ARTIFICIAL INTELLIGENCE

Mandatory Course

Unit 1

Introduction to Artificial Intelligence

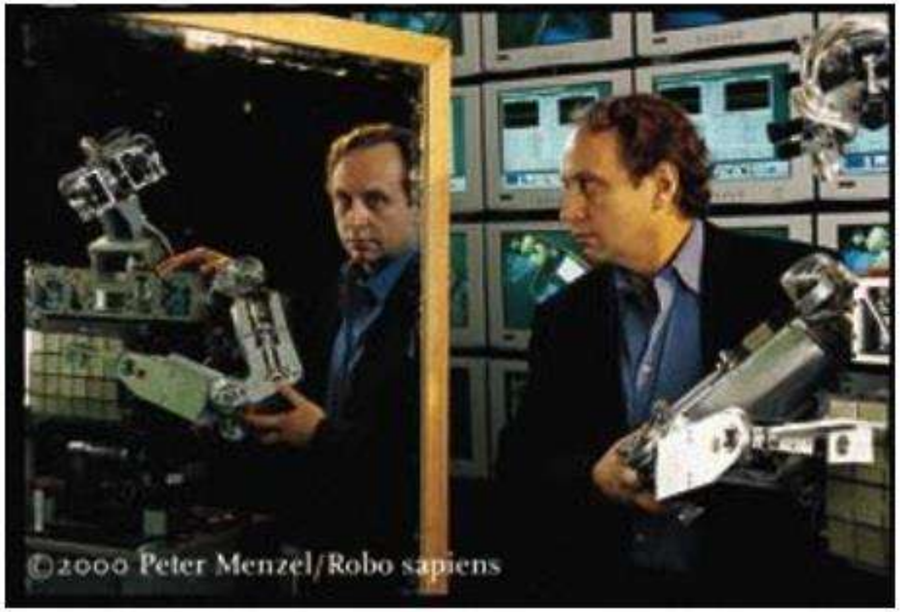
Outline

- What is Intelligence?

Can Machines Think??

Can machines think?

- Today, we use computers to control complex processes, for solving complex problems, decision making, reasoning, natural language . . .



Rodney Brooks i robot Cog, MIT Media Lab

INTELLIGENCE

What is Intelligence ?

"ability to learn, understand and think"

(Oxford dictionary)

- Intelligence : "The capacity to learn and solve problems."
- Artificial Intelligence : Artificial Intelligence (AI) is the simulation of human intelligence by machines.
    1) The ability to solve problems.
    2) The ability to act rationally.
    3) The ability to act like humans.

What is Intelligence??

What is Intelligence???

- Intelligence is the ability to learn about, to learn from, to understand about, and interact with one's environment.

- Intelligence is the faculty of understanding

- Intelligence is not to make no mistakes but quickly to understand how to make them good

(German Poet)

Involved in Intelligence

What's involved in Intelligence?

- Ability to interact with the real world
  - to perceive, understand, and act
  - e.g., speech recognition and understanding and synthesis
  - e.g., image understanding
  - e.g., ability to take actions, have an effect
- Reasoning and Planning
  - modeling the external world, given input
  - solving new problems, planning, and making decisions
  - ability to deal with unexpected problems, uncertainties
- Learning and Adaptation
  - we are continuously learning and adapting
  - our internal models are always being "updated"
    - e.g., a baby learning to categorize and recognize animals

Intelligent Systems

Intelligent Systems in Your Everyday Life

- Post Office
  - automatic address recognition and sorting of mail
- Banks
  - automatic check readers, signature verification systems
  - automated loan application classification
- Customer Service
  - automatic voice recognition
- The Web
  - Identifying your age, gender, location, from your Web surfing
  - Automated fraud detection
- Digital Cameras
  - Automated face detection and focusing

What is AI??

What is artificial intelligence?

- There is **no** agreed definition of the term artificial intelligence. However, there are various definitions that have been proposed. Some will be considered below.

Definitions of AI

- AI is a study in which computer systems are made that think like human beings. Haugeland, 1985 & Bellman, 1978.
- AI is a study in which computer systems are made that act like people. AI is the art of creating computers that perform functions that require intelligence when performed by people. Kurzweil, 1990.
- AI is a study in which computers that rationally think are made. Charniac & McDermott, 1985.
- AI is the study of computations that make it possible to perceive, reason and act. Winston, 1992.
- AI is the study in which systems that rationally act are made. AI is considered to be a study that seeks to explain and emulate intelligent behaviour in terms of computational processes. Schalkeoff, 1990.
- AI is considered to be a branch of computer science that is concerned with the automation of intelligent behavior. Luger & Stubblefield, 1993.

AI Systems

- Speech synthesis, recognition and understanding
  - very useful for limited vocabulary applications
  - unconstrained speech understanding is still too hard
- Computer vision
  - works for constrained problems (hand-written zip-codes)
  - understanding real-world, natural scenes is still too hard
- Learning
  - adaptive systems are used in many applications: have their limits
- Planning and Reasoning
  - only works for constrained problems: e.g., chess
  - real-world is too complex for general systems
- Overall:
  - many components of intelligent systems are "achievable"
  - there are many interesting research problems remaining

HI vs AI (Pros)

Pros

Human Intelligence

- Intuition(Sixth sense), Common sense, Judgement, Creativity, Beliefs etc
- The ability to demonstrate their intelligence by communicating effectively
- Reasoning and Critical thinking

Artificial Intelligence

- Ability to simulate human behavior and cognitive(rational) processes
- Capture and preserve human expertise
- Fast Response. The ability to comprehend large amounts of data quickly.

HI vs AI (Cons)

Cons

Human Intelligence

- Humans are fallible
- They have limited knowledge bases (MS in specialization)
- Information processing of serial nature proceed very slowly in the brain as compared to computers
- Humans are unable to retain large amounts of data in memory.

Artificial Intelligence

- No "common sense"
- Cannot deal with "mixed" knowledge
- May have high development costs.
- Raise legal and ethical concerns

Boundaries of AI ?

Systems that think like humans	Systems that think rationally
Systems that act like humans	Systems that act rationally

	"Like People"	"Rationally"
Think	Cognitive Science	Laws of Thought
Act	Turing Test	Rational Agents

What is AI ?

Systems that think like humans	**Systems that think rationality**
``The exciting new effort to make computers think ... *machines with minds*, in the full and literal sense'' (Haugeland, 1985) ``The automation of activities that we associate with human thinking, activities such as decision-making, problem solving, learning ...'' (Bellman, 1978)	``The study of mental faculties through the use of computational models'' (Charniak and McDermott, 1985) ``The study of the computations that make it possible to perceive, reason, and act'' (Winston, 1992)
Systems that act like humans	**Systems that act like rationality**
``The art of creating machines that perform functions that require intelligence when performed by people'' (Kurzweil, 1990) ``The study of how to make computers do things at which, at the moment, people are better'' (Rich and Knight, 1991)	``A field of study that seeks to explain and emulate intelligent behavior in terms of computational processes'' (Schalkoff, 1990) ``The branch of computer science that is concerned with the automation of intelligent behavior'' (Luger and Stubblefield, 1993)

Acting humanly: The Turing Test approach

- The **Turing Test**, proposed by Alan Turing (Turing, 1950), was designed to provide a satisfactory operational definition of intelligence.
- The computer would need to possess the following capabilities:
  1. **Natural language processing** to enable it to communicate successfully in English (or some other human language);
  2. **Knowledge representation** to store information provided before or during the interrogation;
  3. **Automated reasoning** to use the stored information to answer questions and to draw new conclusions;
  4. **Machine learning** to adapt to new circumstances and to detect and extrapolate patterns.
- To pass the total Turing Test, the computer will need
  - **Computer Vision**
  - **Robotics**

Thinking humanly: The cognitive modeling approach

- Program thinks like a human ..!

We need to get *inside* the actual workings of human minds. There are three ways:

- through introspection--trying to catch our own thoughts as they go by
- or through psychological experiments.
- Brain Imaging

Cognitive science brings together

- Computer Models of AI and
- Experimental Techniques from Psychology

CS vs AI

Cognitive Science vs. Artificial Intelligence

Cognitive science	Artificial intelligence
Intelligence	Artificial intelligence
Knowledge	Knowledge base
Cognition	Information processing
Learning	Machine learning
Learning/understanding language	Natural language processing

Thinking rationality: The Logical approach

- Ensure that all actions performed by computer are justifiable (“rational”)

Facts and Rules in Formal Logic $\longleftrightarrow$ Theorem Prover

- Rational = Conclusions are provable from inputs and prior knowledge
- Problems:
  - Representation of informal knowledge is difficulty
  - Hard to define “provable” plausible reasoning
  - Combinatorial explosion: Not enough time or space to prove desired conclusions.

Acting rationally: The rational agent approach

- **Rational behavior : doing the right thing (that which is expected to maximize goal achievement, given the available information).**
- **Rational Agent is one that acts to achieve the best outcomes or, when there is uncertainty, the best expected outcome.**

Systems Acting Rationally

- Building systems that carry out actions to achieve the **best outcome**
- Rational **behavior**
- May or may not involve rational thinking
  - i.e., consider reflex actions
- This is the definition we will adopt

Rational agents do the best they can given their resources

Rational Agents

very few resources ⟷ lots of resources

no thought

"reflexes"

limited, approximate reasoning

Careful, deliberate reasoning

- Adjust amount of reasoning according to available resources and importance of the result
- This is one thing that makes AI hard

AI (Advantages and Disadvantages)

Advantages :

- More powerful and more useful computers.
- New and improved interfaces.
- Solving new problems.
- Better handling of information.
- Relieves information overload.
- Conversion of information into knowledge.

Disadvantages :

- Increased costs
- Difficulty with software development - slow and expensive
- Few experienced programmers
- Few practical products have reached the market as yet.

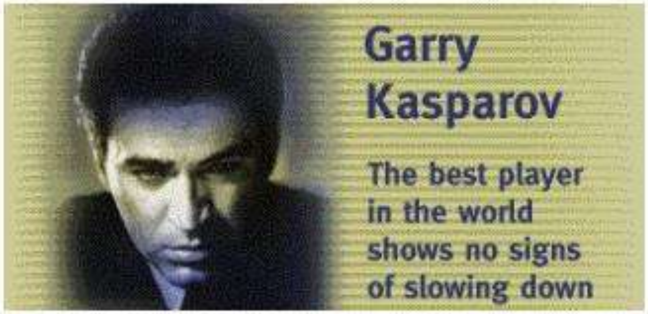
Eliza

ELIZA – example conversation

- Men are all alike
- In what ways?
- They are always bugging us about something or other
- Can you think of specific example?
- Well, my boyfriend made me come here
- Your boyfriend made you come here?
- He says I am depressed much of the time
- I am sorry that you are depressed
- Its true, I am unhappy
- Do you think coming here will help you not to be unhappy?
- I need some help that much seems certain
- What would it mean to you if you got some help?
- Perhaps I could learn to get along with my mother
- Tell me more about your family

Deep Blue

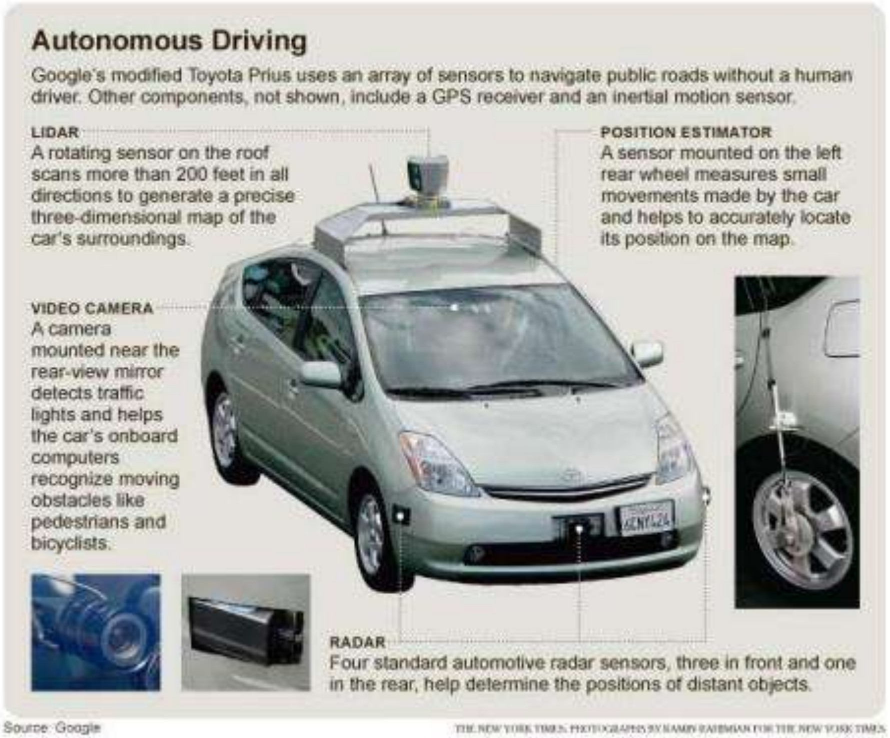
Deep Blue vs. Garry Kasparov (2)



200,000,000 board configurations per second	**3 board configurations** per second
Has **small knowledge** about chess, but **a huge computational capacity**	Has **huge knowledge** about chess, but a considerably **smaller computational capacity**
A machine **has no emotions nor intuition**, it does not forget, cannot be confused or feel uncomfortable	Has feelings and **brilliant intuition**, but can experience **fatigue and boredom** and loss of concentration

Self-Driving Cars

Google self-driving cars



- Google's self-driving car passes 300,000 miles (8/15/2012)

IBM-Watson

IBM Watson – DeepQA project

- February 2011: supercomputer **IBM Watson** defeated the best human competitors in a game of Jeopardy and won $35.734
- Advanced methods of **natural language processing**, **knowledge representation**, **reasoning**, and **information retrieval**

Robotics

Robotics

- Mars rovers
- Autonomous vehicles
  - DARPA Grand Challenge
  - Google self-driving cars
- Autonomous helicopters
- Robot soccer
  - RoboCup
- Personal robotics
  - Humanoid robots
  - Robotic pets
  - Personal assistants?

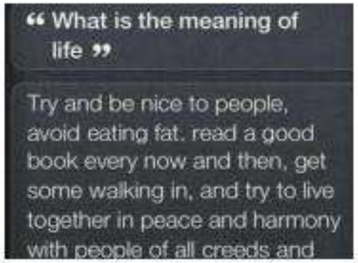
Robocop

RoboCup

Natural Language

- Speech technologies
  - [Google voice search]
  - [Apple Siri]

Machine translation
  - [translate.google.com]
  - [Comparison of several translation systems]



Source	*The Original Text*	*Human Translation*	*Google Translate*
 French "Le Petit Prince" ("The Little Prince") By Antoine de Saint-Exupéry	Le premier soir je me suis donc endormi sur le sable à mille milles de toute terre habitée. J'étais bien plus isolé qu'un naufragé sur un radeau au milieu de l'océan. Alors vous imaginez ma surprise, au lever du jour, quand une drôle de petite voix m'a réveillé. Elle disait: -S'il vous plait... dessine-moi un mouton!	On the first night, I fell asleep on the sand, a thousand miles from any human habitation. I was far more isolated than a shipwrecked sailor on a raft in the middle of the ocean. So you can imagine my surprise at sunrise when an odd little voice woke me up. It said: "Please ... draw me a sheep." - *Wordsworth Children's Classics, 1995*	The first night I went to sleep on the sand a thousand miles from any human habitation. I was more isolated than a shipwrecked sailor on a raft in the middle of the ocean. So imagine my surprise at daybreak, when a funny little voice woke me. She said: "If it pleases you ... draw me a sheep!"

10

Vision

Vision

- handwriting recognition
- Face detection/recognition: many consumer cameras, Apple iPhoto
- Visual search: Google Goggles
- Vehicle safety systems: Mobileye

8

Captcha

Reverse Turing test: CAPTCHA

- CAPTCHA – Completely Automated Public Turing Test to Tell Computers and Humans Apart

- A study (conducted using Amazon Mechanical Turk) shows that CAPTCHA is "often more complex than it should be" – the average solve time is 9.8 seconds (Bursztein et al., 2010)

Wolfram Alpha

Wolfram Alpha – Computational Knowledge Engine

StarCraft

StarCraft AI Competition (2010)

- An agent (an intelligent program that acts autonomously in an environment) must be capable of solving several difficult problems **(planning, optimization, multiagent control)** in a limited time and with limited resources at its disposal

Counting Calories

Deep learning: counting calories (Google)

http://www.popsci.com/google-using-ai-count-calories-food-photos

- A deep learning system estimates the calories based on dish photo

AI-Other Disciplines

Philosophy	Logic, methods of reasoning, mind as physical system, foundations of learning, language, rationality.
Mathematics	Formal representation and proof, algorithms, computation, (un)decidability, (in)tractability
Probability/Statistics	modeling uncertainty, learning from data
Economics	utility, decision theory, rational economic agents
Neuroscience	neurons as information processing units.
Psychology/ Cognitive Science	how do people behave, perceive, process cognitive information, represent knowledge.
Computer engineering	building fast computers
Linguistics	knowledge representation, grammars

History of AI

1943	McCulloch & Pitts: Boolean circuit model of brain
1950	Turing's "Computing Machinery and Intelligence"
1952–69	Look, Ma, no hands!
1950s	Early AI programs, including Samuel's checkers program, Newell & Simon's Logic Theorist, Gelernter's Geometry Engine
1956	Dartmouth meeting: "Artificial Intelligence" adopted
1965	Robinson's complete algorithm for logical reasoning
1966–74	AI discovers computational complexity Neural network research almost disappears
1969–79	Early development of knowledge-based systems
1980–88	Expert systems industry booms
1988–93	Expert systems industry busts: "AI Winter"
1985–95	Neural networks return to popularity
1988–	Resurgence of probability; general increase in technical depth "Nouvelle AI": ALife, GAs, soft computing
1995–	Agents, agents, everywhere . . .
2003–	Human-level AI back on the agenda

Areas of Study in AI

- Reasoning, optimization, resource allocation
  - planning, scheduling, real-time problem solving, intelligent assistants, internet agents
- Natural Language Processing
  - information retrieval, summarization, understanding, generation, translation
- Vision
  - image analysis, recognition, scene understanding
- Robotics
  - grasping/manipulation, locomotion, motion planning, mapping

Where are we now?

- **SKICAT**: a system for automatically classifying the terabytes of data from space telescopes and identifying interesting objects in the sky. 94% classification accuracy, exceeds human abilities.
- **Deep Blue**: the first computer program to defeat champion Garry Kasparov.
- **Pegasus**: a speech understanding program that is a travel agent (1-877-LCS-TALK).
- **Jupiter**: a weather information system (1-888-573-TALK)
- **HipNav**: a robot hip-replacement surgeon.

Where are we now?

- Navlab: a Ford escort that steered itself from Washington DC to San Diego 98% of the way on its own!
- google news: autonomous AI system that assembles “live” newspaper
- DS1: a NASA spacecraft that did an autonomous flyby an asteroid.
- Credit card fraud detection and loan approval
- Search engines: www.citeseer.com, automatic classification and indexing of research papers.
- Proverb: solves NYT puzzles as well as the best humans.

Surprises in AI research

- Tasks difficult for humans have turned out to be “easy”
  - Chess
  - Checkers, Othello, Backgammon
  - Logistics planning
  - Airline scheduling
  - Fraud detection
  - Sorting mail
  - Proving theorems
  - Crossword puzzles

Surprises in AI research

- Tasks easy for humans have turned out to be hard.
  - Speech recognition
  - Face recognition
  - Composing music/art
  - Autonomous navigation
  - Motor activities (walking)
  - Language understanding
  - Common sense reasoning (example: how many legs does a fish have?)

Applications of Artificial Intelligence

- Robotic vehicles
- Speech recognition
- Logistics planning
- Robotics
- Spam filtering
- Game playing
- Machine Translation
- Medicine
- Tele Communications
- Banking

Agent

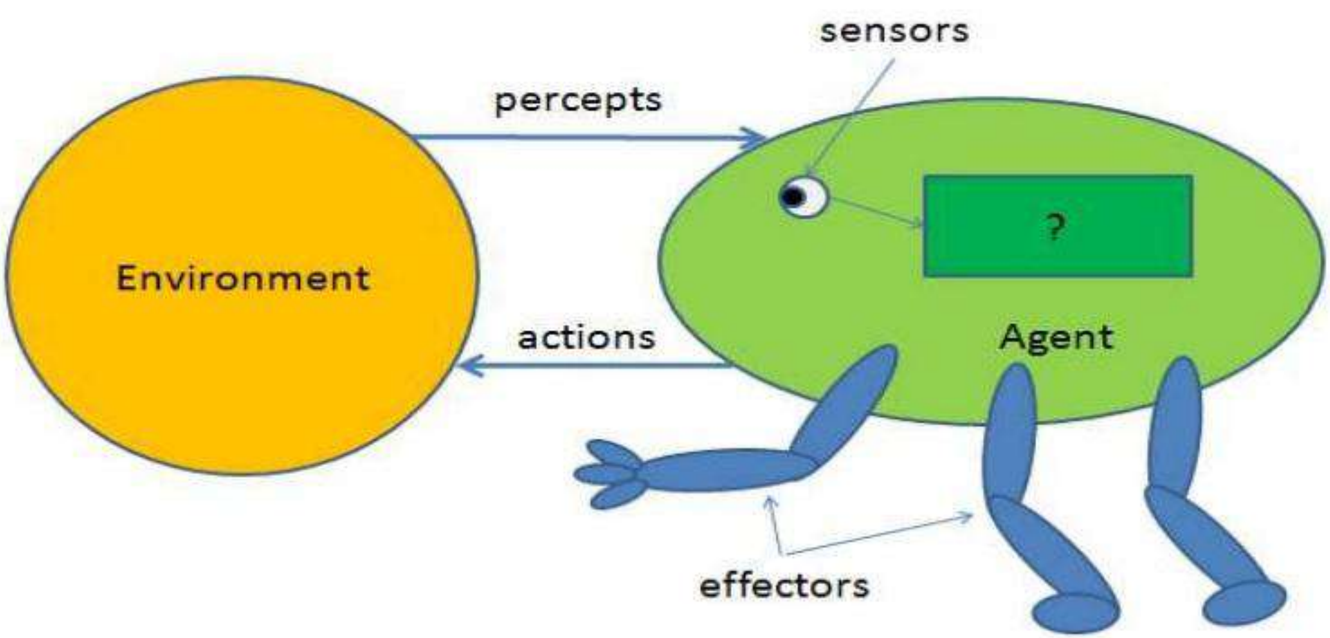
- An **agent** is anything that can be viewed as **perceiving** its environment **through sensors** and **acting** upon that environment **through effectors.**
- **Examples:**

  Human agent

  Robotic agent

  Software agent

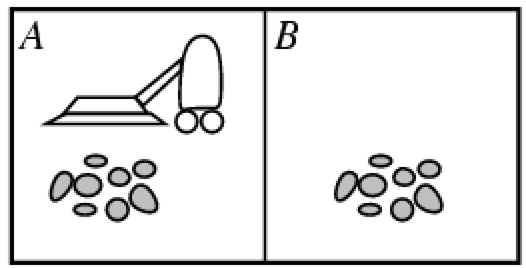
Agent and Environment



Key Definitions

- Agent Percept Sequence
- Agent Function
- Agent Program

Example: Vaccum Cleaner Agent



- Percepts: location and contents, e.g., **[A,Dirty]**
- Actions**: *Left*, *Right*, *Suck*, *NoOp***
- If Current Square is **Dirty**, **then Suck**
  Otherwise, **move** to Other Square

Percept Sequence & Action

Percept sequence	Action
$[A, Clean]$	*Right*
$[A, Dirty]$	*Suck*
$[B, Clean]$	*Left*
$[B, Dirty]$	*Suck*
$[A, Clean], [A, Clean]$	*Right*
$[A, Clean], [A, Dirty]$	*Suck*
$\vdots$	$\vdots$
$[A, Clean], [A, Clean], [A, Clean]$	*Right*
$[A, Clean], [A, Clean], [A, Dirty]$	*Suck*
$\vdots$	$\vdots$

Sensors & Effectors

- Perceives---sensors.
- Percept Sequence.
- The current percept, or a sequence of percepts can influence the actions of an agent.

Sensors & Effectors

- Change the environment- Effectors
- Action
- Action sequences
- Agent Program

Structure of agents

- A simple agent program can be defined mathematically as an **agent function** which **maps** every possible **precepts sequence** to a **possible action** the agent can perform.

  F: p->A*

- The term **percept** is use to the agent's perceptional inputs at any given instant.

Agents

- Autonomous Agent: Decide autonomously which action to take in the current situation to maximize progress towards its goals.
- Performance measure: An objective criterion for success of an agent's behavior.

  E.g., performance measure of a vacuum- cleaner agent could be **amount of dirt cleaned up**, **amount of time taken**, **amount of electricity consumed**, amount of noise generated, etc.

Rational Agent

- AI is about **building rational agents.**
- An agent is something that perceives and acts.
- A **rational agent** always does the **right thing.**

Rational agents

- An agent should strive to **"do the right thing",** based on **what it can perceive** and **the actions it can perform. The right action** is the one that will cause the **agent** to be most **successful.**
- **Rational Agent:** For each possible **percept sequence**, a **rational agent** should **select an action** that is **expected to maximize its performance measure**, given the evidence provided by the percept sequence and whatever built-in knowledge the agent has.

Rationality

Perfect Rationality:

Assumes that the rational agent **knows all** and will take the **action that maximize the utility.**

Human beings do not satisfy this definition of rationality.

Intelligent Agents

IntelligentAgent:

must sense,

must act,

must be autonomous(to some extent)

must be rational.

PEAS

PEAS:

- ✓Performance measure
- ✓Environment
- ✓Actuators
- ✓Sensors

PEAS

Specify Task Environment as fully as Possible

- Consider, e.g., the task of designing an **automated taxi driver:**
  - **Performance measure:** Safe, fast, legal, comfortable trip, maximize profits
  - **Environment:** Roads, other traffic, pedestrians, customers
  - **Actuators:** Steering wheel, accelerator, brake, signal, horn
  - **Sensors:** Cameras, sonar, speedometer, GPS, odometer, engine sensors, keyboard

PEAS

Agent: **Medical diagnosis system**

- **Performance measure:** Healthy patient, minimize costs
- **Environment:** Patient, hospital, staff
- **Actuators:** Screen display (questions, tests, diagnoses, treatments, referrals)
- **Sensors:** Keyboard (entry of symptoms, findings, patient's answers)

PEAS

Agent: **Part-picking robot**

- **Performance measure:** Percentage of parts in correct bins
- **Environment:** Conveyor belt with parts, bins
- **Actuators:** Jointed arm and hand
- **Sensors:** Camera, joint angle sensors

PEAS

Agent: **Interactive English tutor**

- **Performance measure:** Maximize student's score on test
- **Environment:** Set of students
- **Actuators:** Screen display (exercises, suggestions, corrections)
- **Sensors:** Keyboard

Agent Environment

- Environments in which agents operate can be defined in different ways.
- It is helpful to view the following definitions as referring to the way the environment appears from the point of view of the agent itself.

Properties of Task Environment

- Fully Observable vs Partially Observable
- Single Agent vs Multi Agent
- Deterministic vs Stochastic
- Episodic vs Sequential
- Static vs Dynamic
- Discrete vs Continous
- Known vs Unknown

Environment: Observability

- Fully Observable:
  – Access to **Complete State of Environment** at each point of time.
  – Sensors **detect** all aspects that are **relevant** to the **choice of action**
  – **Eg: Chess**
- Partially Observable:
  – Noisy and Inaccurate Sensors
  – **Eg: Vaccum Agent (Local Dirt Sensor)**

Environment: Agents

- **Single Agent:**

Eg: **Crossword Puzzle**

- **Multi Agent:**

Eg: **Chess**

Competitive Multi Agent

Partially Cooperative Multi Agent

Environment: Determinism

- **Deterministic:**

  -Next State is Completely **determined** by **Current State** and **Action executed** by the **Agent.**

Eg: **Crossword Puzzle, Chess**

- **Stochastic:**

  - **Partially Observable**

Eg: **Taxi Driving**

Environment: Episodicity

- **Episodic:**

  - Agents Experience divided into **Atomic Episodes**

  - Each Episode-Single Action

  - Next Episode does not depend on actions taken in previous episodes.

Eg: **Part Picking Robot, Image Analysis**

- **Sequential:**

  - **Current Decision** could **affect** all **Future Decisions.**

Eg: **Chess, Taxi Driving**

Environment: Dynamism

- **Dynamic:**
  – **Changes over time** independent of the actions of the agent
  – Eg: **Taxi Driving, Interactive English Tutor**
- **Static:**
  – **Does not change from one state to next**
  – Eg: **Crossword Puzzle**
- **Semi Dynamic:**
  – **Does not change but Performance Score does**
  – Eg: **Chess**

Environment: Continuity

- **Discrete:**
  - Number of distinct percepts and actions is limited.
  - Eg: **Crossword Puzzle, Chess**
- **Continuous:**
  - Number of distinct percepts and actions is not limited.
  - Eg: **Taxi Driving**

Environment: Known

- **Known:**
  - **Outcomes for all actions are given**
  - **Partially Observable**
  - Eg: **Solitaire Card Games**
- **UnKnown:**
  - **Agent have to Learn to make Good Decisions**
  - **Fully Observable**
  - Eg: **New Video Game**

Examples

Task Environment	Observable	Agents	Deterministic	Episodic	Static	Discrete
Crossword puzzle	Fully	Single	Deterministic	Sequential	Static	Discrete
Chess with a clock	Fully	Multi	Deterministic	Sequential	Semi	Discrete
Poker	Partially	Multi	Stochastic	Sequential	Static	Discrete
Backgammon	Fully	Multi	Stochastic	Sequential	Static	Discrete
Taxi driving	Partially	Multi	Stochastic	Sequential	Dynamic	Continuous
Medical diagnosis	Partially	Single	Stochastic	Sequential	Dynamic	Continuous
Image analysis	Fully	Single	Deterministic	Episodic	Semi	Continuous
Part-picking robot	Partially	Single	Stochastic	Episodic	Dynamic	Continuous
Refinery controller	Partially	Single	Stochastic	Sequential	Dynamic	Continuous
Interactive English tutor	Partially	Multi	Stochastic	Sequential	Dynamic	Discrete

Figure 2.6 Examples of task environments and their characteristics.

Structure of Agents

- Agent Program
- Agent Function

Agent= Architecture + Program

```
function TABLE-DRIVEN-AGENT(percept) returns an action
  persistent: percepts, a sequence, initially empty
              table, a table of actions, indexed by percept sequences, initially fully specified

  append percept to the end of percepts
  action ← LOOKUP(percepts, table)
  return action
```

Figure 2.7 The TABLE-DRIVEN-AGENT program is invoked for each new percept and returns an action each time. It retains the complete percept sequence in memory.

Classes of Intelligent Agents

Intelligent agents are grouped in to five classes based on their degree of **perceived intelligence** and **capability.**

- ✓ Simple reflex agents
- ✓ Model based reflex agents
- ✓ Goal based agents
- ✓ Utility based agents
- ✓ Learning agents

Simple reflex agents

- Simple reflex agents **act** only on the basis of the **current percept**, ignoring the rest of the percept history.
- The agent function is based on **the** ***condition-action rule*****: if condition then action.**
  - **Eg: if** car-infront-isbraking **then** initiate-braking
- Succeeds when the environment is **fully observable.**
- **Randomized Simple Reflex Agent**

Simple reflex agents

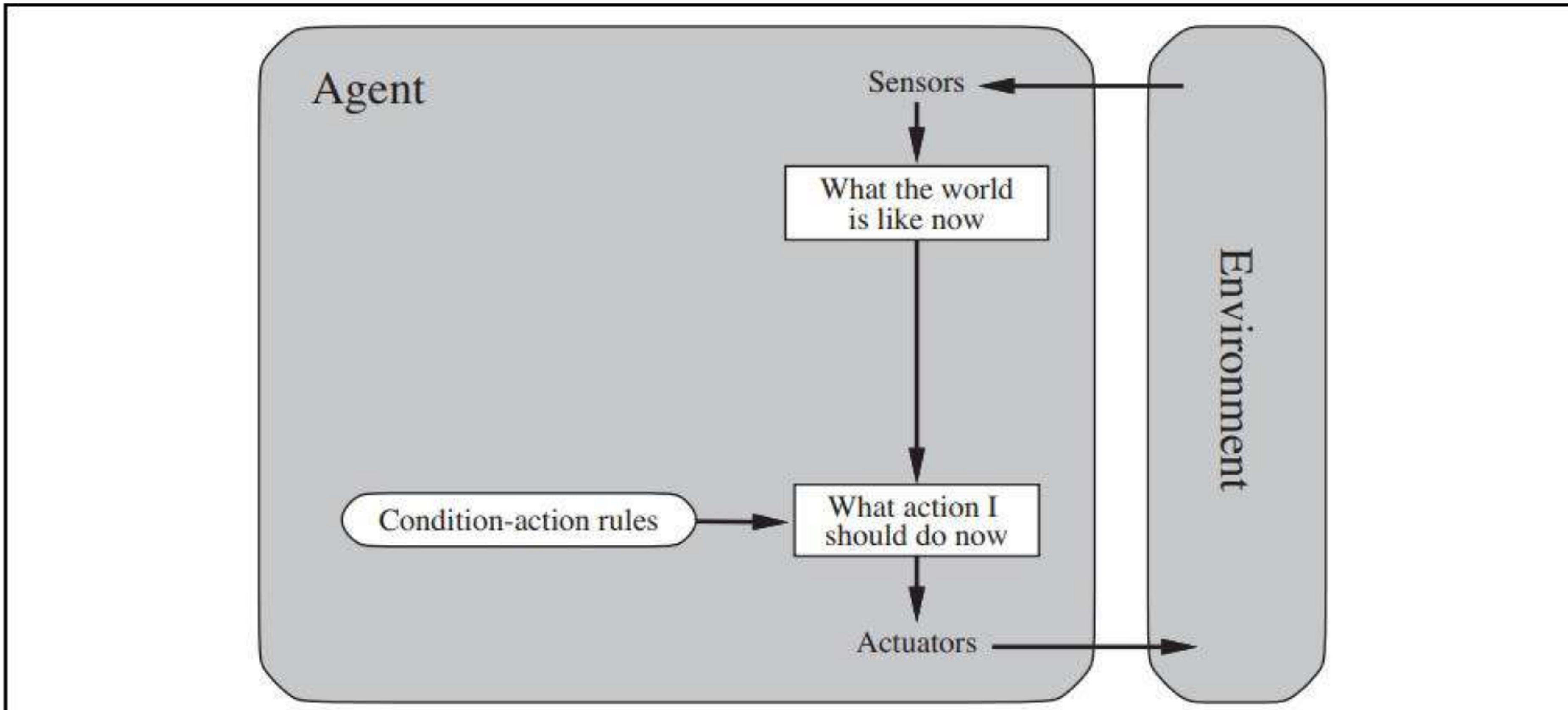


Figure 2.9 Schematic diagram of a simple reflex agent.

Simple reflex agents

```
function SIMPLE-REFLEX-AGENT(percept) returns an action
  persistent: rules, a set of condition–action rules

  state ← INTERPRET-INPUT(percept)
  rule ← RULE-MATCH(state, rules)
  action ← rule.ACTION
  return action
```

Figure 2.10 A simple reflex agent. It acts according to a rule whose condition matches the current state, as defined by the percept.

Model Based Agent

- A model-based agent can **handle** a **partially observable** environment.
- **Maintain Internal State**(Keep Track of Part of World it cant see now).
- Update **Internal State**
    - How World Evolves Independently
    - How Agents Own Actions affect the World
- This knowledge about **"how the world evolves"** is called a **model** of the world, hence the name **"model-based agent".**

Model Based Agent

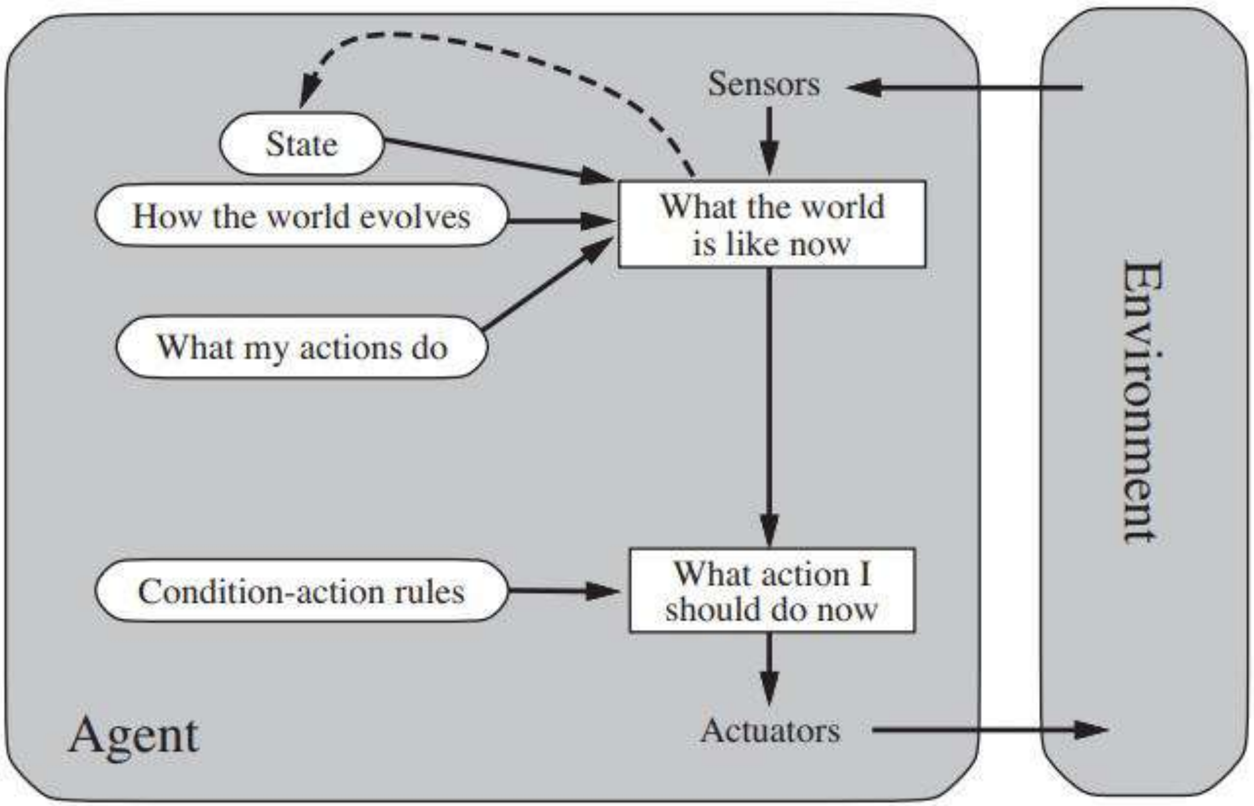


Figure 2.11 A model-based reflex agent.

Model Based Agent

function MODEL-BASED-REFLEX-AGENT(*percept*) **returns** an action
persistent: *state*, the agent's current conception of the world state
model, a description of how the next state depends on current state and action
rules, a set of condition–action rules
action, the most recent action, initially none

state ← UPDATE-STATE(*state*, *action*, *percept*, *model*)
rule ← RULE-MATCH(*state*, *rules*)
action ← *rule*.ACTION
return *action*

Figure 2.12 A model-based reflex agent. It keeps track of the current state of the world, using an internal model. It then chooses an action in the same way as the reflex agent.

Goal Based Agent

- **Goal Information**(Describes Situations that are desirable).
- Combine **Goal Information** with **Model.**
- **Goal Based Action Selection is Straight Forward**
- **Choose** among **multiple possibilities**, **selecting the one which reaches a goal state.**
- **Search and Planning(Goal Based)**

Goal Based Agent

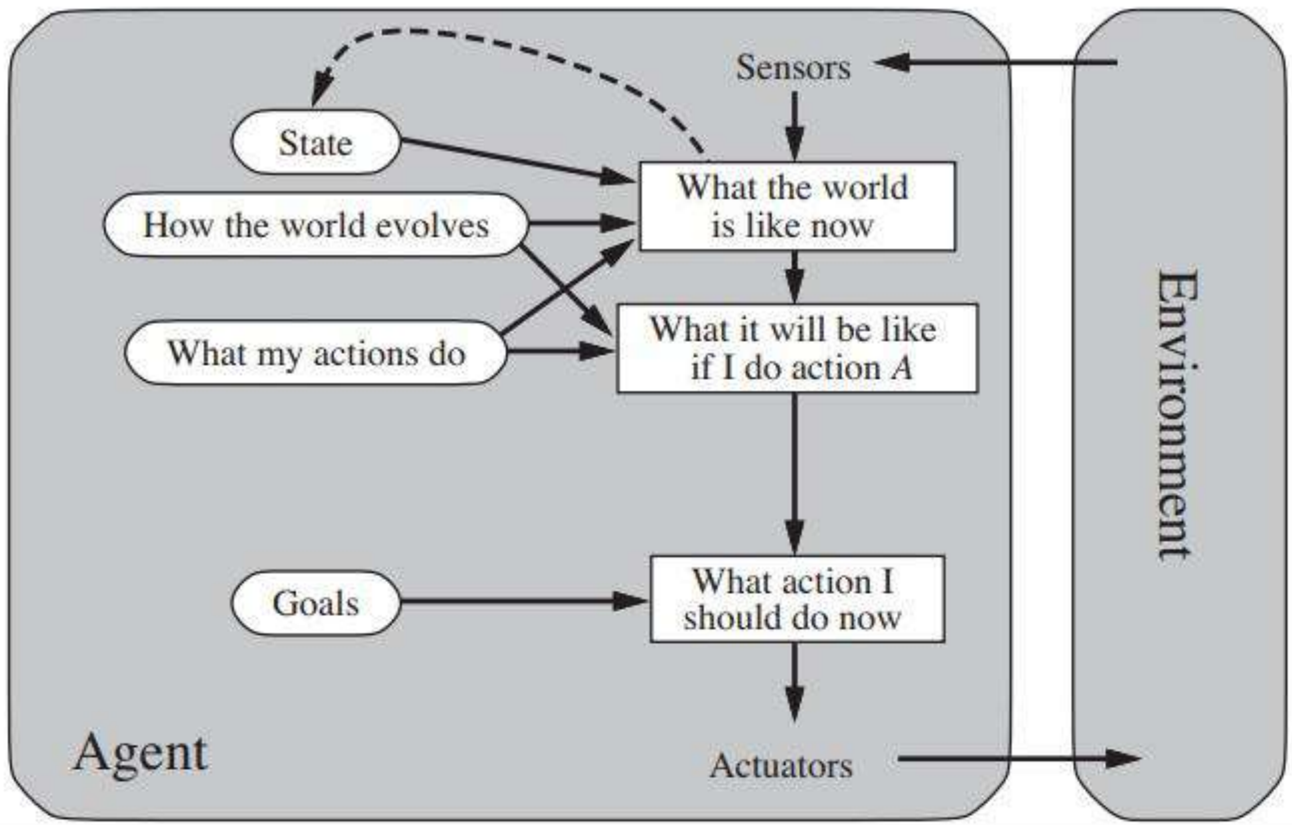


Figure 2.13 A model-based, goal-based agent. It keeps track of the world state as well as a set of goals it is trying to achieve, and chooses an action that will (eventually) lead to the achievement of its goals.

Utility Based Agent

- Goal Based agents only **distinguish** between **goal states** and **non-goal states.**
- Define a **measure** of **how desirable** a **particular state is**.
- This measure can be obtained through the use of a ***utility function*** which **maps** a **state** to a **measure of the utility of the state**.
- **Chooses Action** that **Maximizes Expected Utility of Action Outcomes.**

Utility Based Agent

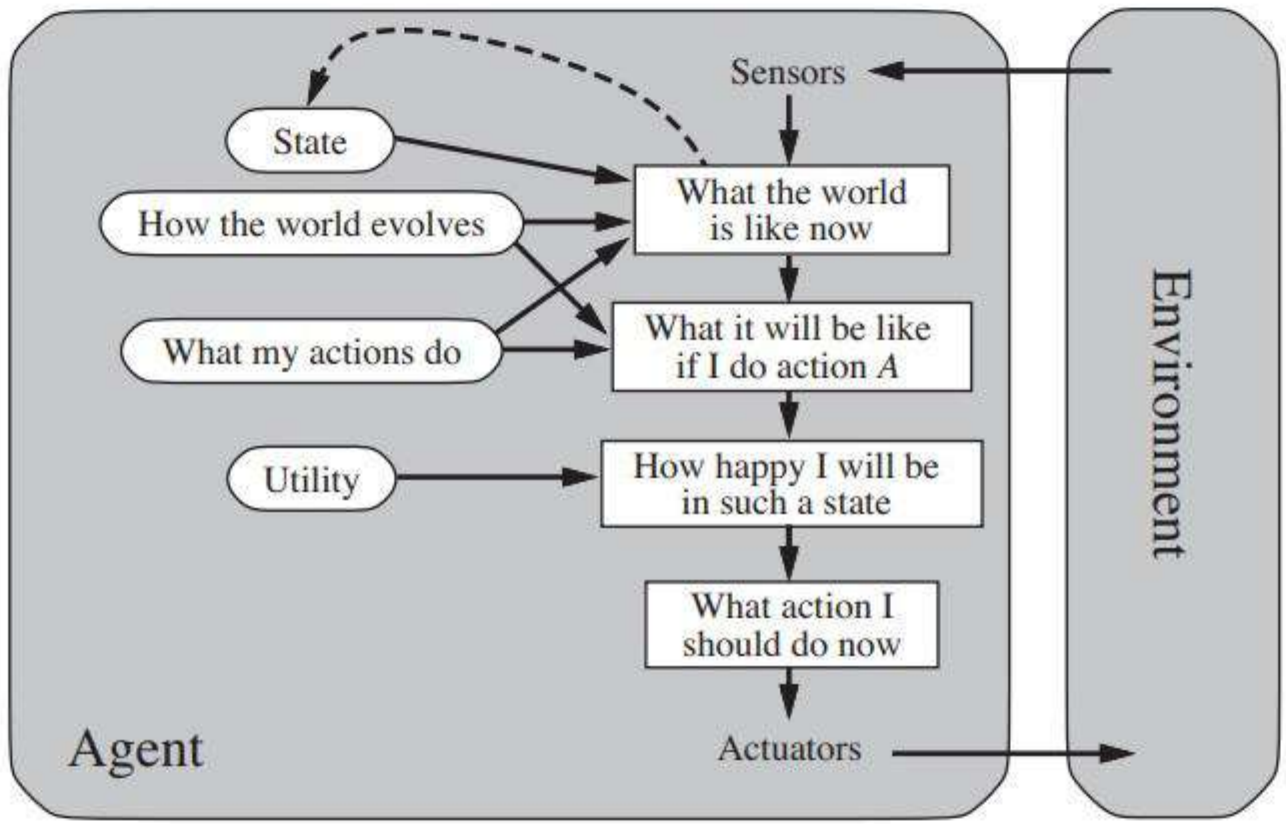


Figure 2.14 A model-based, utility-based agent. It uses a model of the world, along with a utility function that measures its preferences among states of the world. Then it chooses the action that leads to the best expected utility, where expected utility is computed by averaging over all possible outcome states, weighted by the probability of the outcome.

Learning Agent

- It allows the **agents** to **initially operate** in **unknown environments** and to **become more competent**
- **Four Components:**
  - ✓ Critic
  - ✓ Learning Element
  - ✓ Performance Element
  - ✓ Problem Generator
- Eg: **Automated Taxi**

Learning Agent

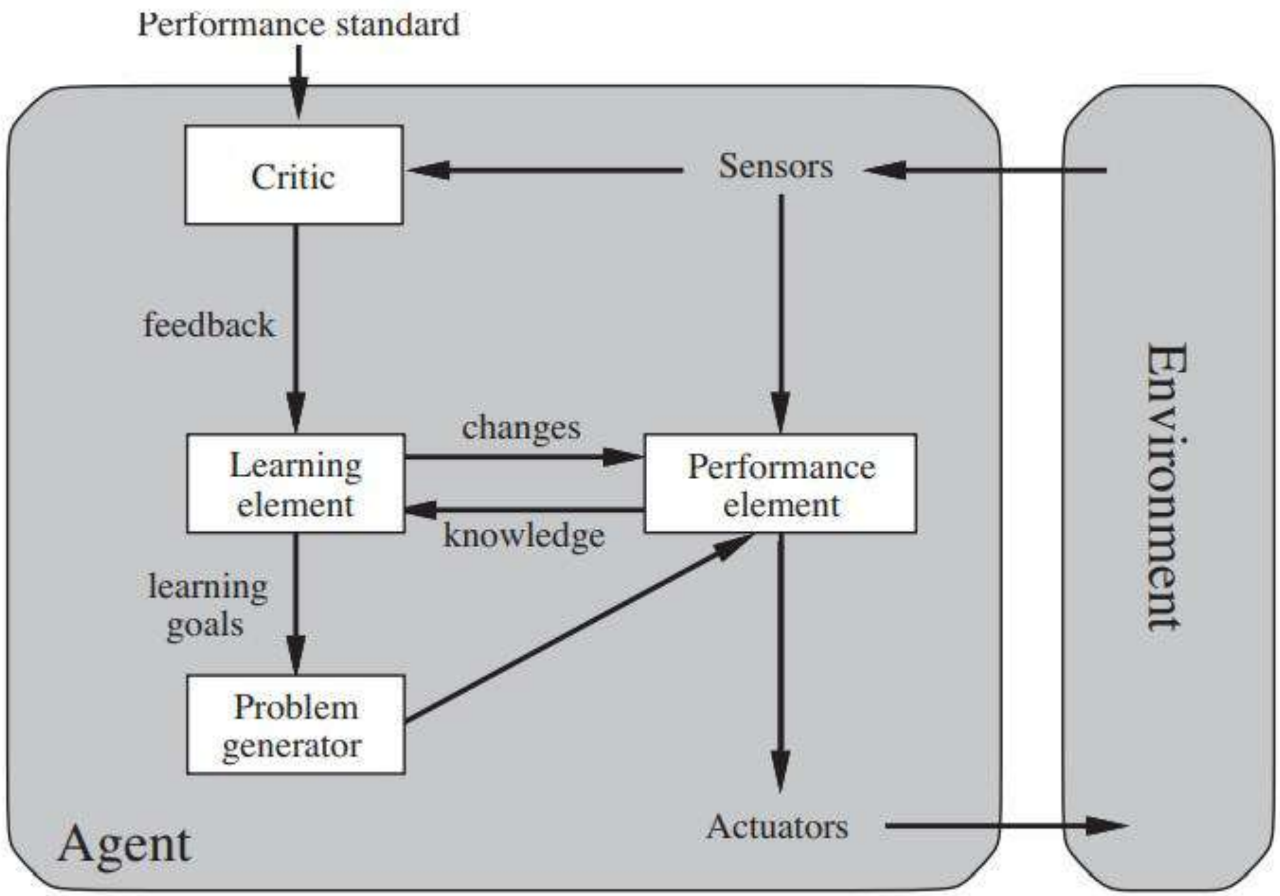


Figure 2.15 A general learning agent.

Applications of Intelligent Agents

- Intelligent Agents are applied as **Automated Online Assistants.**
- Use in **Smart Phones.**

How the components of agent programs work

- There are three ways in which the agent of program work:
  - **Atomic**
  - **Factored**
  - **Structured**

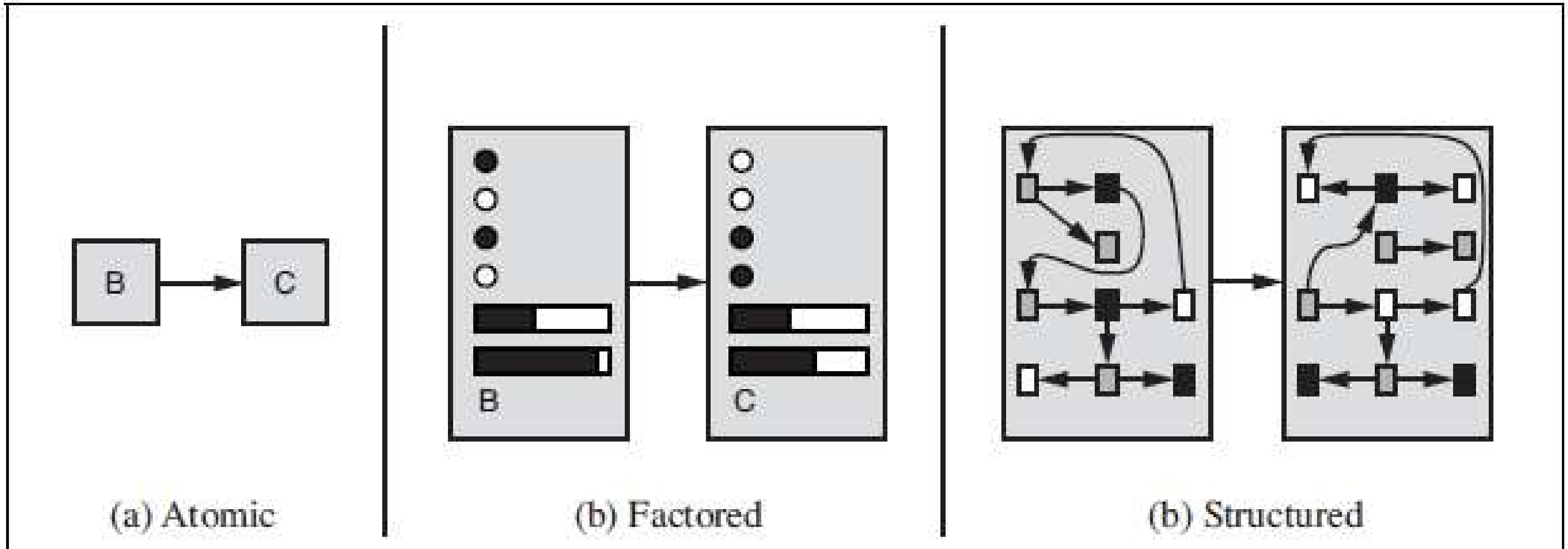


Figure 2.16 Three ways to represent states and the transitions between them. (a) Atomic representation: a state (such as B or C) is a black box with no internal structure; (b) Factored representation: a state consists of a vector of attribute values; values can be Boolean, real-valued, or one of a fixed set of symbols. (c) Structured representation: a state includes objects, each of which may have attributes of its own as well as relationships to other objects.

This chapter has been something of a whirlwind tour of AI, which we have conceived of as the science of agent design. The major points to recall are as follows:

- An **agent** is something that perceives and acts in an environment. The **agent function** for an agent specifies the action taken by the agent in response to any percept sequence.
- The **performance measure** evaluates the behavior of the agent in an environment. A **rational agent** acts so as to maximize the expected value of the performance measure, given the percept sequence it has seen so far.
- A **task environment** specification includes the performance measure, the external environment, the actuators, and the sensors. In designing an agent, the first step must always be to specify the task environment as fully as possible.
- Task environments vary along several significant dimensions. They can be fully or partially observable, single-agent or multiagent, deterministic or stochastic, episodic or sequential, static or dynamic, discrete or continuous, and known or unknown.
- The **agent program** implements the agent function. There exists a variety of basic agent-program designs reflecting the kind of information made explicit and used in the decision process. The designs vary in efficiency, compactness, and flexibility. The appropriate design of the agent program depends on the nature of the environment.
- **Simple reflex agents** respond directly to percepts, whereas **model-based reflex agents** maintain internal state to track aspects of the world that are not evident in the current percept. **Goal-based agents** act to achieve their goals, and **utility-based agents** try to maximize their own expected "happiness."
- All agents can improve their performance through **learning**.

Thank You

INFORMED SEARCH

Informed search

- We have seen that ***uninformed*** search methods that **systematically** explore the ***state space*** and ***find the goals***.
- ***Inefficient*** in most cases.
- ***Informed Search*** methods use **problem specific knowledge**, are more ***efficient***.
- ***Informed Search*** *method* tries to improve problem solving efficiency by using **problem specificknowledge.**

Continued

- A search strategy which searches the most promising branches of the state-space first can:
  - ***find a solution more quickly,***
  - ***find solutions even when there is limited time available,***
  - ***often find a better solution****, since more profitable parts of the state- space can be examined, while ignoring the unprofitable parts.*
- A search strategy which is better than another at identifying the most promising branches of a search-space is said to be more informed.

Continued

- The general approach we consider is called **best-first search**. Best- first search is an instance of the general TREE-SEARCH or GRAPH- SEARCH algorithm in which a **node is selected for expansion** based on an **evaluation function, f(n).**
- The evaluation function is construed as a cost estimate, so the node with the **lowest evaluation is expanded first.**
- The implementation of best-first graph search is **identical** to that for **uniform-cost search** (previous topic), except for the **use of f instead of g** to order the priority queue.

Heuristics

- Heuristic is a ***rule of thumb***.
- "Heuristics are ***criteria***, ***methods*** or ***principles*** for ***deciding*** which among several alternative courses of action promises to be the most effective in ***order to achieve some goals***", Judea Pearl.

Can use heuristics to ***identify the most promising search path***.

Continued

- A heuristic function at a node n is an ***estimate*** of the ***optimum*** cost from the current node to a goal. Denoted by ***h(n)***.

h(n)=estimated cost of the cheapest path from node n to a goal node.

- Example

– *Want to find the path from Vijayawada to Hyderabad*
– *Heuristic for Hyderabad may be straight line distance between Vijayawada and Hyderabad.*
– *h(Vijayawada)=Euclidian distance(Vijayawada, Hyderabad)*

Heuristics Example

- 8-Puzzle: Number of tiles out of place
- ***h(n)=5*** **(1,2,3,4,8 are not in correct location)**

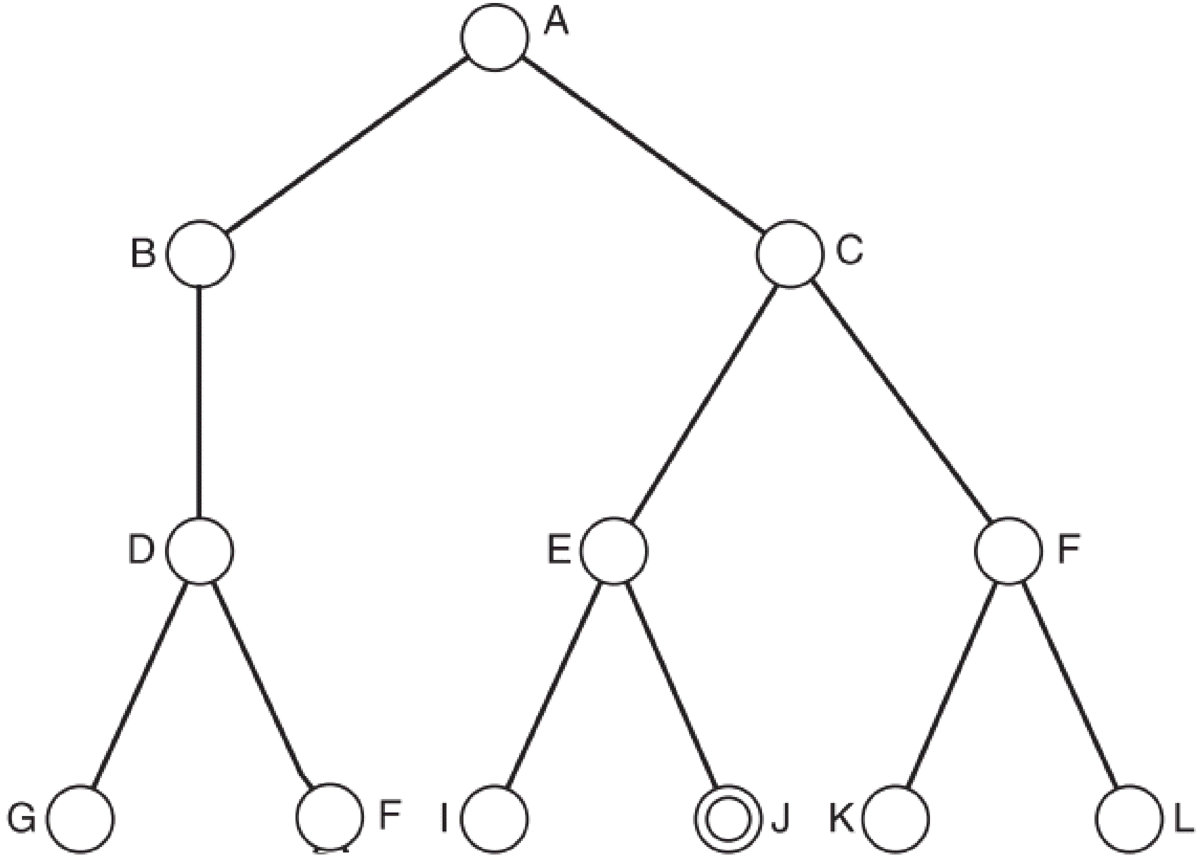
Initial State

1	2	3
8		4
7	6	5

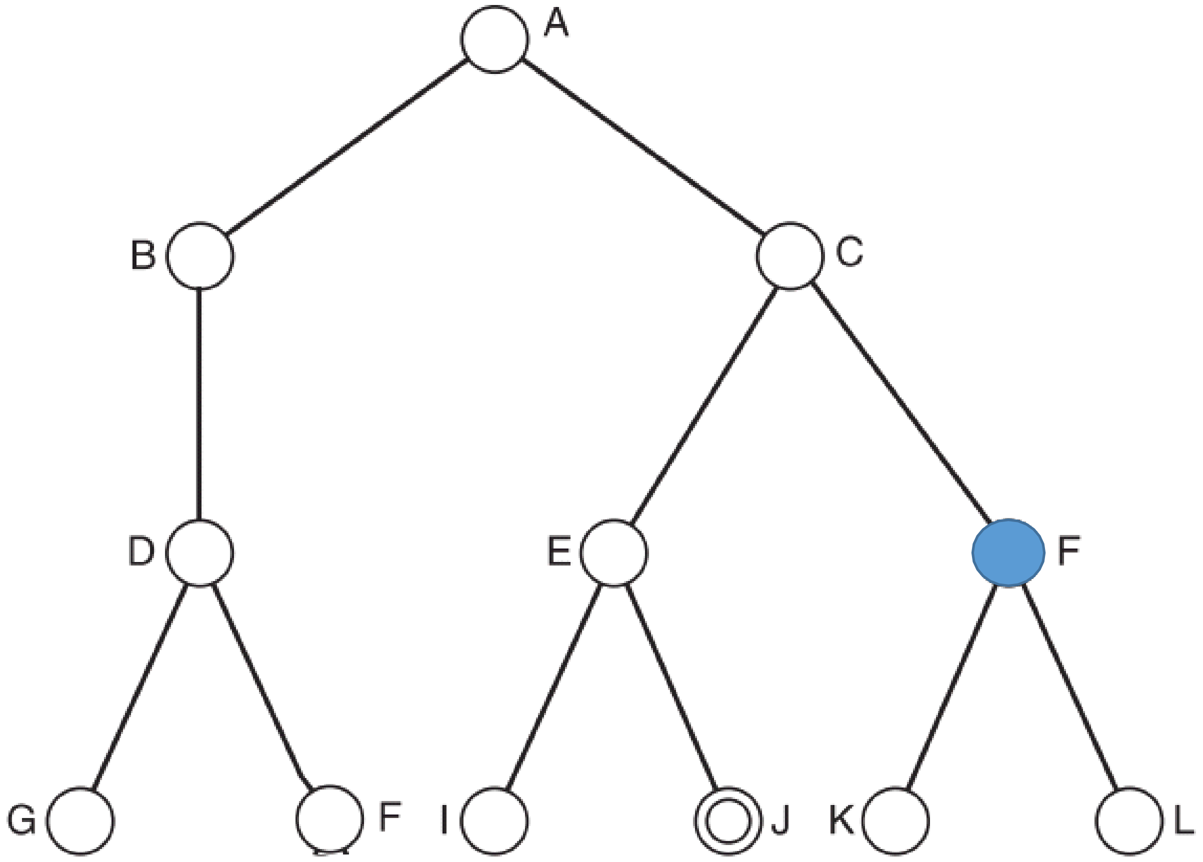
Goal State

2	8	1
	4	3
7	6	5

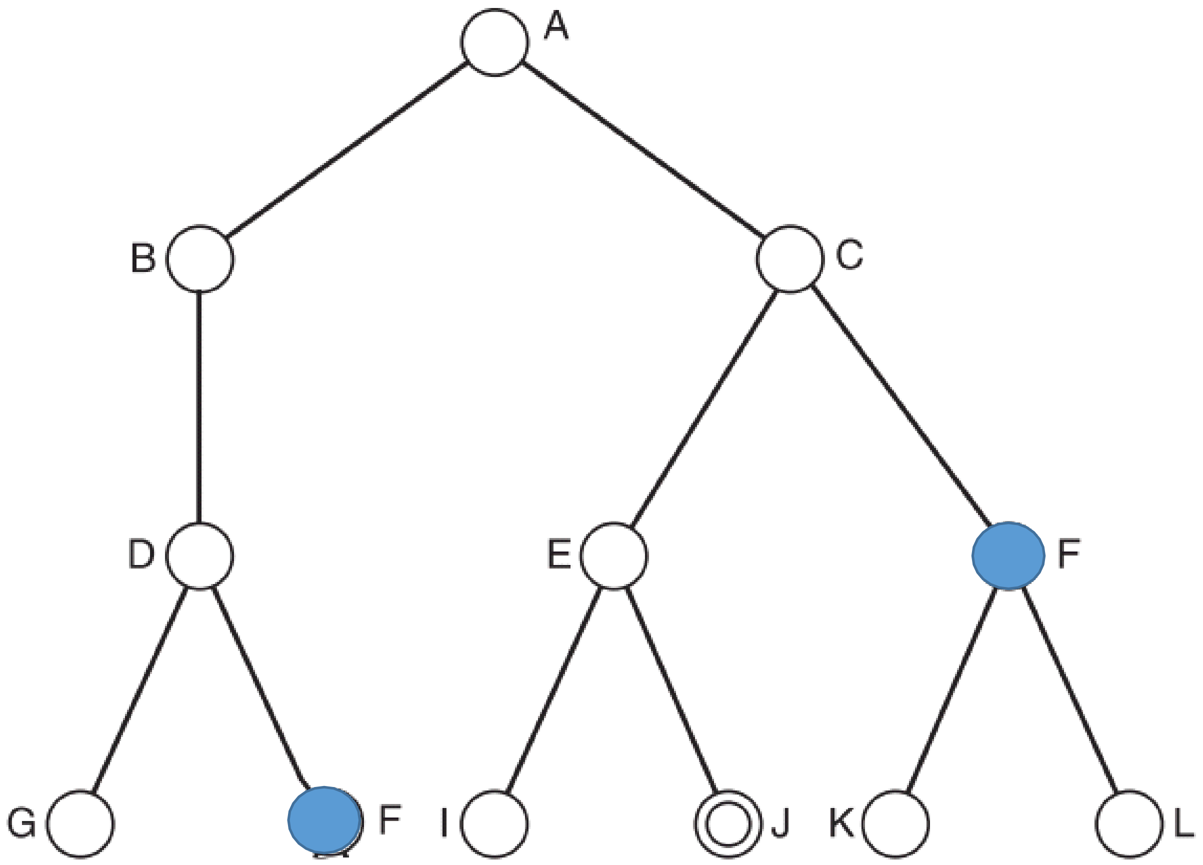
Uninformed Search Vs. Informed Search OR Heuristically Informed Search



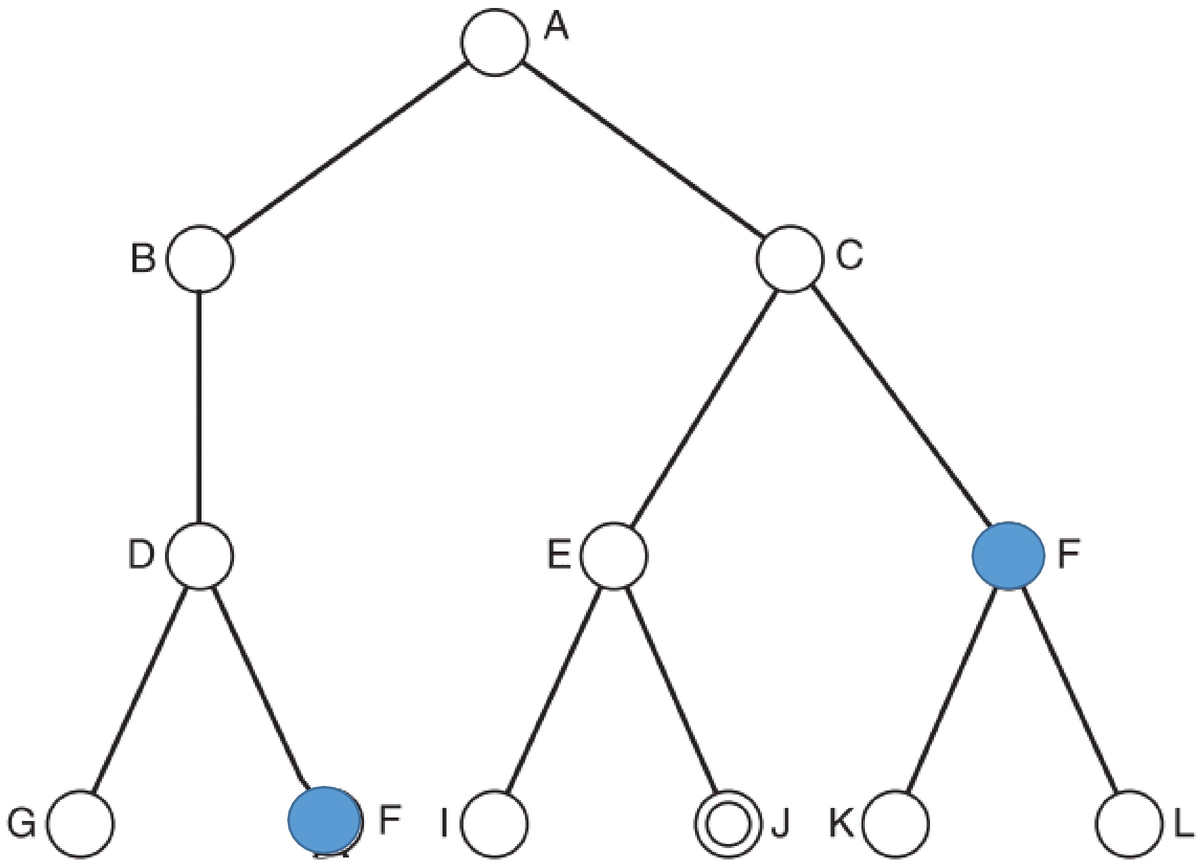
Uninformed Search Vs. Informed Search OR Heuristically Informed Search



Uninformed Search Vs. Informed Search OR Heuristically Informed Search

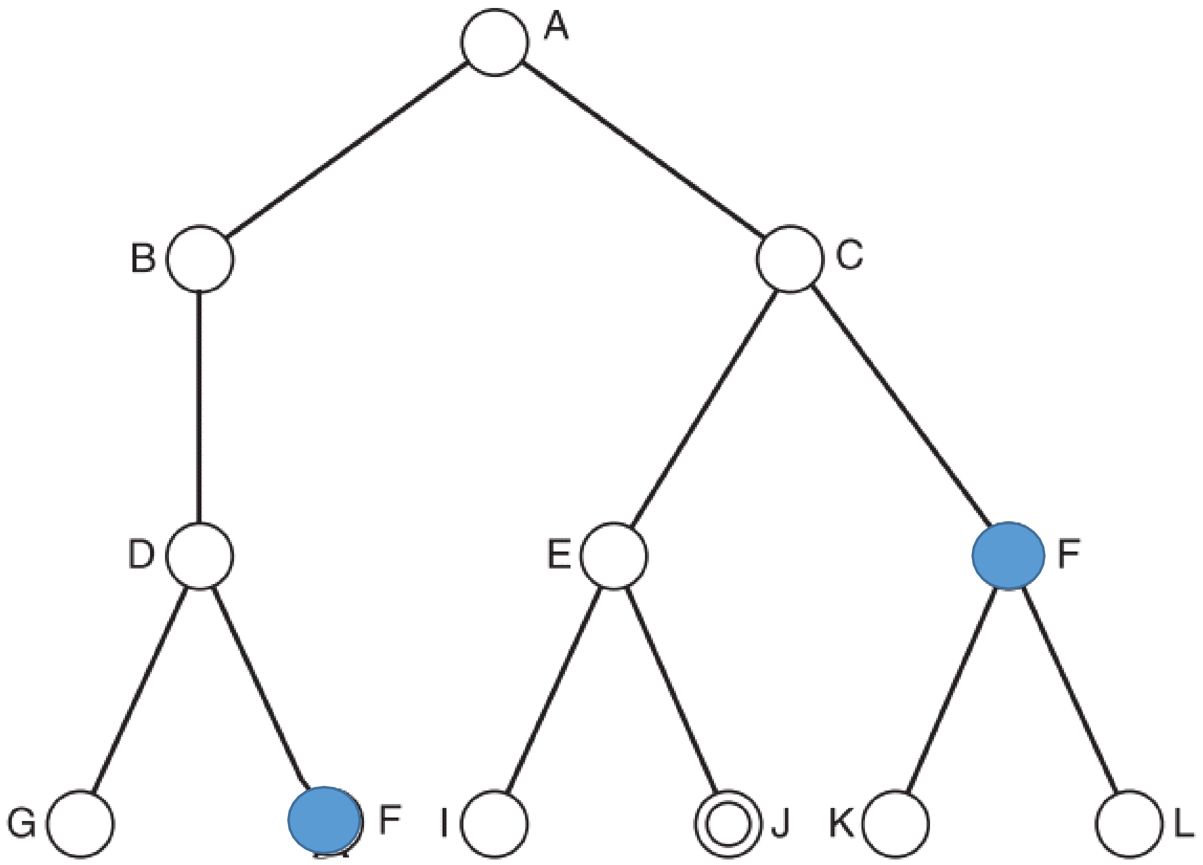


Uninformed Search Vs. Informed Search OR Heuristically Informed Search



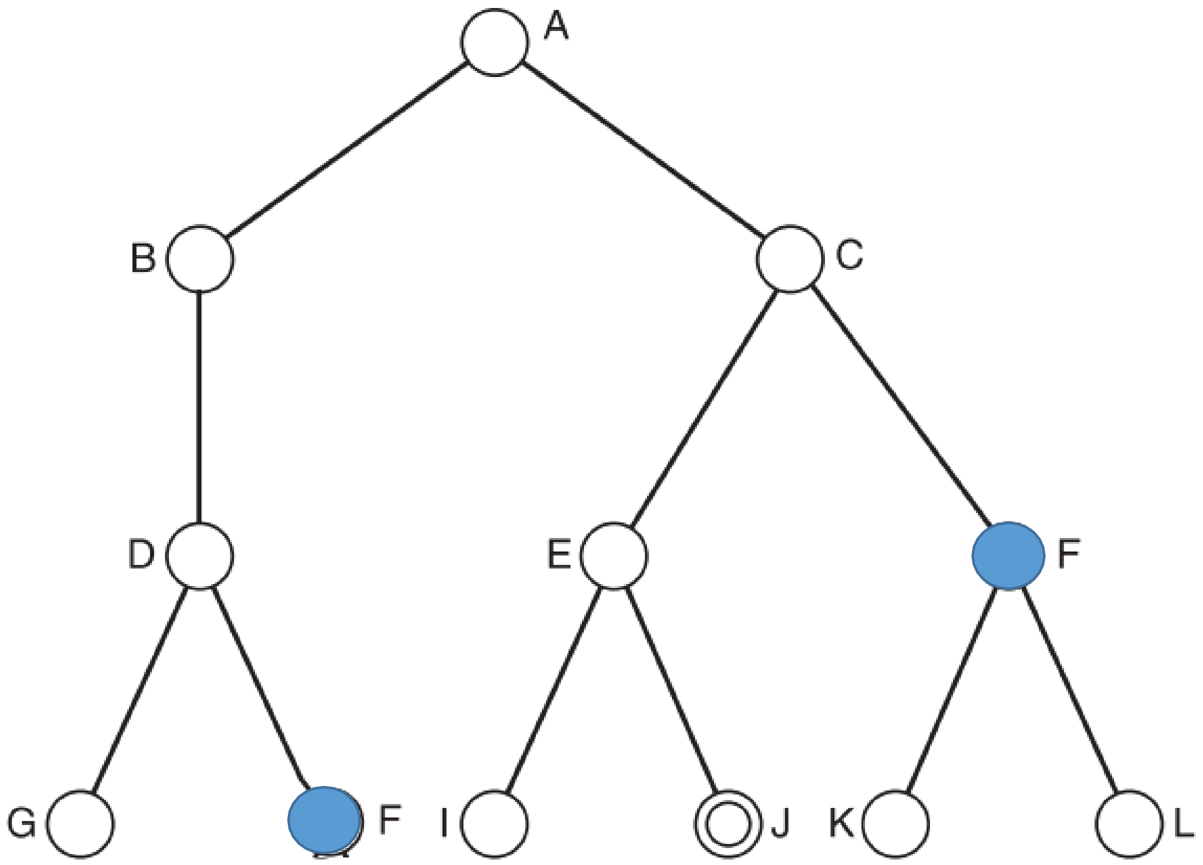
Heuristic Evaluation

Uninformed Search Vs. Informed Search OR Heuristically Informed Search



Heuristic Evaluation

Uninformed Search Vs. Informed Search OR Heuristically Informed Search

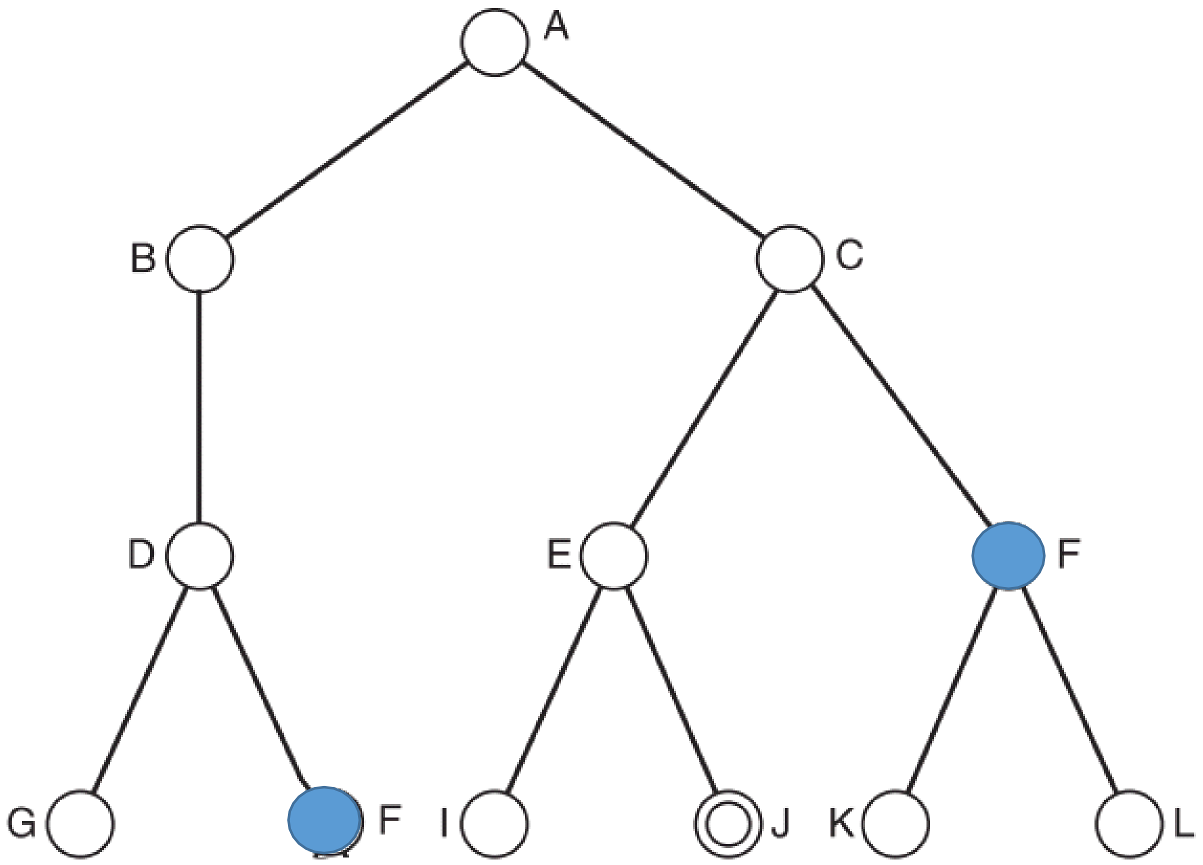


Heuristic Evaluation

↓

Heuristic Value

Uninformed Search Vs. Informed Search OR Heuristically Informed Search

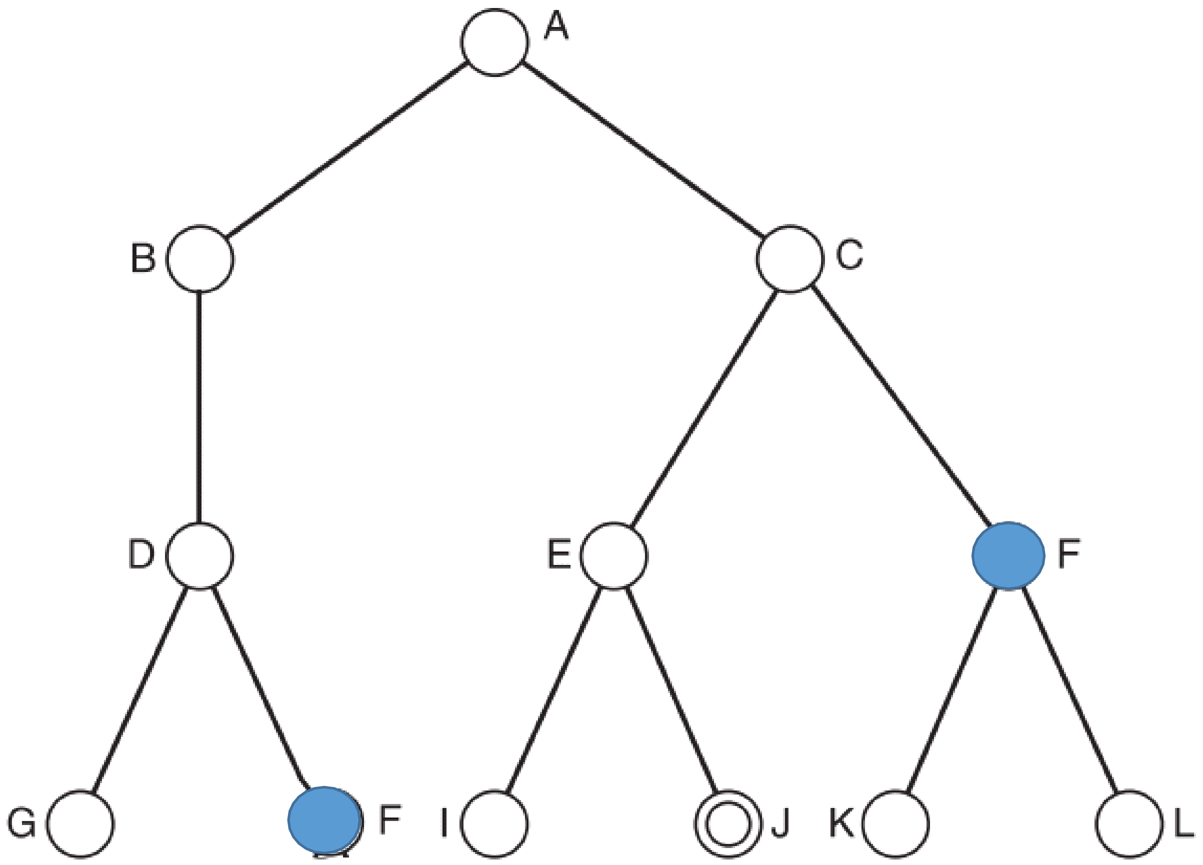


Heuristic Evaluation

Heuristic Value

Small Heuristic Value

Uninformed Search Vs. Informed Search OR Heuristically Informed Search



Heuristic Evaluation

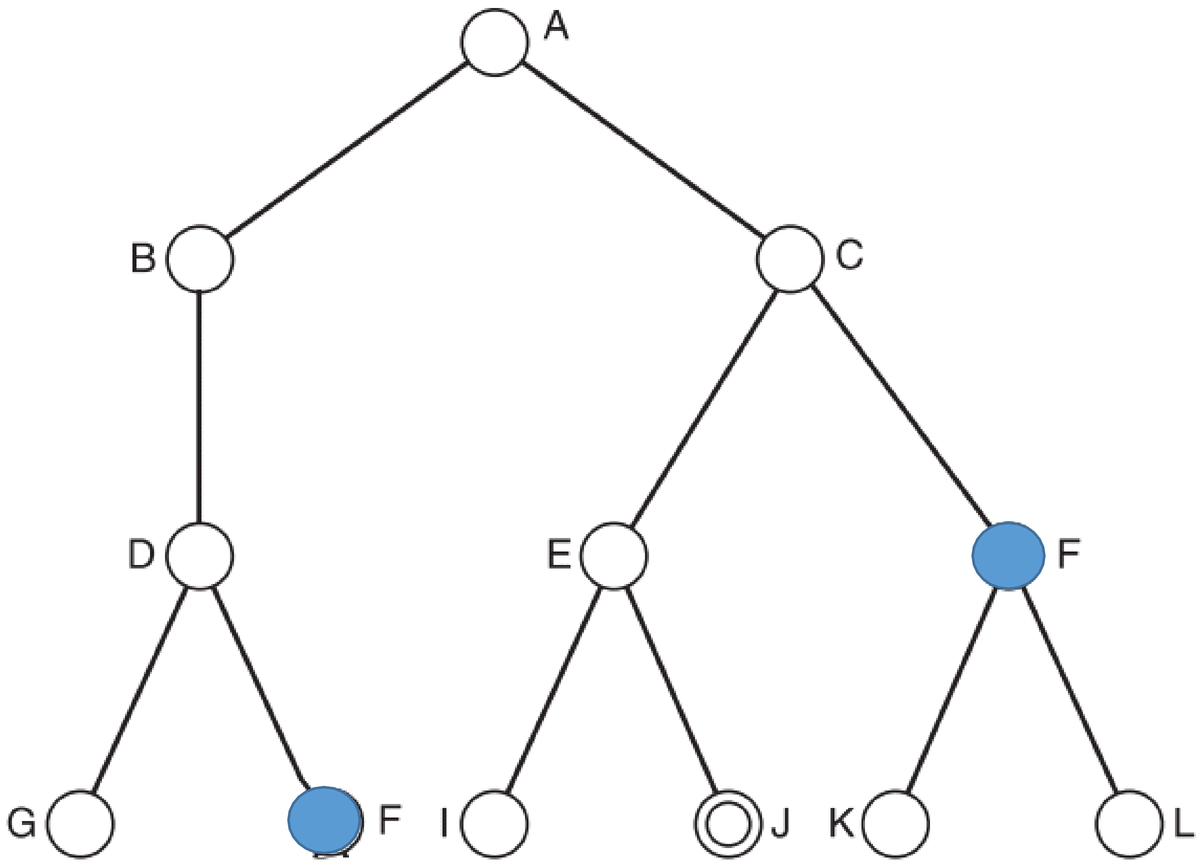
↓

Heuristic Value

Small Heuristic Value

***h*(C)**

Uninformed Search Vs. Informed Search OR Heuristically Informed Search



Heuristic Evaluation

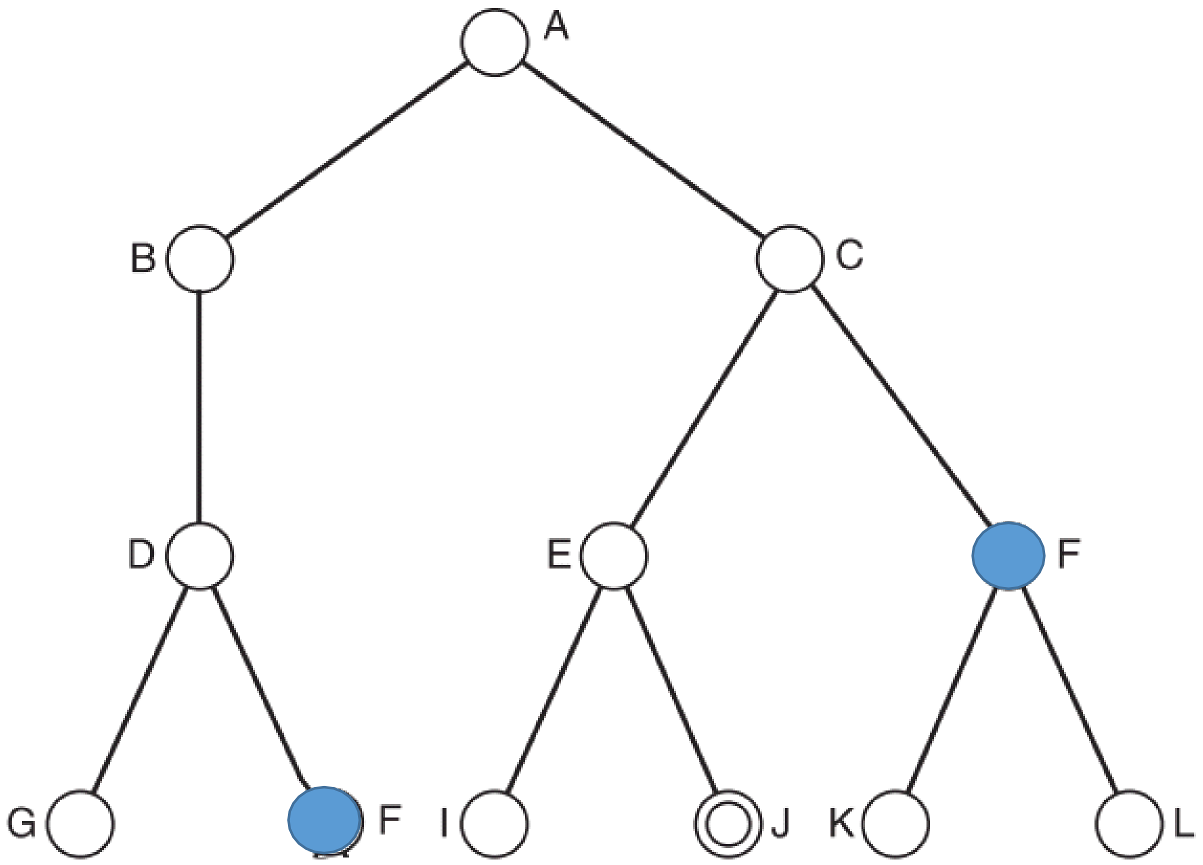
↓

Heuristic Value

Small Heuristic Value

***h*(C)**　　**_h_(B)**

Uninformed Search Vs. Informed Search OR Heuristically Informed Search



Heuristic Evaluation

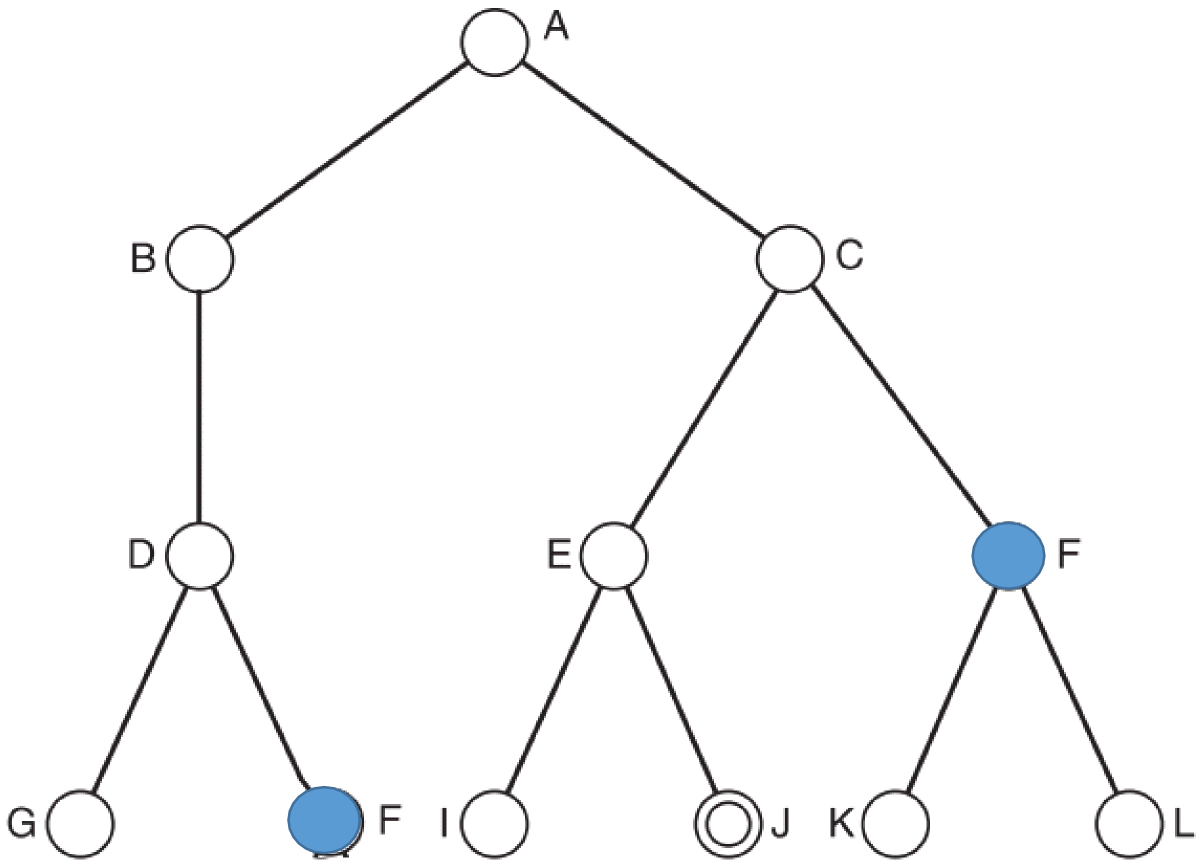
↓

Heuristic Value

Small Heuristic Value

***h*(C)** **<** ***h*(B)**

Uninformed Search Vs. Informed Search OR Heuristically Informed Search



Heuristic Evaluation

Heuristic Value

Small Heuristic Value

C < ***h*(B)**

Uninformed Search Vs. Informed Search OR Heuristically Informed Search

Uninformed Search
NO INFORMATION
Direct Search BEST PATH

Heuristic Evaluation

Heuristic Value

Small Heuristic Value

C **<** ***h*(B)**

Heuristic Evaluation Function

- Good heuristic evaluation function is what directs search to reach goal with the **smallest number of nodes**.

HeuristicFunction h_1

- Good heuristic evaluation function is **not time consuming in** the heuristic value calculation.

Heuristic Evaluation Function

- Good heuristic evaluation function is what directs search to reach goal with the **smallest number of nodes**.

Heuristic Function h_1 h_2

- Good heuristic evaluation function is **not time consuming** in the heuristic value calculation.

Heuristic Evaluation Function

- Good heuristic evaluation function is what directs search to reach goal with the **smallest number of nodes**.

Heuristic Function	h_1	h_2
# of Nodes		

- Good heuristic evaluation function is **not time consuming** in the heuristic value calculation.

Heuristic Evaluation Function

- Good heuristic evaluation function is what directs search to reach goal with the **smallest number of nodes**.

Heuristic Function	h_1	h_2
# of Nodes	5	

- Good heuristic evaluation function is **not time consuming** in the heuristic value calculation.

Heuristic Evaluation Function

- Good heuristic evaluation function is what directs search to reach goal with the **smallest number of nodes**.

Heuristic Function	h_1	h_2
# of Nodes	*5*	*7*

- Good heuristic evaluation function is **not time consuming** in the heuristic value calculation.

Heuristic Evaluation Function

- Good heuristic evaluation function is what directs search to reach goal with the **smallest number of nodes**.

Heuristic Function	h_1	h_2
# of Nodes	5	7

- Good heuristic evaluation function is **not time consuming** in the heuristic value calculation.

Heuristic Evaluation Function

- Good heuristic evaluation function is what directs search to reach goal with the **smallest number of nodes**.

Heuristic Function	h_1	h_2
# of Nodes	5	7

- Good heuristic evaluation function is **not time consuming** in the heuristic value calculation.

Heuristic Evaluation Function

- Good heuristic evaluation function is what directs search to reach goal with the **smallest number of nodes**.

Heuristic Function	h_1	h_2
# *of Nodes*	5	7

- Good heuristic evaluation function is **not time consuming** in the heuristic value calculation.

Heuristic Function	h_1

Heuristic Evaluation Function

- Good heuristic evaluation function is what directs search to reach goal with the **smallest number of nodes**.

Heuristic Function	h_1	h_2
# of Nodes	5	7

- Good heuristic evaluation function is **not time consuming** in the heuristic value calculation.

Heuristic Function	h_1	h_2
Time		

Heuristic Evaluation Function

- Good heuristic evaluation function is what directs search to reach goal with the **smallest number of nodes**.

Heuristic Function	h_1	h_2
# of Nodes	5	7

- Good heuristic evaluation function is **not time consuming** in the heuristic value calculation.

Heuristic Function	h_1	h_2
Time	60s	

Heuristic Evaluation Function

- Good heuristic evaluation function is what directs search to reach goal with the **smallest number of nodes**.

Heuristic Function	h_1	h_2
# of Nodes	5	7

- Good heuristic evaluation function is **not time consuming** in the heuristic value calculation.

Heuristic Function	h_1	h_2
Time	60s	10s

Heuristic Evaluation Function

- Good heuristic evaluation function is what directs search to reach goal with the **smallest number of nodes**.

Heuristic Function	h_1	h_2
# of Nodes	5	7

- Good heuristic evaluation function is **not time consuming** in the heuristic value calculation.

Heuristic Function	h_1	h_2
Time	60s	10s

Heuristic Evaluation Function

- Good heuristic evaluation function is what directs search to reach goal with the **smallest number of nodes**.

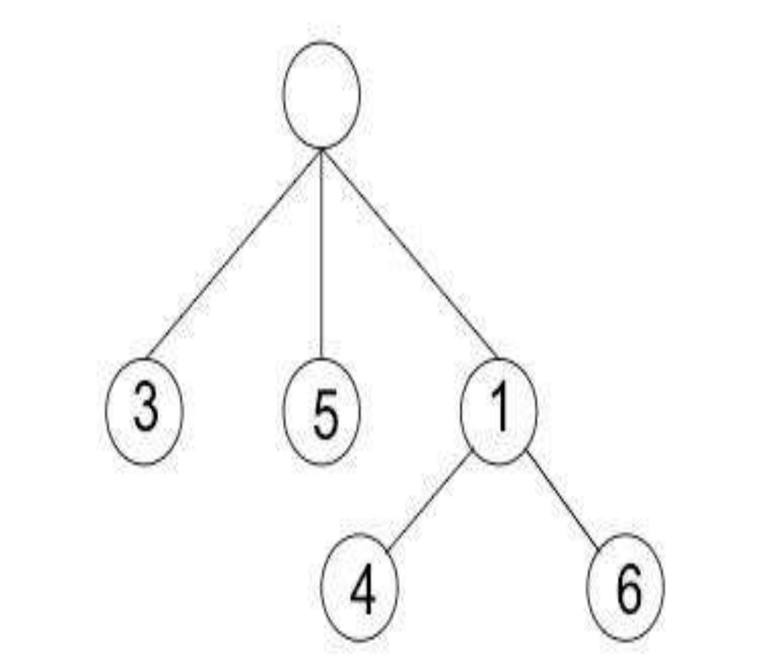
Heuristic Function	h_1	h_2
# of Nodes	*5*	*7*

- Good heuristic evaluation function is **not time consuming** in the heuristic value calculation.

Heuristic Function	h_1	h_2
Time	*60s*	*10s*

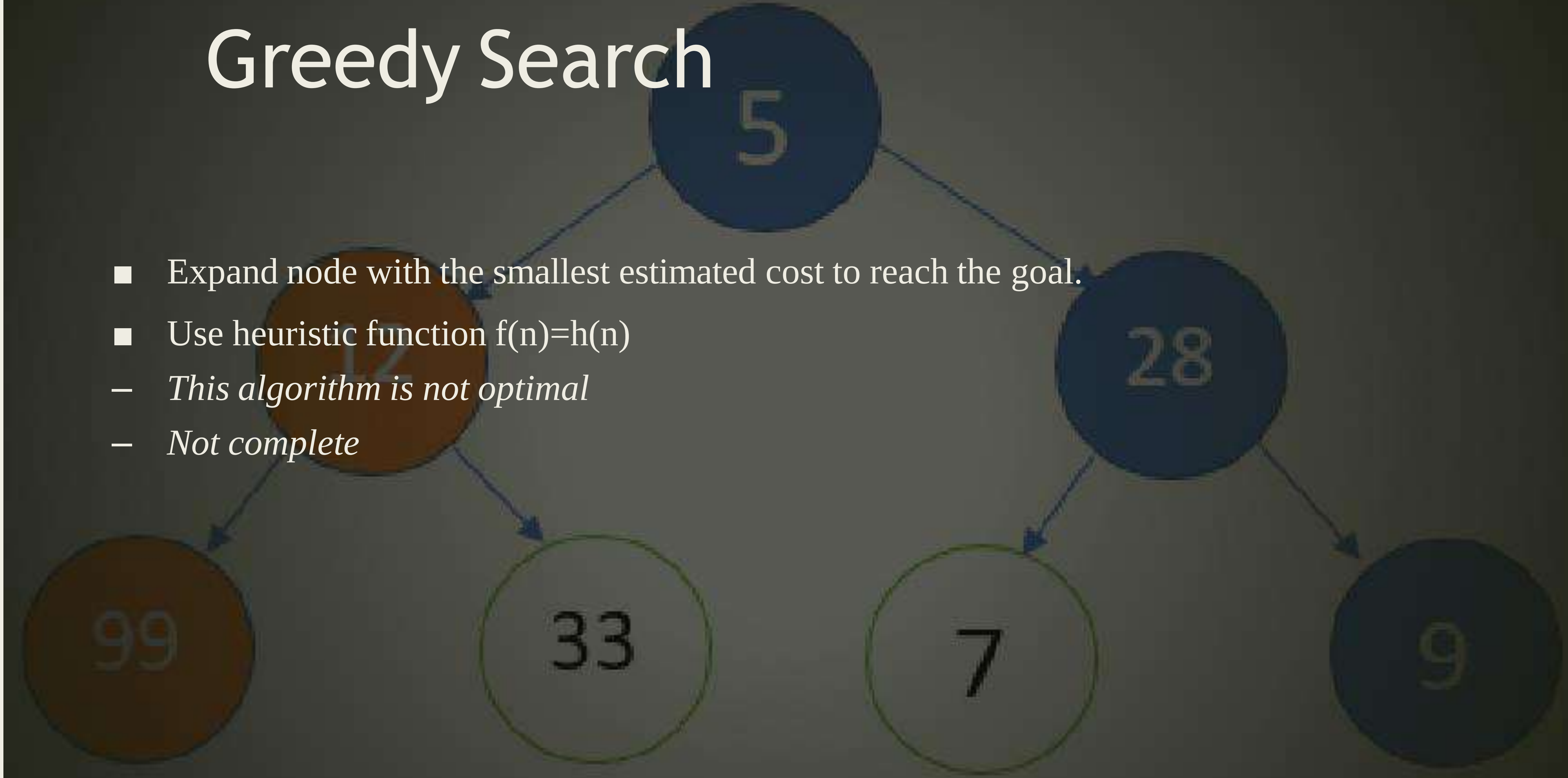
Best First Search

- The generic ***best-first search*** algorithm selects a node for expansion according to an evaluation function.
- It is a generalization of breadth first search.
- Priority queue of nodes to be explored.
- Cost function ***f(n)*** to applied to each node.
- Always choose the node from the frontier that has lowest ***f(n)*** value.



Greedy Search

- Expand node with the smallest estimated cost to reach the goal.
- Use heuristic function f(n)=h(n)
  - *This algorithm is not optimal*
  - *Not complete*



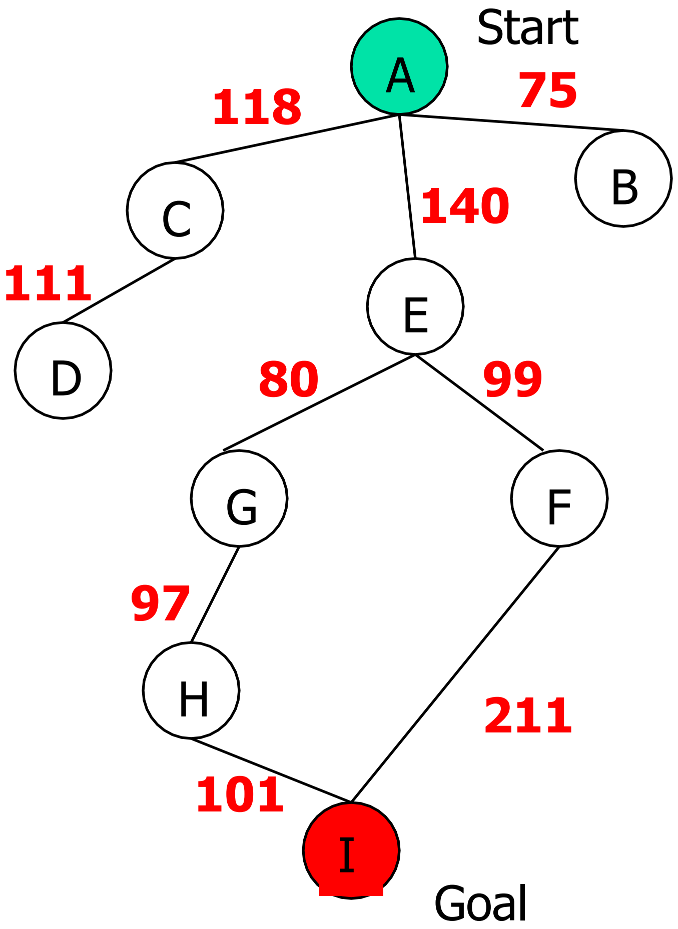
Continued….

- Greedy best-first search tries to expand the node that is closest to the goal, on the grounds that this is likely to lead to a solution quickly
- Thus, the evaluation function is $f(n) = h(n)$
- E.g. in minimizing road distances a heuristic lower bound for distances of cities is their straight-line distance
- Greedy search ignores the cost of the path that has already been traversed to reach n
- Therefore, the solution given is not necessarily optimal
- If repeating states are not detected, greedy best-first search may oscillate forever between two promising states.

Continued..

- Because greedy best-first search can start down an **infinite path** and never return to try other possibilities, **it is incomplete**
- **Because of its greediness** the search makes choices that can lead to a **dead end**; then one backs up in the search tree to the deepest unexpanded node
- Greedy best-first search resembles depth-first search in the way it prefers to follow a single path all the way to the goal, but will back up when it hits a dead end
- The worst-case time and space complexity is **$O(b^m)$**
- The **quality of the heuristic function** determines the practical **usability of greedy search.**

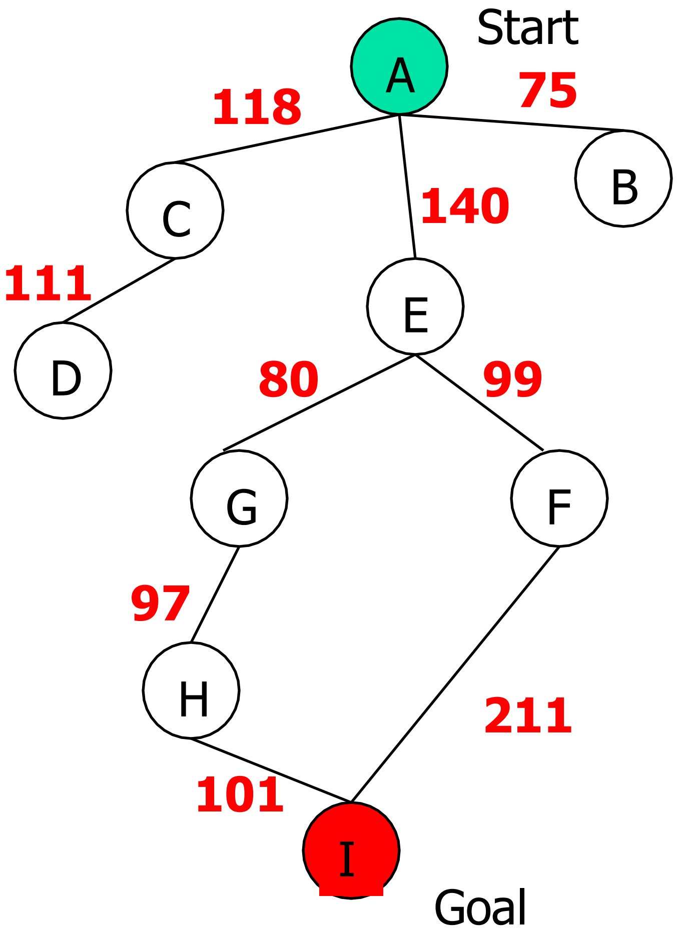
Continue…



State	Heuristic: h(n)
A	366
B	374
C	329
D	244
E	253
F	178
G	193
H	98
I	0

$f(n) = h(n)$ = straight-line distance heuristic

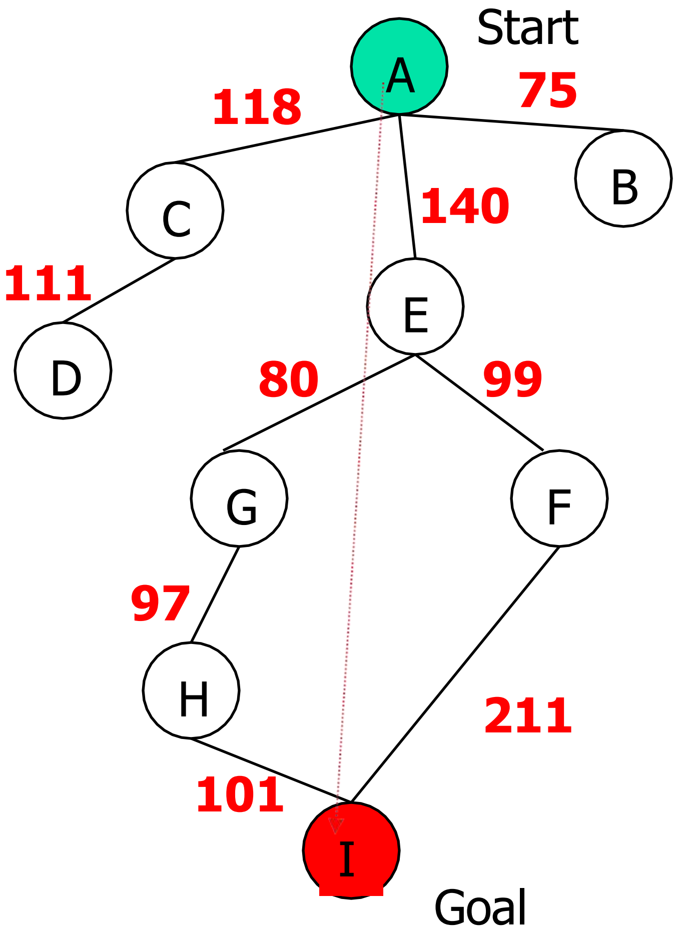
Continue…



State	Heuristic: h(n)
A	366
B	374
C	329
D	244
E	253
F	178
G	193
H	98
I	0

***f(n) = h(n)* = straight-line distance heuristic**

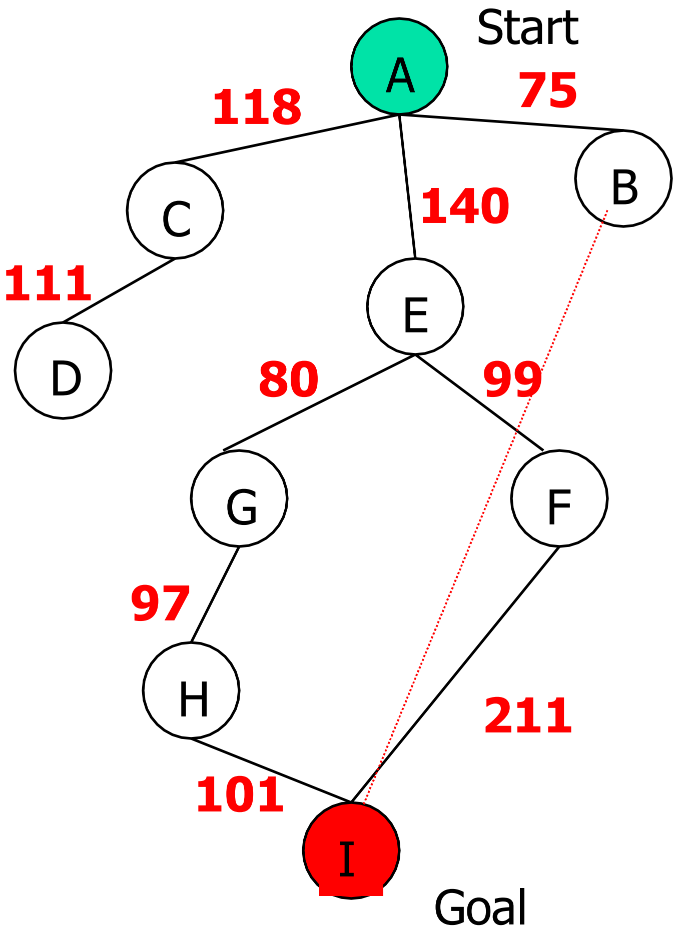
Continue...



State	Heuristic: h(n)
A	**366**
B	374
C	329
D	244
E	253
F	178
G	193
H	98
I	0

***f(n) = h(n)* = straight-line distance heuristic**

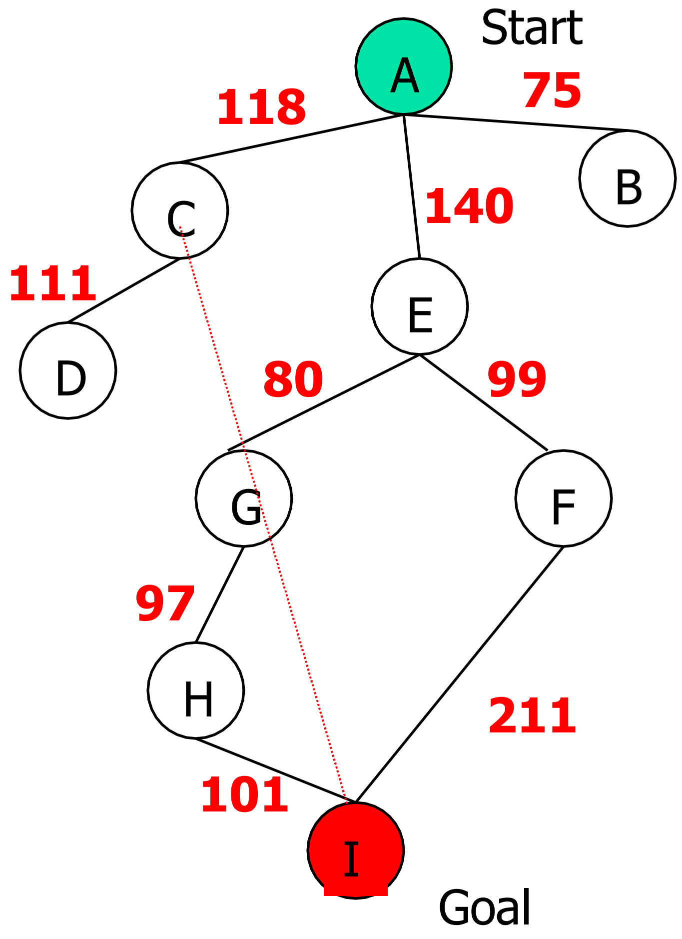
Continue..



State	Heuristic: h(n)
A	366
B	**374**
C	329
D	244
E	253
F	178
G	193
H	98
I	0

***f(n)* = *h(n)* = straight-line distance heuristic**

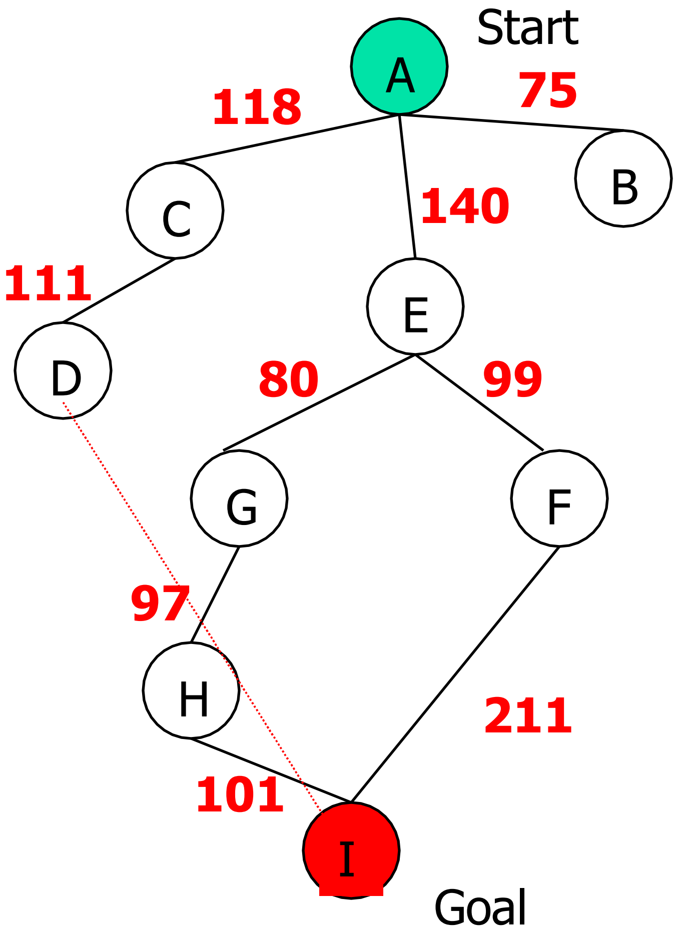
Continue..



State	Heuristic: h(n)
A	366
B	374
C	**329**
D	244
E	253
F	178
G	193
H	98
I	0

***f(n) = h(n)* = straight-line distance heuristic**

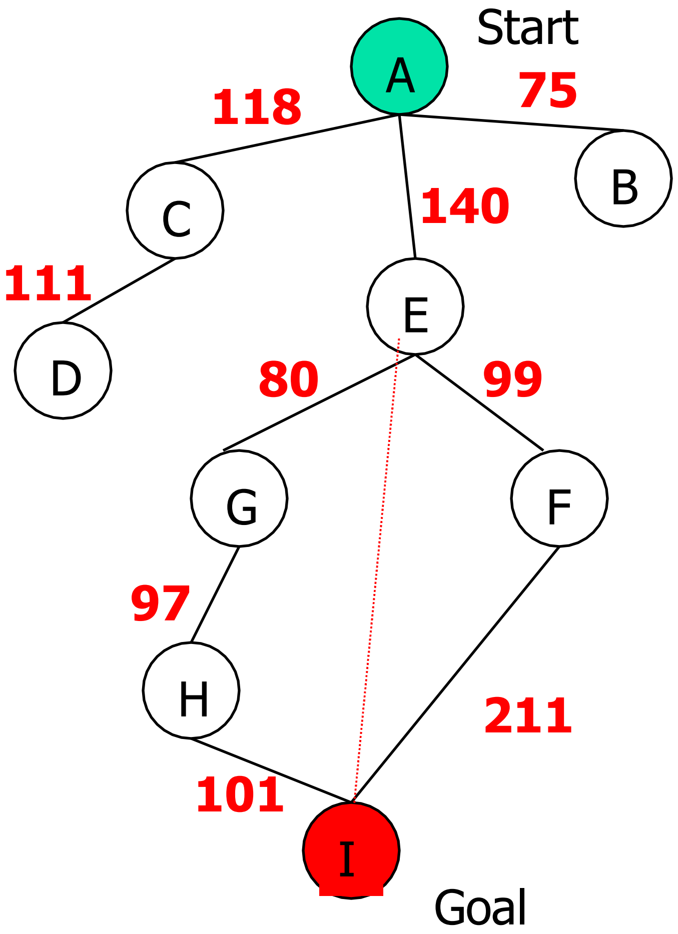
Continue…



State	Heuristic: h(n)
A	366
B	374
C	329
D	**244**
E	253
F	178
G	193
H	98
I	0

$f(n) = h(n)$ = **straight-line distance heuristic**

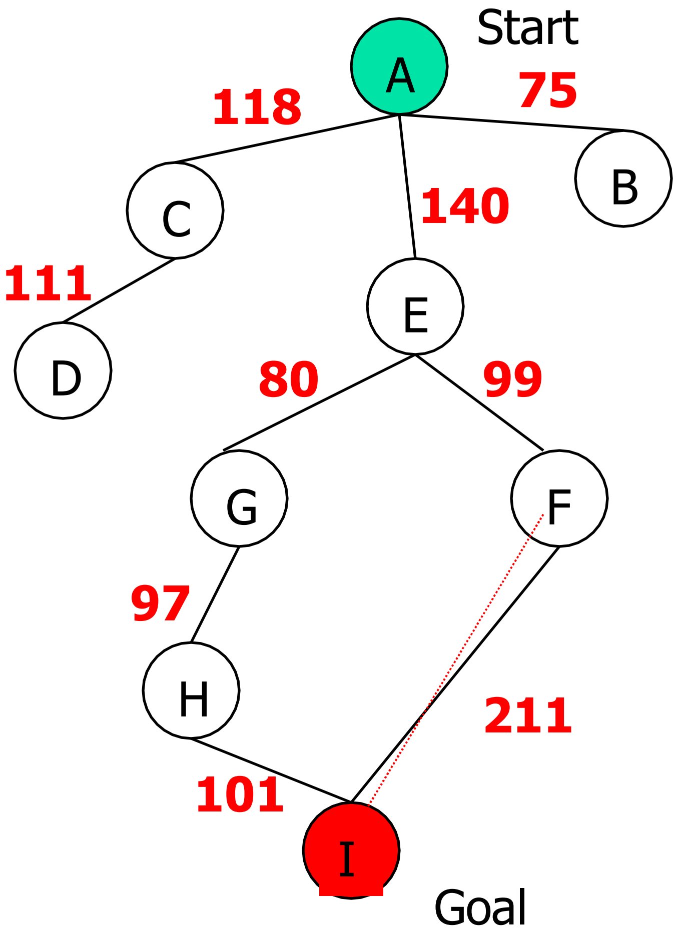
Continue…



State	Heuristic: h(n)
A	366
B	374
C	329
D	244
E	**253**
F	178
G	193
H	98
I	0

***f(n)* = *h*(*n*) = straight-line distance heuristic**

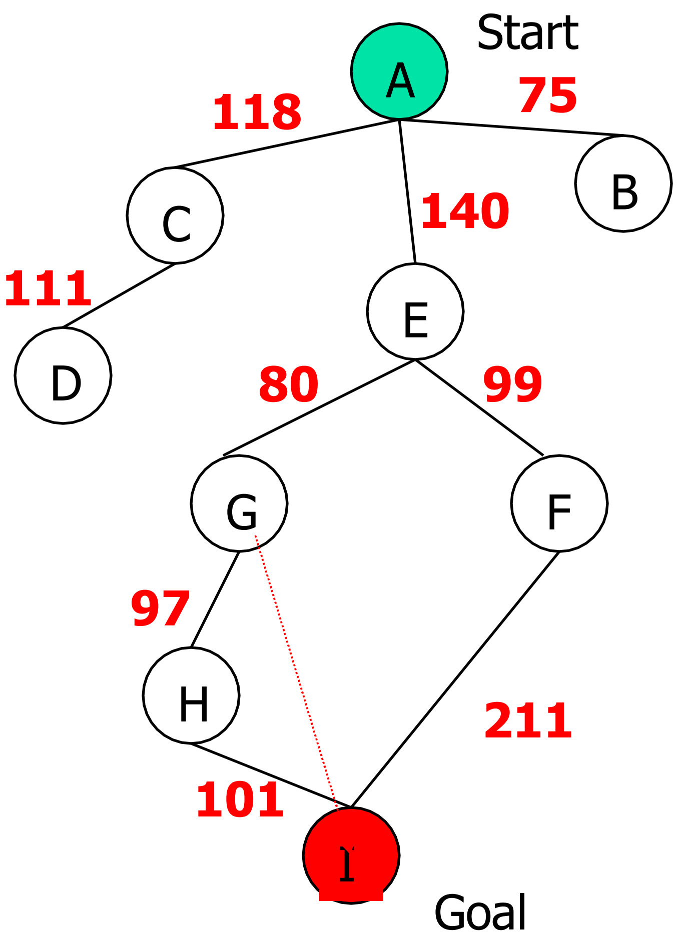
Continue…



State	Heuristic: h(n)
A	366
B	374
C	329
D	244
E	253
F	**178**
G	193
H	98
I	0

$f(n) = h(n)$ = straight-line distance heuristic

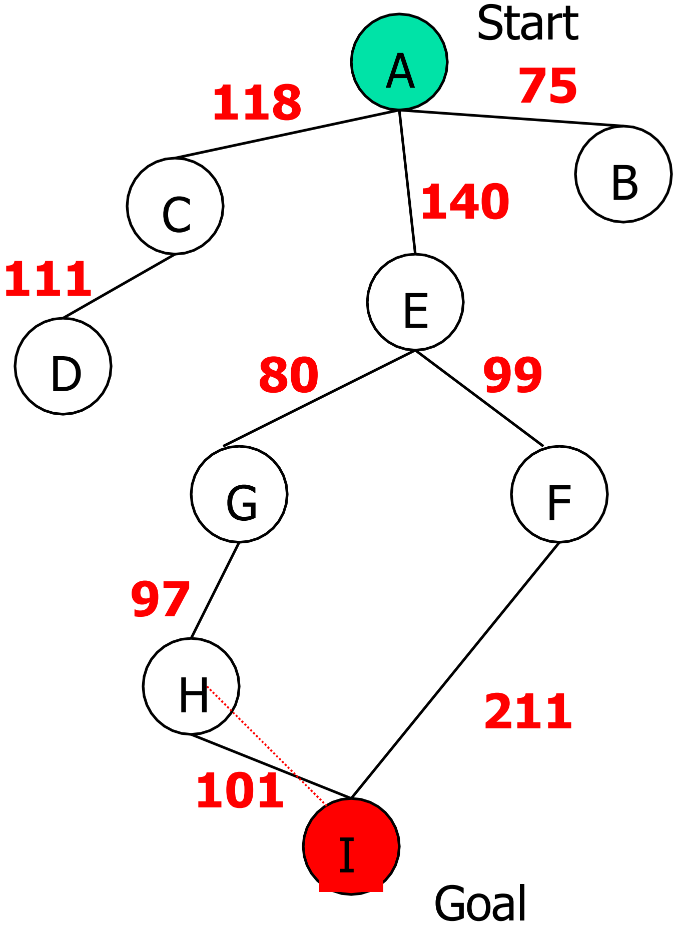
Continue…



State	Heuristic: h(n)
A	366
B	374
C	329
D	244
E	253
F	178
G	**193**
H	98
I	0

***f(n)* = *h* (*n*) = straight-line distance heuristic**

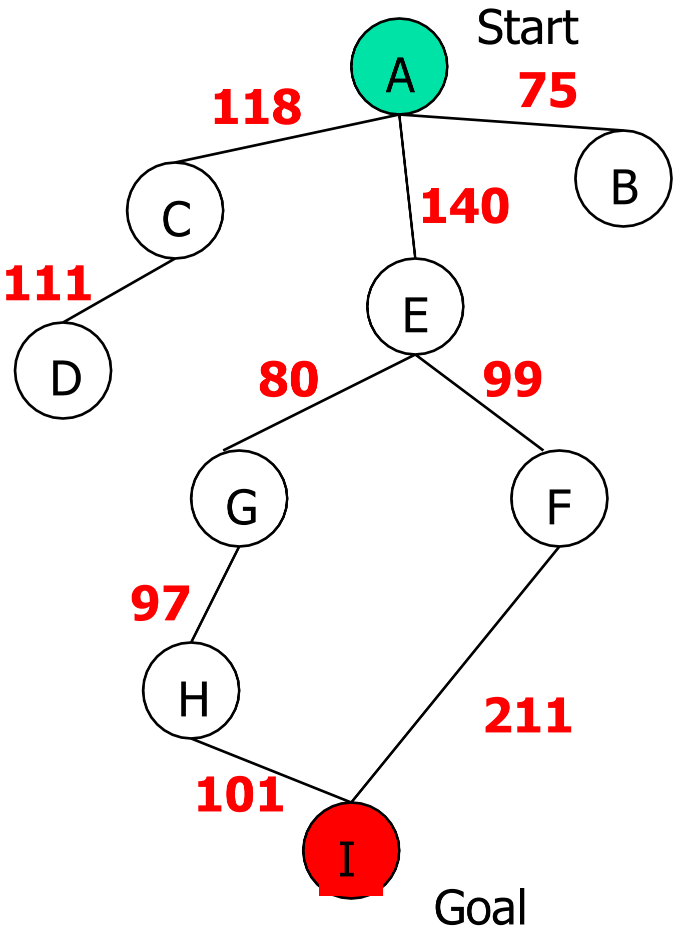
Continue...



State	Heuristic: h(n)
A	366
B	374
C	329
D	244
E	253
F	178
G	193
H	**98**
I	0

***f(n)* = *h(n)* = straight-line distance heuristic**

Continue…

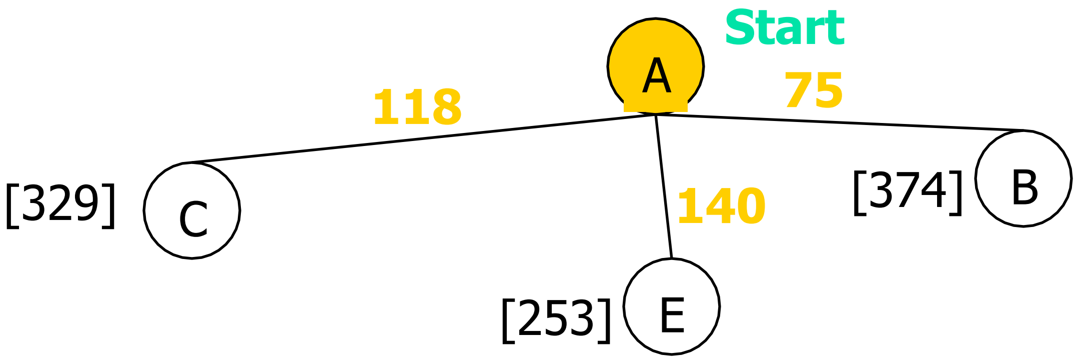


State	Heuristic: h(n)
A	366
B	374
C	329
D	244
E	253
F	178
G	193
H	98
I	**0**

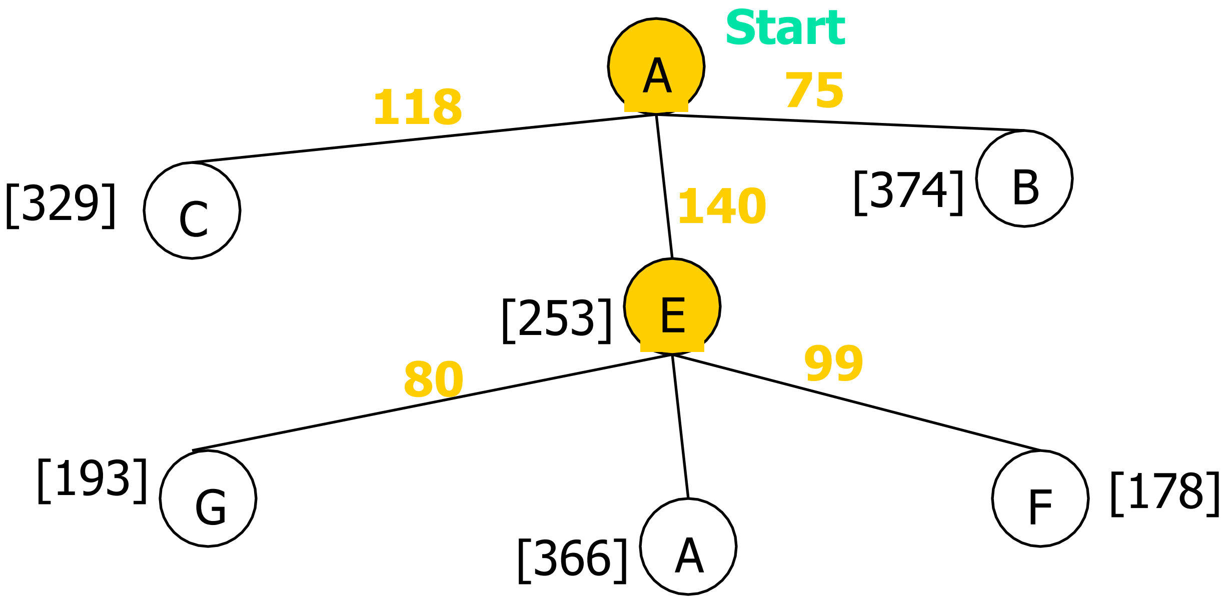
***f(n) = h(n)* = straight-line distance heuristic**

Greedy Search: Tree Search

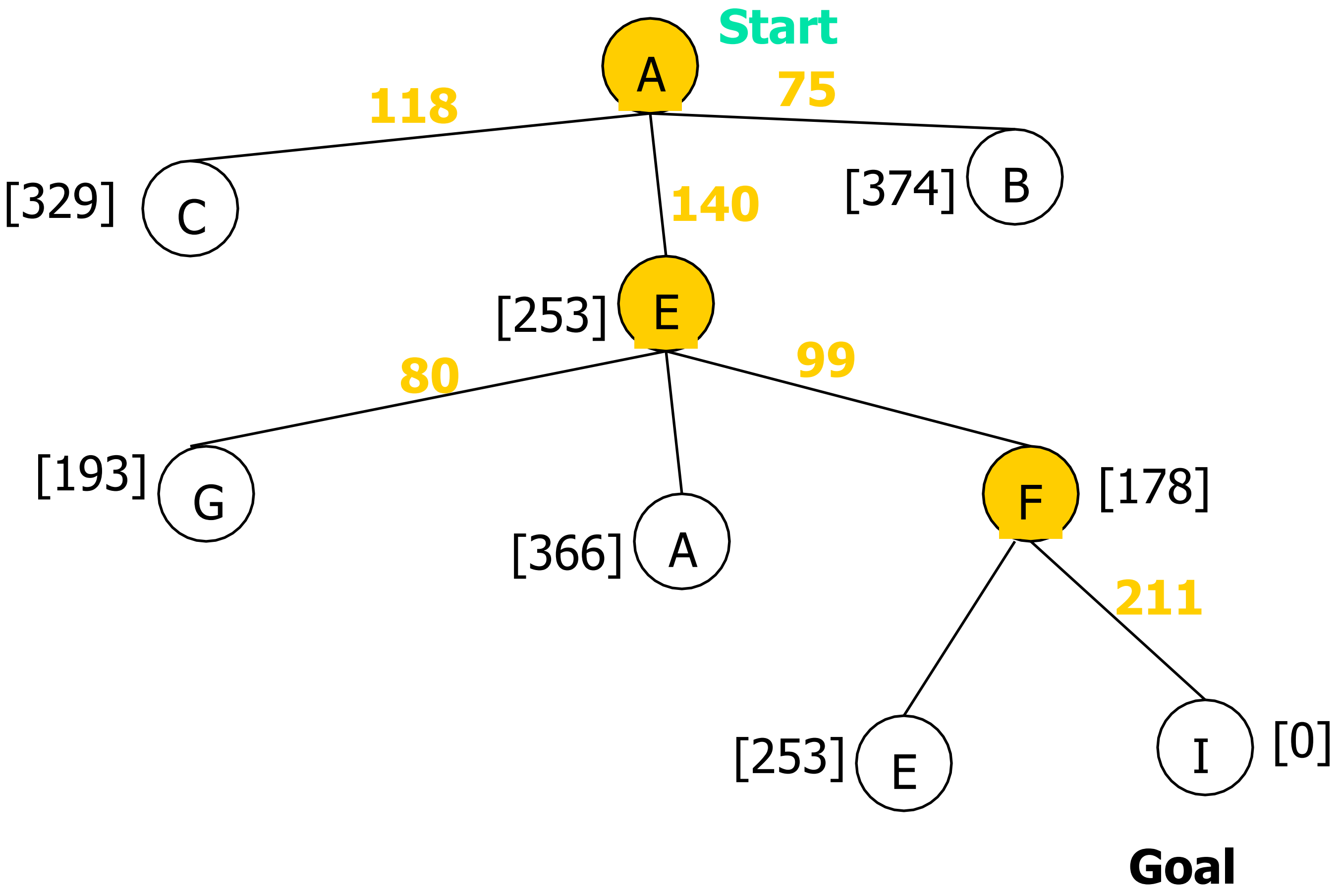
Greedy Search: Tree Search



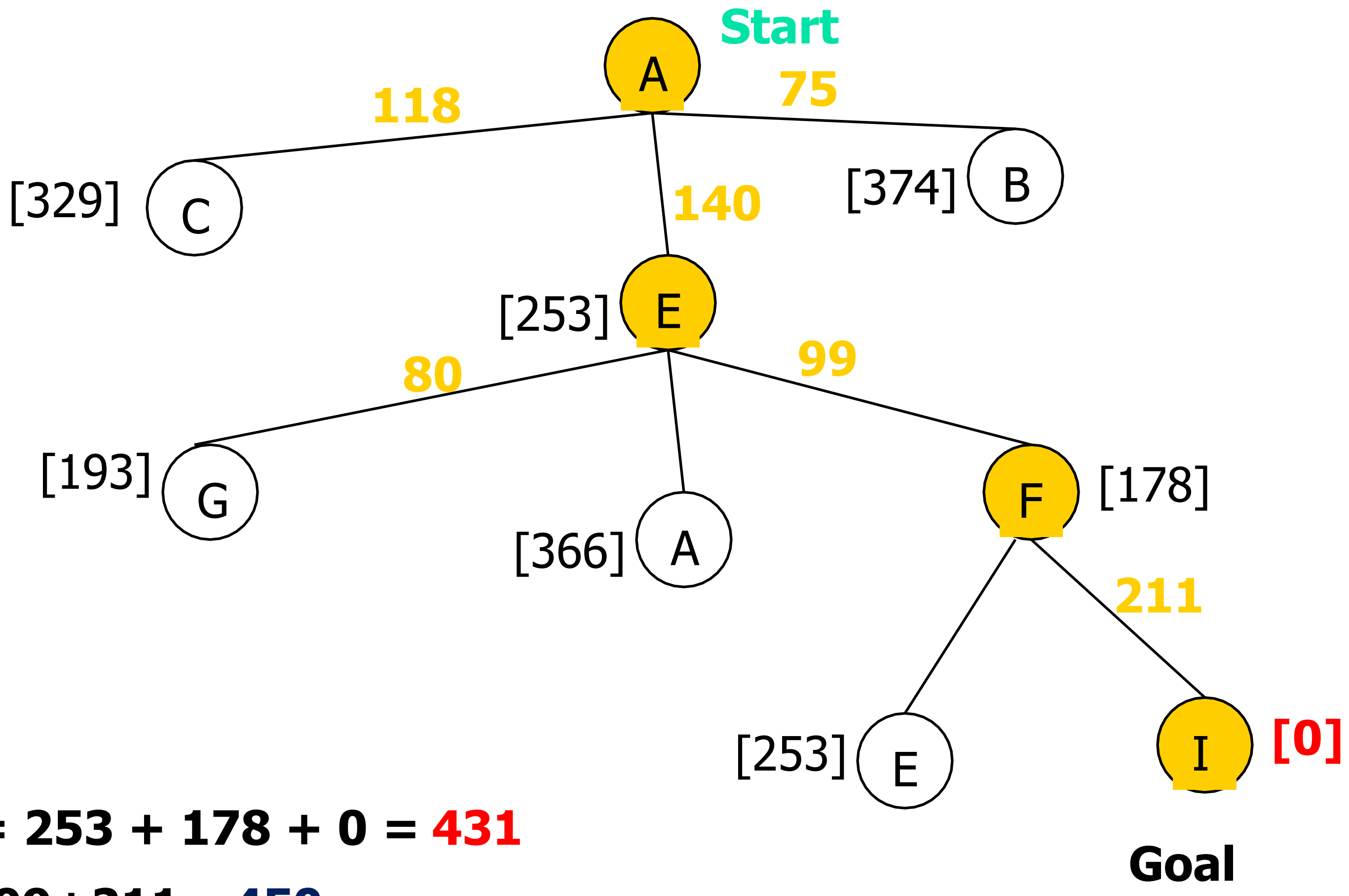
Greedy Search: Tree Search



Greedy Search: Tree Search



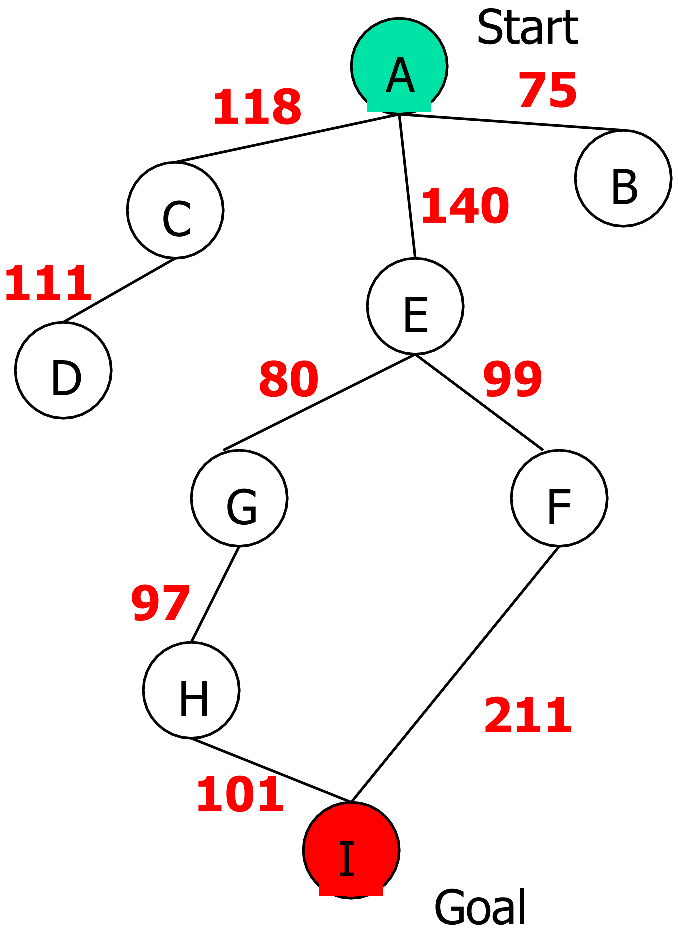
Greedy Search: Tree Search



Path cost(A-E-F-I) = 253 + 178 + 0 = 431

dist(A-E-F-I)=140+99+211= 450

Greedy Search: Complete ?

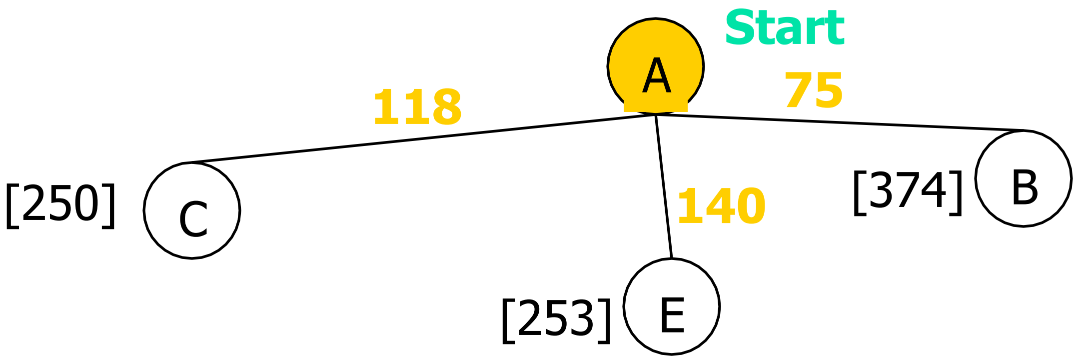


State	Heuristic: h(n)
A	366
B	374
** C	**250**
D	244
E	253
F	178
G	193
H	98
I	0

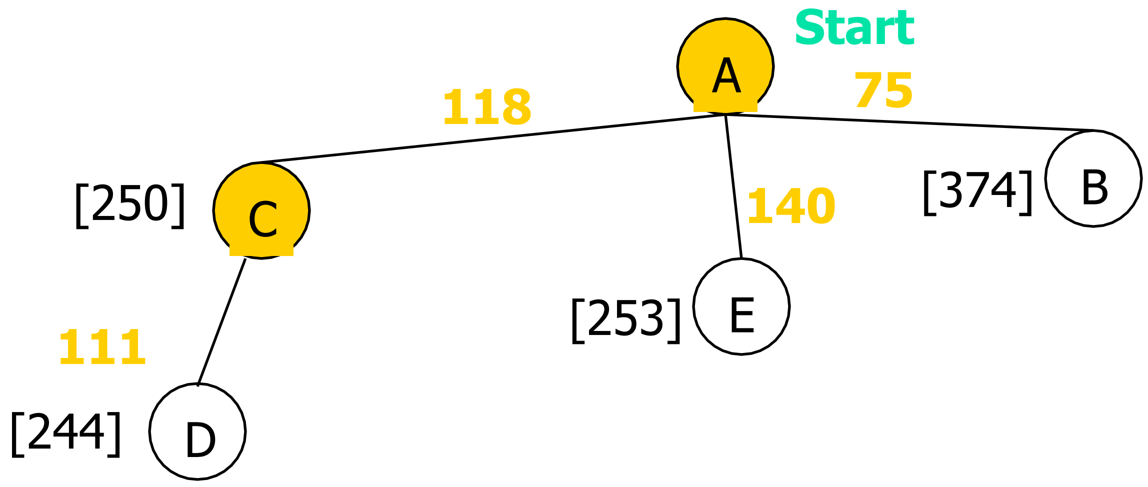
***f(n)* = *h (n)* = straight-line distance heuristic**

Greedy Search: Tree Search

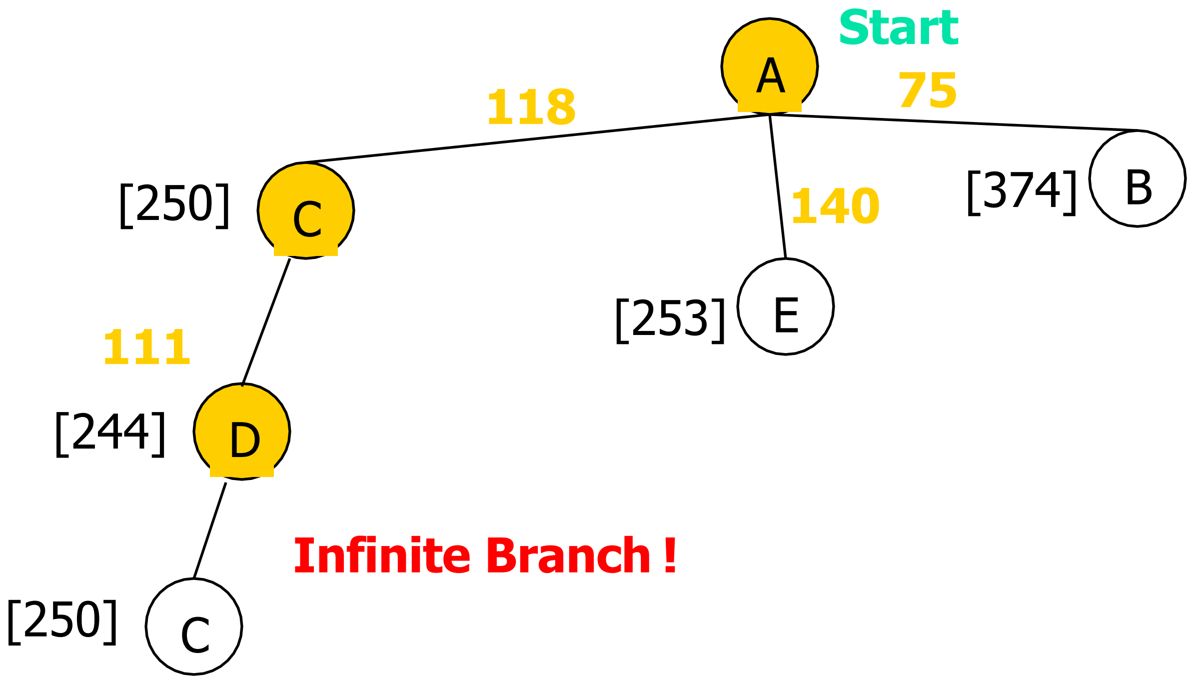
Greedy Search: Tree Search



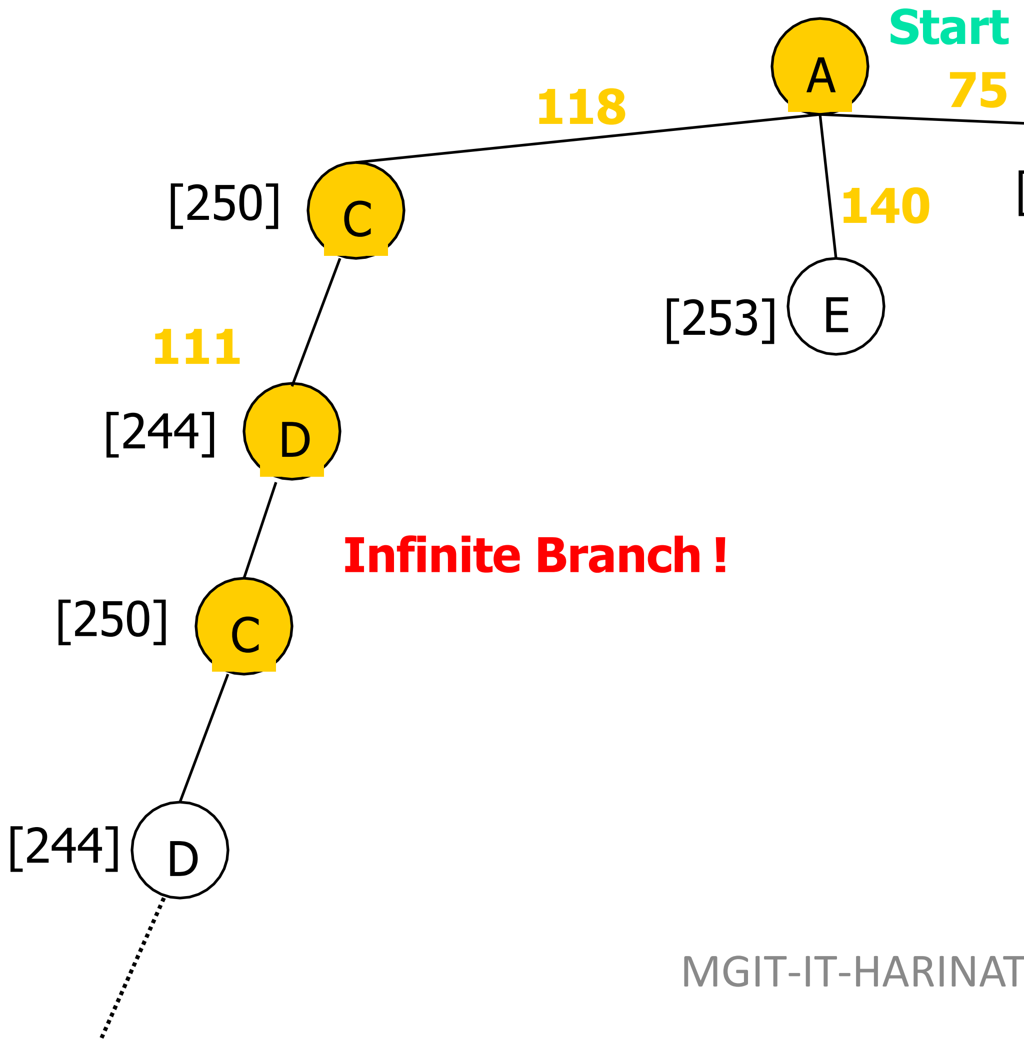
Greedy Search: Tree Search



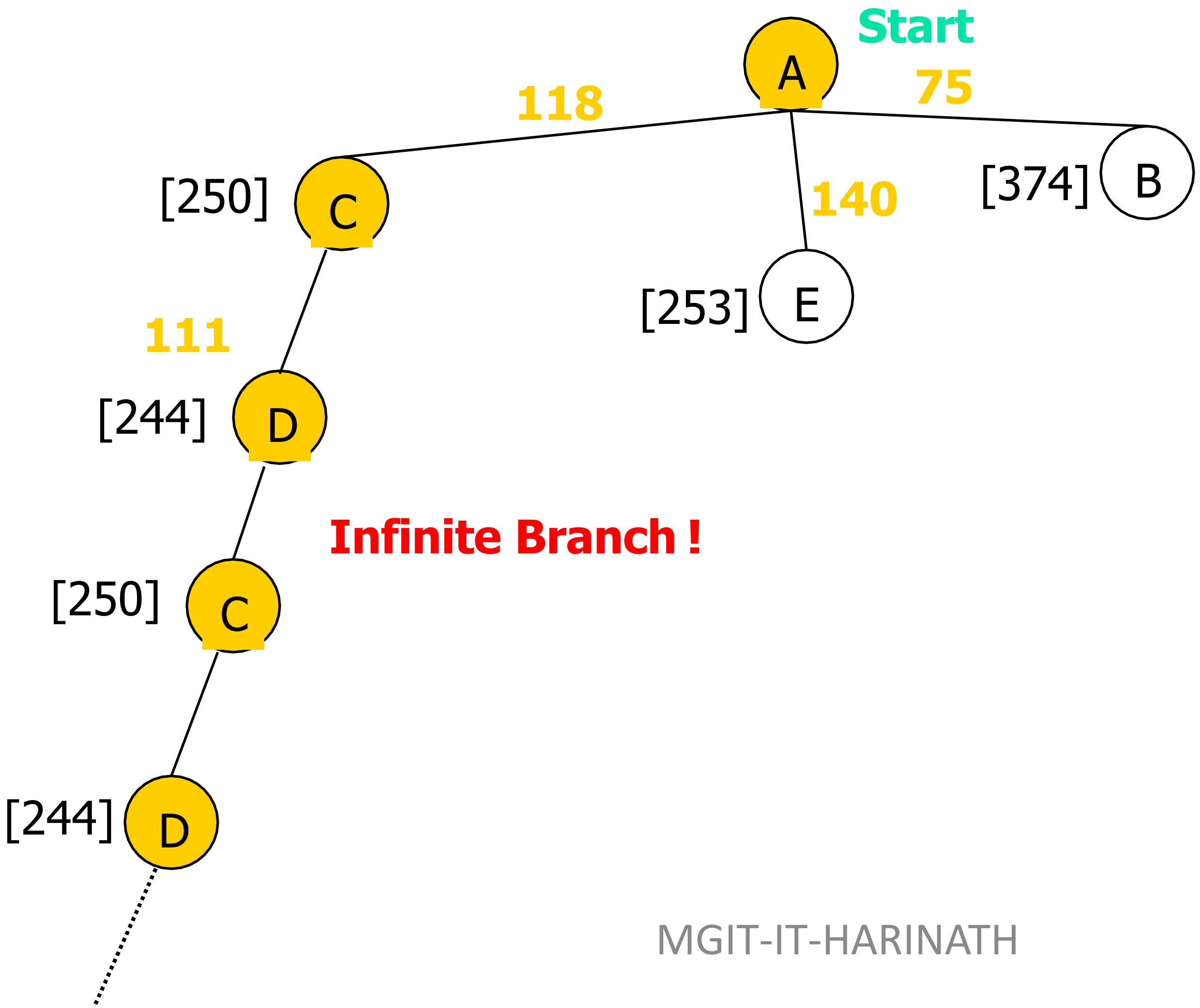
Greedy Search: Tree Search



Greedy Search: Tree Search

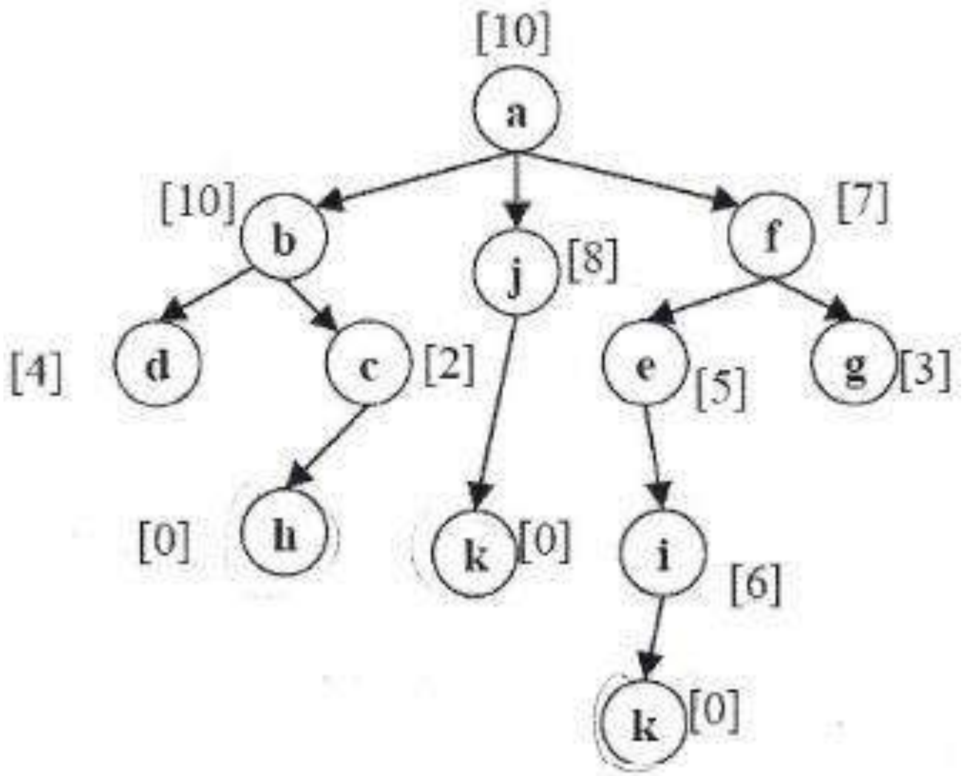


Greedy Search: Tree Search



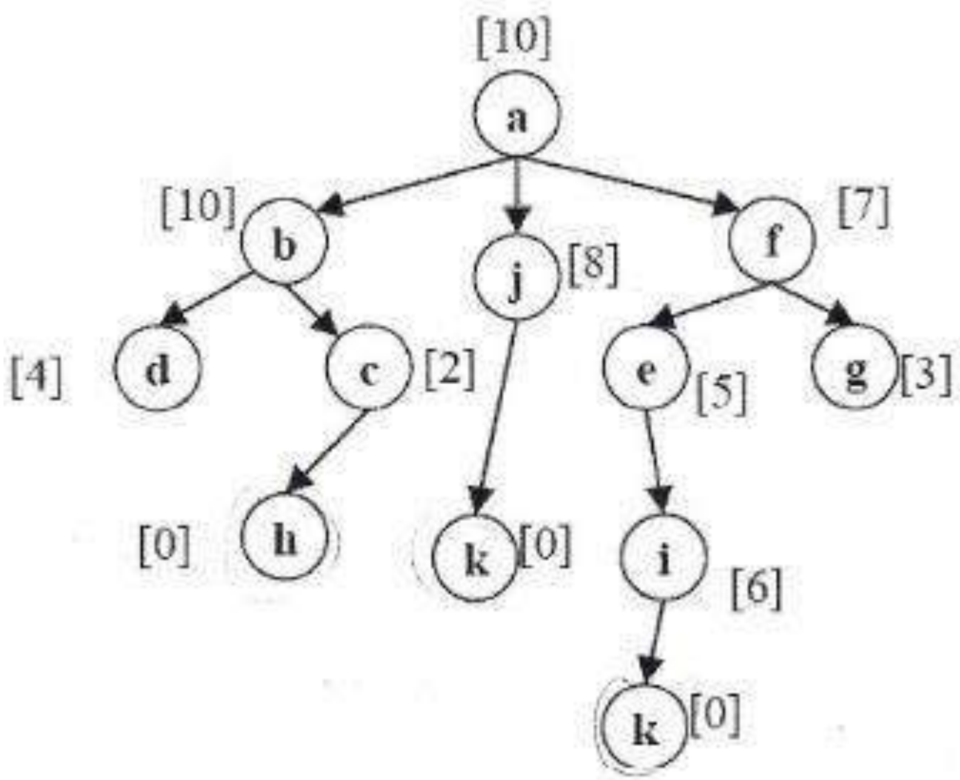
Greedy Best-First Search Goal - Node K

Current	Children
a_{10}	---



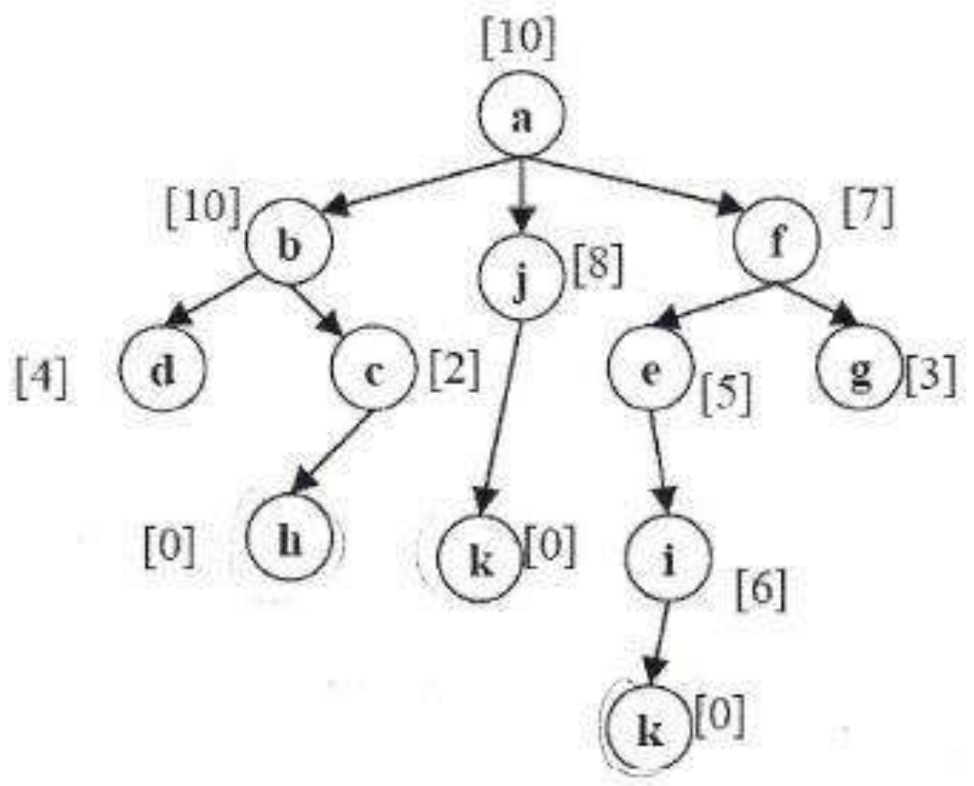
Greedy Best-First Search Goal - Node K

Current	Children
a_{10}	---

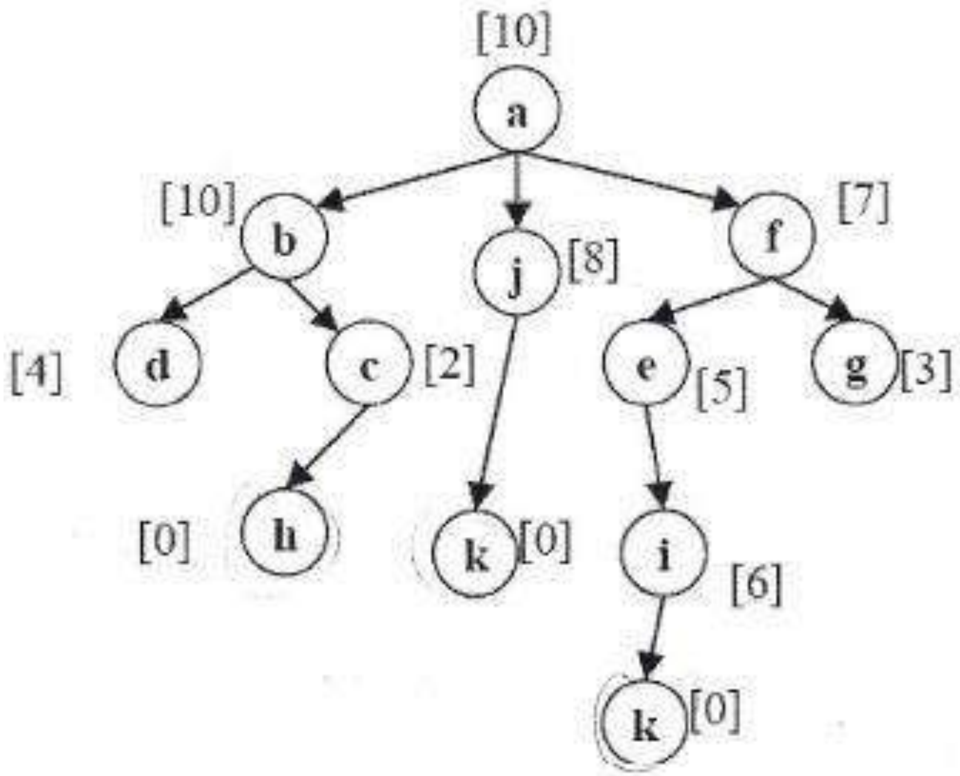


Greedy Best-First Search Goal - Node K

Current	Children
a_{10}	---
a_{10}	



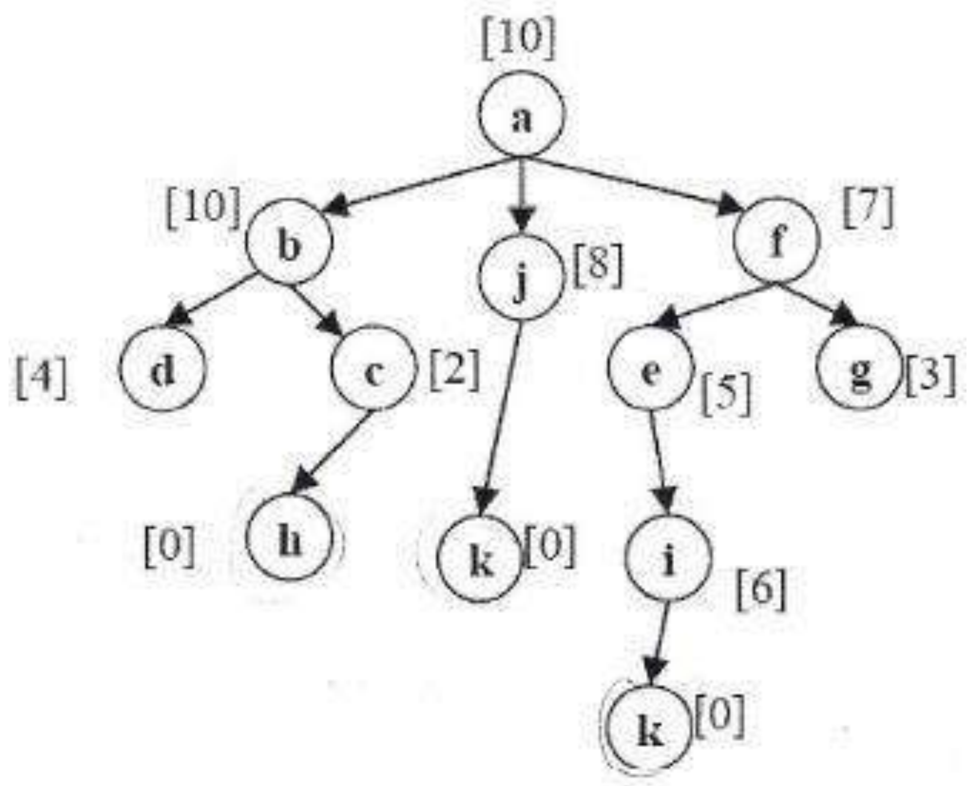
Greedy Best-First Search Goal - Node K



Current	Children
a_{10}	---
a_{10}	$f_7, j_8 b_{10}$

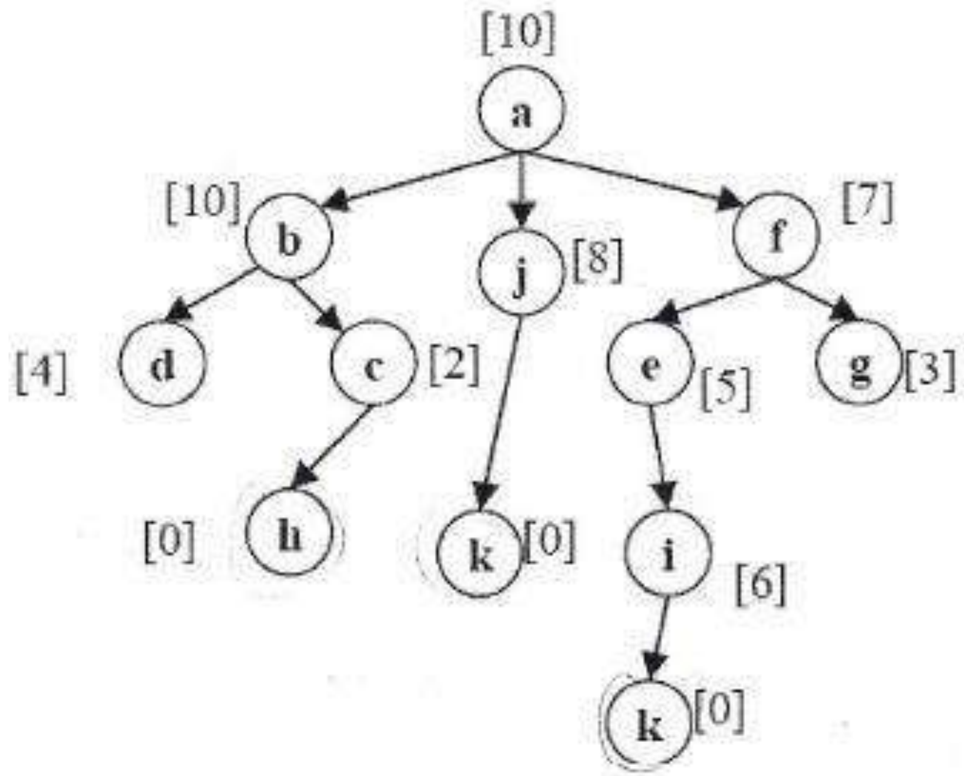
Greedy Best-First Search Goal – Node K

Current	Children
a_{10}	---
a_{10}	$f_7, j_8 b_{10}$
f_7	



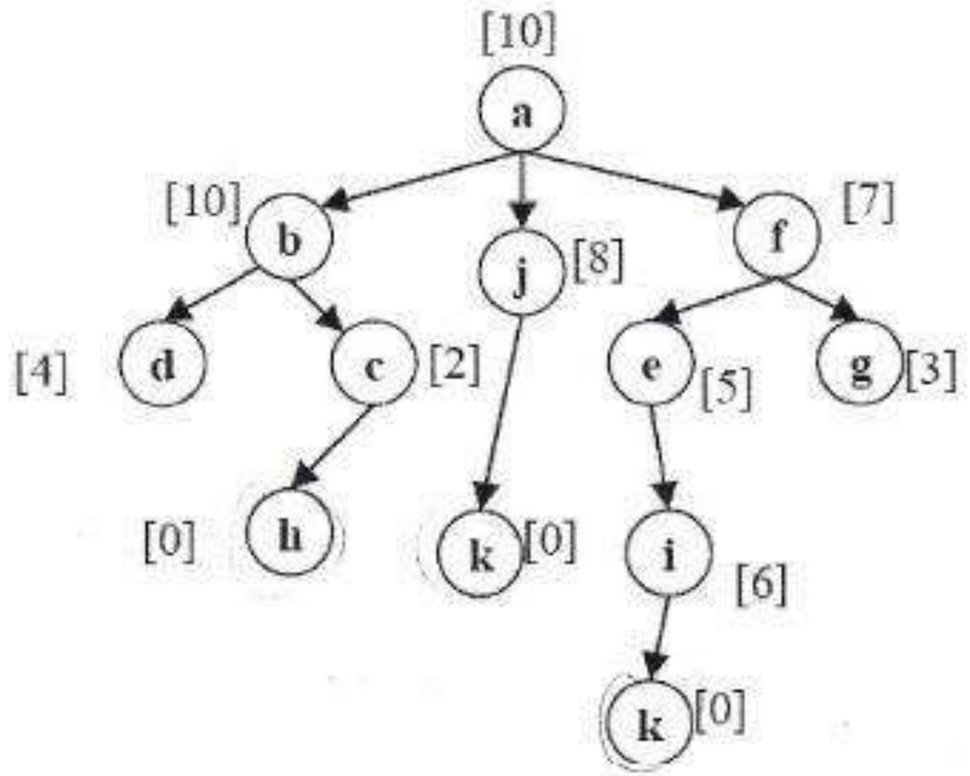
Greedy Best-First Search Goal - Node K

Current	Children
a_{10}	---
a_{10}	$f_7, j_8 b_{10}$
f_7	$j_8 b_{10}$



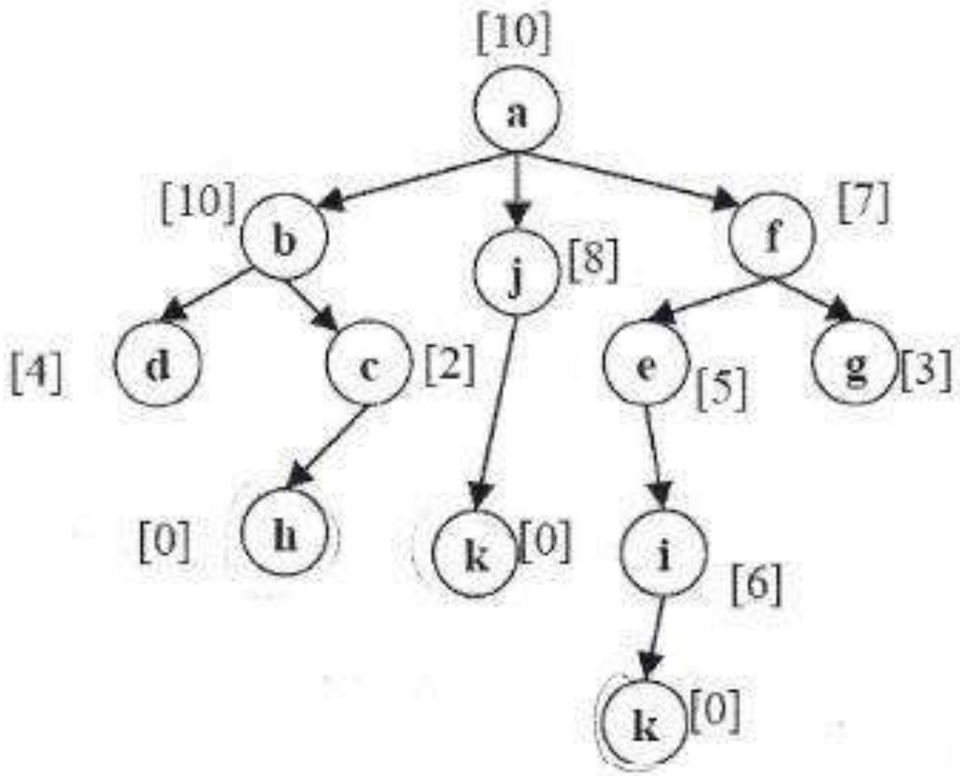
Greedy Best-First Search Goal - Node K

Current	Children
a_{10}	---
a_{10}	$f_7, j_8 b_{10}$
f_7	$j_8 b_{10}$



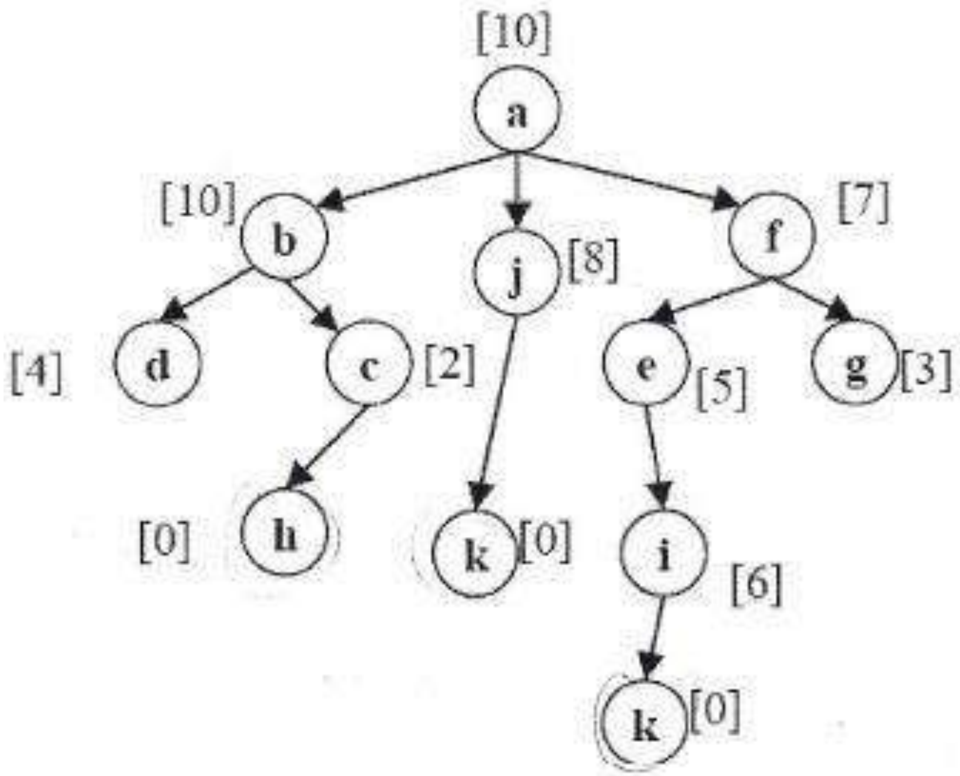
Greedy Best-First Search Goal - Node K

Current	Children
a_{10}	---
a_{10}	$f_7, j_8 b_{10}$
f_7	$j_8 b_{10}$
f_7	

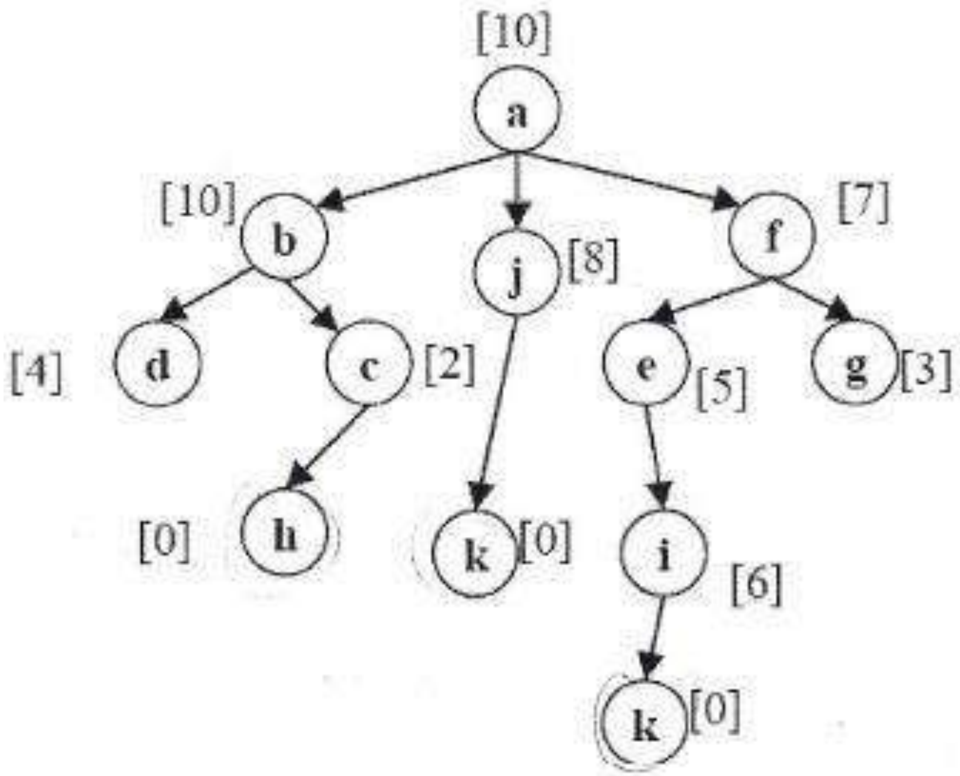


Greedy Best-First Search Goal – Node K

Current	Children
a_{10}	---
a_{10}	$f_{7}, j_{8} b_{10}$
f_{7}	$j_{8} b_{10}$
f_{7}	$g_{3}, e_{5}, j_{8} b_{10}$

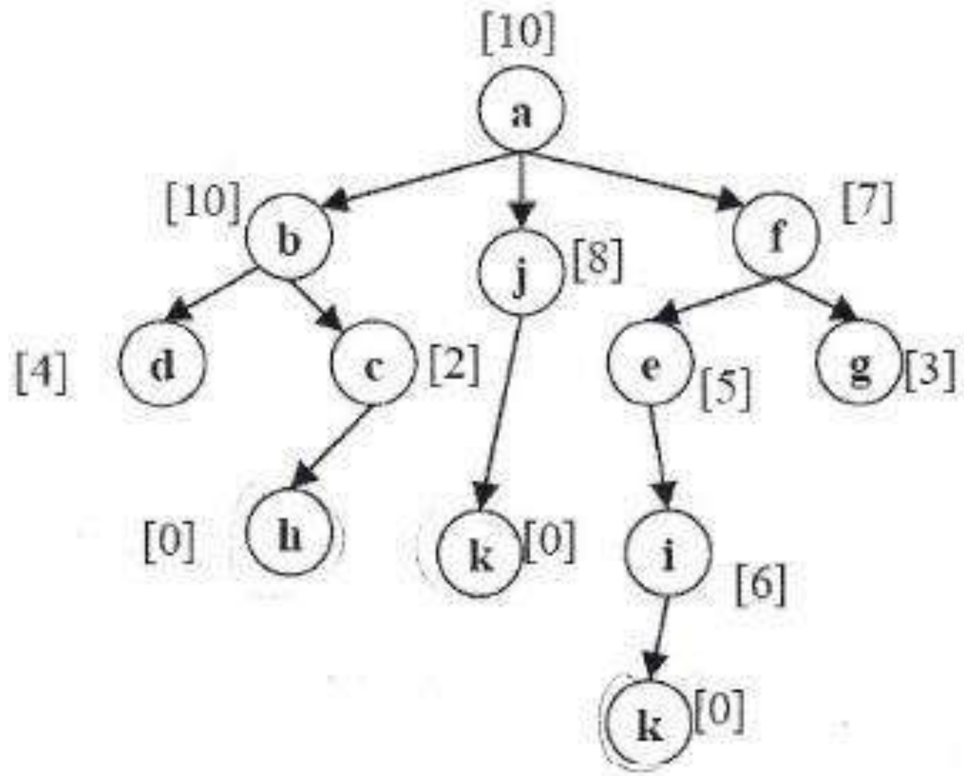


Greedy Best-First Search Goal – Node K



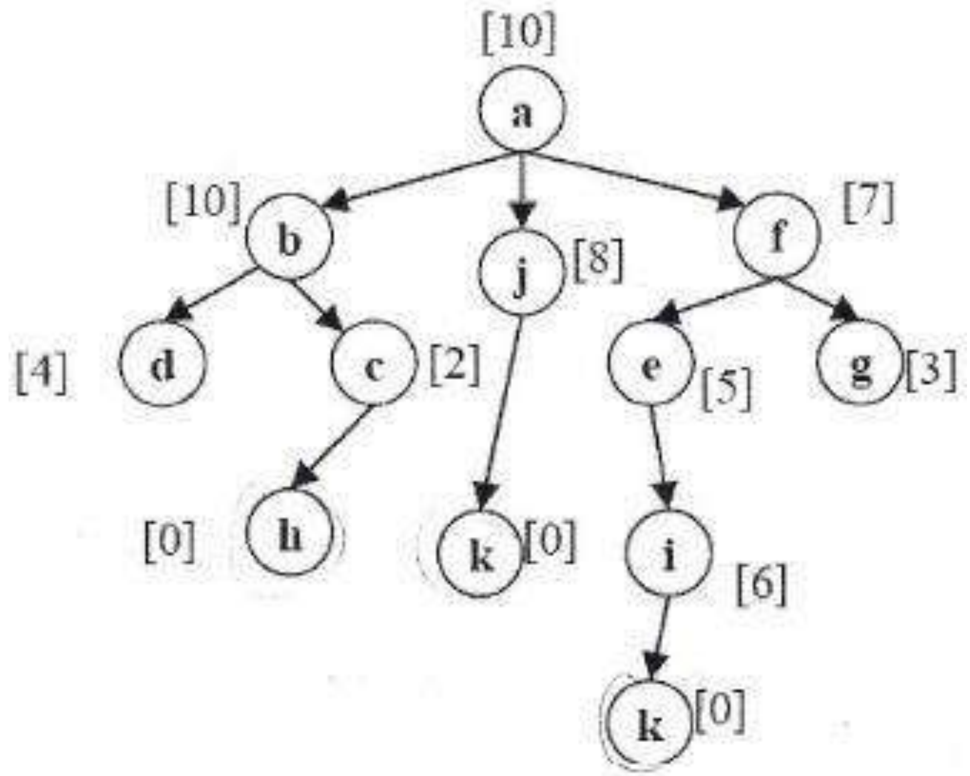
Current	Children
a_{10}	---
a_{10}	$f_7, j_8 b_{10}$
f_7	$j_8 b_{10}$
f_7	$g_3, e_5, j_8 b_{10}$
g_3	

Greedy Best-First Search Goal – Node K



Current	Children
a_{10}	---
a_{10}	f_7, j_8, b_{10}
f_7	j_8, b_{10}
f_7	g_3, e_5, j_8, b_{10}
g_3	e_5, j_8, b_{10}

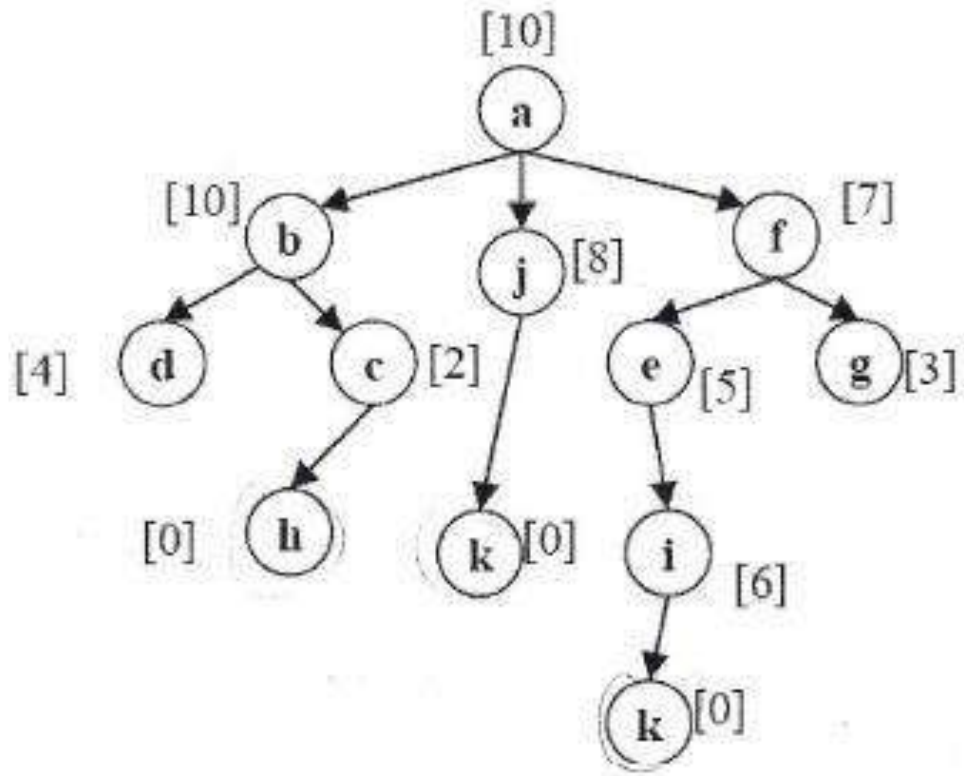
Greedy Best-First Search Goal - Node K



Current	Children
a_{10}	---
a_{10}	$f_7, j_8 b_{10}$
f_7	$j_8 b_{10}$
f_7	$g_3, e_5, j_8 b_{10}$
g_3	$e_5, j_8 b_{10}$
e_5	

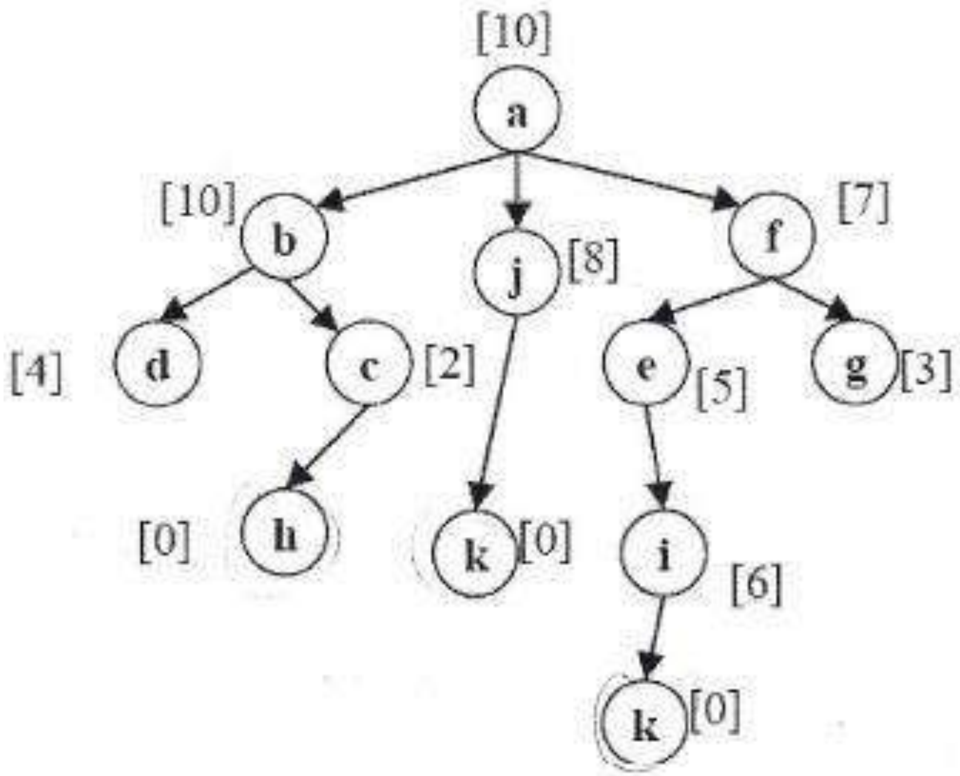
Greedy Best-First Search Goal - Node K

Current	Children
a_{10}	---
a_{10}	f_7, j_8, b_{10}
f_7	$j_8 b_{10}$
f_7	$g_3, e_5, j_8 b_{10}$
g_3	$e_5, j_8 b_{10}$

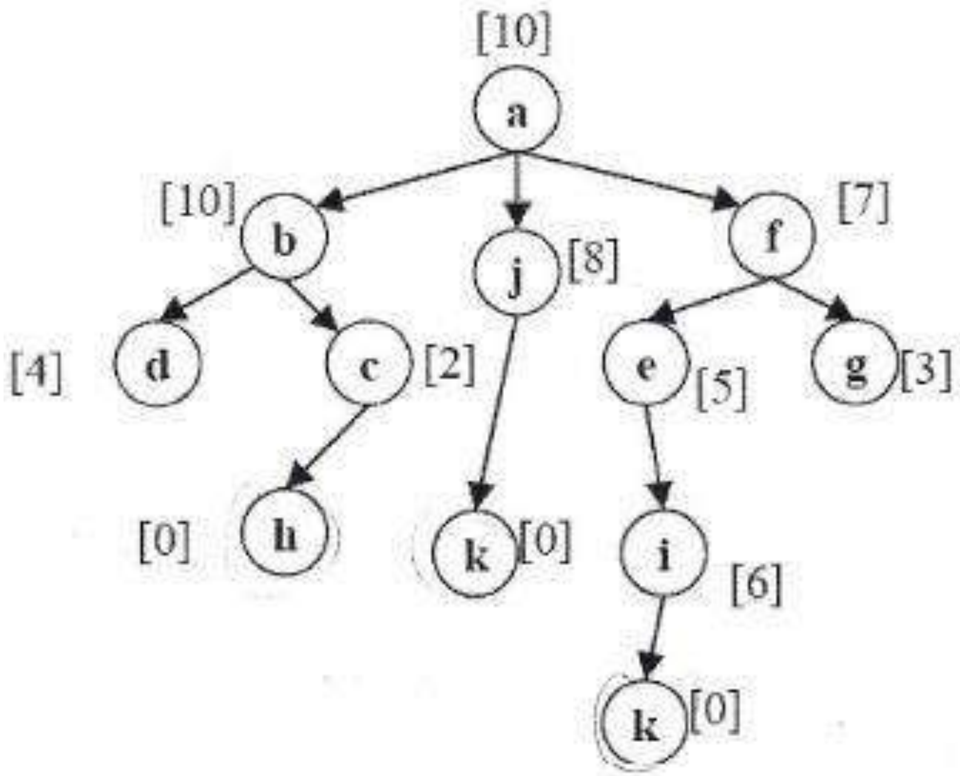


Greedy Best-First Search Goal - Node K

Current	Children
a_{10}	---
a_{10}	f_7, j_8, b_{10}
f_7	j_8, b_{10}
f_7	g_3, e_5, j_8, b_{10}
g_3	e_5, j_8, b_{10}
e_5	

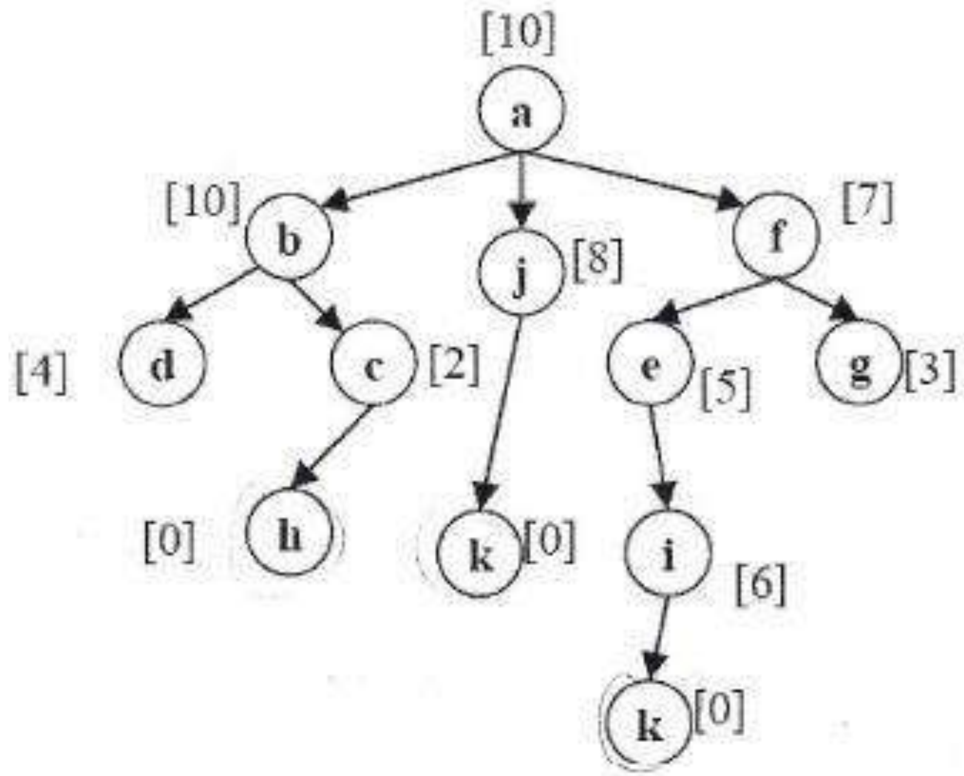


Greedy Best-First Search Goal - Node K



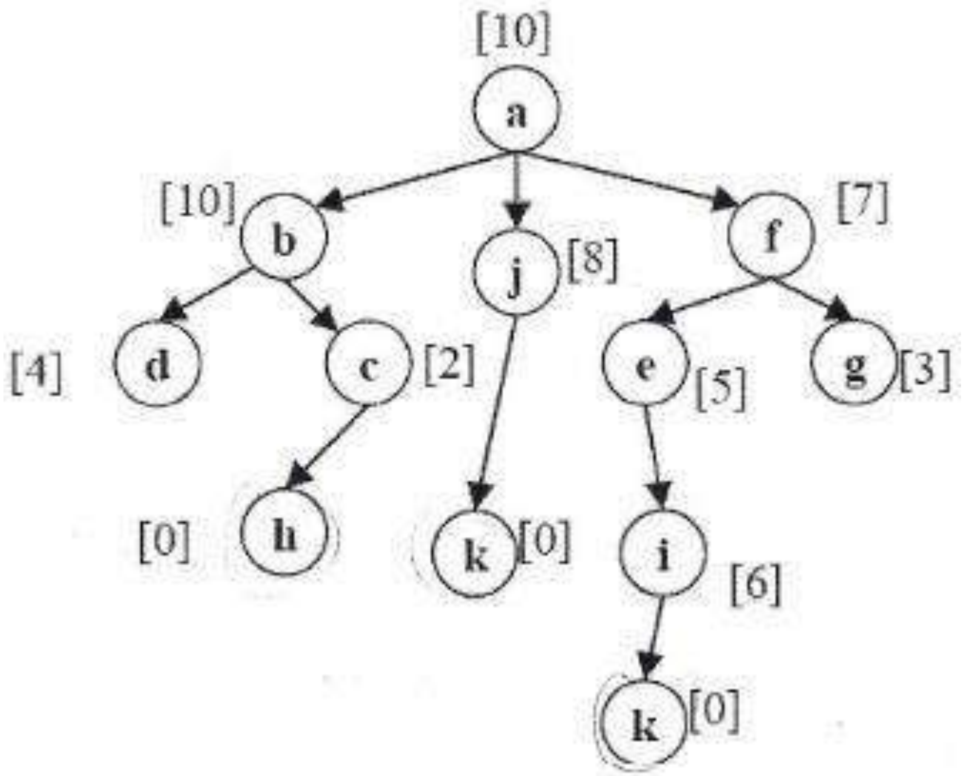
Current	Children
a_{10}	---
a_{10}	f_7, j_8, b_{10}
f_7	j_8, b_{10}
f_7	g_3, e_5, j_8, b_{10}
g_3	e_5, j_8, b_{10}
e_5	j_8, b_{10}

Greedy Best-First Search Goal – Node K



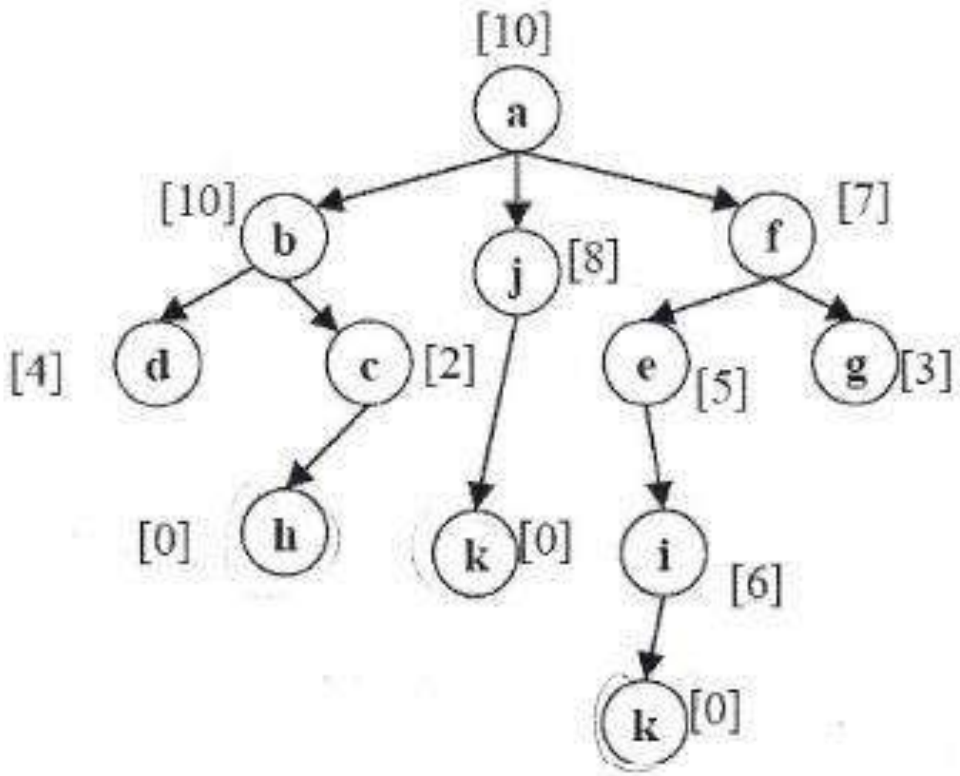
Current	Children
a_{10}	---
a_{10}	$f_7, j_8 b_{10}$
f_7	$j_8 b_{10}$
f_7	$g_3, e_5, j_8 b_{10}$
g_3	$e_5, j_8 b_{10}$
e_5	$j_8 b_{10}$

Greedy Best-First Search Goal – Node K



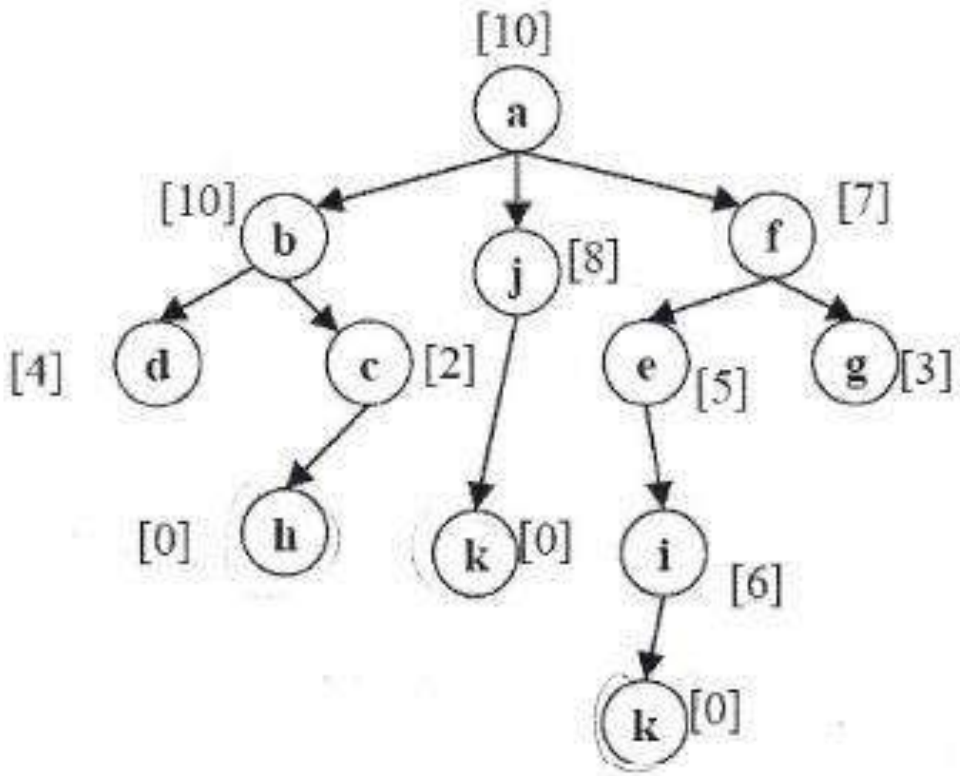
Current	Children
a_{10}	---
a_{10}	f_7, j_8, b_{10}
f_7	j_8, b_{10}
f_7	g_3, e_5, j_8, b_{10}
g_3	e_5, j_8, b_{10}
e_5	j_8, b_{10}
e_5	

Greedy Best-First Search Goal - Node K



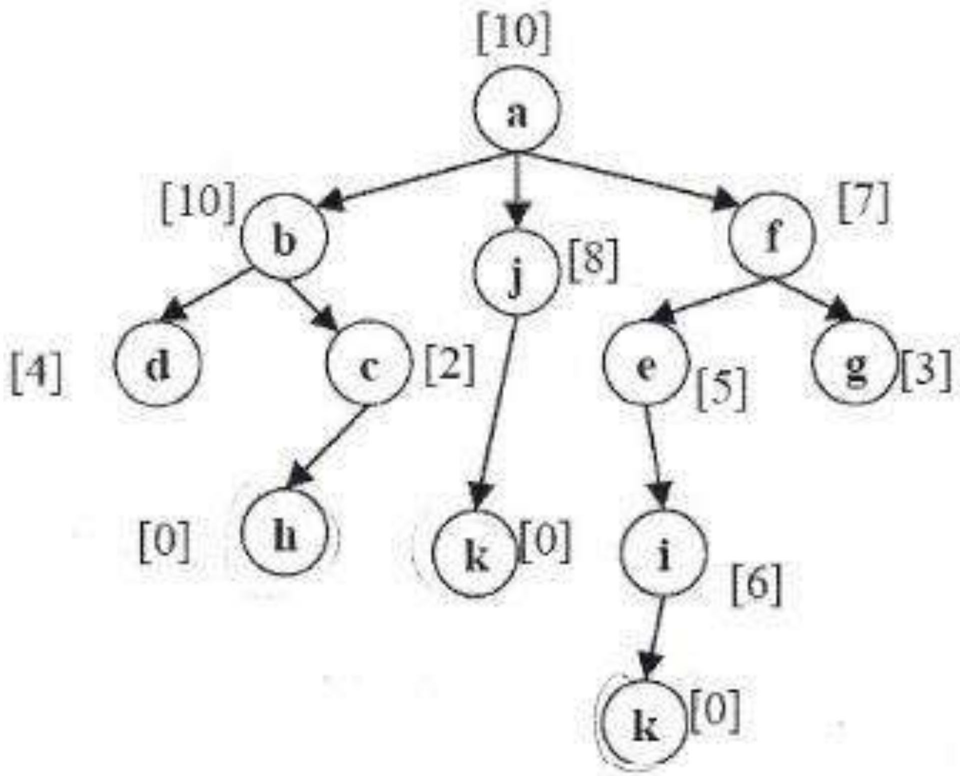
Current	Children
a_{10}	---
a_{10}	f_7, j_8, b_{10}
f_7	j_8, b_{10}
f_7	g_3, e_5, j_8, b_{10}
g_3	e_5, j_8, b_{10}
e_5	j_8, b_{10}
e_5	i_6, j_8, b_{10}

Greedy Best-First Search Goal – Node K



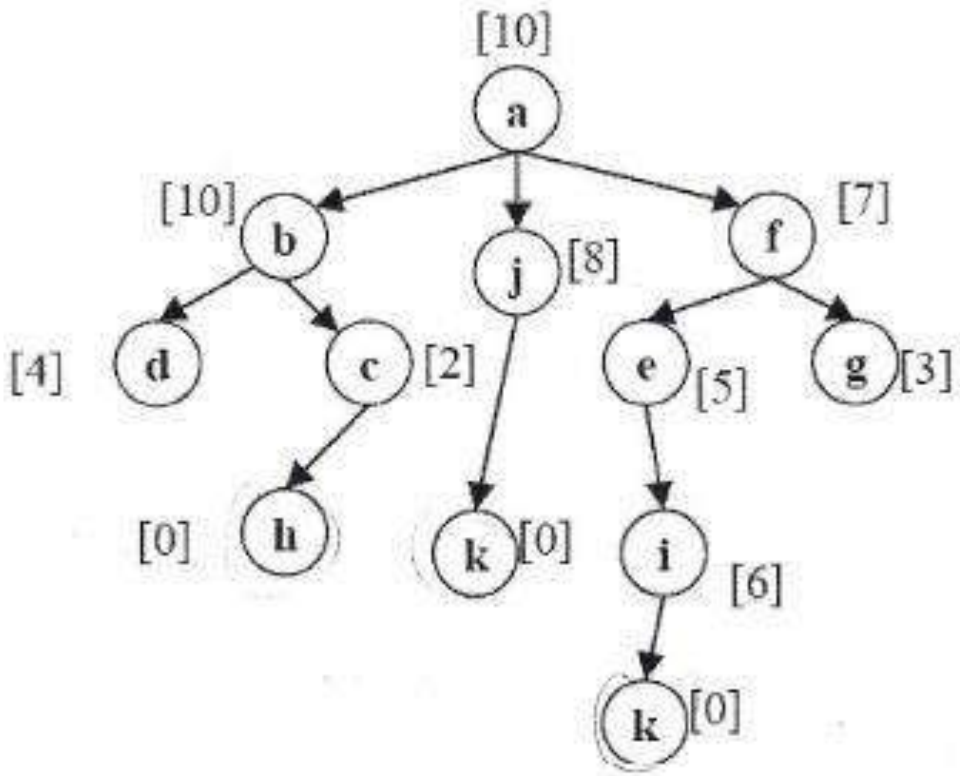
Current	Children
a_{10}	---
a_{10}	$f_7, j_8 b_{10}$
f_7	$j_8 b_{10}$
f_7	$g_3, e_5, j_8 b_{10}$
g_3	$e_5, j_8 b_{10}$
e_5	$j_8 b_{10}$
e_5	$i_6 j_8 b_{10}$
i_6	

Greedy Best-First Search Goal - Node K



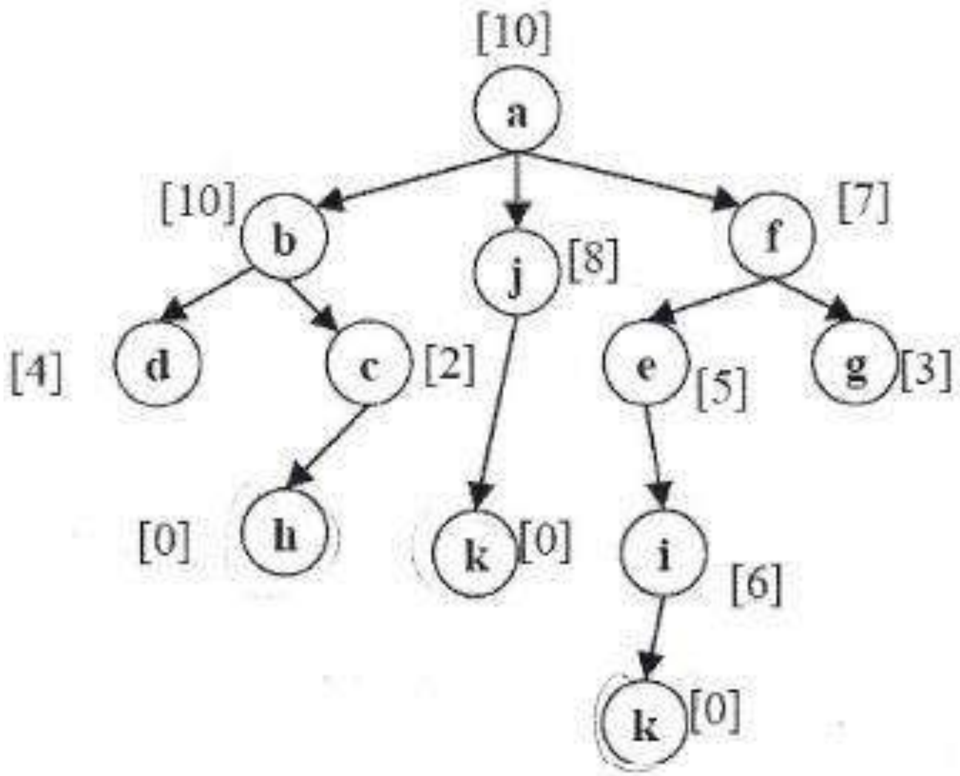
Current	Children
a_{10}	---
a_{10}	f_7, j_8, b_{10}
f_7	$j_8 b_{10}$
f_7	$g_3, e_5, j_8 b_{10}$
g_3	$e_5, j_8 b_{10}$
e_5	$j_8 b_{10}$
e_5	$i_6 j_8 b_{10}$
i_6	$j_8 b_{10}$

Greedy Best-First Search Goal – Node K



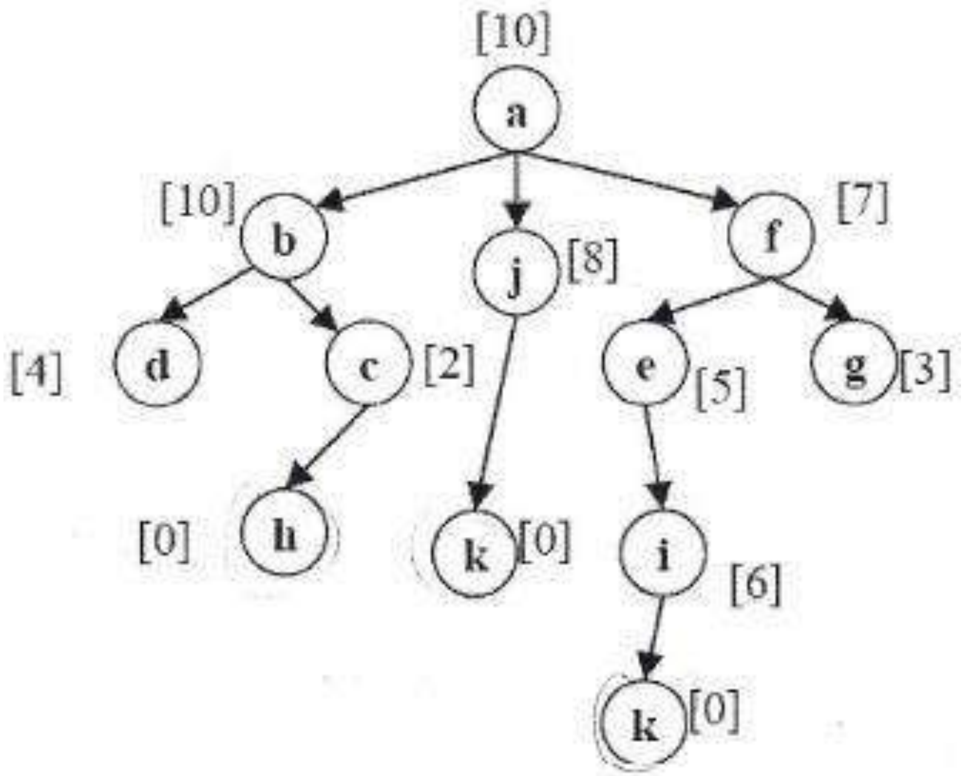
Current	Children
a_{10}	---
a_{10}	f_7, j_8, b_{10}
f_7	$j_8 b_{10}$
f_7	$g_3, e_5, j_8 b_{10}$
g_3	$e_5, j_8 b_{10}$
e_5	$j_8 b_{10}$
e_5	$i_6 j_8 b_{10}$
i_6	$j_8 b_{10}$

Greedy Best-First Search Goal - Node K



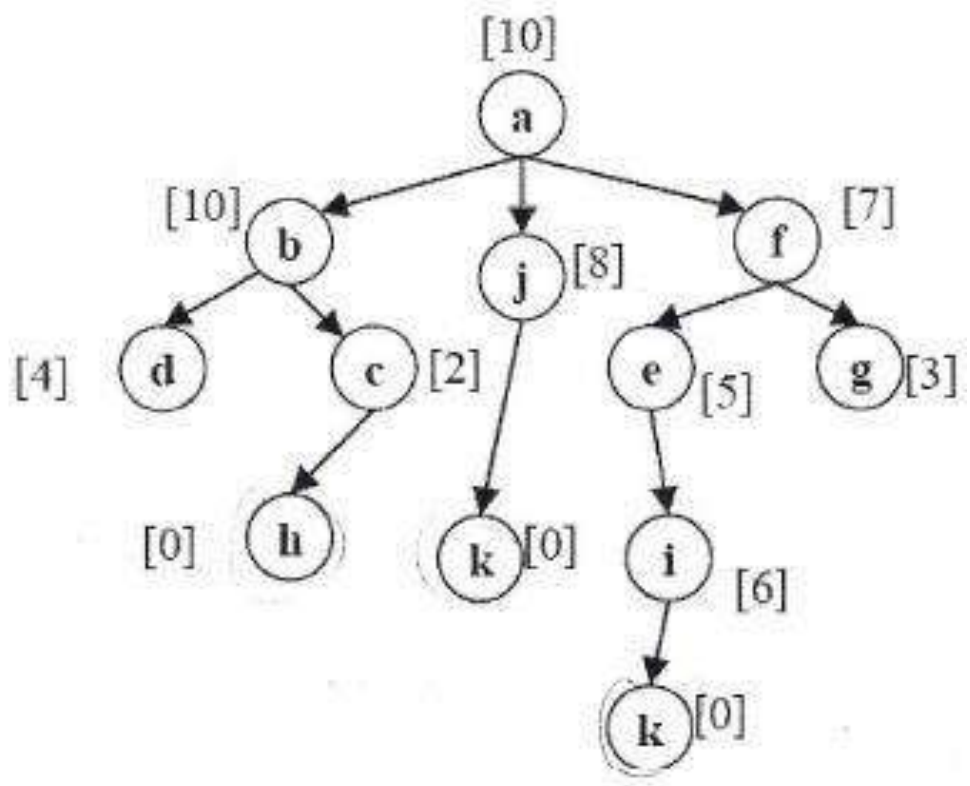
Current	Children
a_{10}	---
a_{10}	f_7, j_8, b_{10}
f_7	$j_8 b_{10}$
f_7	$g_3, e_5, j_8 b_{10}$
g_3	$e_5, j_8 b_{10}$
e_5	$j_8 b_{10}$
e_5	$i_6 j_8 b_{10}$
i_6	$j_8 b_{10}$
i_6	

Greedy Best-First Search Goal - Node K



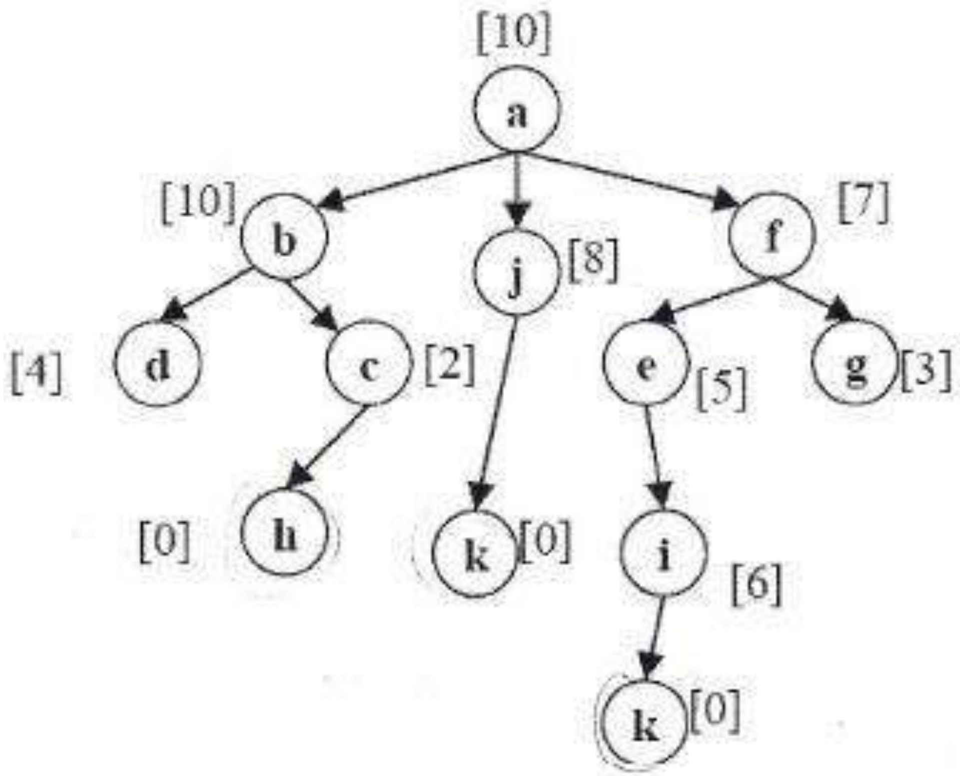
Current	Children
a_{10}	---
a_{10}	f_7, j_8, b_{10}
f_7	$j_8 b_{10}$
f_7	$g_3, e_5, j_8 b_{10}$
g_3	$e_5, j_8 b_{10}$
e_5	$j_8 b_{10}$
e_5	$i_6 j_8 b_{10}$
i_6	$j_8 b_{10}$
i_6	$k_0 j_8 b_{10}$

Greedy Best-First Search Goal - Node K



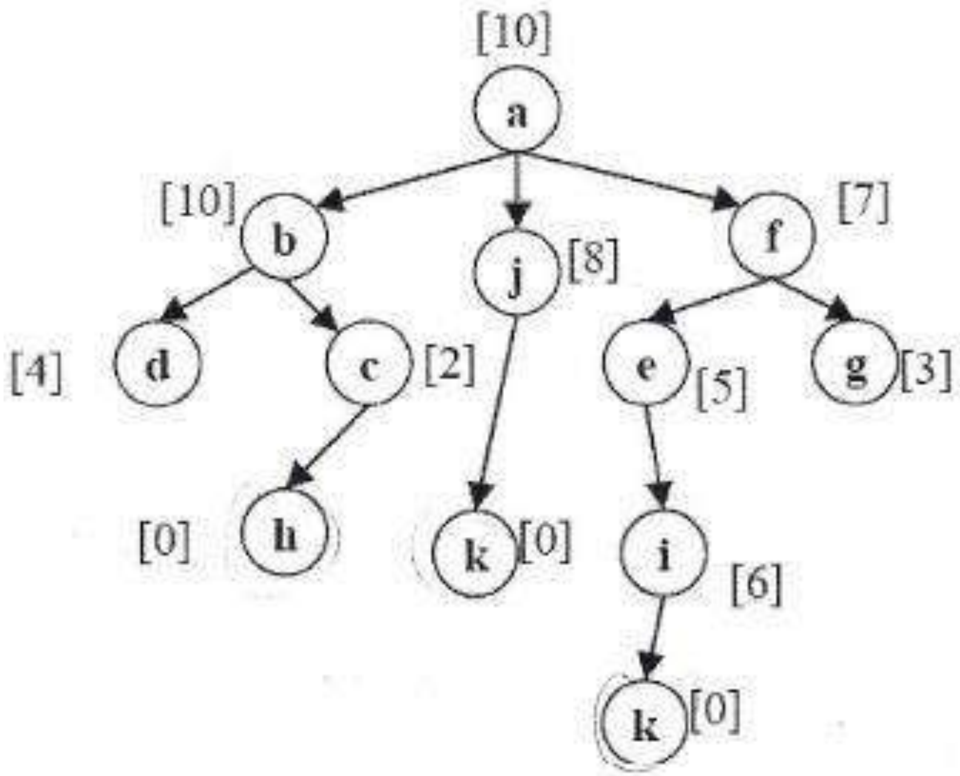
Current	Children
a_{10}	---
a_{10}	f_7, j_8, b_{10}
f_7	j_8, b_{10}
f_7	g_3, e_5, j_8, b_{10}
g_3	e_5, j_8, b_{10}
e_5	j_8, b_{10}
e_5	i_6, j_8, b_{10}
i_6	j_8, b_{10}
i_6	k_0, j_8, b_{10}
k_0	

Greedy Best-First Search Goal - Node K



Current	Children
a_{10}	---
a_{10}	f_7, j_8, b_{10}
f_7	$j_8 b_{10}$
f_7	$g_3, e_5, j_8 b_{10}$
g_3	$e_5, j_8 b_{10}$
e_5	$j_8 b_{10}$
e_5	$i_6 j_8 b_{10}$
i_6	$j_8 b_{10}$
i_6	$k_0 j_8 b_{10}$
k_0	$j_8 b_{10}$

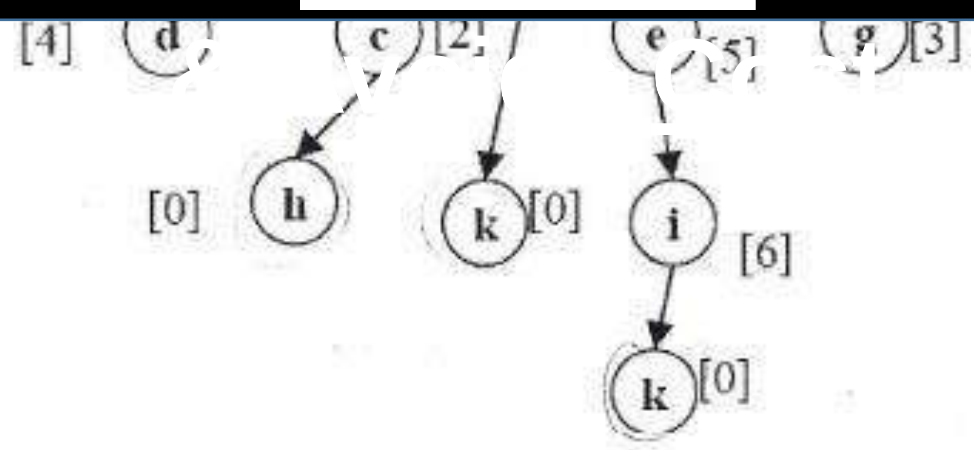
Greedy Best-First Search Goal – Node K



Current	Children
a_{10}	---
a_{10}	f_7, j_8, b_{10}
f_7	j_8, b_{10}
f_7	g_3, e_5, j_8, b_{10}
g_3	e_5, j_8, b_{10}
e_5	j_8, b_{10}
e_5	i_6, j_8, b_{10}
i_6	j_8, b_{10}
i_6	k_0, j_8, b_{10}
k_0	j_8, b_{10}

Greedy Best-First Search Goal - Node K

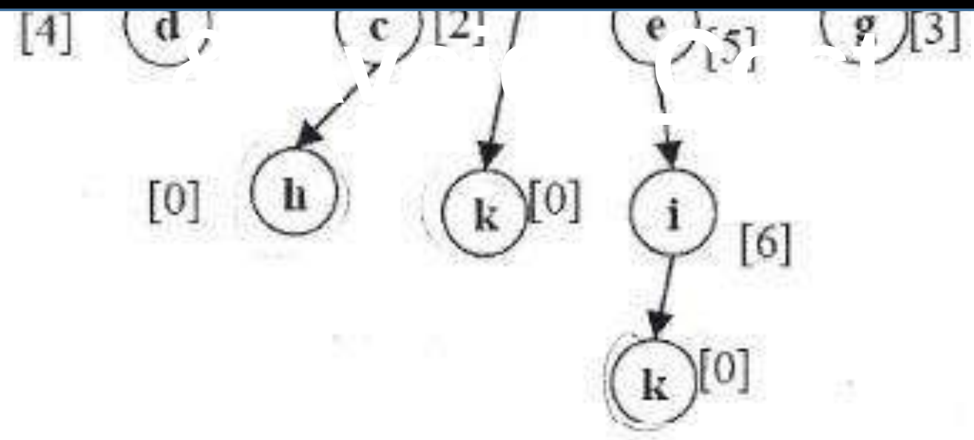
Similar to <u>Uniform Cost</u> Just use <u>Heuristic</u>



Current	Children
a_{10}	---
a_{10}	$f_7, j_8 b_{10}$
f_7	$j_8 b_{10}$
f_7	$g_3, e_5, j_8 b_{10}$
g_3	$e_5, j_8 b_{10}$
e_5	$j_8 b_{10}$
e_5	$i_6 j_8 b_{10}$
i_6	$j_8 b_{10}$
i_6	$k_0 j_8 b_{10}$
k_0	$j_8 b_{10}$

Greedy Best-First Search Goal - Node K

Similar to <u>Uniform Cost</u> Just use <u>Heuristic</u>



<u>Cost</u>: Node to Node
<u>Heuristic</u>: Node to Goal

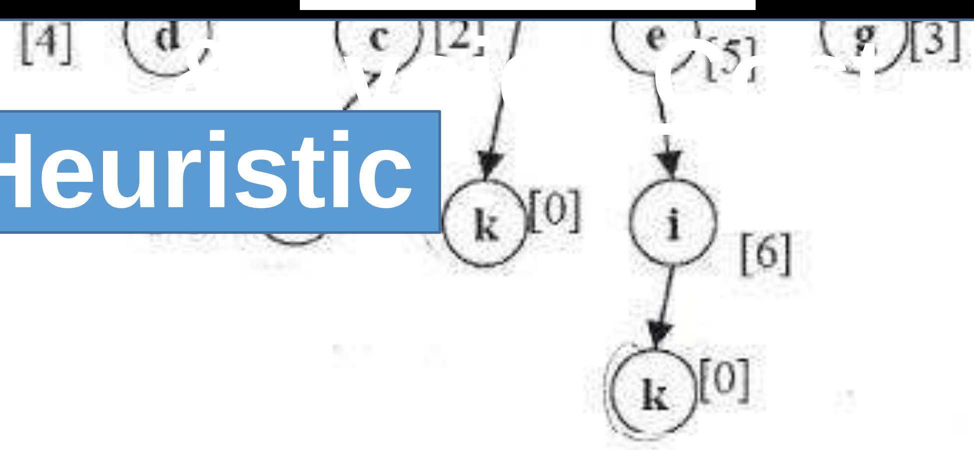
Current	Children
a_{10}	---
a_{10}	f_7, j_8, b_{10}
f [illegible]	$j_8 b_{10}$ [illegible]
5 [illegible]	j_8, b_{10} [illegible]
e_5	$i_6 j_8 b_{10}$
i_6	$j_8 b_{10}$
i_6	$k_0 j_8 b_{10}$
k_0	$j_8 b_{10}$

Greedy Best-First Search Goal - Node K

Similar to Uniform Cost Just use Heuristic

Cost: Node to Node Heuristic: Node to Goal

Heuristic



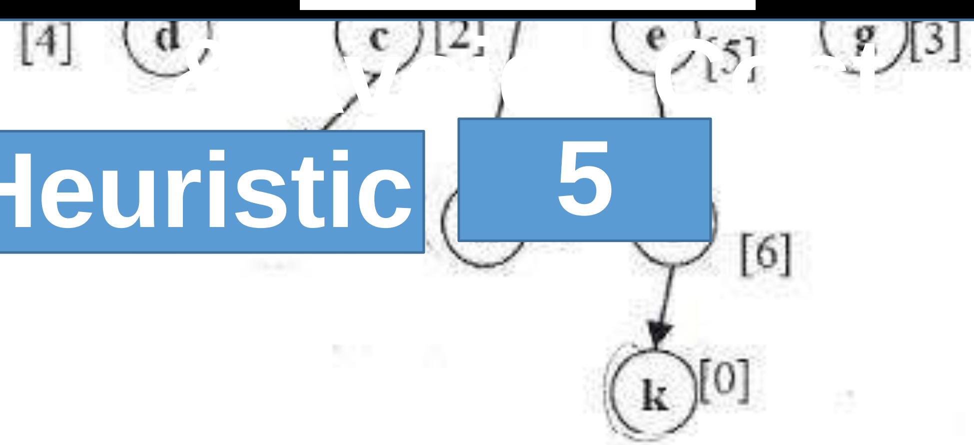
Current	Children
a_{10}	---
a_{10}	f_7, j_8, b_{10}
e_5	i_6, j_8, b_{10}
i_6	j_8, b_{10}
i_6	k_0, j_8, b_{10}
k_0	j_8, b_{10}

Greedy Best-First Search Goal - Node K

Current	Children
a_{10}	---
a_{10}	f_7, j_8, b_{10}
f	$j_8 b_{10}$
	j_8, b_{10}
e_5	$i_6 j_8 b_{10}$
i_6	$j_8 b_{10}$
i_6	$k_0 j_8 b_{10}$
k_0	$j_8 b_{10}$

Similar to <u>Uniform Cost</u> Just use <u>Heuristic</u>

<u>Cost</u>: Node to Node <u>Heuristic</u>: Node to Goal

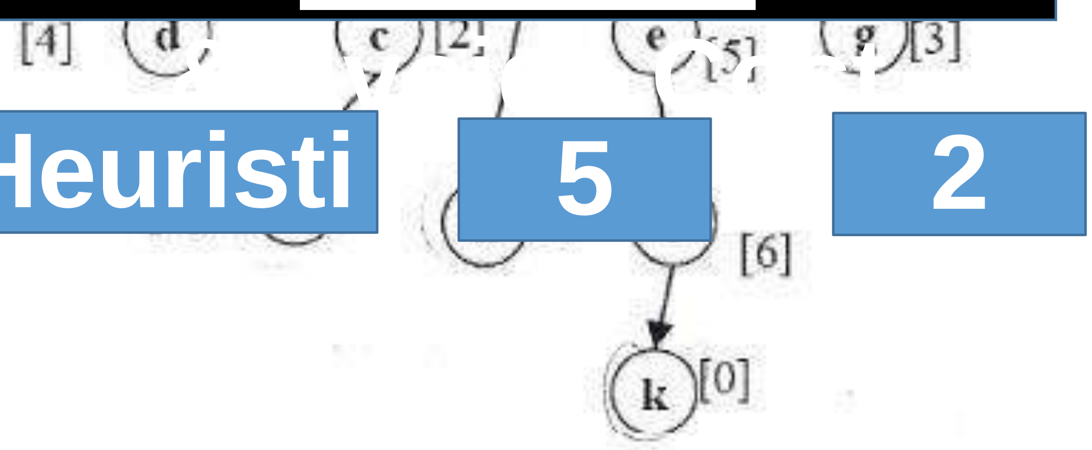


Heuristic 5

Greedy Best-First Search Goal - Node K

Similar to <u>Uniform Cost</u> Just use <u>Heuristic</u>

<u>Cost</u>: Node to Node <u>Heuristic</u>: Node to Goal

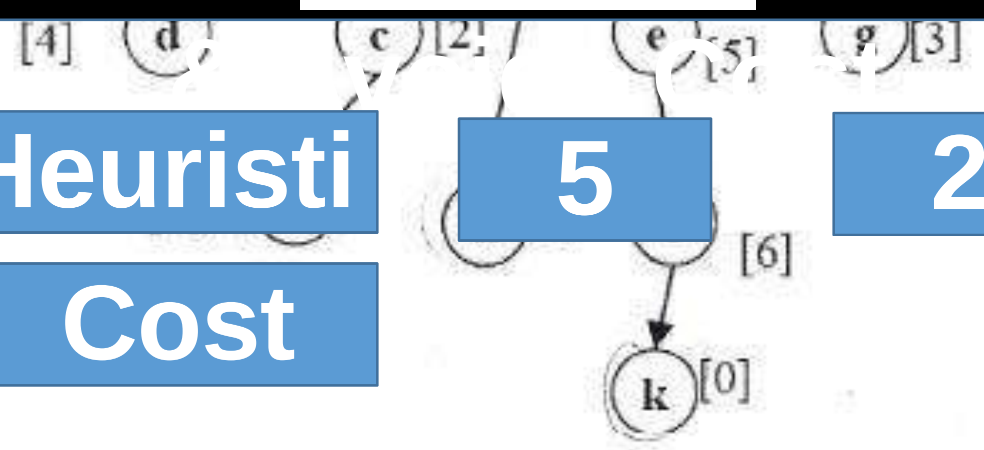


Current	Children
a_{10}	---
a_{10}	$f_7\, j_8 b_{10}$
f	$j_8 b_{10}$
[illegible]	[illegible]
[illegible]	[illegible]
[illegible]	j_8, b_{10}
e_5	$i_6 j_8 b_{10}$
i_6	$j_8 b_{10}$
i_6	$k_0 j_8 b_{10}$
k_0	$j_8 b_{10}$

Greedy Best-First Search Goal - Node K

Similar to <u>Uniform Cost</u> Just use <u>Heuristic</u>

<u>Cost</u>: Node to Node <u>Heuristic</u>: Node to Goal



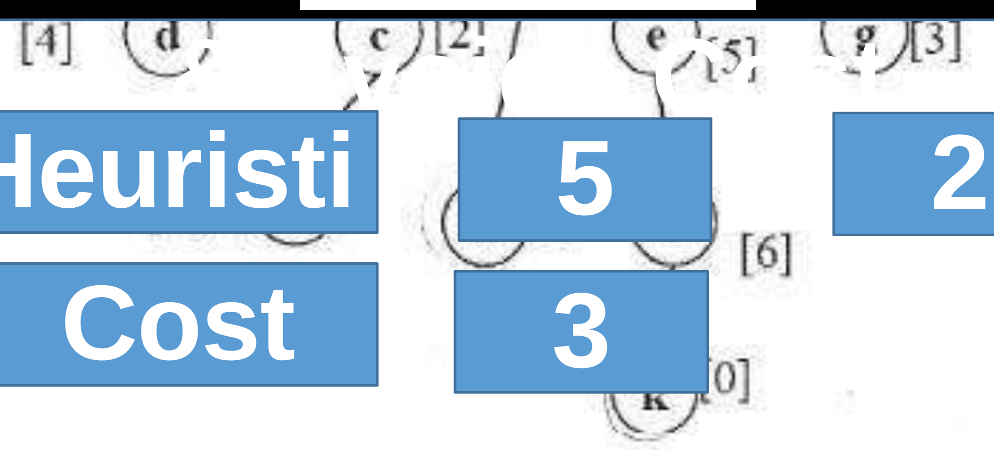
Heuristi	5	2
Cost		

Current	Children
a_{10}	---
a_{10}	$f_7\, j_8 b_{10}$
f	$j_8 b_{10}$
	j_8, b_{10}
e_5	$i_6 j_8 b_{10}$
i_6	$j_8 b_{10}$
i_6	$k_0 j_8 b_{10}$
k_0	$j_8 b_{10}$

Greedy Best-First Search Goal – Node K

<u>Similar to Uniform Cost</u> Just use <u>Heuristic</u>

<u>Cost</u>: Node to Node <u>Heuristic</u>: Node to Goal



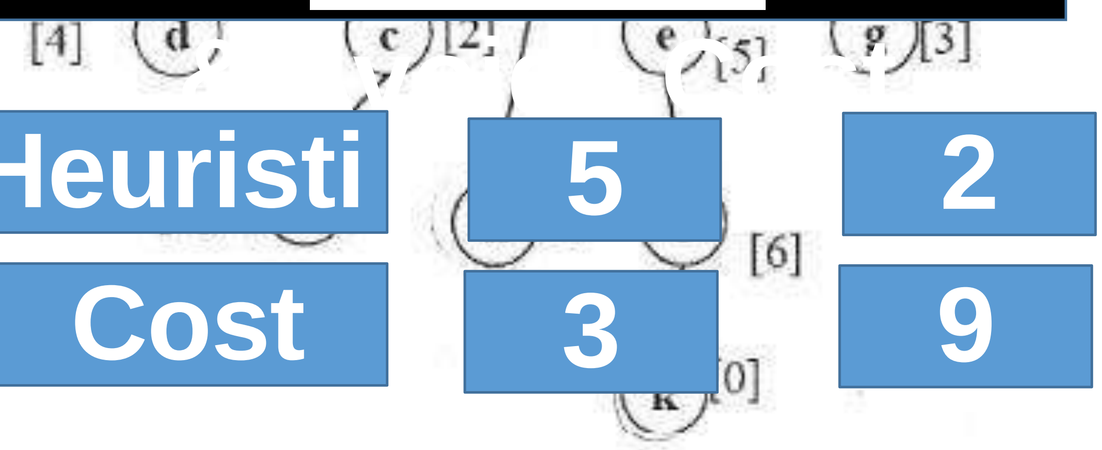
Heuristi	5	2
Cost	3	

Current	Children
a_{10}	---
a_{10}	$f_7, j_8 b_{10}$
e_5	$i_6 j_8 b_{10}$
i_6	$j_8 b_{10}$
i_6	$k_0 j_8 b_{10}$
k_0	$j_8 b_{10}$

Greedy Best-First Search Goal - Node K

Similar to <u>Uniform Cost</u> Just use <u>Heuristic</u>

<u>Cost</u>: Node to Node <u>Heuristic</u>: Node to Goal



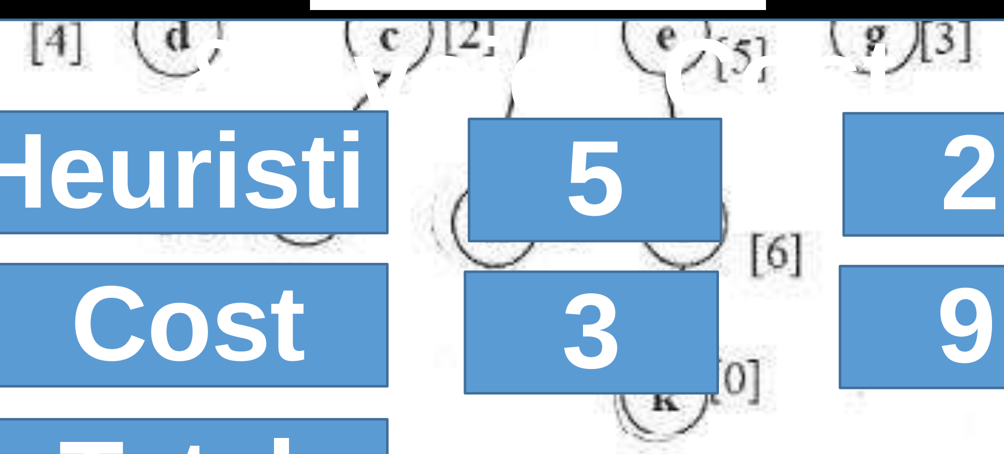
Heuristi	**5**	**2**
Cost	**3**	**9**

Current	Children
a_{10}	---
a_{10}	$f_7\, j_8 b_{10}$
f_7	$j_8 b_{10}$
[illegible]	[illegible]
[illegible]	[illegible]
[illegible]	j_8, b_{10}
e_5	$i_6 j_8 b_{10}$
i_6	$j_8 b_{10}$
i_6	$k_0 j_8 b_{10}$
k_0	$j_8 b_{10}$

Greedy Best-First Search Goal - Node K

Similar to <u>Uniform Cost</u> Just use <u>Heuristic</u>

<u>Cost</u>: Node to Node <u>Heuristic</u>: Node to Goal



Heuristi	5	2
Cost	3	9
Total		

Current	Children
a_{10}	---
a_{10}	$f_7, j_8 b_{10}$
f	$j_8 b$
	j_8, b_{10}
e_5	$i_6 j_8 b_{10}$
i_6	$j_8 b_{10}$
i_6	$k_0 j_8 b_{10}$
k_0	$j_8 b_{10}$

Greedy Best-First Search Goal - Node K

Similar to Uniform Cost Just use Heuristic

Cost: Node to Node Heuristic: Node to Goal

Current	Children
a_{10}	---
a_{10}	$f_7, j_8 b_{10}$
f_7	$j_8 b_{10}$
[illegible]	[illegible]
[illegible]	[illegible]
[illegible]	j_8, b_{10}
e_5	$i_6 j_8 b_{10}$
i_6	$j_8 b_{10}$
i_6	$k_0 j_8 b_{10}$
k_0	$j_8 b_{10}$

Heuristi	5	2
Cost	3	9
Total	8	

Greedy Best-First Search Goal - Node K

Similar to Uniform Cost Just use Heuristic

Heuristi	5	2
Cost	3	9
Total	8	11

Current	Children
a_{10}	---
a_{10}	f_7, j_8, b_{10}
f [illegible]	$j_8 b_{10}$ [illegible]
[illegible]	[illegible]
[illegible]	[illegible]
5	j_8, b_{10}
e_5	$i_6 j_8 b_{10}$
i_6	$j_8 b_{10}$
i_6	$k_0 j_8 b_{10}$
k_0	$j_8 b_{10}$

Cost: Node to Node

Heuristic: Node to Goal

Continue…

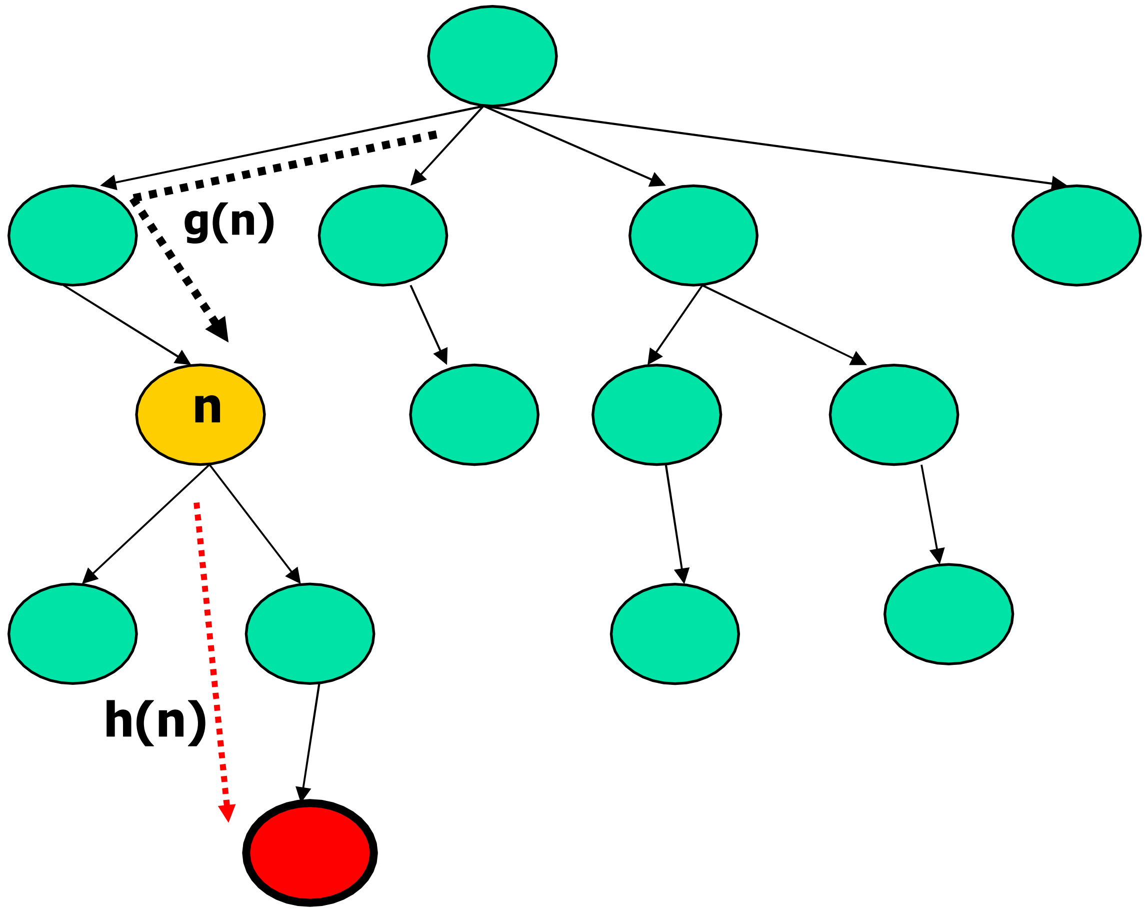
- Greedy search is **not optimal**
- Greedy search is **incomplete** without systematic checking of repeated states.
- In the worst case, the Time and Space Complexity of Greedy Search are both **$O(b^m)$**, Where b is the branching factor and m the maximum path length.

A* Search

- **Greedy Search minimizes a heuristic h(n)** which is an estimated cost from a node n to the goal state. Greedy Search is **efficient but it is not optimal nor complete.**
- **Uniform Cost Search minimizes the cost g(n)** from the initial state to n. **UCS is optimal and complete but not efficient.**
- **New Strategy**: ***Combine*** ***Greedy Search*** and ***UCS*** to get an ***efficient*** algorithm which is ***complete and optimal***.

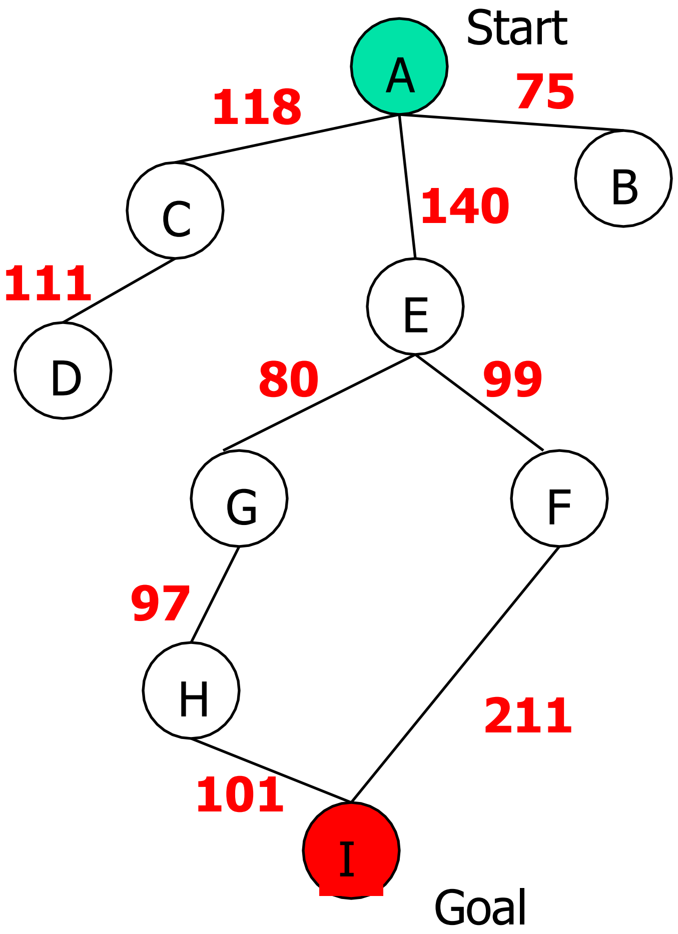
Continue…

- A* uses a heuristic function which $f(n) = g(n) + h(n)$
- **g(n)** is the exact cost to reach node *n* from the initial state.
- **h(n)** is an estimation of the remaining cost to reach the goal.



$$f(n) = g(n)+h(n)$$

A* Search



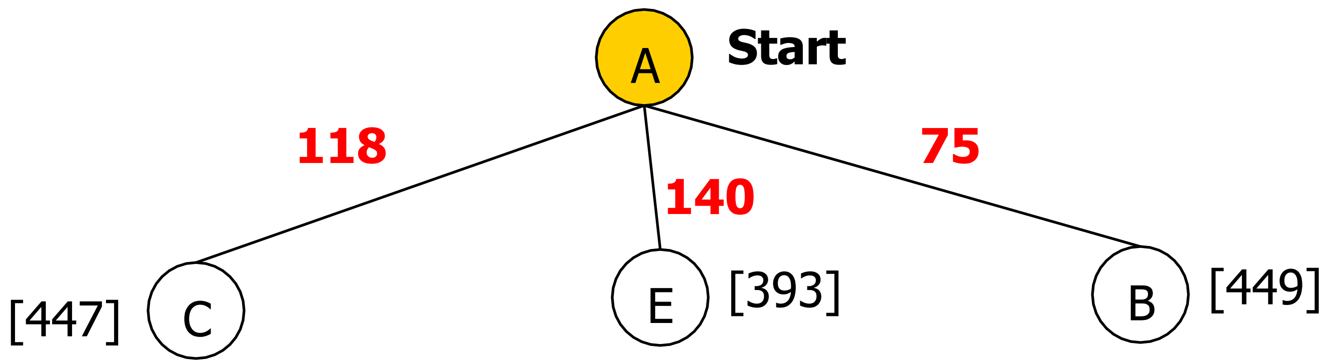
State	Heuristic: h(n)
A	366
B	374
C	329
D	244
E	253
F	178
G	193
H	98
I	0

$$f(n) = g(n) + h(n)$$

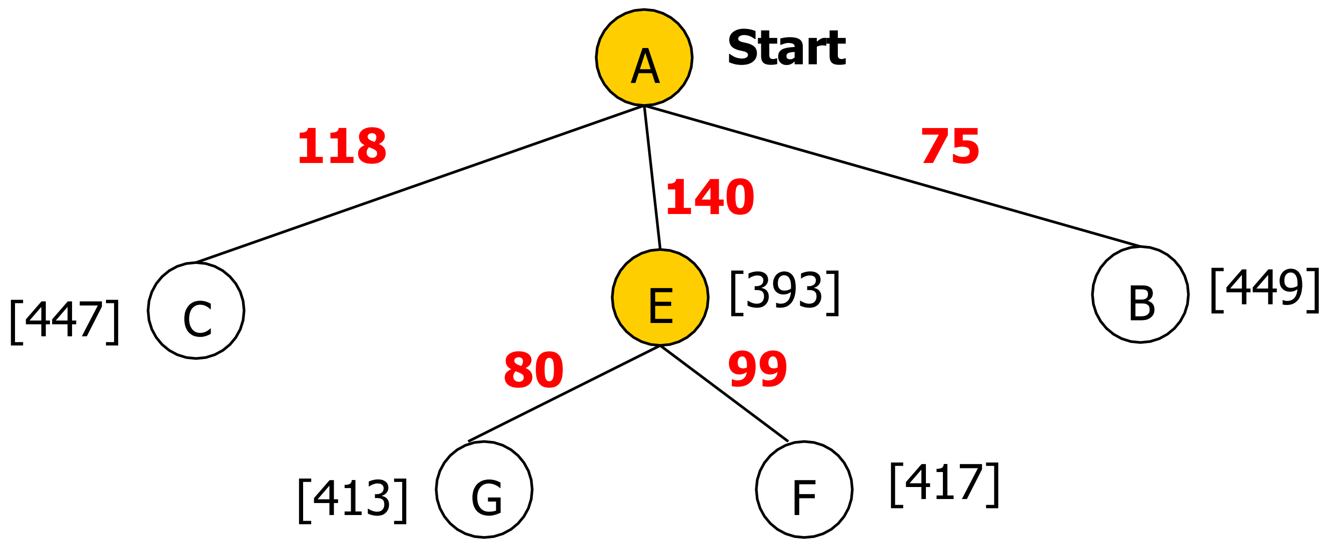
g(n): is the exact cost to reach node *n* from the initial state.

A* Search: Tree Search

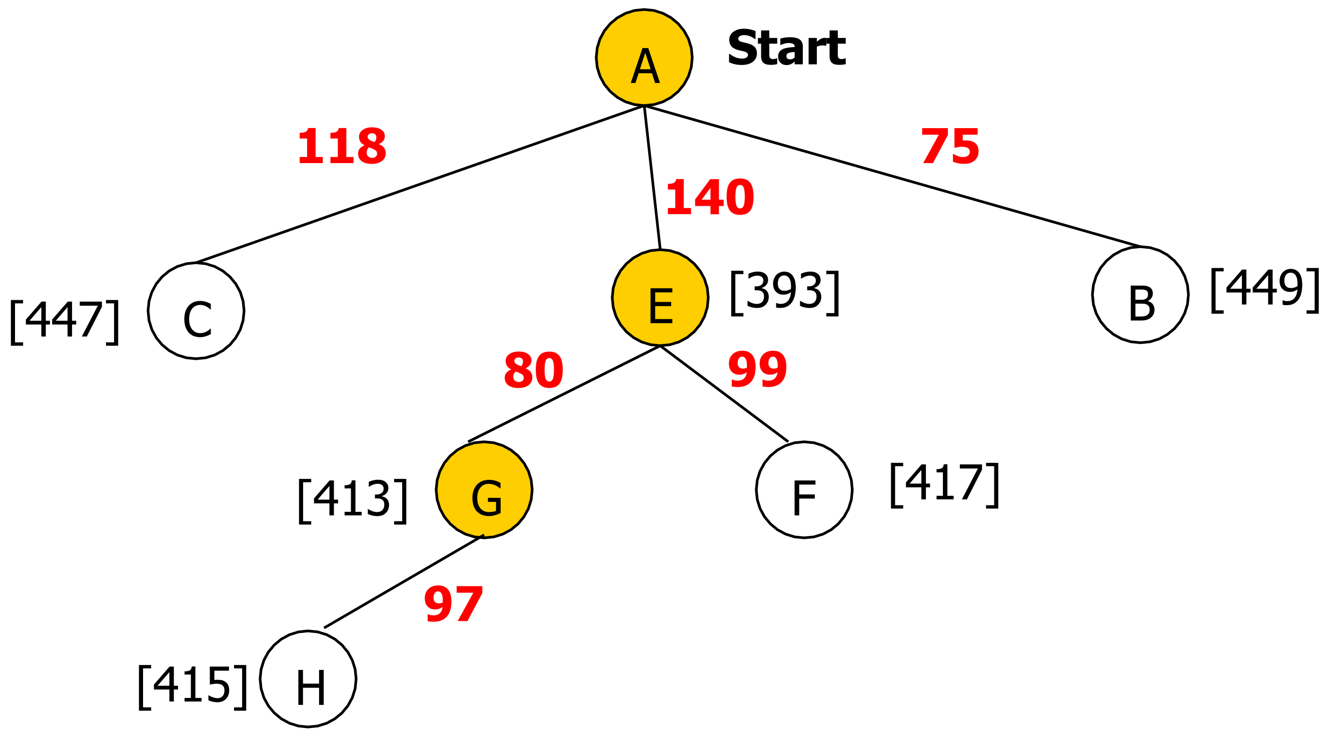
A* Search: Tree Search



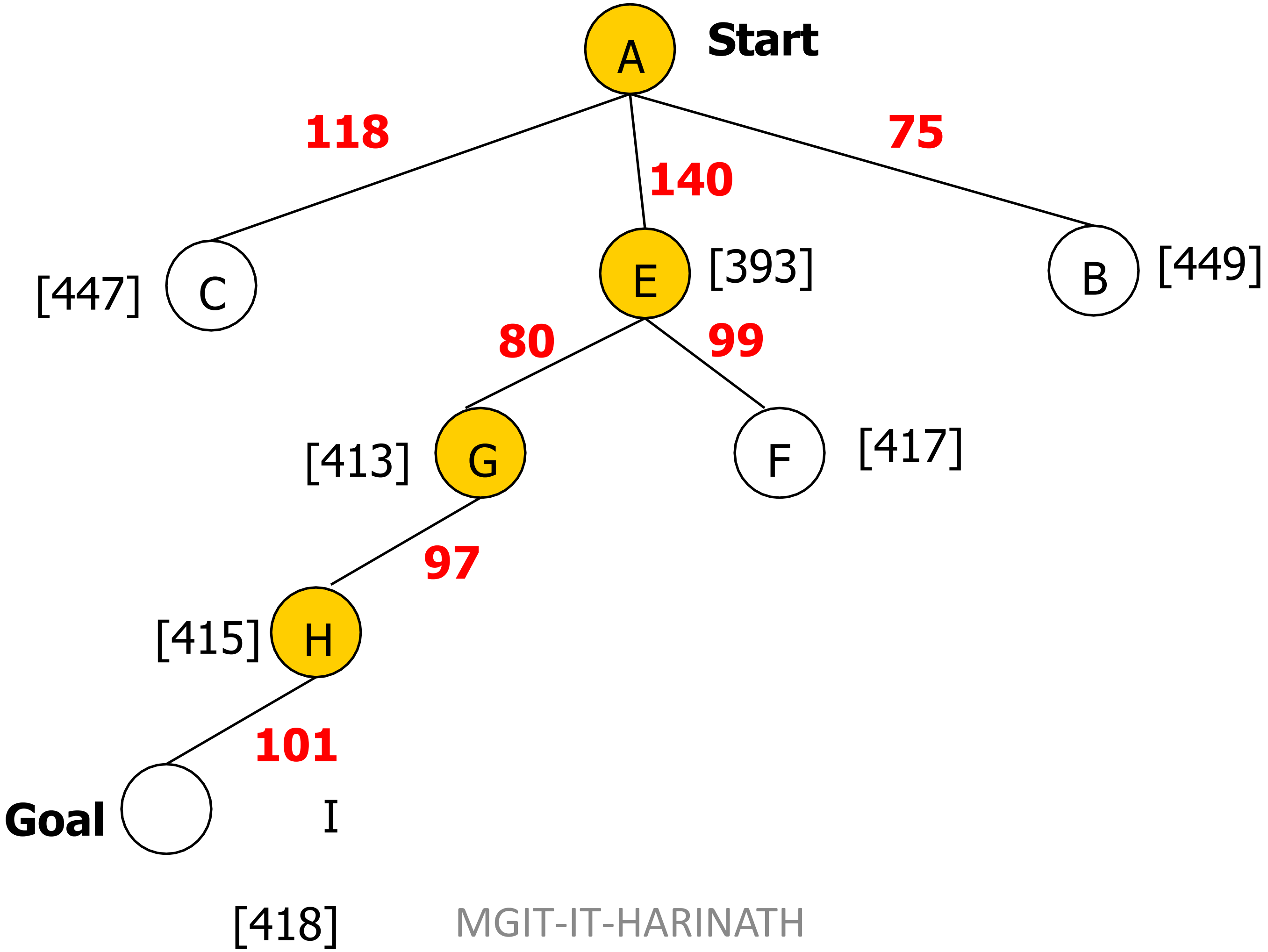
A* Search: Tree Search



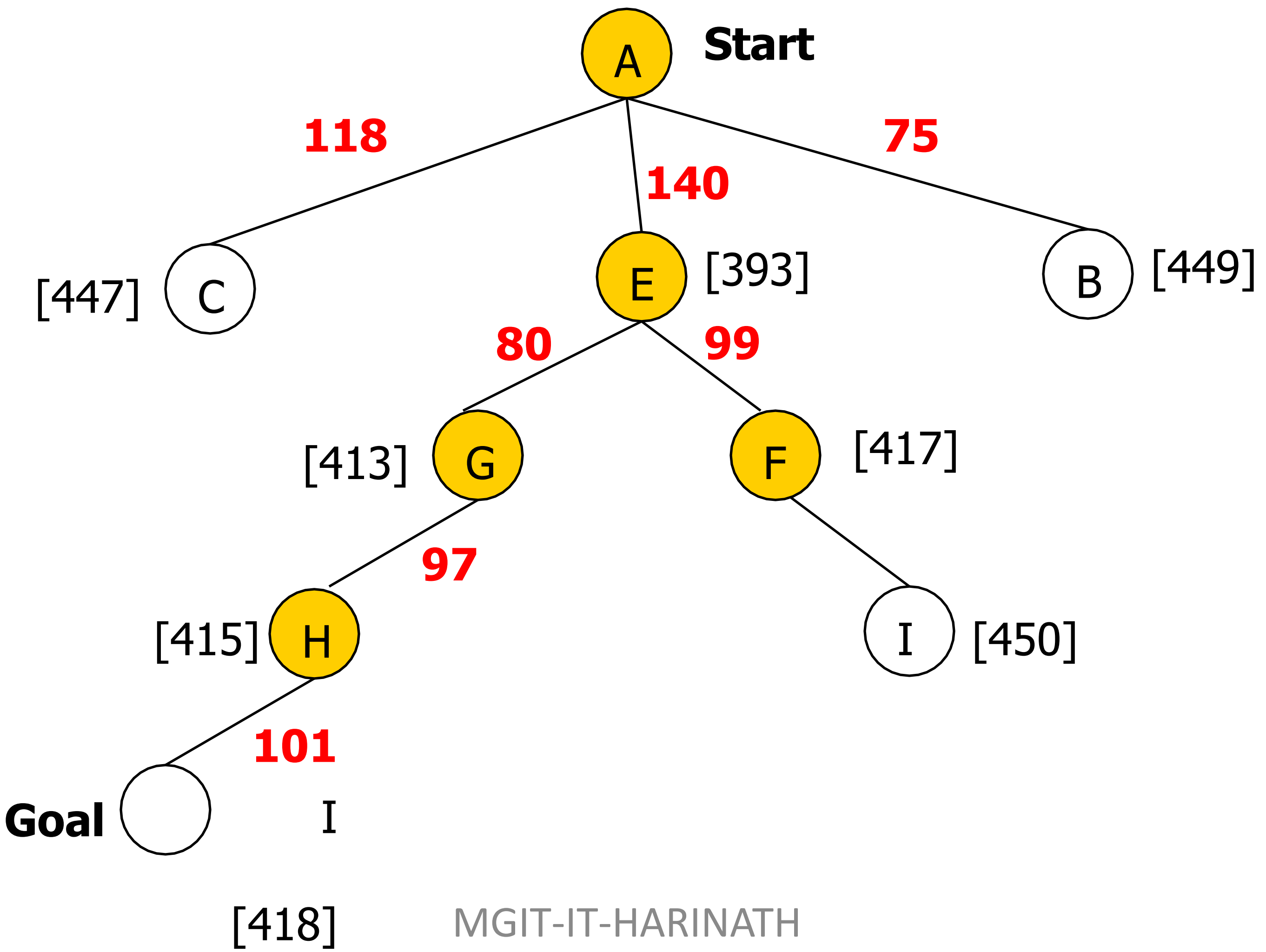
A* Search: Tree Search



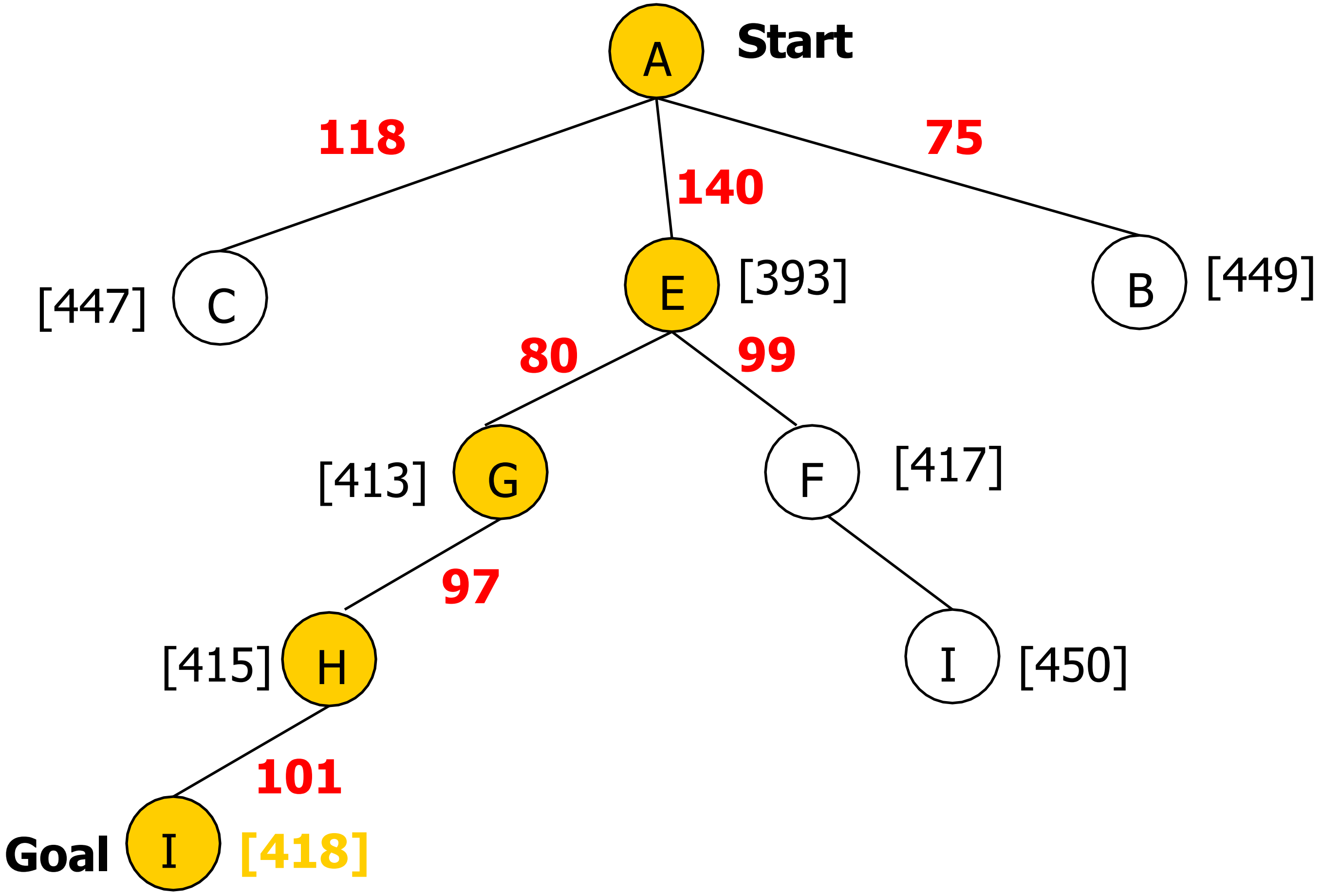
A* Search: Tree Search



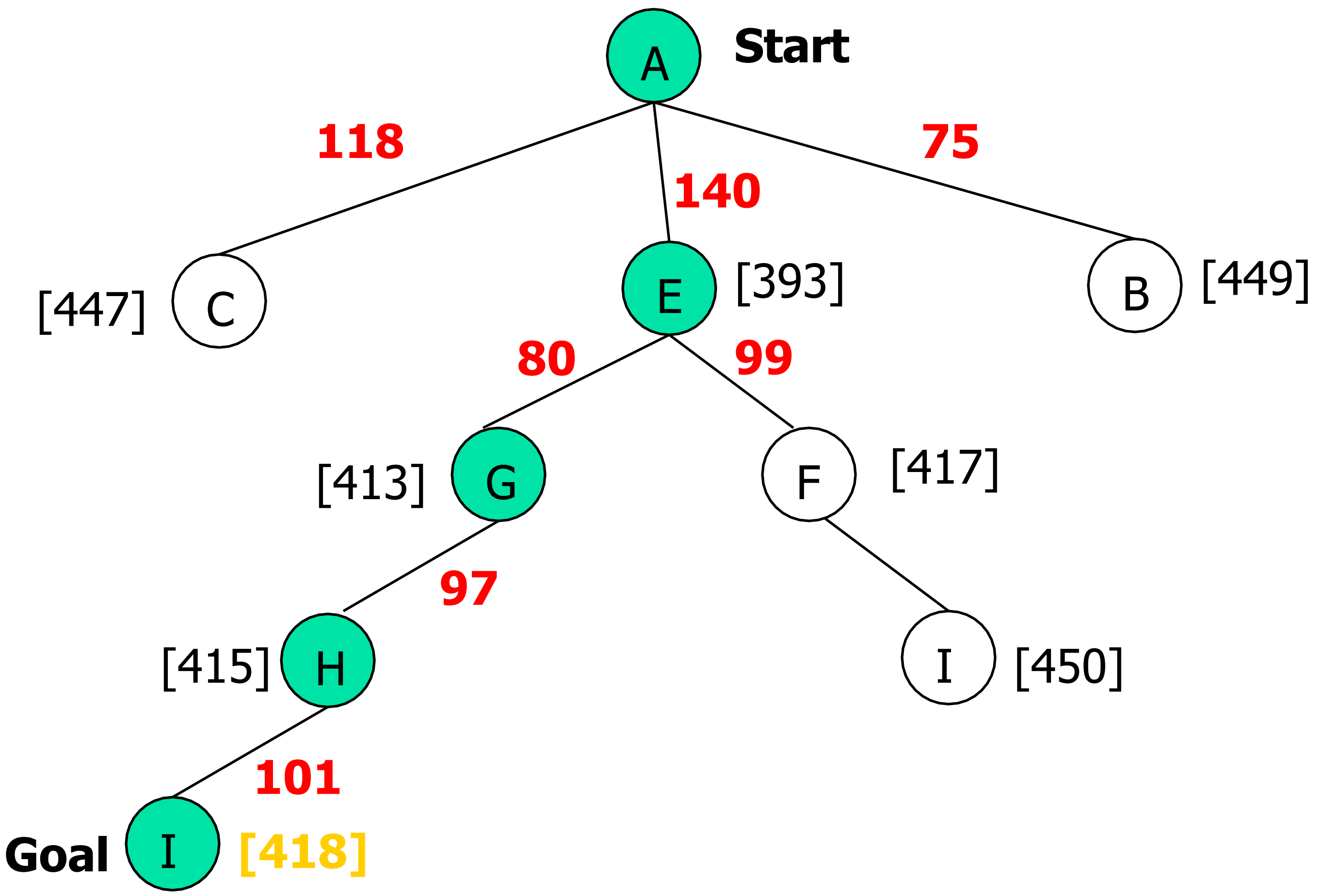
A* Search: Tree Search



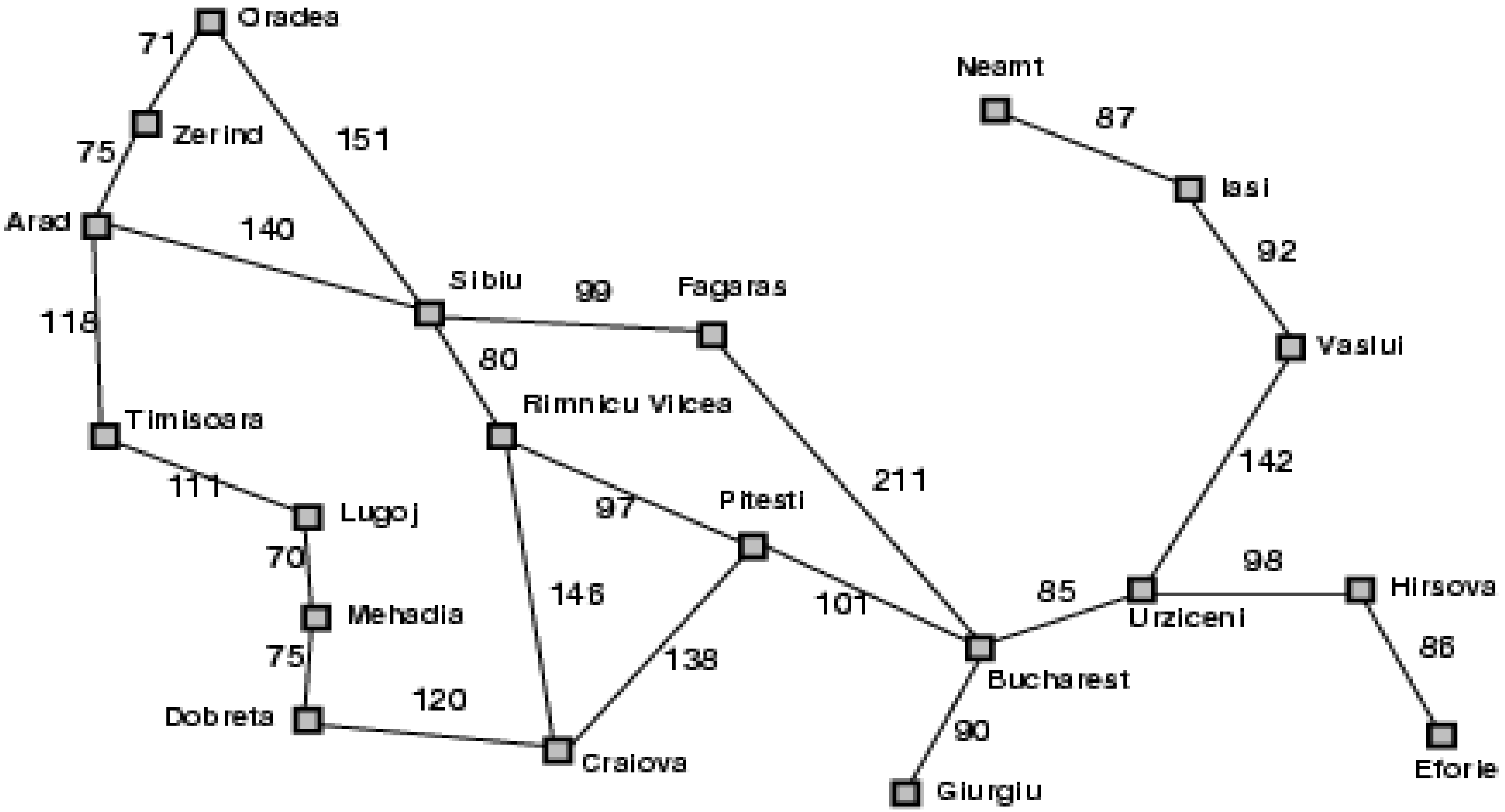
A* Search: Tree Search



A* Search: Tree Search



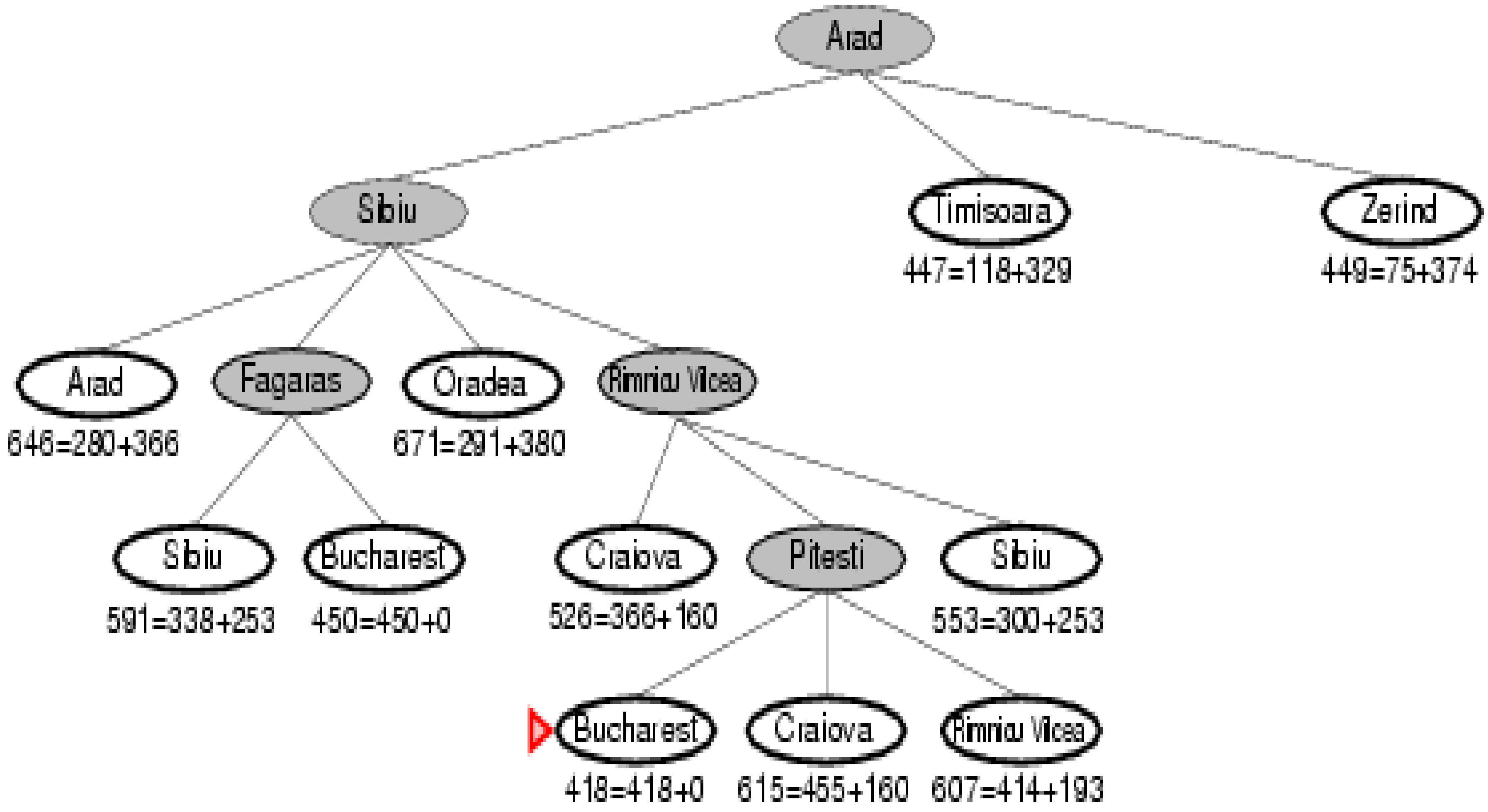
A* Search – Combines Heuristic & Cost Goal - Bucharest



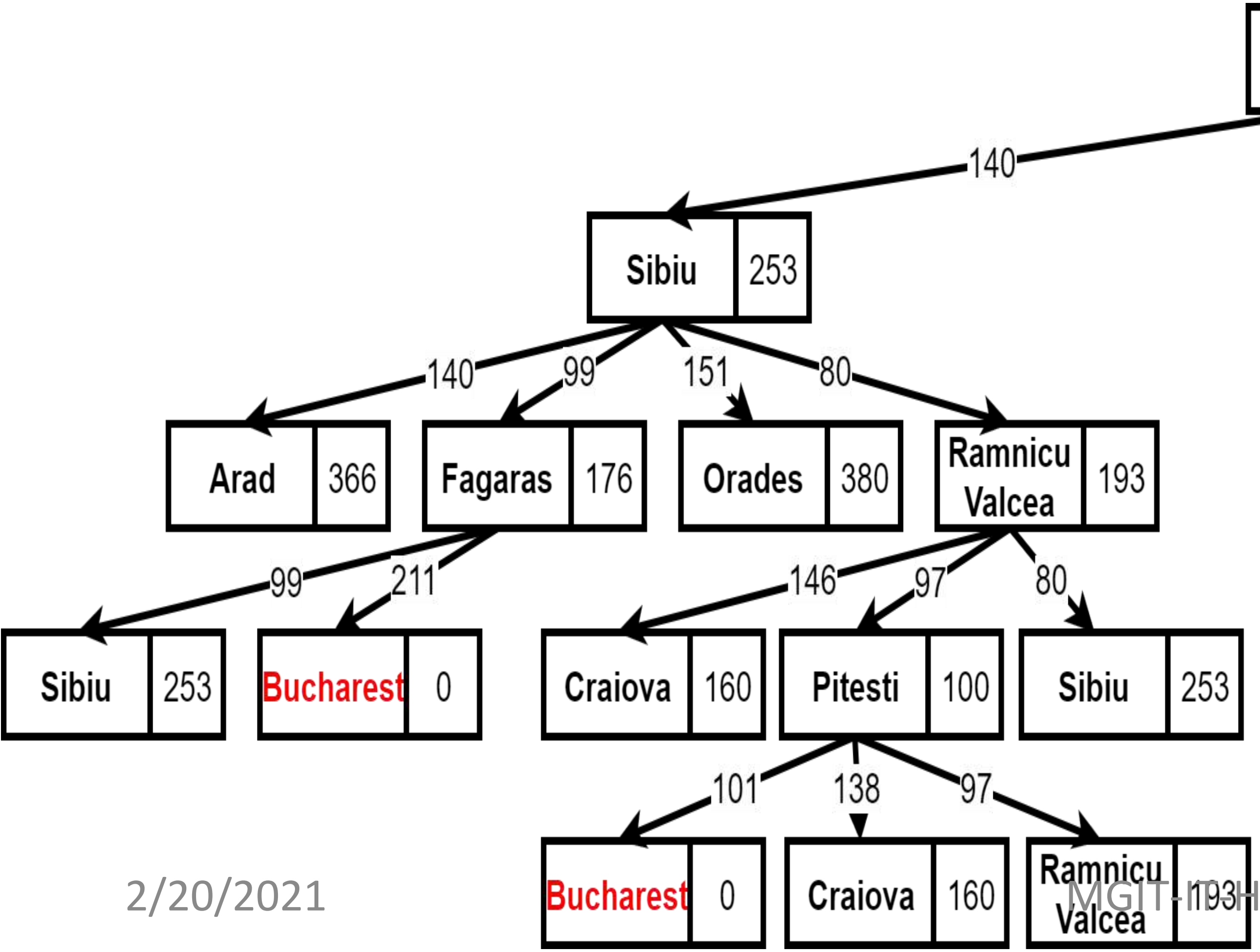
Straight–line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

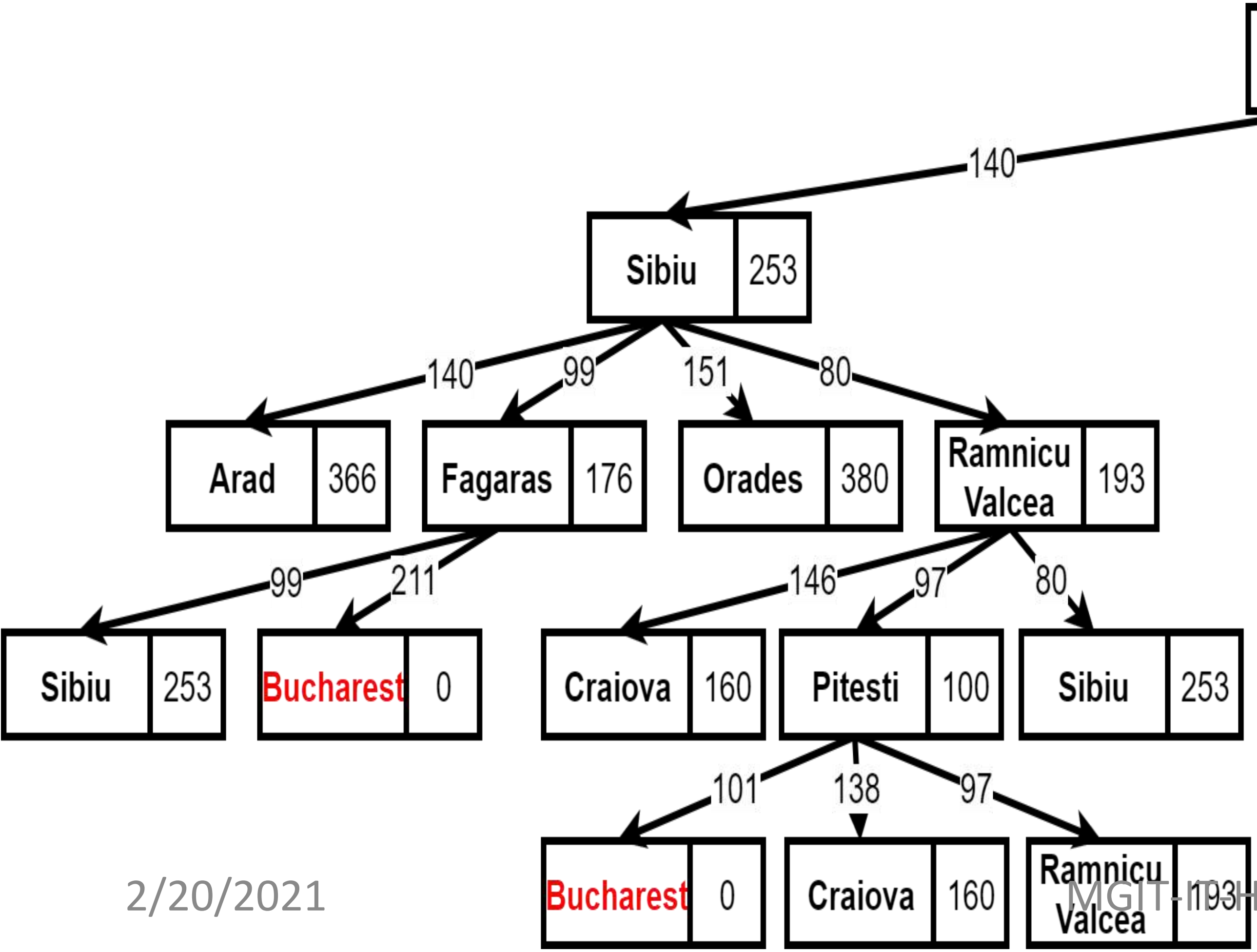
A* Search – Continue Goal -



Arad

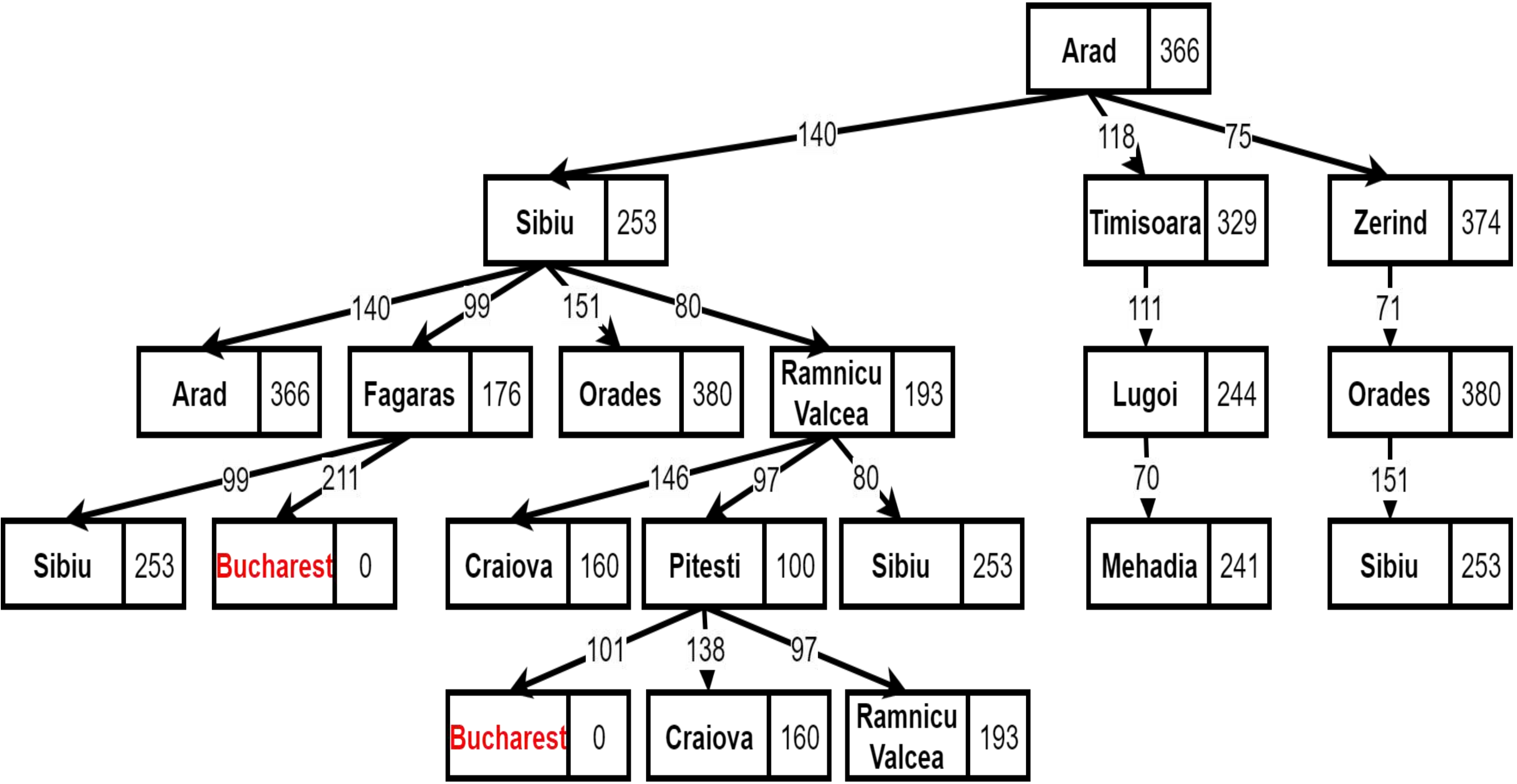

Arad 366
140
118
75
Sibiu 253
Timisoara 329
Zerind 374
140
99
151
80
111
71
Arad 366
Fagaras 176
Orades 380
Ramnicu Valcea 193
Lugoi 244
Orades 380
99
211
146
97
80
70
151
Sibiu 253
Bucharest 0
Craiova 160
Pitesti 100
Sibiu 253
Mehadia 241
Sibiu 253
101
138
97
Bucharest 0
Craiova 160
Ramnicu Valcea 193

Arad



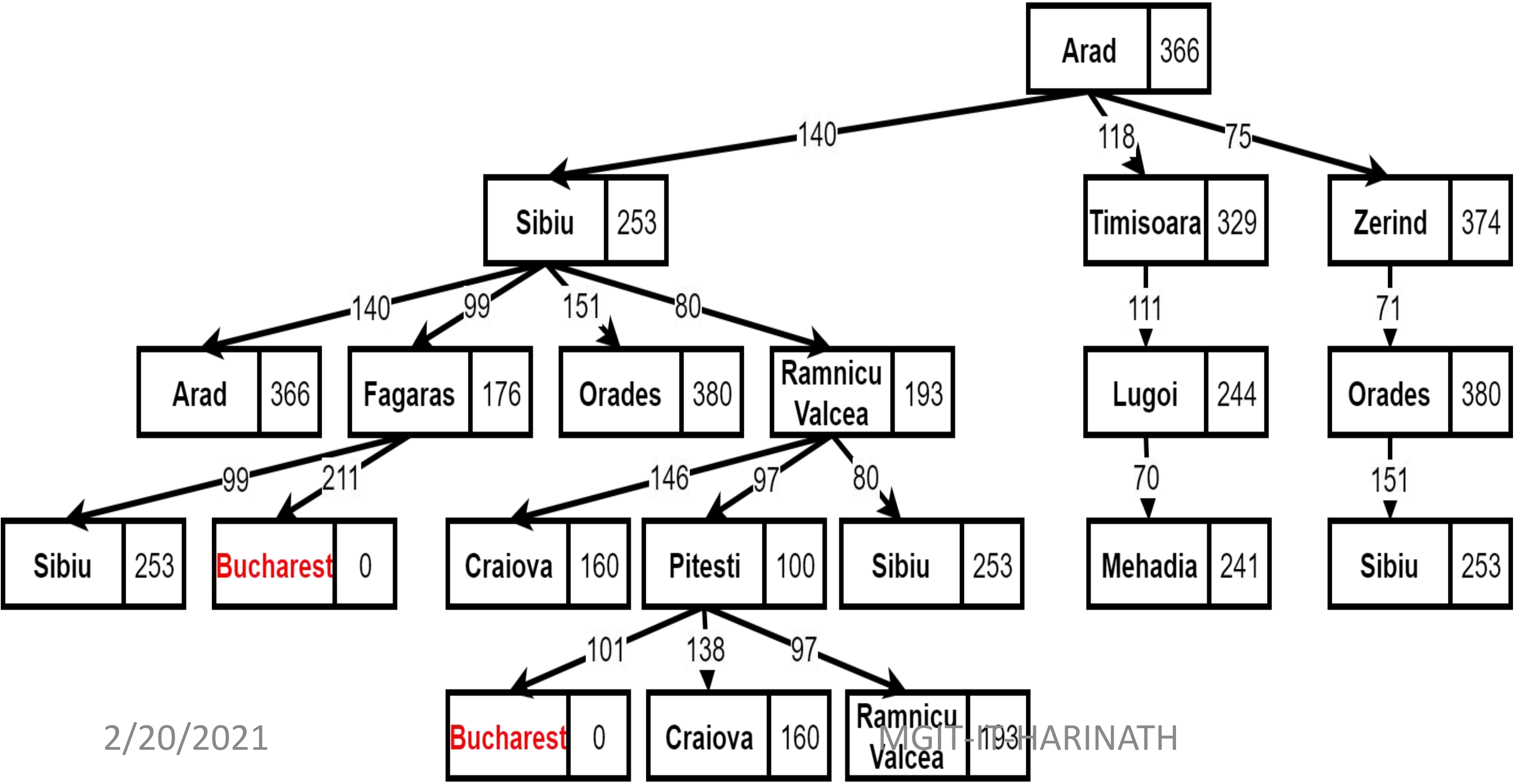
Arad 366
140
118
75
Sibiu 253
Timisoara 329
Zerind 374
140
99
151
80
111
71
Arad 366
Fagaras 176
Orades 380
Ramnicu Valcea 193
Lugoi 244
Orades 380
99
211
146
97
80
70
151
Sibiu 253
Bucharest 0
Craiova 160
Pitesti 100
Sibiu 253
Mehadia 241
Sibiu 253
101
138
97
Bucharest 0
Craiova 160
Ramnicu Valcea 193

Arad



Arad Children

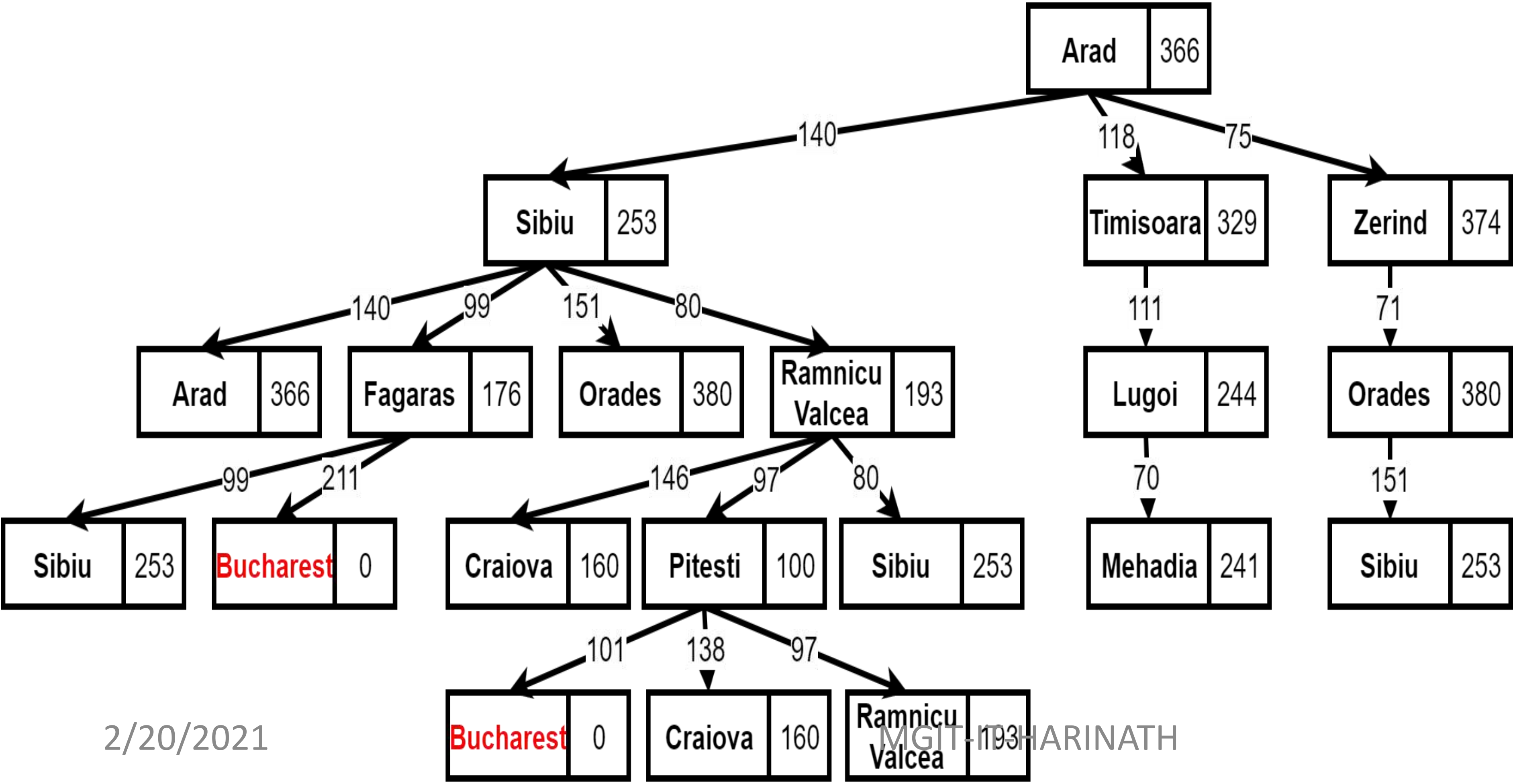
Arad



Arad Children

Sibiu

Arad

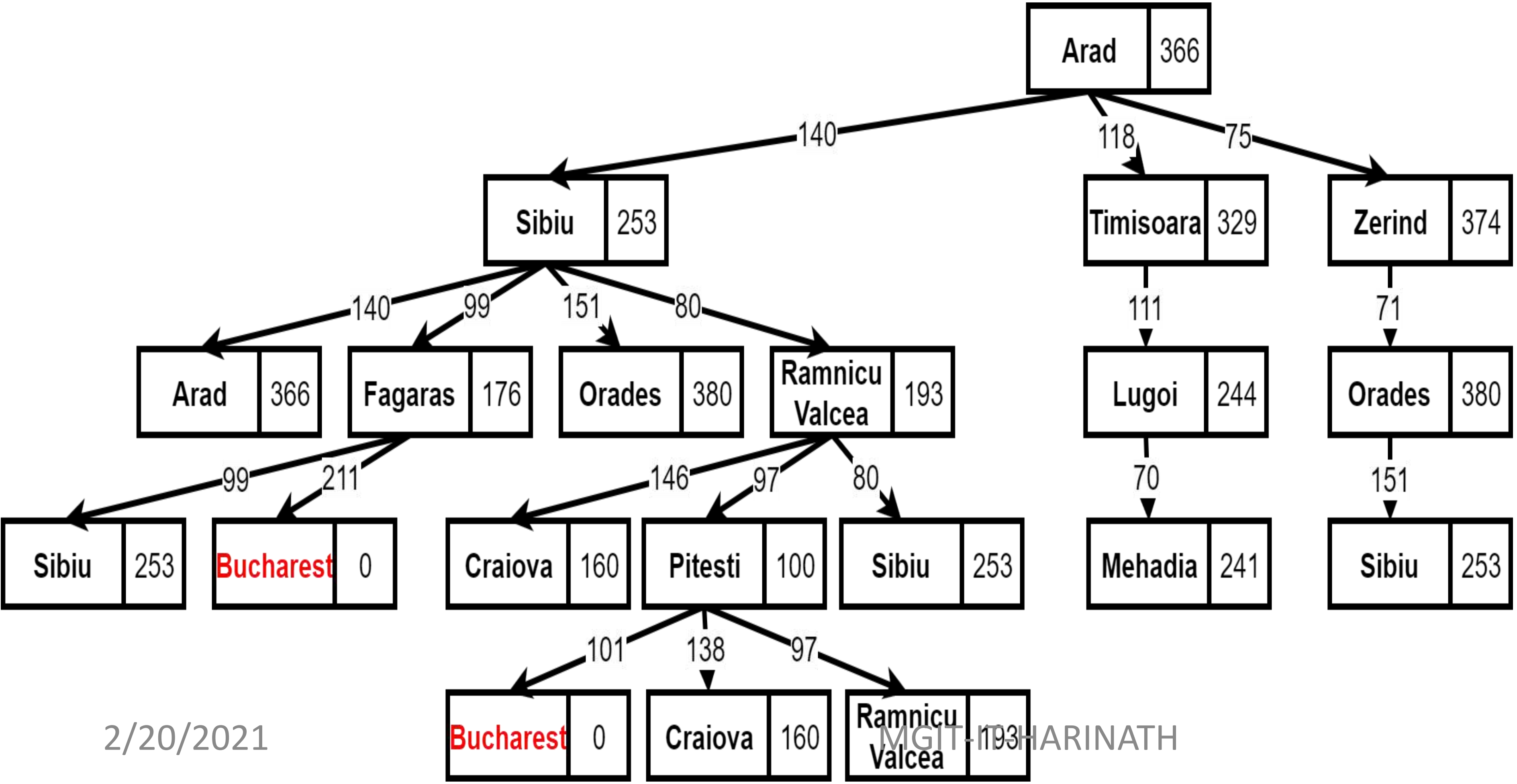


Arad Children

Sibiu

Timisoara

Arad



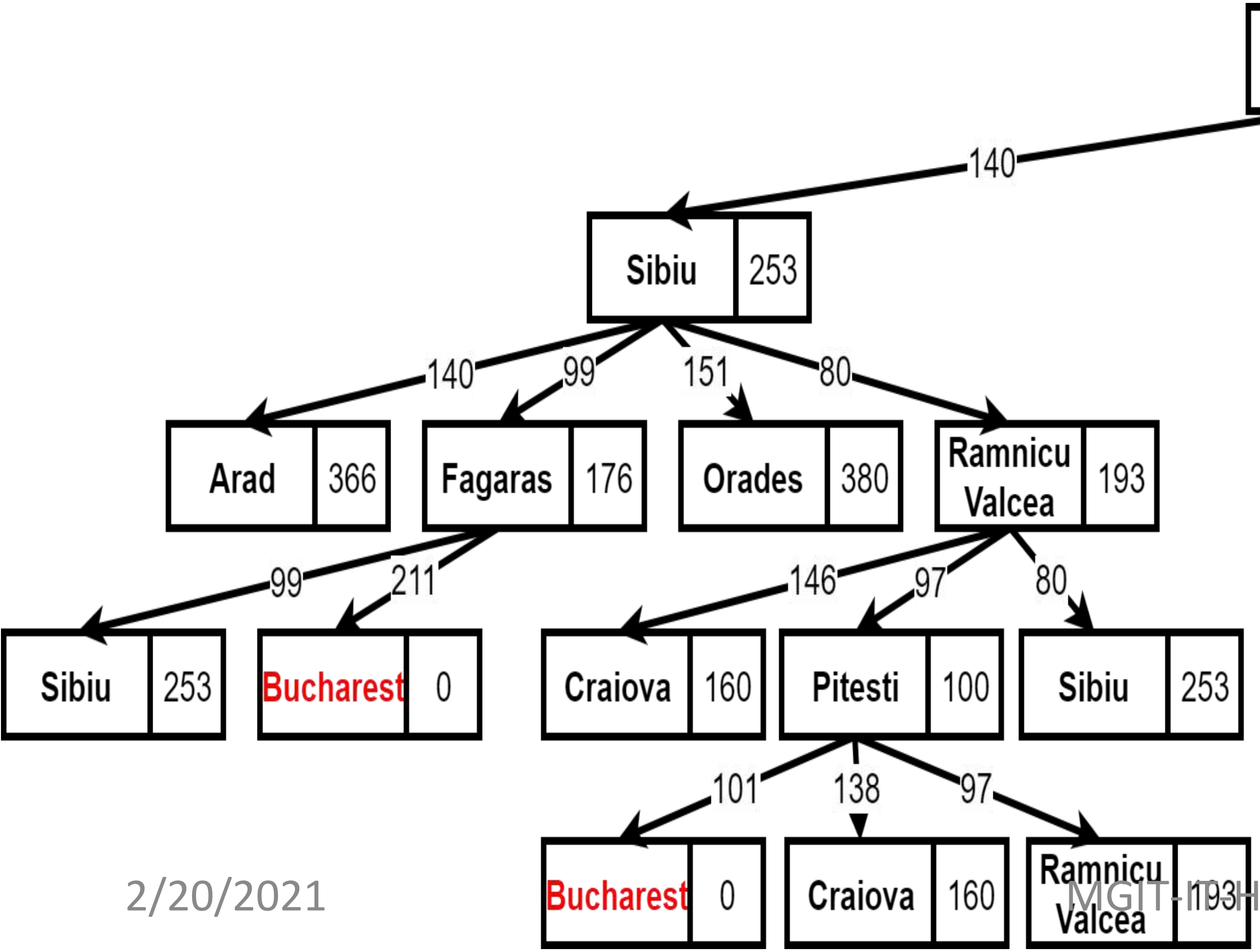
Arad Children

Sibiu

Timisoara

Zerind

Arad



Arad Children

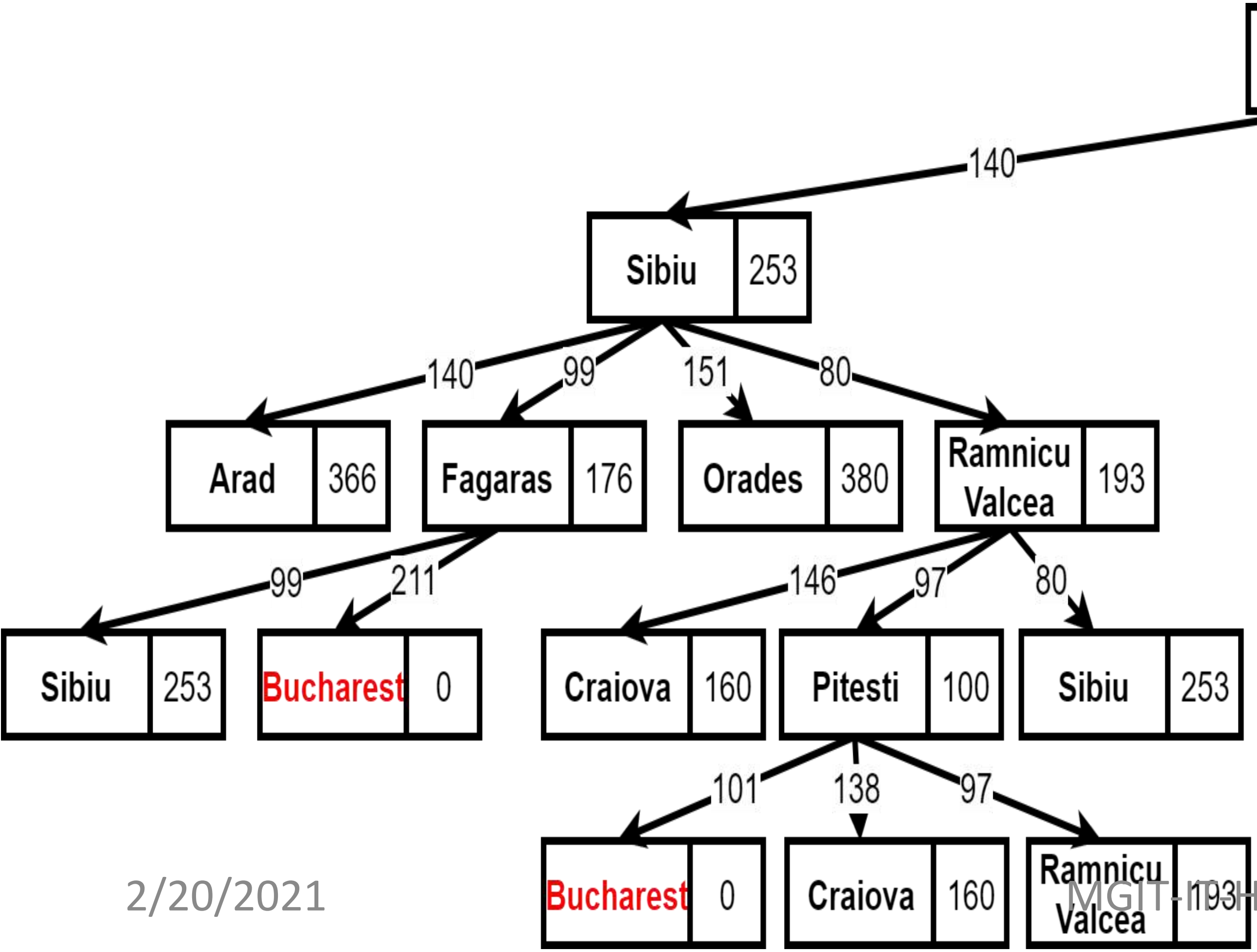
Sibiu

Timisoara

Zerind

Calculate Total Cost

Arad



Arad Children

Sibiu

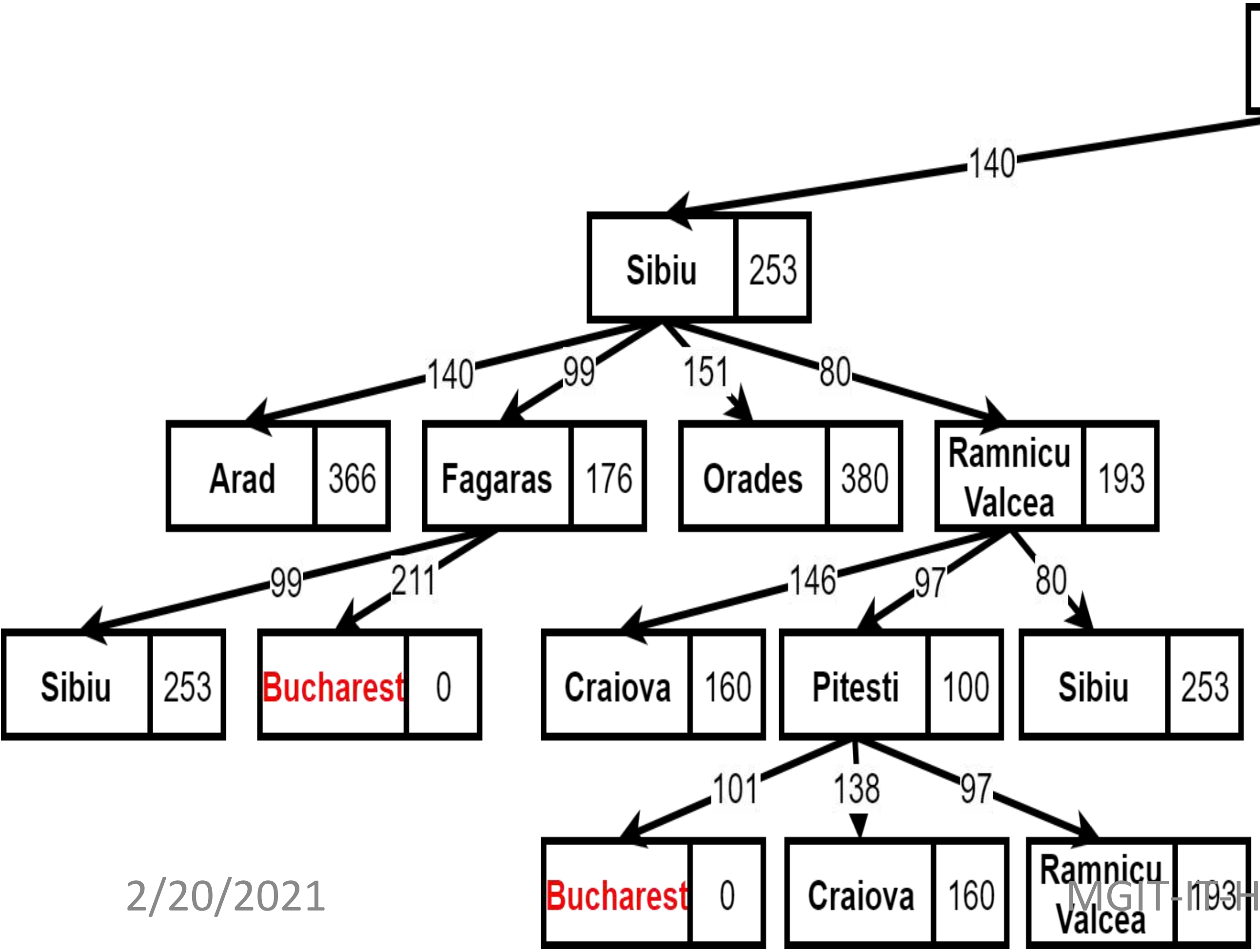
Timisoara

Zerind

Calculate Total Cost

Order Asc.

Arad



Arad Children

Sibiu

Timisoara

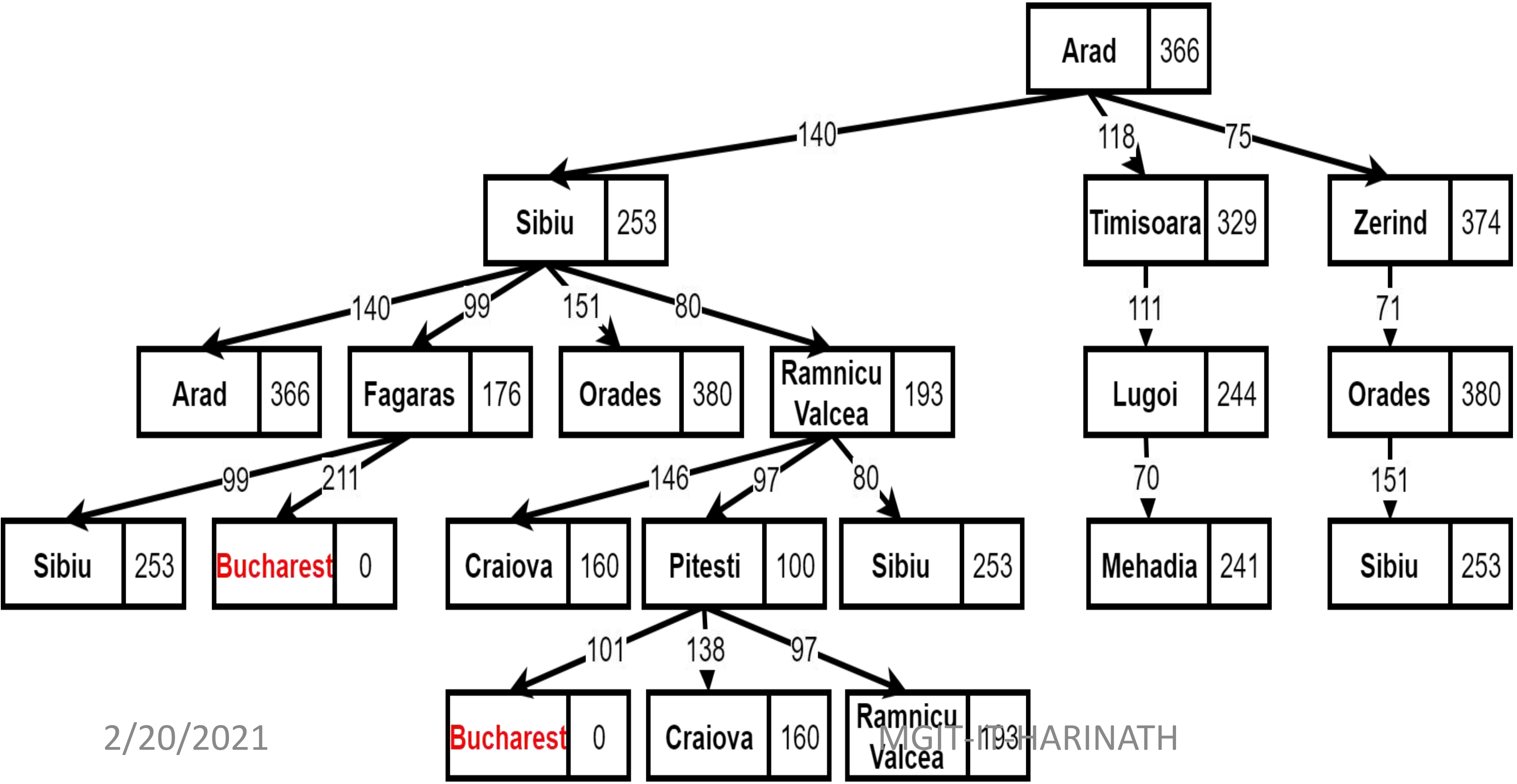
Zerind

Calculate Total Cost

Order Asc.

Select Min. Cost

Sibiu



Arad Children

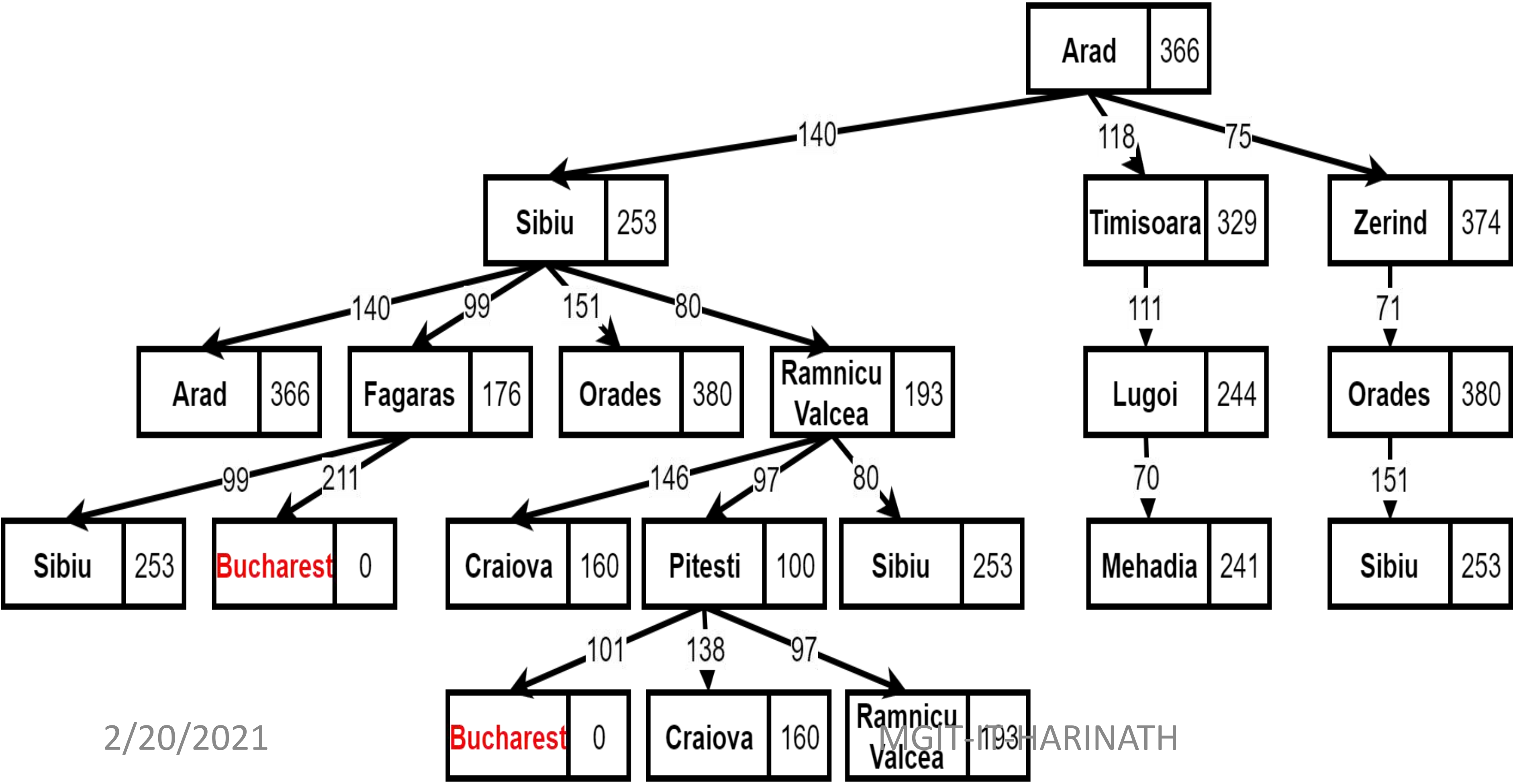
Sibiu

Timisoara

Zerind

Sibiu

Heuristic



Arad Children

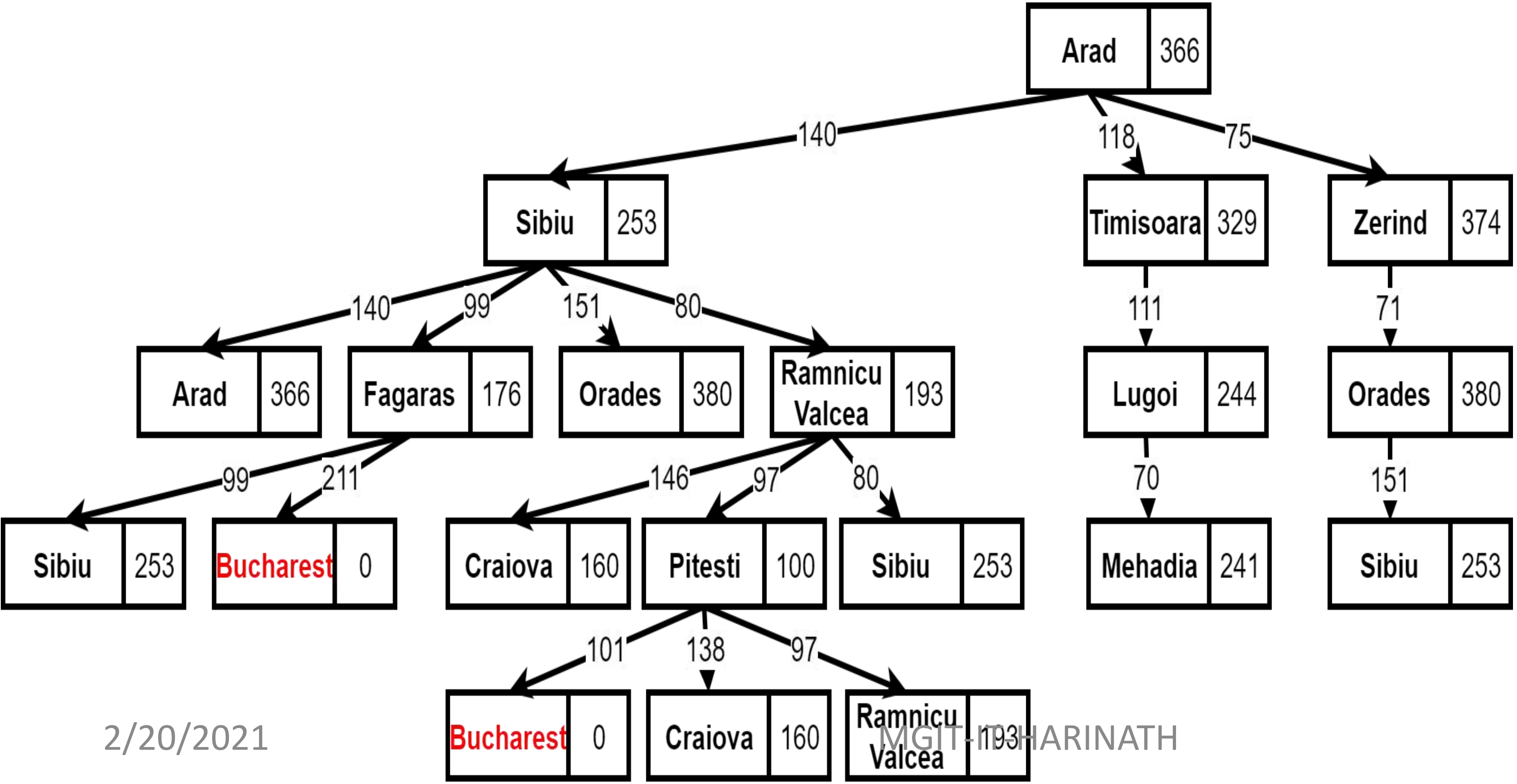
Sibiu

Timisoara

Zerind

Sibiu

Heuristic + **Cost**



Arad Children

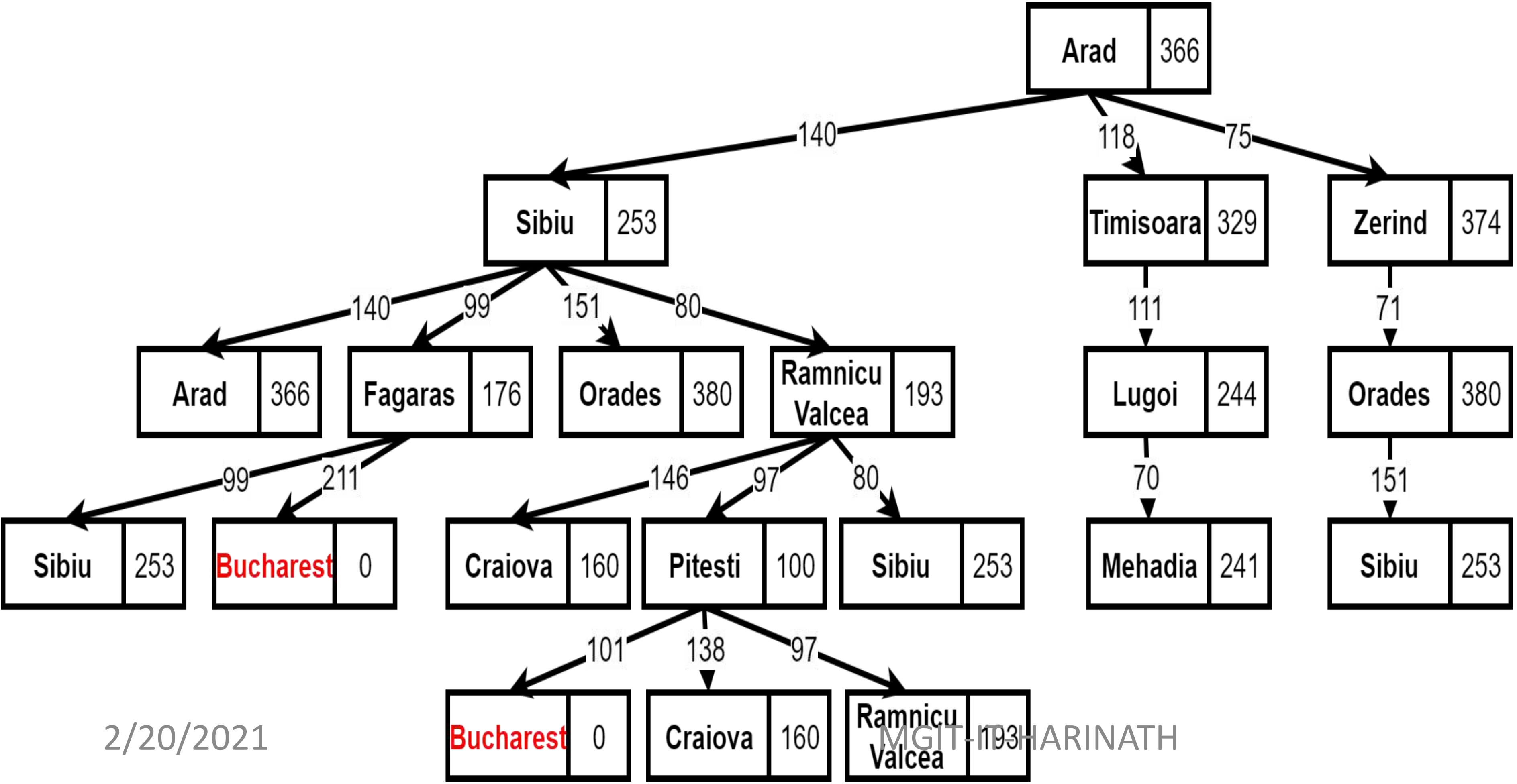
Sibiu

Timisoara

Zerind

Sibiu

Heuristic + **Cost** = **Total**



Arad Children

Sibiu

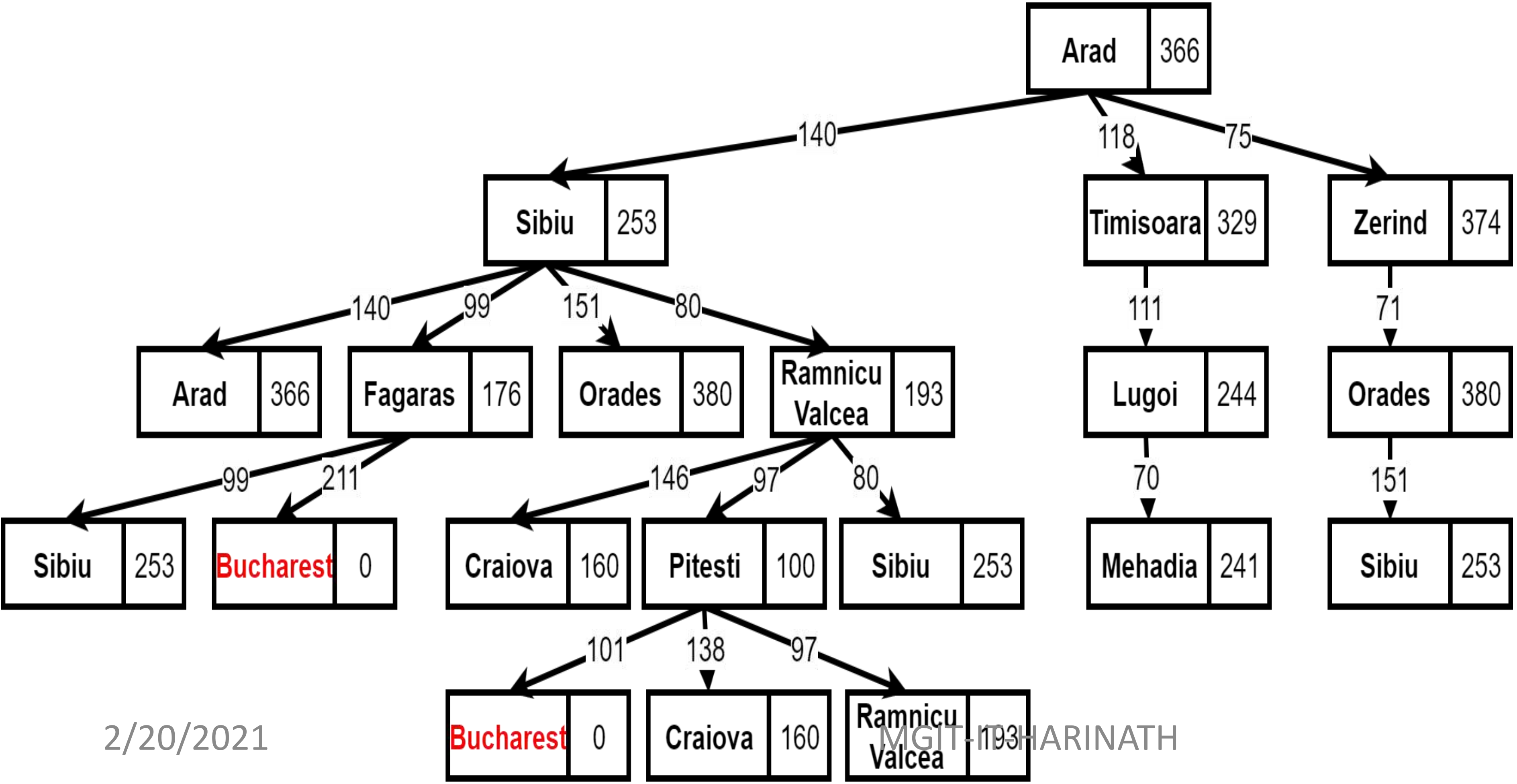
Timisoara

Zerind

Sibiu

Heuristic + Cost = Total

253



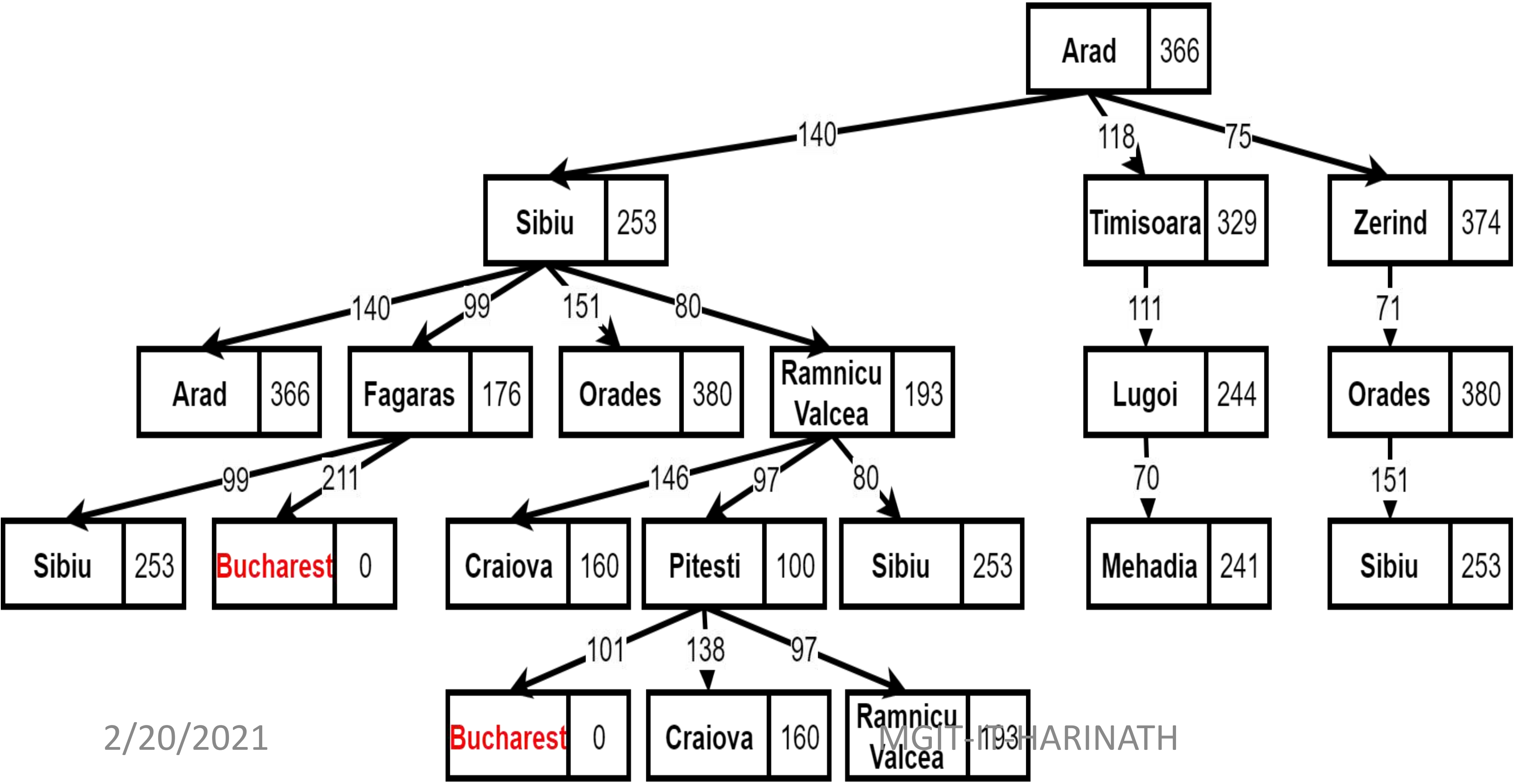
Arad Children

- Sibiu
- Timisoara
- Zerind

Sibiu

Heuristic + Cost = Total

253 + 140

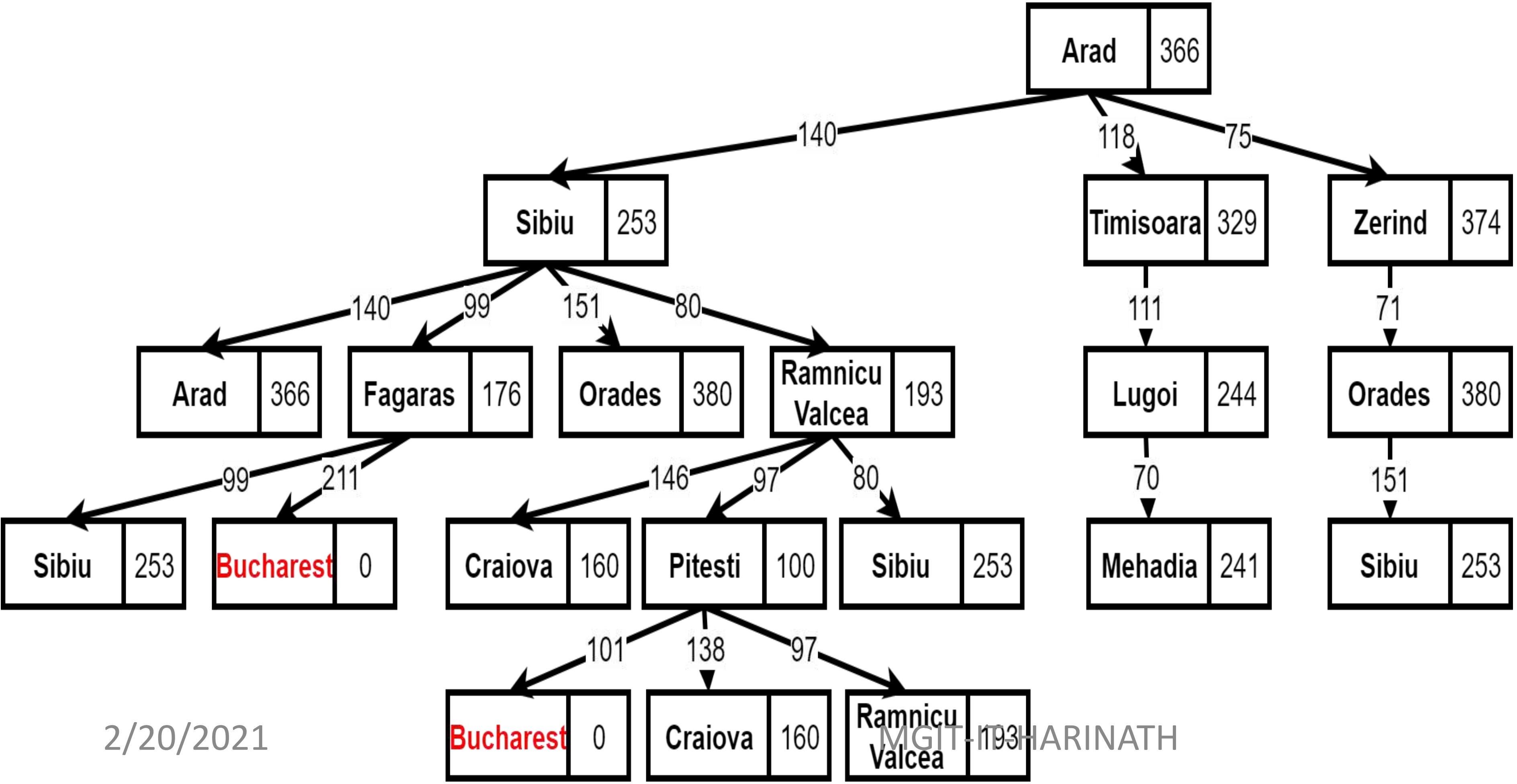


Arad Children

- Sibiu
- Timisoara
- Zerind

Sibiu

Heuristic	+	Cost	=	Total
253	+	140	=	393

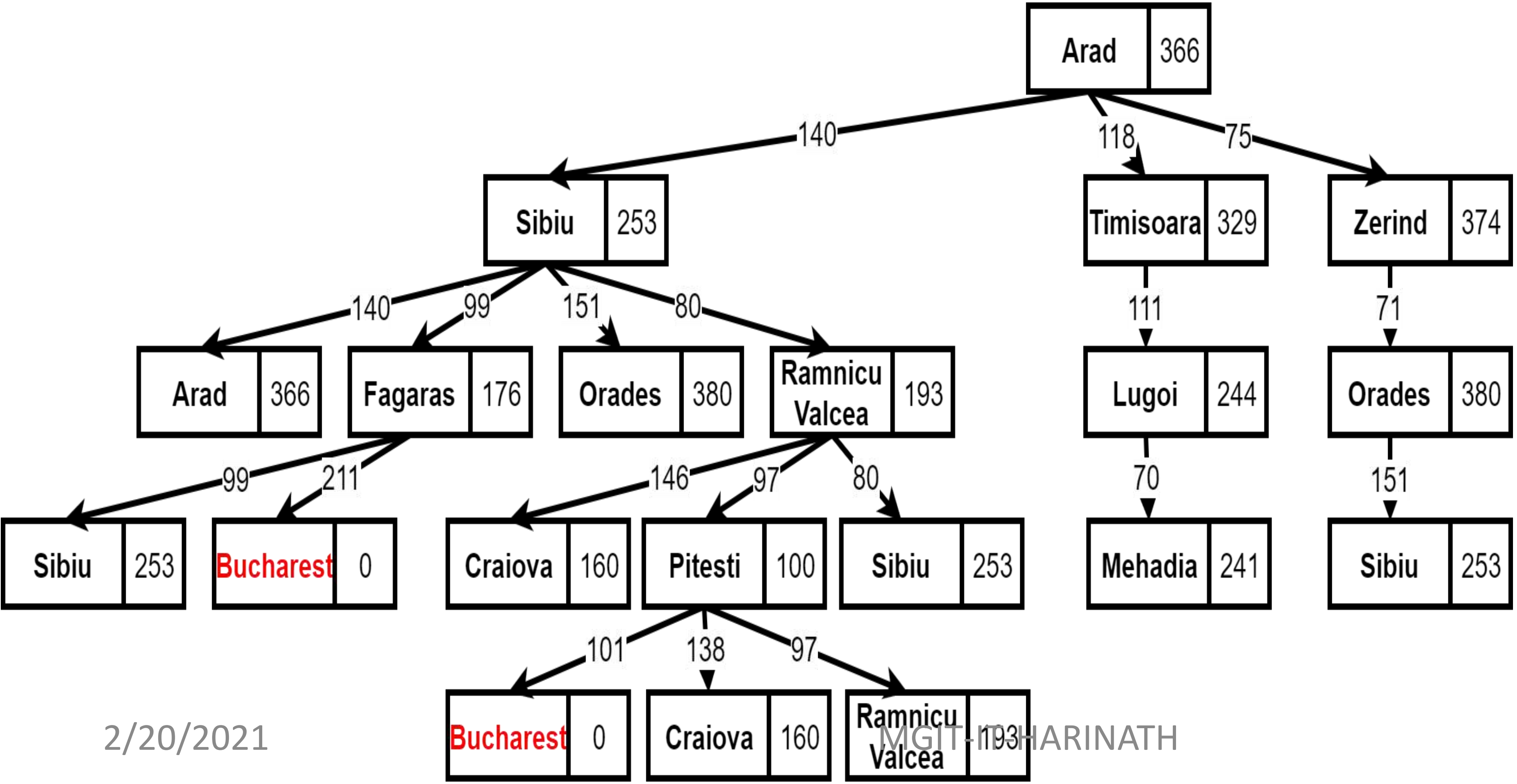


Arad Children

- Sibiu
- Timisoara
- Zerind

Sibiu

Heuristic	+	Cost	=	Total
253	+	140	=	393

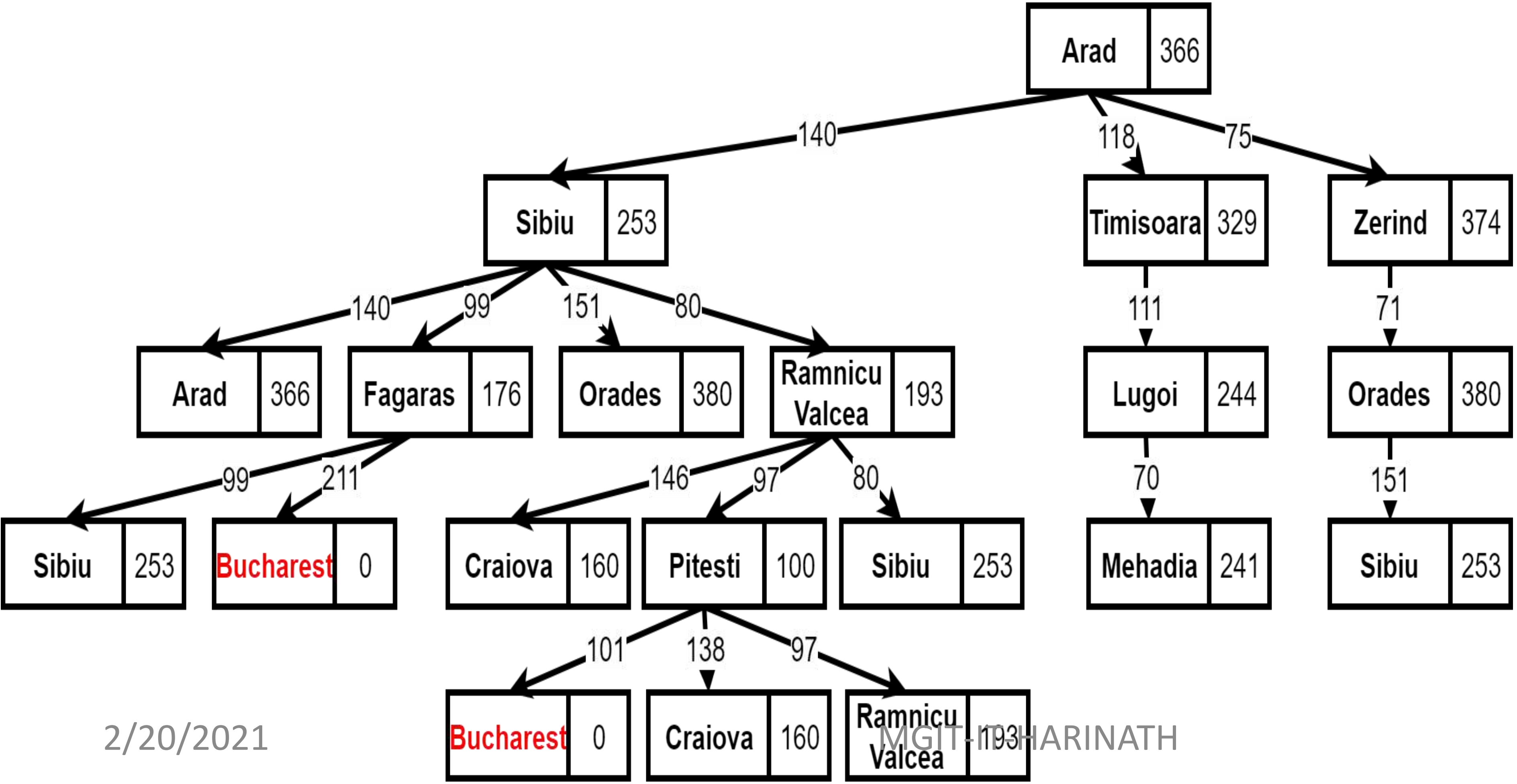


Arad Children

- Sibiu
- Timisoara
- Zerind

Sibiu

Heuristic	+	Cost	=	Total
253	+	140	=	393

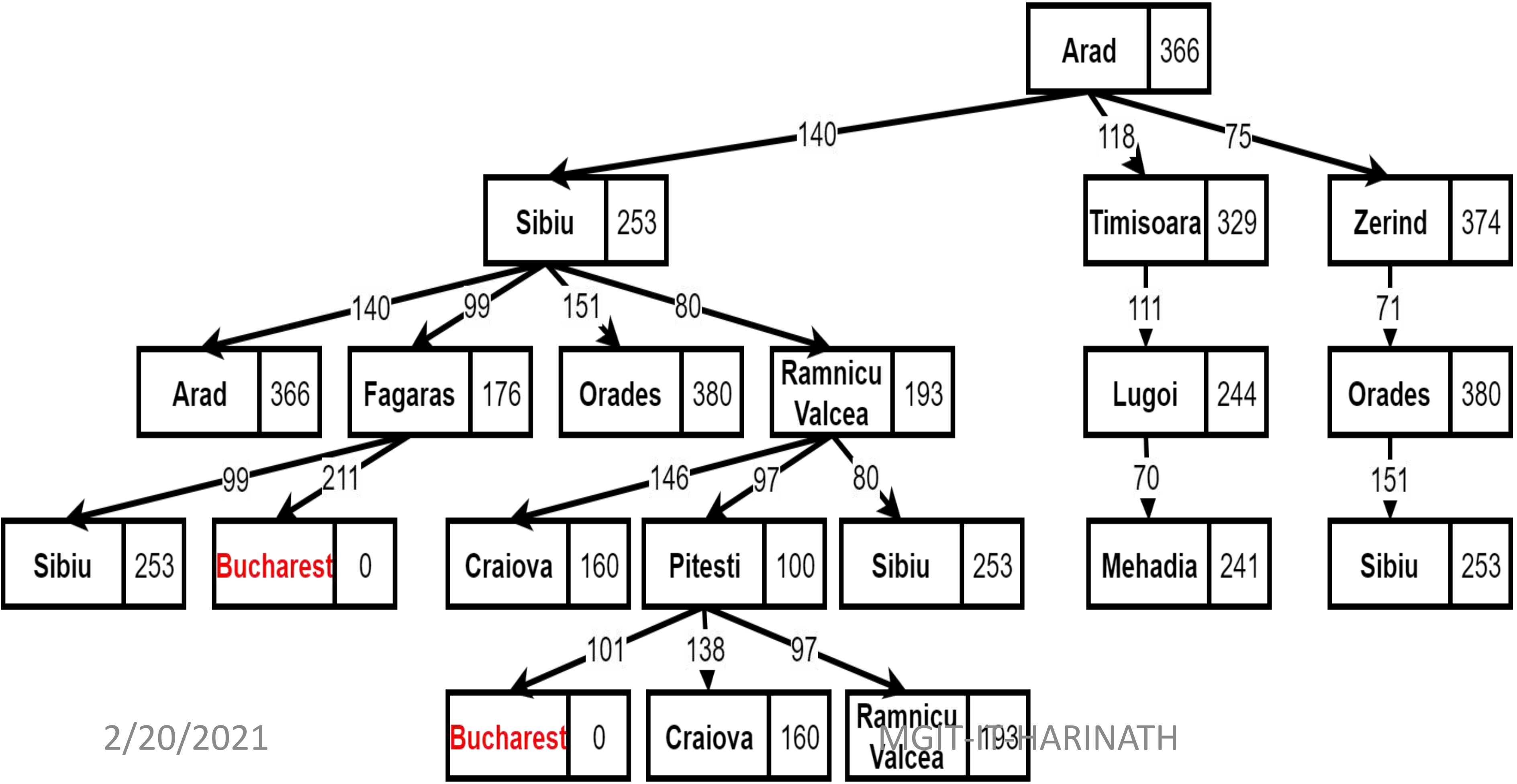


Arad Children

Sibiu	393
Timisoara	
Zerind	

Timisoara

Heuristic + Cost = Total



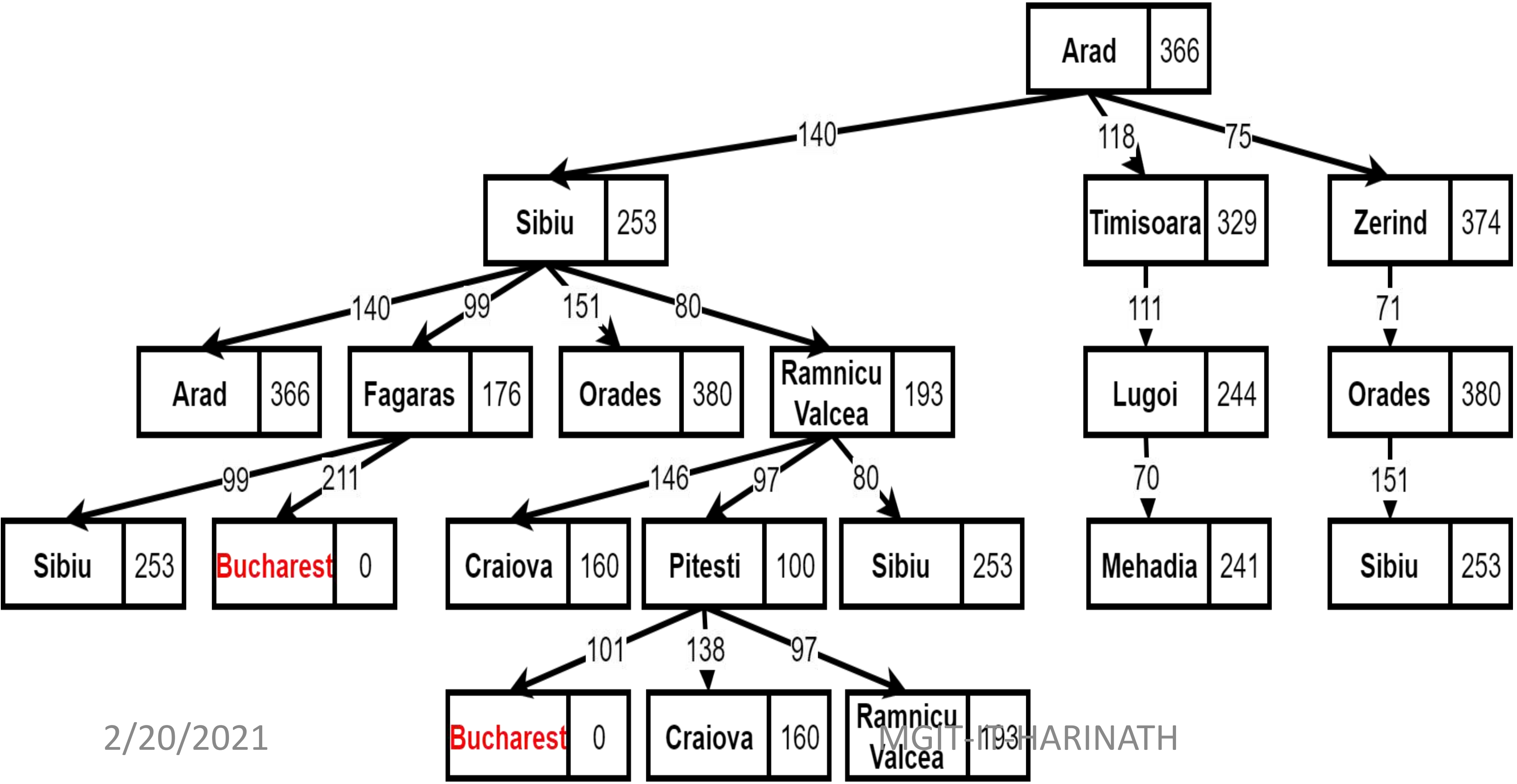
Arad Children

Sibiu	393
Timisoara	
Zerind	

Timisoara

Heuristic + **Cost** = **Total**

329



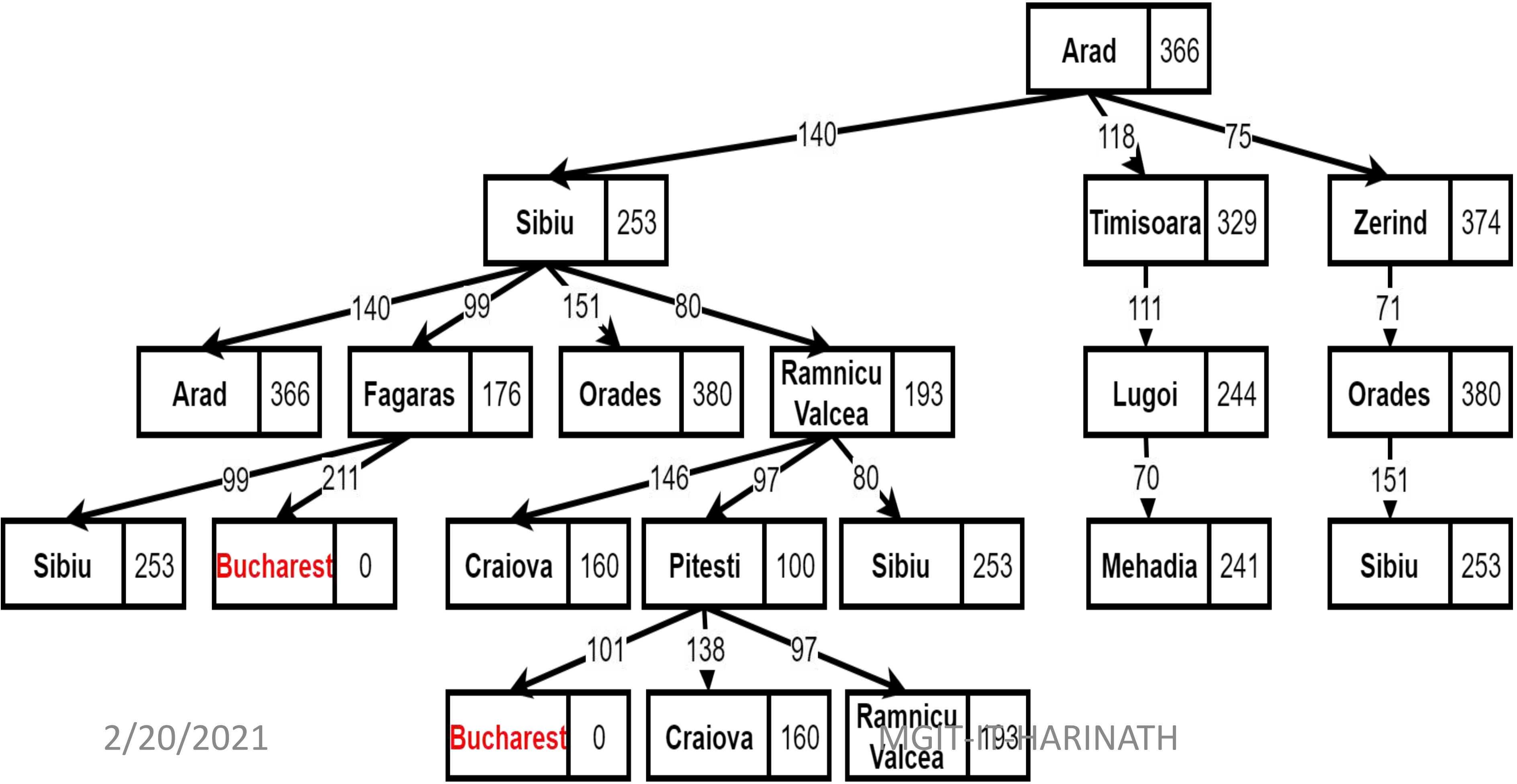
Arad Children

Sibiu	393
Timisoara	
Zerind	

Timisoara

Heuristic + **Cost** = **Total**

329 + 118

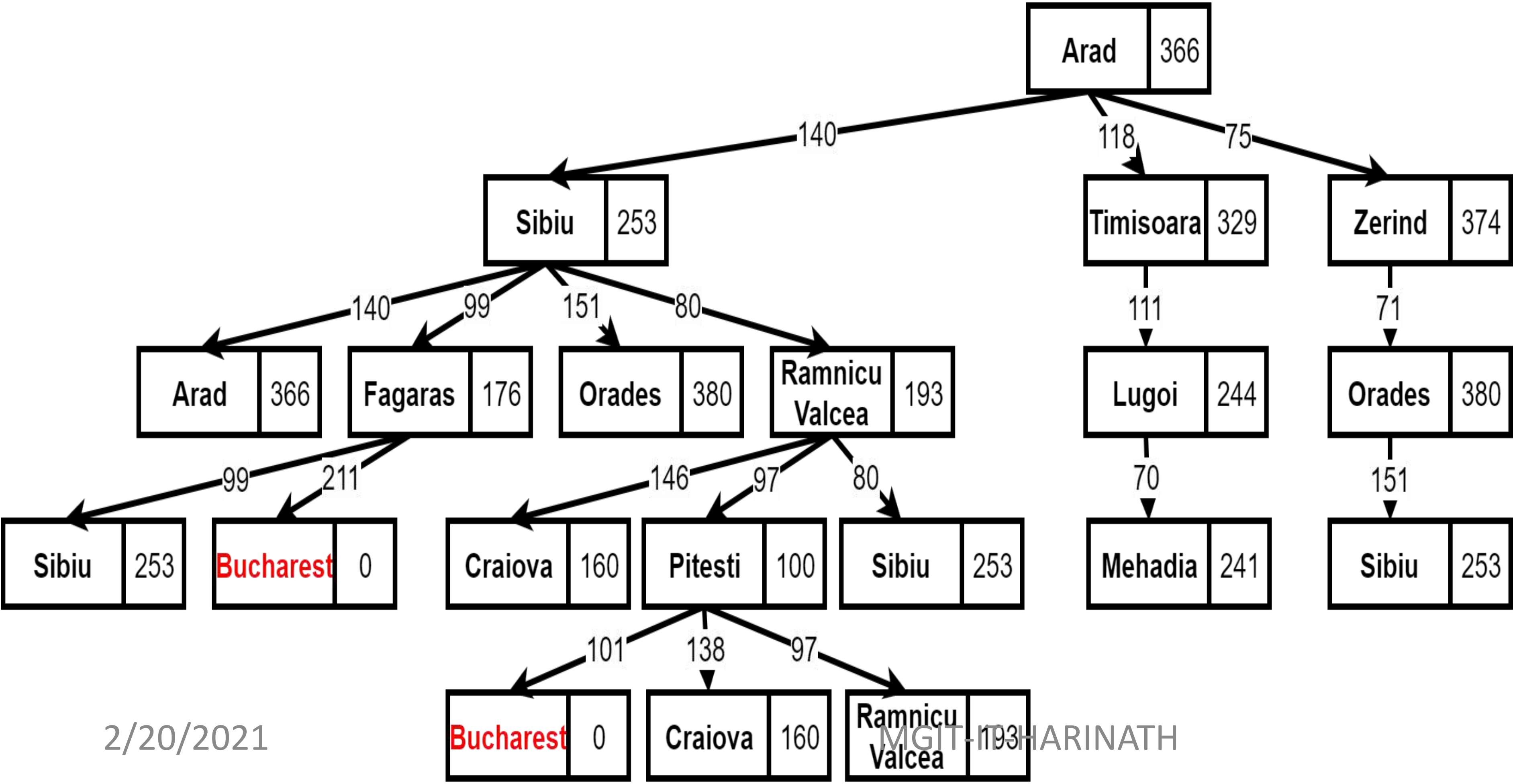


Arad Children

Sibiu	393
Timisoara	
Zerind	

Timisoara

Heuristic	+	Cost	=	Total
329	+	118	=	447

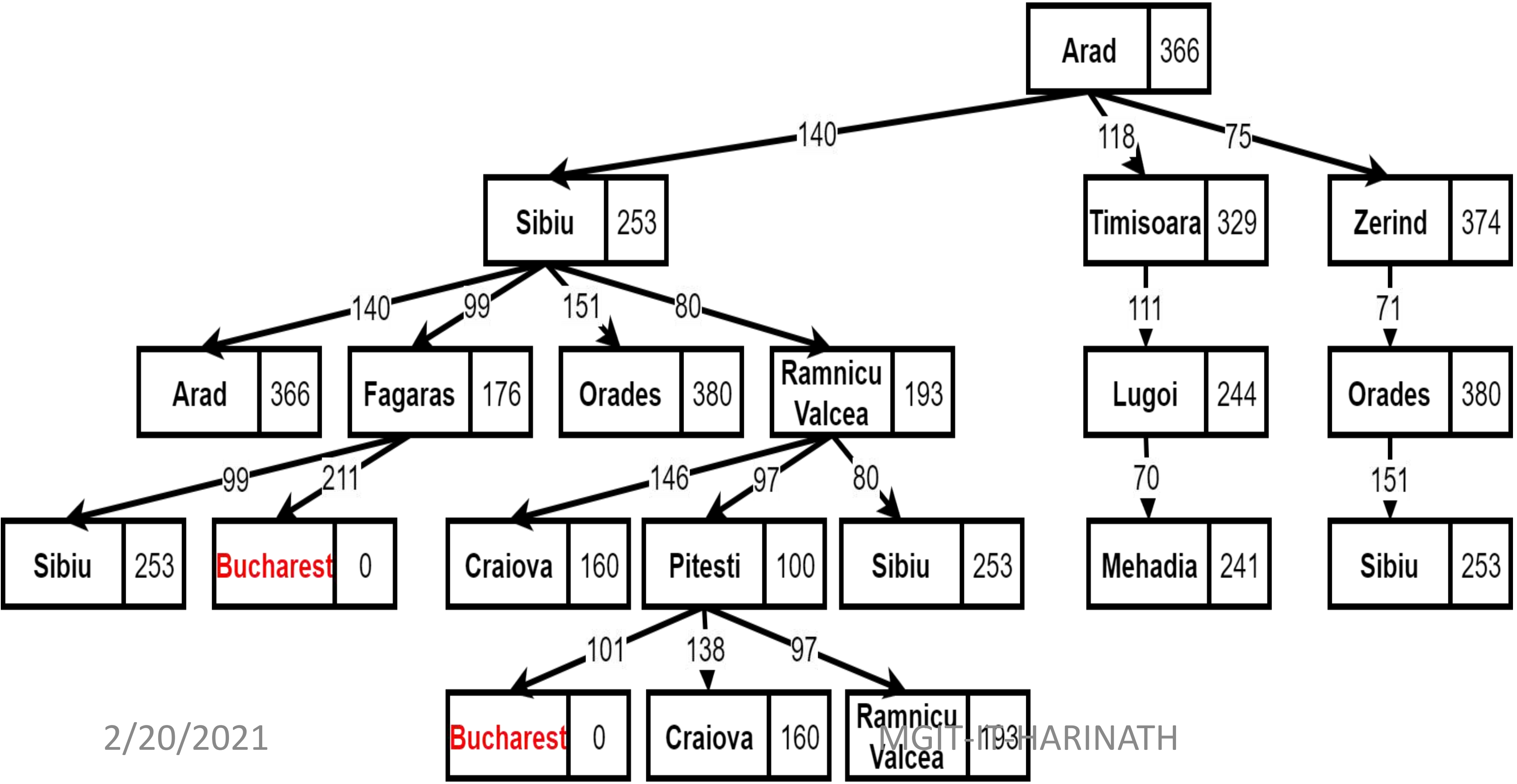


Arad Children

Sibiu	393
Timisoara	
Zerind	

Timisoara

Heuristic	+	Cost	=	Total
329	+	118	=	447

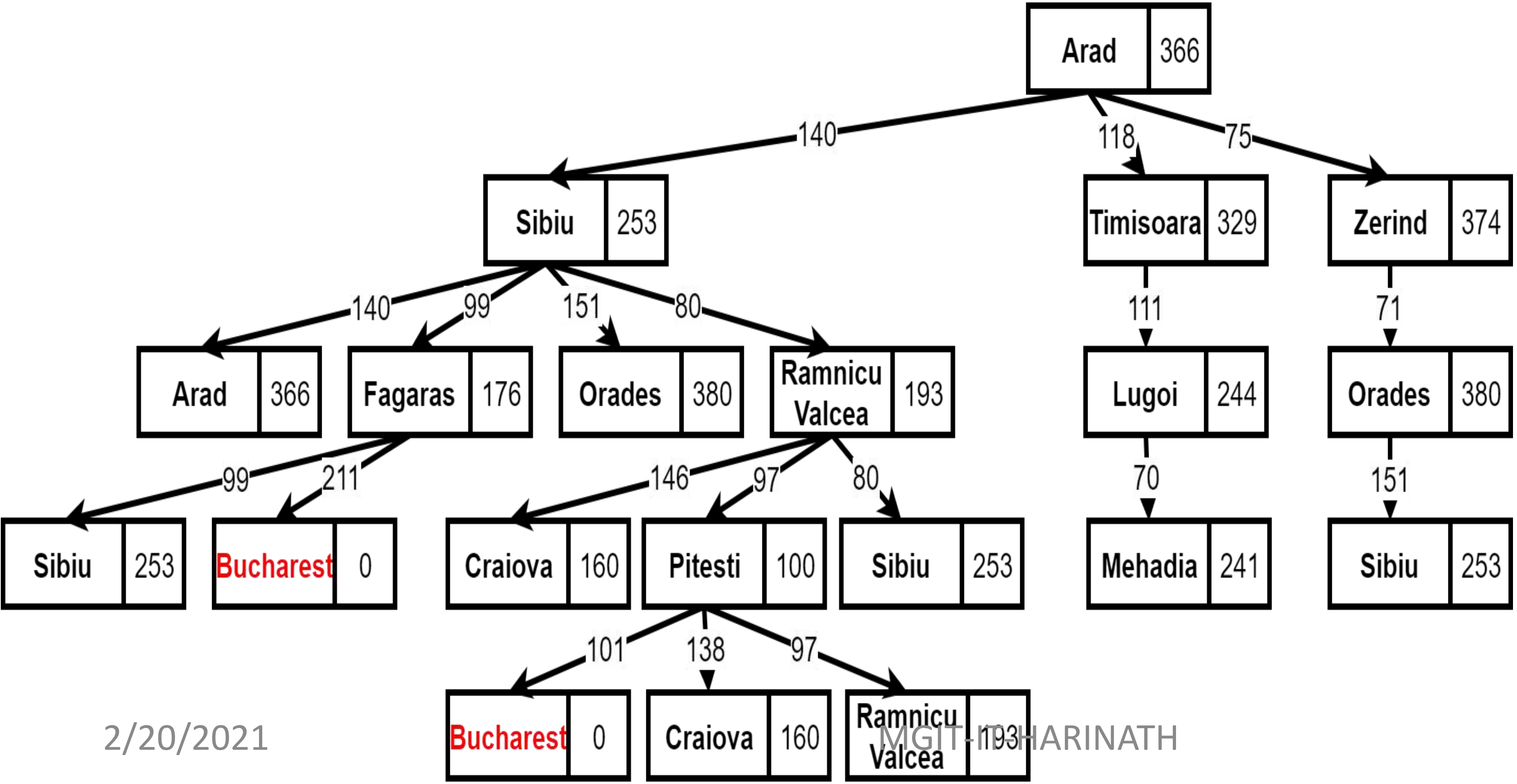


Arad Children

Sibiu	393
Timisoara	447
Zerind	

Zerind

Heuristic + Cost = Total

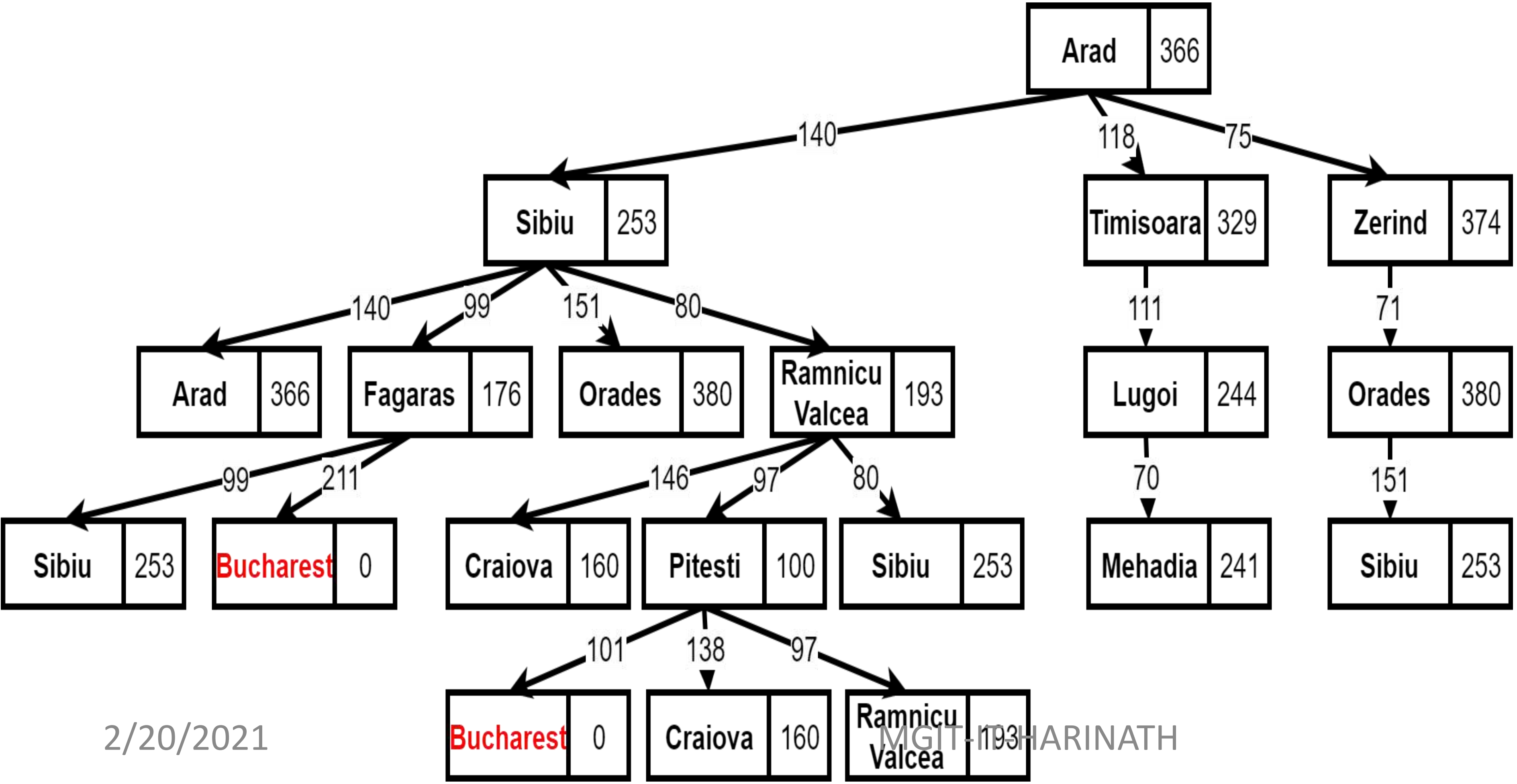


Arad Children	
Sibiu	393
Timisoara	447
Zerind	

Zerind

Heuristic + **Cost** = **Total**

374



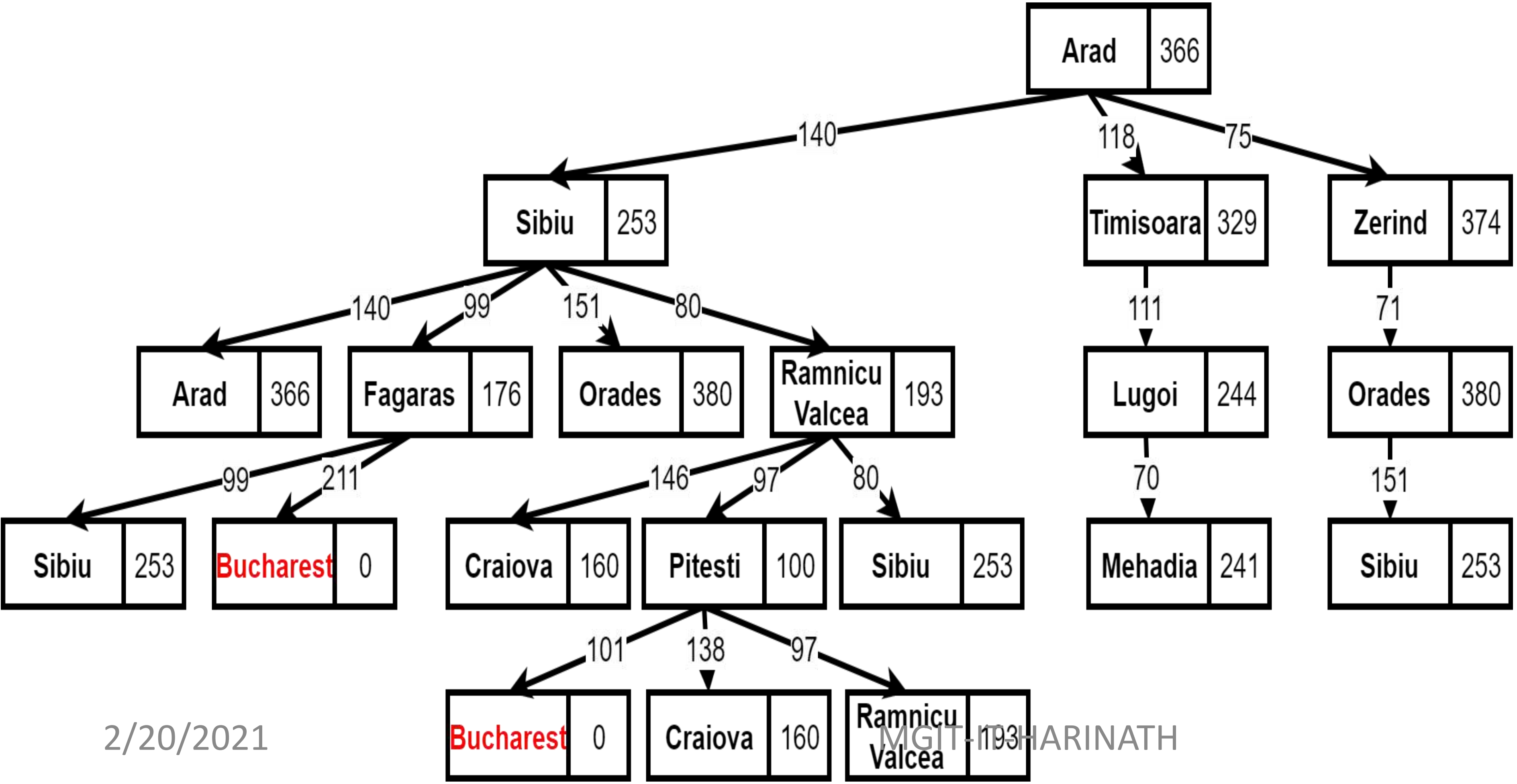
Arad Children

Sibiu	393
Timisoara	447
Zerind	

Zerind

Heuristic + Cost = Total

374 + 75

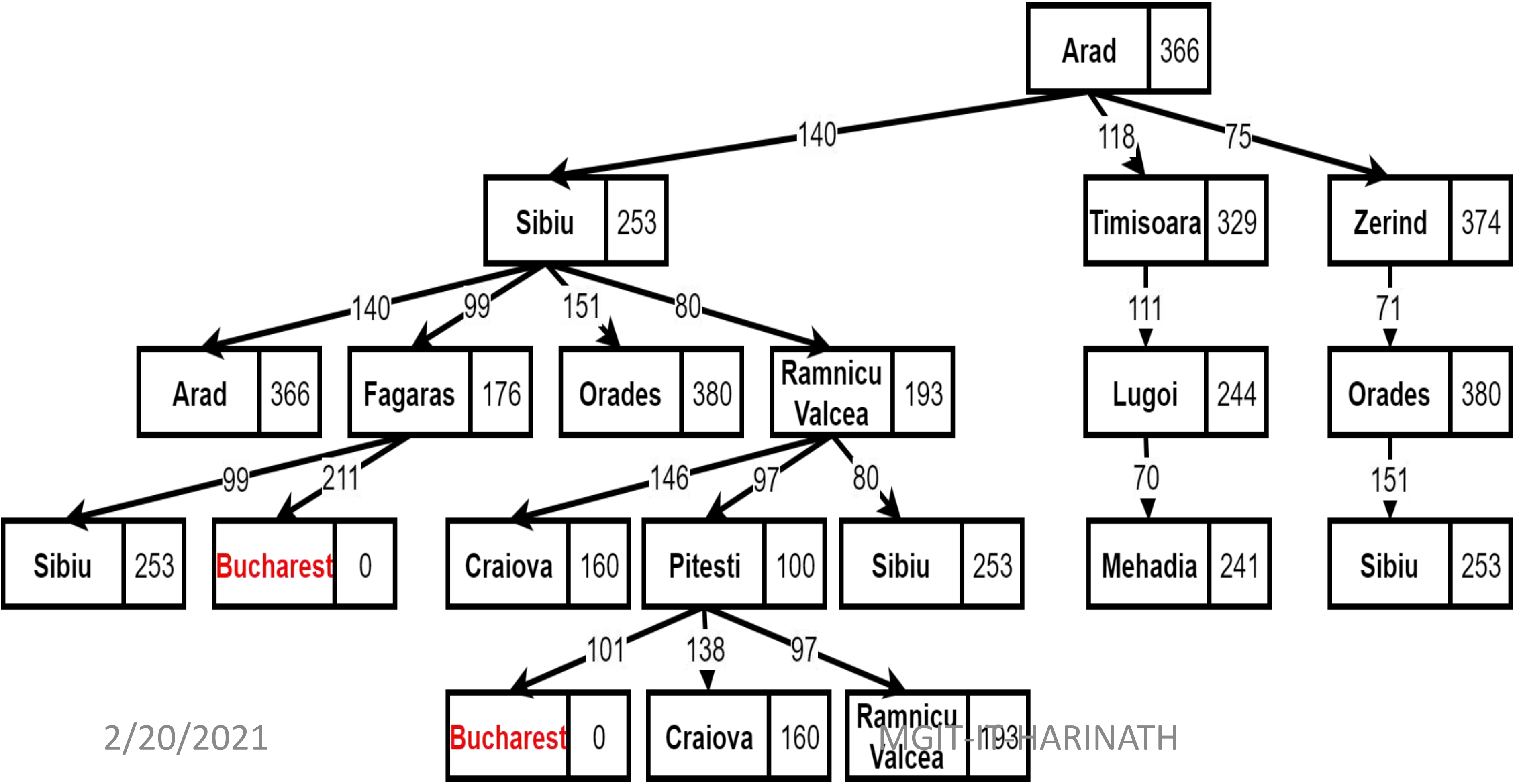


Arad Children

Sibiu	393
Timisoara	447
Zerind	

Zerind

Heuristic	+	Cost	=	Total
374	+	75	=	449

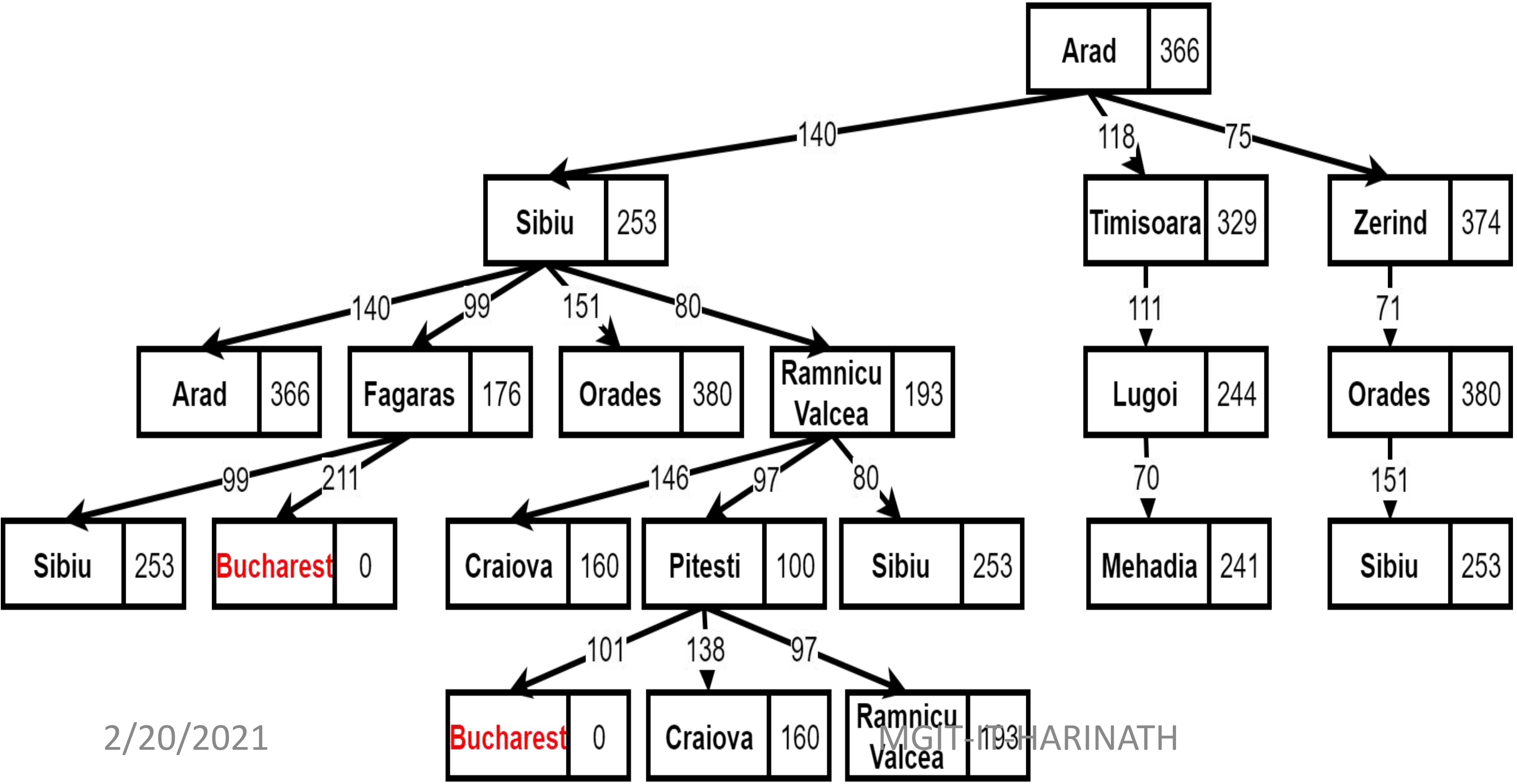


Arad Children

Sibiu	393
Timisoara	447
Zerind	

Zerind

Heuristic	+	Cost	=	Total
374	+	75	=	449



Arad Children	
Sibiu	393
Timisoara	447
Zerind	449

Current Queue

Sibiu
393

Current Queue

Sibiu	Timisoar
393	447

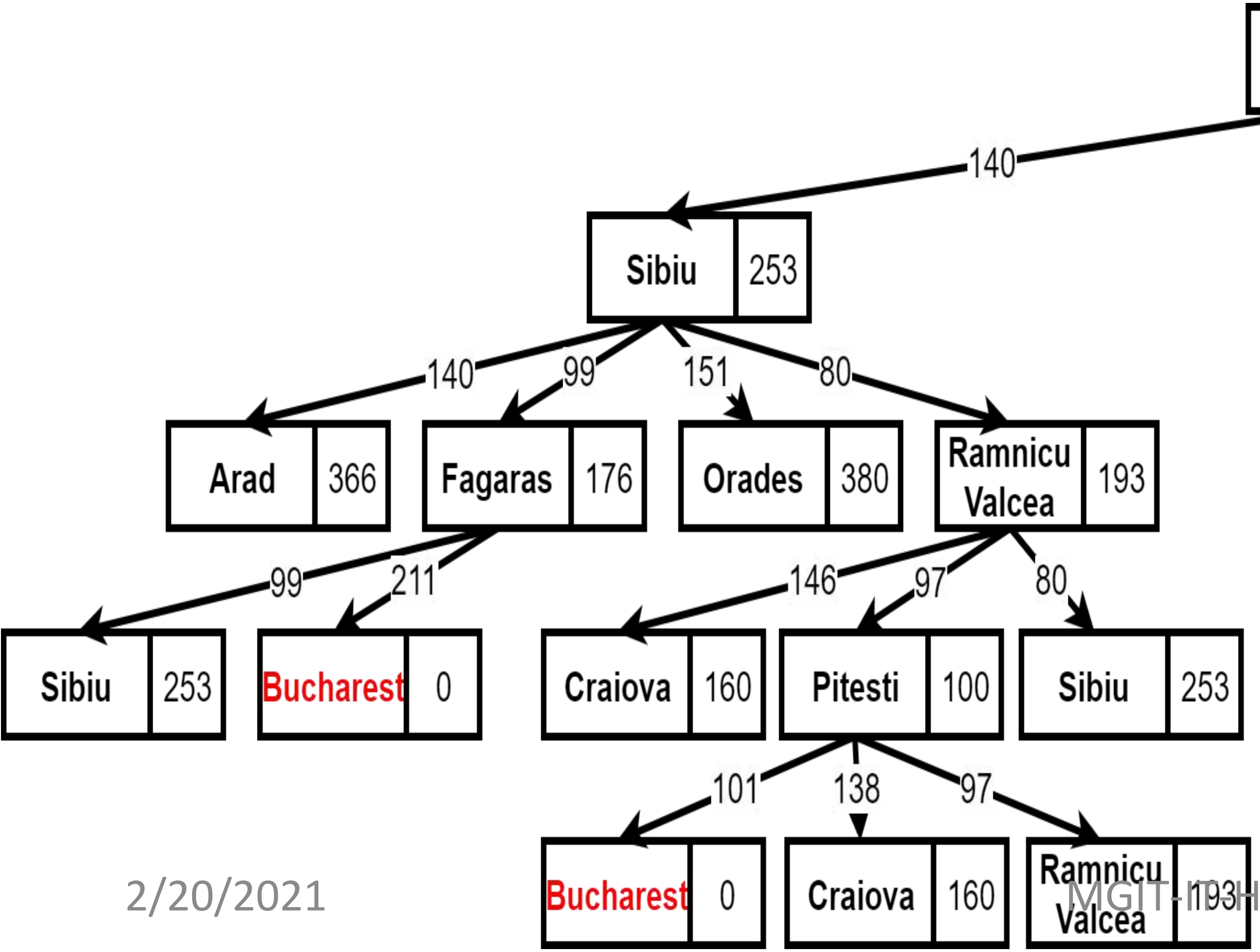
Current Queue

Sibiu	Timisoar	Zerin
393	447	449

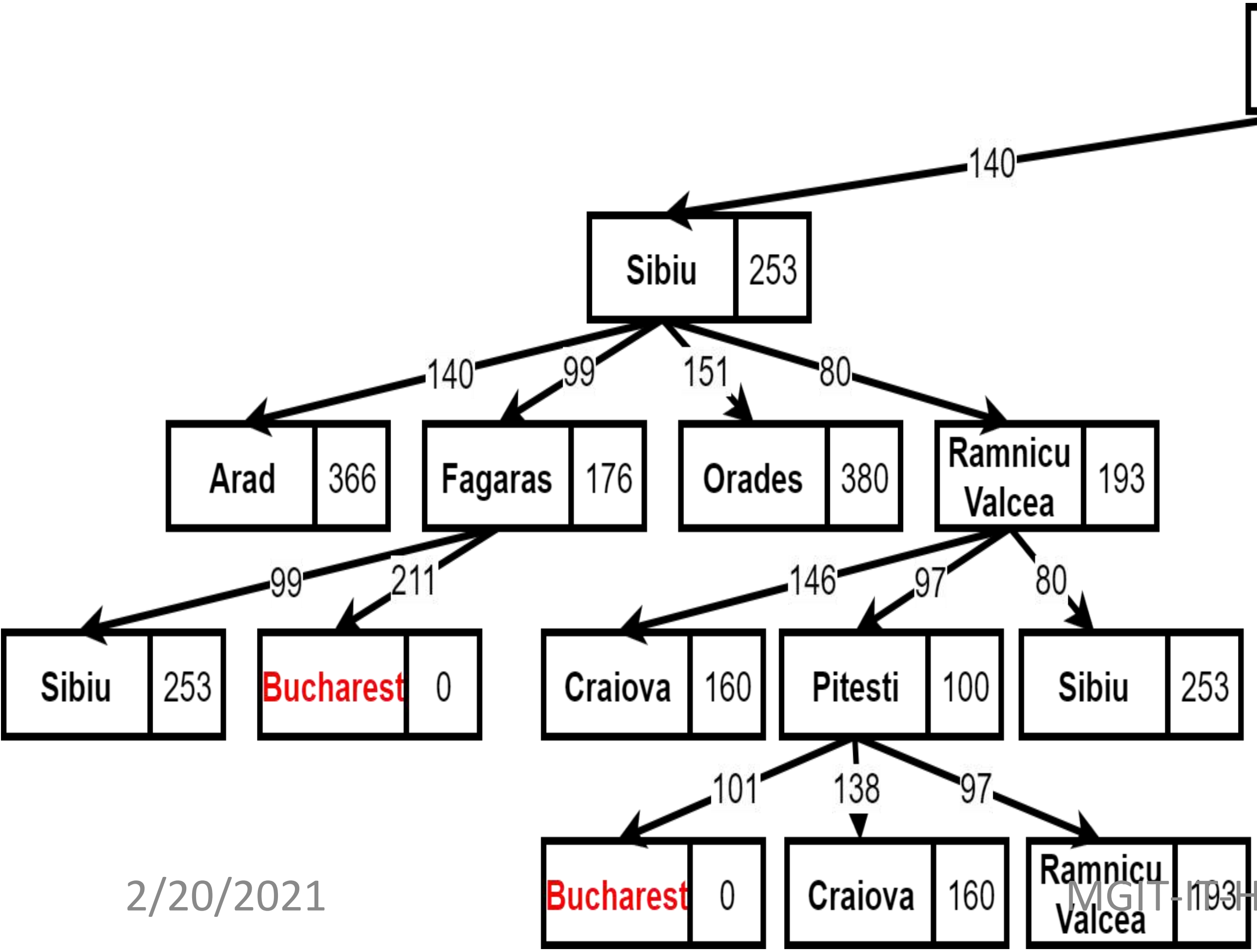
Current Queue

Sibiu	Timisoar	Zerin
393	447	449

Sibiu

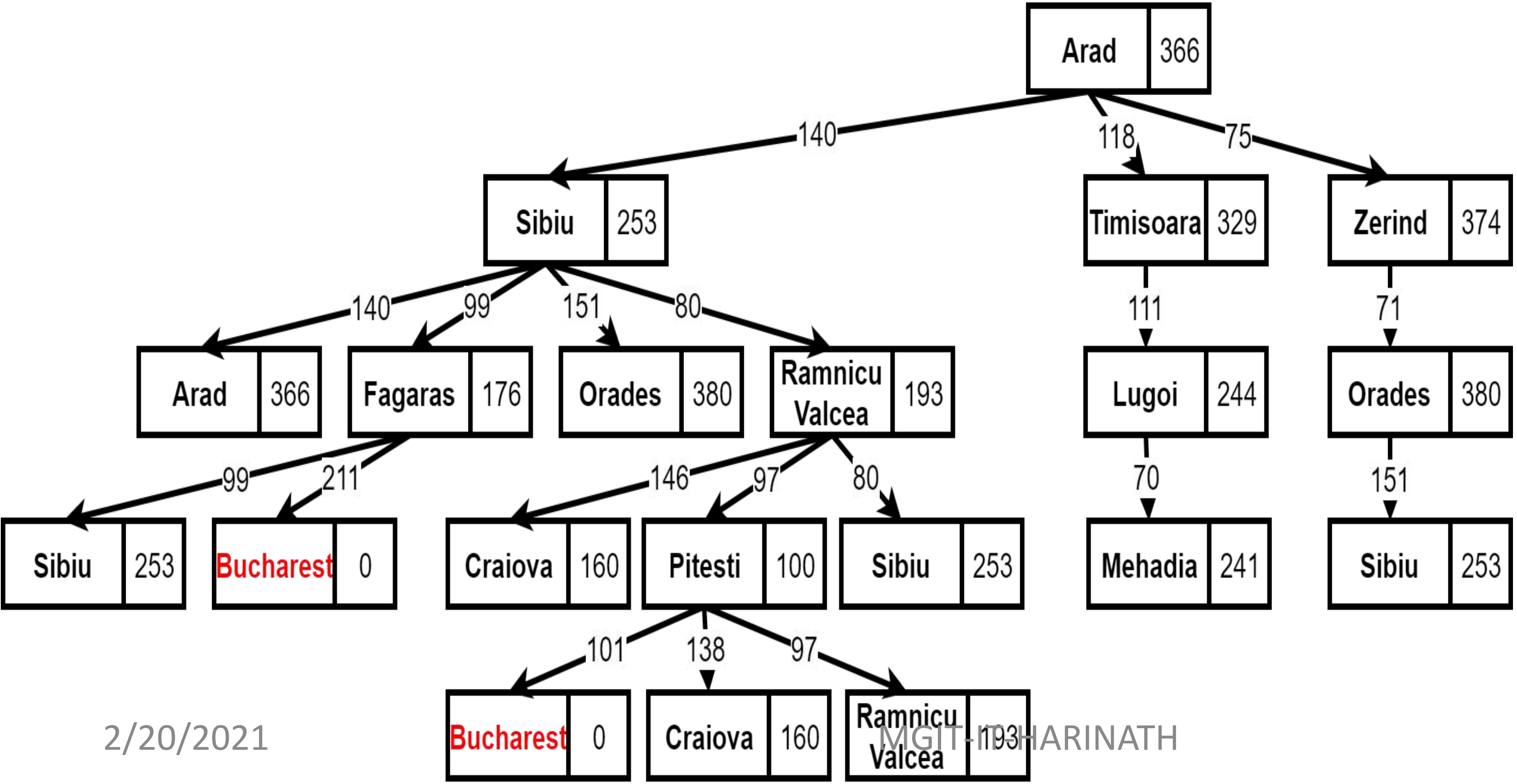

Arad 366
140
118
75
Sibiu 253
Timisoara 329
Zerind 374
140
99
151
80
111
71
Arad 366
Fagaras 176
Orades 380
Ramnicu Valcea 193
Lugoi 244
Orades 380
99
211
146
97
80
70
151
Sibiu 253
Bucharest 0
Craiova 160
Pitesti 100
Sibiu 253
Mehadia 241
Sibiu 253
101
138
97
Bucharest 0
Craiova 160
Ramnicu Valcea 193

Sibiu



Arad 366
140
118
75
Sibiu 253
Timisoara 329
Zerind 374
140
99
151
80
111
71
Arad 366
Fagaras 176
Orades 380
Ramnicu Valcea 193
Lugoi 244
Orades 380
99
211
146
97
80
70
151
Sibiu 253
Bucharest 0
Craiova 160
Pitesti 100
Sibiu 253
Mehadia 241
Sibiu 253
101
138
97
Bucharest 0
Craiova 160
Ramnicu Valcea 193

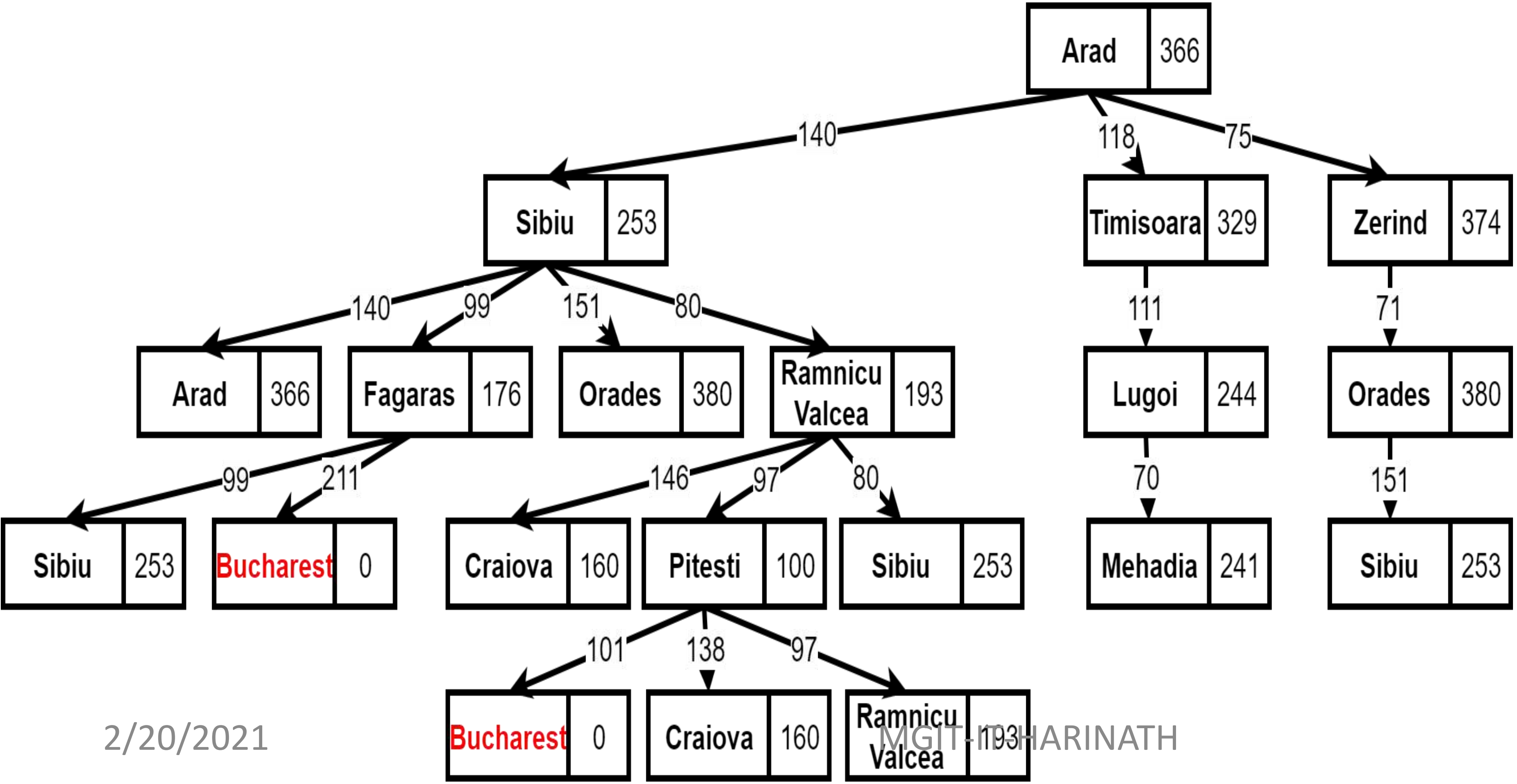
Sibiu



Sibiu Children

Arad

Sibiu

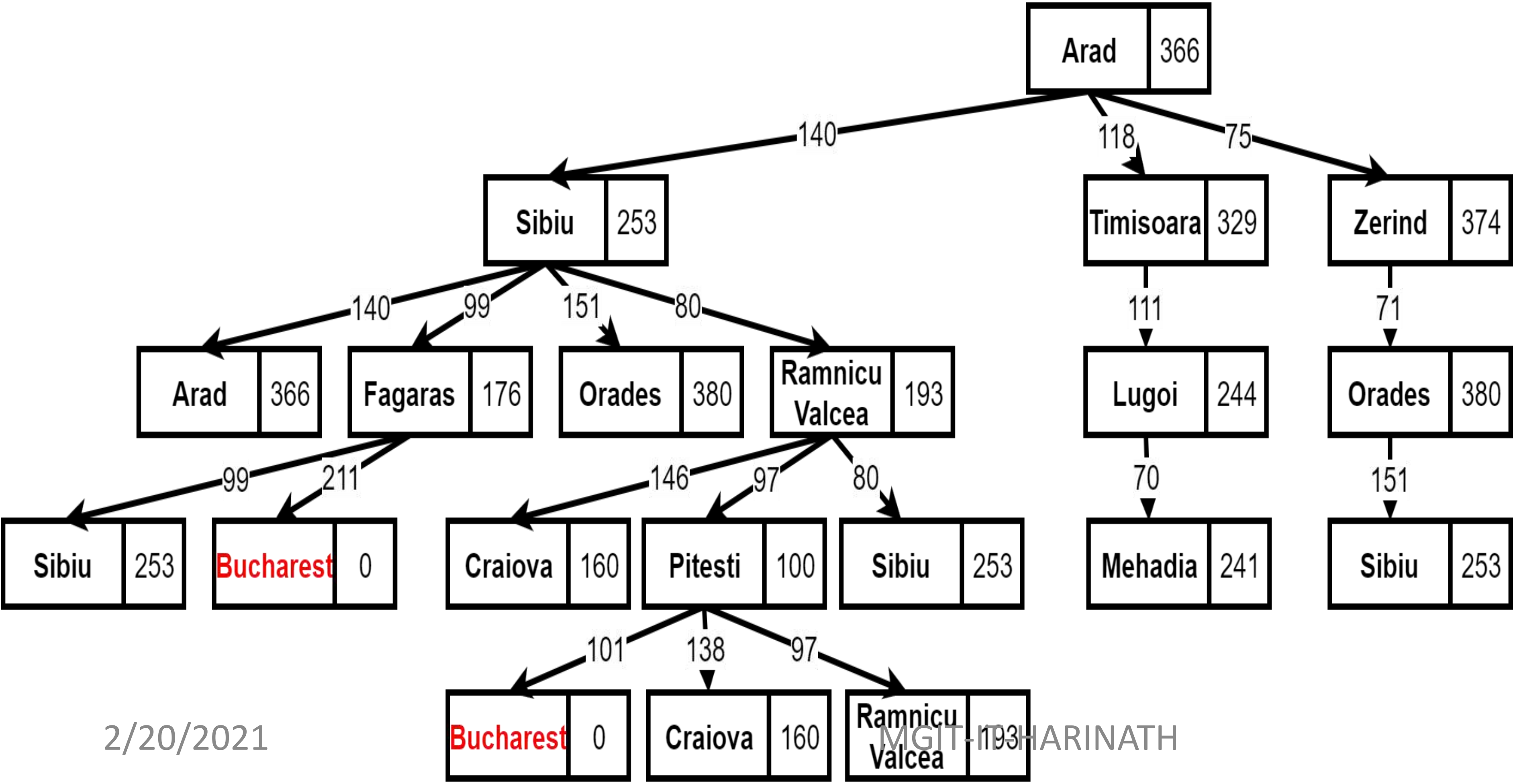


Sibiu Children

Arad

Fagaras

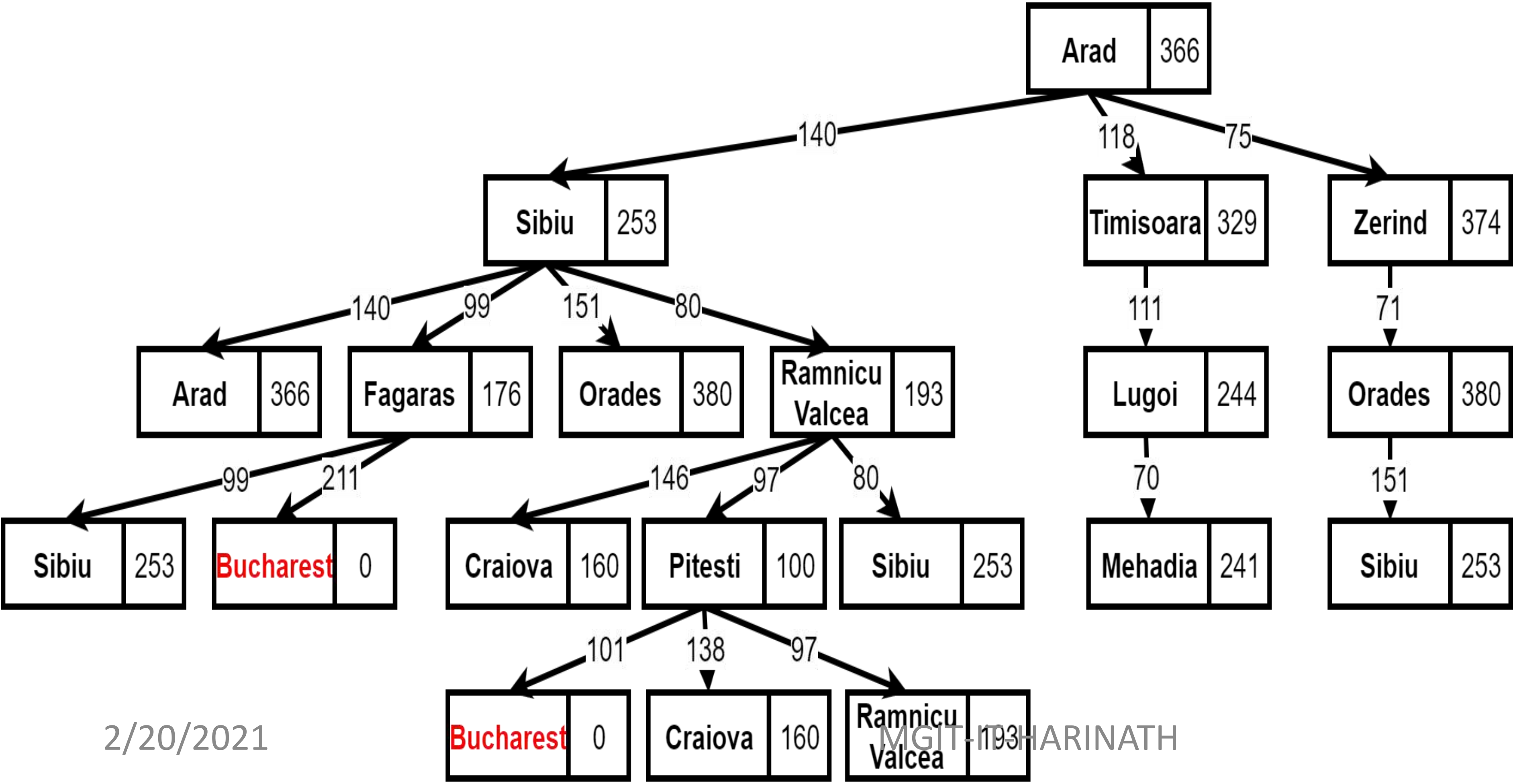
Sibiu



Sibiu Children

- Arad
- Fagaras
- Orades

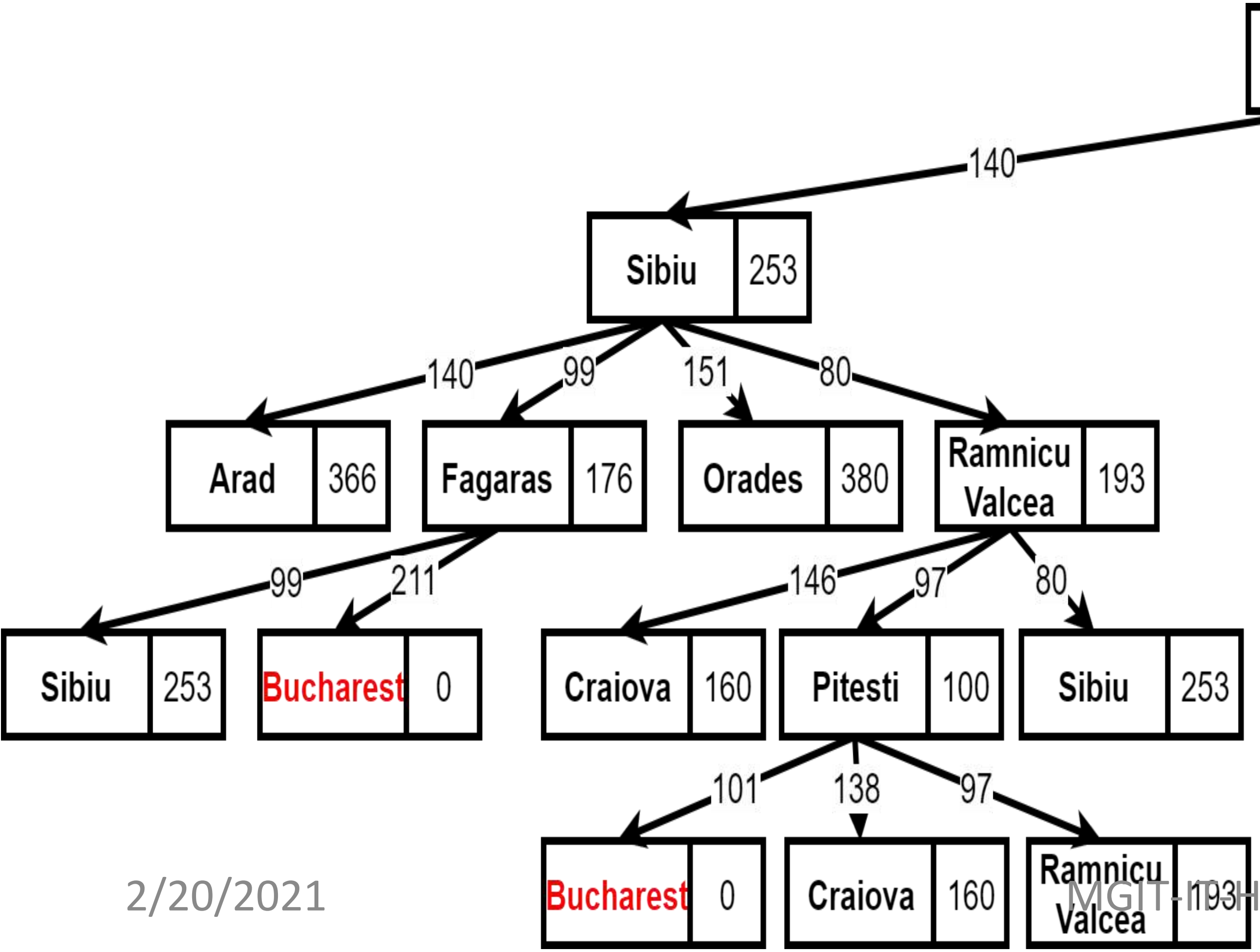
Sibiu



Sibiu Children

- Arad
- Fagaras
- Orades
- Ramnicu

Sibiu

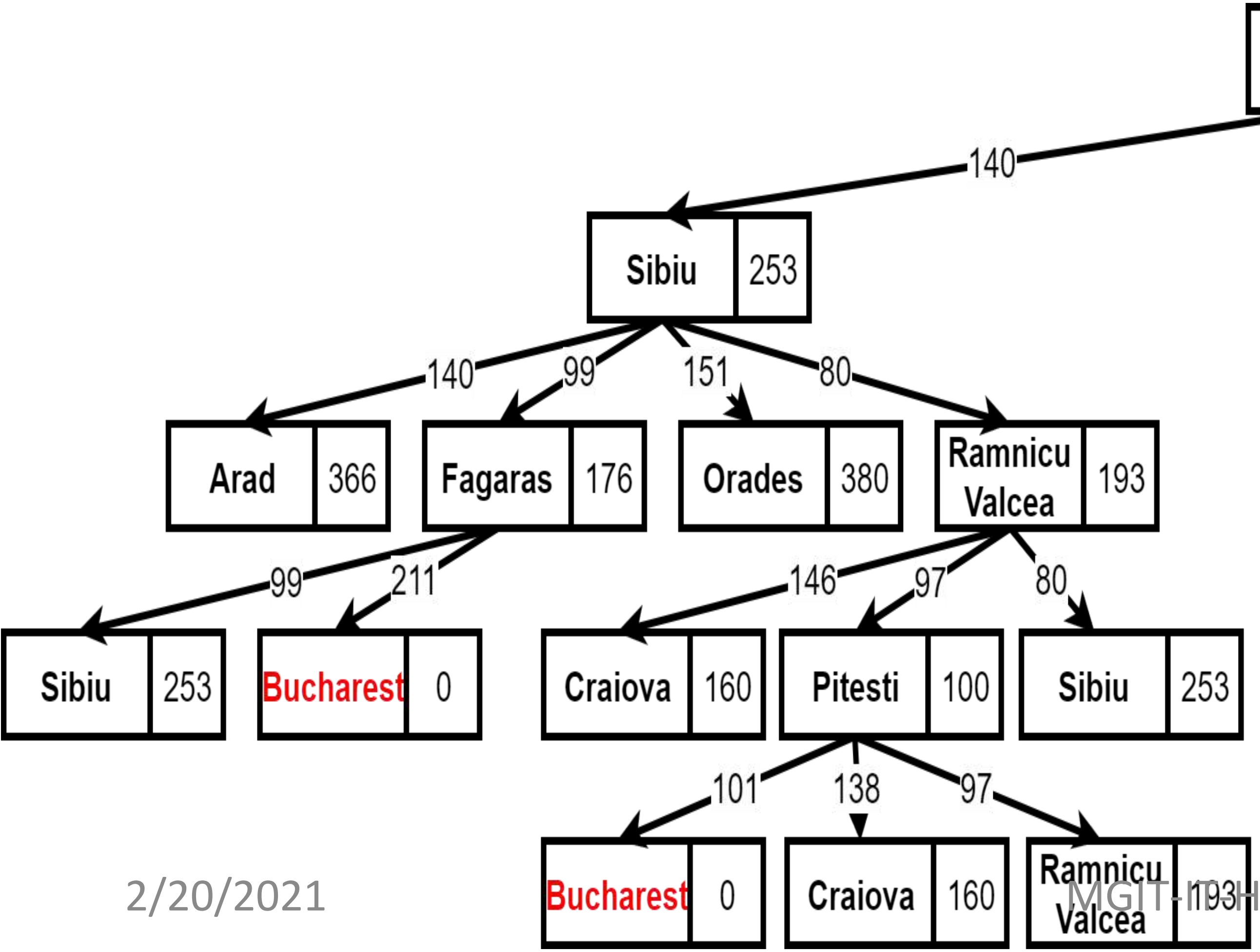


Sibiu Children

- Arad
- Fagaras
- Orades
- Ramnicu

Calculate Total Cost

Sibiu



Sibiu Children

Arad

Fagaras

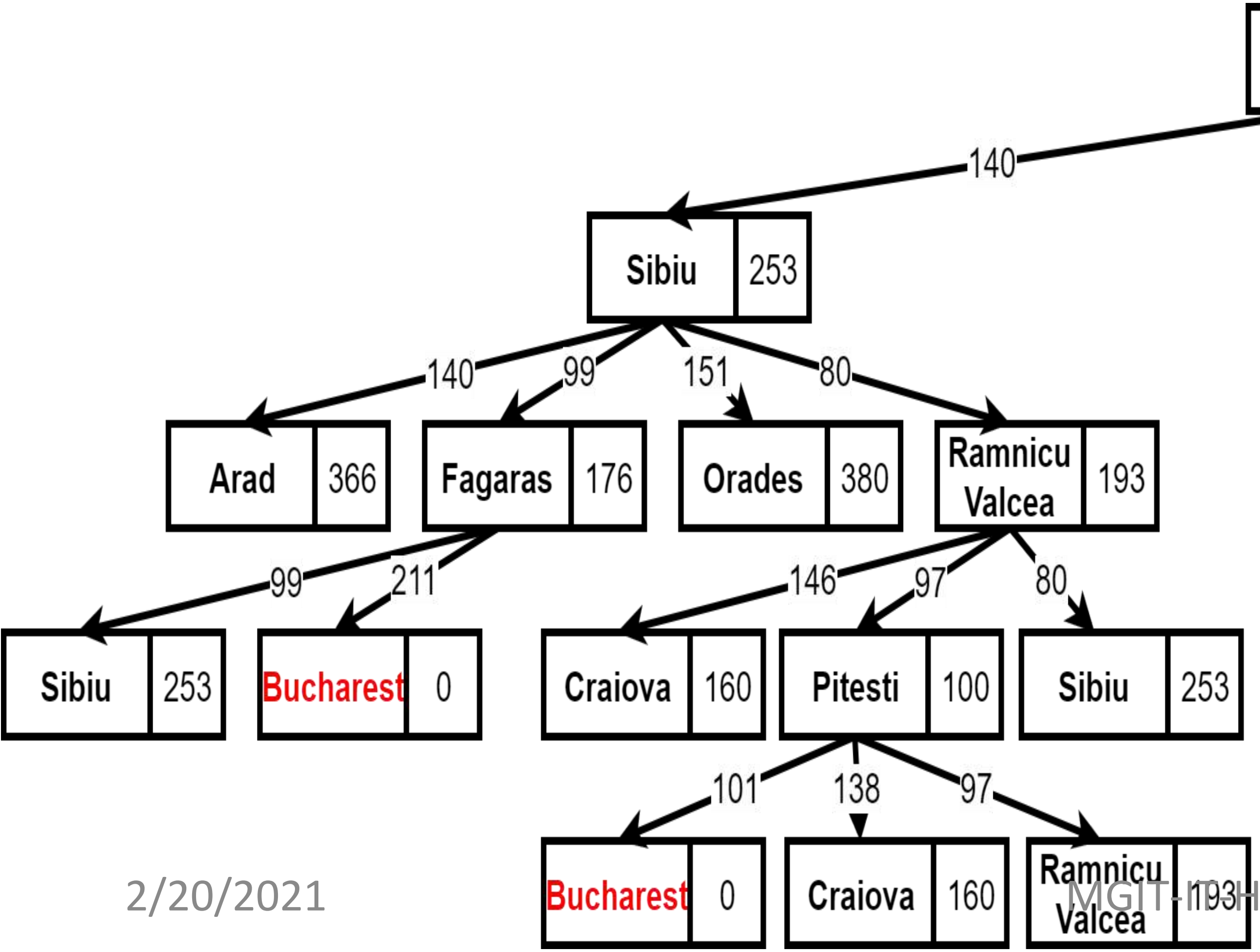
Orades

Ramnicu

Calculate Total Cost

Order Asc.

Sibiu

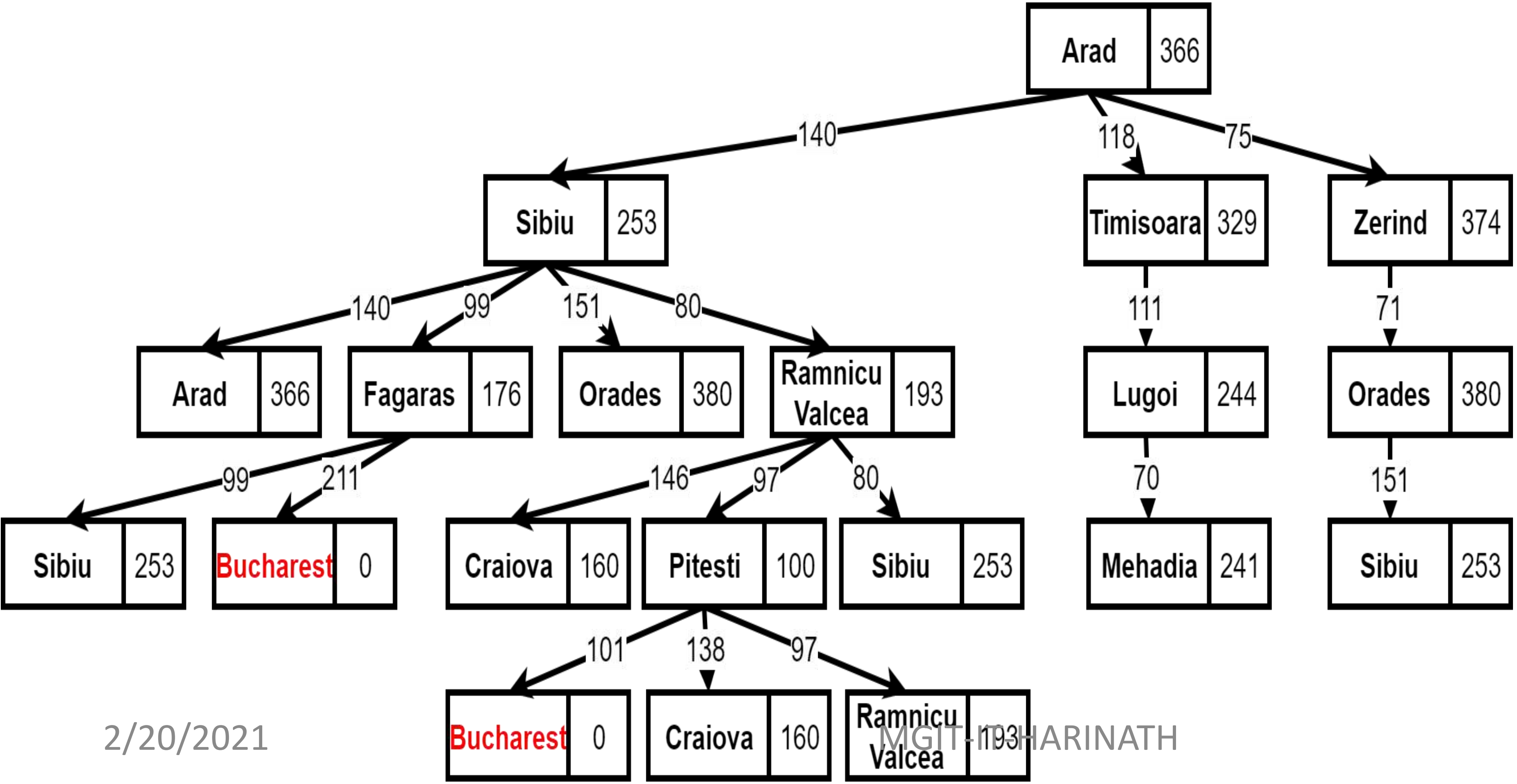


Sibiu Children

- Arad
- Fagaras
- Orades
- Ramnicu

Calculate Total Cost → **Order Asc.** → **Select Min. Cost**

Arad



Sibiu Children

Arad

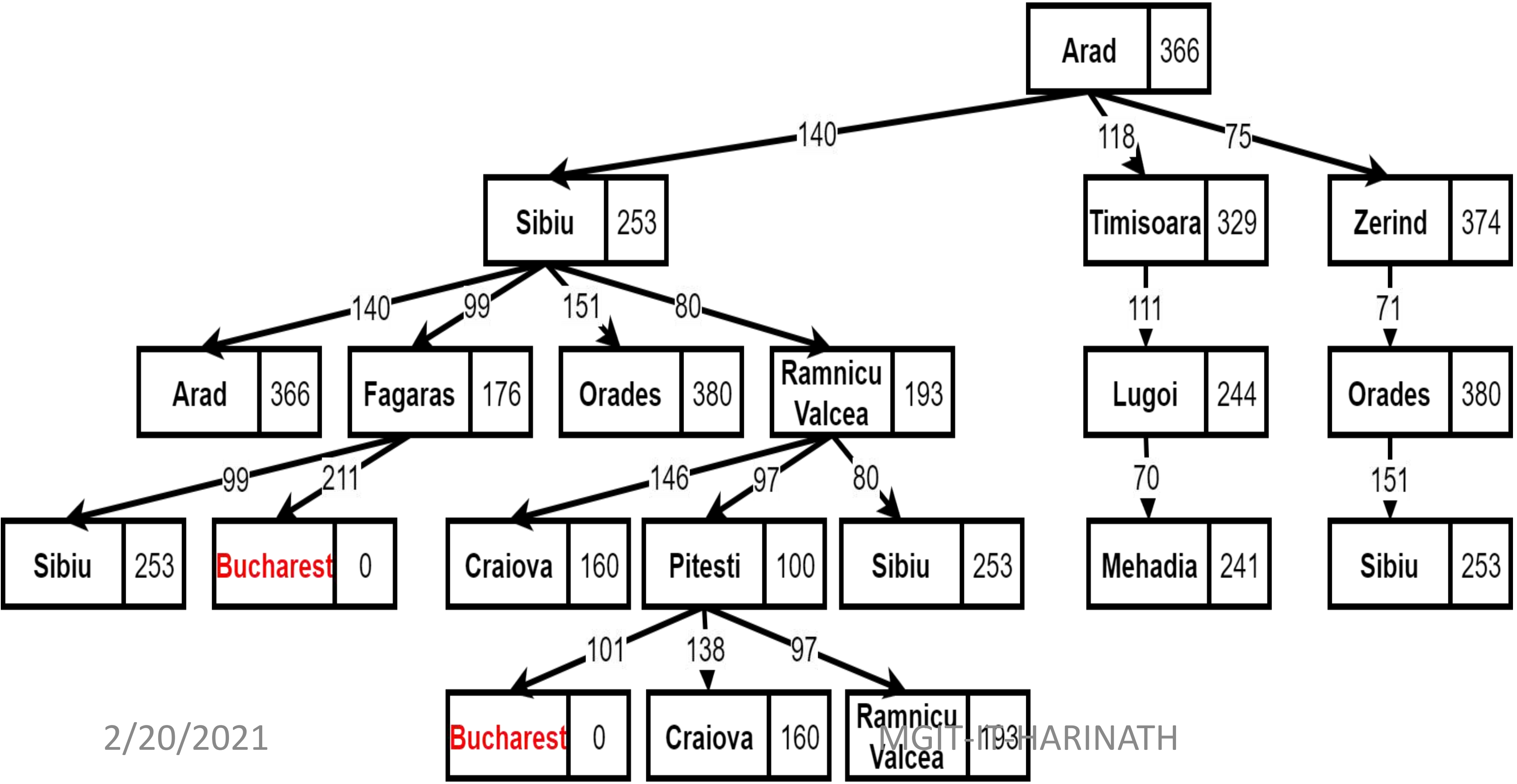
Fagaras

Orades

Ramnicu

Arad

Heuristic



Sibiu Children

Arad

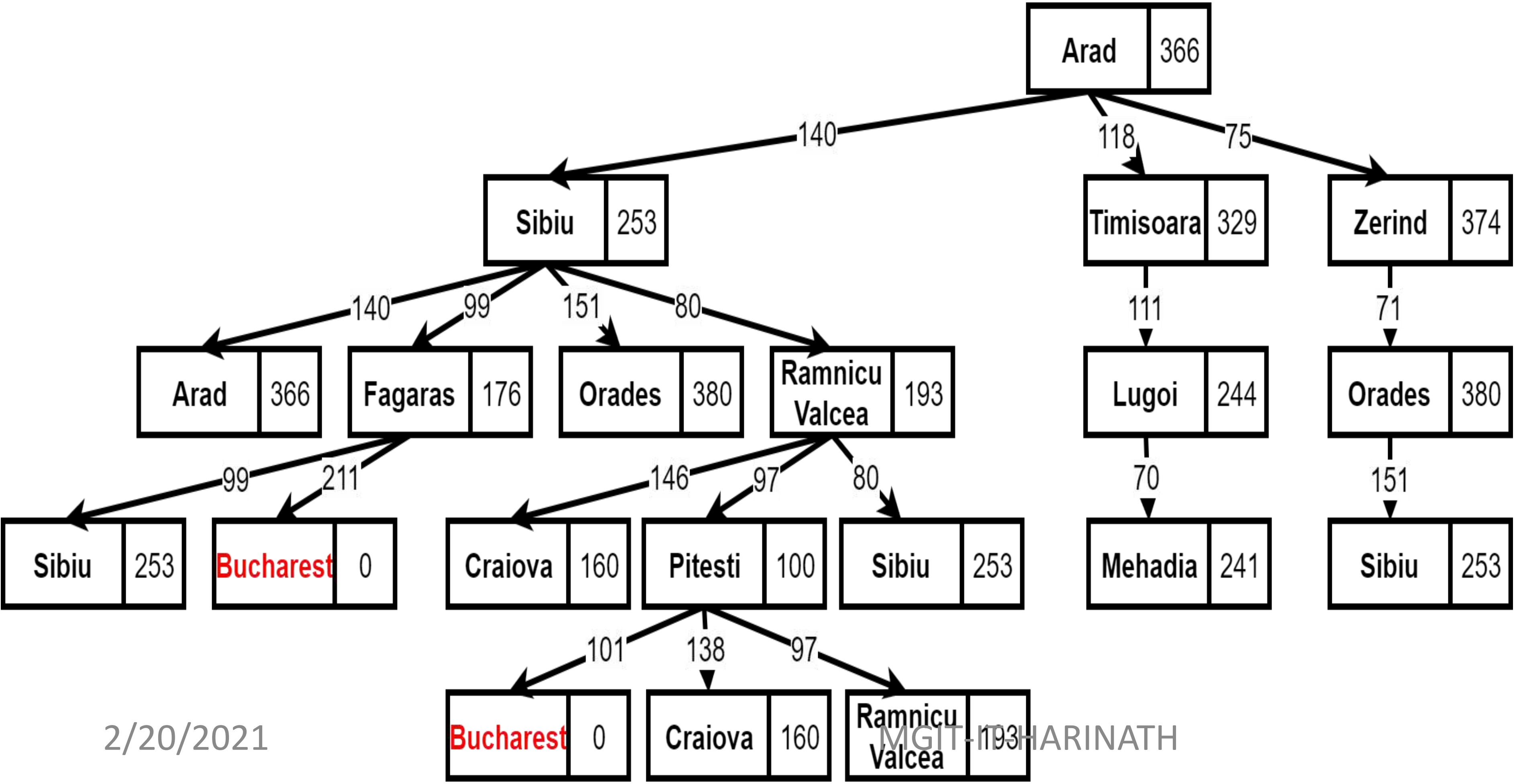
Fagaras

Orades

Ramnicu

Arad

Heuristic + **Cost**

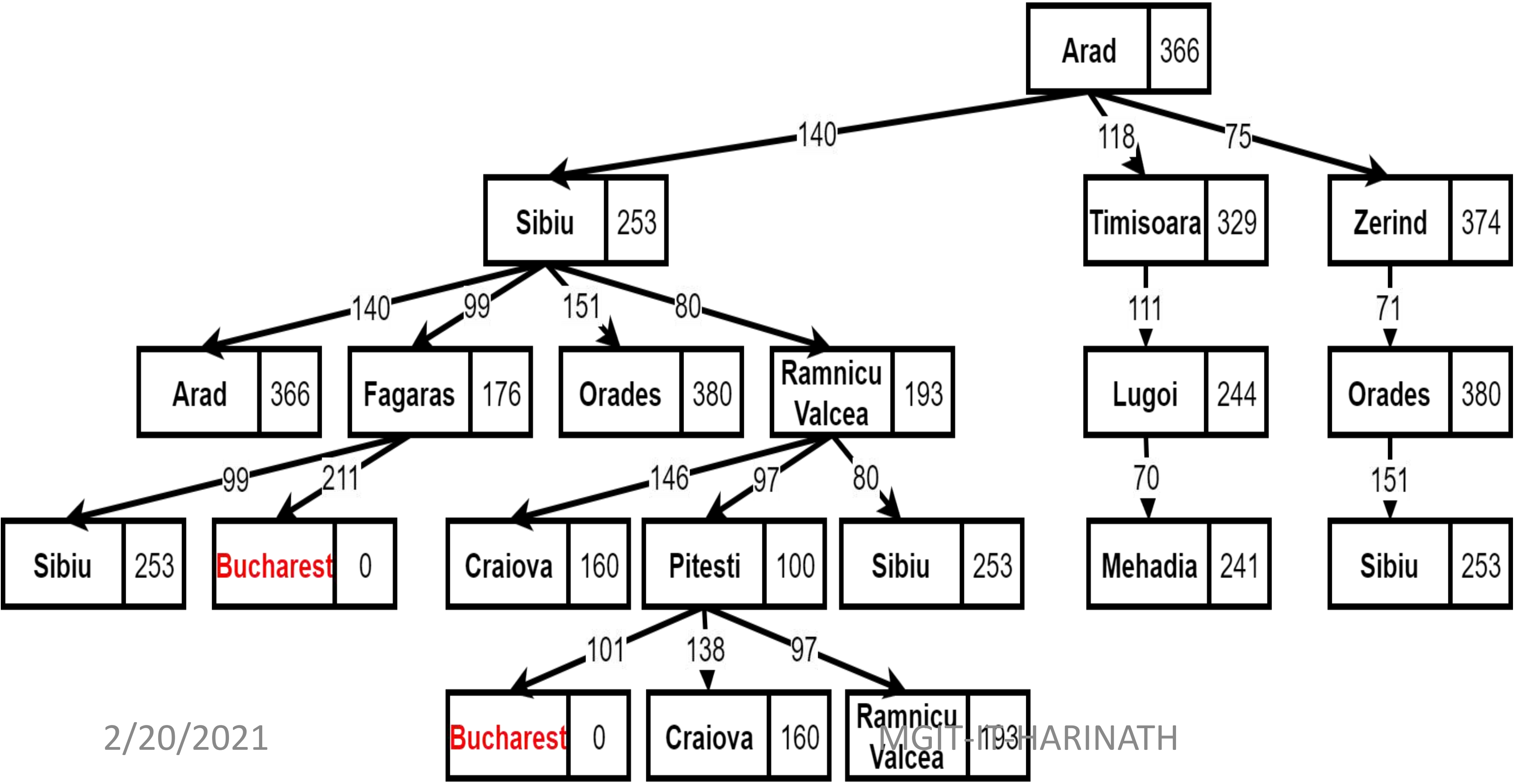


Sibiu Children

- Arad
- Fagaras
- Orades
- Ramnicu

Arad

Heuristic + Cost = Total



Sibiu Children

Arad

Fagaras

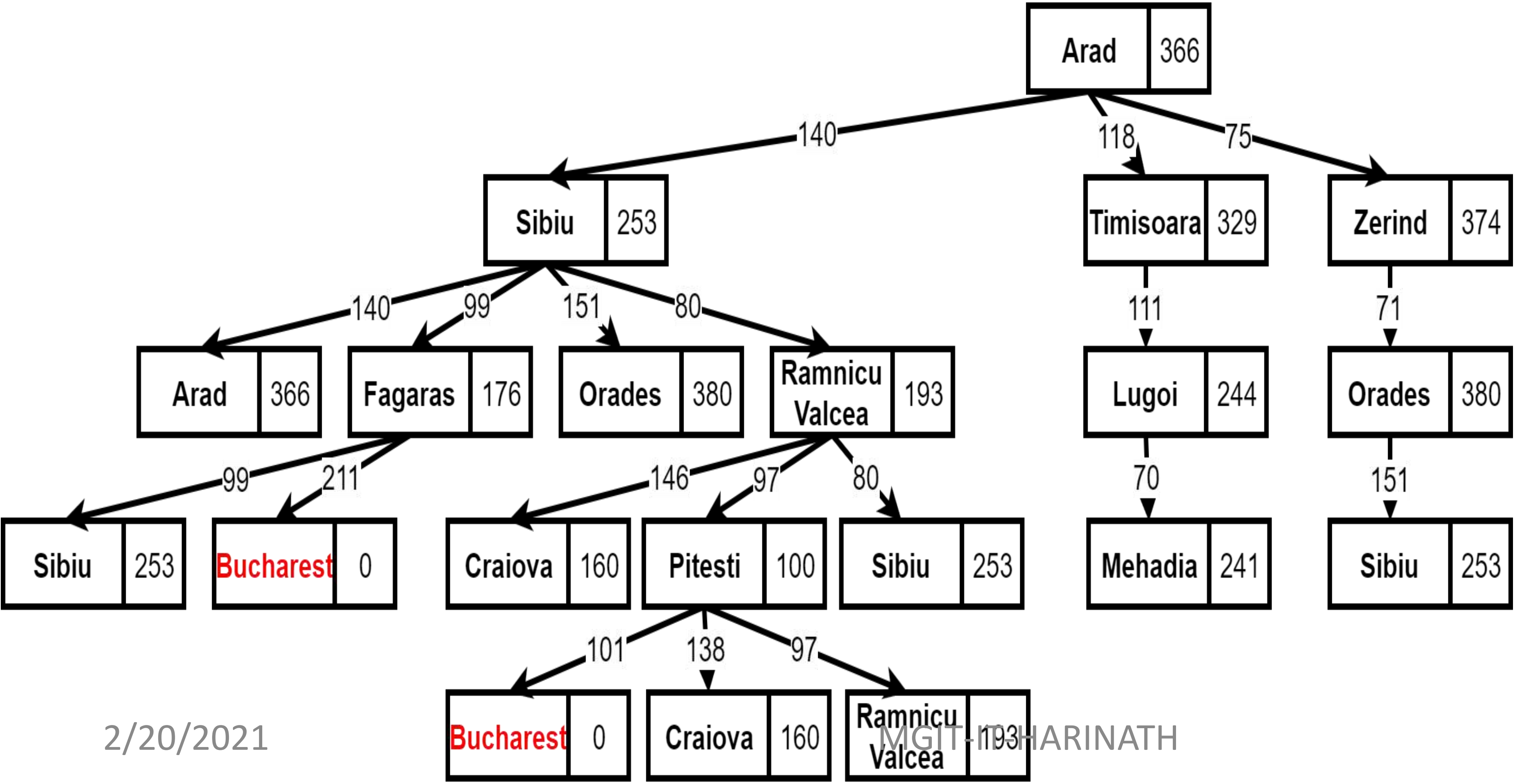
Orades

Ramnicu

Arad

Heuristic + Cost = Total

366



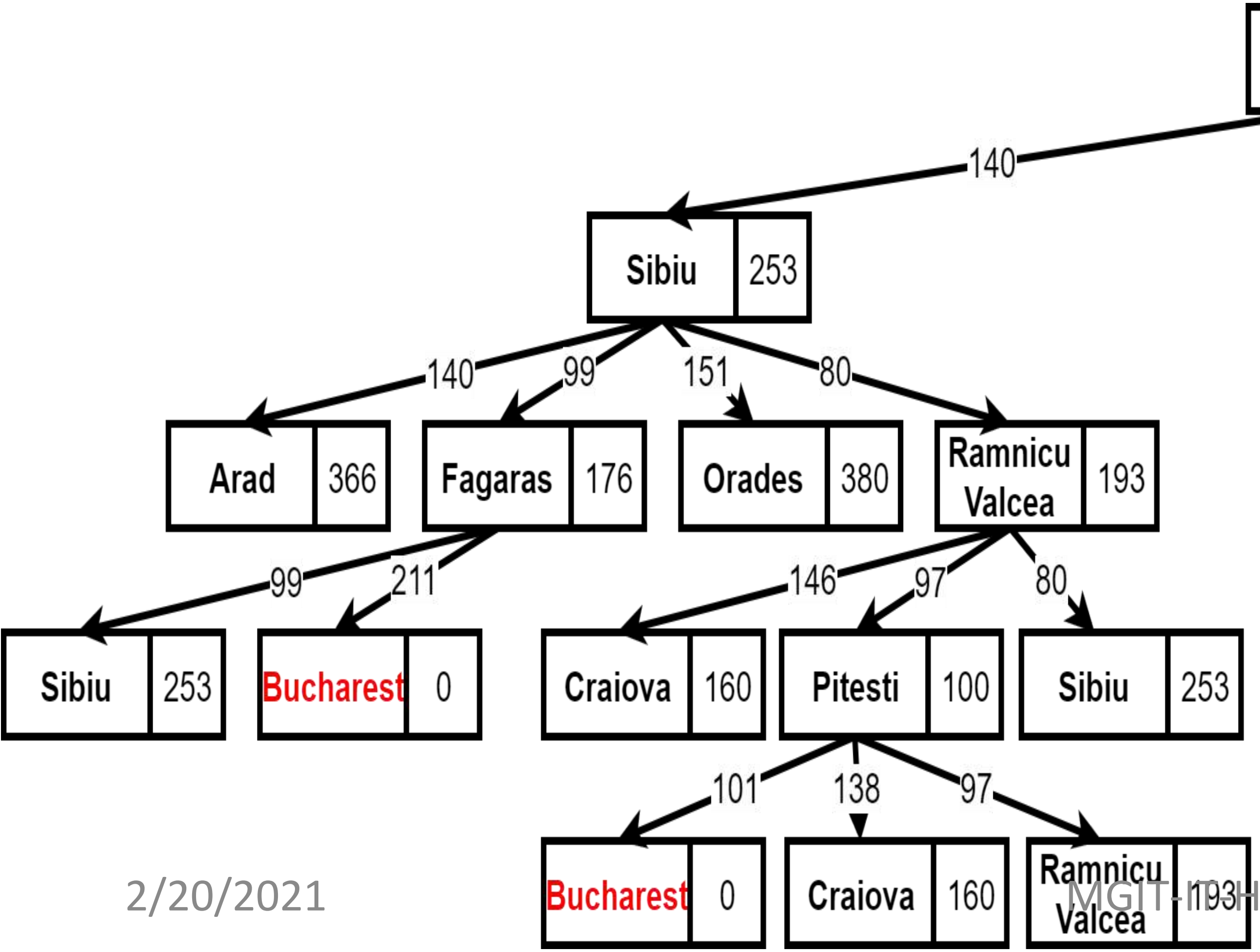
Sibiu Children

- Arad
- Fagaras
- Orades
- Ramnicu

Arad

Heuristic + Cost = Total

366 + 280



Sibiu Children

Arad

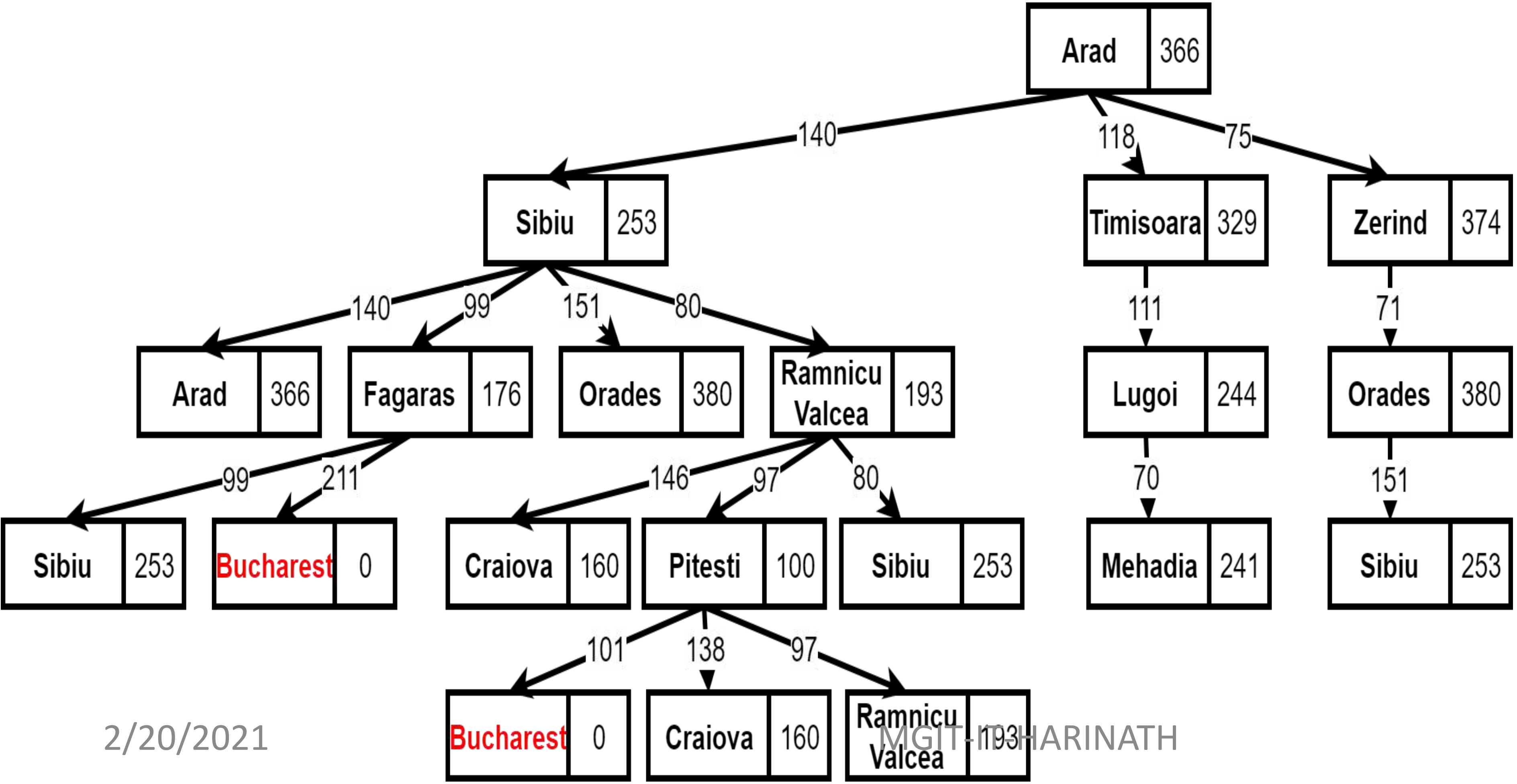
Fagaras

Orades

Ramnicu

Arad

Heuristic	+	Cost	=	Total
366	+	280	=	646

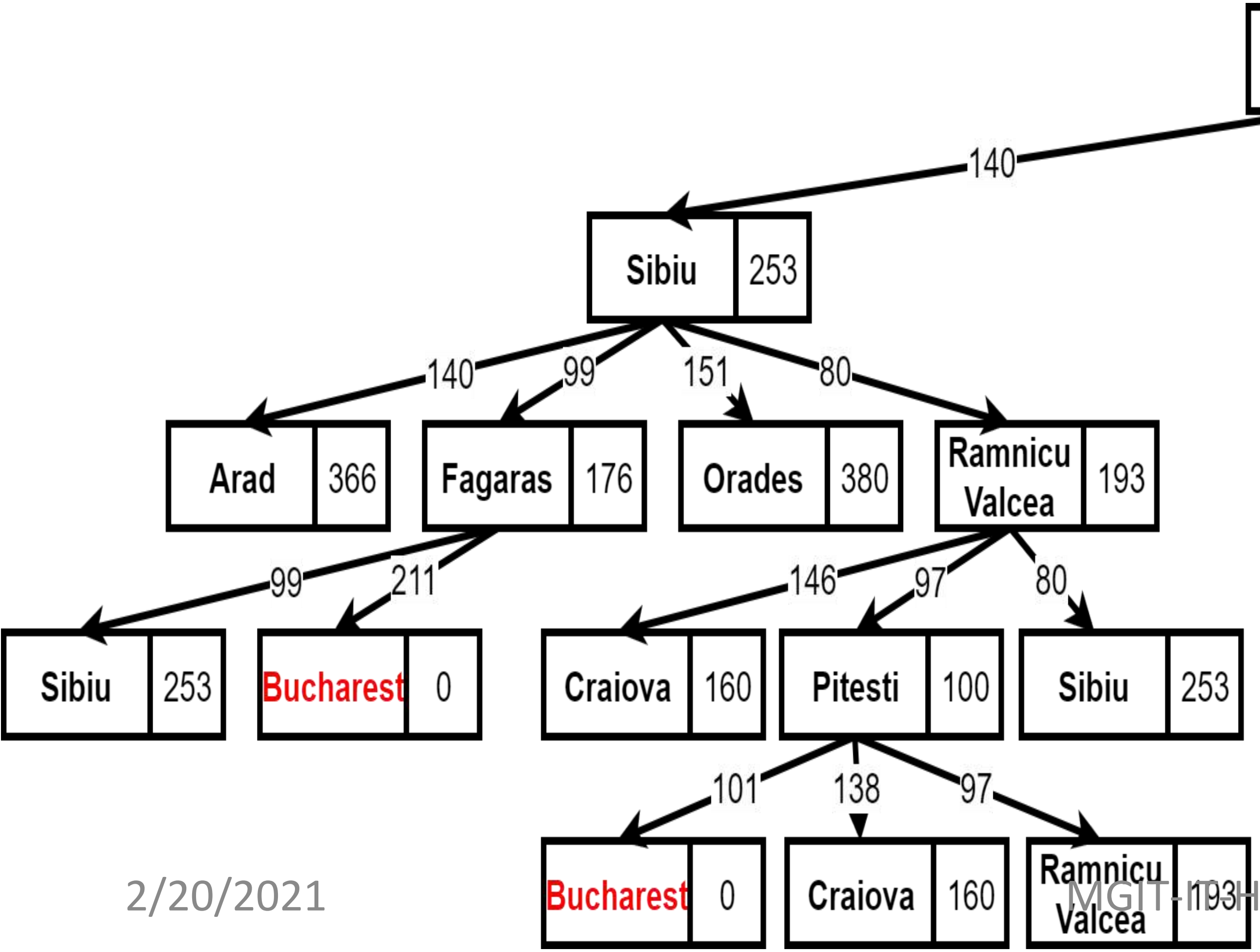


Sibiu Children

- Arad
- Fagaras
- Orades
- Ramnicu

Arad

Heuristic	+	Cost	=	Total
366	+	280	=	646

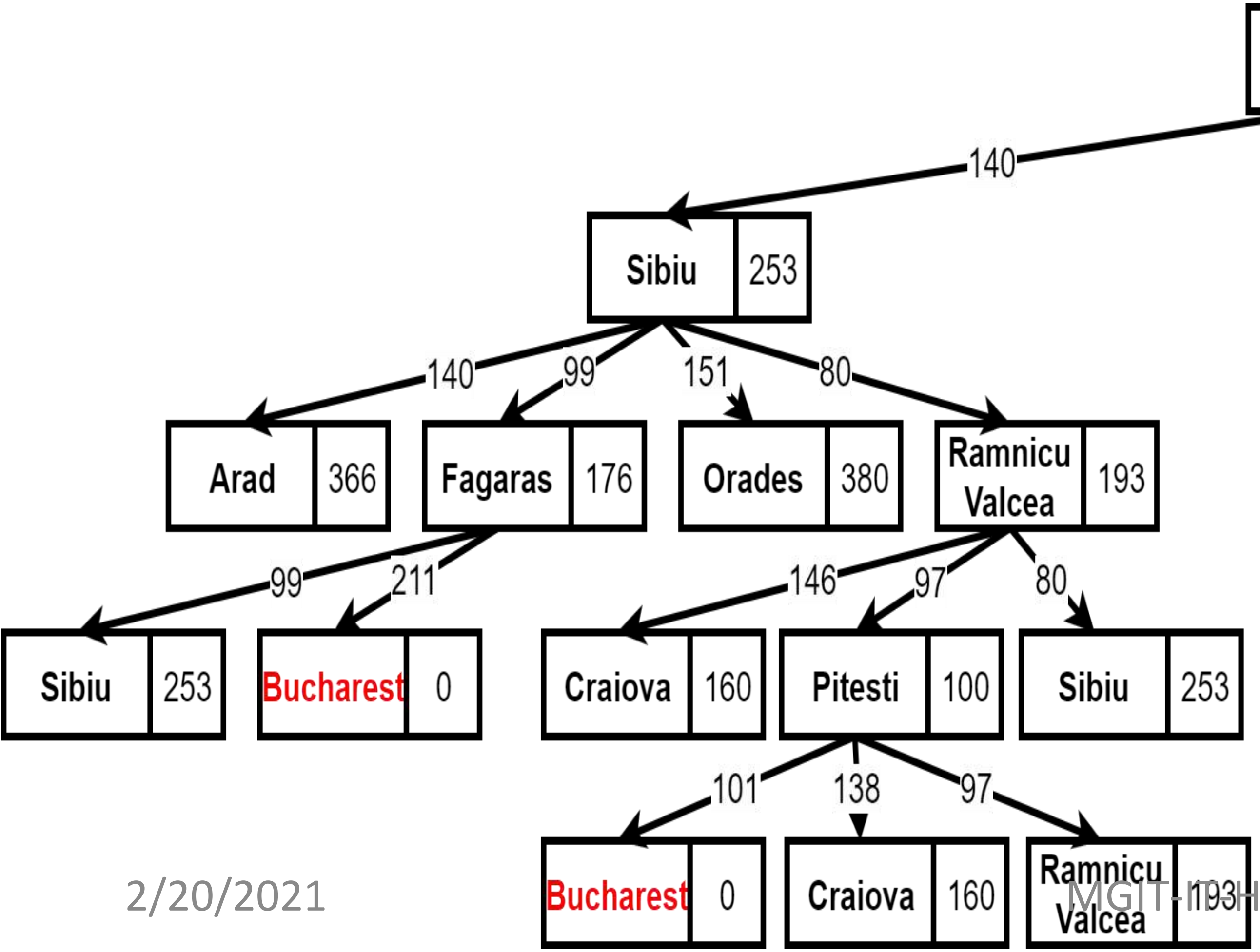


Sibiu Children

Arad	646
Fagaras	
Orades	
Ramnicu	

Fagaras

Heuristic + Cost = Total



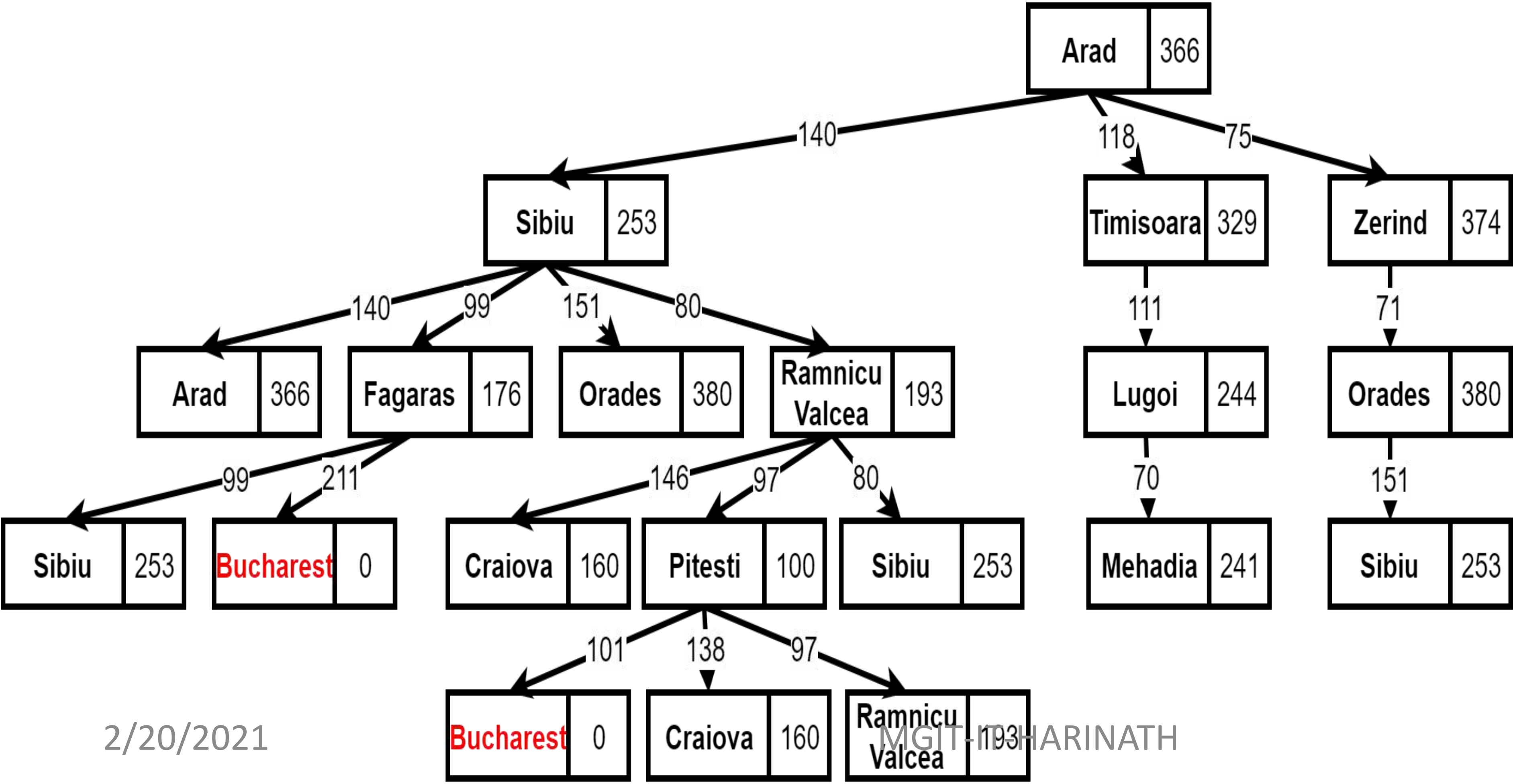
Sibiu Children

Arad	646
Fagaras	
Orades	
Ramnicu	

Fagaras

Heuristic + Cost = Total

176



Sibiu Children

Arad 646

Fagaras

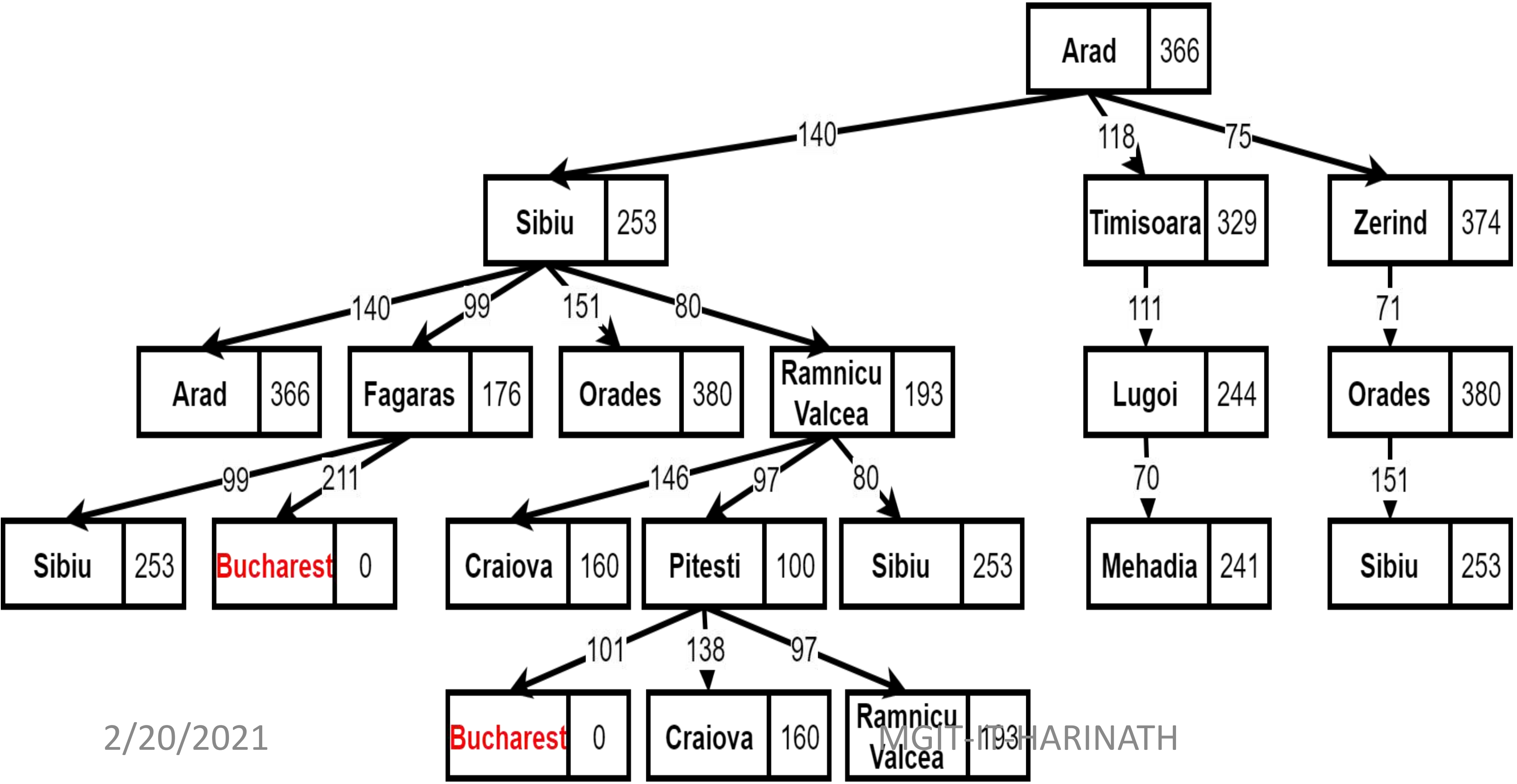
Orades

Ramnicu

Fagaras

Heuristic + **Cost** = **Total**

176 + 239

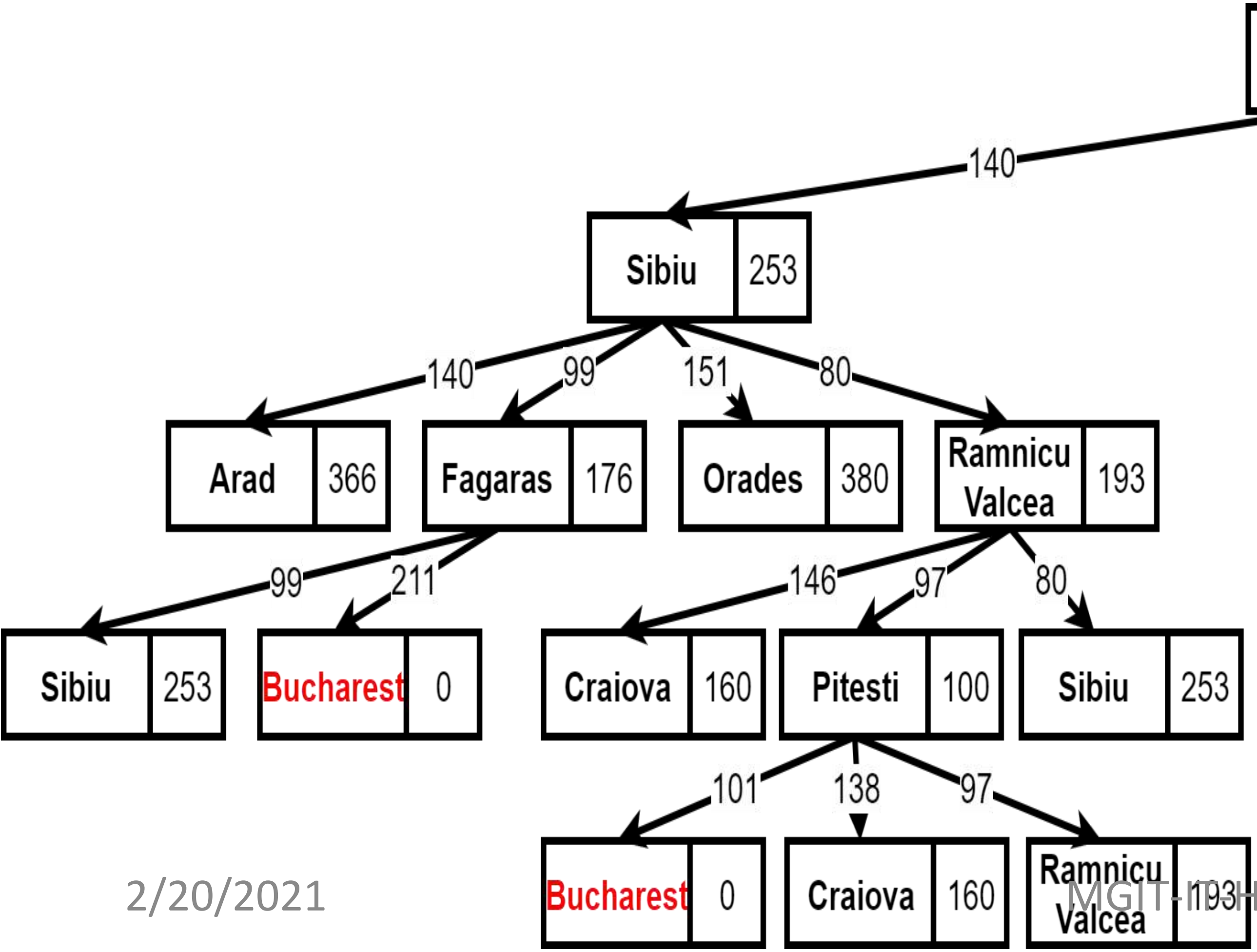


Sibiu Children

Arad	646
Fagaras	
Orades	
Ramnicu	

Fagaras

Heuristic	+	Cost	=	Total
176	+	239	=	415

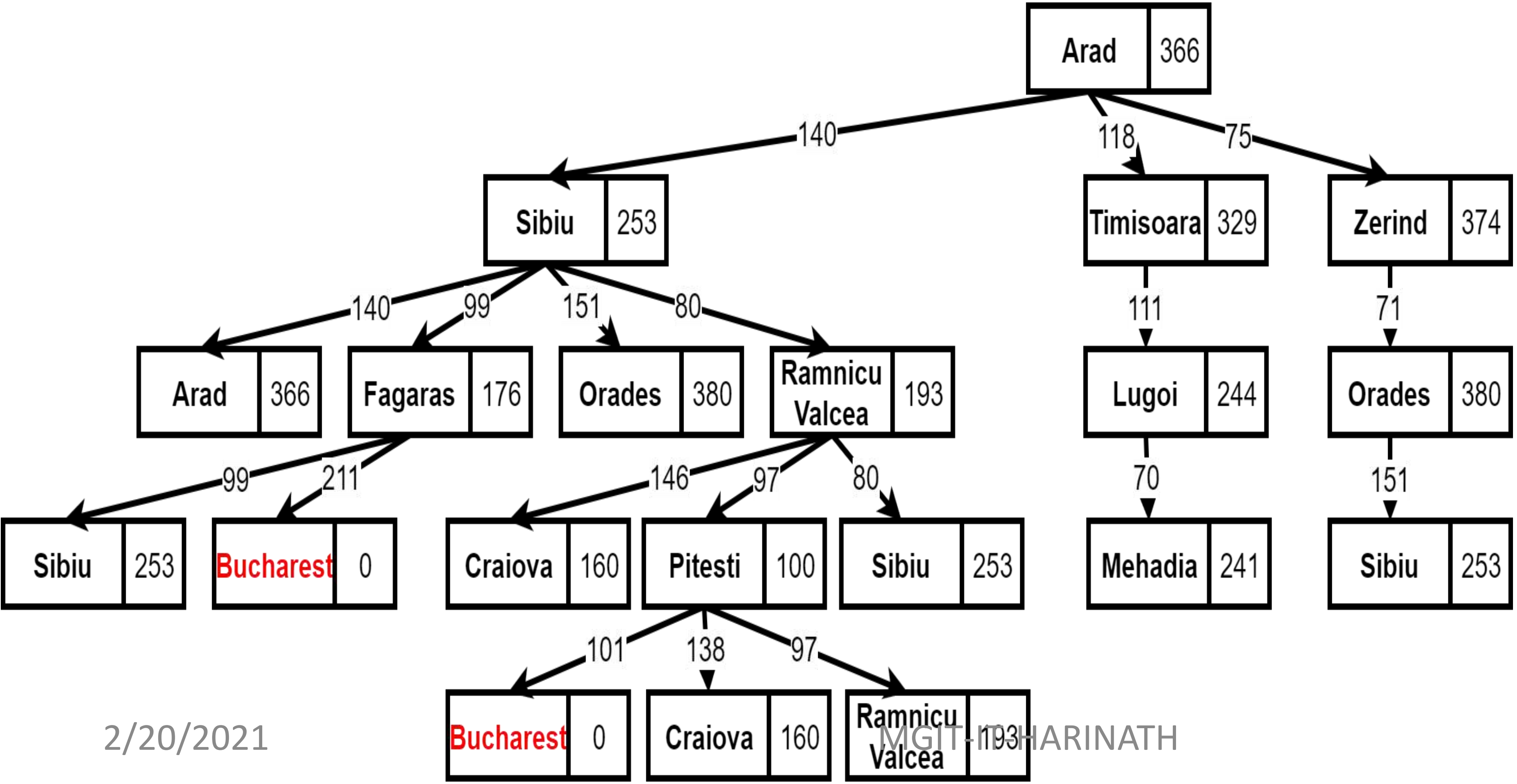


Sibiu Children

Arad	646
Fagaras	
Orades	
Ramnicu	

Fagaras

Heuristic	+	Cost	=	Total
176	+	239	=	415

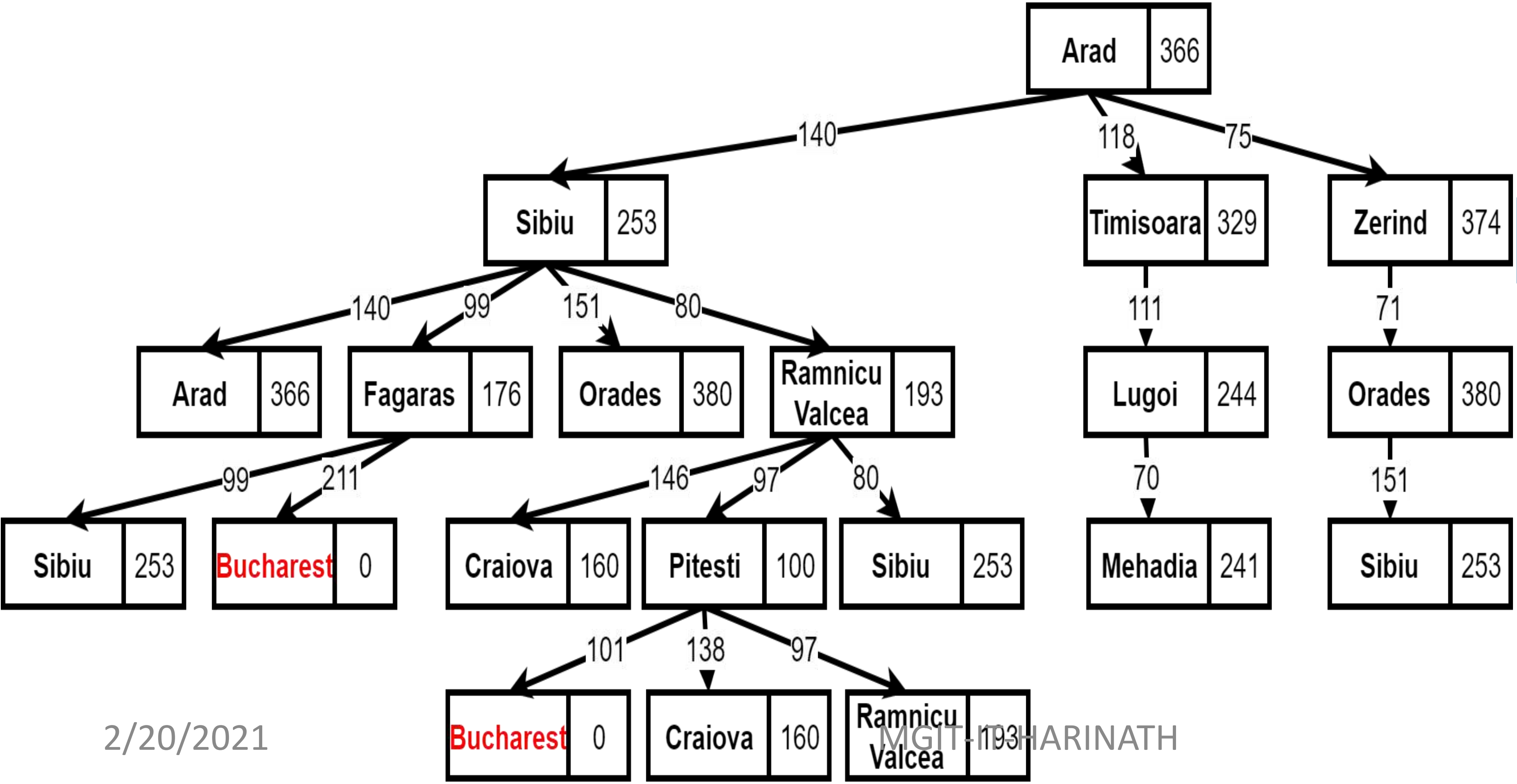


Sibiu Children

Arad	646
Fagaras	415
Orades	
Ramnicu	

Orades

Heuristic + Cost = Total



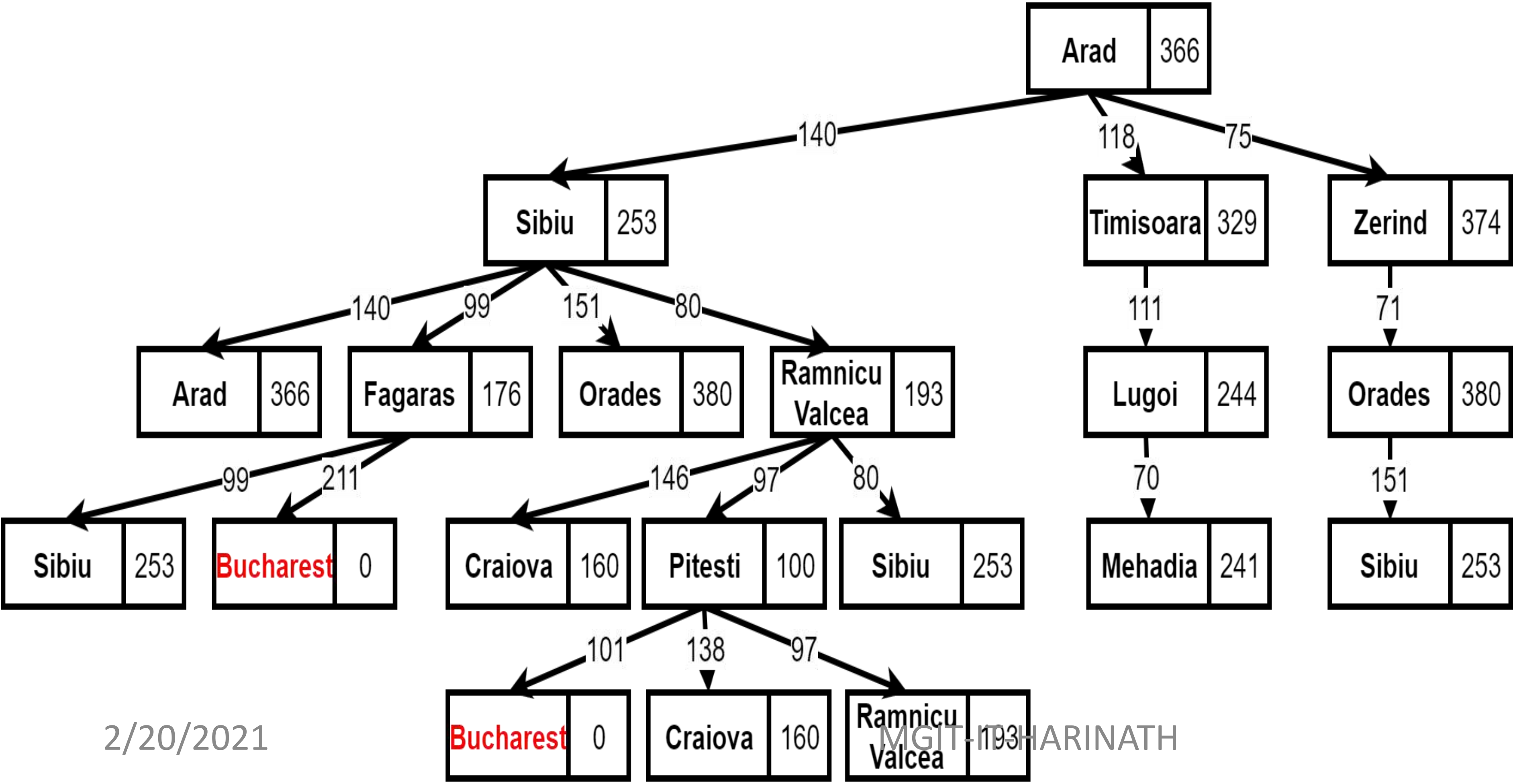
Sibiu Childre

Arad	646
Fagaras	415
Orades	
Ramnicu	

Orades

Heuristic + Cost = Total

380

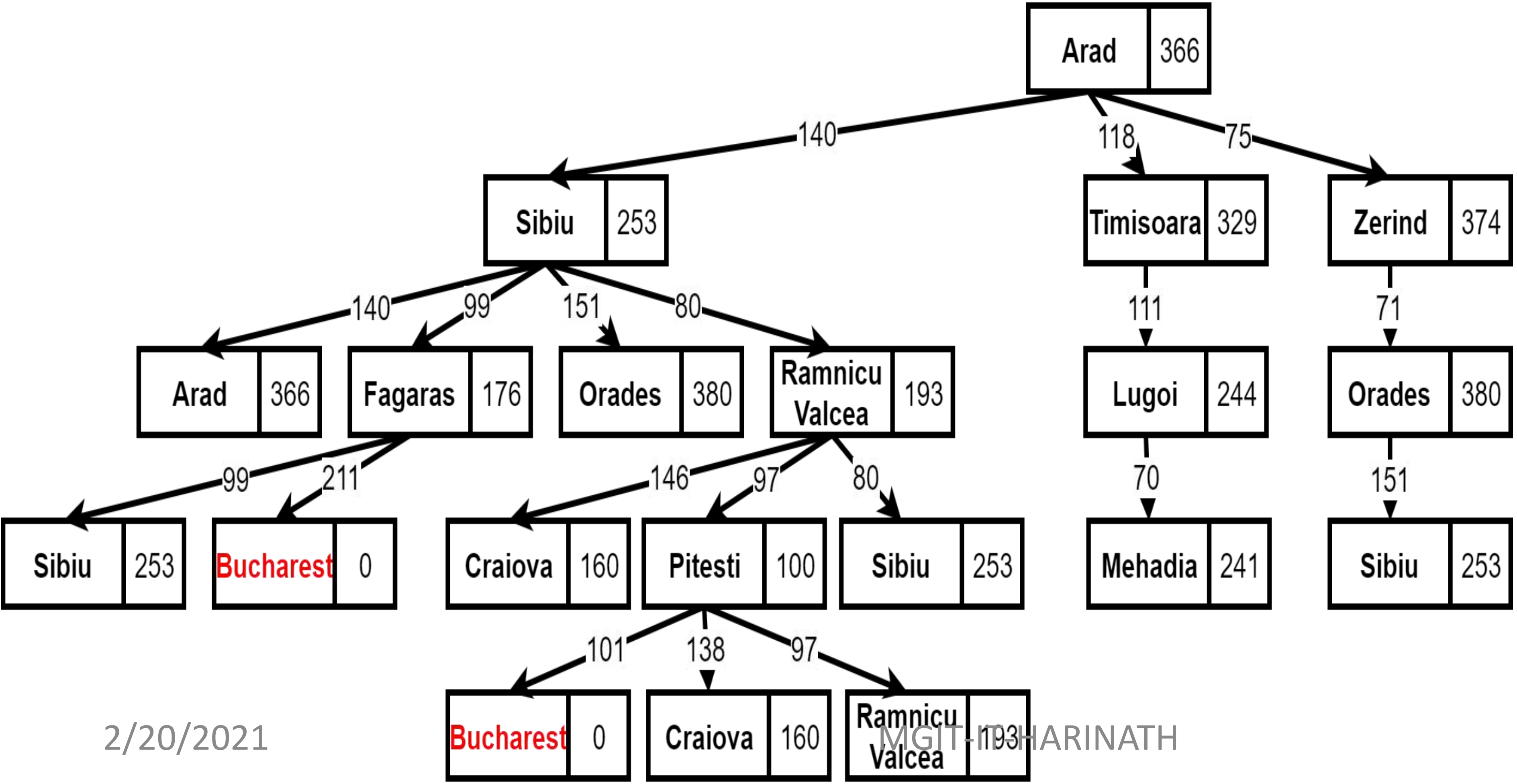


Sibiu Childre

Arad	646
Fagaras	415
Orades	
Ramnicu	

Orades

Heuristic	+	Cost	=	Total
380	+	291		

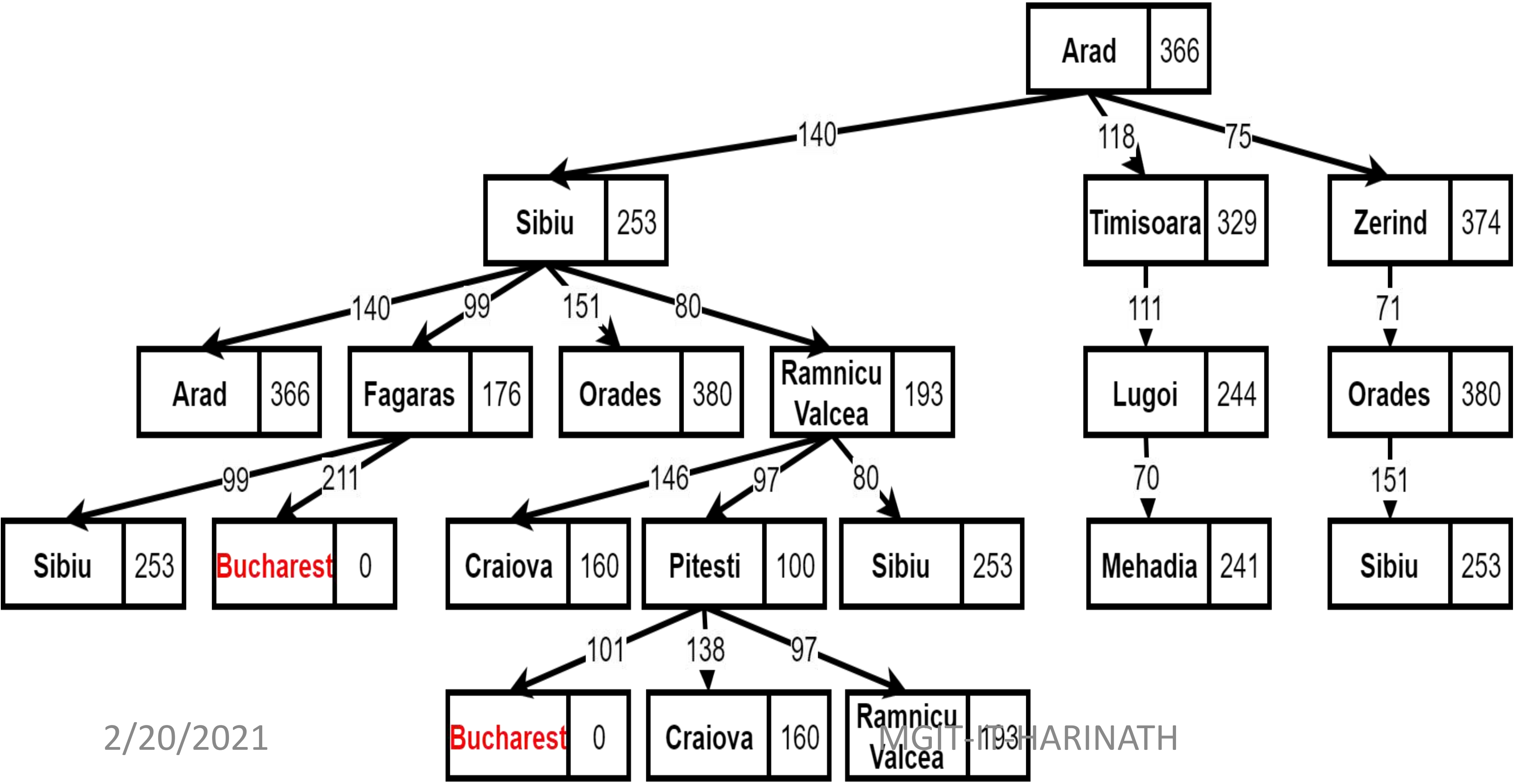


Sibiu Children

Arad	646
Fagaras	415
Orades	
Ramnicu	

Orades

Heuristic	+	Cost	=	Total
380	+	291	=	671

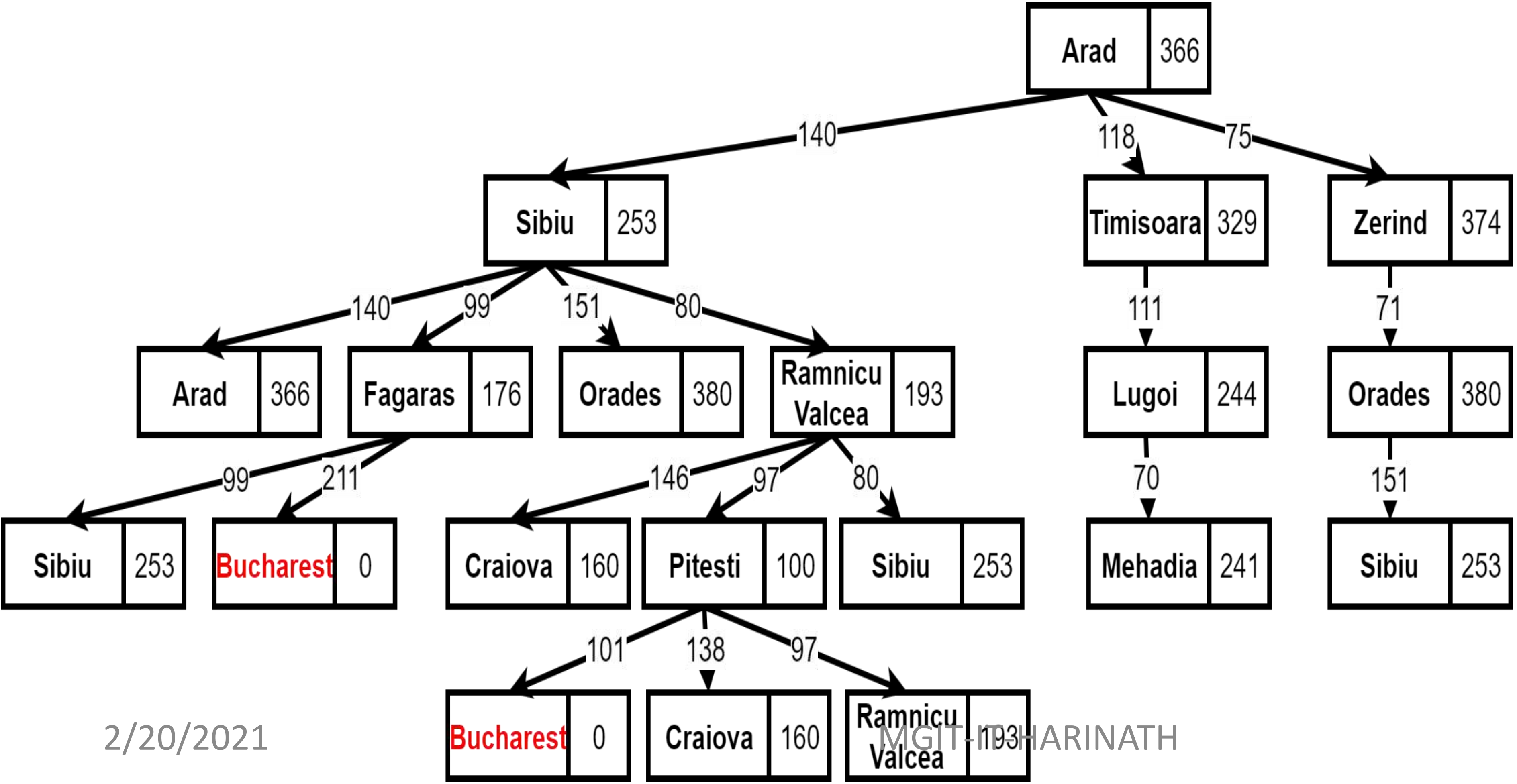


Sibiu Children

Arad	646
Fagaras	415
Orades	
Ramnicu	

Orades

Heuristic	+	Cost	=	Total
380	+	291	=	671

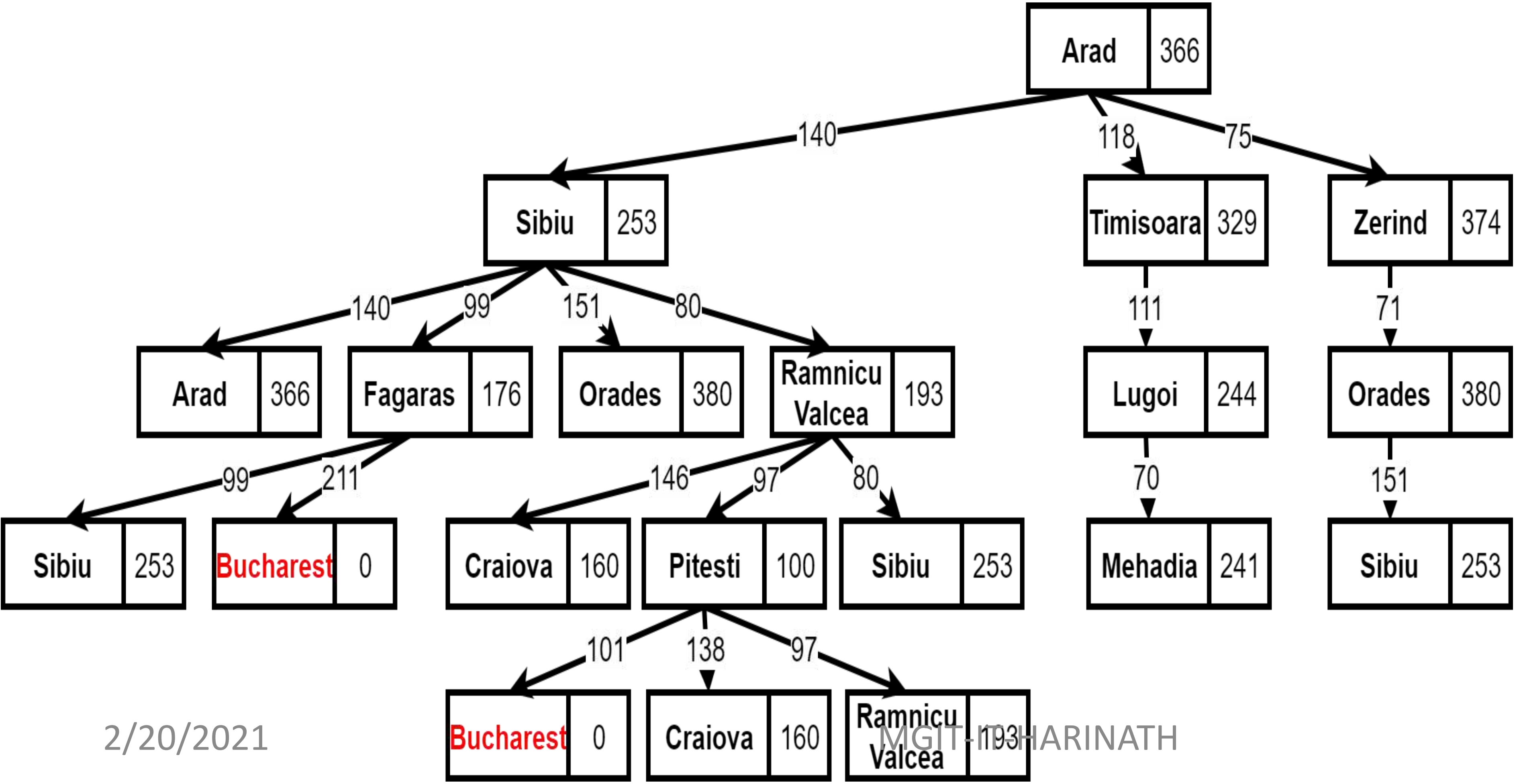


Sibiu Children

Arad	646
Fagaras	415
Orades	671
Ramnicu	

Ramnicu

Heuristic + **Cost** = **Total**



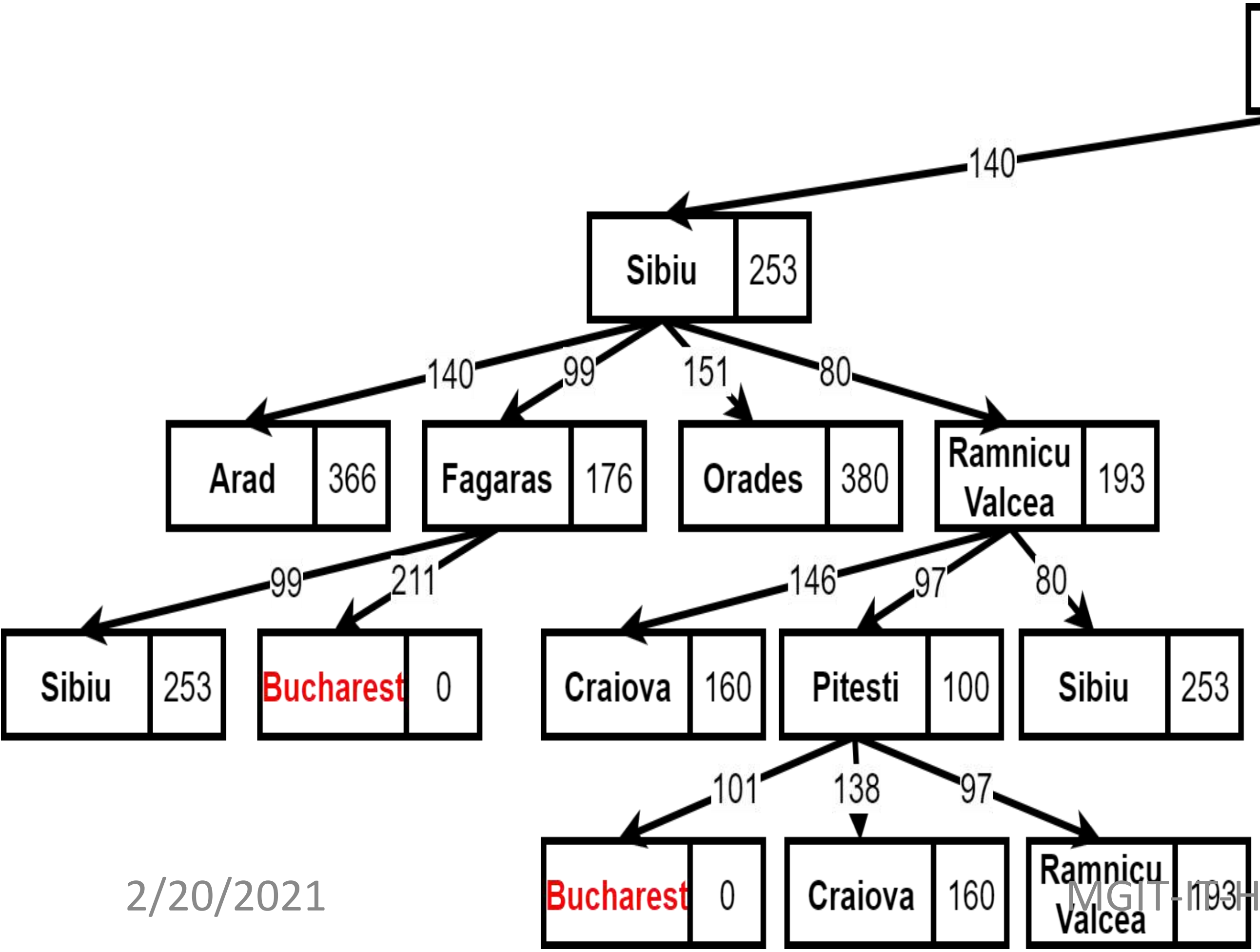
Sibiu Children

Arad	646
Fagaras	415
Orades	671
Ramnicu	

Ramnicu

Heuristic + Cost = Total

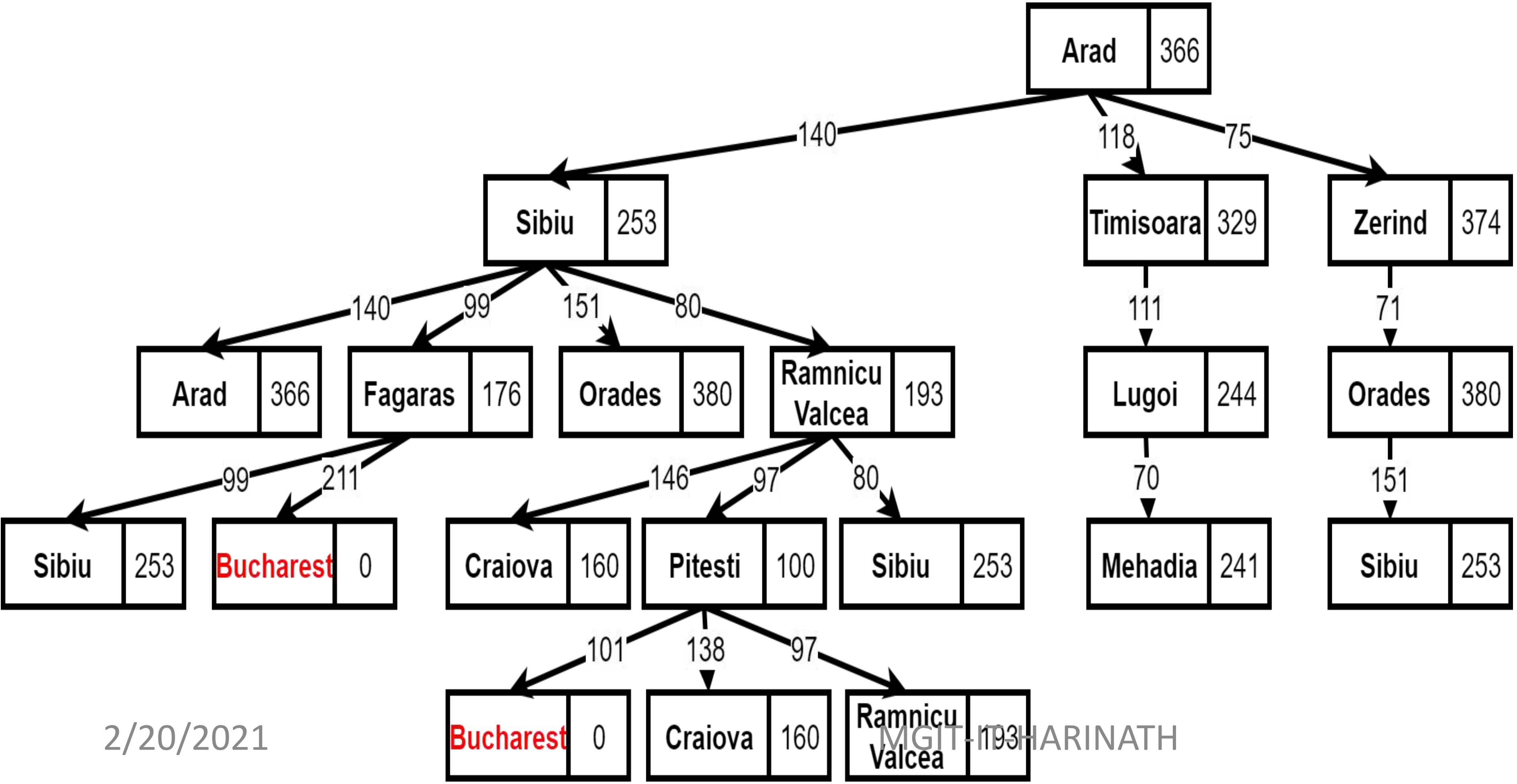
193



Sibiu Children	
Arad	646
Fagaras	415
Orades	671
Ramnicu	

Ramnicu

Heuristic	+	Cost	=	Total
193	+	220		

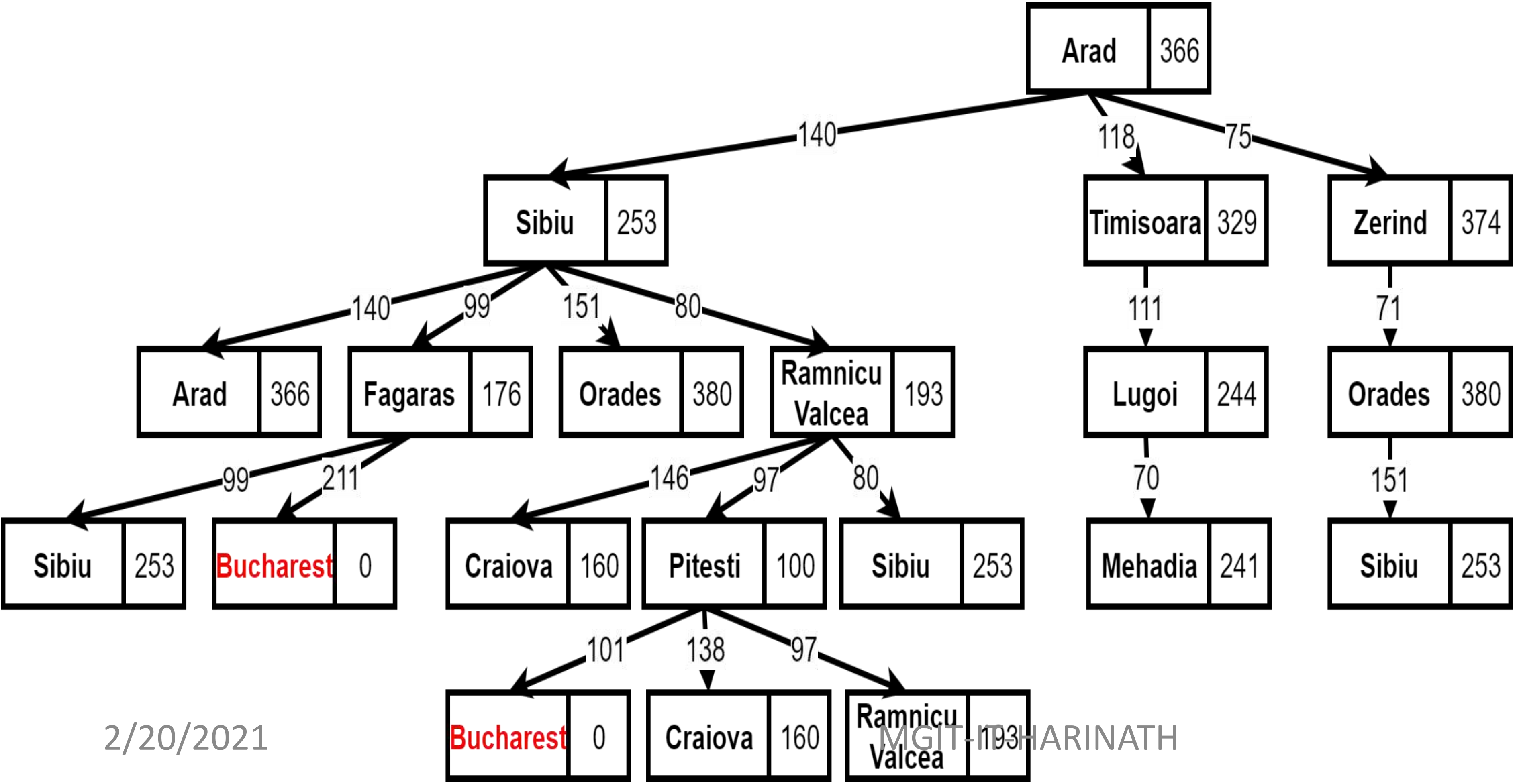


Sibiu Children

Arad	646
Fagaras	415
Orades	671
Ramnicu	

Ramnicu

Heuristic	+	Cost	=	Total
193	+	220	=	413

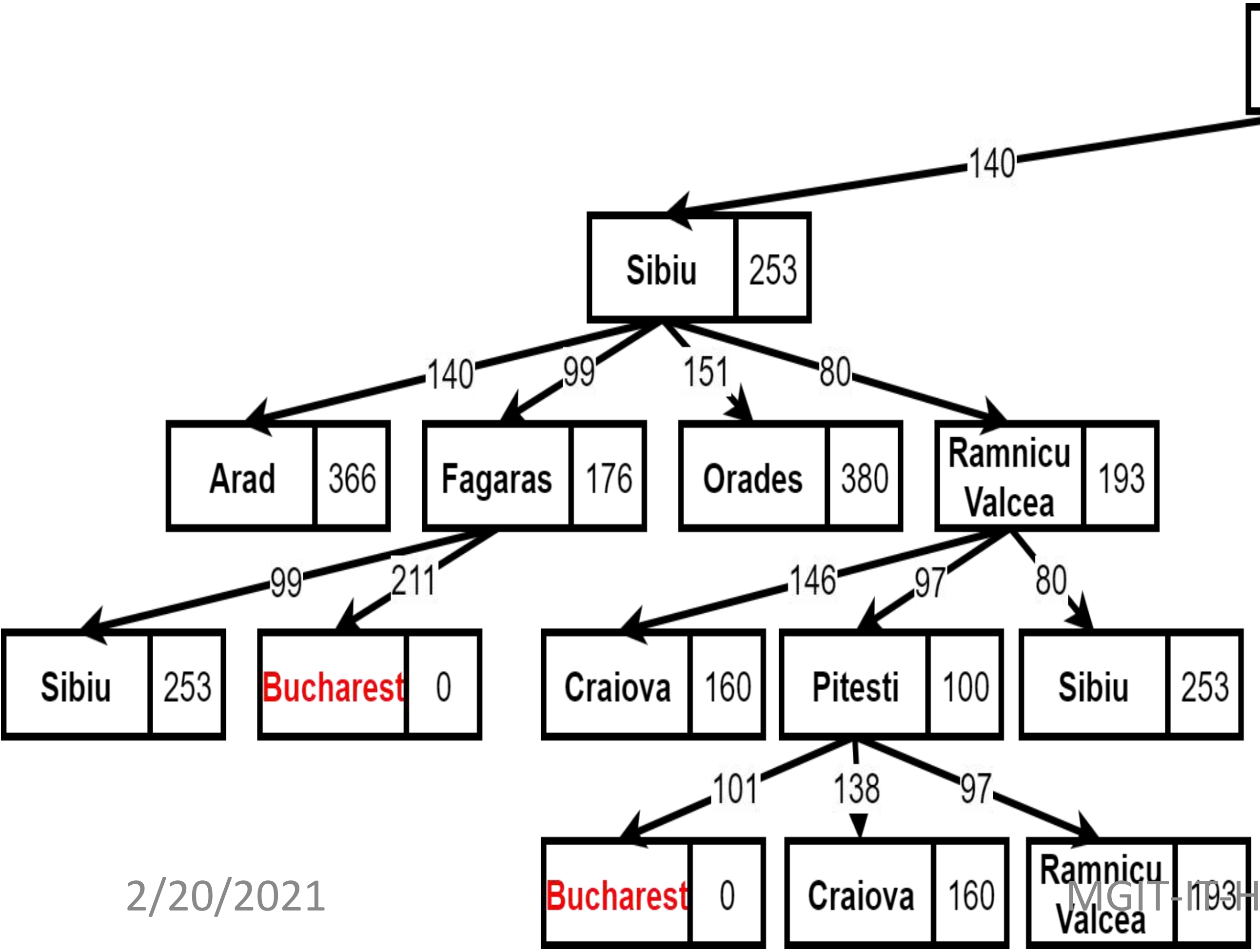


Sibiu Children

Arad	646
Fagaras	415
Orades	671
Ramnicu	

Ramnicu

Heuristic	+	Cost	=	Total
193	+	220	=	413



Sibiu Children

Arad	646
Fagaras	415
Orades	671
Ramnicu	413

Current Queue

Timisoar	Zerin
447	449

Current Queue

Timisoar 447

Zerin 449

Arad 646

Fagara 415

Orade 671

Ramnic 413

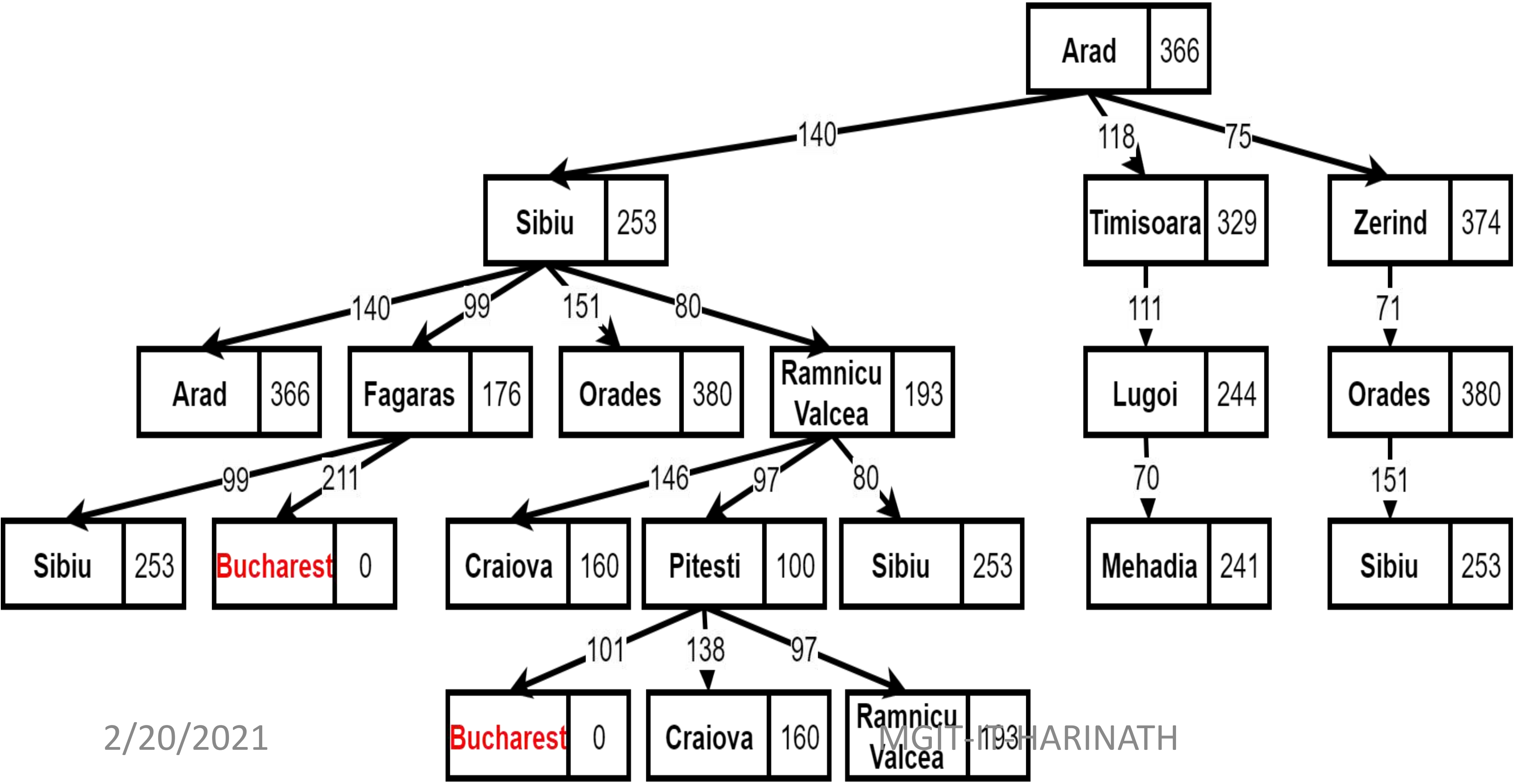
Current Queue

Ramnic	Fagara	Timisoar	Zerin	Arad	Orade
413	415	447	449	646	671

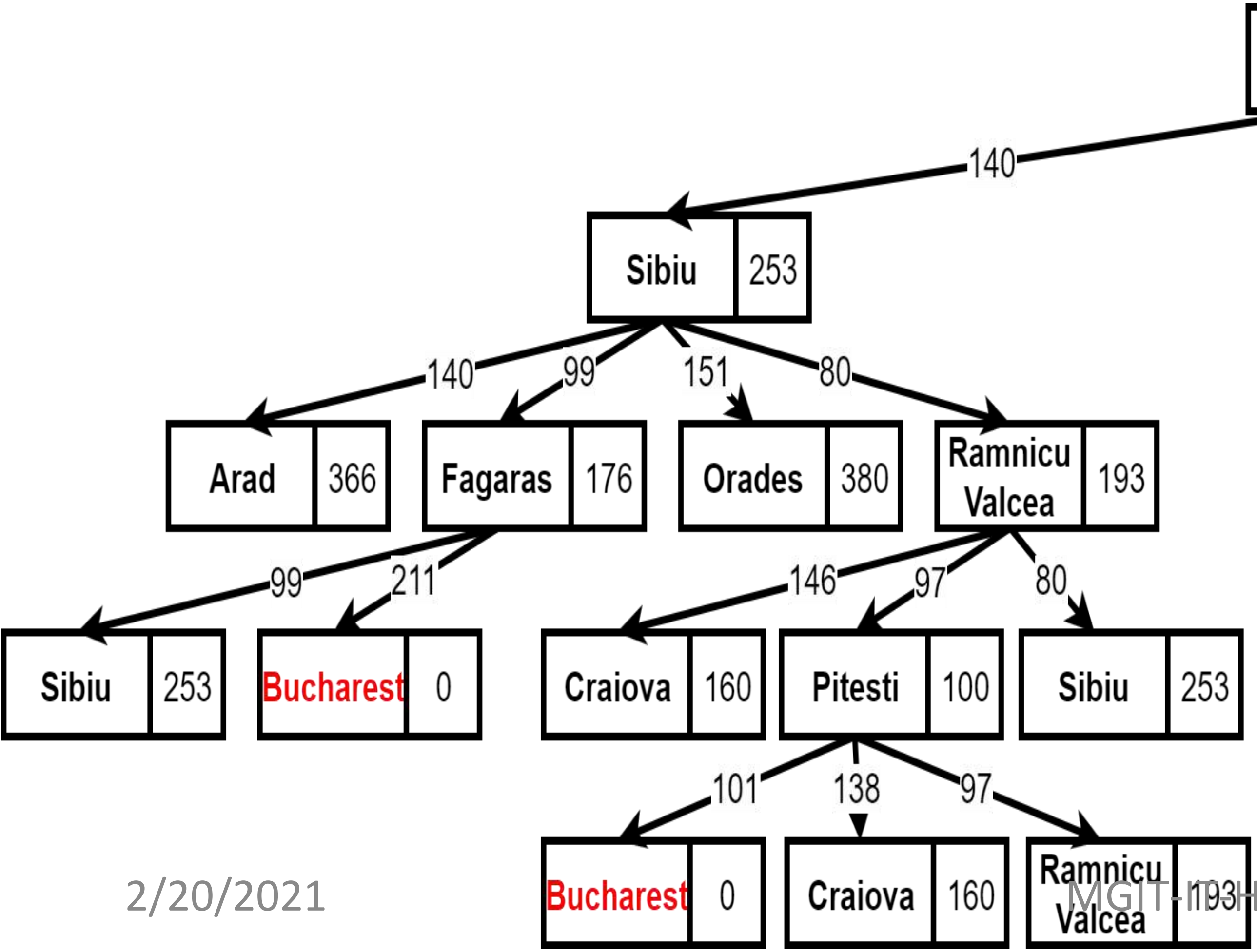
Current Queue

Ramnic	Fagara	Timisoar	Zerin	Arad	Orade
413	415	447	449	646	671

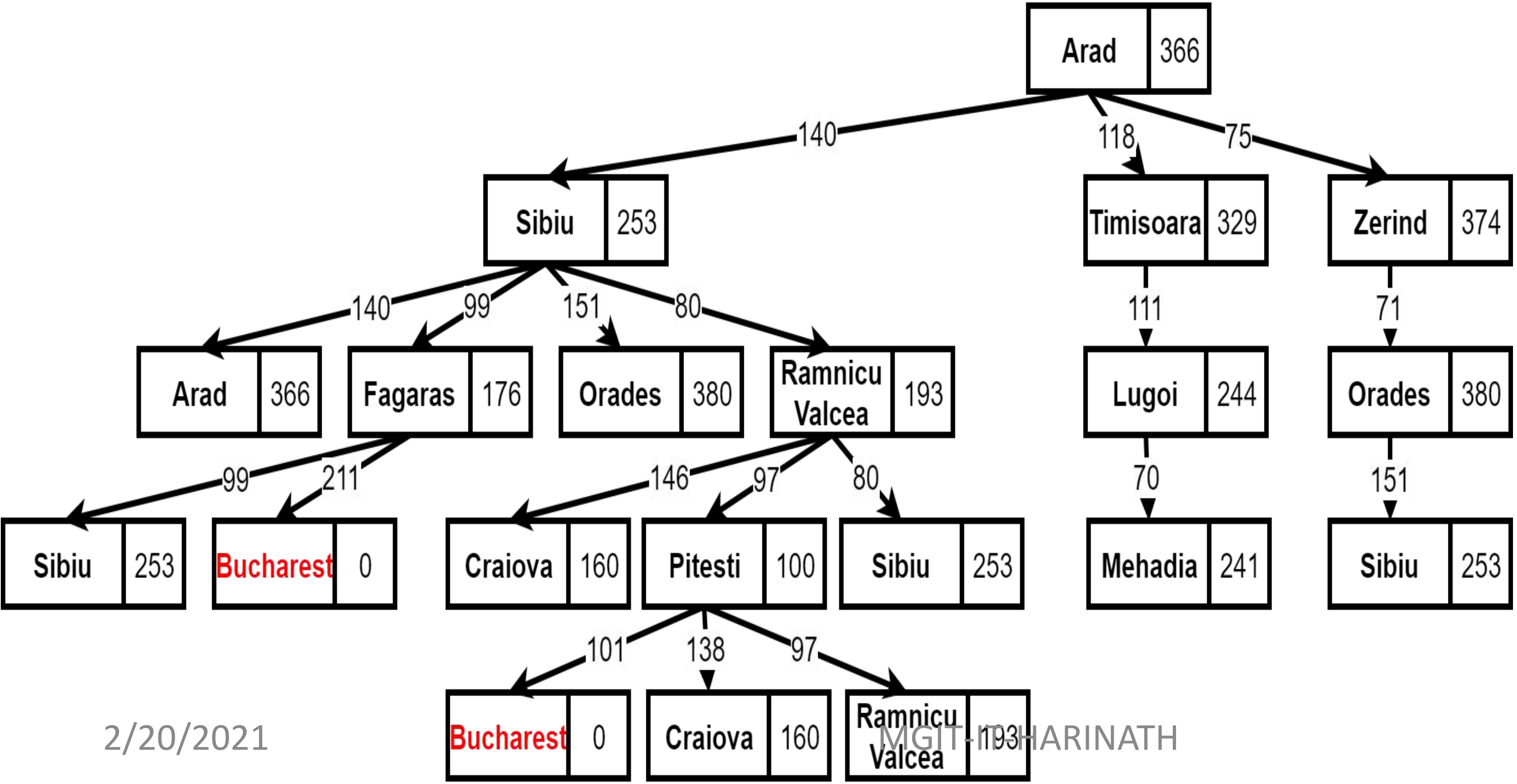
Ramnicu


Arad 366
140
118
75
Sibiu 253
Timisoara 329
Zerind 374
140
99
151
80
111
71
Arad 366
Fagaras 176
Orades 380
Ramnicu Valcea 193
Lugoi 244
Orades 380
99
211
146
97
80
70
151
Sibiu 253
Bucharest 0
Craiova 160
Pitesti 100
Sibiu 253
Mehadia 241
Sibiu 253
101
138
97
Bucharest 0
Craiova 160
Ramnicu Valcea 193

Ramnicu


Arad 366
140
118
75
Sibiu 253
Timisoara 329
Zerind 374
140
99
151
80
111
71
Arad 366
Fagaras 176
Orades 380
Ramnicu Valcea 193
Lugoi 244
Orades 380
99
211
146
97
80
70
151
Sibiu 253
Bucharest 0
Craiova 160
Pitesti 100
Sibiu 253
Mehadia 241
Sibiu 253
101
138
97
Bucharest 0
Craiova 160
Ramnicu Valcea 193

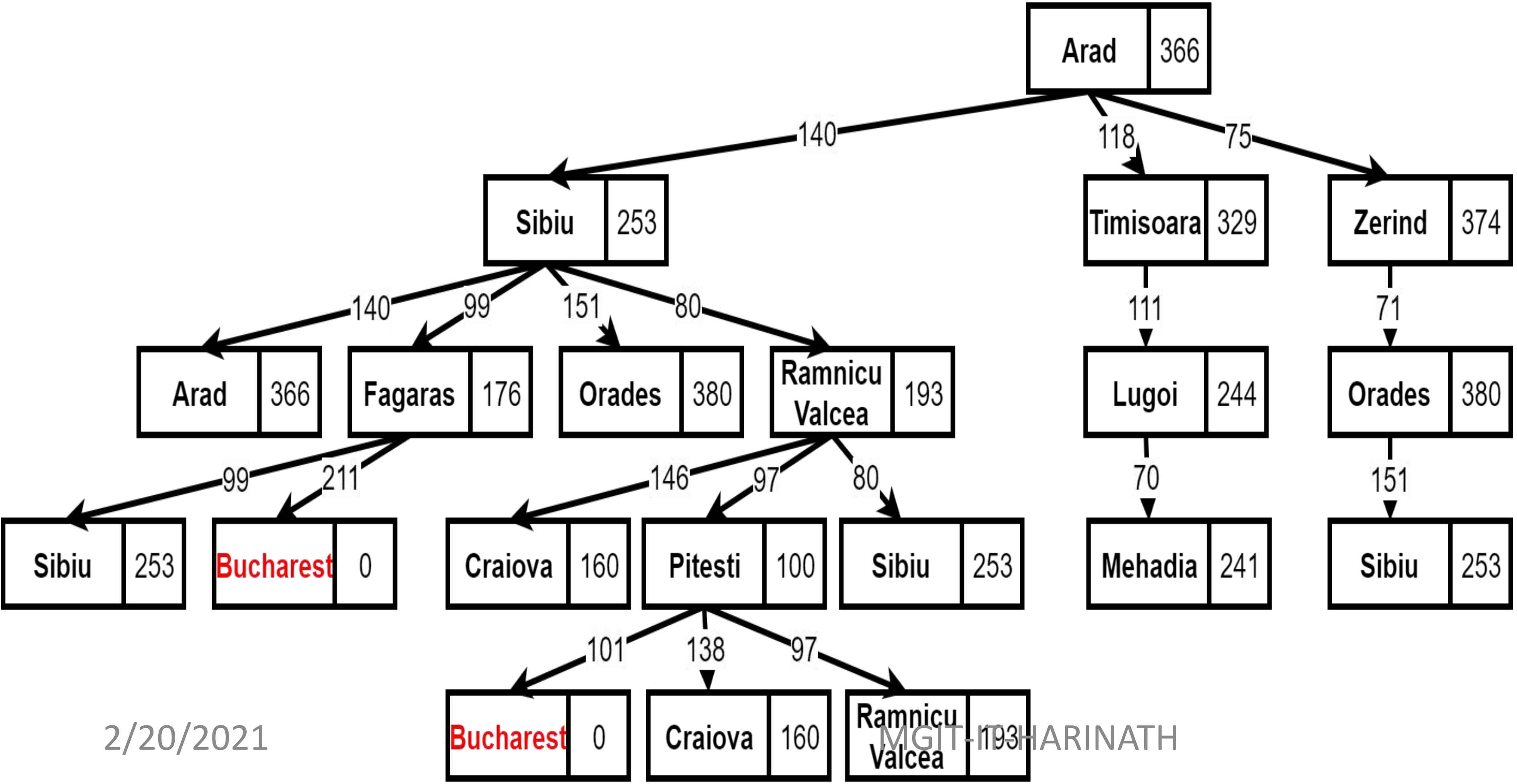
Ramnicu



Ramnicu Children

Craiova

Ramnicu

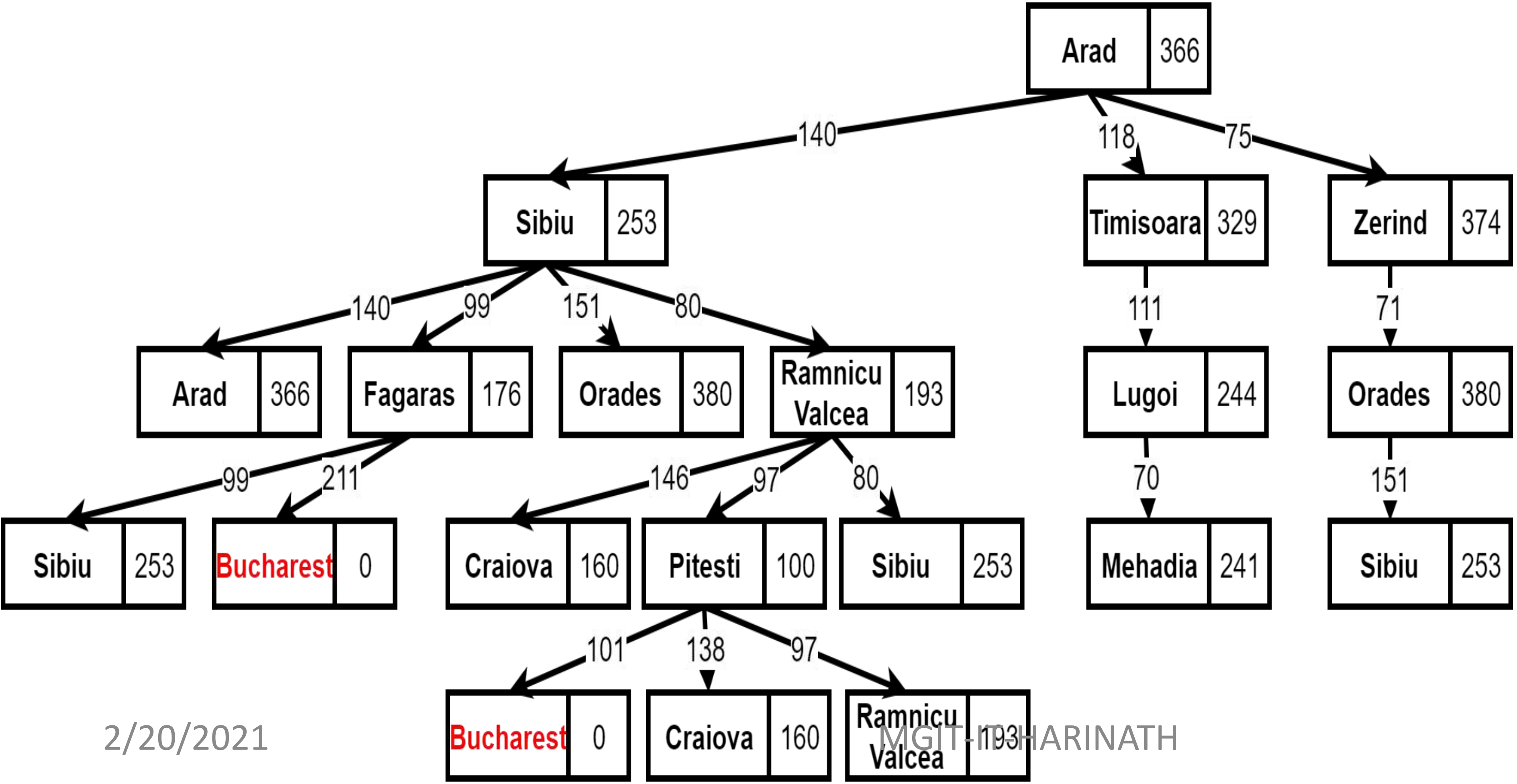


Ramnicu Children

Craiova

Pitesti

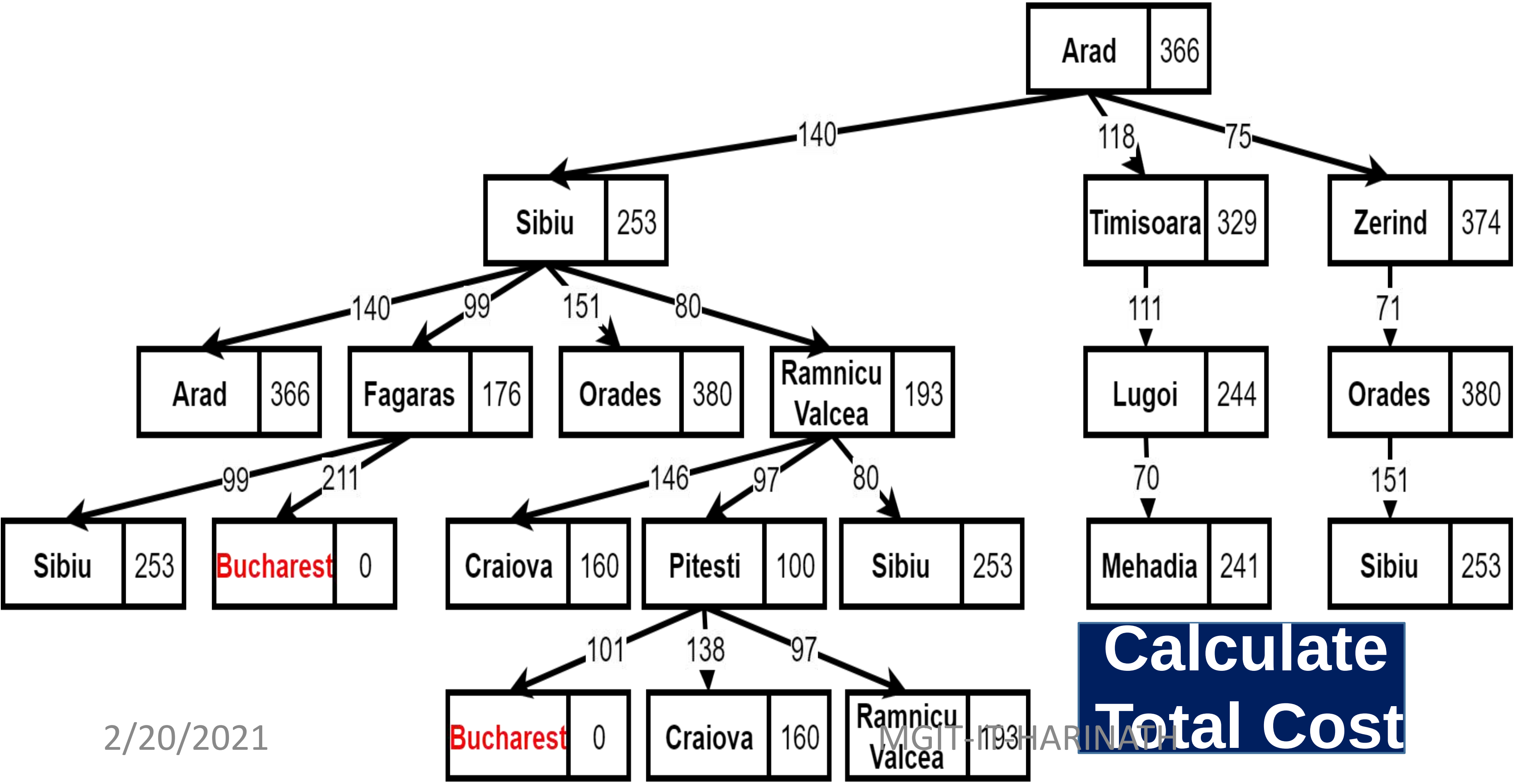
Ramnicu



Ramnicu Children

- Craiova
- Pitesti
- Sibiu

Ramnicu



Ramnicu Children

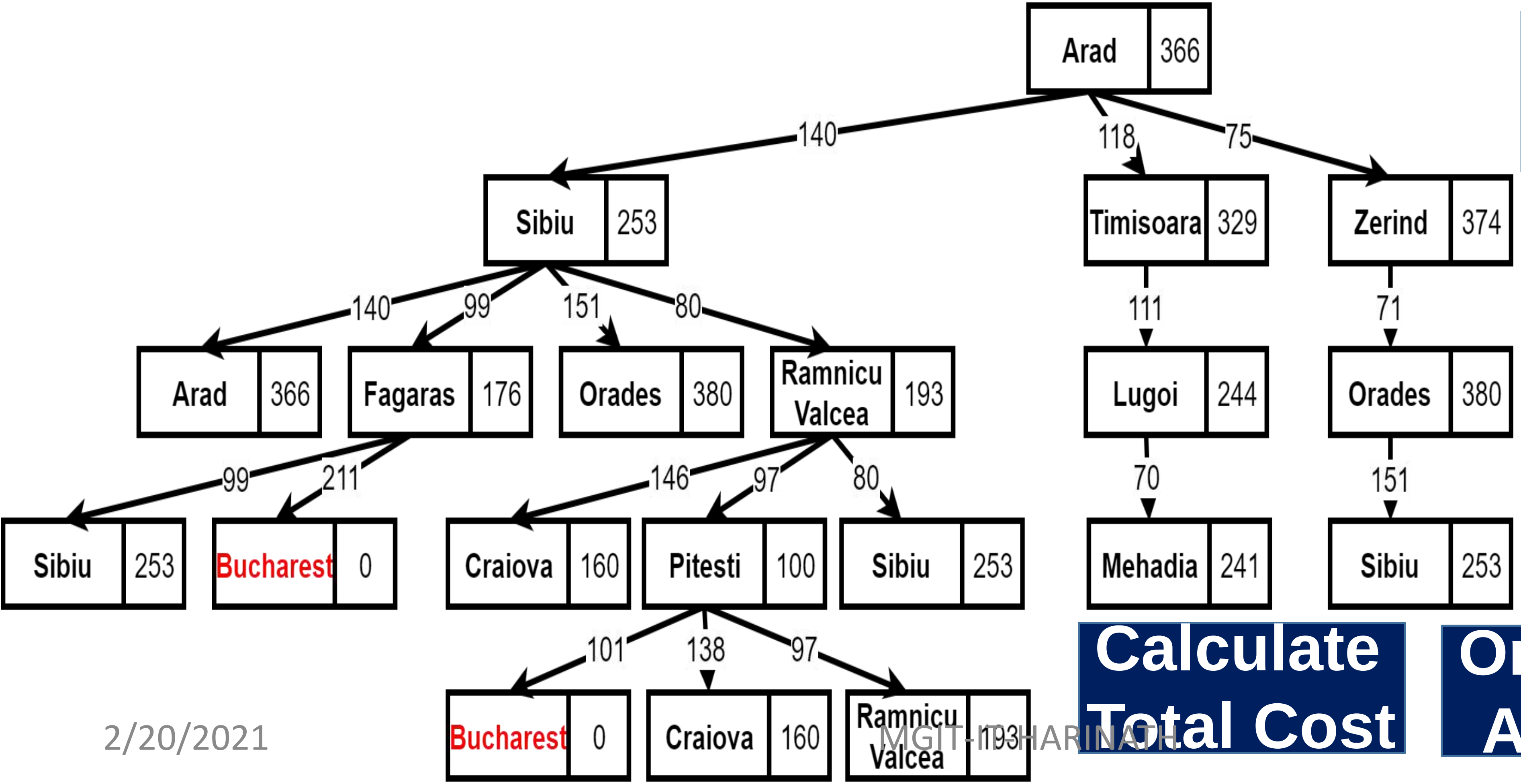
Craiova

Pitesti

Sibiu

Calculate Total Cost

Ramnicu



Ramnicu Children

Craiova

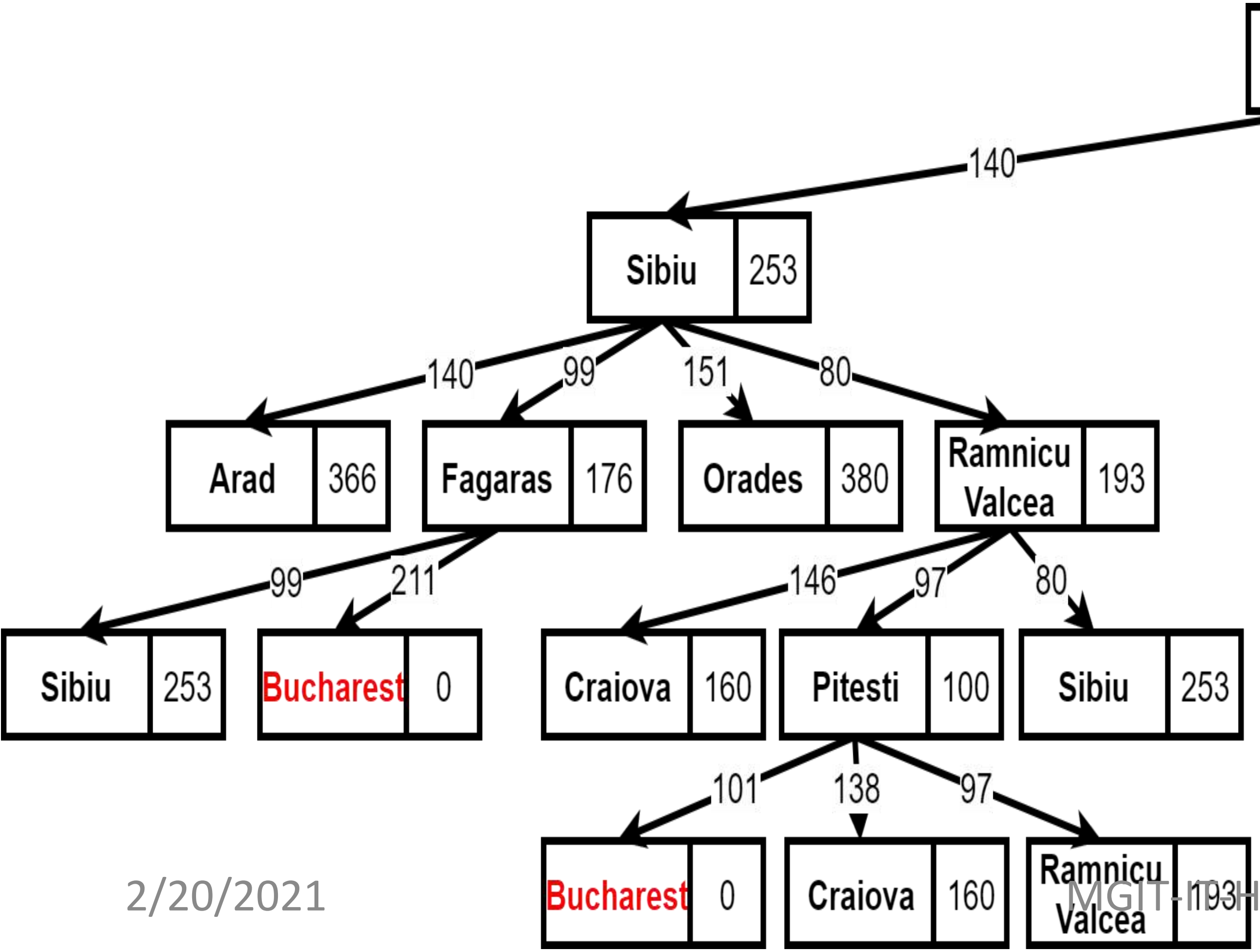
Pitesti

Sibiu

Calculate Total Cost

Order Asc.

Ramnicu



Ramnicu Children

Craiova

Pitesti

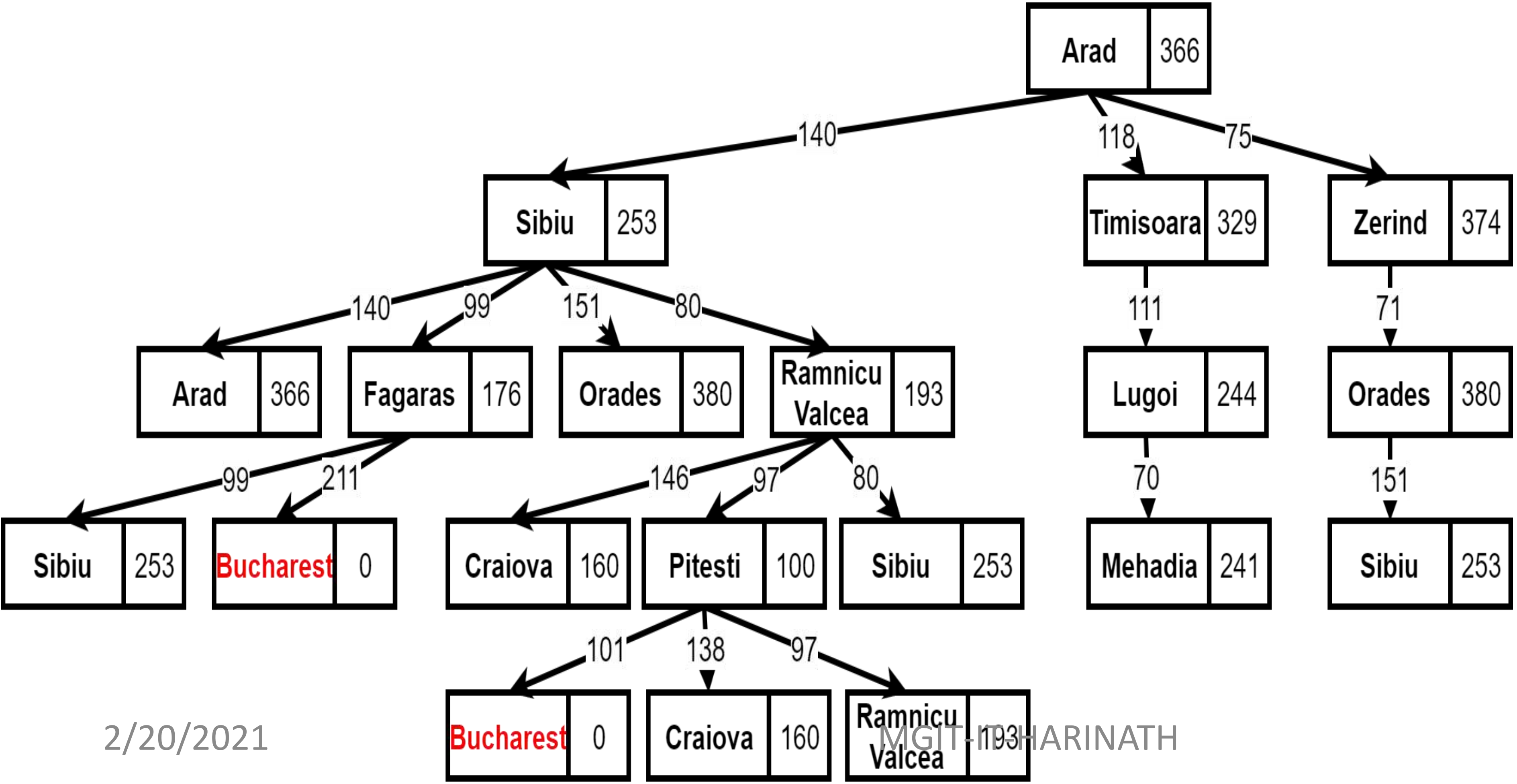
Sibiu

Calculate Total Cost

Order Asc.

Select Min. Cost

Craiova

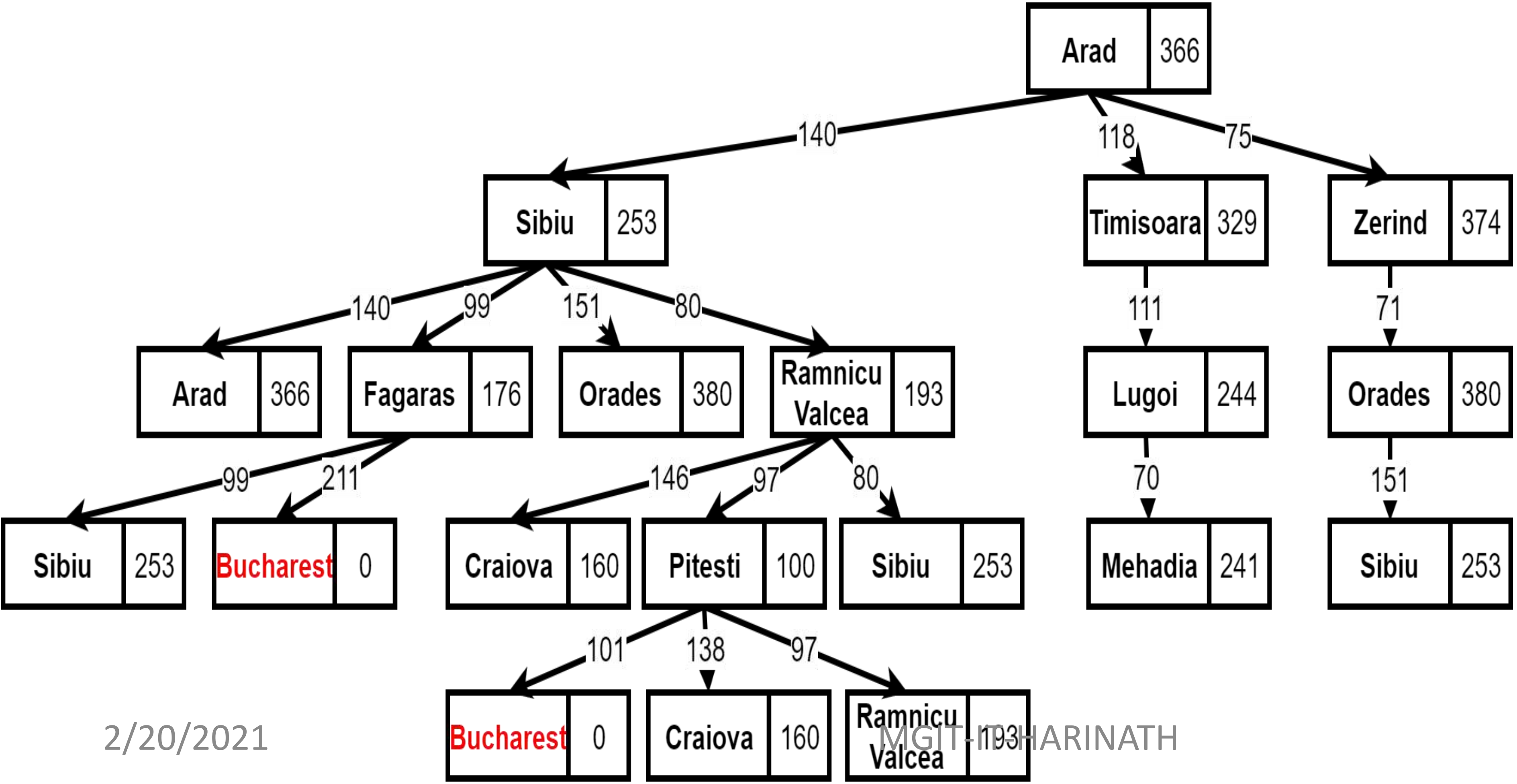


Ramnicu Children

- Craiova
- Pitesti
- Sibiu

Craiova

Heuristic



Ramnicu Children

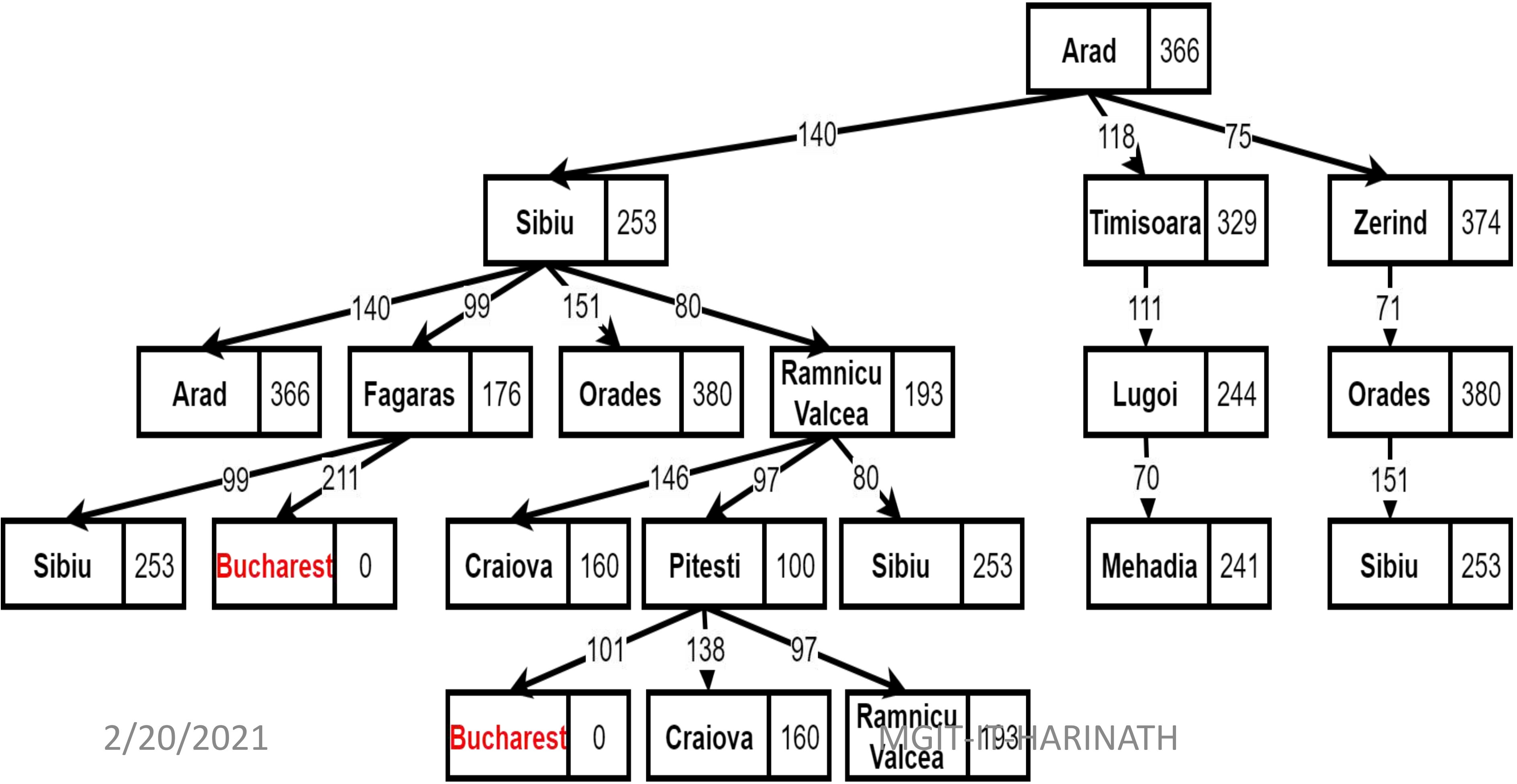
Craiova

Pitesti

Sibiu

Craiova

Heuristic + Cost



Ramnicu Children

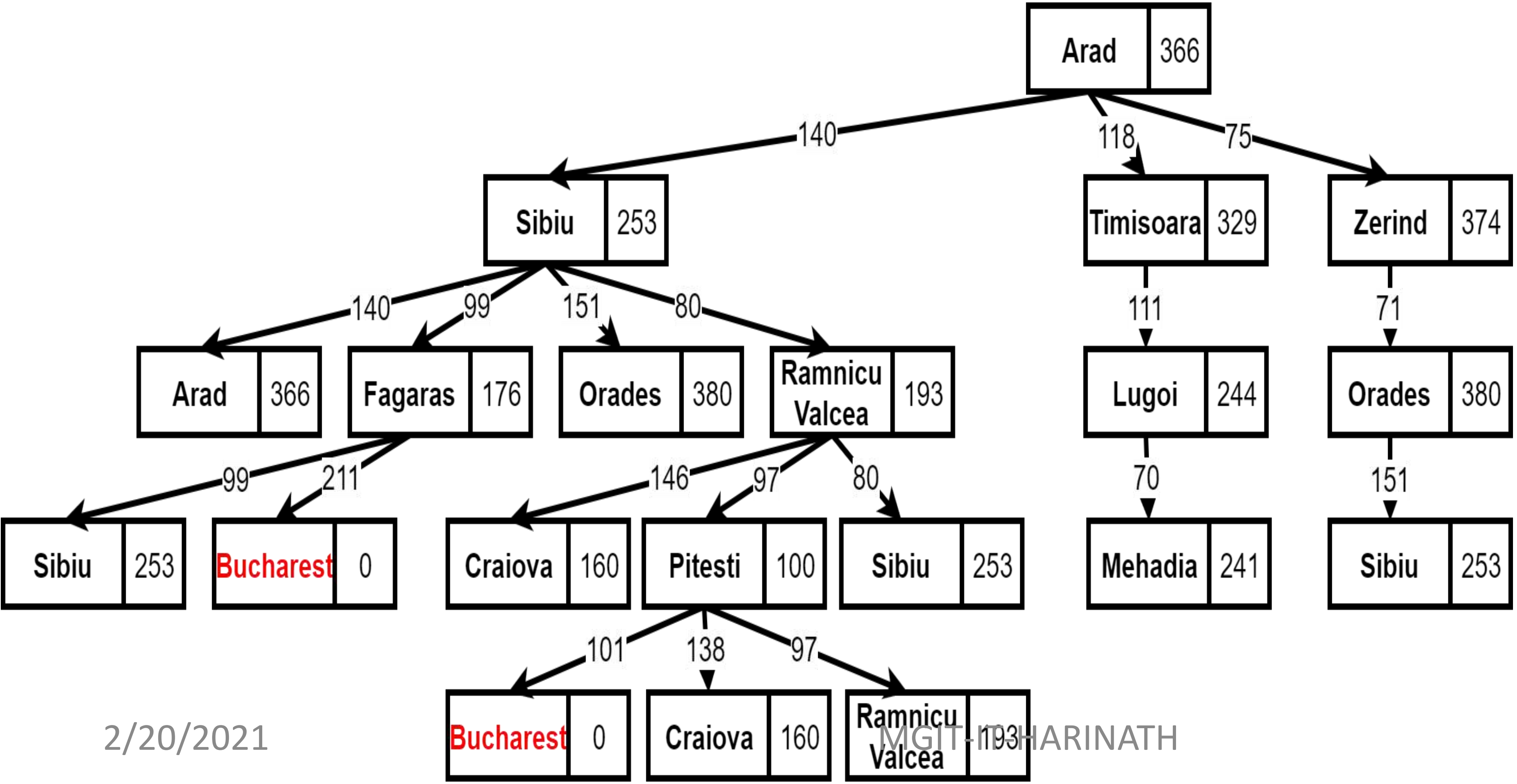
Craiova

Pitesti

Sibiu

Craiova

Heuristic + Cost = Total



Ramnicu Children

Craiova

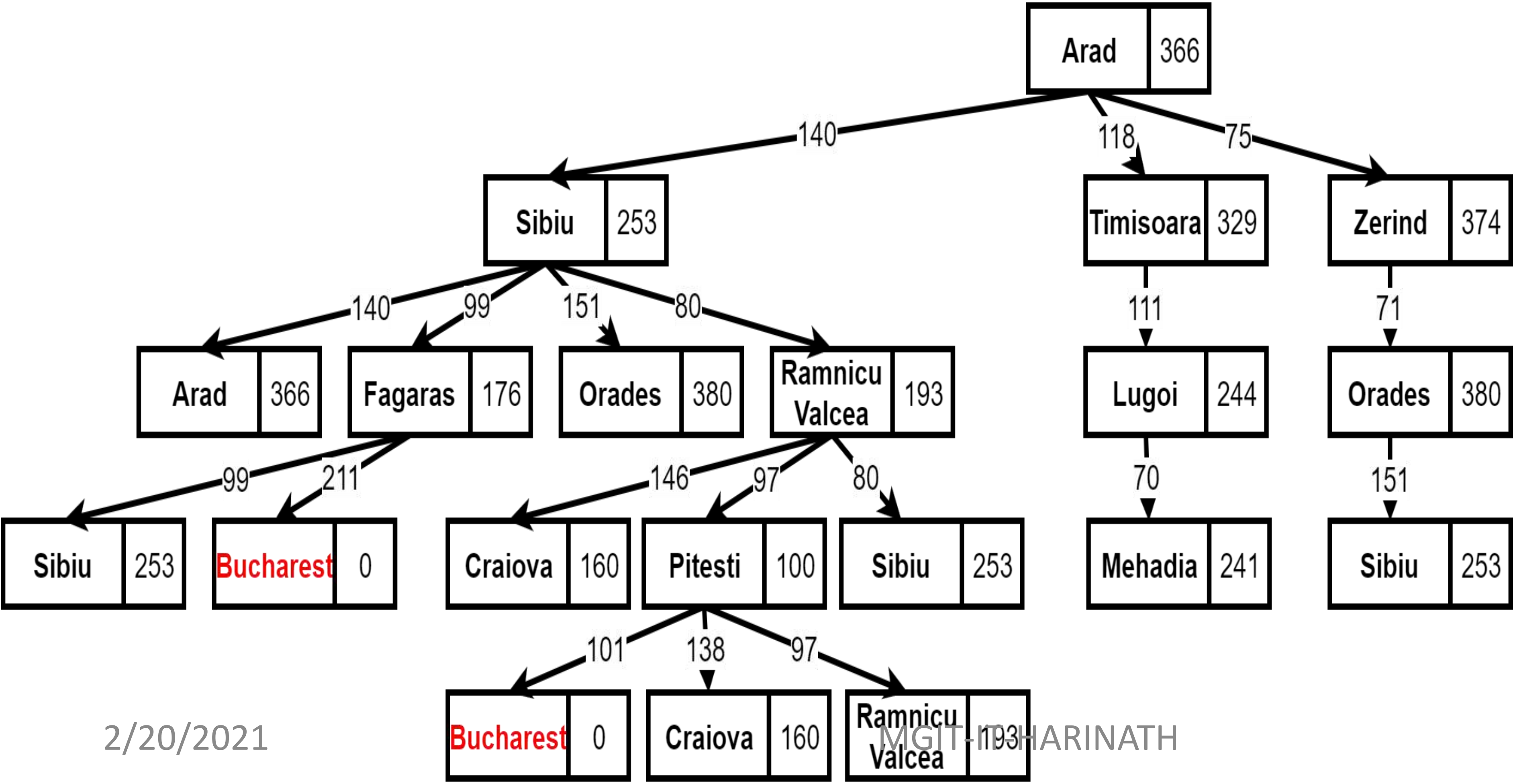
Pitesti

Sibiu

Craiova

Heuristic + **Cost** = **Total**

160



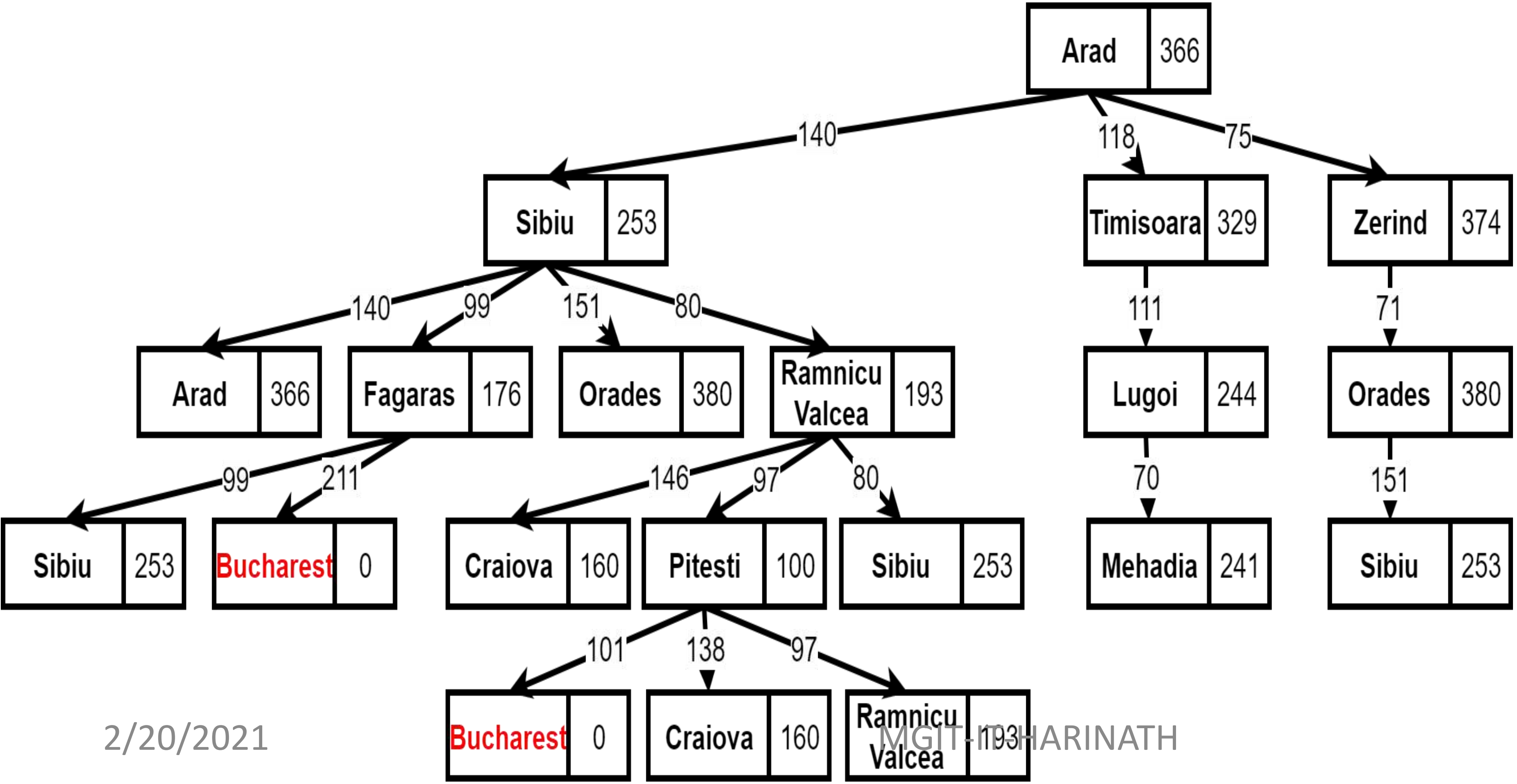
Ramnicu Children

- Craiova
- Pitesti
- Sibiu

Craiova

Heuristic + Cost = Total

160 + 366



Ramnicu Children

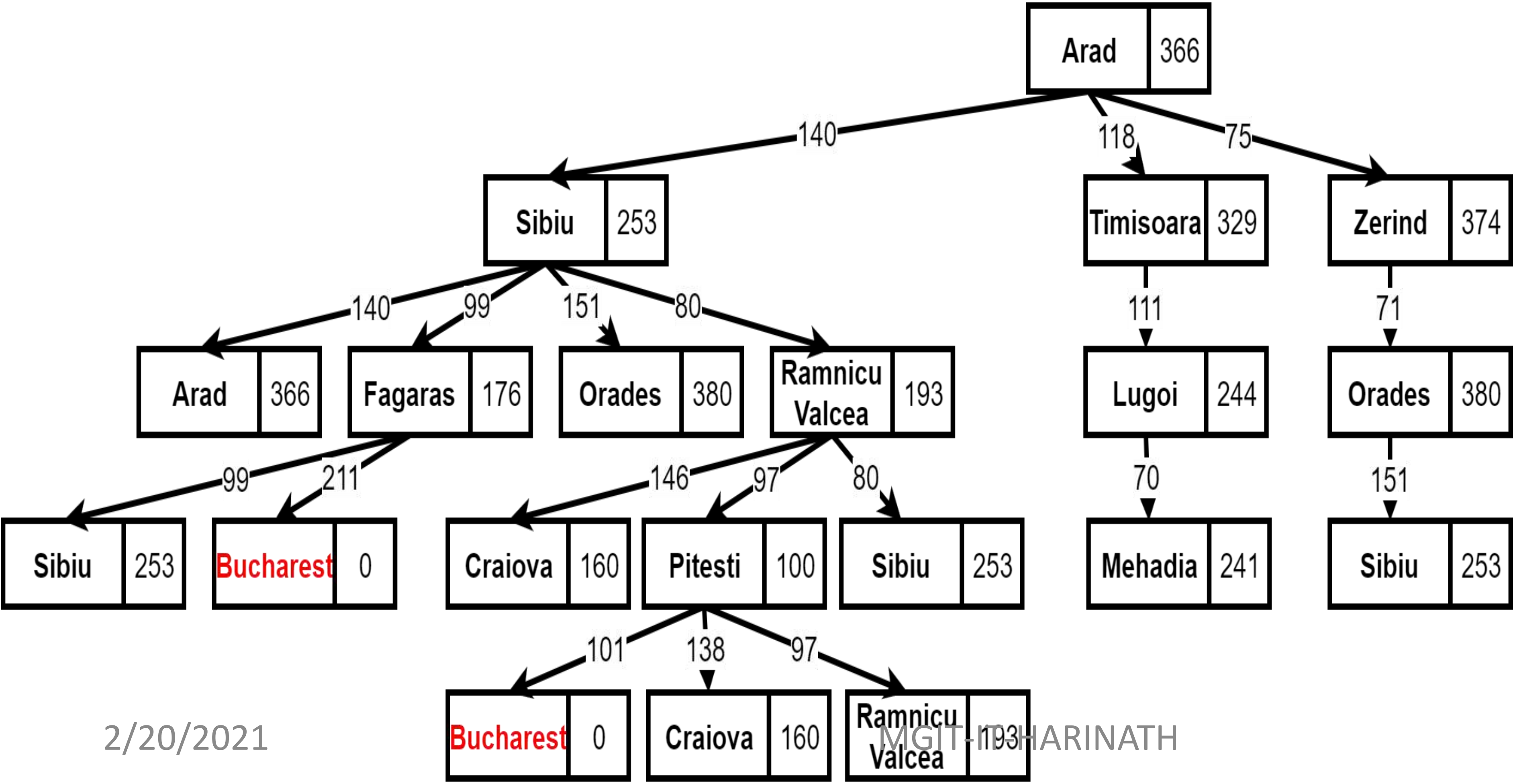
Craiova

Pitesti

Sibiu

Craiova

Heuristic	+	Cost	=	Total
160	+	366	=	526



Ramnicu Children

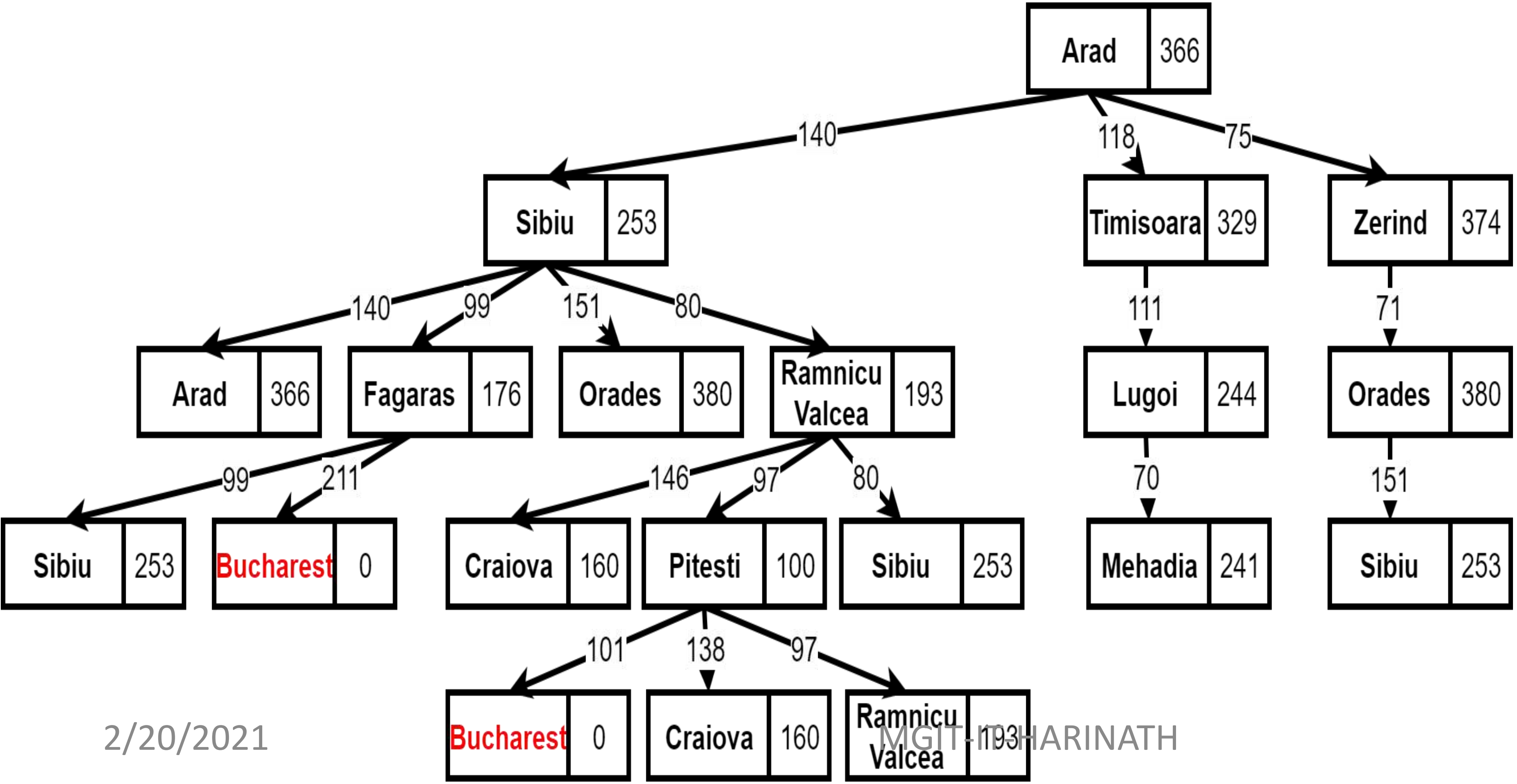
Craiova

Pitesti

Sibiu

Craiova

Heuristic	+	Cost	=	Total
160	+	366	=	526

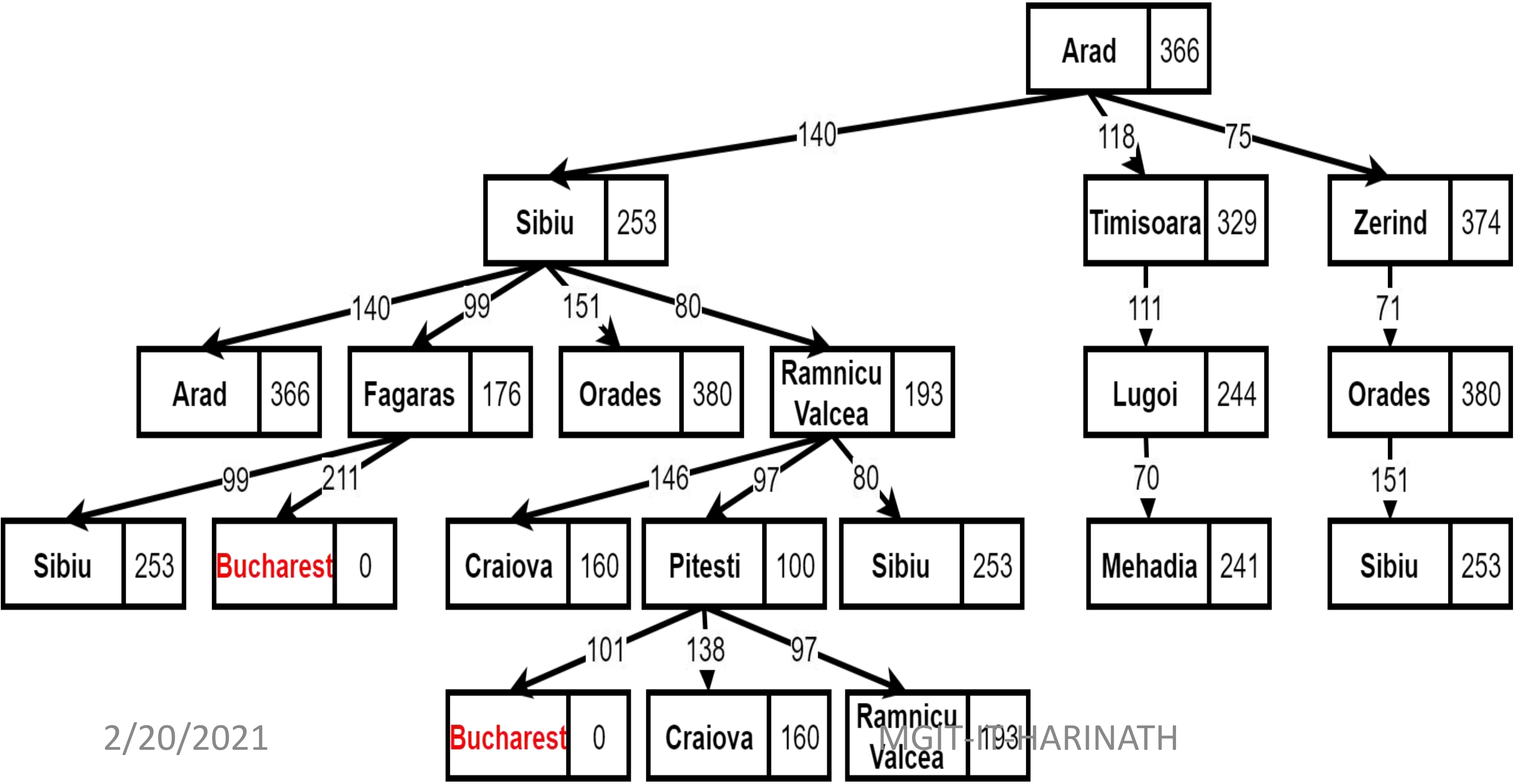


Ramnicu Children

Craiova	526
Pitesti	
Sibiu	

Pitesti

Heuristic + Cost = Total



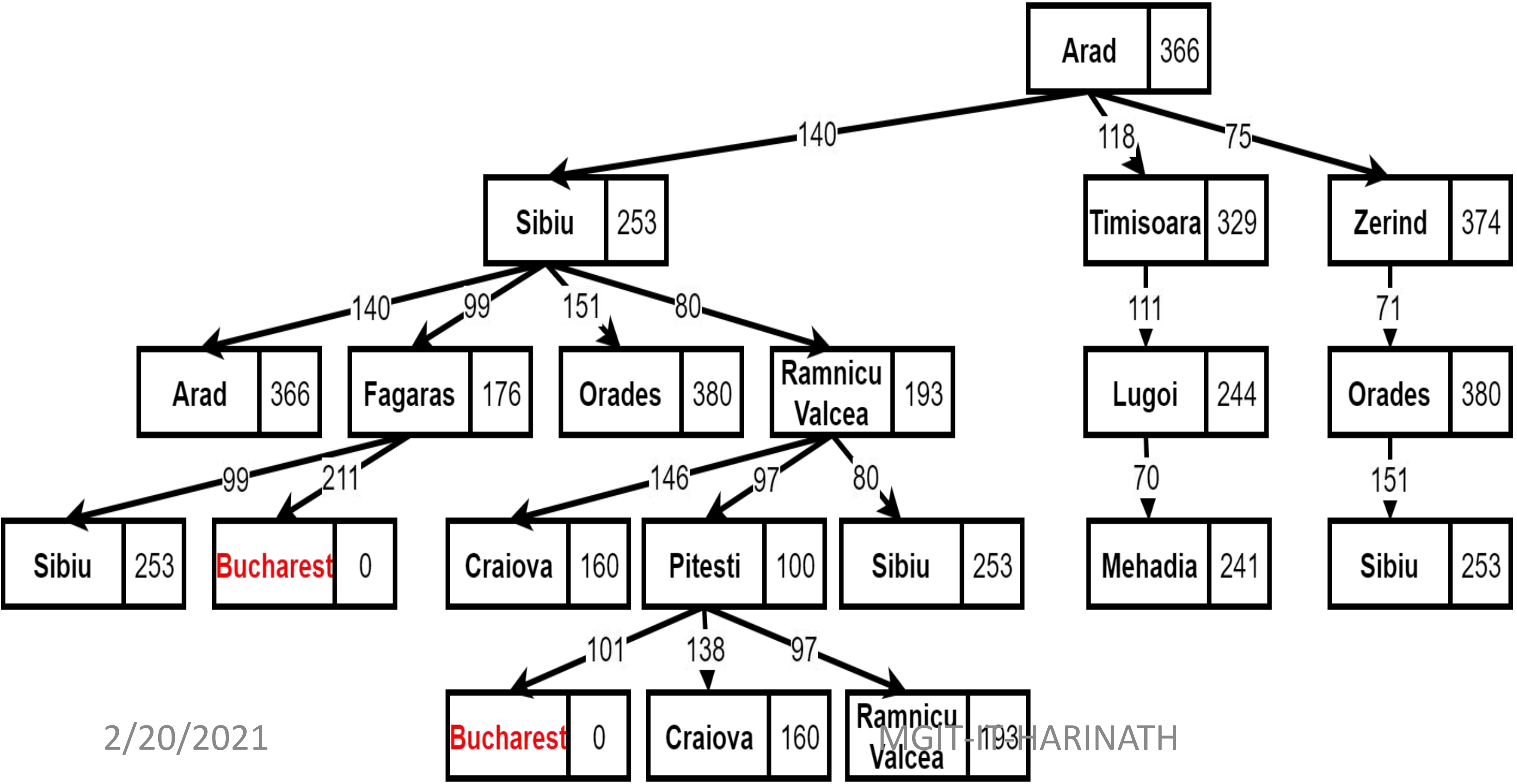
Ramnicu Children

Craiova	526
Pitesti	
Sibiu	

Pitesti

Heuristic + **Cost** = **Total**

100



Ramnicu Children

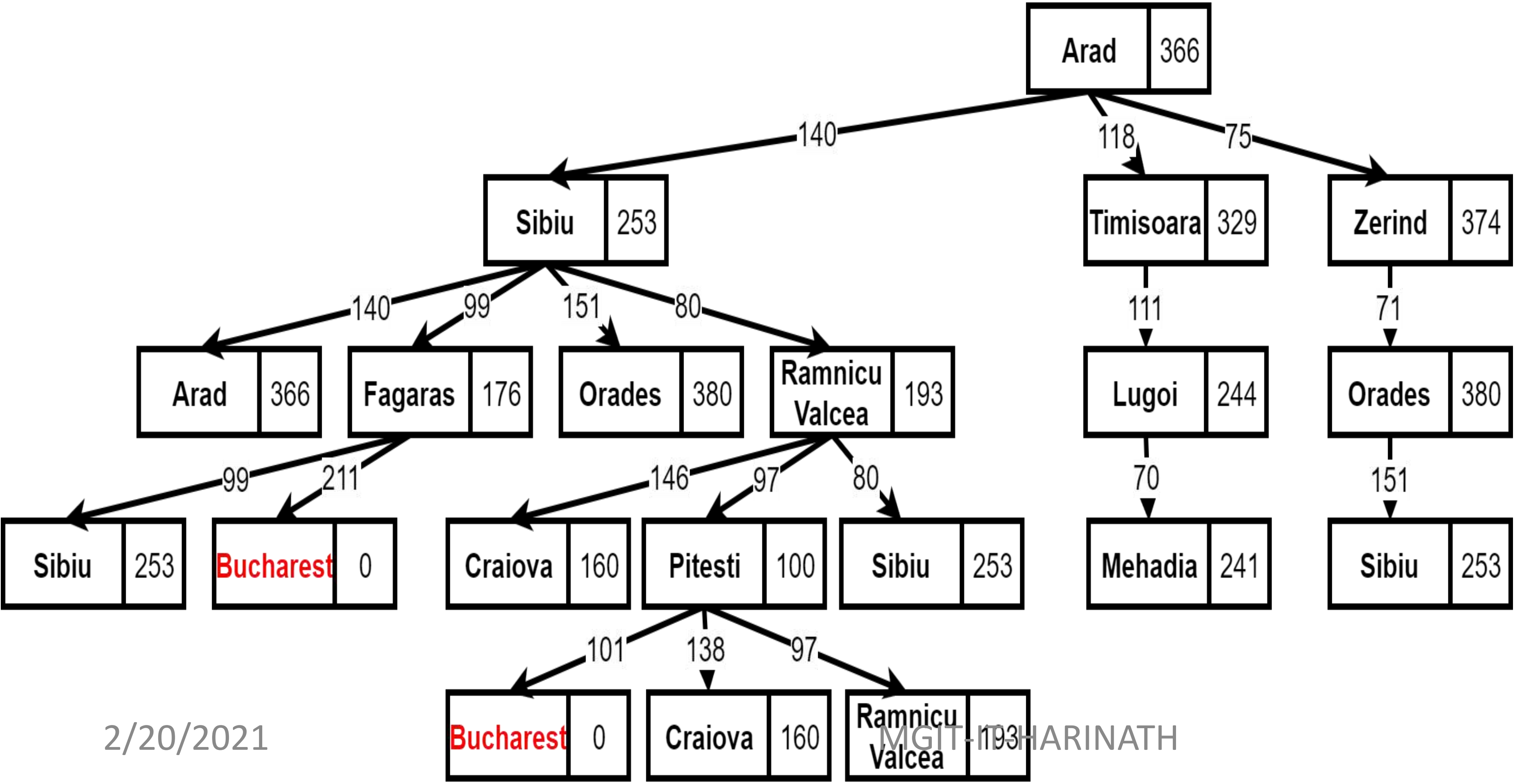
Craiova 526

Pitesti

Sibiu

Pitesti

Heuristic	+	Cost	=	Total
100	+	317		

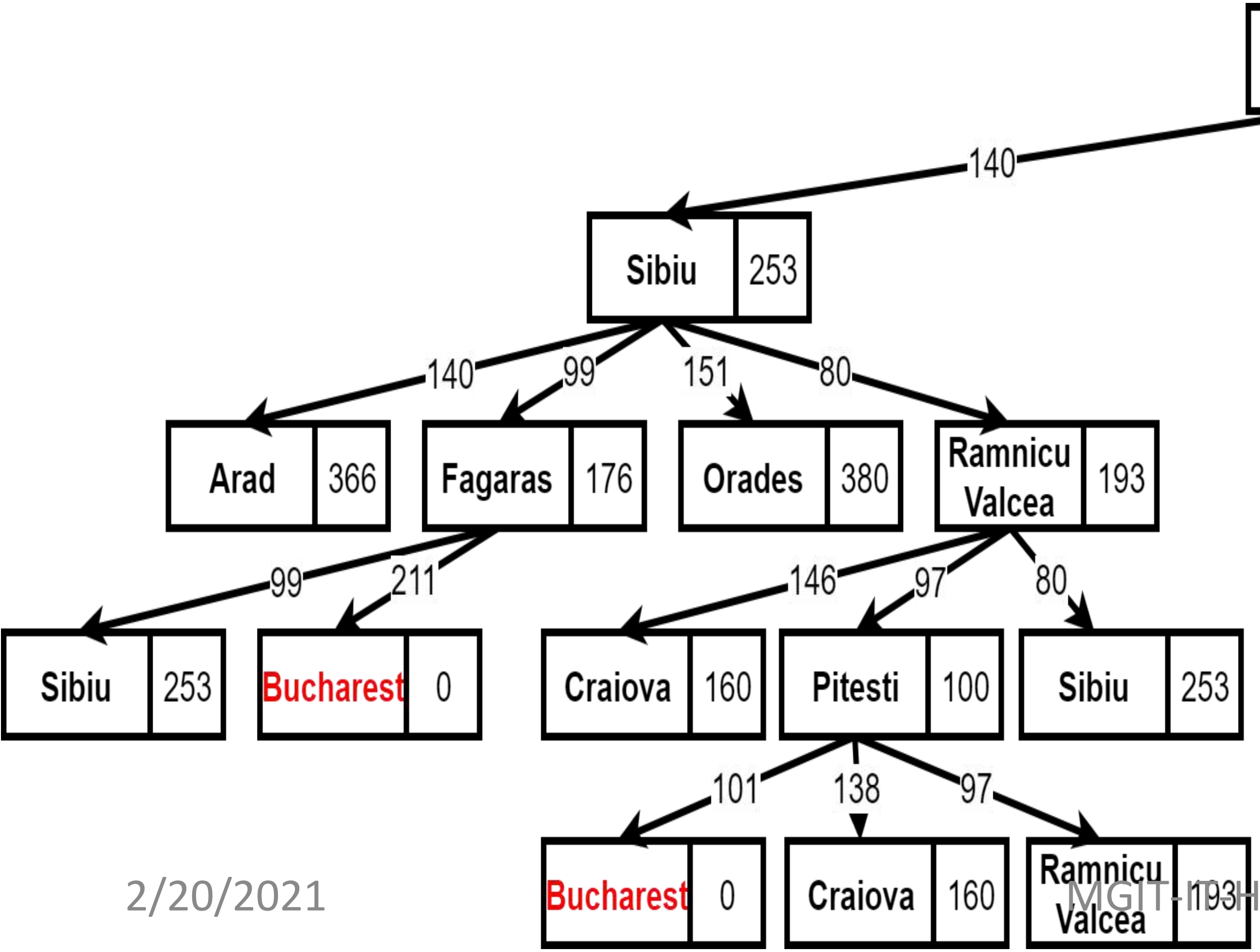


Ramnicu Children

Craiova	526
Pitesti	
Sibiu	

Pitesti

Heuristic	+	Cost	=	Total
100	+	317	=	417

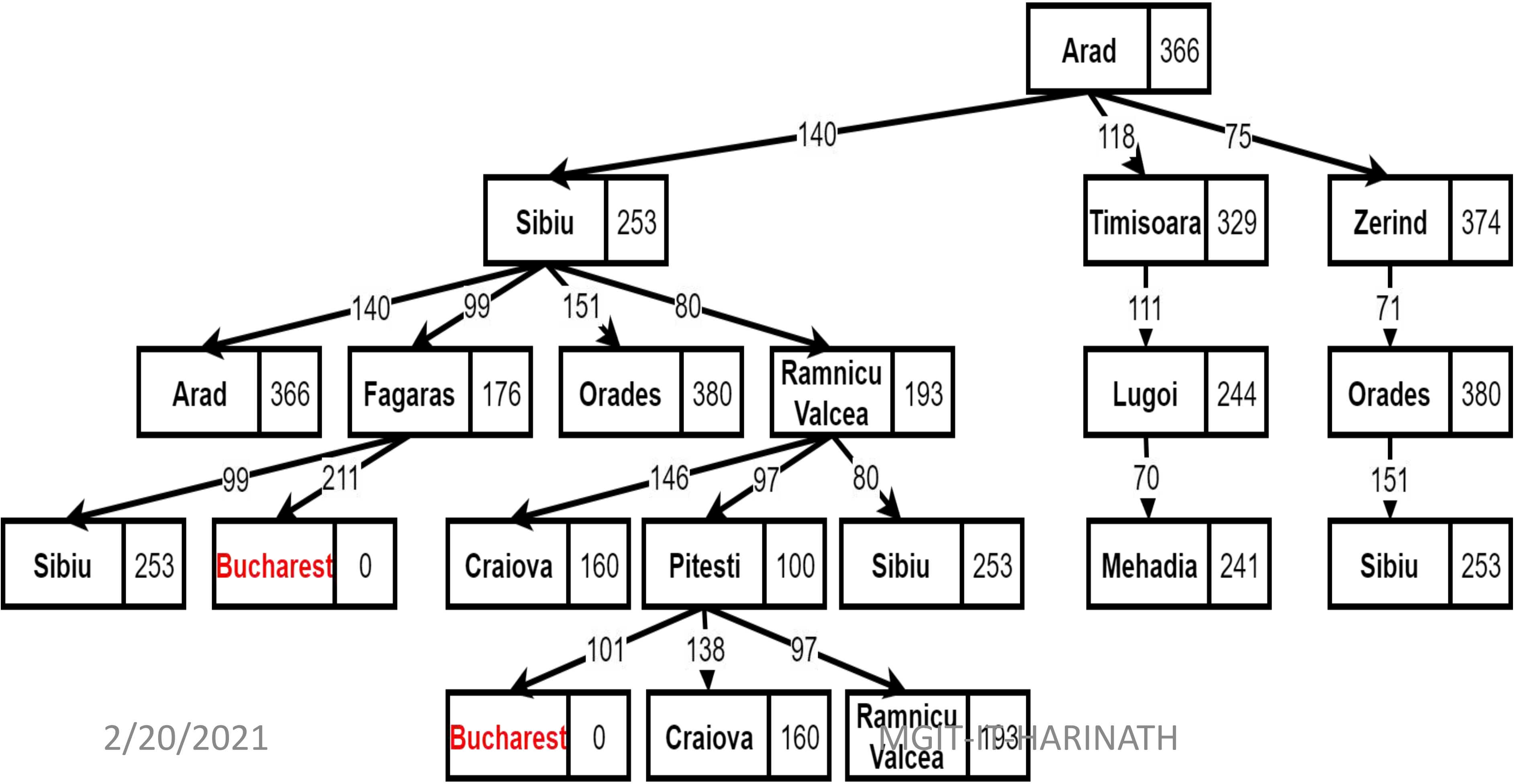


Ramnicu Children

Craiova	526
Pitesti	
Sibiu	

Pitesti

Heuristic	+	Cost	=	Total
100	+	317	=	417

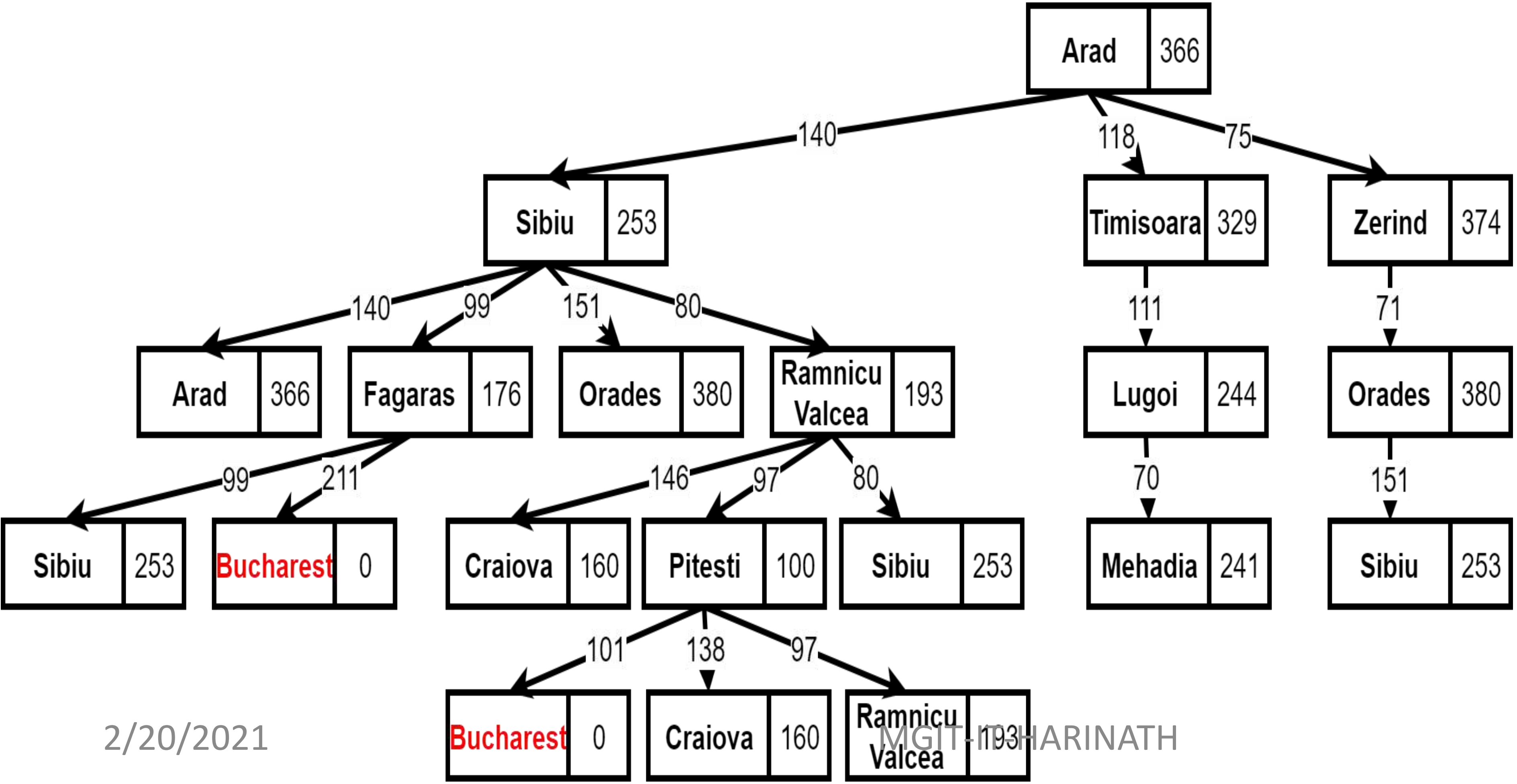


Ramnicu Children

Craiova	526
Pitesti	417
Sibiu	

Sibiu

Heuristic + **Cost** = **Total**



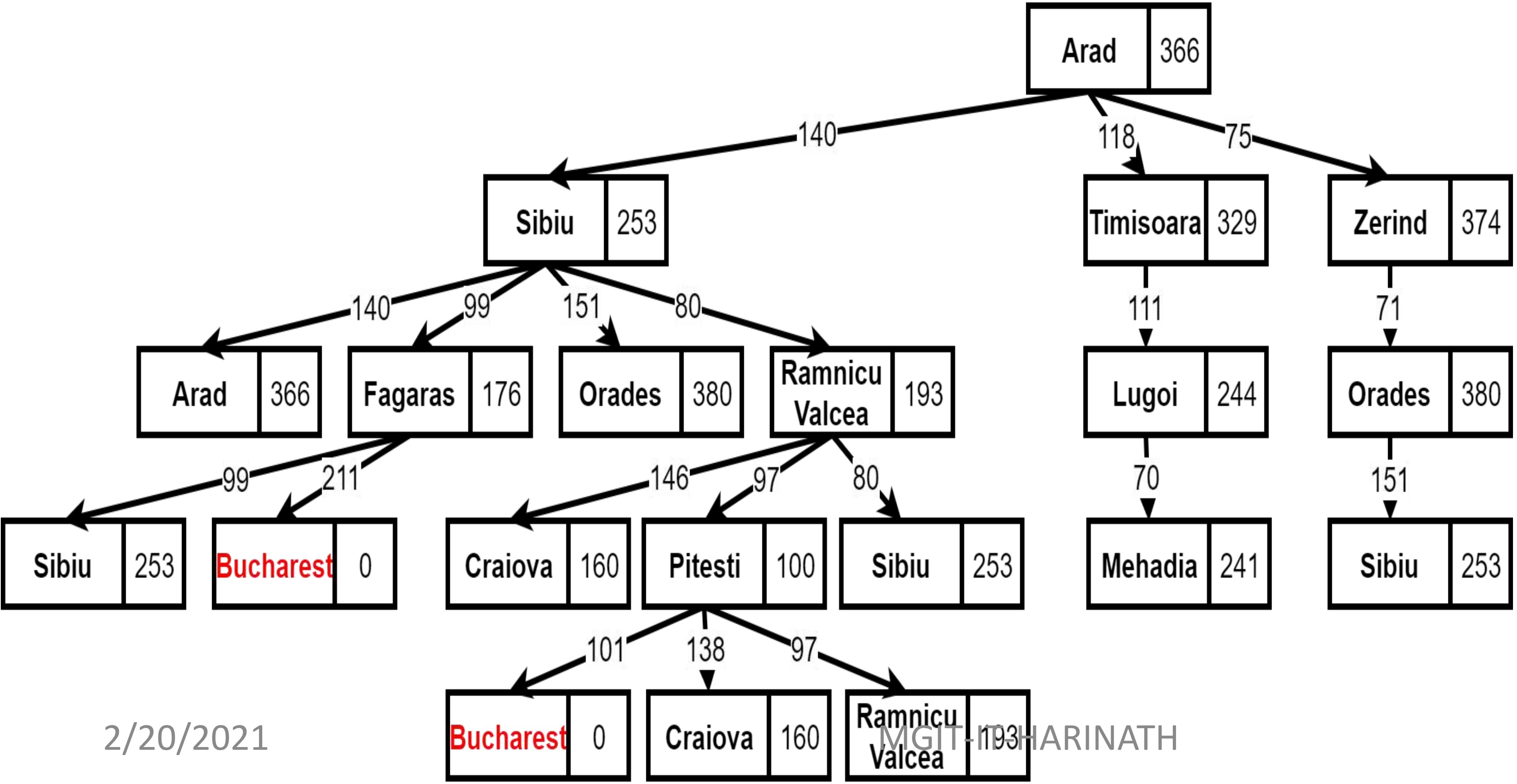
Ramnicu Children

Craiova	526
Pitesti	417
Sibiu	

Sibiu

Heuristic + **Cost** = **Total**

253

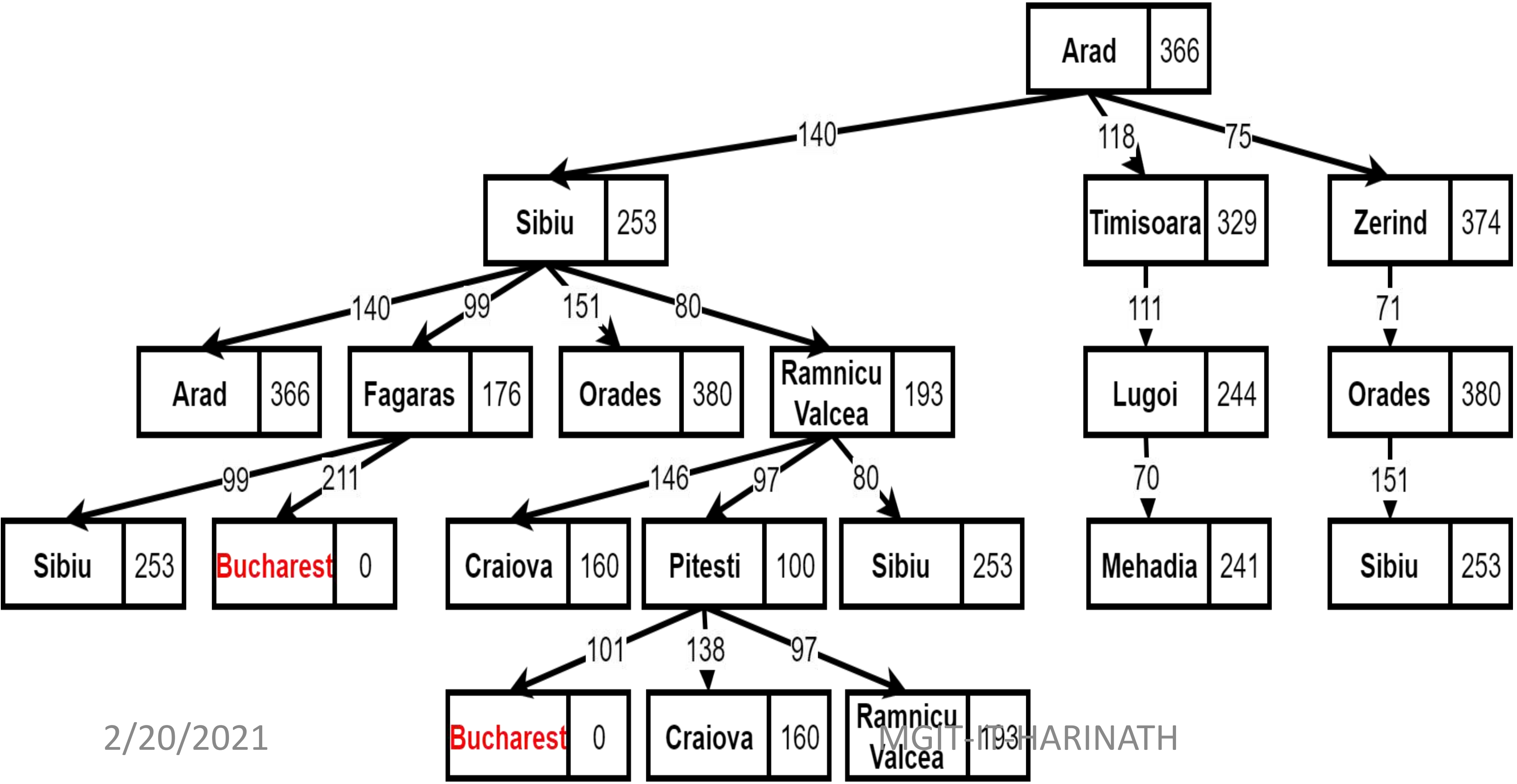


Ramnicu Children

Craiova	526
Pitesti	417
Sibiu	

Sibiu

Heuristic	+	Cost	=	Total
253	+	300		

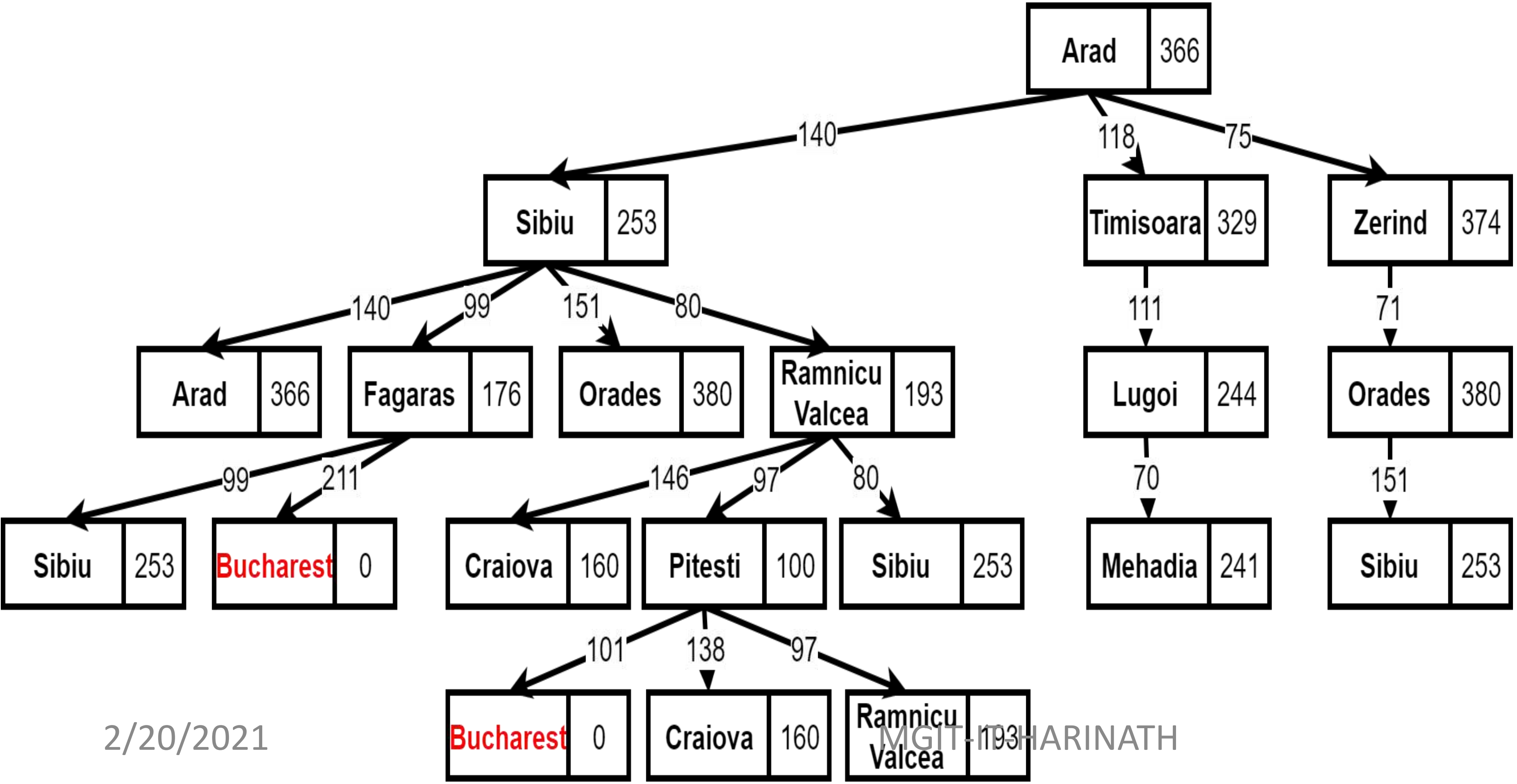


Ramnicu Children

Craiova	526
Pitesti	417
Sibiu	

Sibiu

Heuristic	+	Cost	=	Total
253	+	300	=	553

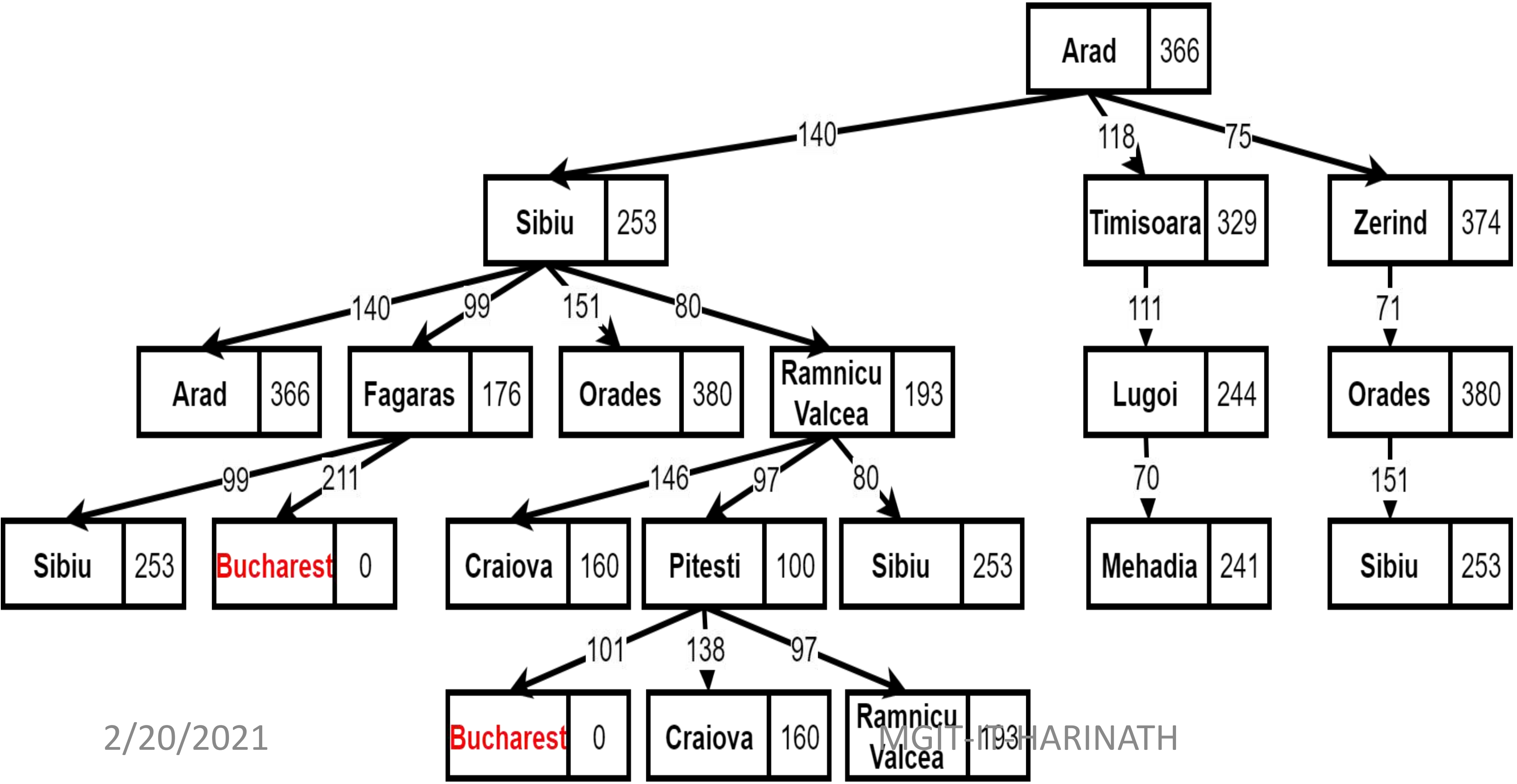


Ramnicu Children

Craiova	526
Pitesti	417
Sibiu	

Sibiu

Heuristic	+	Cost	=	Total
253	+	300	=	553



Ramnicu Children

Craiova	526
Pitesti	417
Sibiu	553

Current Queue

Fagara	Timisoar	Zerin	Arad	Orade
415	447	449	646	671

Current Queue

Fagara	Timisoar	Zerin	Arad	Orade
415	447	449	646	671

Craiova	Pitesti	Sibiu
526	417	553

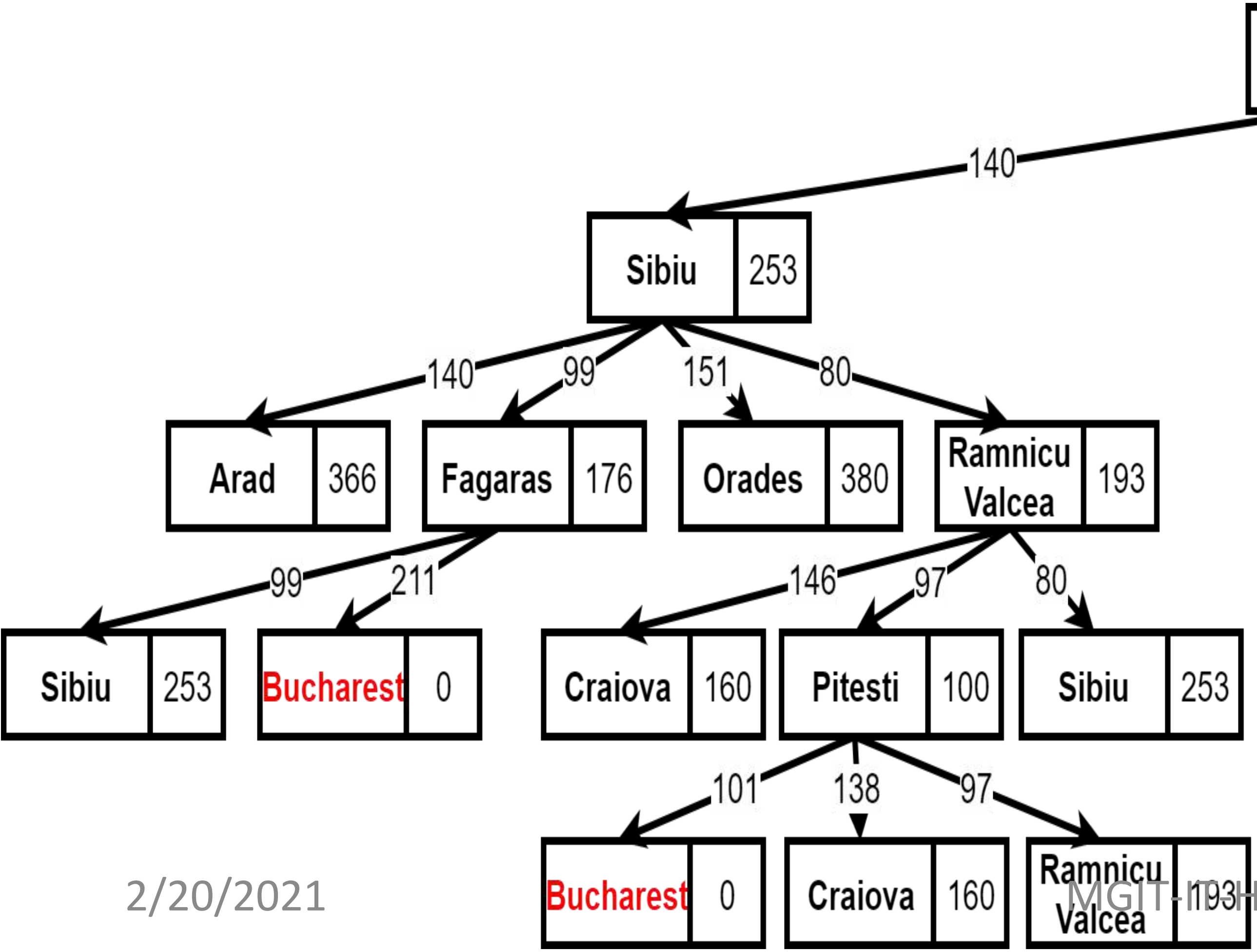
Current Queue

Fagara	Pitesti	Timisoar	Zerin
415	417	447	449
Craiova	Sibiu	Arad	Orade
526	553	646	671

Current Queue

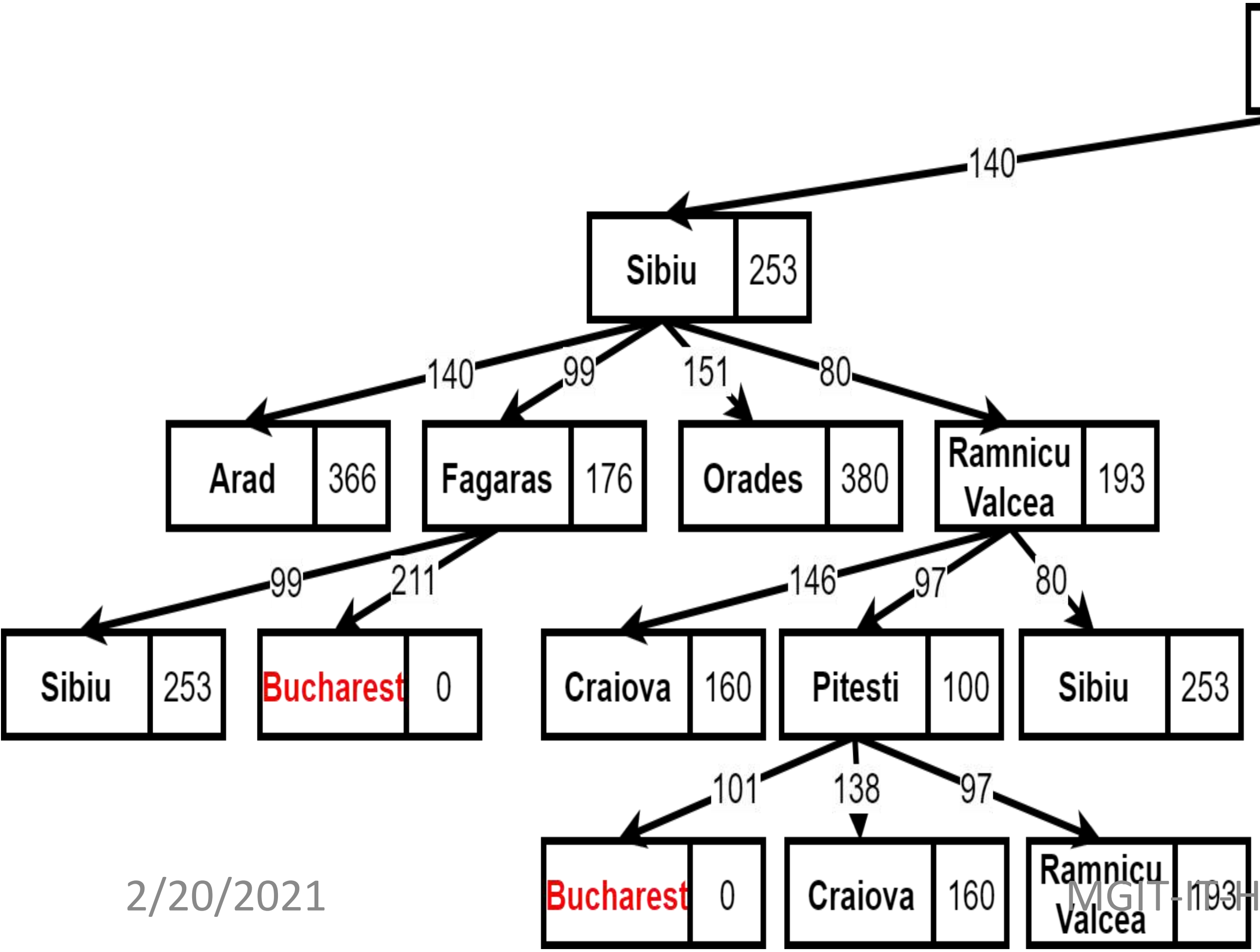
Fagara	Pitesti	Timisoar	Zerin
415	417	447	449
Craiova	Sibiu	Arad	Orade
526	553	646	671

Fagaras



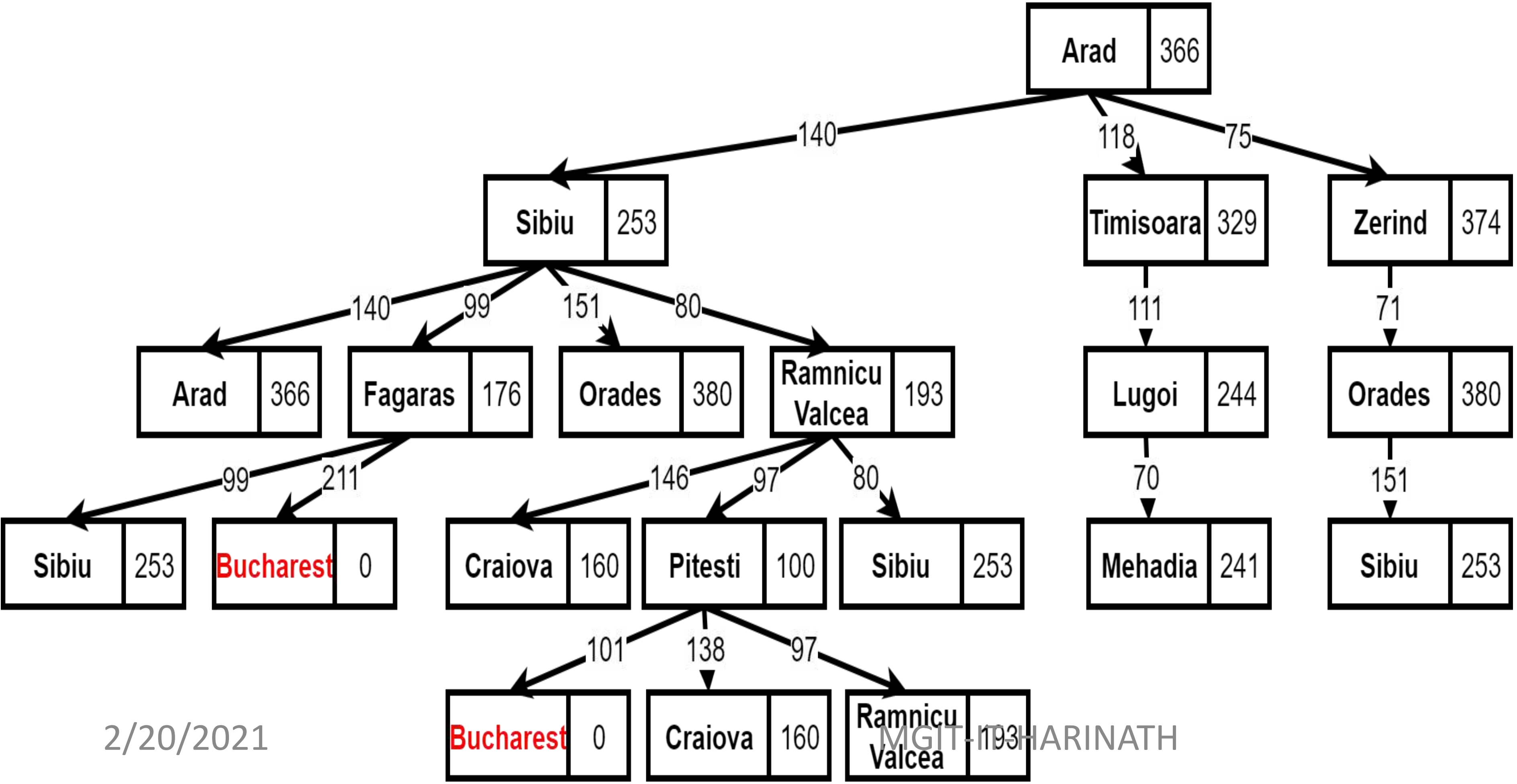
Arad 366
140
118
75
Sibiu 253
Timisoara 329
Zerind 374
140
99
151
80
111
71
Arad 366
Fagaras 176
Orades 380
Ramnicu Valcea 193
Lugoi 244
Orades 380
99
211
146
97
80
70
151
Sibiu 253
Bucharest 0
Craiova 160
Pitesti 100
Sibiu 253
Mehadia 241
Sibiu 253
101
138
97
Bucharest 0
Craiova 160
Ramnicu Valcea 193

Fagaras



Arad 366
140
118
75
Sibiu 253
Timisoara 329
Zerind 374
140
99
151
80
111
71
Arad 366
Fagaras 176
Orades 380
Ramnicu Valcea 193
Lugoi 244
Orades 380
99
211
146
97
80
70
151
Sibiu 253
Bucharest 0
Craiova 160
Pitesti 100
Sibiu 253
Mehadia 241
Sibiu 253
101
138
97
Bucharest 0
Craiova 160
Ramnicu Valcea 193

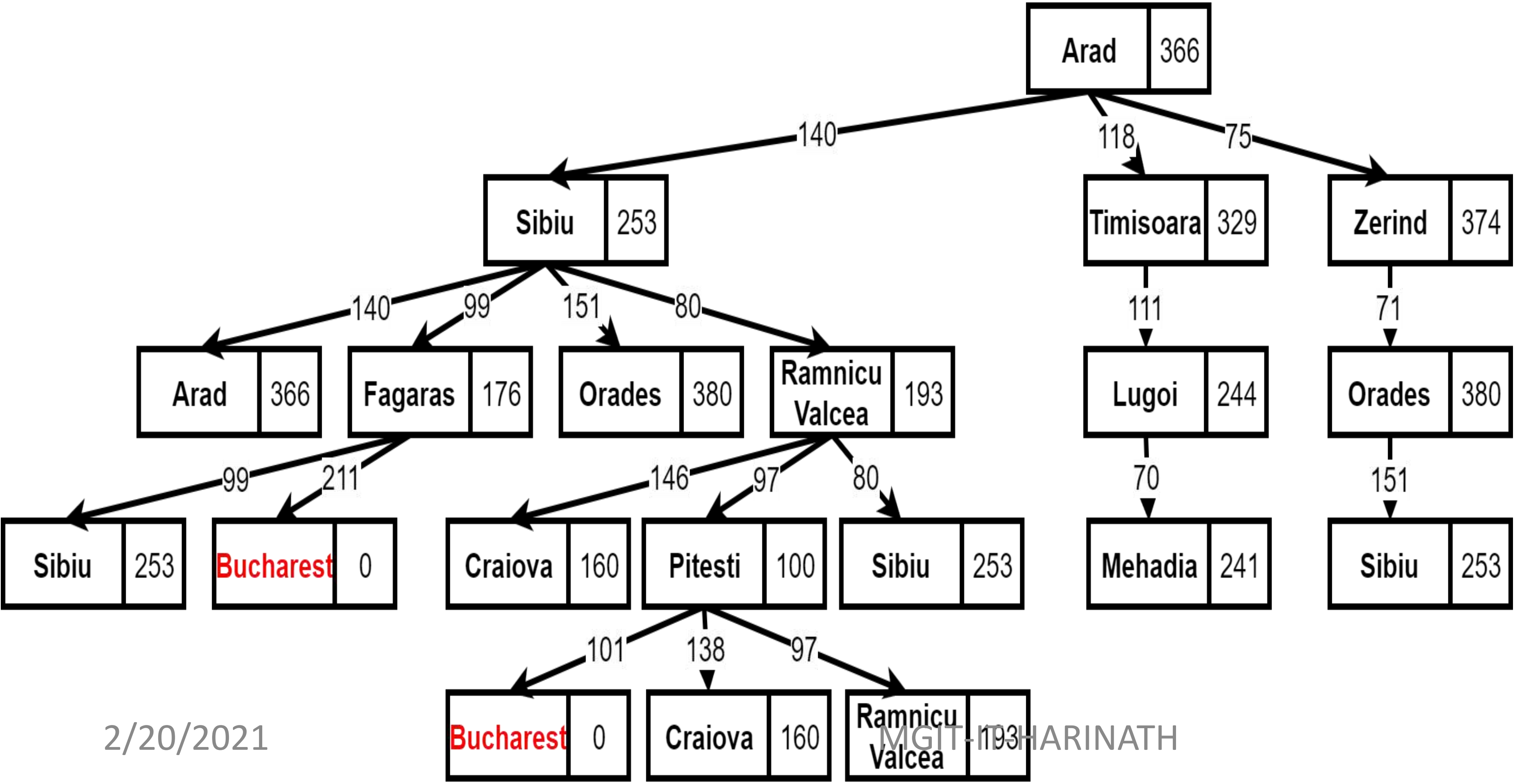
Fagaras



Fagaras Children

Sibiu

Fagaras

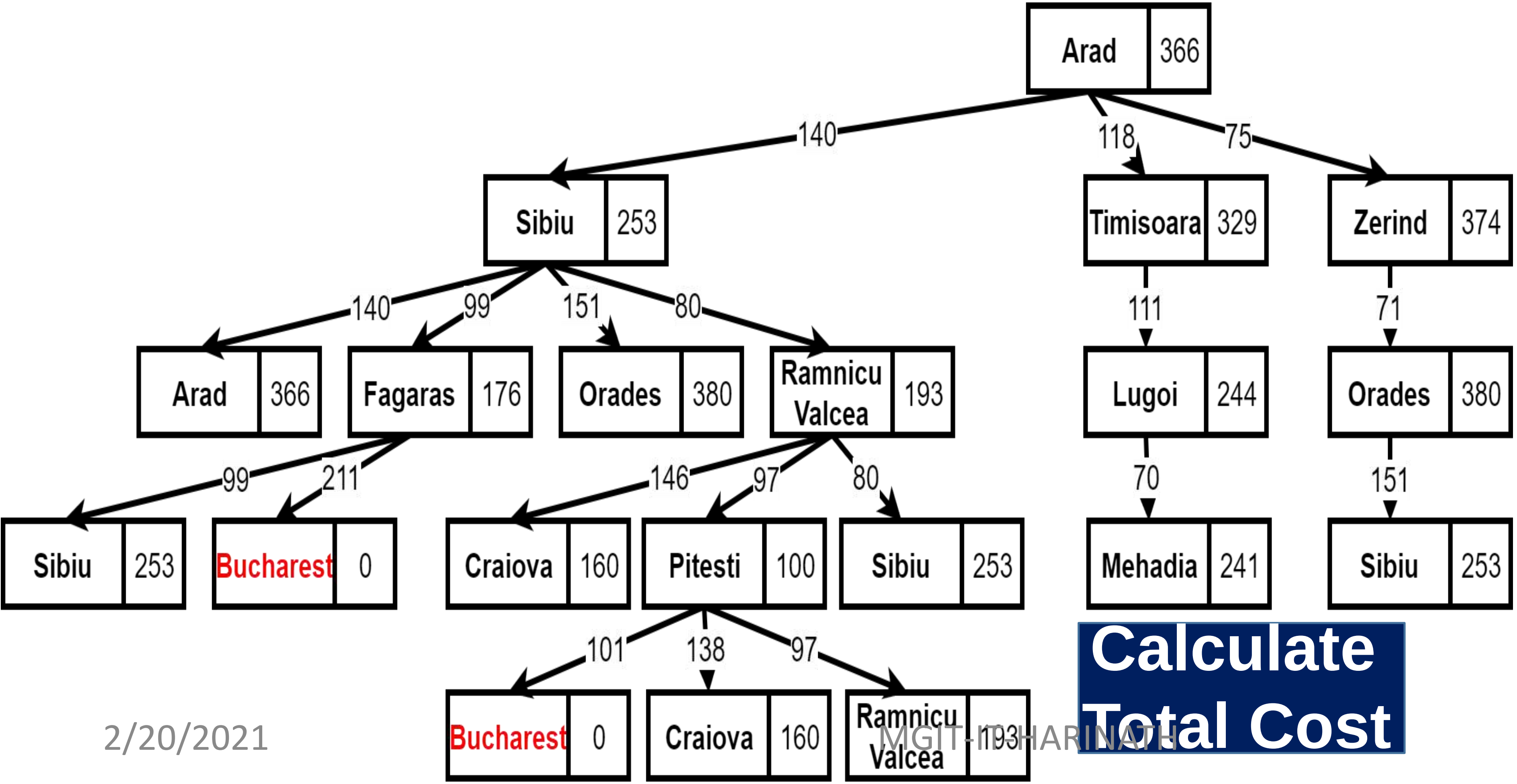


Fagaras Children

Sibiu

Bucharest

Fagaras



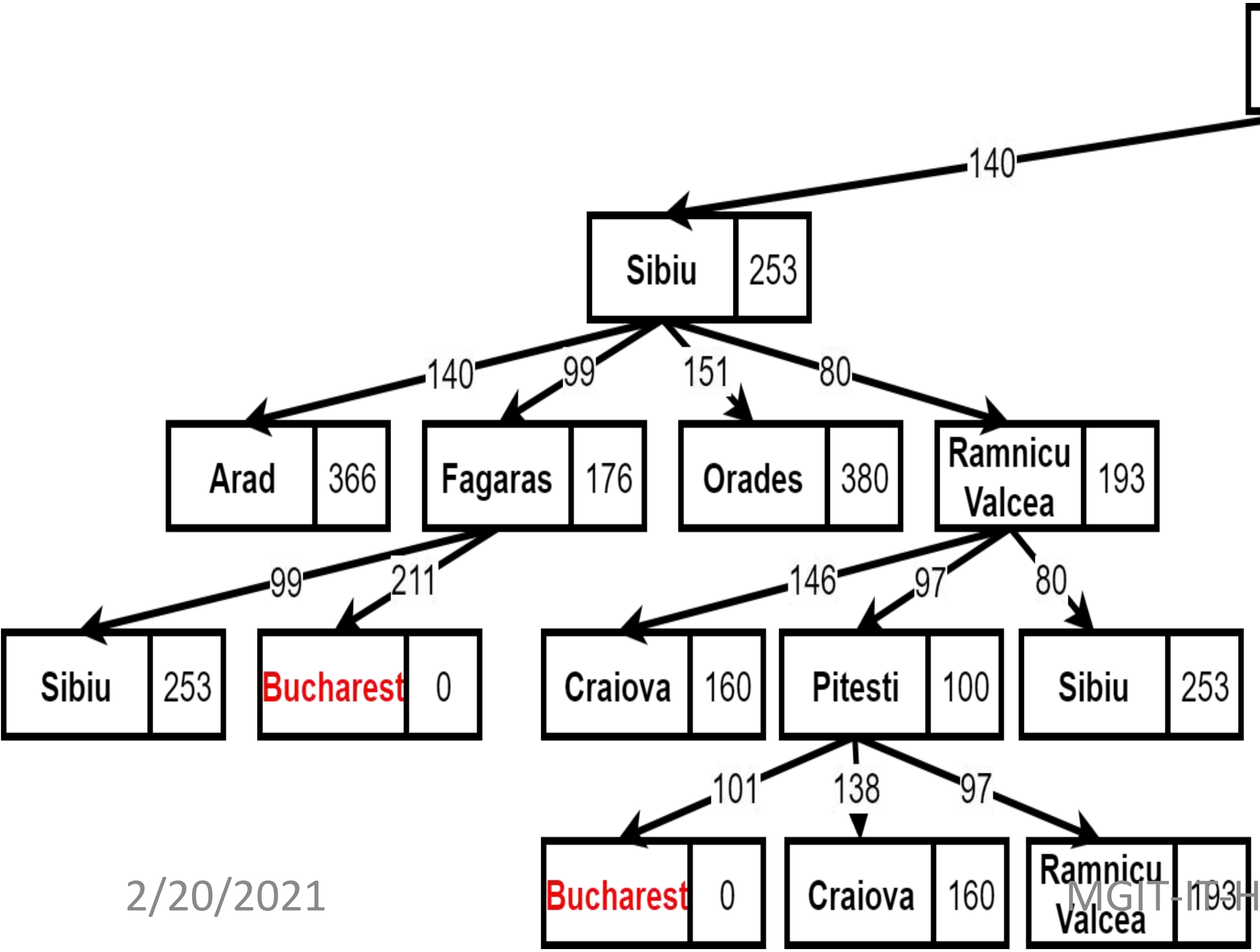
Fagaras Children

Sibiu

Bucharest

Calculate Total Cost

Fagaras



Fagaras Children

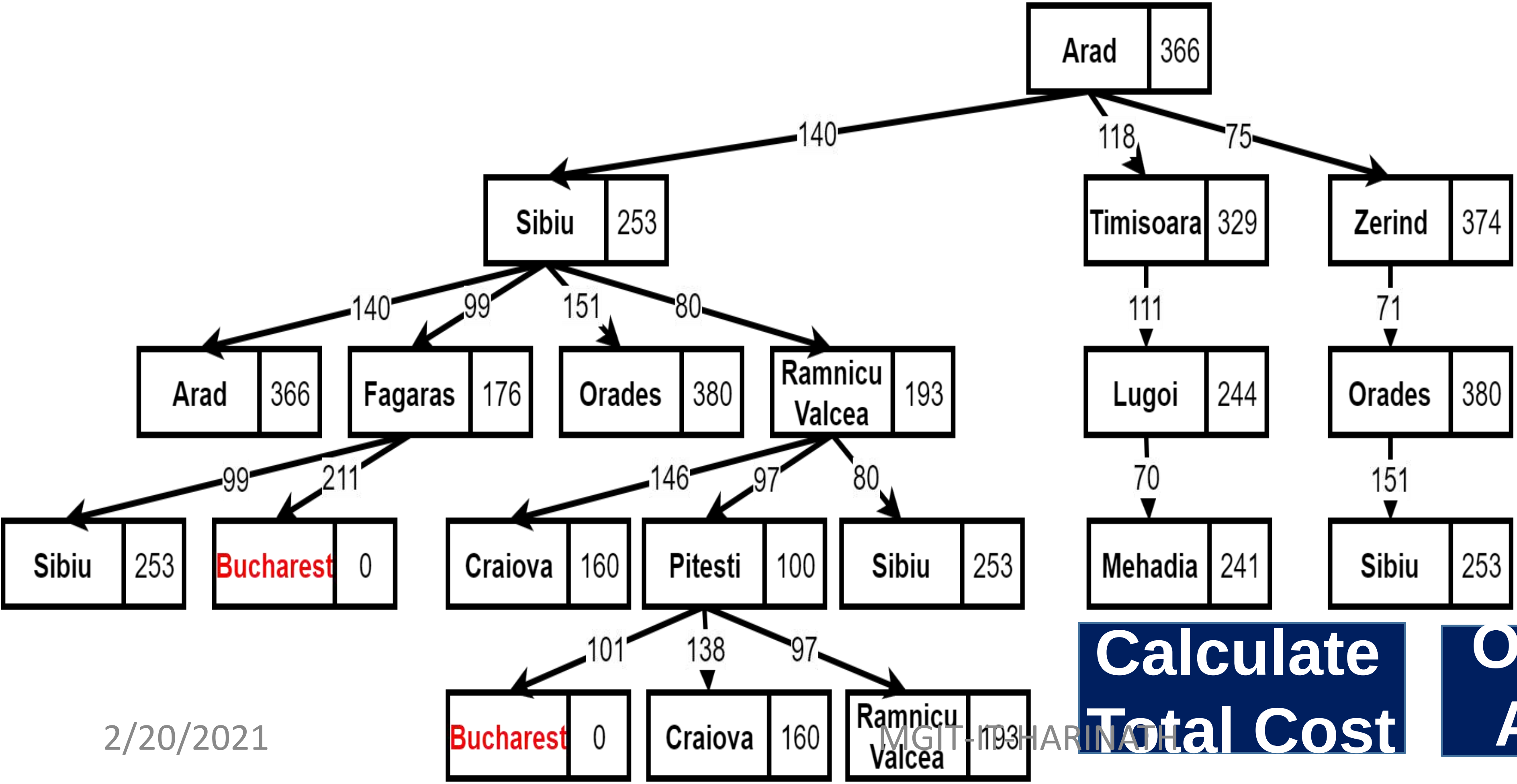
Sibiu

Bucharest

Calculate Total Cost

Order Asc.

Fagaras



Fagaras Children

Sibiu

Bucharest

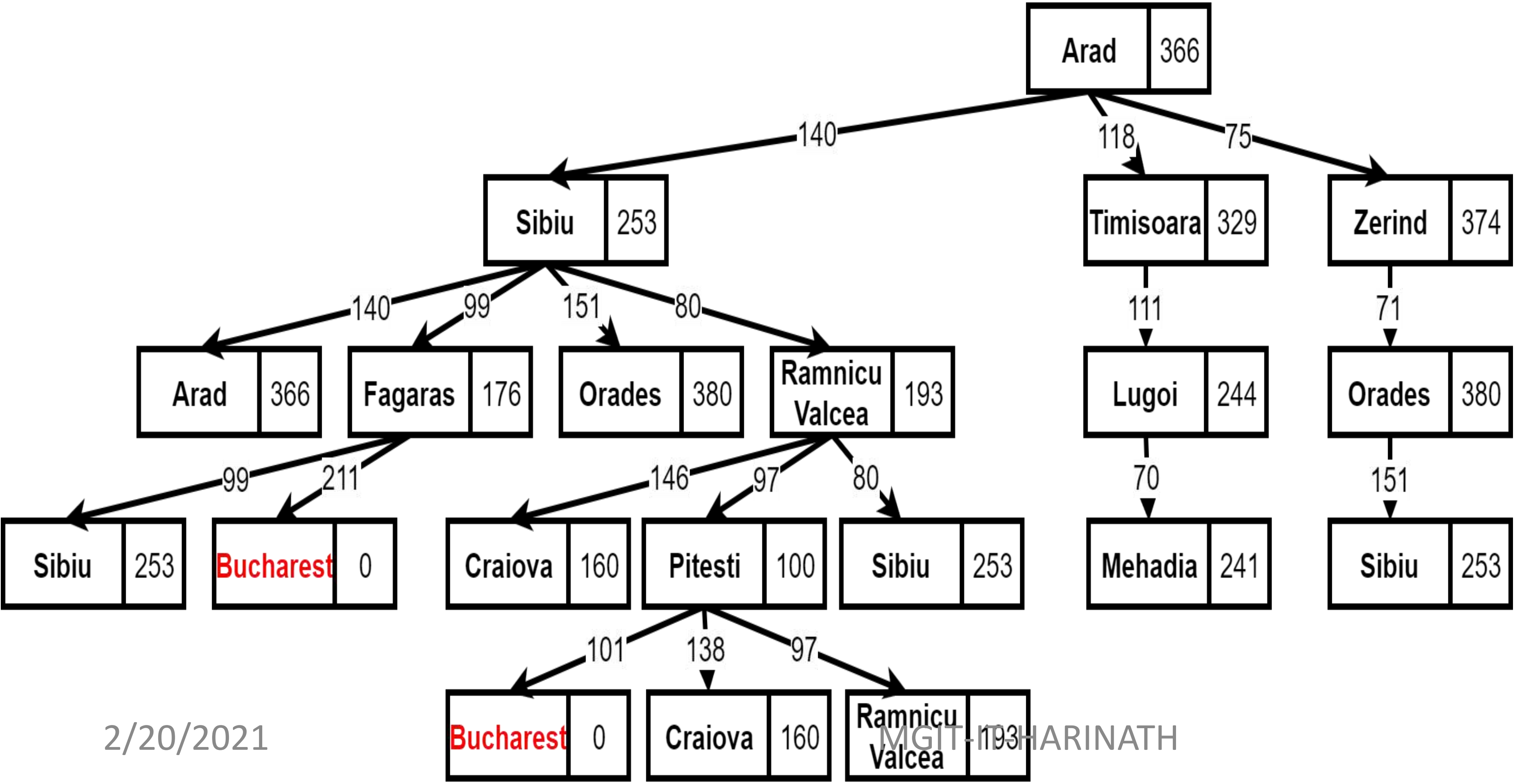
Calculate Total Cost

Order Asc.

Select Min. Cost

Sibiu

Heuristic + **Cost** = **Total**



Fagaras Children

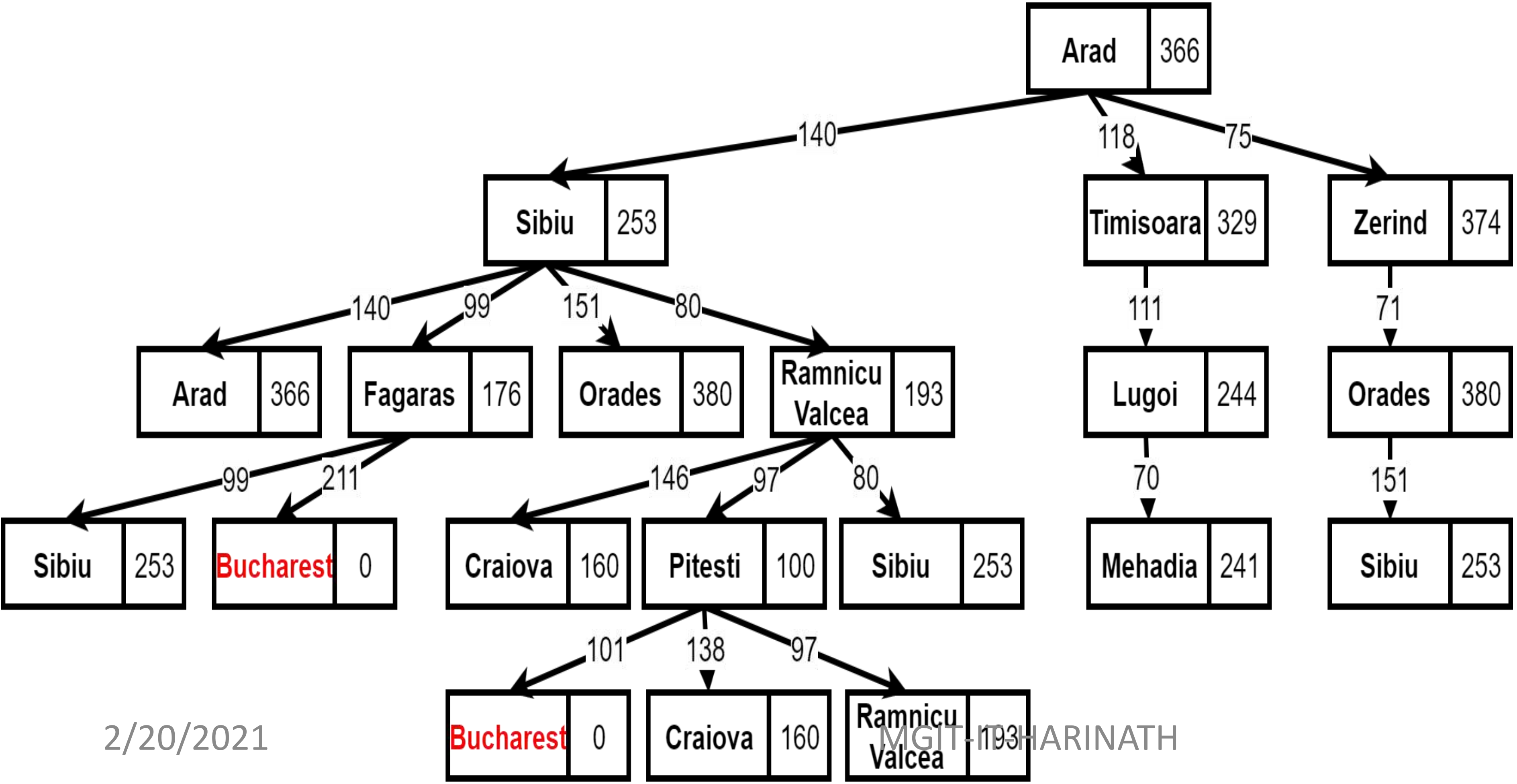
Sibiu

Bucharest

Sibiu

Heuristic + **Cost** = **Total**

253



Fagaras Children

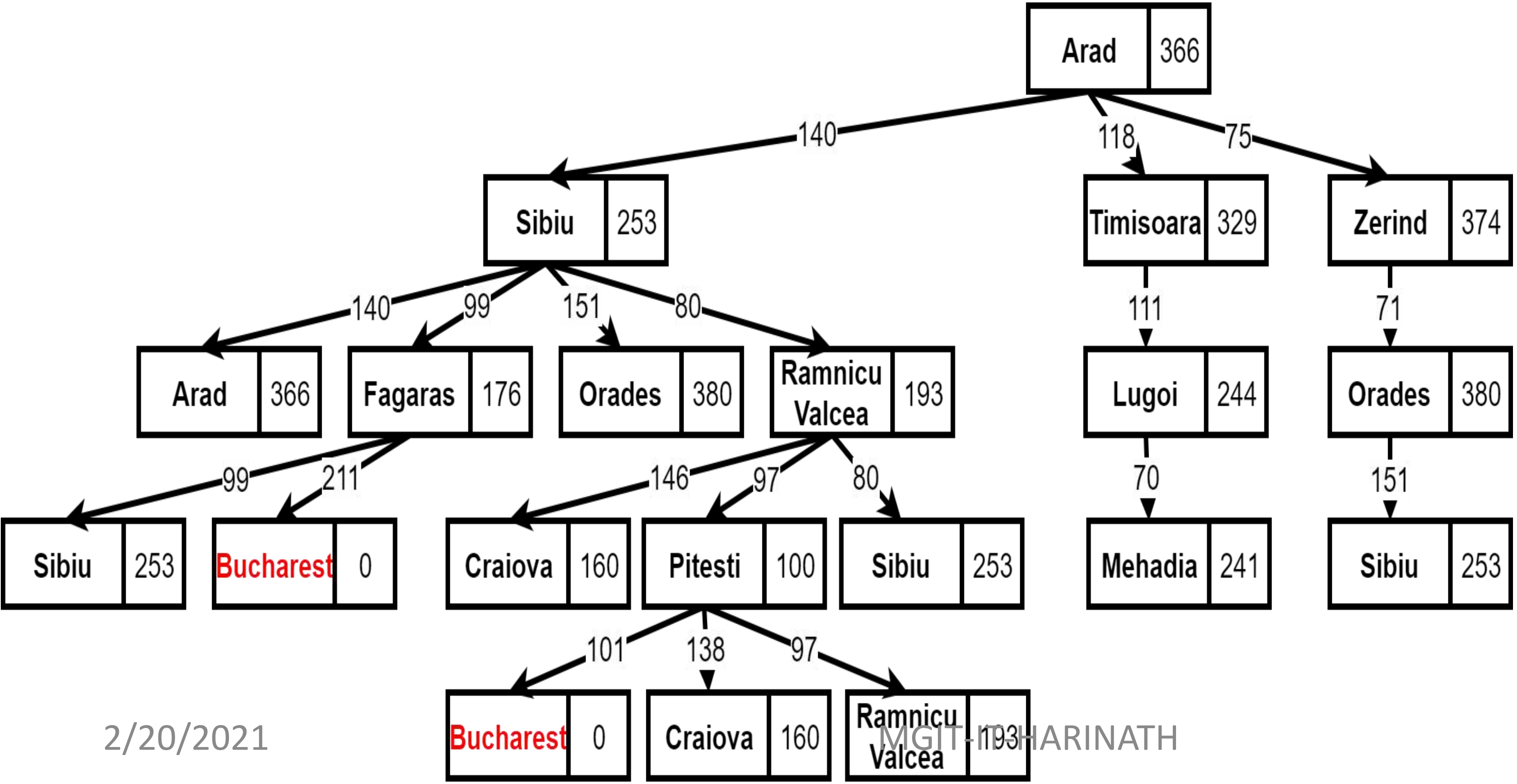
Sibiu

Bucharest

Sibiu

Heuristic + Cost = Total

253 + 338



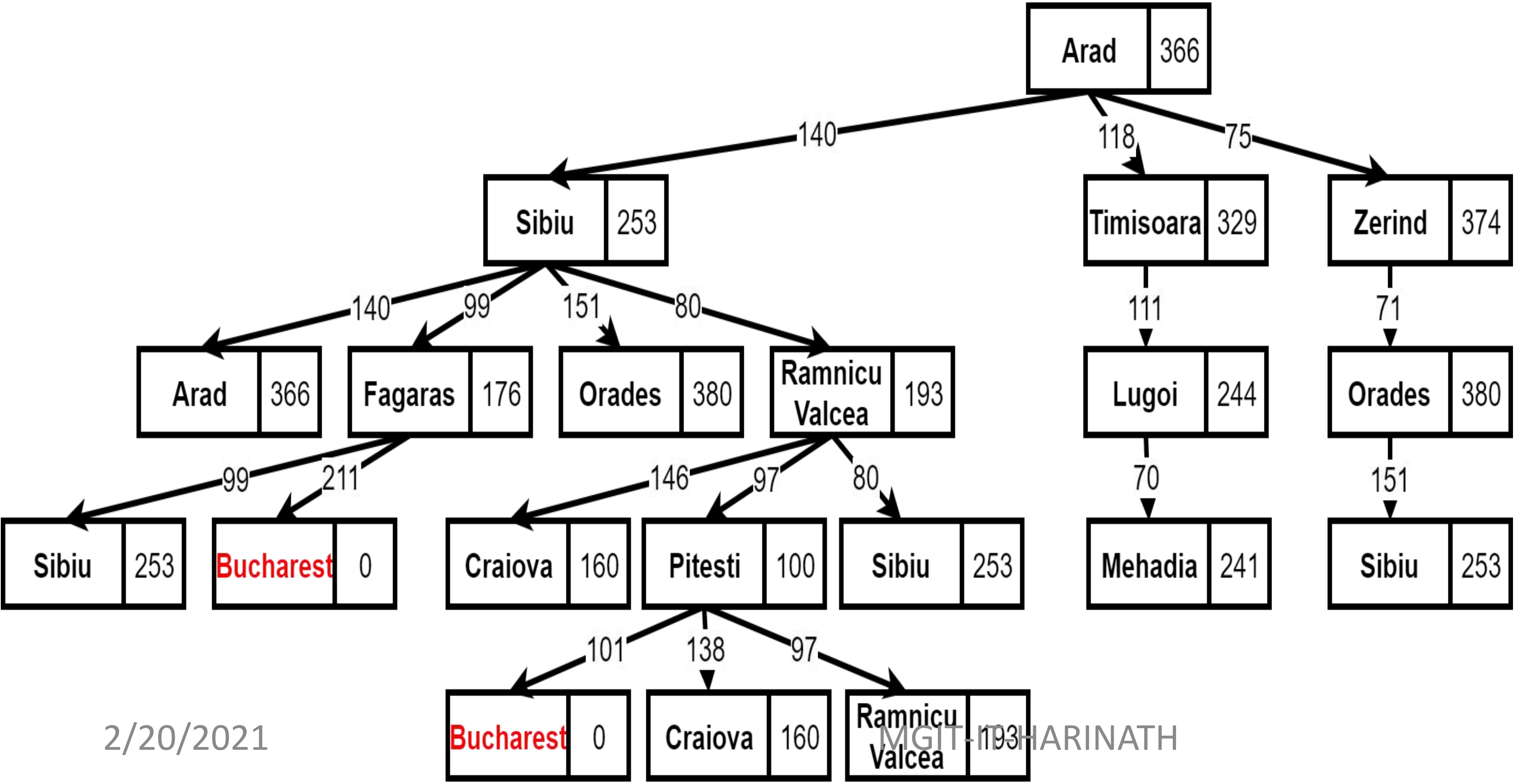
Fagaras Children

Sibiu

Bucharest

Sibiu

Heuristic	+	Cost	=	Total
253	+	338	=	591



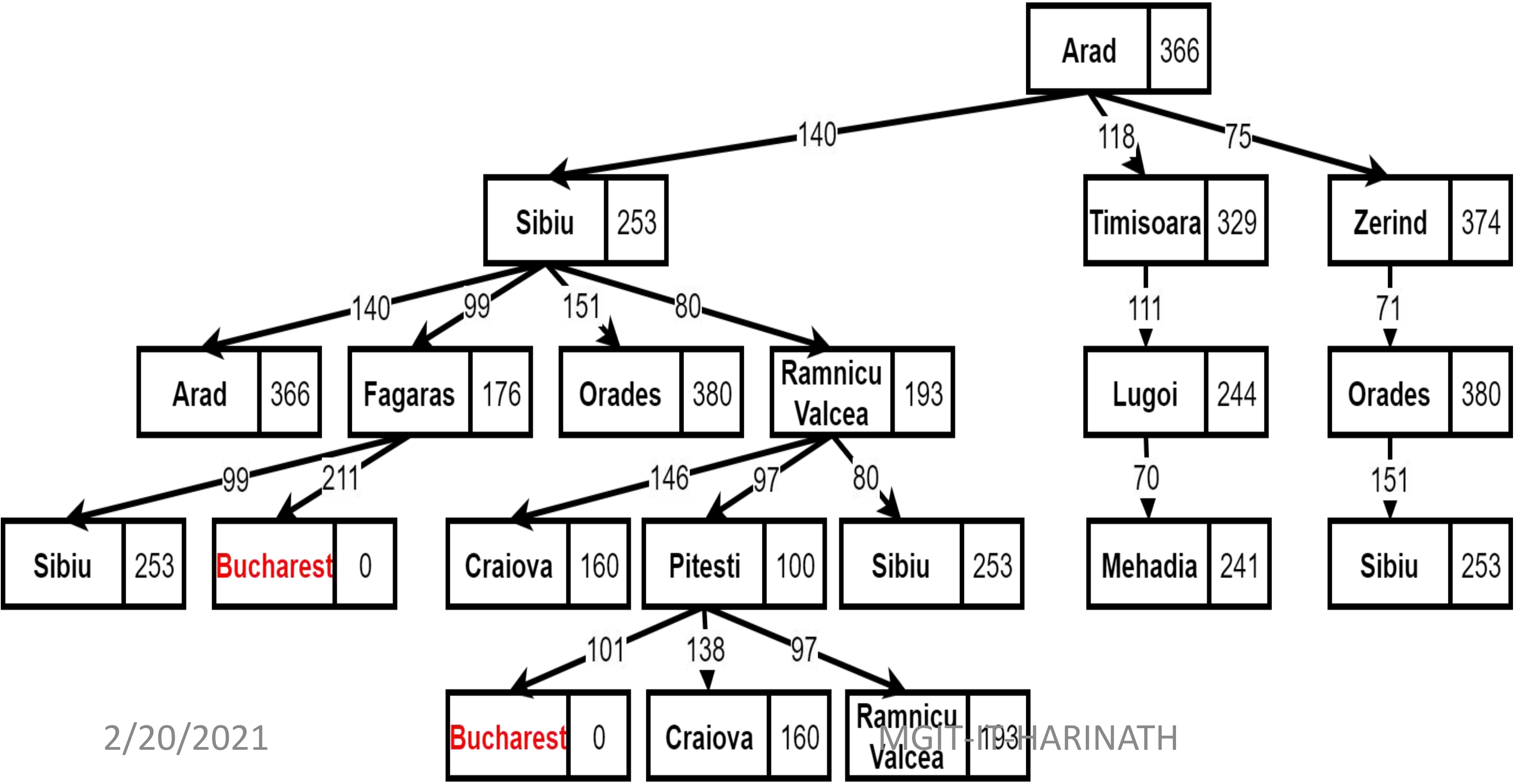
Fagaras Children

Sibiu

Bucharest

Sibiu

Heuristic	+	Cost	=	Total
253	+	338	=	591

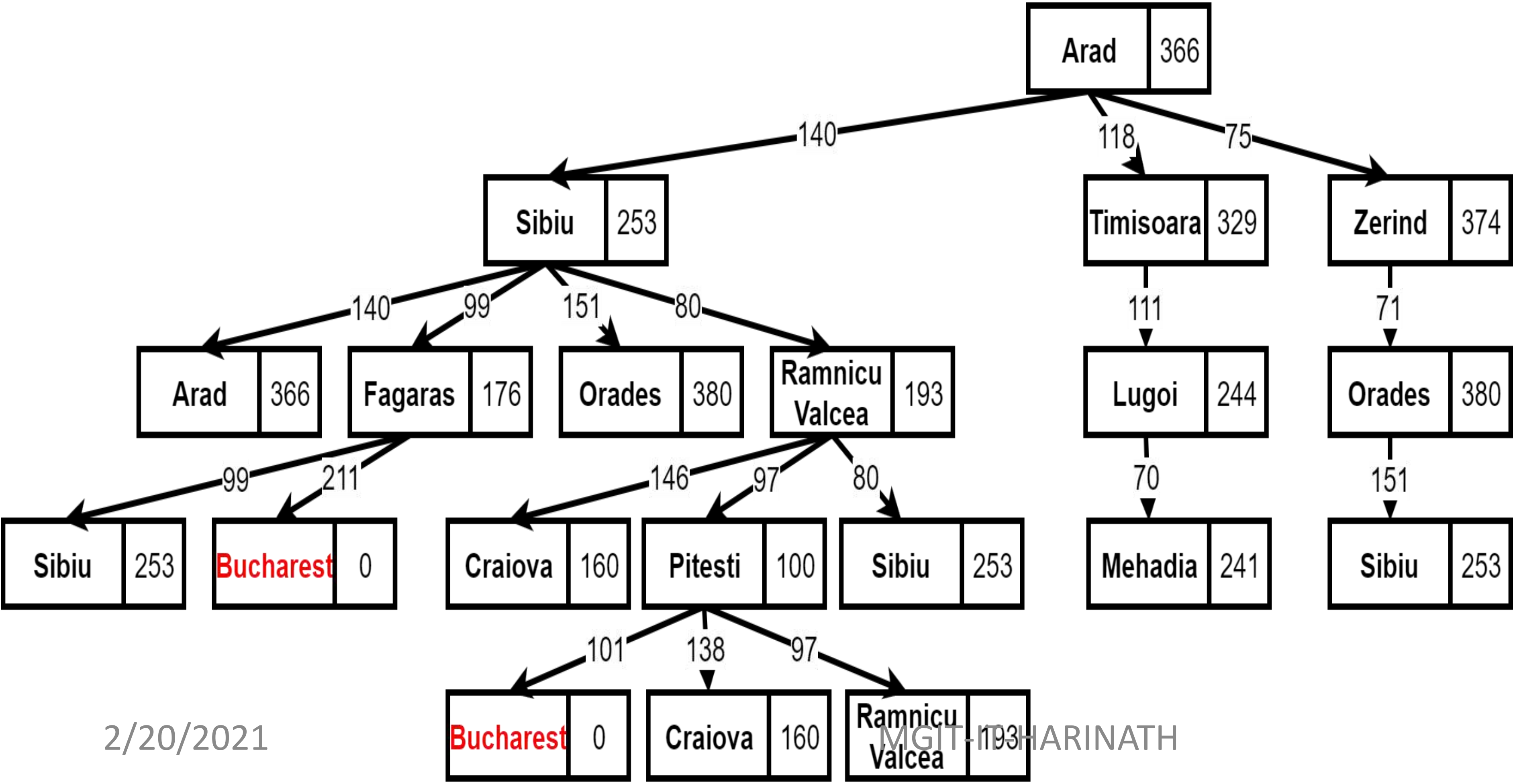


Fagaras Children

Sibiu	591
Bucharest	

Bucharest

Heuristic + Cost = Total



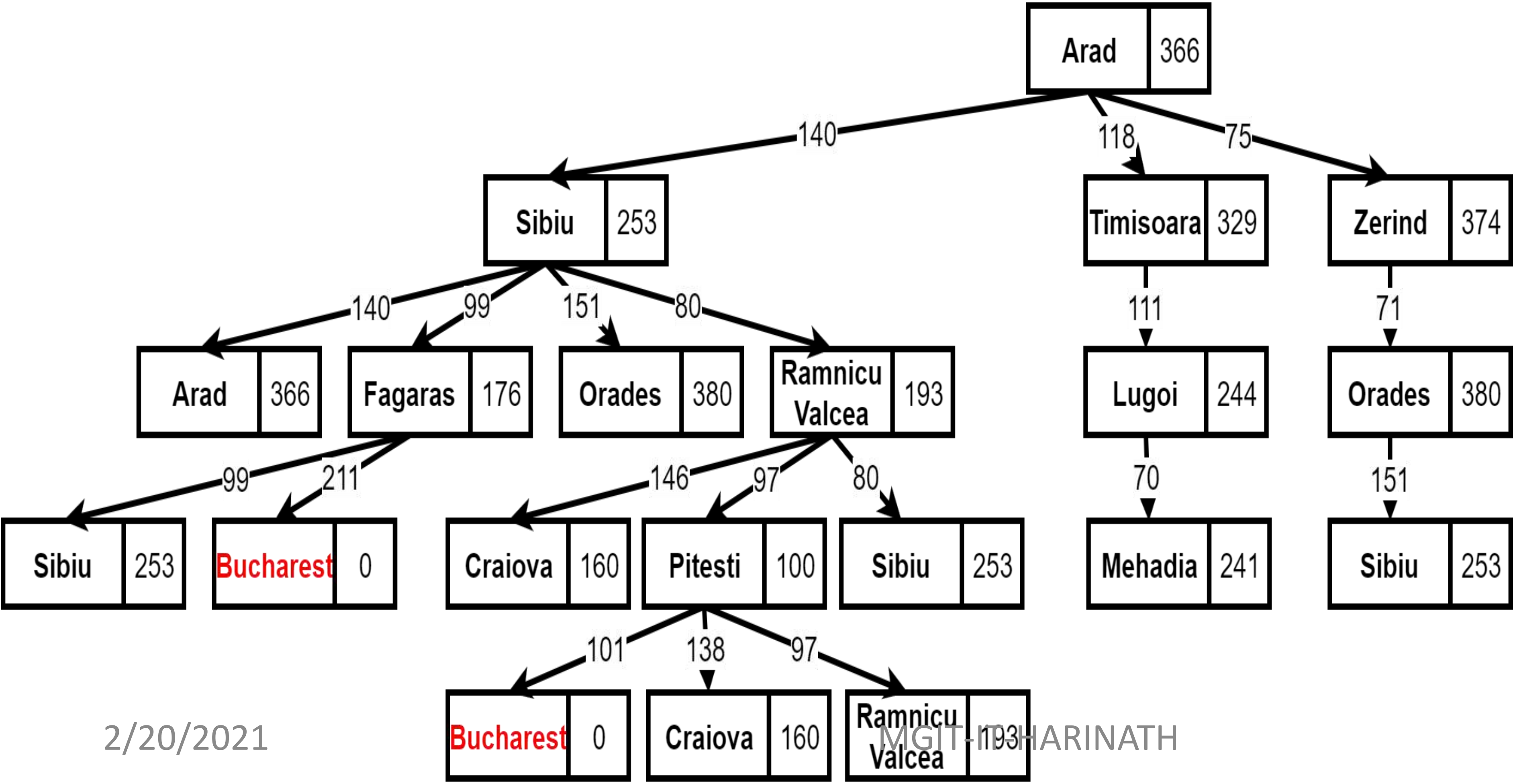
Fagaras Children

Sibiu	591
Bucharest	

Bucharest

Heuristic + Cost = Total

0



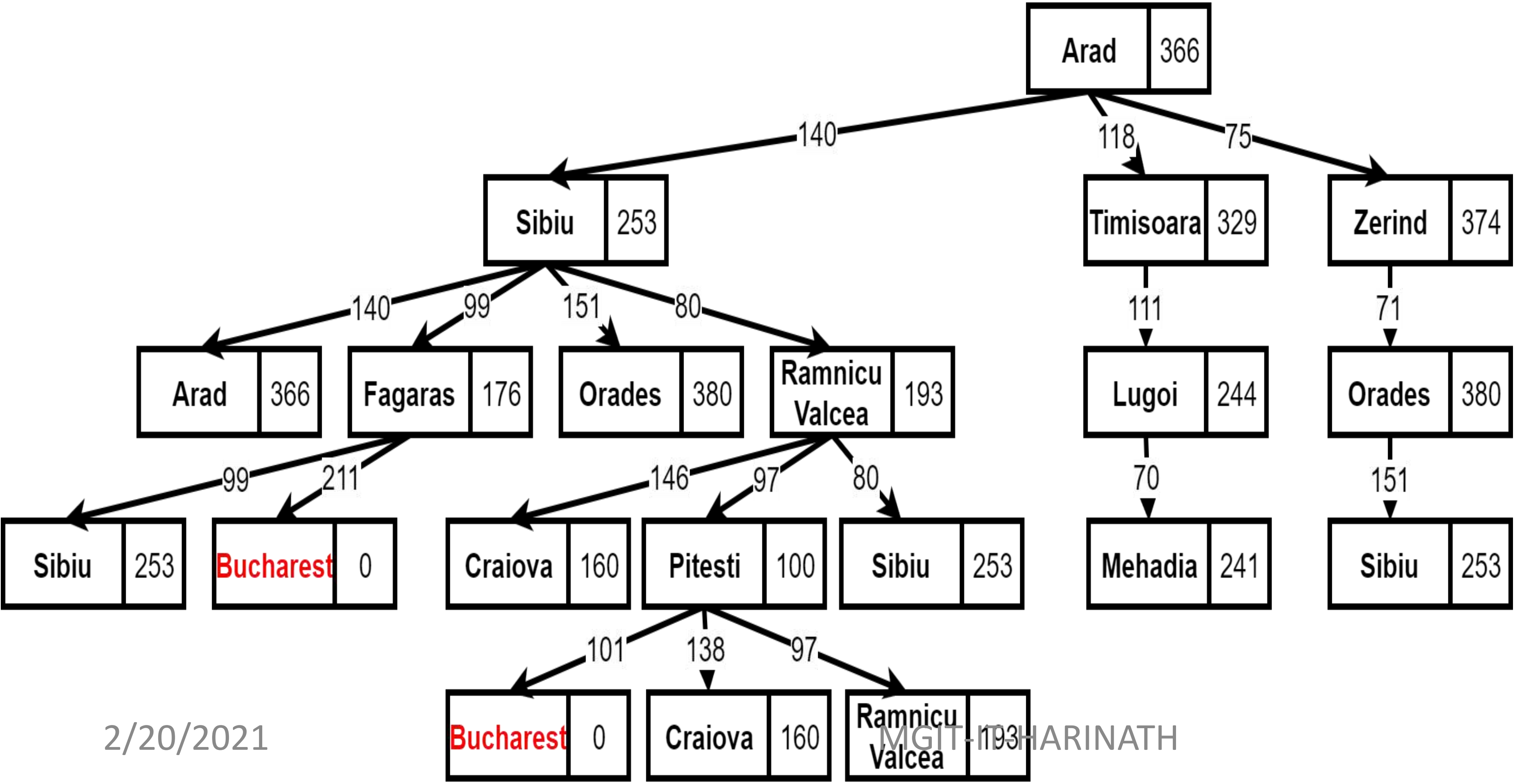
Fagaras Children

Sibiu	591
Bucharest	

Bucharest

Heuristic + Cost = Total

0 + 450



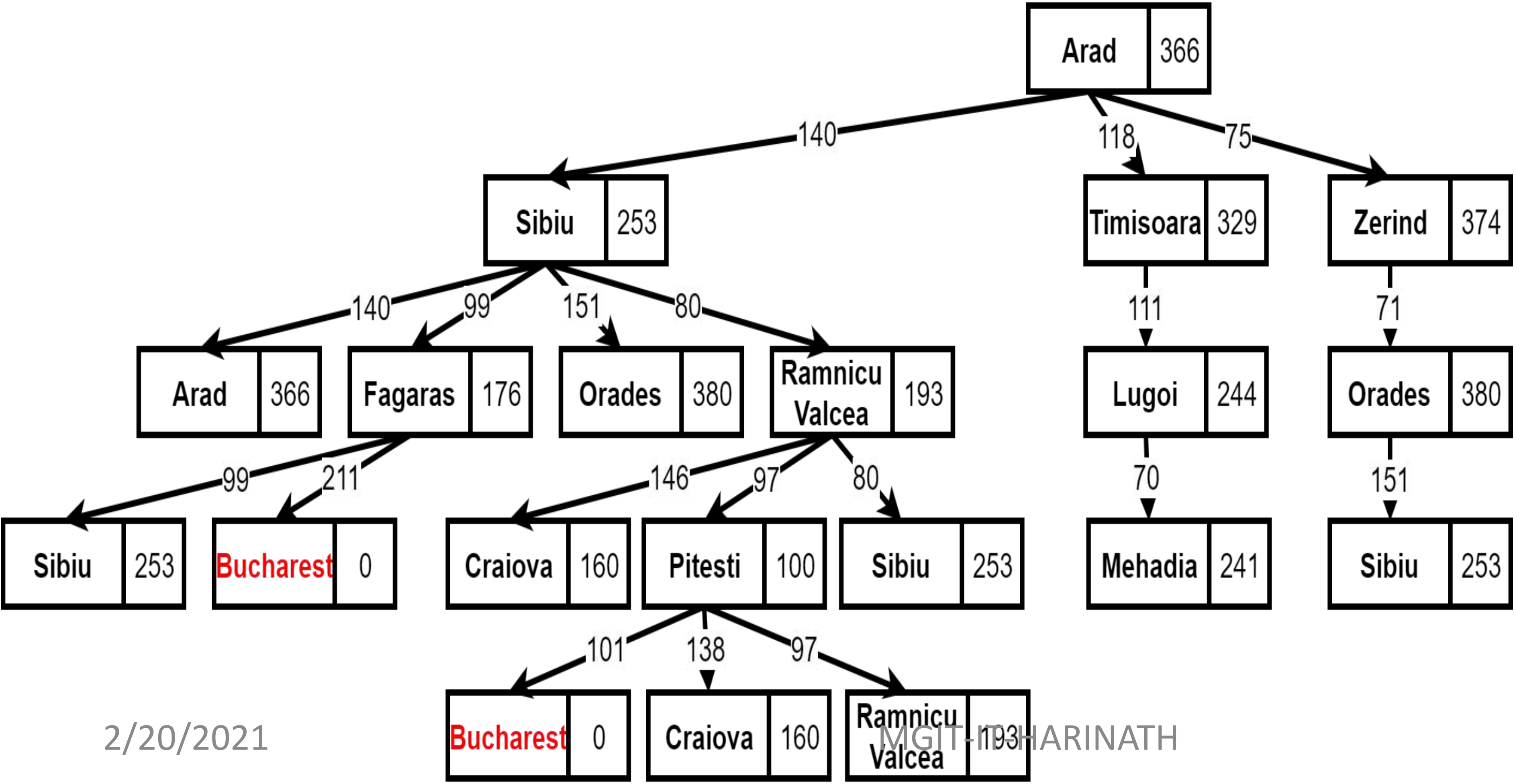
Fagaras Children

Sibiu 591

Bucharest

Bucharest

Heuristic	+	Cost	=	Total
0	+	450	=	450

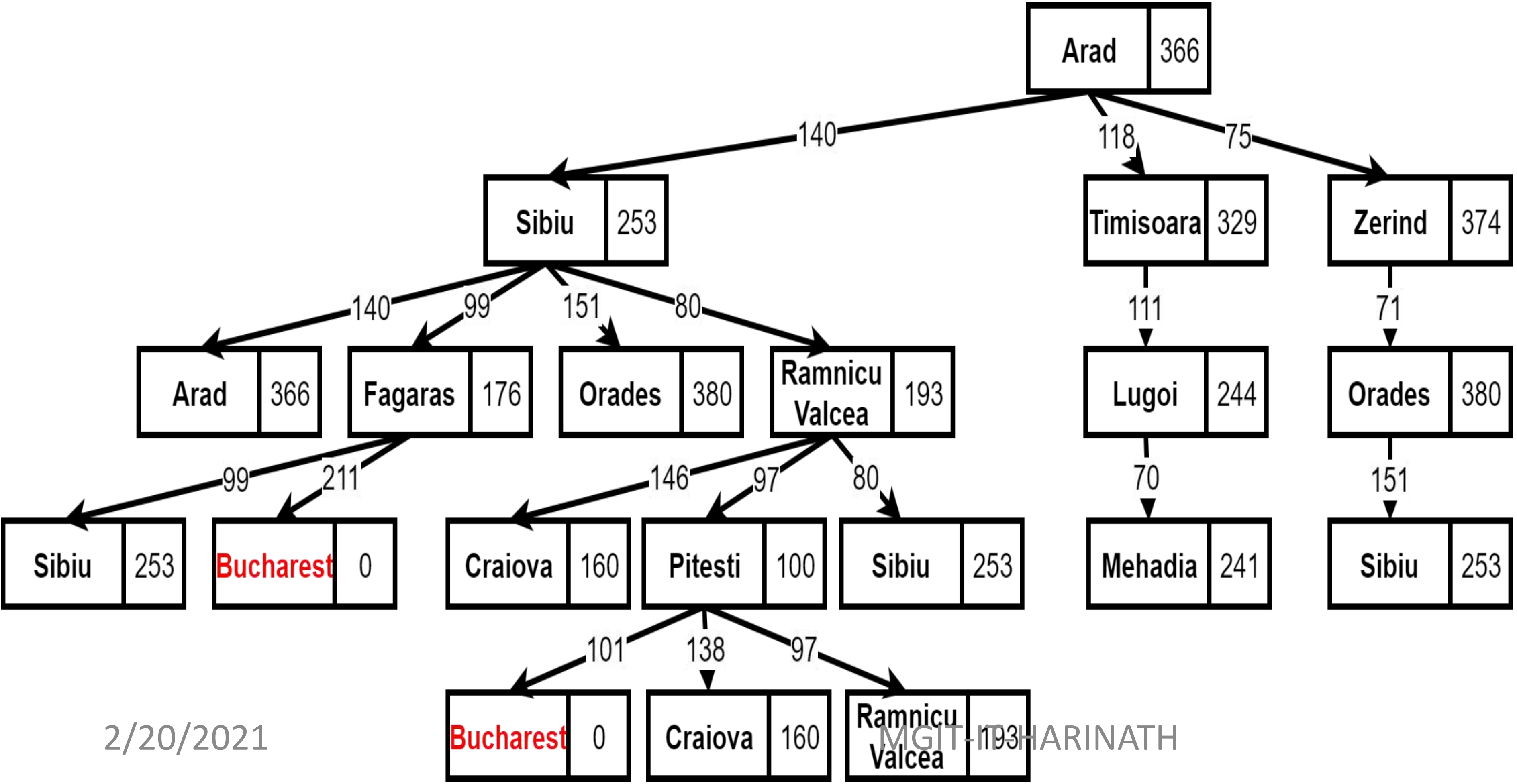


Fagaras Children

Sibiu	591
Bucharest	

Bucharest

Heuristic	+	Cost	=	Total
0	+	450	=	450



Fagaras Children

Sibiu	591
Bucharest	450

Current Queue

Pitesti	Timisoar	Zerin	Craiova	Sibiu	Arad	Orade
417	447	449	526	553	646	671

Current Queue

Pitesti	Timisoar	Zerin	Craiova	Sibiu	Arad	Orade
417	447	449	526	553	646	671

Sibiu	Buchare
591	450

Current Queue

Pitesti	Timisoar	Zerin	Craiova	Sibiu	Buchare
417	447	449	526	553	450

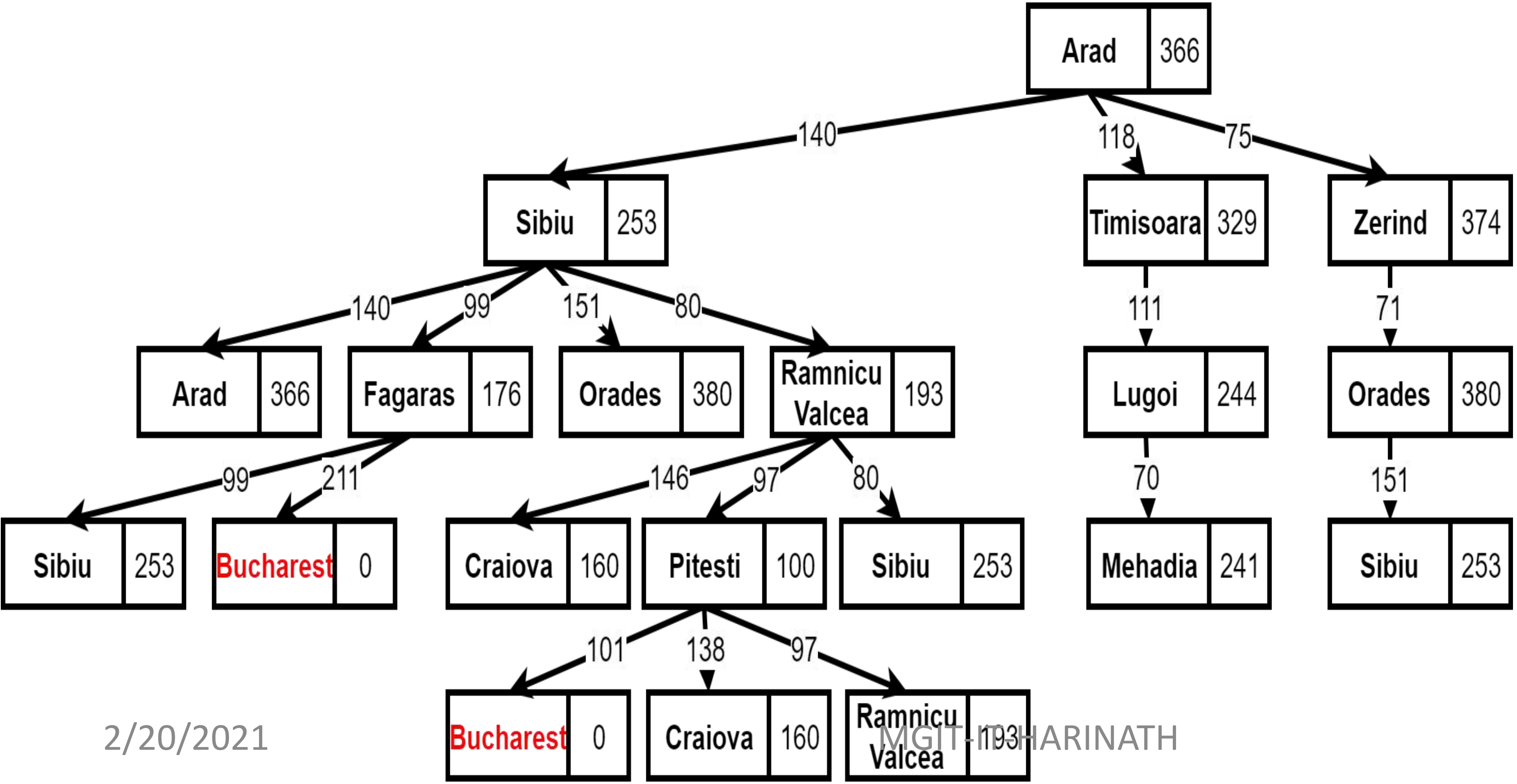
Sibiu	Arad	Orade
591	646	671

Current Queue

Pitesti	Timisoar	Zerin	Craiova	Sibiu	Buchare
417	447	449	526	553	450

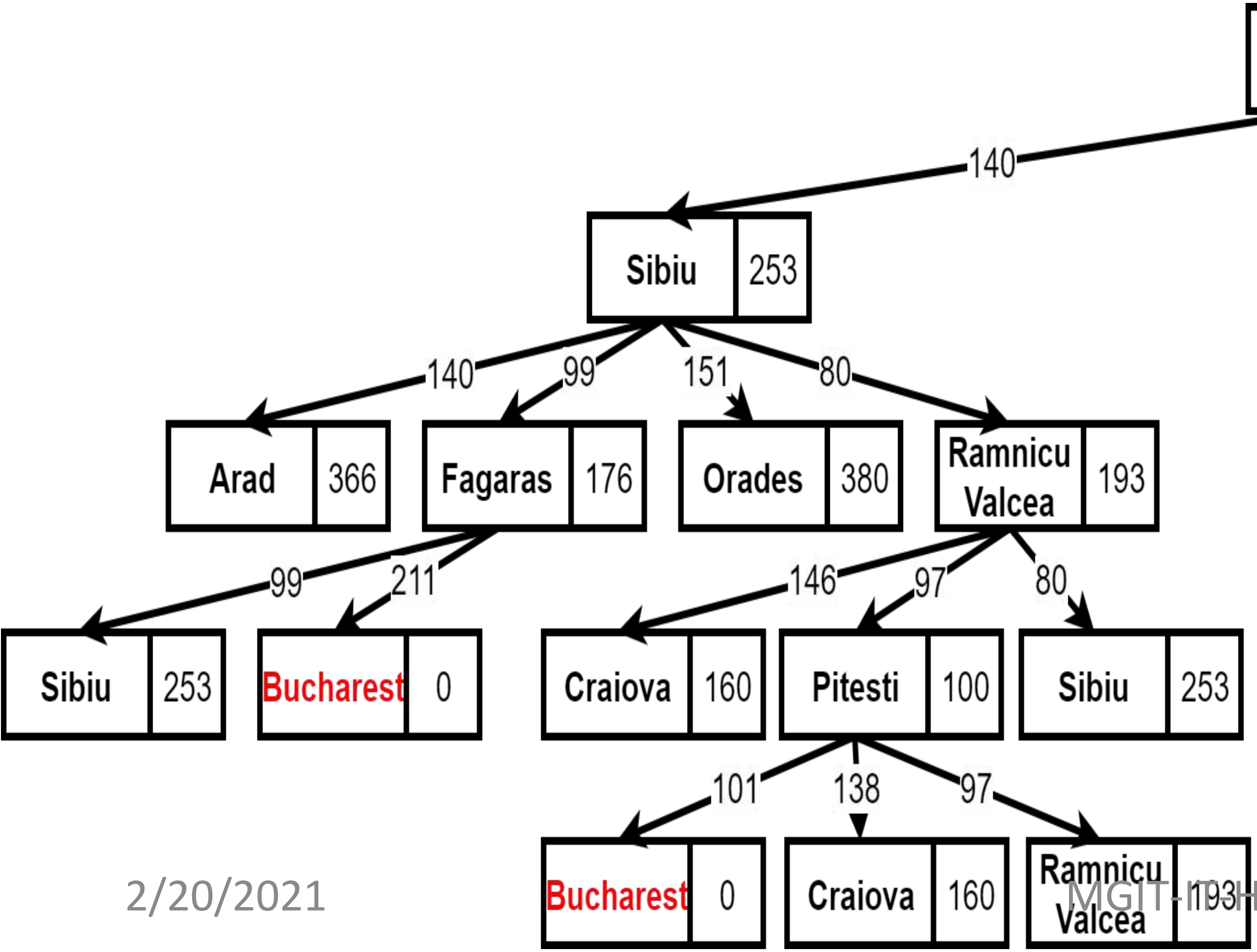
Sibiu	Arad	Orade
591	646	671

Pitesti

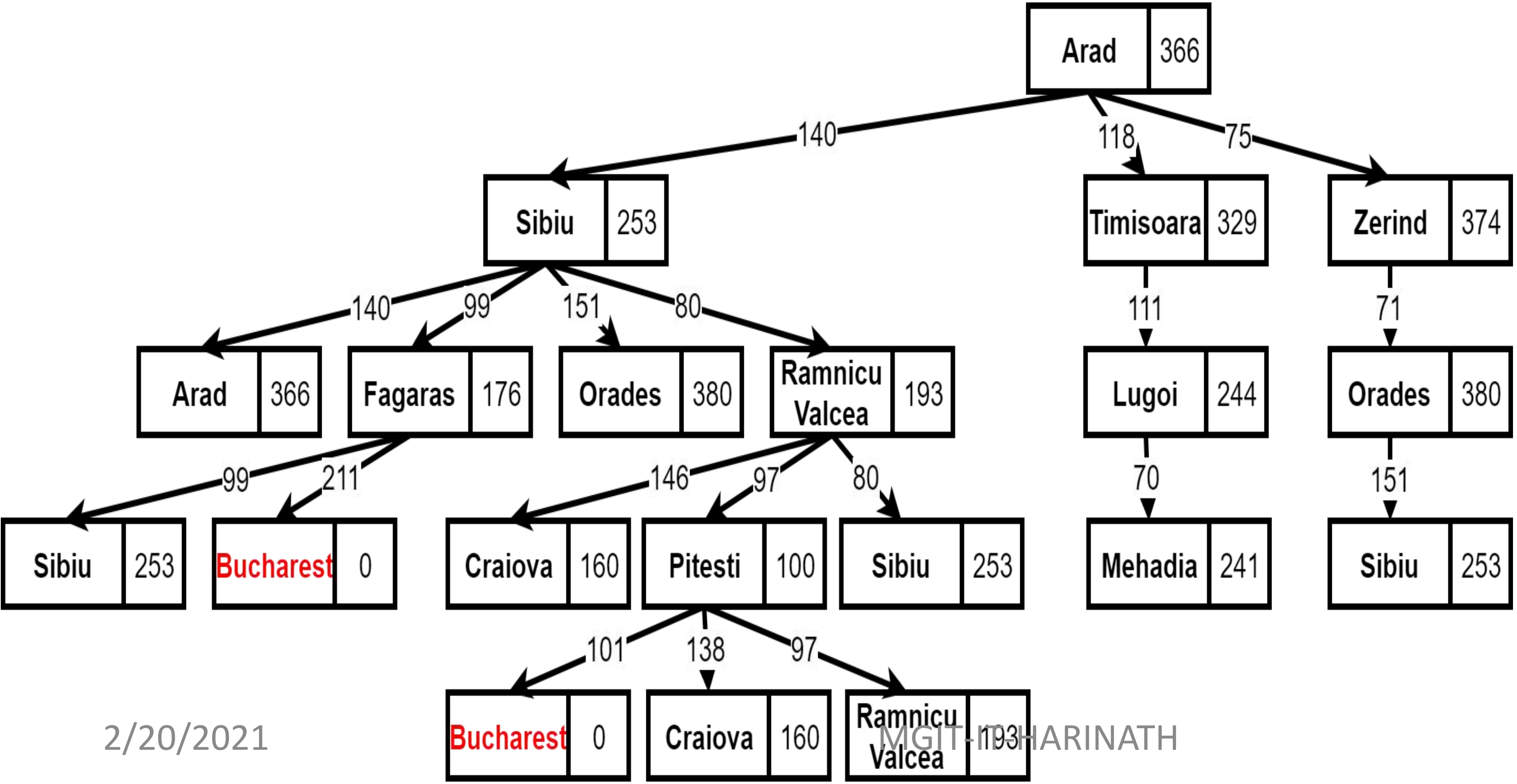


Arad 366
140
118
75
Sibiu 253
Timisoara 329
Zerind 374
140
99
151
80
111
71
Arad 366
Fagaras 176
Orades 380
Ramnicu Valcea 193
Lugoi 244
Orades 380
99
211
146
97
80
70
151
Sibiu 253
Bucharest 0
Craiova 160
Pitesti 100
Sibiu 253
Mehadia 241
Sibiu 253
101
138
97
Bucharest 0
Craiova 160
Ramnicu Valcea 193

Pitesti


Arad 366
140
118
75
Sibiu 253
Timisoara 329
Zerind 374
140
99
151
80
111
71
Arad 366
Fagaras 176
Orades 380
Ramnicu Valcea 193
Lugoi 244
Orades 380
99
211
146
97
80
70
151
Sibiu 253
Bucharest 0
Craiova 160
Pitesti 100
Sibiu 253
Mehadia 241
Sibiu 253
101
138
97
Bucharest 0
Craiova 160
Ramnicu Valcea 193

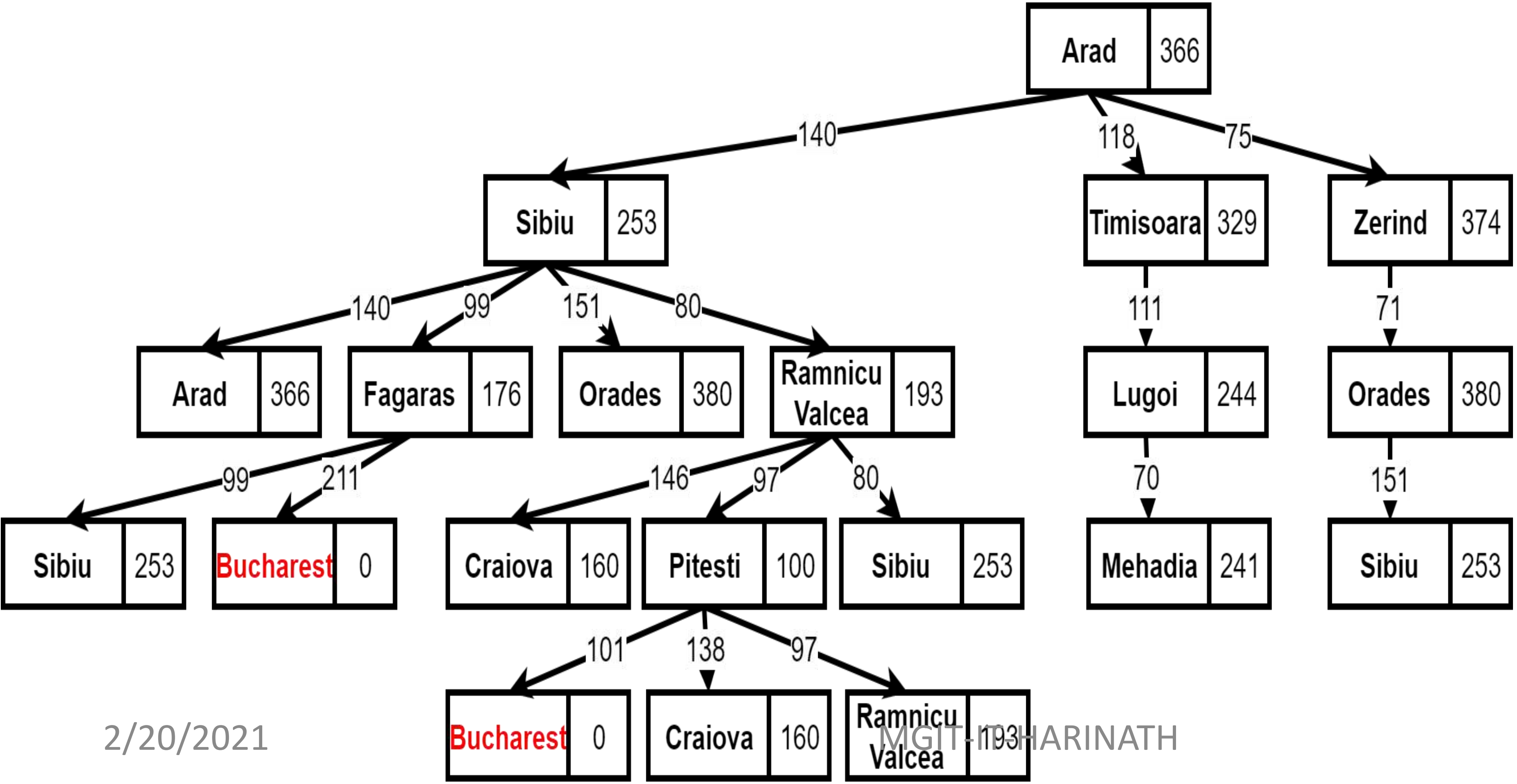
Pitesti



Pitesti Children

Bucharest

Pitesti

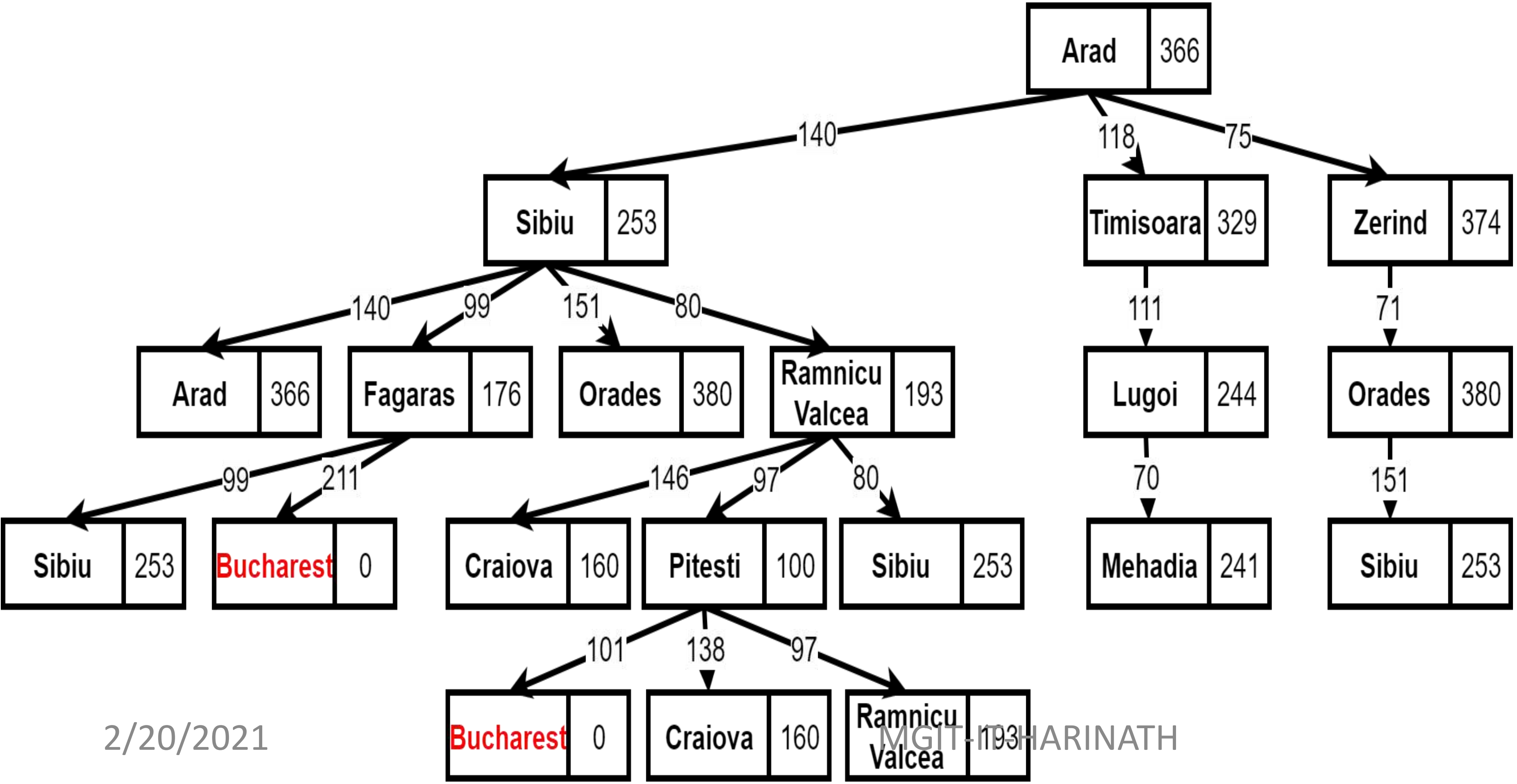


Pitesti Children

Bucharest

Craiova

Pitesti



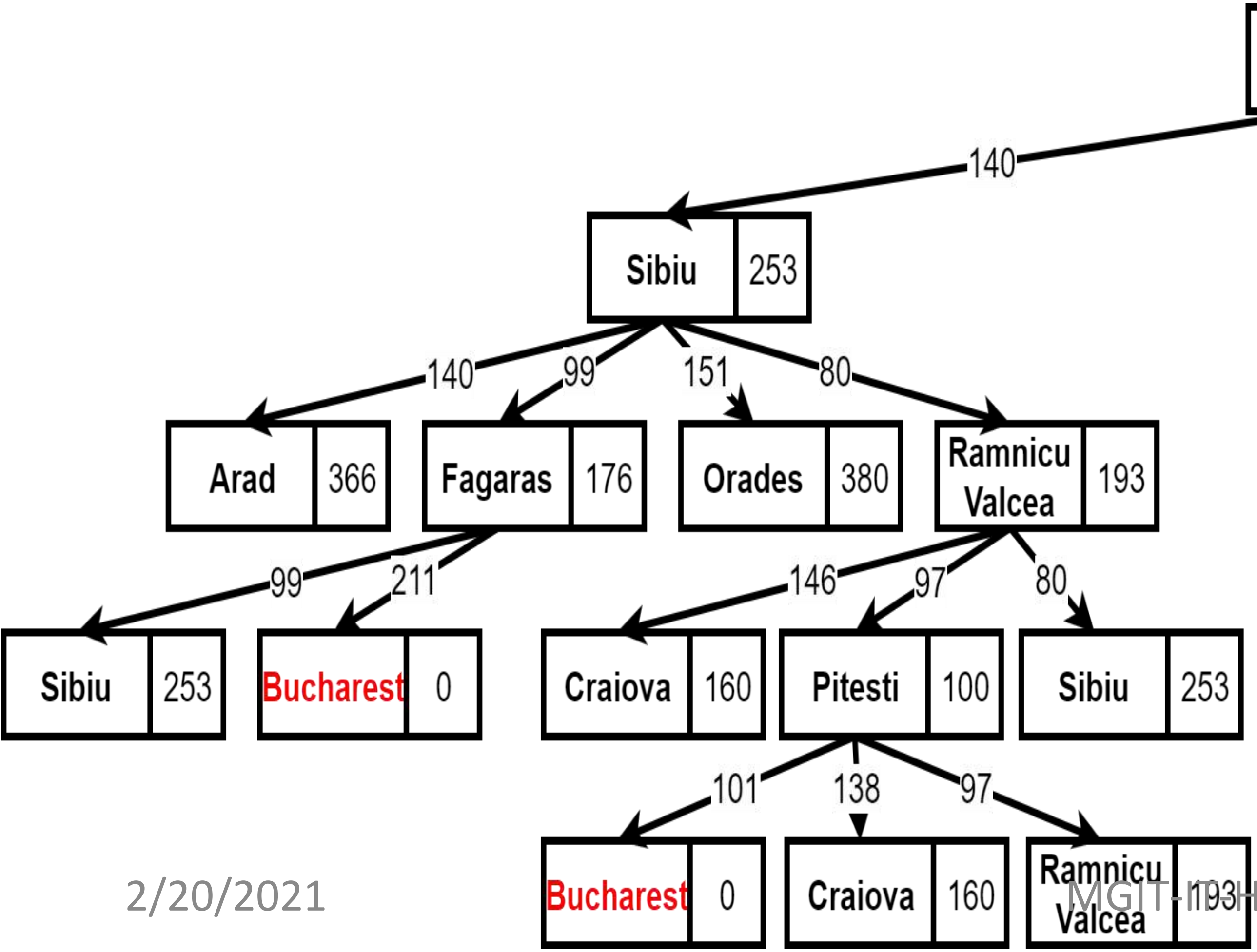
Pitesti Children

Bucharest

Craiova

Ramnicu

Pitesti

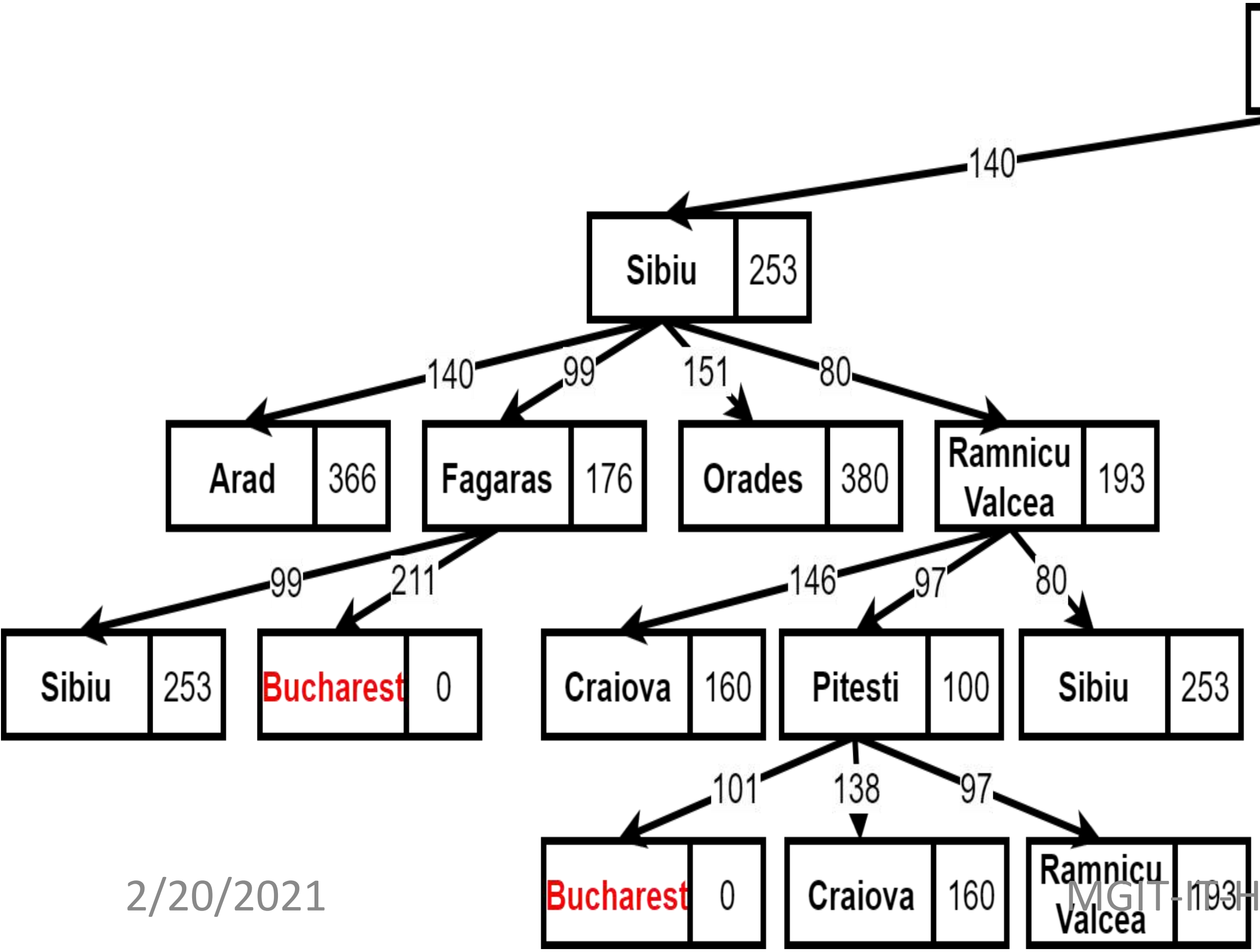


Pitesti Children

- Bucharest
- Craiova
- Ramnicu

Calculate Total Cost

Pitesti



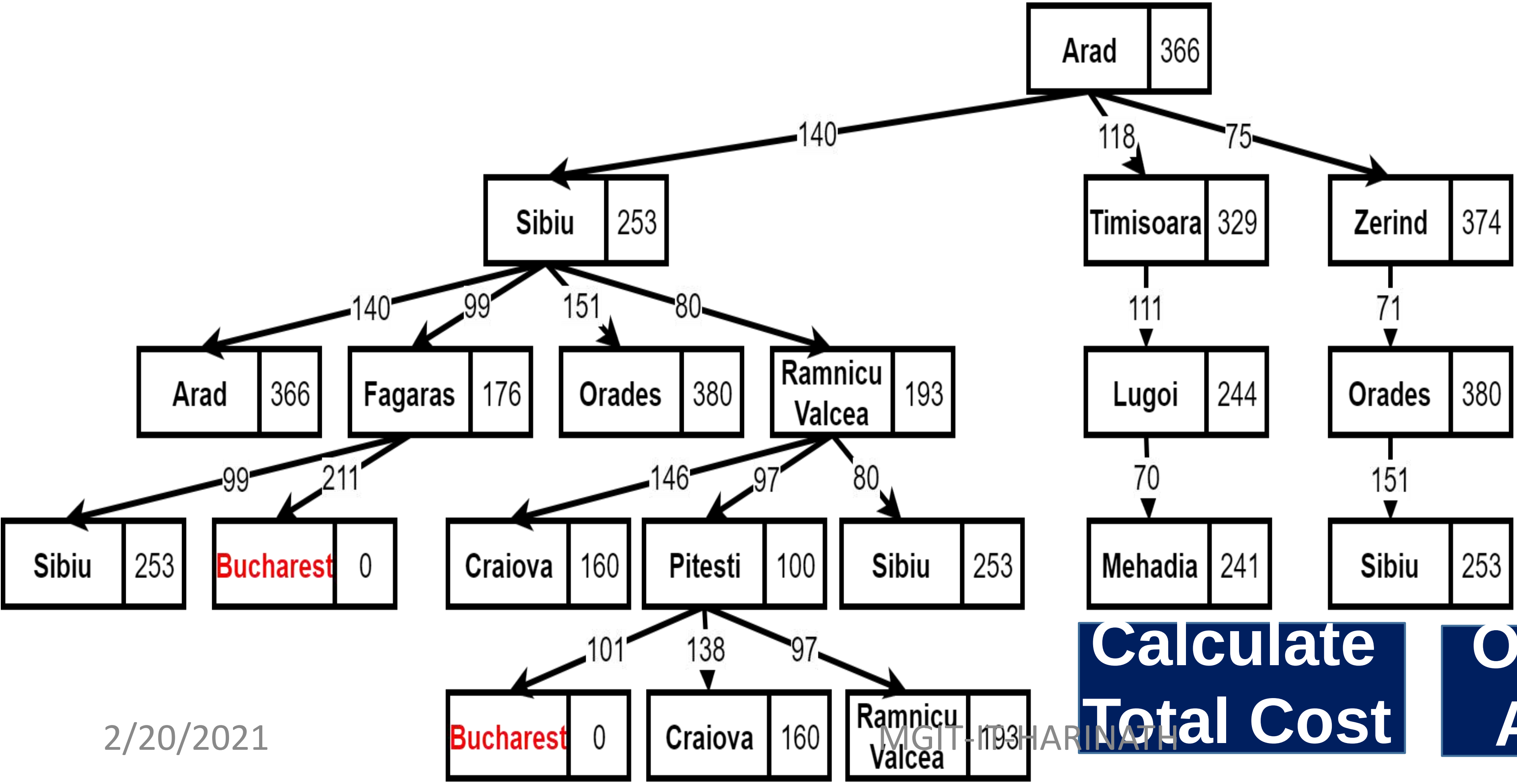
Pitesti Children

- Bucharest
- Craiova
- Ramnicu

Calculate Total Cost

Order Asc.

Pitesti



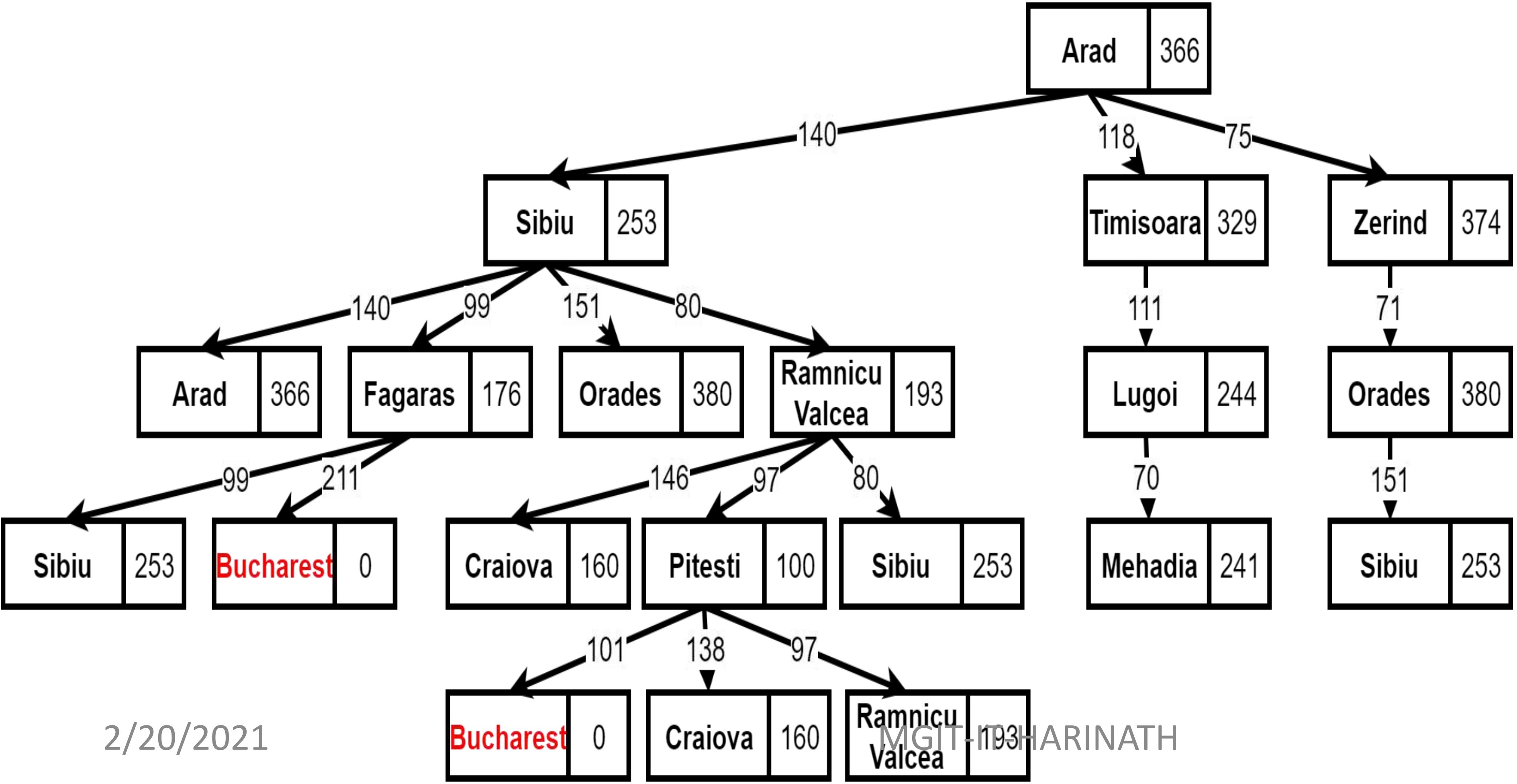
Pitesti Children

- Bucharest
- Craiova
- Ramnicu

Calculate Total Cost | **Order Asc.** | **Select Min. Cost**

Bucharest

Heuristic + Cost = Total



Ramnicu Children

Bucharest

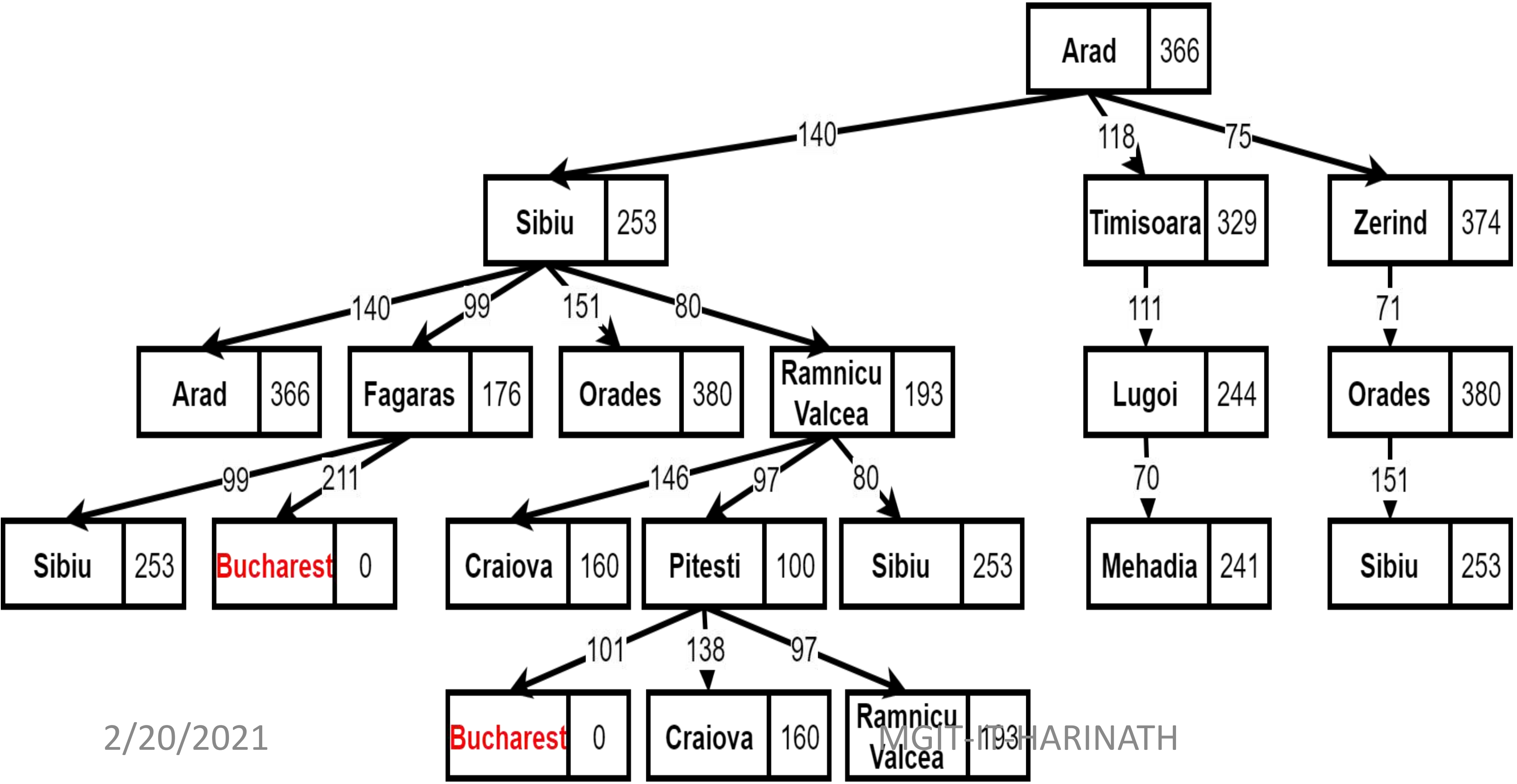
Craiova

Ramnicu

Bucharest

Heuristic + Cost = Total

0



Ramnicu Children

Bucharest

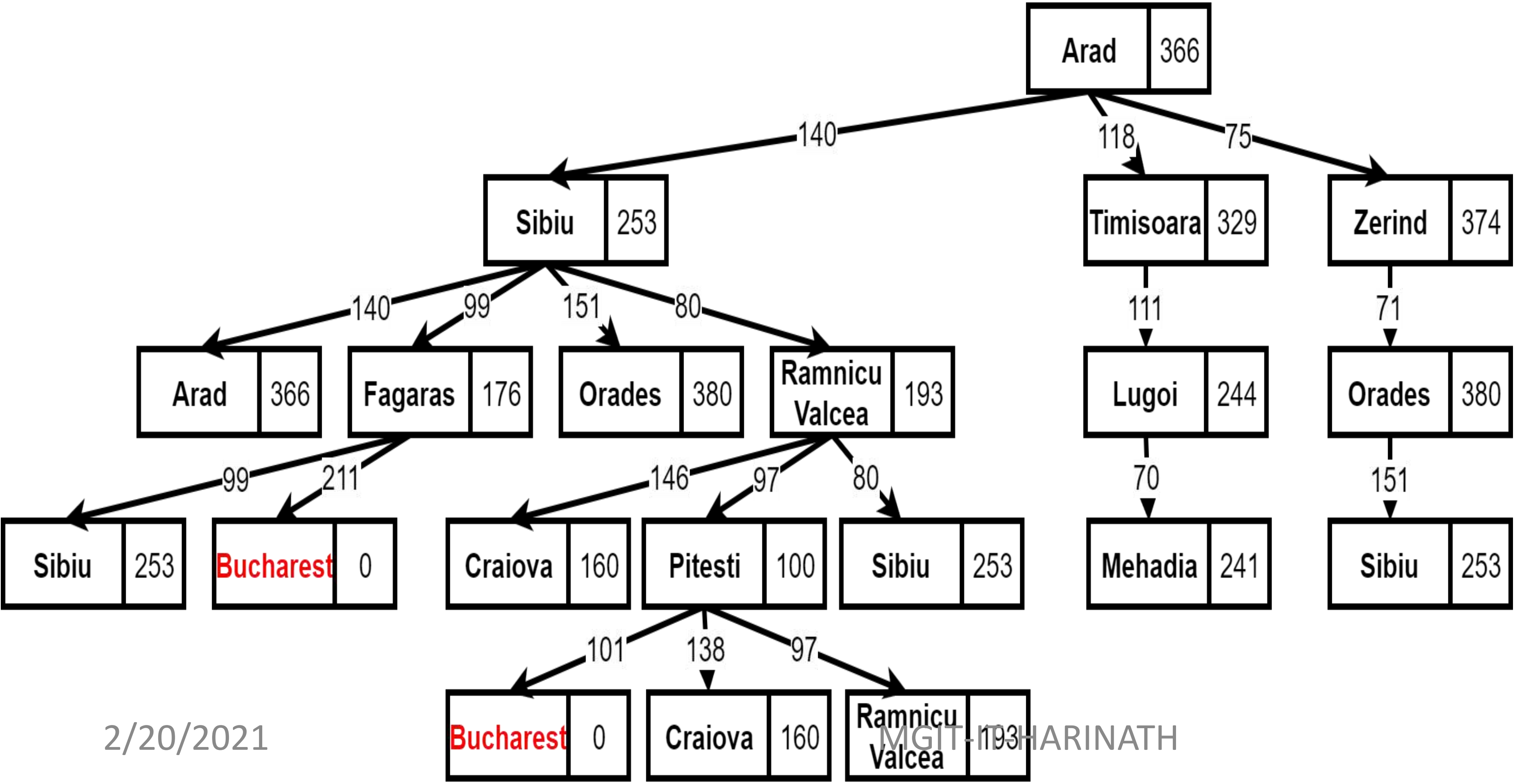
Craiova

Ramnicu

Bucharest

Heuristic + Cost = Total

0 + 418



Ramnicu Children

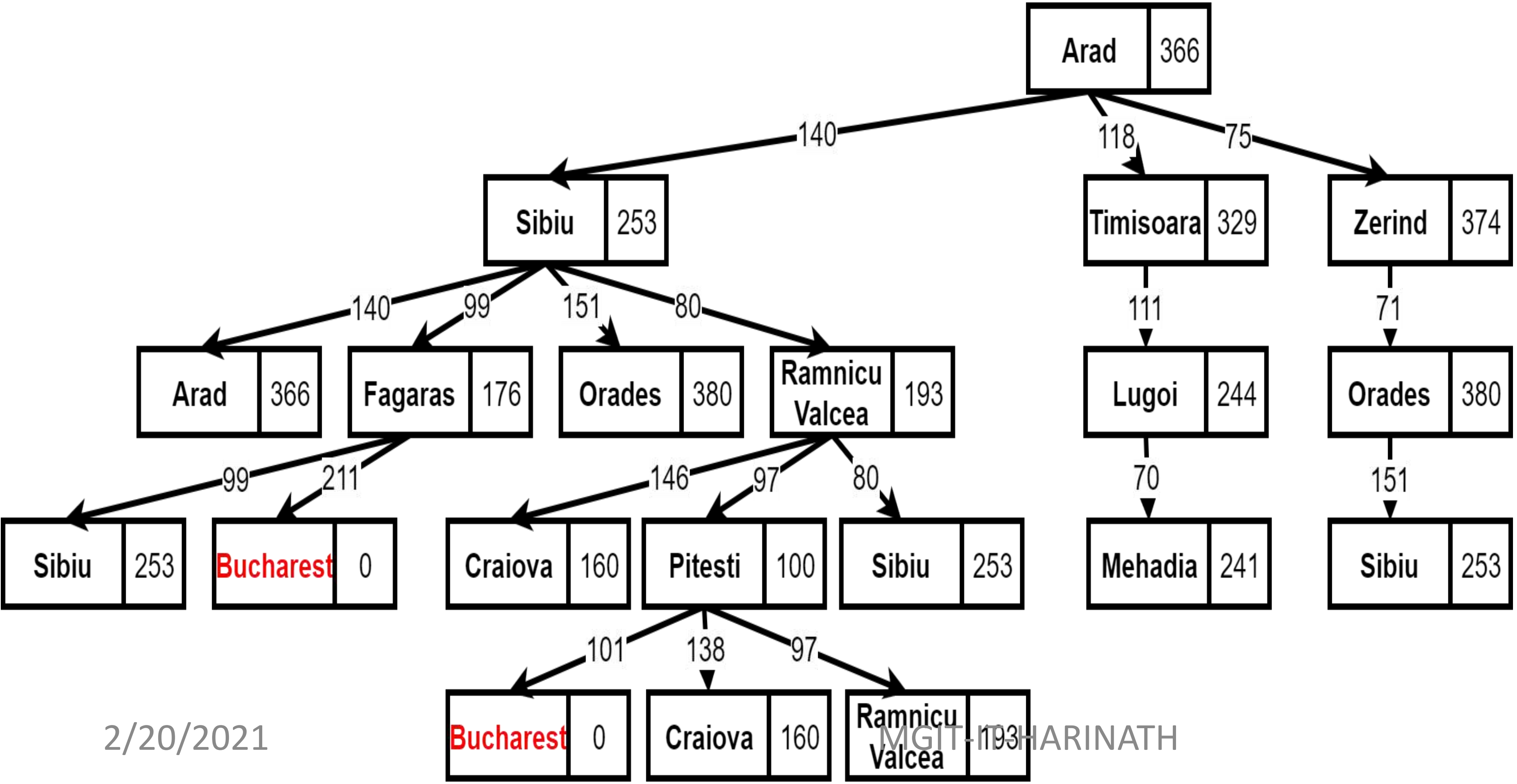
Bucharest

Craiova

Ramnicu

Bucharest

Heuristic	+	Cost	=	Total
0	+	418	=	418

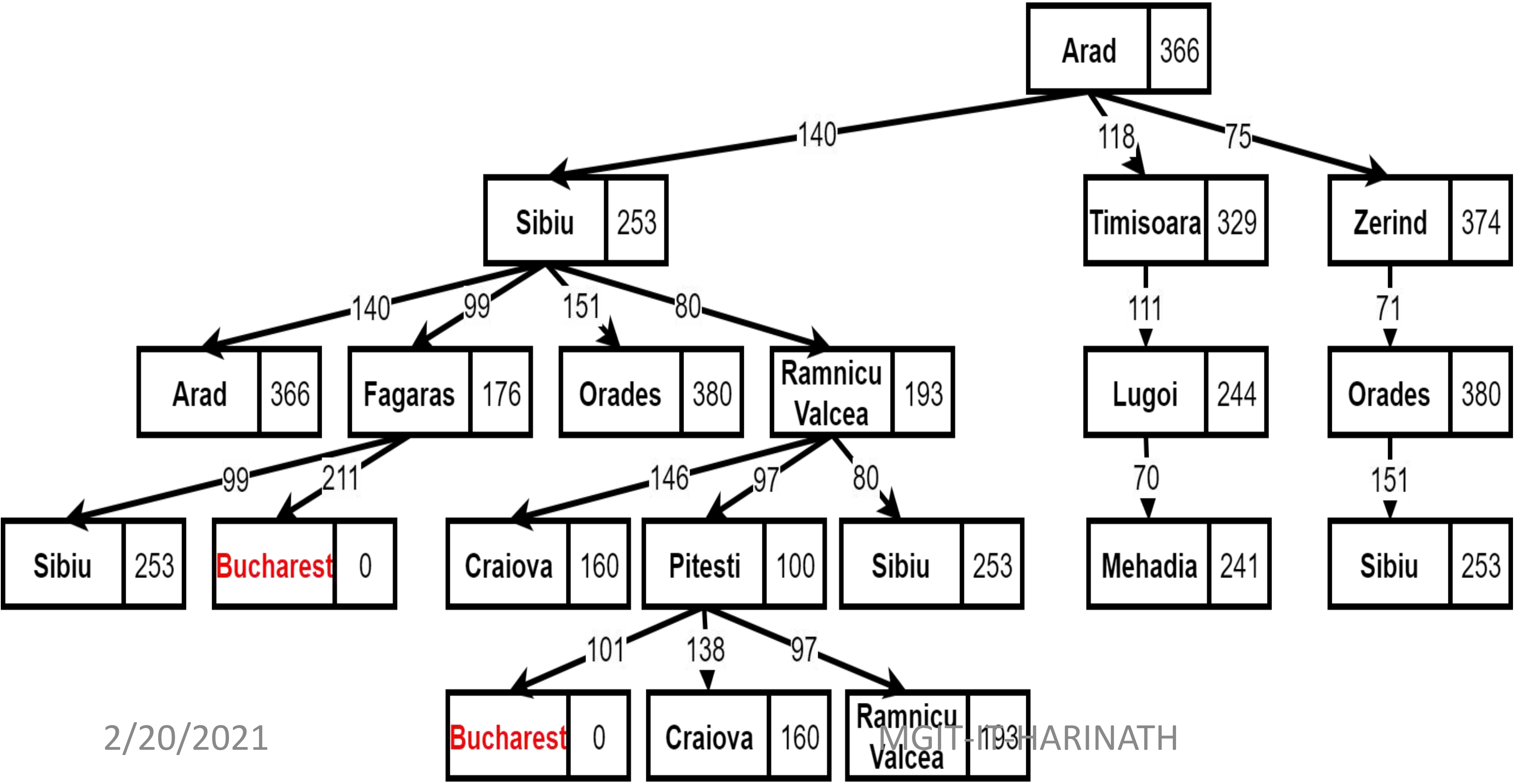


Ramnicu Children

- Bucharest
- Craiova
- Ramnicu

Bucharest

Heuristic	+	Cost	=	Total
0	+	418	=	418

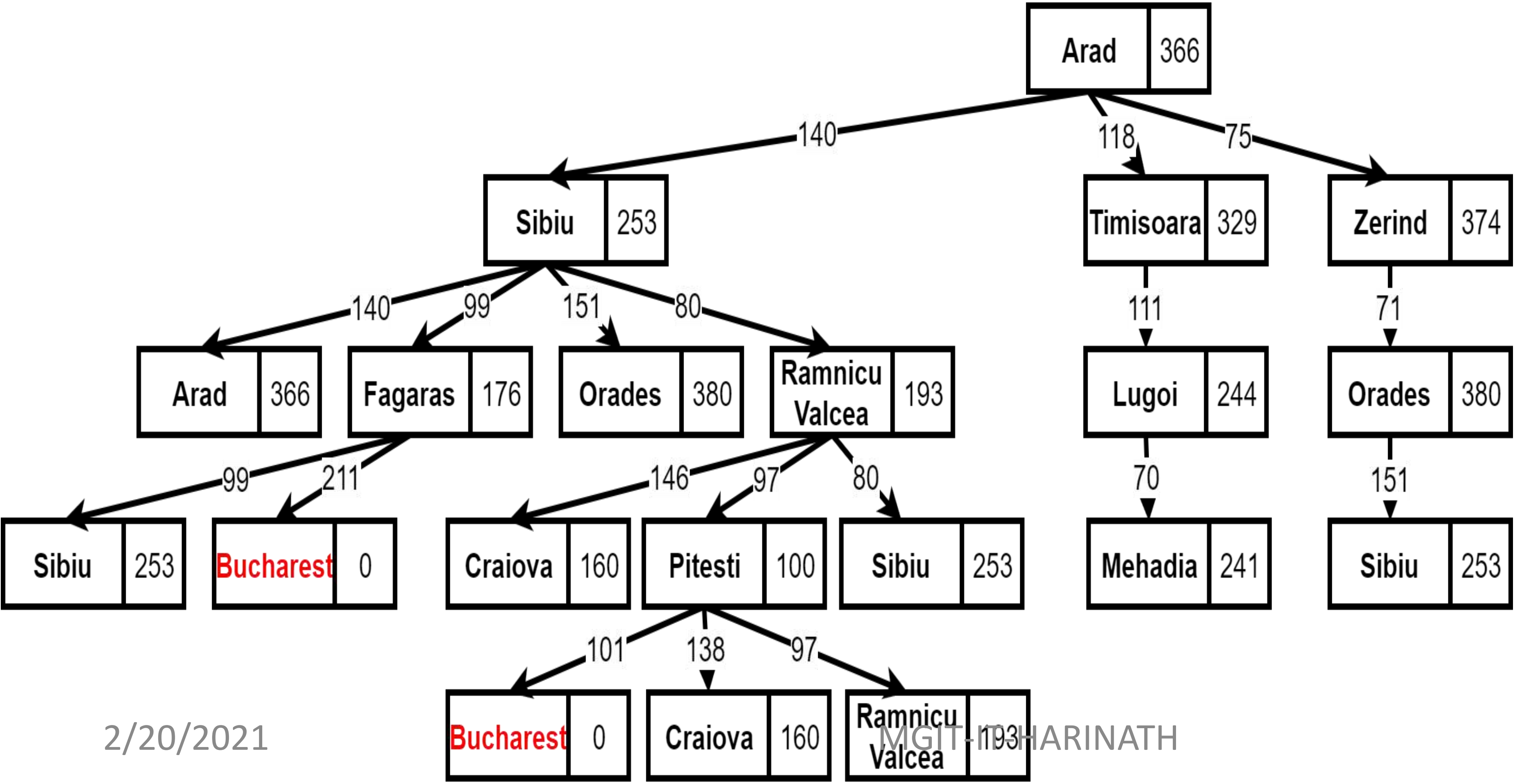


Ramnicu Children

Bucharest	418
Craiova	
Ramnicu	

Craiova

Heuristic + Cost = Total



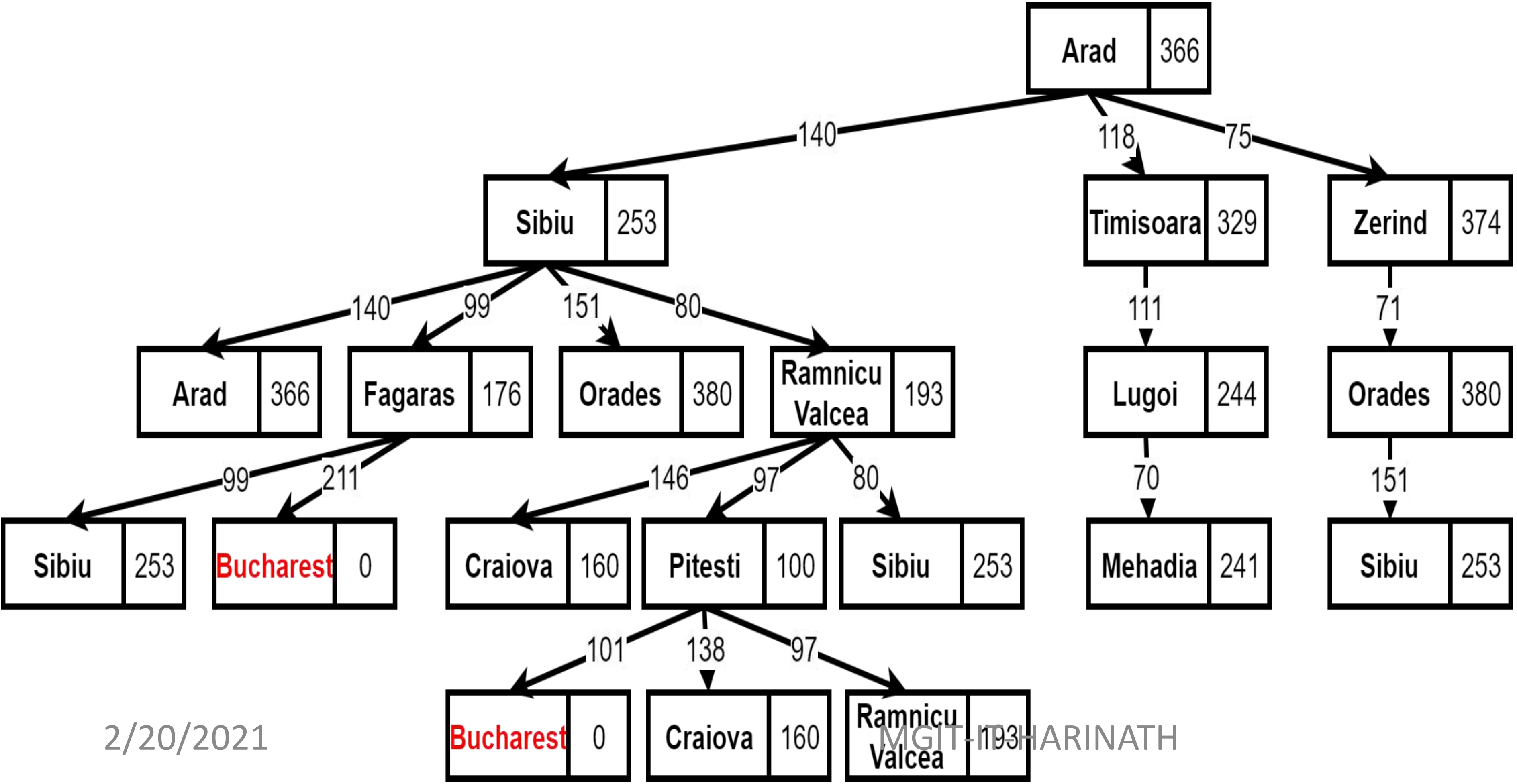
Ramnicu Children

Bucharest	418
Craiova	
Ramnicu	

Craiova

Heuristic + Cost = Total

160



Ramnicu Children

Bucharest 418

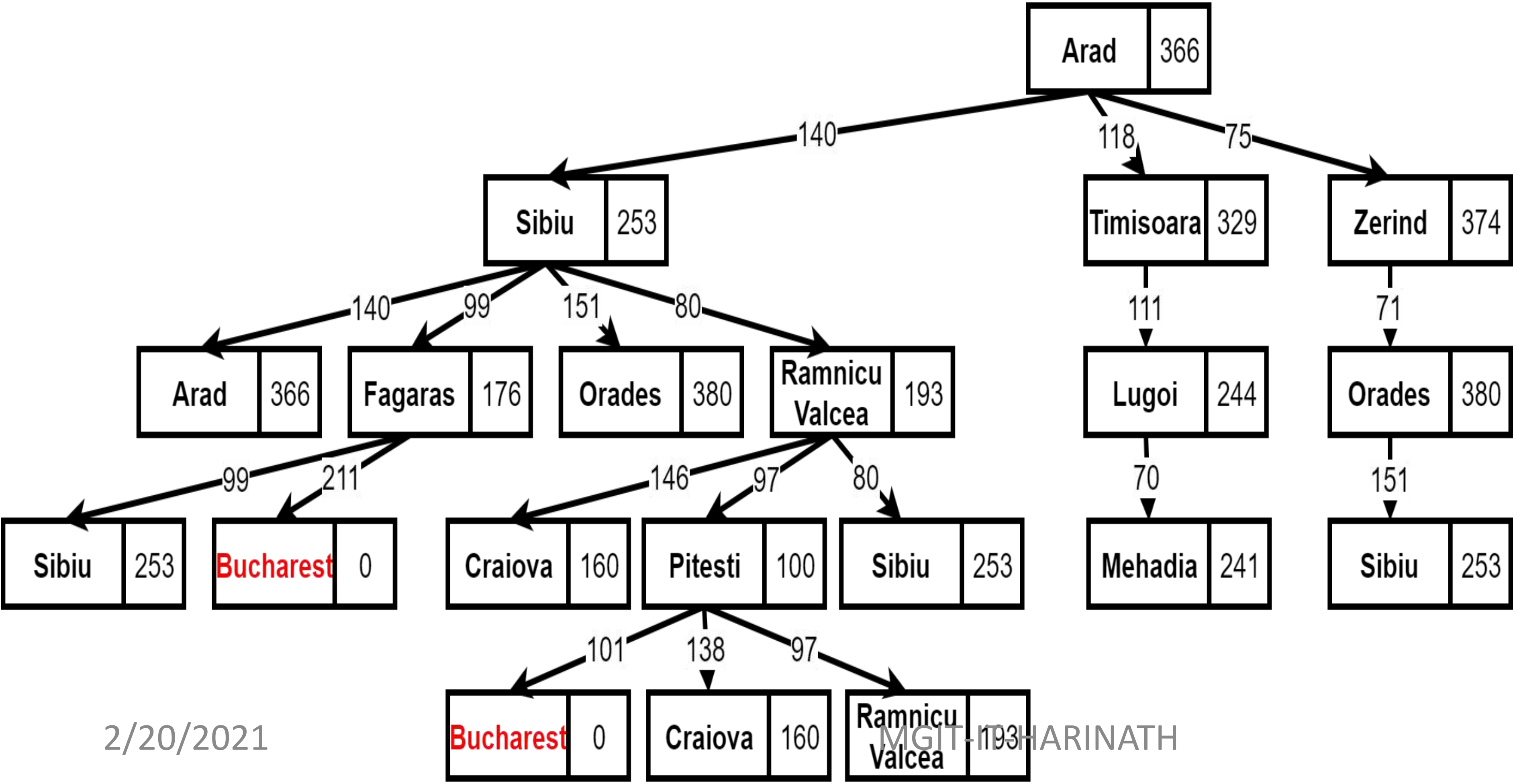
Craiova

Ramnicu

Craiova

Heuristic + **Cost** = **Total**

160 + 455



Ramnicu Children

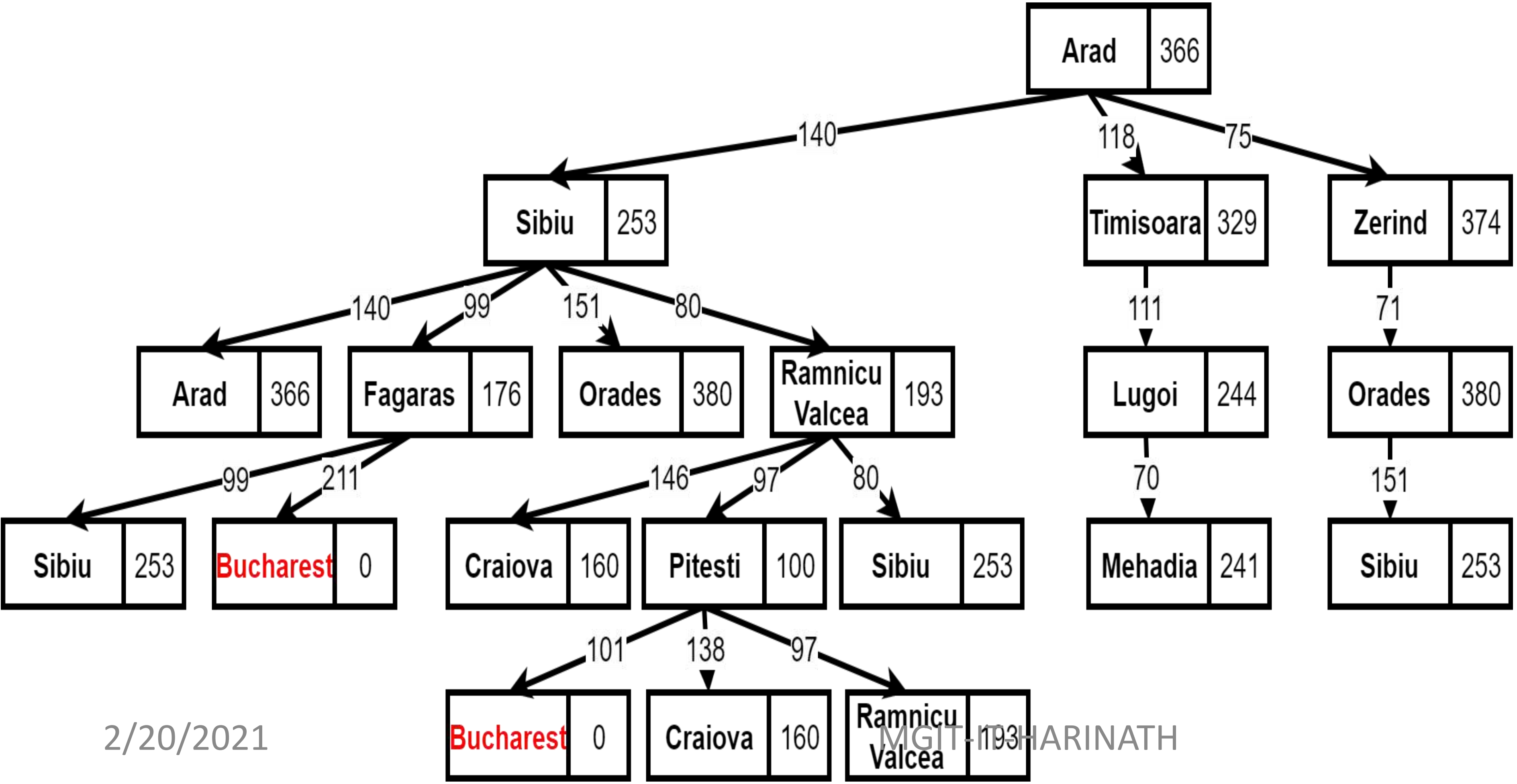
Bucharest 418

Craiova

Ramnicu

Craiova

Heuristic	+	Cost	=	Total
160	+	455	=	615

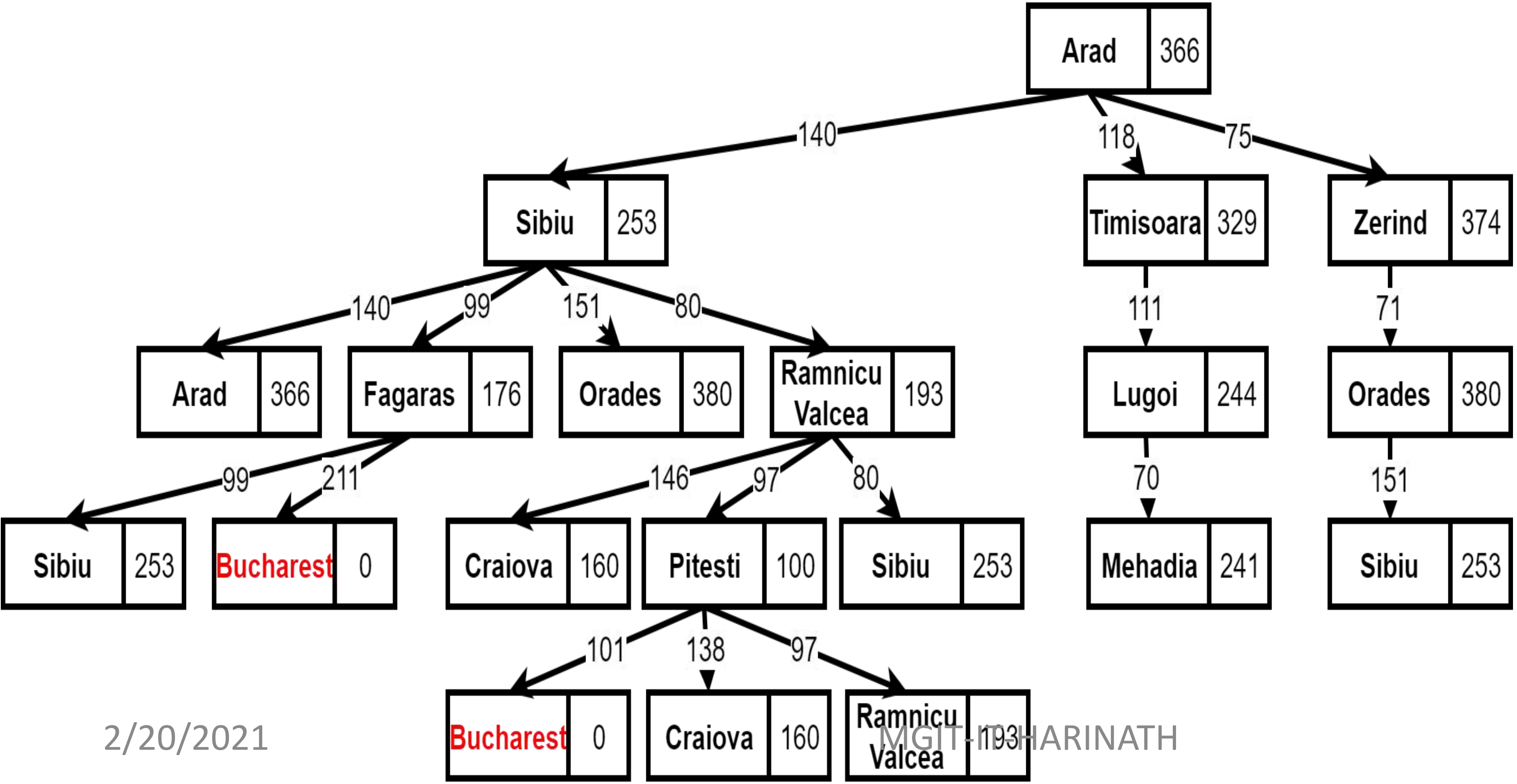


Ramnicu Children

Bucharest	418
Craiova	
Ramnicu	

Craiova

Heuristic	+	Cost	=	Total
160	+	455	=	615

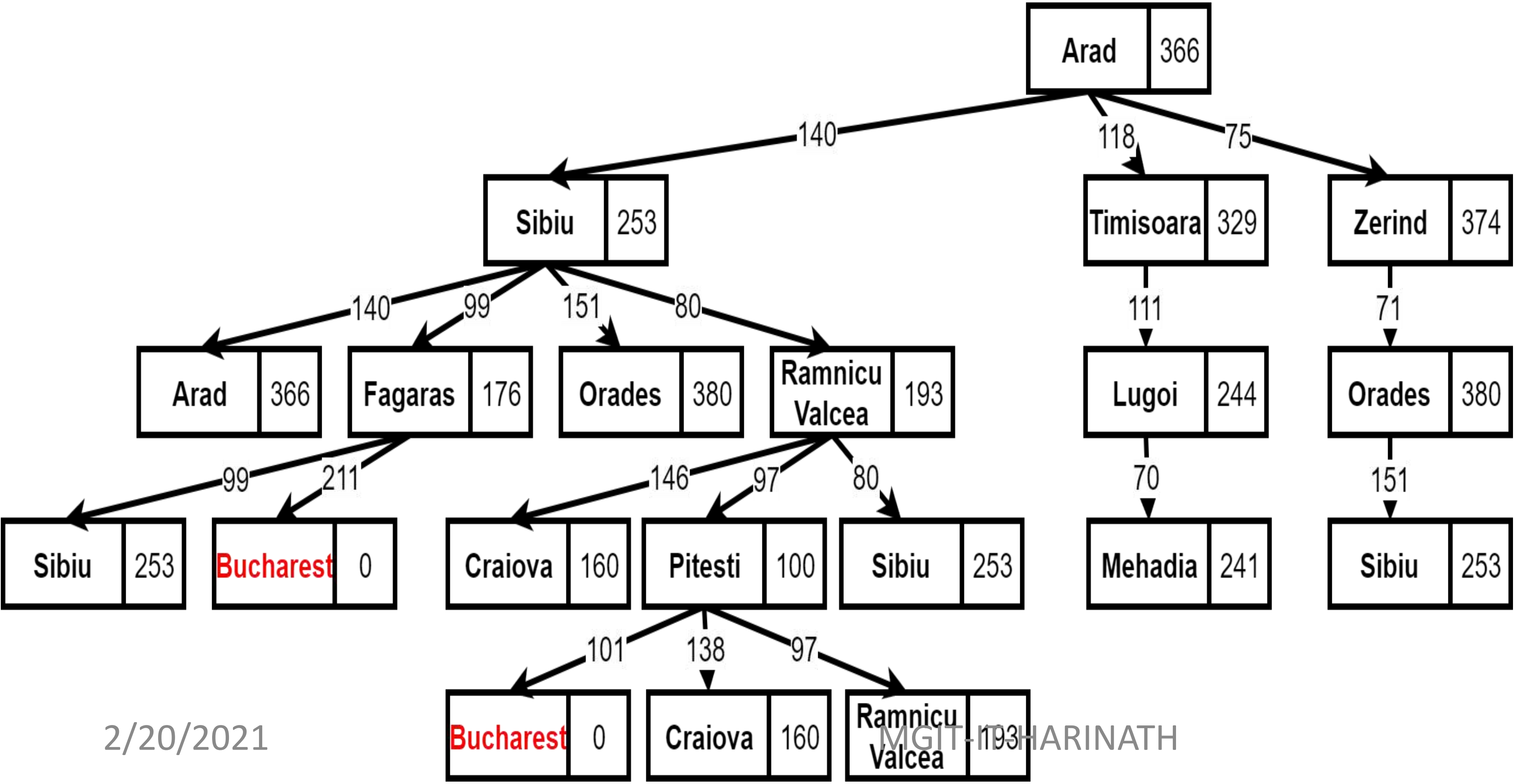


Ramnicu Children

Bucharest	418
Craiova	615
Ramnicu	

Ramnicu

Heuristic + Cost = Total



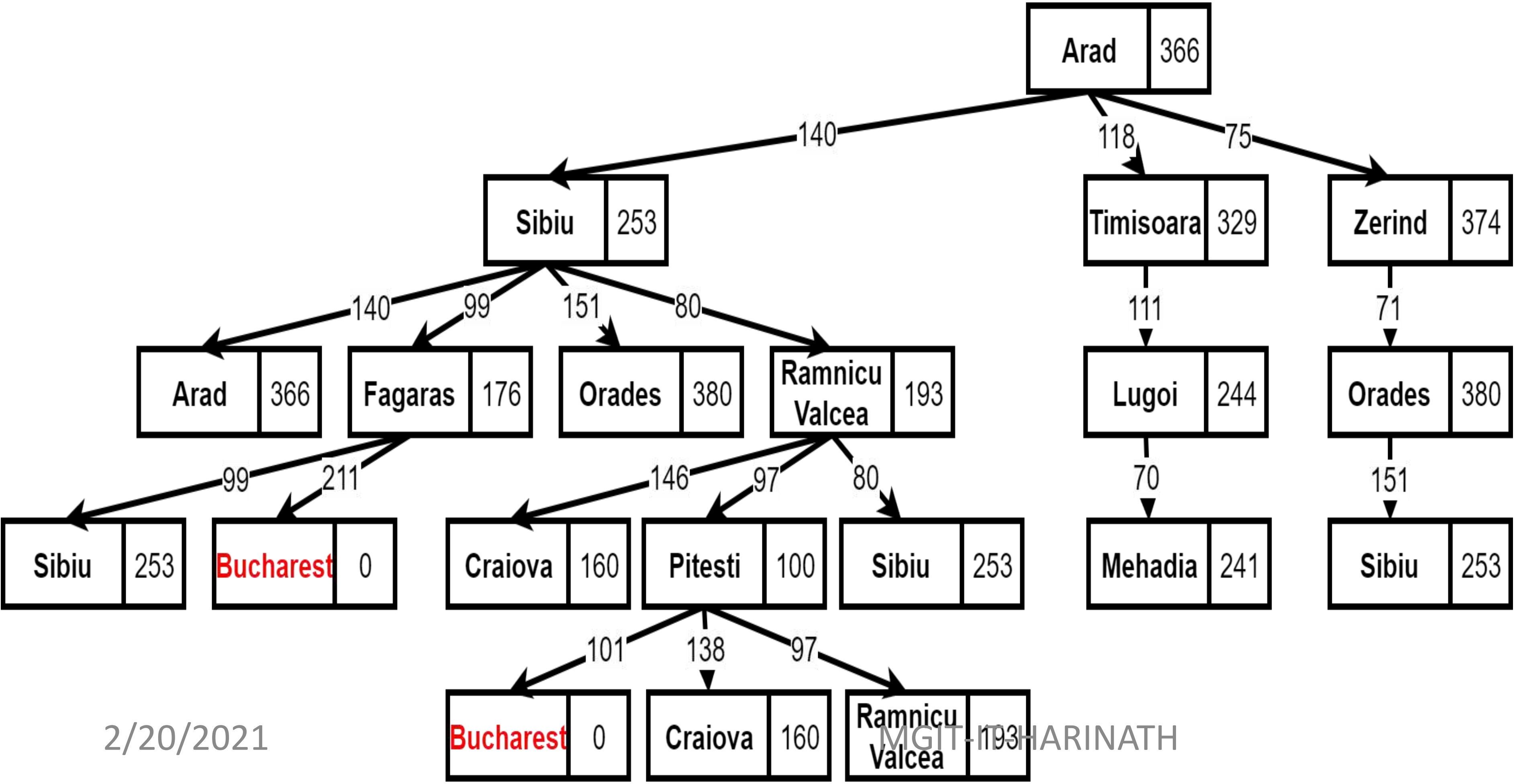
Ramnicu Children

Bucharest	418
Craiova	615
Ramnicu	

Ramnicu

Heuristic + Cost = Total

193



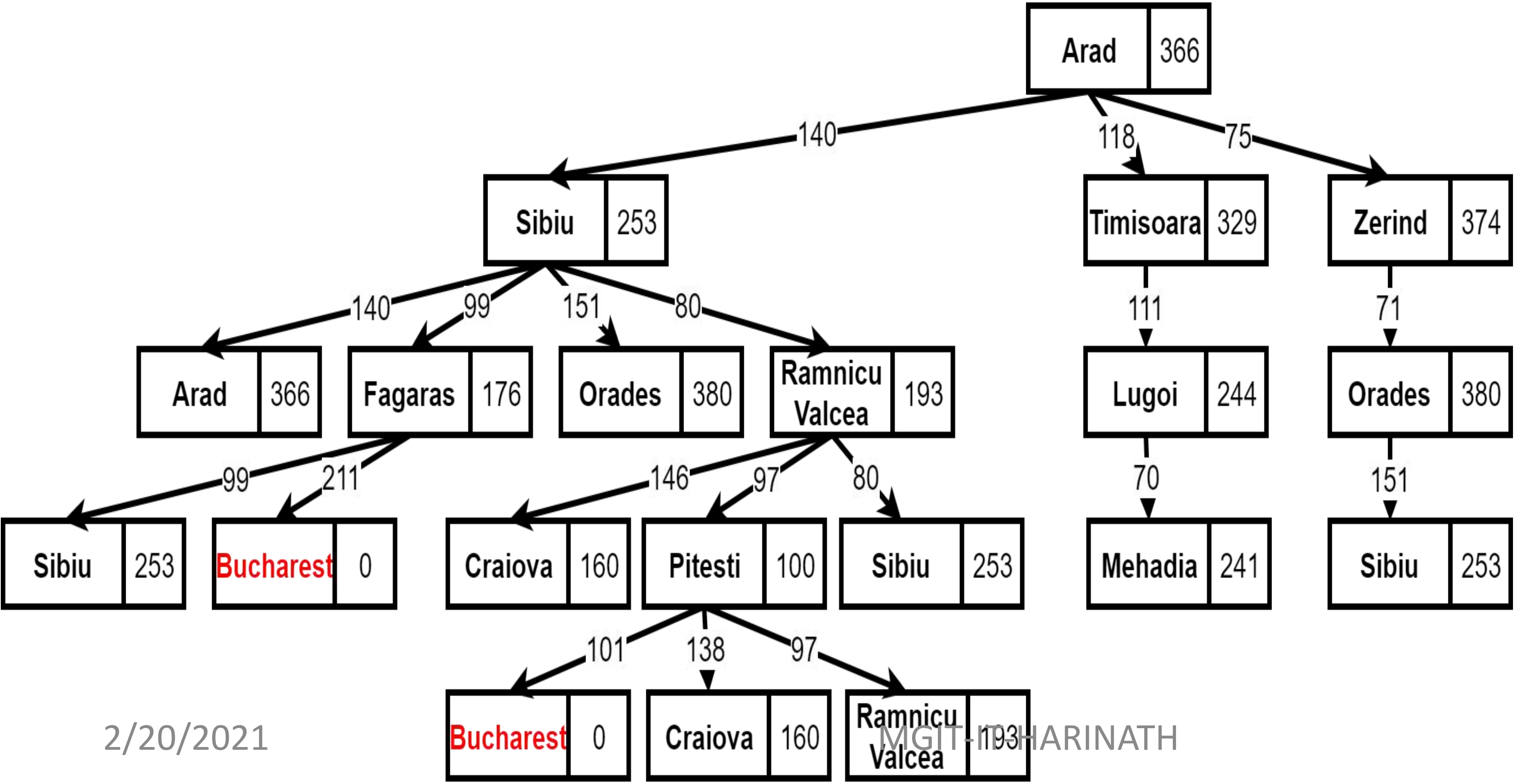
Ramnicu Children

Bucharest	418
Craiova	615
Ramnicu	

Ramnicu

Heuristic + Cost = Total

193 + 414

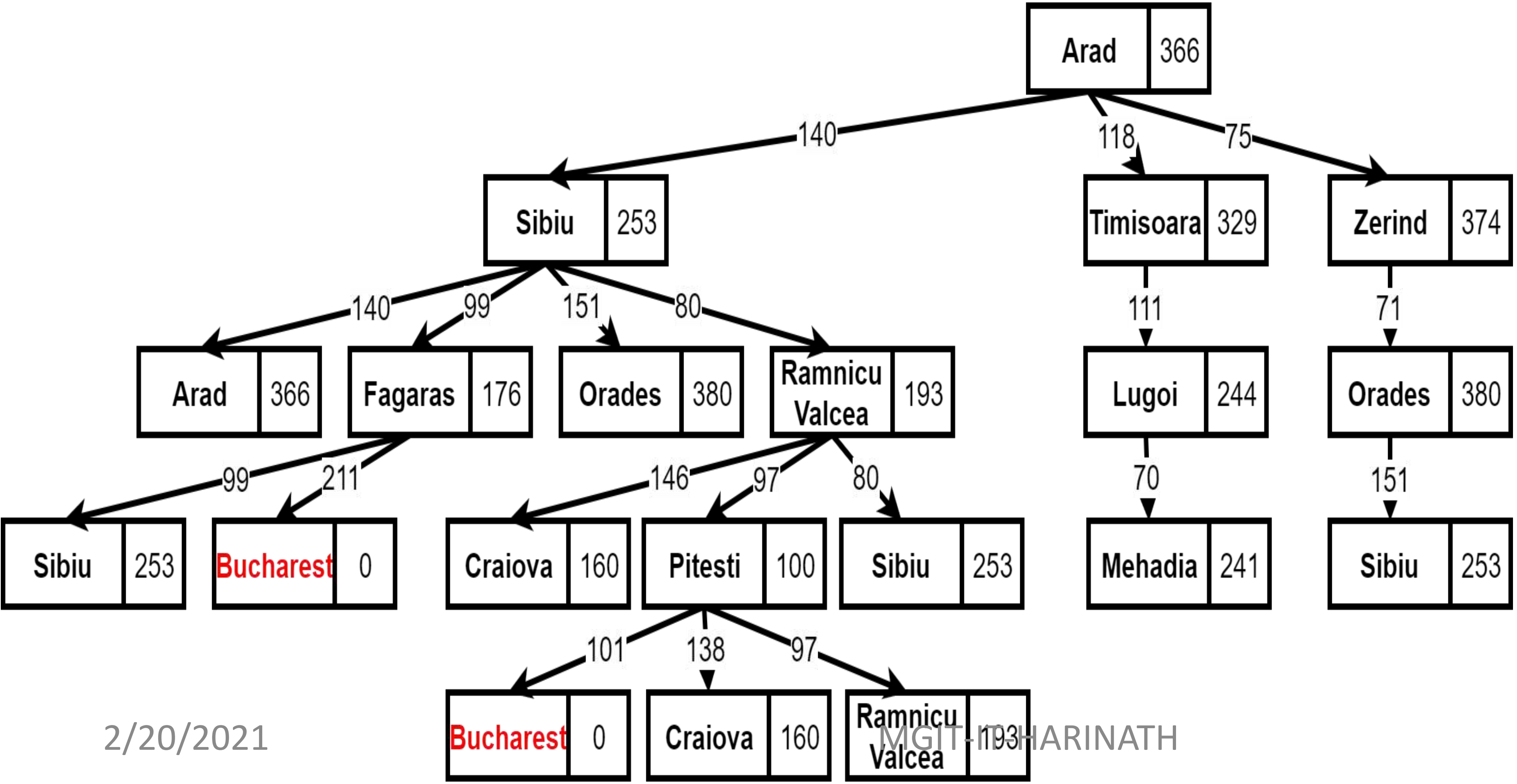


Ramnicu Children

Bucharest	418
Craiova	615
Ramnicu	

Ramnicu

Heuristic	+	Cost	=	Total
193	+	414	=	607

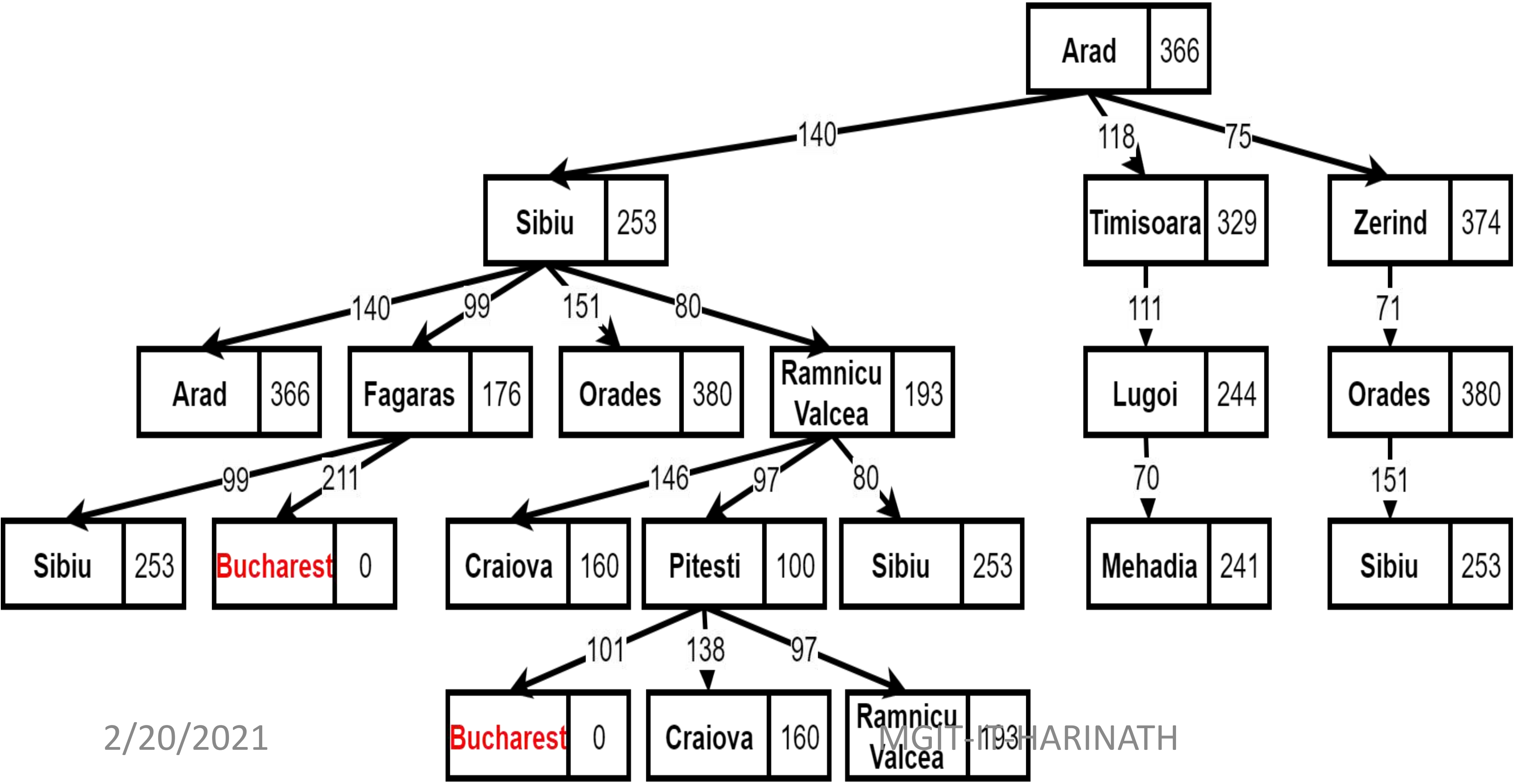


Ramnicu Children

Bucharest	418
Craiova	615
Ramnicu	

Ramnicu

Heuristic	+	Cost	=	Total
193	+	414	=	607



Ramnicu Children

Bucharest	418
Craiova	615
Ramnicu	607

Current Queue

Timisoar 447 | Zerin 449 | Craiova 526 | Sibiu 553 | Buchare 450 | Sibiu 591

Arad 646 | Orade 671

Buchares 418 | Craiova 615 | Ramnic 607

Current Queue

Buchares 418

Timisoar 447

Zerin 449

Craiova 526

Sibiu 553

Buchare 450

Sibiu 591

Ramnic 607

Craiova 615

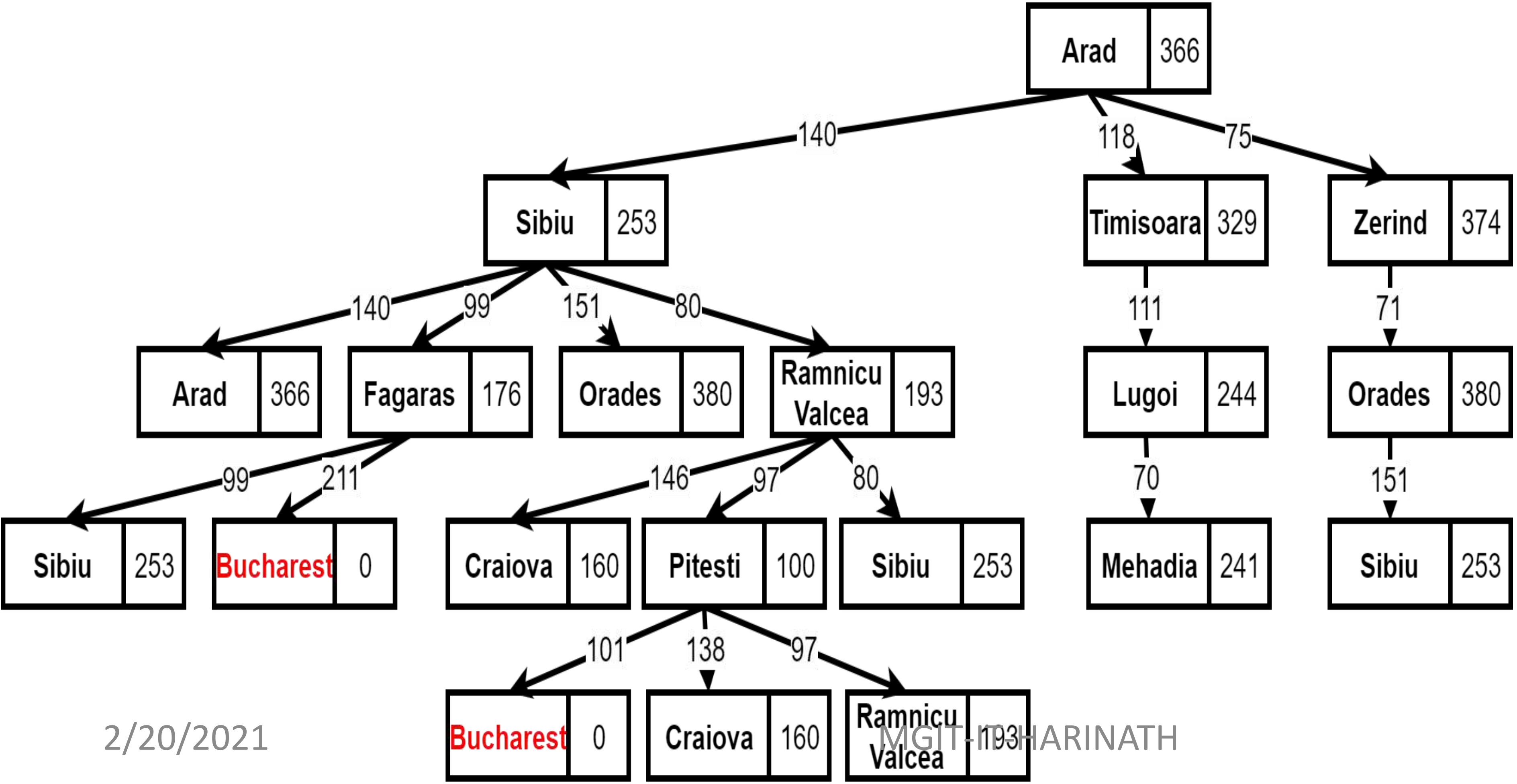
Arad 646

Orade 671

Current Queue

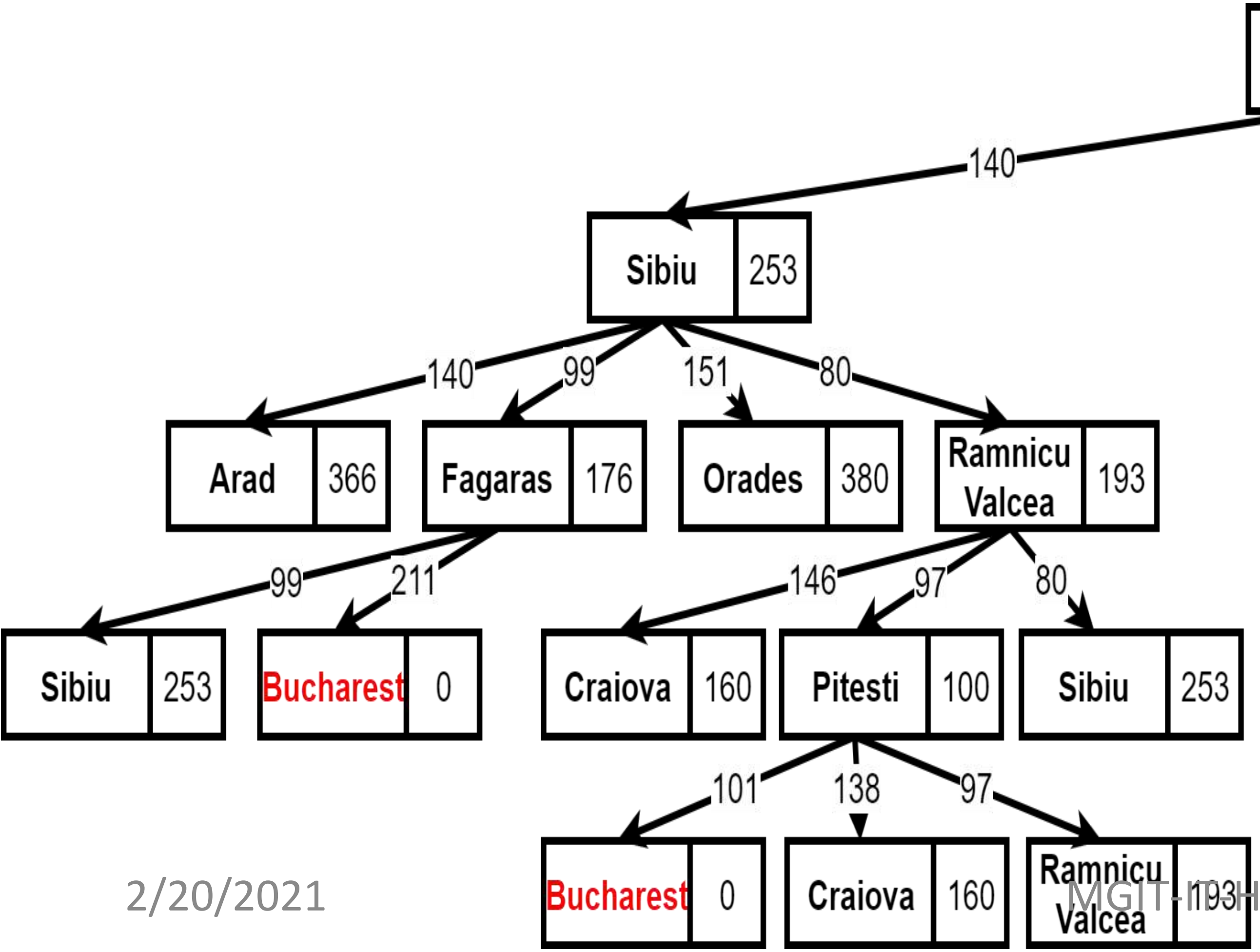
Buchare 418	Timisoar 447	Zerin 449	Craiova 526	Sibiu 553	Buchare 450
Sibiu 591	Ramnic 607	Craiova 615	Arad 646	Orade 671	

Bucharest



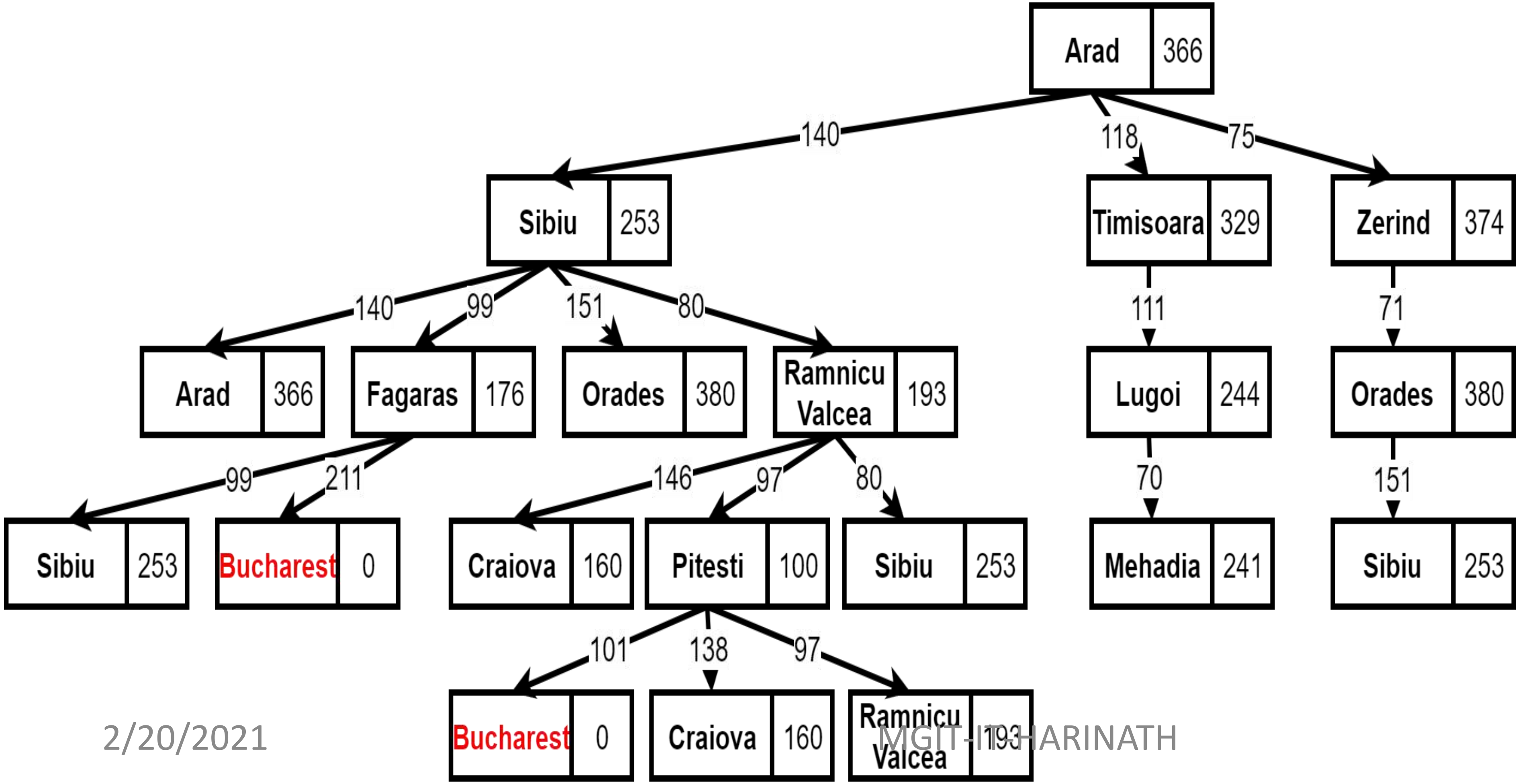
Arad 366
140
118
75
Sibiu 253
Timisoara 329
Zerind 374
140
99
151
80
111
71
Arad 366
Fagaras 176
Orades 380
Ramnicu Valcea 193
Lugoi 244
Orades 380
99
211
146
97
80
70
151
Sibiu 253
Bucharest 0
Craiova 160
Pitesti 100
Sibiu 253
Mehadia 241
Sibiu 253
101
138
97
Bucharest 0
Craiova 160
Ramnicu Valcea 193

Bucharest

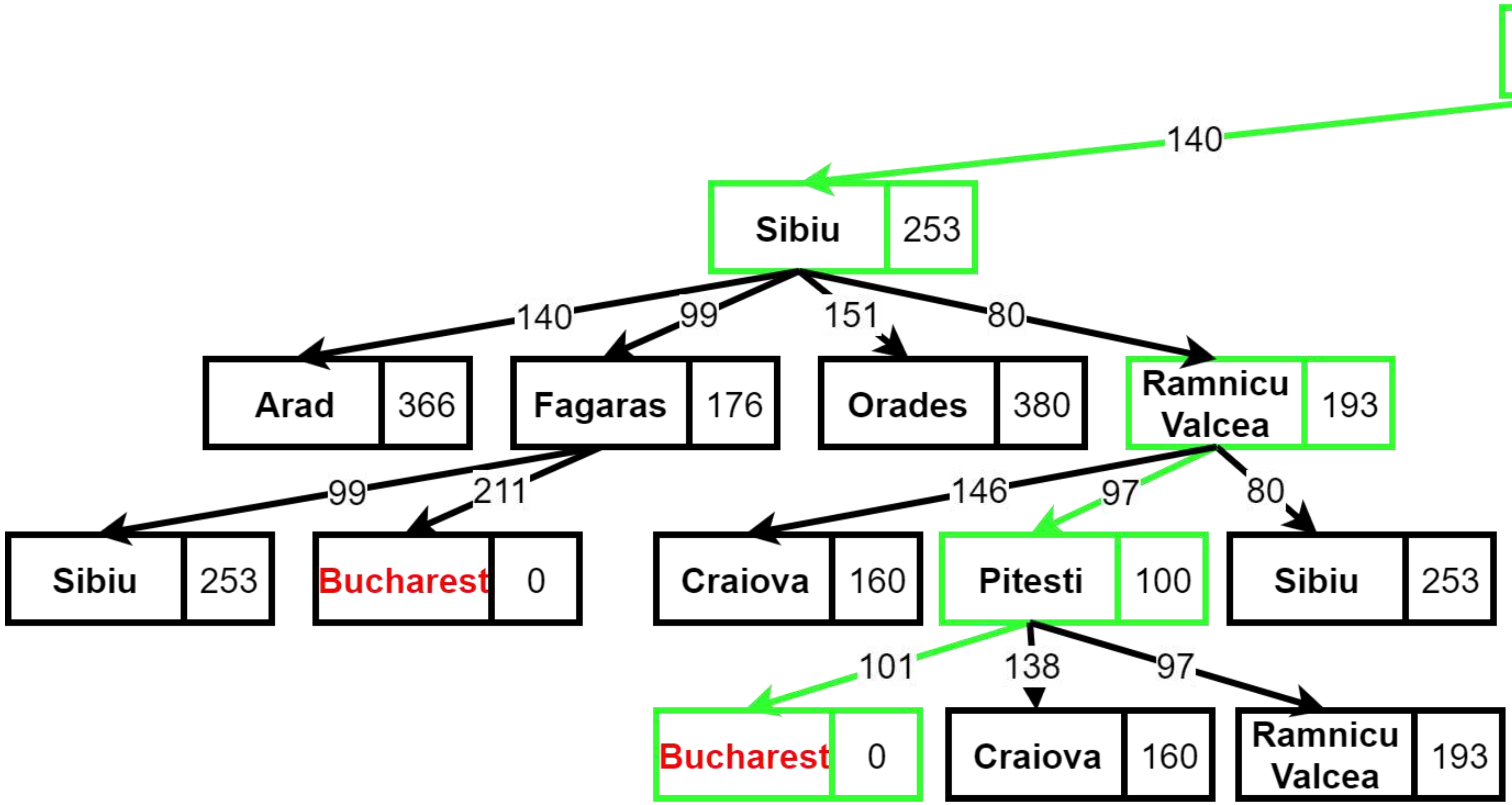

Arad 366
140
118
75
Sibiu 253
Timisoara 329
Zerind 374
140
99
151
80
111
71
Arad 366
Fagaras 176
Orades 380
Ramnicu Valcea 193
Lugoi 244
Orades 380
99
211
146
97
80
70
151
Sibiu 253
Bucharest 0
Craiova 160
Pitesti 100
Sibiu 253
Mehadia 241
Sibiu 253
101
138
97
Bucharest 0
Craiova 160
Ramnicu Valcea 193

Bucharest

GOAL



Path to Goal



Arad 366
140
118
75
Sibiu 253
Timisoara 329
Zerind 374
140
99
151
80
111
71
Arad 366
Fagaras 176
Orades 380
Ramnicu Valcea 193
Lugoi 244
Orades 380
99
211
146
97
80
70
151
Sibiu 253
Bucharest 0
Craiova 160
Pitesti 100
Sibiu 253
Mehadia 241
Sibiu 253
101
138
97
Bucharest 0
Craiova 160
Ramnicu Valcea 193

Analysis

- A* is **optimally efficient** for any given **Consistent heuristic.**
- A* Search is **Complete, Optimal.**
- A* usually keeps **all generated nodes in Memory.**
- A* **runs out of space long before it runs out of time.**
- A* is **not practical for many large-scale Problems.**

A*(Admissible Heuristic)

- h(n) is Admissible Heuristic that **never over estimates** the cost to reach the goal.

estimated — Actual

$h(n) \leq h^*(n)$ - underestimation

$h(n) \geq h^*(n)$ - overestimation

optimal

Memory Bounded Heuristic Search

IDA*

- Simplest way to reduce memory requirements is to adapt idea of **Iterative Deepening** to Heuristic Search content.
- Difference

  -Use **Cutoff f-cost(f+g)** rather than depth.(IDA*)

Memory Bounded Heuristic Search

Recursive BFS(Best First Search)

➢Simple recursive algorithm

➢Similar to Recursive Depth-First Search but uses f-limit variable to keep track of f-value of best alternative path.

➢If Current Node exceeds the limit then back track choose alternate path.

➢**RBFS replaces f-value of each node along the path with the backed up value(best f-value of its children)**

Analysis

- IDA* and RBFS suffer from **using too little memory.**
- IDA* retains only current f-cost limit
- RBFS retains more information in Memory but it uses Linear Space.

Using available Memory in A*

- ➢**MA*(Memory Bound A*)**
- ➢**SMA*(Simplified Memory Bound A*)**
- ➢SMA* is simple, similar to A*(expands best leaf node until Memory is full)
- ➢SMA* always drops the worst leaf node(one with highest f-value).
- ➢SMA* backs up value of the forgotten node to its parent(like RBFS)

Heuristic Functions

- Technique to solve problems quickly
- Eg: 8 Puzzle Problem(3^{20}-Search Space possible)
  - h1- No of misplaced tiles
  - h2-Mahanhatten Distance
- h2- admissible

Generating Admissible Heuristic from Relaxed Problems

- ➢Problem with fewer restrictions(Relaxed Problem)
- ➢Super Graph(State Space Graph of Relaxed Problem)
- ➢Creates additional edges(Removal of restrictions)
- ➢Relaxed Problems better solution if added edges provide shortcuts.

Generating Admissible Heuristic from Sub Problems

- Pattern Databases.
- Store exact solution costs for every possible subproblem instance.
- Compute Admissible Heuristic for Complete State by observing corresponding sub problem configuration in Database.

Learning Heuristic from Experience

- **Experience** (Solving lot of Problems).
- **Construct function h(n)-**(from example problems)
- **Inductive Learning**(Works Best when supplied with features of the State that are relevant to predicting states value)

- Before an agent can start searching for solutions, a **goal** must be identified and a well-defined **problem** must be formulated.
- A problem consists of five parts: the **initial state**, a set of **actions**, a **transition model** describing the results of those actions, a **goal test** function, and a **path cost** function. The environment of the problem is represented by a **state space**. A **path** through the state space from the initial state to a goal state is a **solution**.
- Search algorithms treat states and actions as **atomic**: they do not consider any internal structure they might possess.
- A general TREE-SEARCH algorithm considers all possible paths to find a solution, whereas a GRAPH-SEARCH algorithm avoids consideration of redundant paths.
- Search algorithms are judged on the basis of **completeness**, **optimality**, **time complexity**, and **space complexity**. Complexity depends on b, the branching factor in the state space, and d, the depth of the shallowest solution.

- **Uninformed search** methods have access only to the problem definition. The basic algorithms are as follows:
  - **Breadth-first search** expands the shallowest nodes first; it is complete, optimal for unit step costs, but has exponential space complexity.
  - **Uniform-cost search** expands the node with lowest path cost, $g(n)$, and is optimal for general step costs.
  - **Depth-first search** expands the deepest unexpanded node first. It is neither complete nor optimal, but has linear space complexity. **Depth-limited search** adds a depth bound.
  - **Iterative deepening search** calls depth-first search with increasing depth limits until a goal is found. It is complete, optimal for unit step costs, has time complexity comparable to breadth-first search, and has linear space complexity.
  - **Bidirectional search** can enormously reduce time complexity, but it is not always applicable and may require too much space.

- **Informed search** methods may have access to a **heuristic** function $h(n)$ that estimates the cost of a solution from n.
  - The generic **best-first search** algorithm selects a node for expansion according to an **evaluation function**.
  - **Greedy best-first search** expands nodes with minimal $h(n)$. It is not optimal but is often efficient.
  - **A* search** expands nodes with minimal $f(n) = g(n) + h(n)$. A* is complete and optimal, provided that $h(n)$ is admissible (for TREE-SEARCH) or consistent (for GRAPH-SEARCH). The space complexity of A* is still prohibitive.
  - **RBFS** (recursive best-first search) and **SMA*** (simplified memory-bounded A*) are robust, optimal search algorithms that use limited amounts of memory; given enough time, they can solve problems that A* cannot solve because it runs out of memory.
- The performance of heuristic search algorithms depends on the quality of the heuristic function. One can sometimes construct good heuristics by relaxing the problem definition, by storing precomputed solution costs for subproblems in a pattern database, or by learning from experience with the problem class.

Thank you

Uninformed Search

- An uninformed **(a.k.a. blind, brute-force)** search algorithm generates the search tree **without using any domain specific knowledge.**
- **No additional information** about states beyond that provided in the problem definition.
- All they can do is **generate successors** and **distinguish a goal state from a non-goal state.**
- **All search strategies** are **distinguished** by the **order in which nodes are expanded.**

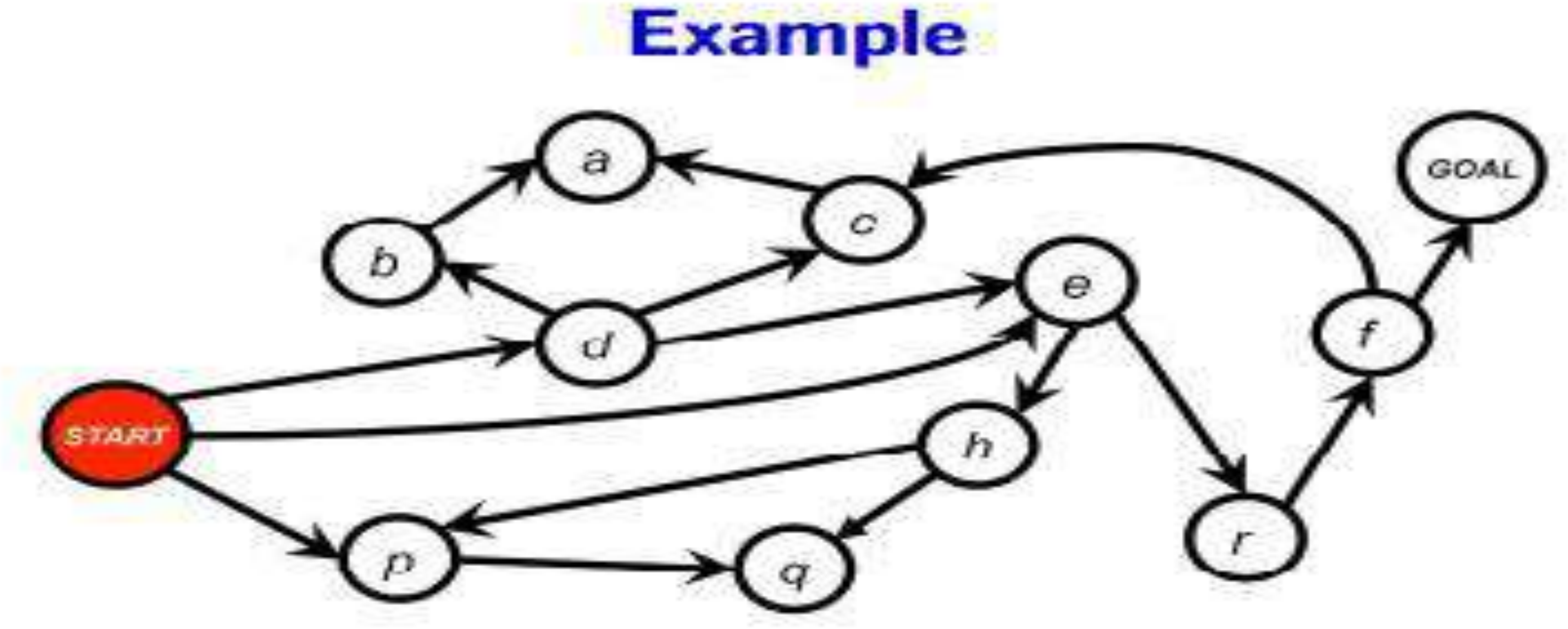
BREADTH - FIRST SEARCH

Breadth-First Search

- **Expand shallowest unexpanded node**
- **Implementation:**

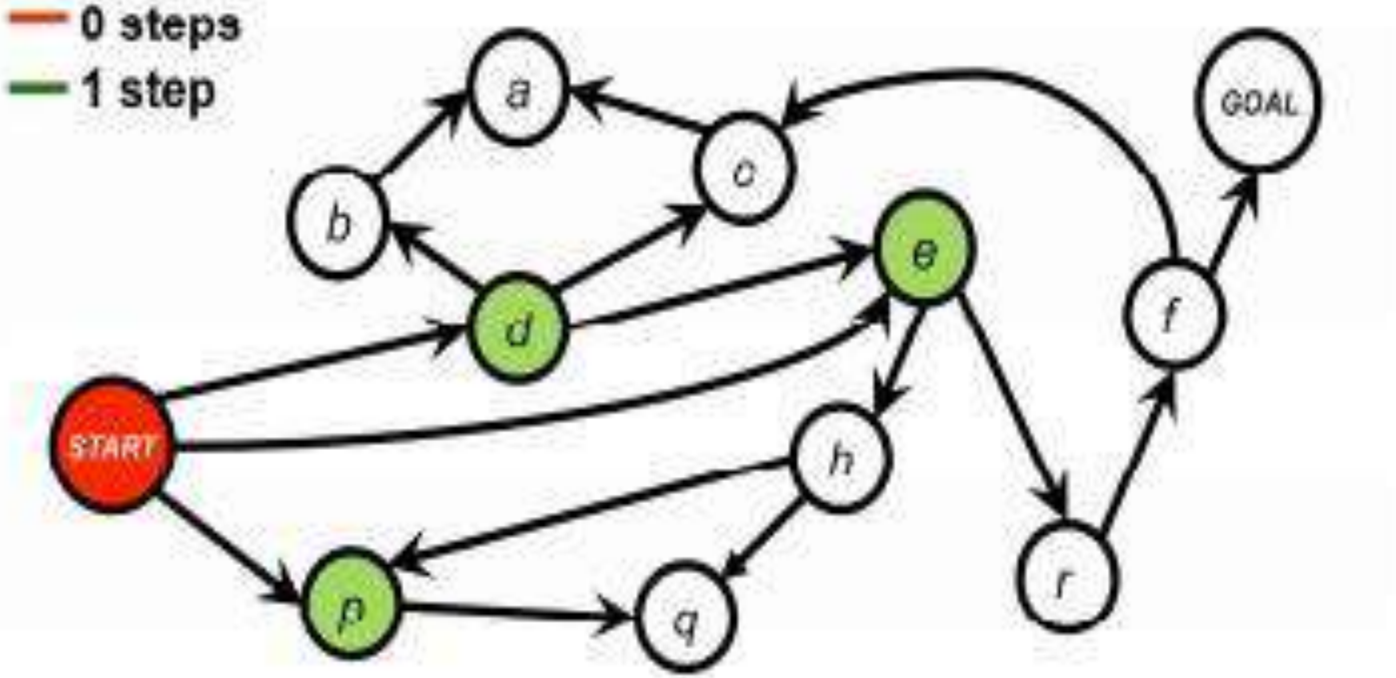
A **FIFO queue**, i.e., new successors go at end.

Example-Graph



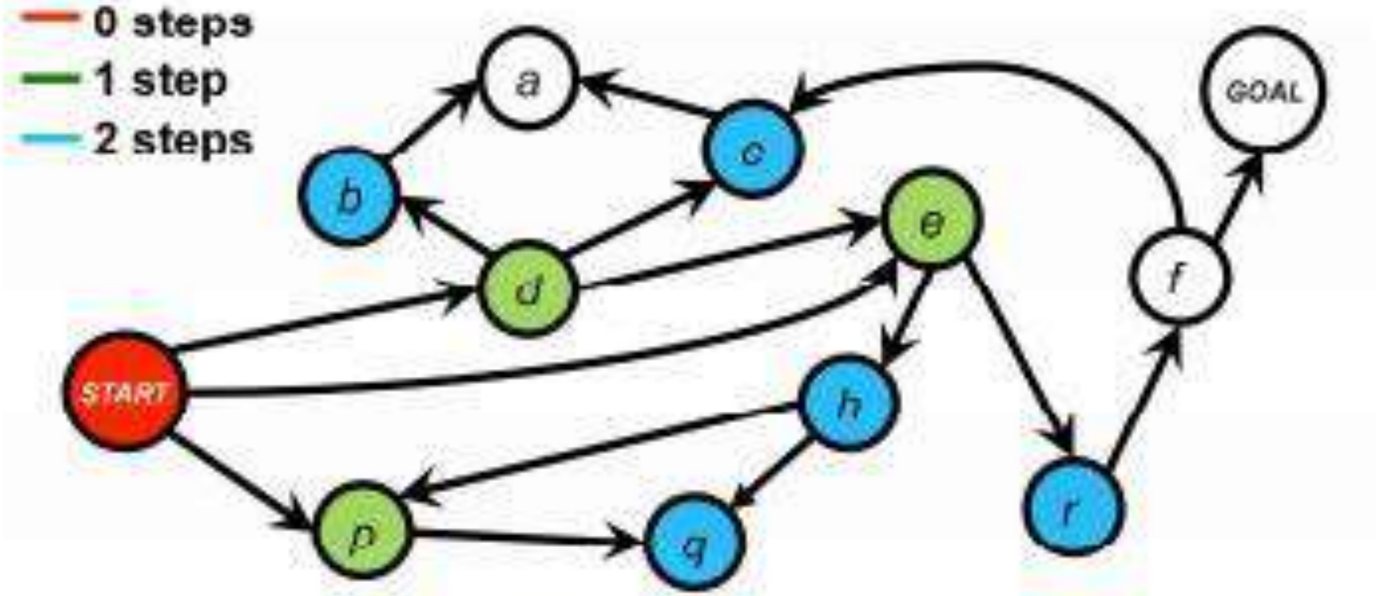
Label all start states as set V_0

Example



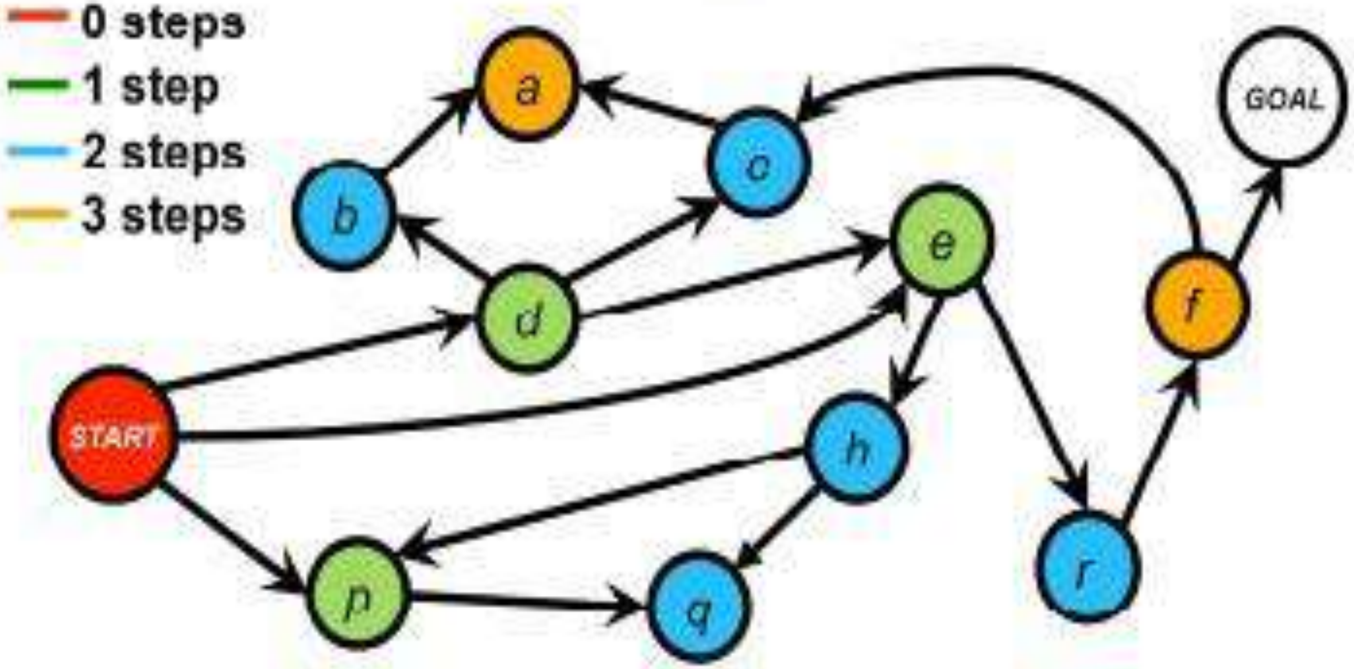
Label all successors of states in V_0 that have not yet been labelled as set V_1

Example



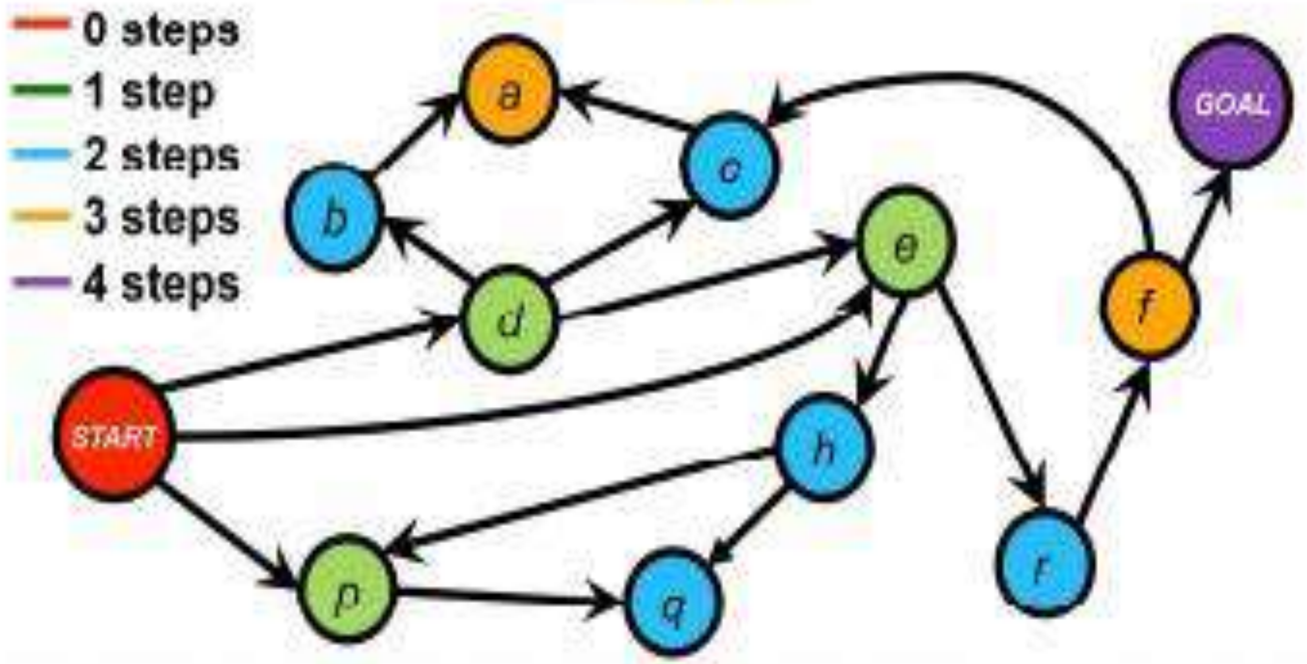
Label all successors of states in V_1 that have not yet been labelled as set V_2

Example



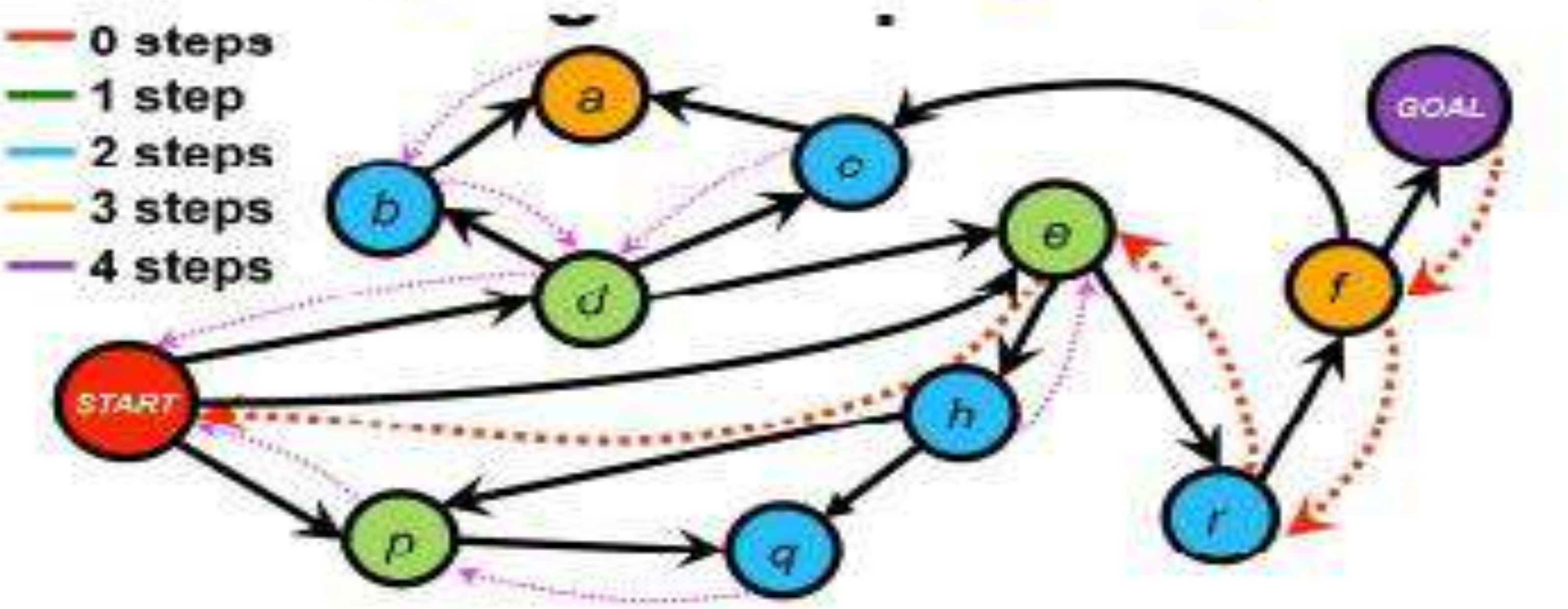
Label all successors of states in V_2 that have not yet been labelled as set V_3

Example



Label all successors of states in V_3 that have not yet been labelled as set V_4

Example: Recovering the path



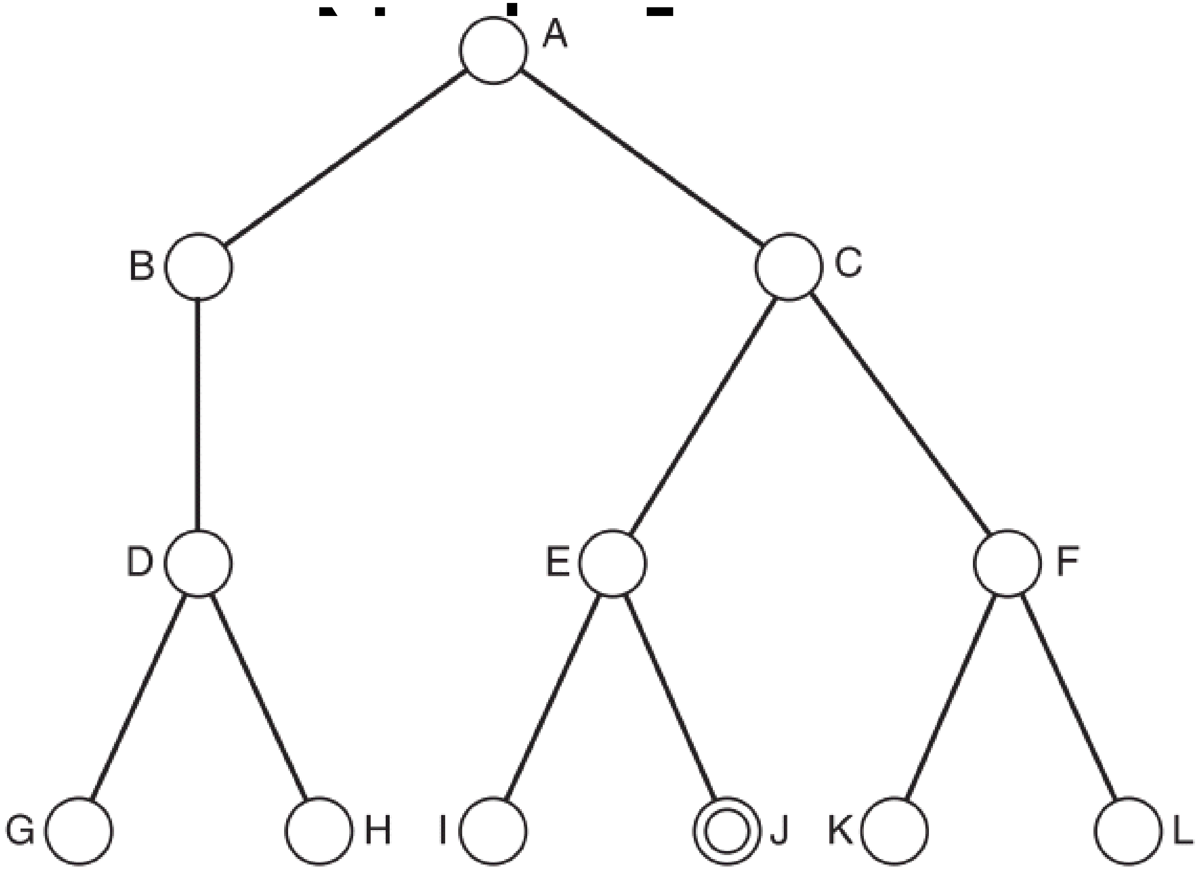
Follow pointers back to the parent node to find the path

Breadth-First Search Goal -

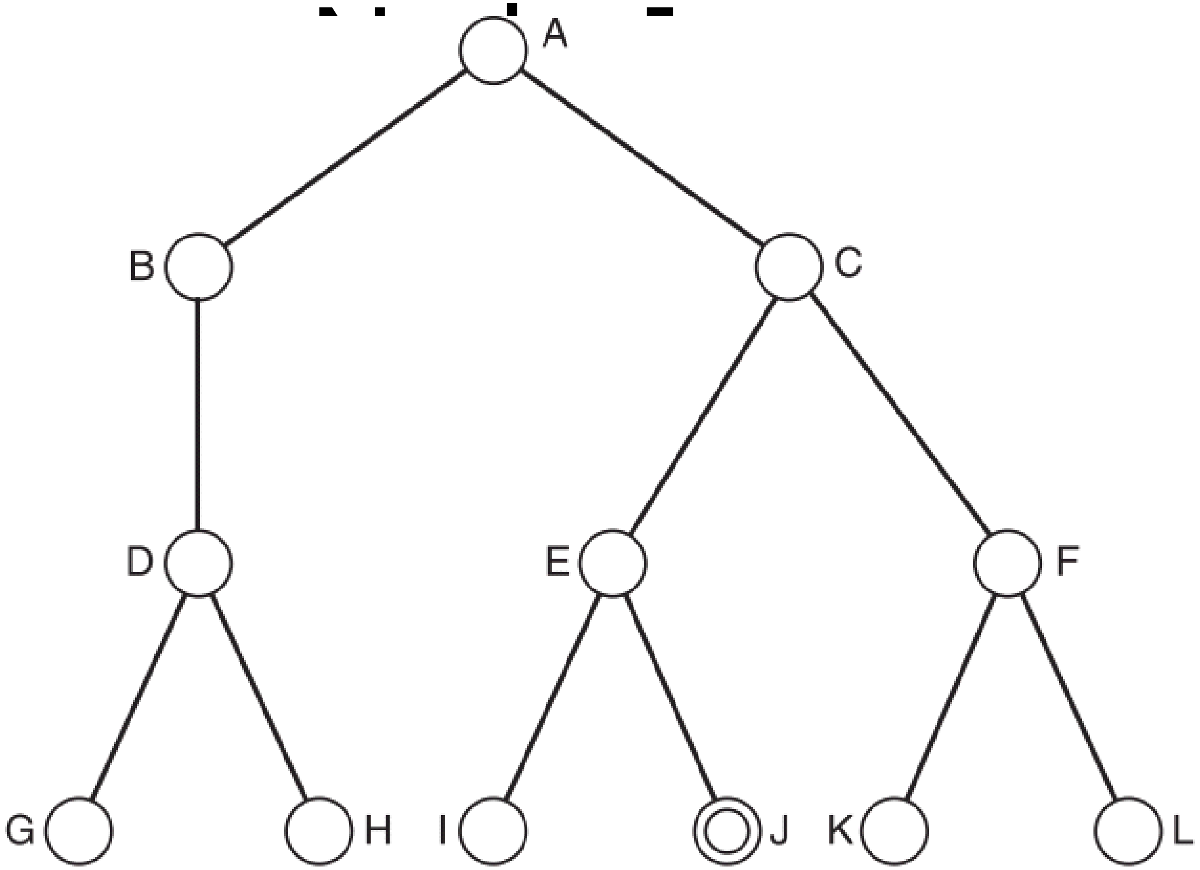
Current

A

Waiting



Breadth-First Search Goal -

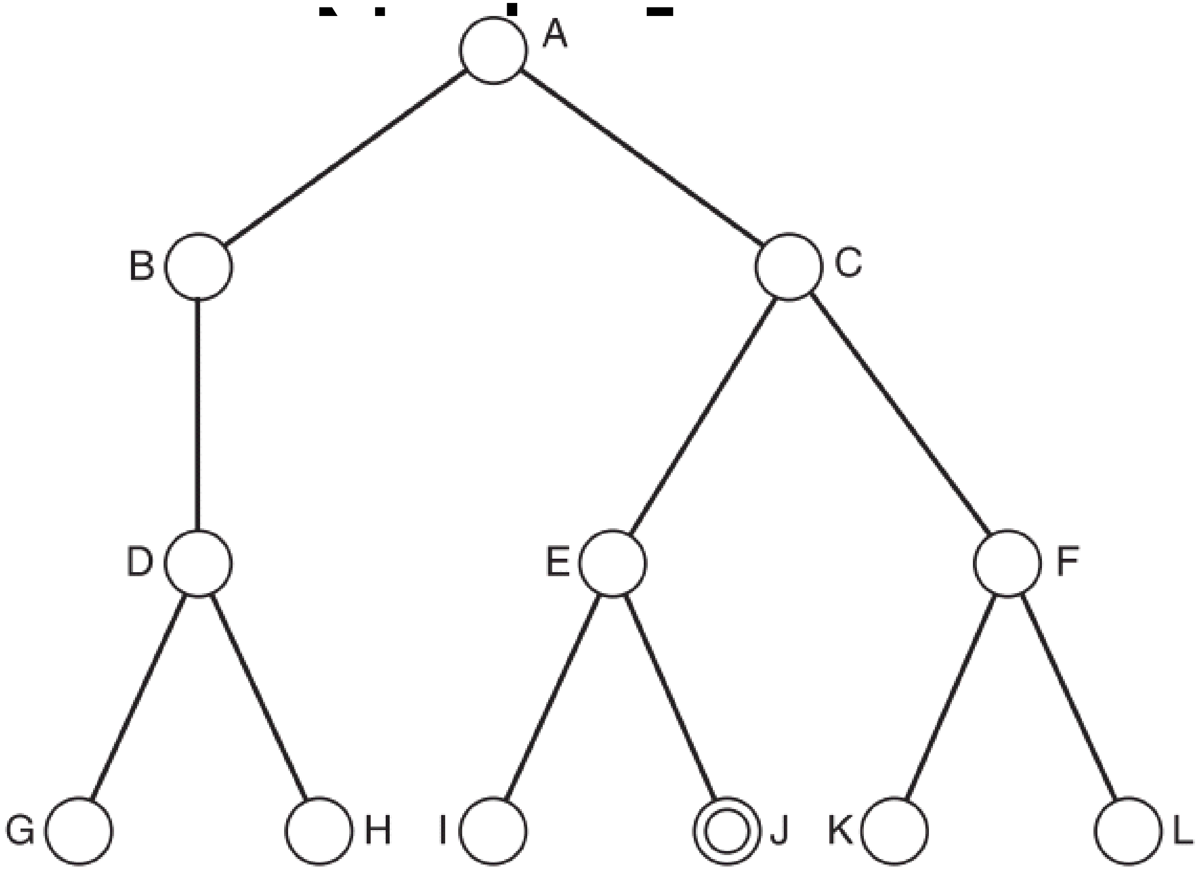


Current

A

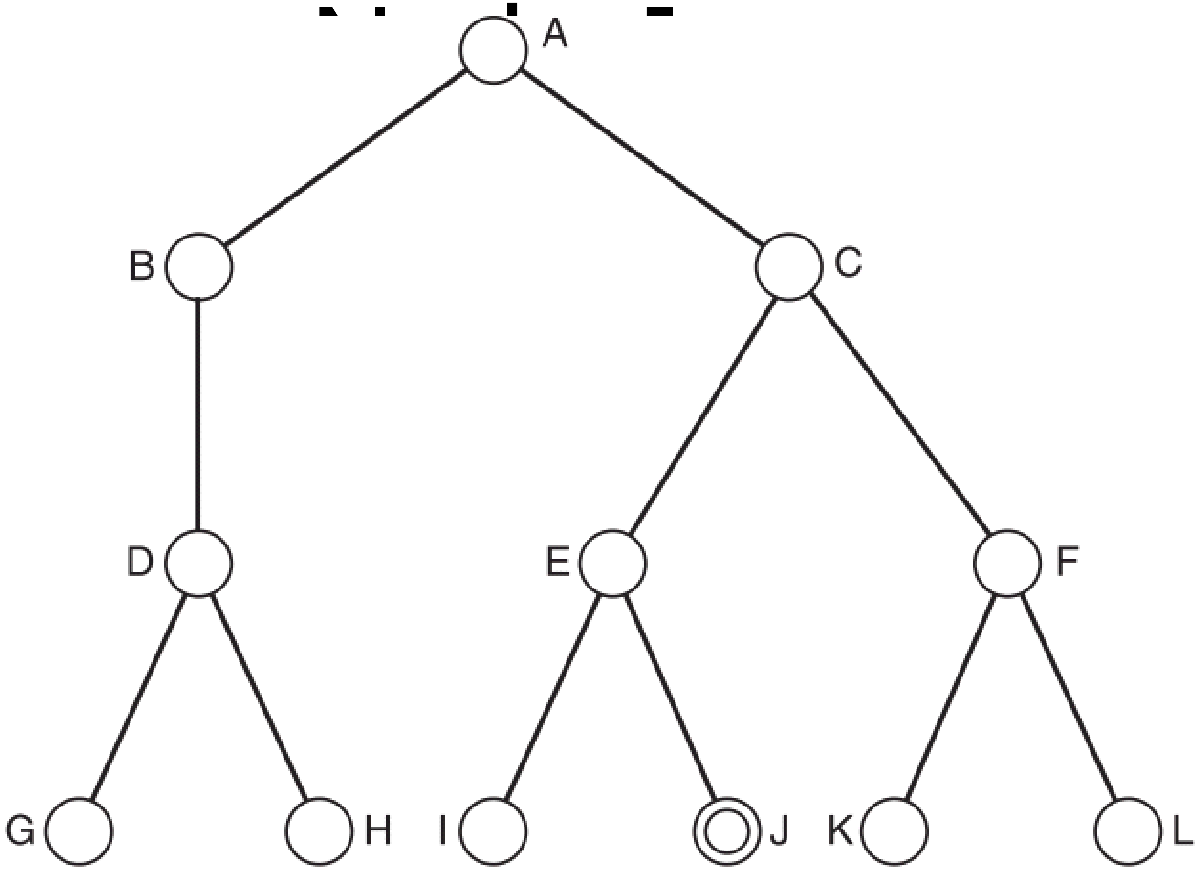
Waiting

Breadth-First Search Goal -



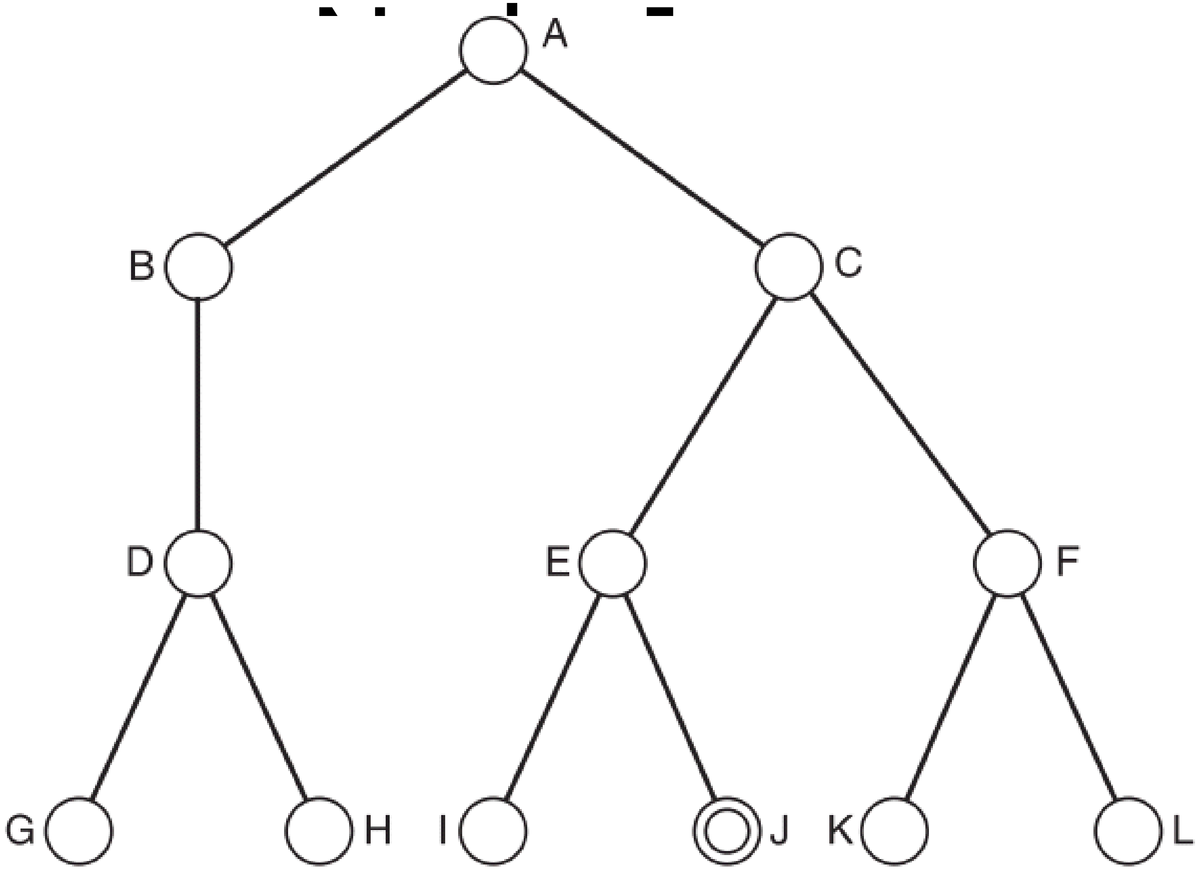
Current	Waiting
A	---
A	

Breadth-First Search Goal -



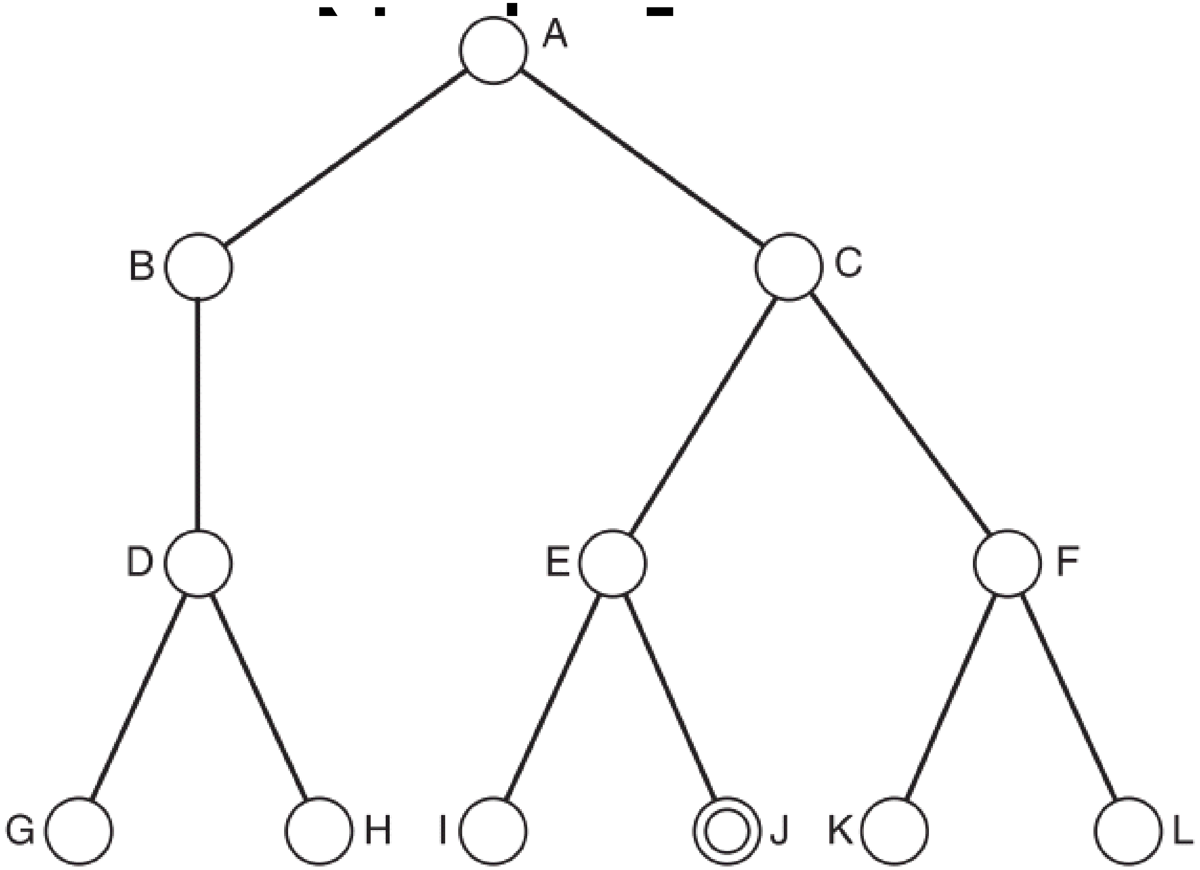
Current	Waiting
A	---
A	B, C

Breadth-First Search Goal -



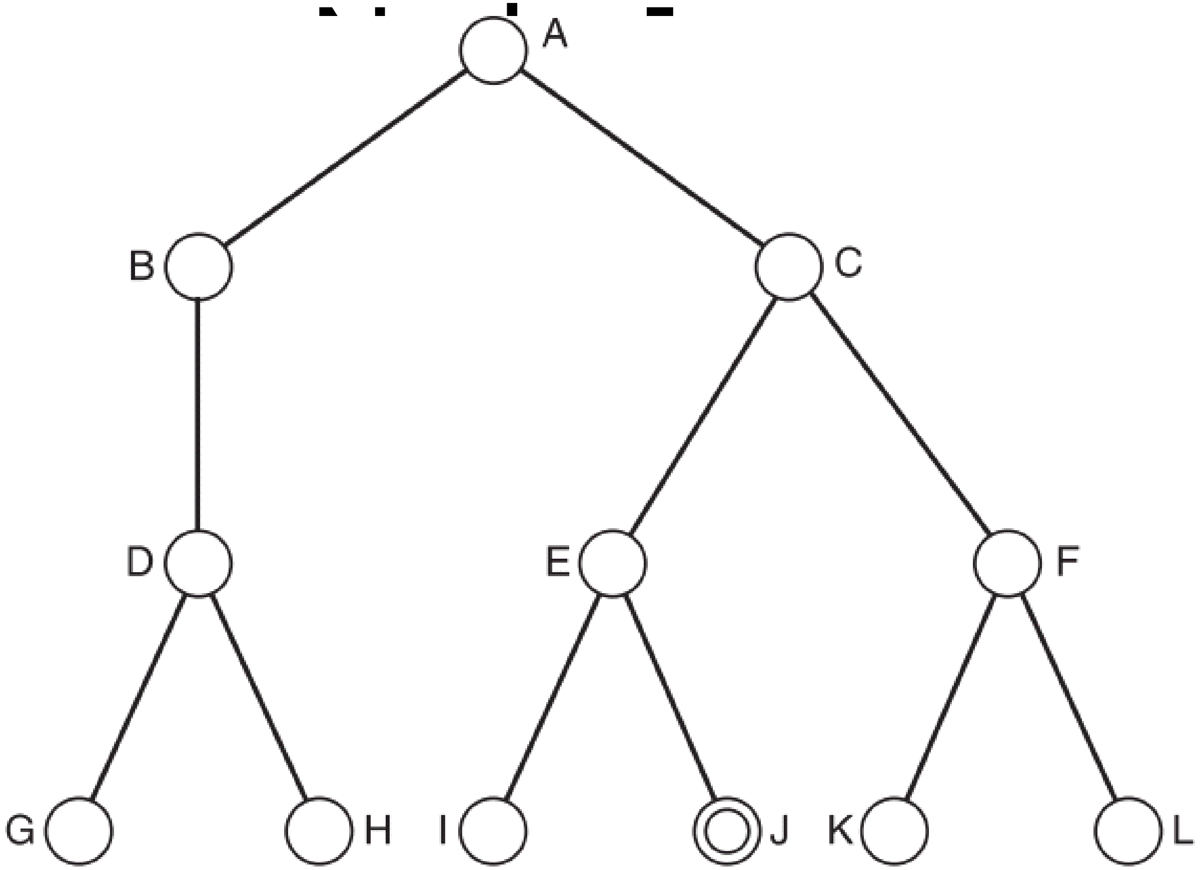
Current	Waiting
A	---
A	B, C
B	

Breadth-First Search Goal -



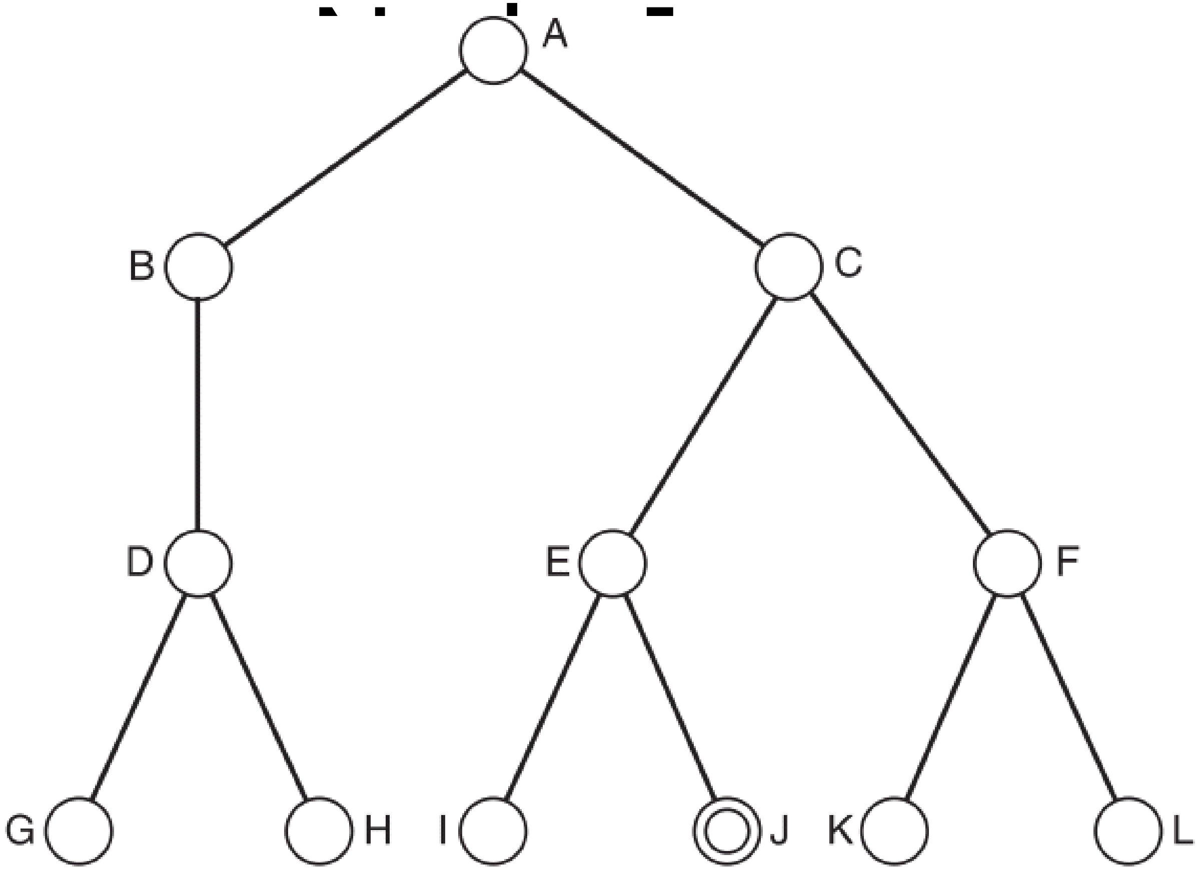
Current	Waiting
A	---
A	B, C
B	C

Breadth-First Search Goal -



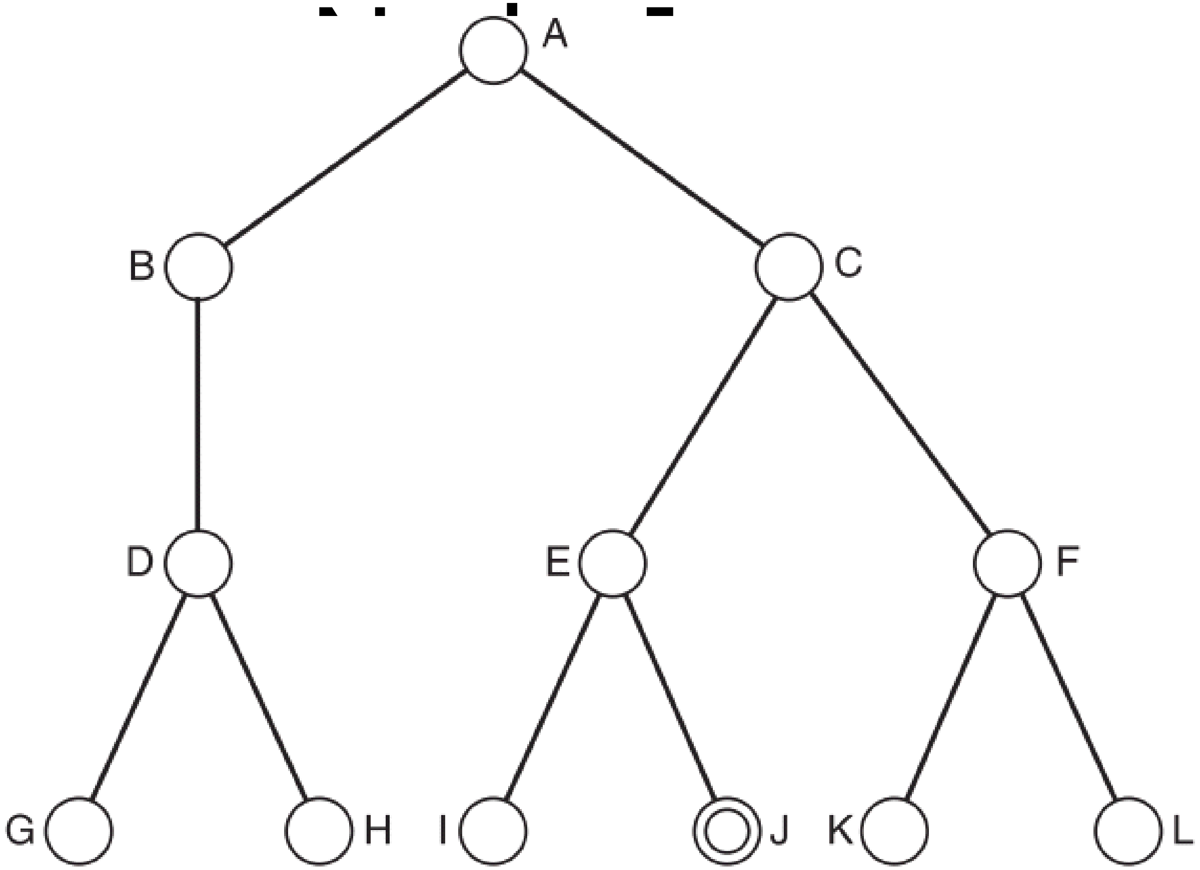
Current	Waiting
A	---
A	B, C
B	C

Breadth-First Search Goal -



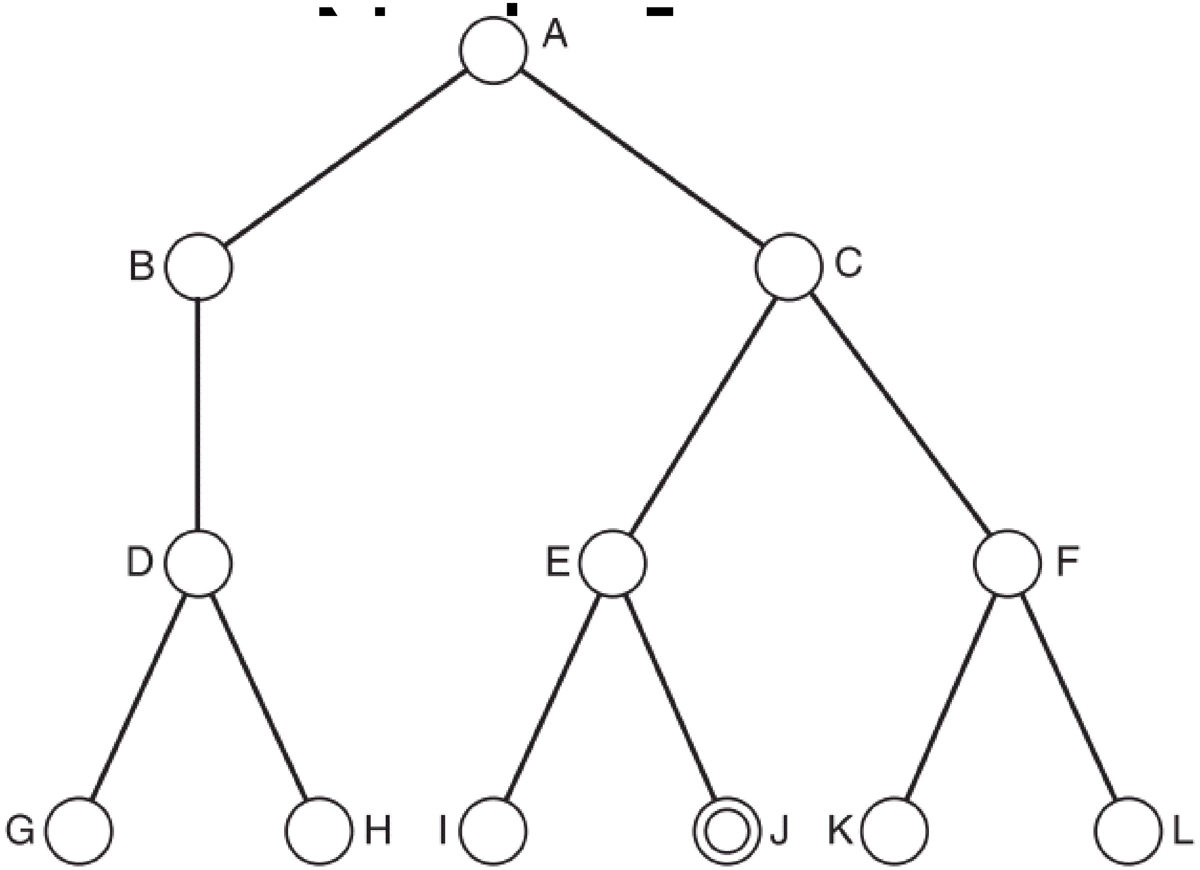
Current	Waiting
A	---
A	B, C
B	C
B	

Breadth-First Search Goal -



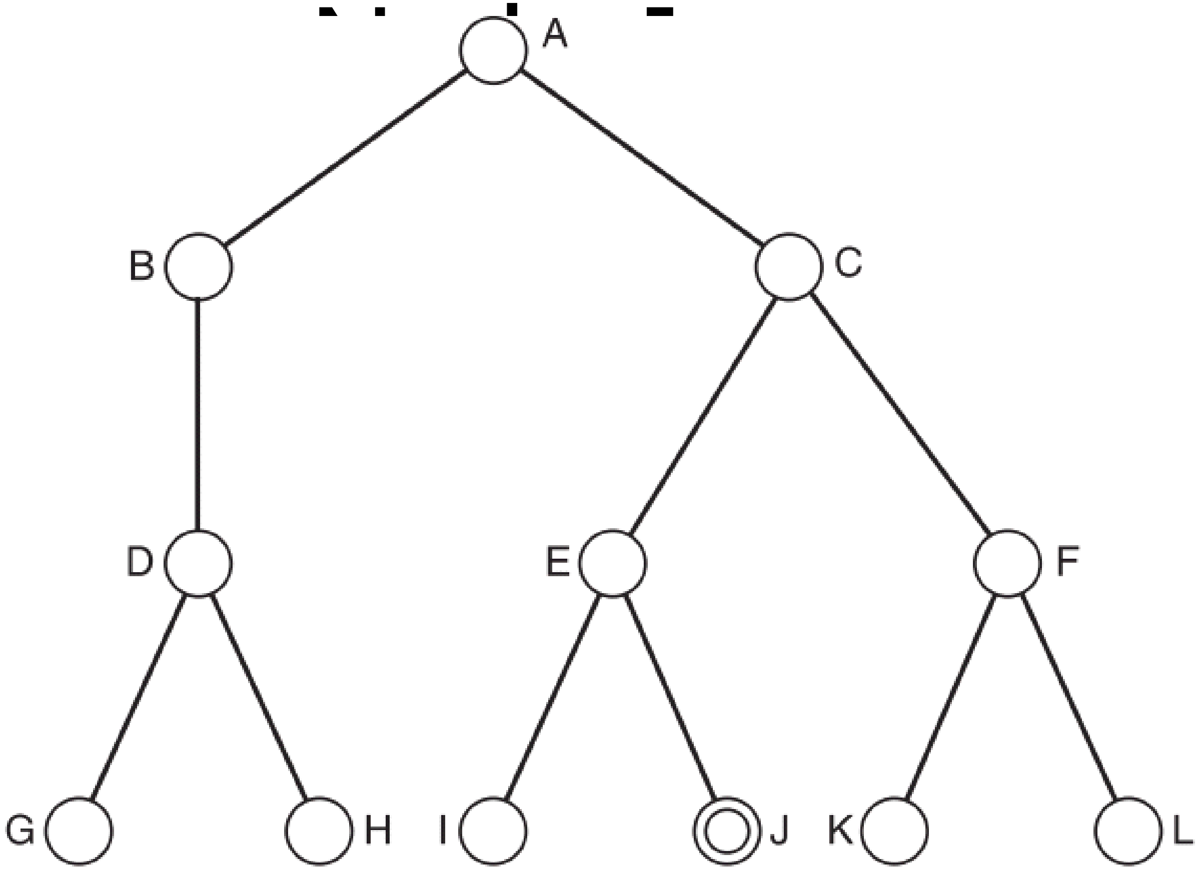
Current	Waiting
A	---
A	B, C
B	C
B	C, D

Breadth-First Search Goal -



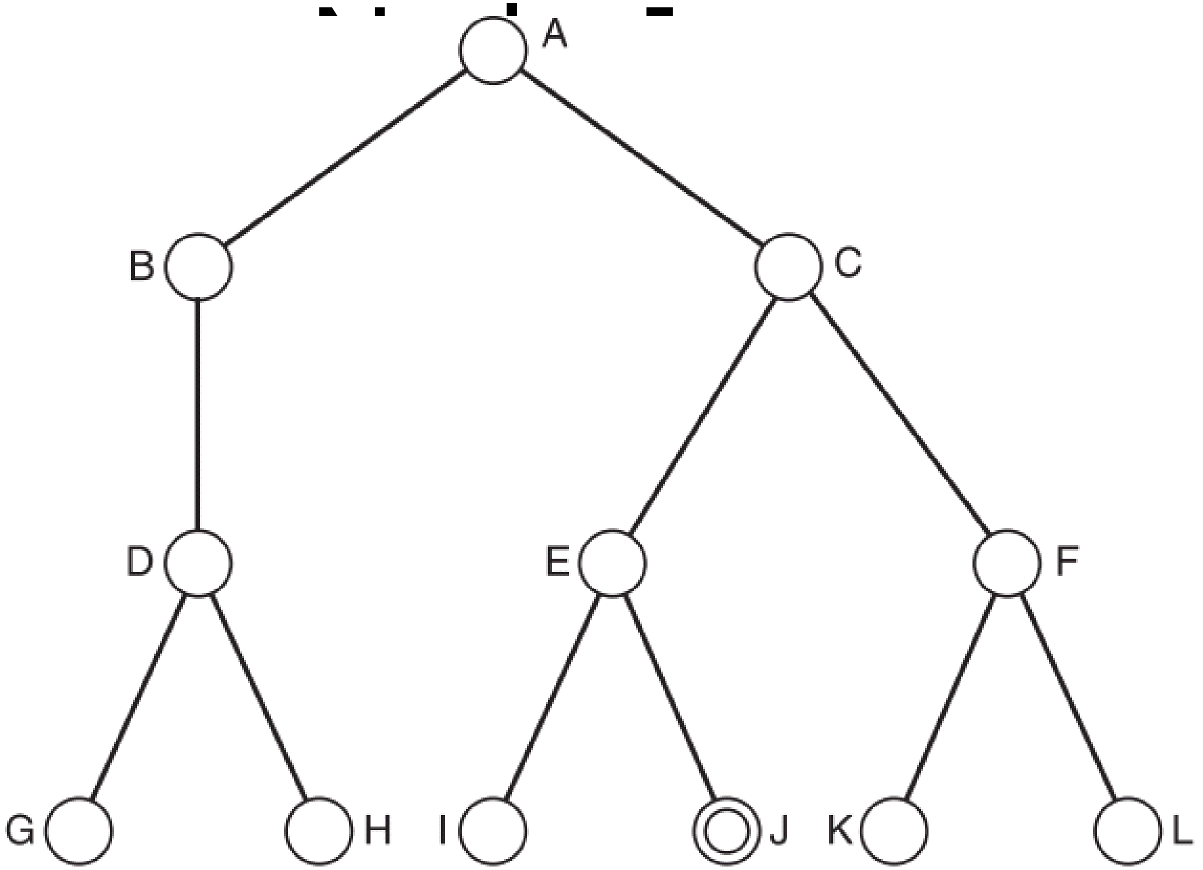
Current	Waiting
A	---
A	B, C
B	C
B	C, D
C	

Breadth-First Search Goal -



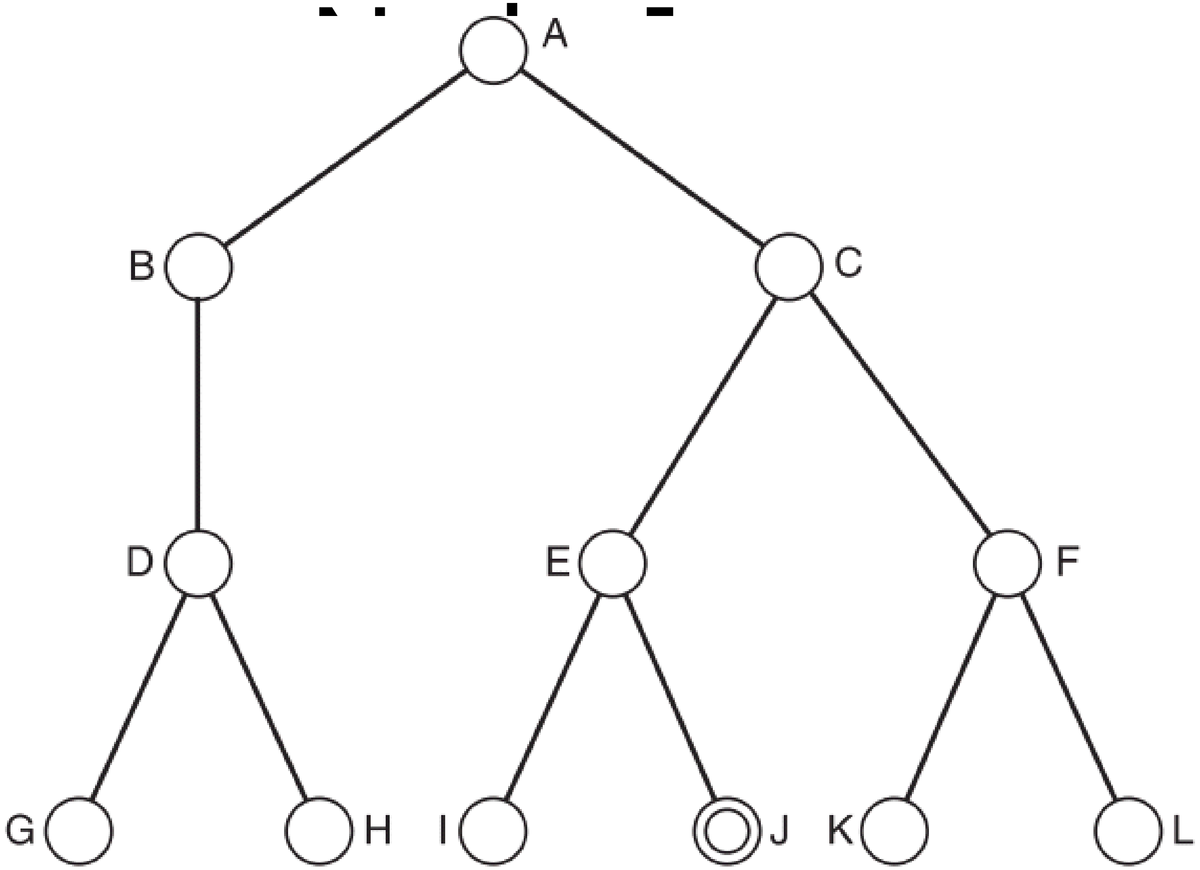
Current	Waiting
A	---
A	B, C
B	C
B	C, D
C	D

Breadth-First Search Goal -



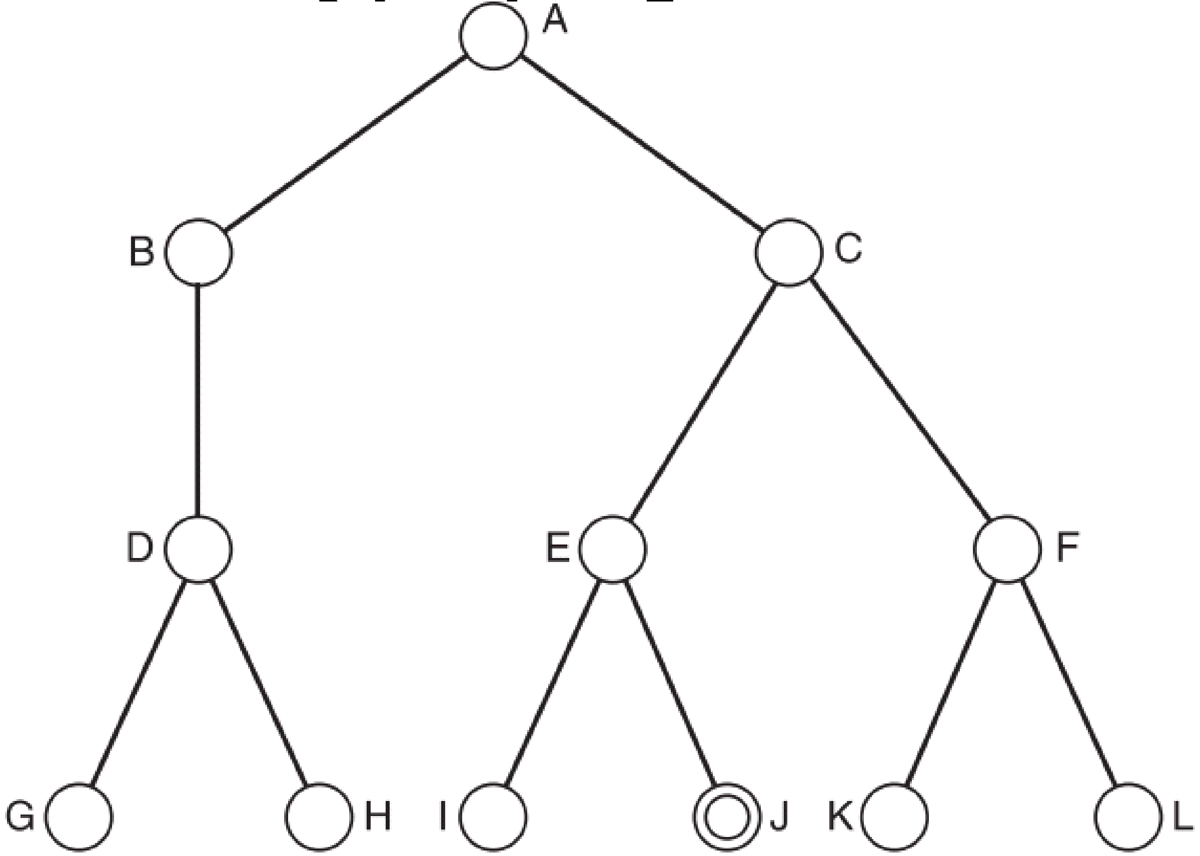
Current	Waiting
A	---
A	B, C
B	C
B	C, D
C	D

Breadth-First Search Goal -



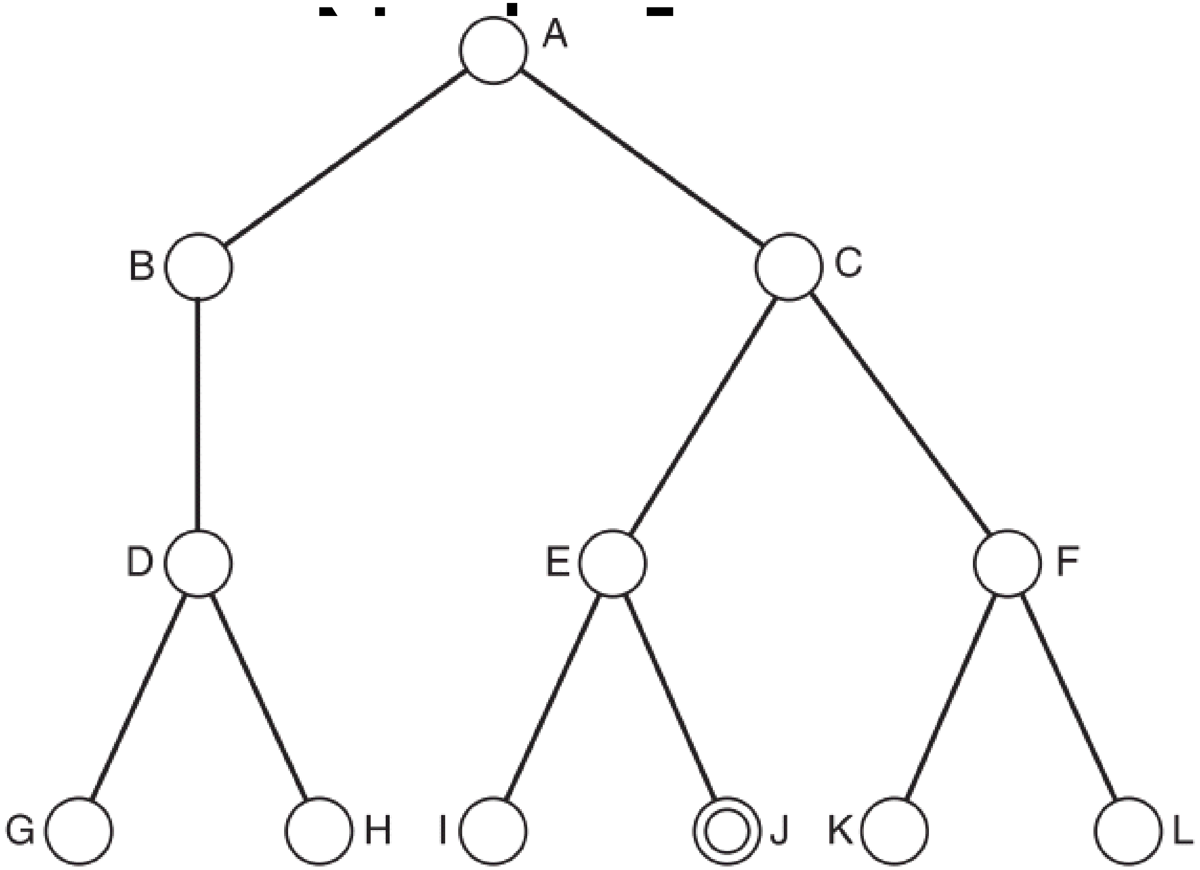
Current	Waiting
A	---
A	B, C
B	C
B	C, D
C	D
C	

Breadth-First Search Goal -



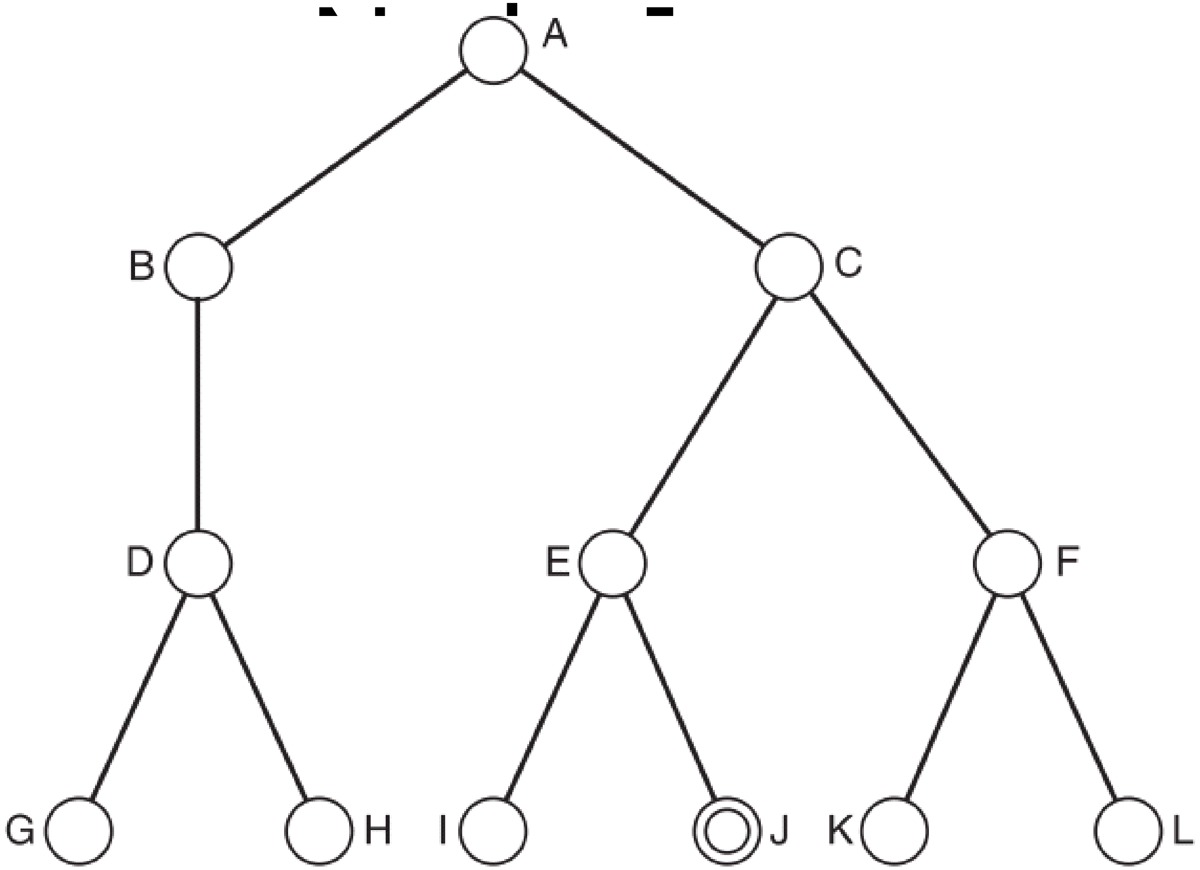
Current	Waiting
A	---
A	B, C
B	C
B	C, D
C	D
C	D, E, F

Breadth-First Search Goal -



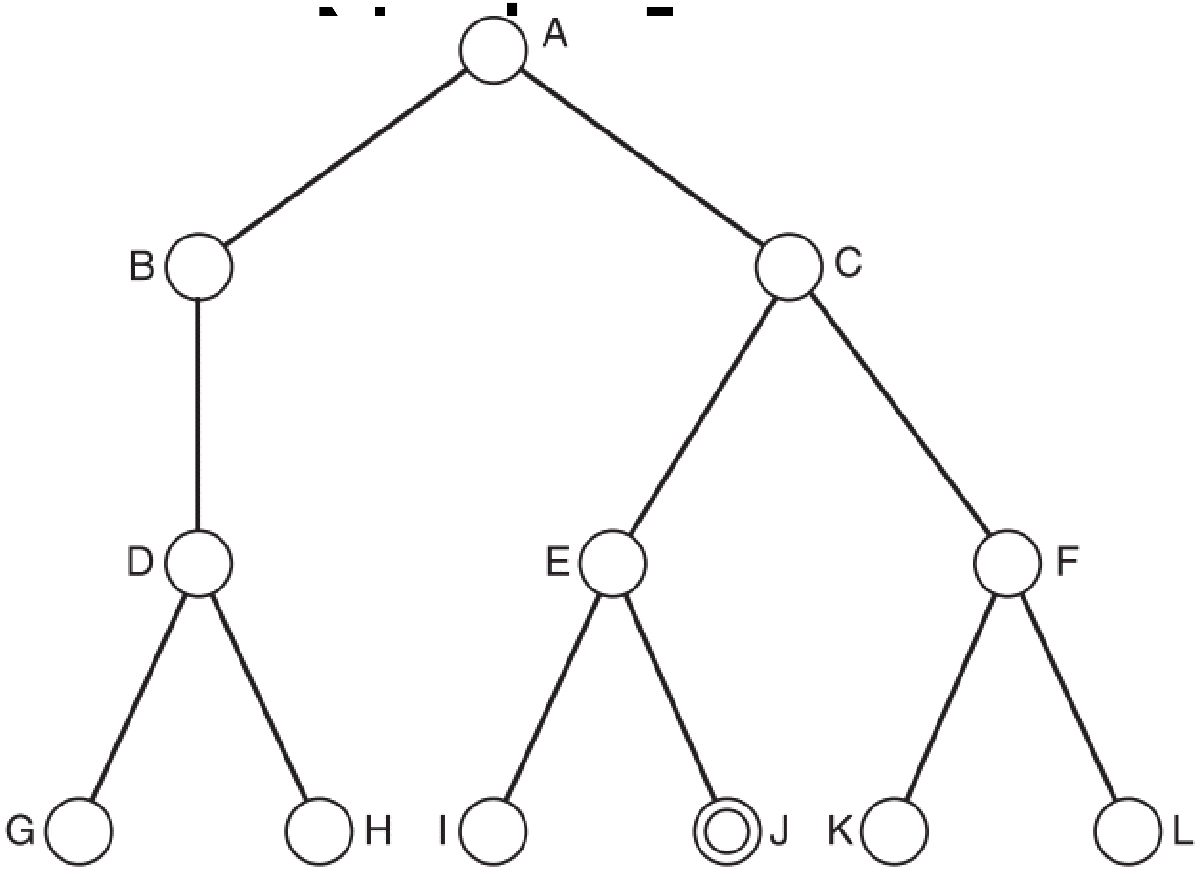
Current	Waiting
A	---
A	B, C
B	C
B	C, D
C	D
C	D, E, F
D	

Breadth-First Search Goal -



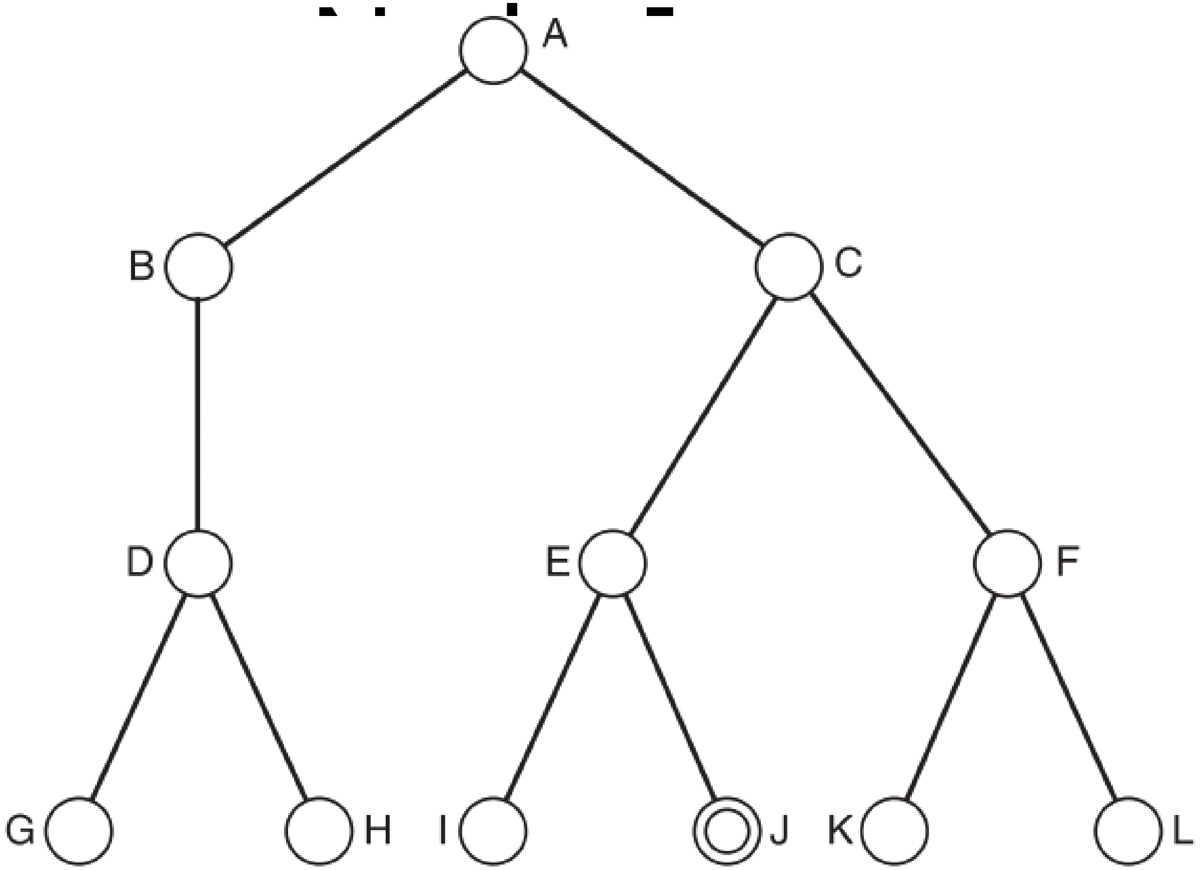
Current	Waiting
A	---
A	B, C
B	C
B	C, D
C	D
C	D, E, F
D	E, F

Breadth-First Search Goal -



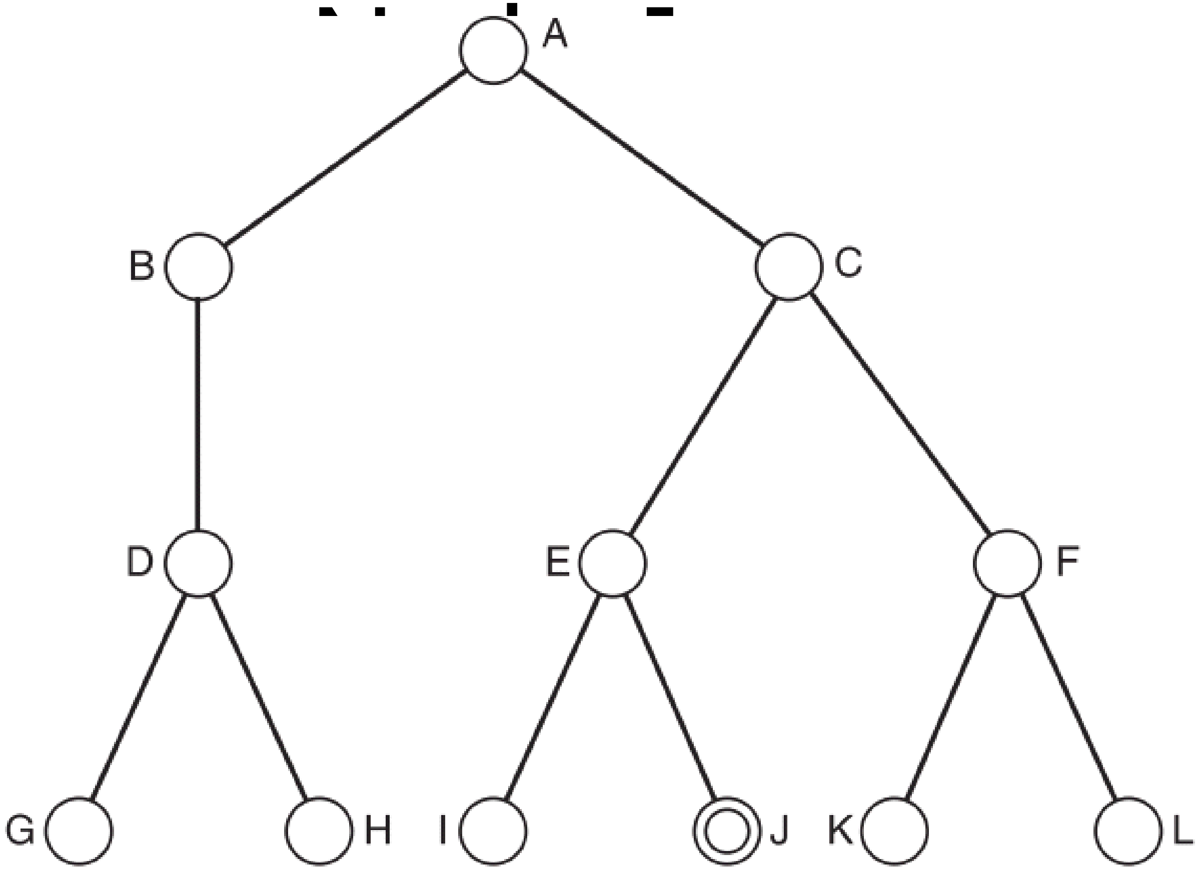
Current	Waiting
A	---
A	B, C
B	C
B	C, D
C	D
C	D, E, F
D	E, F

Breadth-First Search Goal -



Current	Waiting
A	---
A	B, C
B	C
B	C, D
C	D
C	D, E, F
D	E, F
D	

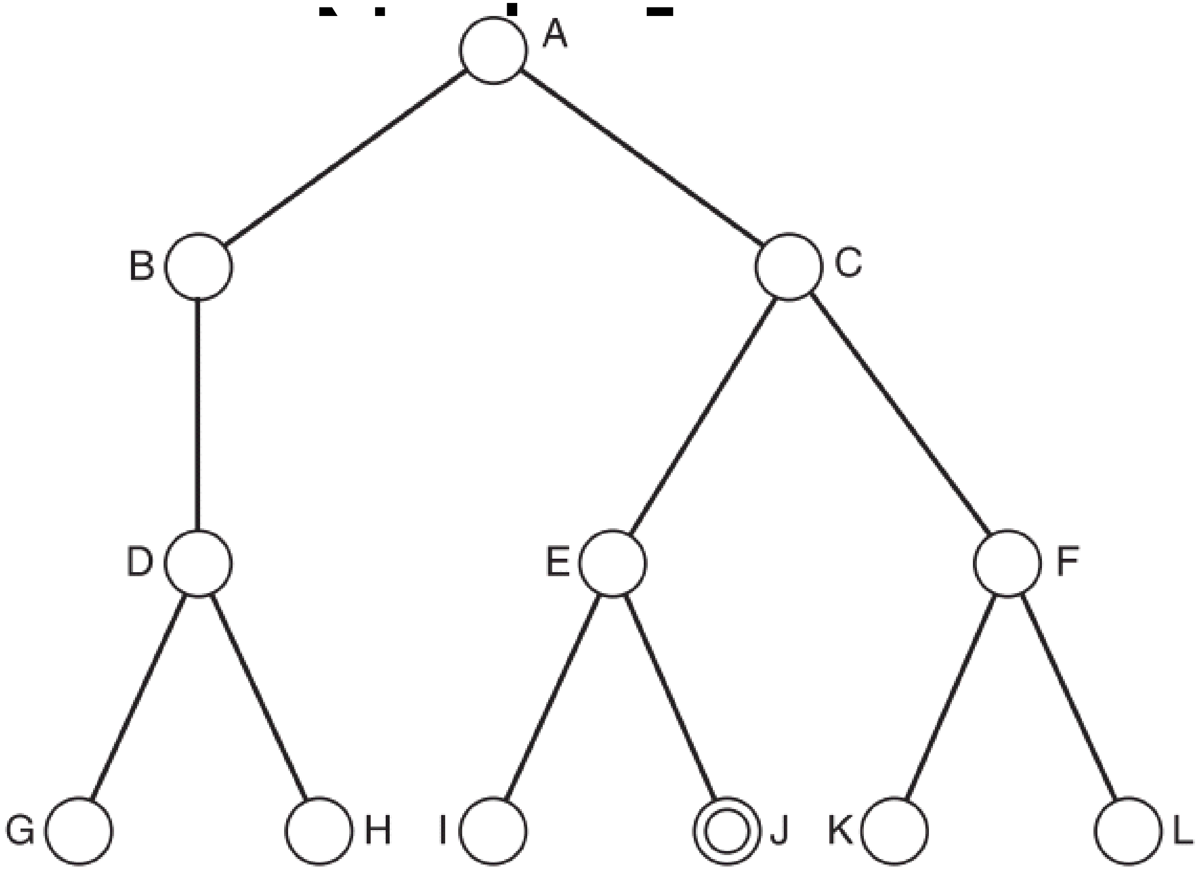
Breadth-First Search Goal -



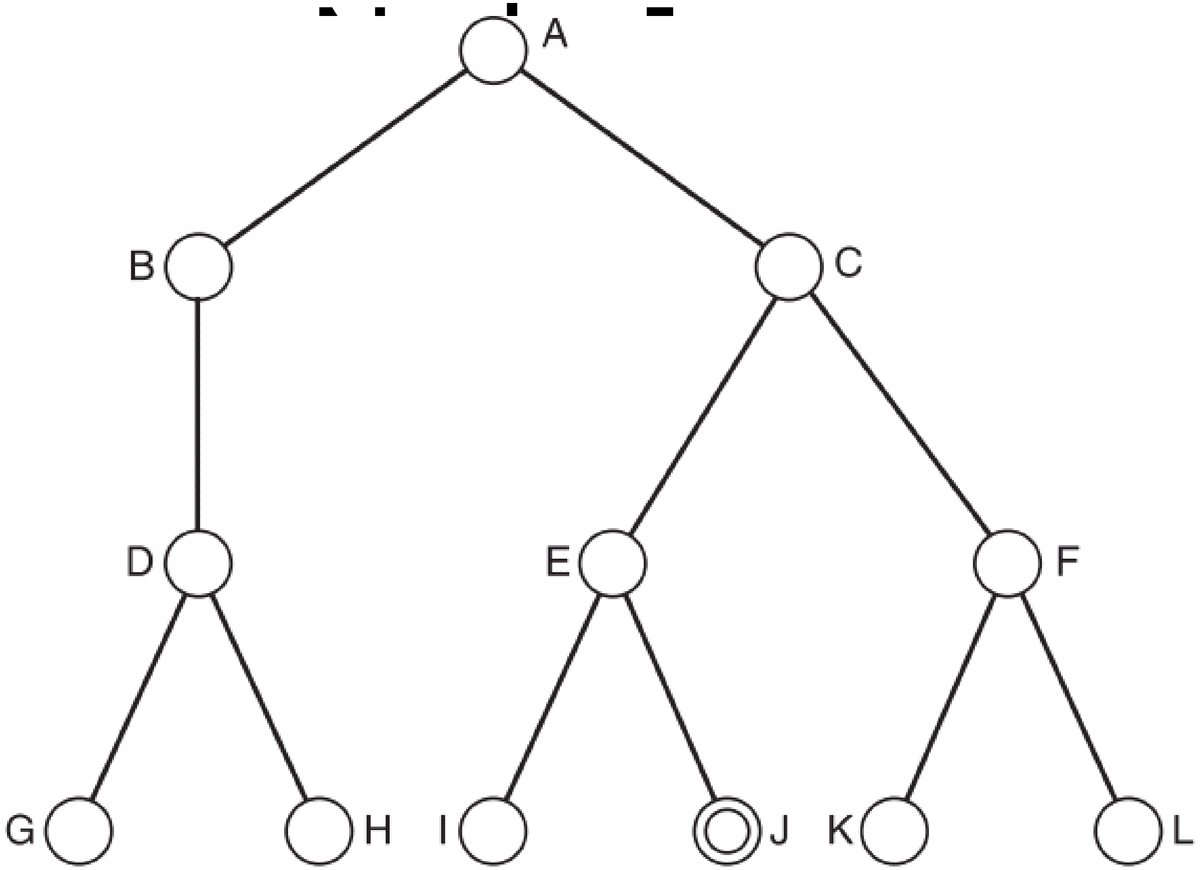
Current	Waiting
A	---
A	B, C
B	C
B	C, D
C	D
C	D, E, F
D	E, F
D	E, F, G,

Breadth-First Search Goal -

Current	Waiting
D	E, F, G, H

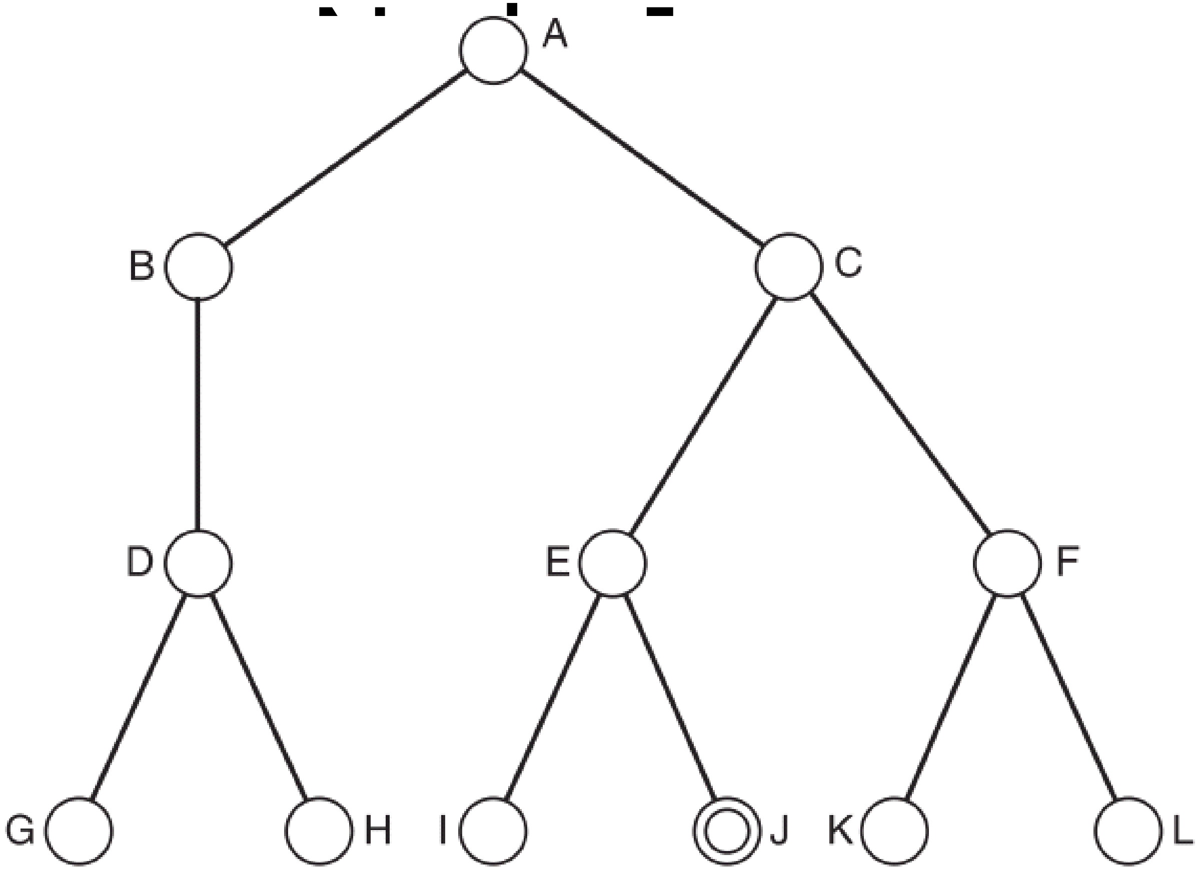


Breadth-First Search Goal -



Current	Waiting
D	E, F, G, H
E	

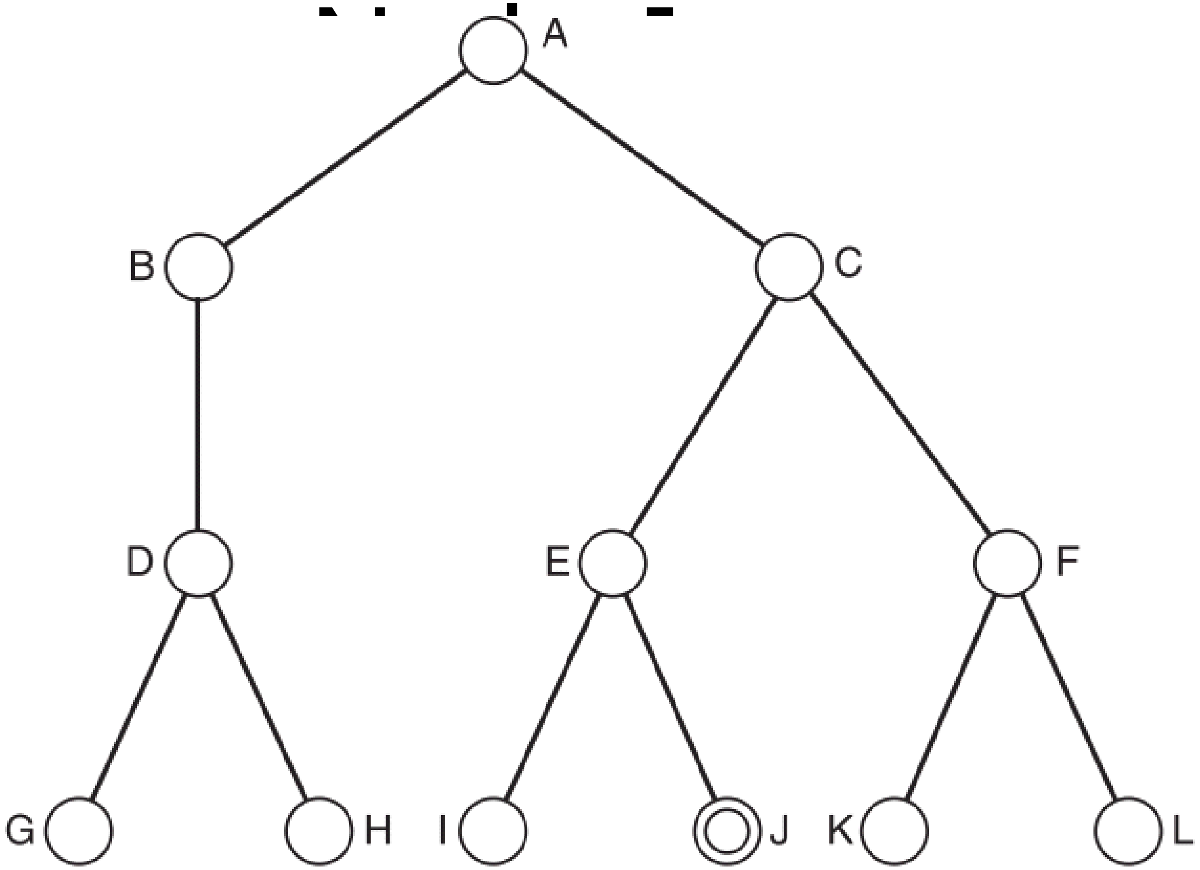
Breadth-First Search Goal -



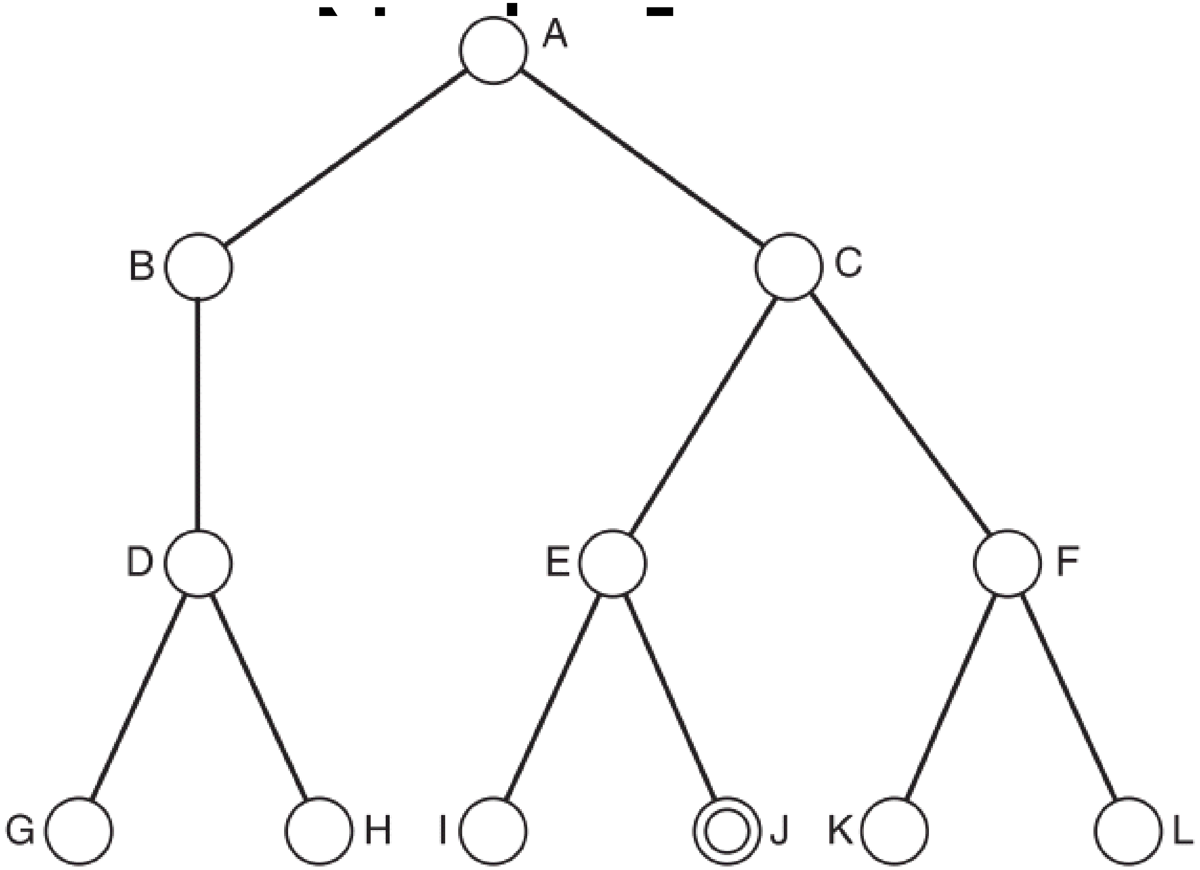
Current	Waiting
D	E, F, G, H
E	F, G, H

Breadth-First Search Goal -

Current	Waiting
D	E, F, G, H
E	F, G, H

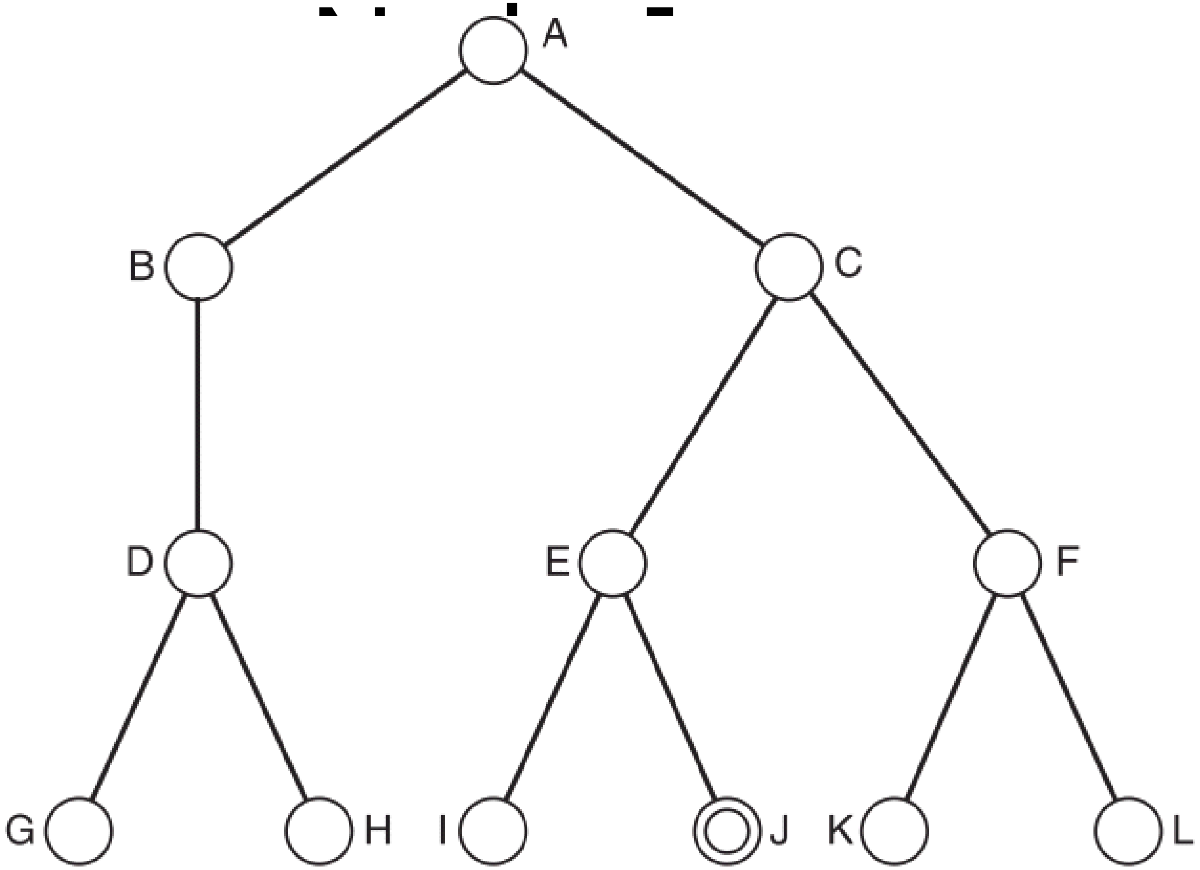


Breadth-First Search Goal -



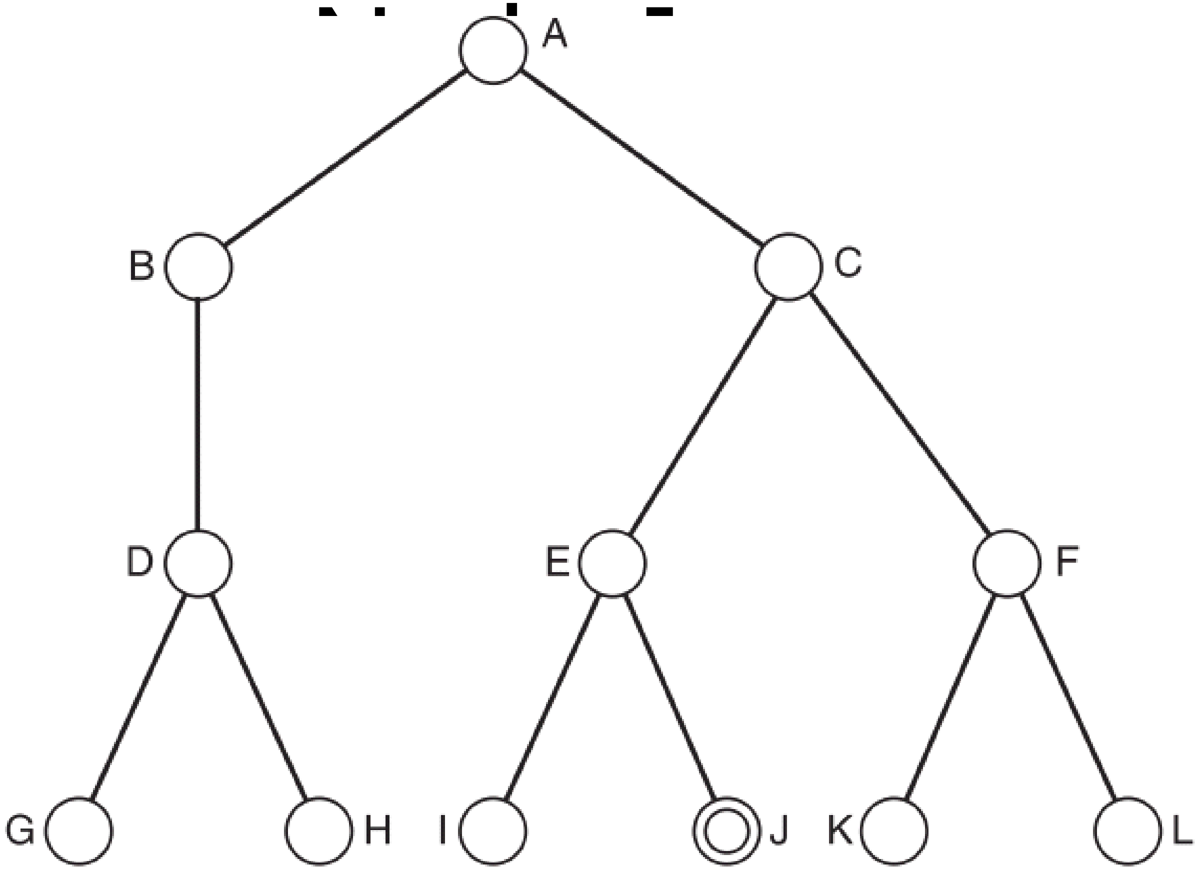
Current	Waiting
D	E, F, G, H
E	F, G, H
E	

Breadth-First Search Goal -



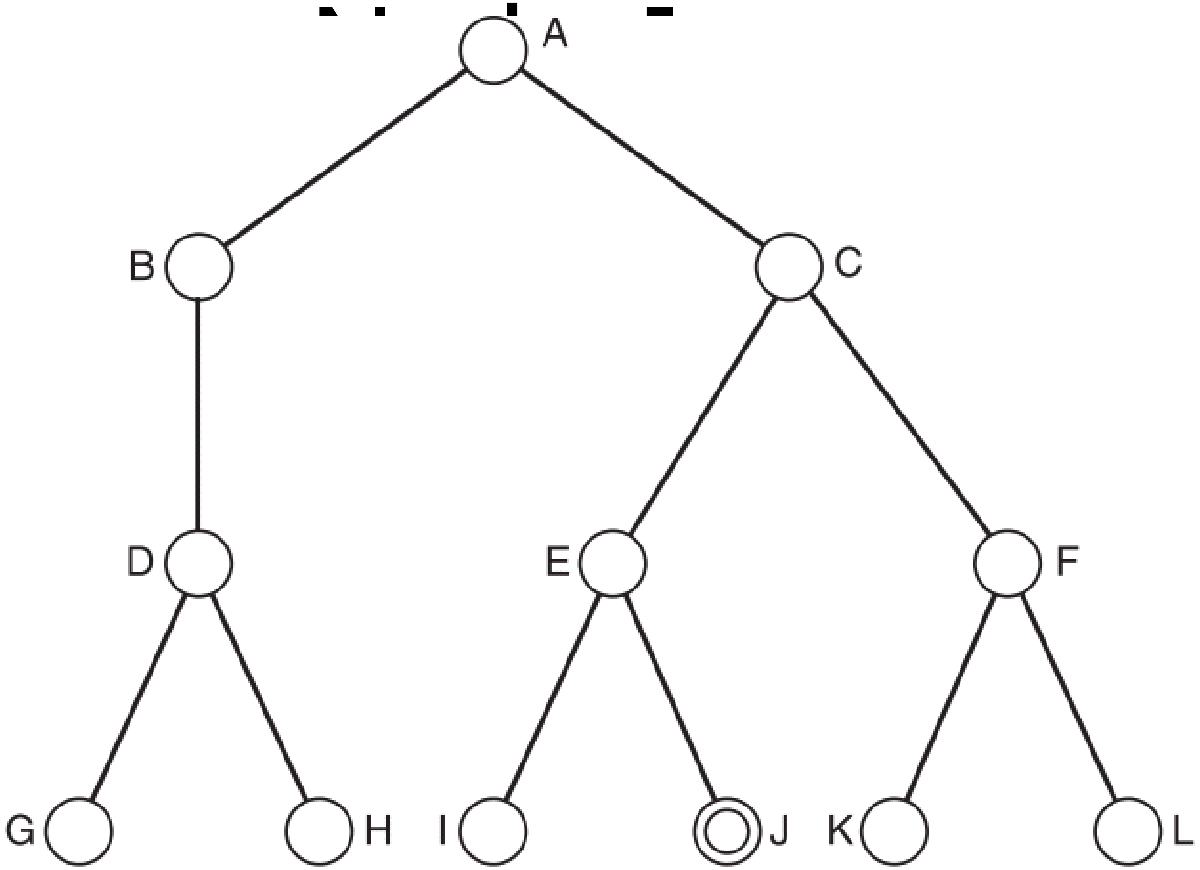
Current	Waiting
D	E, F, G, H
E	F, G, H
E	F, G, H, I, J

Breadth-First Search Goal -



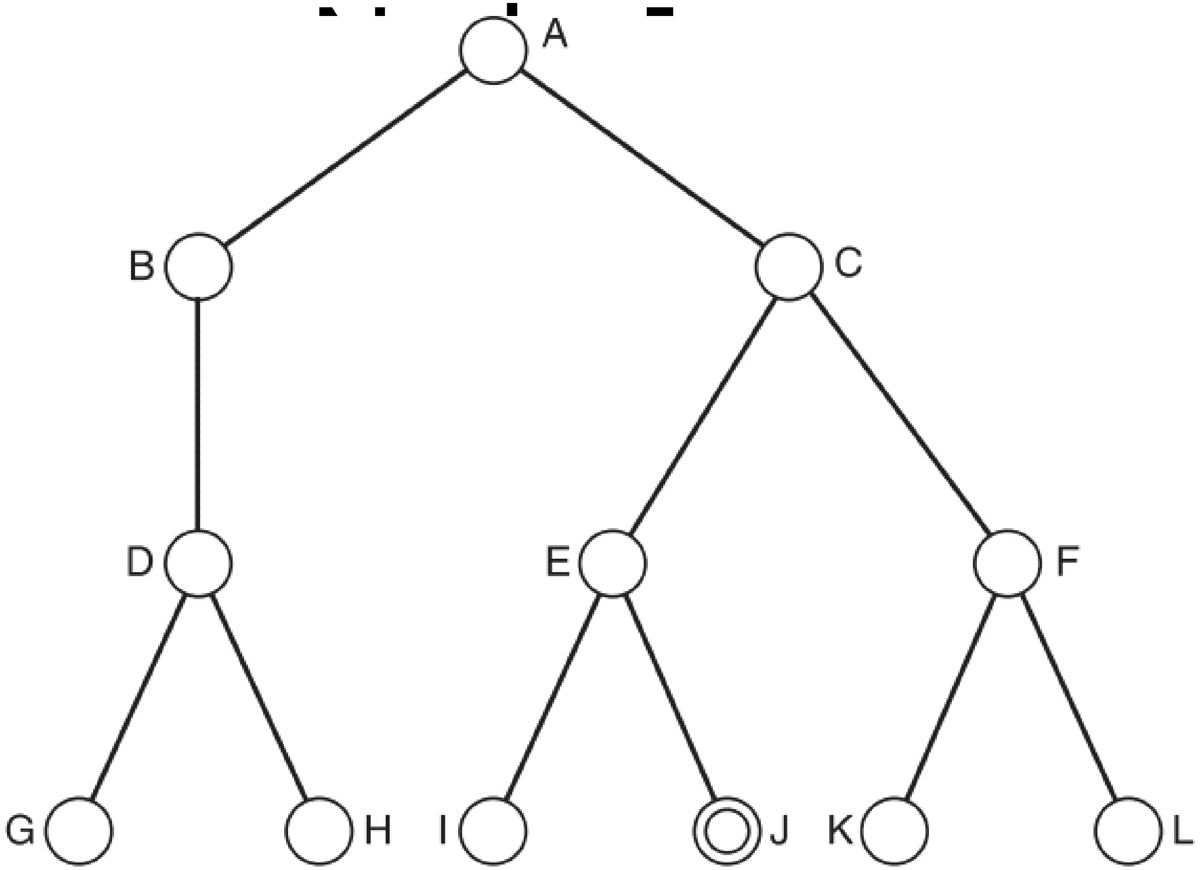
Current	Waiting
D	E, F, G, H
E	F, G, H
E	F, G, H, I, J
F	

Breadth-First Search Goal -



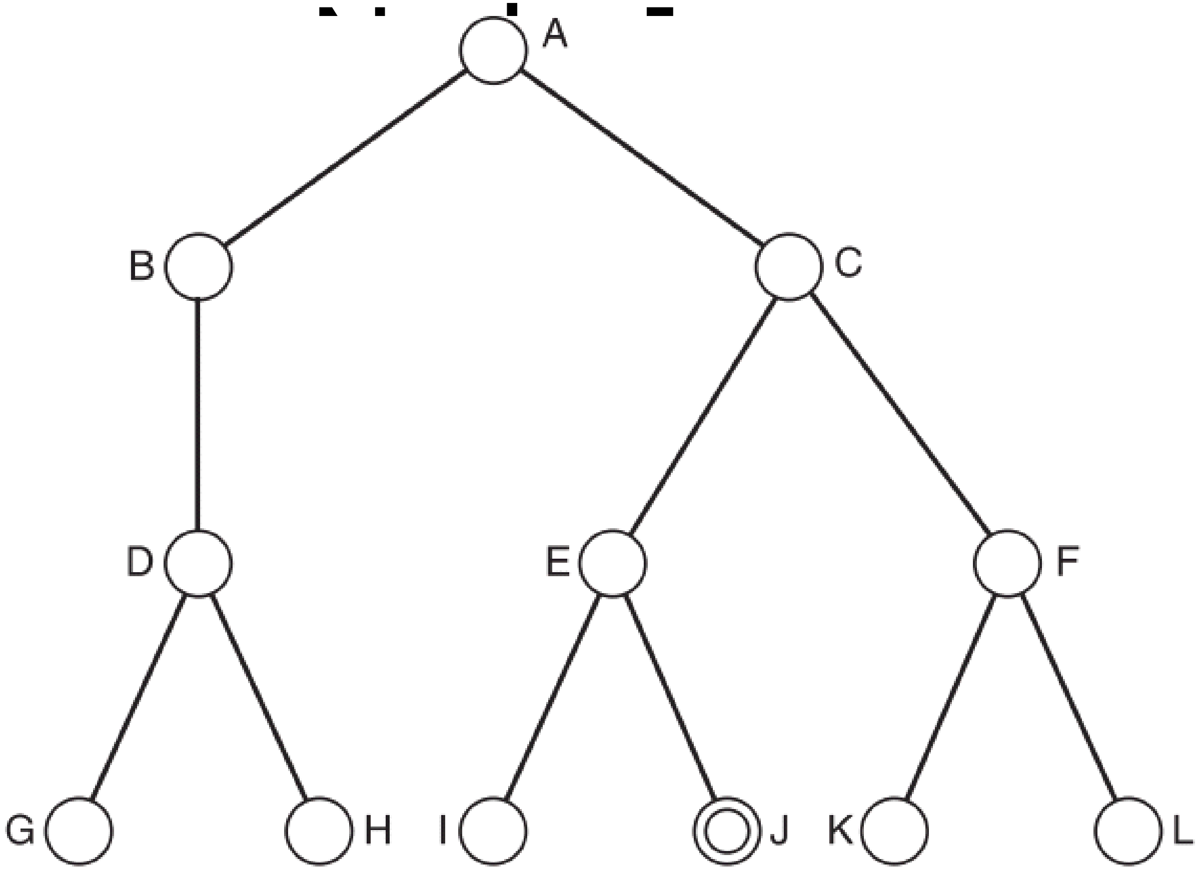
Current	Waiting
D	E, F, G, H
E	F, G, H
E	F, G, H, I, J
F	G, H, I, J

Breadth-First Search Goal -



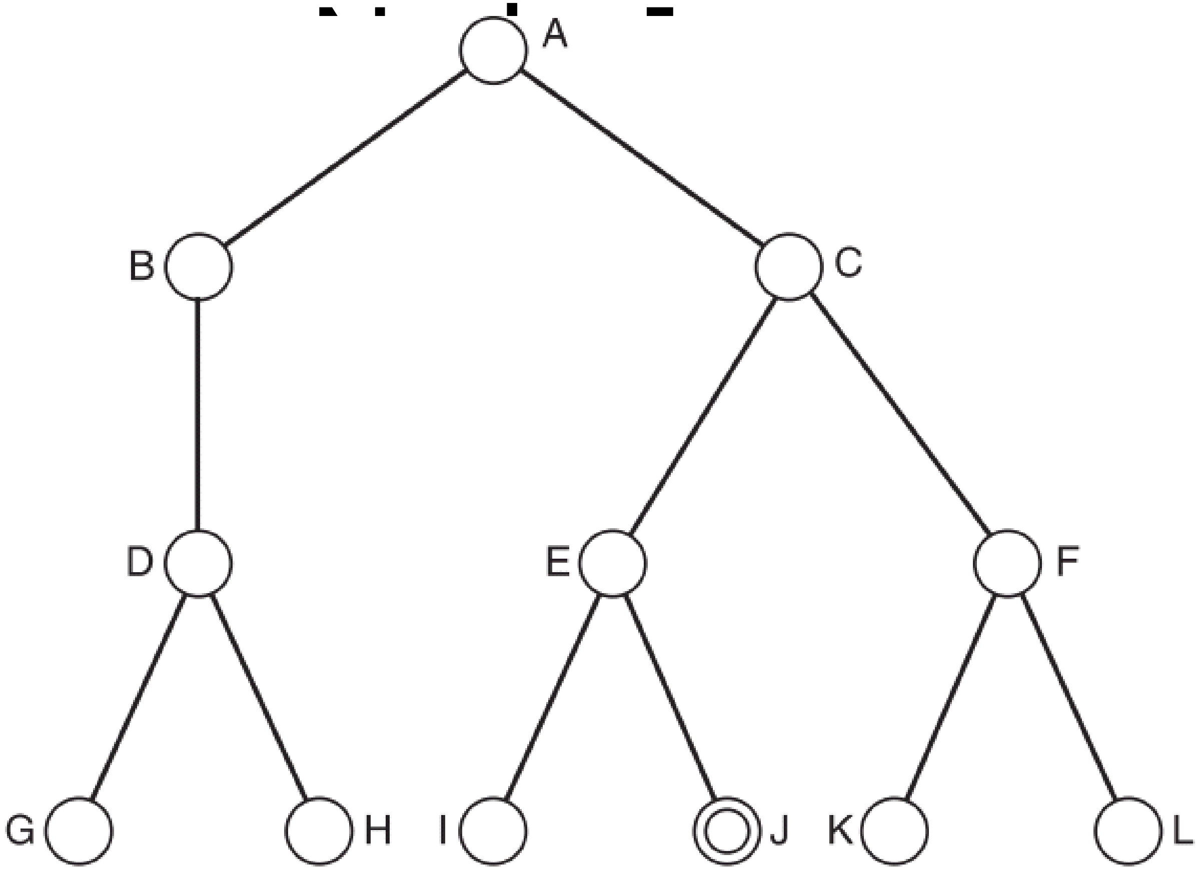
Current	Waiting
D	E, F, G, H
E	F, G, H
E	F, G, H, I, J
F	G, H, I, J

Breadth-First Search Goal -



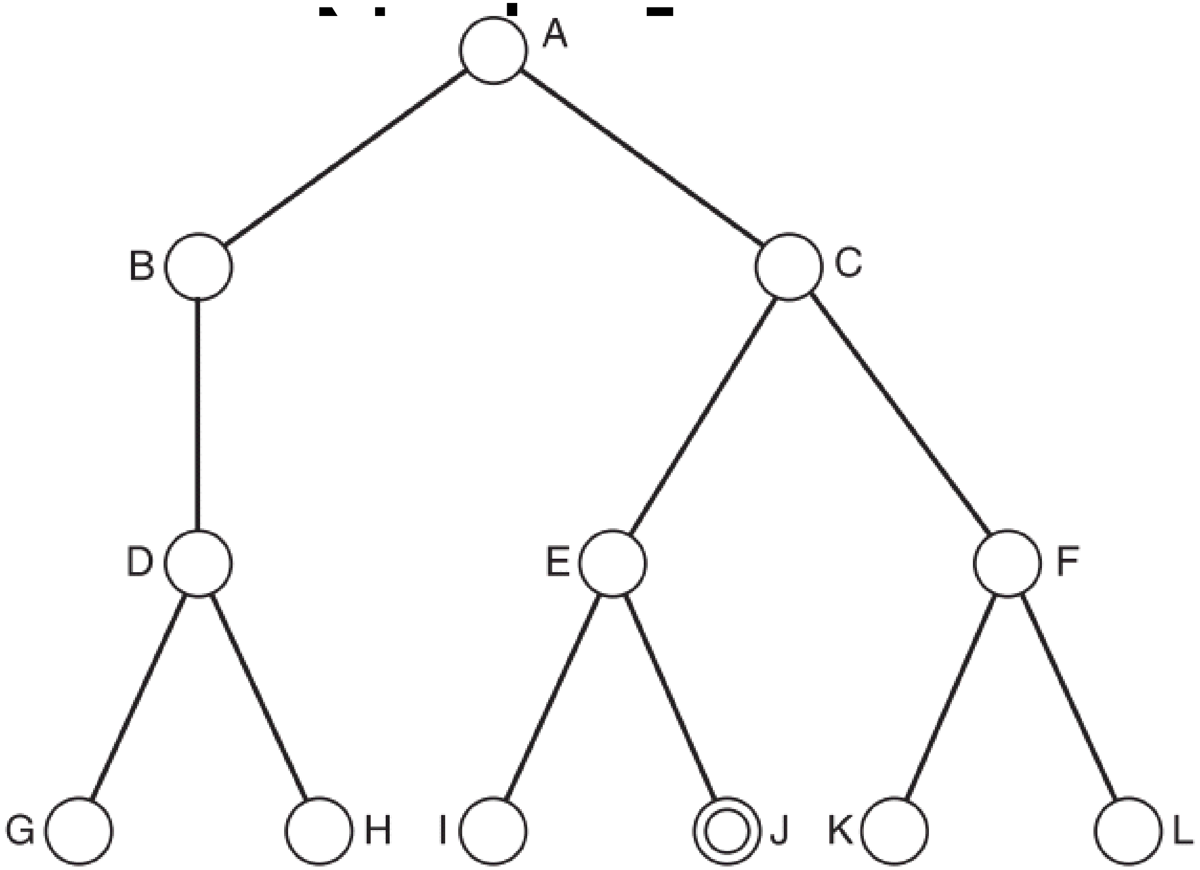
Current	Waiting
D	E, F, G, H
E	F, G, H
E	F, G, H, I, J
F	G, H, I, J
F	

Breadth-First Search Goal -



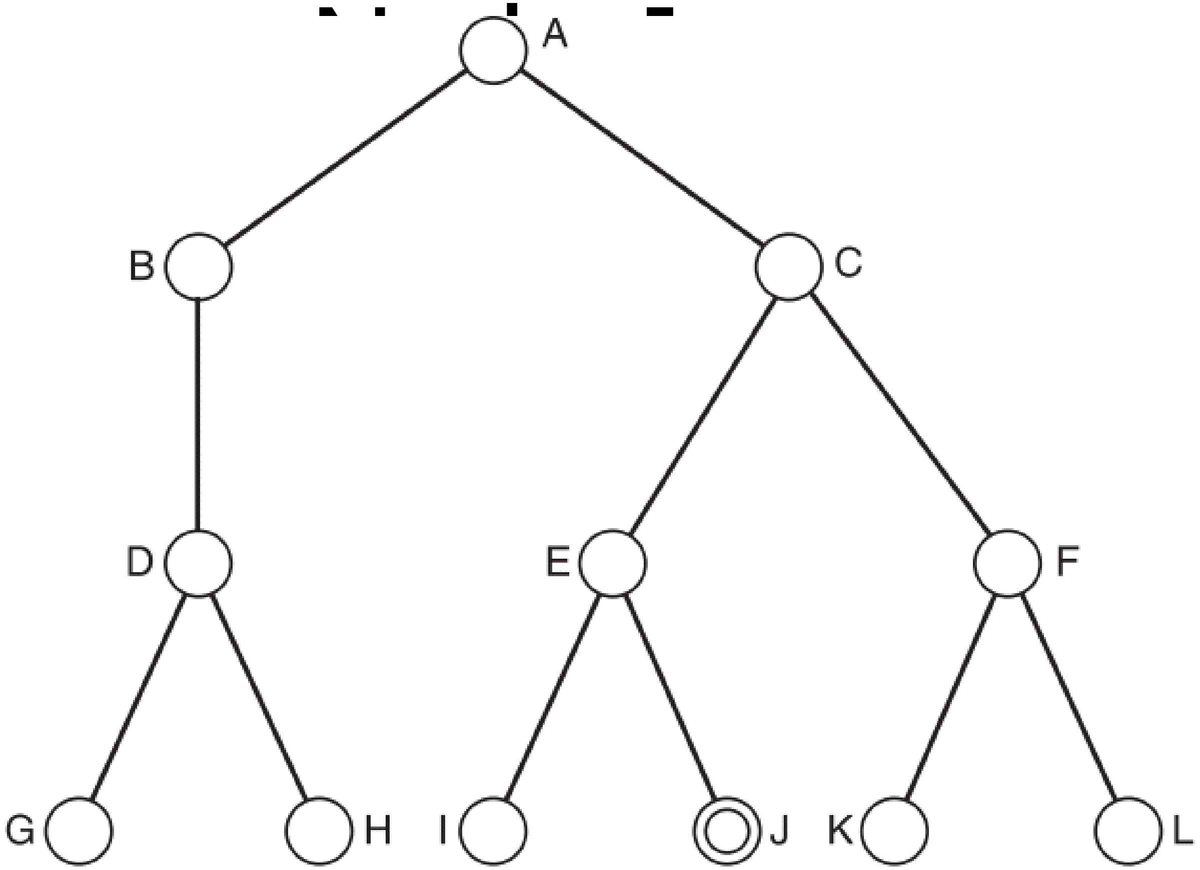
Current	Waiting
D	E, F, G, H
E	F, G, H
E	F, G, H, I, J
F	G, H, I, J
F	G, H, I, J, K, L

Breadth-First Search Goal -



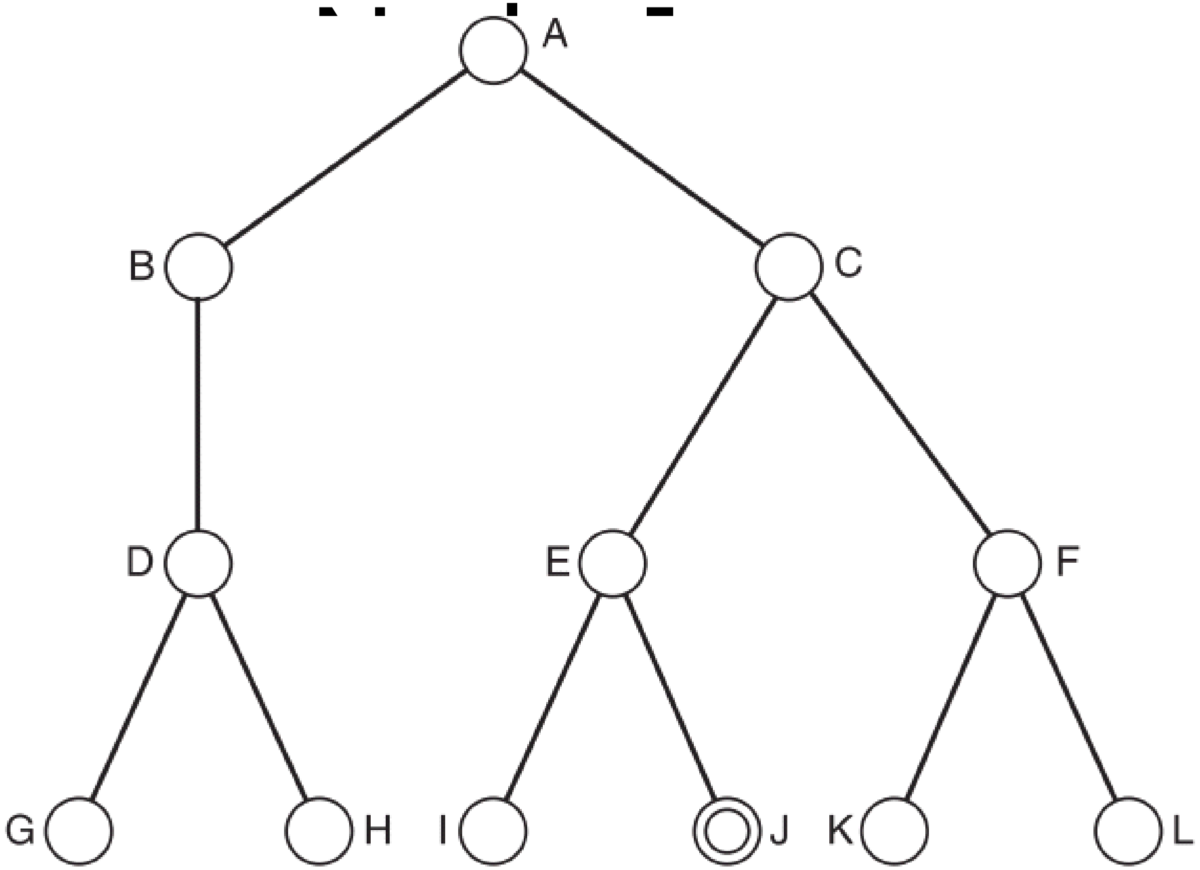
Current	Waiting
D	E, F, G, H
E	F, G, H
E	F, G, H, I, J
F	G, H, I, J
F	G, H, I, J, K, L
G	

Breadth-First Search Goal -



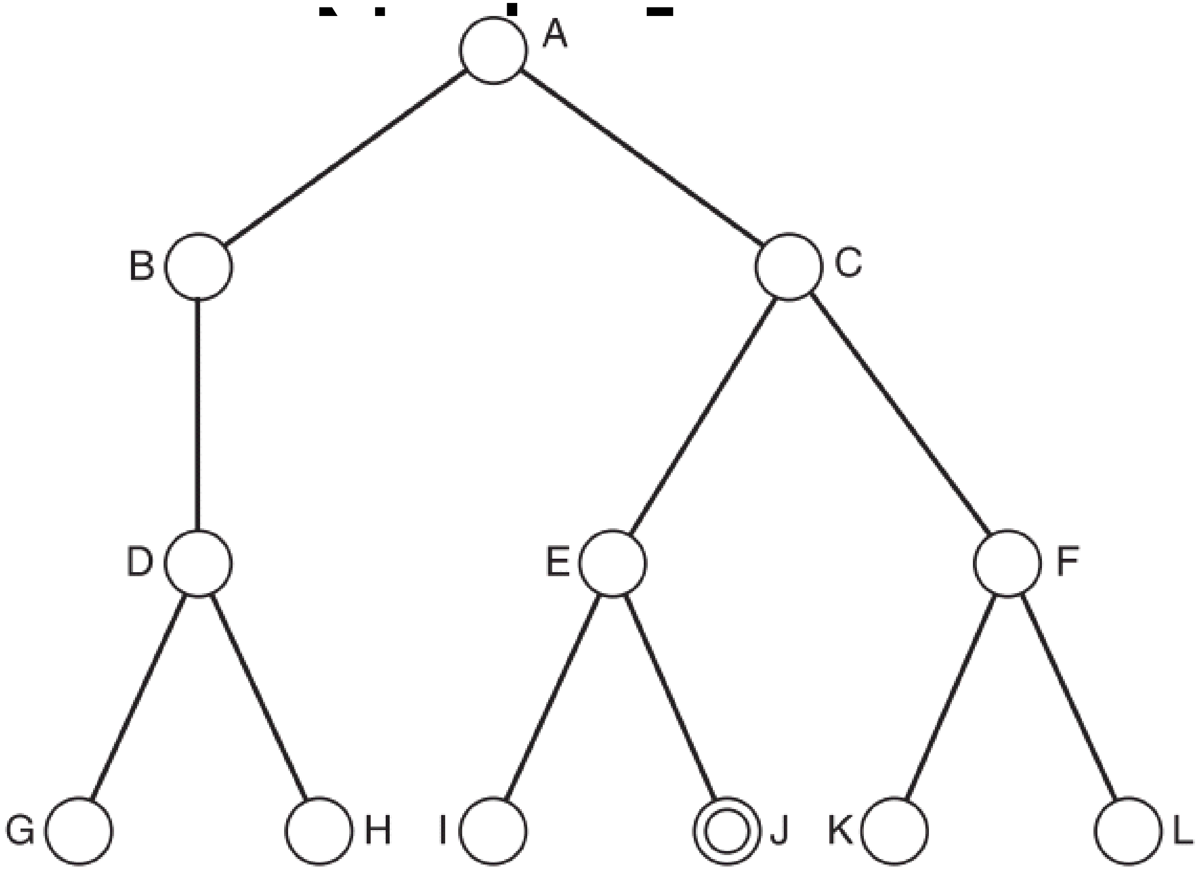
Current	Waiting
D	E, F, G, H
E	F, G, H
E	F, G, H, I, J
F	G, H, I, J
F	G, H, I, J, K, L
G	H, I, J, K, L

Breadth-First Search Goal -



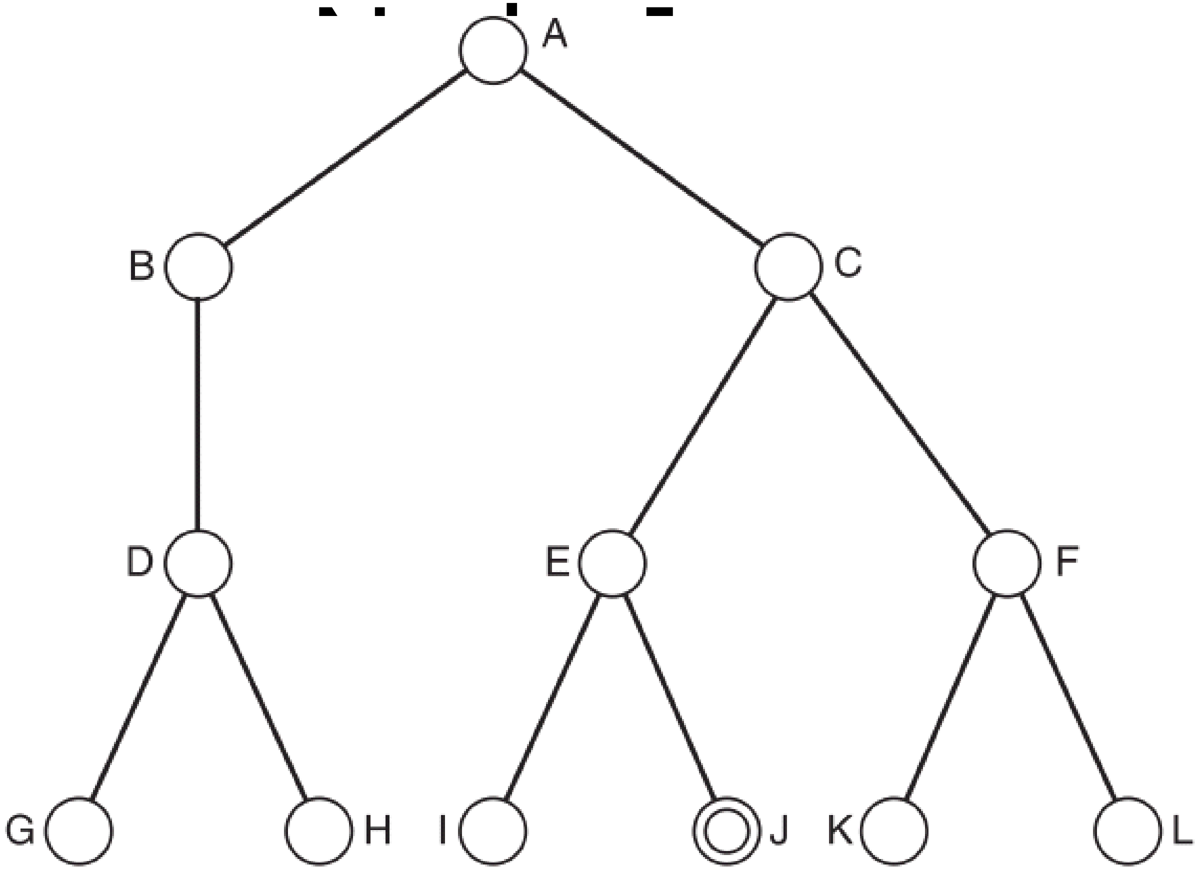
Current	Waiting
D	E, F, G, H
E	F, G, H
E	F, G, H, I, J
F	G, H, I, J
F	G, H, I, J, K, L
G	H, I, J, K, L

Breadth-First Search Goal -



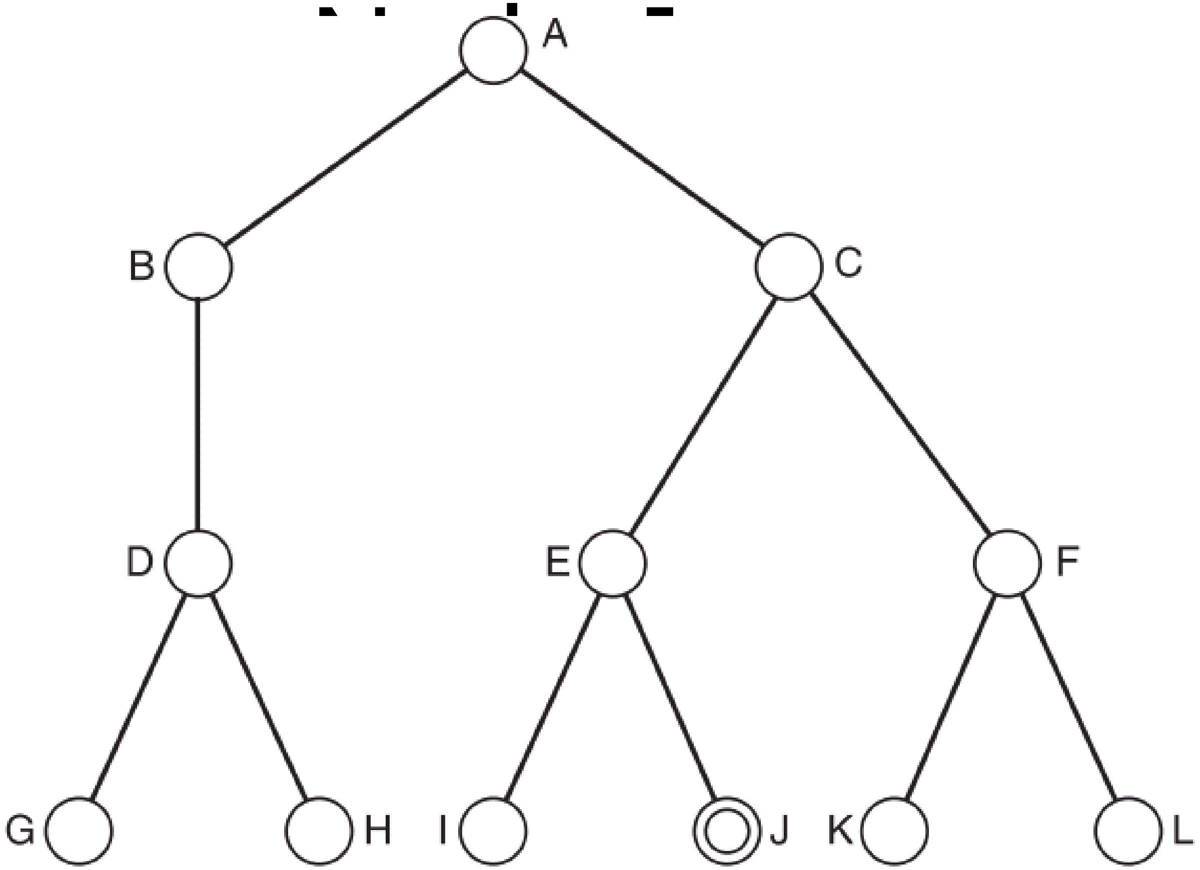
Current	Waiting
D	E, F, G, H
E	F, G, H
E	F, G, H, I, J
F	G, H, I, J
F	G, H, I, J, K, L
G	H, I, J, K, L
H	

Breadth-First Search Goal -



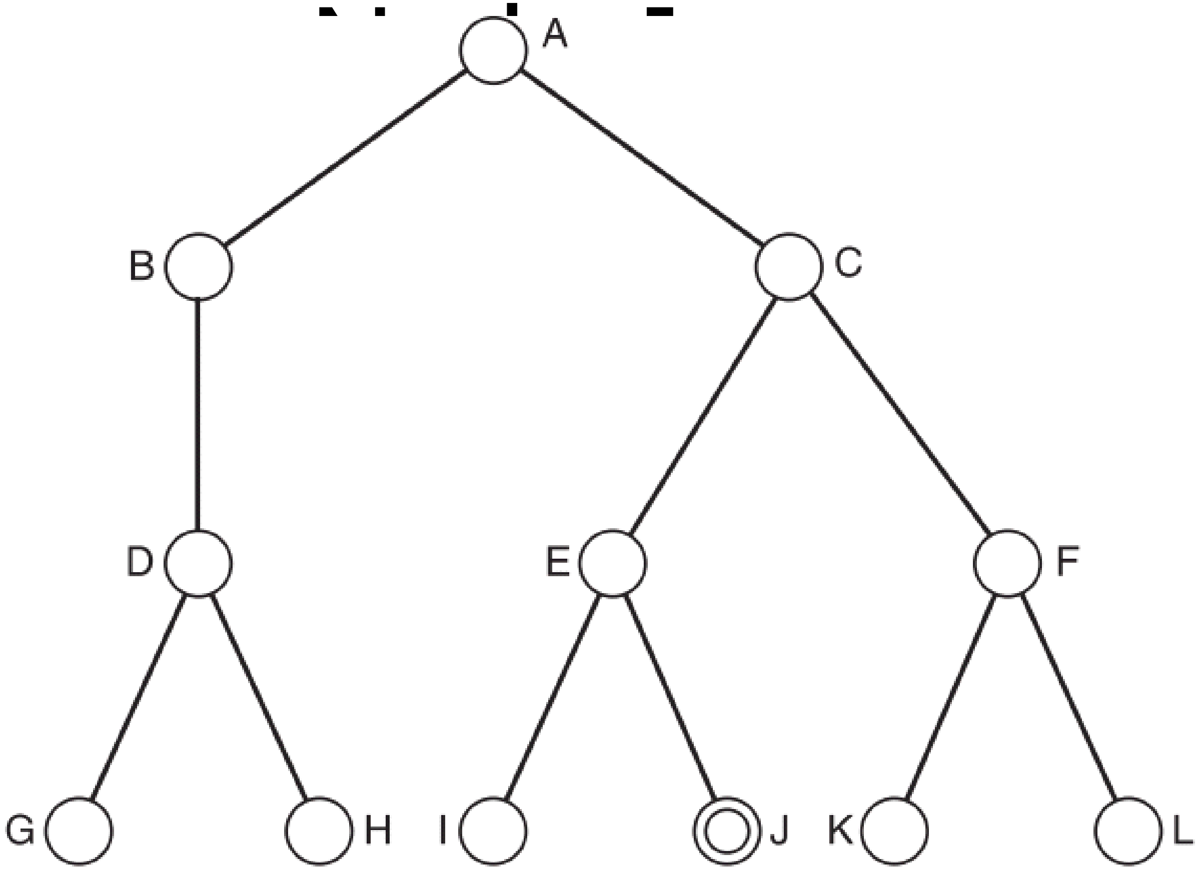
Current	Waiting
D	E, F, G, H
E	F, G, H
E	F, G, H, I, J
F	G, H, I, J
F	G, H, I, J, K, L
G	H, I, J, K, L
H	I, J, K, L

Breadth-First Search Goal -



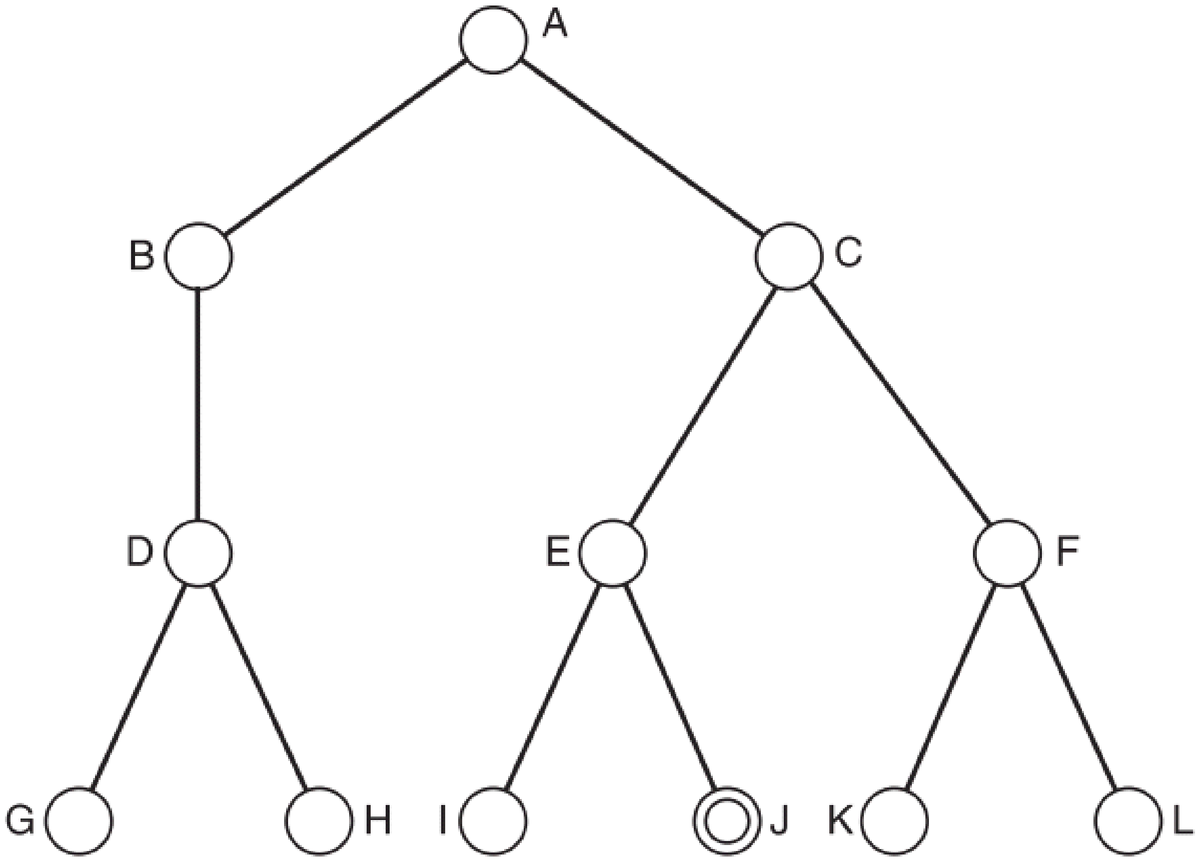
Current	Waiting
D	E, F, G, H
E	F, G, H
E	F, G, H, I, J
F	G, H, I, J
F	G, H, I, J, K, L
G	H, I, J, K, L
H	I, J, K, L

Breadth-First Search Goal -



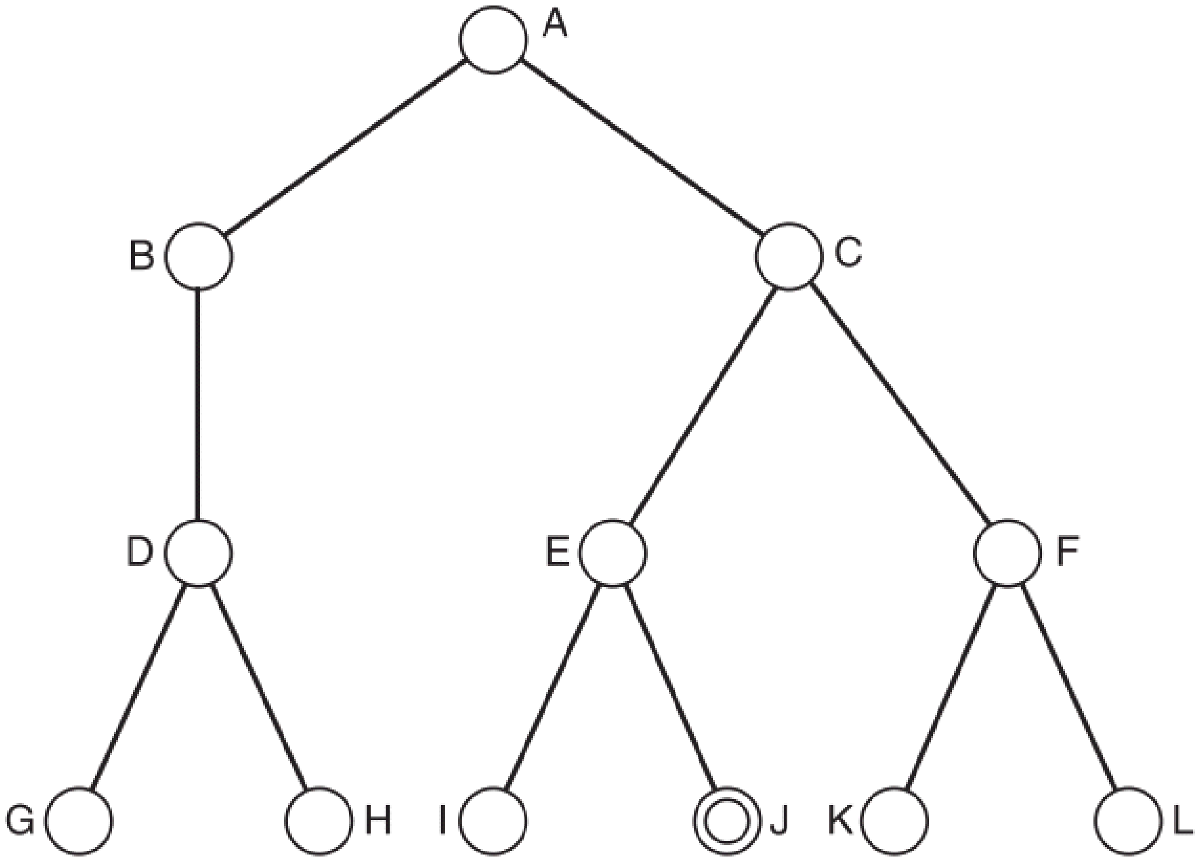
Current	Waiting
D	E, F, G, H
E	F, G, H
E	F, G, H, I, J
F	G, H, I, J
F	G, H, I, J, K, L
G	H, I, J, K, L
H	I, J, K, L
I	

Breadth-First Search
Goal - Node J



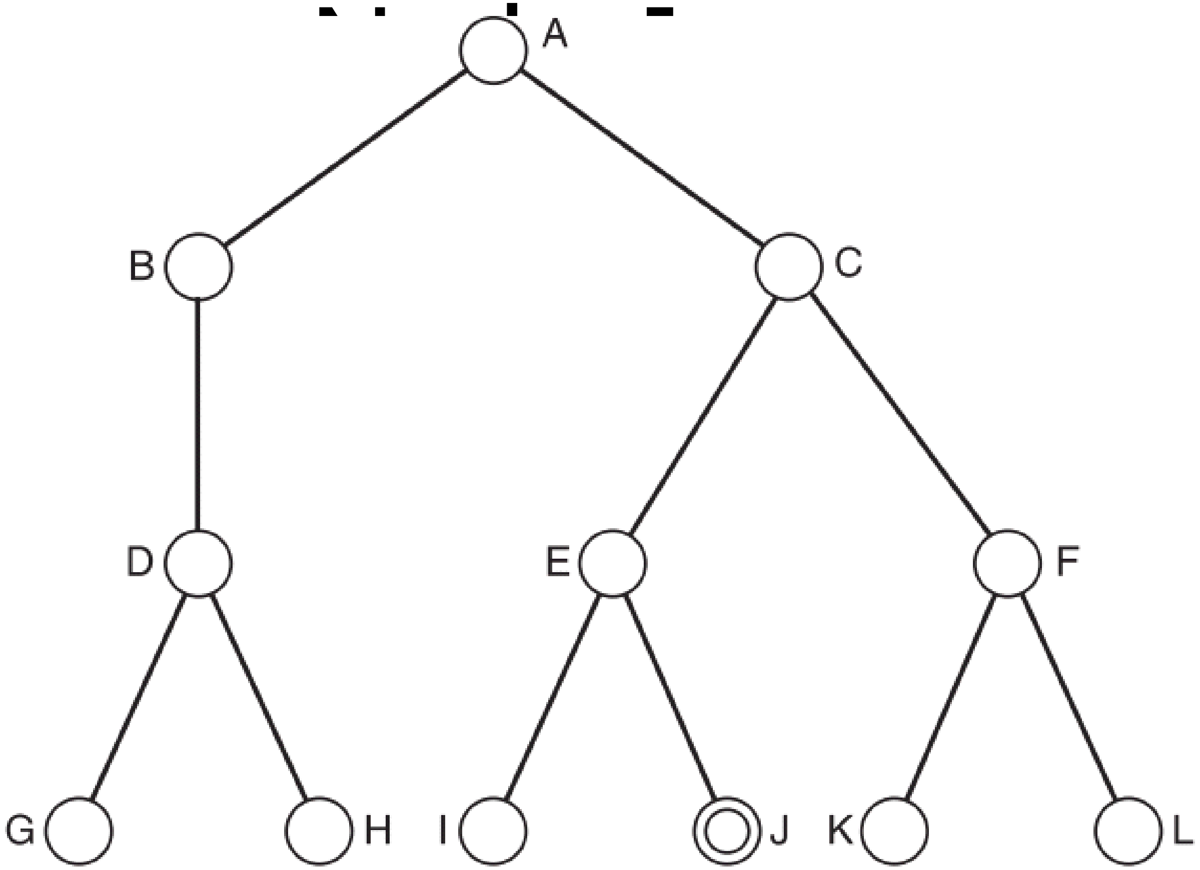
Current	Waiting
D	E, F, G, H
E	F, G, H
E	F, G, H, I, J
F	G, H, I, J
F	G, H, I, J, K, L
G	H, I, J, K, L
H	I, J, K, L
I	J, K, L

Breadth-First Search
Goal - Node J



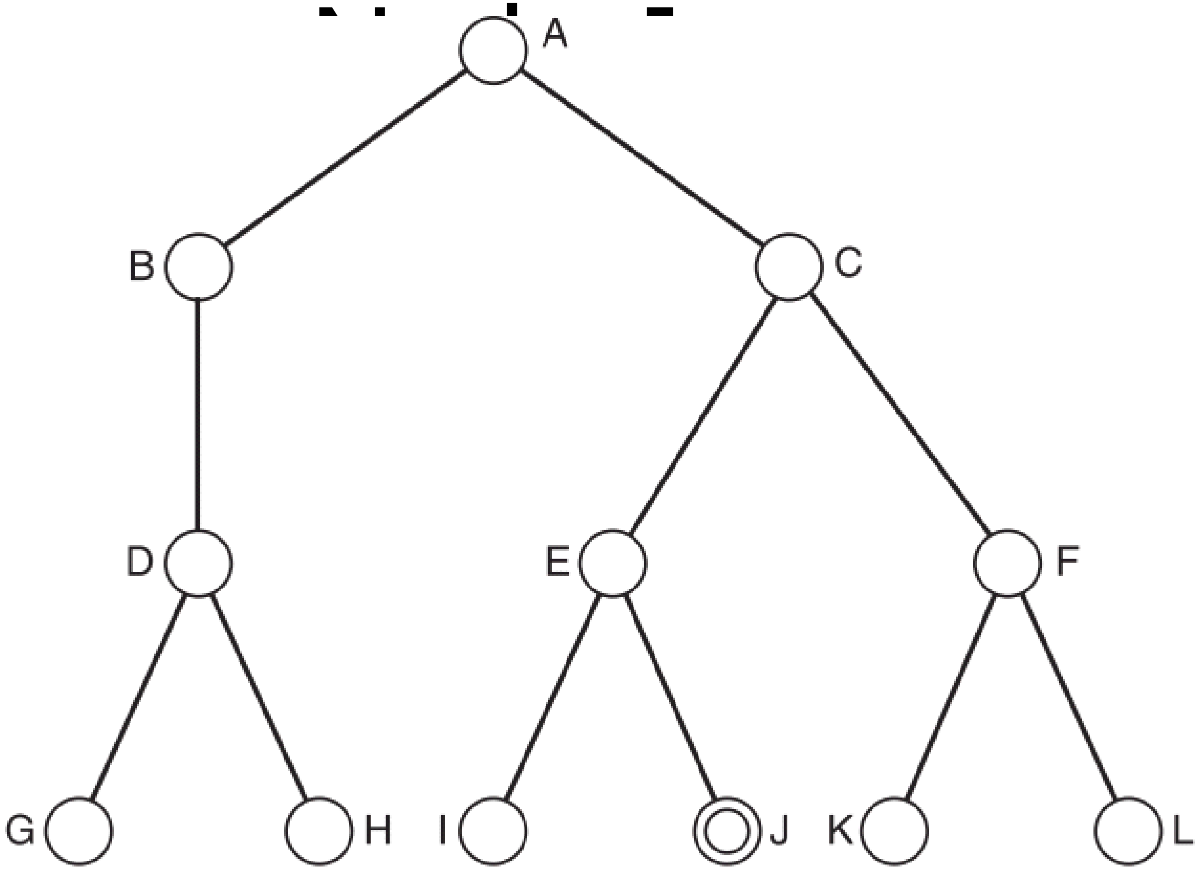
Current	Waiting
D	E, F, G, H
E	F, G, H
E	F, G, H, I, J
F	G, H, I, J
F	G, H, I, J, K, L
G	H, I, J, K, L
H	I, J, K, L
I	J, K, L

Breadth-First Search Goal -



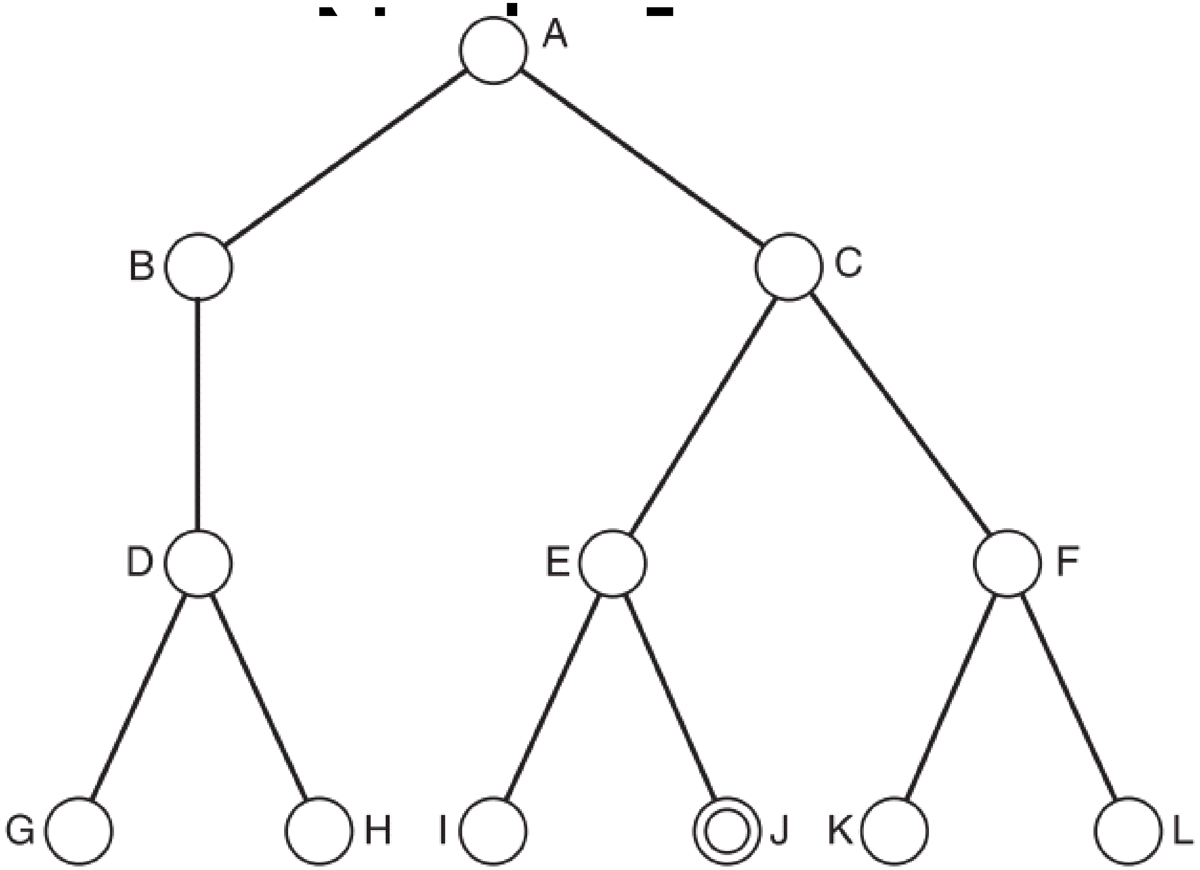
Current	Waiting
D	E, F, G, H
E	F, G, H
E	F, G, H, I, J
F	G, H, I, J
F	G, H, I, J, K, L
G	H, I, J, K, L
H	I, J, K, L
I	J, K, L
J	

Breadth-First Search Goal -



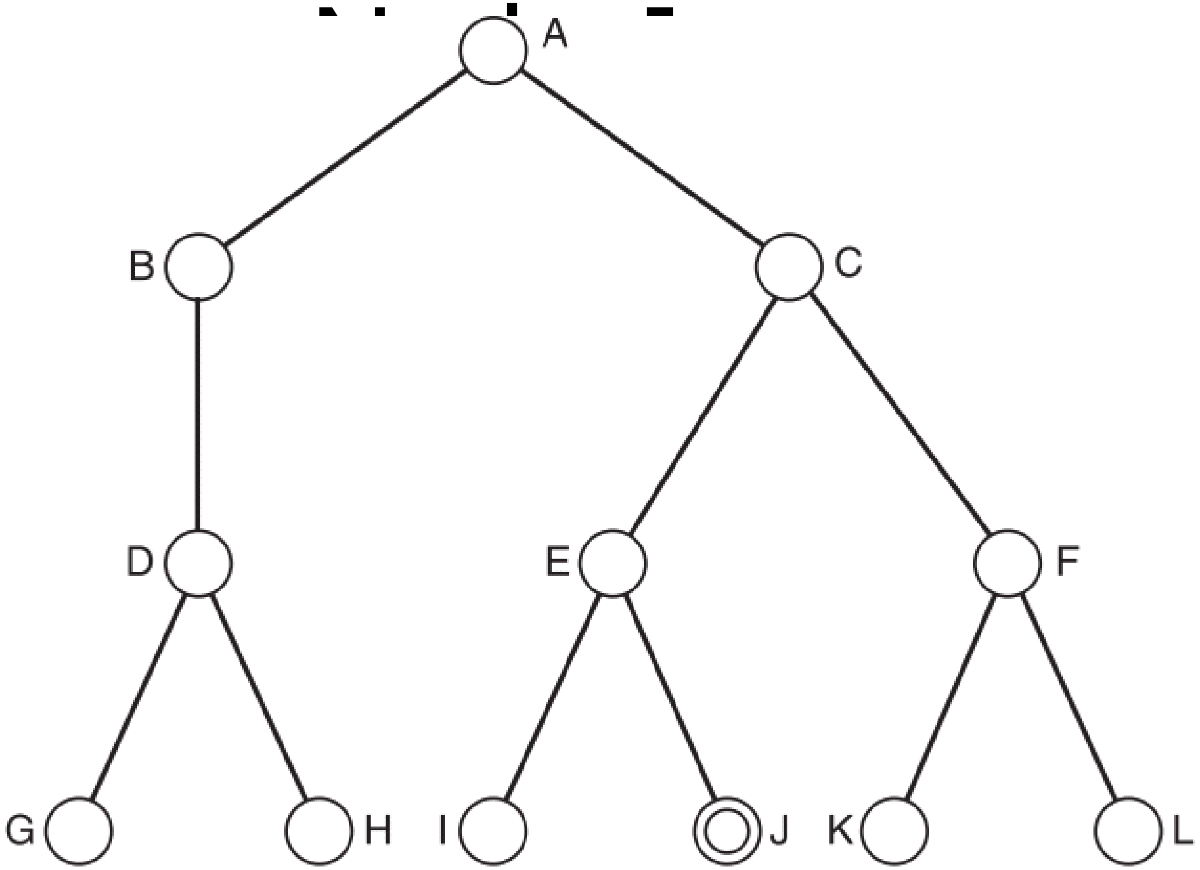
Current	Waiting
D	E, F, G, H
E	F, G, H
E	F, G, H, I, J
F	G, H, I, J
F	G, H, I, J, K, L
G	H, I, J, K, L
H	I, J, K, L
I	J, K, L
J	K, L

Breadth-First Search Goal -



Current	Waiting
D	E, F, G, H
E	F, G, H
E	F, G, H, I, J
F	G, H, I, J
F	G, H, I, J, K, L
G	H, I, J, K, L
H	I, J, K, L
I	J, K, L
J	K, L

Breadth-First Search Goal -



GOAL

Current	Waiting
D	E, F, G, H
E	F, G, H
E	F, G, H, I, J
F	G, H, I, J
F	G, H, I, J, K, L
G	H, I, J, K, L
H	I, J, K, L
I	J, K, L
J	K, L

BFS Algorithm

```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  frontier ← a FIFO queue with node as the only element
  explored ← an empty set
  loop do
      if EMPTY?(frontier) then return failure
      node ← POP(frontier)   /* chooses the shallowest node in frontier */
      add node.STATE to explored
      for each action in problem.ACTIONS(node.STATE) do
          child ← CHILD-NODE(problem, node, action)
          if child.STATE is not in explored or frontier then
              if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
              frontier ← INSERT(child, frontier)
```

Figure 3.11 Breadth-first search on a graph.

Analyzing BFS

Good news:

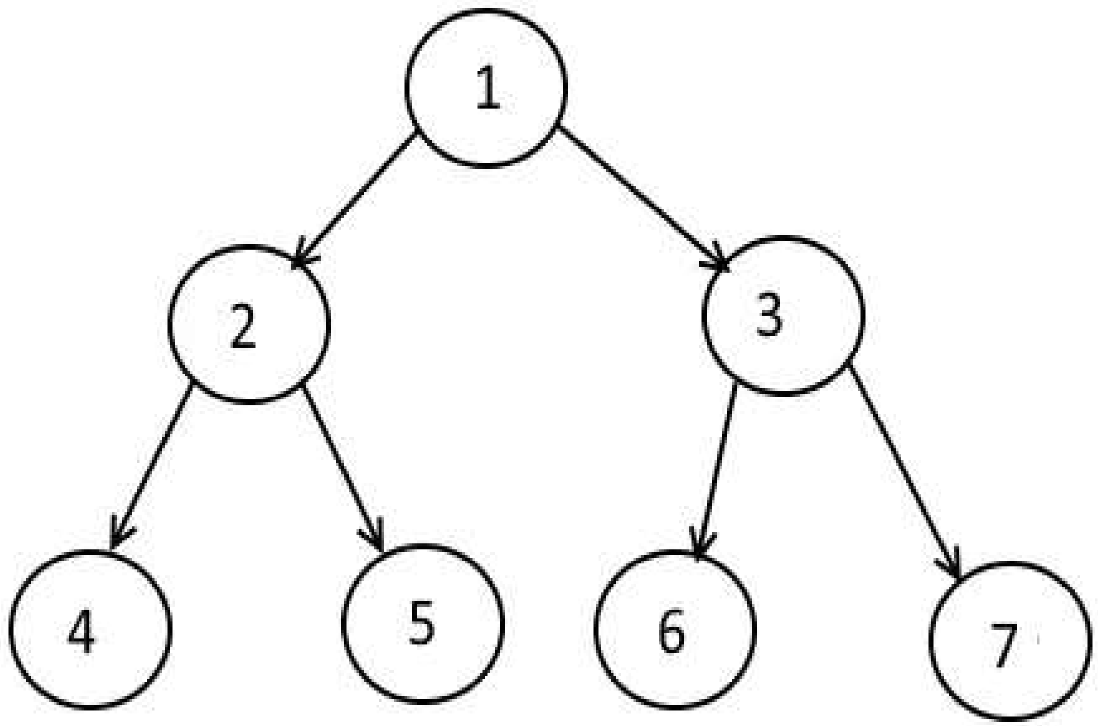
- *Complete*
- ***Guaranteed** to find the shallowest path to the goal This is **not necessarily the best path!** But we can "fix" the algorithm to get the best path.*
- *Different start-goal combinations can be explored at the same time*

Bad news:

- *Exponential time complexity: $O(b^d)$ (why?) This is the same for all uninformed search methods*
- ***Exponential memory requirements!** $O(b^d)$ (why?) This is not good...*

DEPTH - FIRST SEARCH

DFS

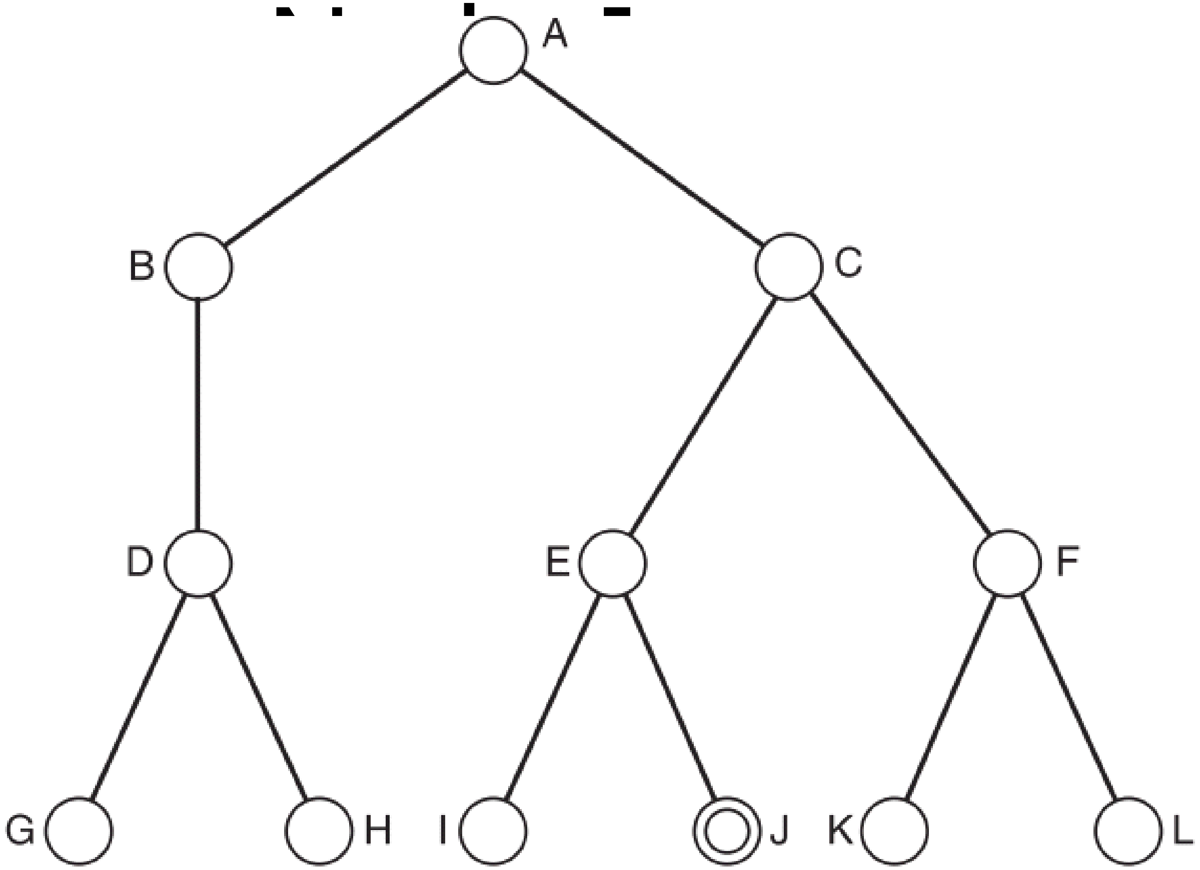


- In depth-first search, we start with the root node and completely **explore the descendants of a node before exploring its siblings** (and siblings are explored in a left- to-right fashion).
- Depth-first search always **expands the deepest node** in the current frontier of the search tree.
- **LIFO queue**

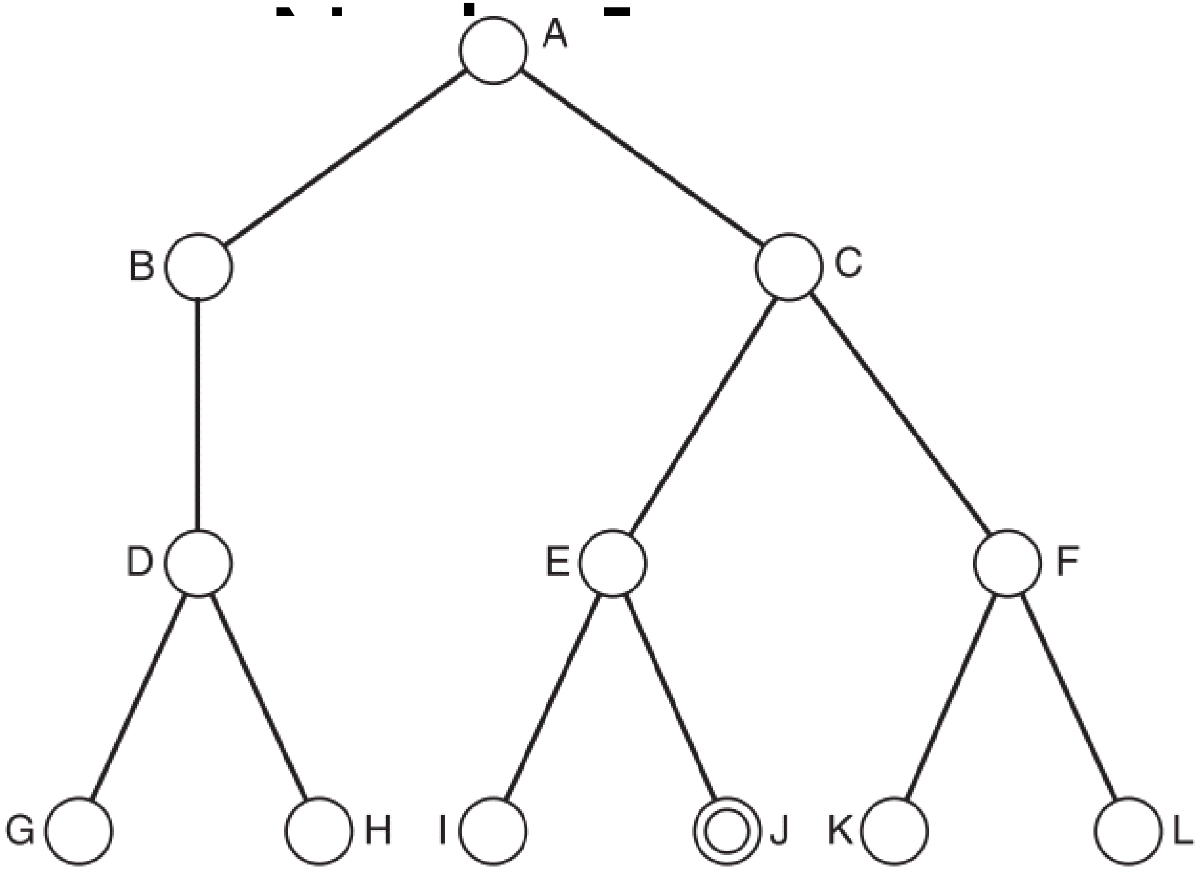
Depth-first traversal: 1 → 2 → 4 → 5 → 3 → 6 → 7

Depth-First Search Goal -

Current



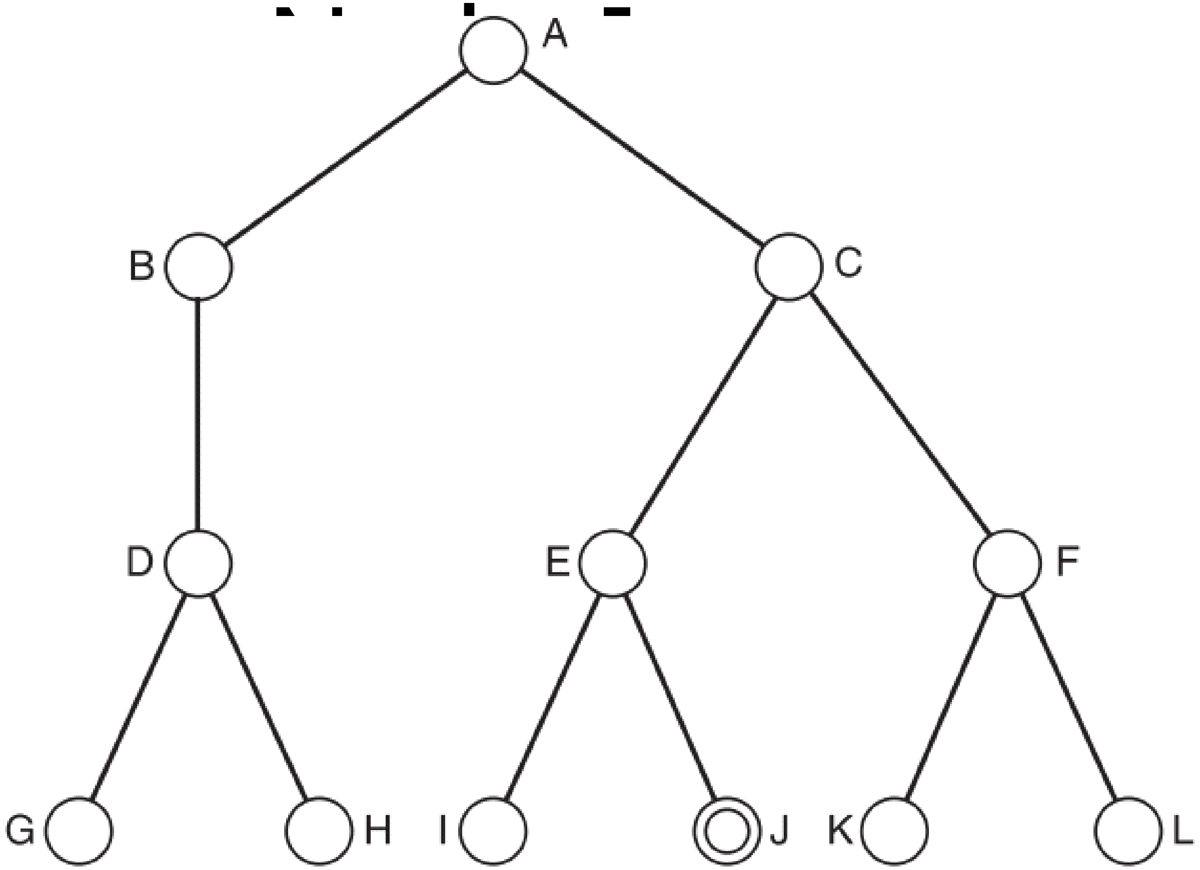
Depth-First Search Goal -



Current

A

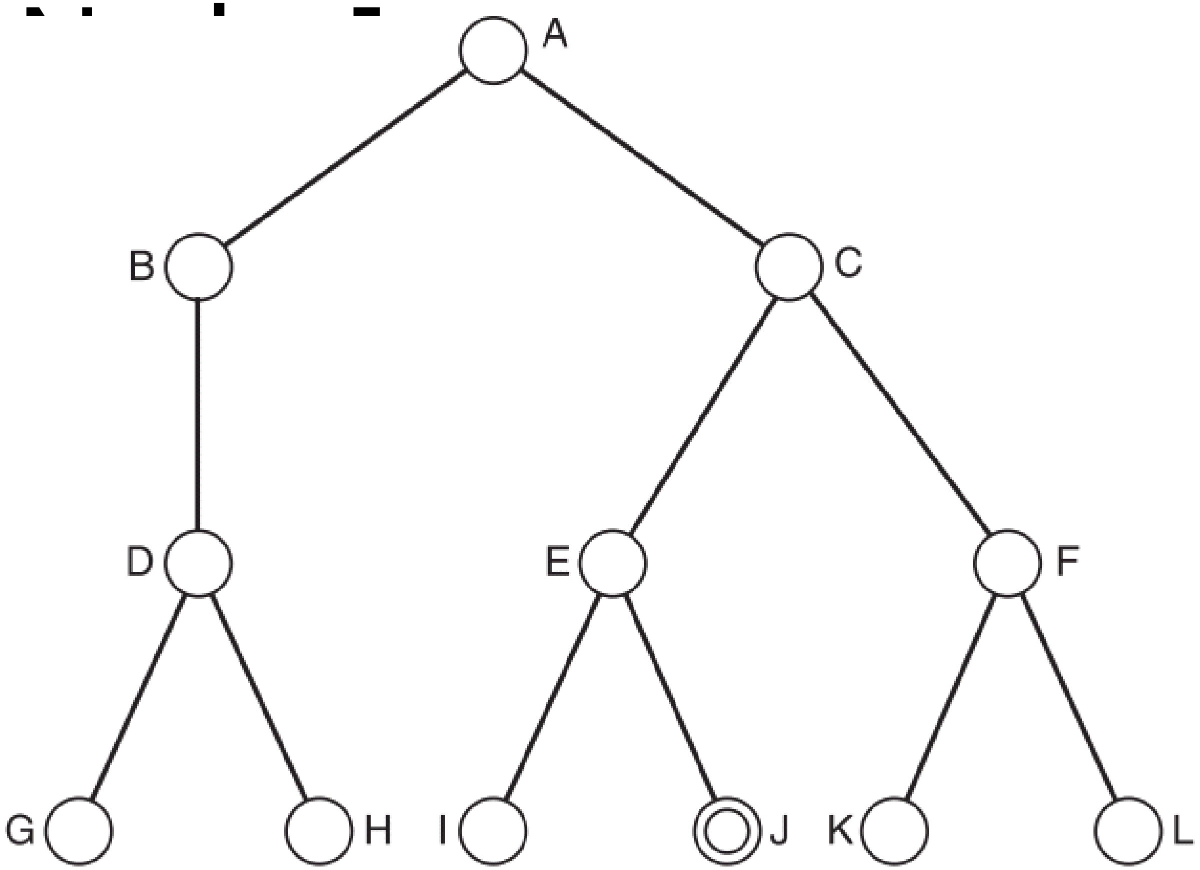
Depth-First Search Goal -



Current

A

Depth-First Search Goal -

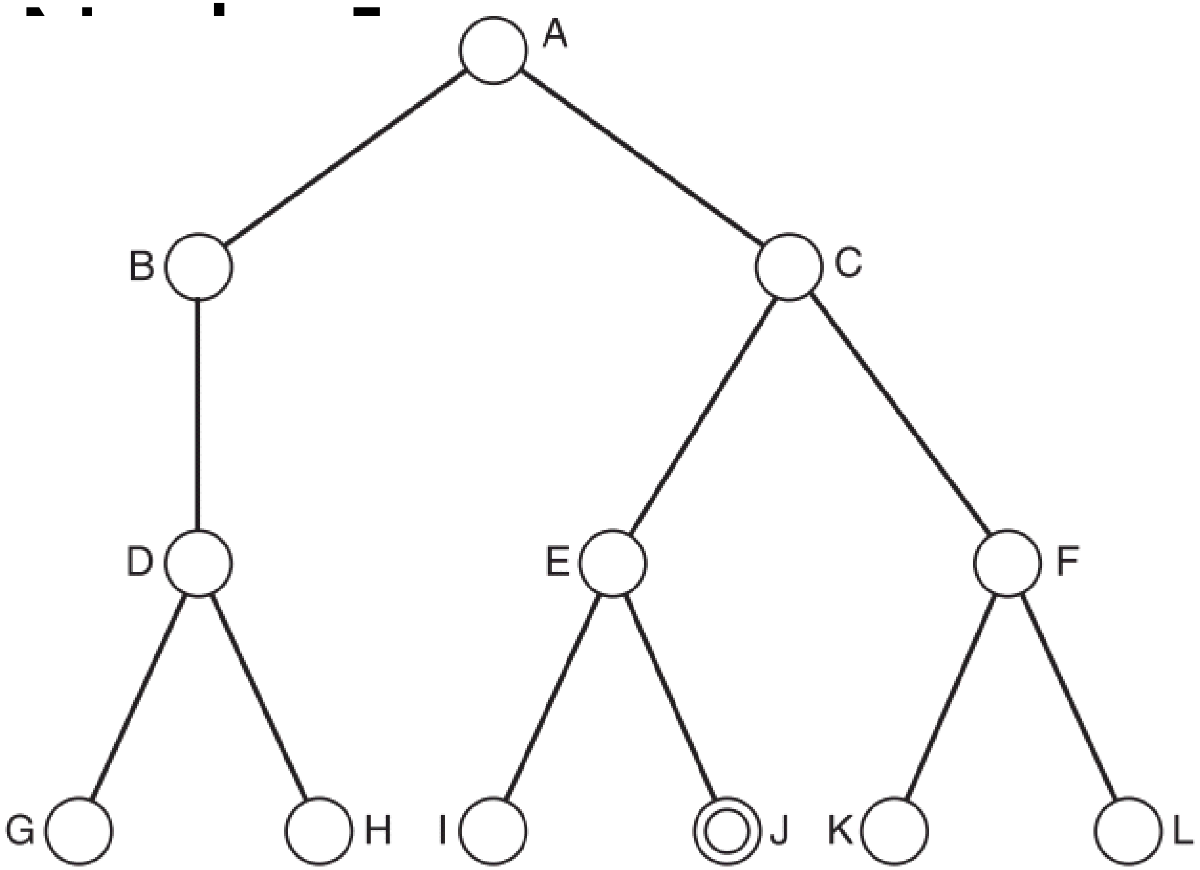


Current

A

B

Depth-First Search Goal -

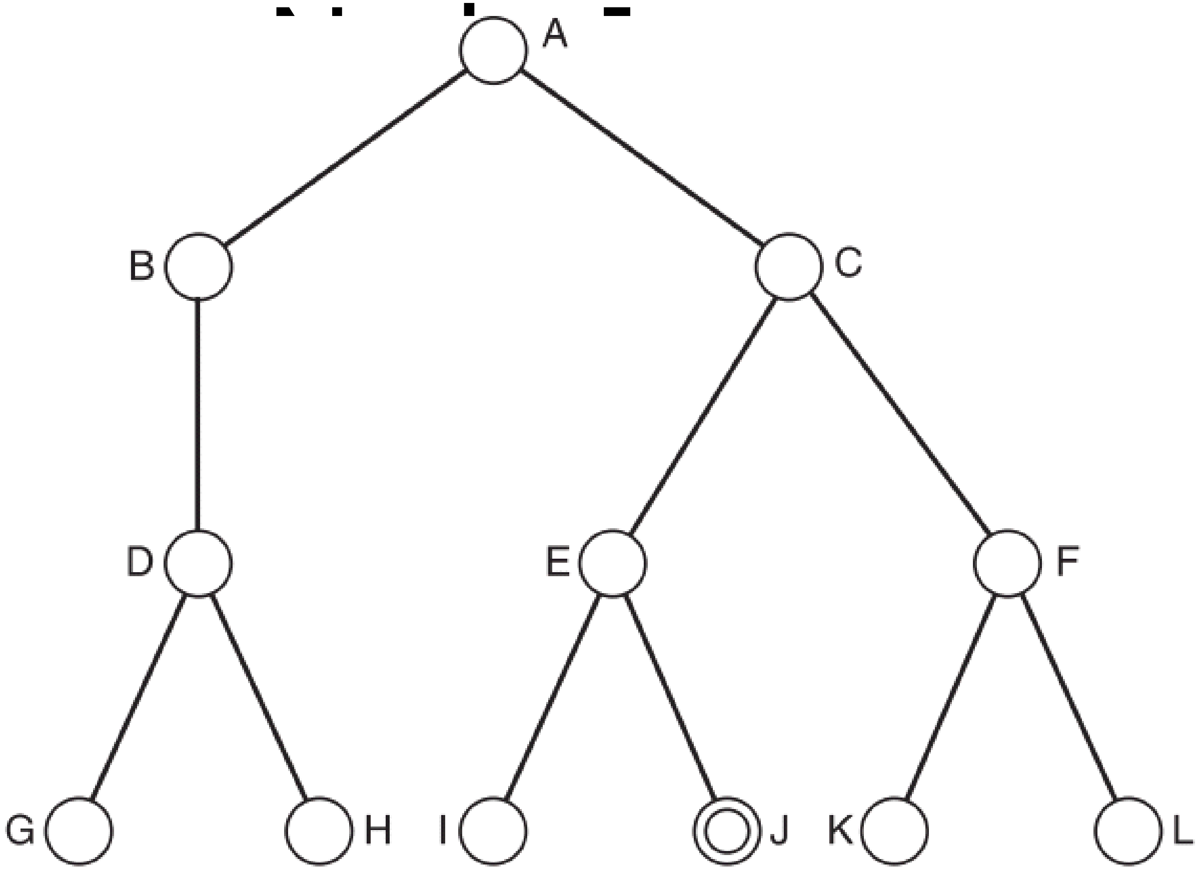


Current

A

B

Depth-First Search Goal -



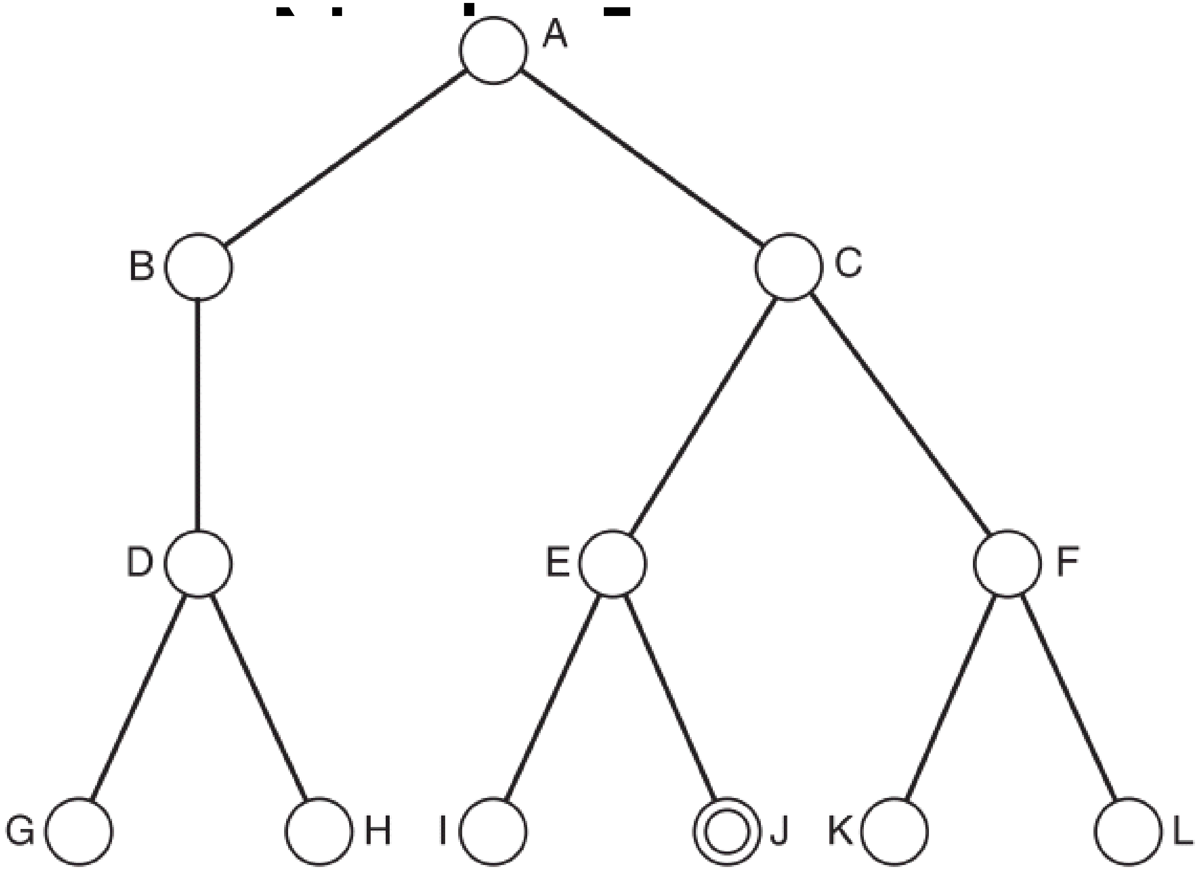
Current

A

B

D

Depth-First Search Goal -



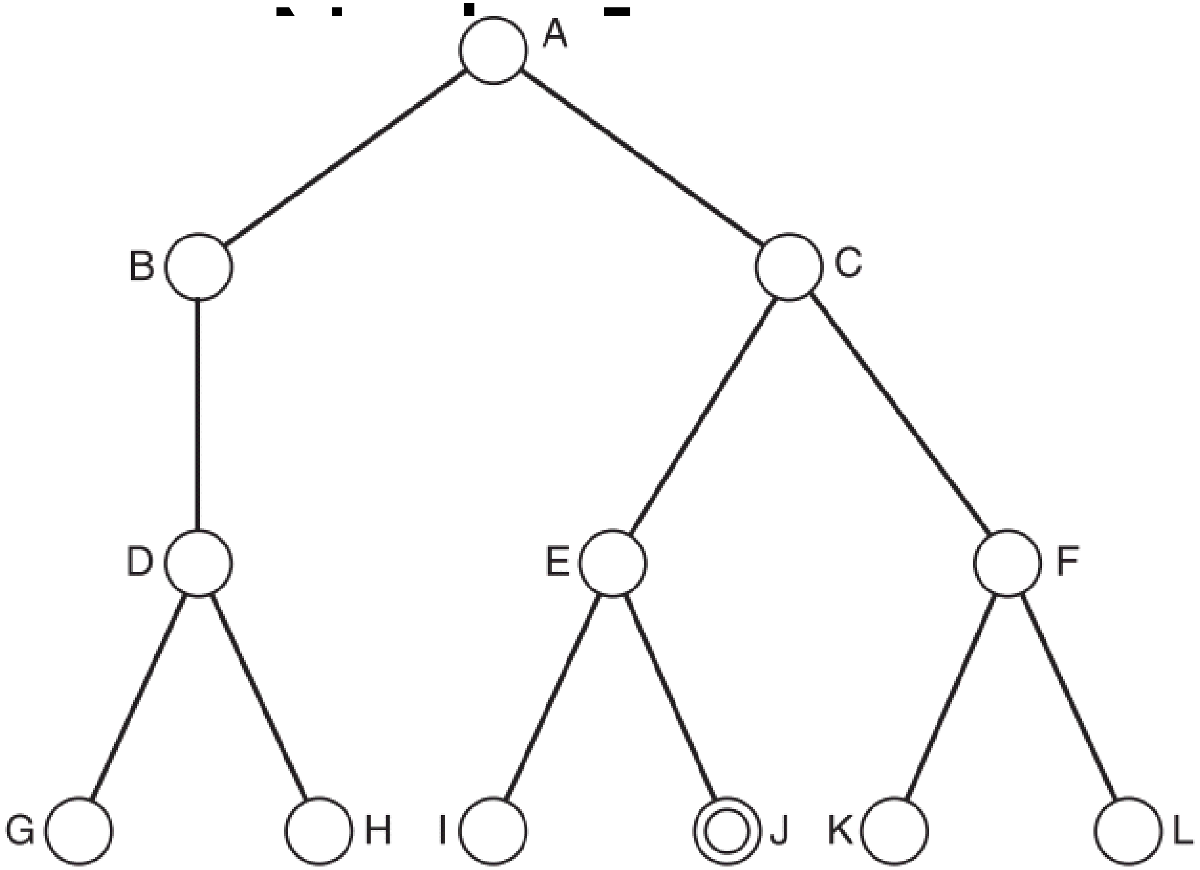
Current

A

B

D

Depth-First Search Goal -



Current

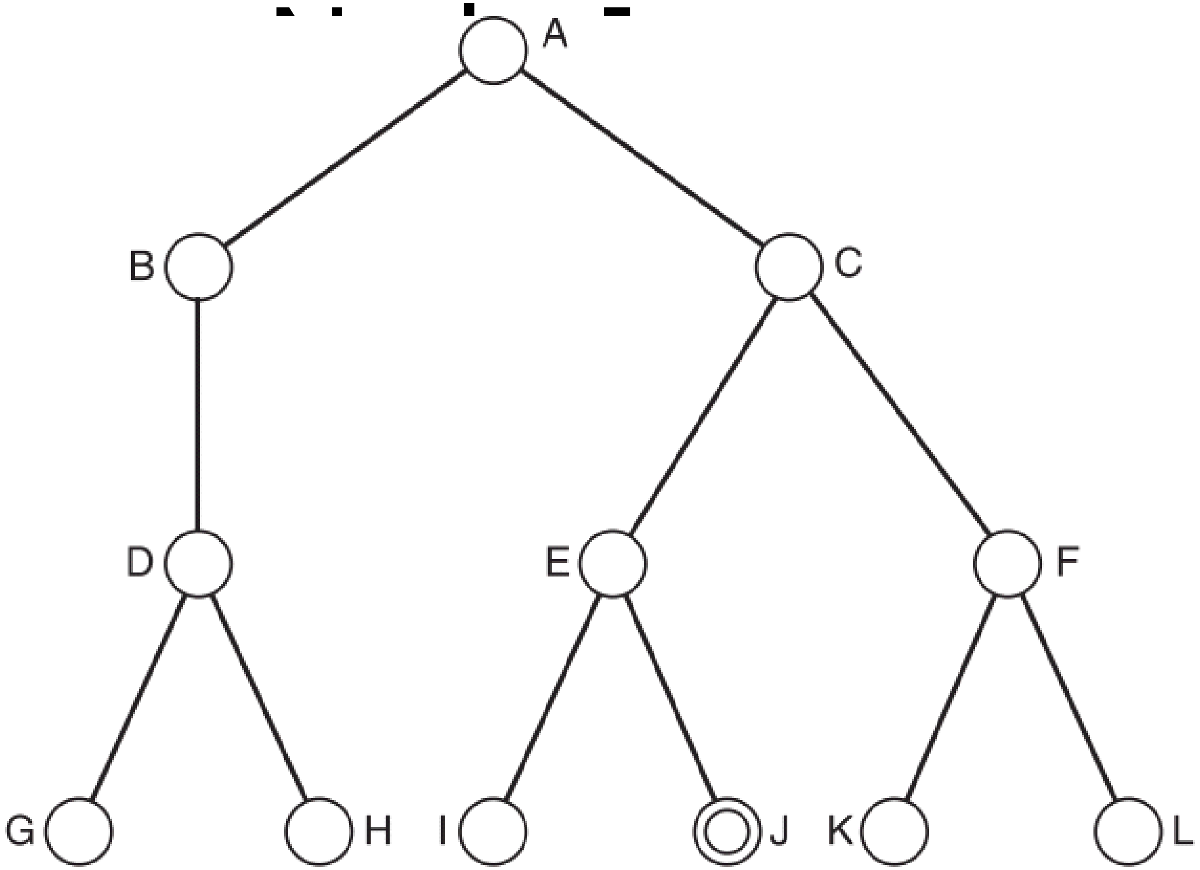
A

B

D

G

Depth-First Search Goal -



Current

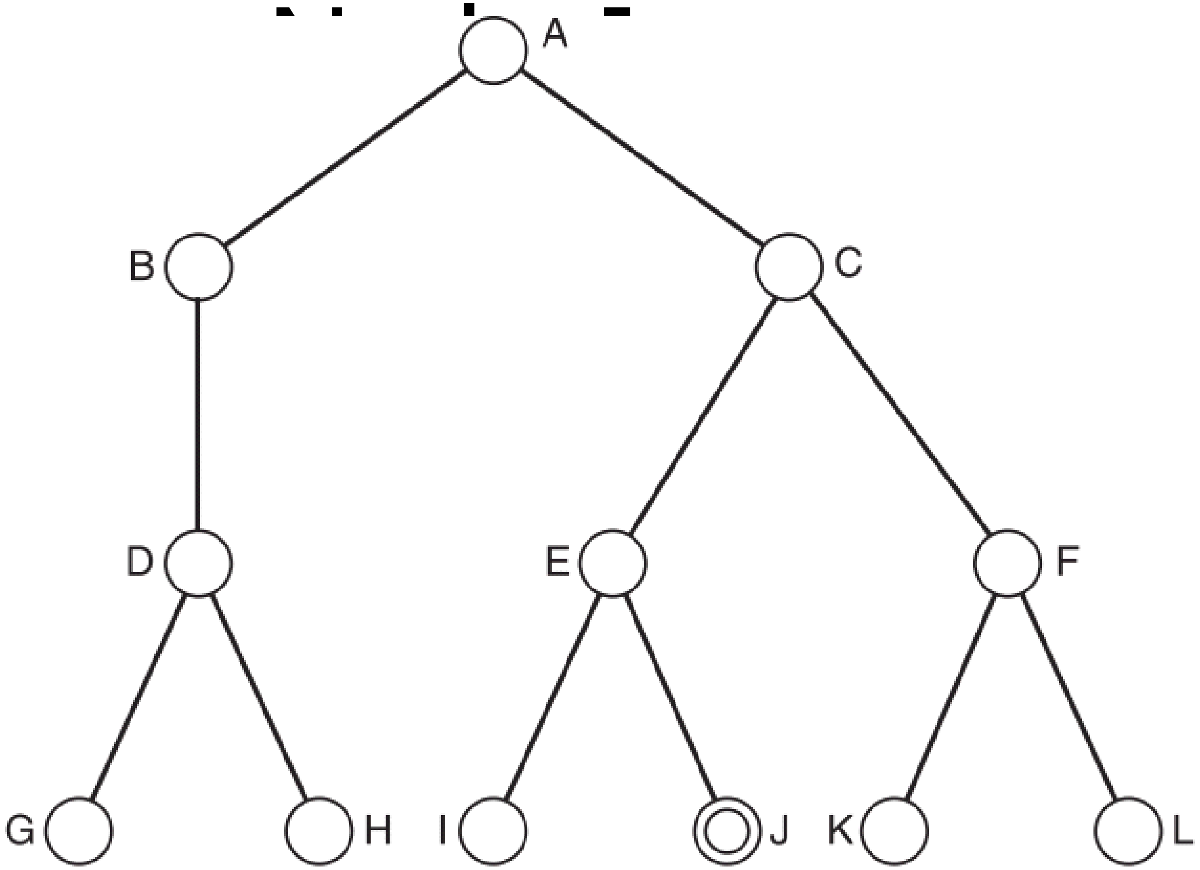
A

B

D

G

Depth-First Search Goal -



Current

A

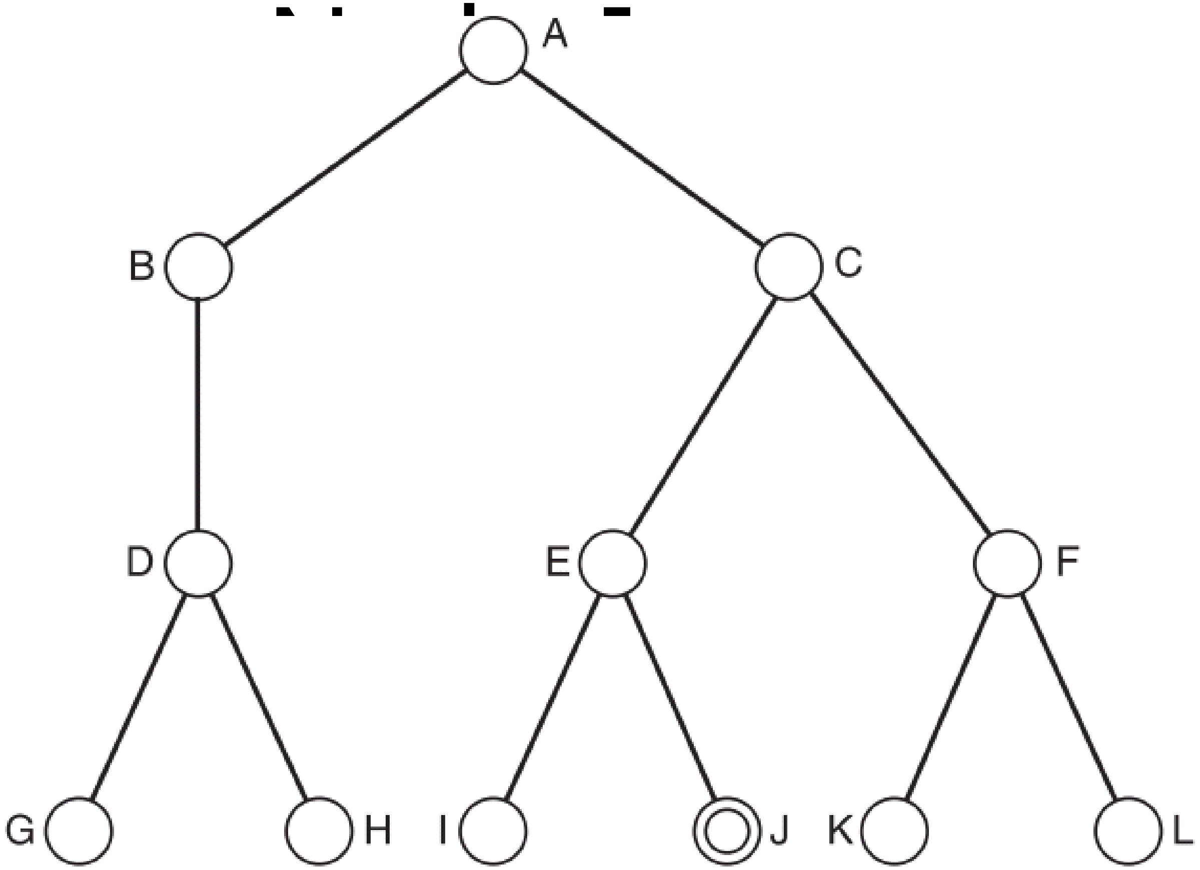
B

D

G

D

Depth-First Search Goal -



Current

A

B

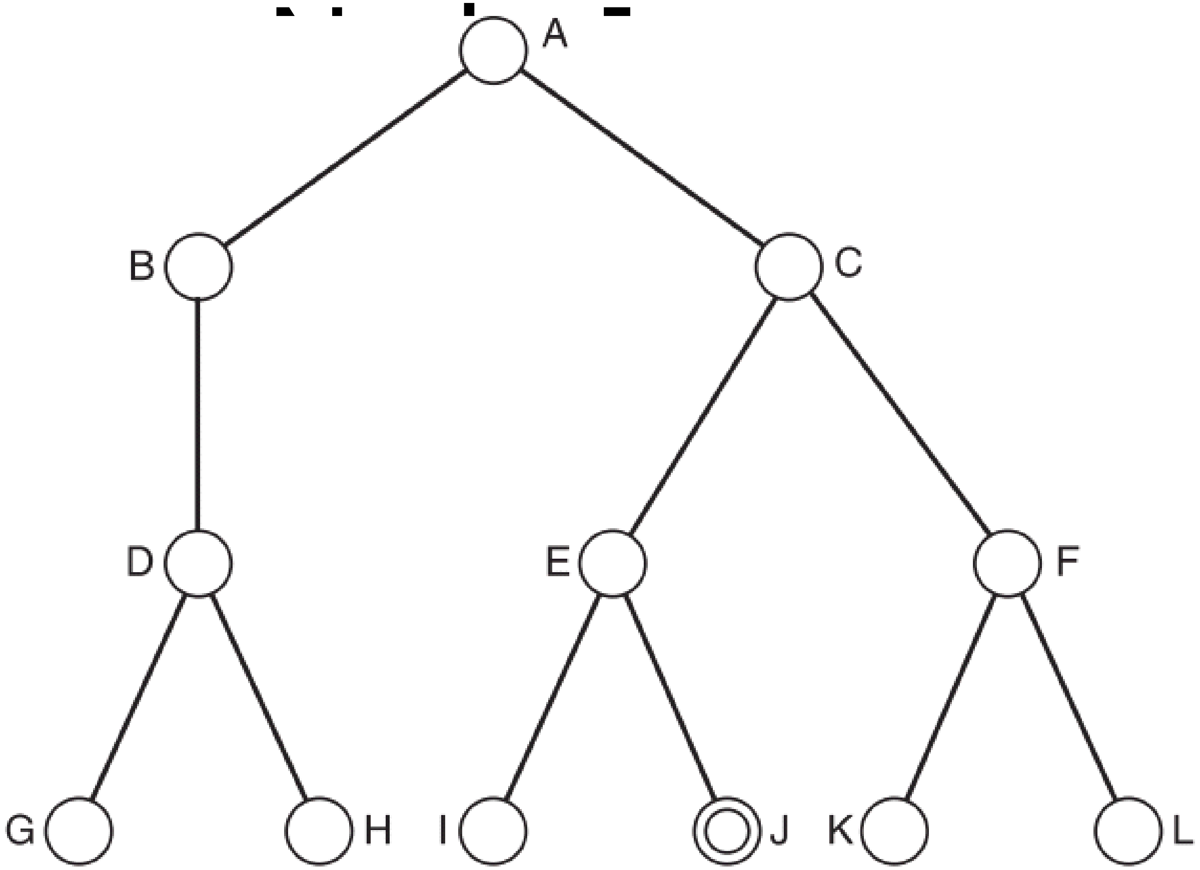
D

G

D

H

Depth-First Search Goal -



Current

A

B

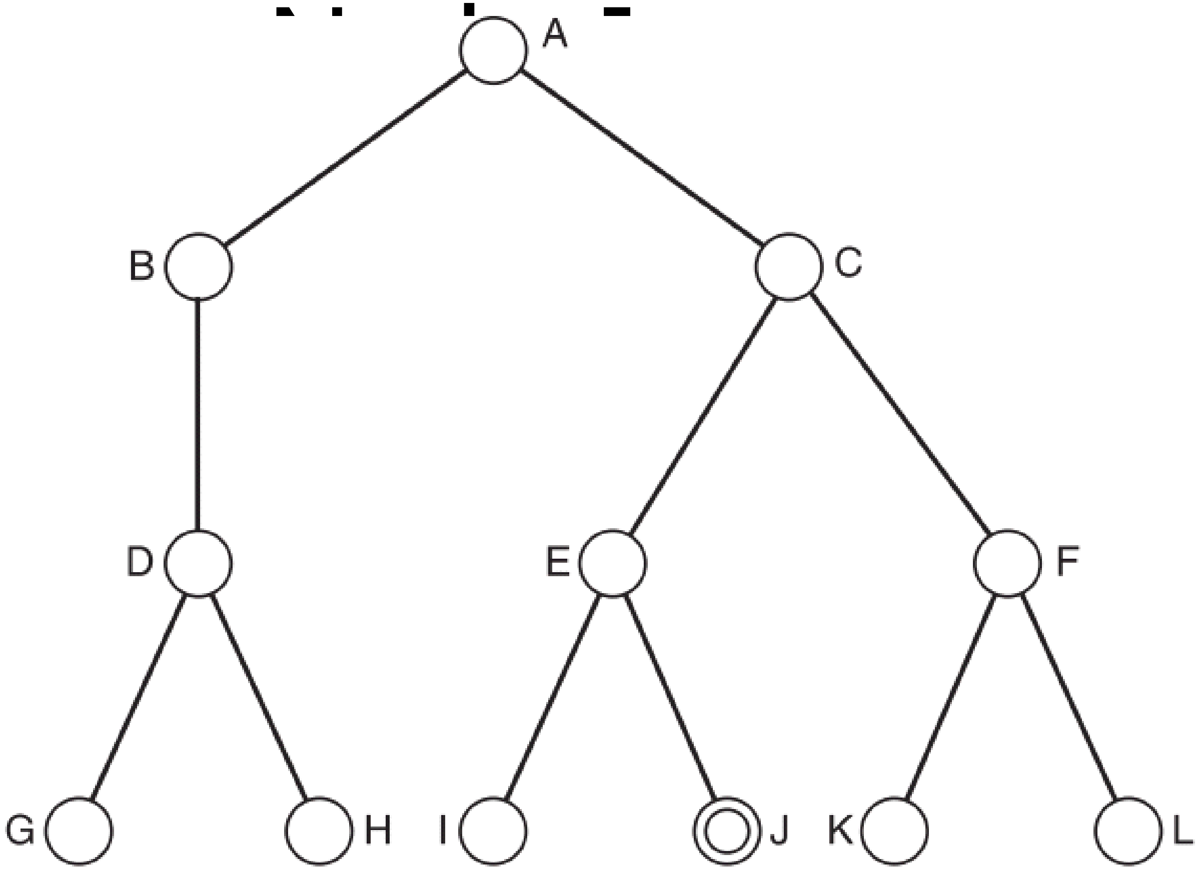
D

G

D

H

Depth-First Search Goal -



Current

A

B

D

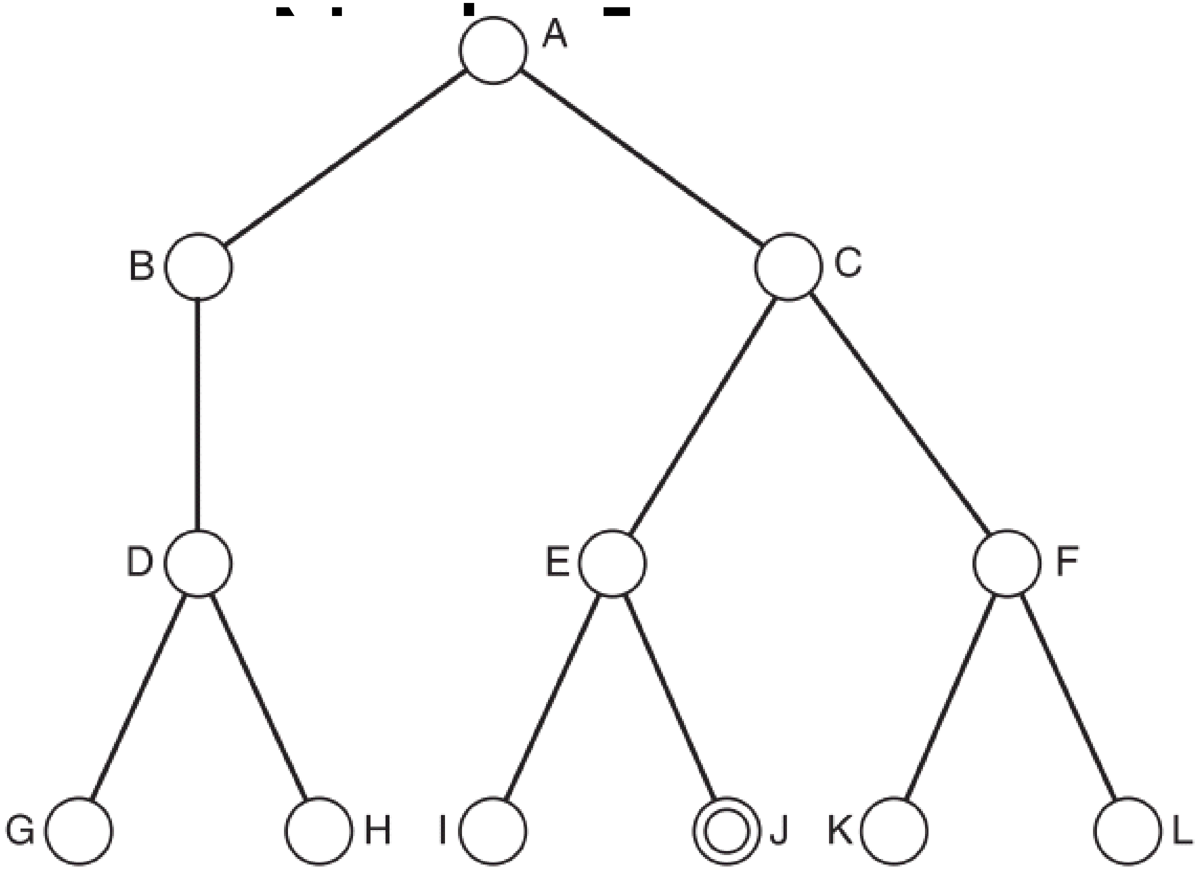
G

D

H

D

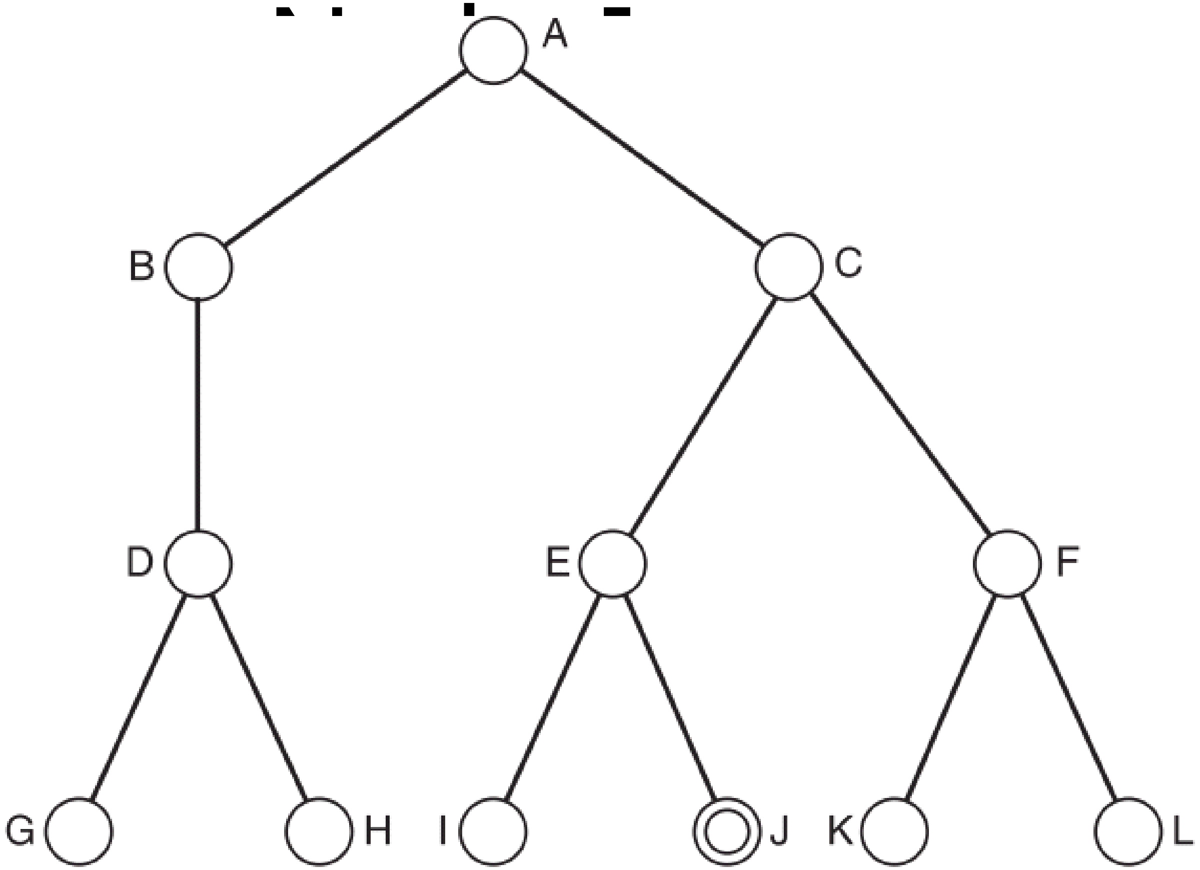
Depth-First Search Goal -



Current

- A
- B
- D
- G
- D
- H
- D
- B

Depth-First Search Goal -



Current

A

B

D

G

D

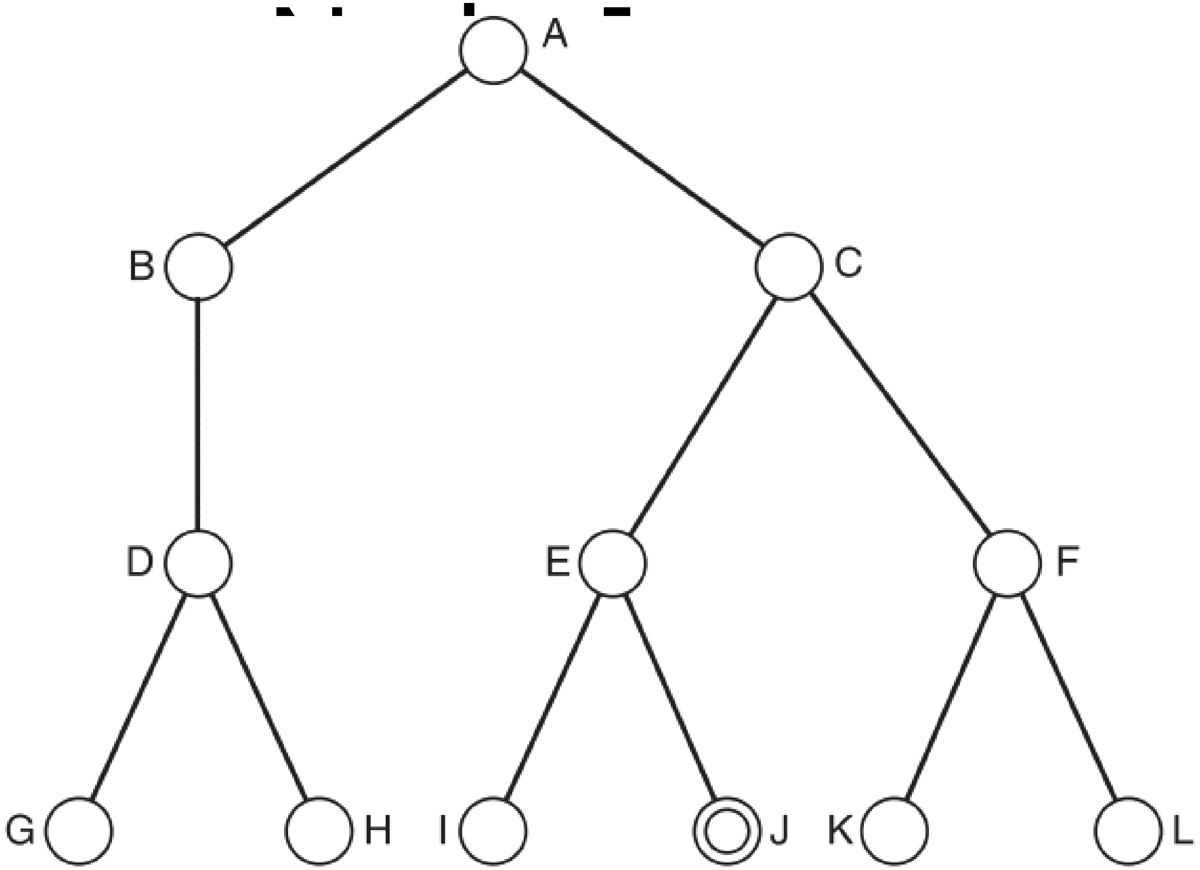
H

D

B

A

Depth-First Search Goal -



Current

A

C

B

D

G

D

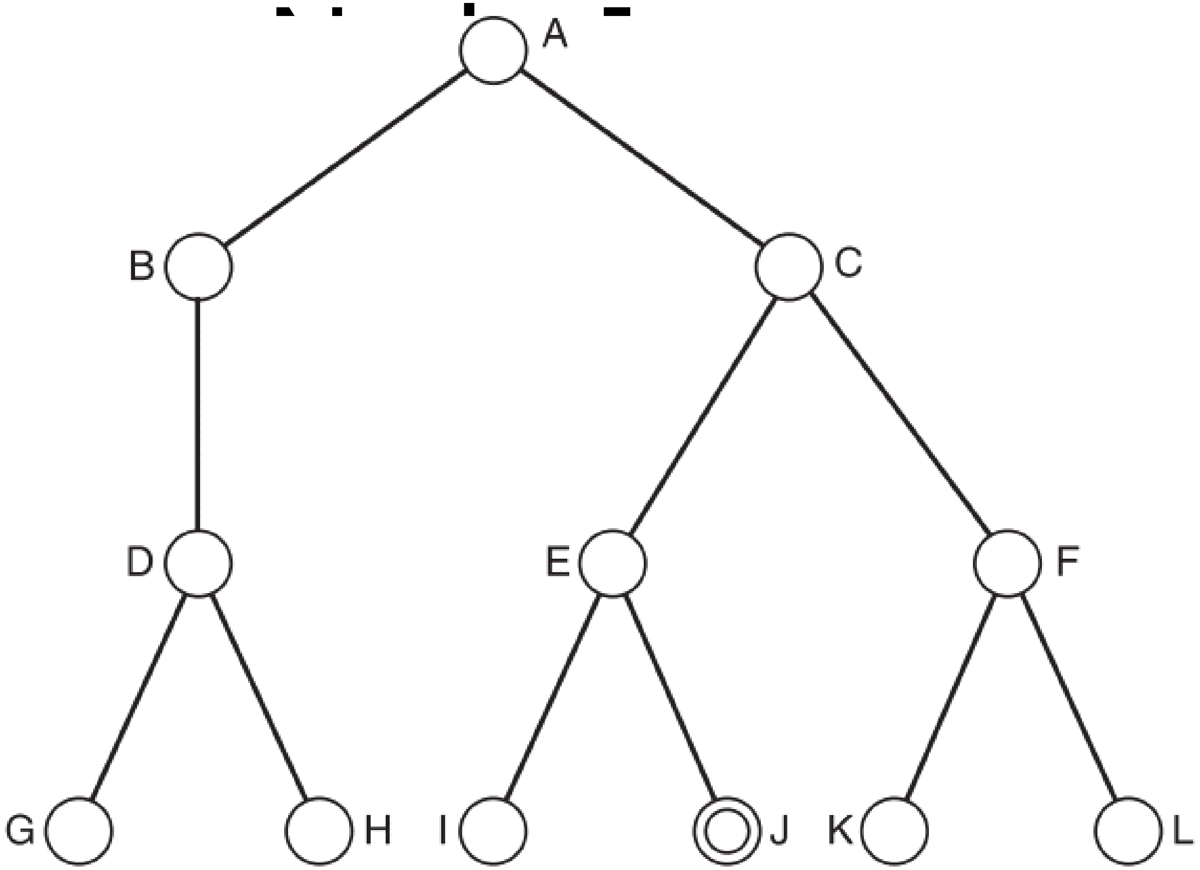
H

D

B

A

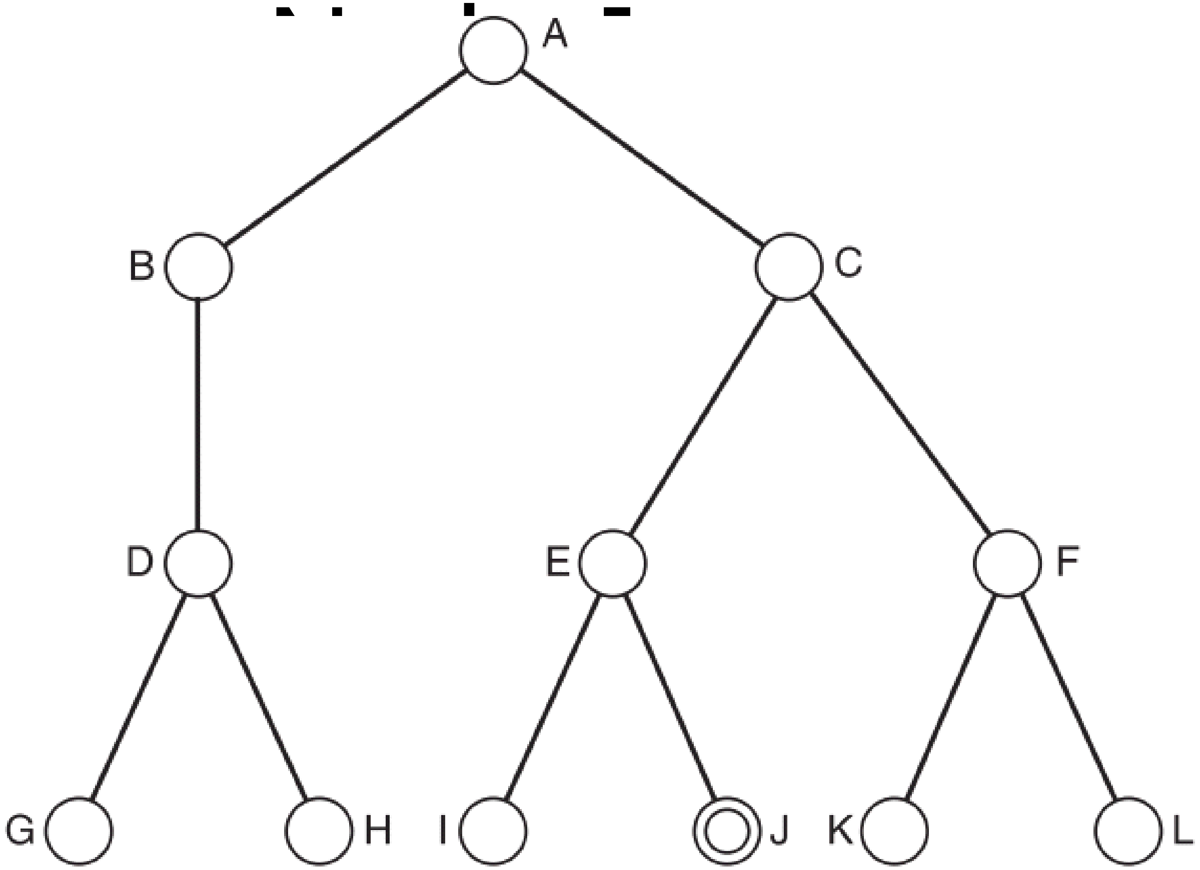
Depth-First Search Goal -



Current

A	C
B	
D	
G	
D	
H	
D	
B	
A	

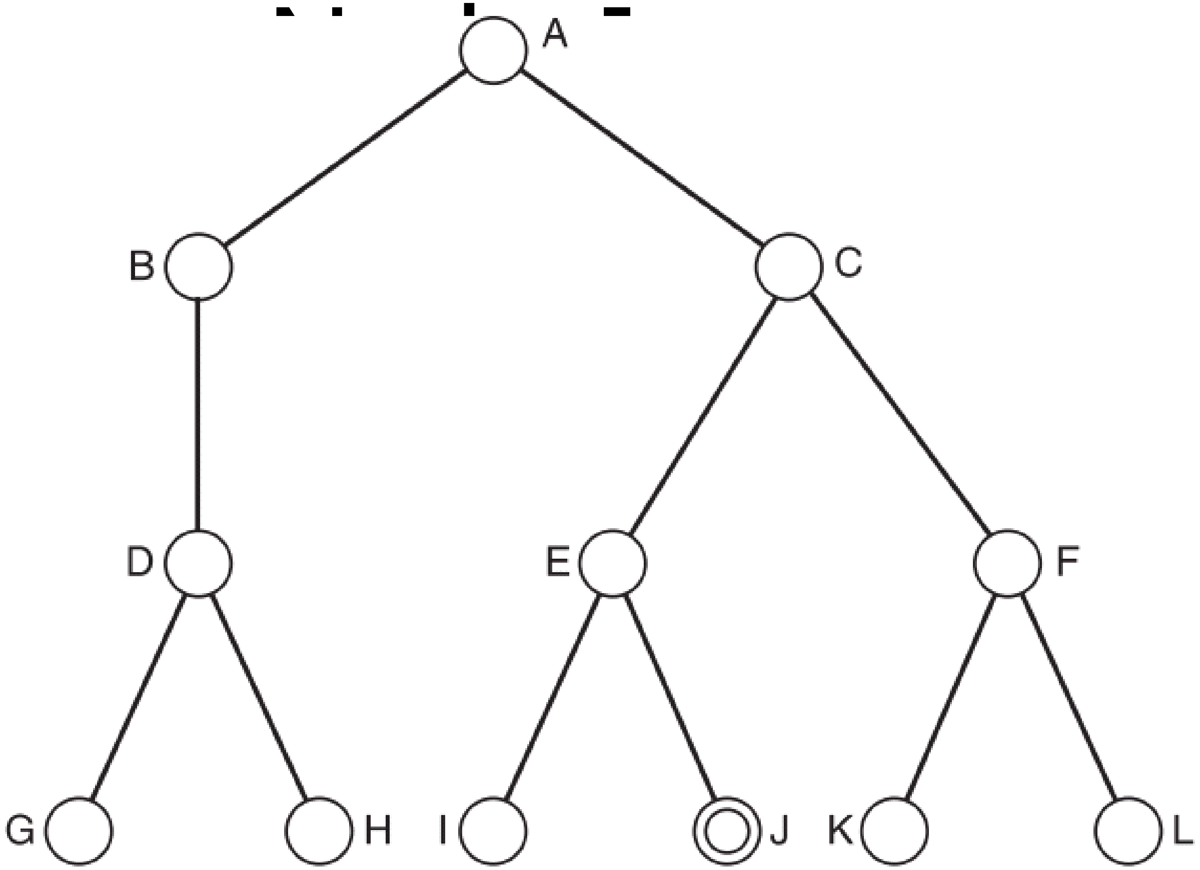
Depth-First Search Goal -



Current

A C
B E
D
G
D
H
D
B
A

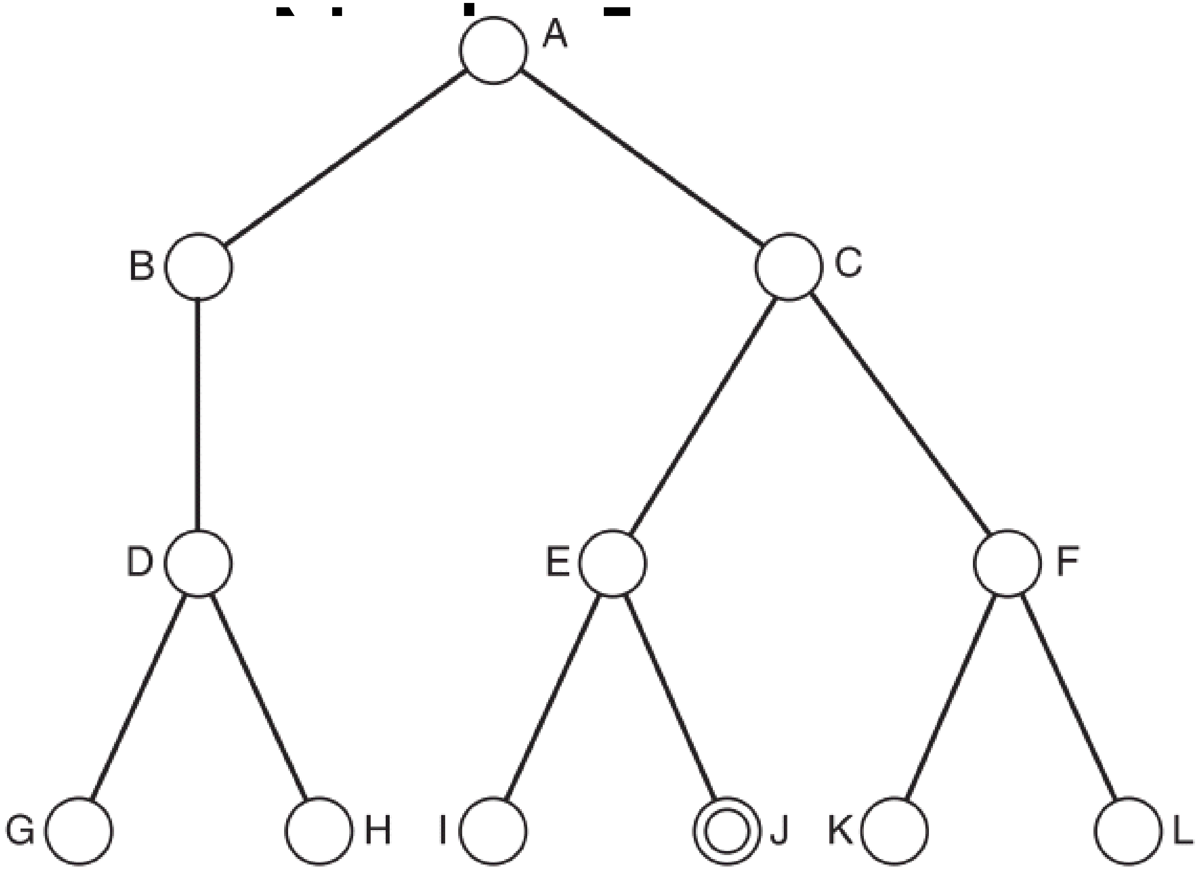
Depth-First Search Goal -



Current

A	C
B	E
D	
G	
D	
H	
D	
B	
A	

Depth-First Search Goal -

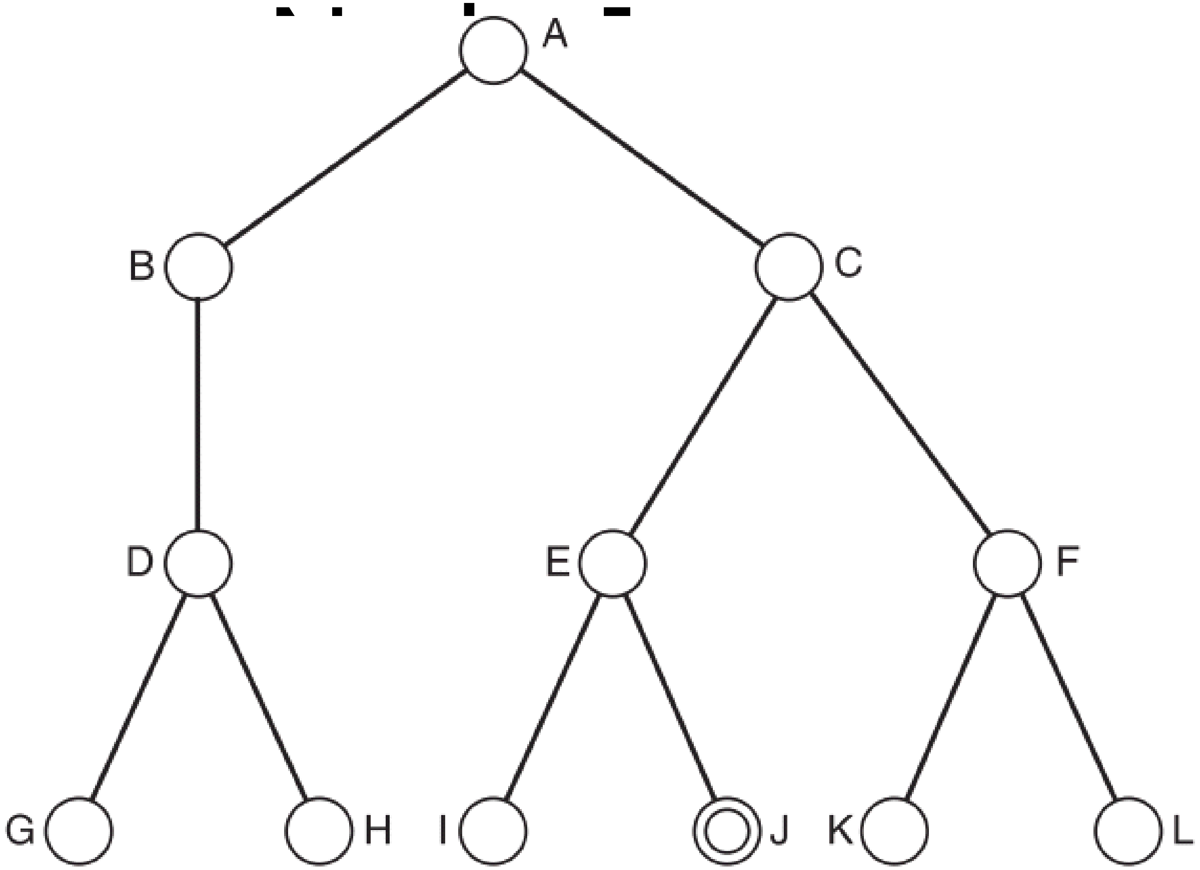


Current

A
B
D
G
D
H
D
B
A

C
E
I

Depth-First Search Goal -

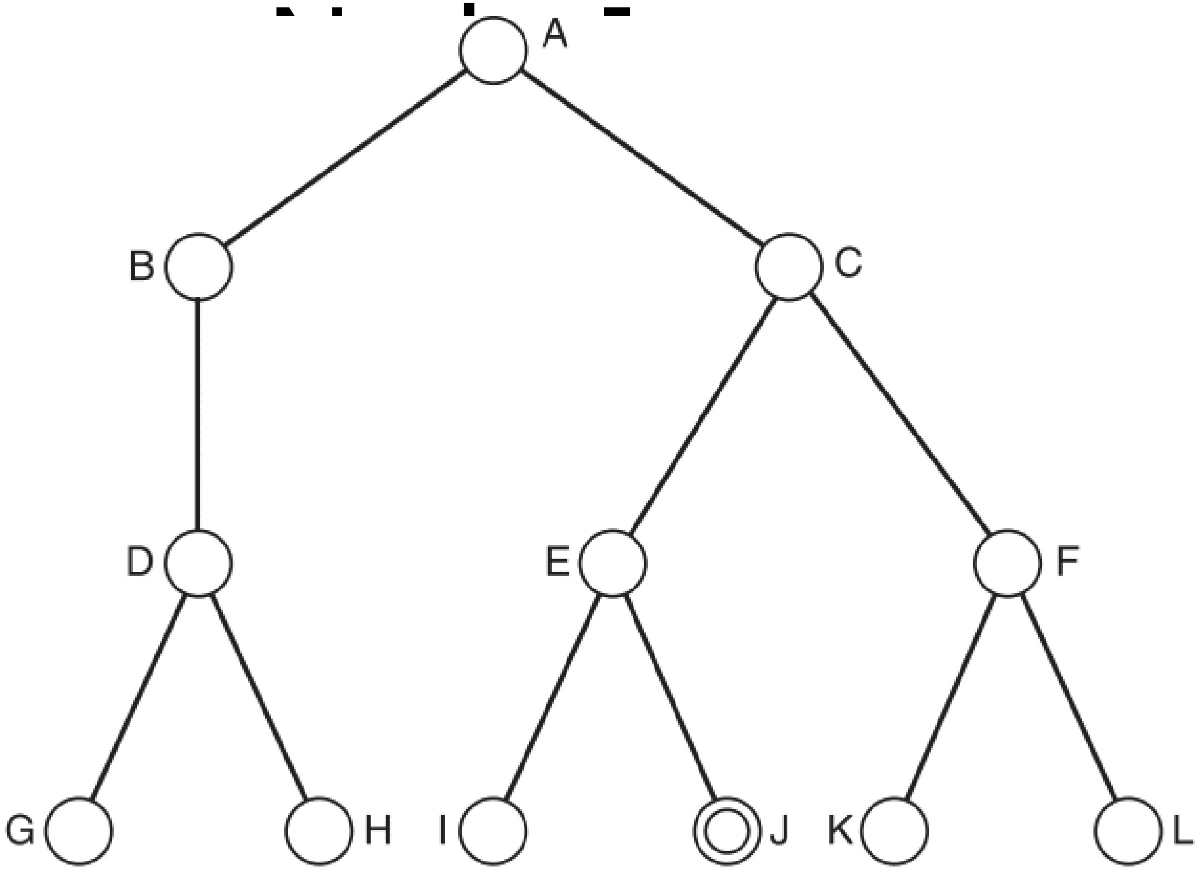


Current

A
B
D
G
D
H
D
B
A

C
E
I

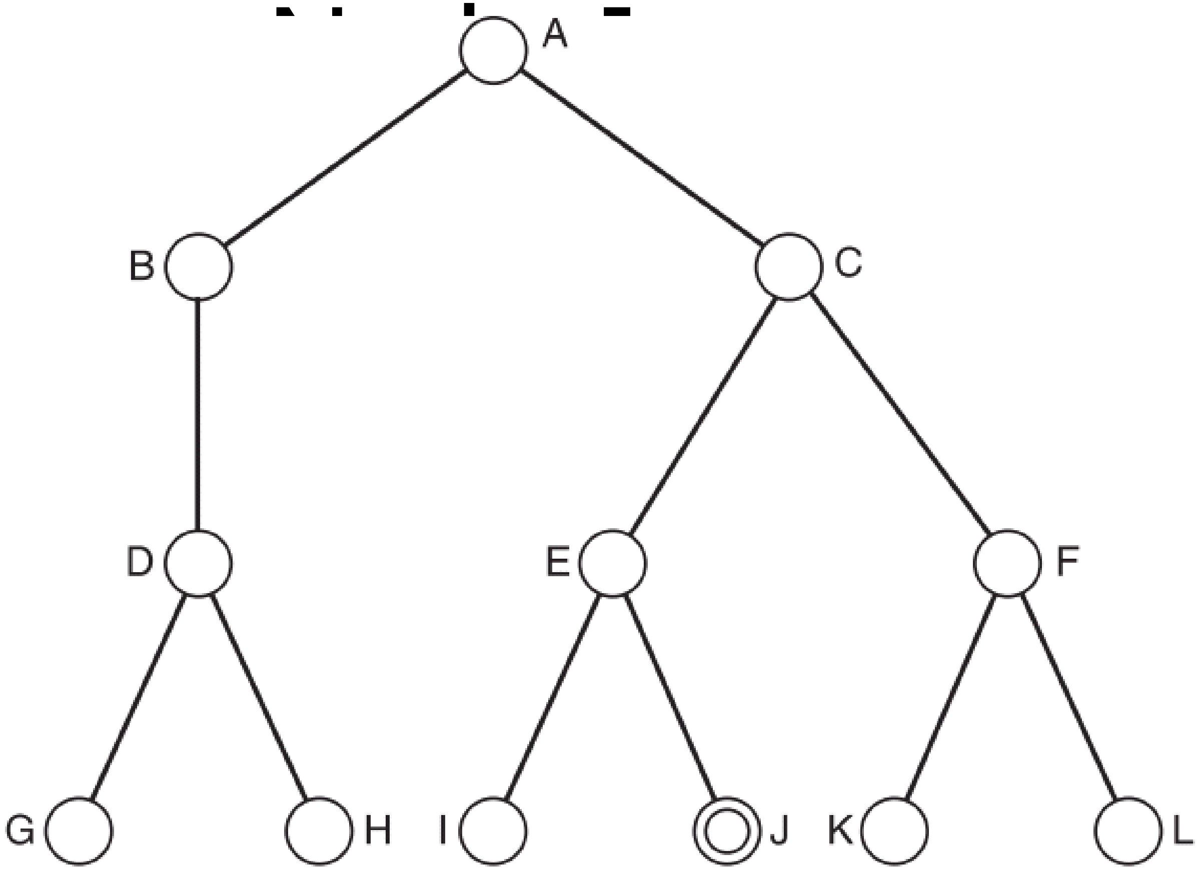
Depth-First Search Goal -



Current

A	C
B	E
D	I
G	E
D	
H	
D	
B	
A	

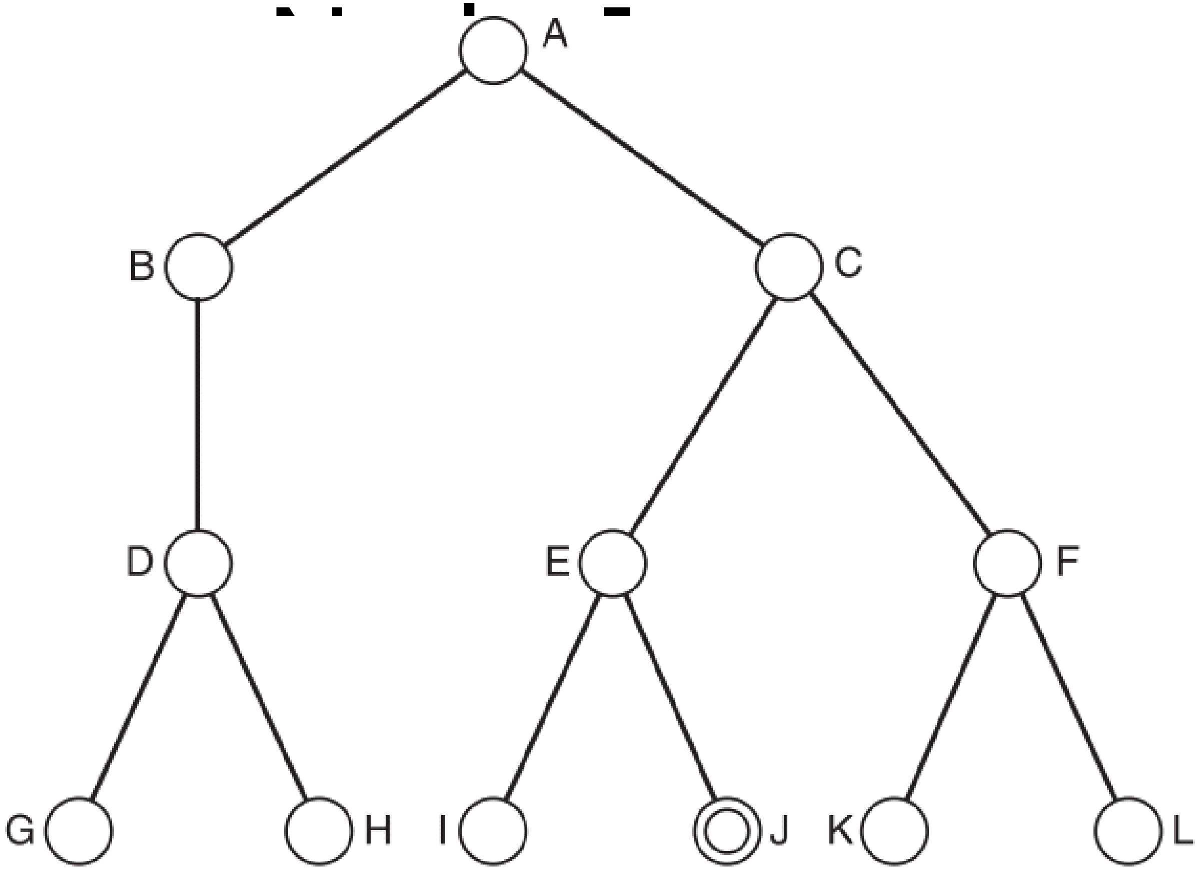
Depth-First Search Goal -

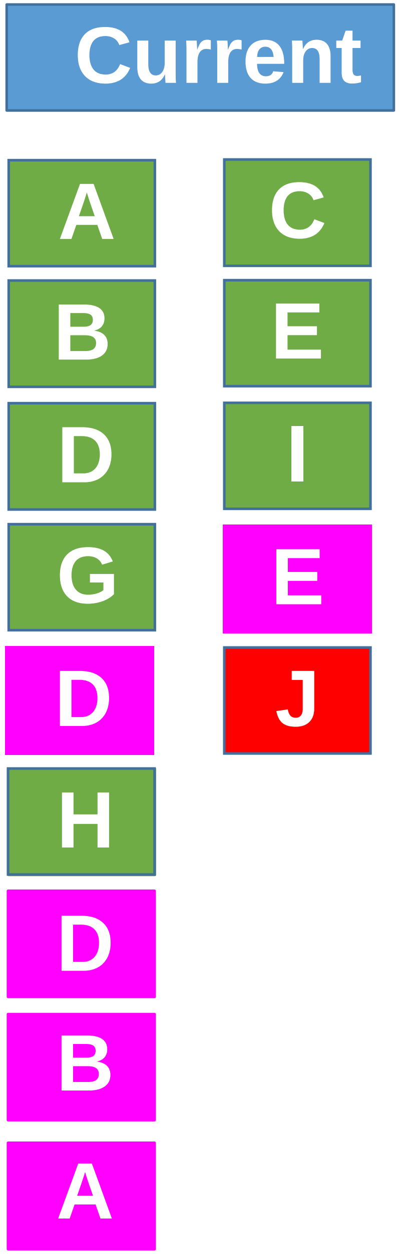


Current

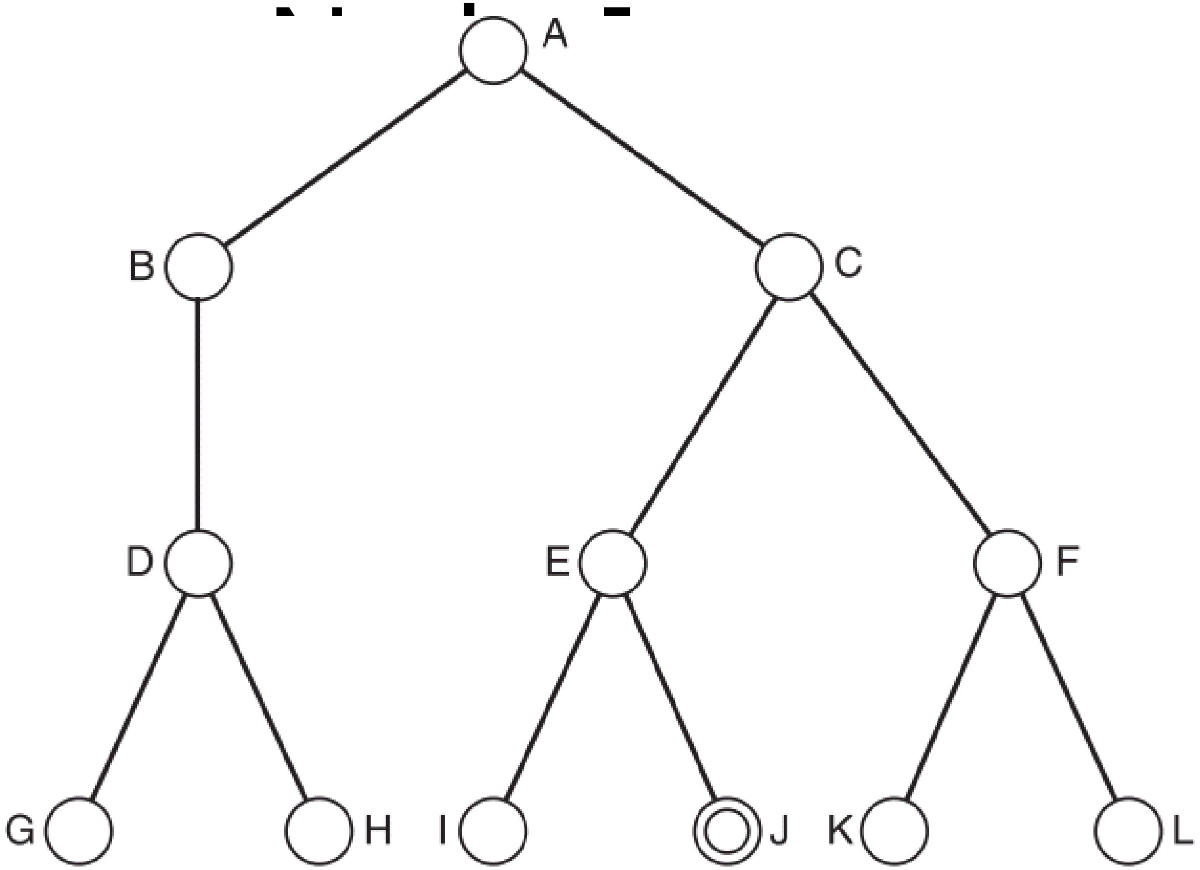
A C
B E
D I
G E
D J
H
D
B
A

Depth-First Search Goal -





Depth-First Search Goal -



Current

A
B
D
G
D
H
D
B
A

C
E
I
E
J

GOAL

Analyzing DFS

- **Not Optimal**
- **Not Complete**
- Time Complexity-**O(b^m)**-(m is maximum depth of any node)
- Space Complexity-**O(bm)**

UNIFORM-COST SEARCH

Fixing BFS To Get An Optimal Path

- Use a **priority queue** instead of a simple queue
- Insert nodes in the **increasing order** of the cost of the path so far
- Guaranteed to find an **optimal solution!**
- This algorithm is called **uniform-cost search**

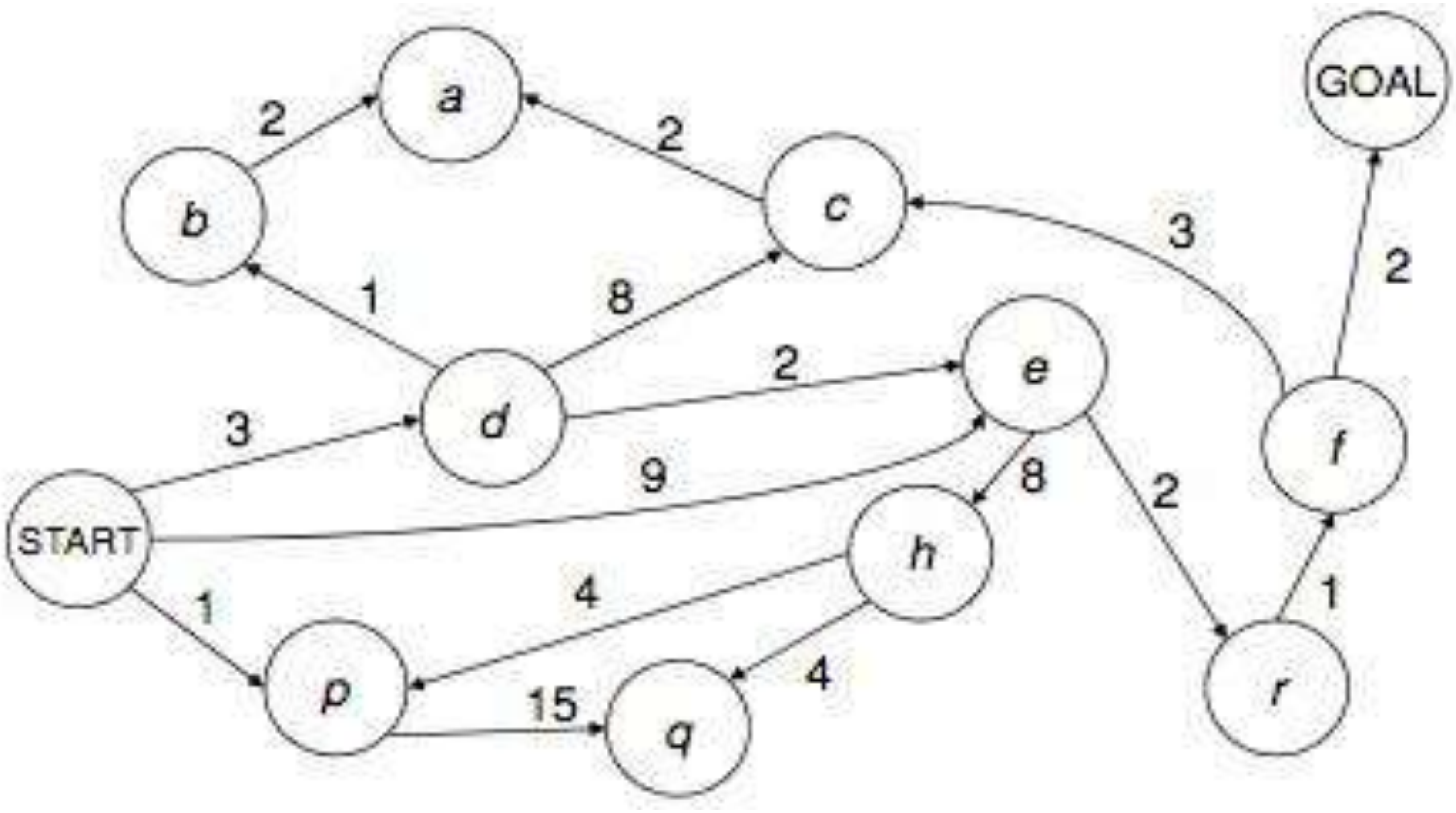
Continued

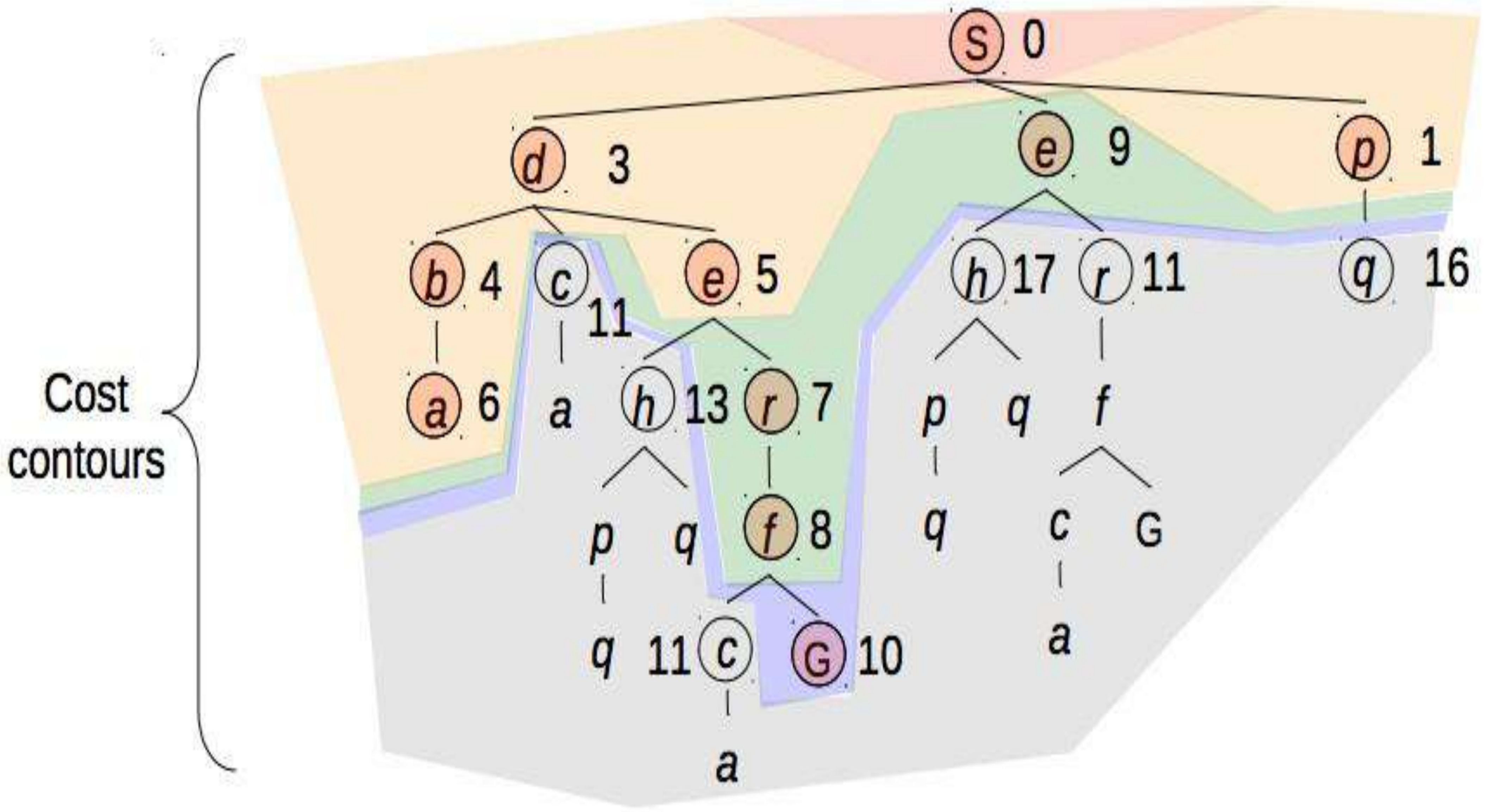
- ❖Instead of Expanding shallowest node the node n with the Lowest Path Cost g(n) is expanded
- ❖Differences

  --Goal Test is applied to a node when it is selected for expansion

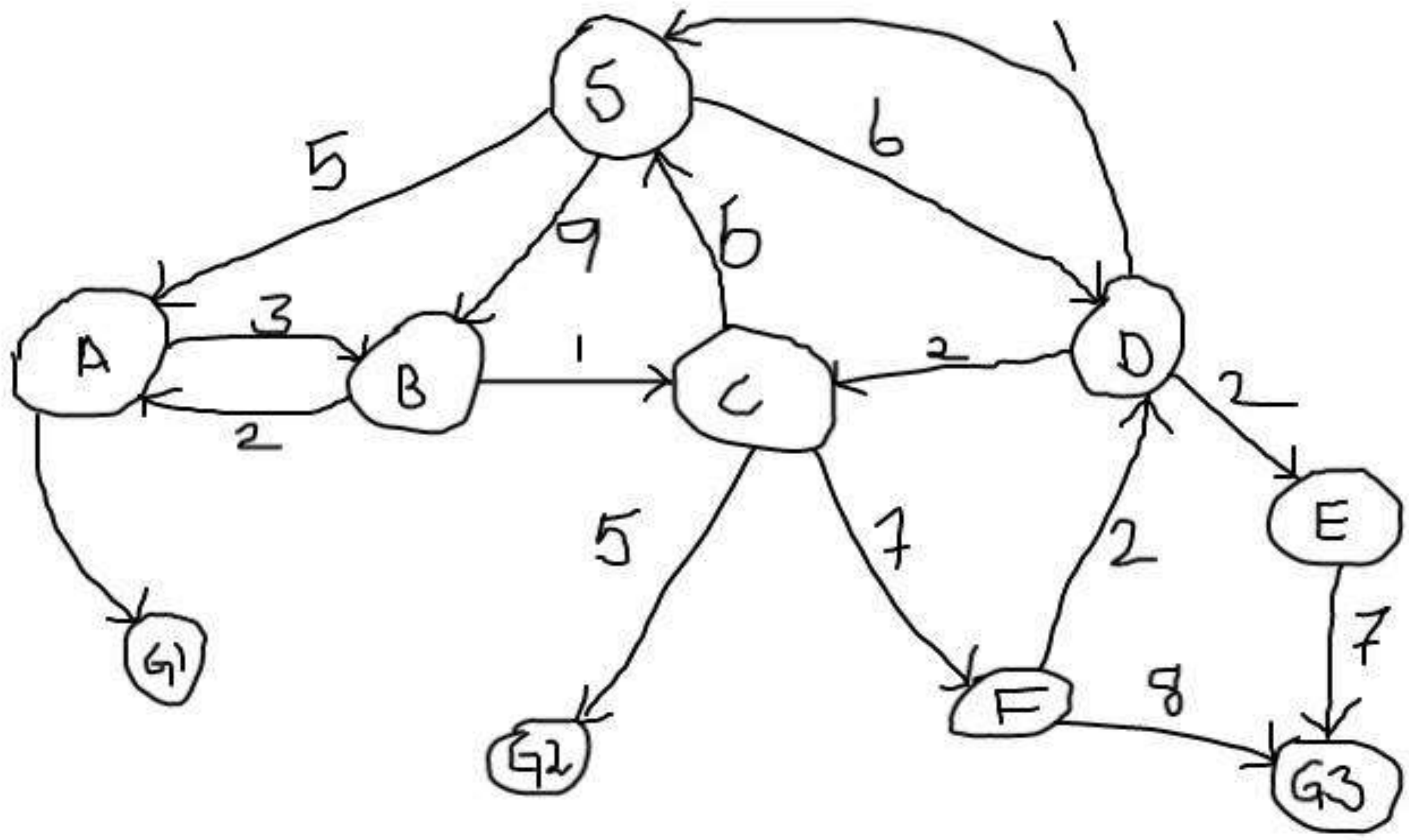
  --Test is added in case a better path is found to a node currently on frontier

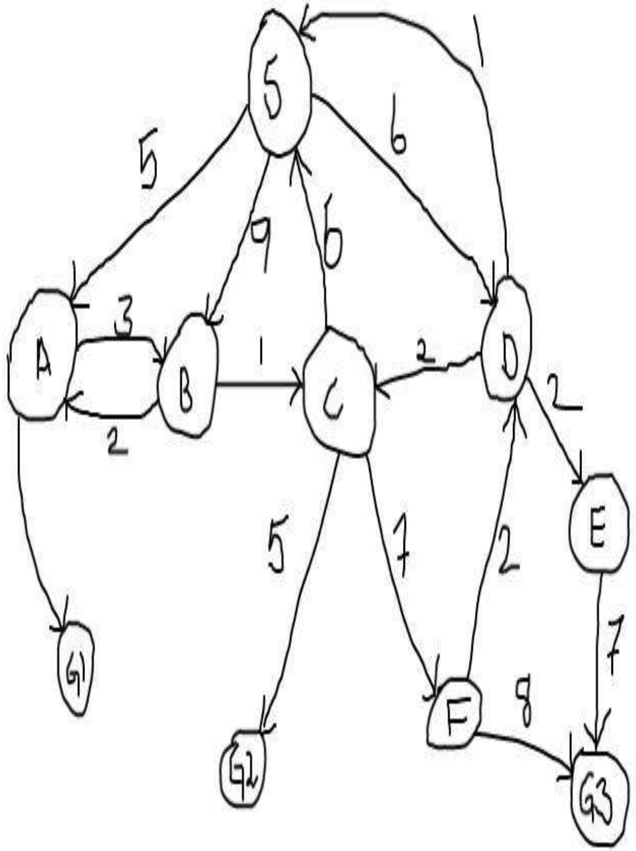
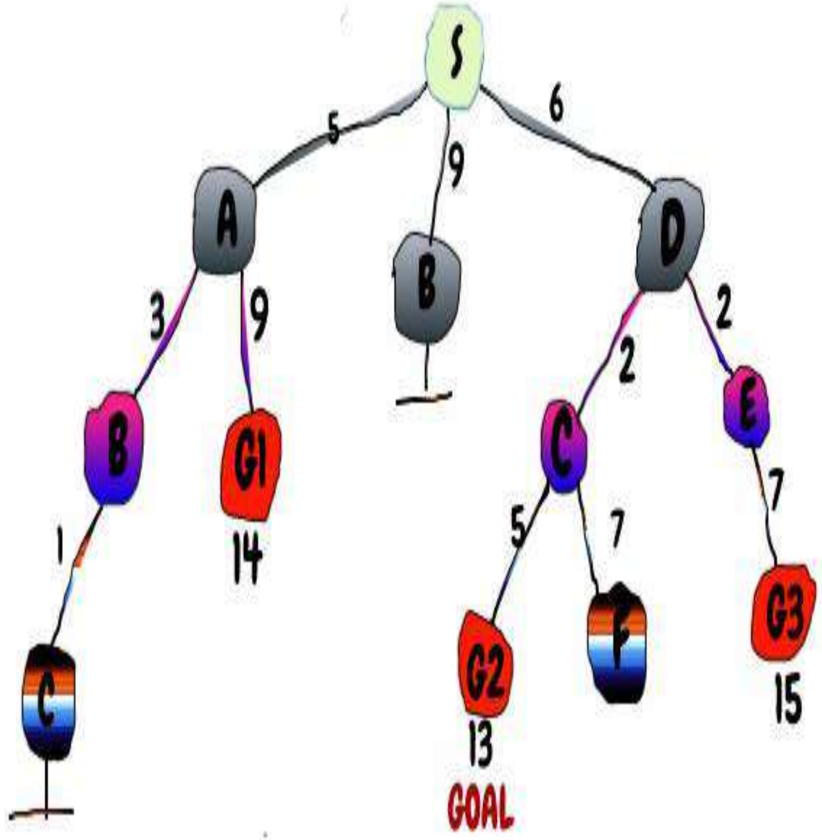
Example 1



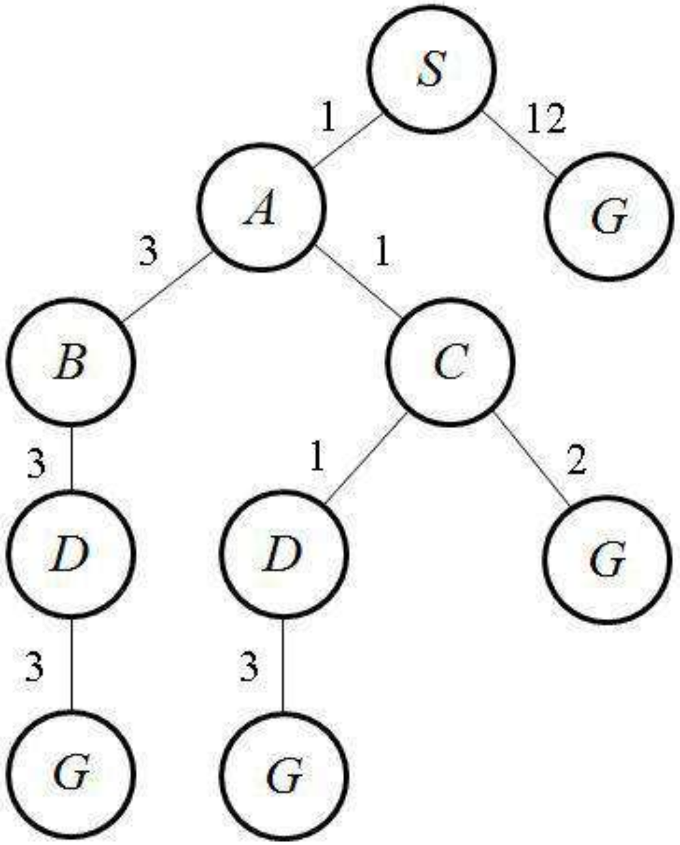

Cost contours
S 0
d 3
e 9
p 1
b 4
c 11
e 5
h 17
r 11
q 16
a 6
a
h 13
r 7
p
q
f
p
q
f 8
q
c
G
q
11 c
G 10
a
a

Example 2



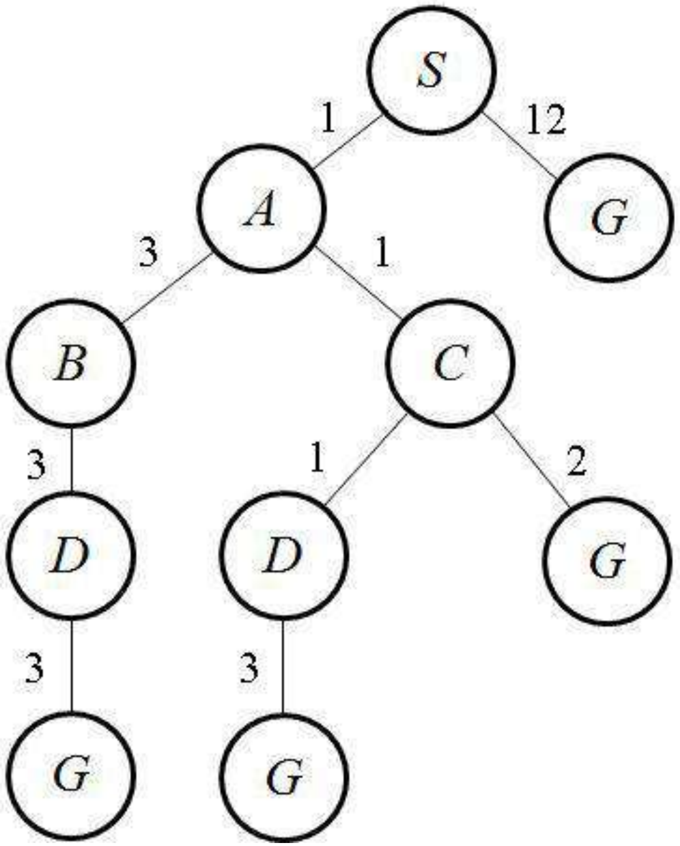

S
1
5
6
9
6
A
3
B
1
C
2
D
2
E
2
5
7
2
7
G1
G2
F
8
G3

S
5
9
6
A
B
D
3
9
2
2
B
G1
C
E
14
1
5
7
7
C
G2
F
G3
13
15
GOAL

Uniform Cost Search Goal - Node G



Current	Waiting Ordered
S_0	---

Uniform Cost Search Goal - Node G

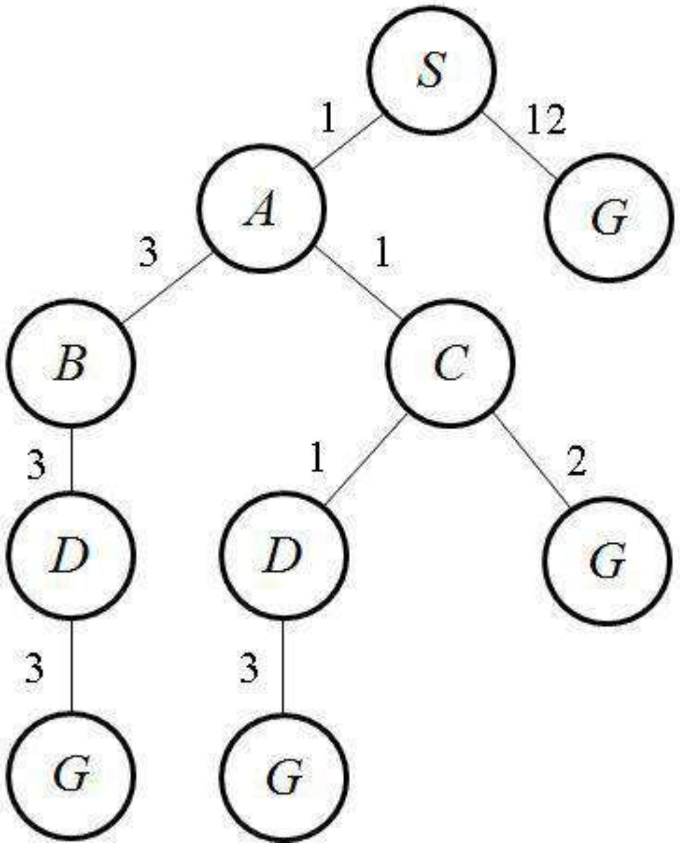


Current

S_0

Waiting Ordered

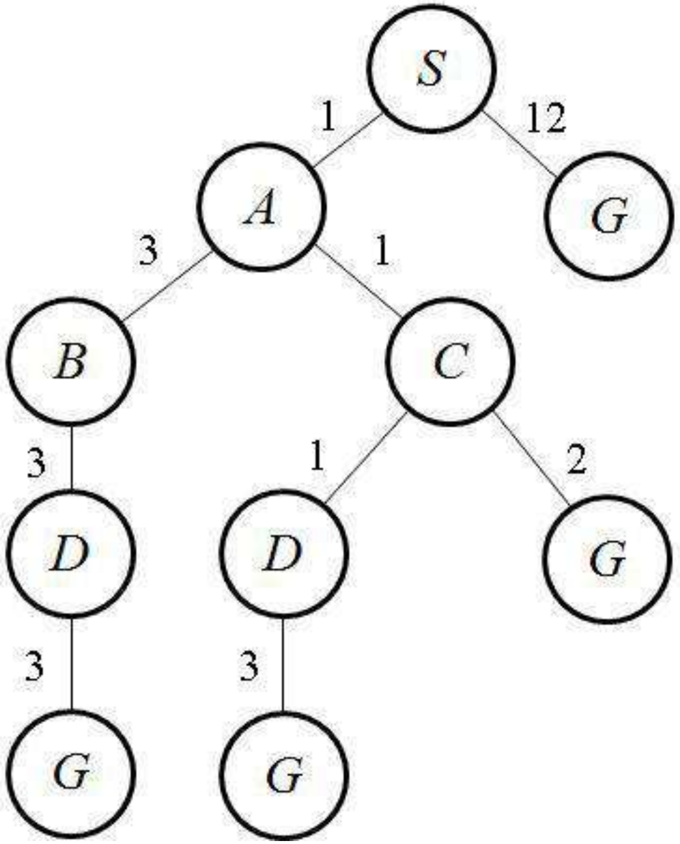
Uniform Cost Search Goal - Node G



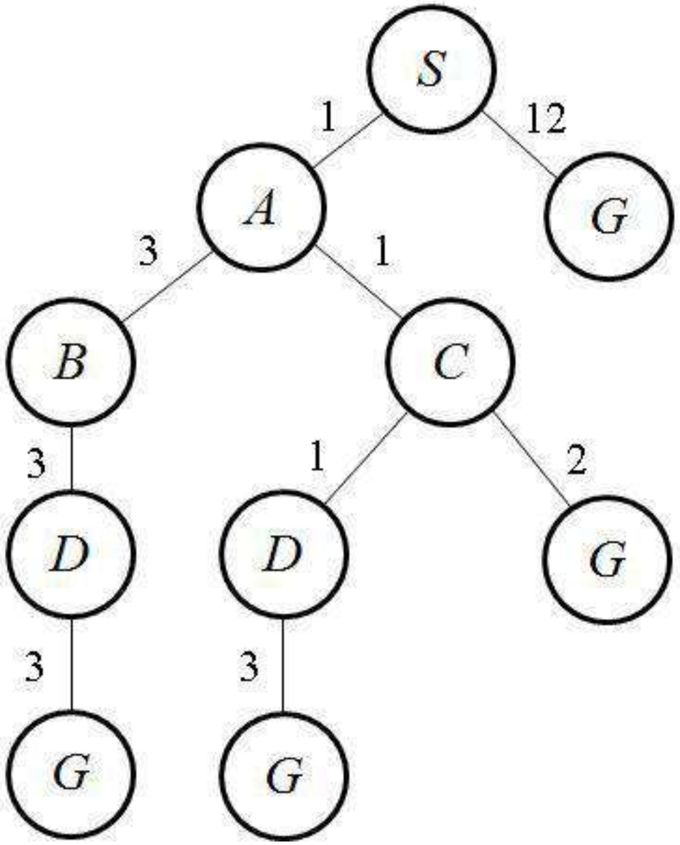
Current	Waiting Ordered
S_0	---
S_0	

Uniform Cost Search Goal - Node G

Current	Waiting Ordered
S_0	---
S_0	A_1, G_{12}

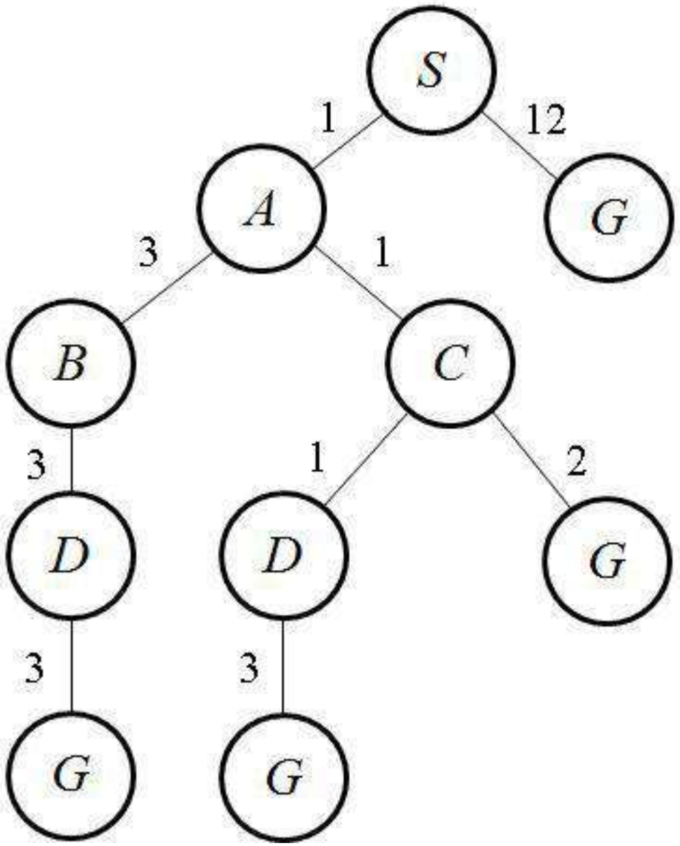


Uniform Cost Search Goal - Node G



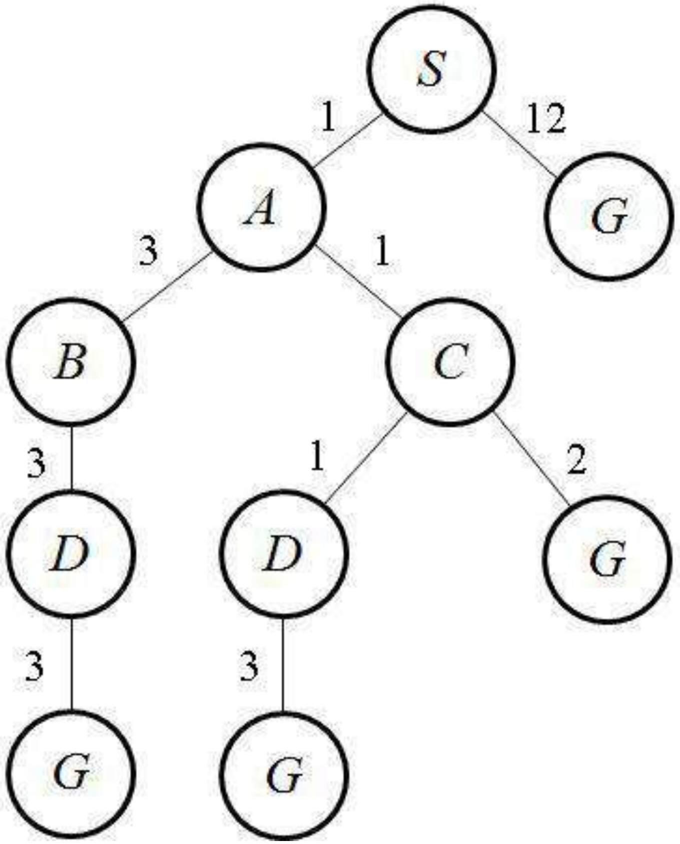
Current	Waiting Ordered
S_0	---
S_0	A_1, G_{12}
A_1	

Uniform Cost Search Goal - Node G



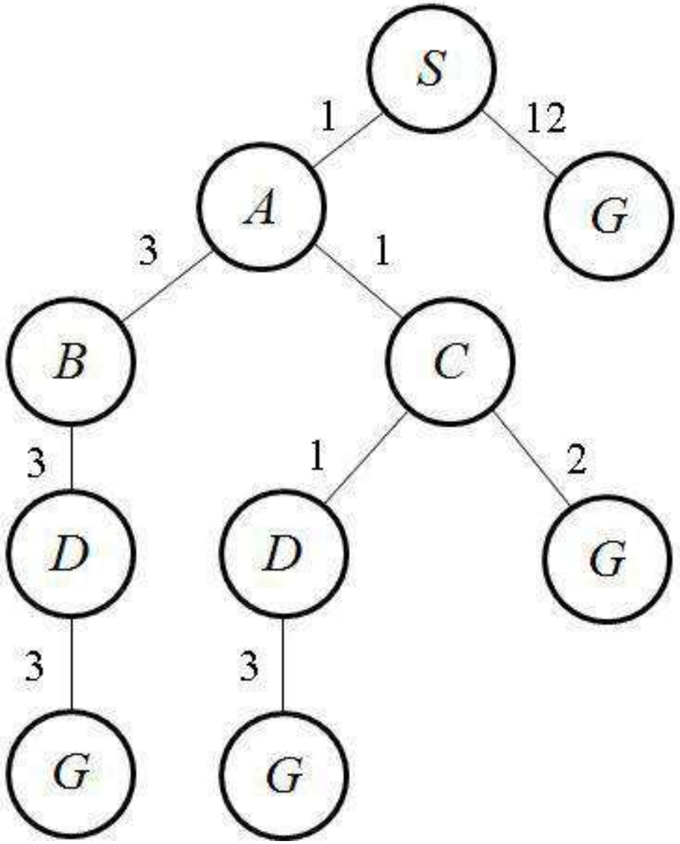
Current	Waiting Ordered
S_0	---
S_0	A_1, G_{12}
A_1	G_{12}

Uniform Cost Search Goal - Node G



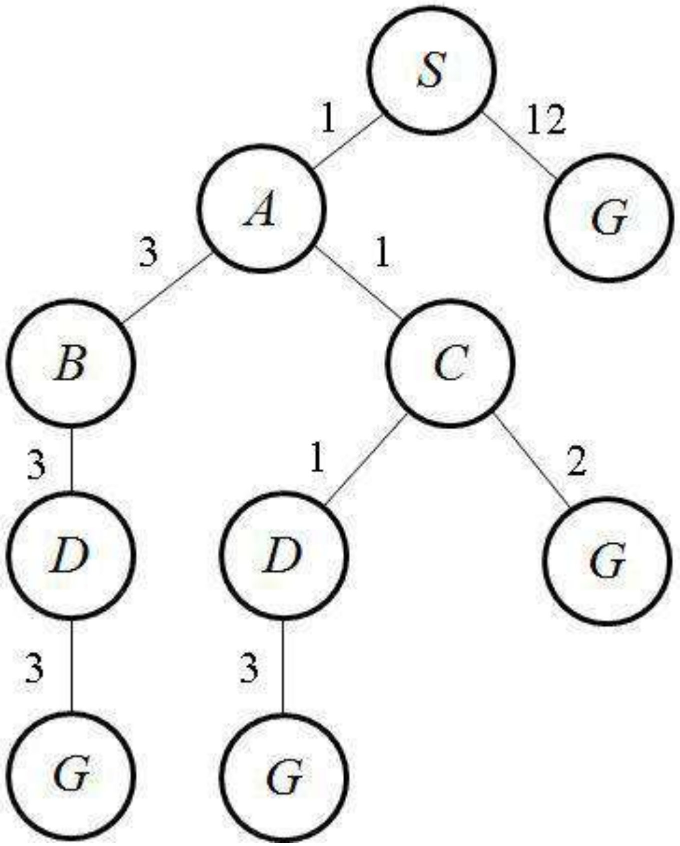
Current	Waiting Ordered
S_0	---
S_0	A_1, G_{12}
A_1	G_{12}

Uniform Cost Search Goal - Node G



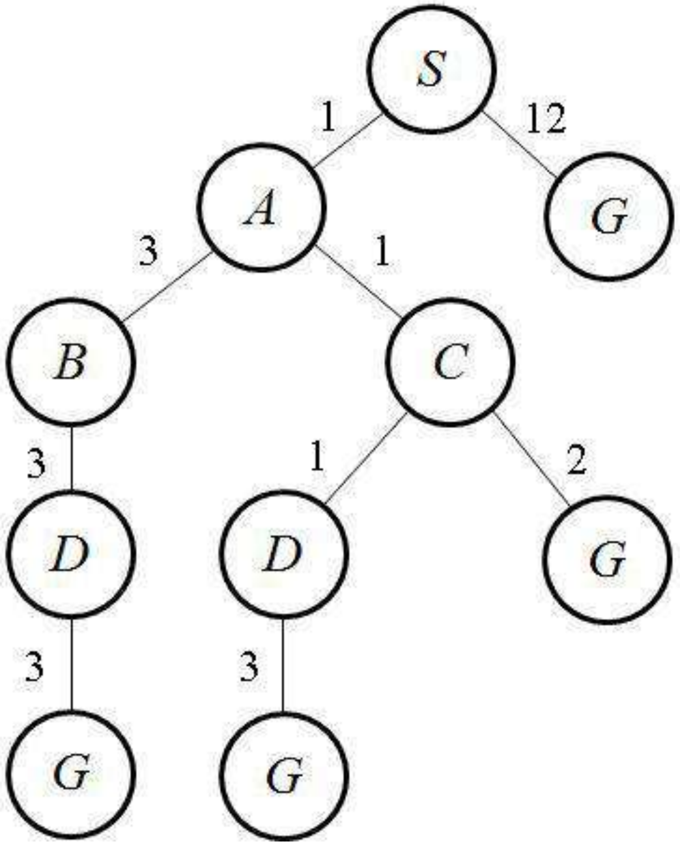
Current	Waiting Ordered
S_0	---
S_0	A_1, G_{12}
A_1	G_{12}
A_1	

Uniform Cost Search Goal - Node G



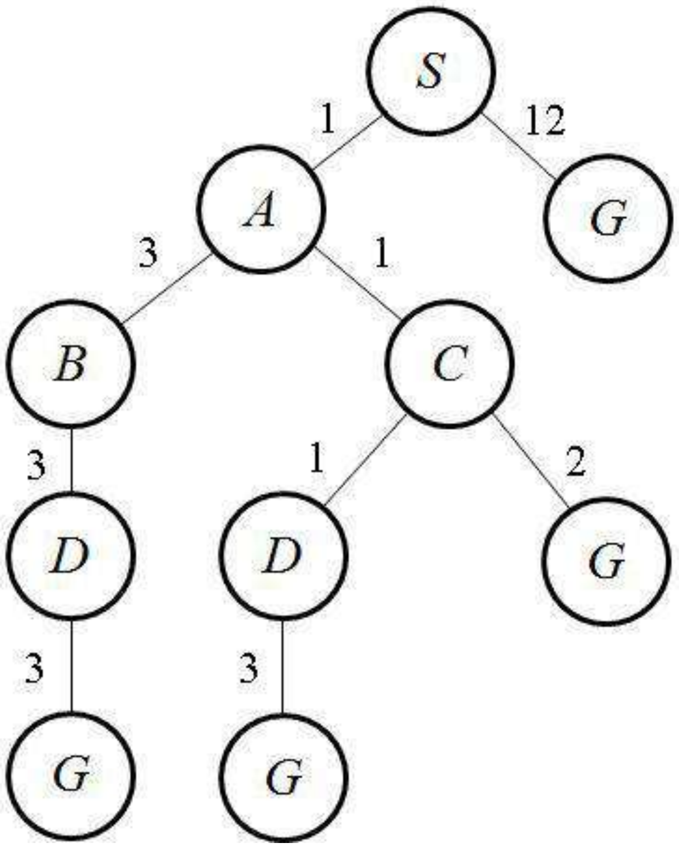
Current	Waiting Ordered
S_0	---
S_0	A_1, G_{12}
A_1	G_{12}
A_1	C_2, B_4, G_{12}

Uniform Cost Search Goal - Node G



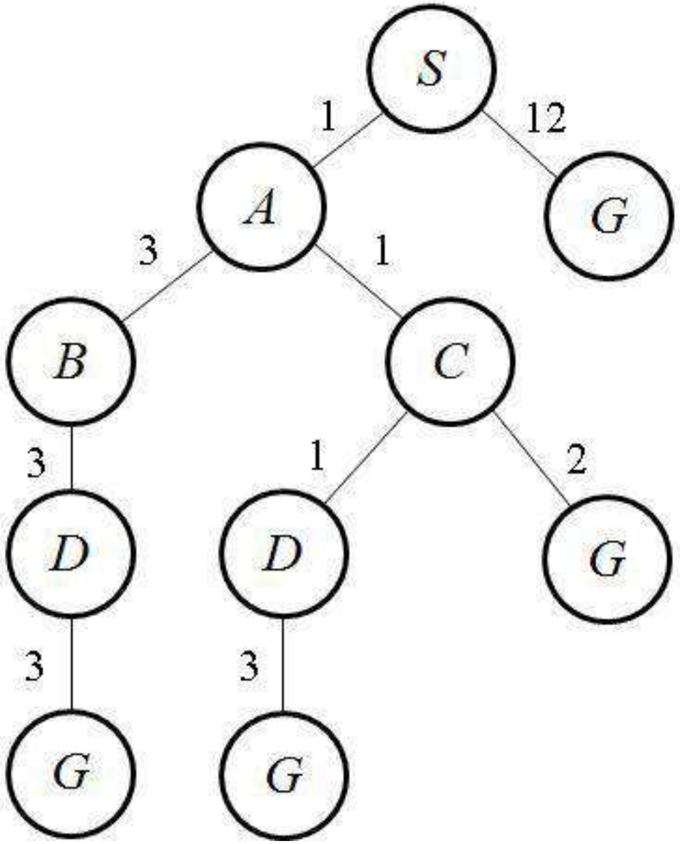
Current	Waiting Ordered
S_0	---
S_0	A_1, G_{12}
A_1	G_{12}
A_1	C_2, B_4, G_{12}
C_2	

Uniform Cost Search Goal - Node G



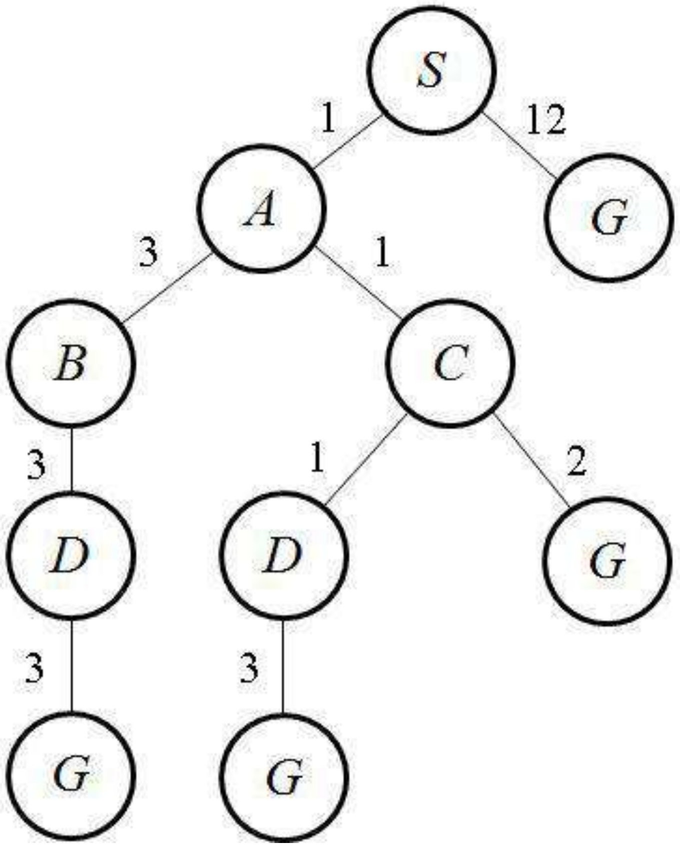
Current	Waiting Ordered
S_0	---
S_0	A_1, G_{12}
A_1	G_{12}
A_1	C_2, B_4, G_{12}
C_2	B_4, G_{12}

Uniform Cost Search Goal - Node G



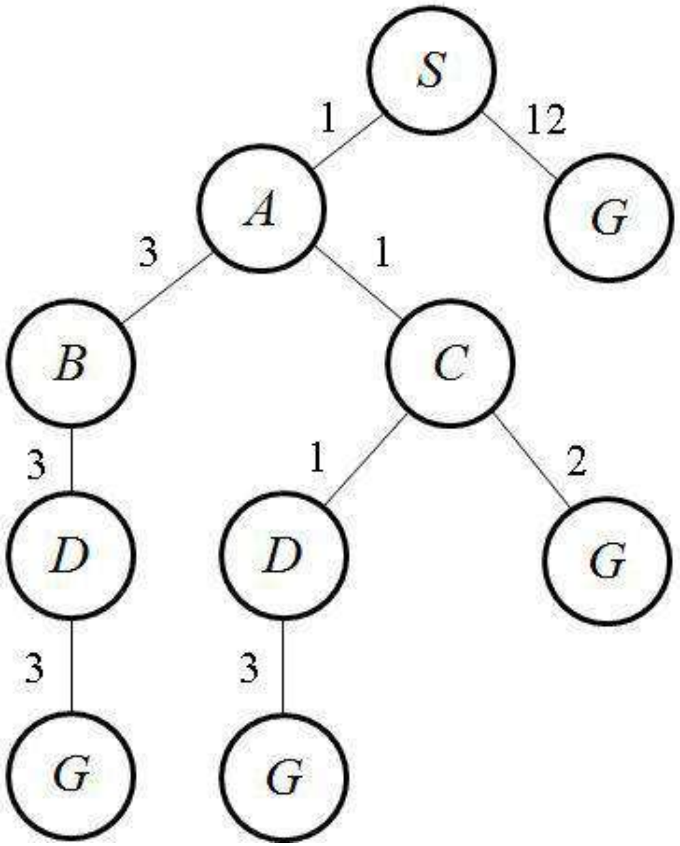
Current	Waiting Ordered
S_0	---
S_0	A_1, G_{12}
A_1	G_{12}
A_1	C_2, B_4, G_{12}
C_2	B_4, G_{12}

Uniform Cost Search Goal - Node G



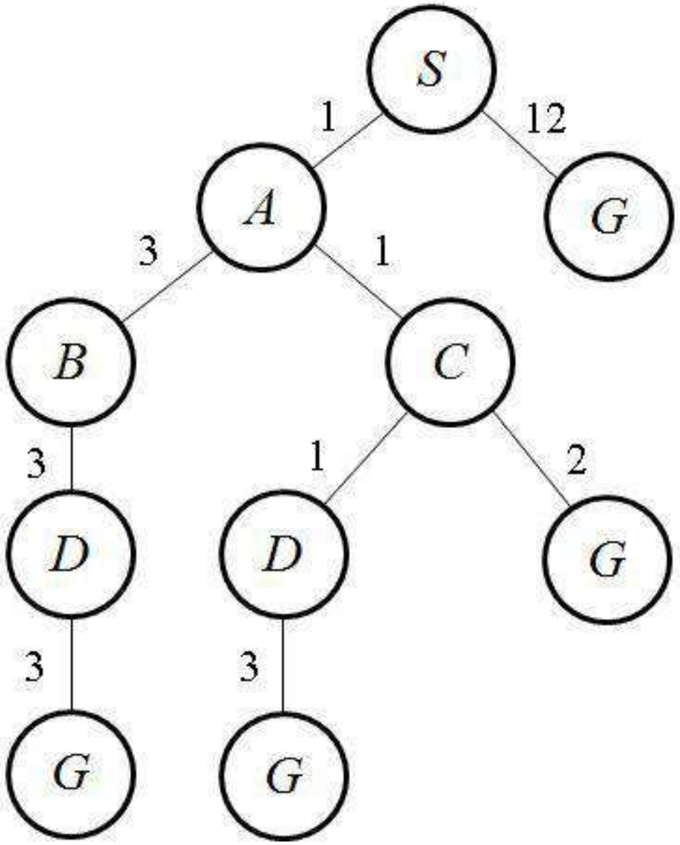
Current	Waiting Ordered
S_0	---
S_0	A_1, G_{12}
A_1	G_{12}
A_1	C_2, B_4, G_{12}
C_2	B_4, G_{12}
C_2	

Uniform Cost Search Goal - Node G



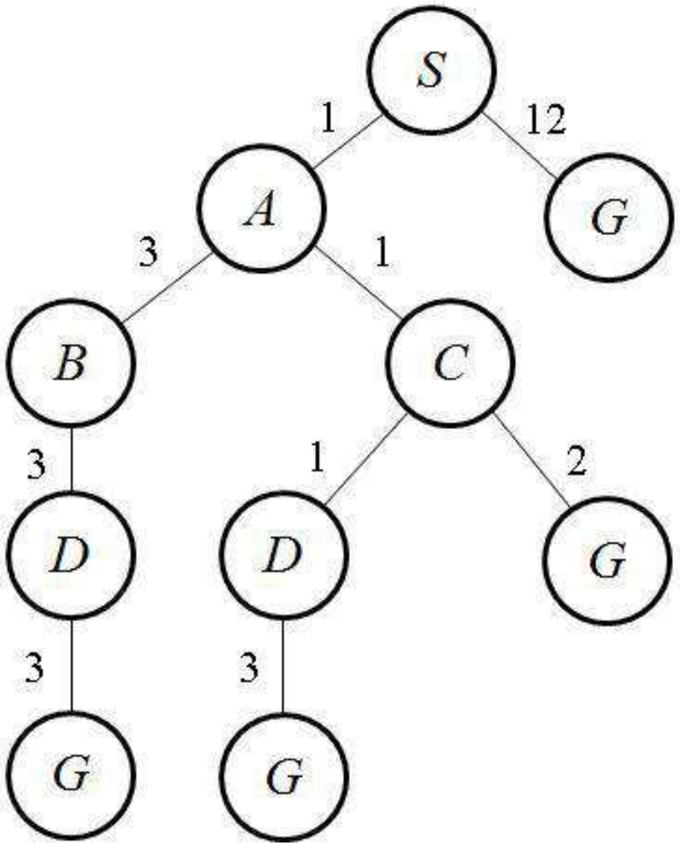
Current	Waiting Ordered
S_0	---
S_0	A_1, G_{12}
A_1	G_{12}
A_1	C_2, B_4, G_{12}
C_2	B_4, G_{12}
C_2	D_3, B_4, G_4, G_{12}

Uniform Cost Search Goal - Node G



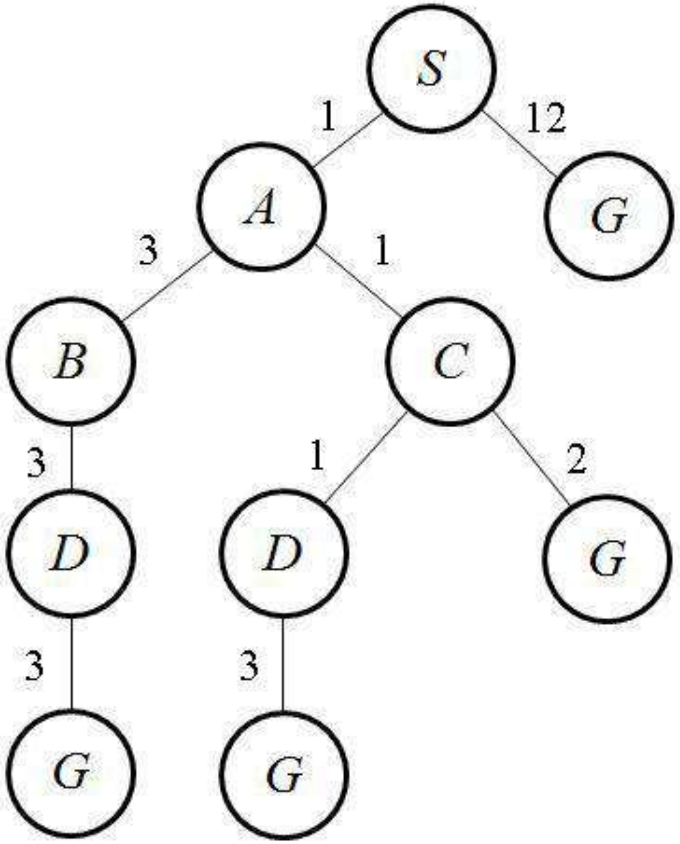
Current	Waiting Ordered
S_0	---
S_0	A_1, G_{12}
A_1	G_{12}
A_1	C_2, B_4, G_{12}
C_2	B_4, G_{12}
C_2	D_3, B_4, G_4, G_{12}
D_3	

Uniform Cost Search Goal - Node G



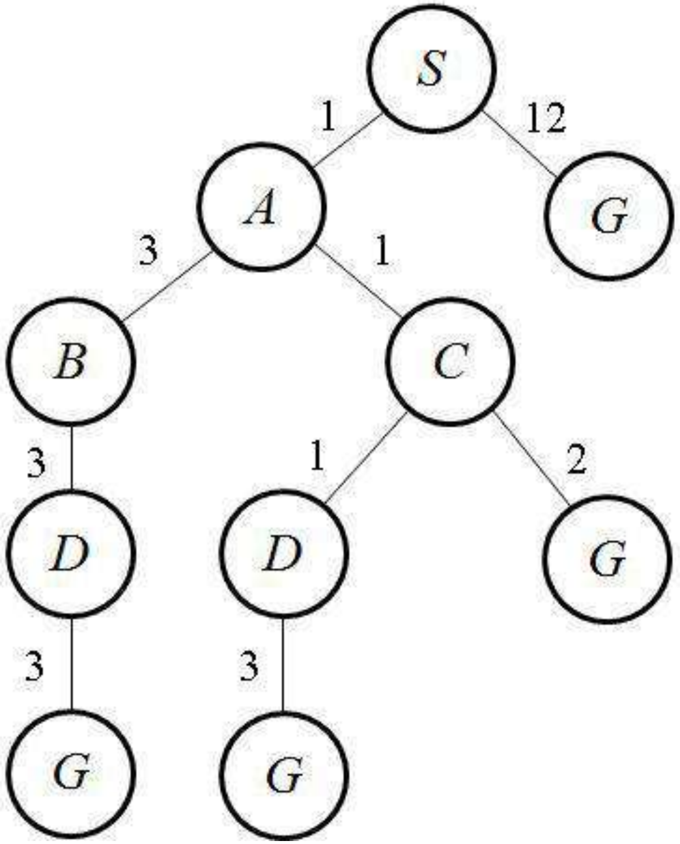
Current	Waiting Ordered
S_0	---
S_0	A_1, G_{12}
A_1	G_{12}
A_1	C_2, B_4, G_{12}
C_2	B_4, G_{12}
C_2	D_3, B_4, G_4, G_{12}
D_3	B_4, G_4, G_{12}

Uniform Cost Search Goal - Node G



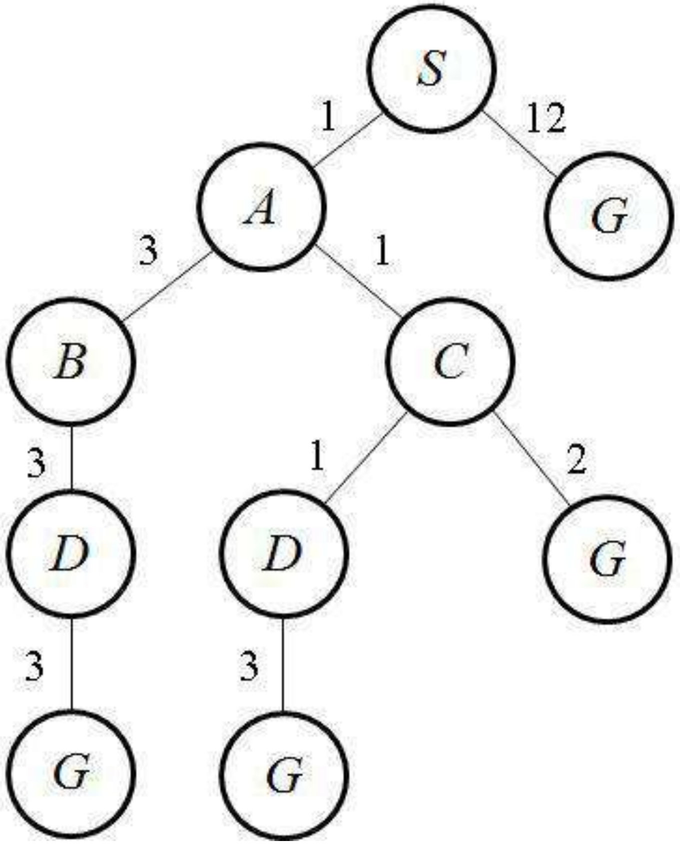
Current	Waiting Ordered
S_0	---
S_0	A_1, G_{12}
A_1	G_{12}
A_1	C_2, B_4, G_{12}
C_2	B_4, G_{12}
C_2	D_3, B_4, G_4, G_{12}
D_3	B_4, G_4, G_{12}

Uniform Cost Search Goal - Node G



Current	Waiting Ordered
S_0	---
S_0	A_1, G_{12}
A_1	G_{12}
A_1	C_2, B_4, G_{12}
C_2	B_4, G_{12}
C_2	D_3, B_4, G_4, G_{12}
D_3	B_4, G_4, G_{12}
D_3	

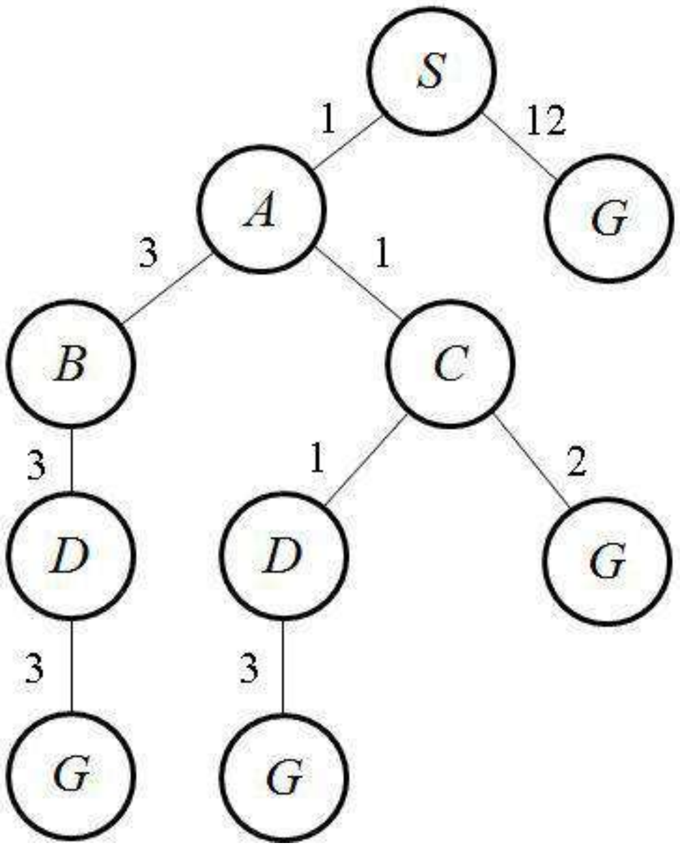
Uniform Cost Search Goal - Node G



Current	Waiting Ordered
S_0	---
S_0	A_1, G_{12}
A_1	G_{12}
A_1	C_2, B_4, G_{12}
C_2	B_4, G_{12}
C_2	D_3, B_4, G_4, G_{12}
D_3	B_4, G_4, G_{12}
D_3	B_4, G_4, G_6, G_{12}

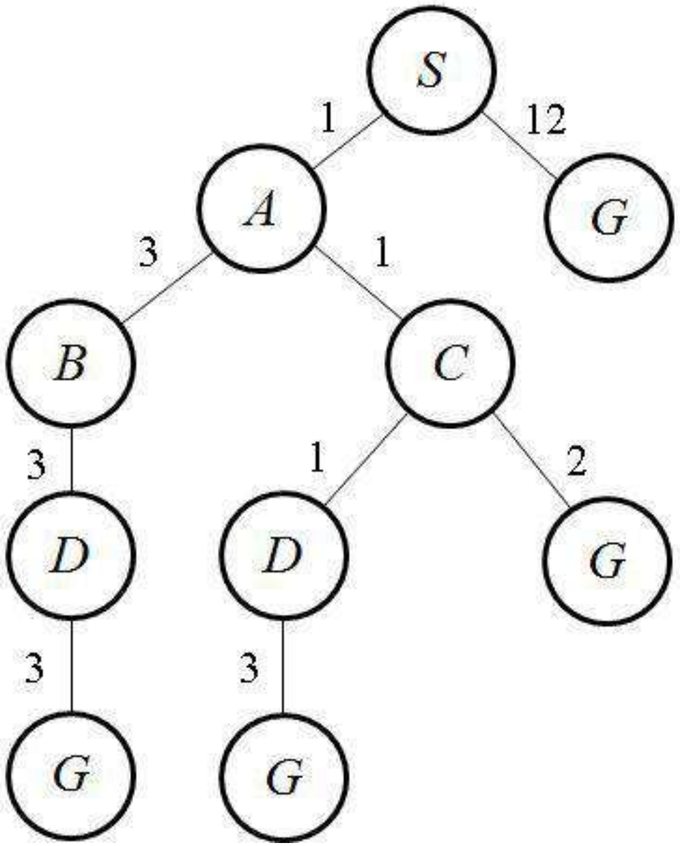
Uniform Cost Search Goal - Node G

Current	Waiting Ordered
B_4	G_4, G_6, G_{12}

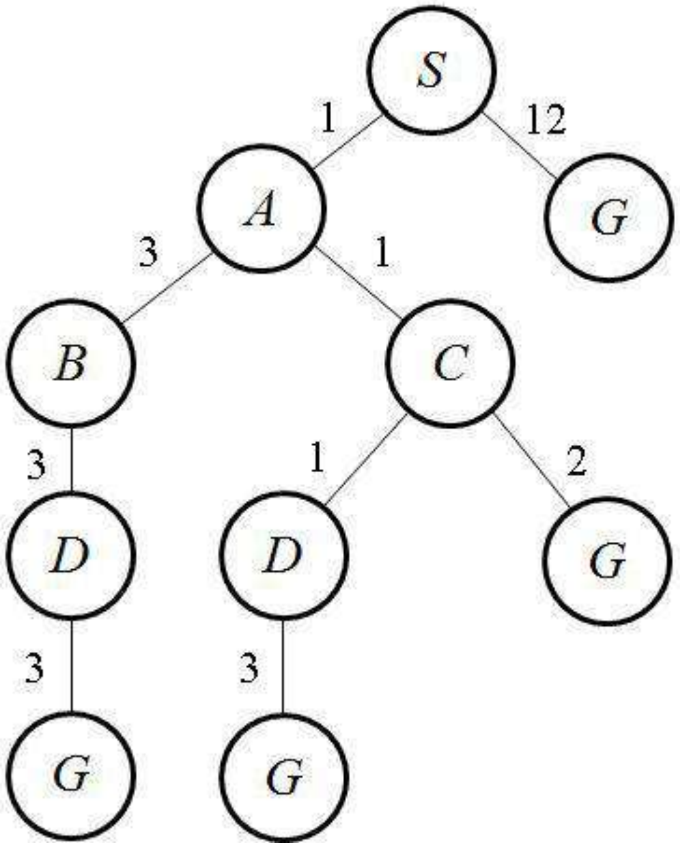


Uniform Cost Search Goal - Node G

Current	Waiting Ordered
B_4	G_4, G_6, G_{12}

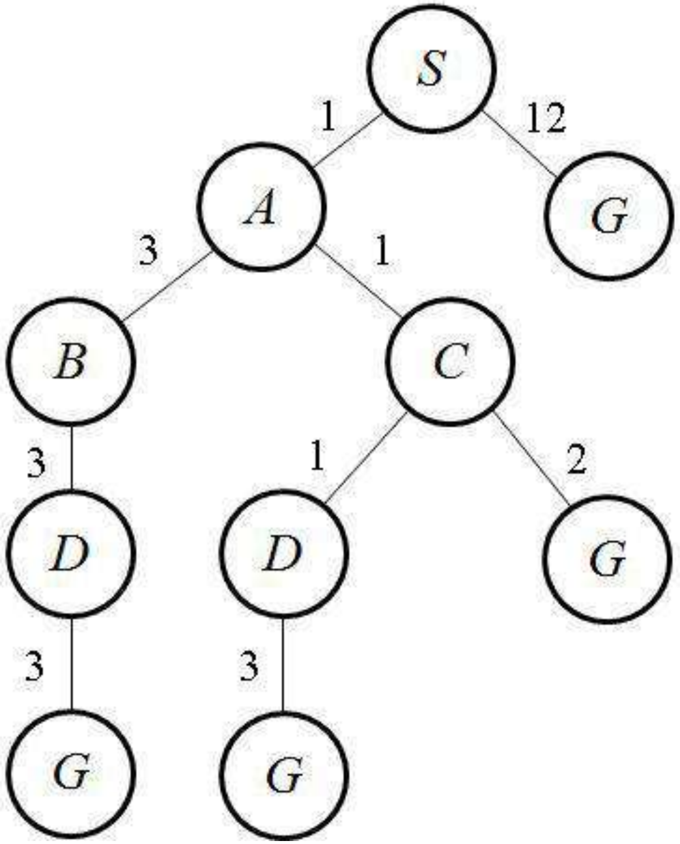


Uniform Cost Search Goal - Node G



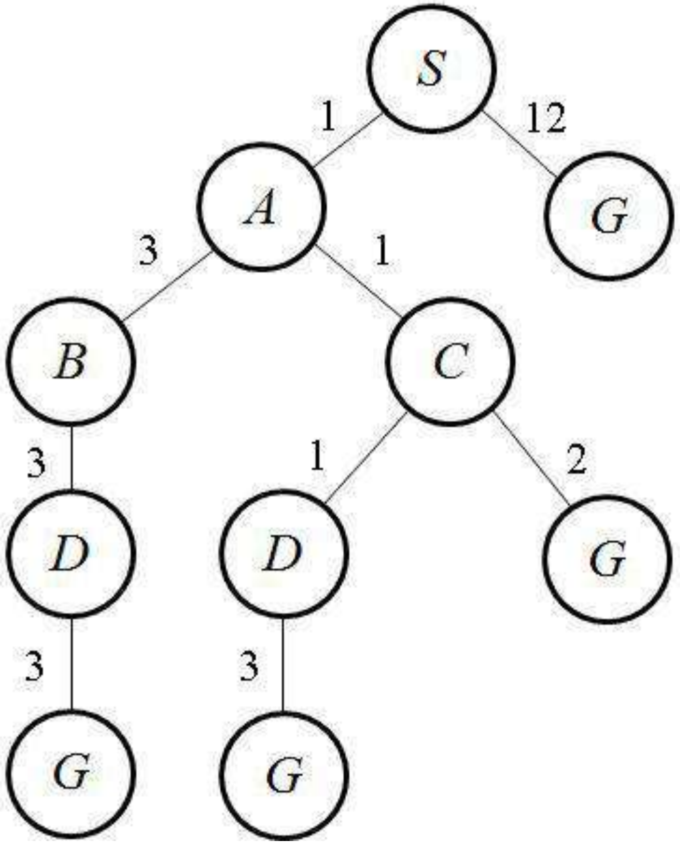
Current	Waiting Ordered
B_4	G_4, G_6, G_{12}
B_4	

Uniform Cost Search Goal - Node G



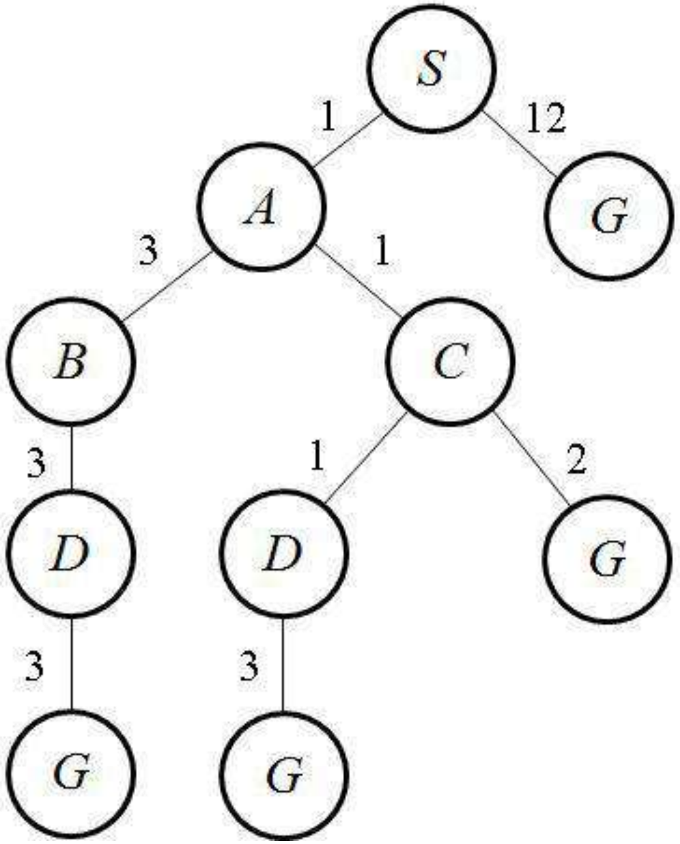
Current	Waiting Ordered
B_4	G_4, G_6, G_{12}
B_4	G_4, G_6, D_7, G_{12}

Uniform Cost Search Goal - Node G



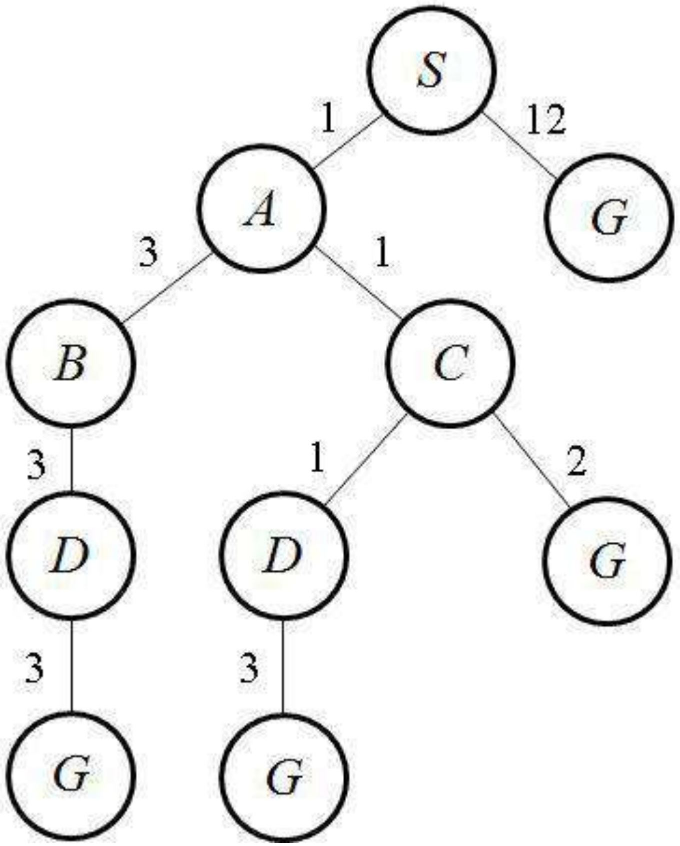
Current	Waiting Ordered
B_4	G_4, G_6, G_{12}
B_4	G_4, G_6, D_7, G_{12}
G_4	

Uniform Cost Search Goal - Node G



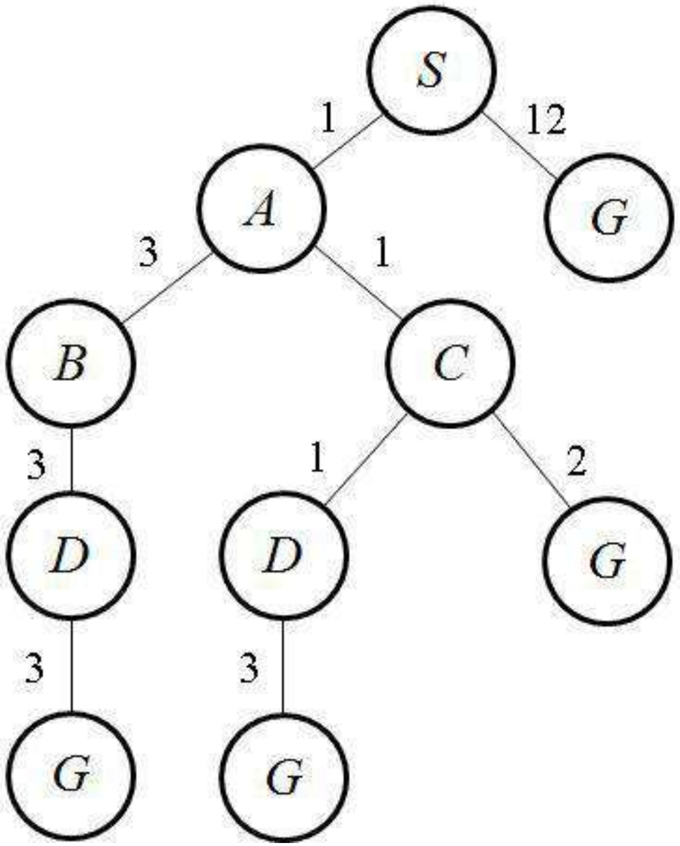
Current	Waiting Ordered
B_4	G_4, G_6, G_{12}
B_4	G_4, G_6, D_7, G_{12}
G_4	G_6, D_7, G_{12}

Uniform Cost Search Goal - Node G



Current	Waiting Ordered
B_4	G_4, G_6, G_{12}
B_4	G_4, G_6, D_7, G_{12}
G_4	G_6, D_7, G_{12}

Uniform Cost Search Goal - Node G

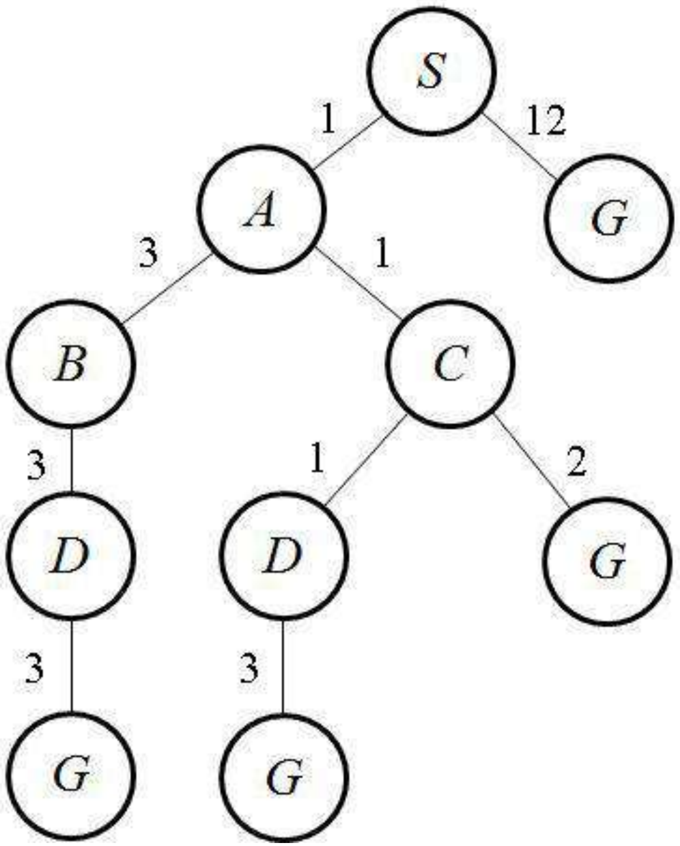


Current	Waiting Ordered
B_4	G_4, G_6, G_{12}
B_4	G_4, G_6, D_7, G_{12}
G_4	G_6, D_7, G_{12}

GOAL

Uniform Cost Search Goal - Node G

Current	Waiting Ordered
B_4	G_4, G_6, G_{12}
B_4	G_4, G_6, D_7, G_{12}
G_4	G_6, D_7, G_{12}



GOAL

Solve using BFS & DFS Compare Costs

```
function UNIFORM-COST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  frontier ← a priority queue ordered by PATH-COST, with node as the only element
  explored ← an empty set
  loop do
    if EMPTY?(frontier) then return failure
    node ← POP(frontier)  /* chooses the lowest-cost node in frontier */
    if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
    add node.STATE to explored
    for each action in problem.ACTIONS(node.STATE) do
      child ← CHILD-NODE(problem, node, action)
      if child.STATE is not in explored or frontier then
        frontier ← INSERT(child, frontier)
      else if child.STATE is in frontier with higher PATH-COST then
        replace that frontier node with child
```

Analyzing Uniform Cost Search

- **Optimal**
- **Complete**(If Cost of every step exceeds some positive constant Ɛ)
- **Time Complexity-O($b^{1+(c^*/\varepsilon)}$)**
- **Space Complexity-O($b^{1+(c^*/\varepsilon)}$)**
- **UCS examines all the nodes at Goal Depth to see if one has a lower cost.**

COMPARISION

Algorithm		Complete	Optimal	Time	Space
DFS	w/ Path Checking	Y	N	$O(b^m)$	$O(bm)$
BFS		Y	Y*	$O(b^d)$	$O(b^d)$
UCS		Y*	Y	$O(b^{C*/\varepsilon})$	$O(b^{C*/\varepsilon})$

DEPTH - LIMITED SEARCH

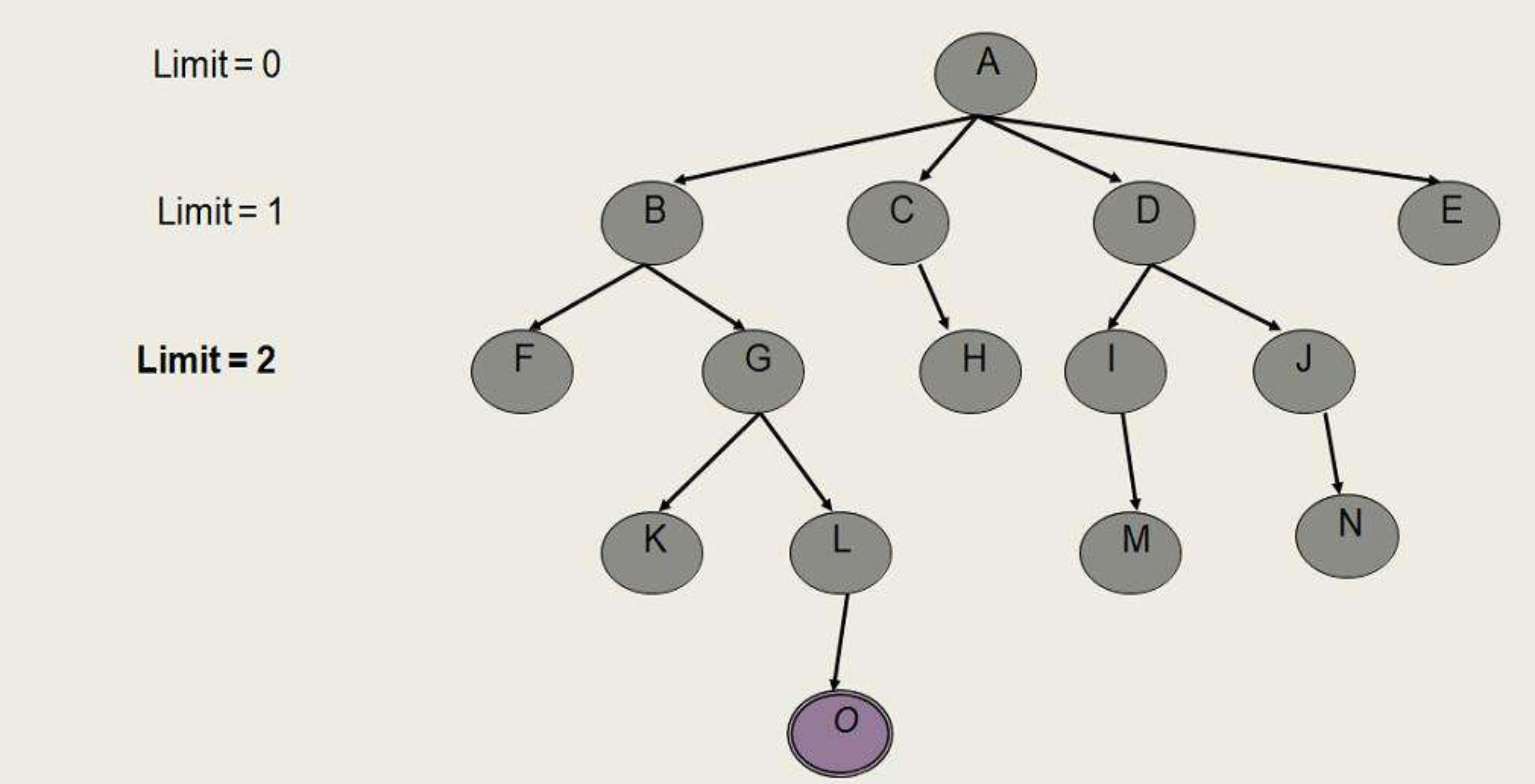
DEPTH-LIMITED-SEARCH

- The embarrassing failure of depth-first search in infinite state spaces can be alleviated by supplying depth-first search with a predetermined depth limit .
- That is, nodes at depth are treated as if they have no successors. This approach is called depth-limited search. The depth limit solves the infinite-path problem.
- Depth-limited search can be implemented as a simple modification to the general tree or graph-search algorithm.
- Notice that depth-limited search can terminate with two kinds of failure:

  The **standard failure** value indicates **no solution.**

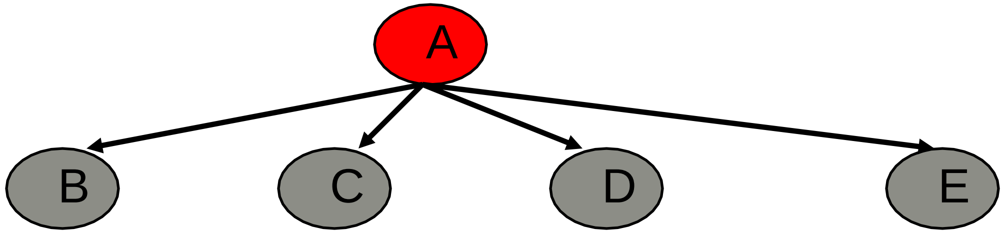
  The **cutoff value indicates no solution within the depth limit.**

DEPTH-LIMITED SEARCH (EXAMPLE-1)



Depth-Limited Search (DLS)

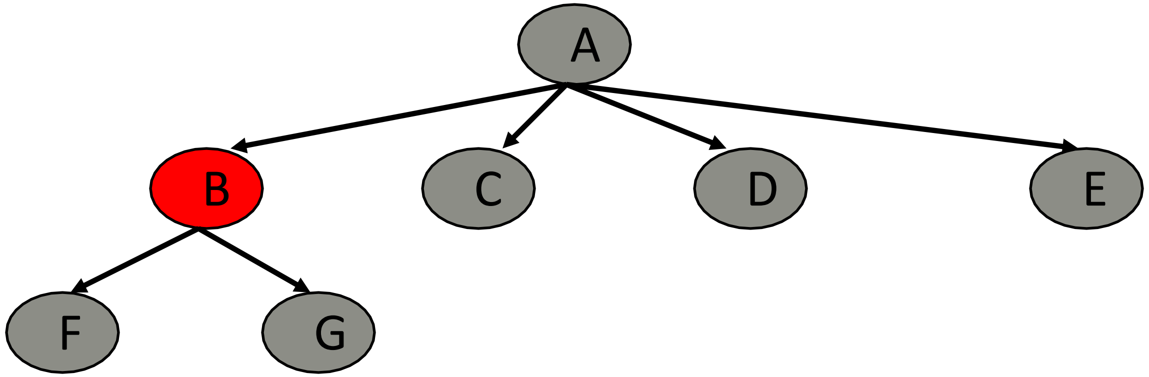
- **A,**



Limit = 2

Depth-Limited Search (DLS)

- **A,B,**

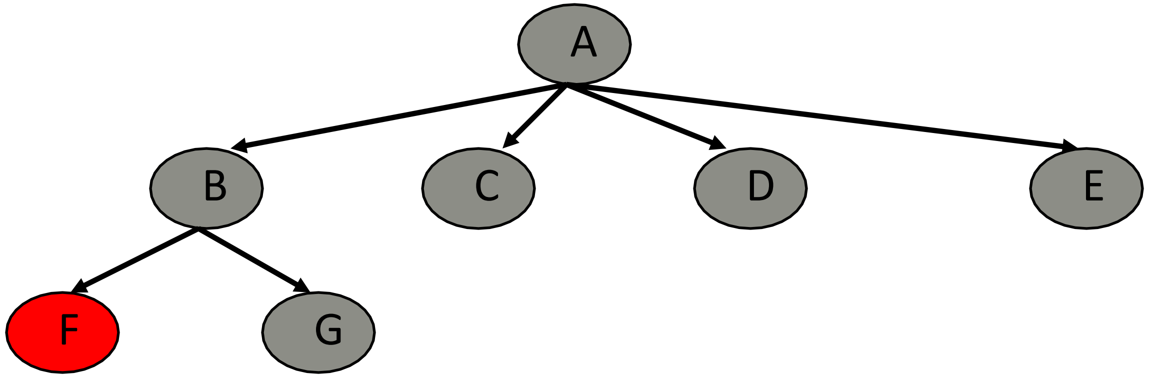


Limit = 2

Depth-Limited Search (DLS)

- **A,B,F,**

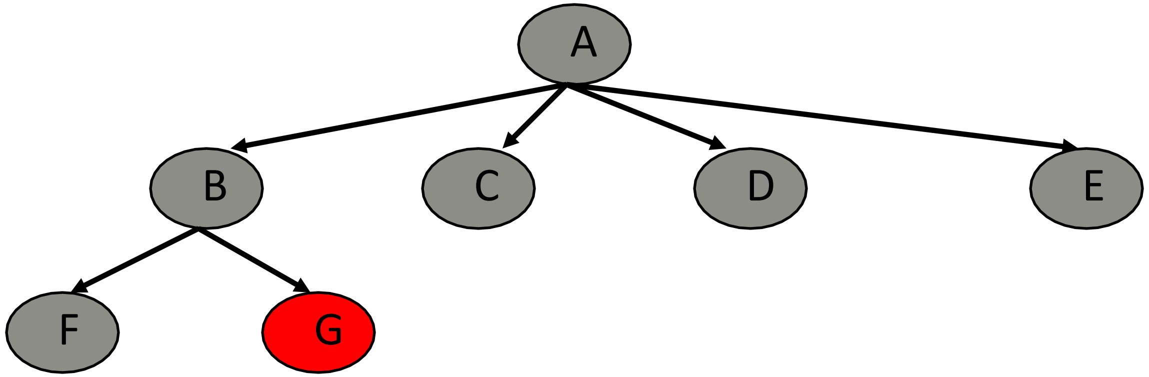
Limit = 2



Depth-Limited Search (DLS)

- A,B,F,
- G,

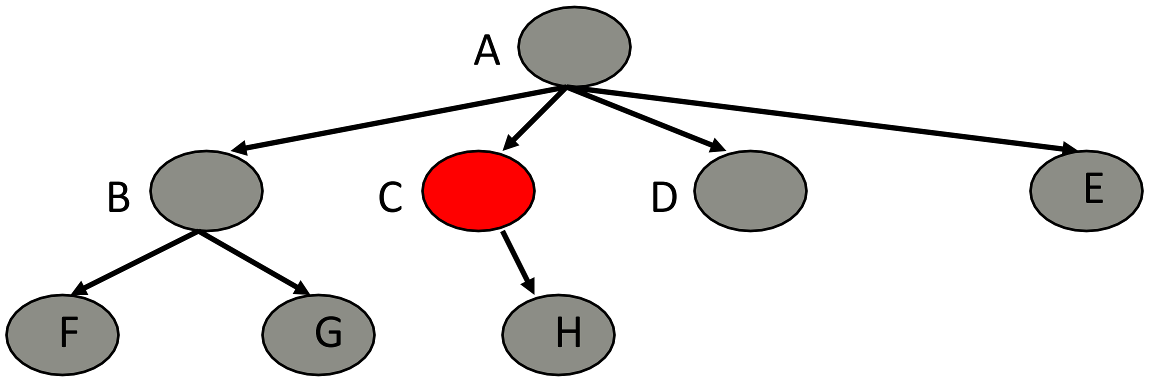
Limit = 2



Depth-Limited Search (DLS)

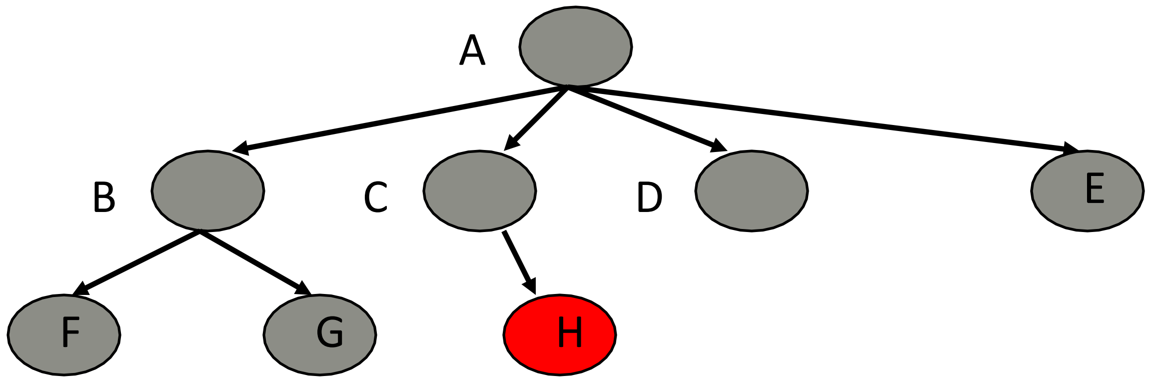
- **A,B,F,**
- **G,**
- **C,**

Limit = 2



Depth-Limited Search (DLS)

- **A,B,F,**
- **G,**
- **C,H,**

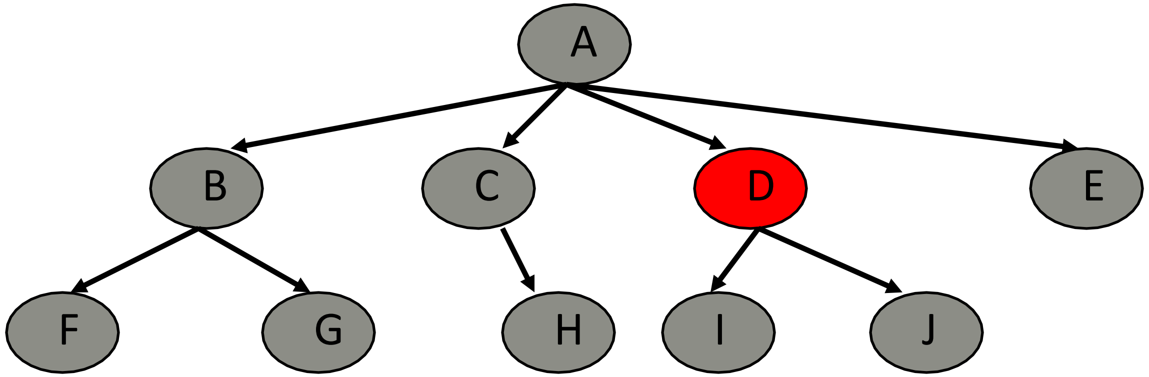


Limit = 2

Depth-Limited Search (DLS)

- A,B,F,
- G,
- C,H,
- D,

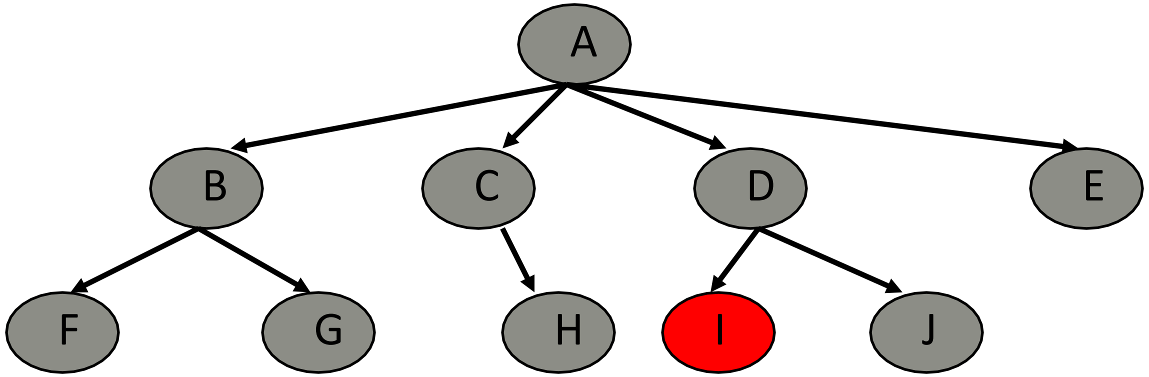
Limit = 2



Depth-Limited Search (DLS)

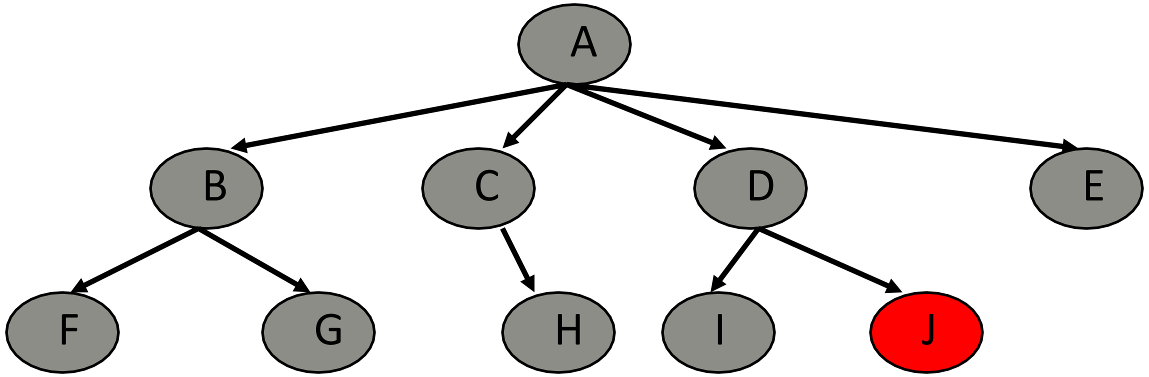
- A,B,F,
- G,
- C,H,
- D,I

Limit = 2



Depth-Limited Search (DLS)

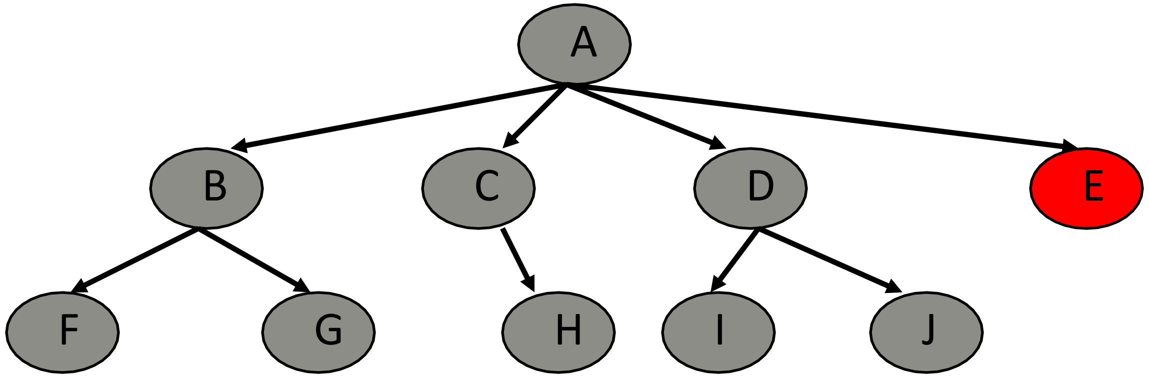
- A,B,F,
- G,
- C,H,
- D,I
- J,



Limit = 2

Depth-Limited Search (DLS)

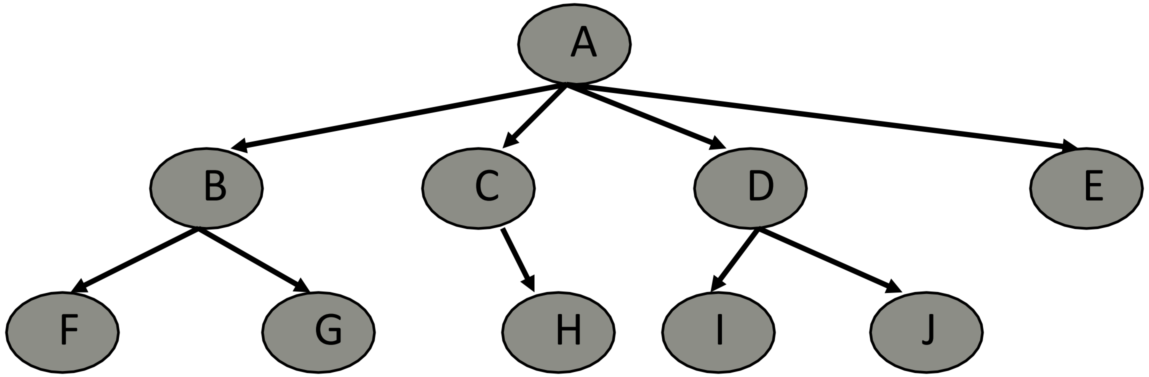
- A,B,F,
- G,
- C,H,
- D,I
- J,
- E



Limit = 2

Depth-Limited Search (DLS)

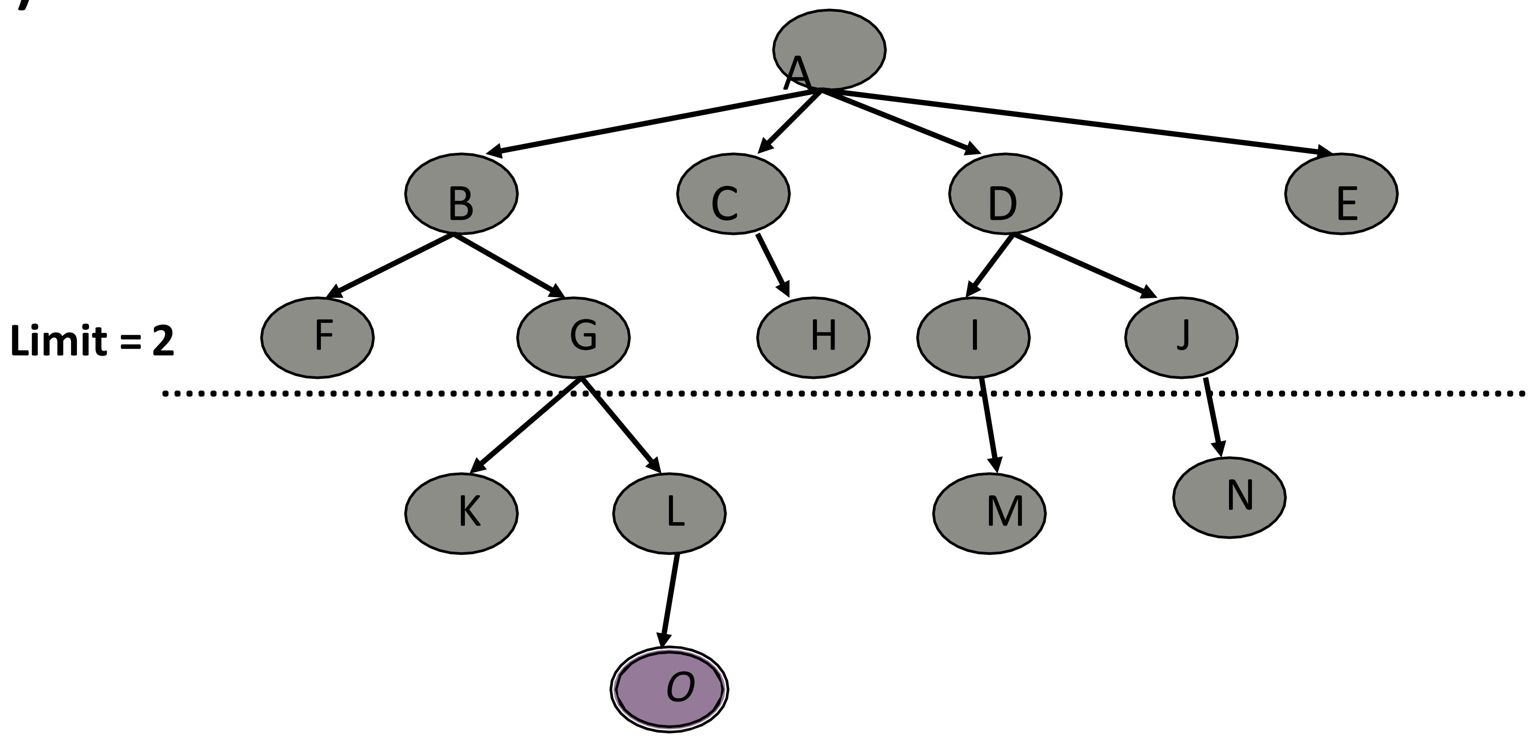
- A,B,F,
- G,
- C,H,
- D,I
- J,
- E, **Failure**



Limit = 2

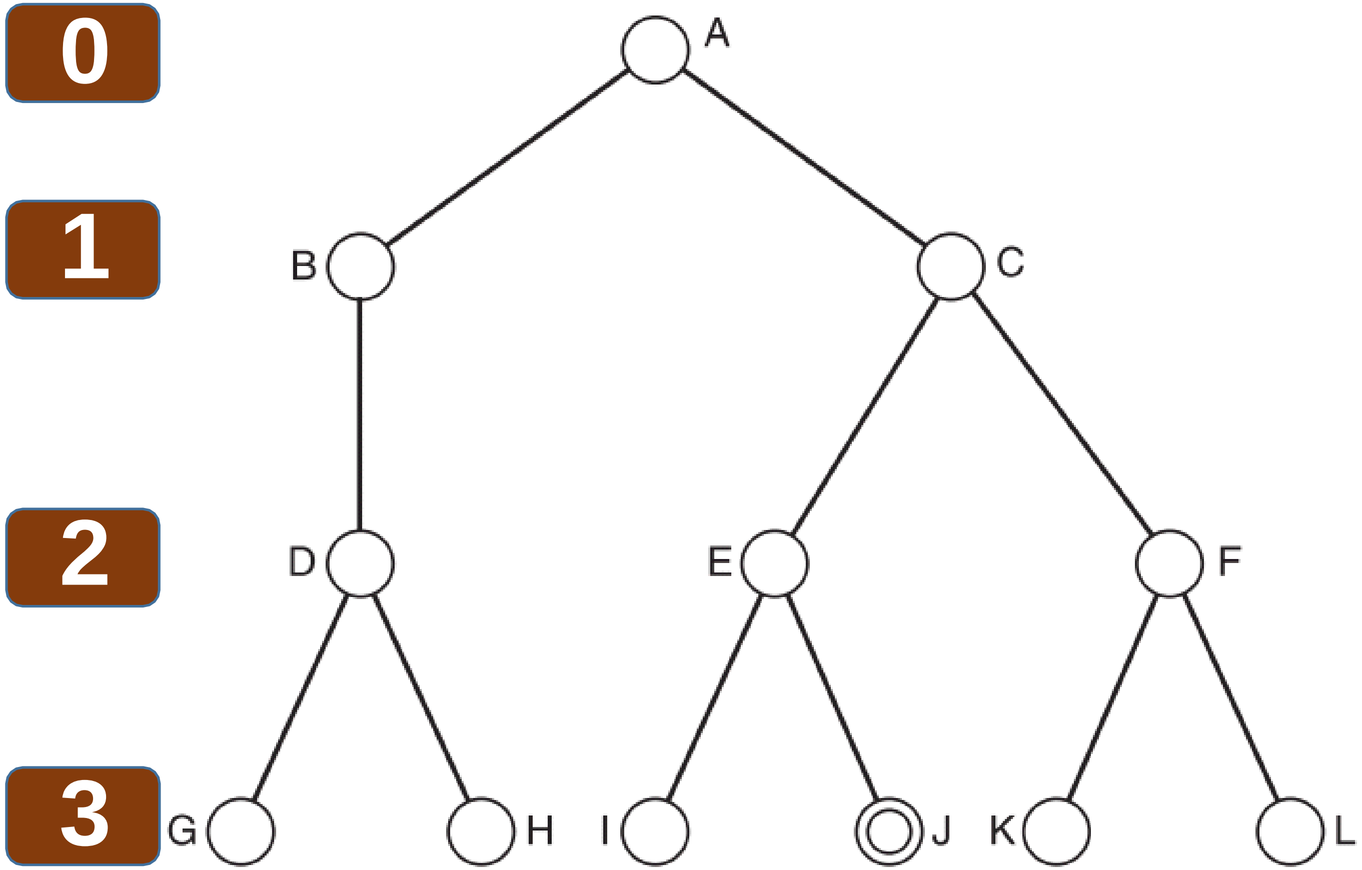
Depth-Limited Search (DLS)

- DLS algorithm returns Failure (no solution)
- The reason is that the goal is beyond the limit (Limit =2): the goal depth is (d=4)



Depth-Limited Search
Depth – 3, Goal – Node J

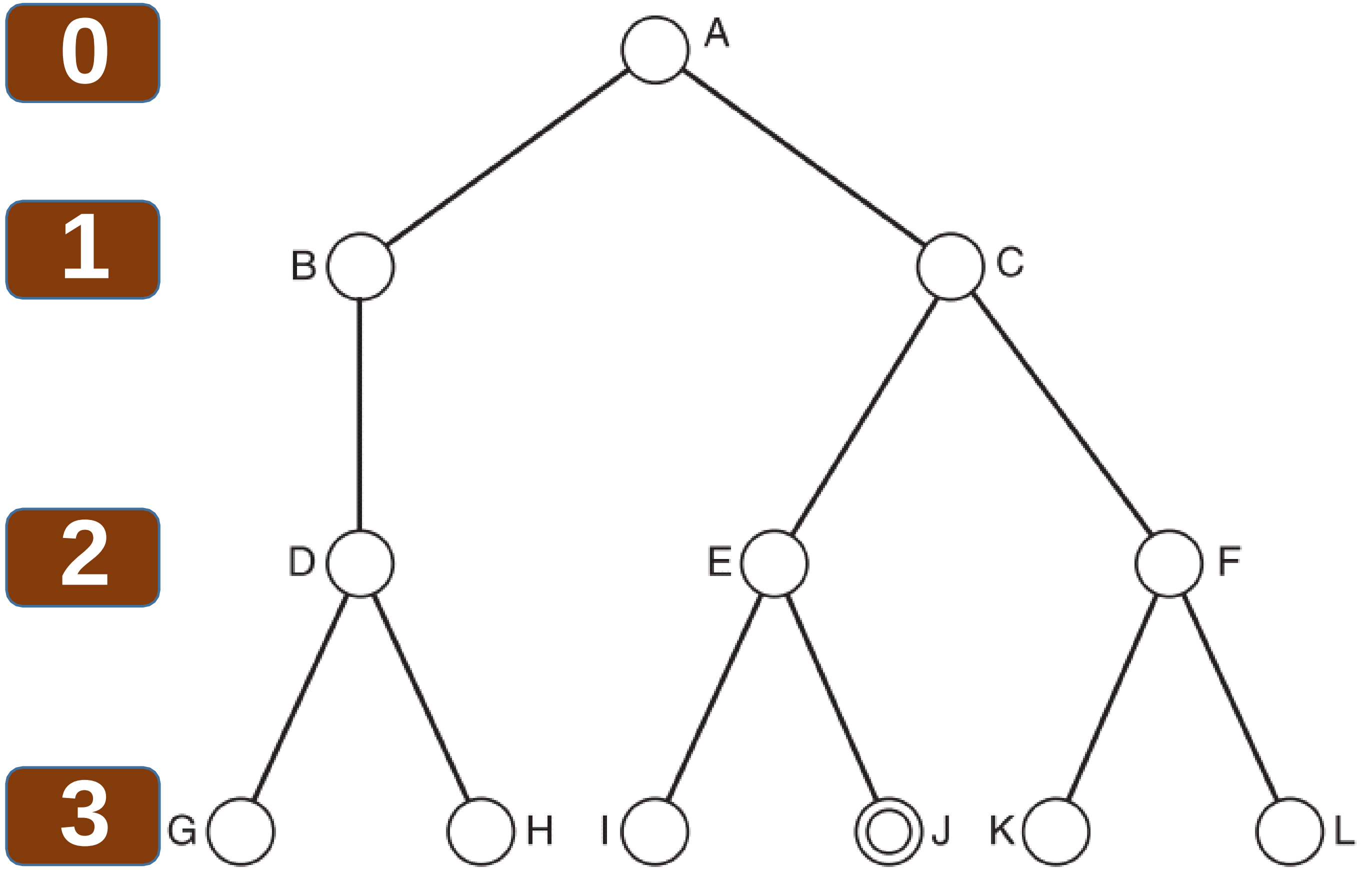
Current



Depth-Limited Search
Depth – 3, Goal – Node J

Current

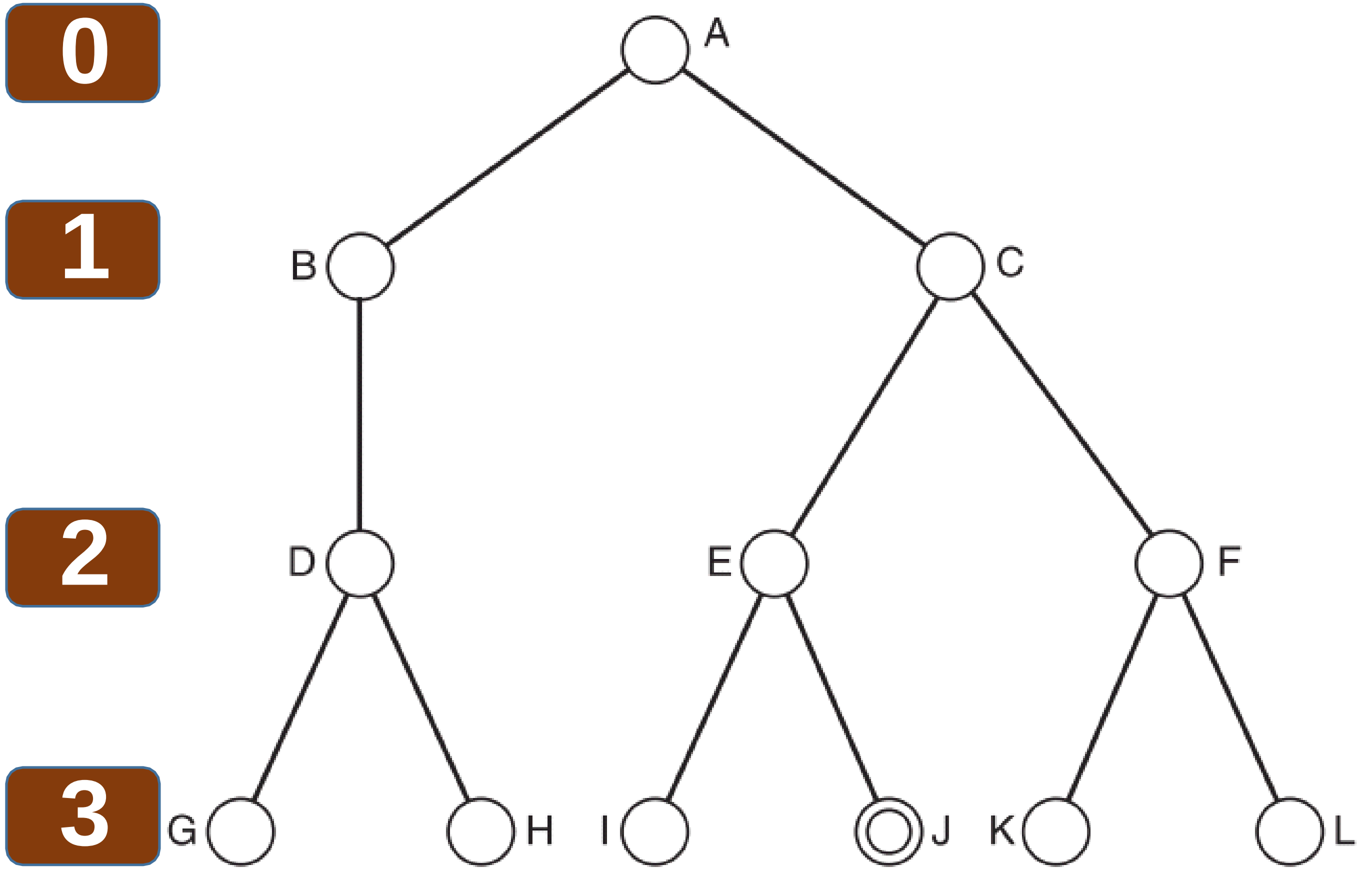
A



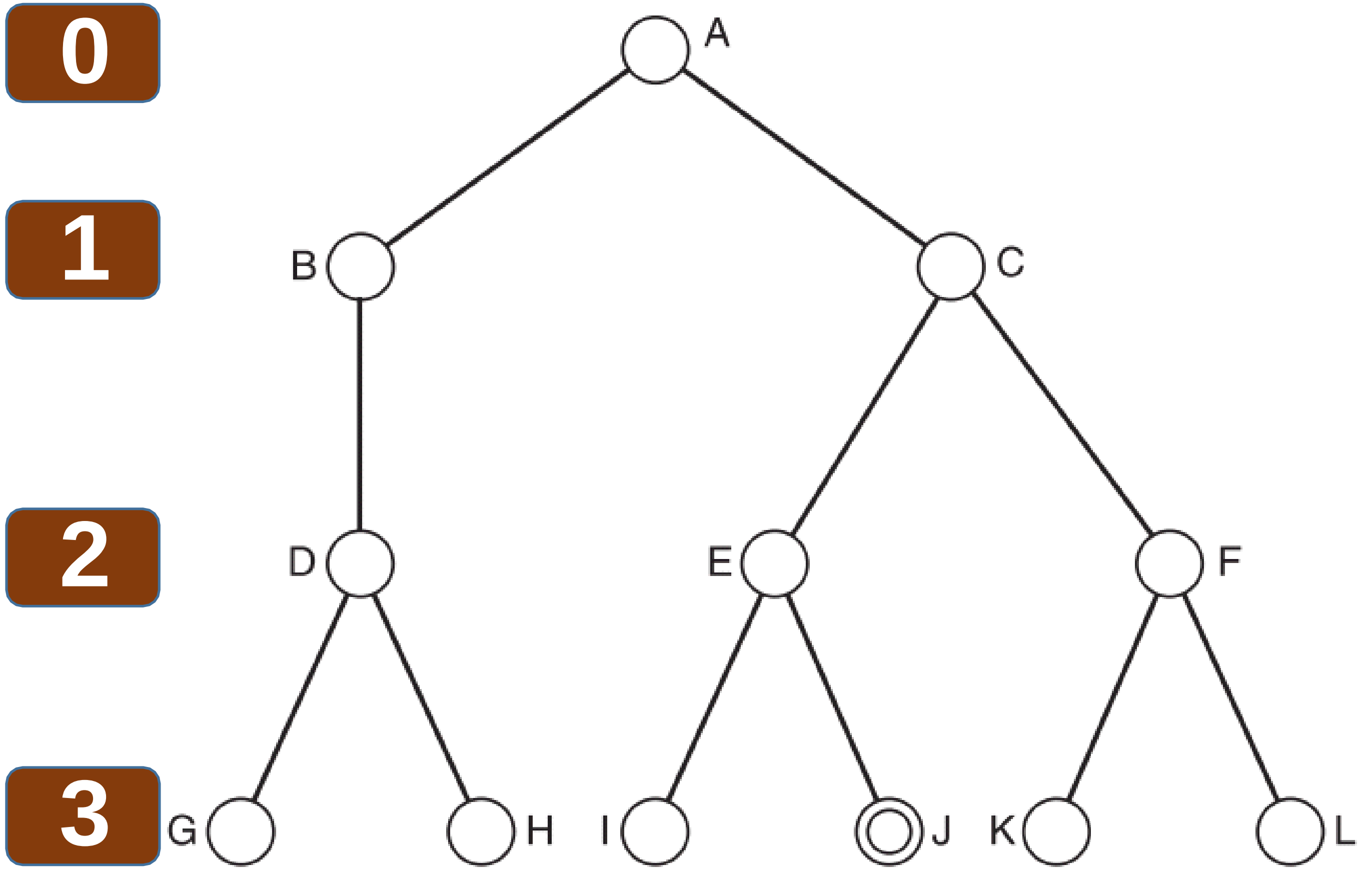
Depth-Limited Search
Depth – 3, Goal – Node J

Current

A



Depth-Limited Search
Depth – 3, Goal – Node J

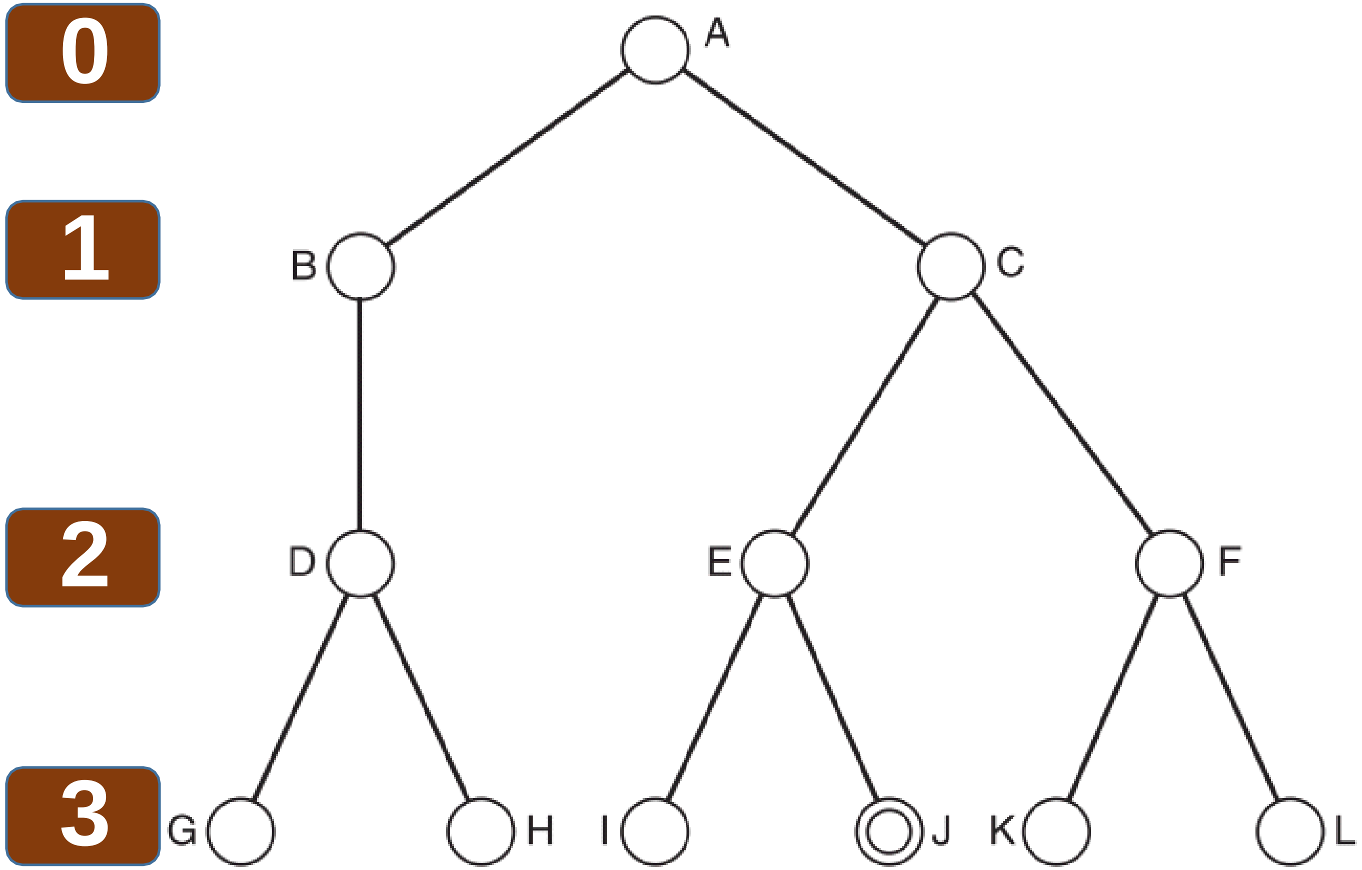


Current

A

B

Depth-Limited Search
Depth – 3, Goal – Node J

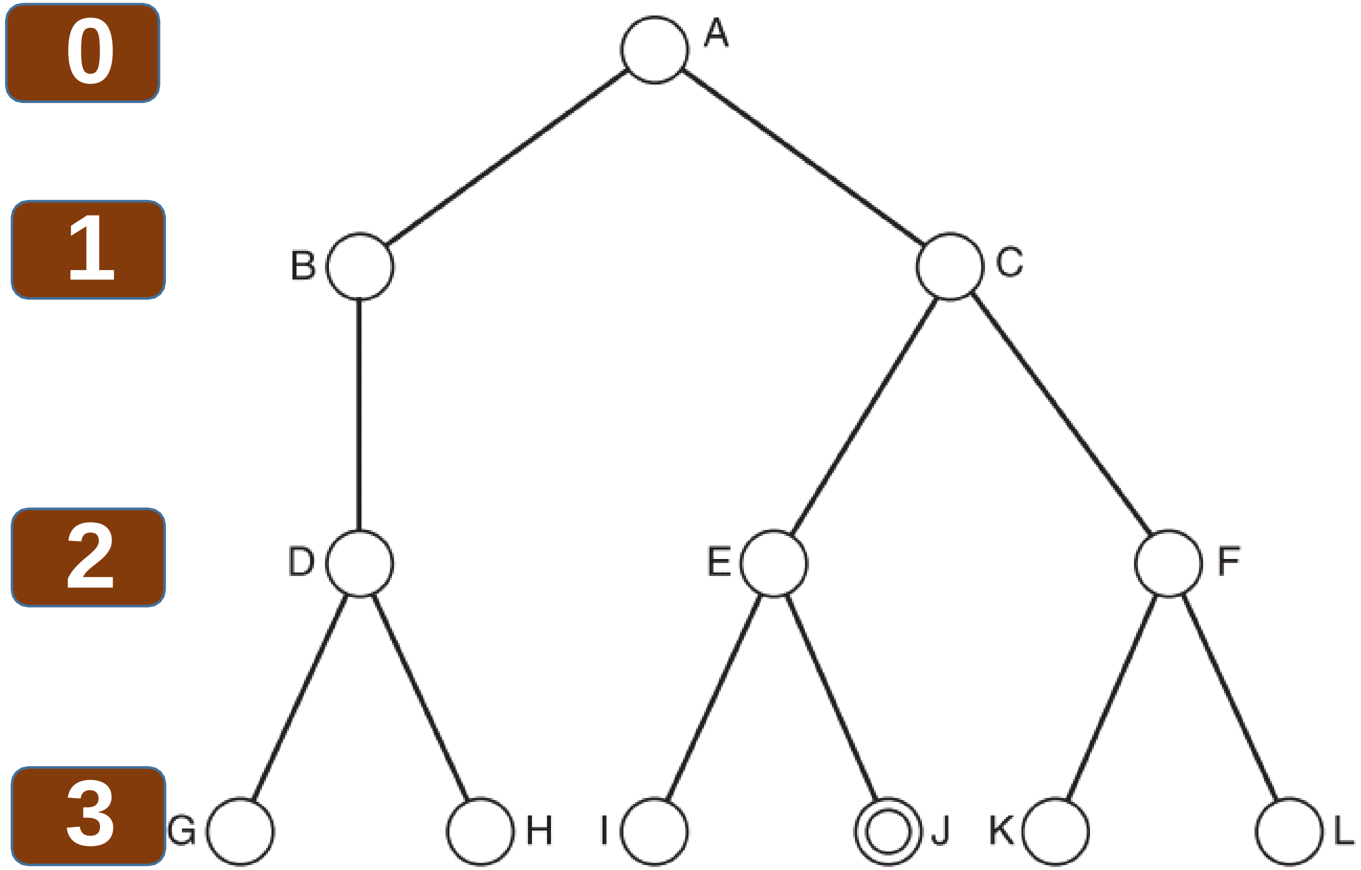


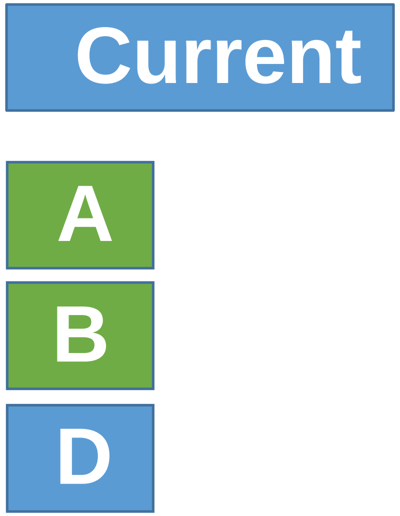
Current

A

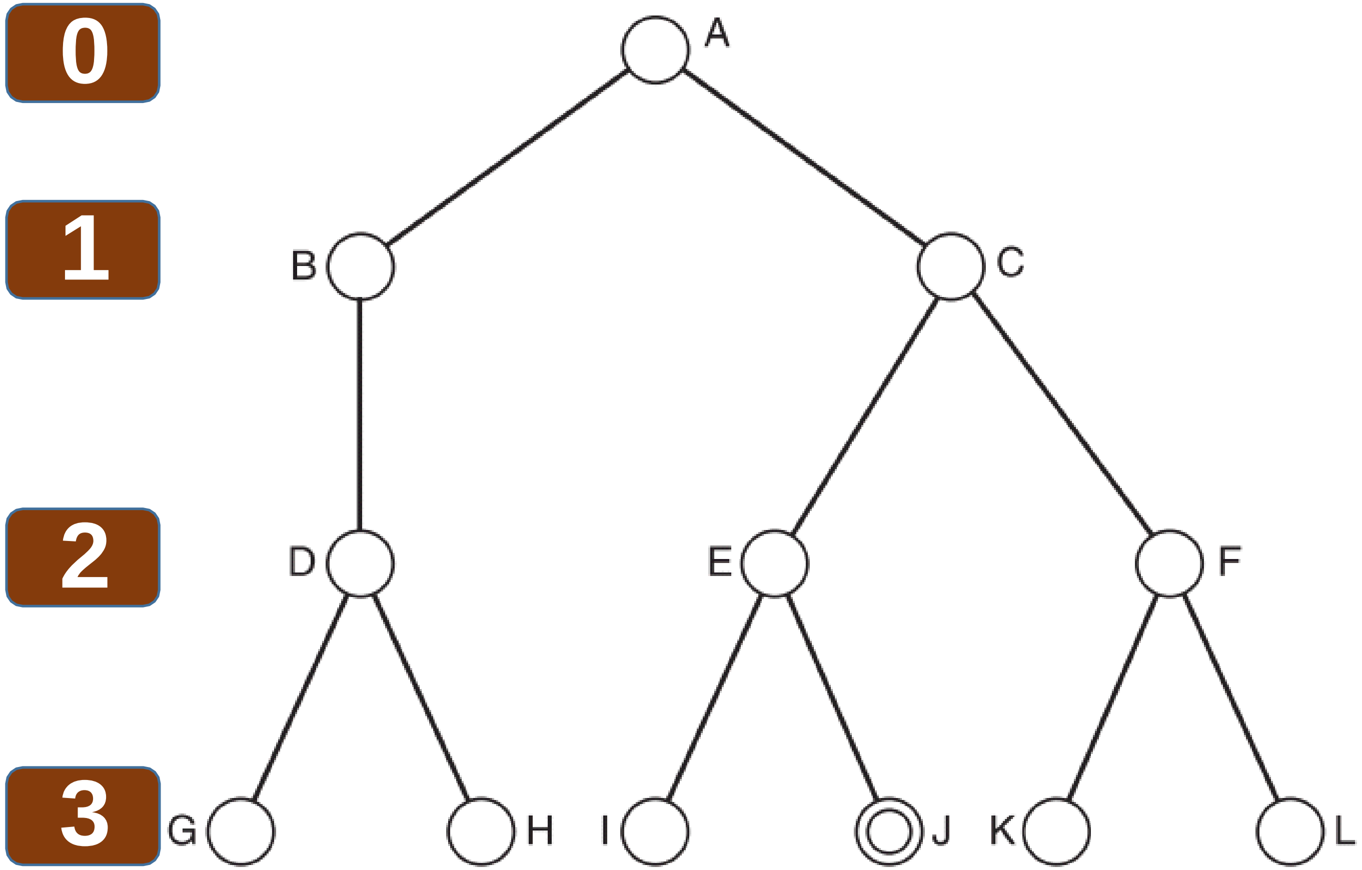
B

Depth-Limited Search
Depth – 3, Goal – Node J





Depth-Limited Search
Depth – 3, Goal – Node J



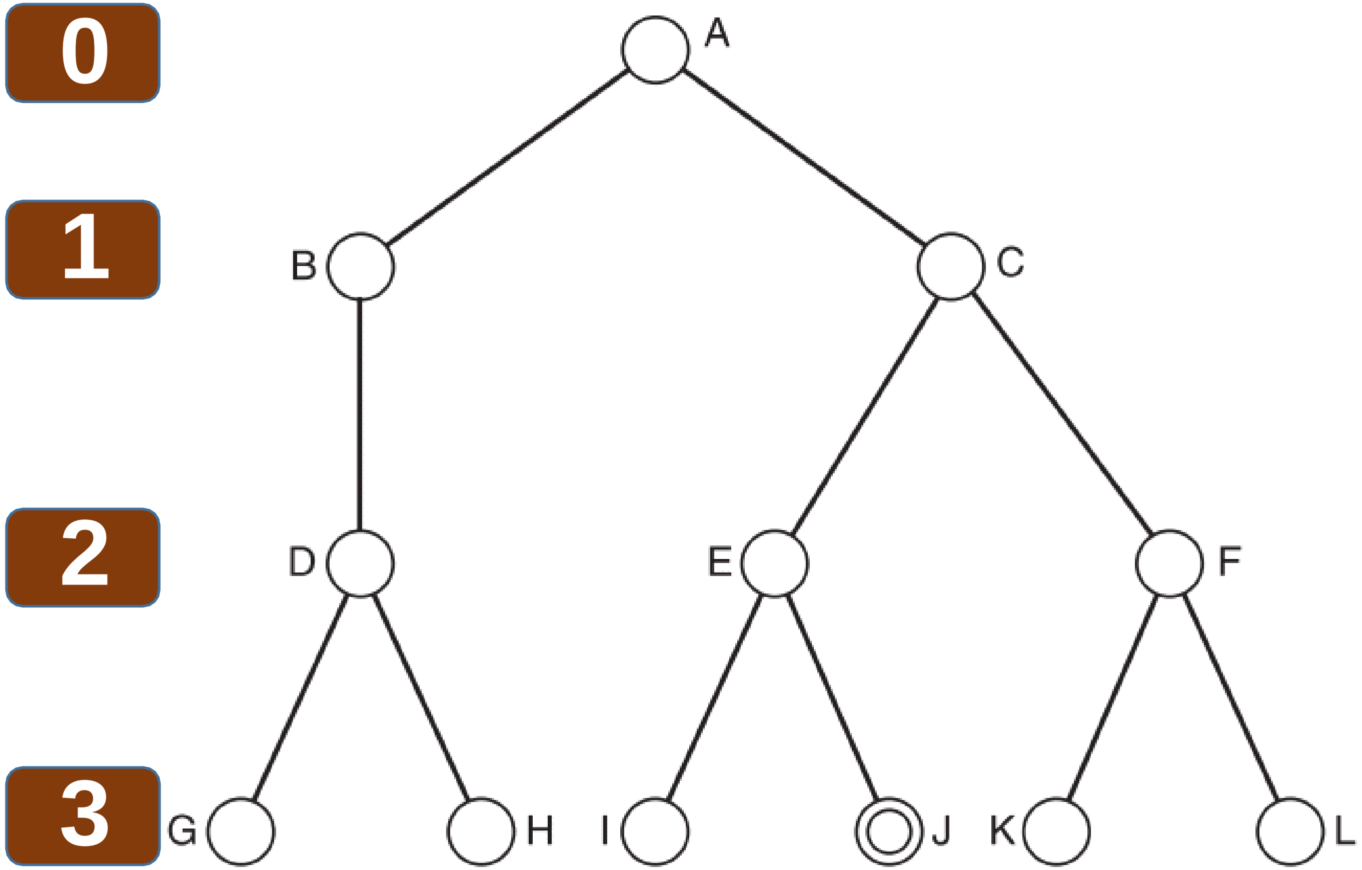
Current

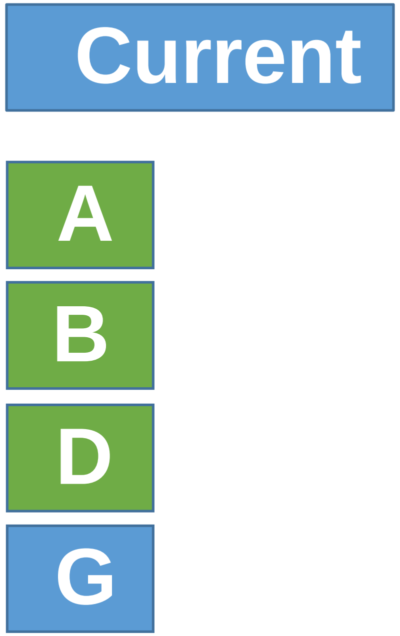
A

B

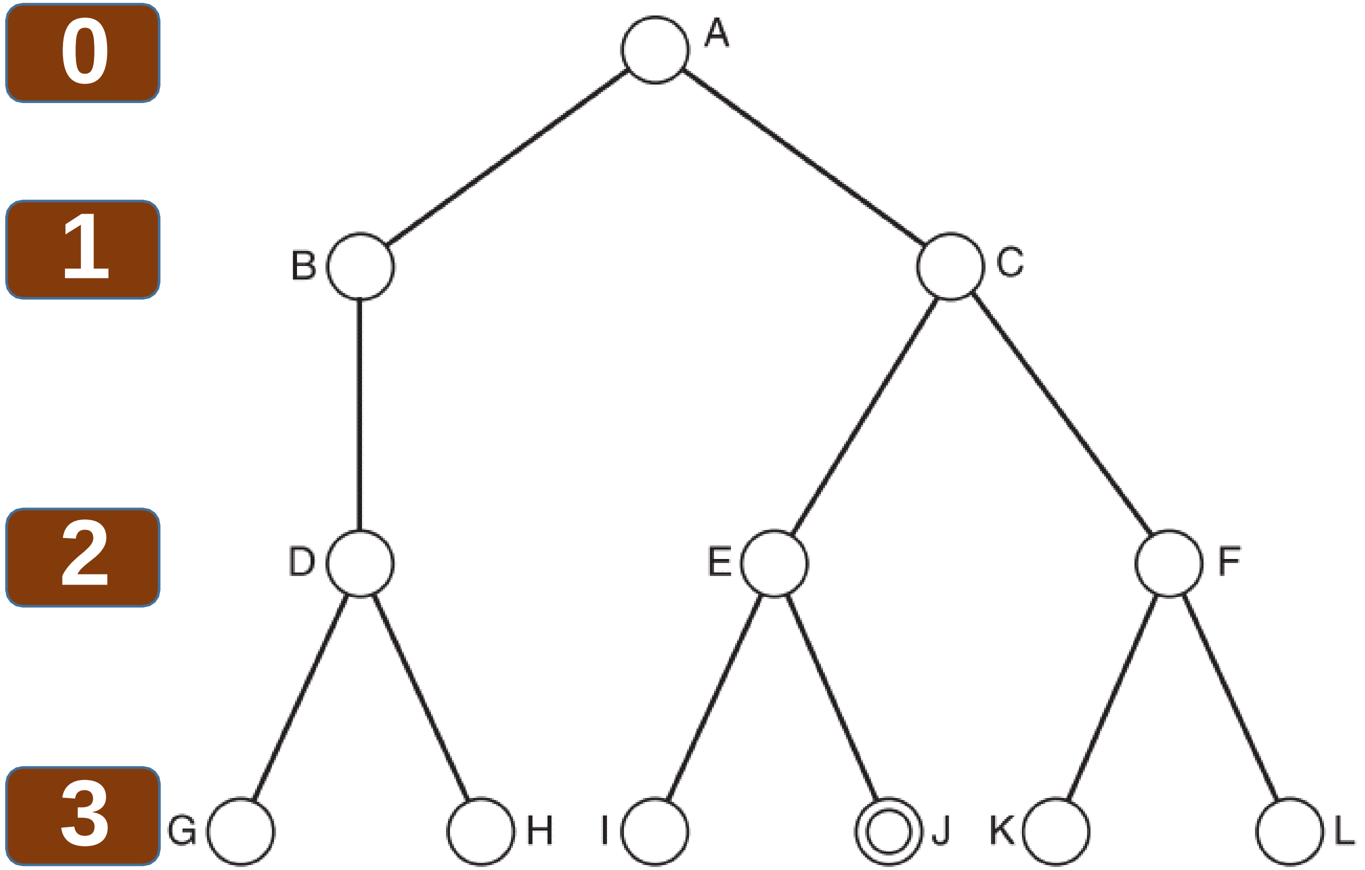
D

Depth-Limited Search
Depth – 3, Goal – Node J





Depth-Limited Search
Depth – 3, Goal – Node J



Current

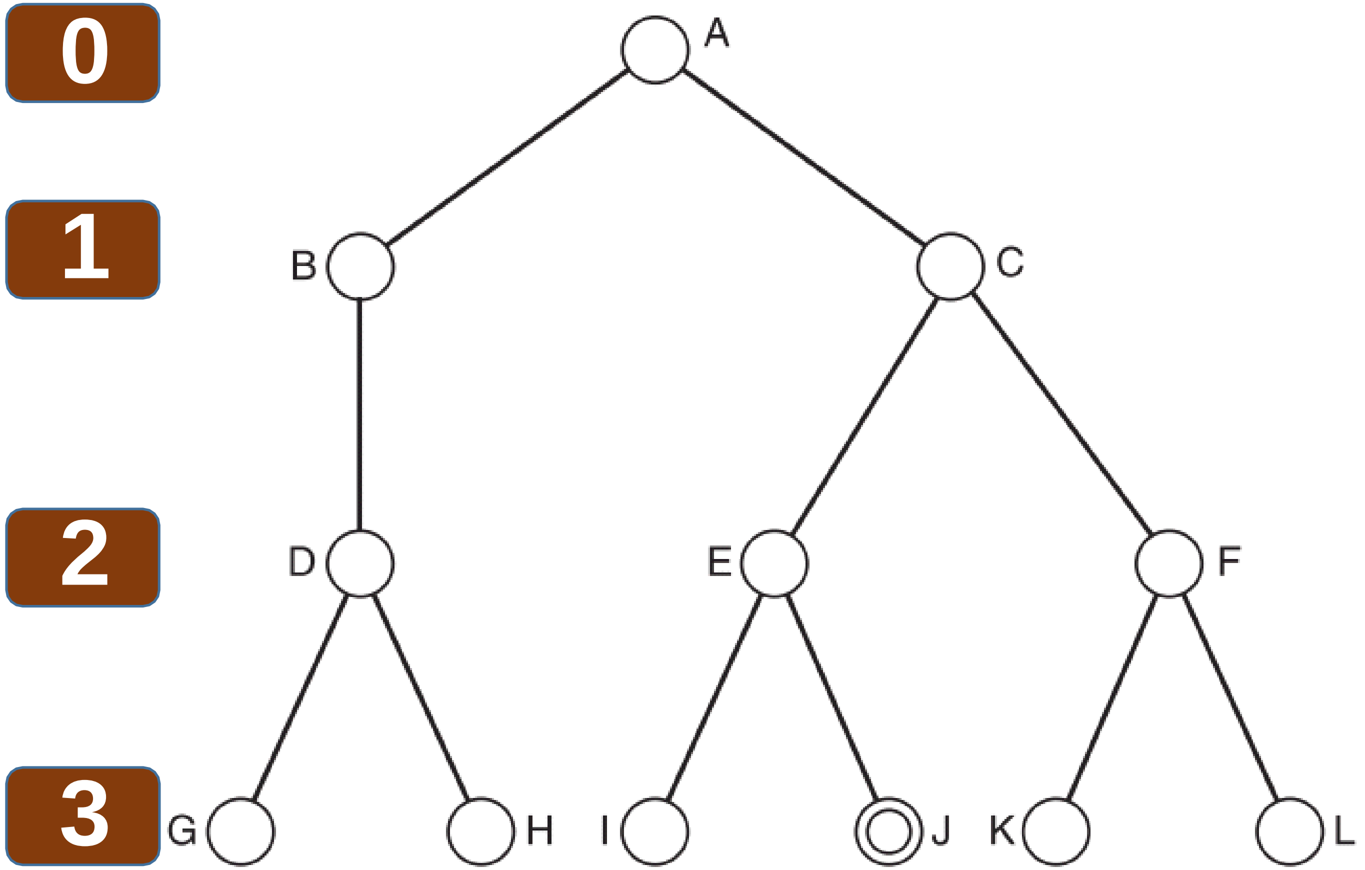
A

B

D

G

Depth-Limited Search
Depth – 3, Goal – Node J



Current

A

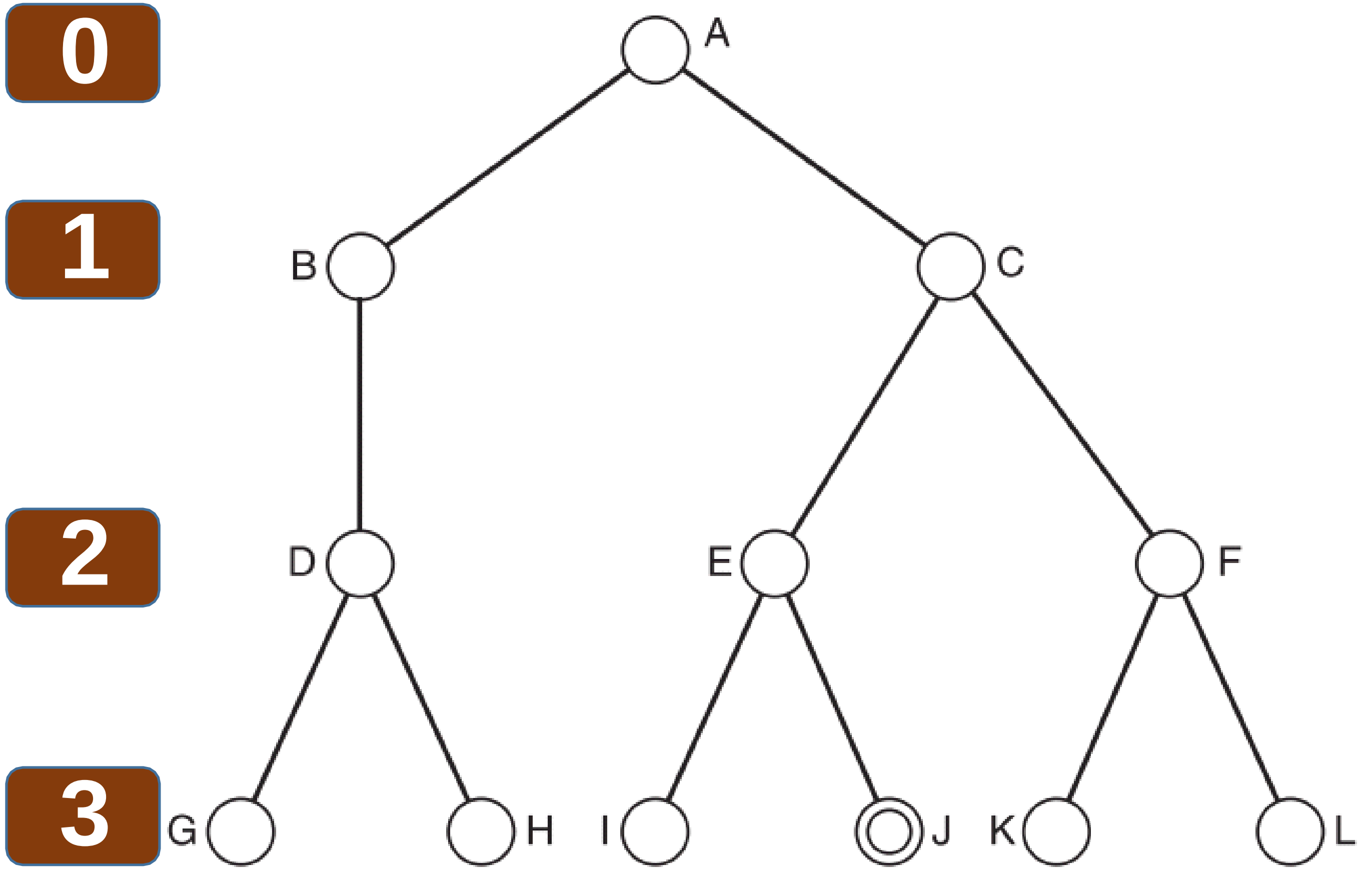
B

D

G

D

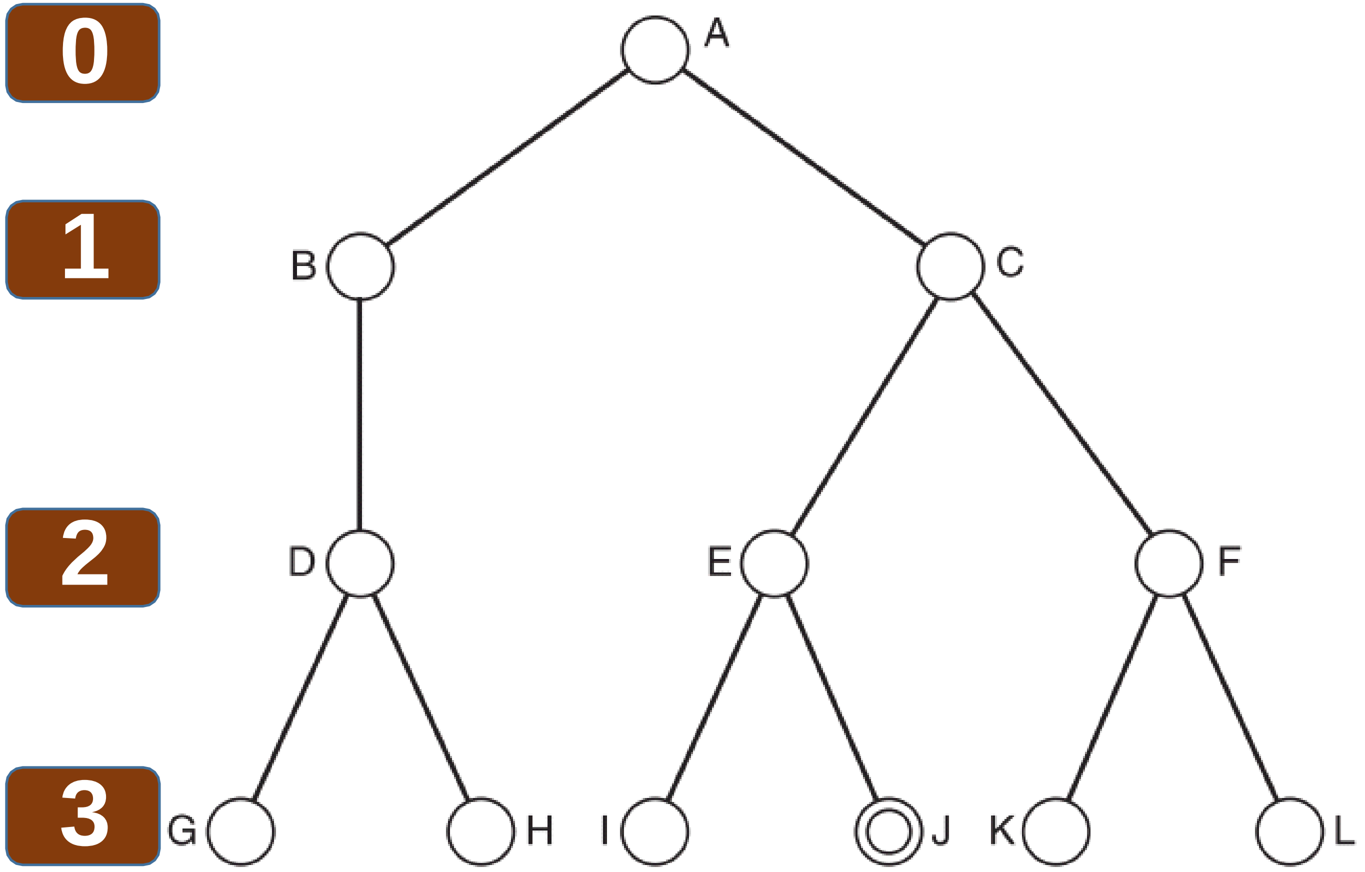
Depth-Limited Search
Depth – 3, Goal – Node J



Current

A
B
D
G
D
H

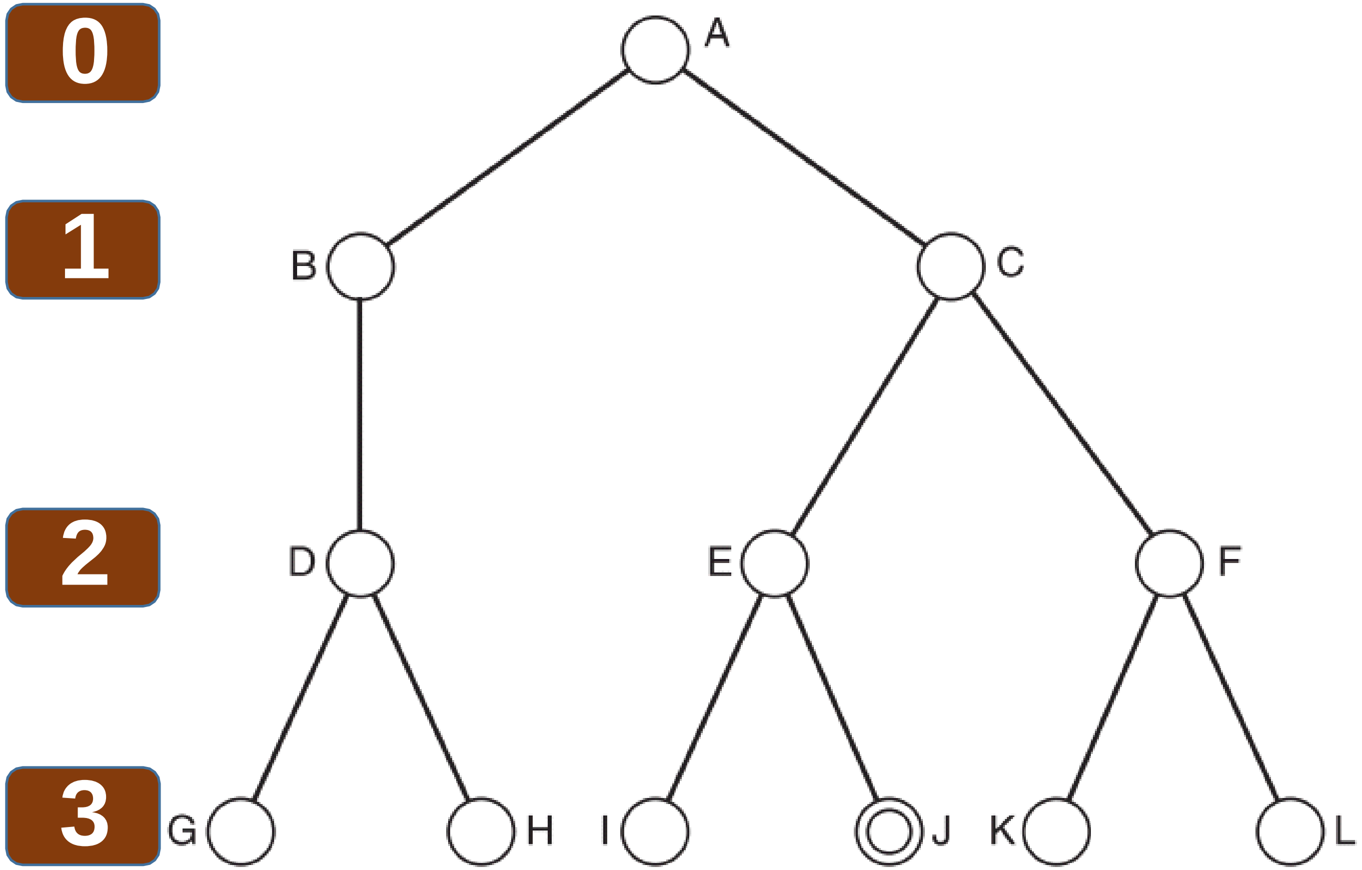
Depth-Limited Search
Depth – 3, Goal – Node J

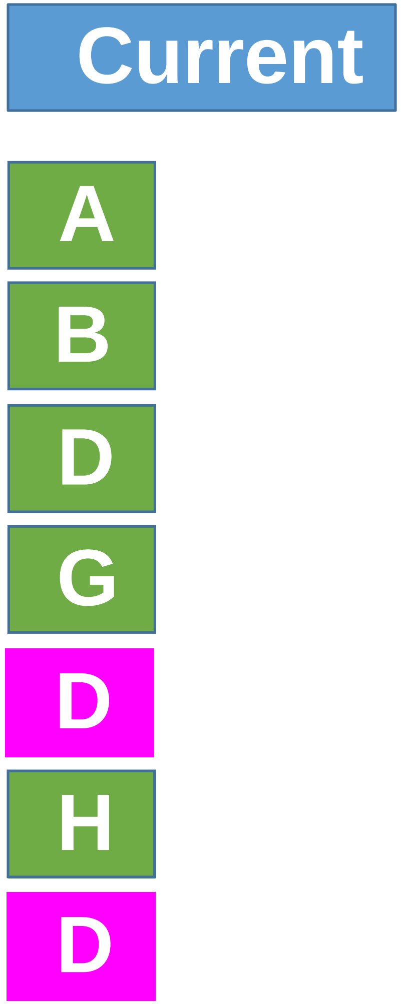


Current

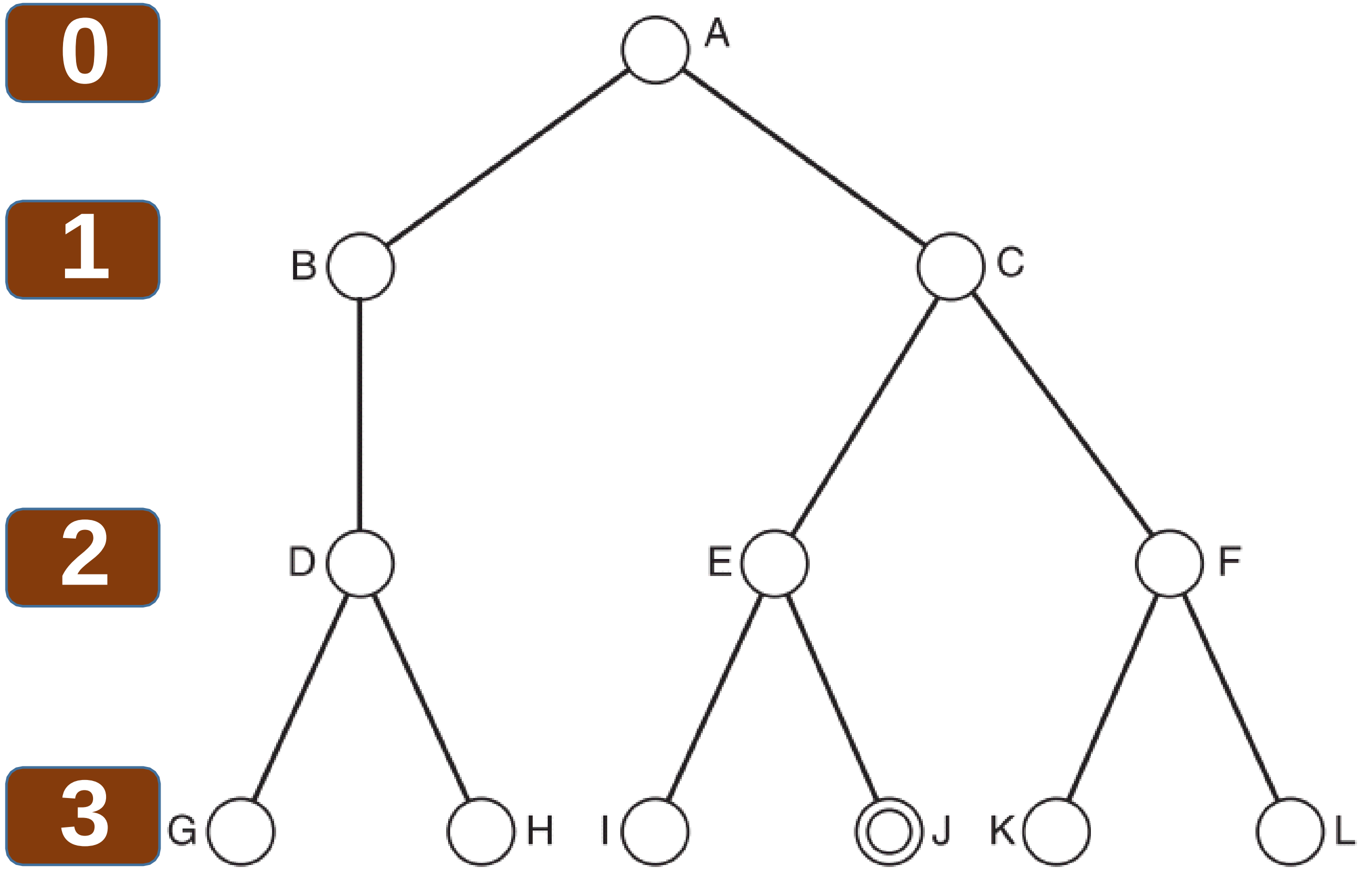
A
B
D
G
D
H

Depth-Limited Search
Depth – 3, Goal – Node J



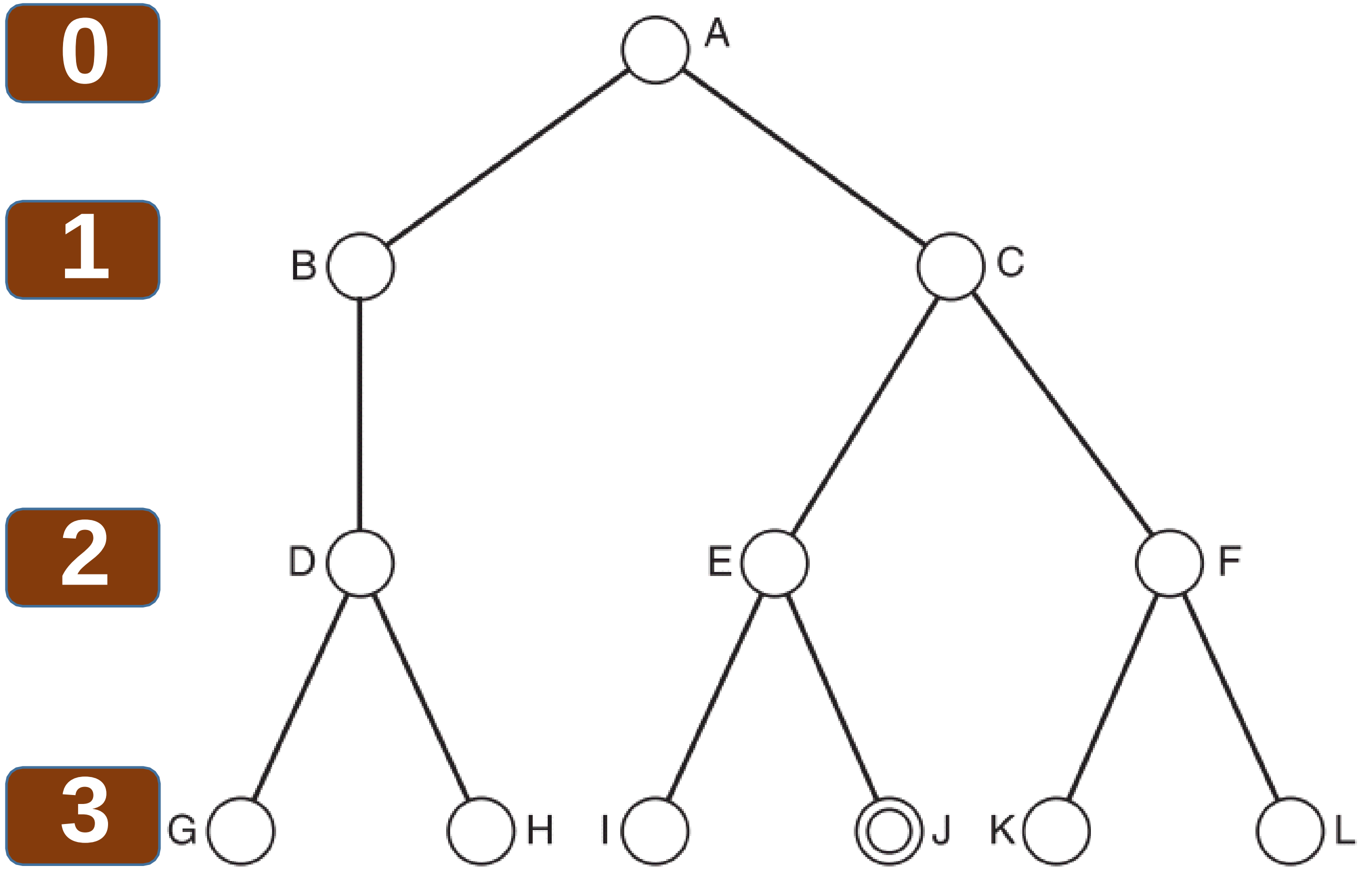


Depth-Limited Search
Depth – 3, Goal – Node J



Current
A
B
D
G
D
H
D
B

Depth-Limited Search
Depth – 3, Goal – Node J



Current

A

B

D

G

D

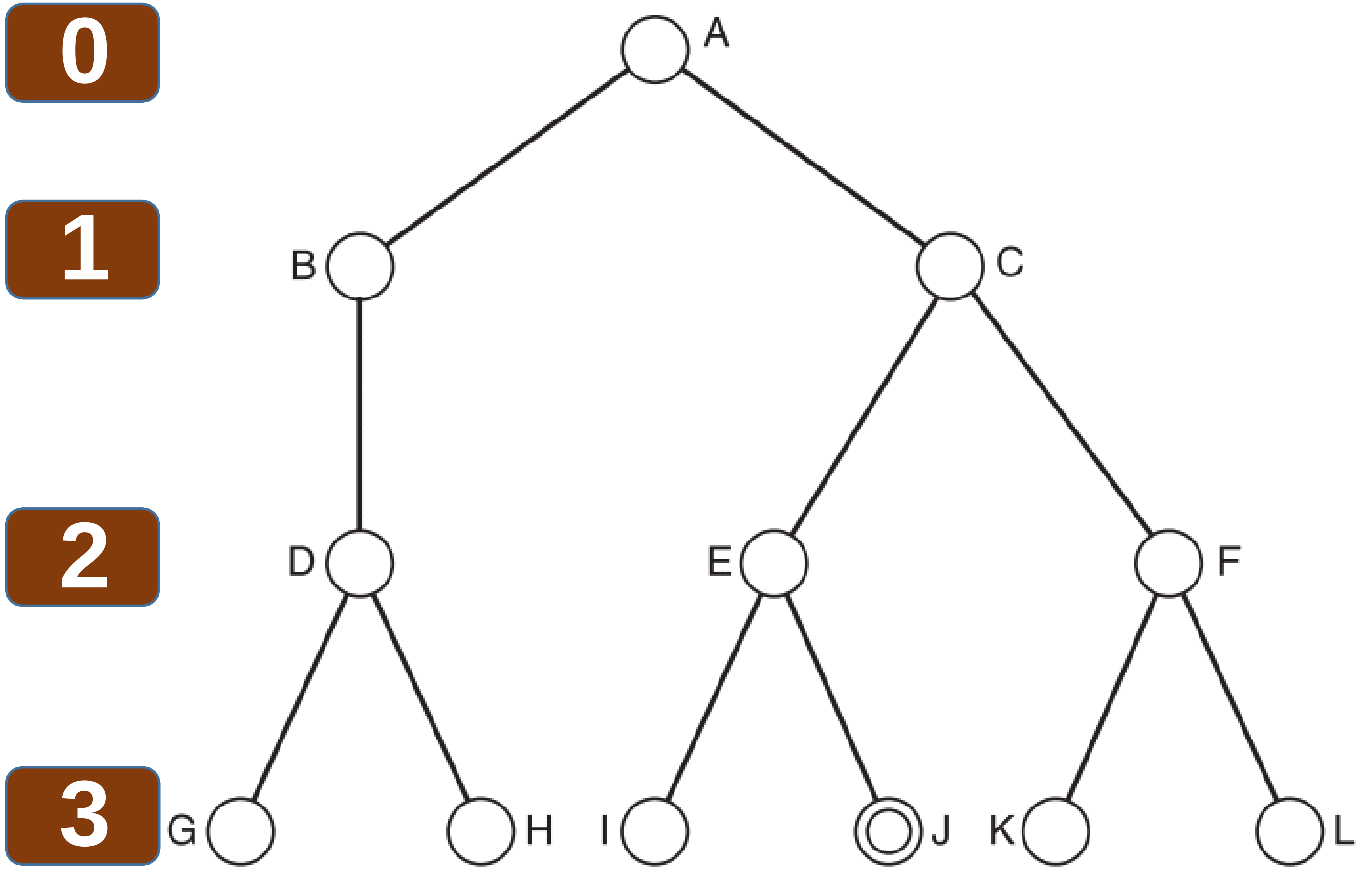
H

D

B

A

Depth-Limited Search
Depth – 3, Goal – Node J



Current

A C

B

D

G

D

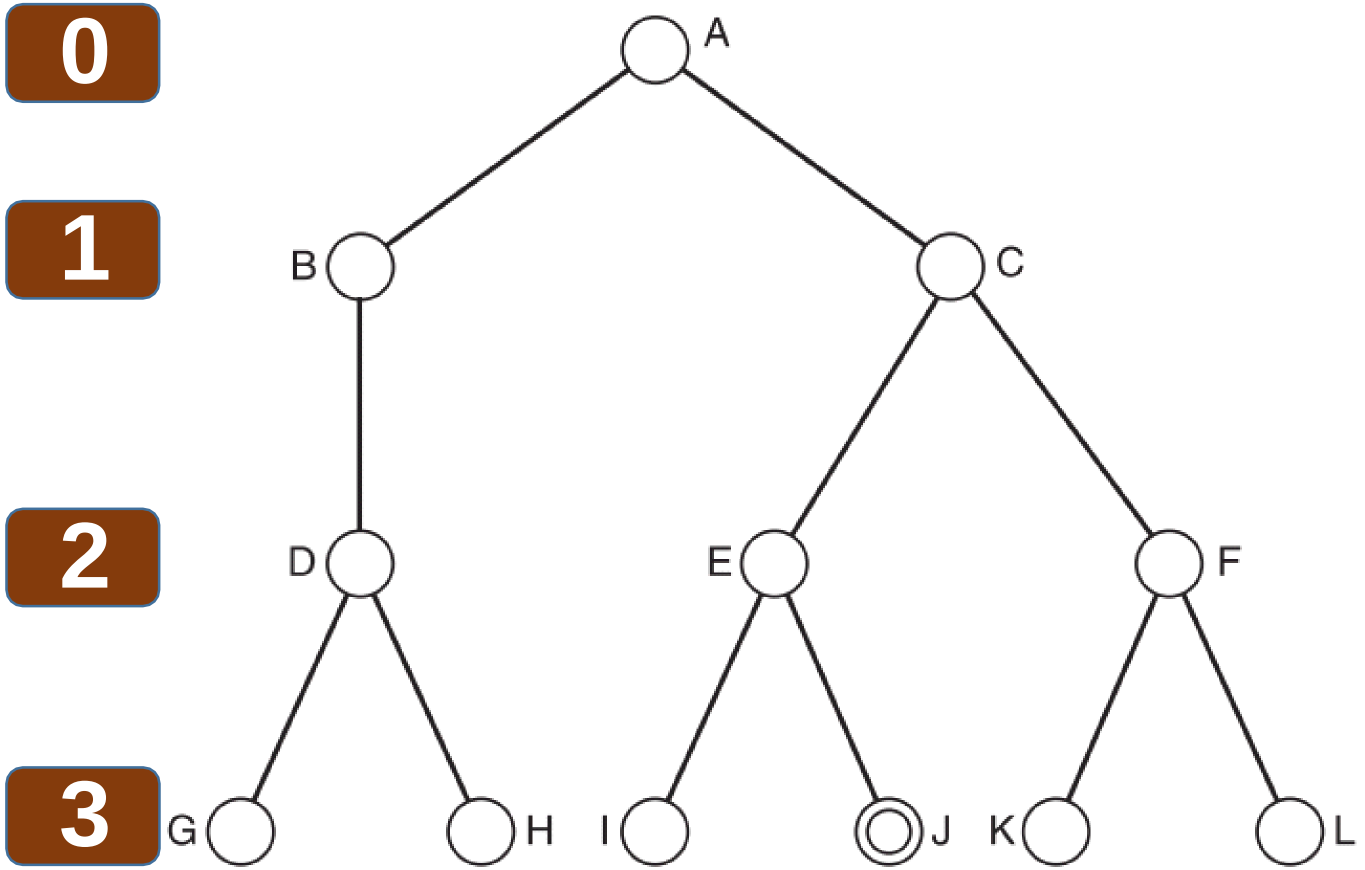
H

D

B

A

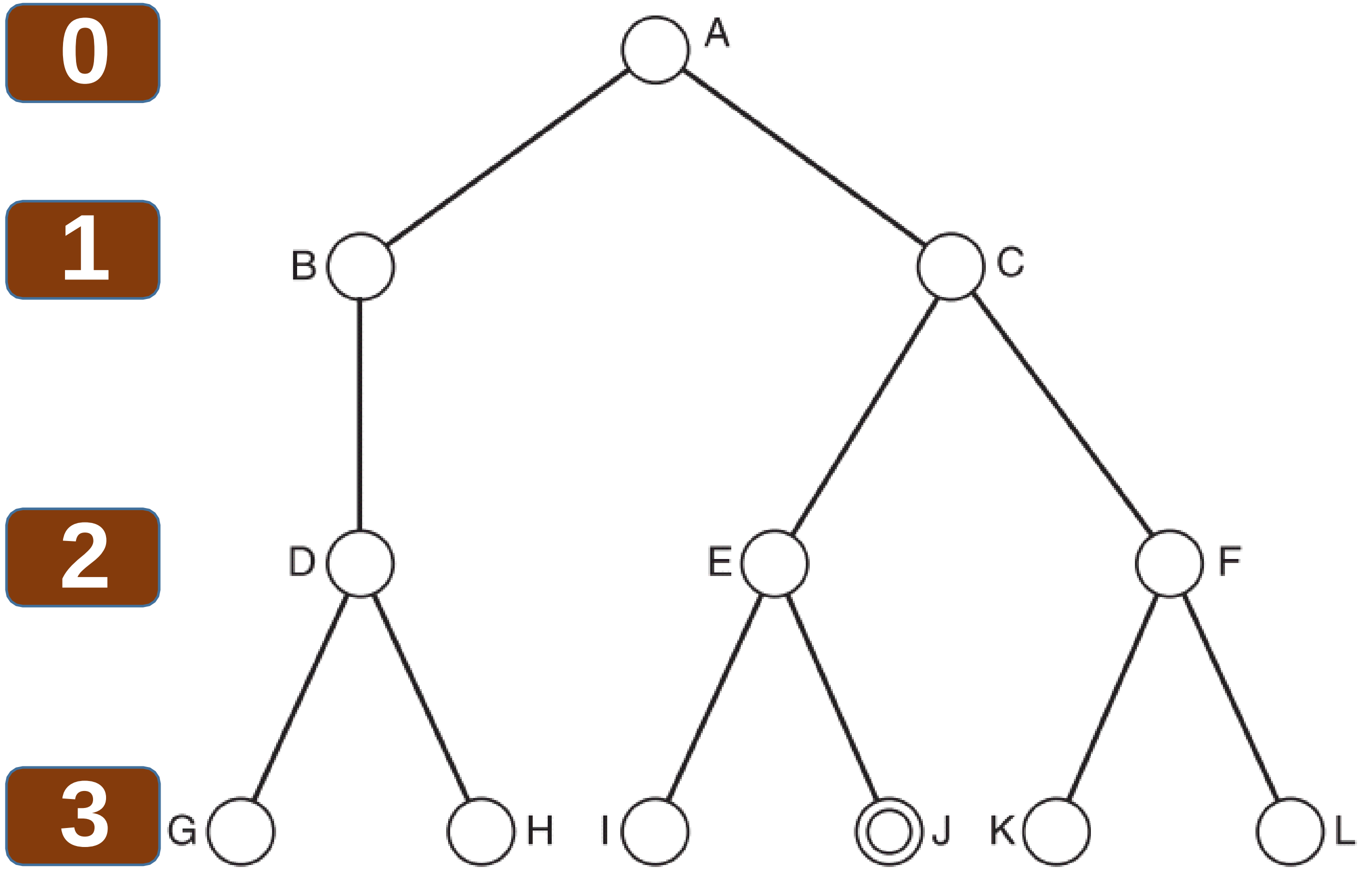
Depth-Limited Search
Depth – 3, Goal – Node J

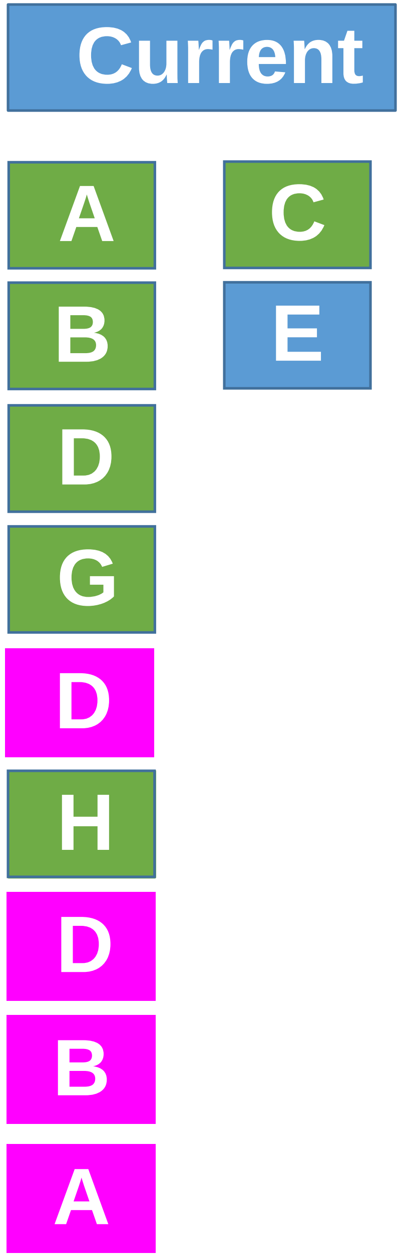


Current

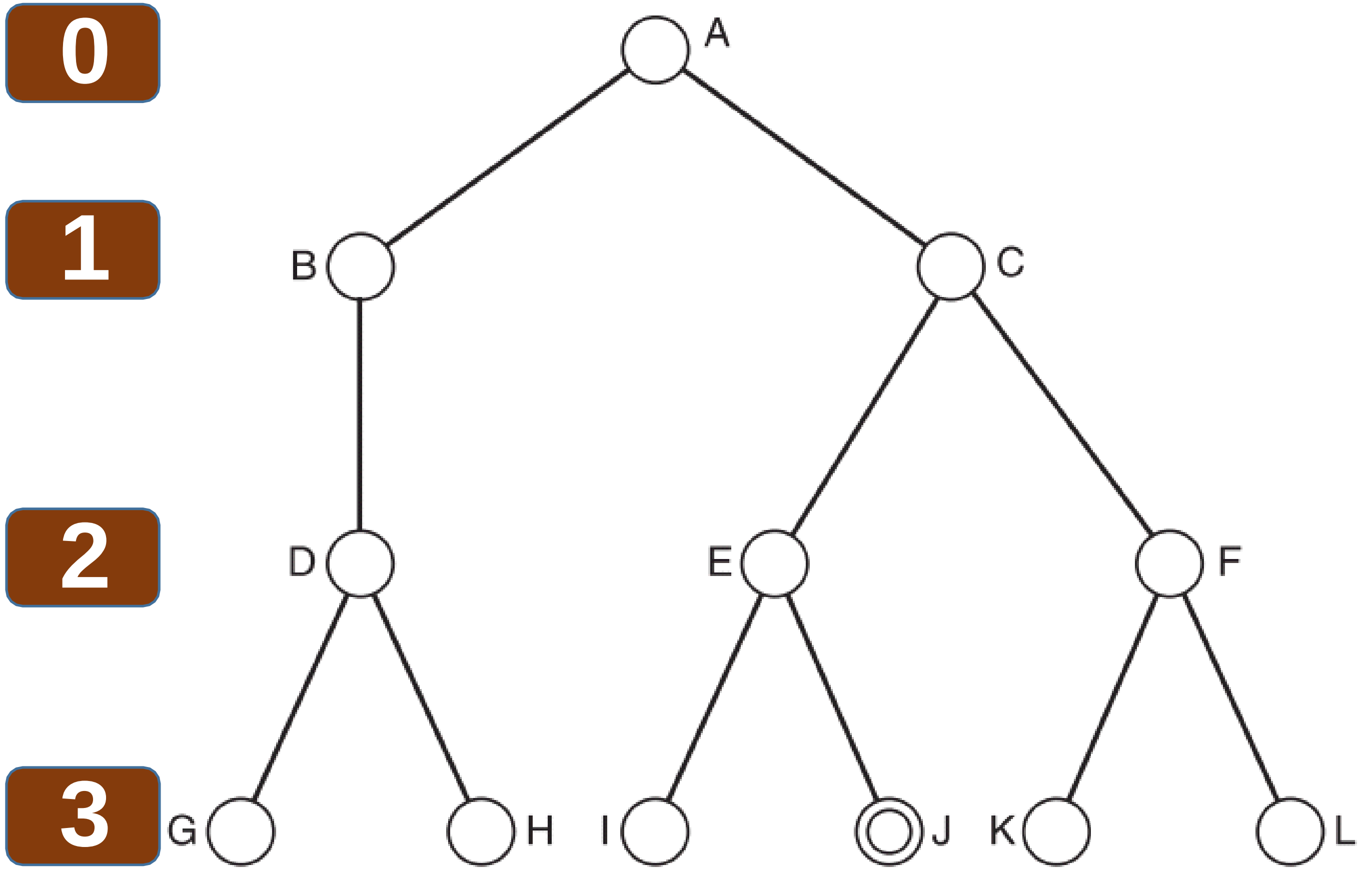
A C
B
D
G
D
H
D
B
A

Depth-Limited Search
Depth – 3, Goal – Node J





Depth-Limited Search
Depth – 3, Goal – Node J

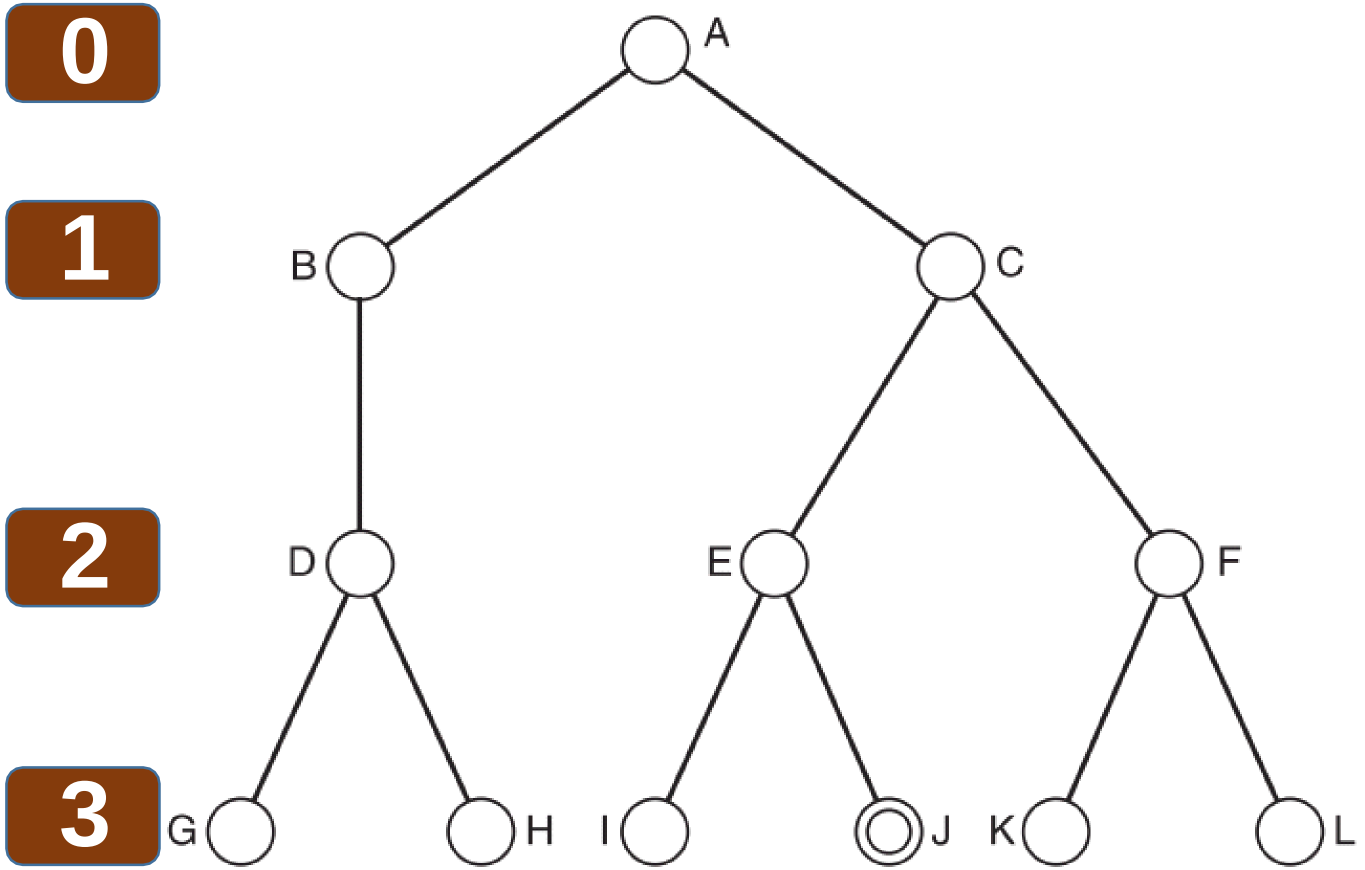


Current

A
B
D
G
D
H
D
B
A

C
E

Depth-Limited Search
Depth – 3, Goal – Node J

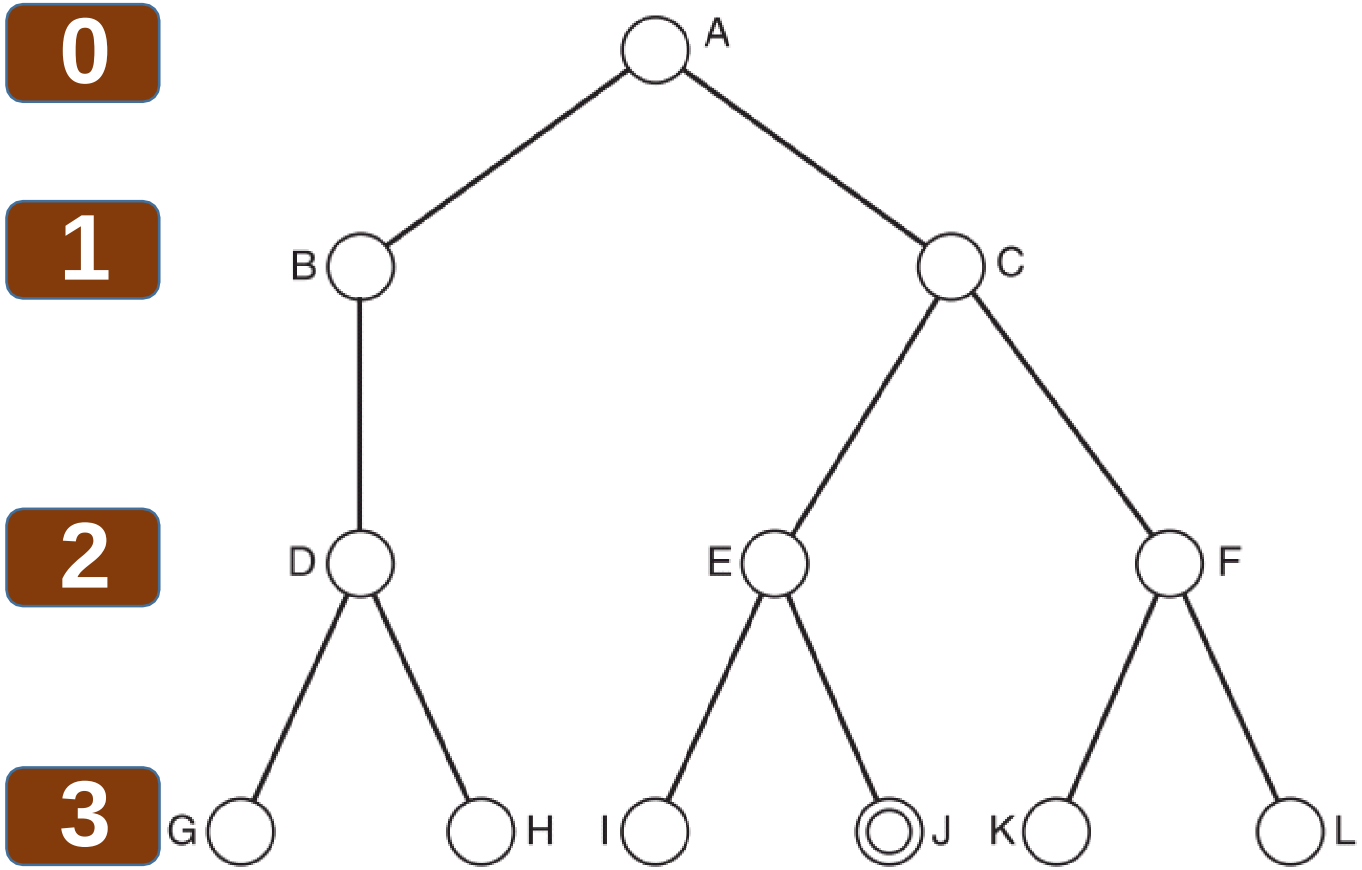


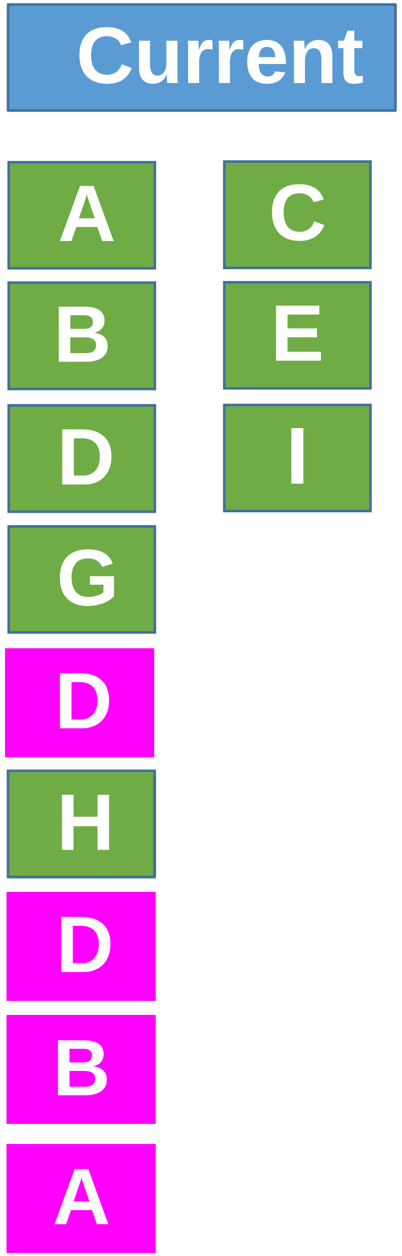
Current

A
B
D
G
D
H
D
B
A

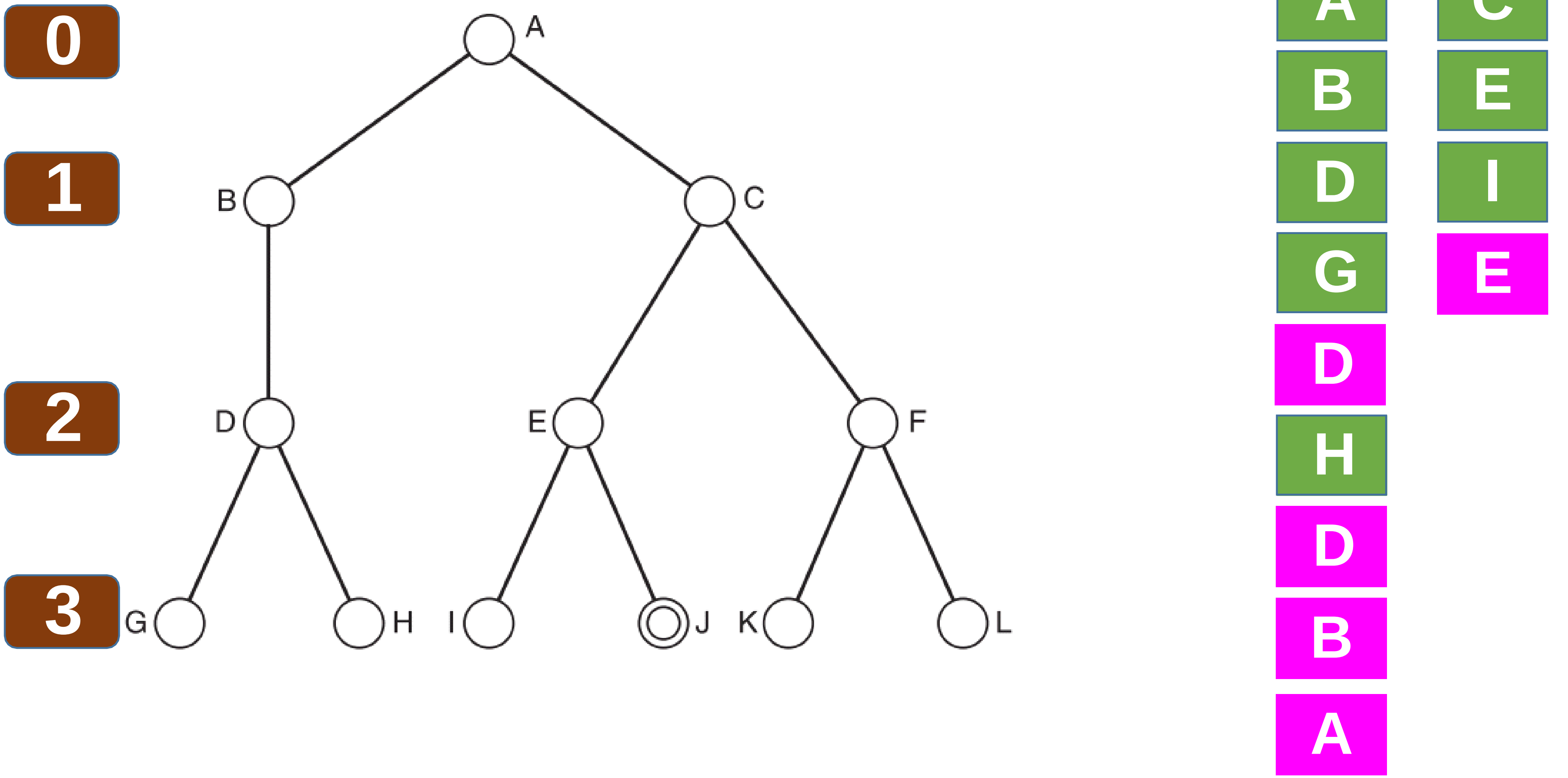
C
E
I

Depth-Limited Search
Depth – 3, Goal – Node J

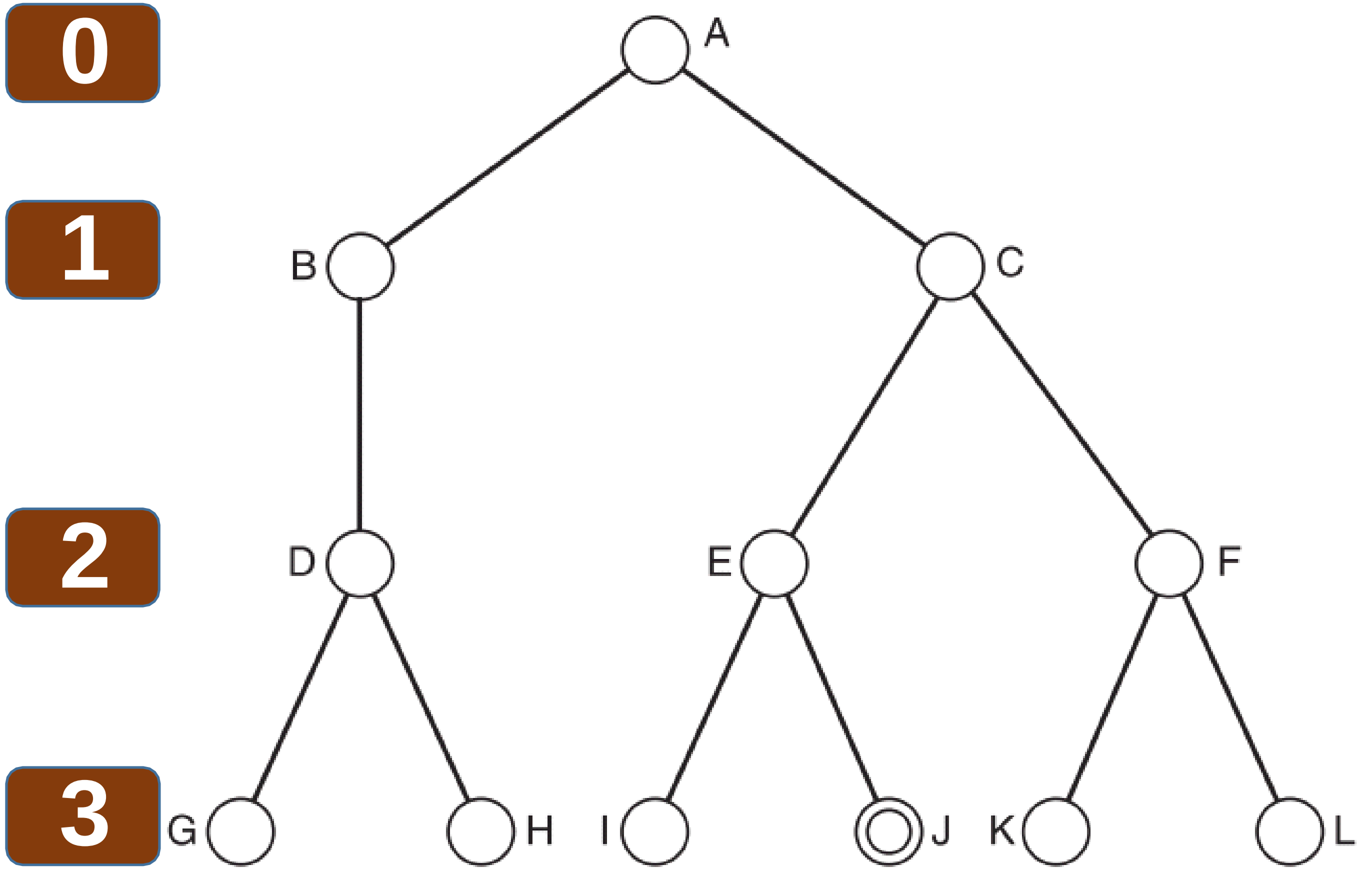




Depth-Limited Search
Depth – 3, Goal – Node J



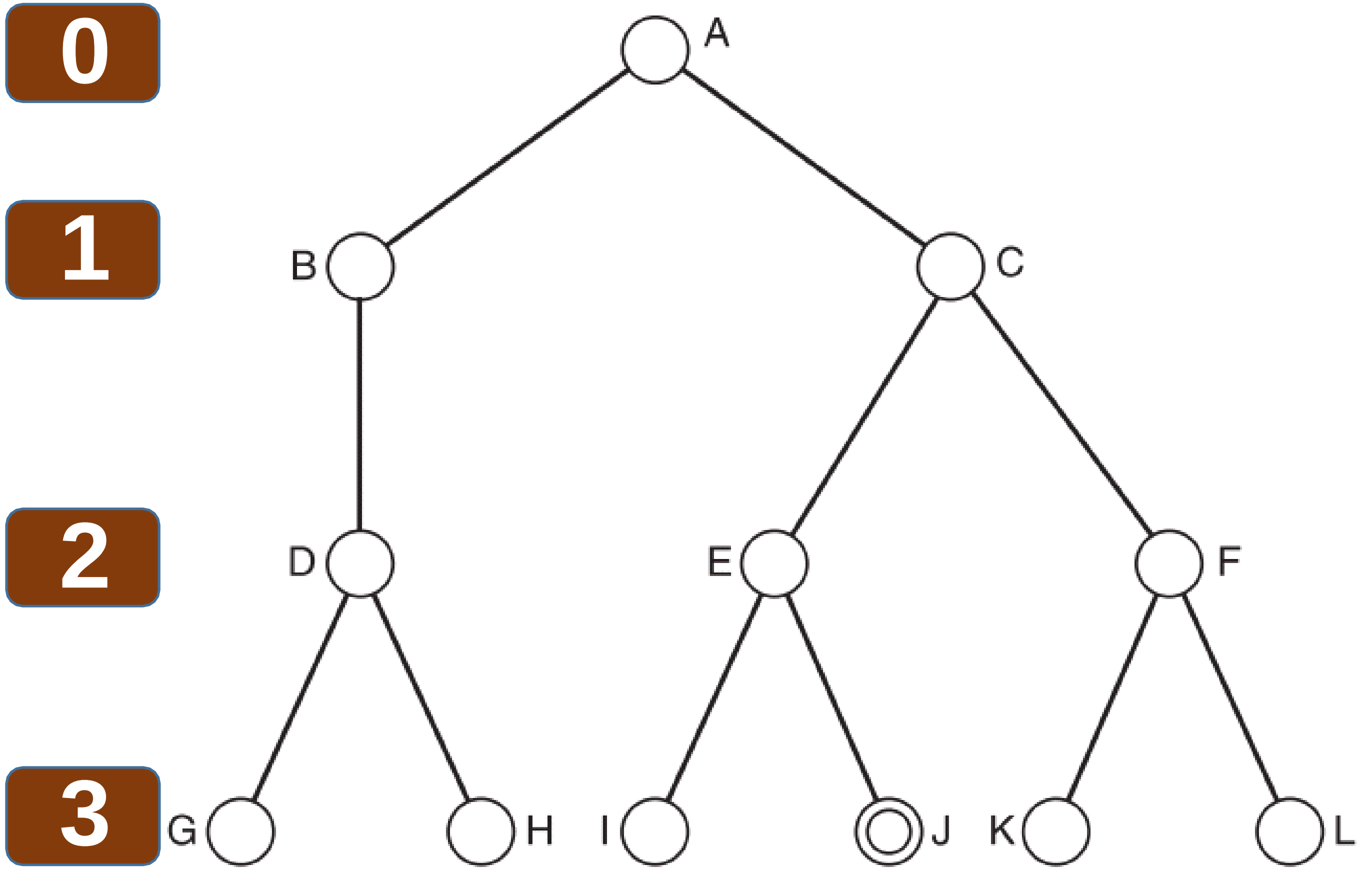
Depth-Limited Search
Depth – 3, Goal – Node J



Current

A	C
B	E
D	I
G	E
D	J
H	
D	
B	
A	

Depth-Limited Search
Depth – 3, Goal – Node J



Current

A
B
D
G
D
H
D
B
A

C
E
I
E
J

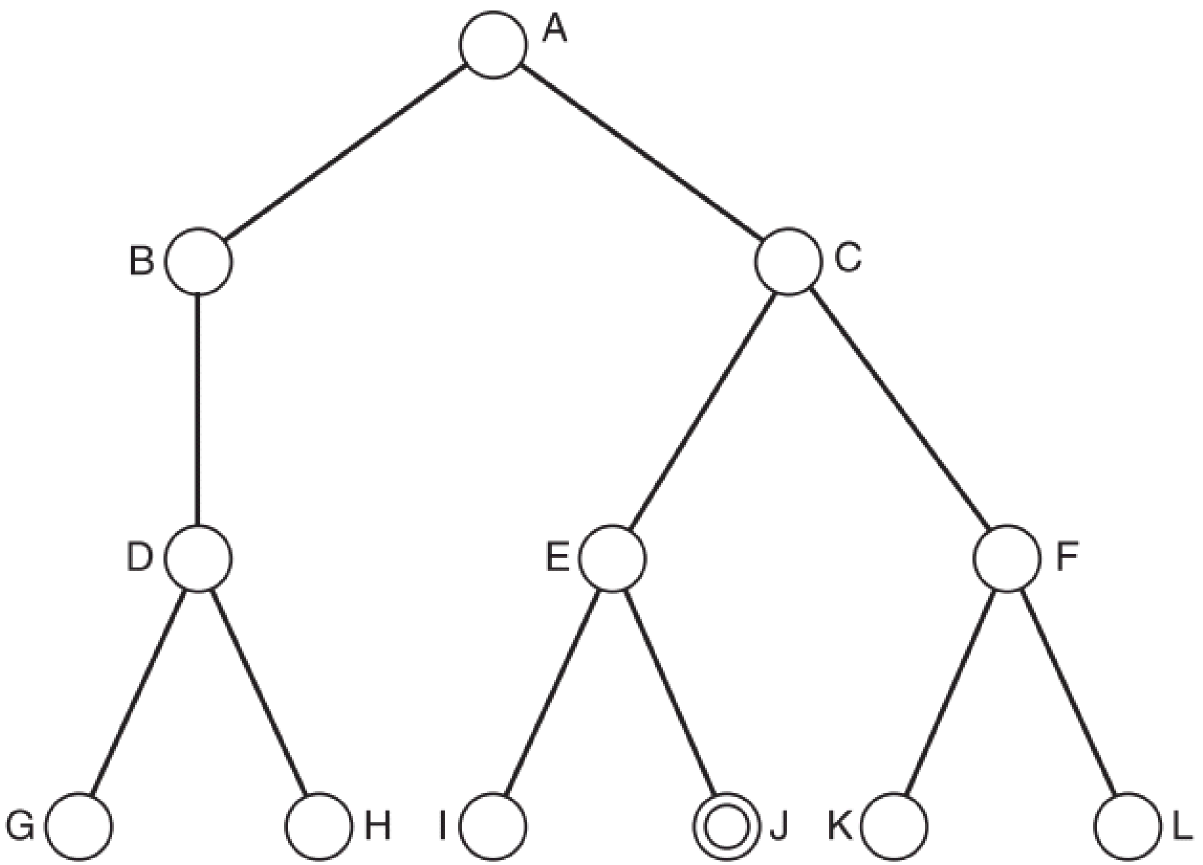
Depth-Limited Search
Depth – 3, Goal – Node J

0

1

2

3



Current

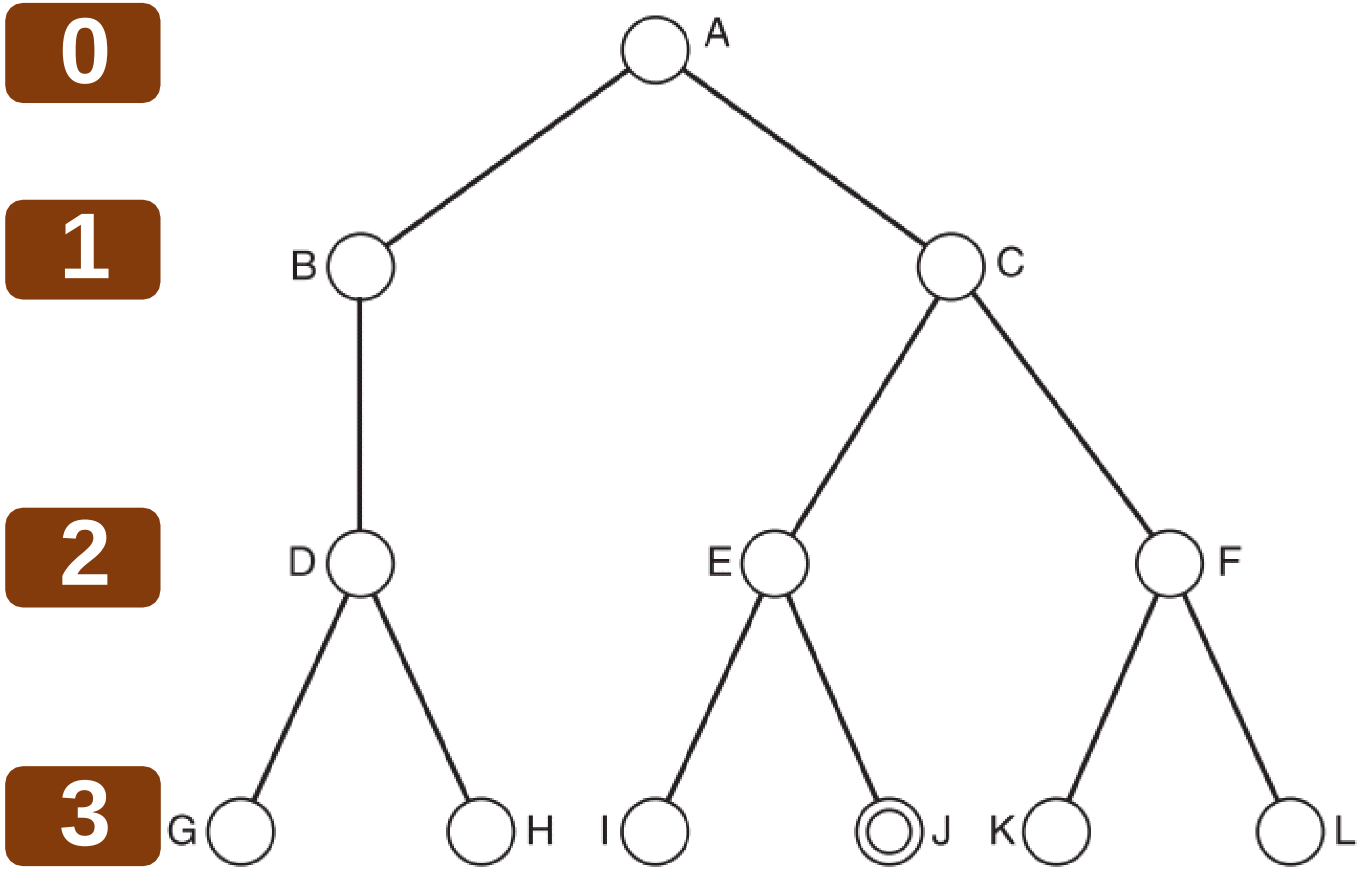
A
B
D
G
D
H
D
B
A

C
E
I
E
J

GOAL

Depth-Limited Search
Depth – 3, Goal – Node C

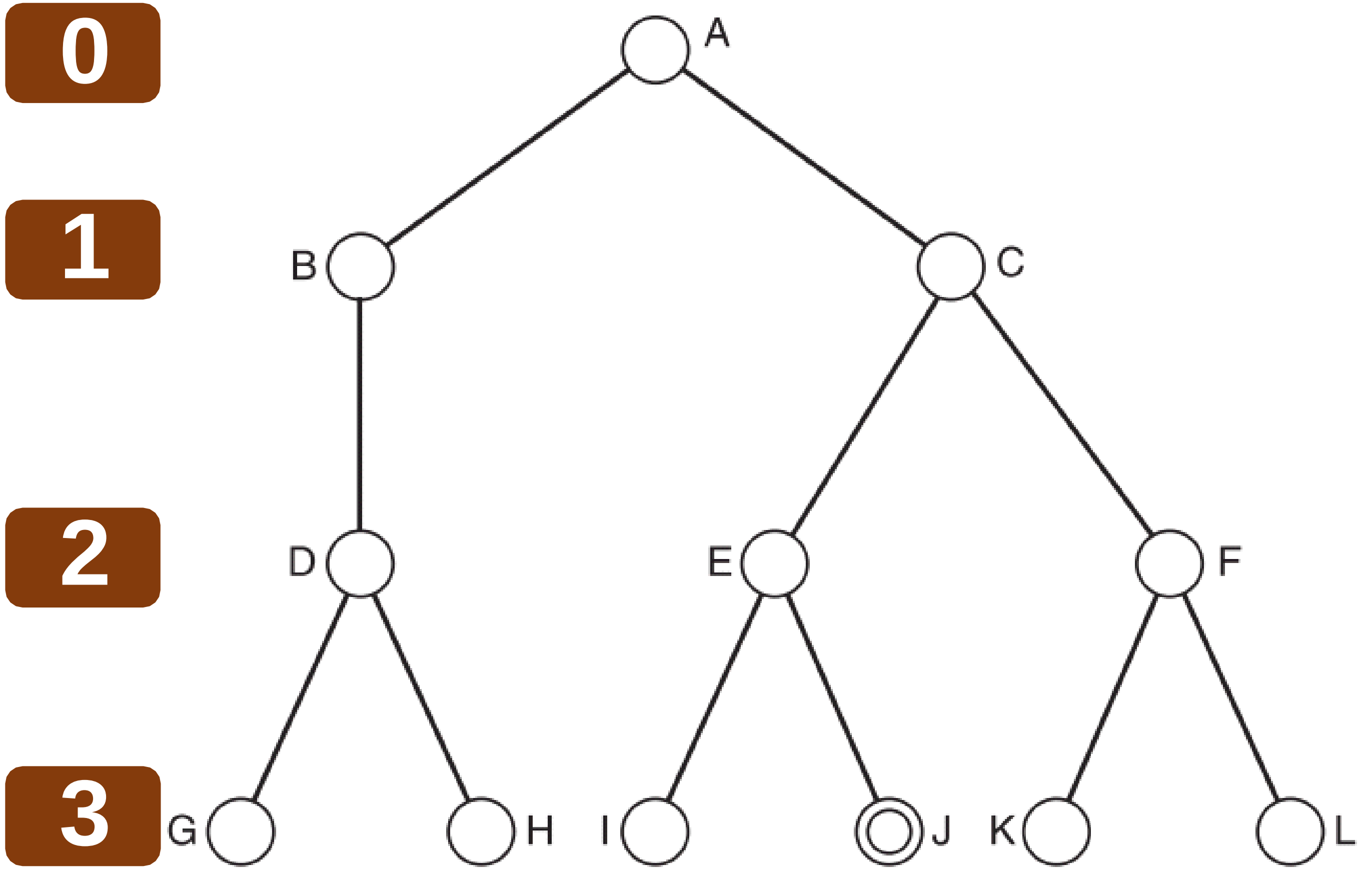
Current



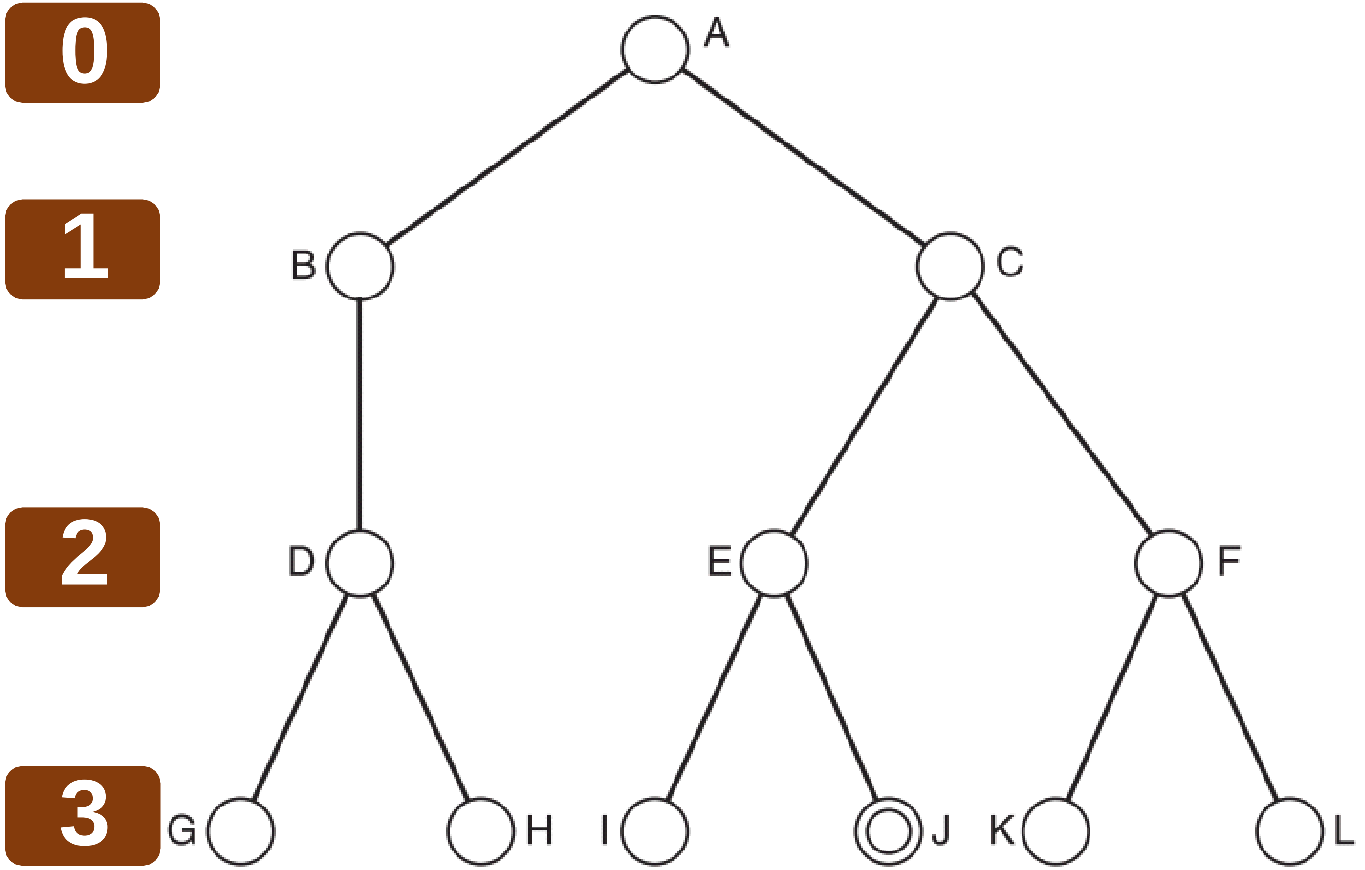
Depth-Limited Search
Depth – 3, Goal – Node C

Current

A



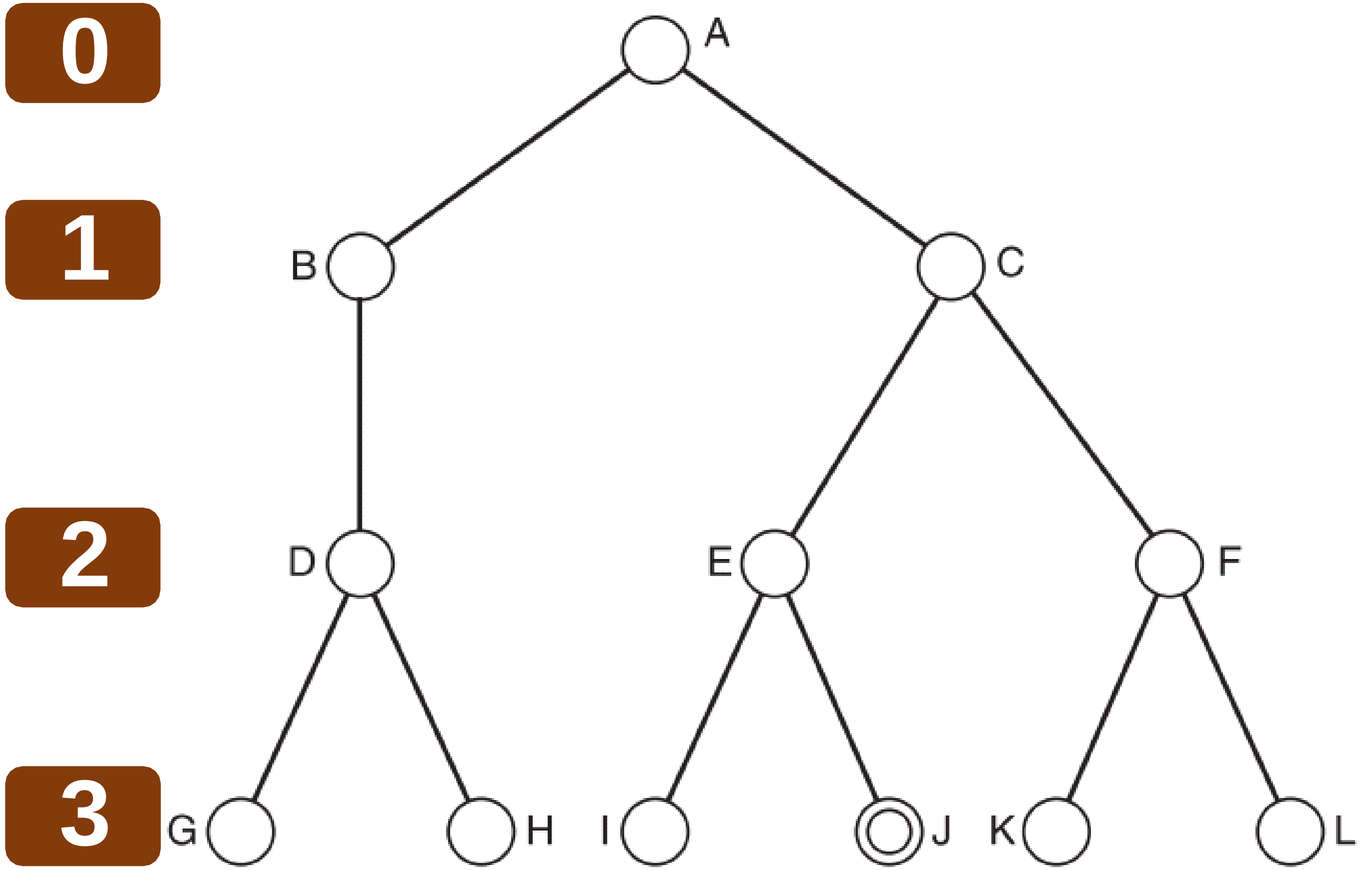
Depth-Limited Search
Depth – 3, Goal – Node C



Current

A

Depth-Limited Search
Depth – 3, Goal – Node C

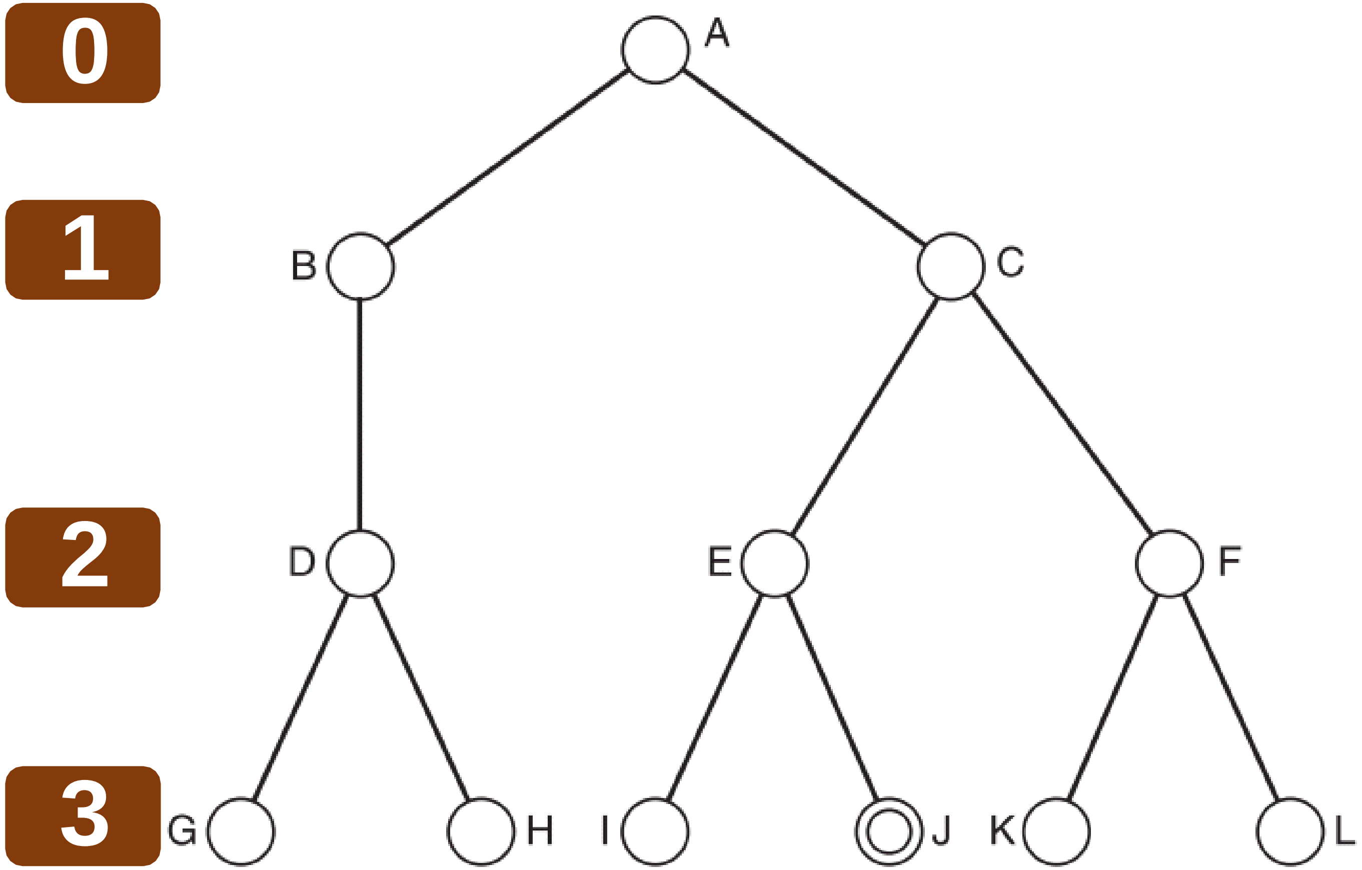


Current

A

B

Depth-Limited Search
Depth – 3, Goal – Node C

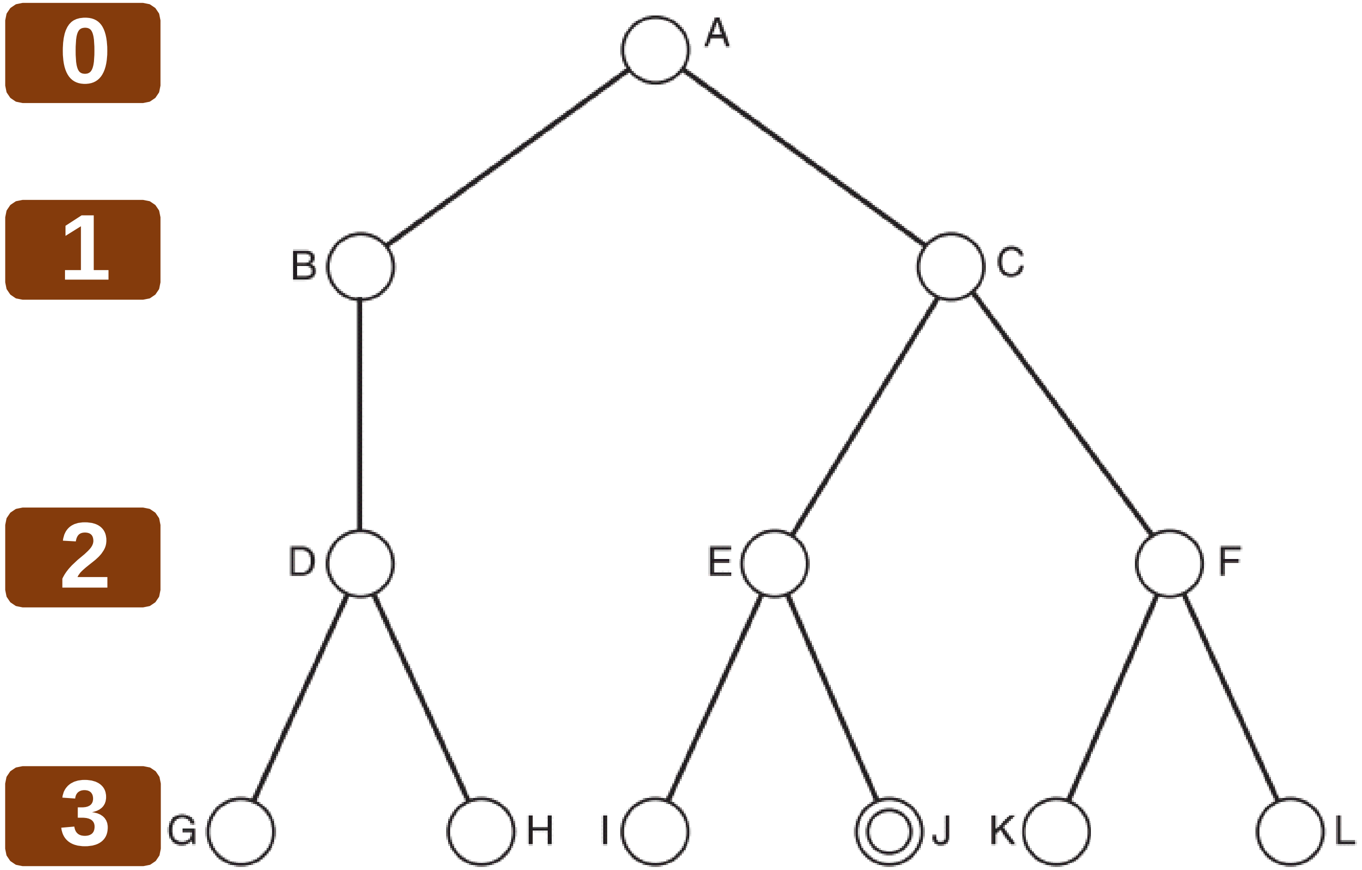


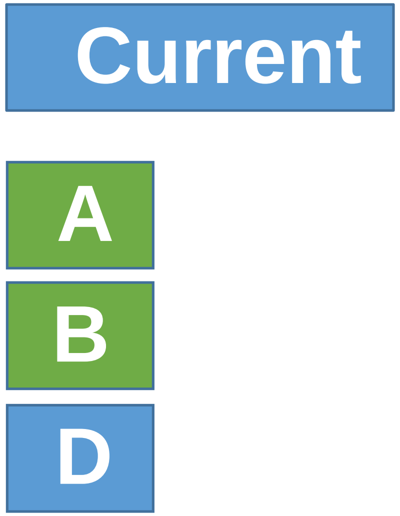
Current

A

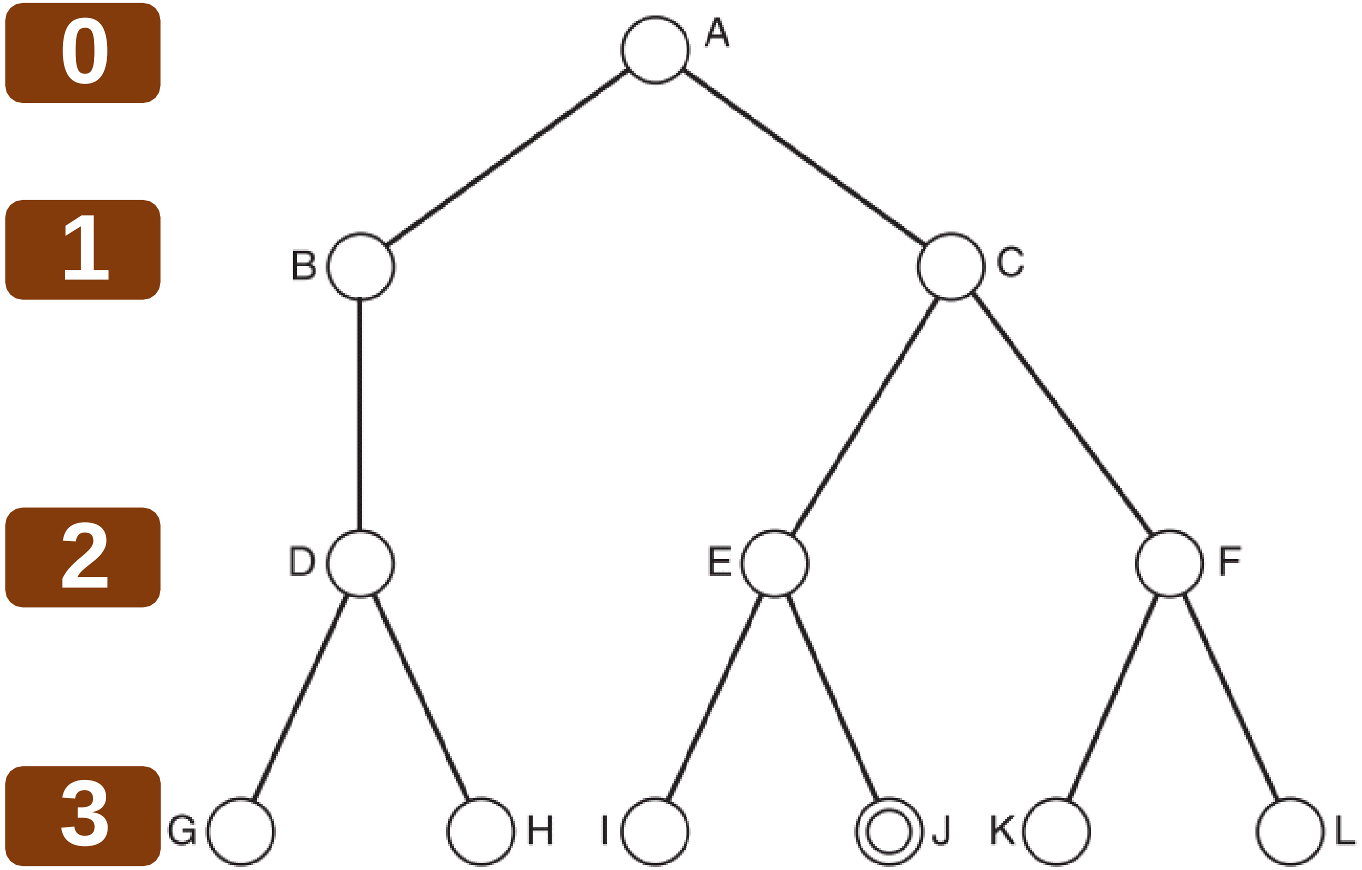
B

Depth-Limited Search
Depth – 3, Goal – Node C





Depth-Limited Search
Depth – 3, Goal – Node C



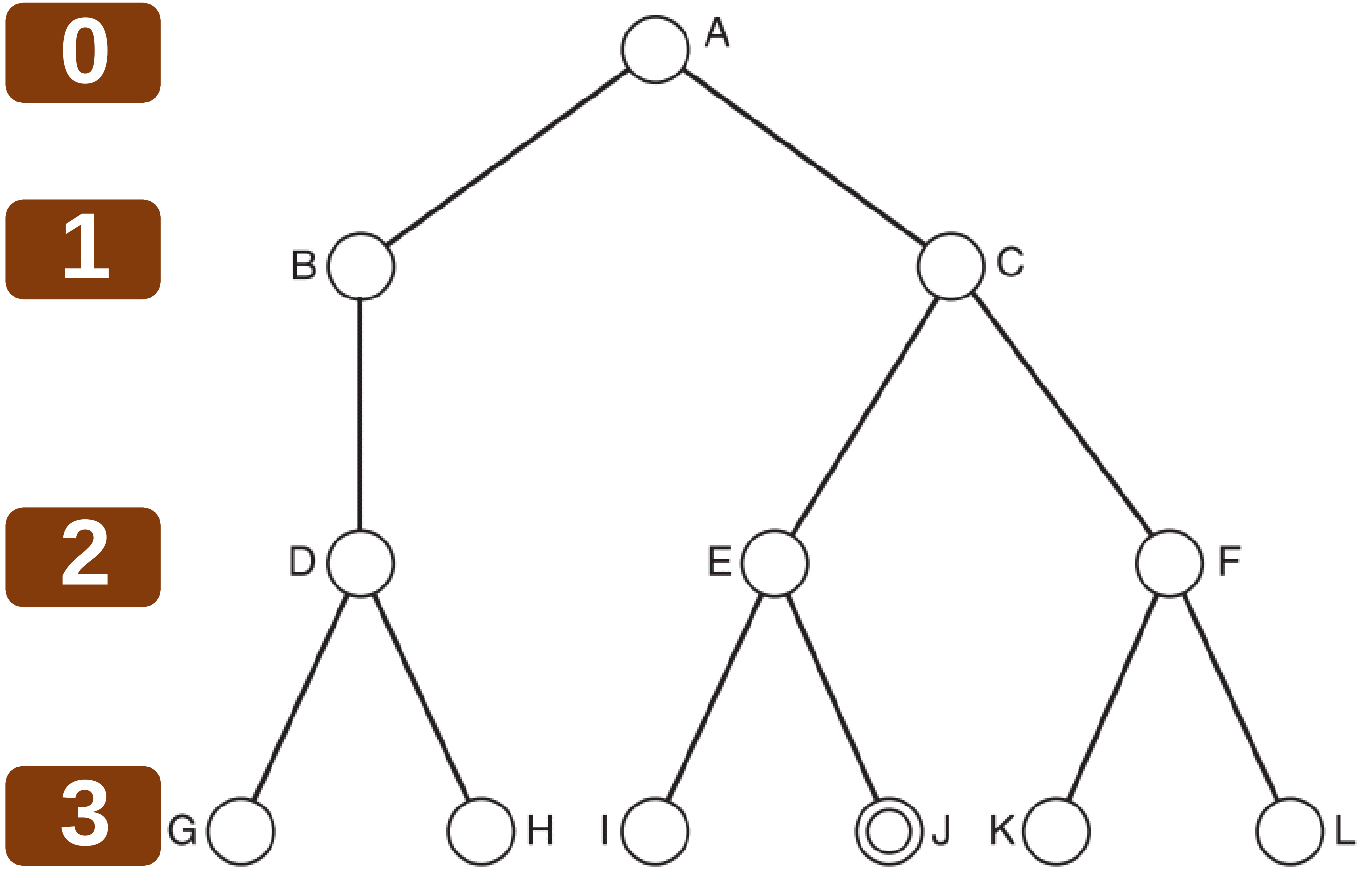
Current

A

B

D

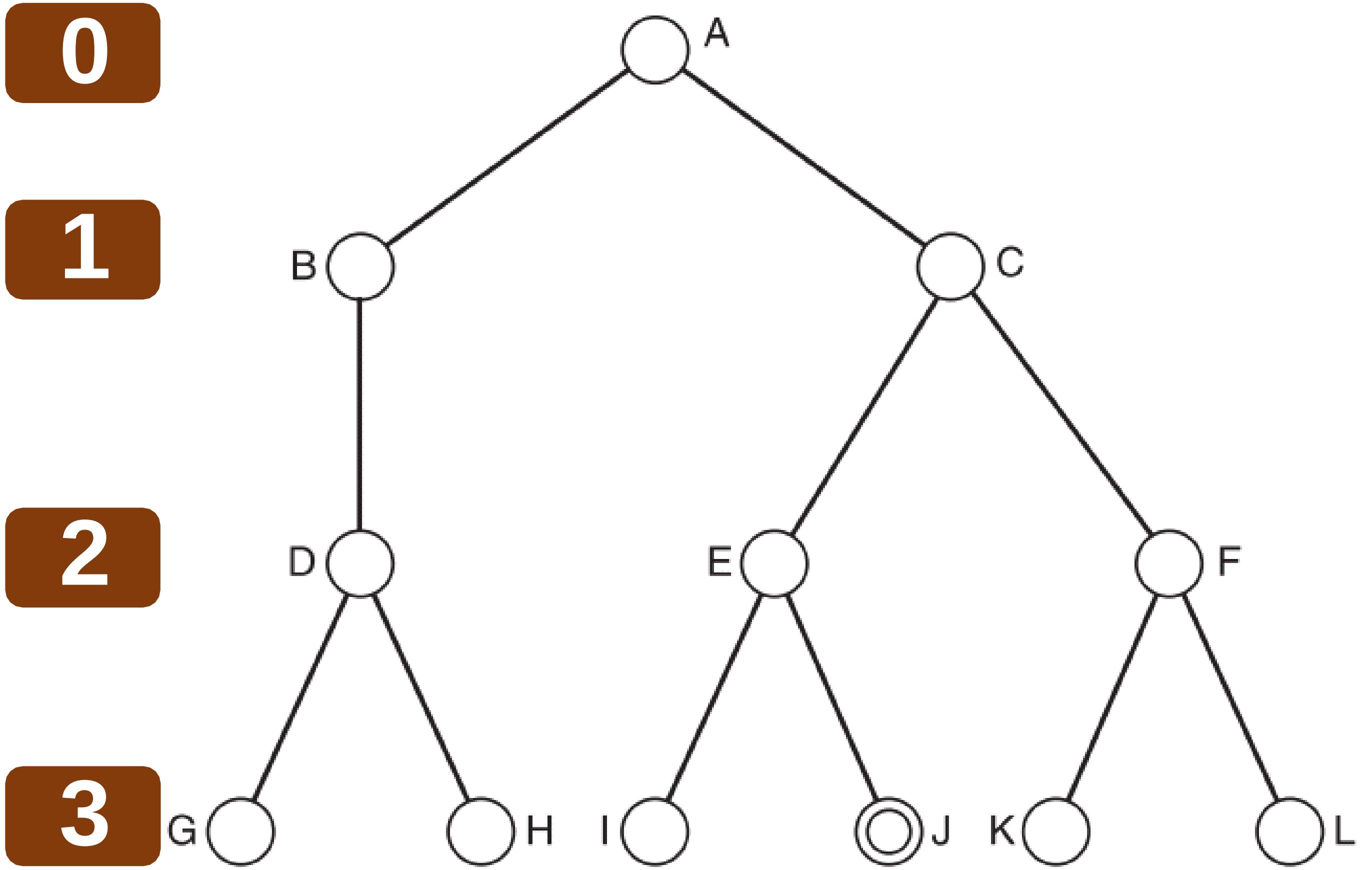
Depth-Limited Search
Depth – 3, Goal – Node C



Current

A
B
D
G

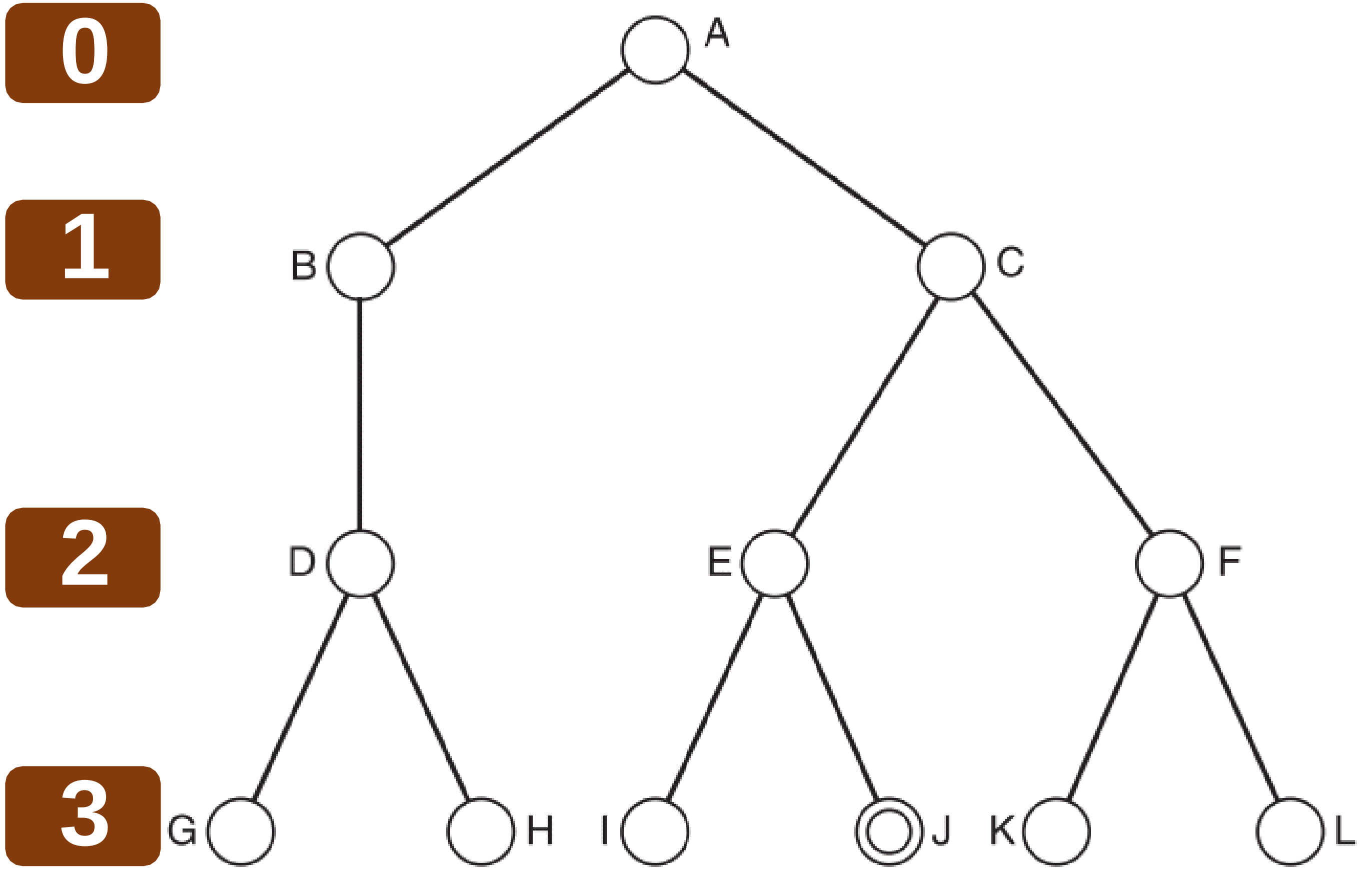
Depth-Limited Search
Depth – 3, Goal – Node C



Current

A
B
D
G

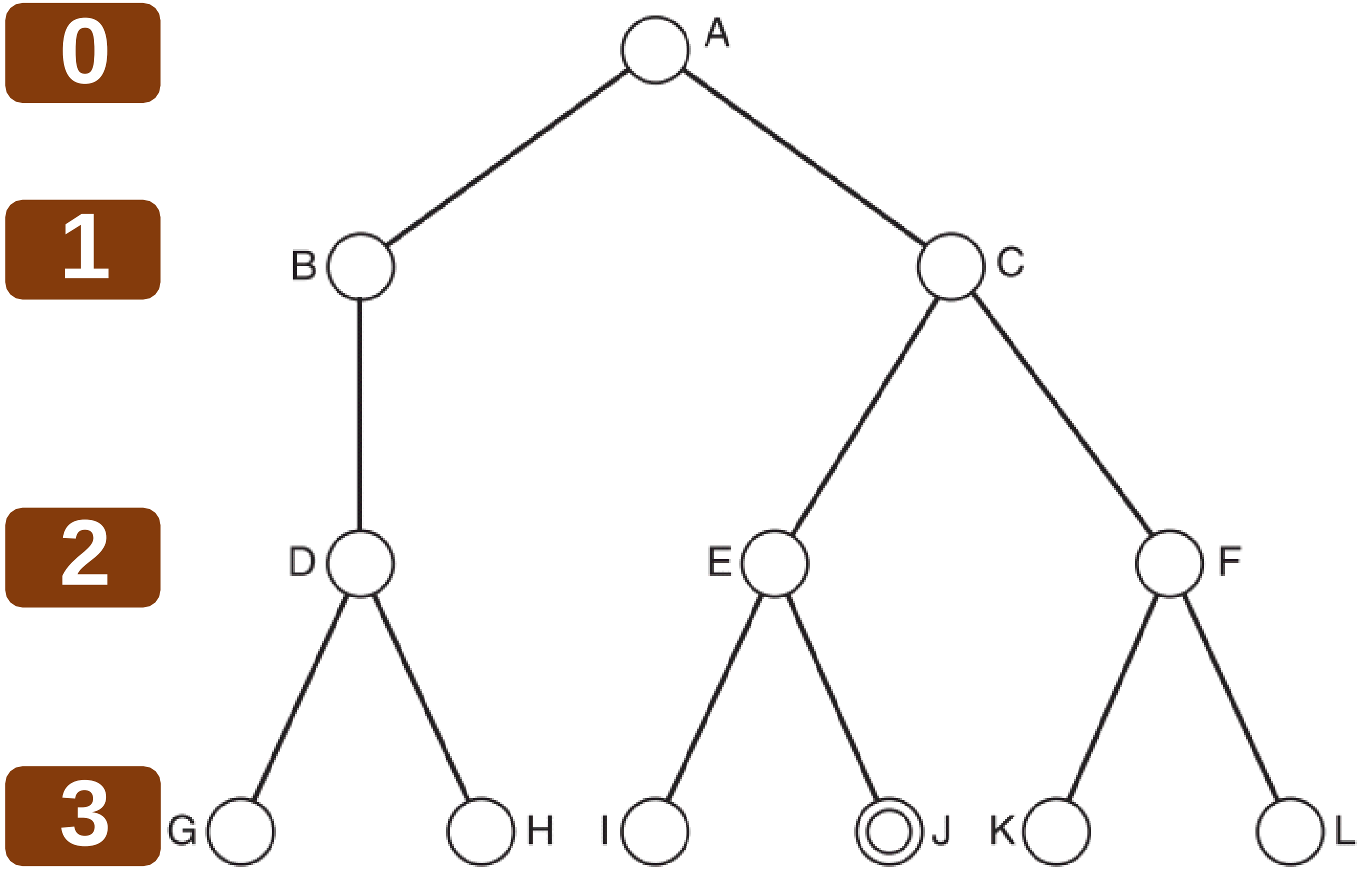
Depth-Limited Search
Depth – 3, Goal – Node C



Current

A
B
D
G
D

Depth-Limited Search
Depth – 3, Goal – Node C



Current

A

B

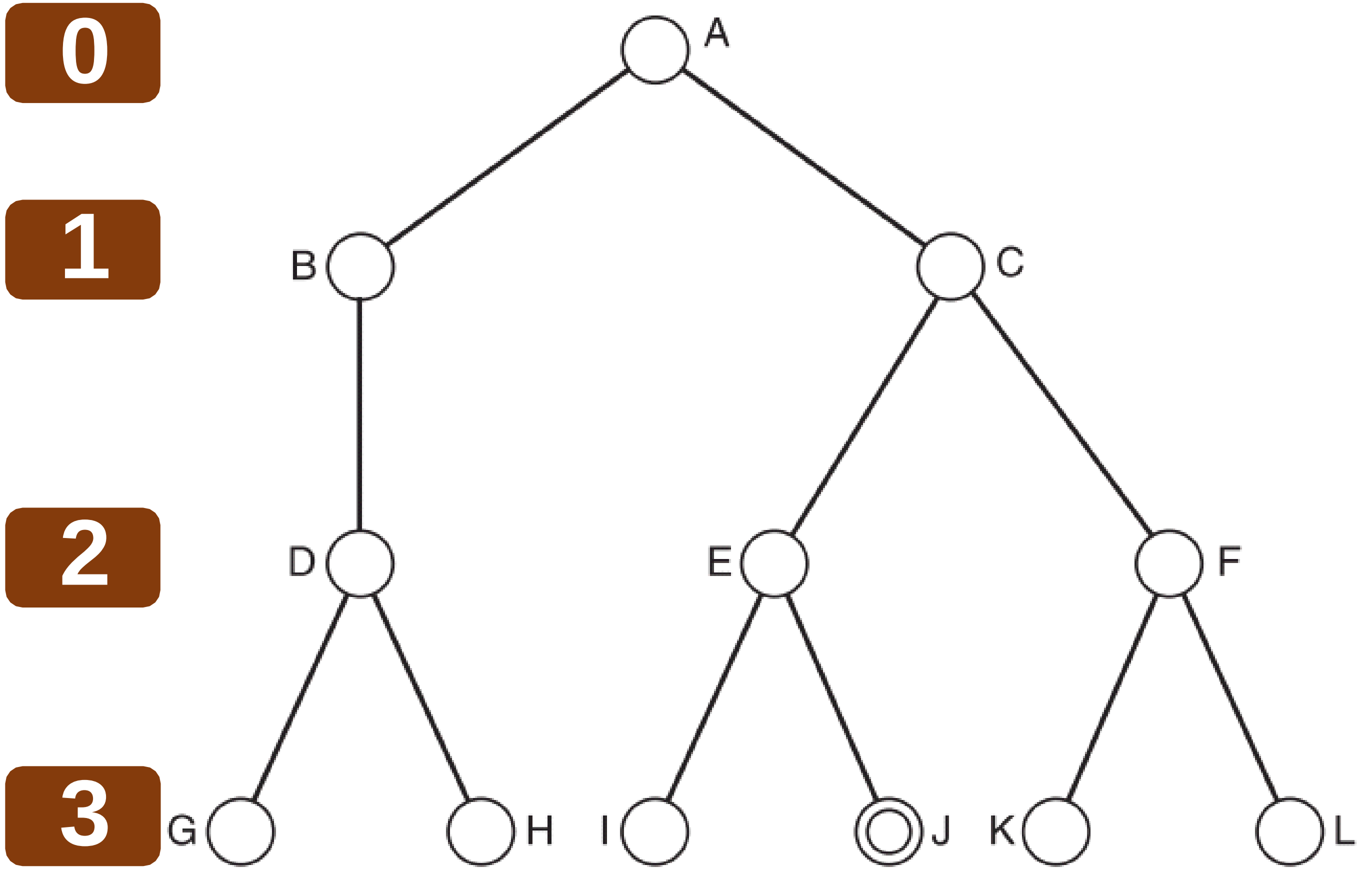
D

G

D

H

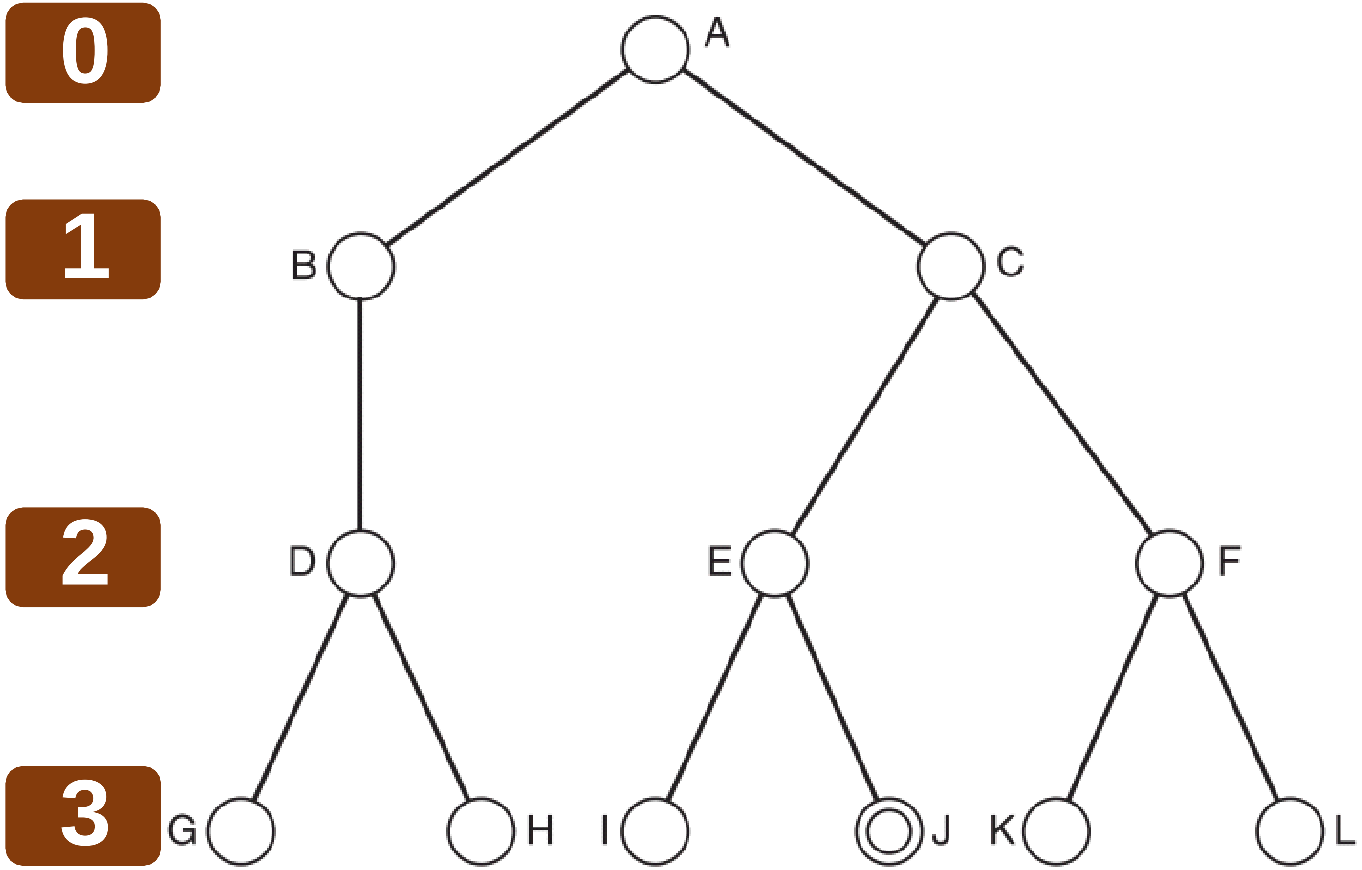
Depth-Limited Search
Depth – 3, Goal – Node C



Current

A
B
D
G
D
H

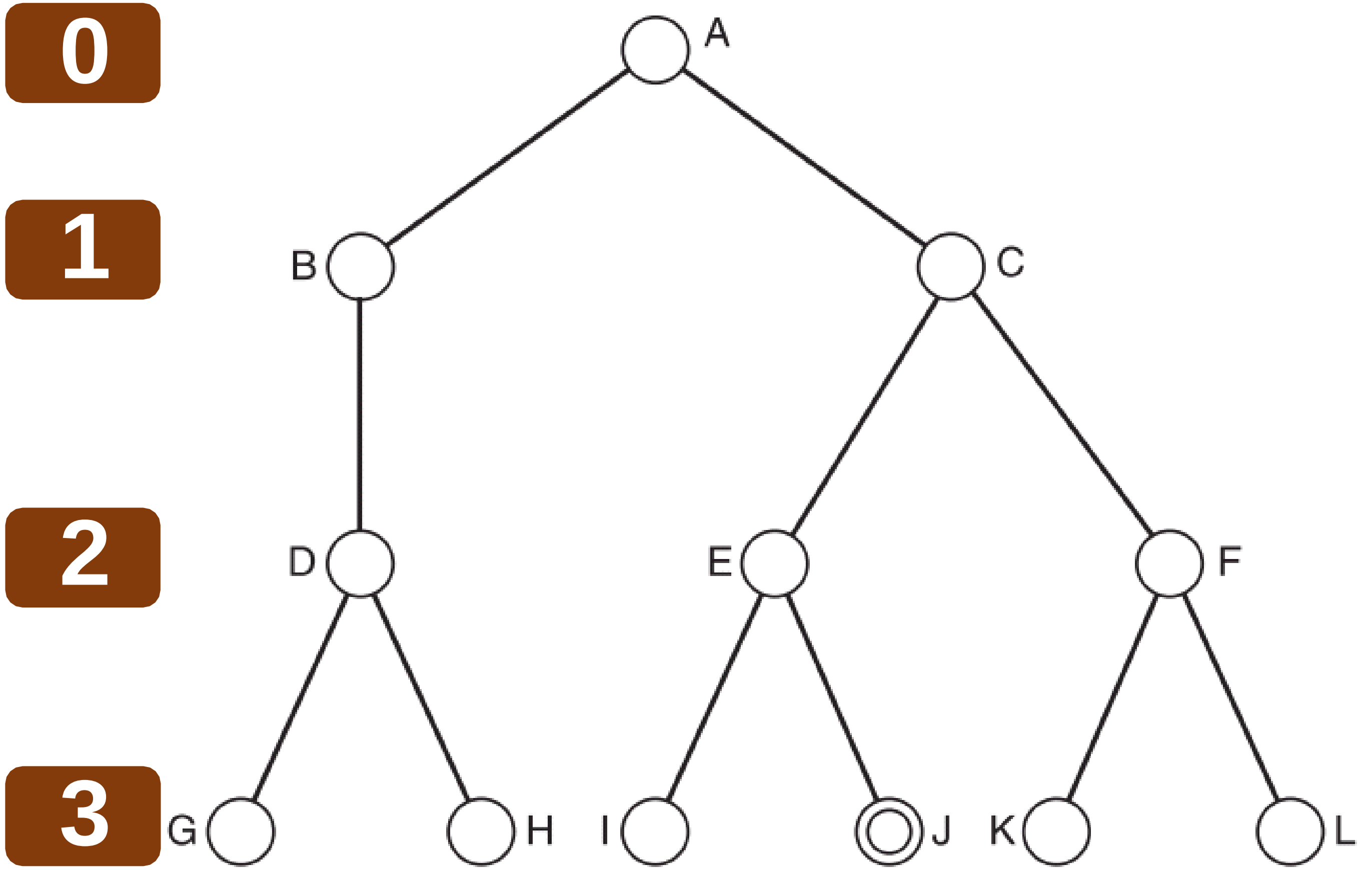
Depth-Limited Search
Depth – 3, Goal – Node C

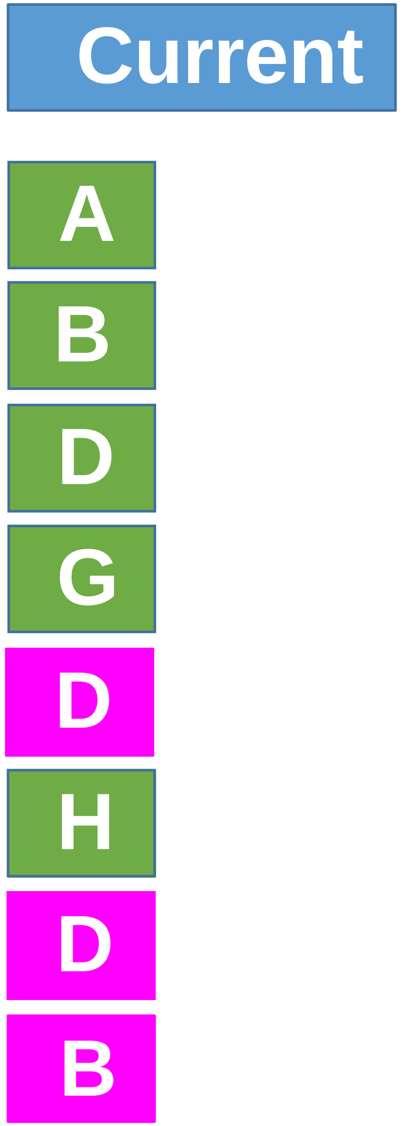


Current

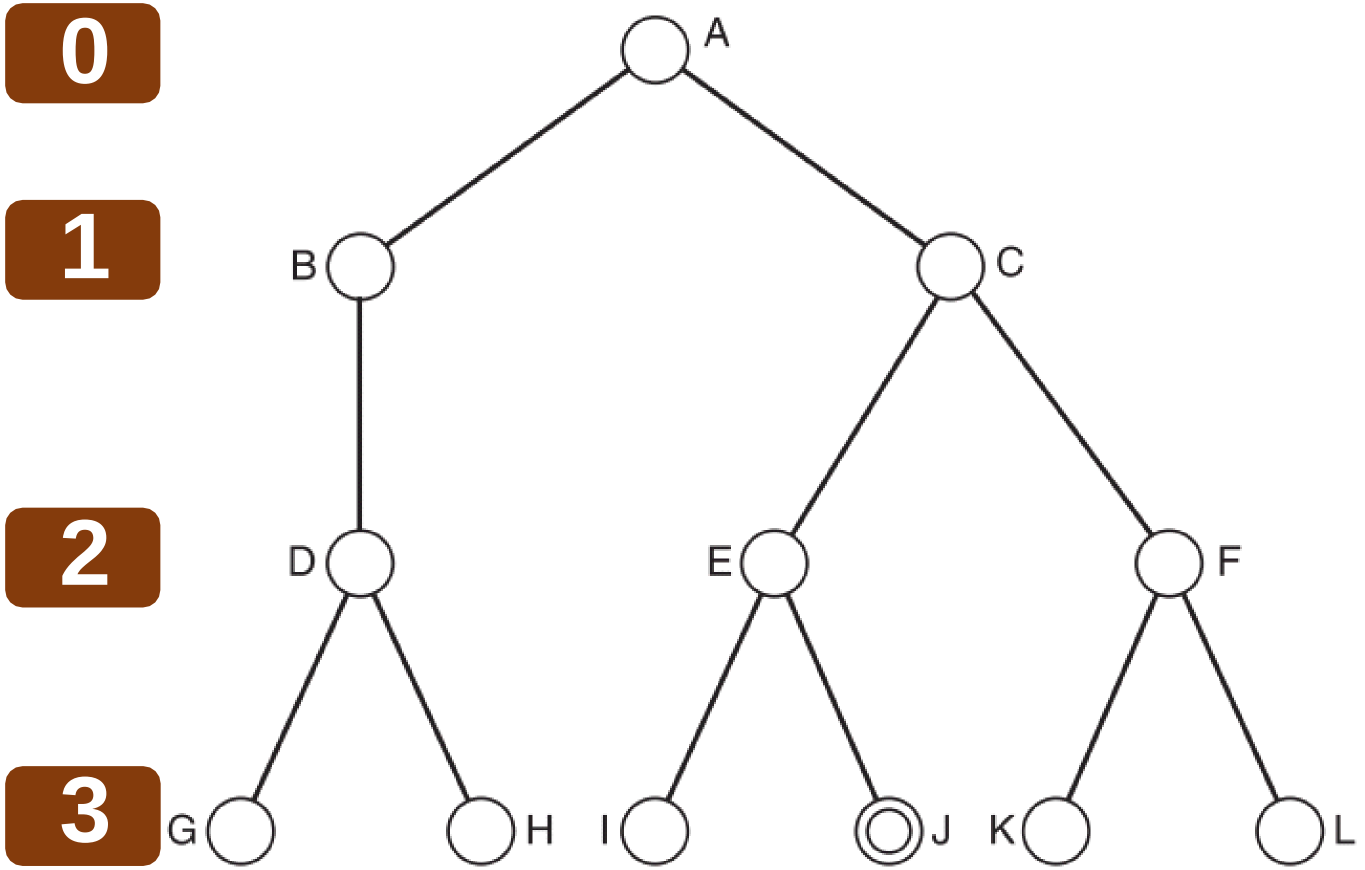
- A
- B
- D
- G
- D
- H
- D

Depth-Limited Search
Depth – 3, Goal – Node C





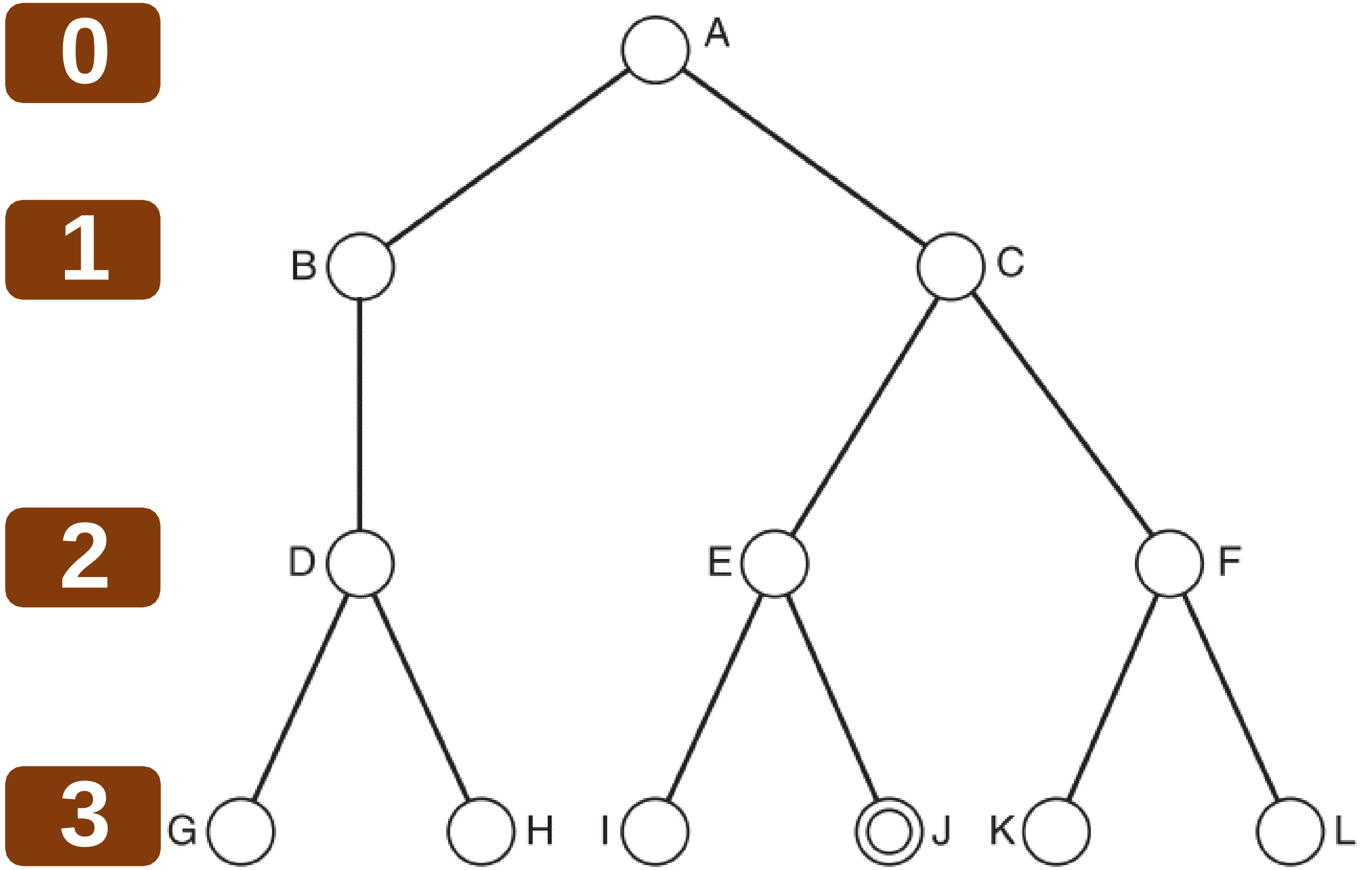
Depth-Limited Search
Depth – 3, Goal – Node C



Current

- A
- B
- D
- G
- D
- H
- D
- B
- A

Depth-Limited Search
Depth – 3, Goal – Node C



Current

A

B

D

G

D

H

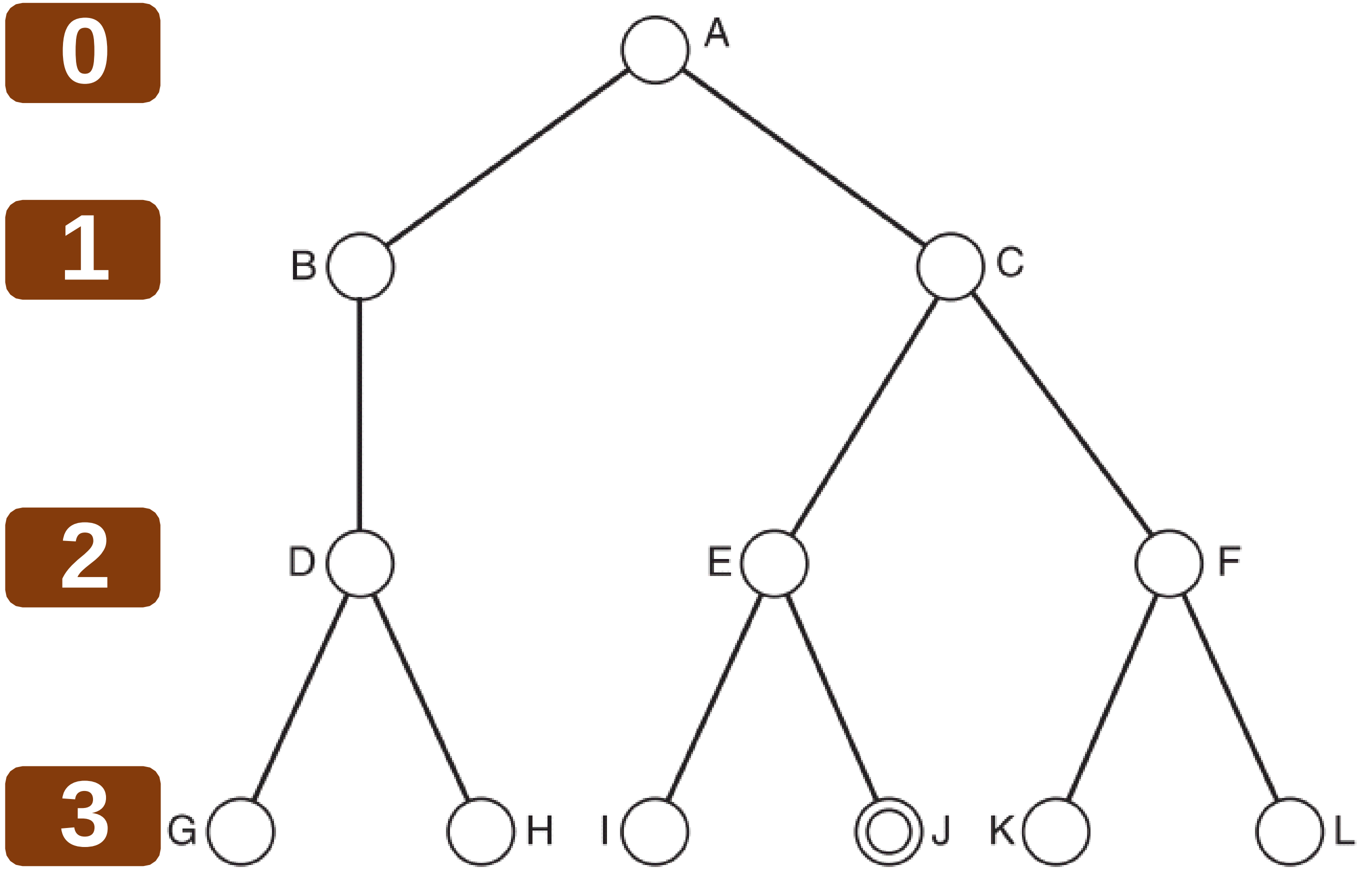
D

B

A

C

Depth-Limited Search
Depth – 3, Goal – Node C



Current

A C

B

D

G

D

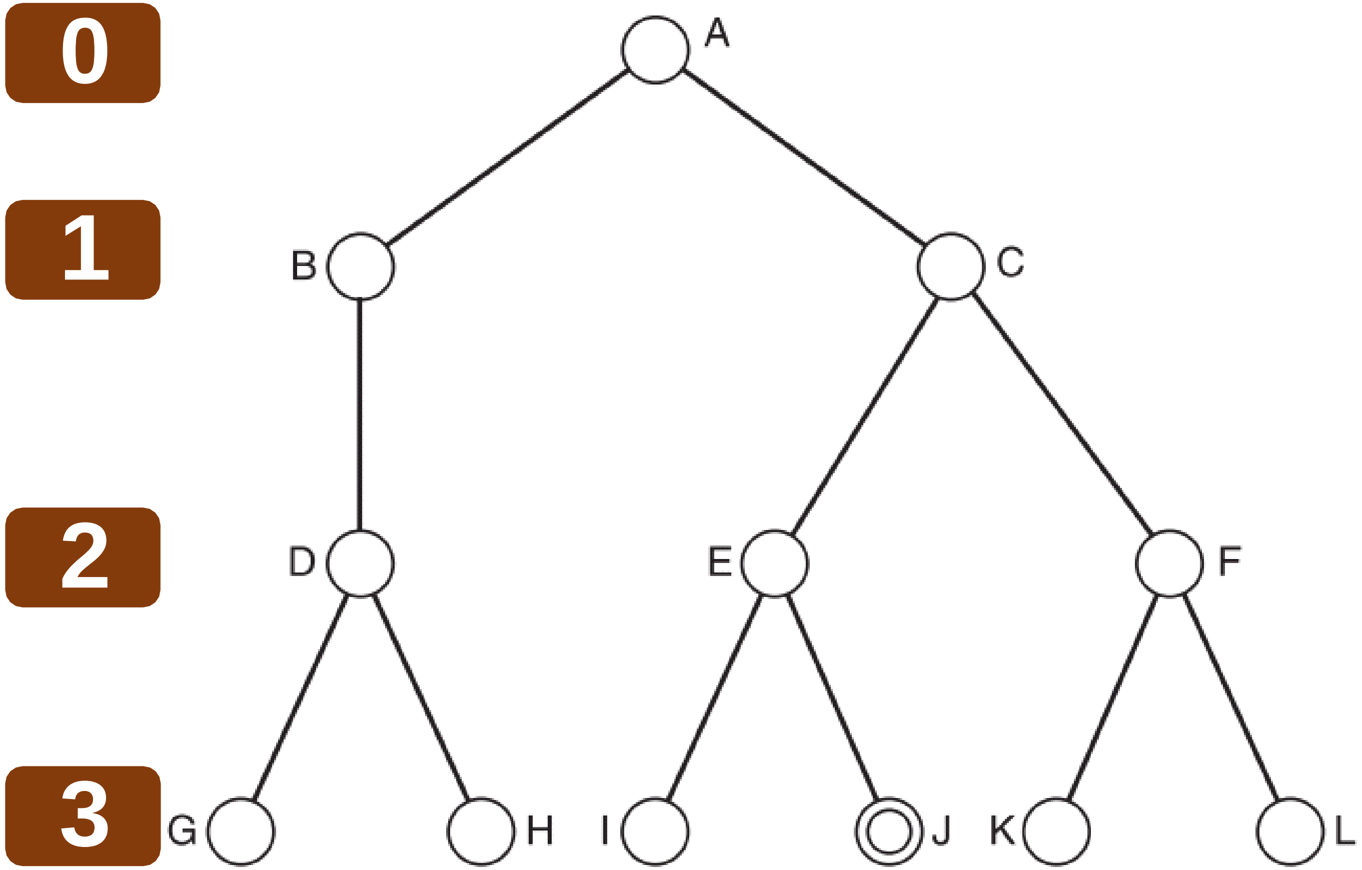
H

D

B

A

Depth-Limited Search
Depth – 3, Goal – Node C



Current

A
B
D
G
D
H
D
B
A

C

GOAL

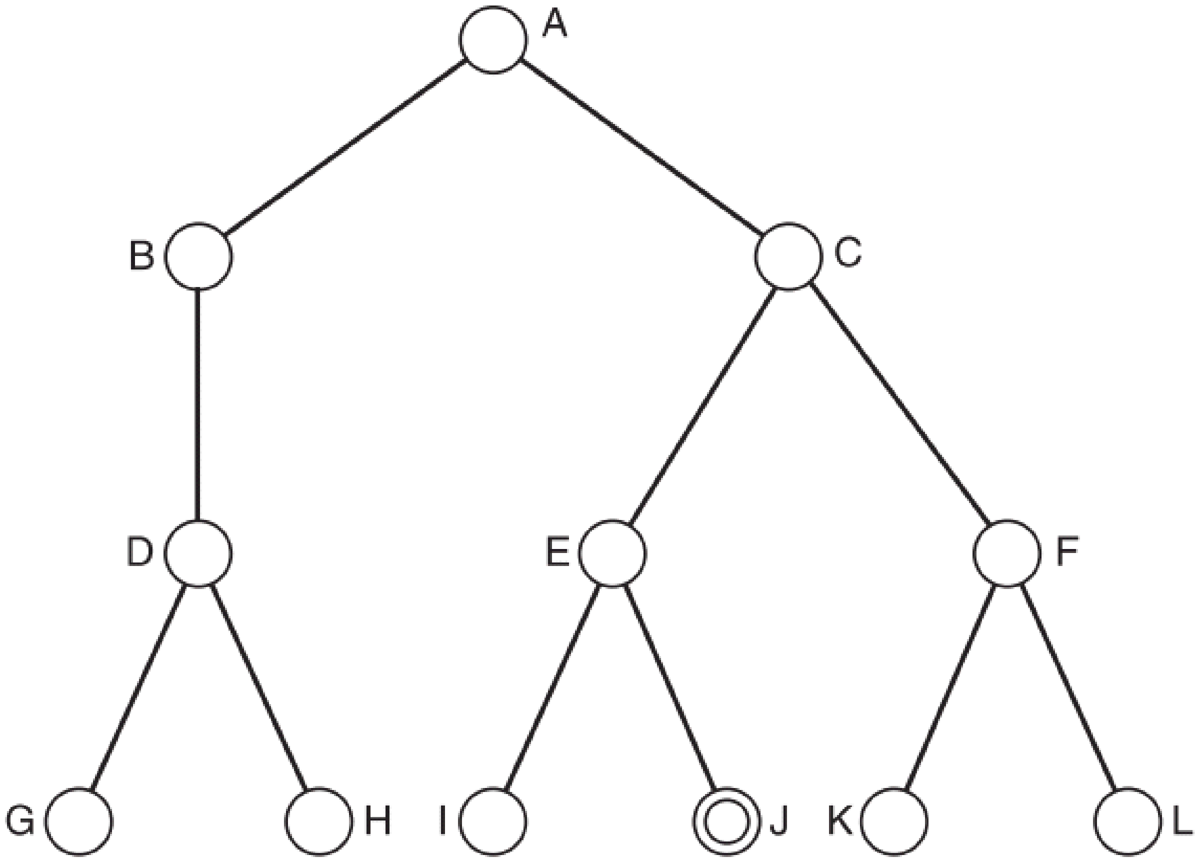
Depth-Limited Search
Depth – 3, Goal – Node C

0

1

2

3



Depth is Large

Current

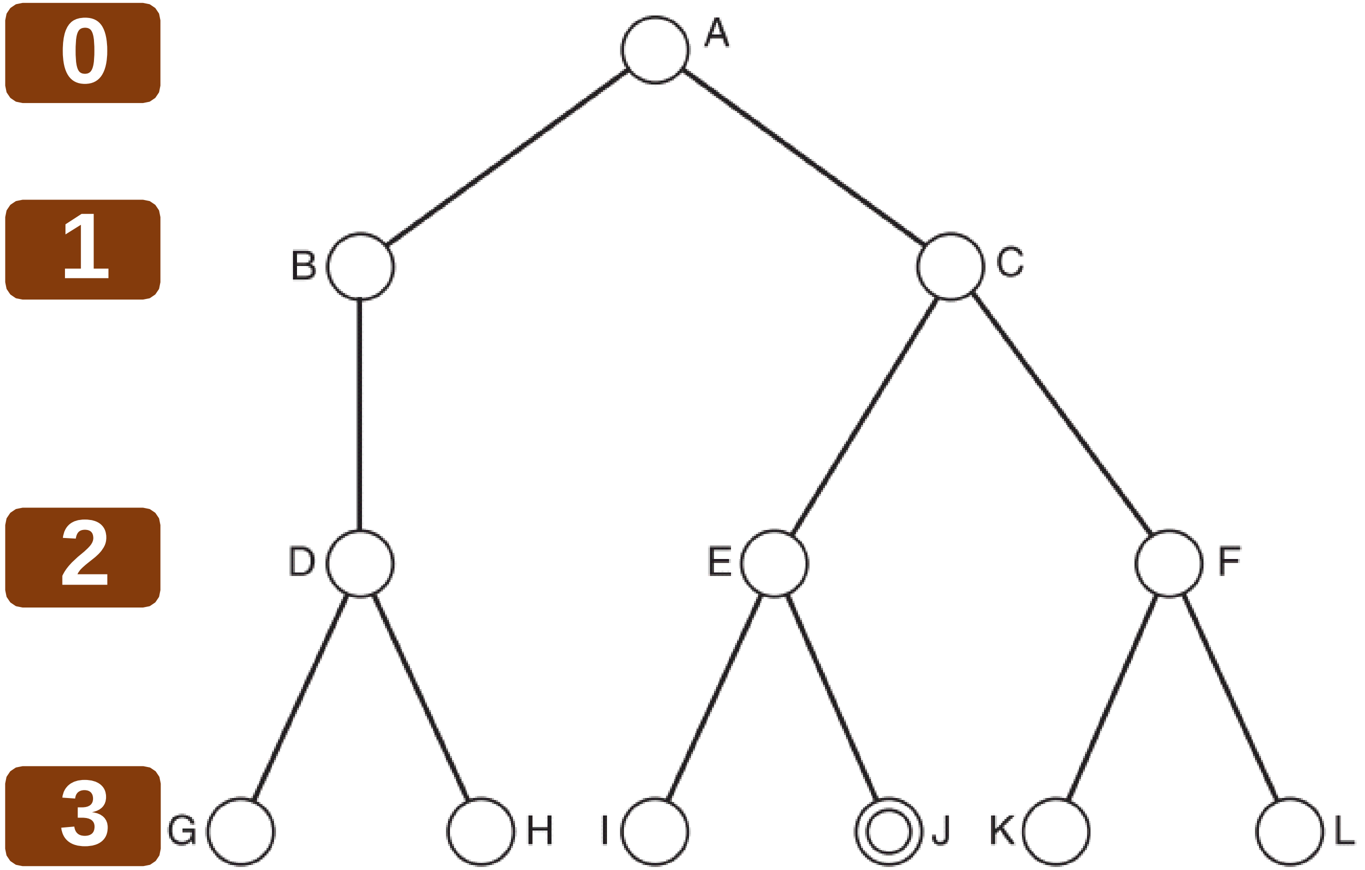
A
B
D
G
D
H
D
B
A

C

GOAL

Depth-Limited Search
Depth – 2, Goal – Node J

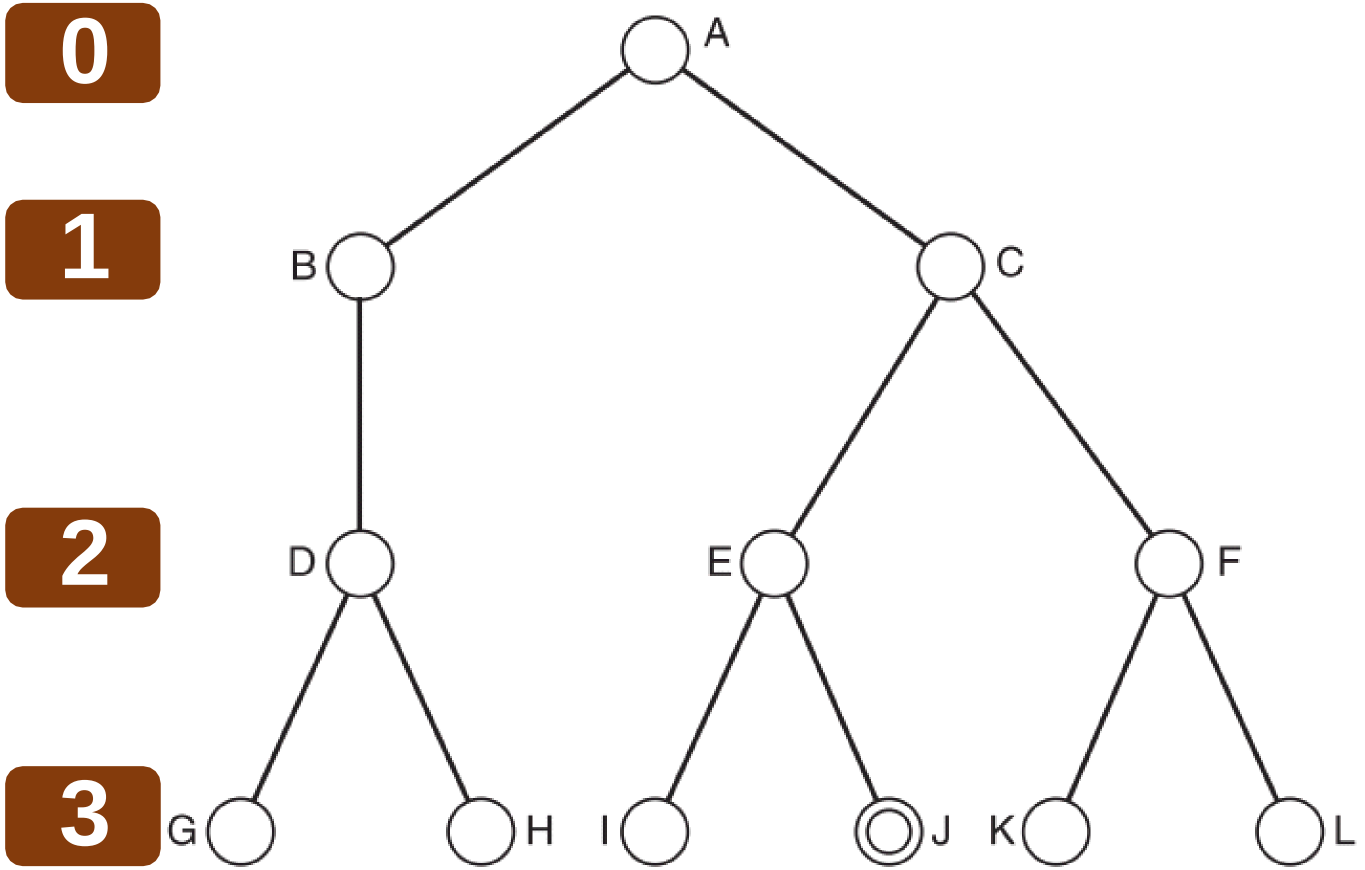
Current



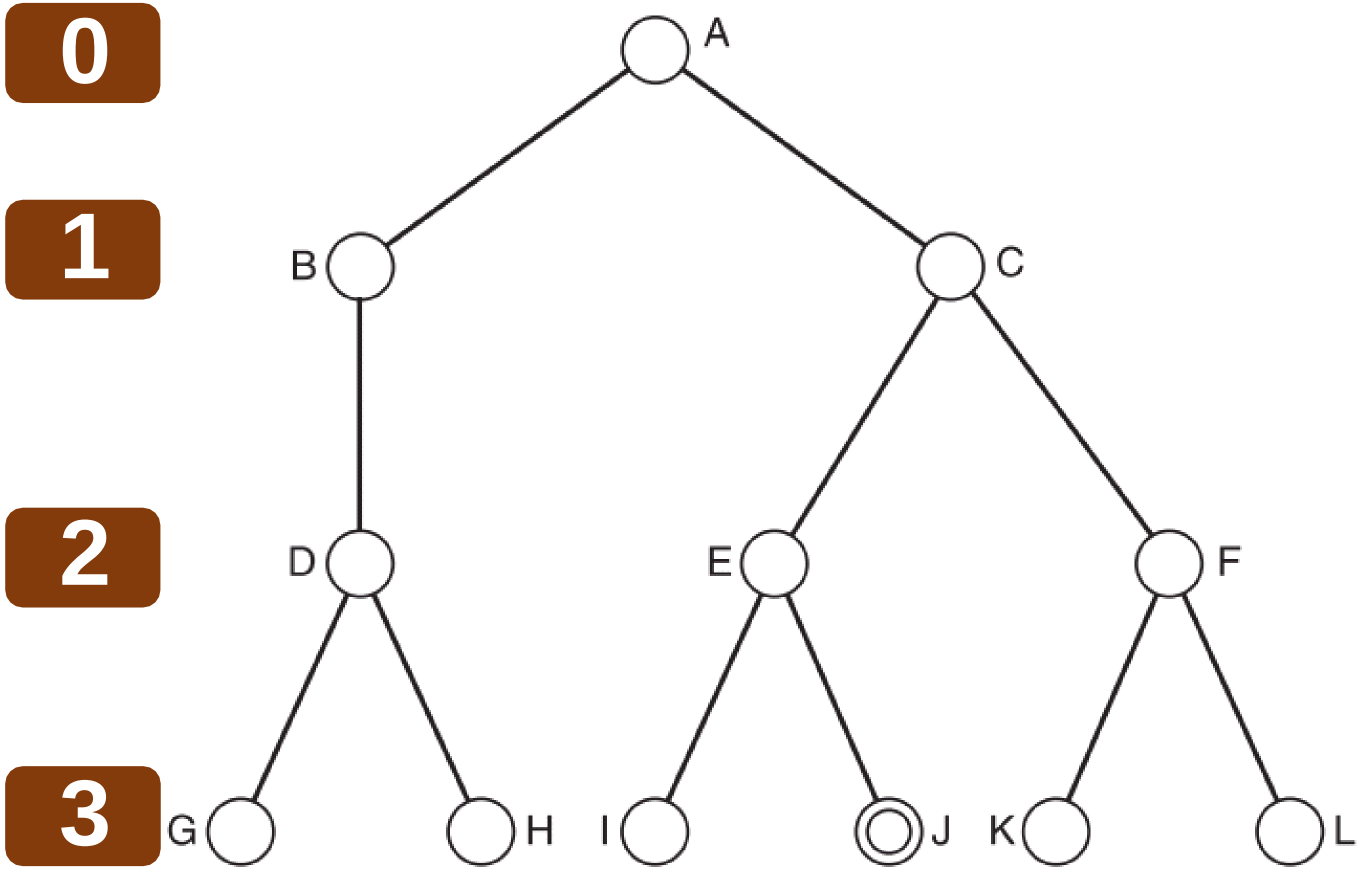
Depth-Limited Search
Depth – 2, Goal – Node J

Current

A



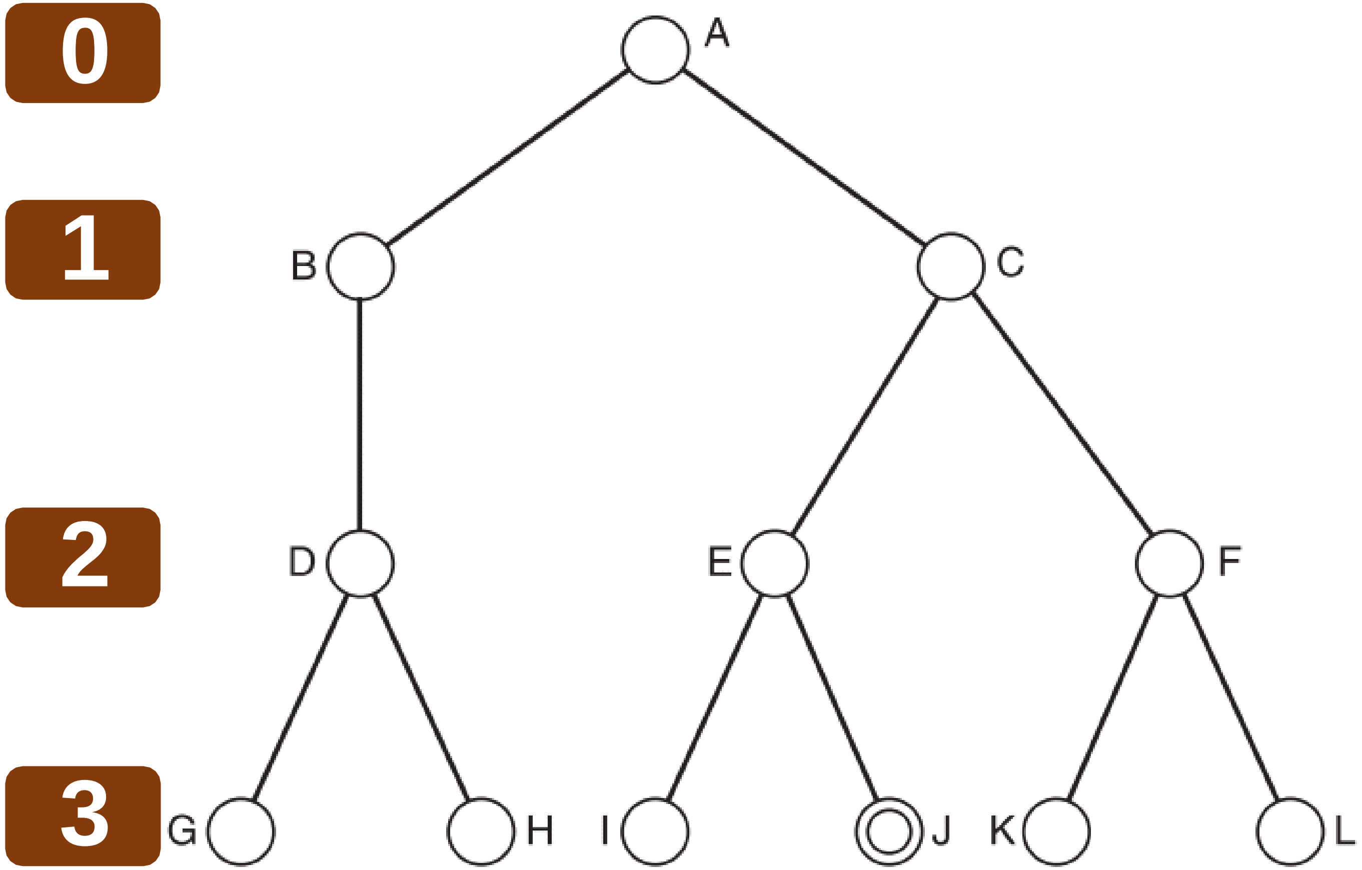
Depth-Limited Search
Depth – 2, Goal – Node J



Current

A

Depth-Limited Search
Depth – 2, Goal – Node J

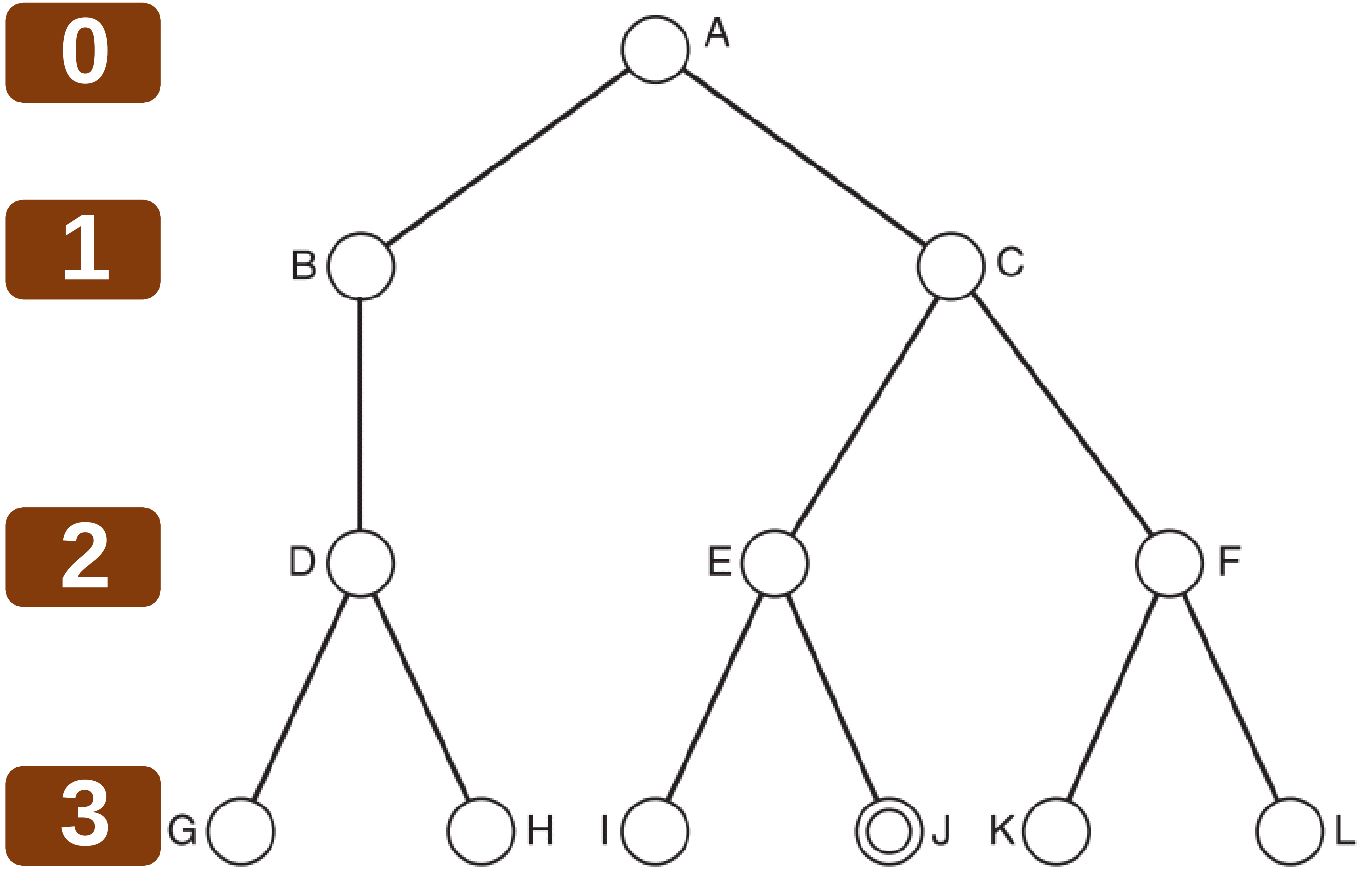


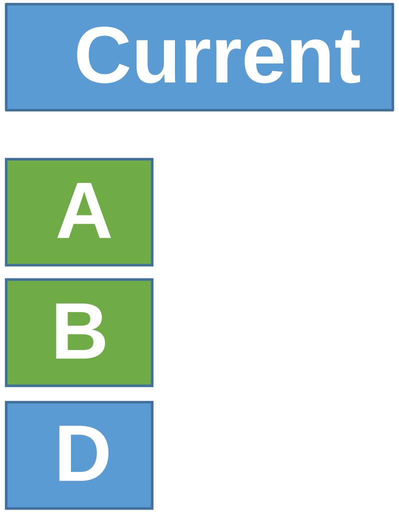
Current

A

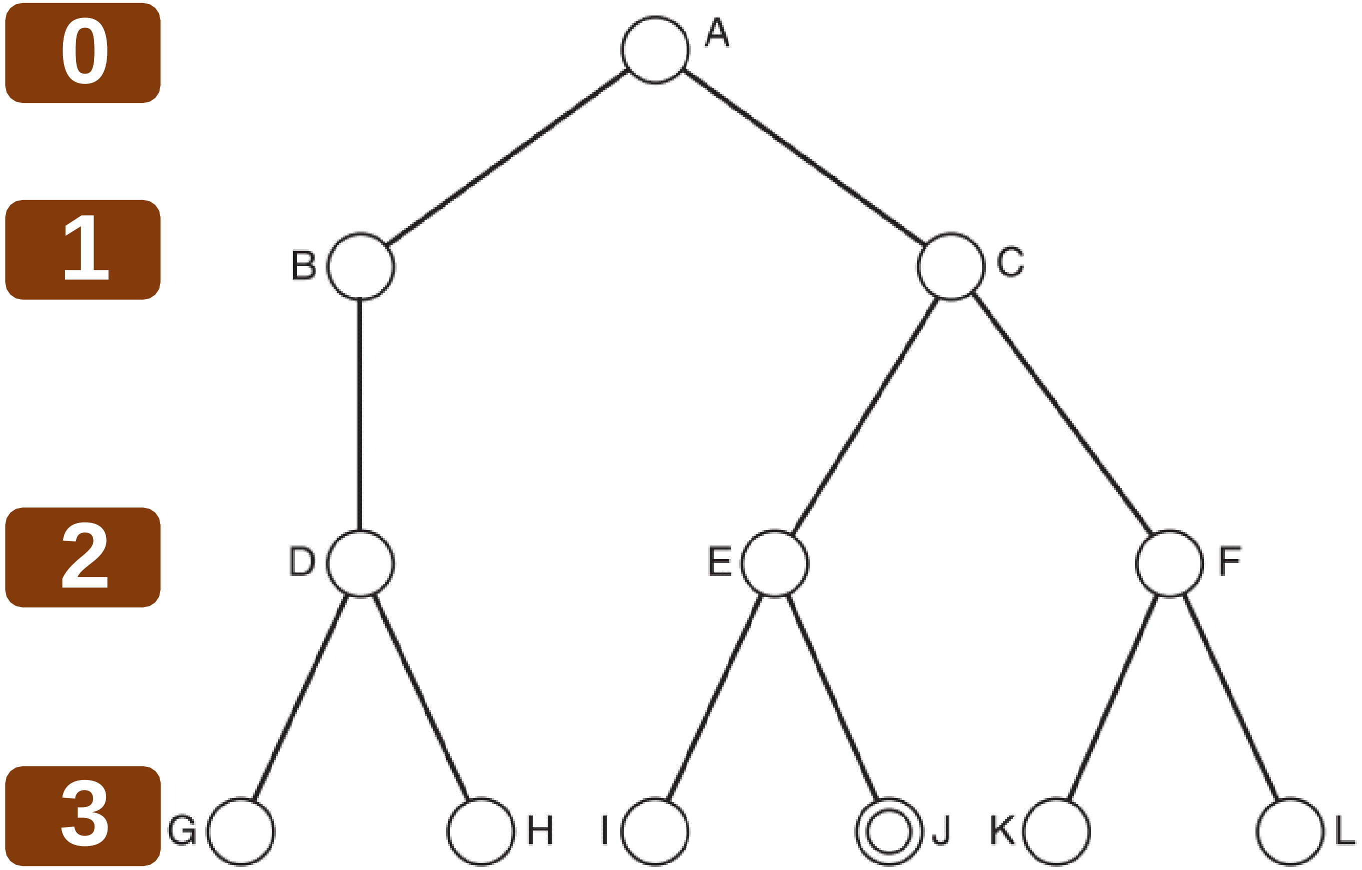
B

Depth-Limited Search
Depth – 2, Goal – Node J



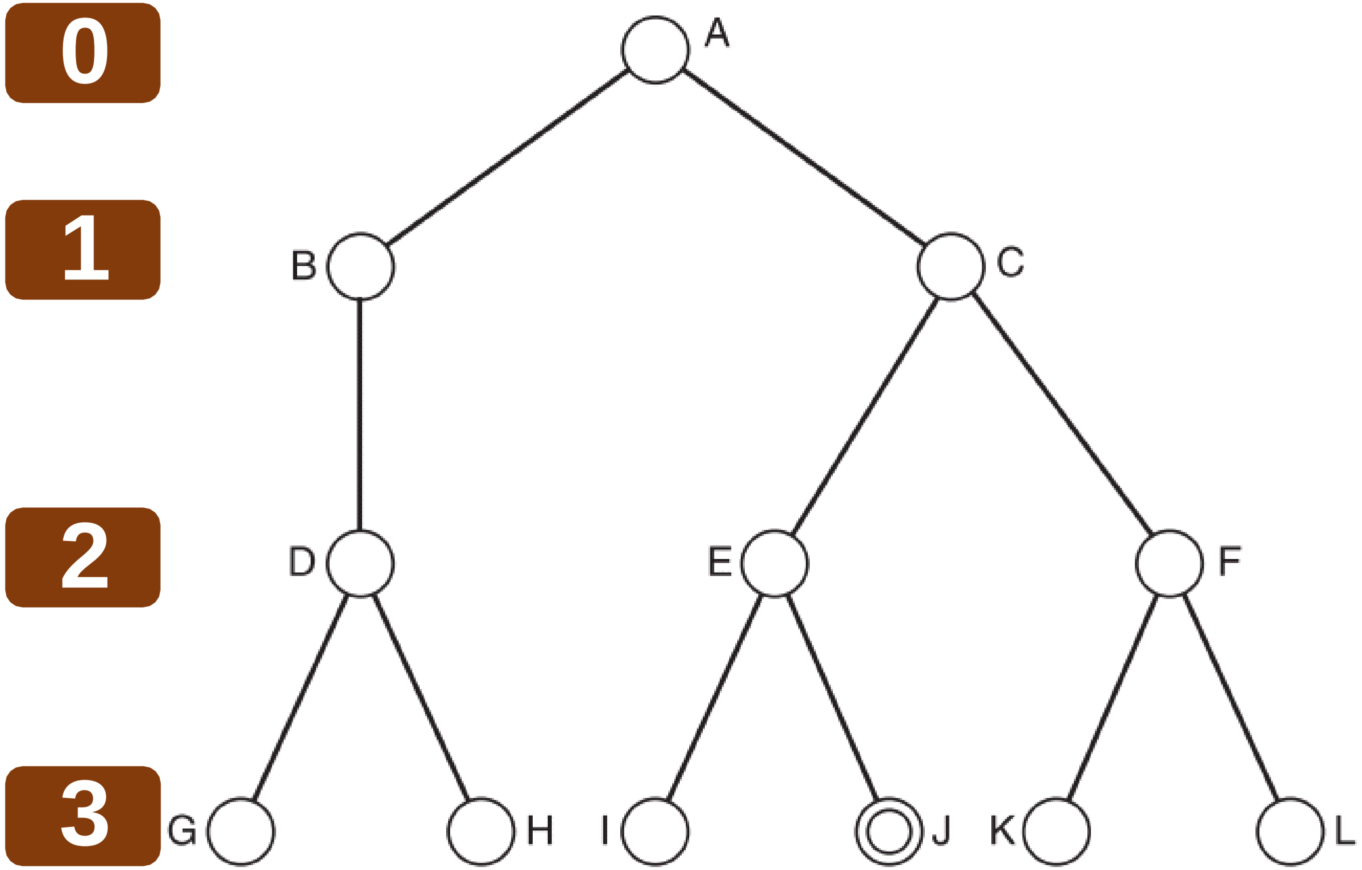


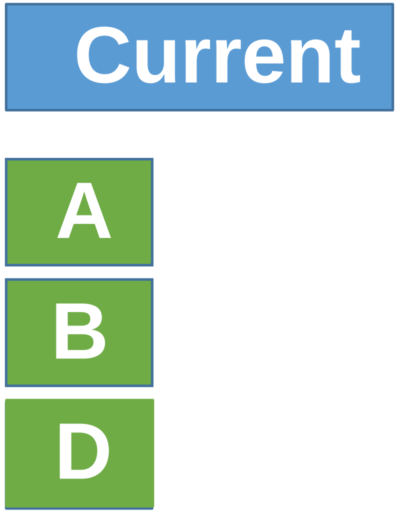
Depth-Limited Search
Depth – 2, Goal – Node J



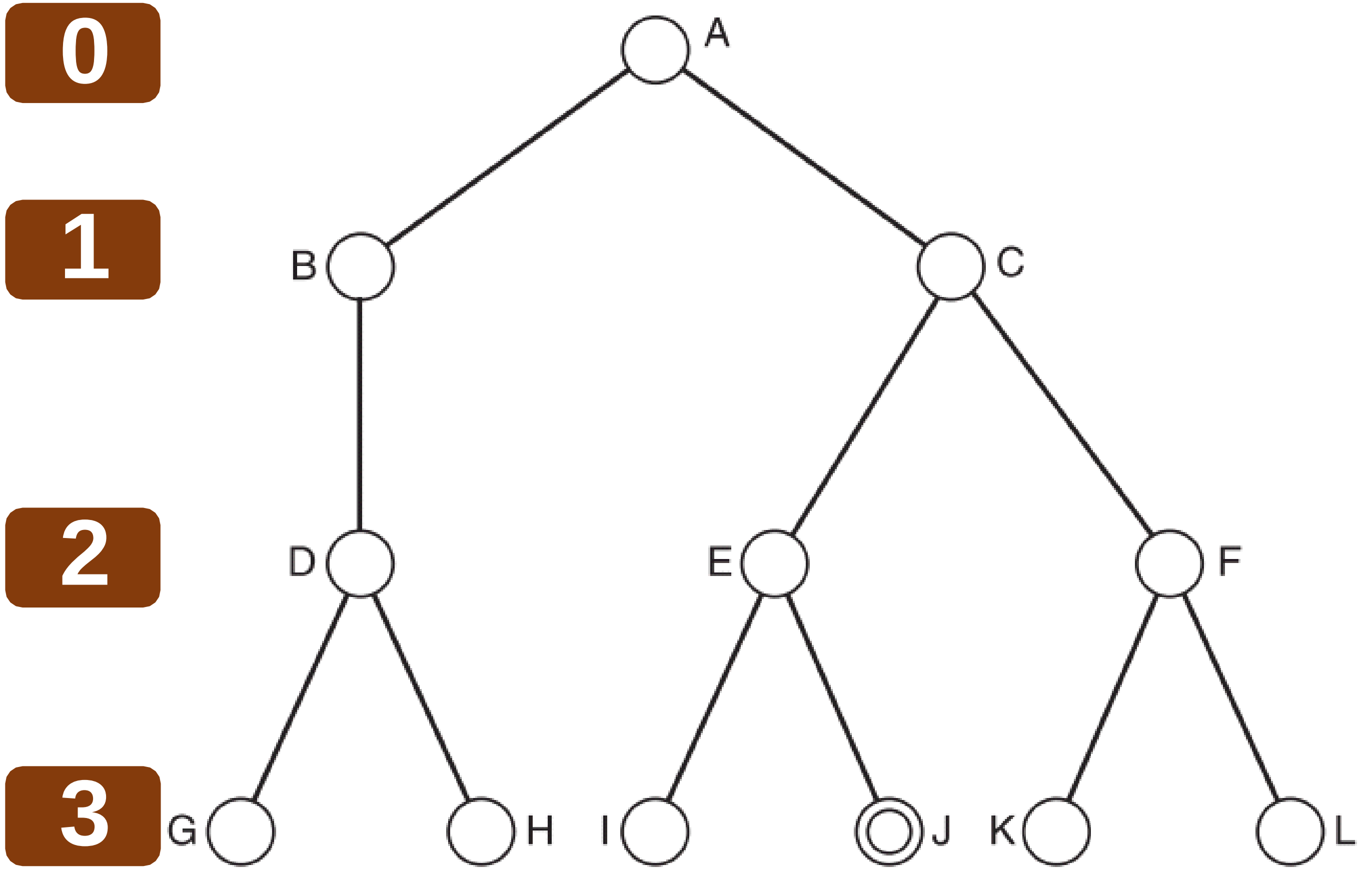
Current
A
B
D

Depth-Limited Search
Depth – 2, Goal – Node J





Depth-Limited Search
Depth – 2, Goal – Node J



Current

A

B

D

B

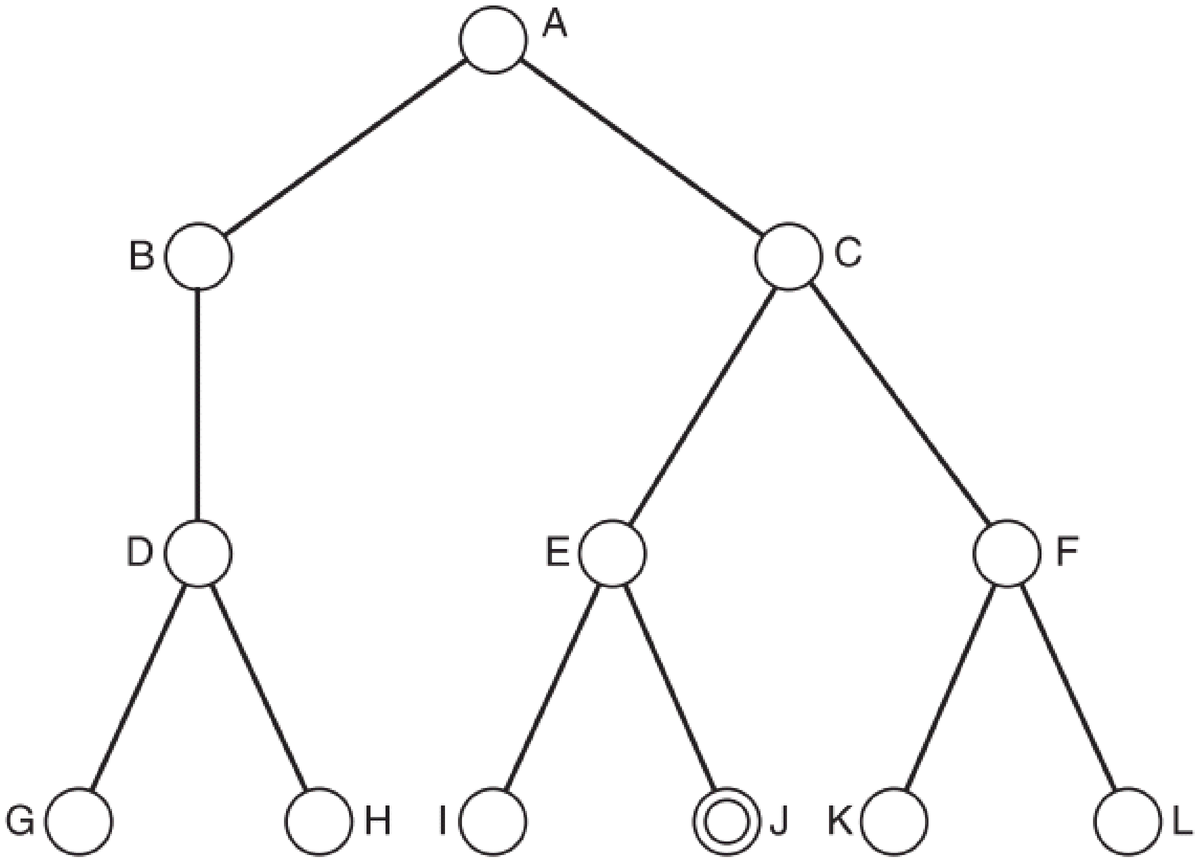
Depth-Limited Search
Depth – 2, Goal – Node J

0

1

2

3



Current

A

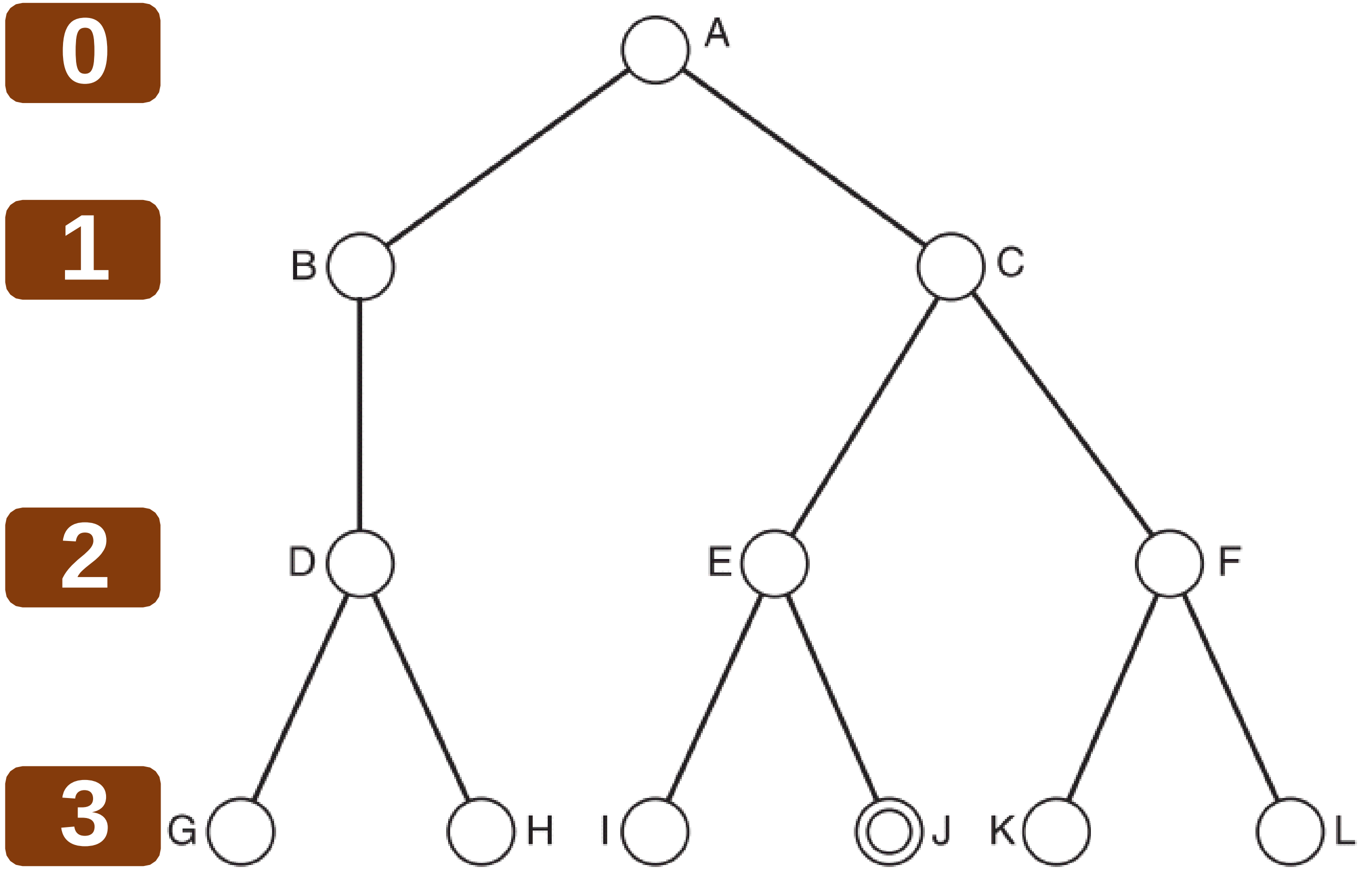
B

D

B

A

Depth-Limited Search
Depth – 2, Goal – Node J

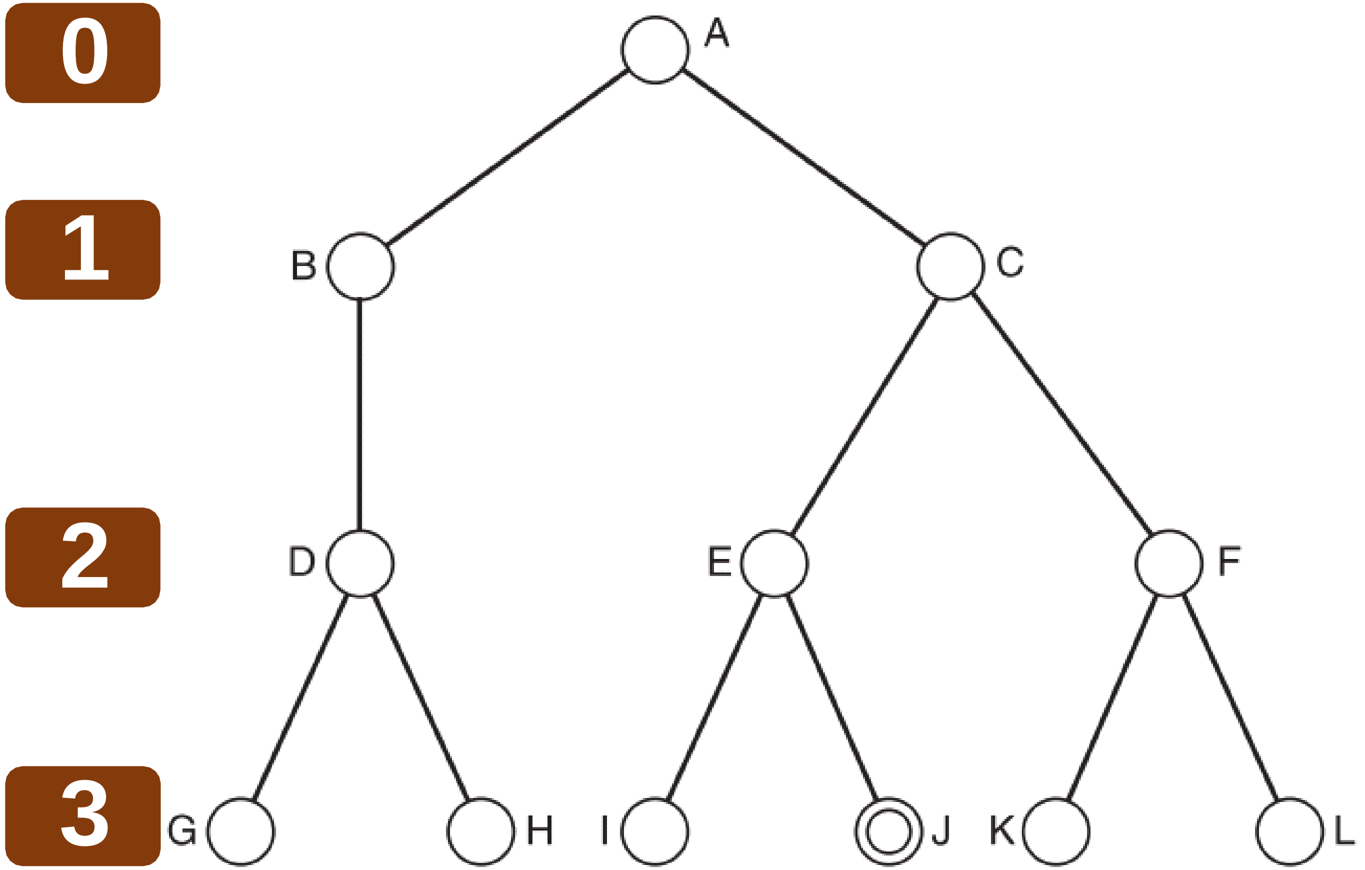


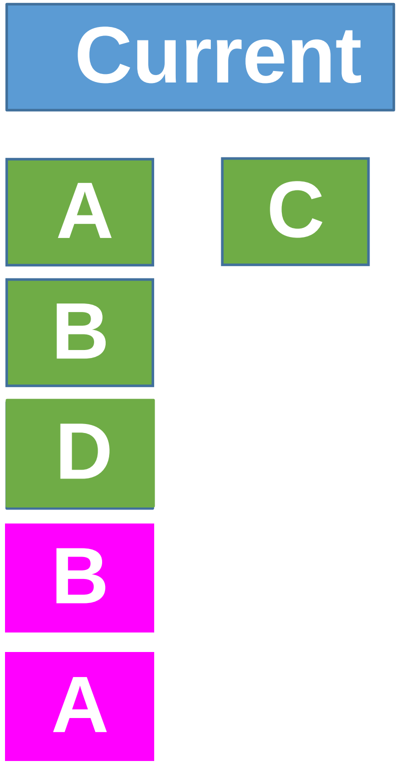
Current

A
B
D
B
A

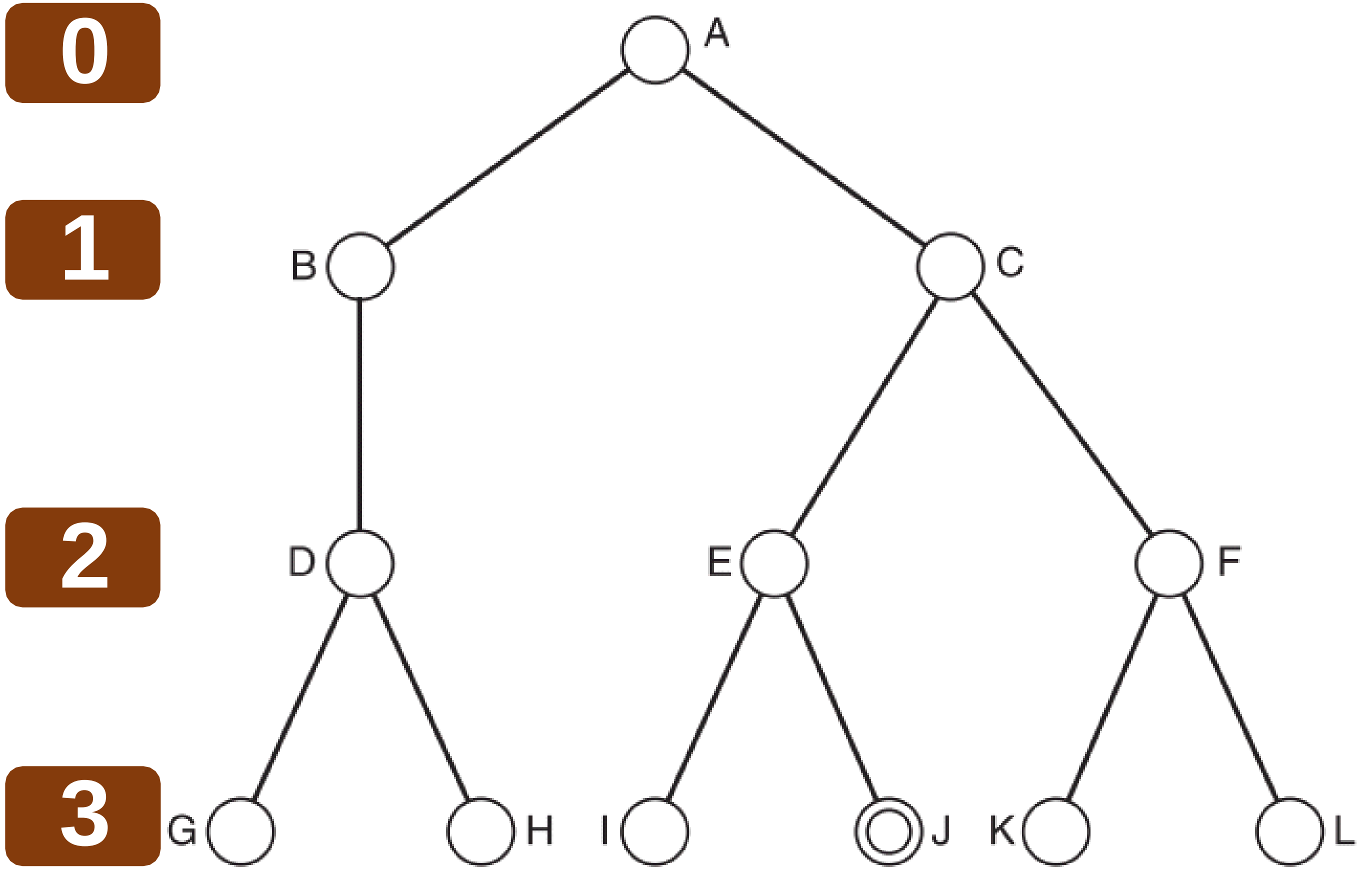
C

Depth-Limited Search
Depth – 2, Goal – Node J





Depth-Limited Search
Depth – 2, Goal – Node J

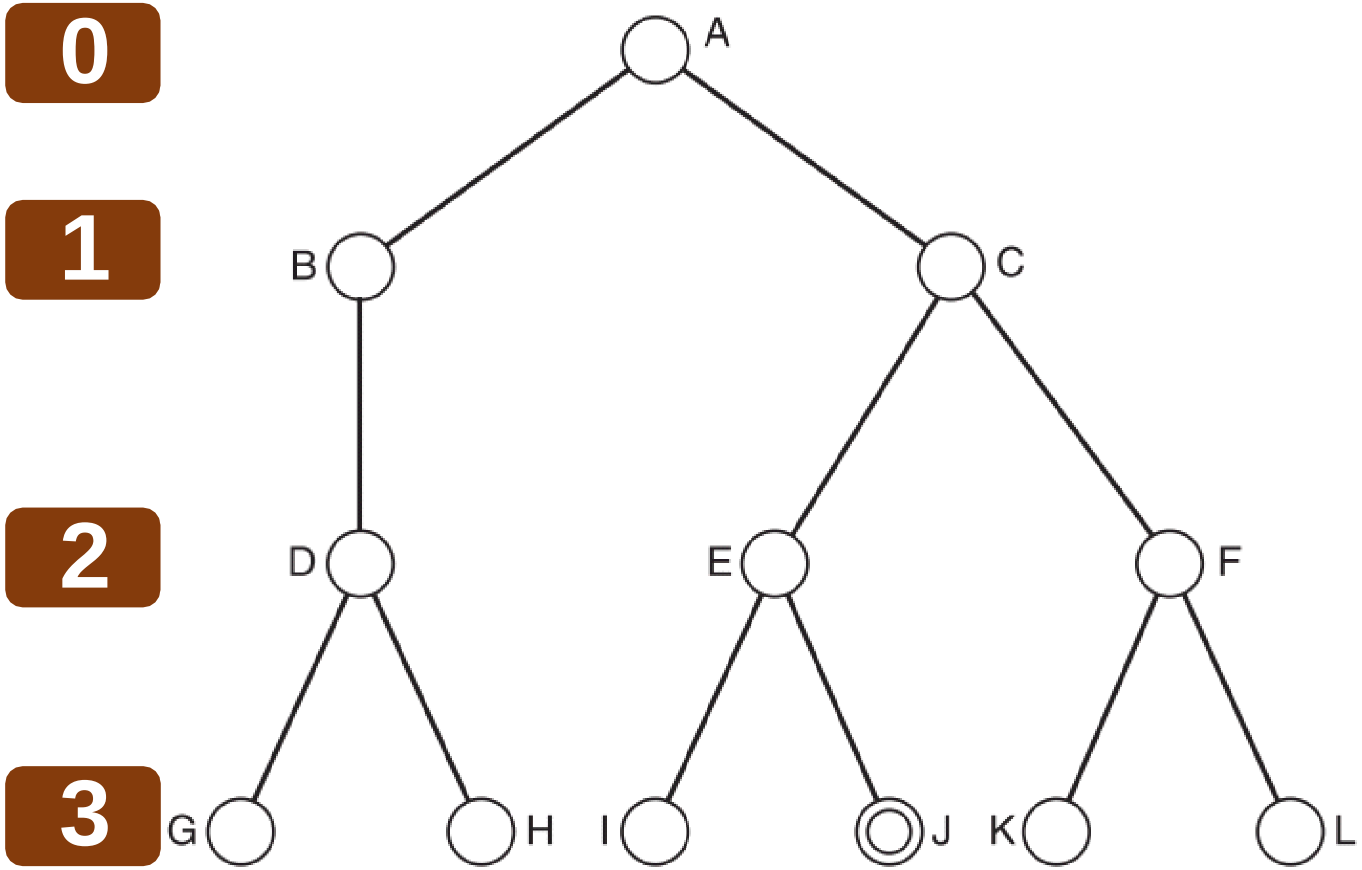


Current

A
B
D
B
A

C
E

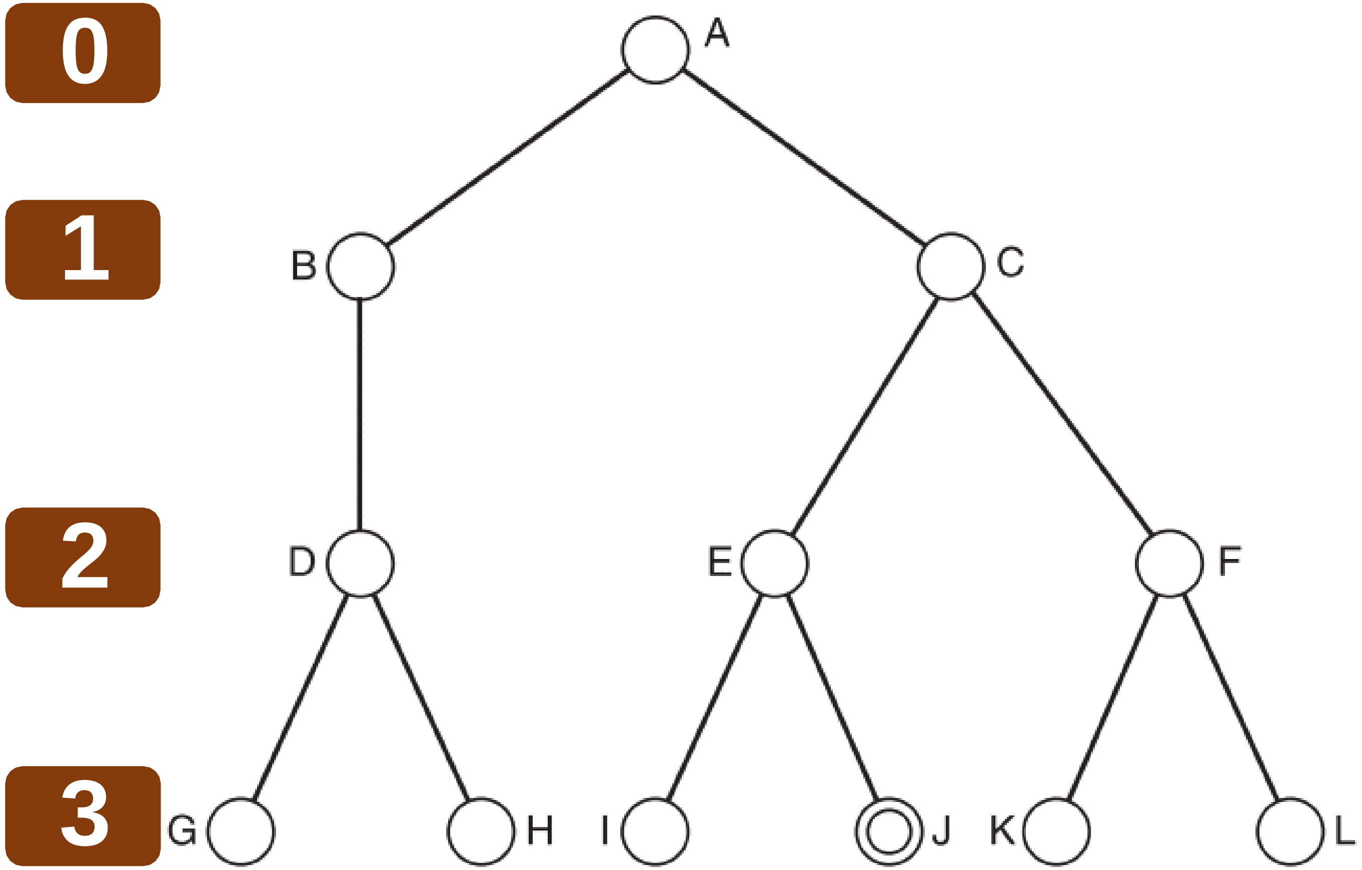
Depth-Limited Search
Depth – 2, Goal – Node J



Current

A	C
B	E
D	
B	
A	

Depth-Limited Search
Depth – 2, Goal – Node J

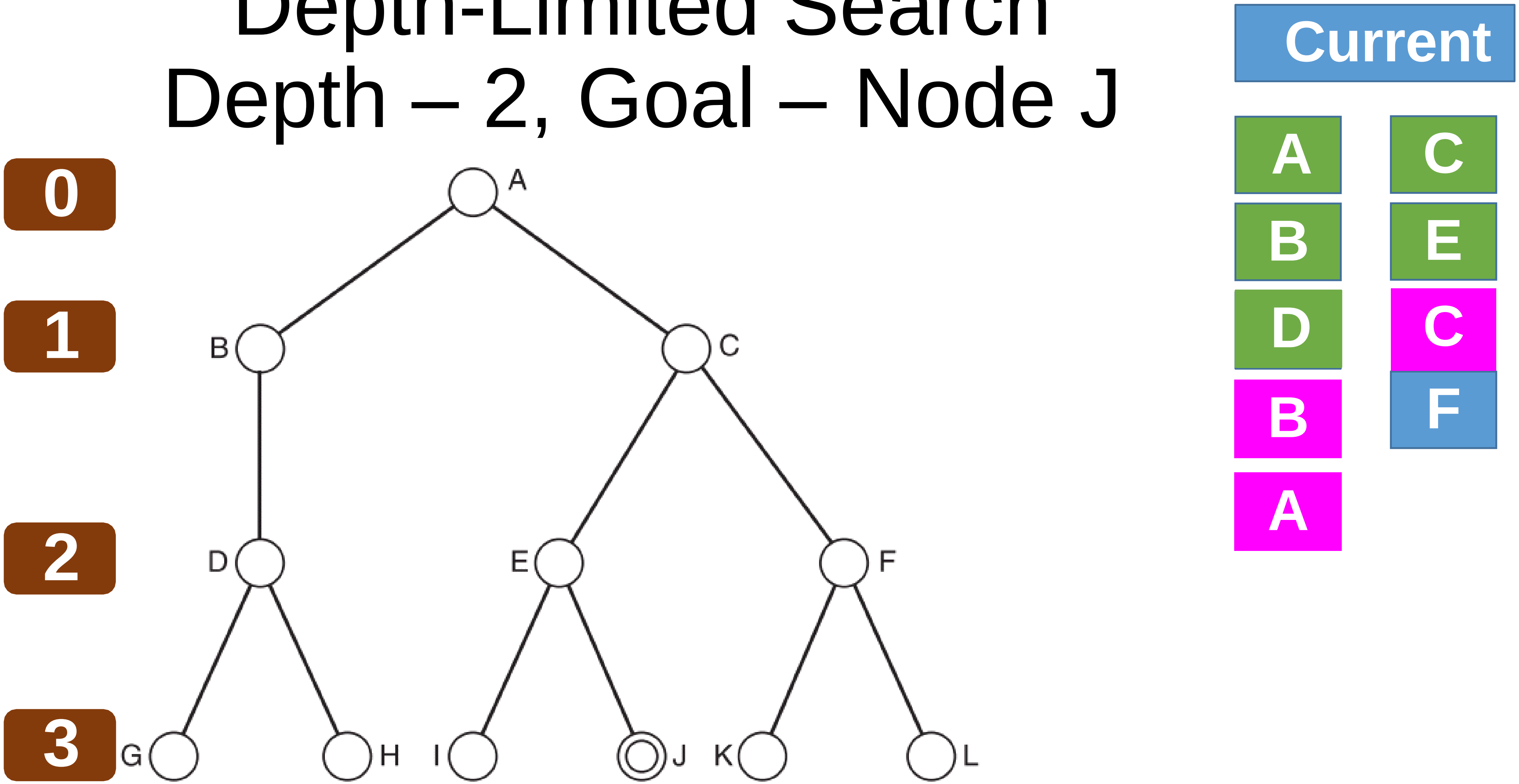


Current

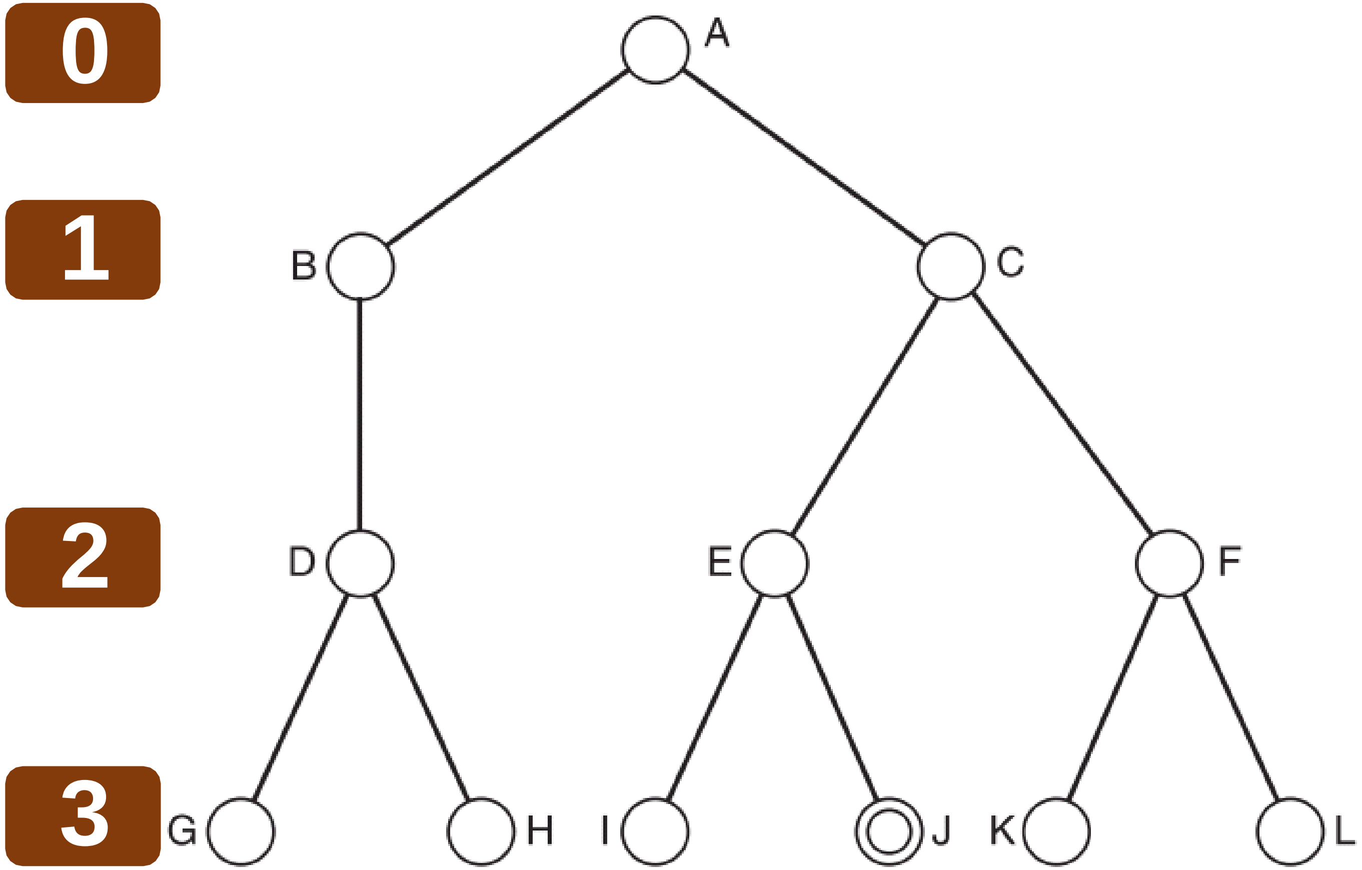
A
B
D
B
A

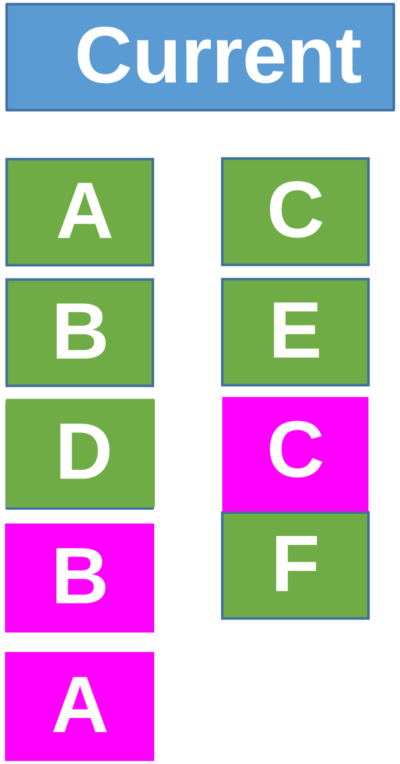
C
E
C

Depth-Limited Search
Depth – 2, Goal – Node J

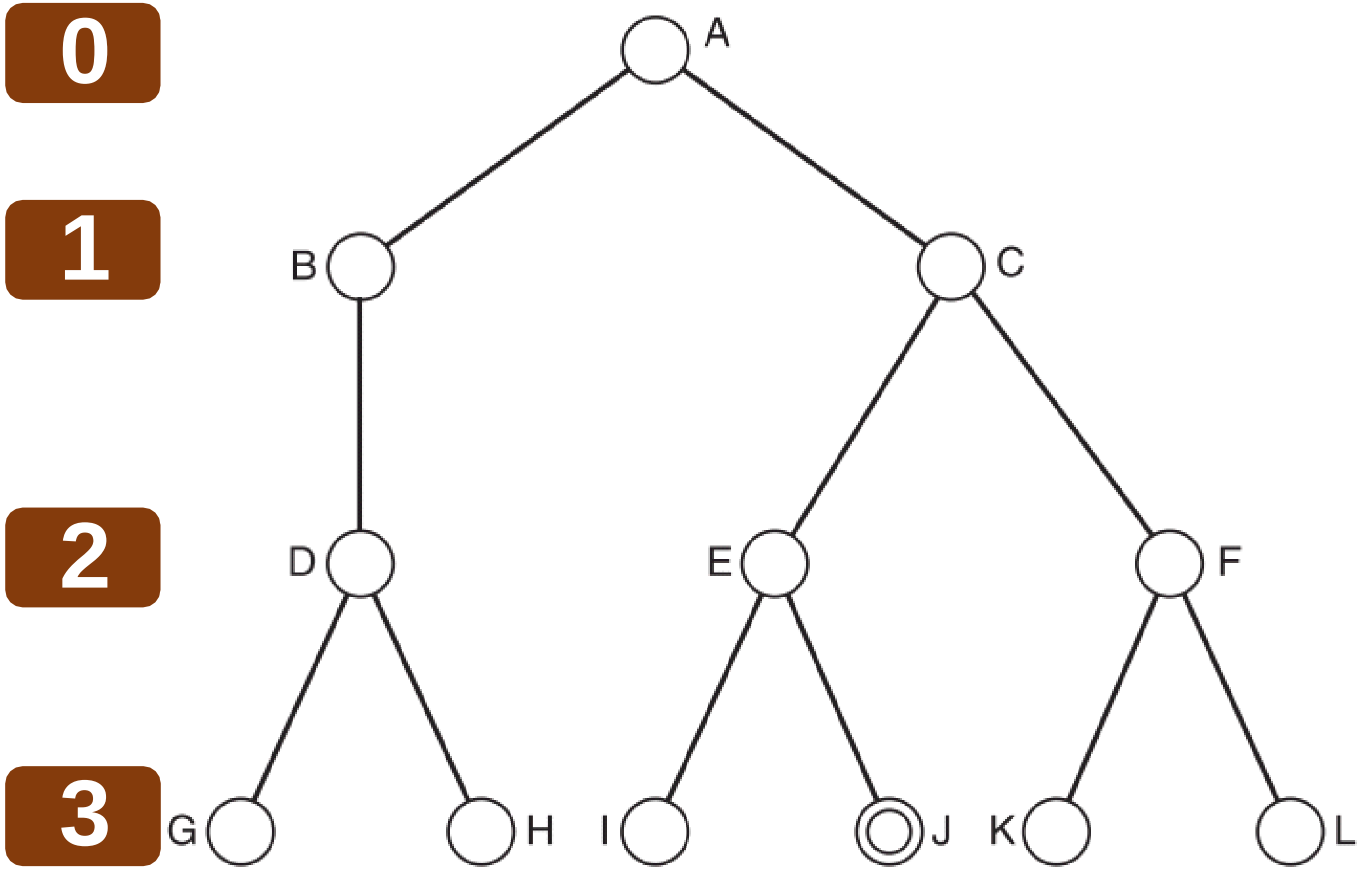


Depth-Limited Search
Depth – 2, Goal – Node J





Depth-Limited Search
Depth – 2, Goal – Node J



Current

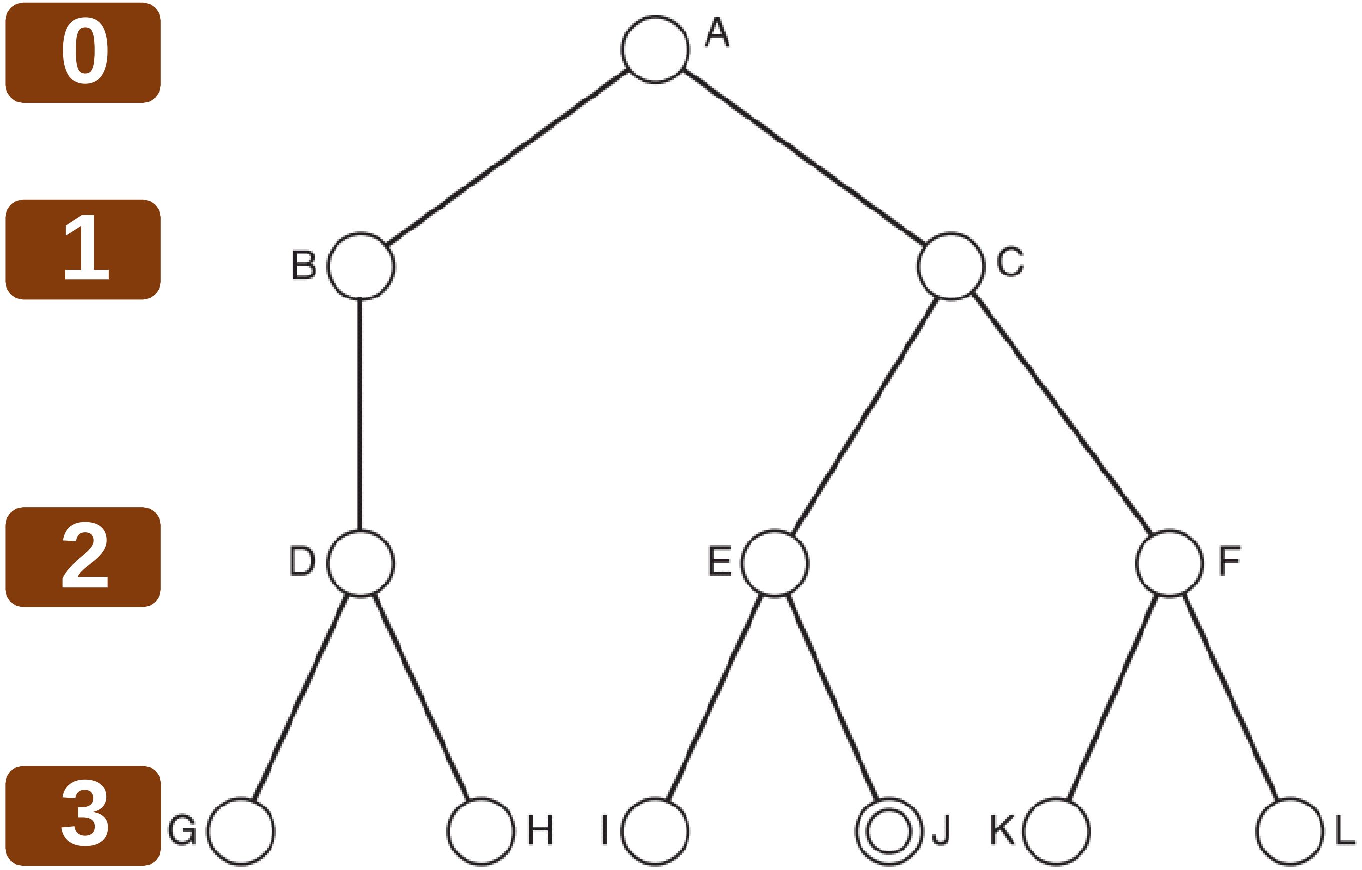
A
B
D
B
A

C
E
C
F

Search Finished

NO GOAL

Depth-Limited Search
Depth – 2, Goal – Node J



Depth is Small

Current

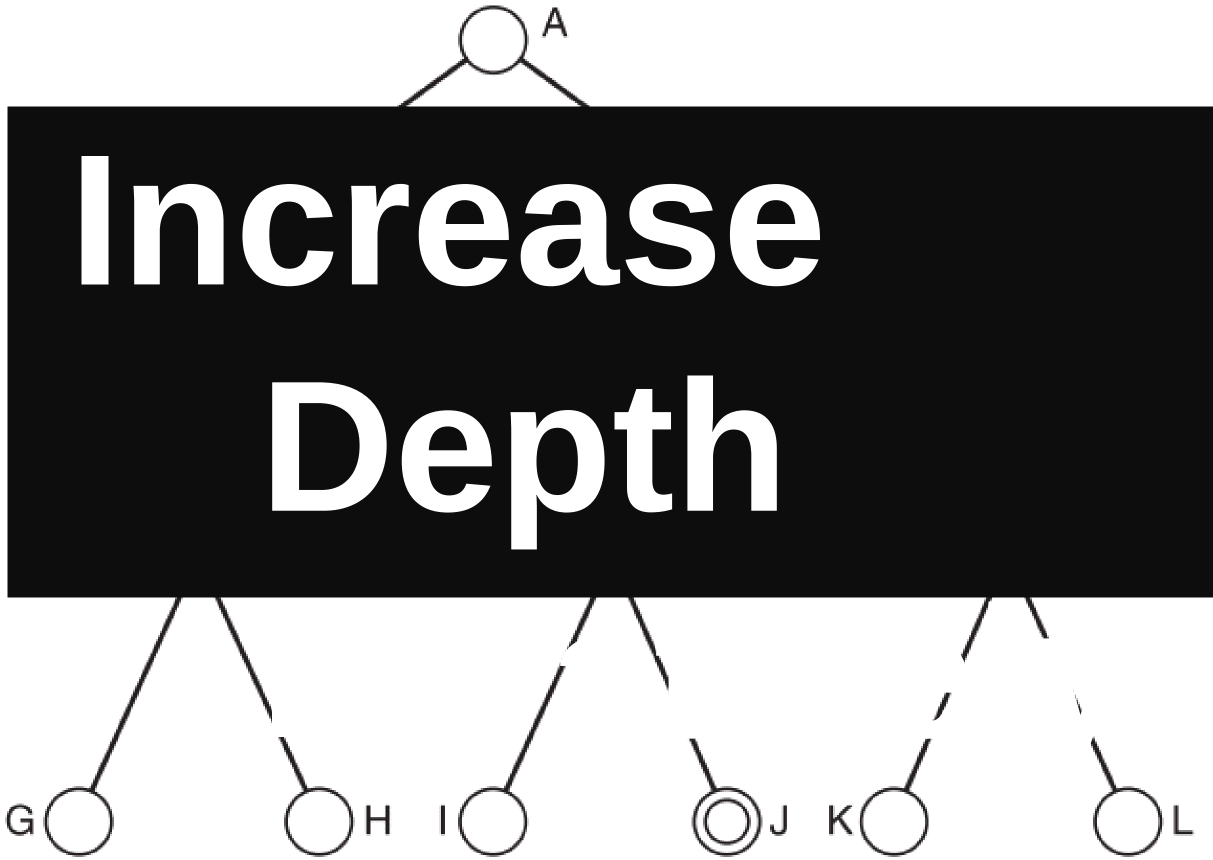
A	C
B	E
D	C
B	F
A	

Search Finished

NO GOAL

Depth-Limited Search
Depth – 2, Goal – Node J

0
1
2
3



Increase Depth

Depth is Small

Current

A
B
D
B
A

C
E
C
F

Search Finished NO GOAL

```
function DEPTH-LIMITED-SEARCH(problem, limit) returns a solution, or failure/cutoff
  return RECURSIVE-DLS(MAKE-NODE(problem.INITIAL-STATE), problem, limit)

function RECURSIVE-DLS(node, problem, limit) returns a solution, or failure/cutoff
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  else if limit = 0 then return cutoff
  else
      cutoff_occurred? ← false
      for each action in problem.ACTIONS(node.STATE) do
          child ← CHILD-NODE(problem, node, action)
          result ← RECURSIVE-DLS(child, problem, limit − 1)
          if result = cutoff then cutoff_occurred? ← true
          else if result ≠ failure then return result
      if cutoff_occurred? then return cutoff else return failure
```

Figure 3.17 A recursive implementation of depth-limited tree search.

Analysing Depth-Limit Search

- Not Optimal
- Not Complete
- Time Complexity- $O(b^l)$
- Space Complexity-$O(bl)$

ITERATIVE DEEPENING SEARCH

Iterative Deepening Search

- It's a Depth First Search, but it does it one level at a time, gradually increasing the limit, until a goal is found.
- Combine the benefits of depth-first and breadth-first search
- Like DFS, modest memory requirements
- Like BFS, it is complete when branching factor is finite, and optimal when the path cost is a non decreasing function of the dept of the node.

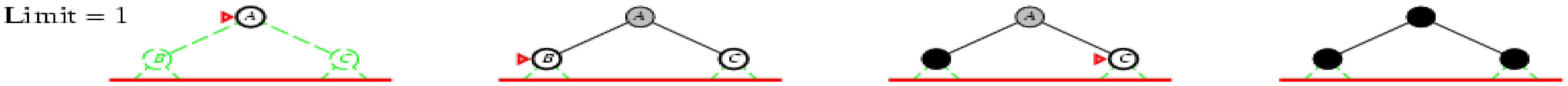
Iterative Deepening Search

- May seem wasteful because states are generated multiple times
- But actually not very costly, because nodes at the bottom level are generated only once.
- In practice, however, the overhead of these multiple expansions is small, because most of the nodes are towards leaves (bottom) of the search tree:
  - *Thus, the nodes that are evaluated several times (towards top of tree) are in relatively small number.*
- **Iterative depending is the preferred uninformed search method when the search space is large and the depth of the solution is unknown**

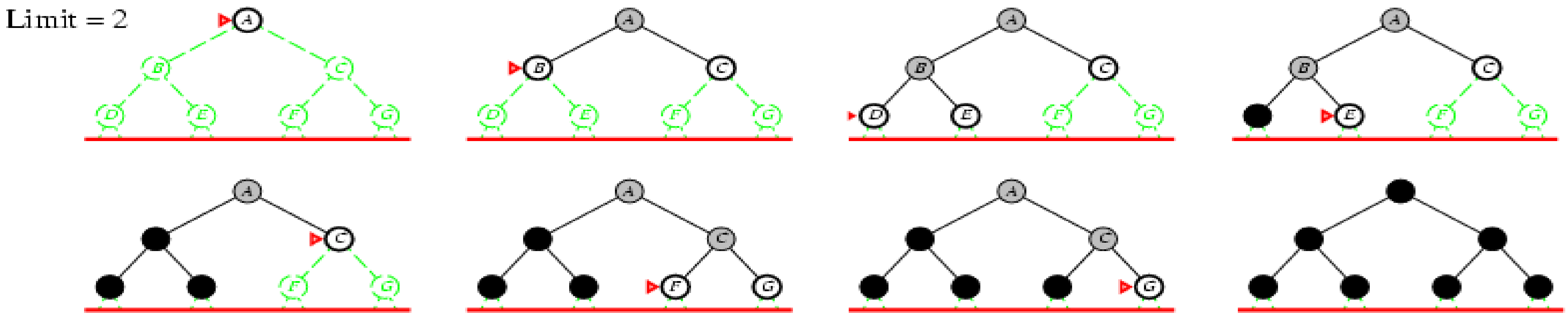
Iterative Deepening Search with l=0

Limit = 0

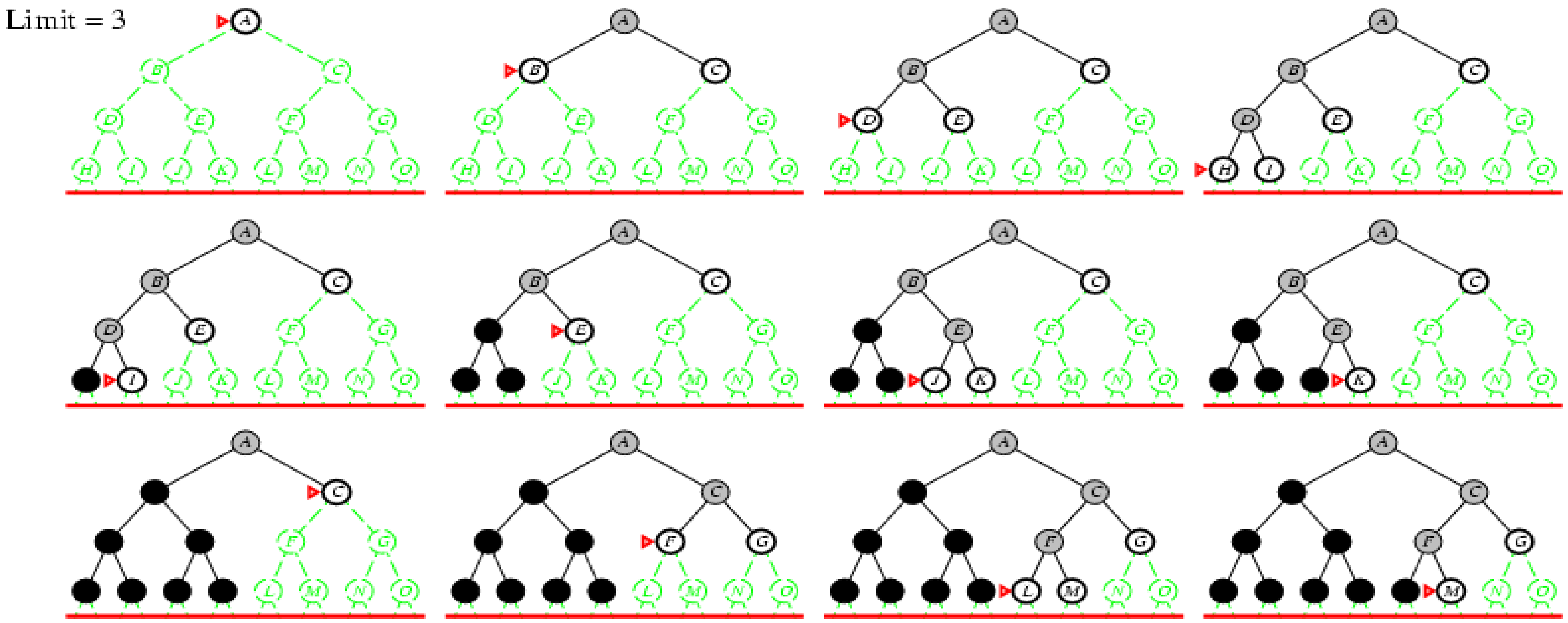
Iterative Deepening Search with l=1



Iterative Deepening Search with l=2

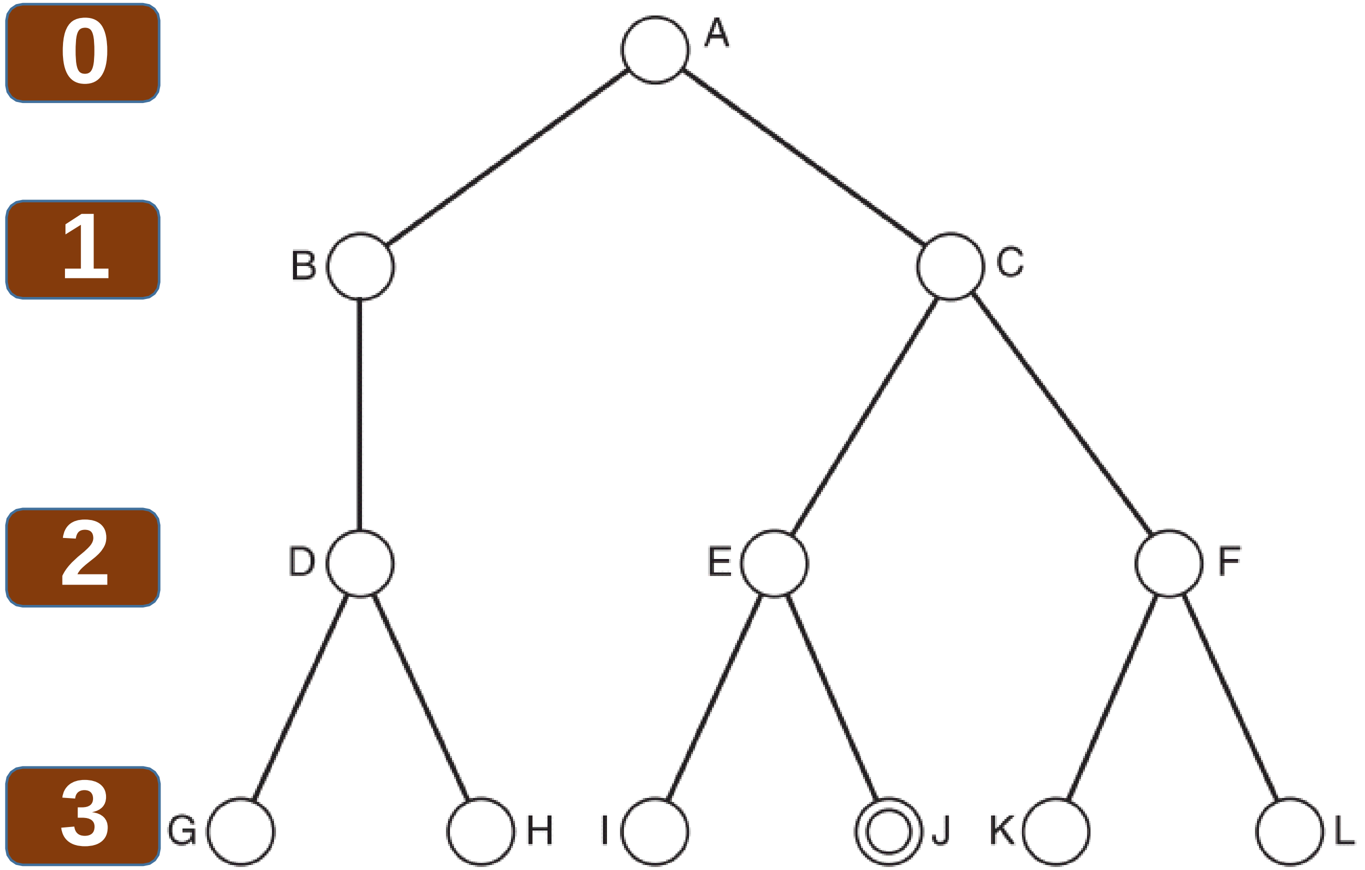


Iterative Deepening Search with l=3



Iterative Deepening Search
Depth – 0, Goal – Node J

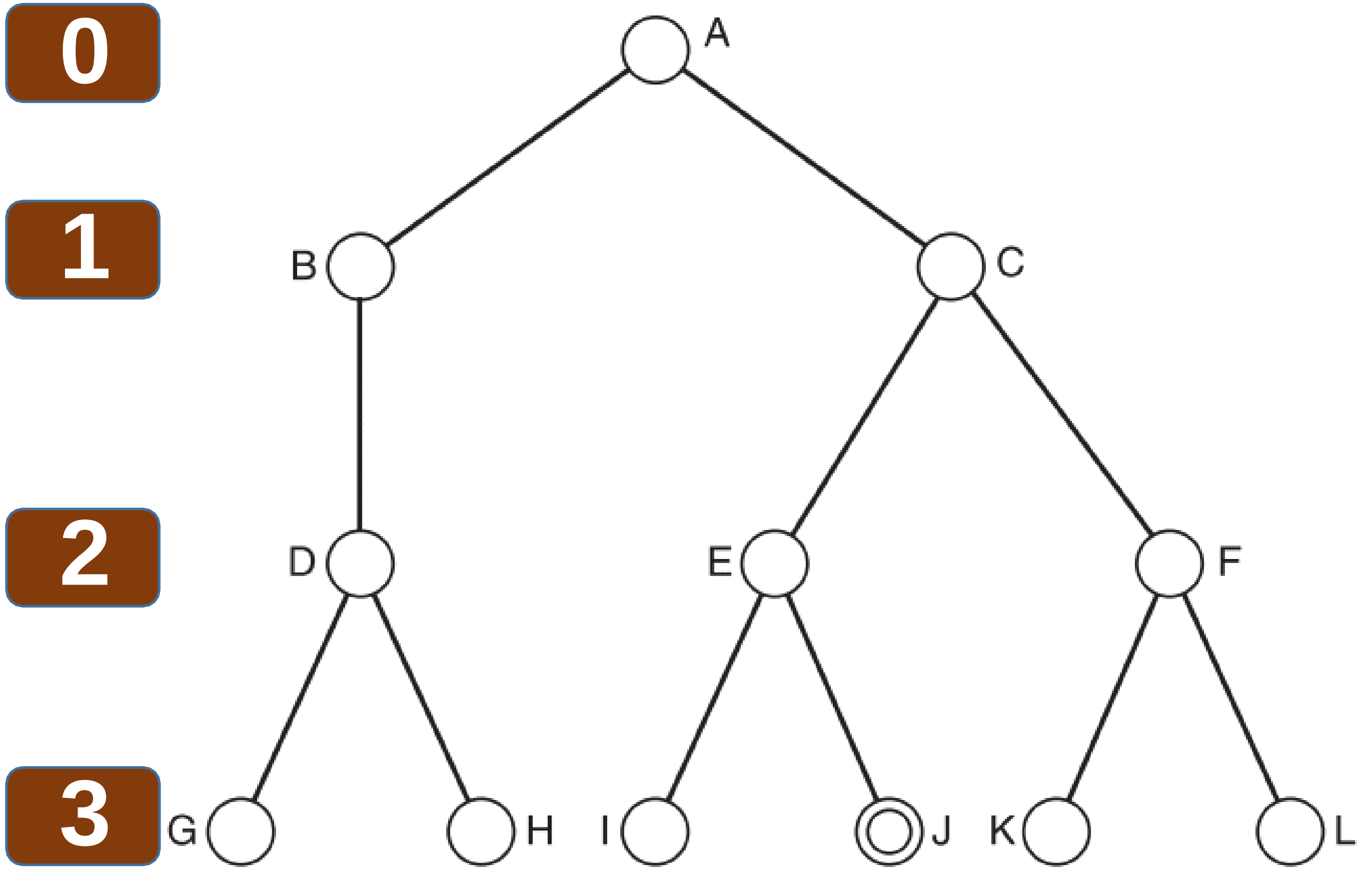
Current



Iterative Deepening Search
Depth – 0, Goal – Node J

Current

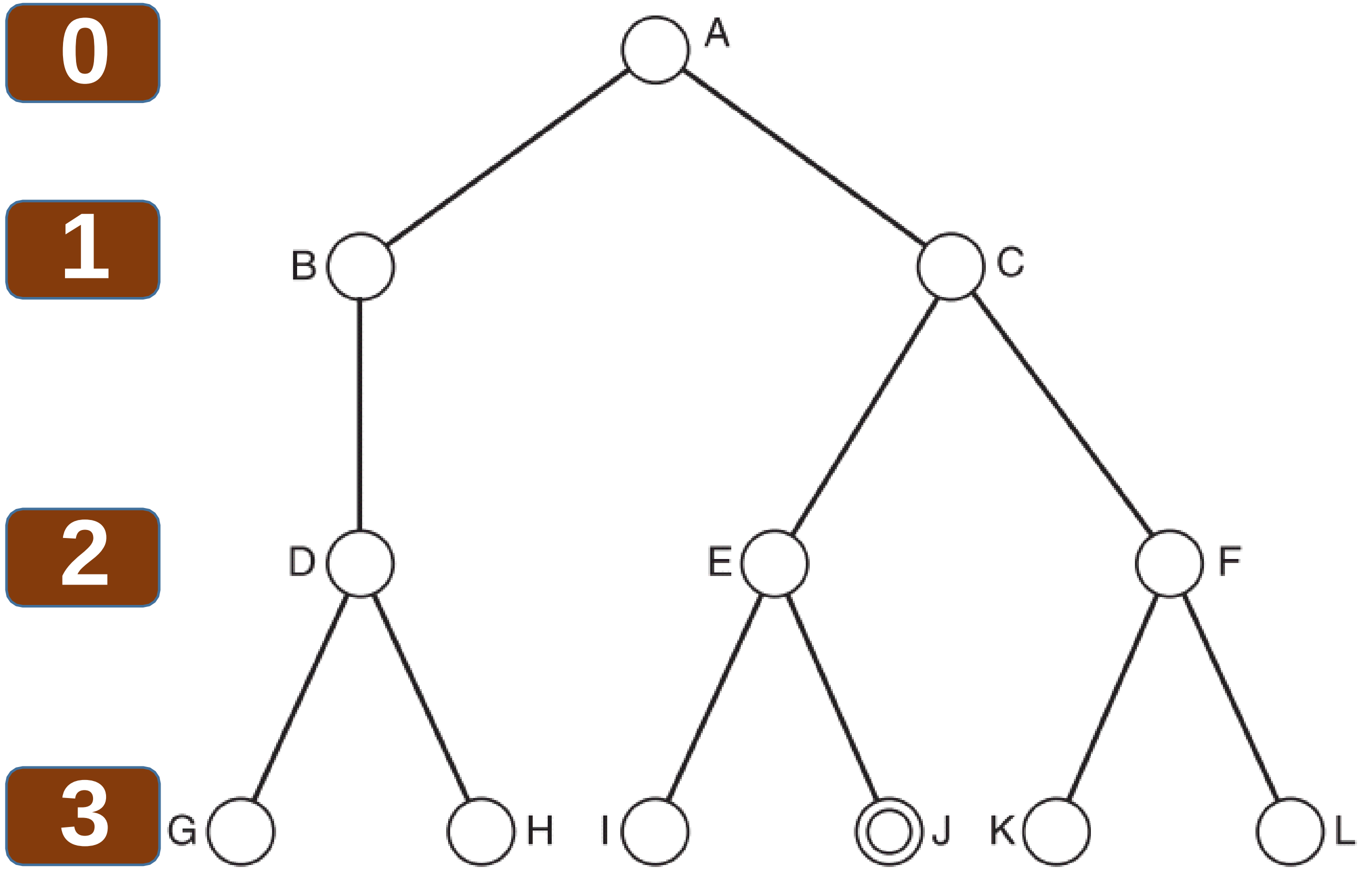
A



Iterative Deepening Search
Depth – 0, Goal – Node J

Current

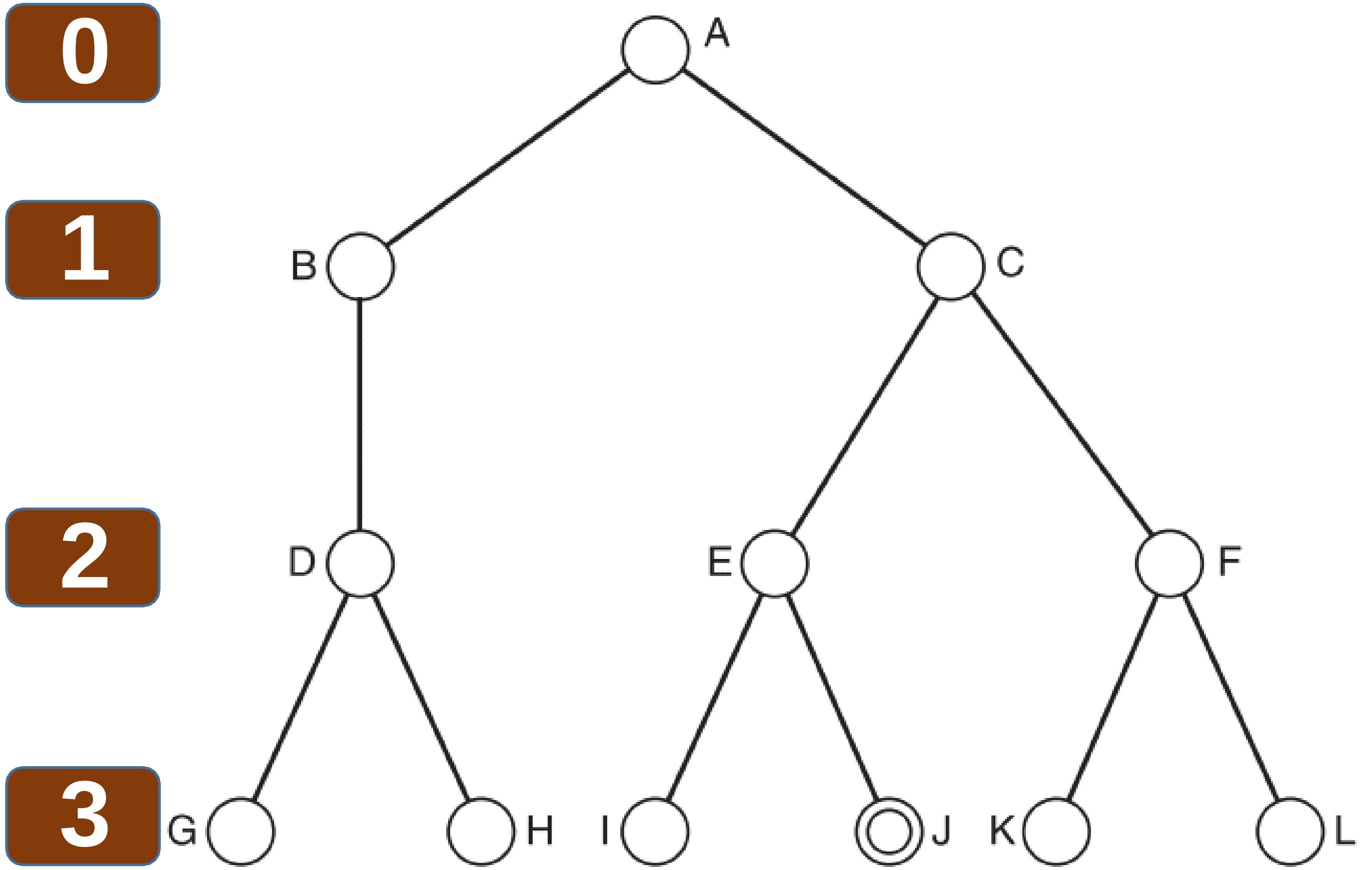
A



Iterative Deepening Search
Depth – 0, Goal – Node J

Current

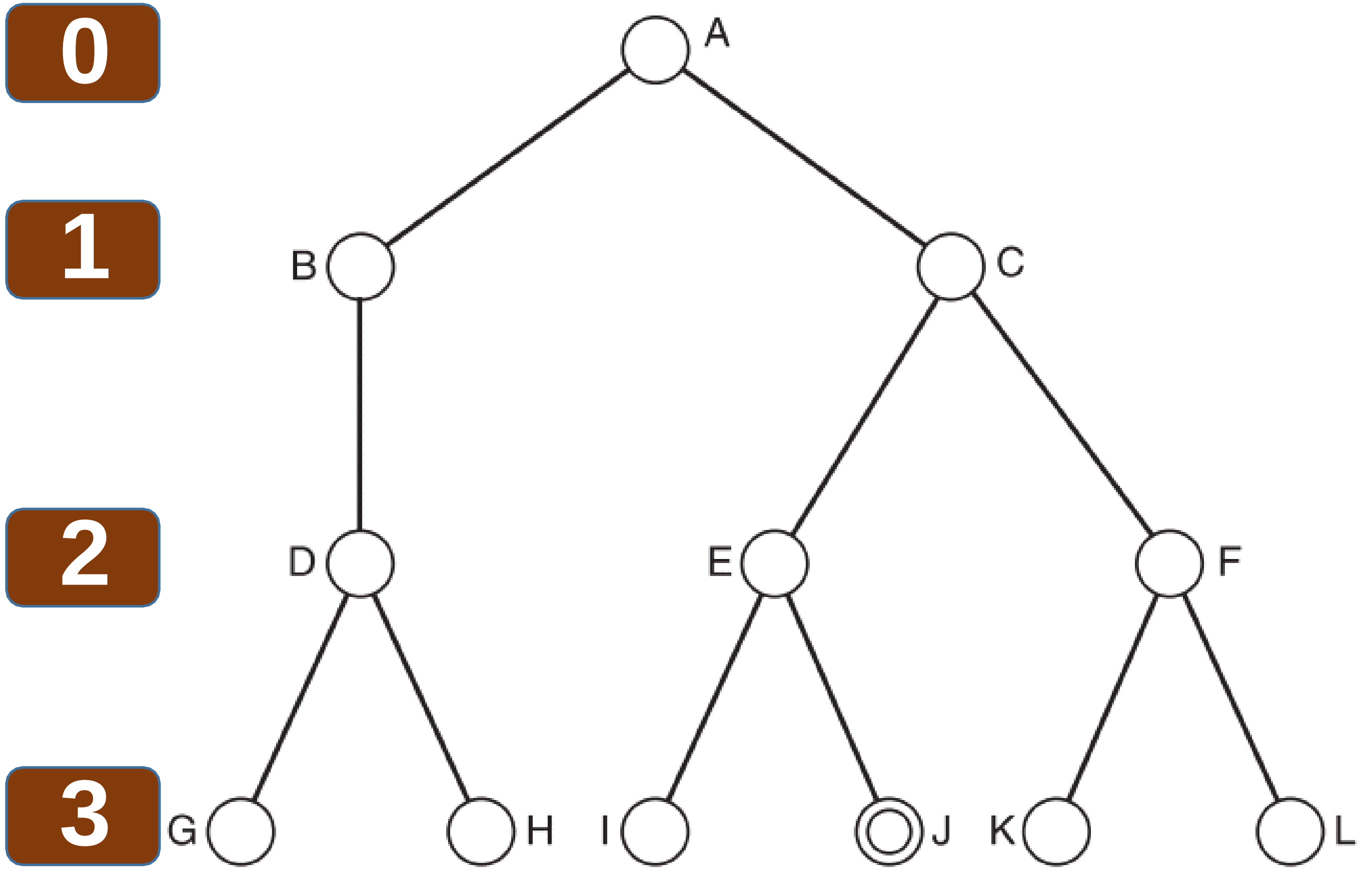
A



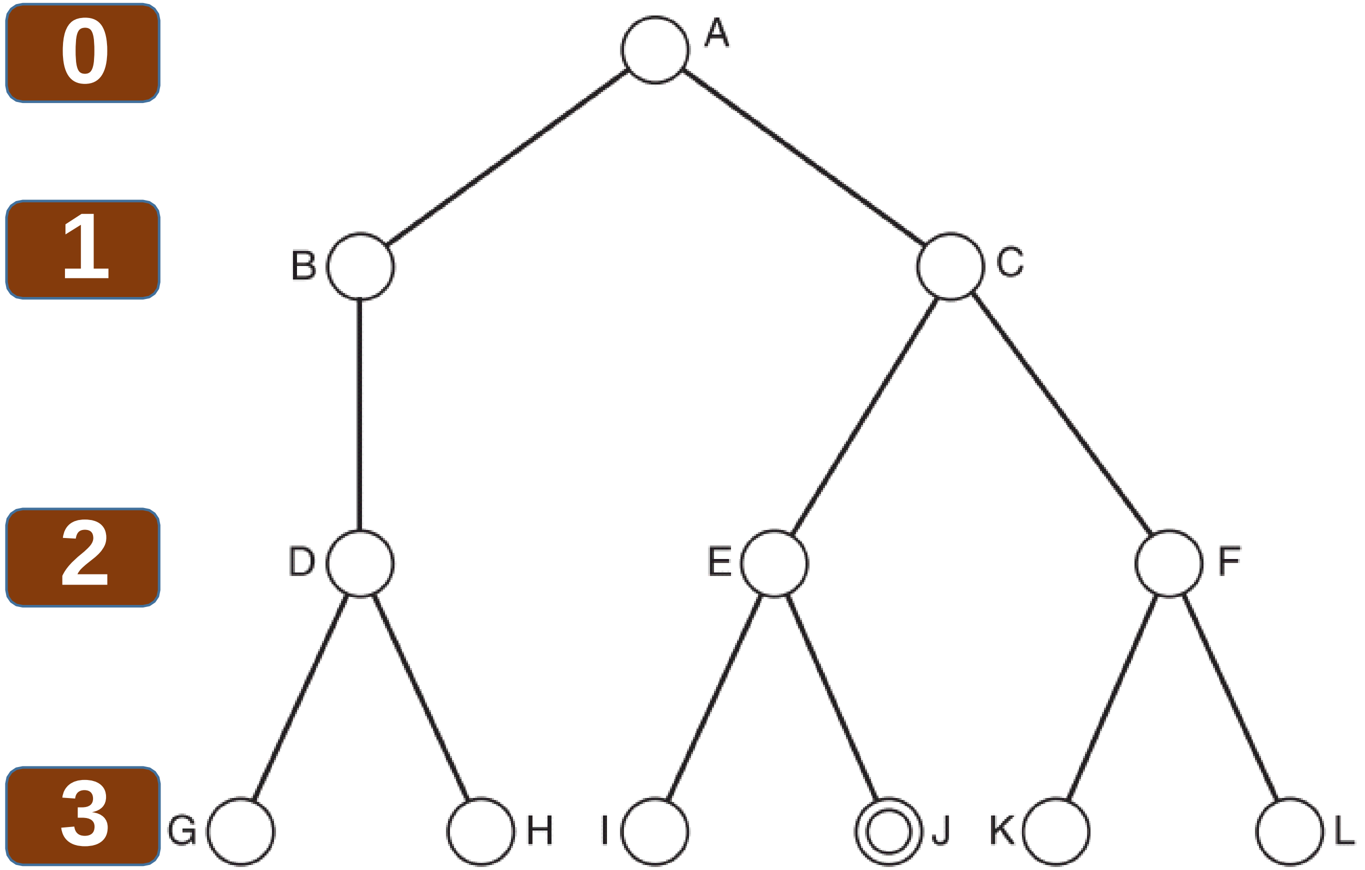
Search Finished NO GOAL
Increase Depth by 1

Iterative Deepening Search
Depth – 1, Goal – Node J

Current



Iterative Deepening Search
Depth – 1, Goal – Node J



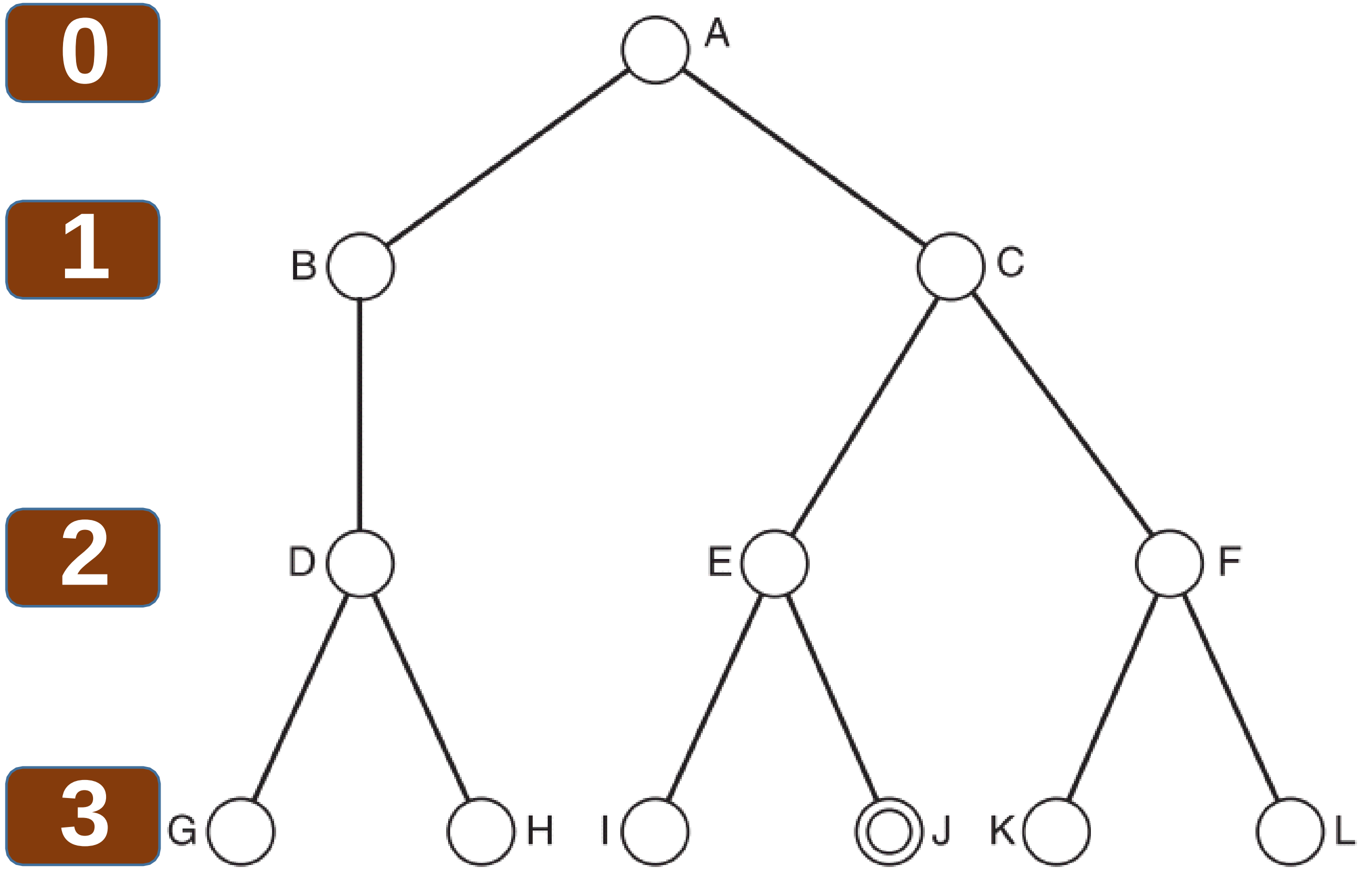
Current

A

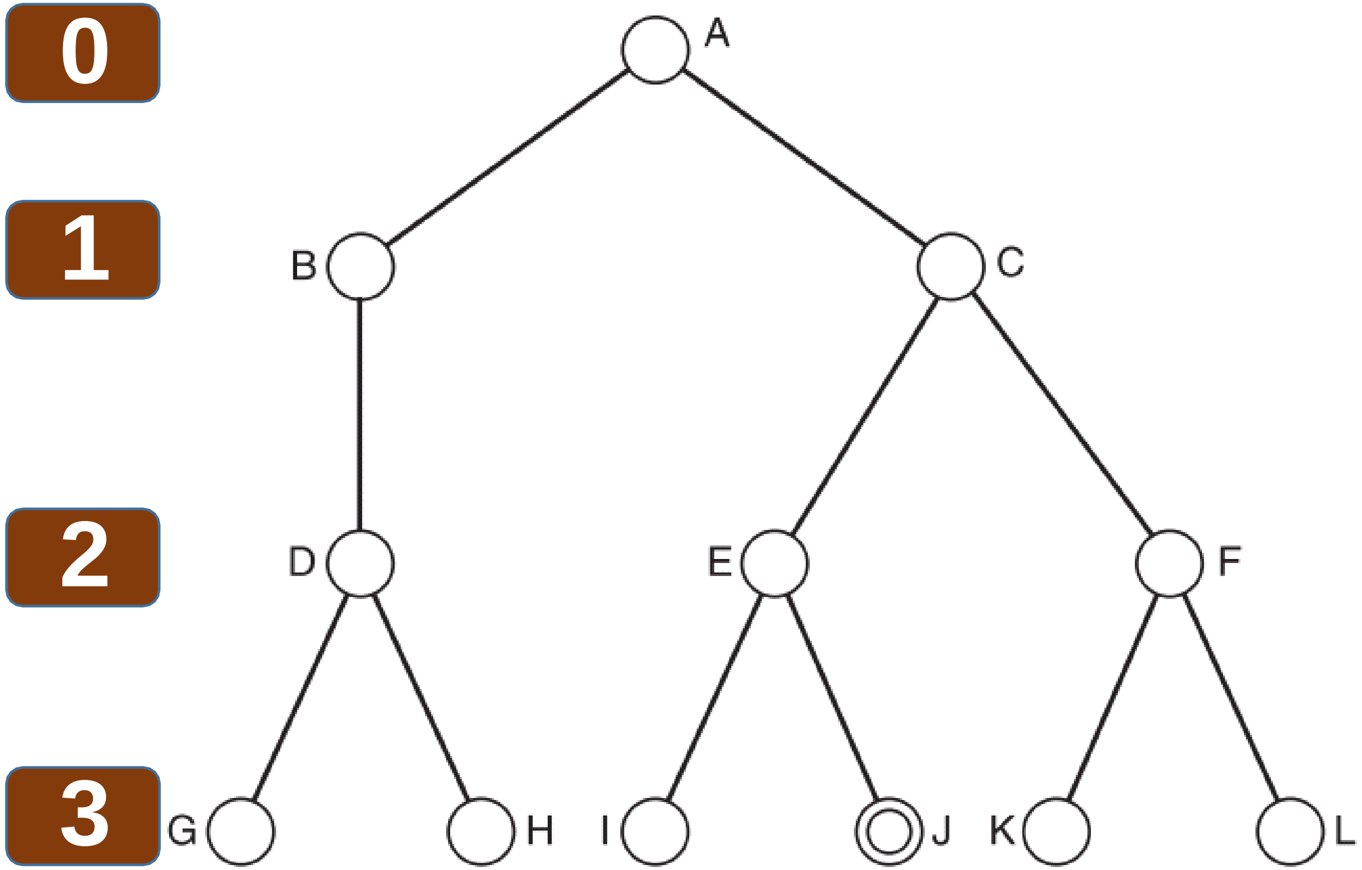
Iterative Deepening Search
Depth – 1, Goal – Node J

Current

A



Iterative Deepening Search
Depth – 1, Goal – Node J

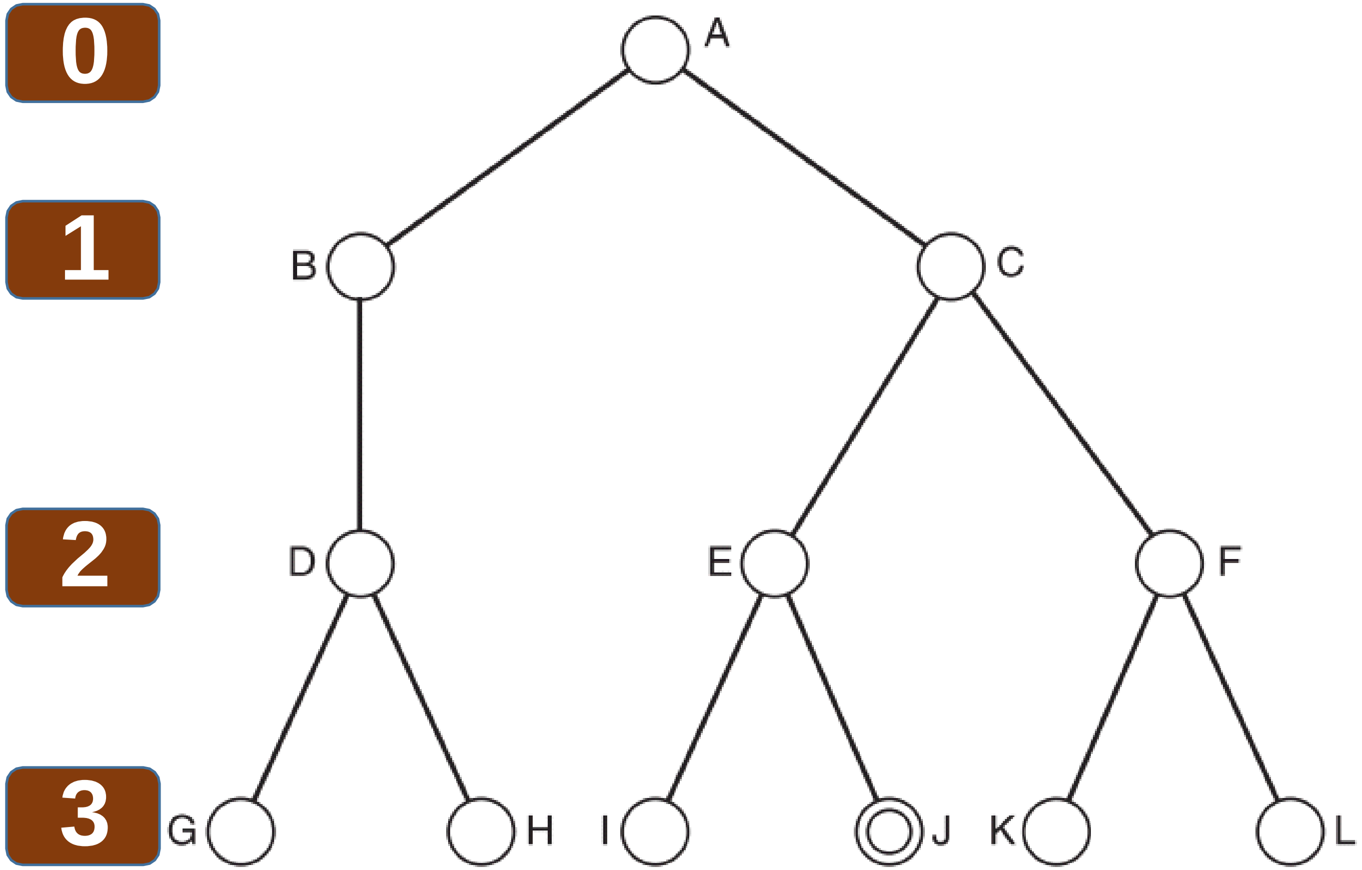


Current

A

B

Iterative Deepening Search
Depth – 1, Goal – Node J

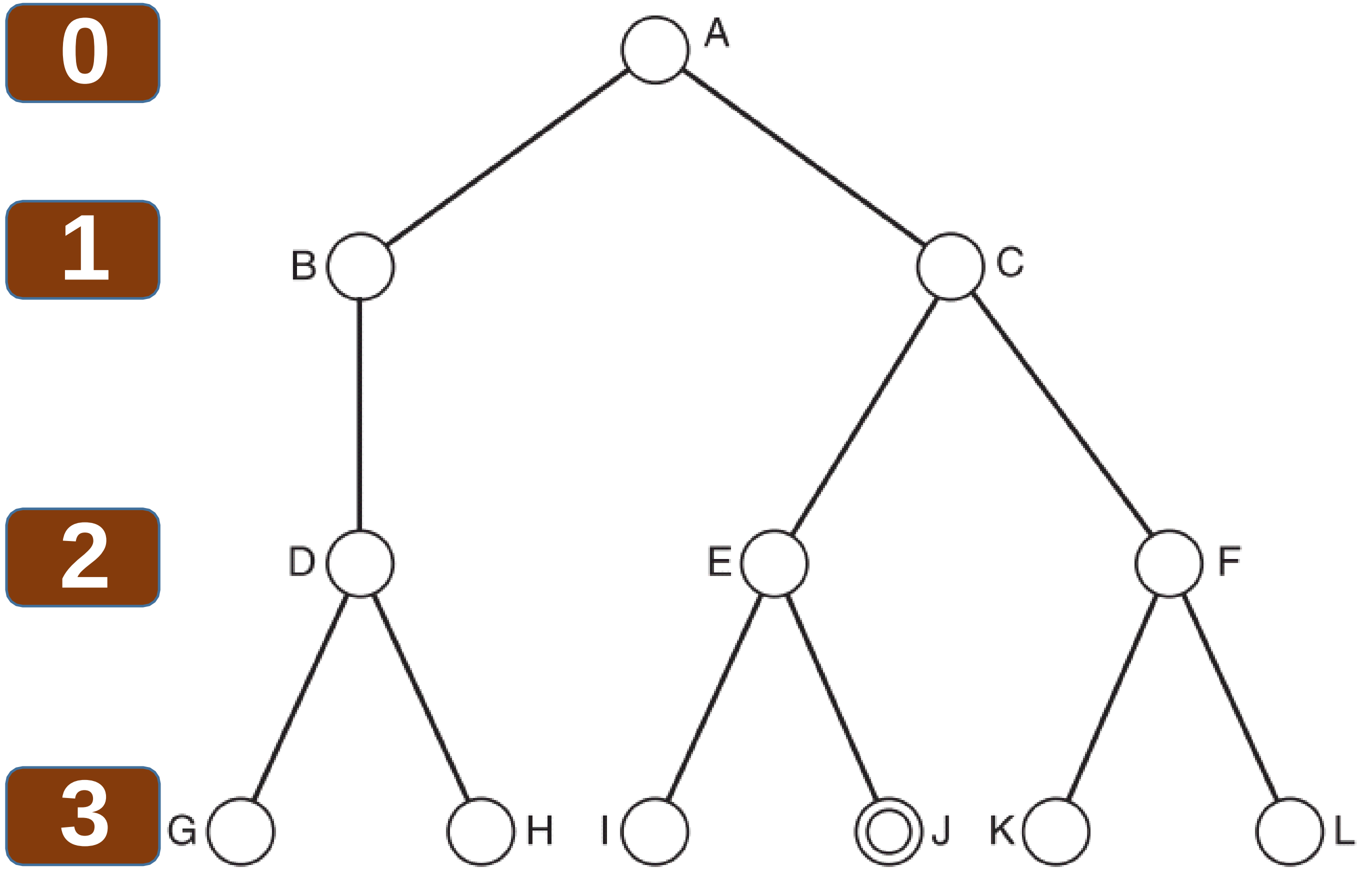


Current

A

B

Iterative Deepening Search
Depth – 1, Goal – Node J



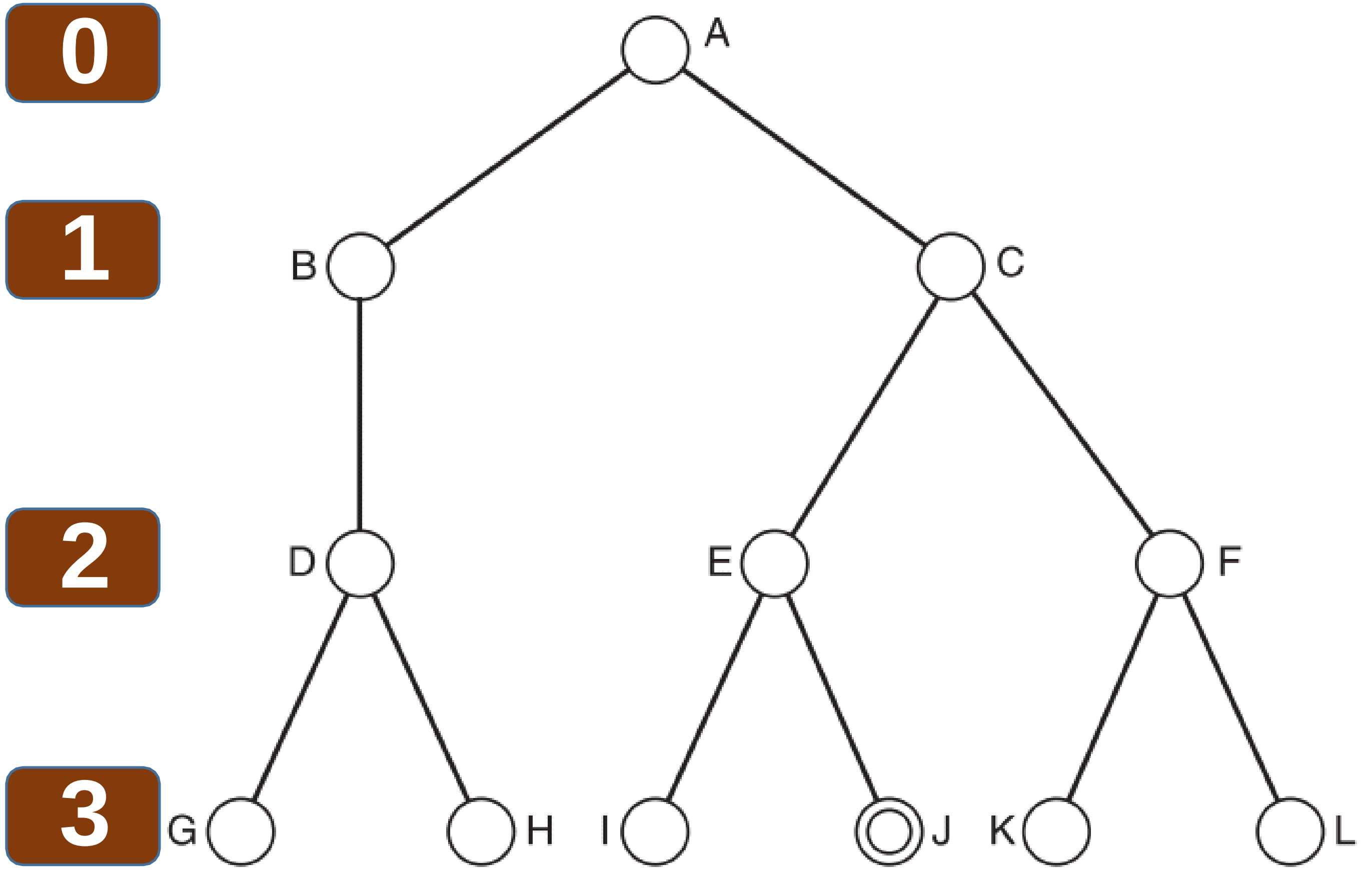
Current

A

B

A

Iterative Deepening Search
Depth – 1, Goal – Node J

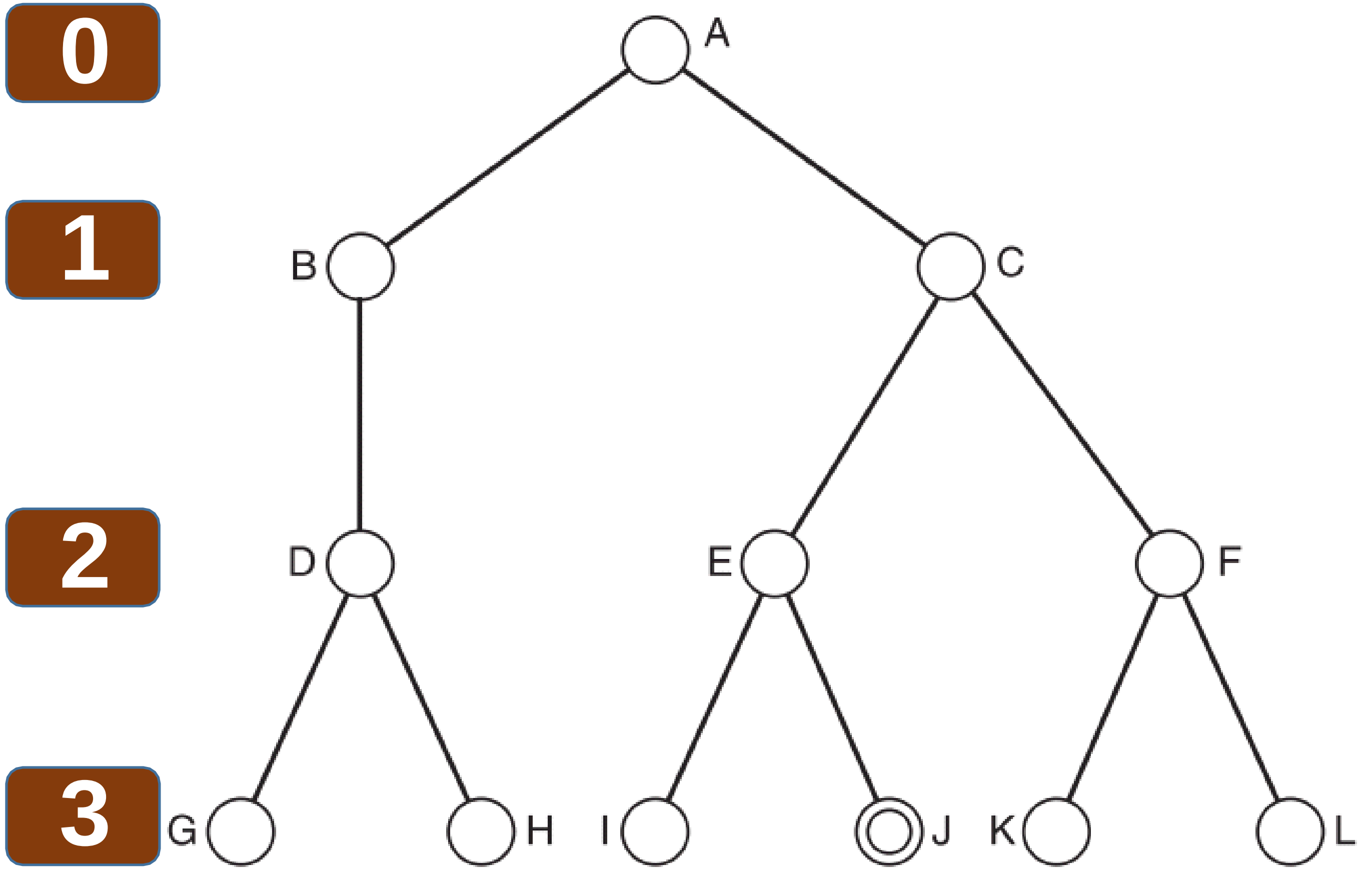


Current

A
B
A

C

Iterative Deepening Search
Depth – 1, Goal – Node J



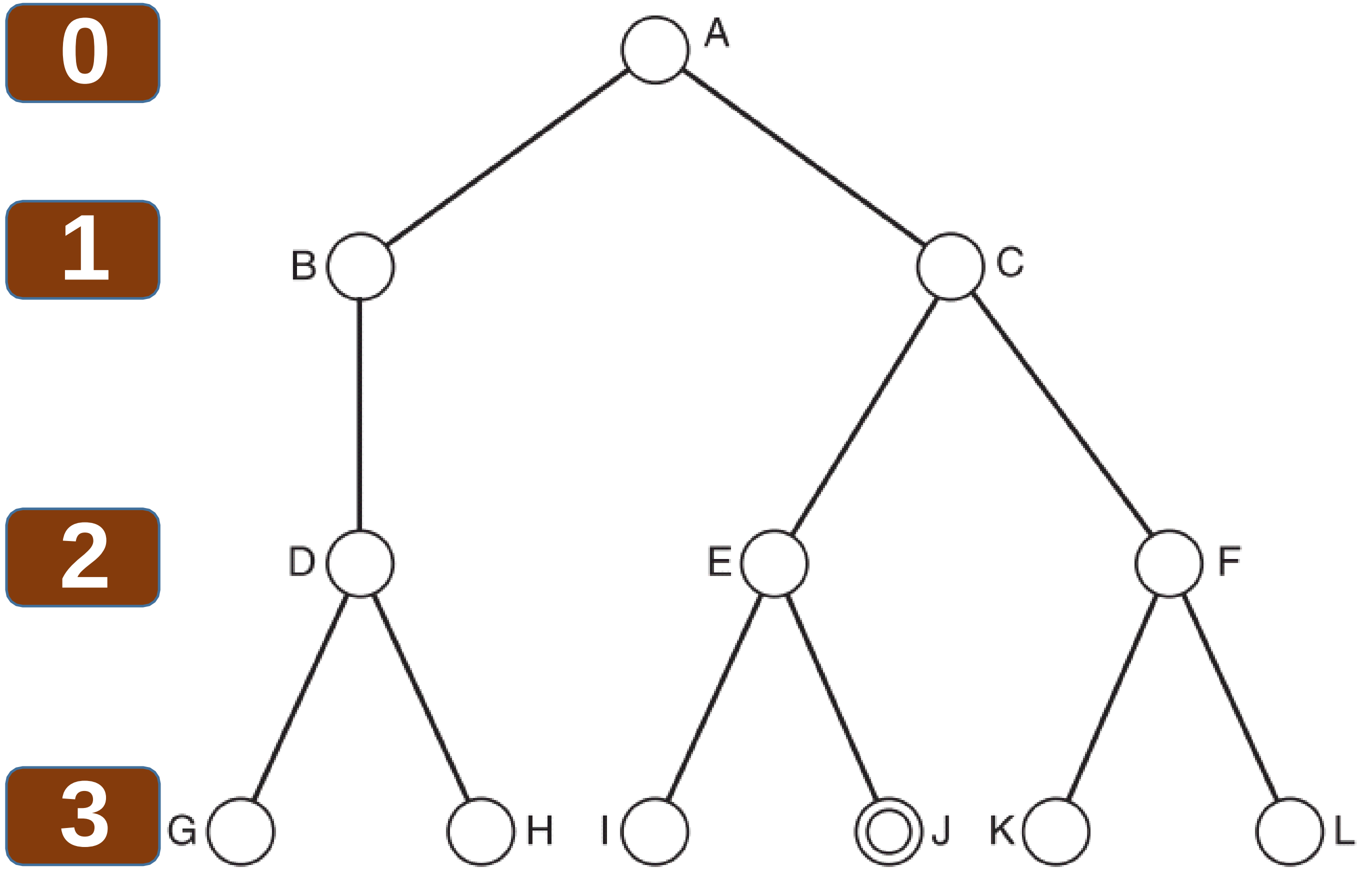
Current

A C

B

A

Iterative Deepening Search
Depth – 1, Goal – Node J



Current

A C

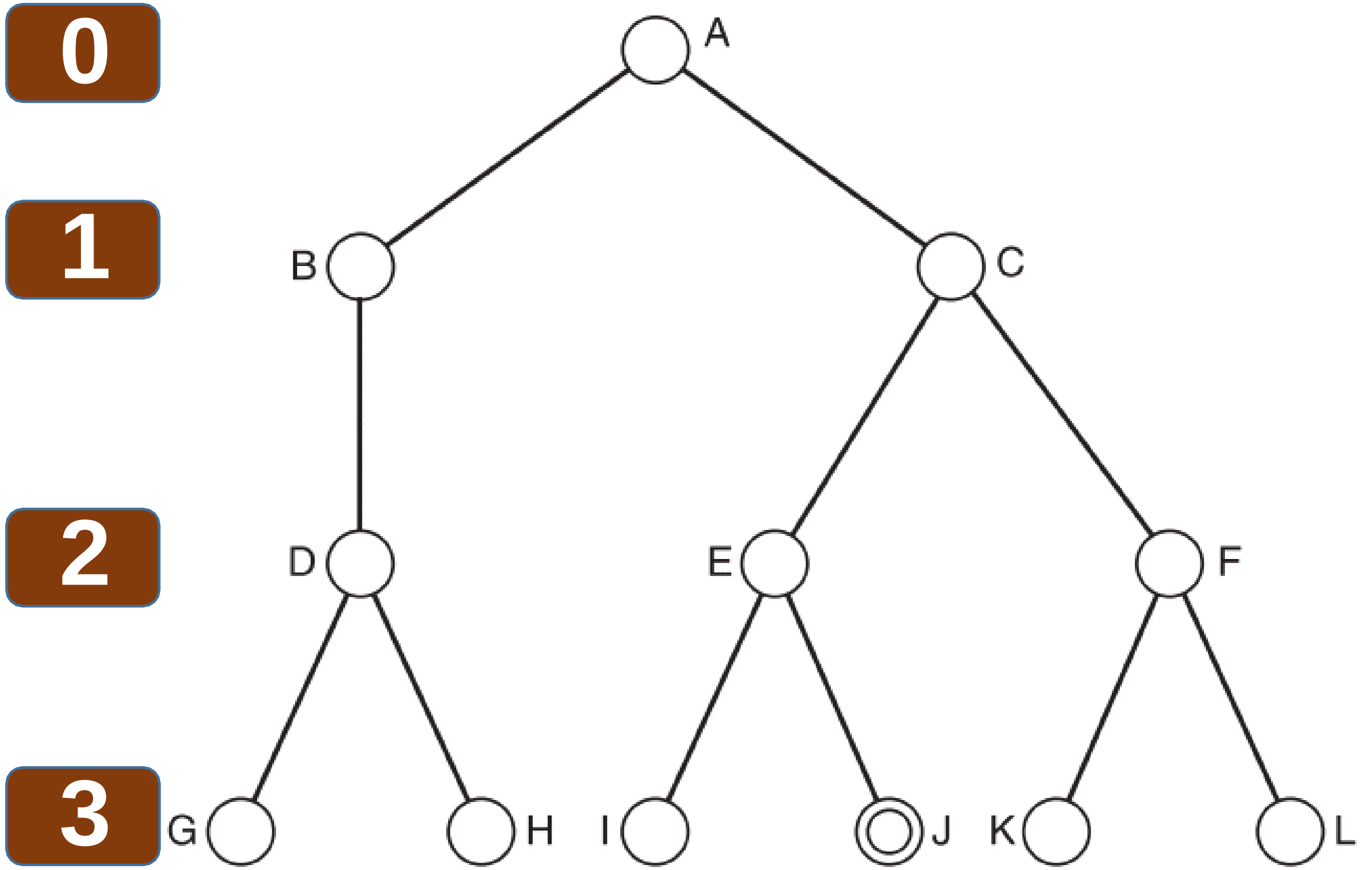
B

A

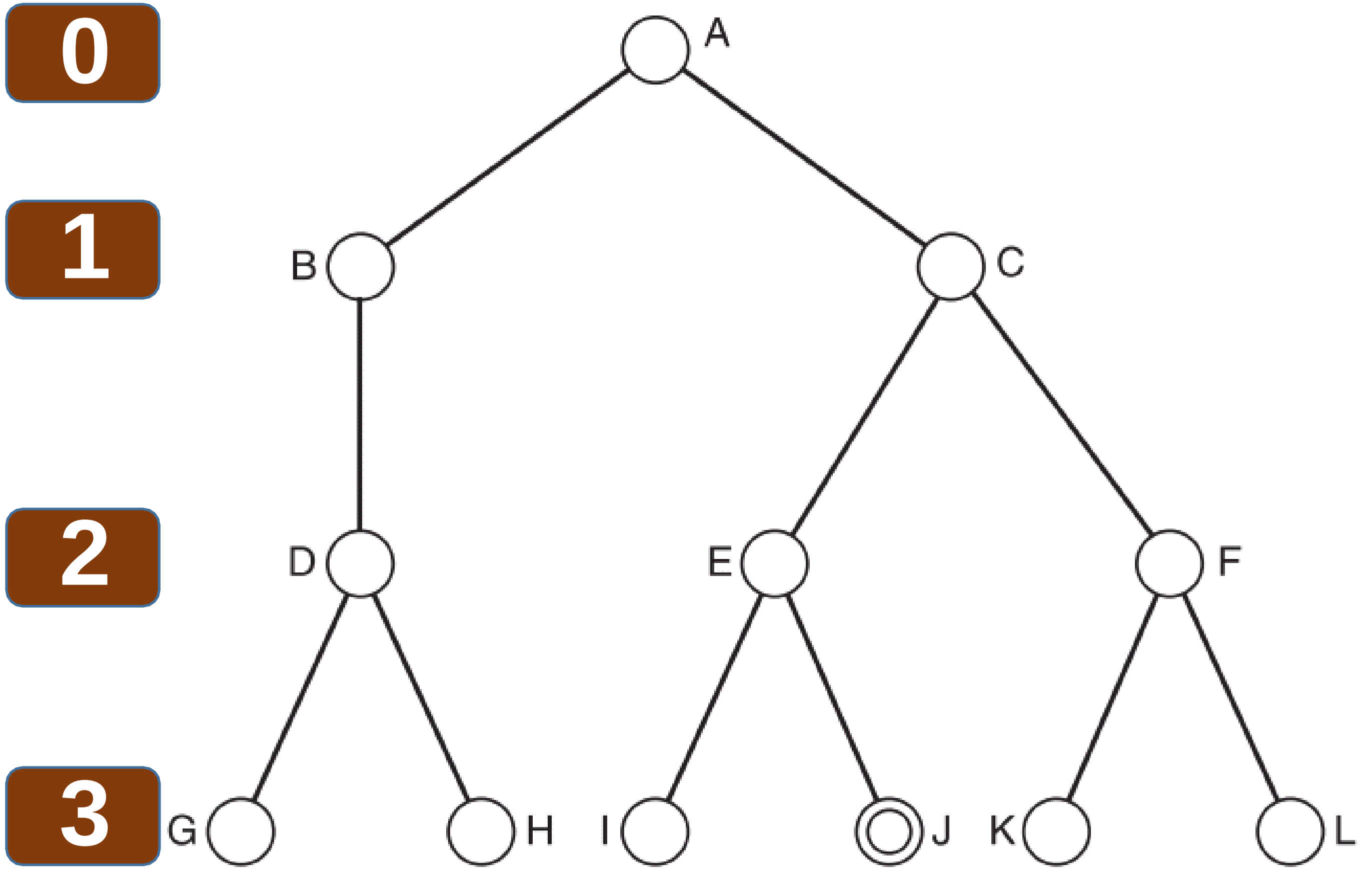
Search Finished
NO GOAL
Increase Depth by 1

Iterative Deepening Search
Depth – 2, Goal – Node J

Current



Iterative Deepening Search
Depth – 2, Goal – Node J



Current

A

Iterative Deepening Search
Depth – 2, Goal – Node J

Current

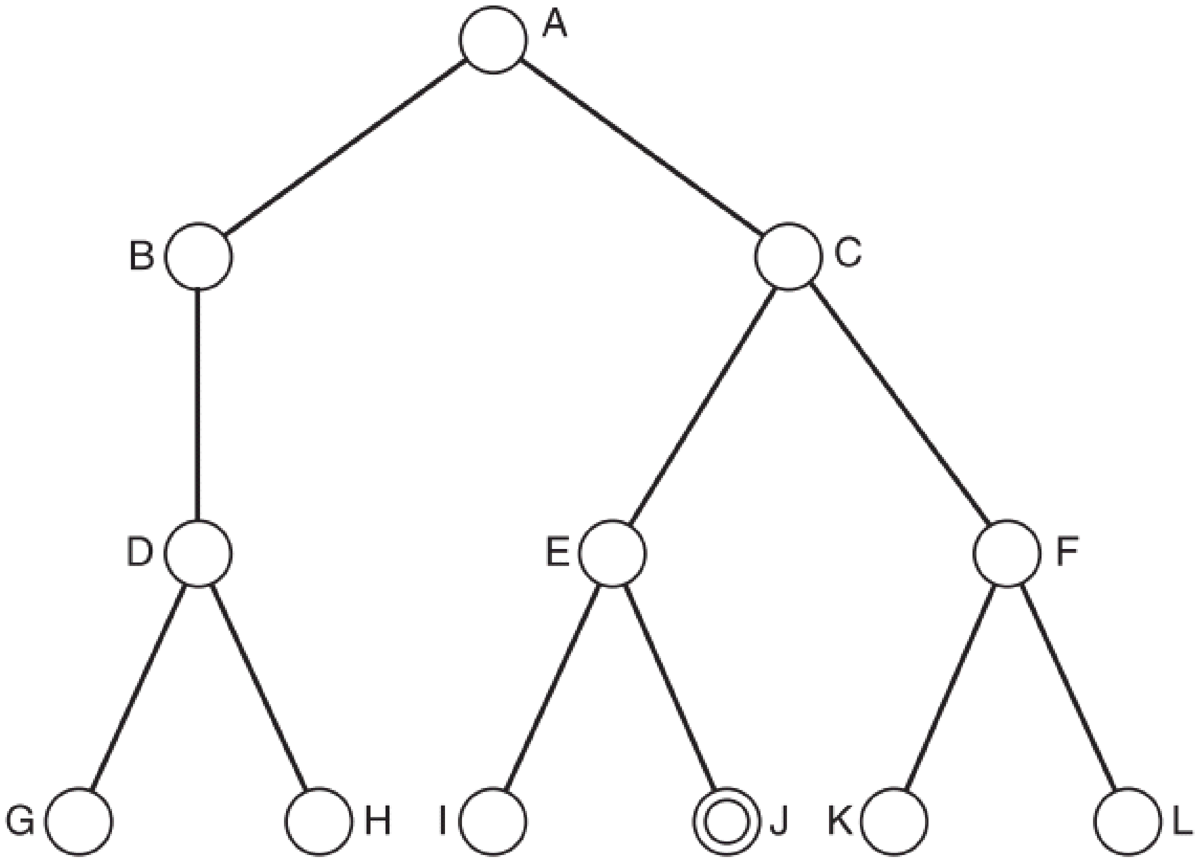
A

0

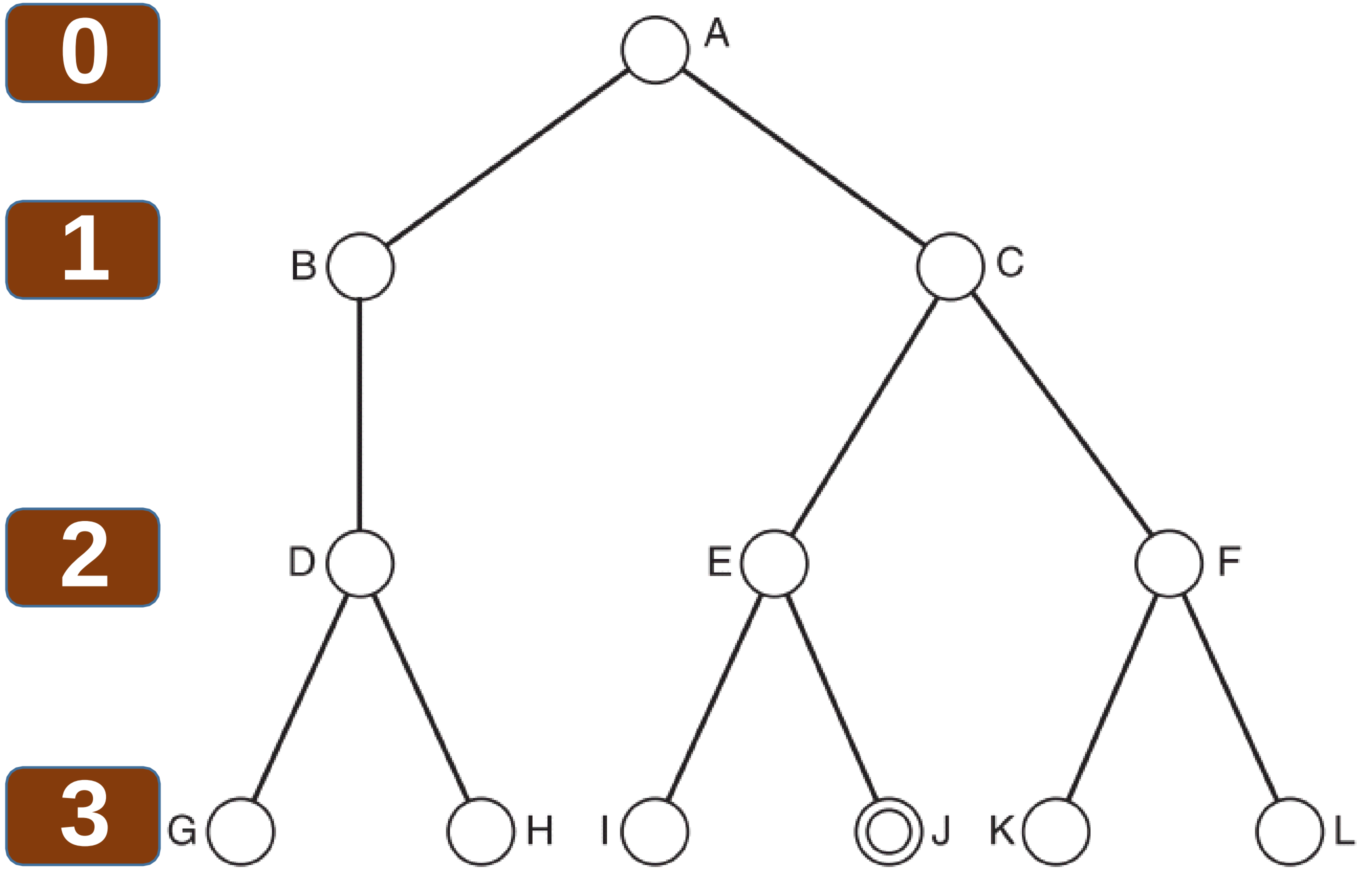
1

2

3



Iterative Deepening Search
Depth – 2, Goal – Node J



Current

A

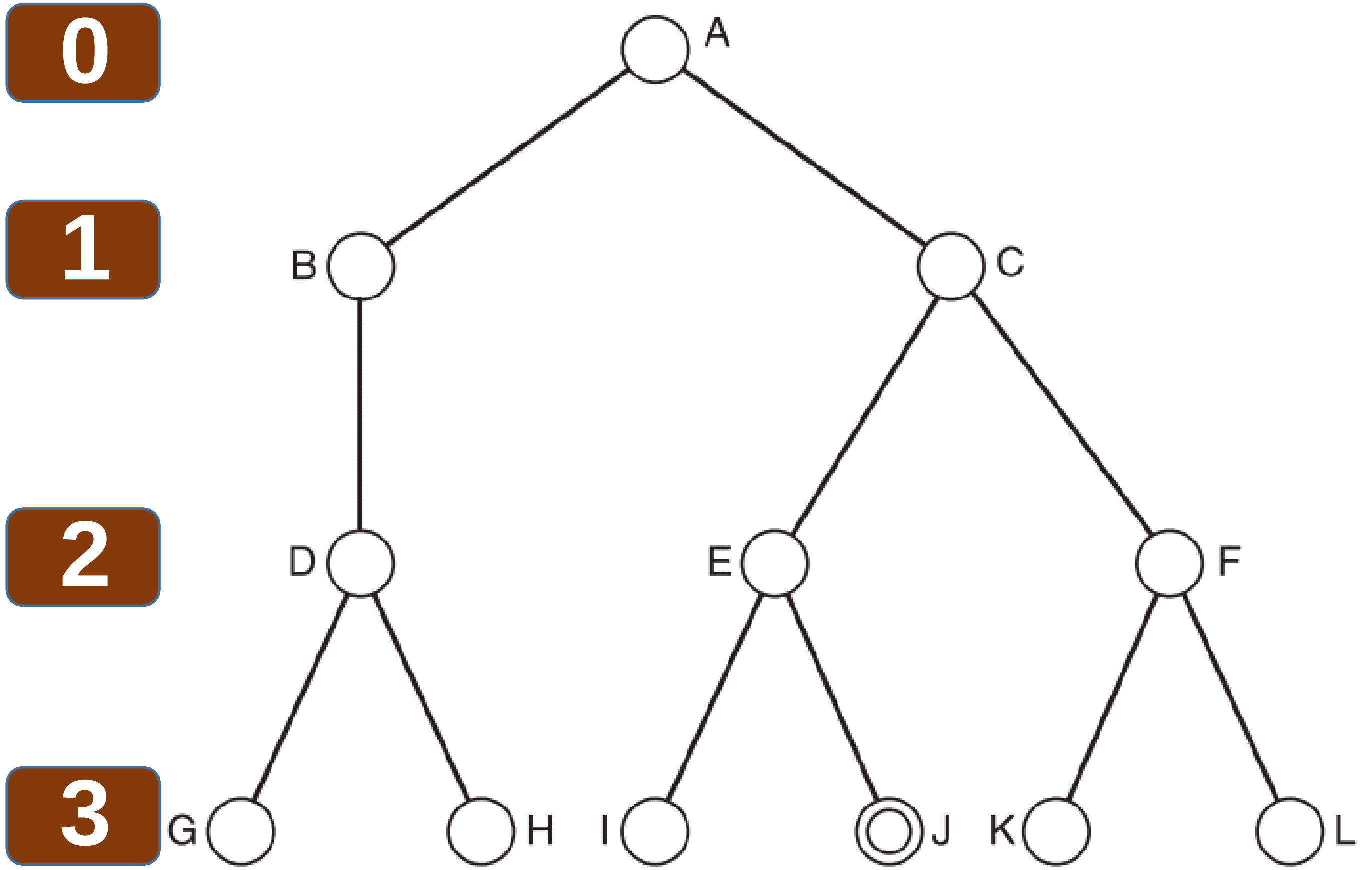
B

Iterative Deepening Search
Depth – 2, Goal – Node J

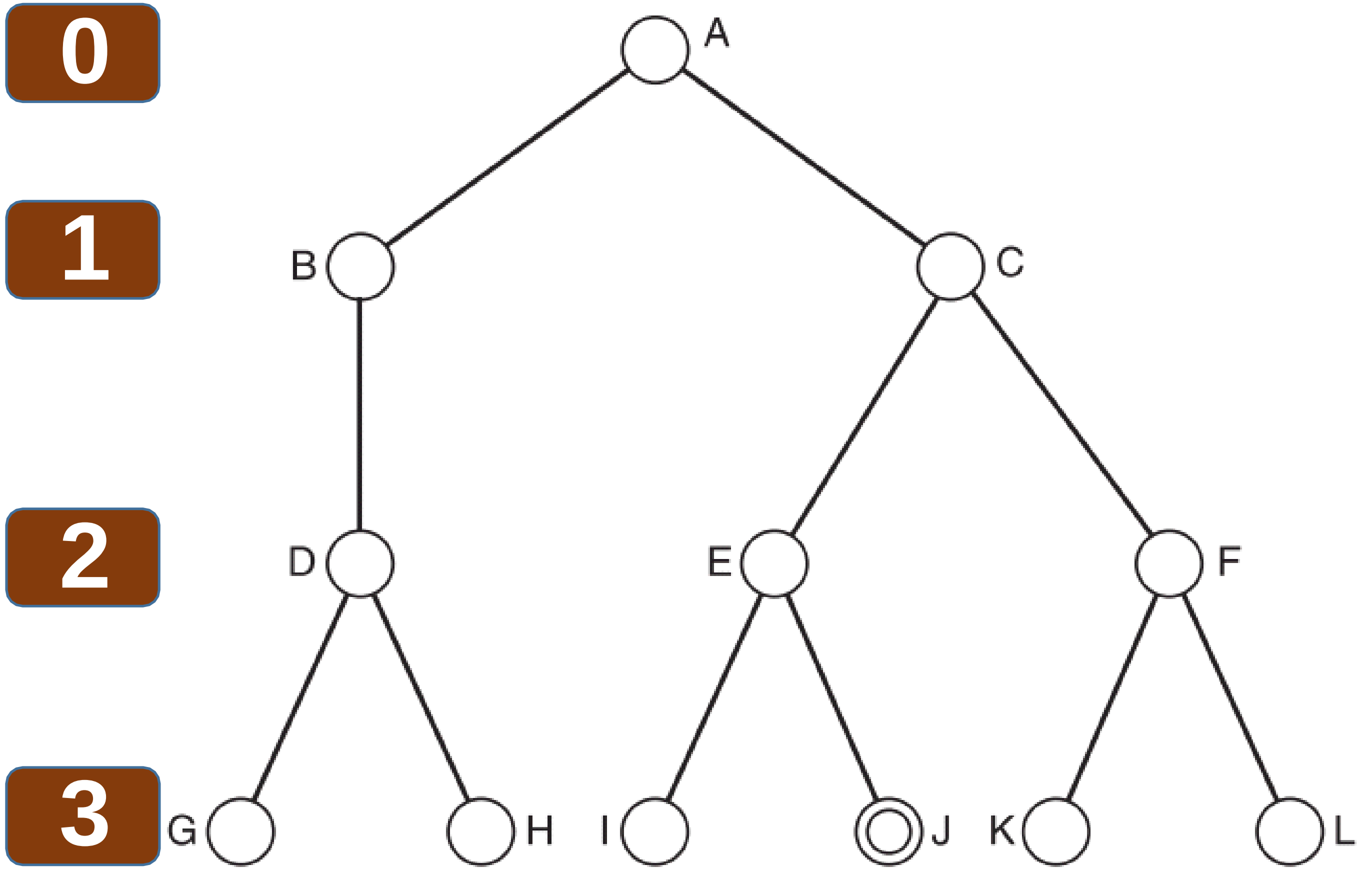
Current

A

B



Iterative Deepening Search
Depth – 2, Goal – Node J



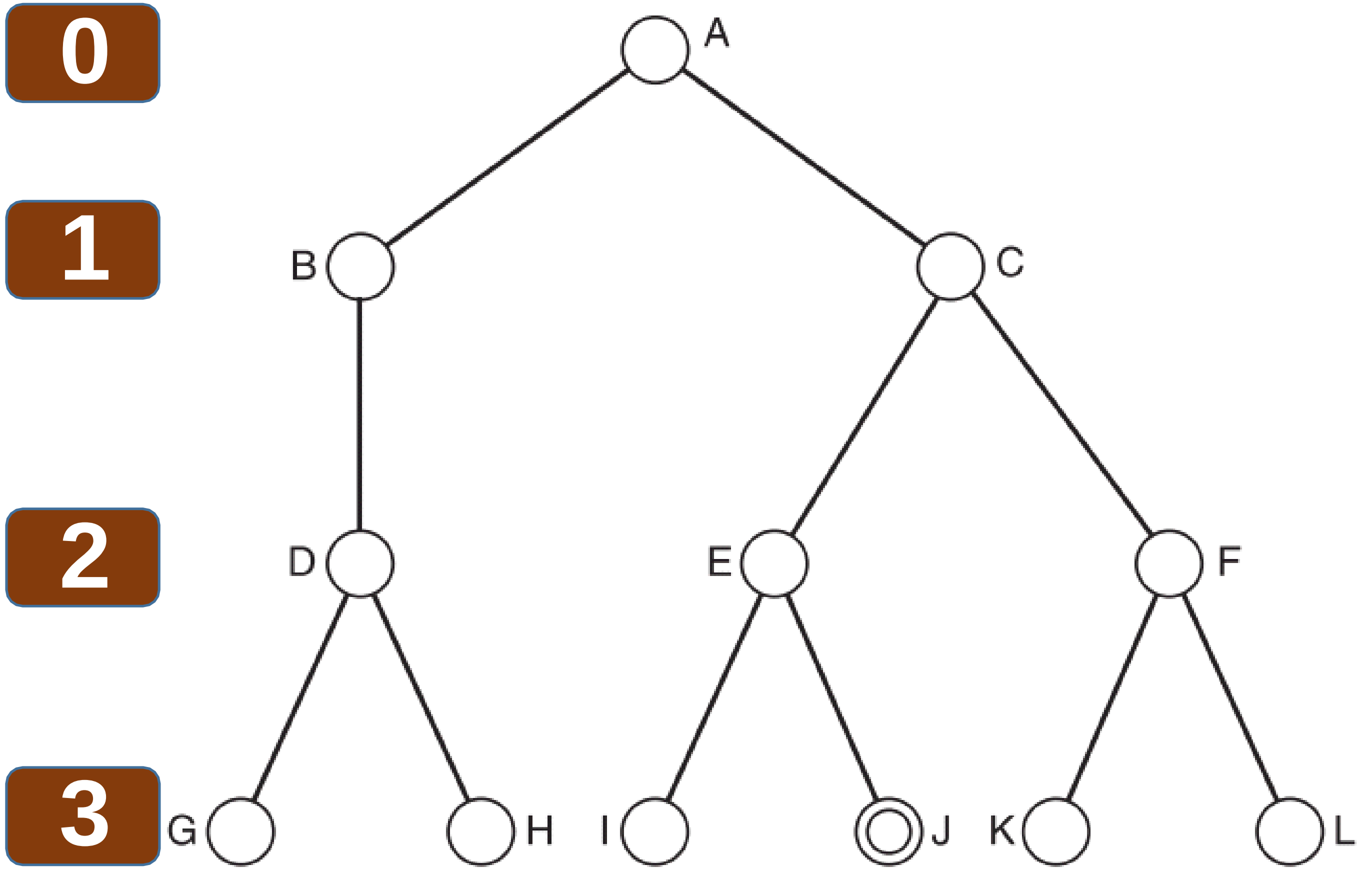
Current

A

B

D

Iterative Deepening Search
Depth – 2, Goal – Node J



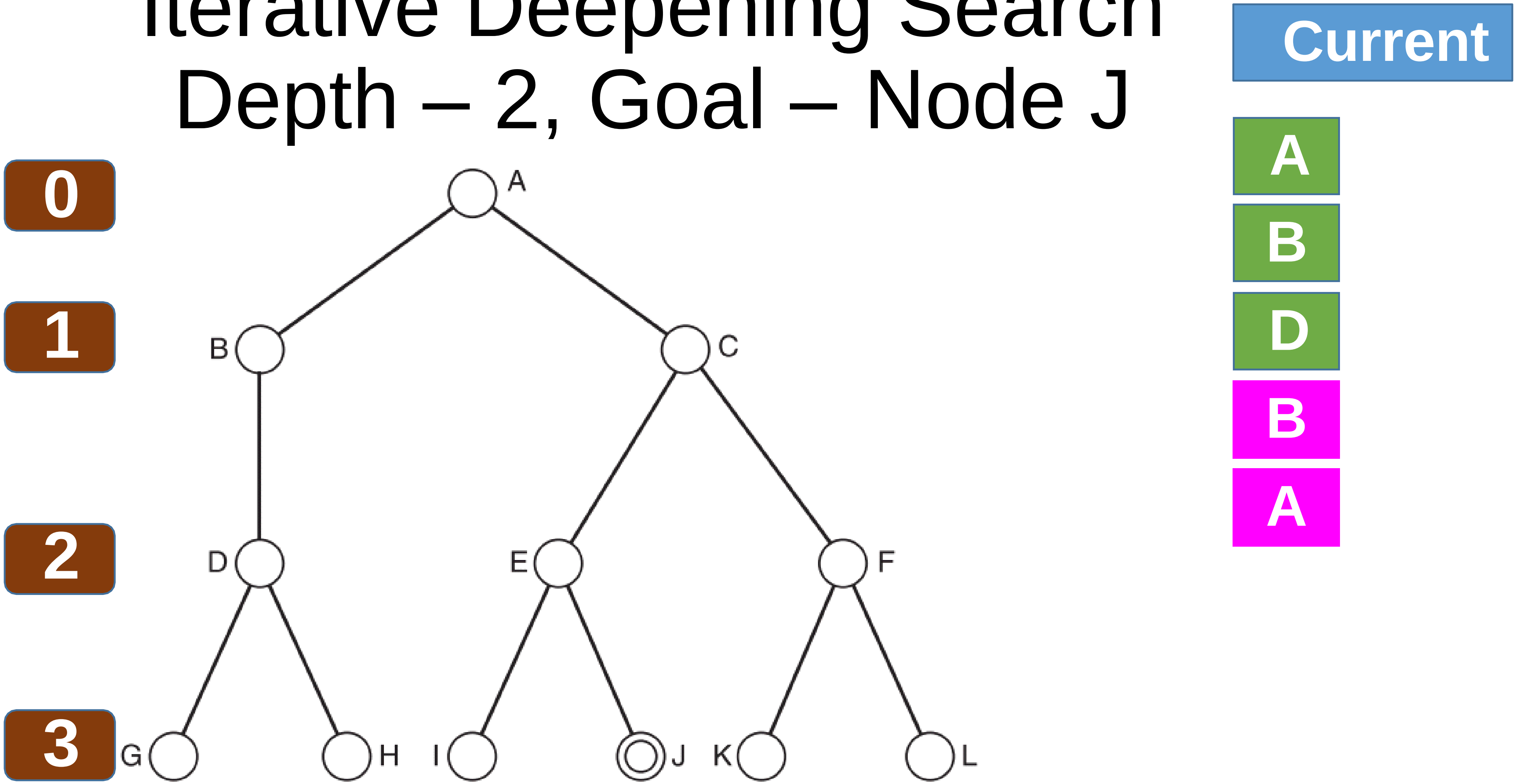
Current

A

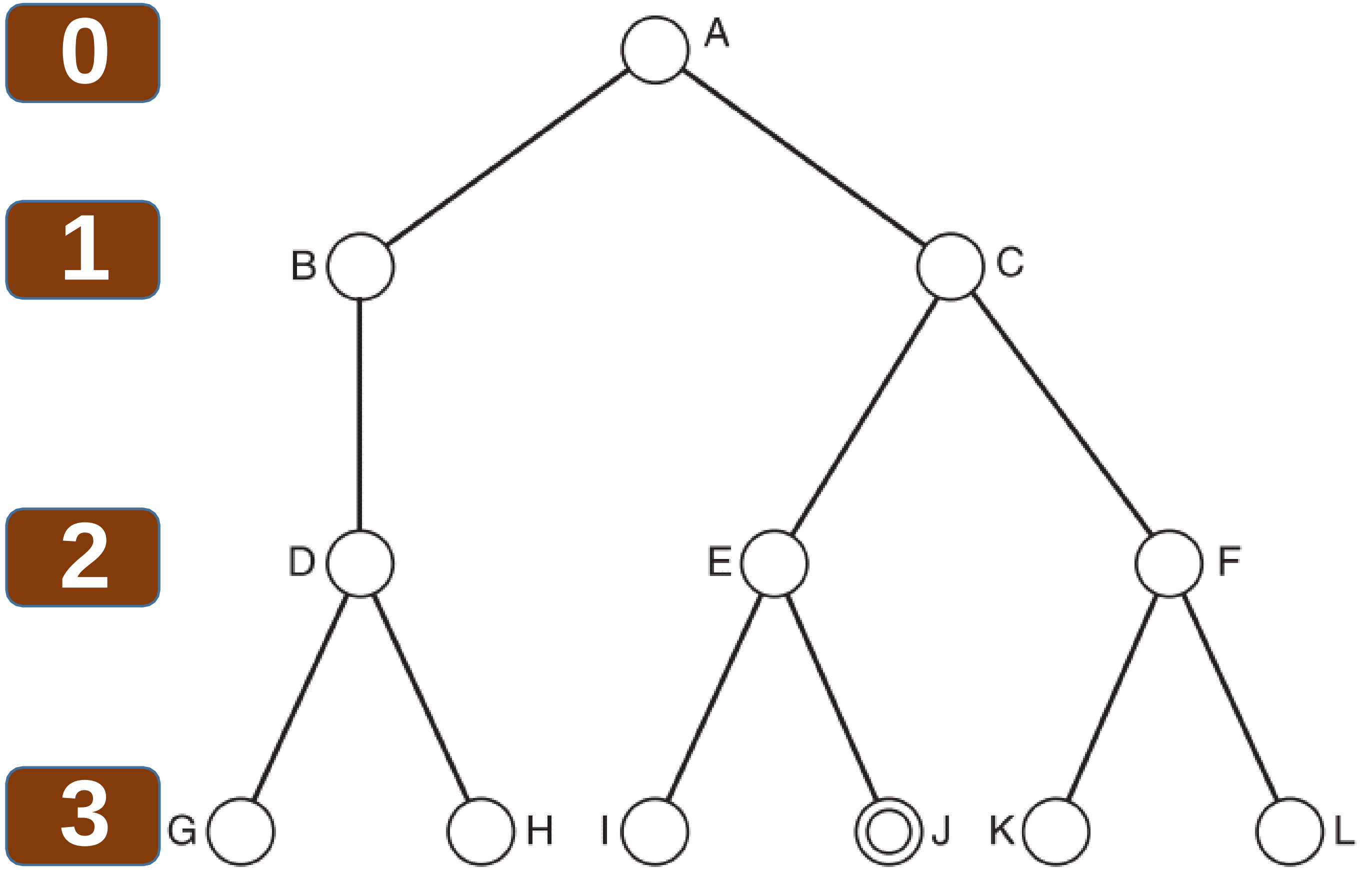
B

D

Iterative Deepening Search
Depth – 2, Goal – Node J



Iterative Deepening Search
Depth – 2, Goal – Node J



Current

A

B

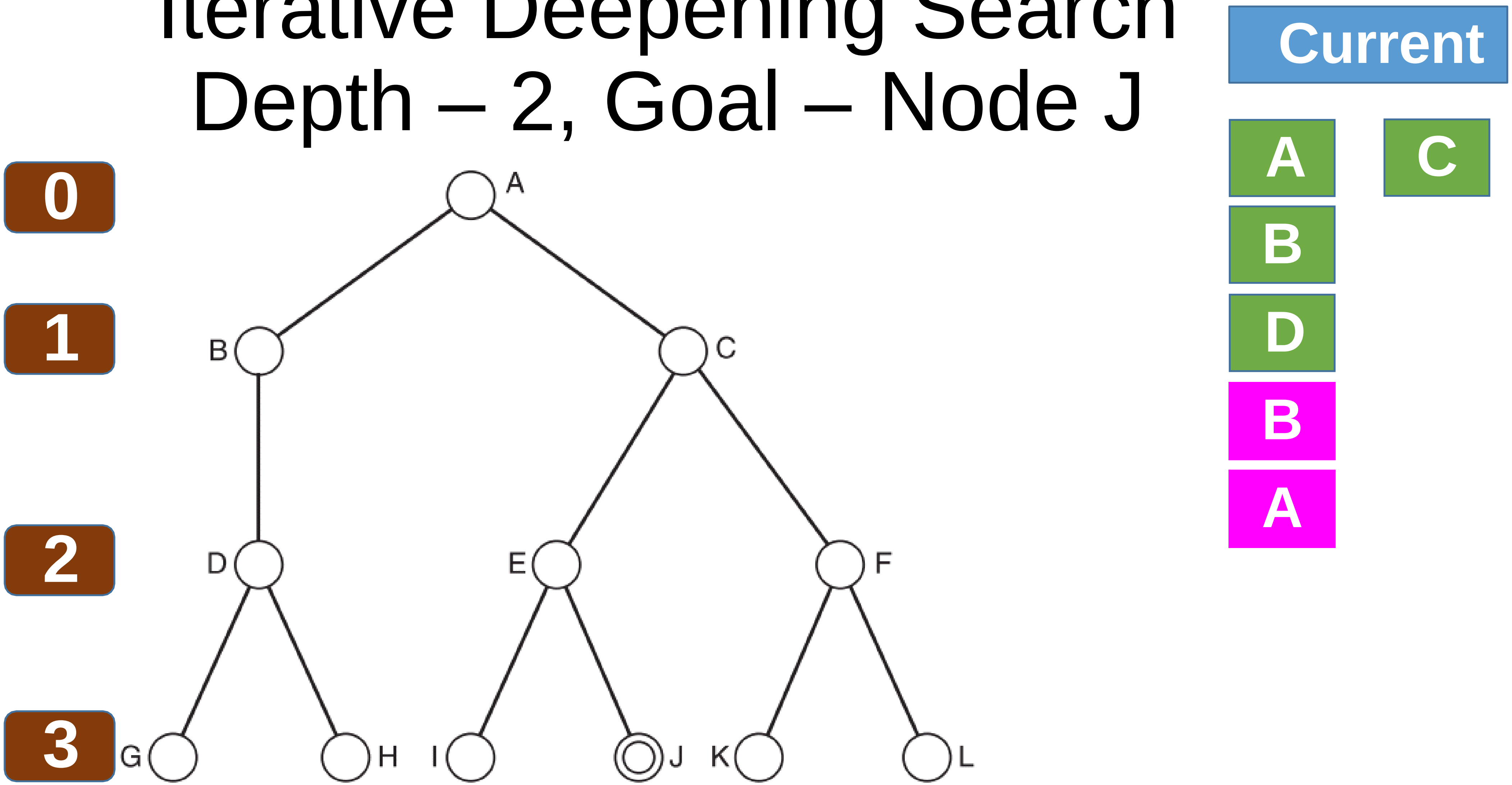
D

B

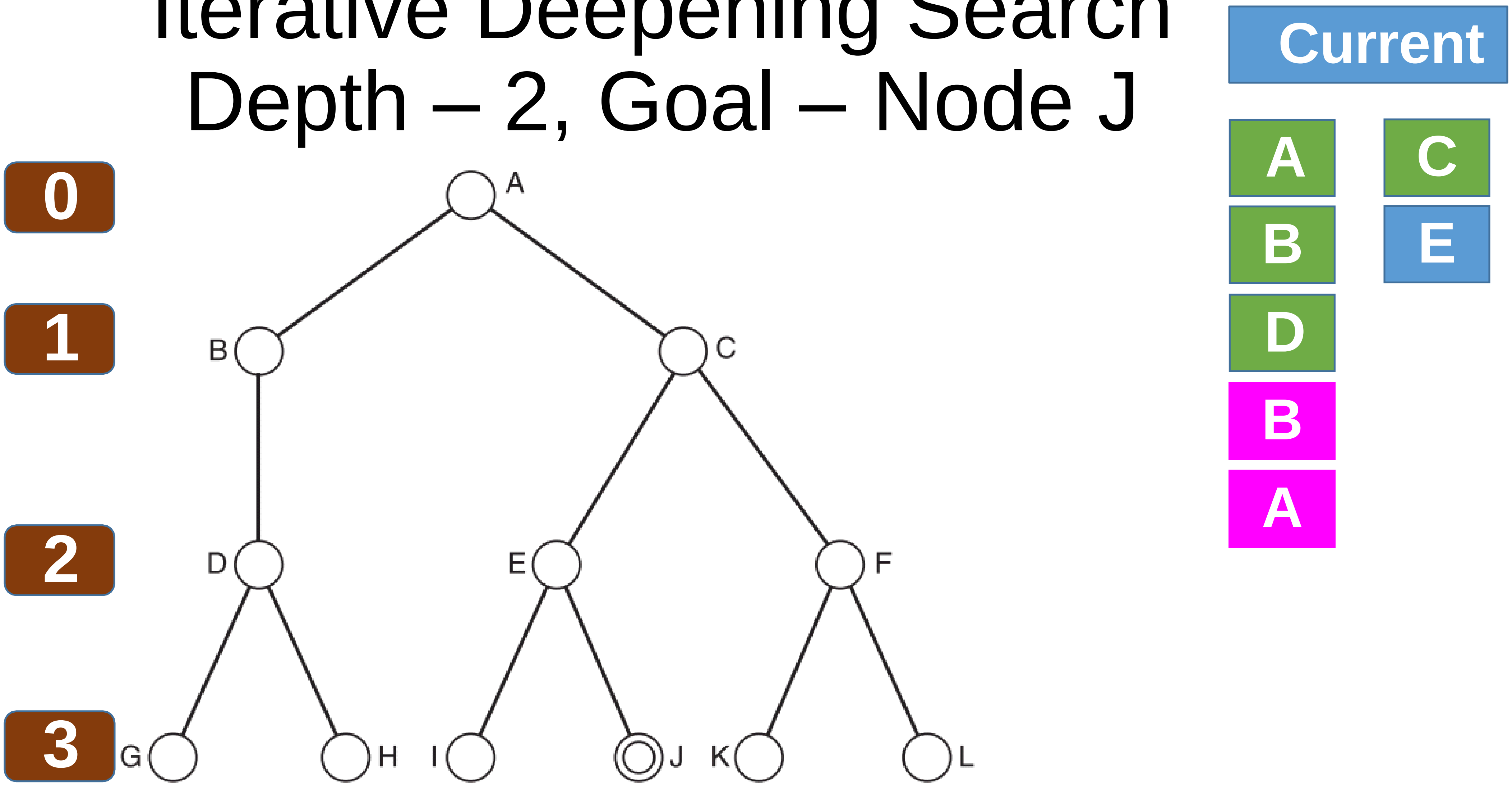
A

C

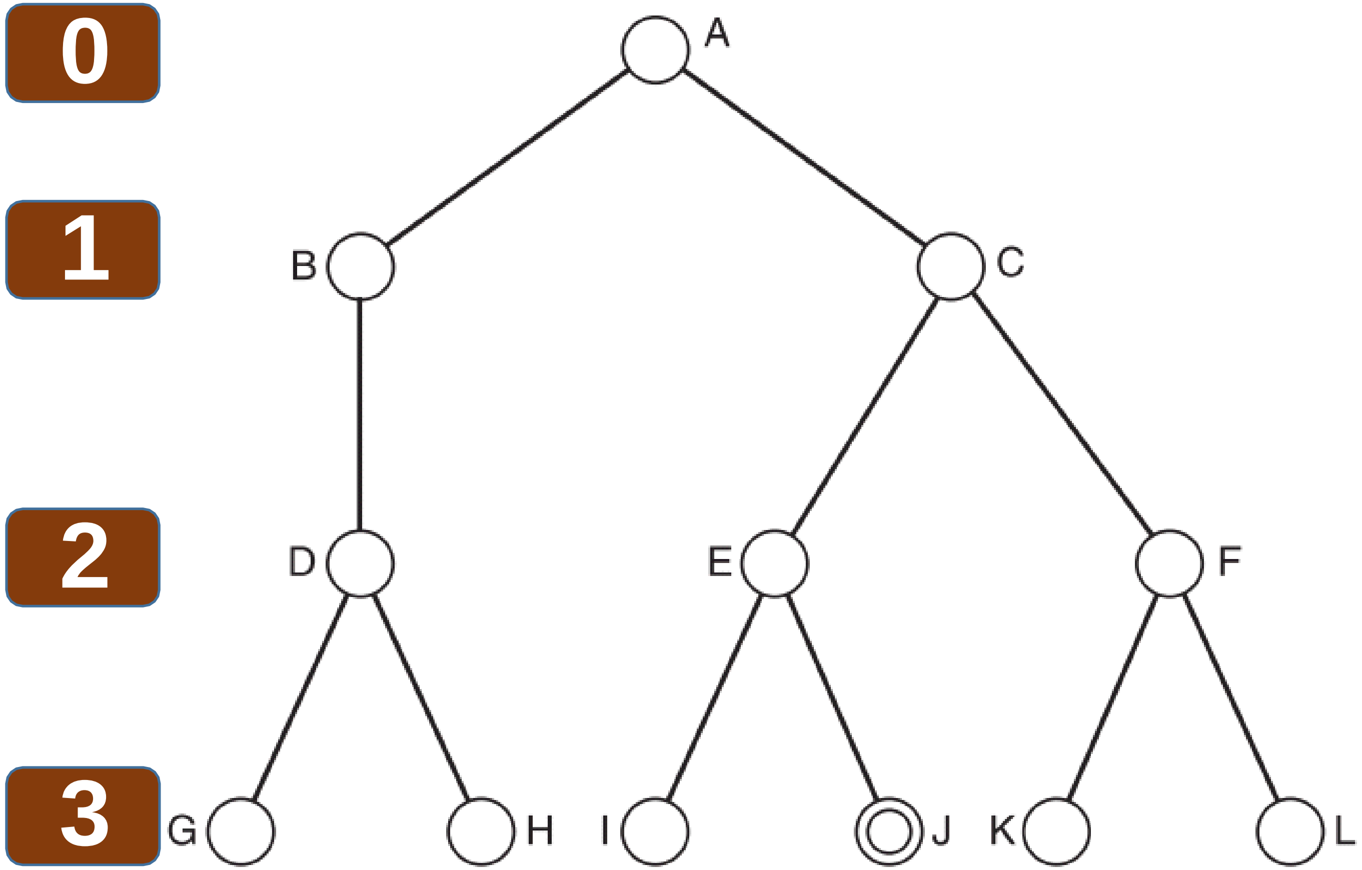
Iterative Deepening Search
Depth – 2, Goal – Node J



Iterative Deepening Search
Depth – 2, Goal – Node J



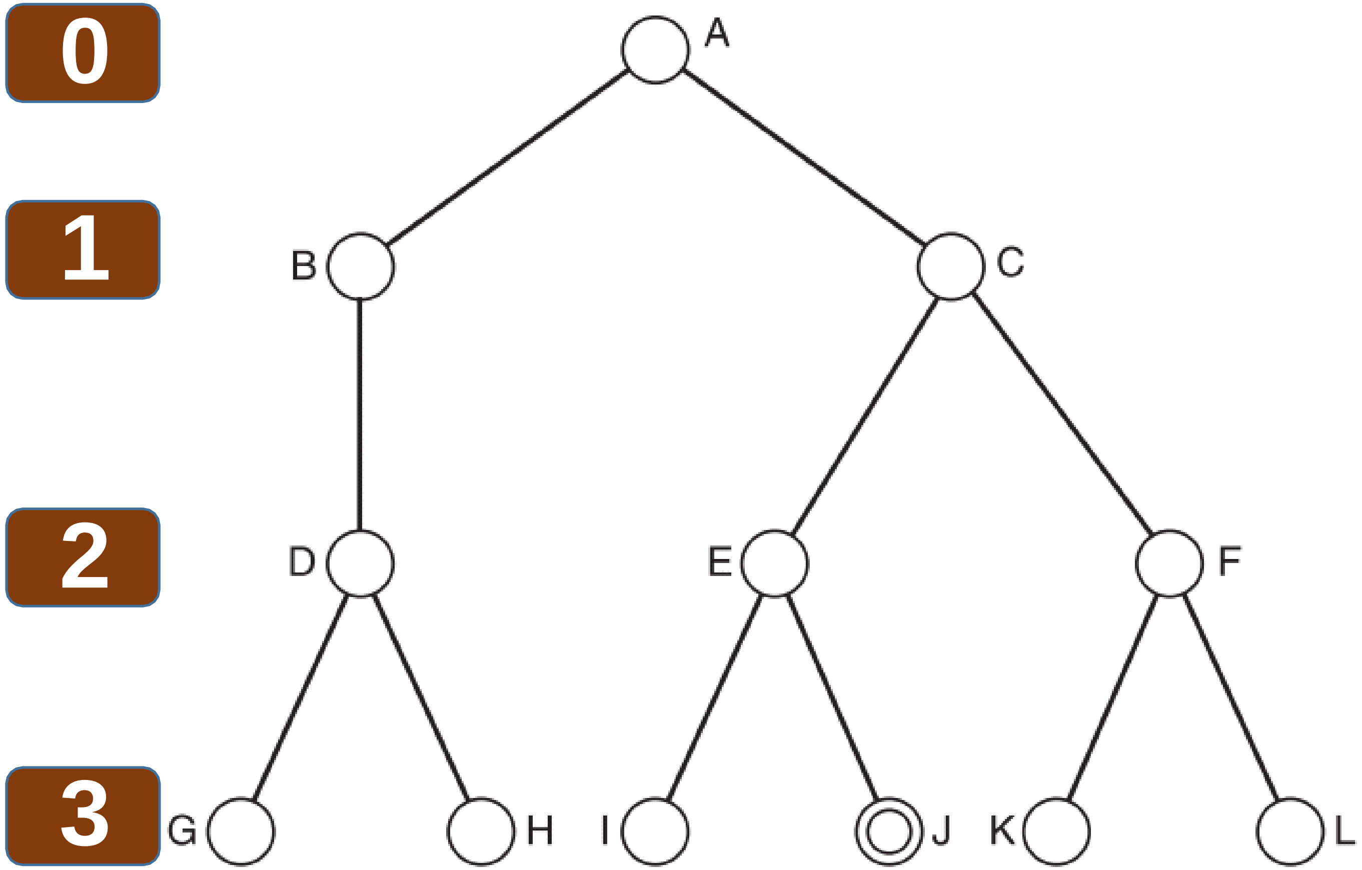
Iterative Deepening Search
Depth – 2, Goal – Node J

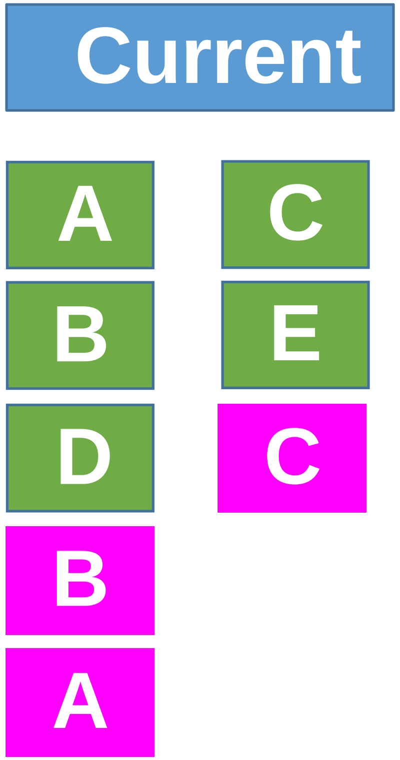


Current

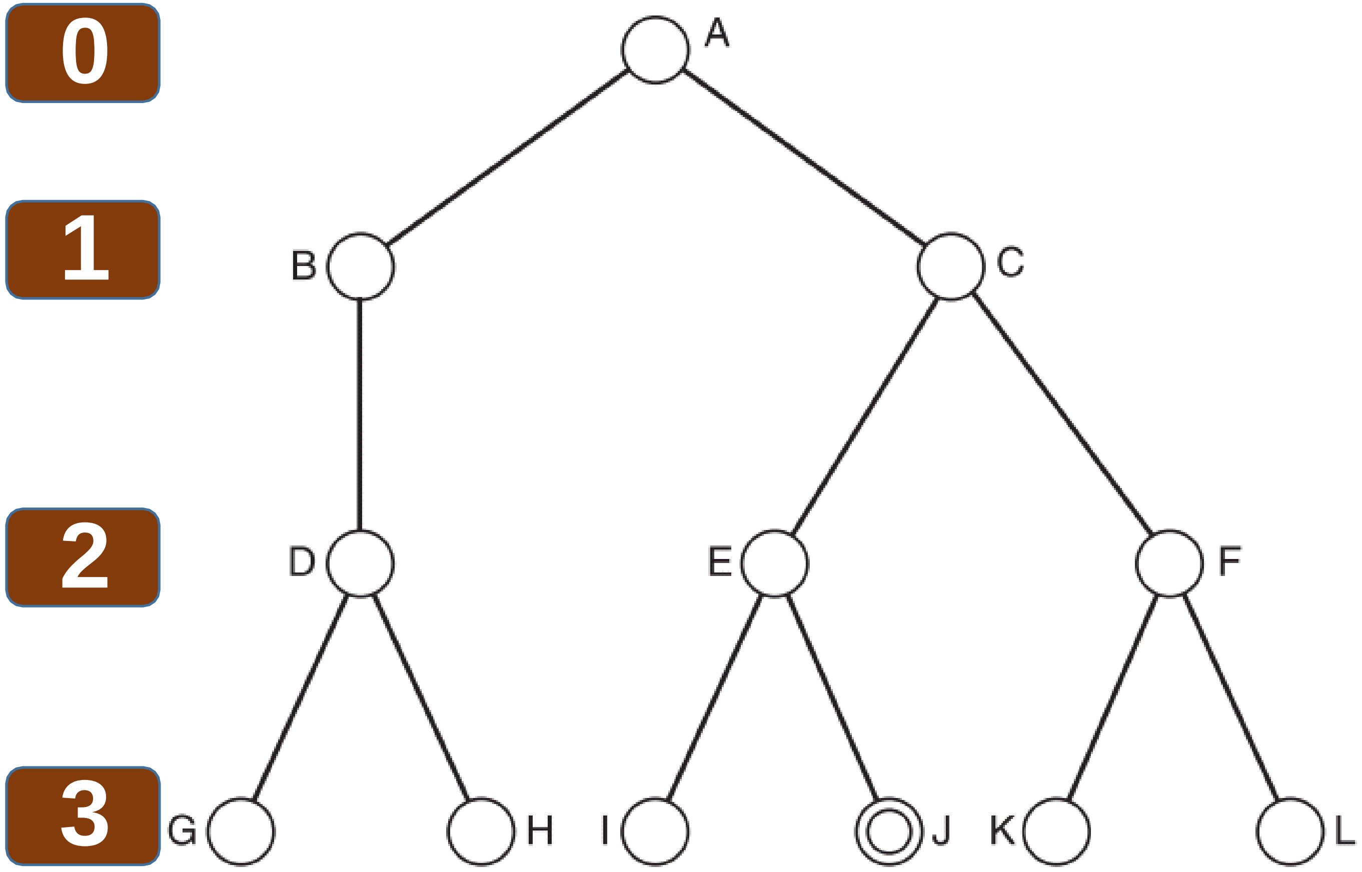
A C
B E
D
B
A

Iterative Deepening Search
Depth – 2, Goal – Node J





Iterative Deepening Search
Depth – 2, Goal – Node J

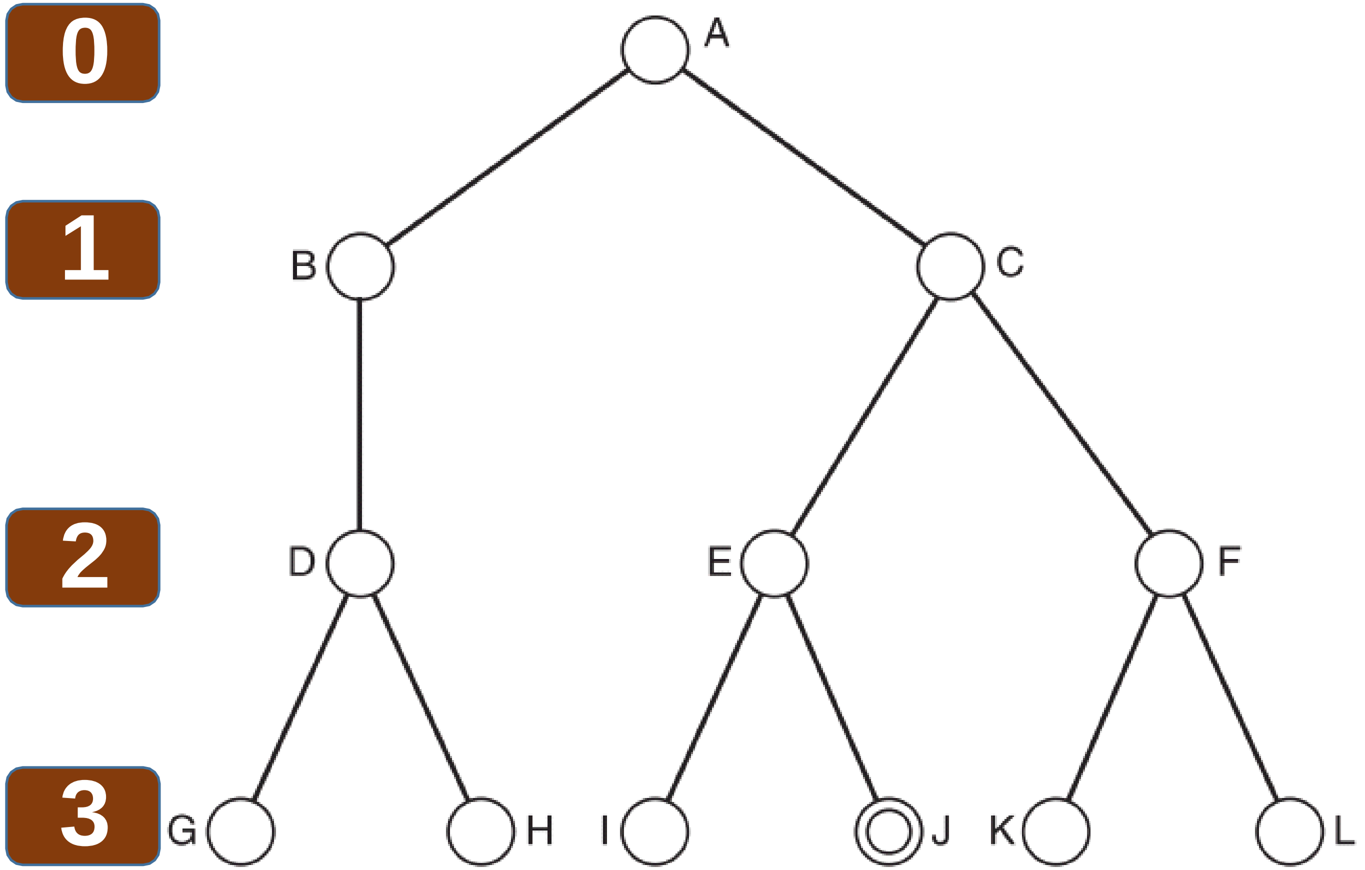


Current

A
B
D
B
A

C
E
C
F

Iterative Deepening Search
Depth – 2, Goal – Node J

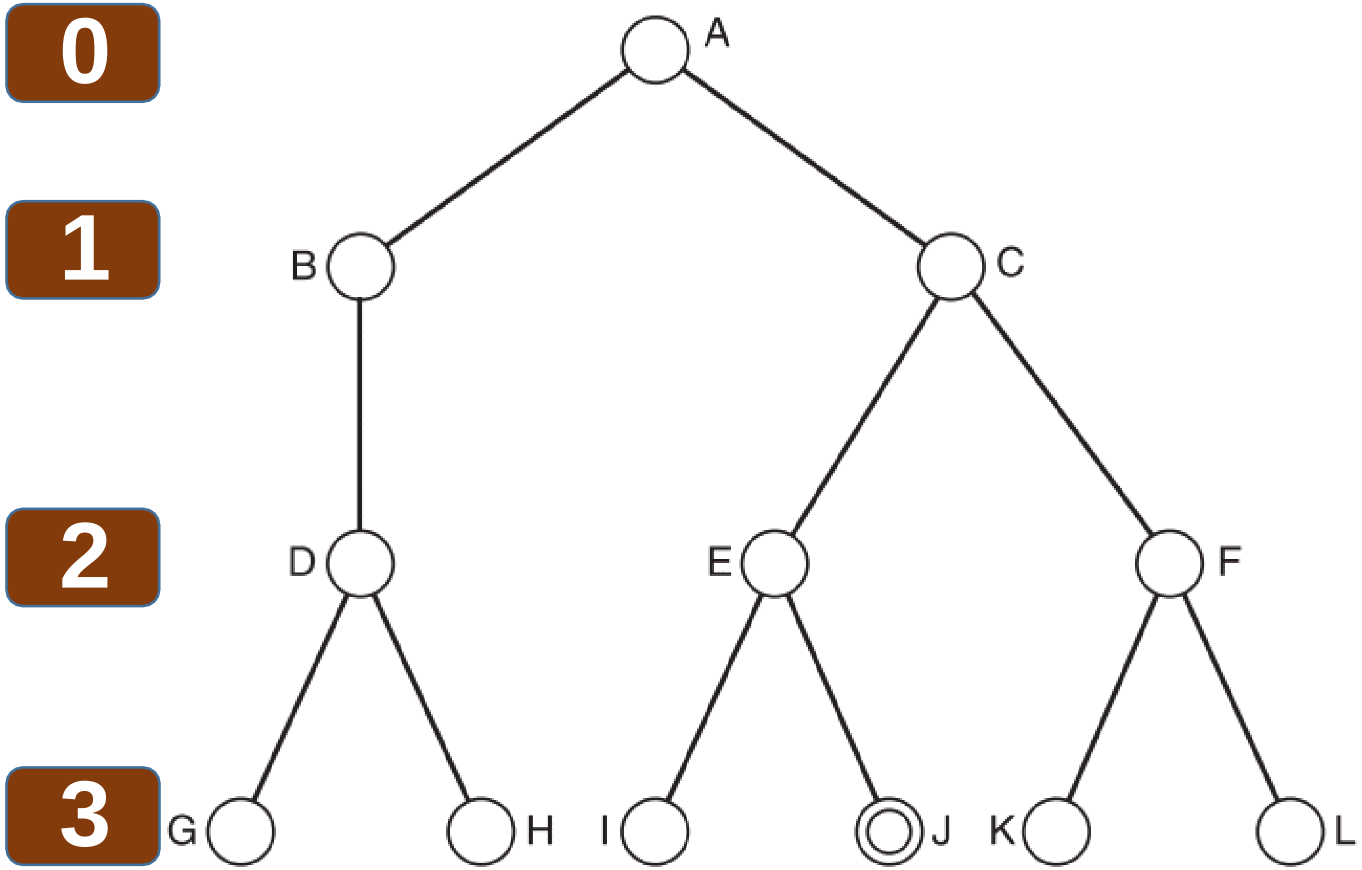


Current

A
B
D
B
A

C
E
C
F

Iterative Deepening Search
Depth – 2, Goal – Node J



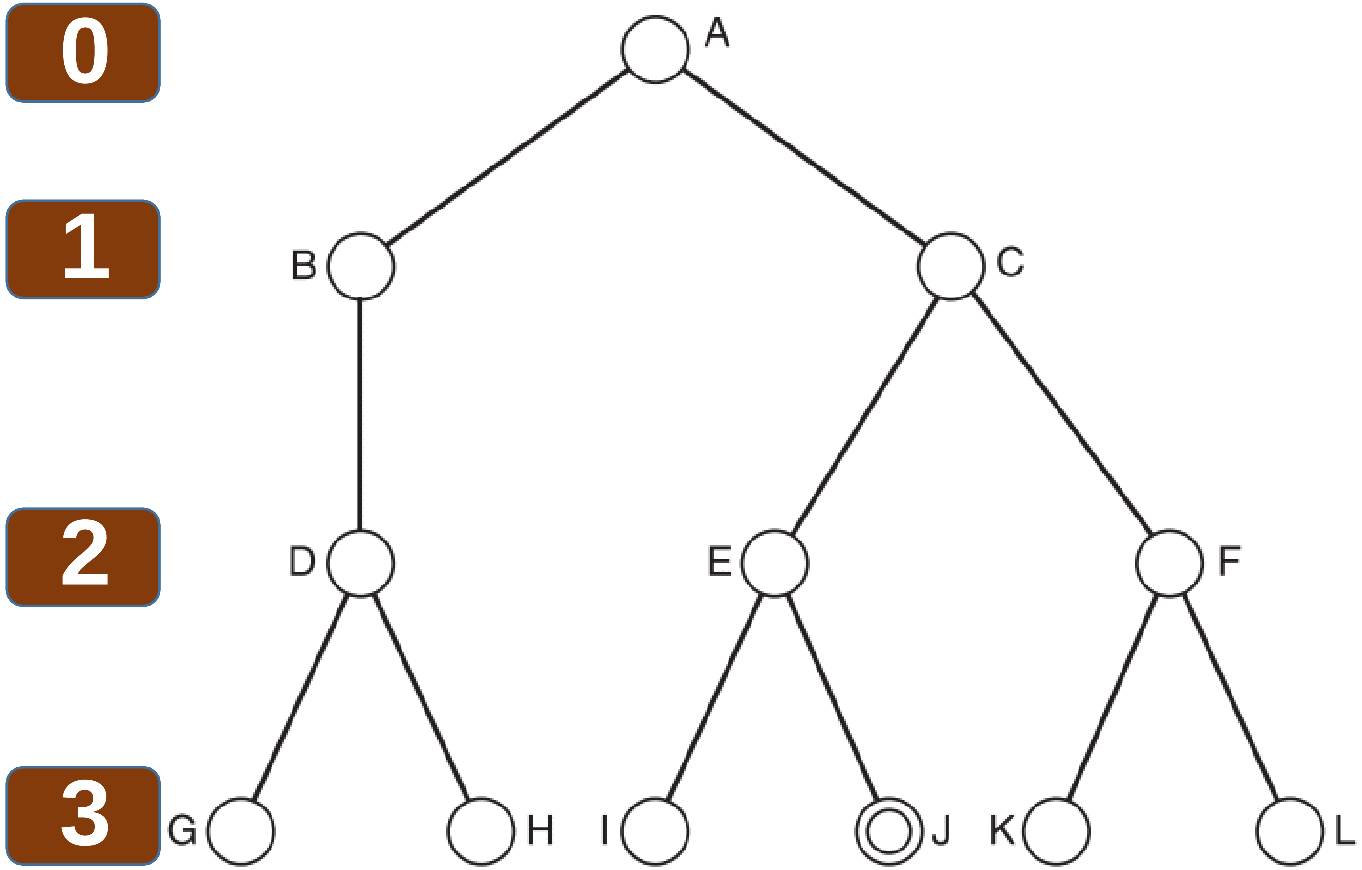
Current

A	C
B	E
D	C
B	F
A	

Search Finished
NO GOAL
Increase Depth by

Iterative Deepening Search
Depth – 3, Goal – Node J

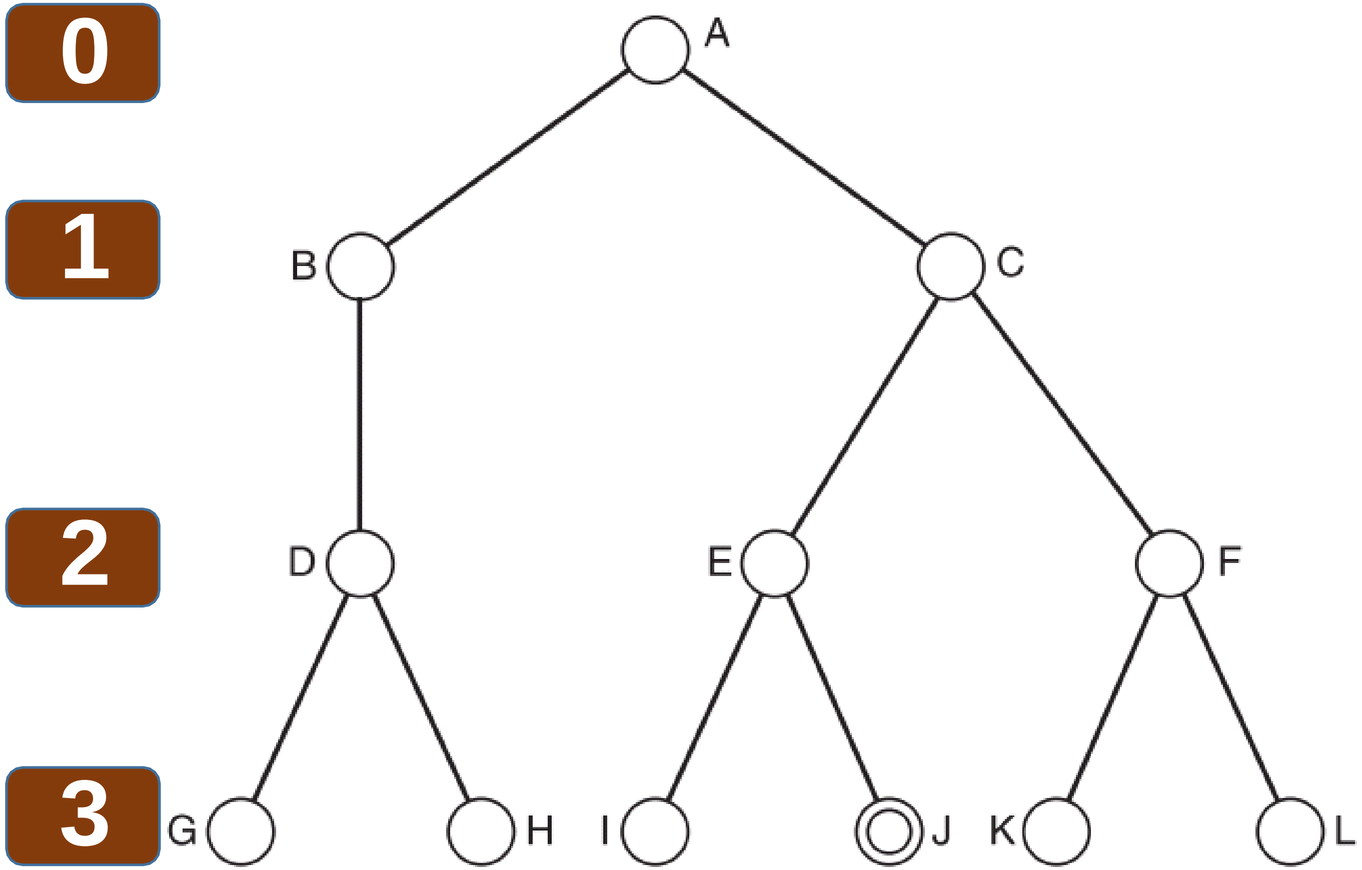
Current



Iterative Deepening Search
Depth – 3, Goal – Node J

Current

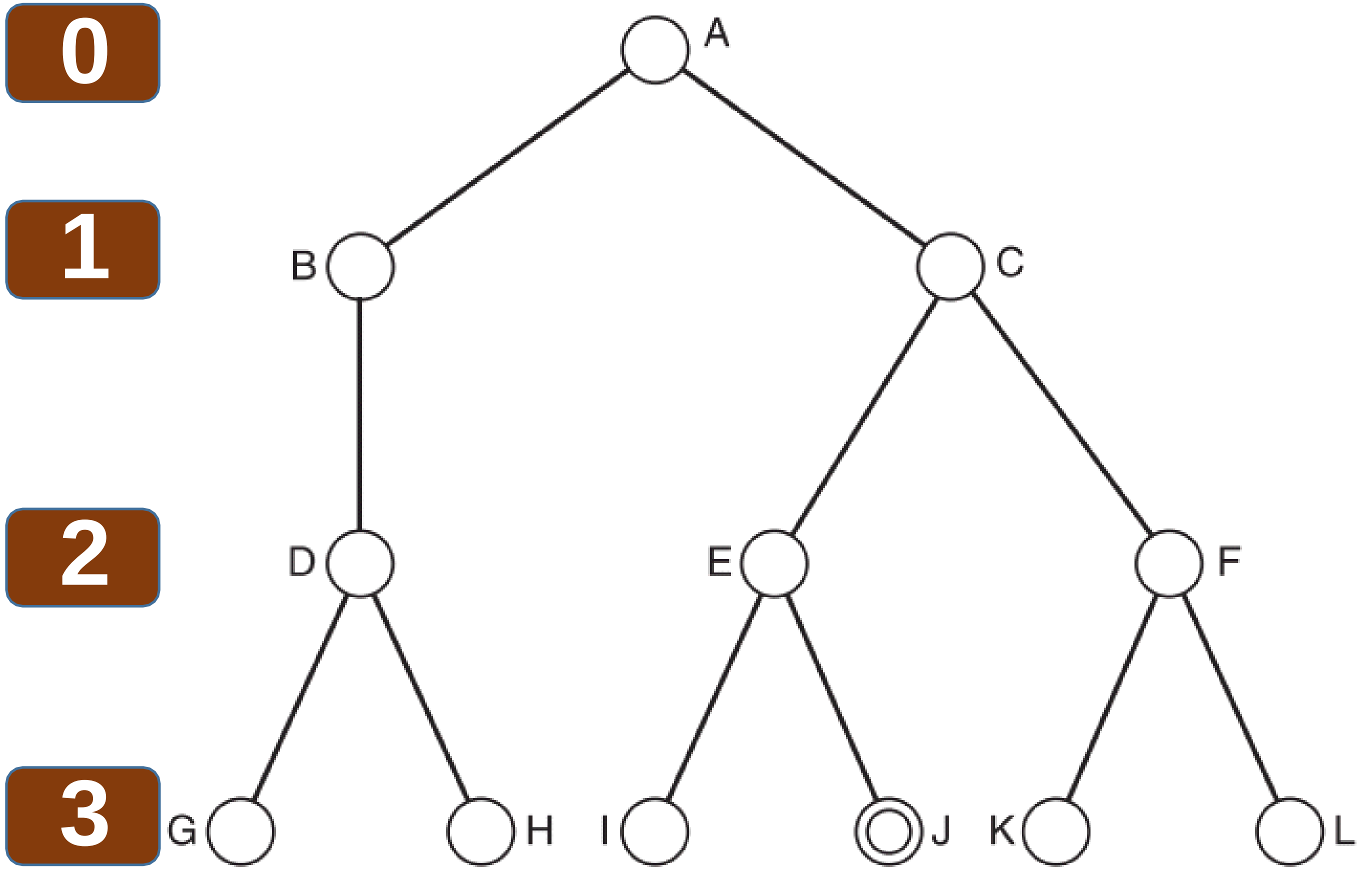
A



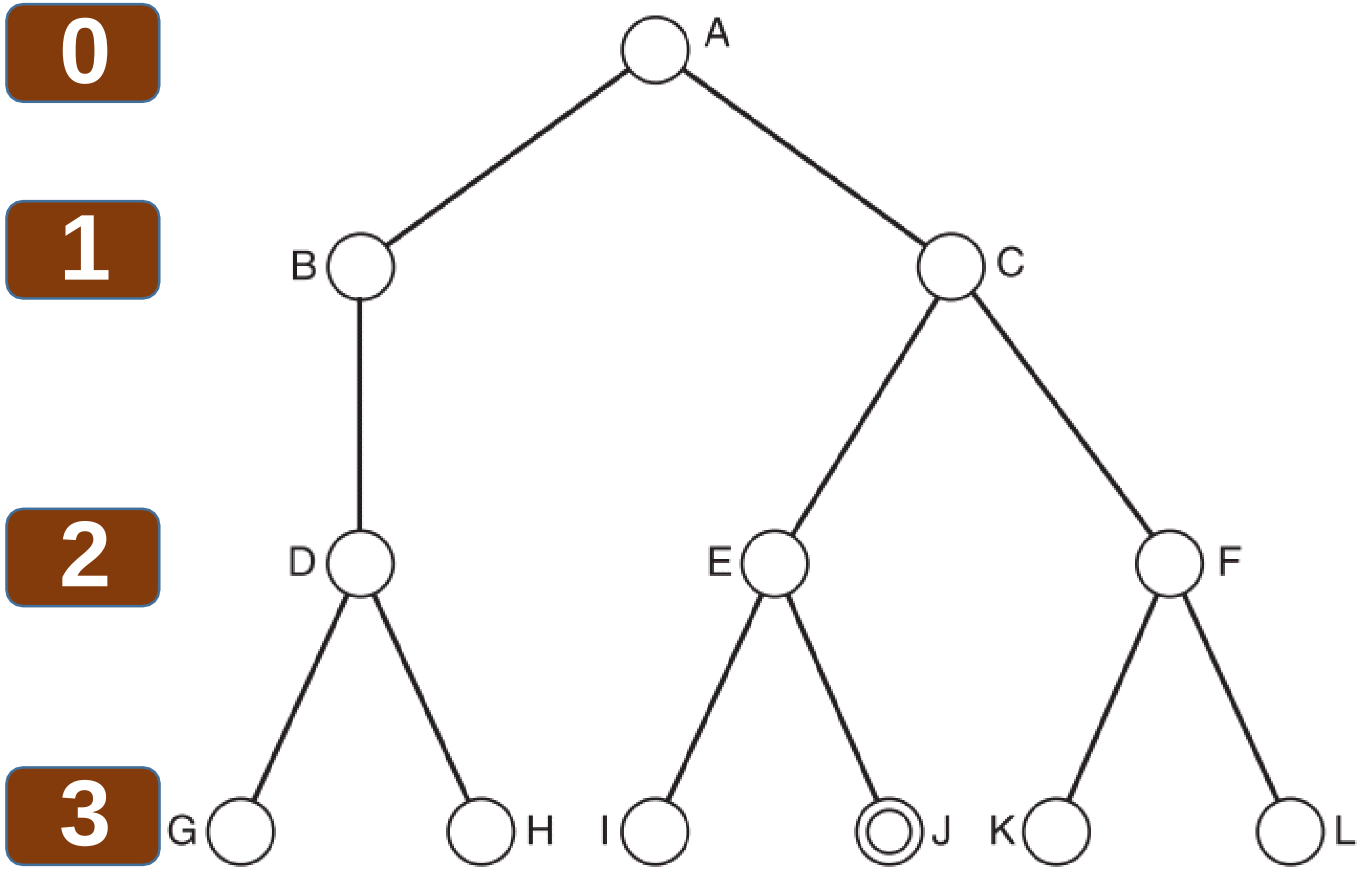
Iterative Deepening Search
Depth – 3, Goal – Node J

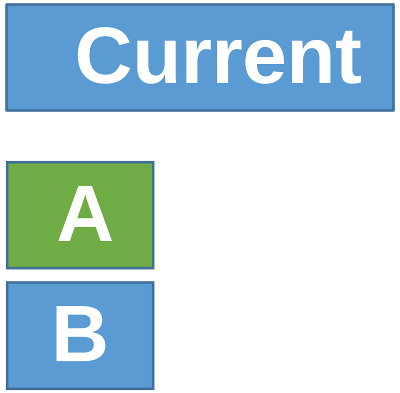
Current

A

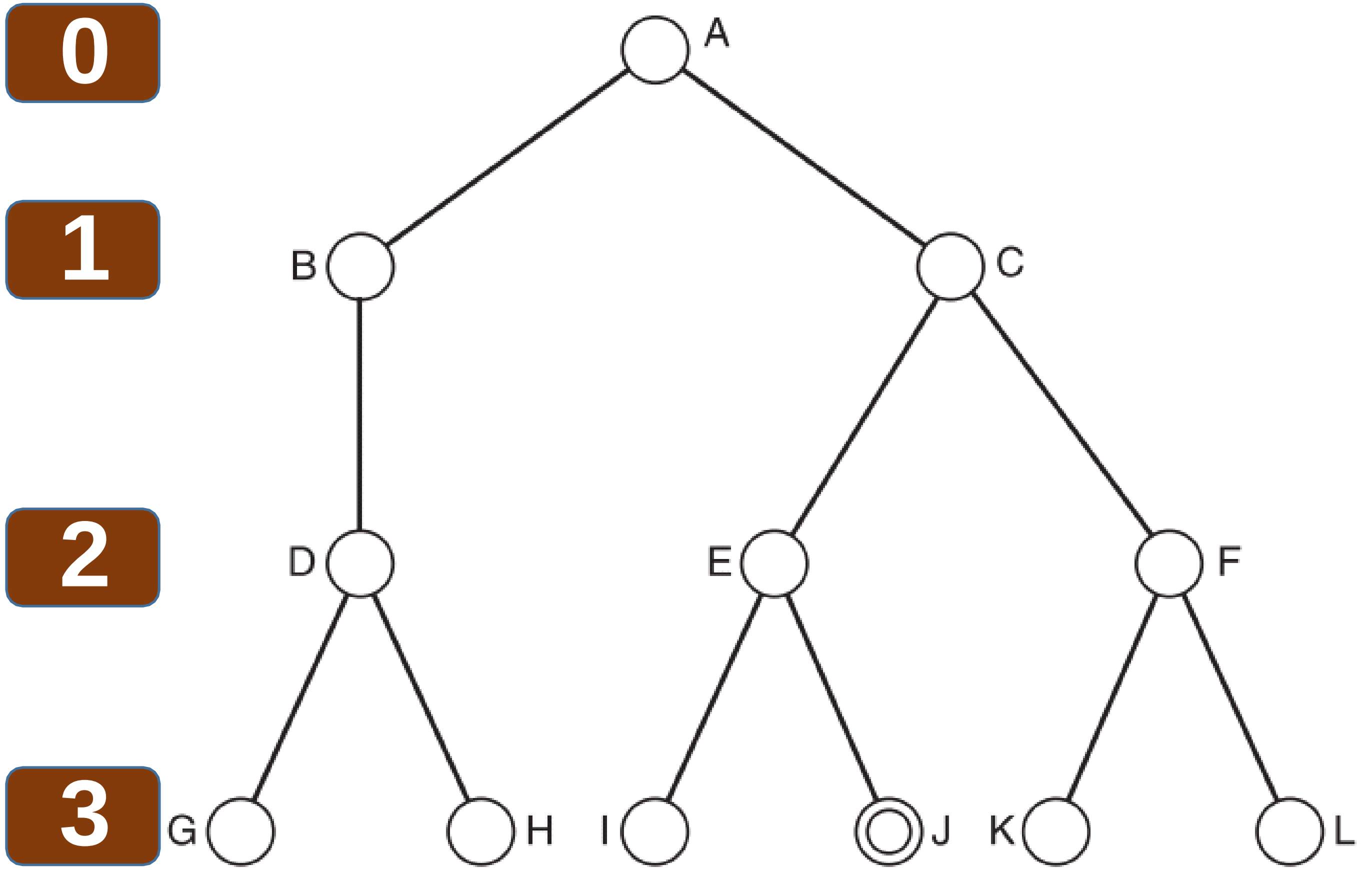


Iterative Deepening Search
Depth – 3, Goal – Node J





Iterative Deepening Search
Depth – 3, Goal – Node J

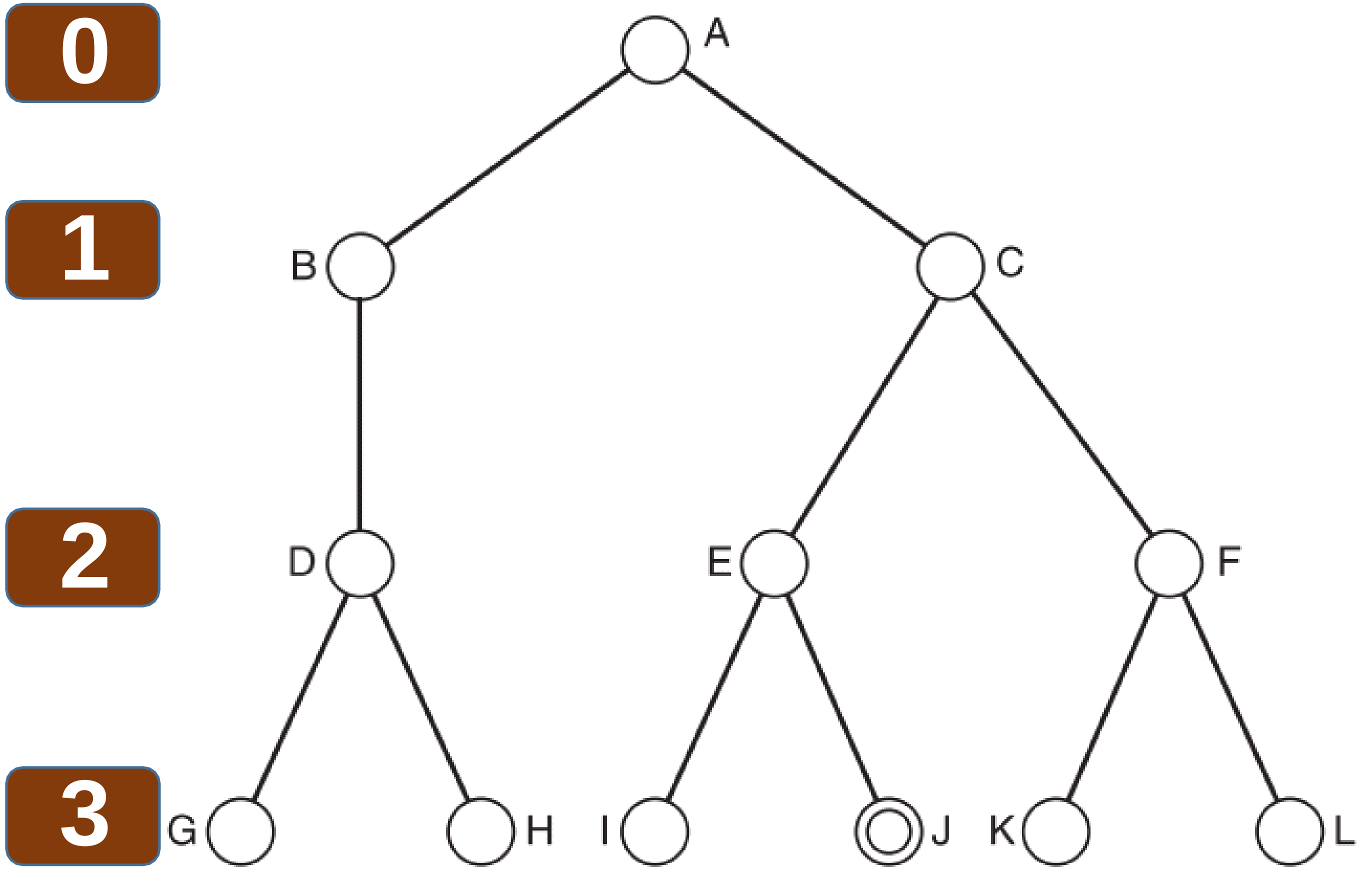


Current

A

B

Iterative Deepening Search
Depth – 3, Goal – Node J



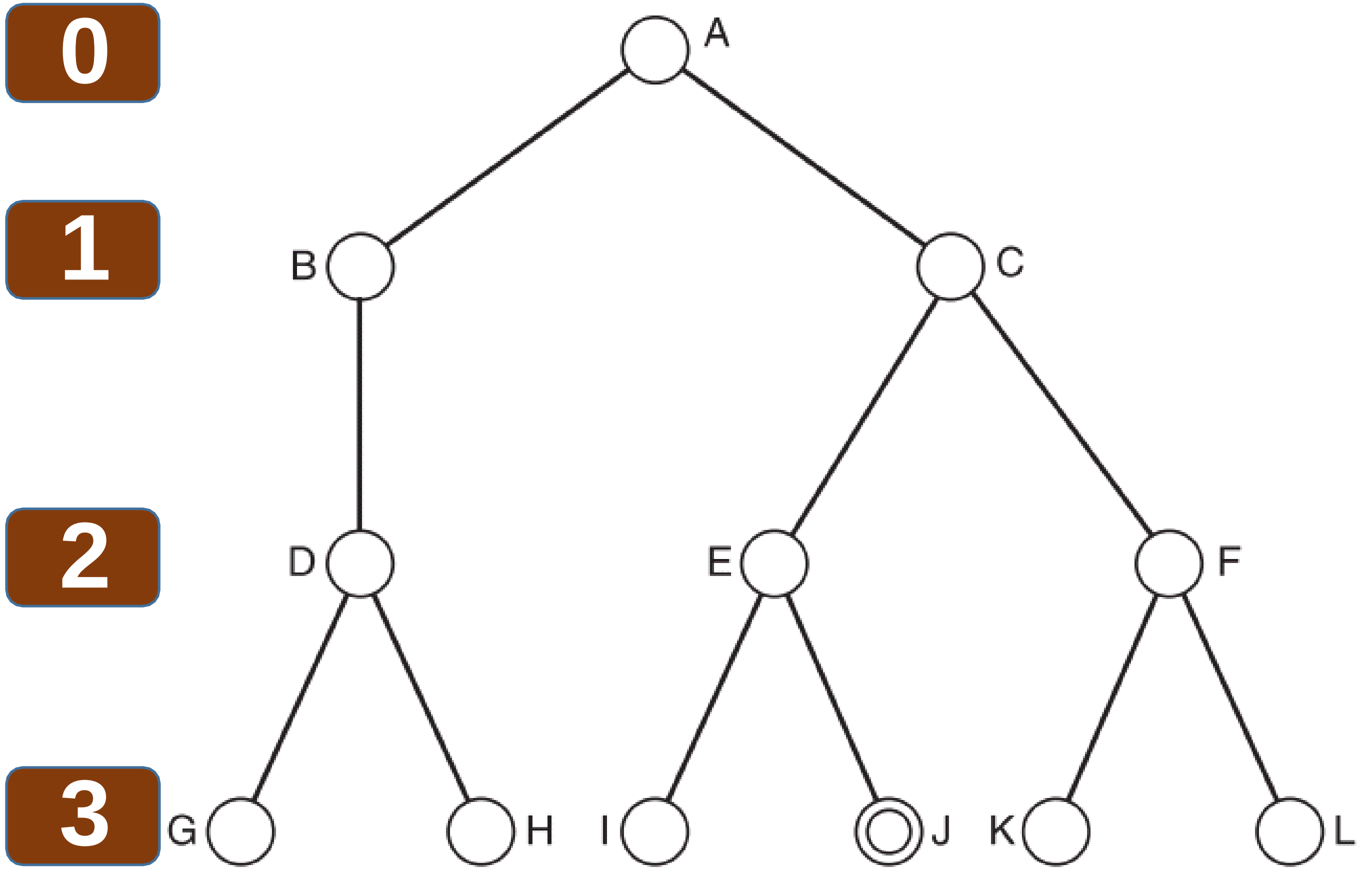
Current

A

B

D

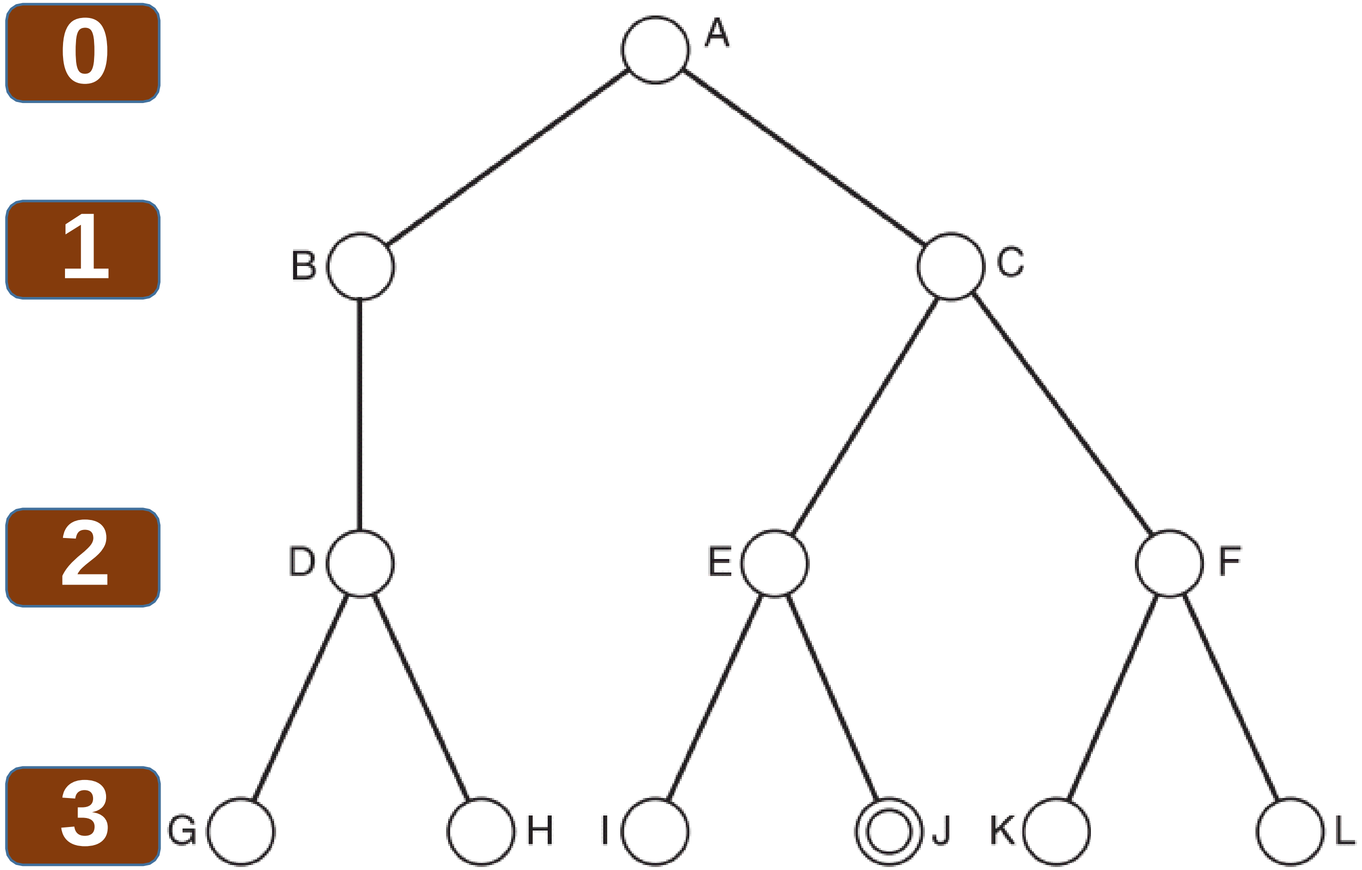
Iterative Deepening Search
Depth – 3, Goal – Node J



Current

A
B
D

Iterative Deepening Search
Depth – 3, Goal – Node J



Current

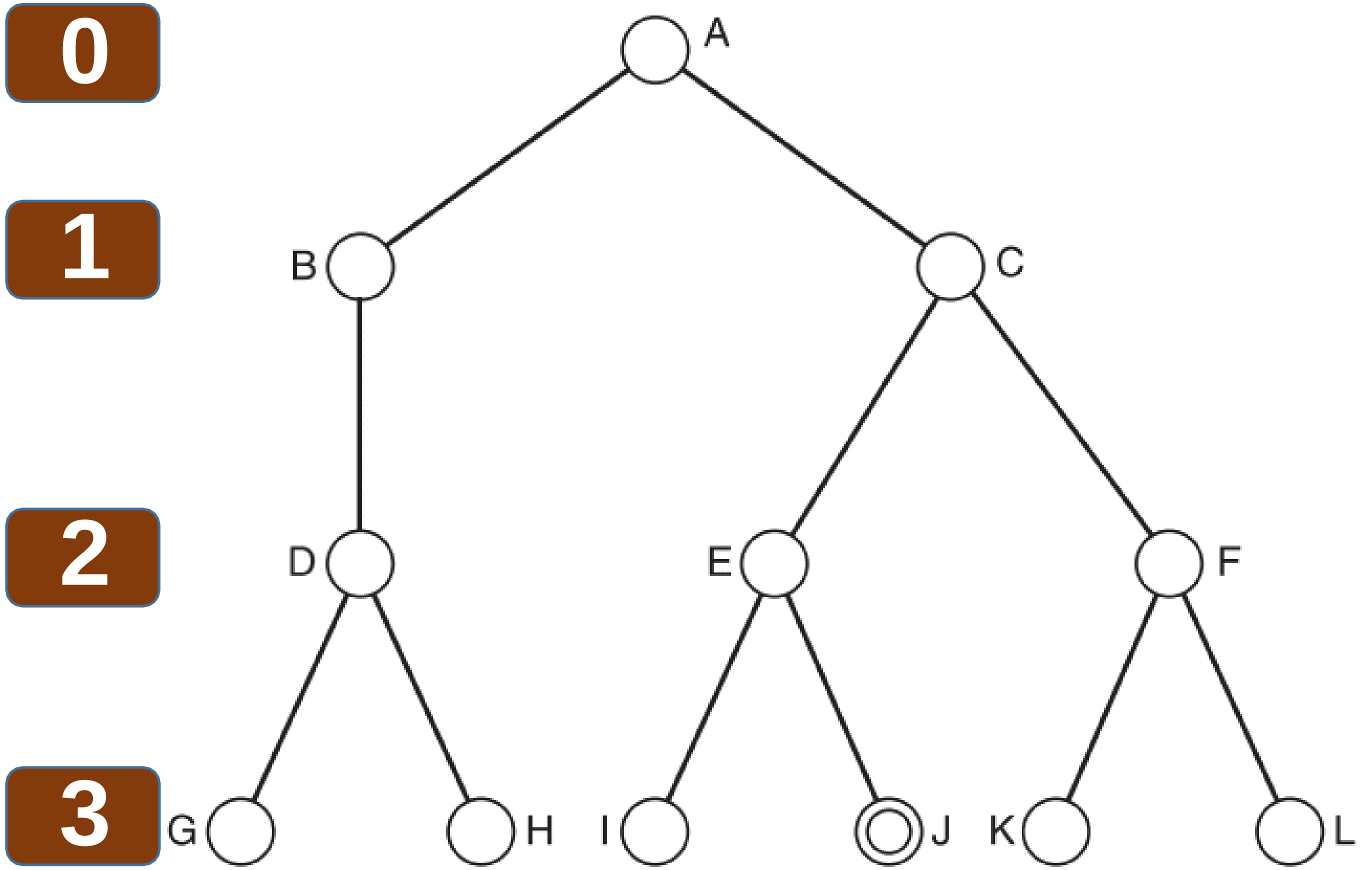
A

B

D

G

Iterative Deepening Search
Depth – 3, Goal – Node J



Current

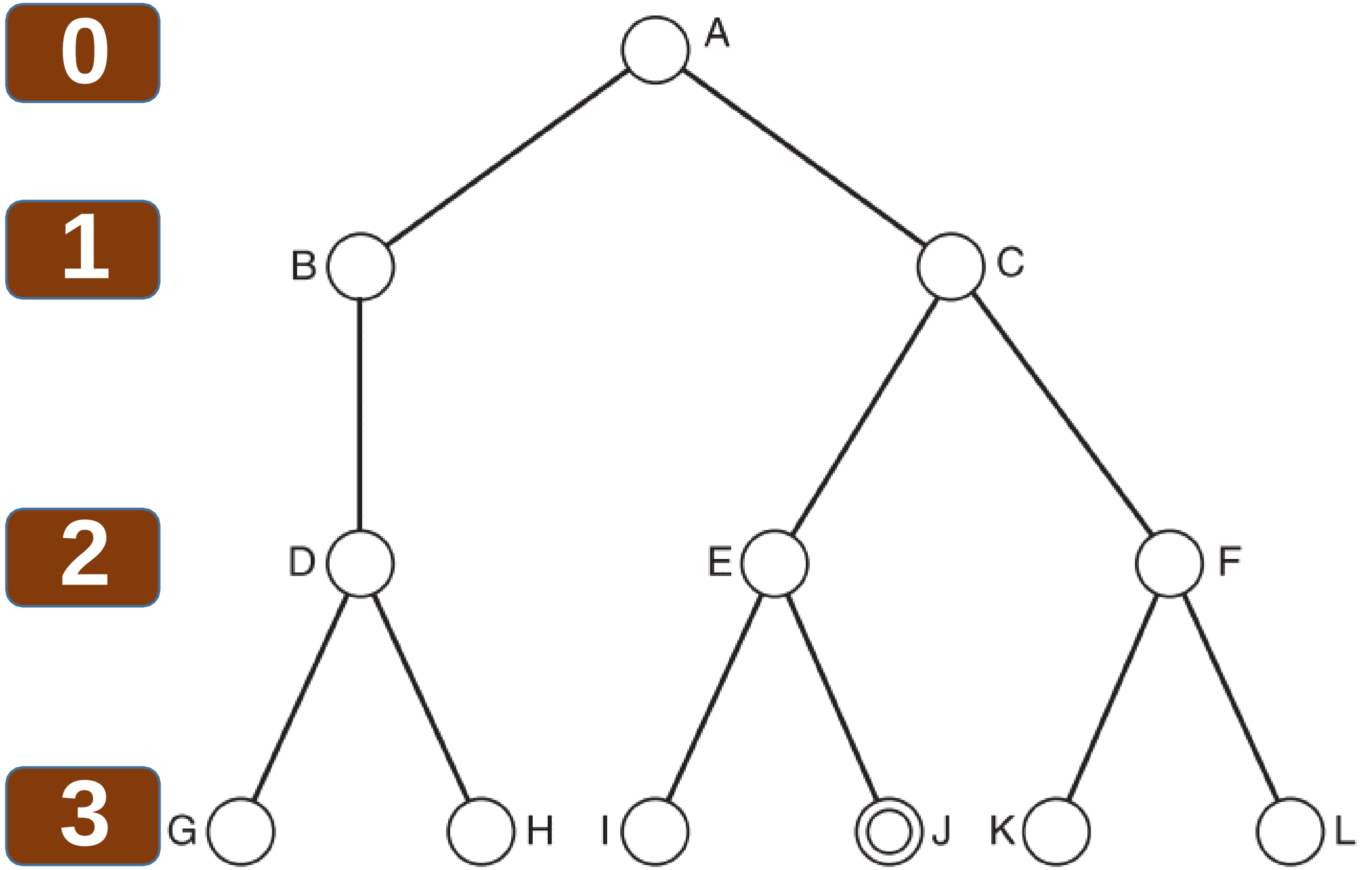
A

B

D

G

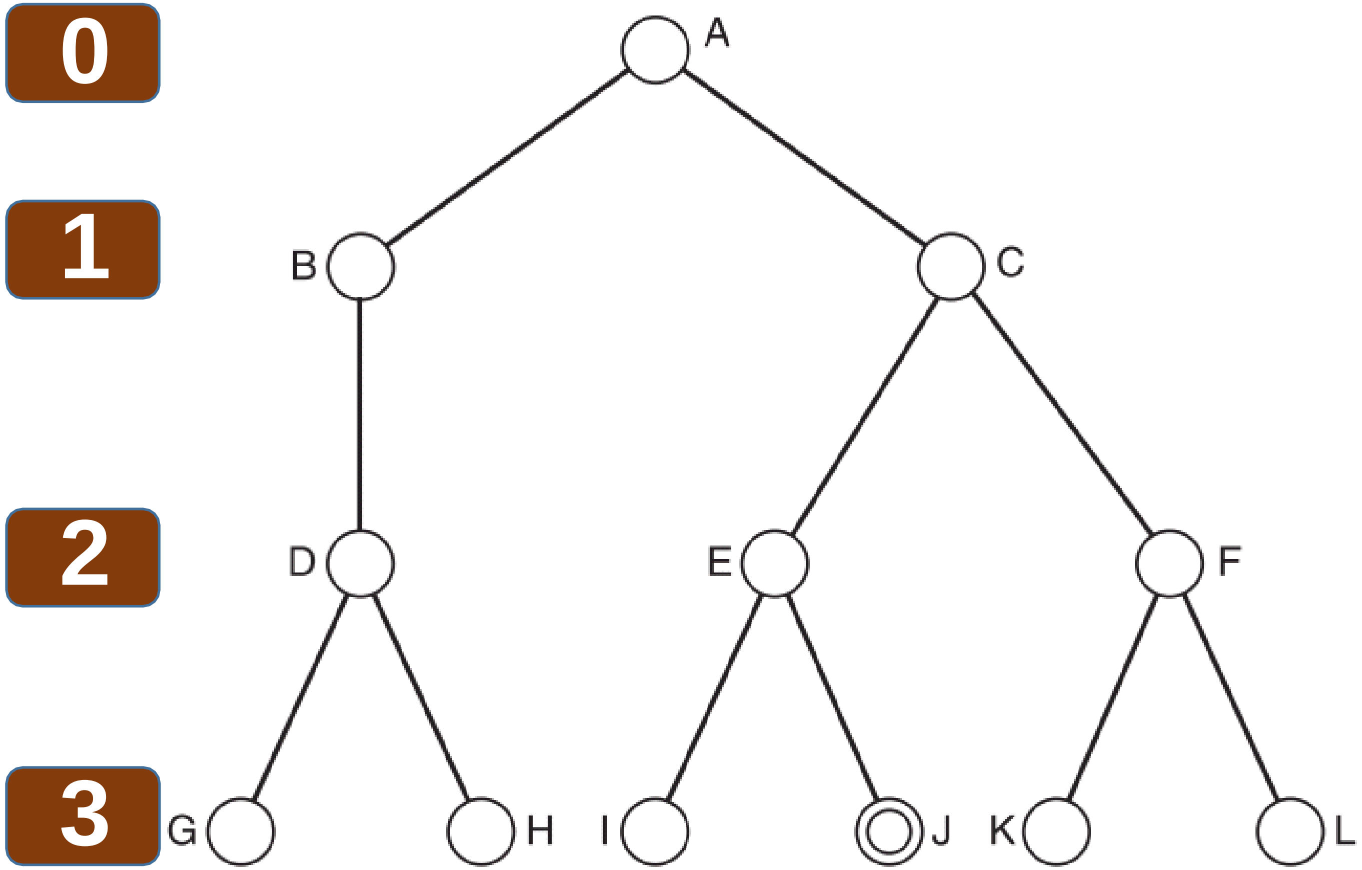
Iterative Deepening Search
Depth – 3, Goal – Node J

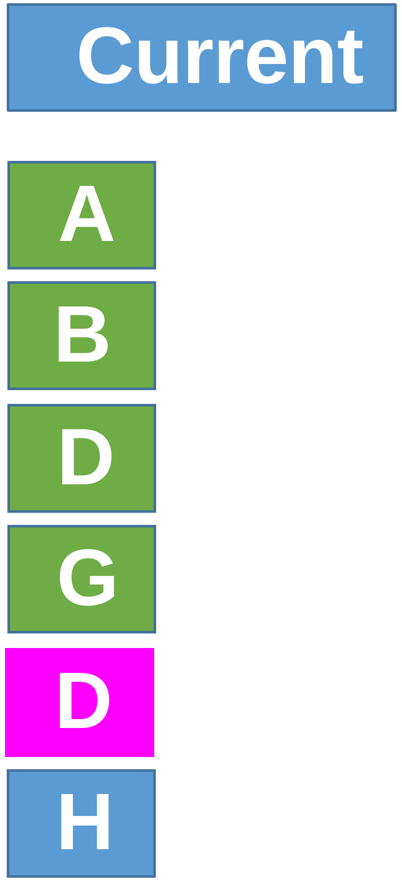


Current

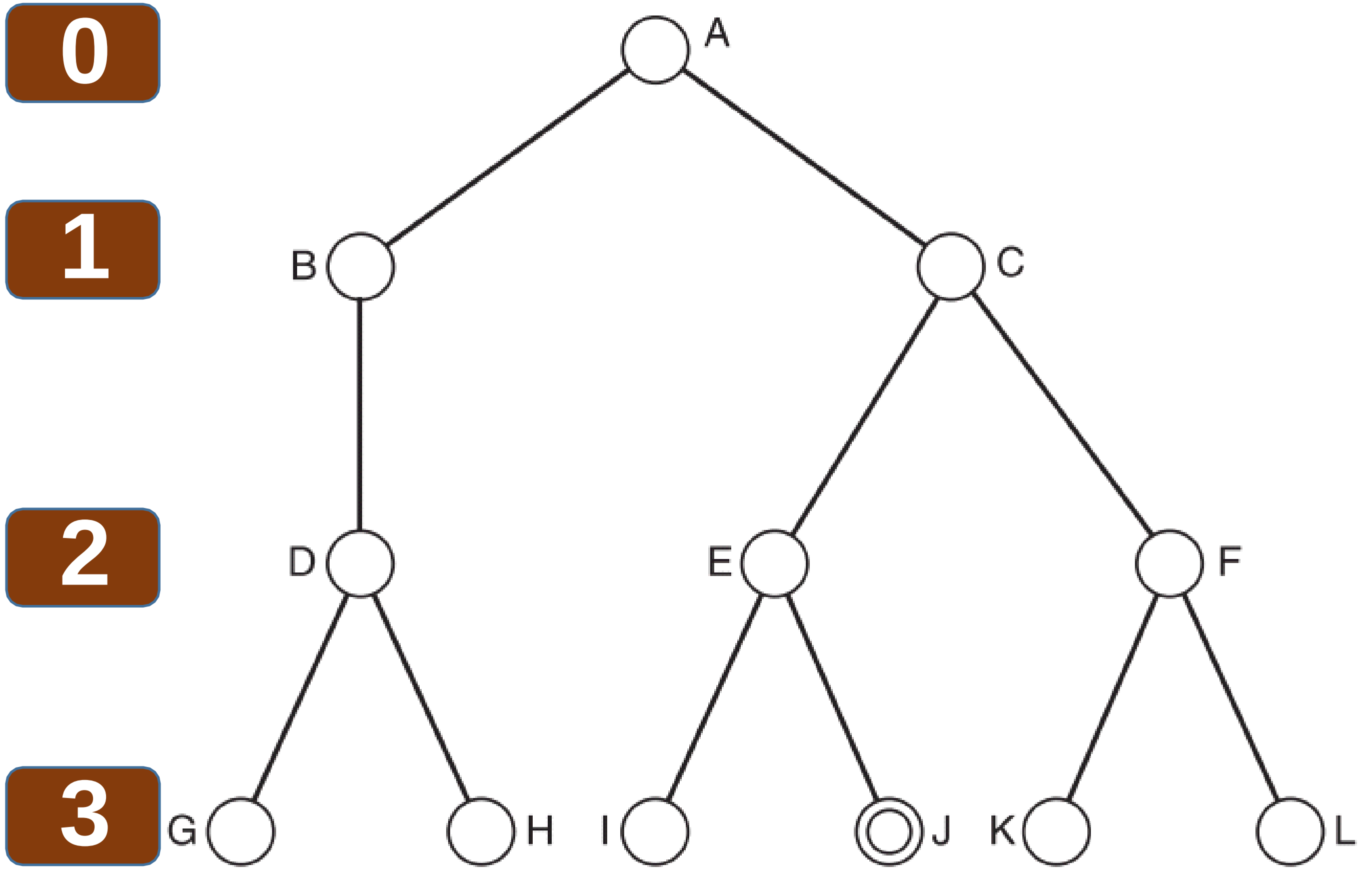
A
B
D
G
D

Iterative Deepening Search
Depth – 3, Goal – Node J





Iterative Deepening Search
Depth – 3, Goal – Node J



Current

A

B

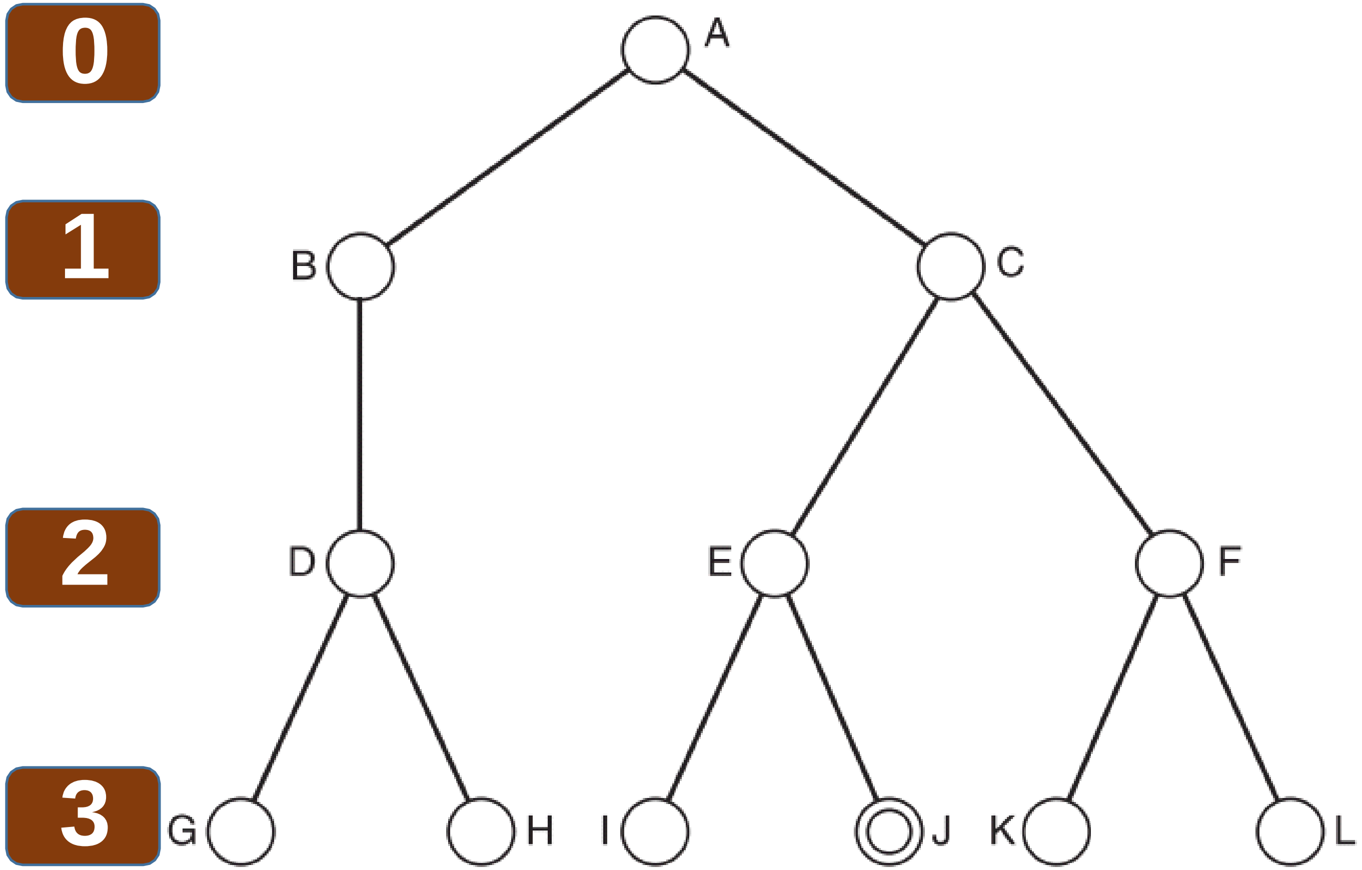
D

G

D

H

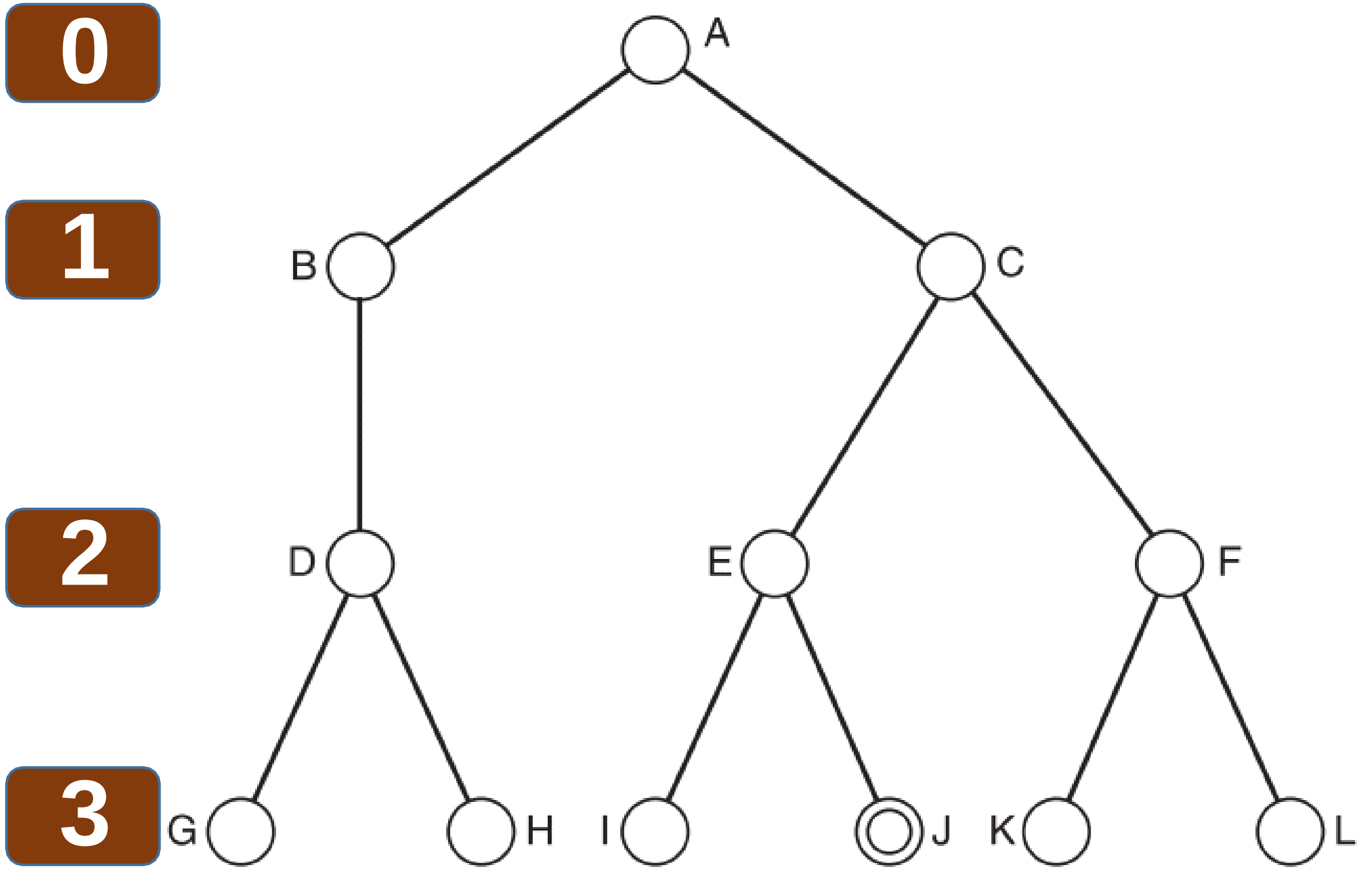
Iterative Deepening Search
Depth – 3, Goal – Node J

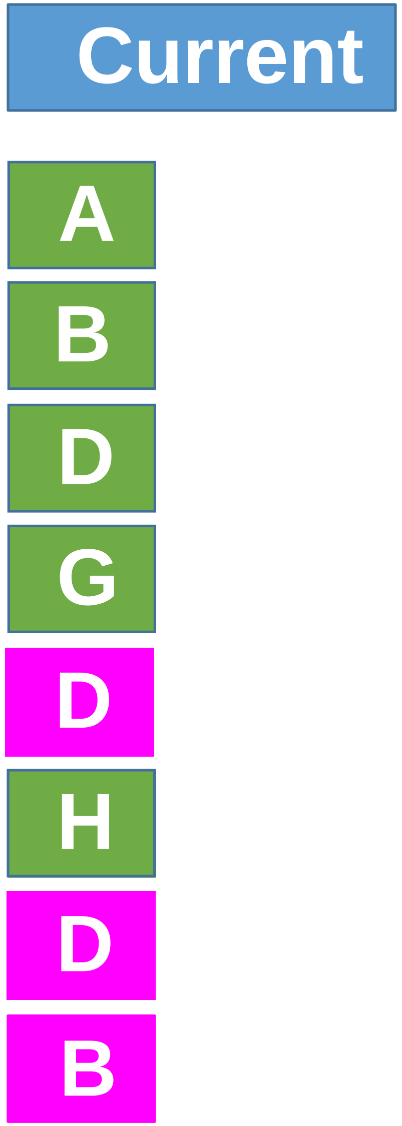


Current

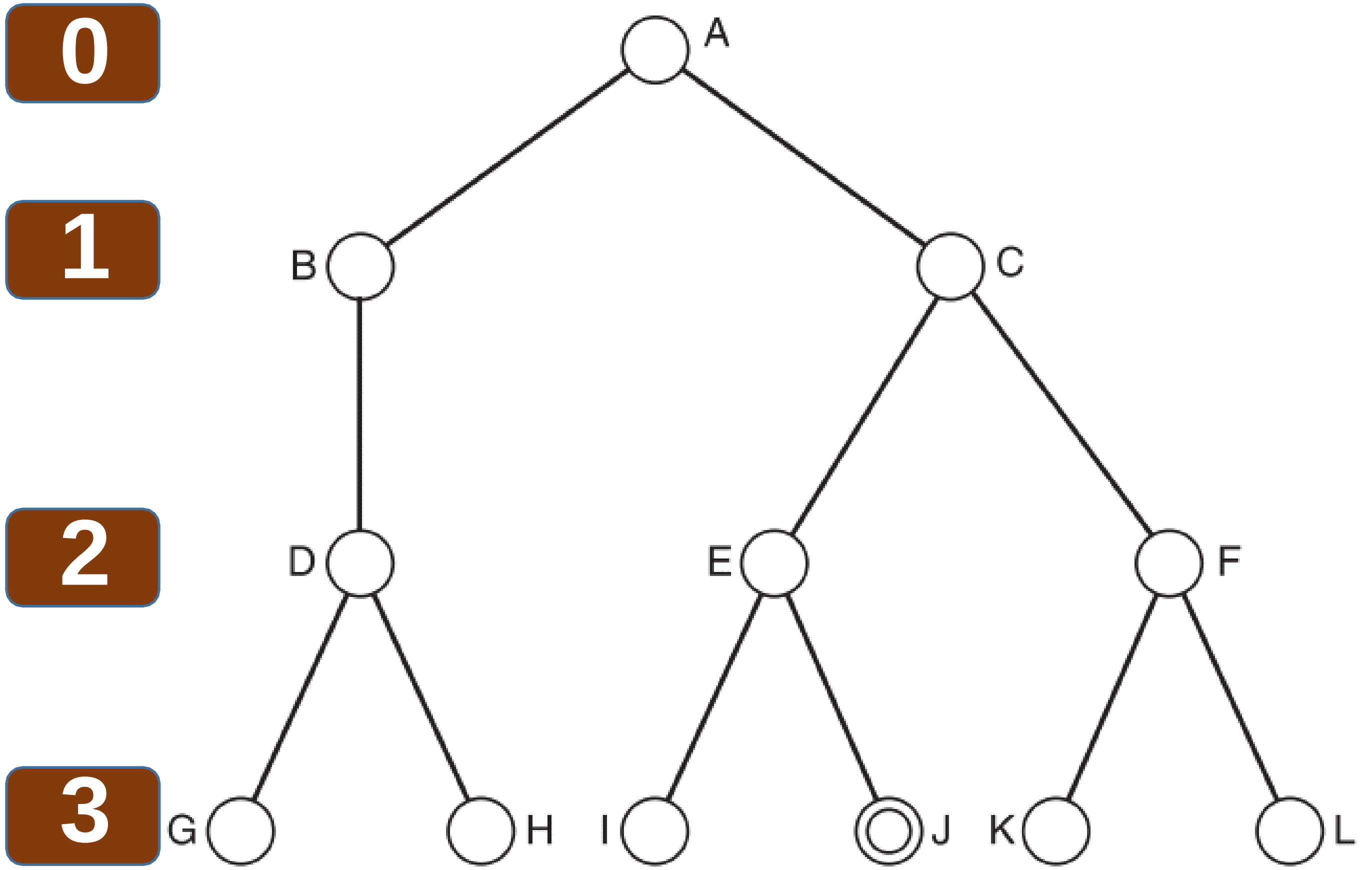
A
B
D
G
D
H
D

Iterative Deepening Search
Depth – 3, Goal – Node J





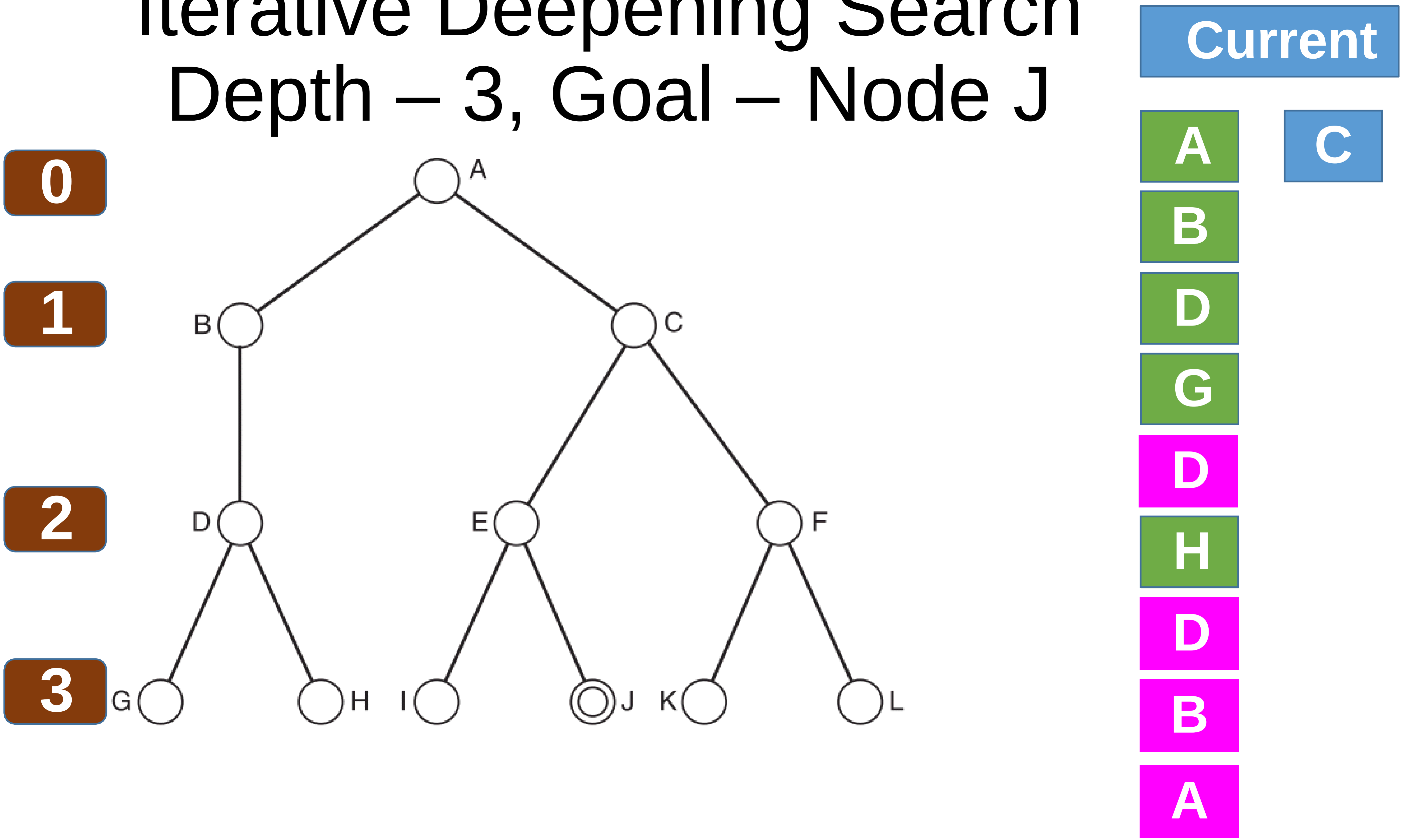
Iterative Deepening Search
Depth – 3, Goal – Node J



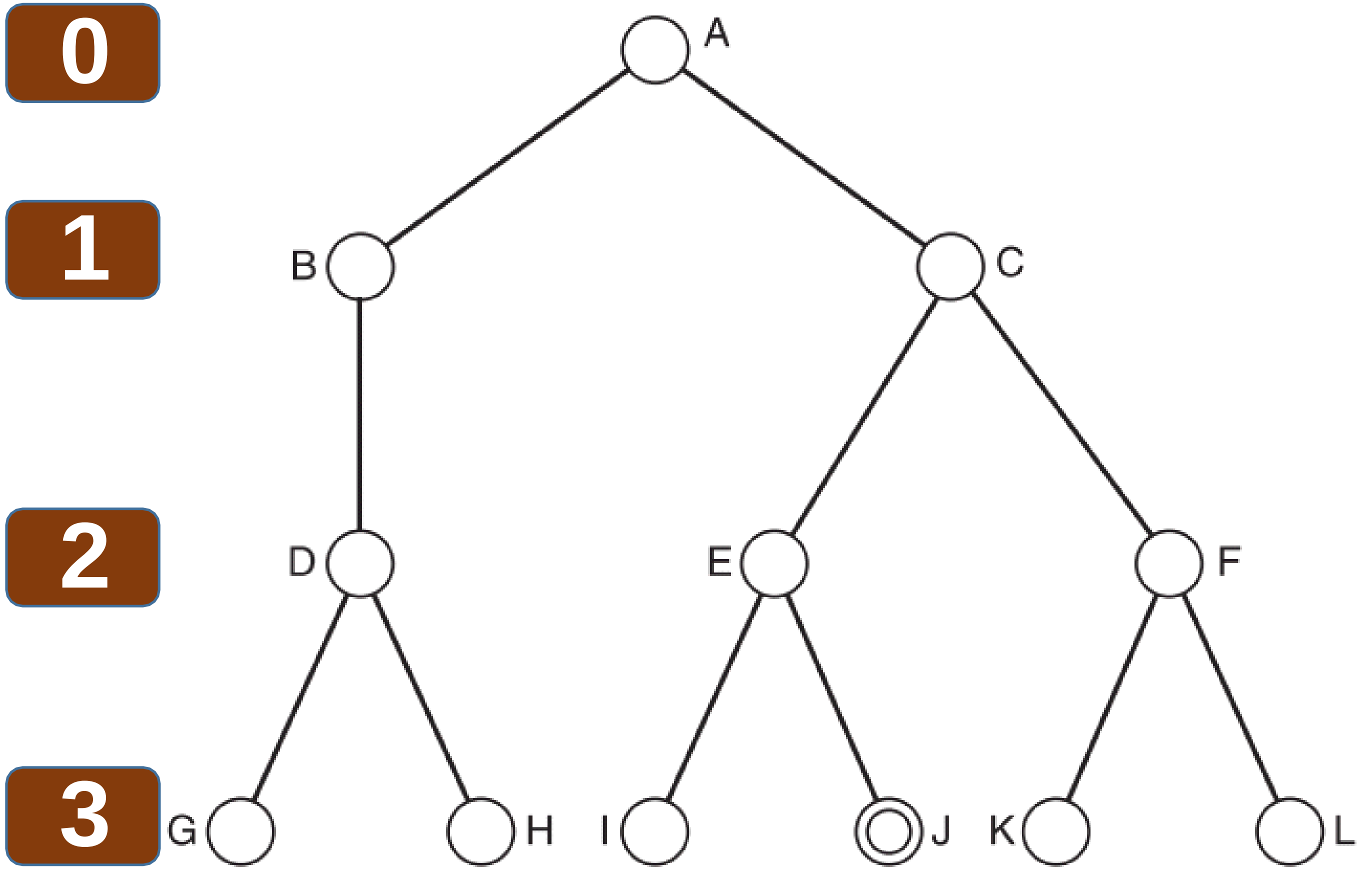
Current

A
B
D
G
D
H
D
B
A

Iterative Deepening Search
Depth – 3, Goal – Node J



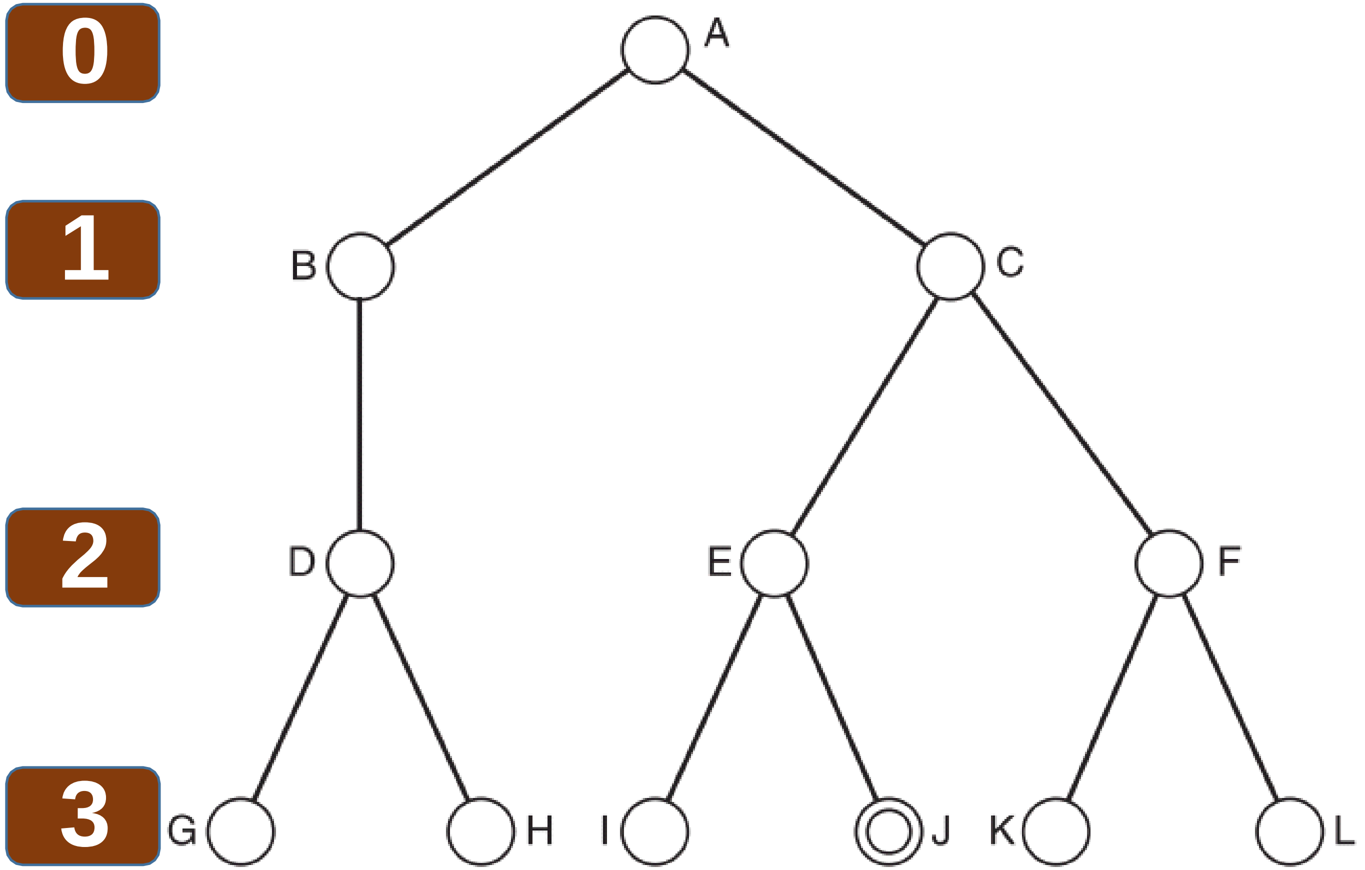
Iterative Deepening Search
Depth – 3, Goal – Node J



Current

A C
B
D
G
D
H
D
B
A

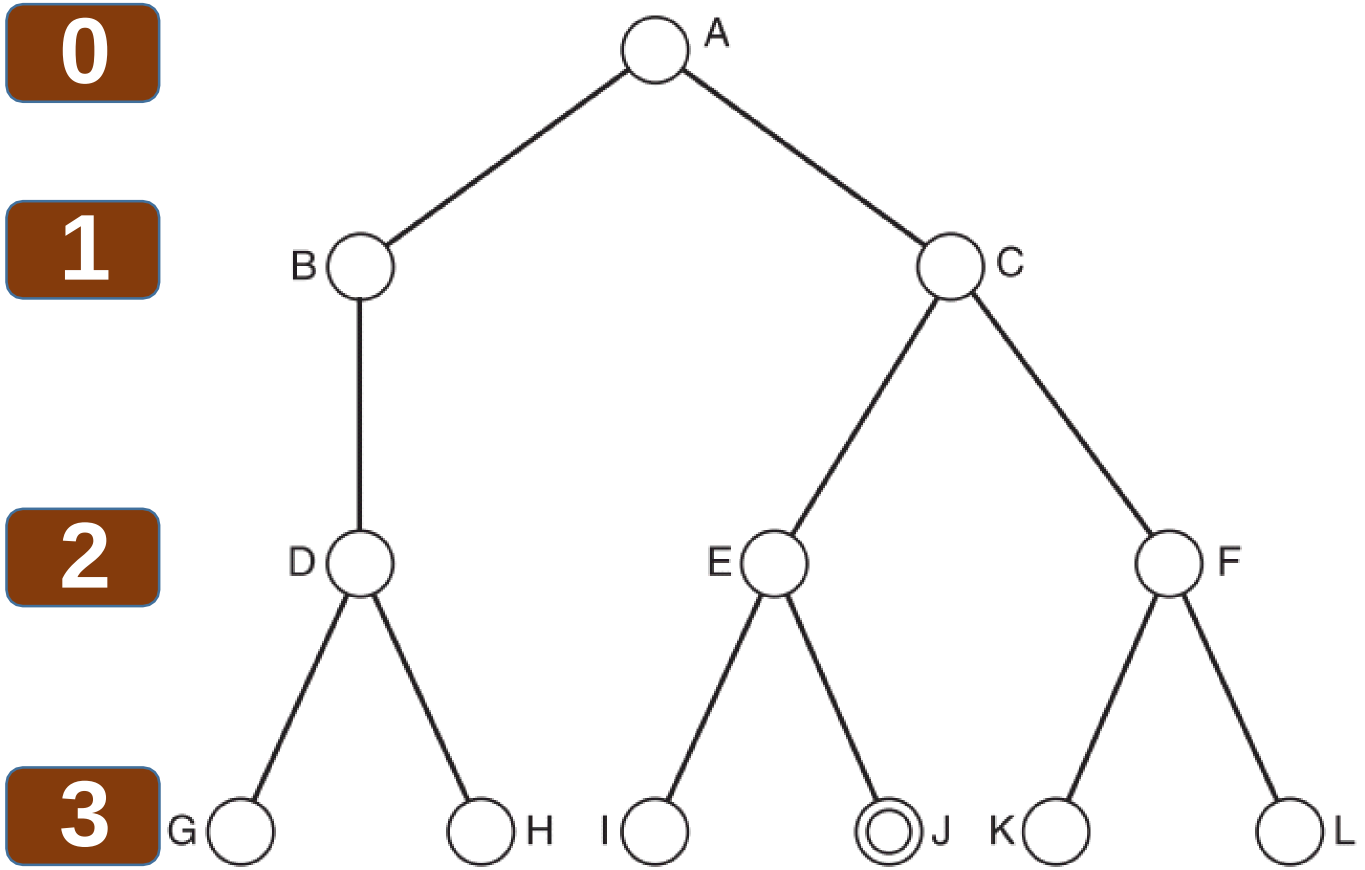
Iterative Deepening Search
Depth – 3, Goal – Node J



Current

A C
B E
D
G
D
H
D
B
A

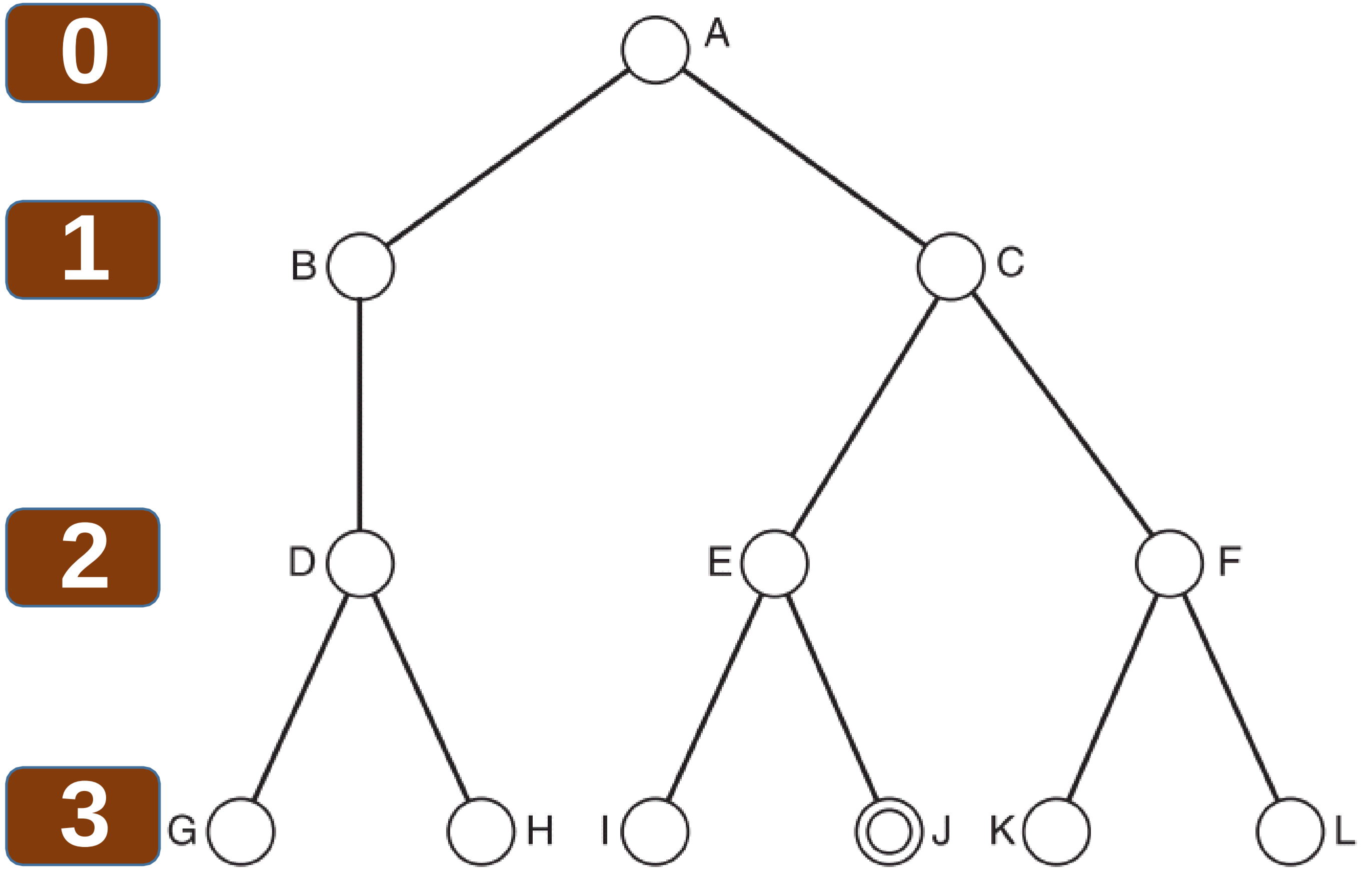
Iterative Deepening Search
Depth – 3, Goal – Node J



Current

A C
B E
D
G
D
H
D
B
A

Iterative Deepening Search
Depth – 3, Goal – Node J



Current

A
B
D
G
D
H
D
B
A

C
E
I

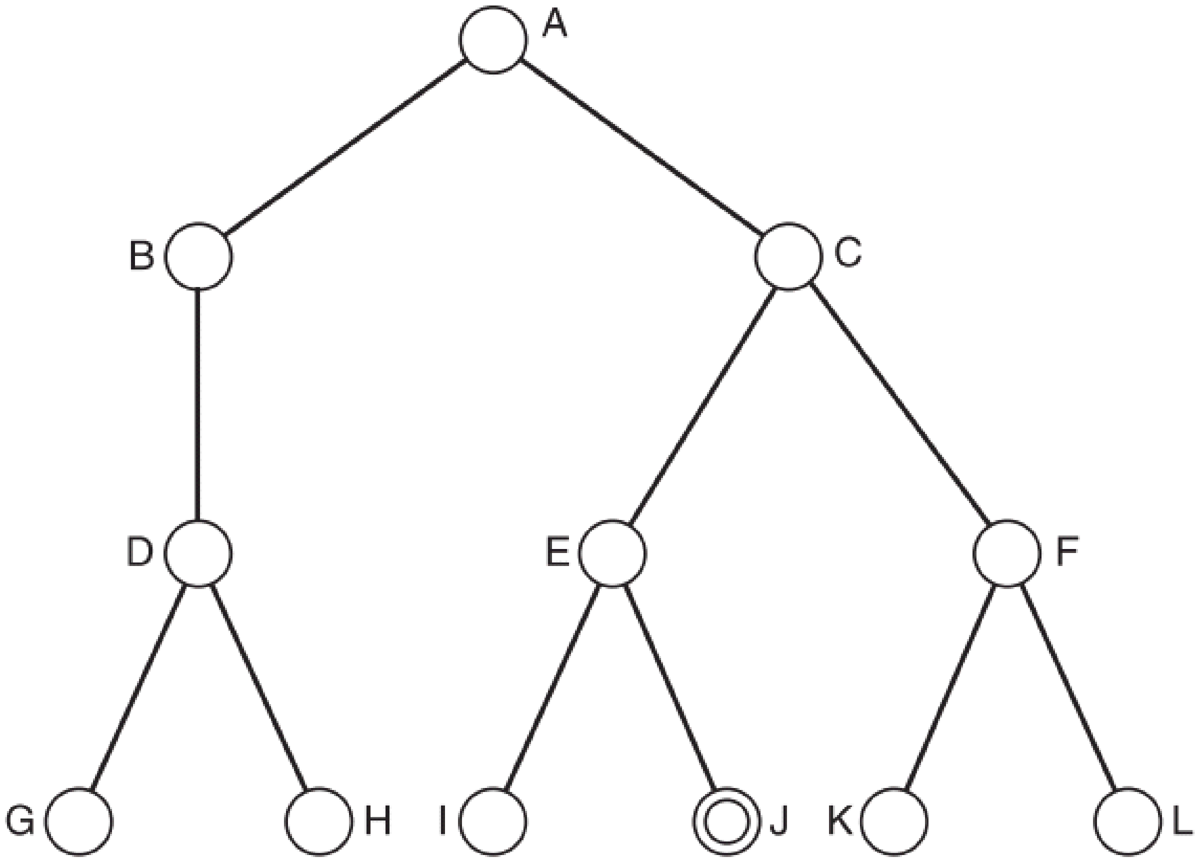
Iterative Deepening Search
Depth – 3, Goal – Node J

0

1

2

3

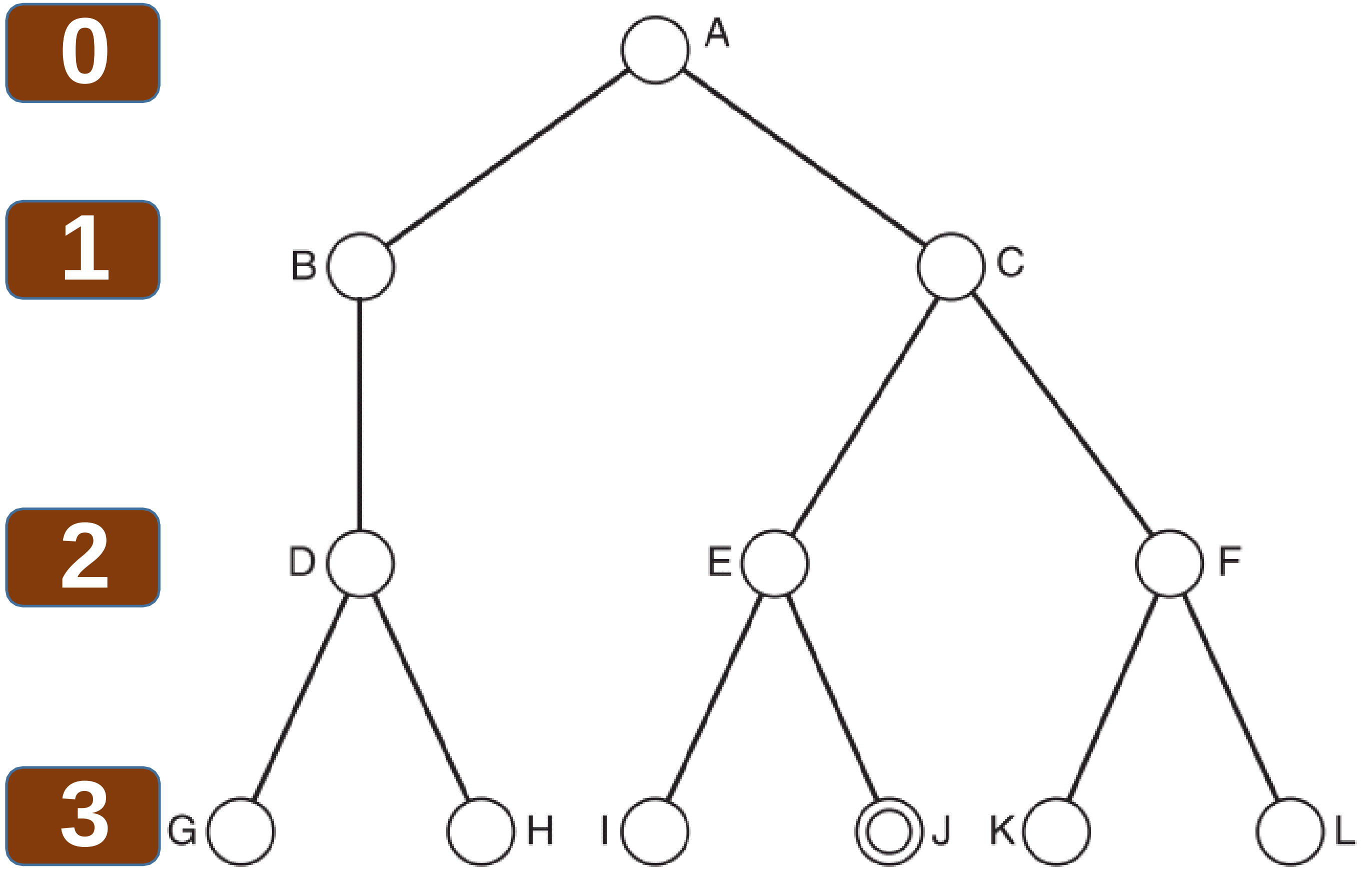


Current

A
B
D
G
D
H
D
B
A

C
E
I

Iterative Deepening Search
Depth – 3, Goal – Node J

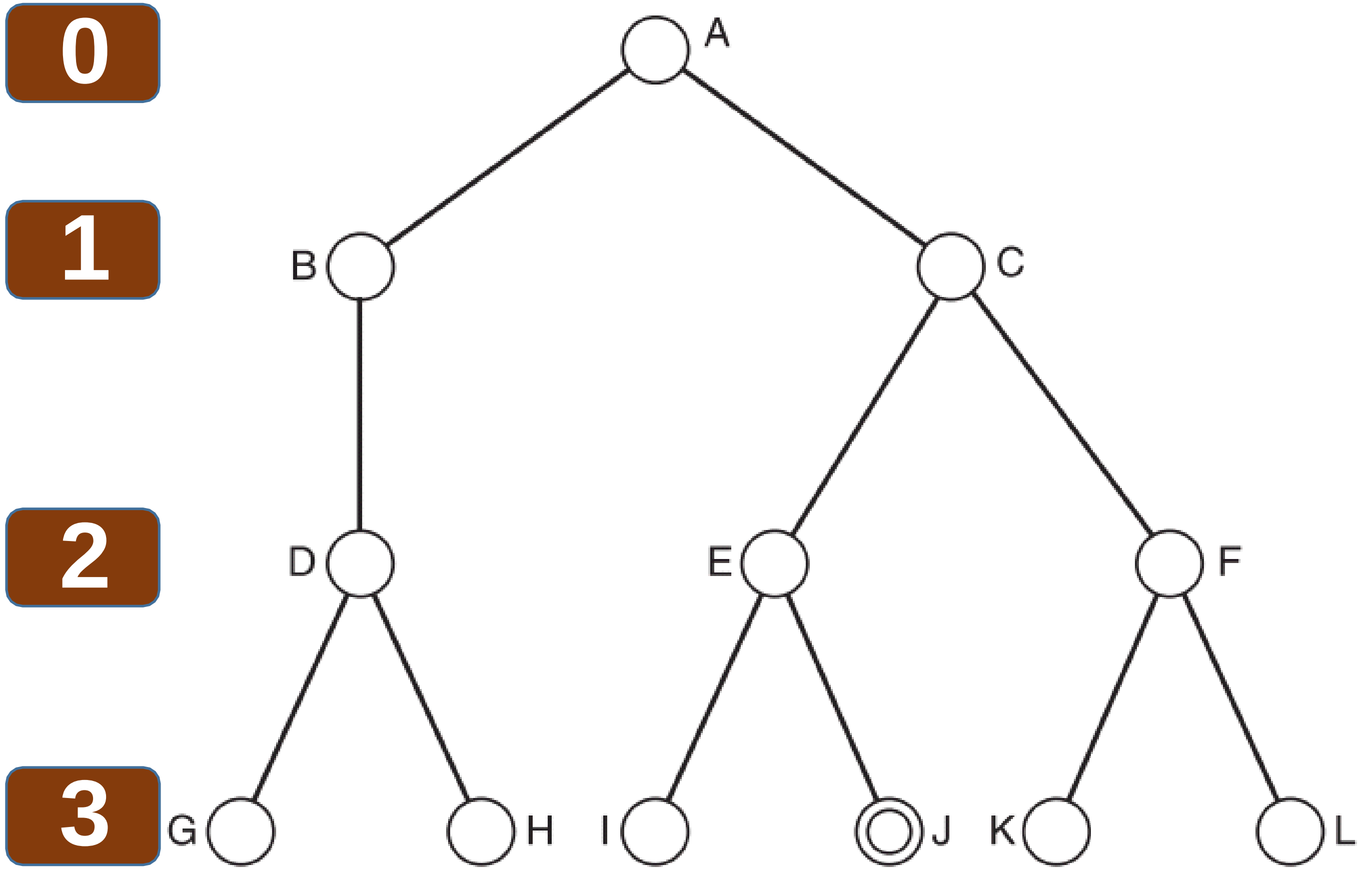


Current

A
B
D
G
D
H
D
B
A

C
E
I
E

Iterative Deepening Search
Depth – 3, Goal – Node J

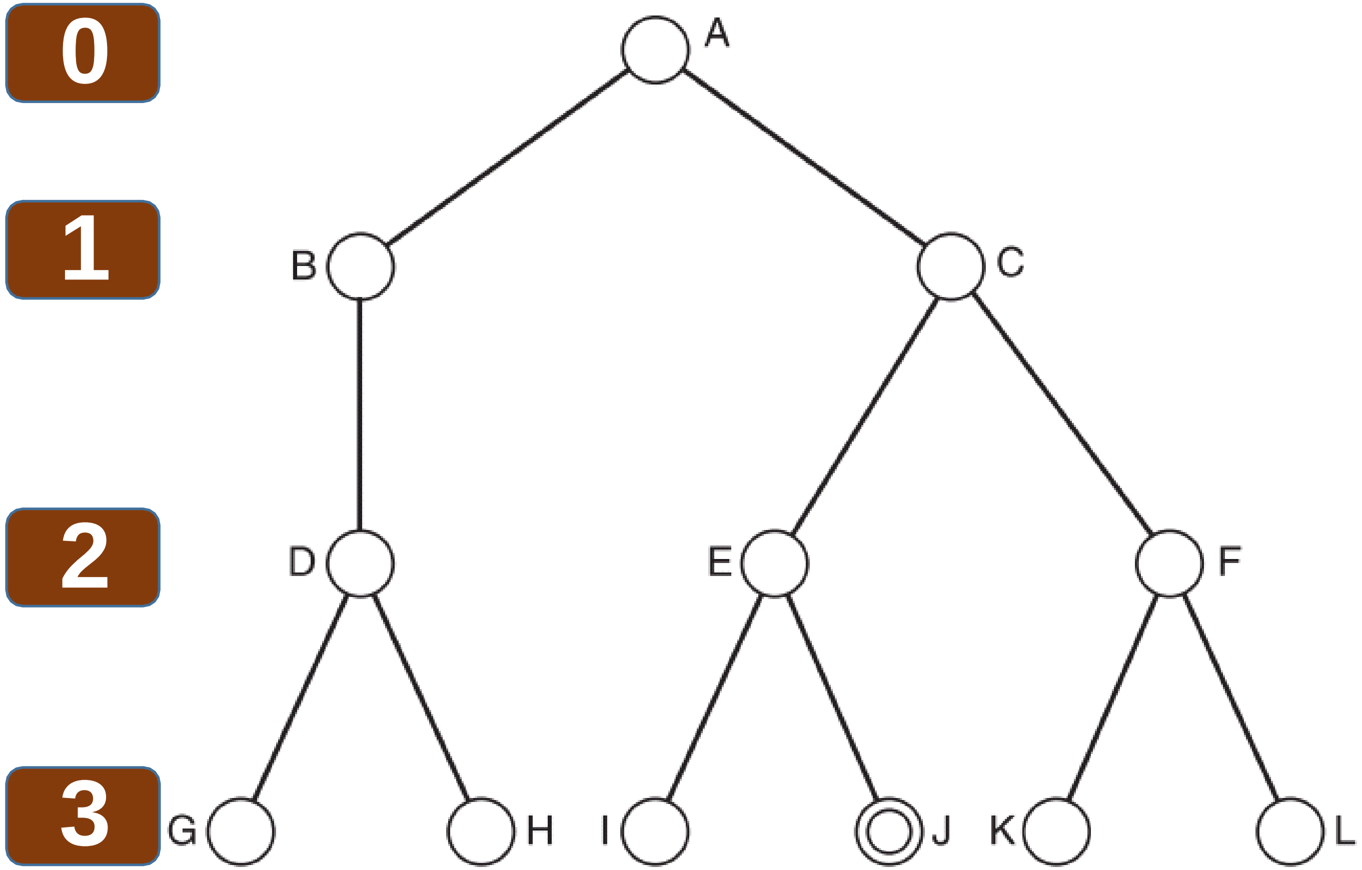


Current

A
B
D
G
D
H
D
B
A

C
E
I
E
J

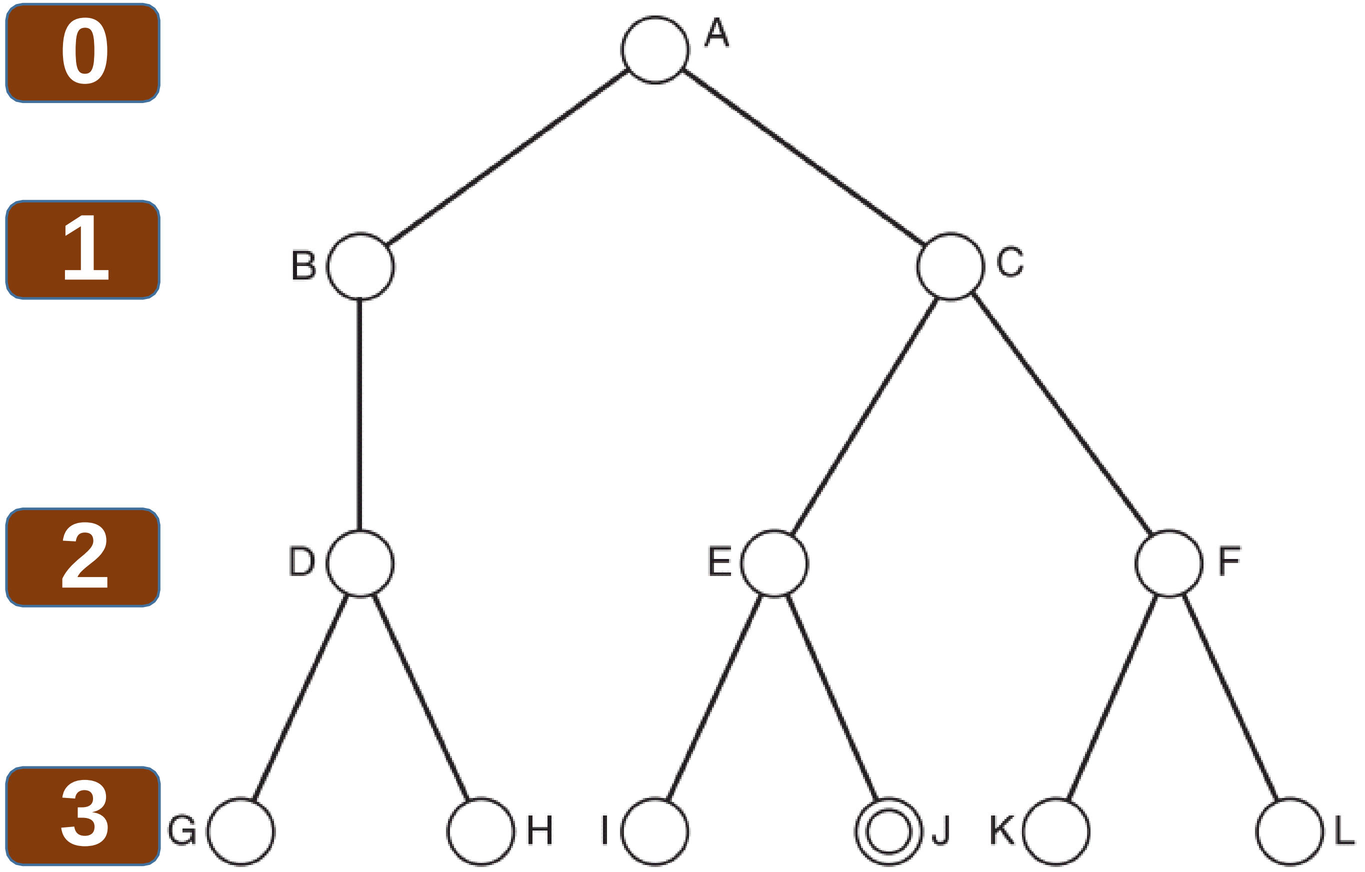
Iterative Deepening Search
Depth – 3, Goal – Node J



Current

A	C
B	E
D	I
G	E
D	J
H	
D	
B	
A	

Iterative Deepening Search
Depth – 3, Goal – Node J



Current

A B D G D H D B A

C E I E J

GOAL

```
function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution, or failure
  for depth = 0 to ∞ do
    result ← DEPTH-LIMITED-SEARCH(problem, depth)
    if result ≠ cutoff then return result
```

Figure 3.18 The iterative deepening search algorithm, which repeatedly applies depth-limited search with increasing limits. It terminates when a solution is found or if the depth-limited search returns *failure*, meaning that no solution exists.

Iterative Deepening Search

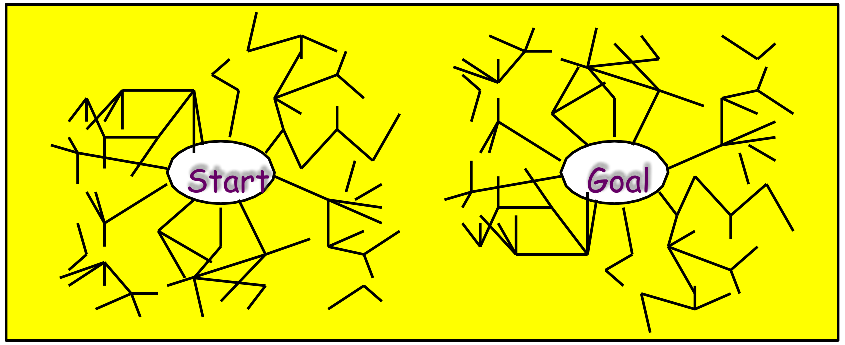
Combines the best of breadth-first and depth-first search strategies.

- Completeness: Yes,
- Time complexity: $O(b^d)$
- Space complexity: $O(bd)$
- Optimality: Yes, if step cost = 1

Properties of Iterative Deepening Search

- Complete? Yes (b finite)
- Time? $d\, b^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$
- Space? $O(bd)$
- Optimal? Yes, if step costs identical

Bidirectional Search



Both search forward from initial state, and backwards from goal.

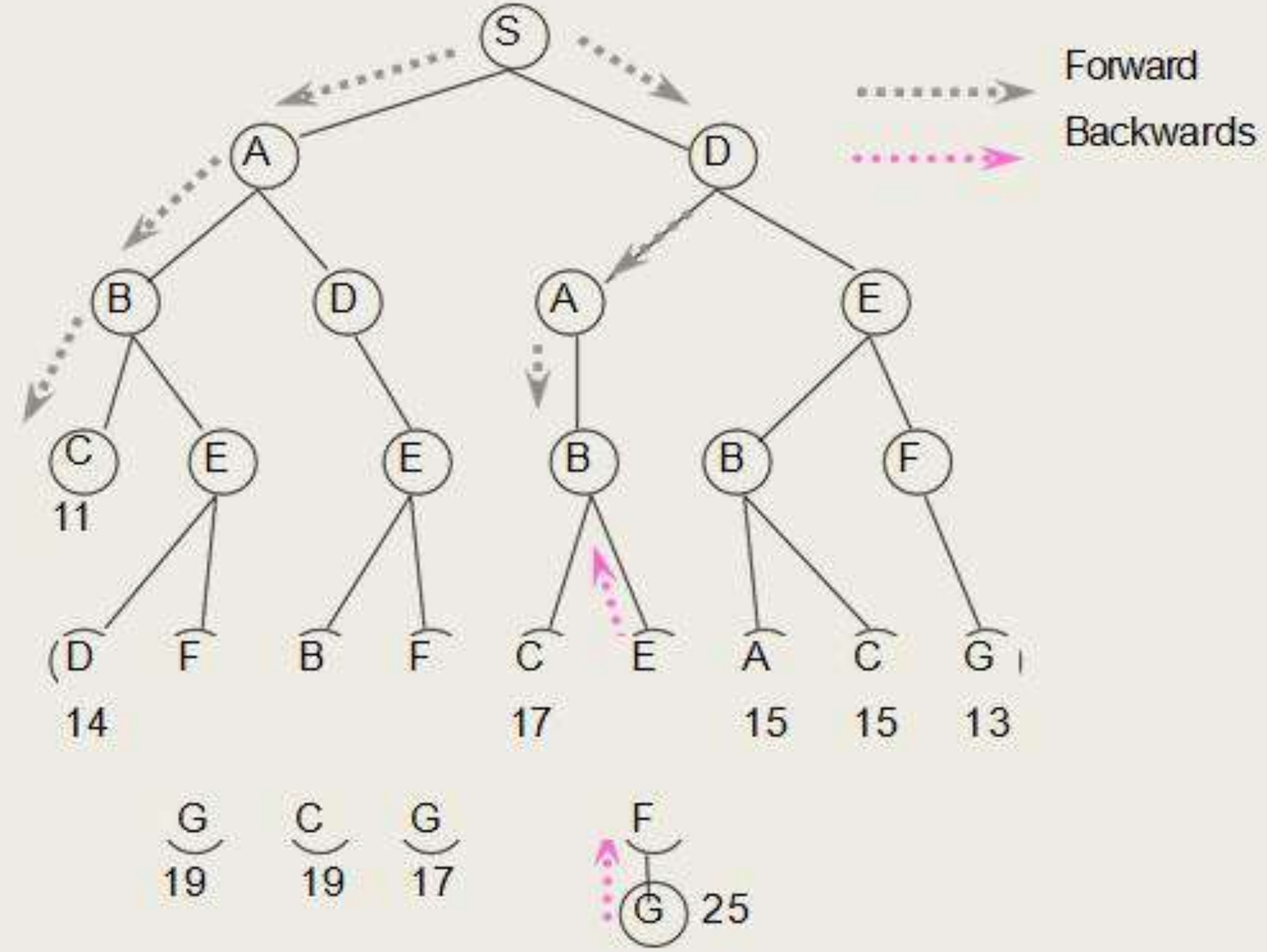
Stop when the two searches meet in the middle.

Motivation: $b^{d/2} + b^{d/2}$ is much less than b^d

Implementation

Replace the goal test with a check to see whether the frontiers of the two searches intersect, if yes → solution is found

Bi directional Search



Bidirectional Search

- Not always optimal, even if both searches are BFS
- Check when each node is expanded or selected for expansion
- Can be implemented using BFS or iterative deepening (but at least one frontier needs to be kept in memory)
- Significant weakness

  Space requirement
- Time Complexity is good

Bidirectional Search

- Problem: how do we search backwards from goal??
  - *predecessor of node n = all nodes that have n as successor*
  - *this may not always be easy to compute!*
  - *if several goal states, apply predecessor function to them just as we applied successor (only works well if goals are explicitly known; may be difficult if goals only characterized implicitly).*
  - *for bidirectional search to work well, there must be an efficient way to check whether a given node belongs to the other search tree.*
  - *select a given search algorithm for each half.*

Bidirectional Search

- Completeness: Yes,
- Time complexity: $2 * O(b^{d/2}) = O(b^{d/2})$
- Space complexity: $O(b^{m/2})$
- Optimality: Yes

- To avoid one by one comparison, we need a hash table of size $O(b^{m/2})$
- *If hash table is used, the cost of comparison is O(1)*

Comparison Uninformed Search Strategies

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes^{a}	Yes^{a,b}	No	No	Yes^{a}	Yes^{a,d}
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes^{c}	Yes	No	No	Yes^{c}	Yes^{c,d}

Figure 3.21 Evaluation of tree-search strategies. b is the branching factor; d is the depth of the shallowest solution; m is the maximum depth of the search tree; l is the depth limit. Superscript caveats are as follows: ^{a} complete if b is finite; ^{b} complete if step costs $\geq \epsilon$ for positive ϵ; ^{c} optimal if step costs are all identical; ^{d} if both directions use breadth-first search.

Unit 2

Knowledge Based Agent

- Central Component of Knowledge Base Agent is **Knowledge Base**
- A knowledge base is a **set of sentences**.
- Each sentence is expressed in a language called a **knowledge representation language**.
- **TELL -** Add new sentences to the knowledge base
- **ASK -** Query what is known.
- The agent maintains a **knowledge base, KB**, which may **initially** contain **some background knowledge.**

Examples of sentences

The moon is made of green cheese
If A is true then B is true
A is false
All humans are mortal
Confucius is a human

Knowledge Based Agent

Each time the agent program is called, it does three things.

- ➢**TELLs** the knowledge base what it perceives.
- ➢**ASKs** the knowledge base what action it should perform.
- ➢The agent program **TELLs** the knowledge base which action was chosen, and the agent executes the action.

- ➢**MAKE-PERCEPT-SENTENCE** constructs a sentence asserting that the agent perceived the given percept at the given time.
- ➢**MAKE-ACTION-QUERY** constructs a sentence that asks what action should be done at the current time.
- ➢**MAKE-ACTION-SENTENCE** constructs a sentence asserting that the chosen action was executed.

Knowledge Based Agent Algorithm

function KB-AGENT(*percept*) **returns** an *action*
  persistent: *KB*, a knowledge base
        t, a counter, initially 0, indicating time

  TELL(*KB*, MAKE-PERCEPT-SENTENCE(*percept*, *t*))
  action ← ASK(*KB*, MAKE-ACTION-QUERY(*t*))
  TELL(*KB*, MAKE-ACTION-SENTENCE(*action*, *t*))
  $t \leftarrow t + 1$
  return *action*

Figure 7.1 A generic knowledge-based agent. Given a percept, the agent adds the percept to its knowledge base, asks the knowledge base for the best action, and tells the knowledge base that it has in fact taken that action.

Architecture of a knowledge-based agent

Knowledge Level.

The most abstract level: describe agent by saying what it knows.

Example: A taxi agent might know that the Golden Gate Bridge connects San Francisco with the Marin County.

Logical Level.

The level at which the knowledge is encoded into sentences.

Example: Links(GoldenGateBridge, SanFrancisco, MarinCounty).

Implementation Level.

The physical representation of the sentences in the logical level.

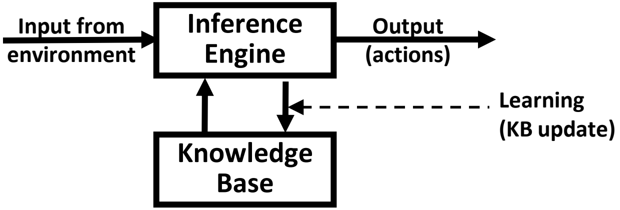
Example: `‘(links goldengatebridge sanfrancisco marincounty)`

➢The **Inference engine** derives new sentences from the input and KB

➢The inference mechanism depends on representation in KB

➢The agent operates as follows:

1. It receives percepts from environment
2. It computes what action it should perform (by IE and KB)
3. It performs the chosen action (some actions are simply inserting inferred new facts into KB).



The <u>Wumpus World</u> environment

- The Wumpus computer game
- The agent explores a cave consisting of rooms connected by passageways.
- Lurking somewhere in the cave is **the Wumpus**, a beast that <u>eats any agent</u> that enters its room.
- Some rooms contain bottomless pits that *trap any agent* that wanders into the room.
- Occasionally, there is a heap of gold in a room.
- **The goal is:**

To collect the gold and exit the world without being eaten

Agent in a Wumpus world: Percepts

The agent perceives

a stench in the square containing the wumpus and in the adjacent squares (not diagonally)

a breeze in the squares adjacent to a pit

a glitter in the square where the gold is

a bump, if it walks into a wall

a woeful scream everywhere in the cave, if the wumpus is killed

The percepts will be given as a five-symbol list:

If there is a stench, and a breeze, but no glitter, no bump, and no scream, the percept is

[Stench, Breeze, None, None, None]

The agent can not perceive its own location.

The actions of the agent in Wumpus game are:

go forward
turn right 90 degrees
turn left 90 degrees
grab means pick up an object that is in the same square as the agent
shoot means fire an arrow in a straight line in the direction the agent is looking.

The arrow continues until it either hits and kills the wumpus or hits the wall. The agent has only one arrow. Only the first shot has any effect.

climb is used to leave the cave.

Only effective in start field.

die, if the agent enters a square with a pit or a live wumpus.

(No take-backs!)

The agent's goal

The agent's goal is to find the gold and bring it back to the start as quickly as possible, without getting killed.

1000 points reward for climbing out of the cave with the gold

1 point deducted for every action taken

10000 points penalty for getting killed

Wumpus World description

Performance measure gold +1000, death -1000

-1 per step, -10 for using the arrow Environment

Squares adjacent to wumpus are smelly

Squares adjacent to pit are breezy Glitter iff gold is in the same square Shooting kills wumpus if you are facing it

The wumpus kills you if in the same square

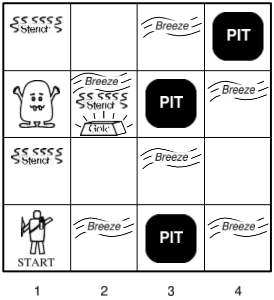
Shooting uses up the only arrow

Grabbing picks up gold if in same square

Releasing drops the gold in same square

ActuatorsLeft turn, Right turn, Forward, Grab,Release,Shoot,Climb

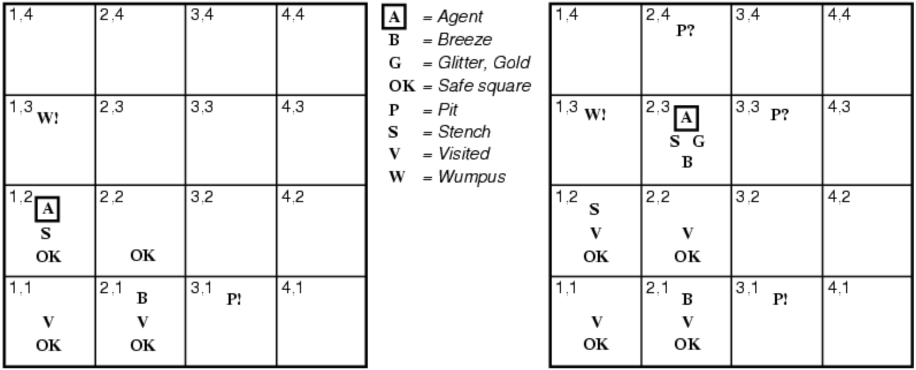
SensorsBreeze,Glitter,Stench,Bump, Scream



The Wumpus agent's first step

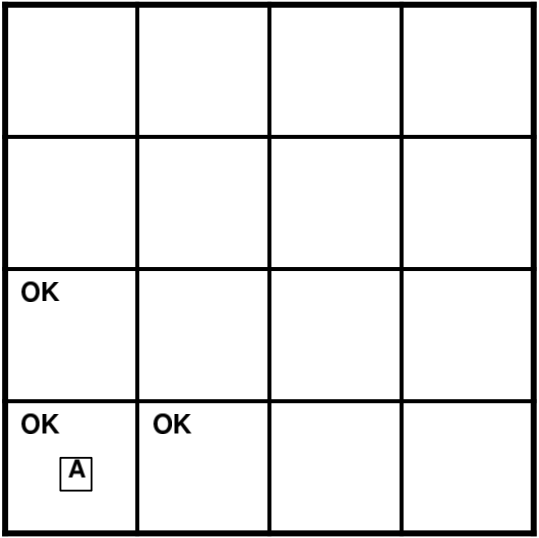
(a) (b)

Later

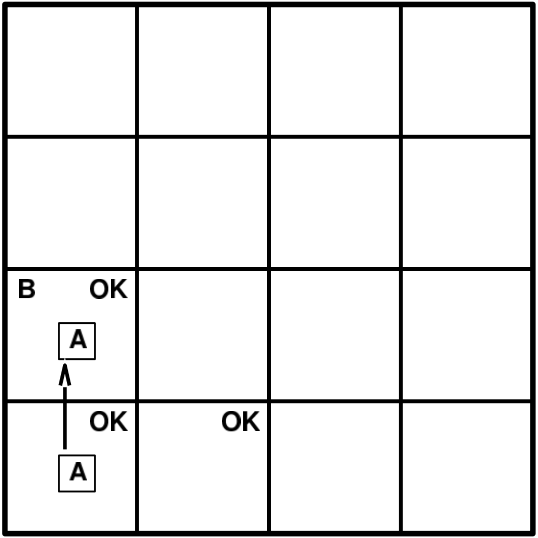

1,4
2,4
3,4
4,4
1,3 W!
2,3
3,3
4,3
1,2 A S OK
2,2 OK
3,2
4,2
1,1 V OK
2,1 B V OK
3,1 P!
4,1
A = Agent
B = Breeze
G = Glitter, Gold
OK = Safe square
P = Pit
S = Stench
V = Visited
W = Wumpus
1,4
2,4 P?
3,4
4,4
1,3 W!
2,3 A S G B
3,3 P?
4,3
1,2 S V OK
2,2 V OK
3,2
4,2
1,1 V OK
2,1 B V OK
3,1 P!
4,1

(a) (b)

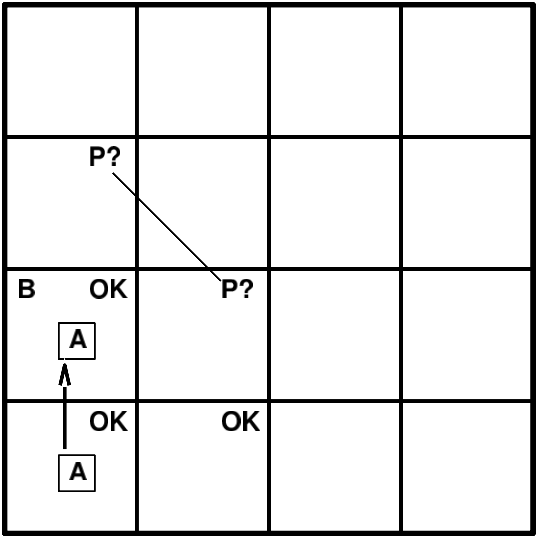
Exploring a wumpus world



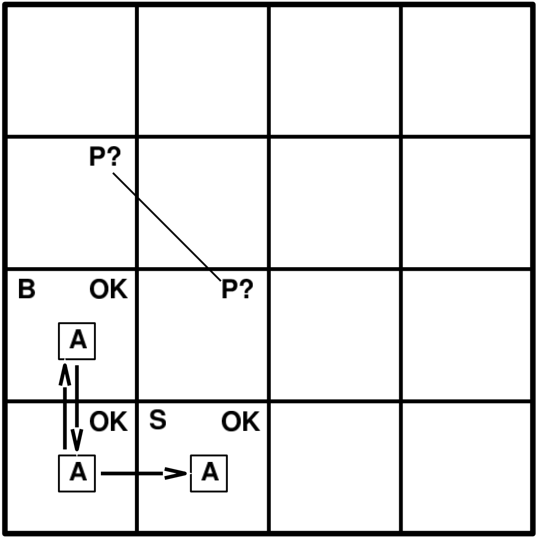
Exploring a wumpus world



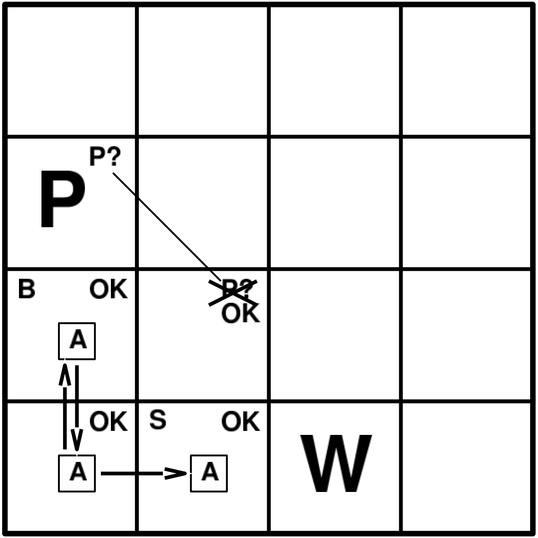
Exploring a wumpus world



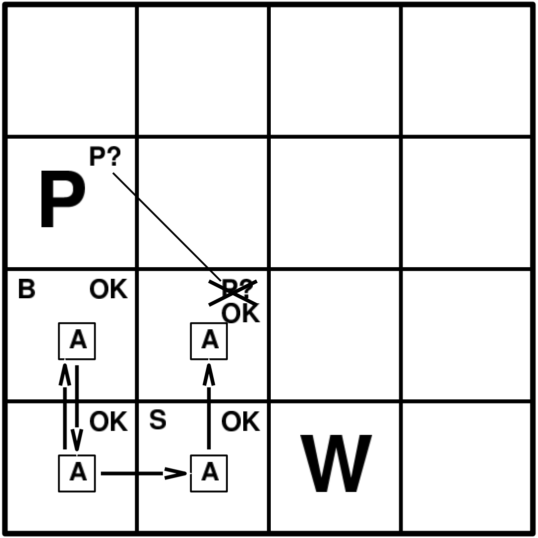
Exploring a wumpus world



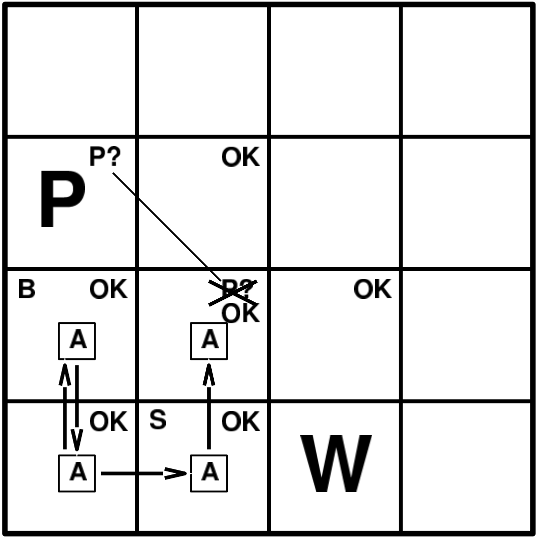
Exploring a wumpus world



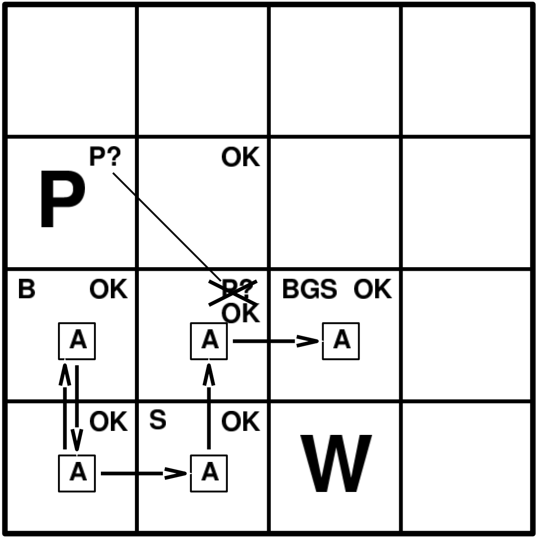
Exploring a wumpus world



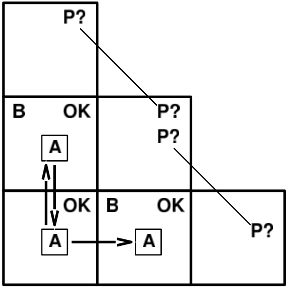
Exploring a wumpus world



Exploring a wumpus world

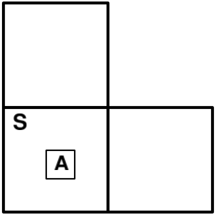


Other tight spots



Breeze in (1,2) and (2,1)
⇒ no safe actions

Assuming pits uniformly distributed,
(2,2) has pit w/ prob 0.86, vs. 0.31



Smell in (1,1) ⇒ cannot move Can use a strategy ofcoercion:
shoot straight ahead
wumpus was there ⇒ dead ⇒ safe
wumpus wasn't there ⇒ safe

Logic

- Knowledge bases consist of sentences.
- These sentences **SYNTAX** are expressed according to the syntax of the representation language, which specifies all the sentences that are well formed.
- The notion of syntax is clear enough in ordinary arithmetic: “x + y = 4” is a well-formed sentence, whereas “x4y+ =” is not.
- **Logics** are formal languages for representing information such that conclusions can be drawn.
- **Syntax** defines the sentences in the language .
- **Semantics** define the **“meaning”** of sentences

i.e., **define truth of a sentence in a world**

E.g., the language of arithmetic

$x + 2 \geq y$ is a sentence; $x2 + y >$ is not a sentence

$x + 2 \geq y$ is true iff the number $x + 2$ is no less than the number y

$x + 2 \geq y$ is true in a world where $x = 7, y = 1$

$x + 2 \geq y$ is false in a world where $x = 0, y = 6$

Entailment

- The possible models are just **all possible assignments** of real numbers to the variables x and y
- **Each such assignment fixes the truth of any sentence of arithmetic whose variables are x and y.**
- If **a sentence α is true in model m**, we say that **m satisfies α** or sometimes **m is a model of α.**
- **The notation M(α) to mean the set of all models of α**
- **Entailment** means that one thing ***follows*** *from* another:
- $KB \models \alpha$
- Knowledge base KB entails sentence α

  if and only if

α is true in all worlds where KB is true

E.g., the KB containing "the Giants won" and "the Reds won" entails "Either the Giants won or the Reds won"

E.g., $x + y = 4$ entails $4 = x + y$

Entailment is a relationship between sentences (i.e., *syntax*) that is based on ***semantics***

Models

- Given a logical sentence, when is its truth uniquely defined in a world?
- Logicians typically think in terms of **models,** which are formally structured worlds.

(e.g., full abstract description of a world, configuration of all variables, world state)

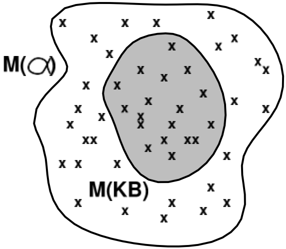
We say m is a model of a sentence α if α is true in m

$M(\alpha)$ is the set of all models of α

Then $KB \models \alpha$ if and only if $M(KB) \subseteq M(\alpha)$

E.g. KB = Giants won and Reds won

α = Giants won

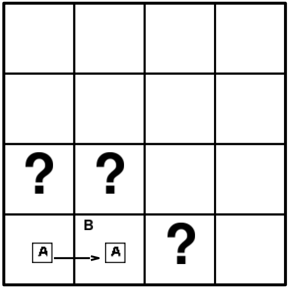


Entailment in the wumpus world

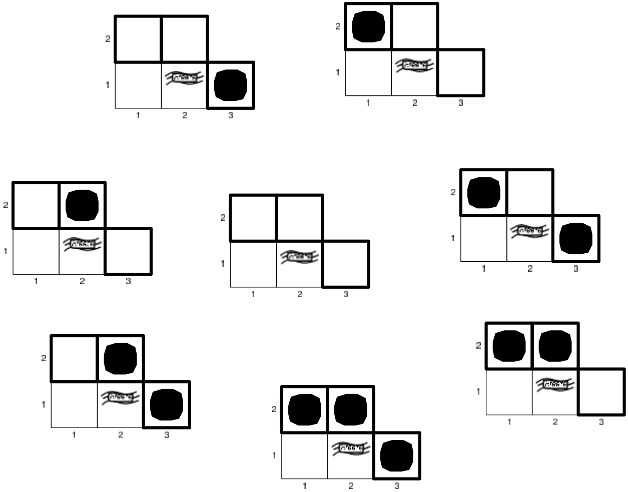
- Situation after detecting nothing in [1,1], moving right, breeze in [2,1]

- Consider possible models for ?s assuming only pits

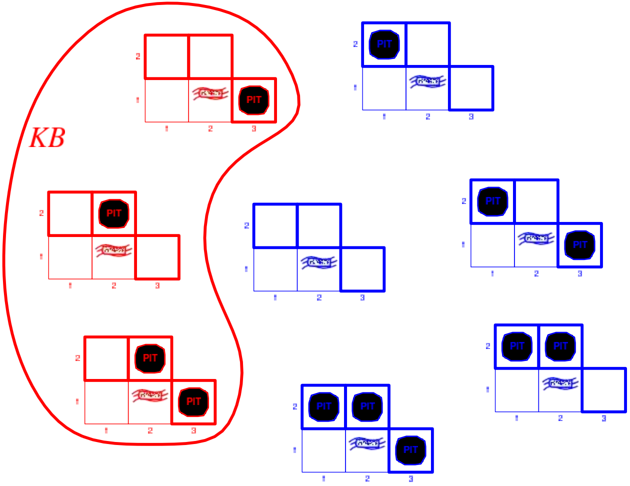
- 3 Boolean choices ⇒ 8 possible models



Wumpus models

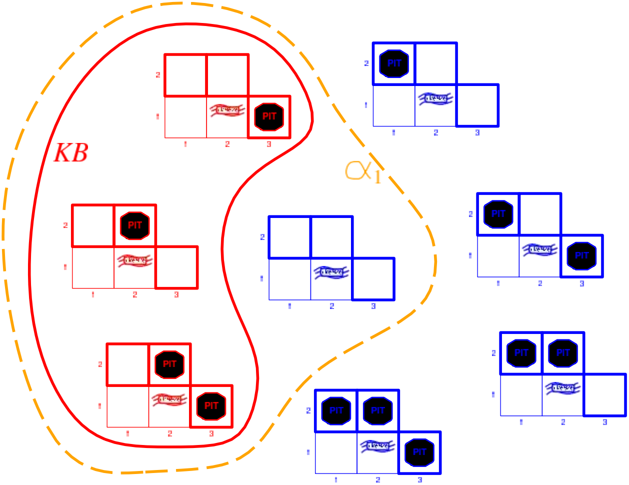


Wumpus models



KB = wumpus-world rules + observations

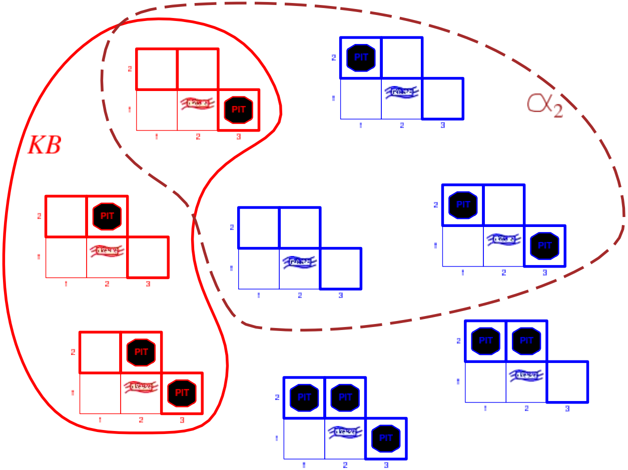
Wumpus models



KB = wumpus-world rules + observations

α_1 = "[1,2] is safe", $KB \models \alpha_1$, proved by **model checking**

Wumpus models



KB = wumpus-world rules + observations

α_2 = "[2,2] is safe", $KB \models \alpha_2$

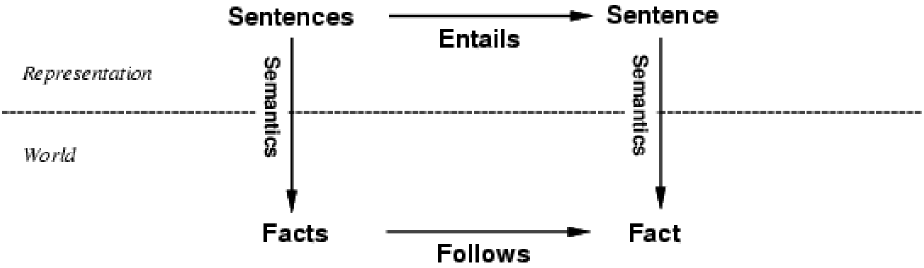
Representation, Reasoning, and Logic

The object of *knowledge representation* is to **express knowledge** in a **computer-tractable form**, so that agents can perform well.

A **knowledge representation language** is defined by:

- Its **syntax**, which defines all possible sequences of symbols that constitute sentences of the language.
  - Examples: Sentences in a book, bit patterns in computer memory.
- Its **semantics**, which determines the facts in the world to which the sentences refer.
  - Each sentence makes a claim about the world.
  - An agent is said to believe a sentence about the world.

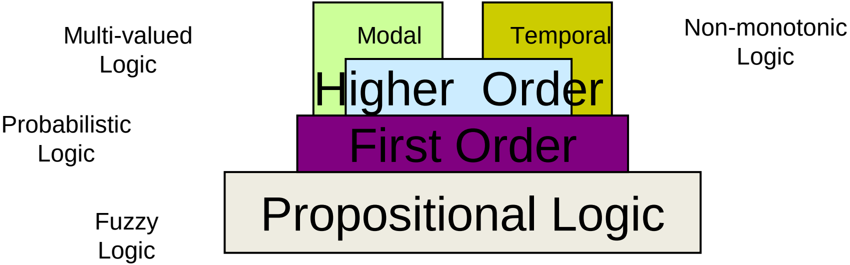
The connection between sentences and facts



➢**Semantics** **maps sentences** **in logic** **to facts** **in the world.**

➢**The property of *one fact following from another* is mirrored by the property of *one sentence being entailed by another*.**

Logic as a Knowledge-Representation (KR) language



Inference

Inference in the general sense means:

Given some pieces of information (prior, observed variabes, knowledge base) what is the implication (the implied information, the posterior) on other things (non-observed variables, sentence)

$KB \vdash_i \alpha$ **= sentence** α **can be derived from** KB **by procedure** I

Eg: Consequences of KB are a haystack; α is a needle.

Entailment = needle in haystack;

Inference = finding it

Soundness: i is sound if whenever $KB \vdash_i \alpha$, it is also true that $KB \models \alpha$

Completeness: i is complete if whenever $KB \models \alpha$, it is also true that $KB \vdash_i \alpha$

Preview: we will define a logic (first-order logic) which is expressive enough to say almost anything of interest, and for which there exists a sound and complete inference procedure. That is, the procedure will answer any question whose answer follows from what is known by the KB**.**

Summary

- Intelligent agents need knowledge about the world for making good decisions.
- The knowledge of an agent is stored in a knowledge base in the form of **sentences** in a knowledge representation language.
- A knowledge-based agent needs a **knowledge base** and an **inference mechanism**.
    - ✓ **It operates by storing sentences in its knowledge base,**
    - ✓ **Inferring new sentences with the inference mechanism,**
    - ✓ **and using them to deduce which actions to take.**
- A **representation language** is defined by its syntax and semantics, which specify the structure of sentences and how they relate to the facts of the world.
- The **interpretation** of a sentence is the fact to which it refers.
    - ✓ **If this fact is part of the actual world, then the sentence is true**.

Summary

- The process of deriving new sentences from old one is called **inference**.
  - ✓**Sound inference processes derives true conclusions given true premises.**
  - ✓**Complete inference processes derive all true conclusions from a set of premises.**
- A **valid sentence** is true in all worlds under all interpretations.
- If an implication sentence can be shown to be valid, then - given its premise - its consequent can be derived.

PROPOSITIONAL LOGIC

Propositional logic (PL)

- A simple language useful for showing key ideas and definitions
- User defines a set of propositional symbols, like P and Q.
- User defines the semantics of each of these symbols, e.g.:
  - ✓P means "It is hot"
  - ✓Q means "It is humid"
  - ✓R means "It is raining"
- A sentence (aka formula, well-formed formula, wff) defined as:
  - ✓A symbol
  - ✓If S is a sentence, then ~S is a sentence (e.g., "not”)
  - ✓If S is a sentence, then so is (S)
  - ✓If S and T are sentences, then (S v T), (S ^ T), (S => T) , and (S <=> T) are sentences (e.g., "or," "and," "implies," and "if and only if”)
  - ✓A finite number of applications of the above

Propositional logic: Syntax

- **Propositional logic is the simplest logic—illustrates basic ideas**
- **The proposition symbols P_1, P_2 etc are sentences**
- If S is a sentence, $\neg S$ is a sentence (negation)
- If S_1 and S_2 are sentences, $S_1 \wedge S_2$ is a sentence (conjunction)
- If S_1 and S_2 are sentences, $S_1 \vee S_2$ is a sentence (disjunction)
- If S_1 and S_2 are sentences, $S_1 \Rightarrow S_2$ is a sentence (implication)
- If S_1 and S_2 are sentences, $S_1 \Leftrightarrow S_2$ is a sentence (biconditional)

Propositional logic:

- Complex sentences are constructed from simpler sentences, using parentheses and logical COMPLEX SENTENCES connectives.
- There are five connectives in common use: **LOGICAL CONNECTIVES**
- **NEGATION ¬ (not):** A sentence such as **¬W1,3** is called the negation of **W1,3**. A literal is either an atomic sentence (a positive literal) or a negated atomic sentence (a negative literal).
- **CONJUNCTION ∧ (and):** A sentence whose main connective is ∧, such as **W1,3 ∧ P3,1**, is called a conjunction; its parts are the conjuncts. (The ∧ looks like an "A" for "And.")
- **DISJUNCTION ∨ (or):** A sentence using ∨, such as **(W1,3∧P3,1)∨W2,2**, is a disjunction of the disjuncts (W1,3 ∧ P3,1) and W2,2.
- **IMPLICATION ⇒ (implies):** A sentence such as **(W1,3∧P3,1) ⇒ ¬W2,2** is called an implication (or conditional). Its **premise or antecedent** is (W1,3 ∧P3,1), and its **conclusion or consequent** is ¬W2,2. Implications are also known as rules or if–then statements.
- **BICONDITIONAL ⇔ (if and only if):** The sentence **W1,3 ⇔ ¬W2,2** is a biconditional.

Examples of PL sentences

(P ^ Q) => R

“If it is hot and humid, then it is raining”

Q => P

“If it is humid, then it is hot”

Q

“It is humid.”

A better way:

Ho = “It is hot”

Hu = “It is humid”

R = “It is raining”

Propositional logic: Syntax grammar

$$
\begin{array}{rcl}
\mathit{Sentence} & \rightarrow & \mathit{AtomicSentence} \mid \mathit{ComplexSentence} \\
\mathit{AtomicSentence} & \rightarrow & \mathit{True} \mid \mathit{False} \mid P \mid Q \mid R \mid \ldots \\
\mathit{ComplexSentence} & \rightarrow & (\,\mathit{Sentence}\,) \mid [\,\mathit{Sentence}\,] \\
& \mid & \neg\, \mathit{Sentence} \\
& \mid & \mathit{Sentence} \land \mathit{Sentence} \\
& \mid & \mathit{Sentence} \lor \mathit{Sentence} \\
& \mid & \mathit{Sentence} \;\Rightarrow\; \mathit{Sentence} \\
& \mid & \mathit{Sentence} \;\Leftrightarrow\; \mathit{Sentence} \\
\\
\text{OPERATOR PRECEDENCE} & : & \neg, \land, \lor, \Rightarrow, \Leftrightarrow
\end{array}
$$

Figure 7.7 A BNF (Backus–Naur Form) grammar of sentences in propositional logic, along with operator precedences, from highest to lowest.

Propositional logic: Semantics

Each model specifies true/false for each proposition symbol

E.g. $P_{1,2}$ $P_{2,2}$ $P_{3,1}$

True True false

(With these symbols, 8 possible models, can be enumerated automatically.)

Rules for evaluating truth with respect to a model m:

$\neg S$	is true iff	S	is false		
$S_1 \wedge S_2$	is true iff	S_1	is true *and*	S_2	is true
$S_1 \vee S_2$	is true iff	S_1	is true *or*	S_2	is true
$S_1 \Rightarrow S_2$	is true iff	S_1	is false *or*	S_2	is true
i.e.,	is false iff	S_1	is true *and*	S_2	is false
$S_1 \Leftrightarrow S_2$	is true iff	$S_1 \Rightarrow S_2$	is true *and*	$S_2 \Rightarrow S_1$	is true

Simple recursive process evaluates an arbitrary sentence,

e.g., $\neg P_{1,2} \wedge (P_{2,2} \vee P_{3,1})$ **= true $\wedge$ (false $\vee$ true) = true $\wedge$ true = true**

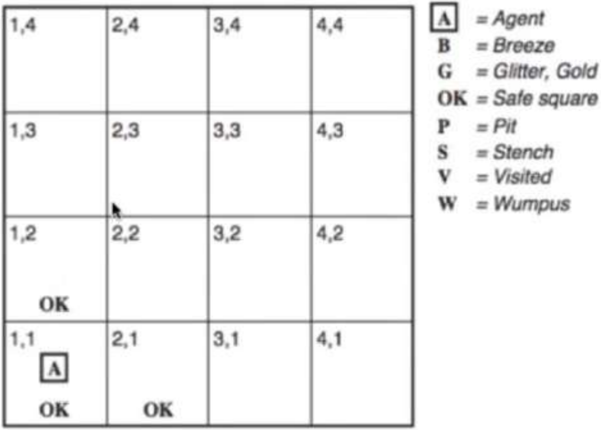
Semantics

- Sentences have a Truth value with respect to a model
- For example: $m = \{P_{1,2} = \mathit{false}, P_{2,2} = \mathit{false}, P_{2,1} = \mathit{True}\}$
- Or: $m = \{P_{1,2} = \mathit{false}, P_{2,2} = \mathit{true}, P_{2,1} = \mathit{false}\}$
- $P_{1,2}$ is just a symbol. It can mean anything.
- Truth value is computed recursively according to...

Truth tables for connectives

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
false	*false*	*true*	*false*	*false*	*true*	*true*
false	*true*	*true*	*false*	*true*	*true*	*false*
true	*false*	*false*	*false*	*true*	*false*	*false*
true	*true*	*false*	*true*	*true*	*true*	*true*

Wumpus: Inference
Starting State



[None,None,None,None,None]

- $\neg P$ is true if P is false in m (negation)
- $P \wedge Q$ is true iff both P and Q are true in m (conjunction)
- $P \vee Q$ is true iff either P or Q are true in m (disjunction)
- $P \Rightarrow Q$ is true unless P is true and Q is false (implication)[4]
- $P \leftrightarrow Q$ is true iff P and Q are both true or both false[5] (biconditional)

---

[4] if P is true I claim that Q is true. Otherwise no claim

[5] if P is true I claim that Q is true, if P is false I claim that Q is false. otherwise no claim

in the model $m = \{P_{1,2} = \mathit{false}, P_{2,2} = \mathit{false}, P_{2,1} = \mathit{True}\}$

Evaluate $\neg P_{1,2} \wedge P_{2,2} \vee P_{3,1}$

Evaluate it for $m = \{P_{1,2} = \mathit{true}, P_{2,2} = \mathit{true}, P_{2,1} = \mathit{false}\}$

A Simple Knowledge Base

- **Px,y** is true if there is a pit in [x, y].
- **Wx,y** is true if there is a wumpus in [x, y], dead or alive.
- **Bx,y** is true if the agent perceives a breeze in [x, y].
- **Sx,y** is true if the agent perceives a stench in [x, y].

A Simple KB
Definitions

- $P_{x,y}$ is true if there's a pit in $[x, y]$
- $W_{x,y}$ is true if there is a Wumpus in $[x, y]$
- $B_{x,y}$ is true if the agent perceives a breeze in $[x, y]$
- $S_{x,y}$ is true if the agent perceives a stench in $[x, y]$

Wumpus world sentences

- Let $P_{i,j}$ be true if there is a pit in $[i, j]$.
- Let $B_{i,j}$ be true if there is a breeze in $[i, j]$.

$\neg P_{1,1}$

$\neg B_{1,1}$

$B_{2,1}$

- **"Pits cause breezes in adjacent squares"**

Wumpus world sentences

Let $P_{i,j}$ be true if there is a pit in $[i, j]$.
Let $B_{i,j}$ be true if there is a breeze in $[i, j]$.

$\neg P_{1,1}$

$\neg B_{1,1}$

$B_{2,1}$

"Pits cause breezes in adjacent squares"

$$B_{1,1} \Leftrightarrow (P_{1,2} \lor P_{2,1})$$
$$B_{2,1} \Leftrightarrow (P_{1,1} \lor P_{2,2} \lor P_{3,1})$$

"A square is breezy *if and only if* there is an adjacent pit"

A Simple Knowledge Base

- There is no pit in [1,1]:

  R1 : ¬P1,1
- A square is breezy if and only if there is a pit in a neighboring square.
- This has to be stated for each square; for now, we include just the relevant squares:

  R2 : B1,1 ⇔ (P1,2 ∨ P2,1)

  R3 : B2,1 ⇔ (P1,1 ∨ P2,2 ∨ P3,1)
- The preceding sentences are true in all wumpus worlds.
- Now we include the breeze percepts for the first two squares visited in the specific world the agent is in.

  R4 : ¬B1,1

  R5 : B2,1

A simple KB
Rules

For the Wumpus world in general.

- $R_1 : \neg P_{1,1}$
- $R_2 : B_{1,1} \leftrightarrow (P_{1,2} \vee P_{2,1})$
- $R_3 : B_{2,1} \leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$

Now, after visiting [1,1]; [1,2] and [2,1]

- $R_4 : \neg B_{1,1}$
- $R_5 : B_{2,1}$

$KB = R_1 \wedge R_2 \wedge R_3 \wedge R_4 \wedge R_5$

I want to find whether my KB says there's no pit in [1,2]

That is, does $KB \models \neg P_{1,2}$?

We say that $\neg P_{1,2}$ is a sentence α

Main goal: decide whether $KB \models \alpha$

α can be a much more complex query

- enumerate the models
- for each model, check that:
- if it is true in α is has to be true in KB

In the Wumpus world: 7 relevant symbols:
$B_{1,1}, B_{2,1}, P_{1,1}, P_{1,2}, P_{2,1}, P_{2,2}, P_{3,1}$
$2^7 = 128$ models. Only 3 are true

Inference
All Possible Models

$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	R_1	R_2	R_3	R_4	R_5	KB
false	*false*	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*false*	*false*
false	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*false*	*true*	*false*	*false*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
false	*true*	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*false*	*true*	*true*	*false*
false	*true*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*false*	*true*	*false*	*true*	*true*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*true*	*false*	*false*	*true*	*false*	*false*	*true*	*true*	*false*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
true	*true*	*true*	*true*	*true*	*true*	*true*	*false*	*true*	*true*	*false*	*true*	*false*

Does $KB \models \neg P_{1,1}$?

Inference
TT-Entails

```
function TT-Entails(KB,q) //q is the query in prop. logic
    symbols=list of the proposition symbols in KB and q
    return TT-Check-All(KB,q,symbols,{})

function TT-Check-All(KB,q,symbols,model)
    if isEmpty(symbols)
        if PL-True(KB,model)
            return PL-True(q,model)
        else
            return true // if KB is false, always return true
    else do
        P=First(symbols)
        rest=Rest(symbols)
        return (TT-Check-All(KB,q,rest,model + {P=true}) AND
               (TT-Check-All(KB,q,rest,model + {P=false})

function PL-True(sentence, model)
    //returns true if sentence holds within the model
```

Inference
Model Checking Complexity

if KB and α contain n symbols in all:

Time complexity: $O(2^n)$

Space complexity: $O(n)$ because it is depth first.

propositional entailment is co-NP complete (probably no easier than NP-Complete)

Logical equivalence

Two sentences are logically equivalent iff true in same models:

$\alpha \equiv \beta$ if and only if $\alpha \models \beta$ and $\beta \models \alpha$

$(\alpha \land \beta)$	$\equiv$	$(\beta \land \alpha)$ commutativity of $\land$
$(\alpha \lor \beta)$	$\equiv$	$(\beta \lor \alpha)$ commutativity of $\lor$
$((\alpha \land \beta) \land \gamma)$	$\equiv$	$(\alpha \land (\beta \land \gamma))$ associativity of $\land$
$((\alpha \lor \beta) \lor \gamma)$	$\equiv$	$(\alpha \lor (\beta \lor \gamma))$ associativity of $\lor$
$\neg(\neg\alpha)$	$\equiv$	α double-negation elimination
$(\alpha \Rightarrow \beta)$	$\equiv$	$(\neg\beta \Rightarrow \neg\alpha)$ contraposition
$(\alpha \Rightarrow \beta)$	$\equiv$	$(\neg\alpha \lor \beta)$ implication elimination
$(\alpha \Leftrightarrow \beta)$	$\equiv$	$((\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha))$ biconditional elimination
$\neg(\alpha \land \beta)$	$\equiv$	$(\neg\alpha \lor \neg\beta)$ De Morgan
$\neg(\alpha \lor \beta)$	$\equiv$	$(\neg\alpha \land \neg\beta)$ De Morgan
$(\alpha \land (\beta \lor \gamma))$	$\equiv$	$((\alpha \land \beta) \lor (\alpha \land \gamma))$ distributivity of $\land$ over $\lor$
$(\alpha \lor (\beta \land \gamma))$	$\equiv$	$((\alpha \lor \beta) \land (\alpha \lor \gamma))$ distributivity of $\lor$ over $\land$

Validity and satisfiability

- A sentence is **valid** if it is true in ***all*** models,
- e.g., true, $A \lor \neg A$, $A \Rightarrow A$, $(A \land (A \Rightarrow B)) \Rightarrow B$
- Validity is connected to inference via the Deduction Theorem:
- $KB \models \alpha$ if and only if $(KB \Rightarrow \alpha)$ is valid
- A sentence is **satisfiable** if it is true in ***some*** model
- e.g., $R1 \land R2 \land R3 \land R4 \land R5$ *is satisfiable for three models*
- A sentence is **unsatisfiable** if it is true in ***no*** models
- e.g., $A \land \neg A$
- Satisfiability is connected to inference via the following:
- **$KB \models \alpha$ if and only if $(KB \land \neg \alpha)$ is unsatisfiable**

INFERENCE AND THEOREM PROVING

Inference
Logical Equivalences[6]

$(\alpha \wedge \beta)$	$\equiv (\beta \wedge \alpha)$	commutativity of $\wedge$
$(\alpha \vee \beta)$	$\equiv (\beta \vee \alpha)$	commutativity of $\vee$
$((\alpha \wedge \beta) \wedge \gamma)$	$\equiv (\alpha \wedge (\beta \wedge \gamma))$	associativity of $\wedge$
$((\alpha \vee \beta) \vee \gamma)$	$\equiv (\alpha \vee (\beta \vee \gamma))$	associativity of $\vee$
$\neg(\neg\alpha)$	$\equiv \alpha$	double-negation elimination
$(\alpha \Rightarrow \beta)$	$\equiv (\neg\beta \Rightarrow \neg\alpha)$	contraposition
$(\alpha \Rightarrow \beta)$	$\equiv (\neg\alpha \vee \beta)$	implication elimination
$(\alpha \Leftrightarrow \beta)$	$\equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha))$	biconditional elimination
$\neg(\alpha \wedge \beta)$	$\equiv (\neg\alpha \vee \neg\beta)$	De Morgan
$\neg(\alpha \vee \beta)$	$\equiv (\neg\alpha \wedge \neg\beta)$	De Morgan
$(\alpha \wedge (\beta \vee \gamma))$	$\equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma))$	distributivity of $\wedge$ over $\vee$
$(\alpha \vee (\beta \wedge \gamma))$	$\equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma))$	distributivity of $\vee$ over $\wedge$

[6] There are many more, but these are the main ones

Inference By Theorem Proving
Concepts

- Logical Equivalence: $\alpha \equiv \beta$ iff $\alpha \models \beta$ and $\beta \models \alpha$
- Validity: A tautology: it is true in all models. e.g. $P \vee \neg P$
- Deduction: $\alpha \models \beta$ iff $\alpha \Rightarrow \beta$
- Satisfiability: if some model makes it true.

Modus Ponens

$$\frac{\alpha \Rightarrow \beta,\ \alpha}{\beta}$$

If α implies β and α is true, then β is true

And Elimination

$$\frac{\alpha \wedge \beta}{\alpha}$$

Other rules can also be inference rules

$$\frac{\alpha \iff \beta}{(\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha)}$$

$$\frac{(\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha)}{\alpha \iff \beta}$$

Inference
In our Wumpus World: Is there a pit in 1,2?

- $R_1 : \neg P_{1,1}$
- $R_2 : B_{1,1} \iff (P_{1,2} \vee P_{2,1})$
- $R_3 : B_{2,1} \iff (P_{1,1} \vee P_{2,2} \vee P_{3,1})$
- $R_4 : \neg B_{1,1}$
- $R_5 : B_{2,1}$

We have $KB = R_1 \wedge R_2 \wedge R_3 \wedge R_4 \wedge R_5$. We want to prove $\neg P_{1,2}$

- $R_6 : (B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$ by bicond. elim R_2
- $R_7 : ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$ by And-Elimination to R_6
- $R_8 : (\neg B_{1,1} \Rightarrow \neg(P_{1,2} \vee P_{2,1}))$ by contrapositives
- $R_9 : \neg(P_{1,2} \vee P_{2,1})$ by Modus Ponens with R_8 and R_4
- $R_{10} : \neg P_{1,2} \wedge \neg P_{2,1}$

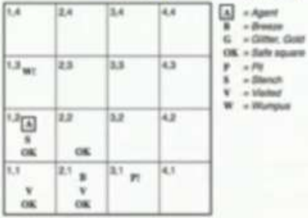
That is: Neither [1,2] nor [2,1] contains a pit.

Inference
As Search

- Intial State: The initial Knowledge Base
- Actions: The set of all the inference rules applied to all sentences that match top half
- Result: Add sentence in the bottom half of the inference rule
- Goal: The goal is a state that contains sentence we want to prove

Inference
By Resolution

Let's say agent returns to [1,1] from [2,1] and goes to [1,2]



We add:

- $R_{11} : \neg B_{1,2}$
- $R_{12} : B_{1,2} \iff (P_{1,1} \vee P_{2,2} \vee P_{1,3})$

Inference
By Resolution

We can continue using same process as earlier.

- $R_{13} : \neg P_{2,2}$ Contrapositive R_{12} and AND elimination
- $R_{14} : \neg P_{1,3}$ Same as above.
- $R_{15} : P_{1,1} \vee P_{2,2} \vee P_{3,1}$ bi-conditional elem. R_3 and modus ponens R_5

And the literal $\neg P_{2,2}$ in R_{13} resolves with $P_{2,2}$ in R_{15} to give the resolvent

- $R_{16} : P_{1,1} \vee P_{3,1}$

more generally...

$$\frac{A \vee B, \neg A \vee C}{B \vee C}$$

Anything else that resolves?

Resolution
Conjunctive Normal Form (CNF)

Every sentence in propositional logic can be expressed as conjunctions of disjunctions of literals.

e.g. $(A \vee B) \wedge (\neg C \vee D \vee \neg E) \wedge \ldots$

$B_{1,1} \iff (P_{1,2} \vee P_{2,1})$ in CNF?

- Eliminate $\iff$ replacing $\alpha \iff \beta$ with $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$
- $(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$
- Eliminate $\Rightarrow$ by replacing $\alpha \Rightarrow \beta$ with $\neg\alpha \vee \beta$
- $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,1})$
- Symbol $\neg$ should appear next to each literal: DeMorgan $\neg(\alpha \vee \beta) \equiv \neg\alpha \wedge \neg\beta$ and $\neg(\alpha \wedge \beta) \equiv \neg\alpha \vee \neg\beta$
- $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1})$
- Distribute $\vee$ over $\wedge$ and flatten
- $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$

RESOLUTION

Resolution
An algorithm

Algorithm works using **proof by contradiction**.

To show $KB \models \alpha$ we show that $KB \wedge \neg\alpha$ is not satisfiable

Apply resolution to $KB \wedge \neg\alpha$ in CNF

and Resolve pairs with complementary literals

$$\frac{l_1 \vee \ldots \vee l_k, \quad m_1 \vee \ldots \vee m_n}{l_1 \vee \ldots l_{i-1} \vee l_{i+1} \ldots \vee l_k \vee m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \ldots \vee m_n}$$

if l_i and m_j are complimentary literals

and add new clauses

until

- there are no new clauses to be added
- two clauses resolve to the *empty* class, which means $KB \models \alpha$

Resolution
An algorithm

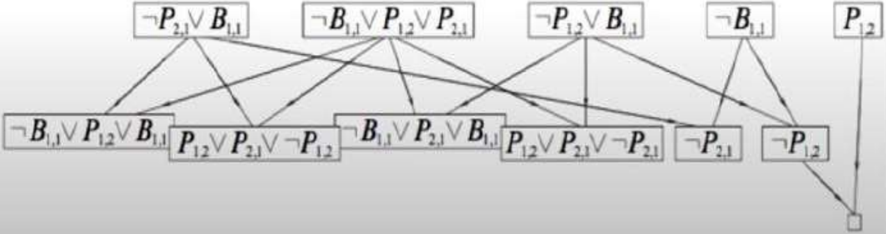
```
function PL-Resolution(KB,q)
// KB, the knowledge base. a sentence in prop logic.
// q, the query, a sentence in prop logic
    clauses= contra(KB,q) //CNF representation of KB ∧ ¬q
    new = {}
    do
        for each pair of clauses Ci,Cj in clauses
            resolvents=PL-Resolve(Ci,Cj)
            if resolvents contains the empty clause
                return true
            new = new + resolvents
        if new is subset of clauses
            return false
        clauses = clauses + new
```

Resolution
Algorithm

Say the agent is in [1,1], no breeze, so no pits can be in there.

$\alpha = \neg P_{1,2}$

$$
\begin{aligned}
KB &= R_2 \wedge R_4 \\
KB &= (B_{1,1} \iff (P_{1,2} \vee P_{2,1})) \wedge \neg B_{1,1} \\
KB \wedge \neg\alpha &= (\neg P_{2,1} \vee B_{1,1}) \wedge (\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg B_{1,1}) \wedge (P_{1,2})
\end{aligned}
$$



FORWARD CHAINING & BACKWARD CHAINING

Horn Form

- KB conjunction of Horn clauses
- Horn Clause (at most one literal is Positive[7])
- For example: $(\neg P \vee \neg Q \vee V)$ is a Horn Clause.
- so is $(\neg P \vee \neg W)$. But, $(\neg P \vee Q \vee V)$ is not.
- Definite Clauses: exactly one literal is positive.

- Horn clauses can be re-written as implications
  - proposition symbol (fact) or
  - conjunction of symbols (body or premise) $\Rightarrow$ symbol (head)
  - Example: $(\neg C \vee \neg B \vee A)$ becomes $(C \wedge B \Rightarrow A)$

Modus ponens for Horn KB:

$$\frac{\alpha_1 \ldots \alpha_n, \alpha_1 \wedge \ldots \alpha_n \Rightarrow \beta}{\beta}$$

[7] Not negated

Forward chaining

Idea: fire any rule whose premises are satisfied in the *KB*, add its conclusion to the *KB*, until query is found.

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

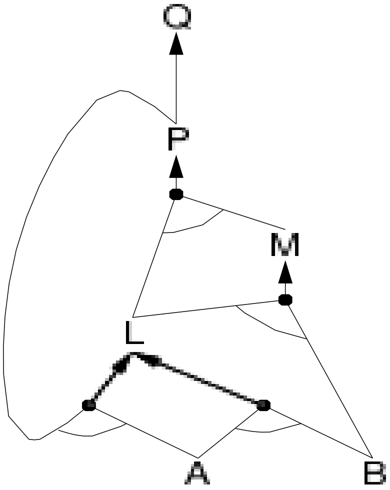
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

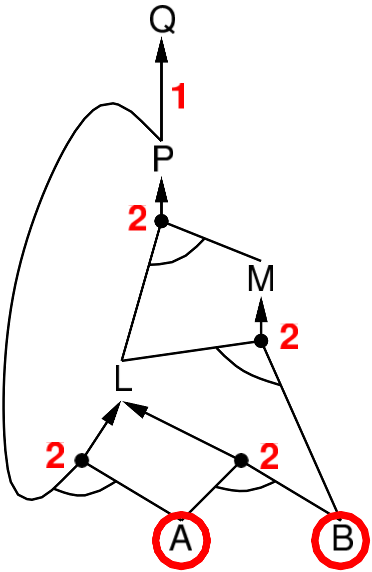
$A \wedge B \Rightarrow L$

A

B



Forward chaining example



$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

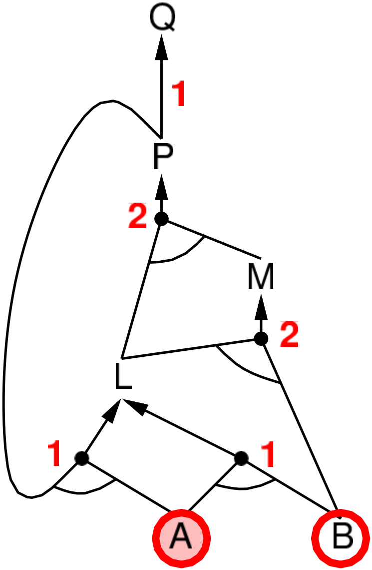
$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B

Forward chaining example



$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

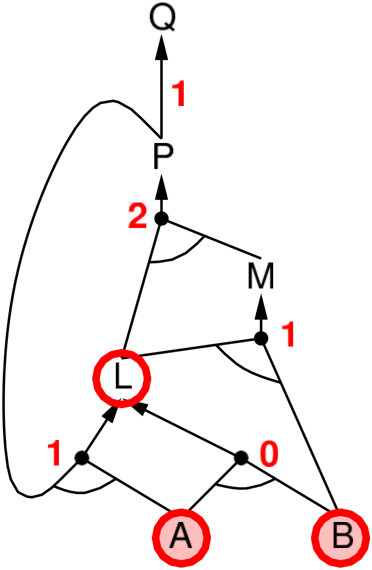
$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B

Forward chaining example



$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

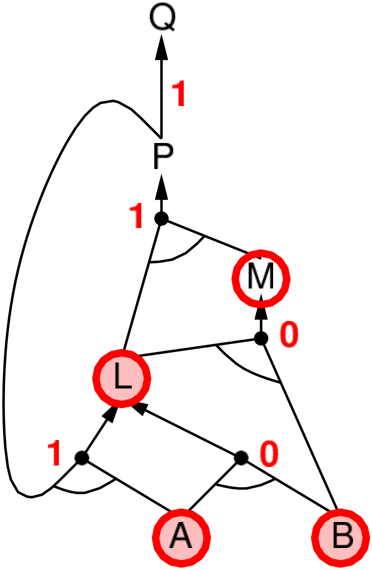
$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B

Forward chaining example



$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

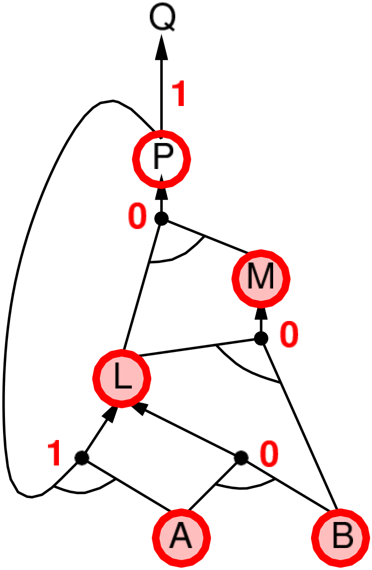
$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

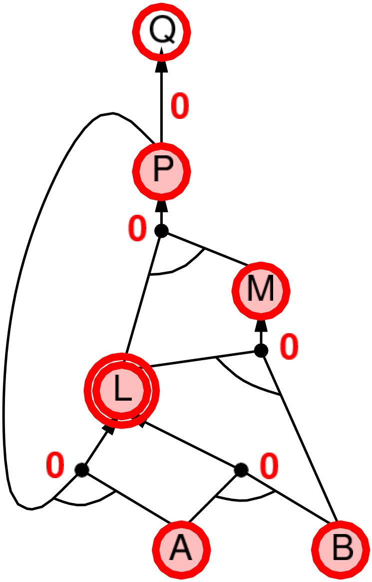
B

Forward chaining example



$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$

Forward chaining example



$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

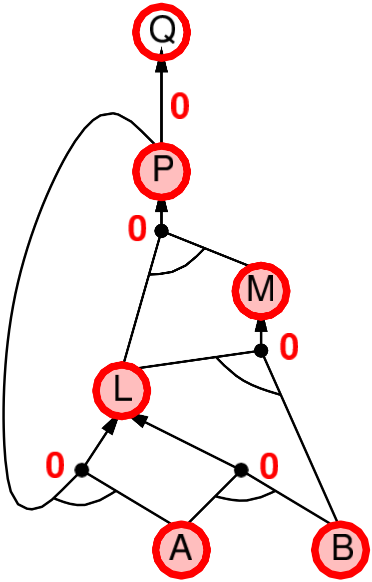
$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B

Forward chaining example



$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

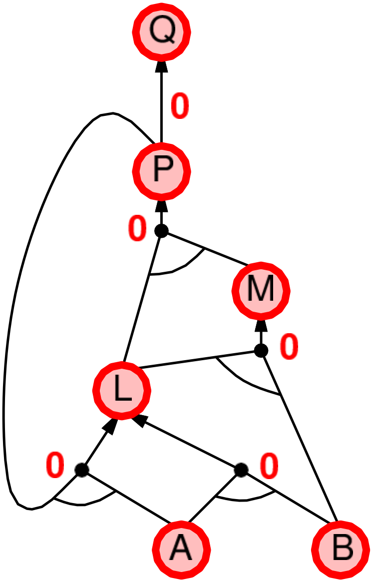
$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B

Forward chaining example



$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B

Inference
Forward Chaining

```
function PL-FC-Entails(KB,q)
// KB, the knowledge base, a prop. sentence
// q, the query, a prop. sentence
    count = a table //count[c] is num of symbols in c's premise
    inferred = a table //inferred[s] initially false for all s
    agenda = a queue of symbols //Init w/symbols that are true

    while agenda is not empty
        p = pop(agenda)
        if p=q then return true
        if inferred[p]=false
            inferred[p]=true
            for each clause c in KB that contains p in premise
                decrement count[c]
                if count[c]=0
                    add c.conclusion to agenda
    return false
```

Backward chaining

- **Idea: work backwards from the query *q*:** to prove *q* by BC, check if *q* is known already, or prove by BC all premises of some rule concluding *q*
- **Avoid loops:** check if new subgoal is already on the goal stack
- **Avoid repeated work:** check if new subgoal has already been proved true, or has already failed

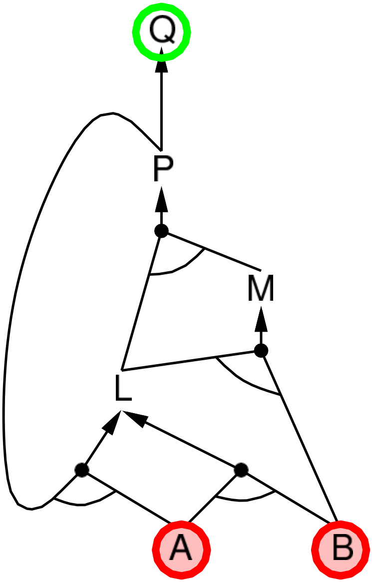
Work backwards from query q

- To prove q by B.C.
- check if q is known or
- prove by B.C. all premises of some rule concluding q

Avoid Loops: Check if new subgoal is already in goal stack
Avoid repeat work: Check if new subgoal

- has already been proved true or
- has already failed

Backward chaining example



$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

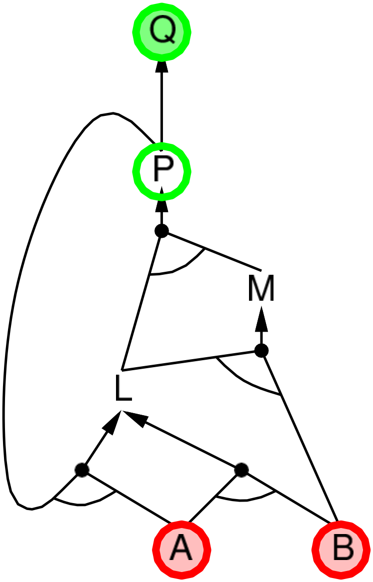
$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B

Backward chaining example



$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

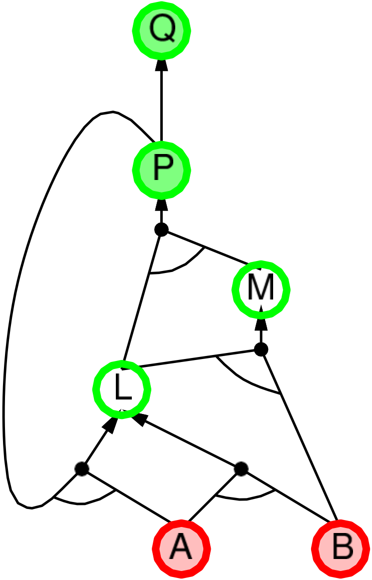
$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B

Backward chaining example



$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

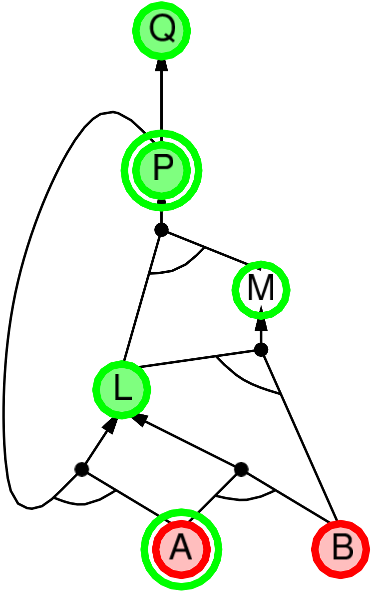
$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B

Backward chaining example



$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

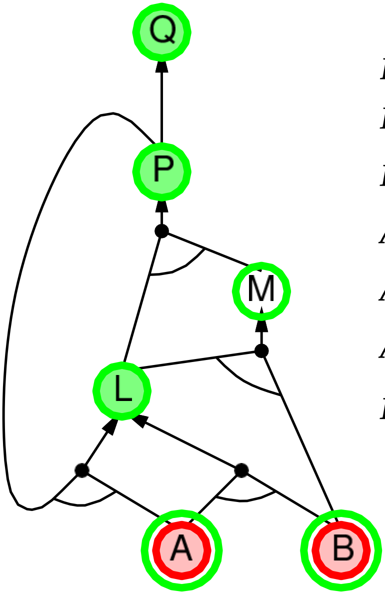
$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B

Backward chaining example



$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

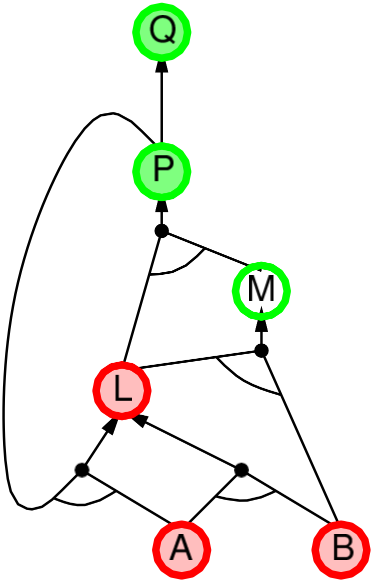
$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B

Backward chaining example



$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

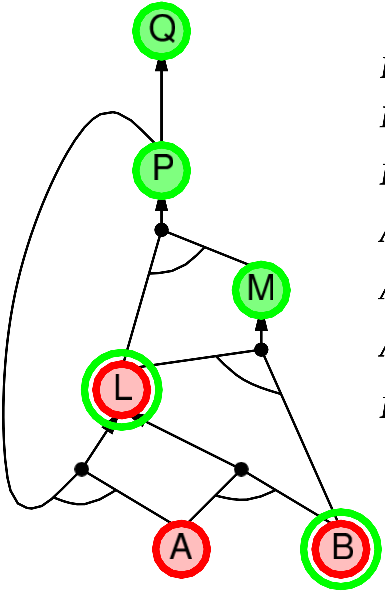
$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B

Backward chaining example



$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

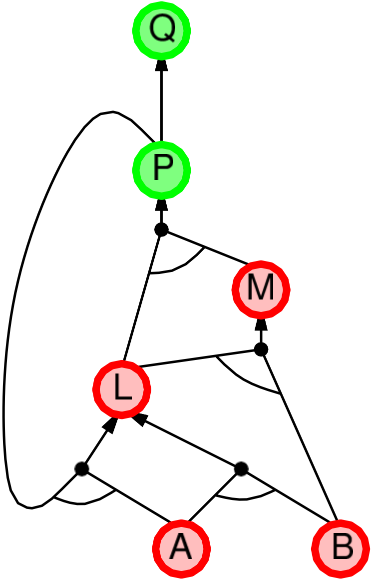
$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B

Backward chaining example



$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

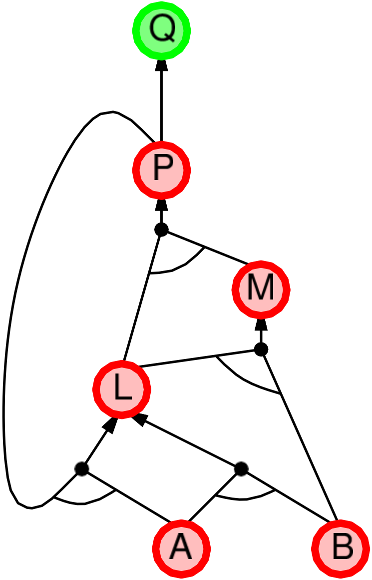
$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B

Backward chaining example



$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

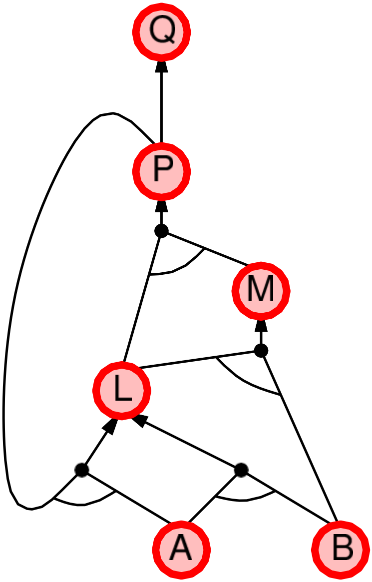
$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B

Backward chaining example



$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B

Forward and Backward Chaining
Discussion

- FC is data driven. E.g. object recognition, routine decision
- FC may do a lot of work irrelevant to the goal
- BC is goal driven. Appropriate for problem solving. I.e. Where is home? What's the result of equation x?
- Complexity of BC can be much less than linear size of KB

Summary

Logical agents apply inference to a knowledge base to derive new information and make decisions

Basic concepts of logic:

– syntax: formal structure of sentences

– semantics: truth of sentences wrt models

– entailment: necessary truth of one sentence given another

– inference: deriving sentences from other sentences

– soundness: derivations produce only entailed sentences

– completeness: derivations can produce all entailed sentences

Wumpus world requires the ability to represent partial and negated information, reason by cases, etc.

Forward, backward chaining are linear-time, complete for Horn clauses

Resolution is complete for propositional logic

Propositional logic lacks expressive power

Dictionary: logic in general

alogic: a language, elements α aresentences, (grammar example: slide 34)

model m: a world/state description that allows us to evaluate $\alpha(m) \in$ {true, false} uniquely for any sentence α, $M(\alpha) = \{m : \alpha(m) = \text{true}\}$

entailment $\alpha \models \beta$: $M(\alpha) \subseteq M(\beta)$, " $\forall_m : \alpha(m) \Rightarrow \beta(m)$"
(Folgerung)

equivalence $\alpha \equiv \beta$: iff ($\alpha \models \beta$ and $\beta \models \alpha$)

KB: a set of sentences

inferenceprocedure i can infer α from KB: KB €$_i$ α soundnessof i: KB €$_i$ α implies $KB \models \alpha$ (Korrektheit) completenessof i: $KB \models \alpha$ implies KB €$_i$ α

Dictionary: propositional logic

conjunction: $\alpha \wedge \beta$,disjunction: $\alpha \vee \beta$,negation: $\neg\alpha$

implication: $\alpha \Rightarrow \beta \equiv \neg\alpha \vee \beta$,biconditional:

$\alpha \Leftrightarrow \beta \equiv (\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$

Note: $\models$ and $\equiv$ are statements about sentences in a logic; $\Rightarrow$ and $\Leftrightarrow$ are symbols in the grammar of propositional logic

α valid: true for *any* model, e.g.: $KB \models \alpha$ iff [$(KB \Rightarrow \alpha)$ is valid] (allgemeingültig)

α unsatisfiable: true for *no* model, e.g.: $KB \models \alpha$ iff [$(KB \wedge \neg\alpha)$ is unsatisfiable]

literal: A or $\neg A$,clause: disjunction of literals,CNF: conjunction of clauses

Horn clause: symbol | (conjunction of symbols $\Rightarrow$ symbol),Horn form: conjunction of Horn clauses

Modus Ponensrule: complete for Horn KBs $\dfrac{\alpha_1,\ldots,\alpha_n, \quad \alpha_1 \wedge \cdots \wedge \alpha_n \Rightarrow \beta}{\beta}$

Resolutionrule: complete for propositional logic in CNF, let "$A_i = \neg m_j$":

$$\frac{A_1 \vee \cdots \vee A_k, \quad m_1 \vee \cdots \vee m_n}{A_1 \vee \cdots \vee A_{i-1} \vee A_{i+1} \vee \cdots \vee A_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n}$$

Effective Propositional Model Checking

- Two families of efficient algorithms for propositional inference based on model checking:
- Mainly used for checking satisfiability
- Complete Backtracking Search Algorithms

    DPLL Algorithm (Davis, Putnam, Logemann, Loveland)
- Incomplete Local Search Algorithms

    WalkSAT Algorithm

Conversion to CNF

$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$

Eliminate $\Leftrightarrow$, replacing $\alpha \Leftrightarrow$ ß with $(\alpha \Rightarrow$ ß$)\wedge($ß $\Rightarrow \alpha)$.

$(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$

Eliminate $\Rightarrow$, replacing $\alpha \Rightarrow$ ß with $\neg\, \alpha \vee$ ß.

$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,1})$

Move $\neg$ inwards using de Morgan's rules and double-negation:

$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1})$

Apply distributivity law ($\wedge$ over $\vee$) and flatten:

$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$

The DPLL algorithm

Determine if an input propositional logic sentence (in CNF) is satisfiable. This is just backtracking search for a CSP.

Improvements:

1. **Early termination**

   A clause is true if any literal is true.

   A sentence is false if any clause is false.

2. **Pure symbol heuristic**

   Pure symbol: always appears with the same "sign" in all clauses.

   e.g., In the three clauses ($A \vee \neg B$), ($\neg B \vee \neg C$), ($C \vee A$), A and B are pure, C is impure.

   Make a pure symbol literal true. (if there is a model for S, then making a pure symbol true is also a model).

3. **Unit clause heuristic**

   Unit clause: only one literal in the clause

   The only literal in a unit clause must be true.

   Note: literals can become a pure symbol or a unit clause when other literals obtain truth values. e.g.

   $(A \vee \cancel{True}) \wedge (\neg A \vee B)$

   $A = \cancel{pure}$

The DPLL algorithm

- Determine if an input propositional logic sentence (in CNF) is satisfiable by assigning values to variables.

1. **Pure symbol heuristic**

   Pure symbol: always appears with the same "sign" in all clauses.

   e.g., In the three clauses $(A \vee \neg B)$, $(\neg B \vee \neg C)$, $(C \vee A)$, A and B are pure, C is impure.

   Assign a pure symbol so that their literals are true.

2. **Unit clause heuristic**

   Unit clause: only one literal in the clause or only one literal which has not yet received a value. The only literal in a unit clause must be true.

The DPLL algorithm

```
function DPLL-SATISFIABLE?(s) returns true or false
   inputs: s, a sentence in propositional logic

   clauses ← the set of clauses in the CNF representation of s
   symbols ← a list of the proposition symbols in s
   return DPLL(clauses, symbols, [])
```

```
function DPLL(clauses, symbols, model) returns true or false

   if every clause in clauses is true in model then return true
   if some clause in clauses is false in model then return false
   P, value ← FIND-PURE-SYMBOL(symbols, clauses, model)
   if P is non-null then return DPLL(clauses, symbols–P, [P = value|model])
   P, value ← FIND-UNIT-CLAUSE(clauses, model)
   if P is non-null then return DPLL(clauses, symbols–P, [P = value|model])
   P ← FIRST(symbols); rest ← REST(symbols)
   return DPLL(clauses, rest, [P = true|model]) or
          DPLL(clauses, rest, [P = false|model])
```

The `WalkSAT` algorithm

- Incomplete, local search algorithm.
- **Evaluation function:** The min-conflict heuristic of minimizing the number of unsatisfied clauses.
- Steps are taken in the space of complete assignments, flipping the truth value of one variable at a time.
- Balance between greediness and randomness.
    - To avoid local minima

The `WalkSAT` algorithm

```
function WALKSAT(clauses, p, max-flips) returns a satisfying model or failure
   inputs: clauses, a set of clauses in propositional logic
           p, the probability of choosing to do a "random walk" move
           max-flips, number of flips allowed before giving up

   model ← a random assignment of true/false to the symbols in clauses
   for i = 1 to max-flips do
       if model satisfies clauses then return model
       clause ← a randomly selected clause from clauses that is false in model
       with probability p flip the value in model of a randomly selected symbol
              from clause
      else flip whichever symbol in clause maximizes the number of satisfied clauses
   return failure
```

Hard satisfiability problems

Consider random 3-CNF sentences. e.g.,

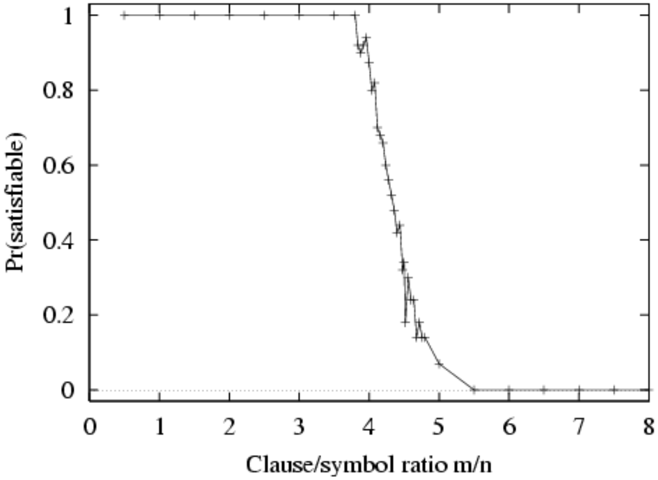
$(\neg D \lor \neg B \lor C) \land (B \lor \neg A \lor \neg C) \land (\neg C \lor \neg B \lor E) \land (E \lor \neg D \lor B) \land (B \lor E \lor \neg C)$

m = number of clauses

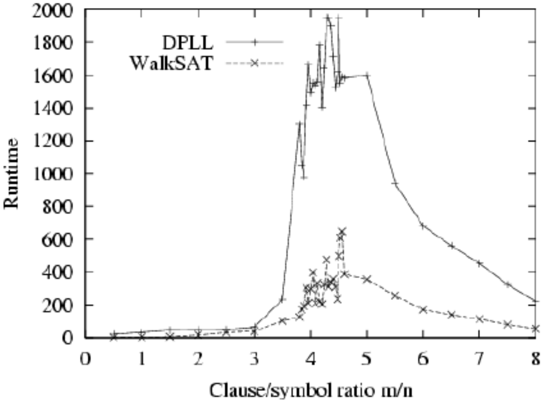
n = number of symbols

Hard problems seem to cluster near $m/n = 4.3$ (critical point)

Hard satisfiability problems



Hard satisfiability problems



Median runtime for 100 satisfiable random 3-CNF sentences, $n = 50$

Inference-based agents in the wumpus world

A wumpus-world agent using propositional logic:

$\neg P_{1,1}$

$\neg W_{1,1}$

$B_{x,y} \Leftrightarrow (P_{x,y+1} \vee P_{x,y-1} \vee P_{x+1,y} \vee P_{x-1,y})$

$S_{x,y} \Leftrightarrow (W_{x,y+1} \vee W_{x,y-1} \vee W_{x+1,y} \vee W_{x-1,y})$

$W_{1,1} \vee W_{1,2} \vee \ldots \vee W_{4,4}$

$\neg W_{1,1} \vee \neg W_{1,2}$

$\neg W_{1,1} \vee \neg W_{1,3}$

…

$\Rightarrow$ 64 distinct proposition symbols, 155 sentences

function PL-WUMPUS-AGENT(*percept*) **returns** an *action*
    inputs: *percept*, a list, [*stench*,*breeze*,*glitter*]
    static: *KB*, initially containing the "physics" of the wumpus world
        x, *y*, *orientation*, the agent's position (init. [1,1]) and orient. (init. *right*)
        visited, an array indicating which squares have been visited, initially *false*
        action, the agent's most recent action, initially null
        plan, an action sequence, initially empty

    update *x*,*y*,*orientation*, *visited* based on *action*
    if *stench* **then** TELL(KB, $S_{x,y}$) **else** TELL(KB, $\neg S_{x,y}$)
    if *breeze* **then** TELL(KB, $B_{x,y}$) **else** TELL(KB, $\neg B_{x,y}$)
    if *glitter* **then** *action* ← *grab*
    else if *plan* is nonempty then *action* ← POP(*plan*)
    else if for some fringe square [*i*,*j*], ASK(KB, $(\neg P_{i,j} \wedge \neg W_{i,j})$) is *true* **or**
            for some fringe square [*i*,*j*], ASK(KB, $(P_{i,j} \vee W_{i,j})$) is *false* **then do**
        plan ← A*-GRAPH-SEARCH(ROUTE-PB([*x*,*y*], *orientation*, [*i*,*j*],*visited*))
        action ← POP(*plan*)
    else *action* ← a randomly chosen move
    return *action*

Expressiveness limitation of propositional logic

- KB contains "physics" sentences for every single square

- For every time t and every location $[x,y]$,
- $L_{x,y} \wedge FacingRight^t \wedge Forward^t \Rightarrow L_{x+1,y}$

- Rapid proliferation of clauses

Maintaining Location and Orientation

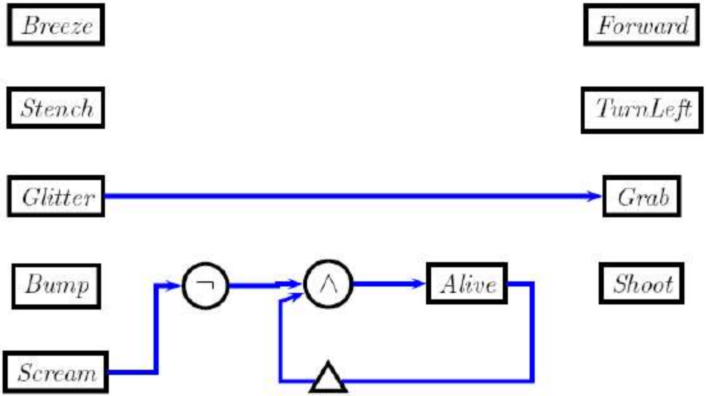
- KB contains "physics" sentences for every single square
- PL-Wumpus cheats – it keeps x,y & direction variables outside the KB.
- To keep them in the KB we would need propositional statement for every location. ***Also need to add time denotation to symbols***
- $\boldsymbol{L^{t}_{x,y} \wedge FacingRight^{\,t} \wedge Forward^{\,t} \Rightarrow L^{t}_{x+1,y}}$
- $\boldsymbol{FacingRight^{\,t} \wedge TurnLeft^{\,t} \Rightarrow FacingRight^{\,t+1}}$
- We need these statements in the initial KB for every location and for every time.
- This is tens of thousands of statements for time steps of [0,100]

Circuit-based Agents

- Reflex agent with State.
- Formed of logical gates and registers (stores a value)
- **Inputs** are registers **holding current percepts**
- **Outputs** are registers **giving the action to take**
- At each time step, **inputs are set and signals propagate through the circuit**
- Handles time **'more satisfactorily'** than previous agent. No need for a hundreds of rules encoding states

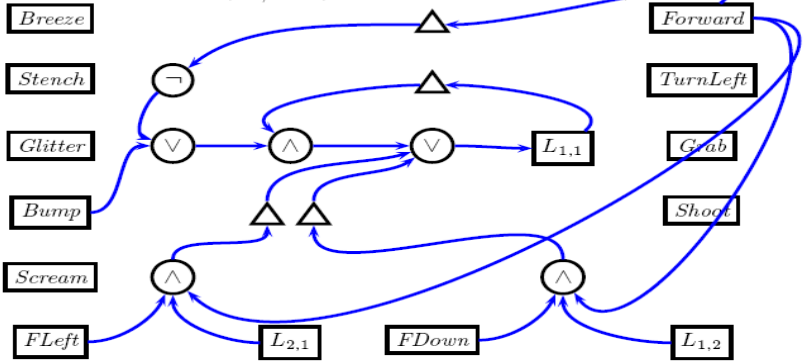
Example Circuit

$$Glitter^t \Rightarrow Grab^t$$
$$\neg Scream^t \wedge Alive^{t-1} \Rightarrow Alive^t$$



Location Circuit

$$
\begin{aligned}
L_{2,1}^{t} \quad \Leftrightarrow \quad & (L_{1,1}^{t-1} \wedge (\neg Forward^{t-1} \vee Bump^{t})) \\
\vee \quad & (L_{1,2}^{t-1} \wedge (FDown^{t-1} \wedge Forward^{t-1})) \\
\vee \quad & (L_{2,1}^{t-1} \wedge (FLeft^{t-1} \wedge Forward^{t-1}))
\end{aligned}
$$



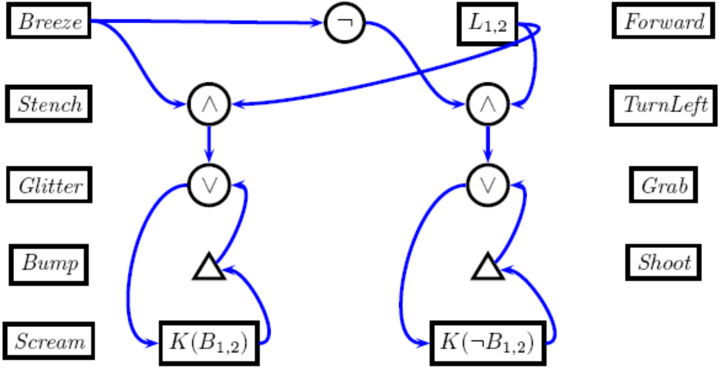
Need a similar circuit for each location register.

Unknown Information in Circuits

- **Propositions *Alive* and $L^t_{x+1,y}$ are always known**
- What about $\boldsymbol{B_{1,2}}$ **?** Unknown at the beginning of Wumpus world simulation. This is OK in a propositional KB, but not OK in a circuit.
- **Use two bits K($B_{1,2}$) and K($\neg B_{1,2}$)**
- **If both are false we know nothing! One of them is set by visiting the square.**
- $K(B_{1,2})^t \Leftrightarrow K(B_{1,2})^{t-1} \vee (L^t_{1,2} \wedge \text{Breeze}^t)$
- Pit in 4,4?
- $K(\neg P_{4,4})^t \Leftrightarrow K(\neg B_{3,4})^t \vee K(\neg B_{4,3})^t$
- $K(P_{4,4})^t \Leftrightarrow (K(B_{3,4})^t \wedge K(\neg P_{2,4})^t \wedge K(\neg P_{3,3})^t)$
- $\vee\ (K(B_{4,3})^t \wedge K(\neg P_{4,2})^t \wedge K(\neg P_{3,3})^t)$
- **Hairy Circuits, but still only a constant number of gates**
- ***Note: Assume that Pits cannot be close enough together such that you can build a counter example to* K($B_{1,2}$)^t *above***

Example of 2-bit K(x) usage

$$K(B_{1,2})^t \Leftrightarrow K(B_{1,2})^{t-1} \vee (L^t_{1,2} \wedge Breeze^t)$$
$$K(\neg B_{1,2})^t \Leftrightarrow K(\neg B_{1,2})^{t-1} \vee (L^t_{1,2} \wedge \neg Breeze^t)$$



Avoid cyclic circuits

So far all 'feedback' loops have a delay. Why? Otherwise the circuit would go from being acyclic to cyclic

Physical cyclic circuits do not work and/or are unstable.

$K(B_{4,4})^t \Leftrightarrow K(B_{4,4})^{t-1} \vee (L^t_{4,4} \wedge Breeze^t) \vee K(P_{3,4})^t \vee K(P_{4,3})^t$

$K(P_{3,4})^t$ and $K(P_{4,3})^t$ depend on breeziness in adjacent pits, and pits depend on more adjacent breeziness. The circuit would contain cycles

These statements are not wrong, just not representable in a boolean circuit

Thus the corrected acyclic (using direct observation) version is incomplete. The Circuit-based agent might know less than the corresponding inference based agent at that time

Example: $B_{1,1} \Rightarrow B_{2,2}$. This is OK for IBA, but not for CBA

A complete circuit can be built, but it would be much more complex

IBA vs. CBA

Conciseness: Neither deals with time very well. Both are very verbose in their own way. Adding more complex objects will swamp both types. Both are poorly suited to path-finding between safe squares (PL-Wumpus uses A* search to get around this)

Computational Efficiency: Inference can take exponential time in the number of symbols. Evaluating a circuit is linear in size/depth of circuit. However in practice good inference algorithms are very quick

Completeness: The incompleteness of CBA is deeper than acyclicity. For some environments a complete circuit must be exponentially larger than the IBA's KB to execute in linear time. CBAs also forgets knowledge learned in previous times

Ease: Both agents can require lots and lots of work to build. Many seemingly redundant statements or very large and ugly circuits.

Hybrid???

Summary

Logical agents apply inference to a knowledge base to derive new information and make decisions

Basic concepts of logic:

- syntax: formal structure of sentences
- semantics: truth of sentences wrt models
- entailment: necessary truth of one sentence given another
- inference: deriving sentences from other sentences
- soundness: derivations produce only entailed sentences
- completeness: derivations can produce all entailed sentences

Wumpus world requires the ability to represent partial and negated information, reason by cases, etc.

Resolution is complete for propositional logic

Forward, backward chaining are linear-time, complete for Horn clauses

Propositional logic lacks expressive power

Local Search Algorithms and Optimization Problems

- **Local search:**
  - Use single current state and move to neighboring states.
- **Idea: start with an initial guess at a solution and incrementally improve it until it is one**
- **Advantages:**
  - Use very little memory
  - Find often *reasonable* solutions in large or infinite state spaces.
- **Useful for pure optimization problems.**
  - Find or approximate best state according to some *objective function*
  - ***Optimal if the space to be searched is convex***

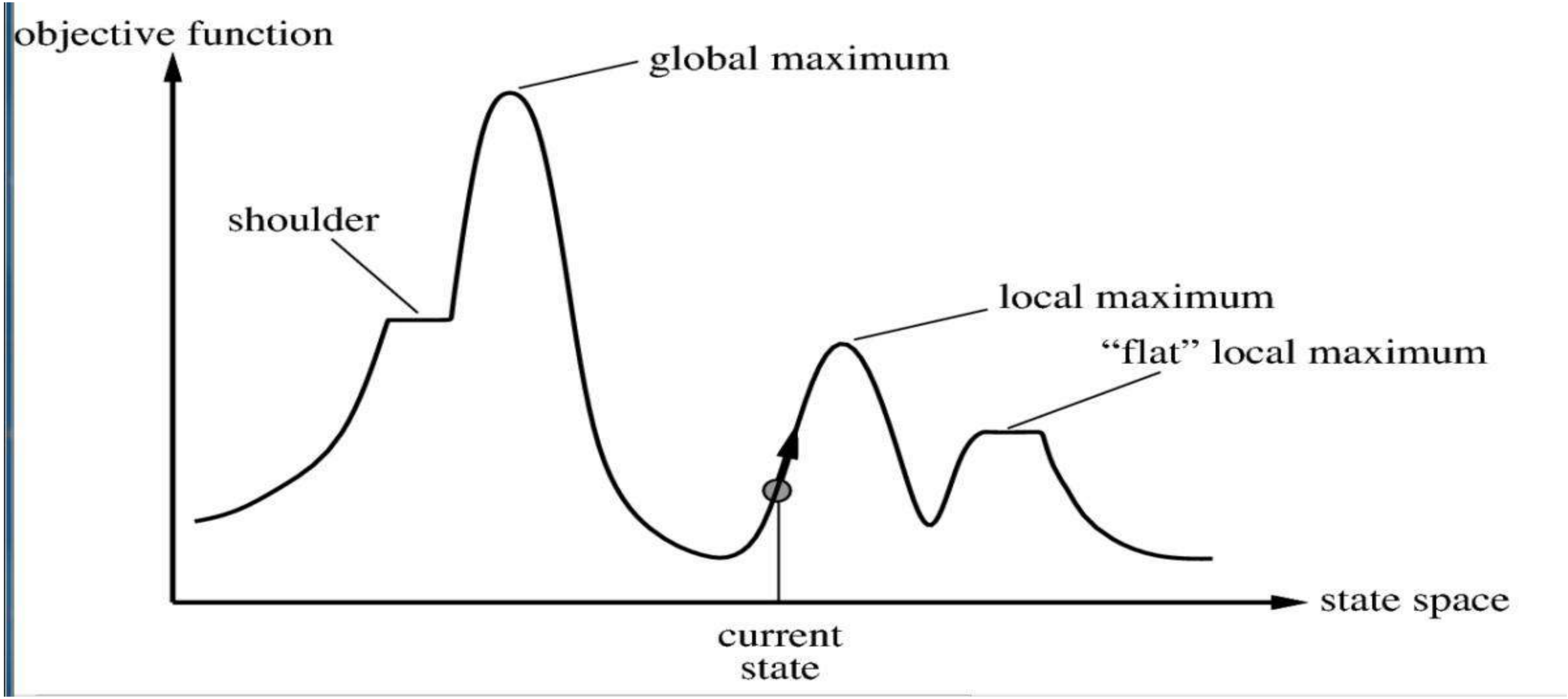
Local search vs Systematic search

	Systematic search	Local search
Solution	Path from initial state to the goal	Solution state itself
Method	Systematically trying different paths from an initial state	Keeping a single or more "current" states and trying to improve them
State space	Usually incremental	Complete configuration
Memory	Usually very high	Usually very little (constant)
Time	Finding optimal solutions in small state spaces	Finding reasonable solutions in large or infinite (continuous) state spaces
Scope	Search	Search & optimization problems

Understand Local Search

- ➢**State- Space Landscape**
- ➢Landscape
  - --**Location**(defined by state)
  - --**Elevation**(defined by heuristic function)
- ➢If Elevation corresponds to
  - --**Cost**-Find Lowest Valley (Global Minimum)
  - --**Objective Function**-Find Highest Peak(Global Maximum)
- ➢A Complete Local Search Algorithm always **finds a Goal if one exists, Optimal Solution always finds a Global Maximum/Minimum.**

State- Space Landscape Features



Local Search Algorithms

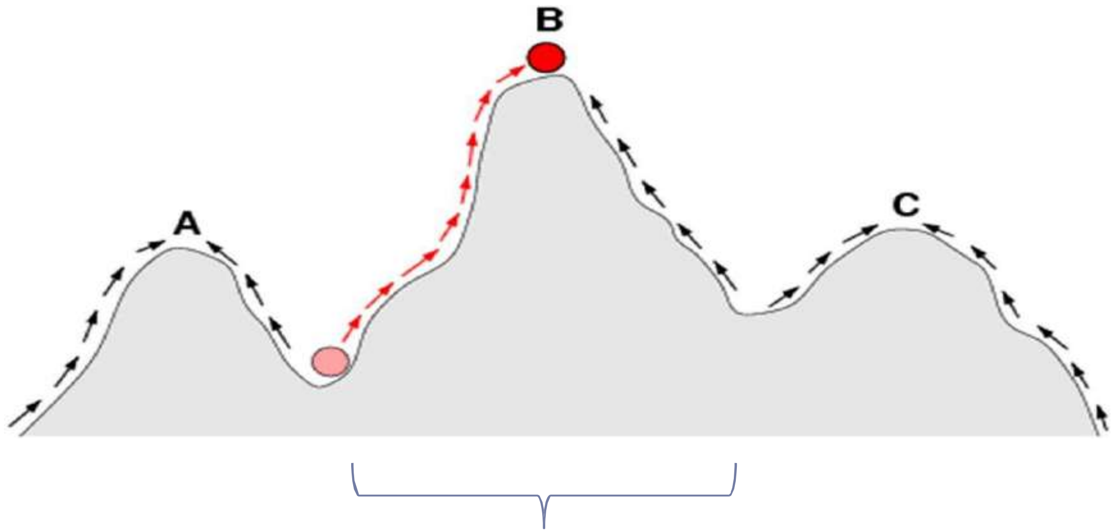
- **Hill Climbing Search**
- **Simulated Annealing Search**
- **Local Beam Search**
- **Genetic Algorithms**

Hill Climbing Search

- Local Search Algorithm
- **Steepest-Ascent** (Simply a Loop that Continuously moves in direction of increasing value-**UPHILL**)
- **Terminates when reaches Peak**(No Neighbour has higher value)
- **Does not maintain a search tree**(only Current State)
- **No Back Tracking**
- **Greedy Local Search**(grabs good neighbour without thinking other)

Hill-climbing search is greedy

- Greedy local search: considering only one step ahead and select the best successor state (steepest ascent)
- Rapid progress toward a solution
- Usually quite easy to improve a bad solution



Optimal when starting in one of these states

Hill Climbing - Algorithm

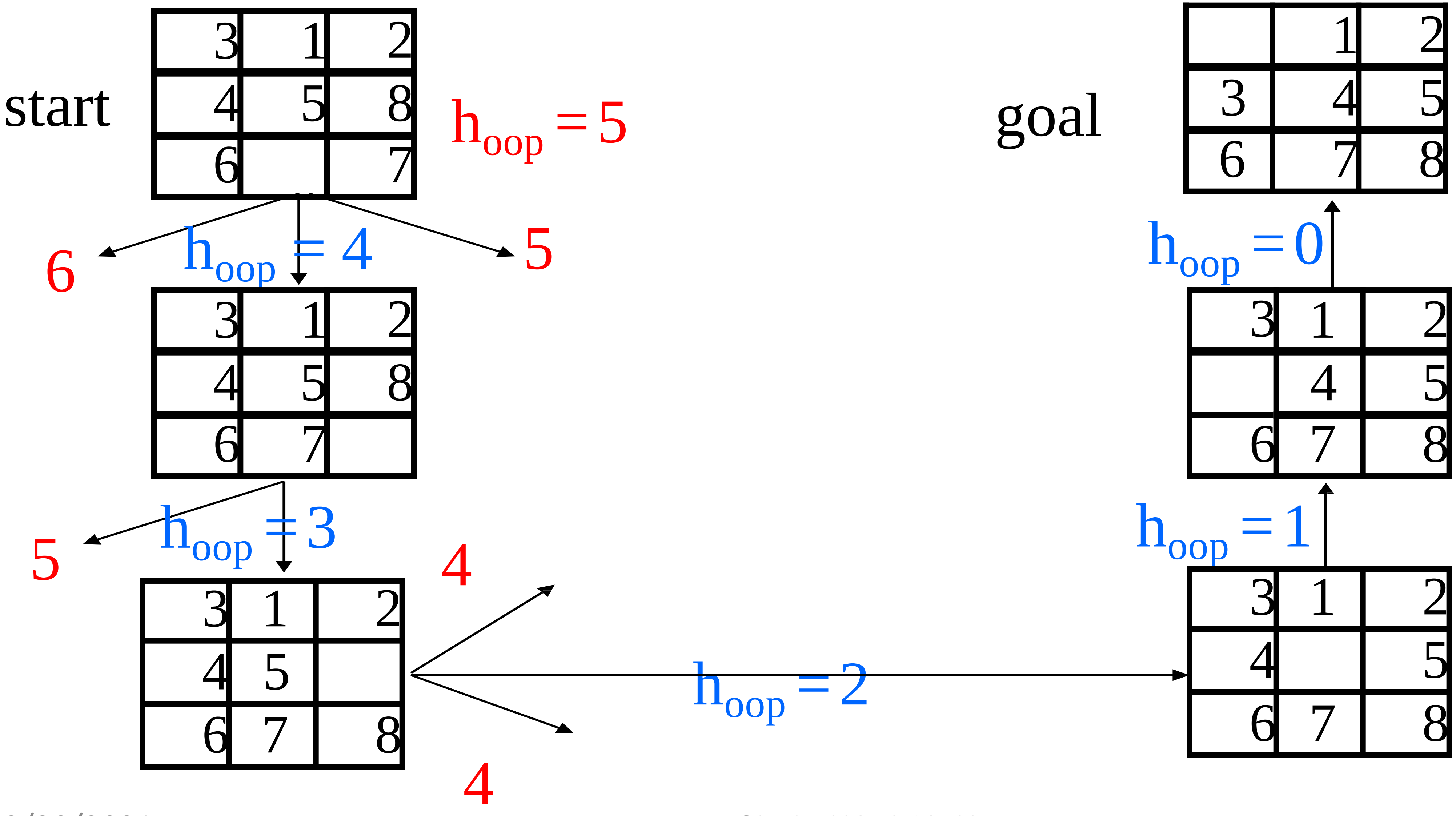
1. **Pick a random point in the search space**
2. **Consider all the neighbors of the current state**
3. **Choose the neighbor with the best quality and move to that state.**
4. **Repeat 2 to 4 until all the neighboring states are of lower quality.**
5. **Return the current state as the solution state.**

```
function HILL-CLIMBING(problem) returns a state that is a local maximum

  current ← MAKE-NODE(problem.INITIAL-STATE)
  loop do
     neighbor ← a highest-valued successor of current
     if neighbor.VALUE ≤ current.VALUE then return current.STATE
     current ← neighbor
```

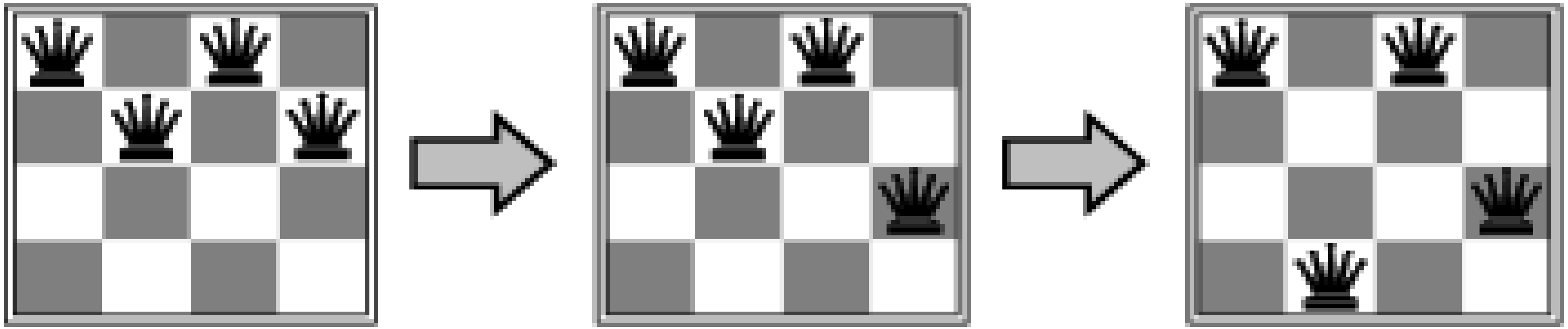
Figure 4.2 The hill-climbing search algorithm, which is the most basic local search technique. At each step the current node is replaced by the best neighbor; in this version, that means the neighbor with the highest VALUE, but if a heuristic cost estimate h is used, we would find the neighbor with the lowest h.

Hill climbing example 1 (*minimizing* h)



Hill-climbing Example: *n*-queens

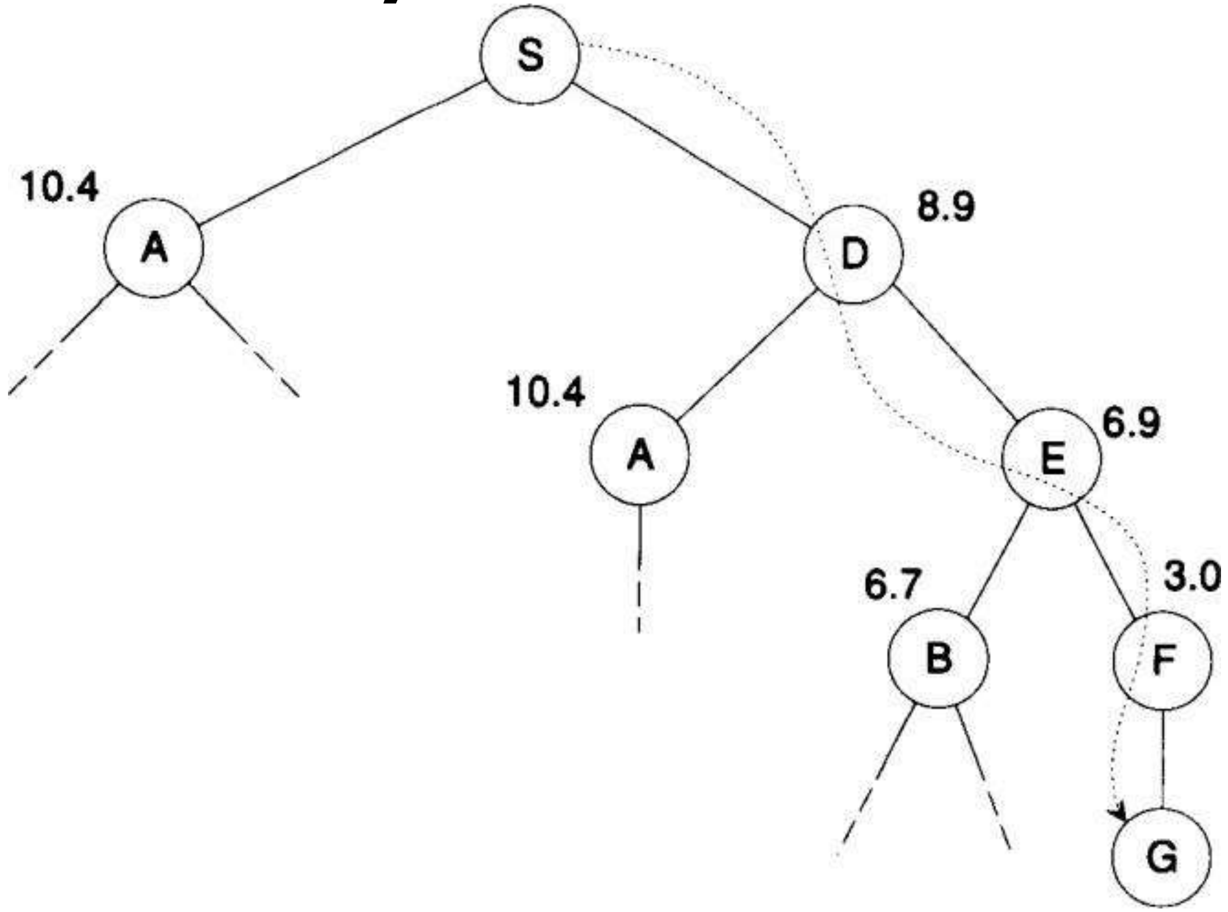
- ***n*-queens problem:** **Put *n* queens on an *n* × *n* board with no two queens on the same row, column, or diagonal**
- ***Good heuristic:*** ***h* = number of pairs of queens that are attacking each other**



h=5 *h=3 (for illustration)* *h=1*

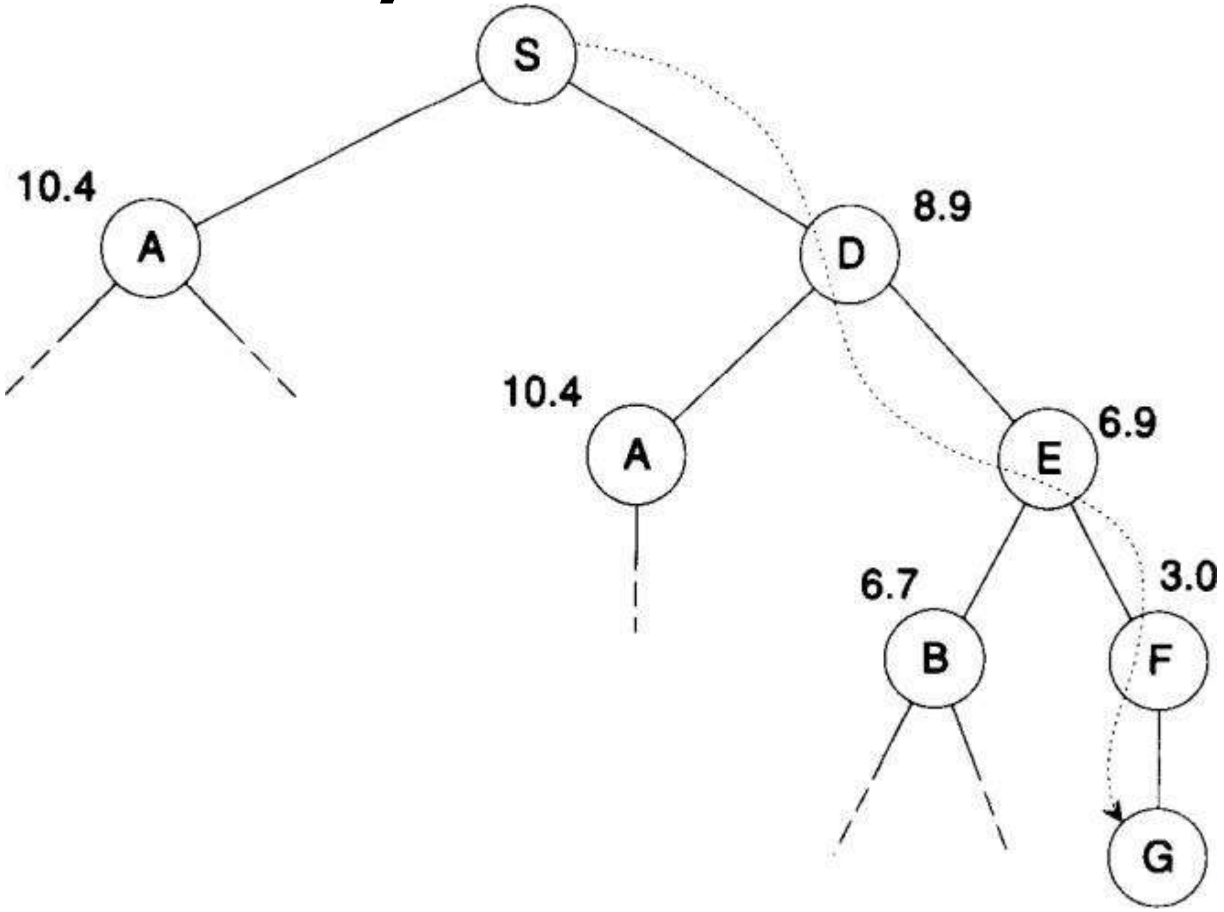
Hill Climbing Goal - Node G

Current	Children
S	---

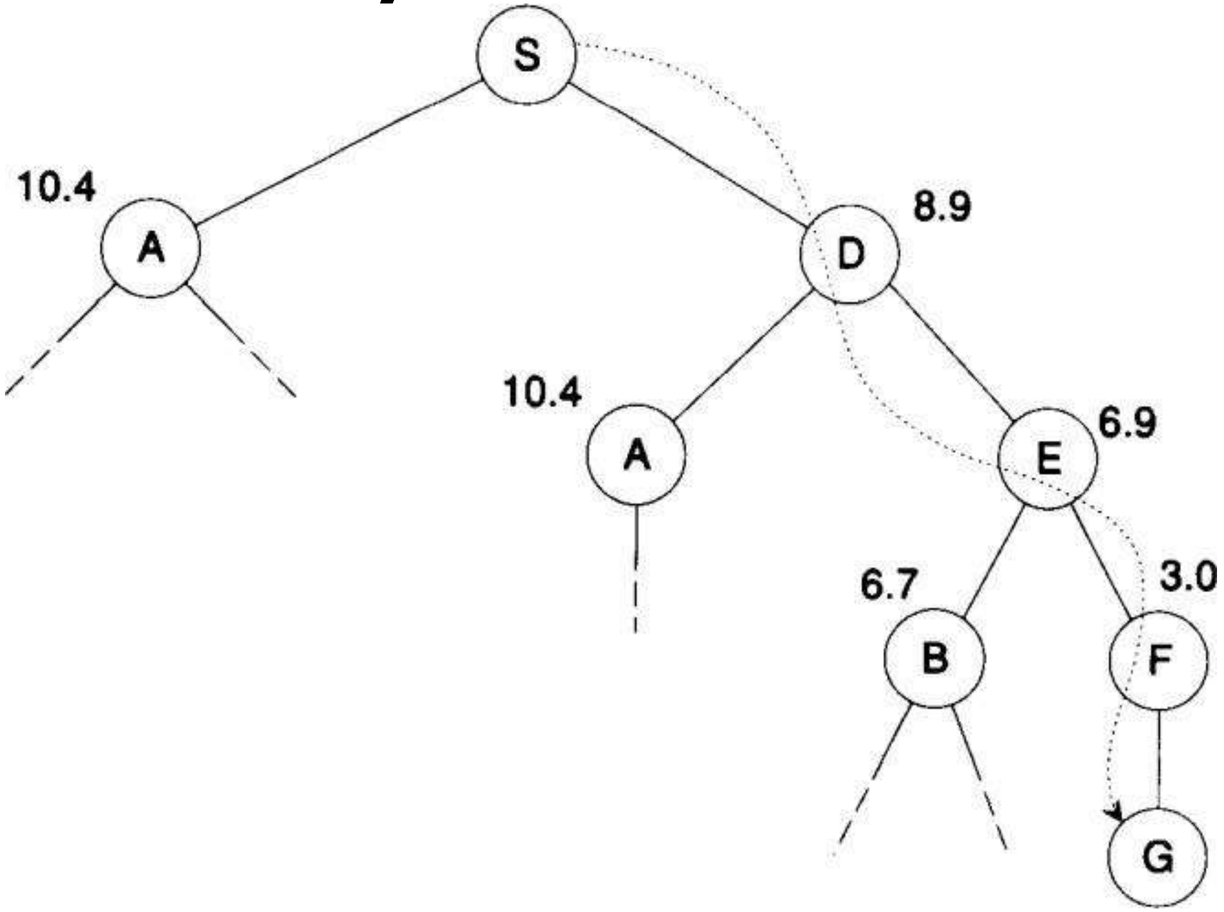


Hill Climbing
Goal - Node

Current	Children
S	---

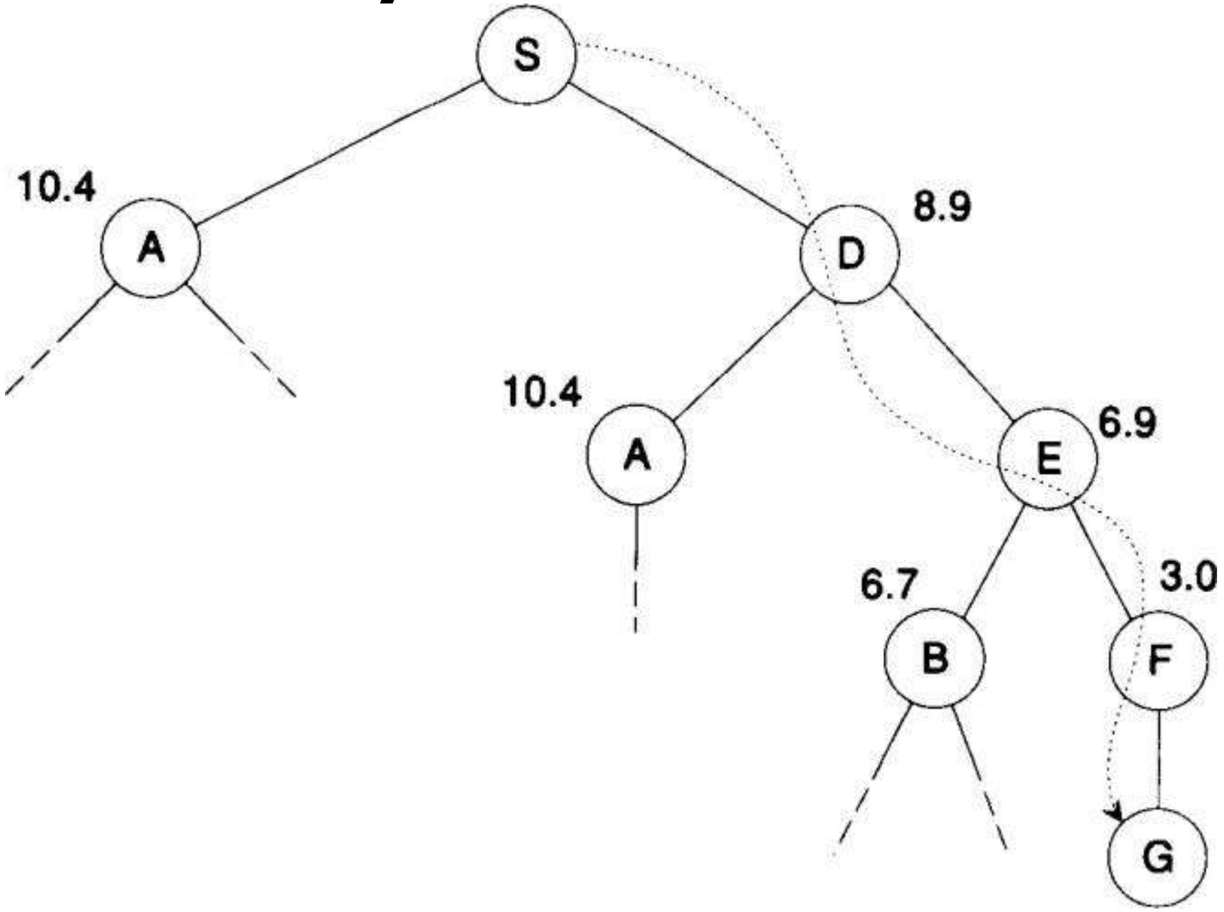


Hill Climbing
Goal - Node



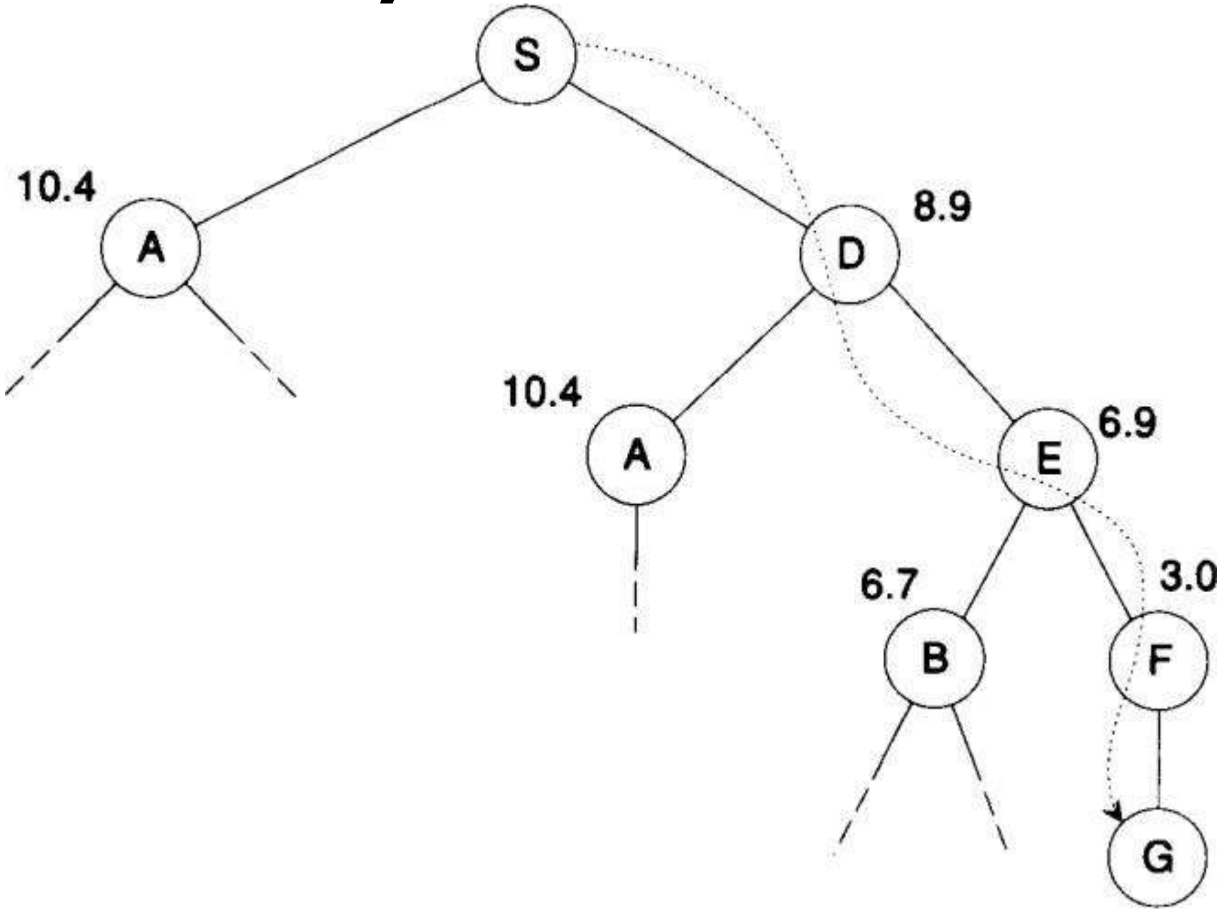
Current	Children
S	---
S	

Hill Climbing
Goal – Node



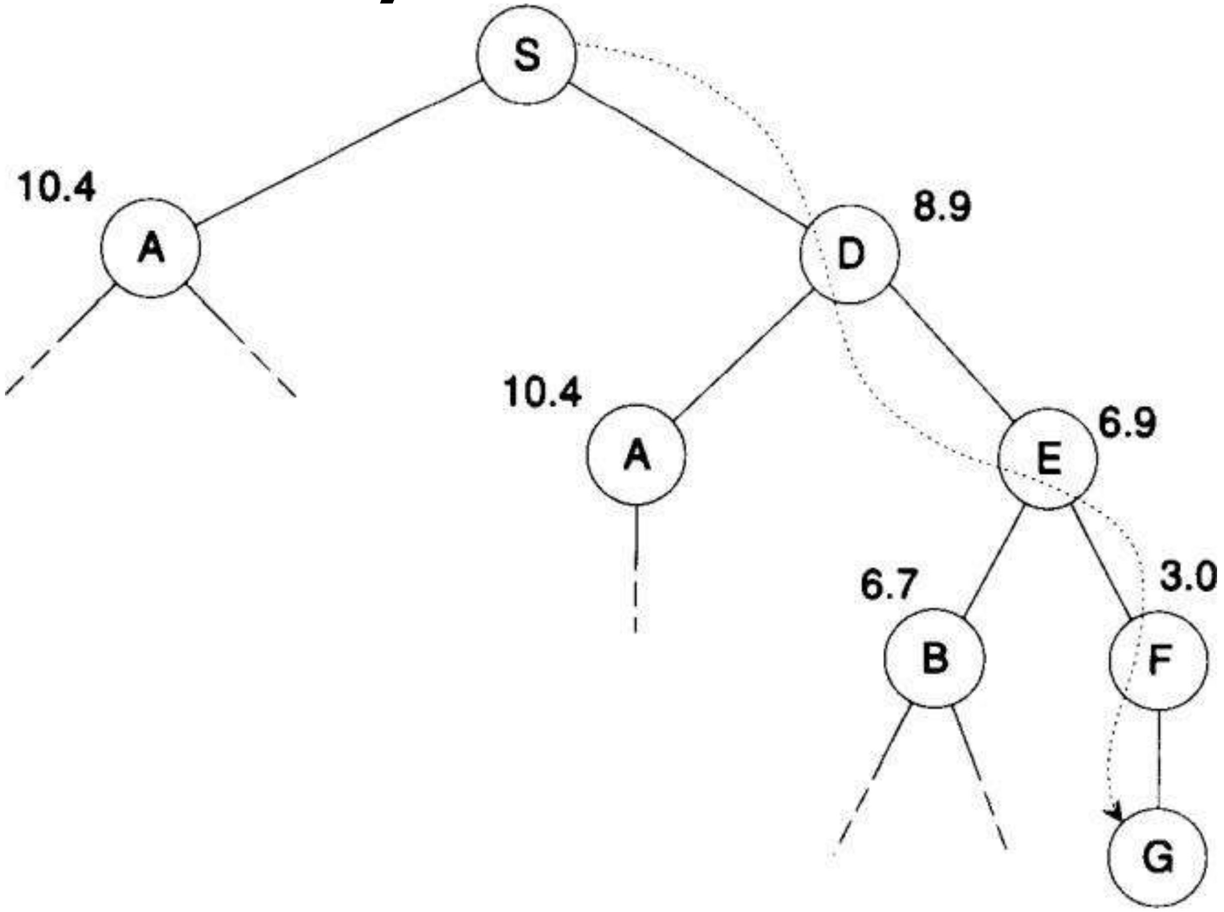
Current	Children
S	---
S	$D_{8.9}, A_{10.4}$

Hill Climbing
Goal - Node



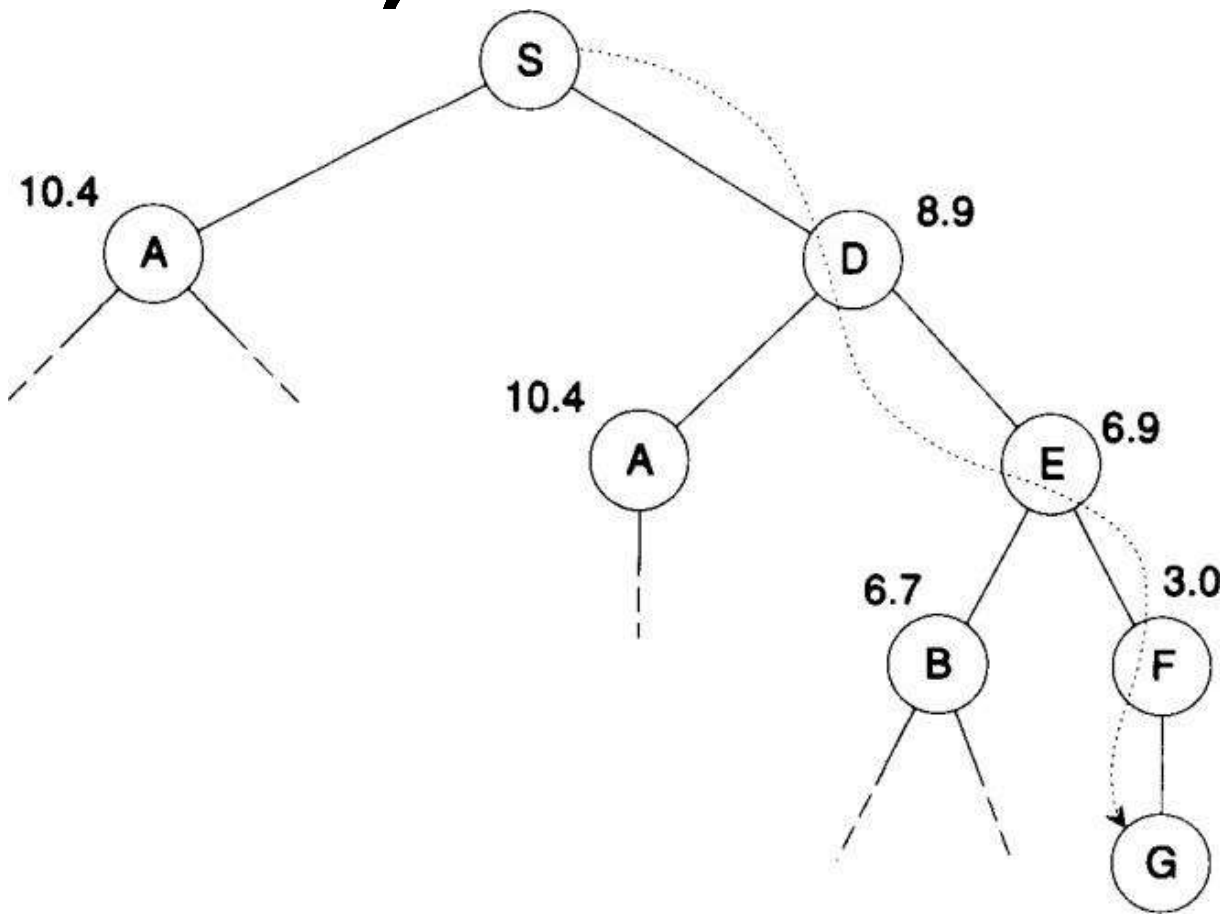
Current	Children
S	---
S	$D_{8.9}$, $A_{10.4}$
D	

Hill Climbing Goal - Node G



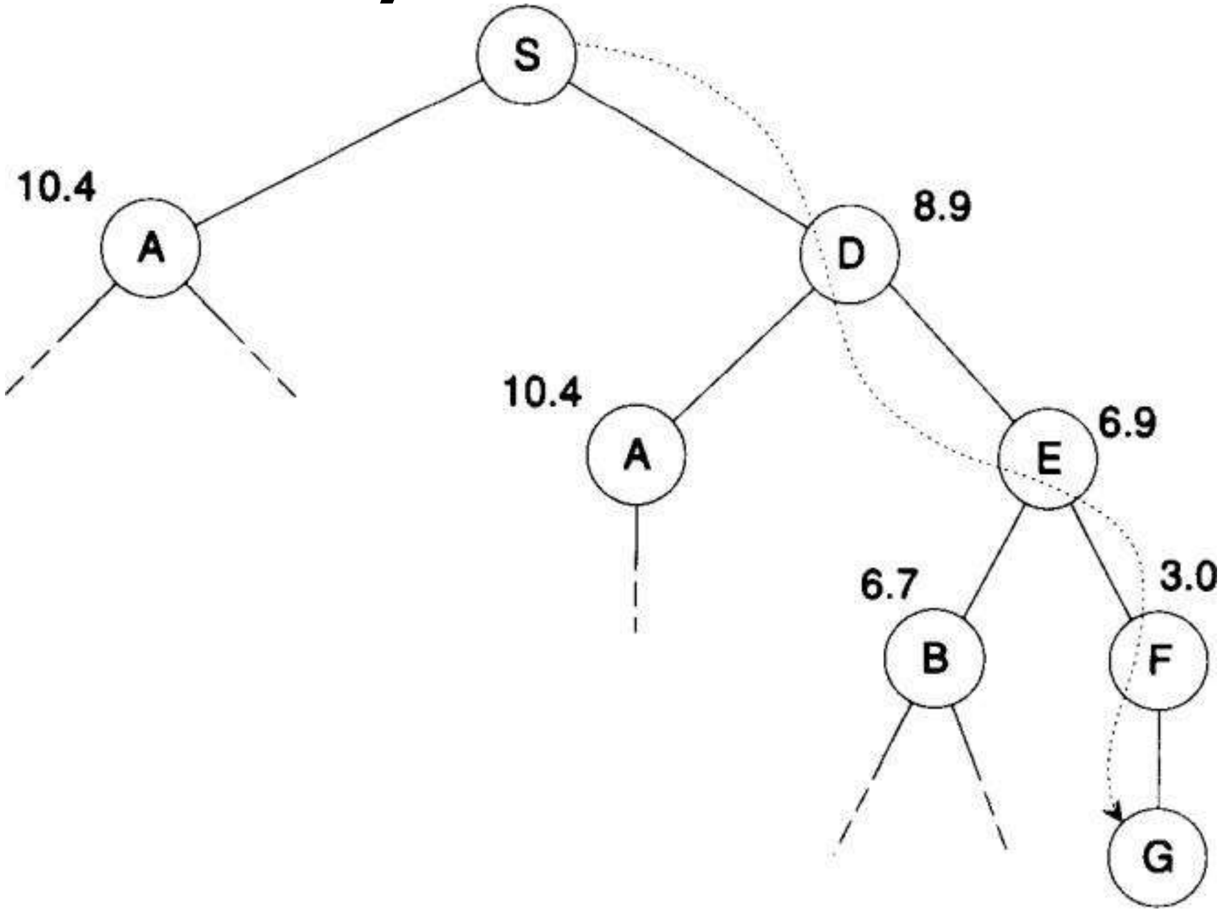
Current	Children
S	---
S	$D_{8.9}$, $A_{10.4}$
D	---

Hill Climbing Goal – Node



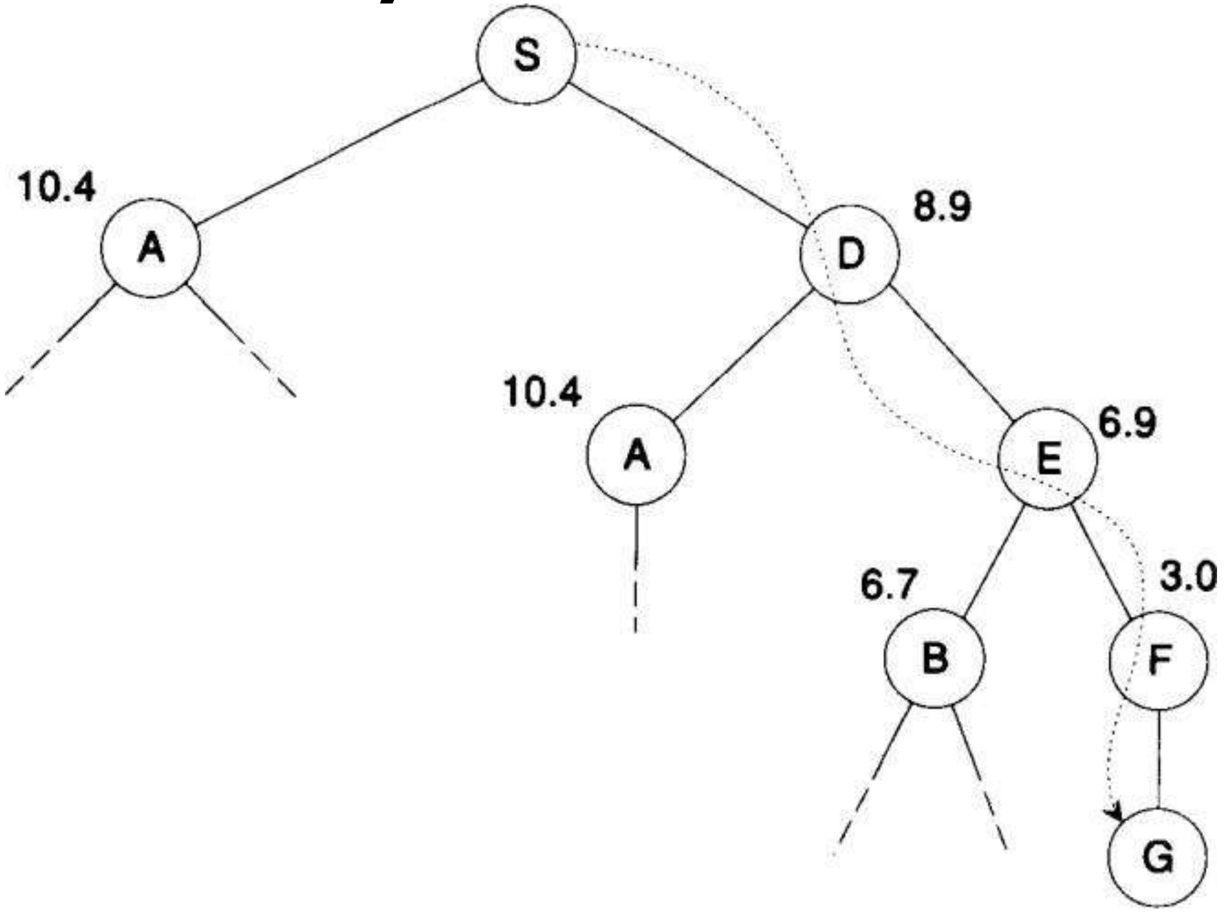
Current	Children
S	---
S	$D_{8.9}$, $A_{10.4}$
D	---

Hill Climbing Goal - Node



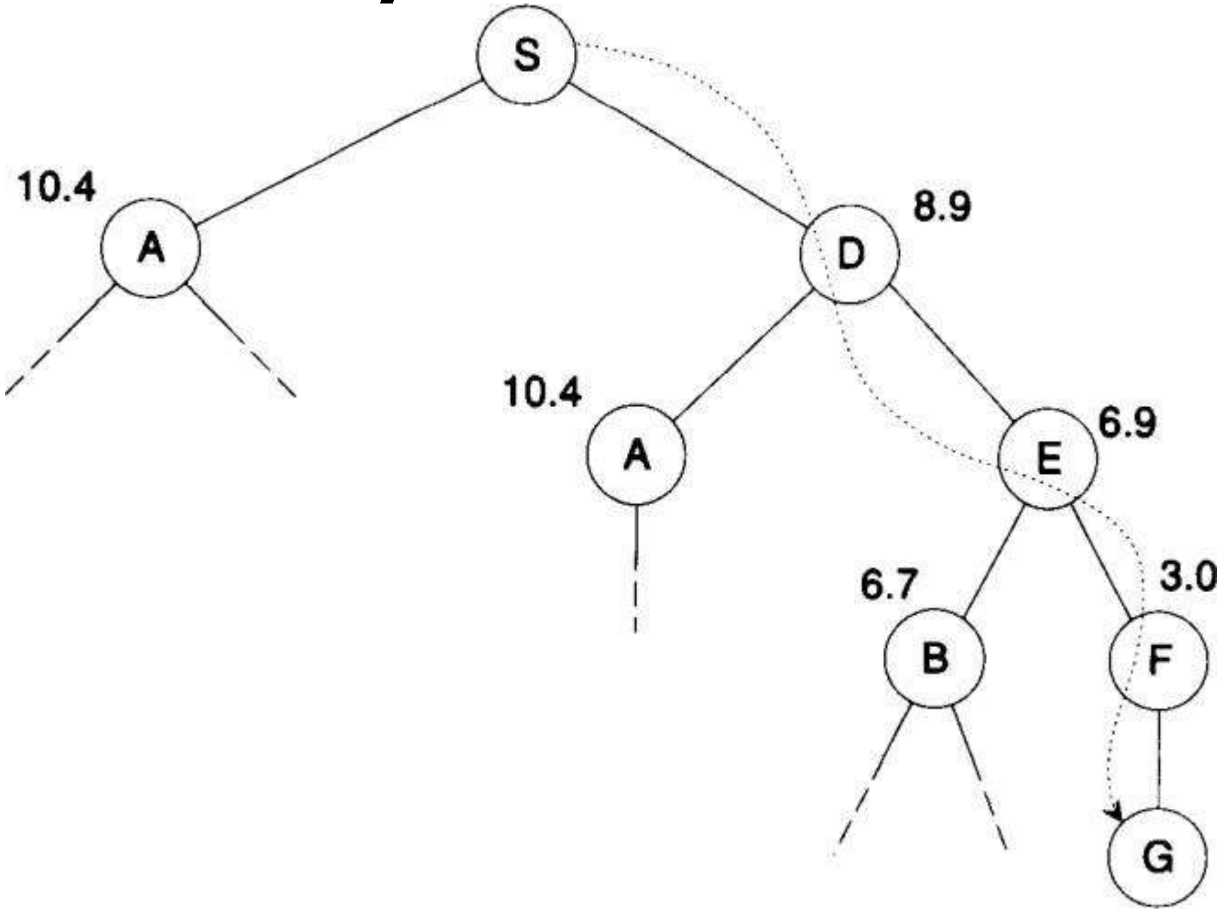
Current	Children
S	---
S	$D_{8.9}, A_{10.4}$
D	---
D	

Hill Climbing Goal - Node



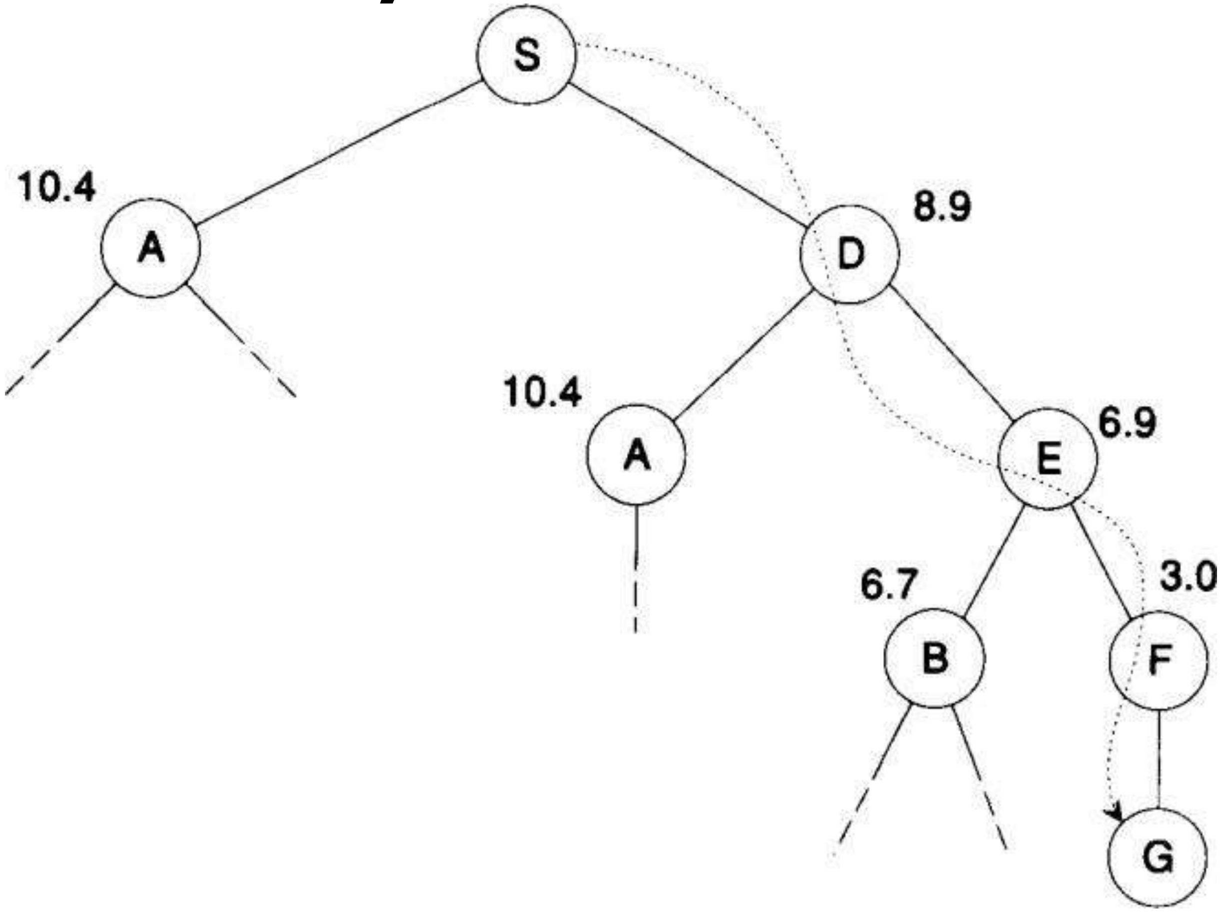
Current	Children
S	---
S	$D_{8.9}, A_{10.4}$
D	---
D	$E_{6.9}, A_{10.4}$

Hill Climbing
Goal – Node



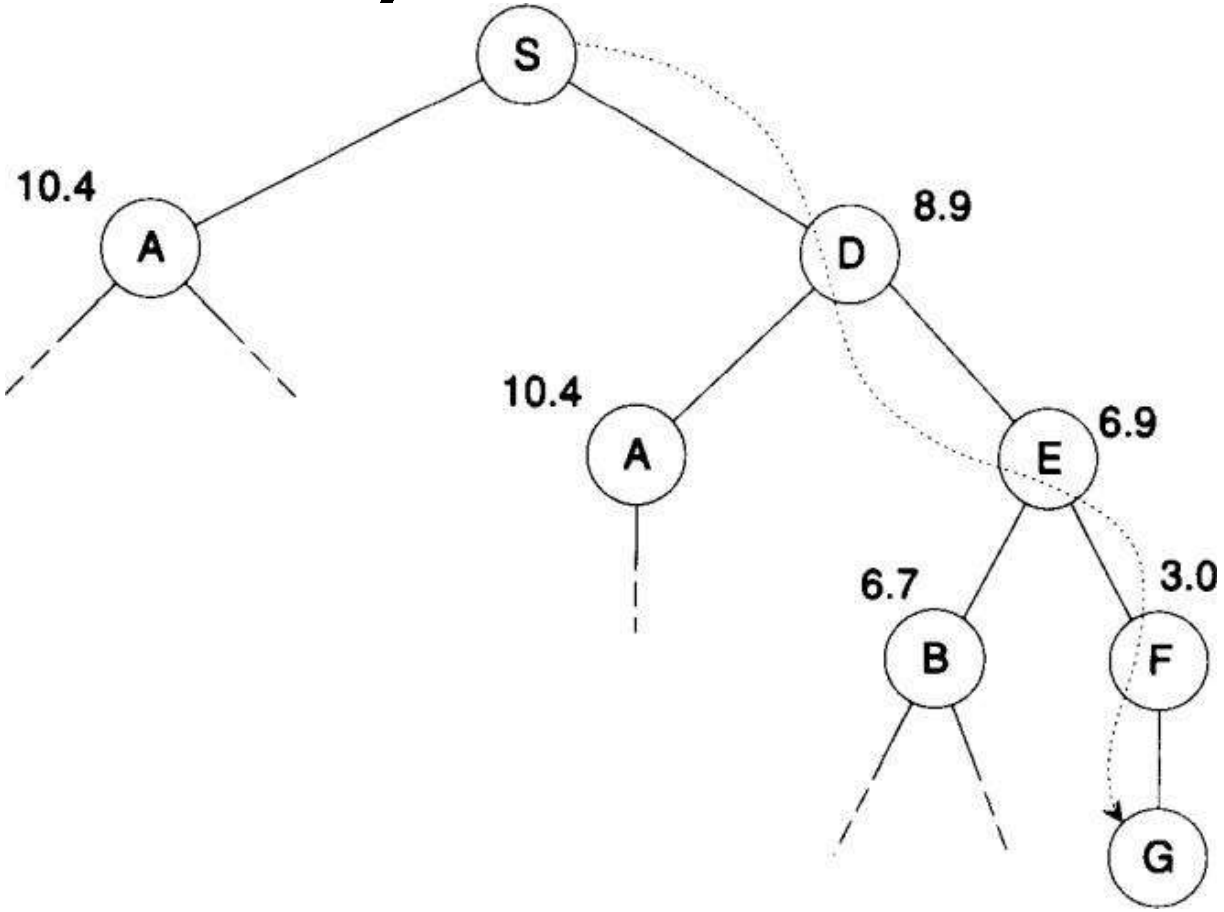
Current	Children
S	---
S	$D_{8.9}, A_{10.4}$
D	---
D	$E_{6.9}, A_{10.4}$
E	

Hill Climbing Goal – Node



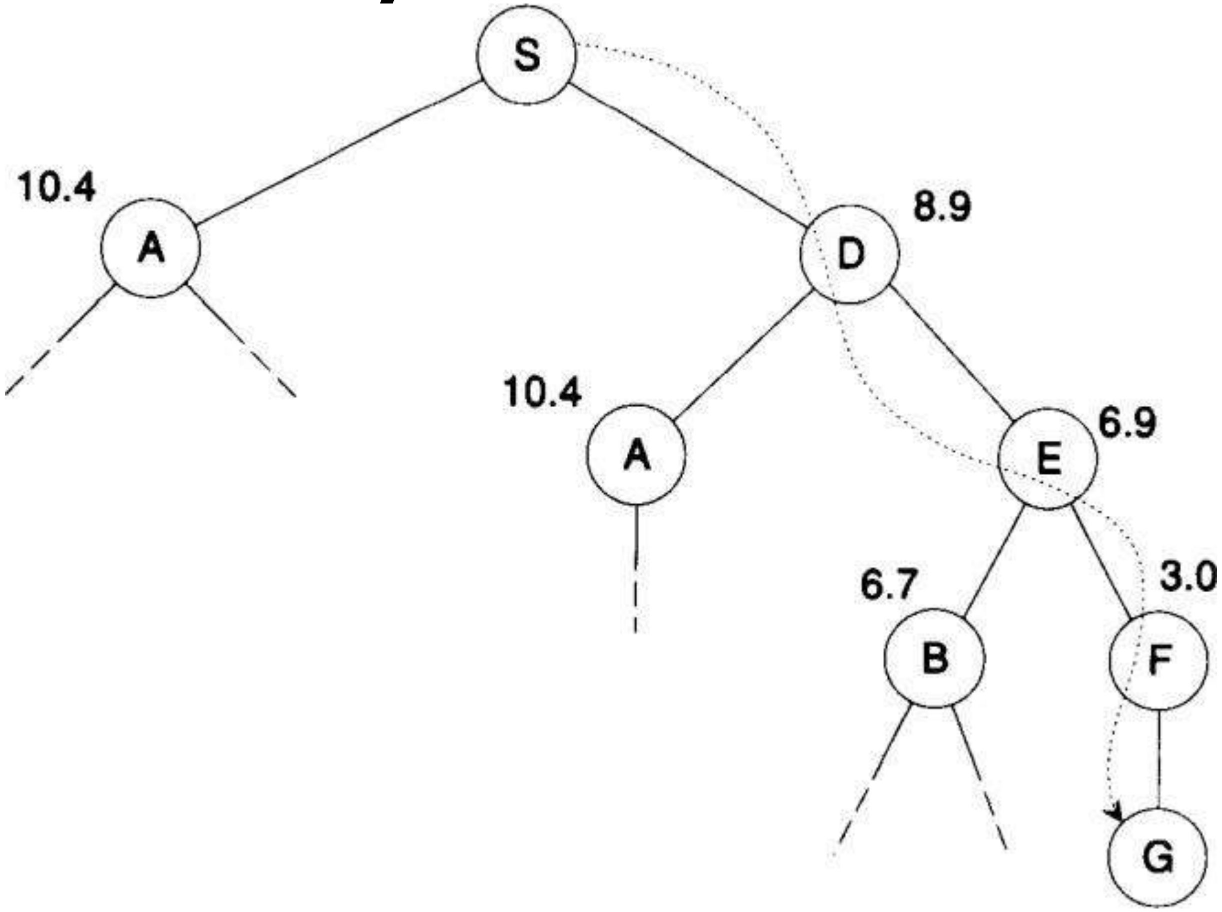
Current	Children
S	---
S	$D_{8.9}$, $A_{10.4}$
D	---
D	$E_{6.9}$, $A_{10.4}$
E	---

Hill Climbing Goal – Node



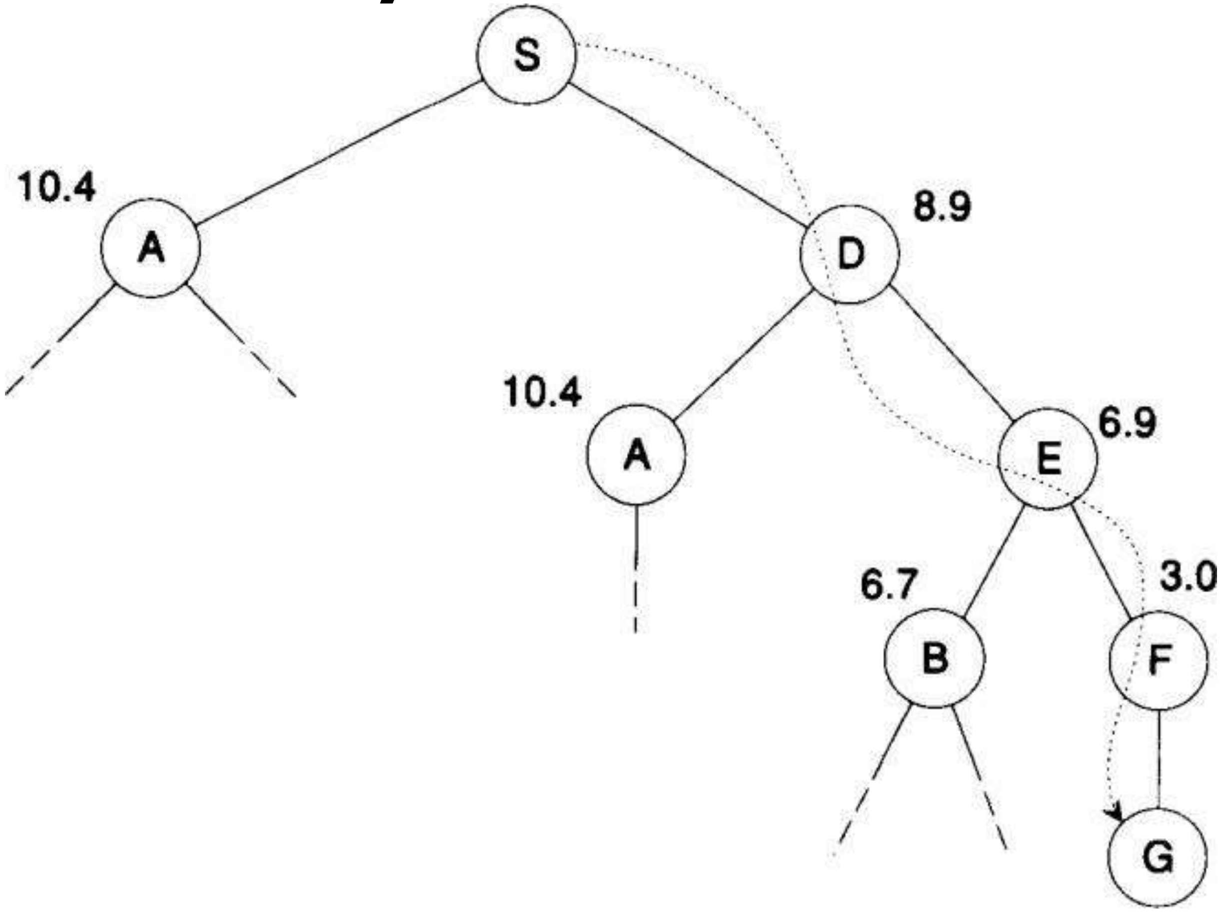
Current	Children
S	---
S	$D_{8.9}, A_{10.4}$
D	---
D	$E_{6.9}, A_{10.4}$
E	---

Hill Climbing Goal – Node



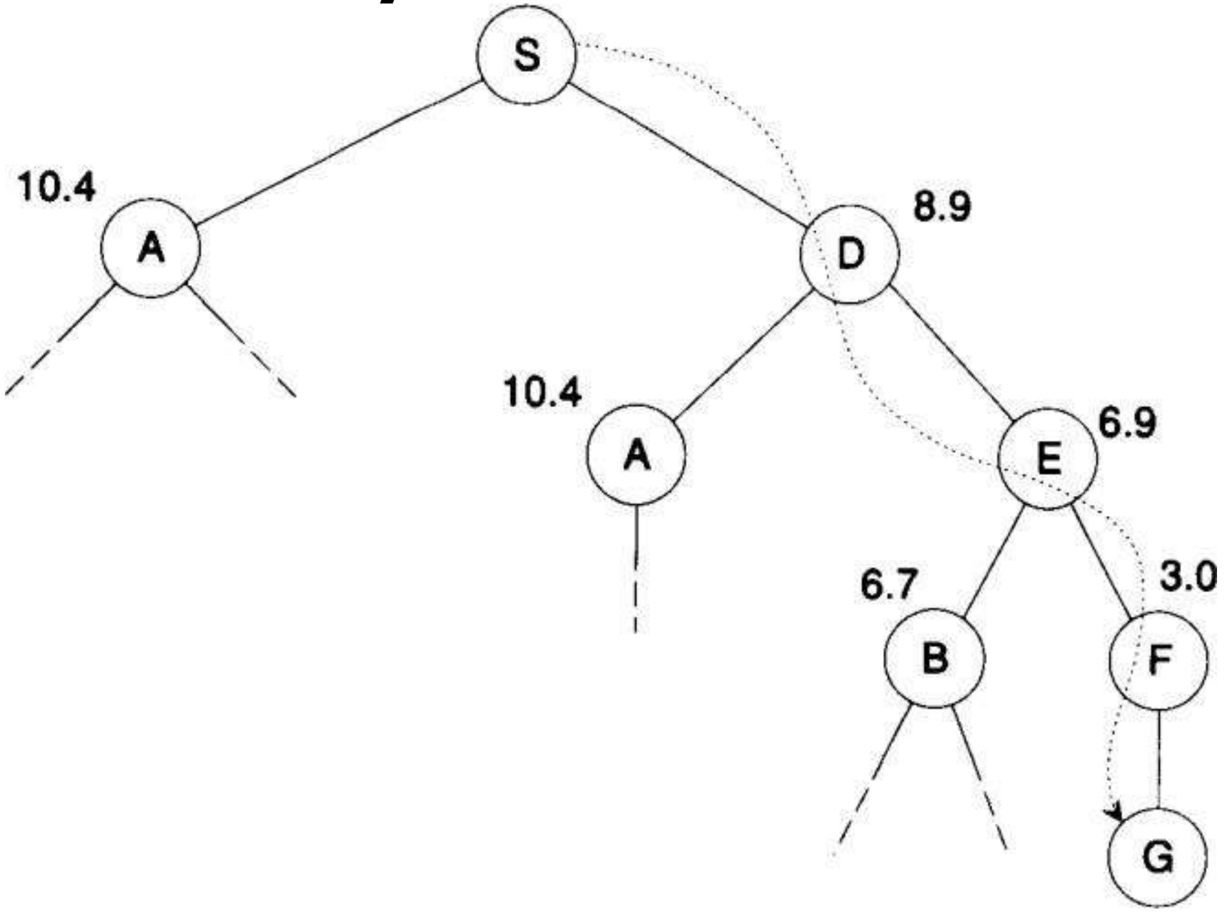
Current	Children
S	---
S	$D_{8.9}, A_{10.4}$
D	---
D	$E_{6.9}, A_{10.4}$
E	---
E	

Hill Climbing Goal - Node



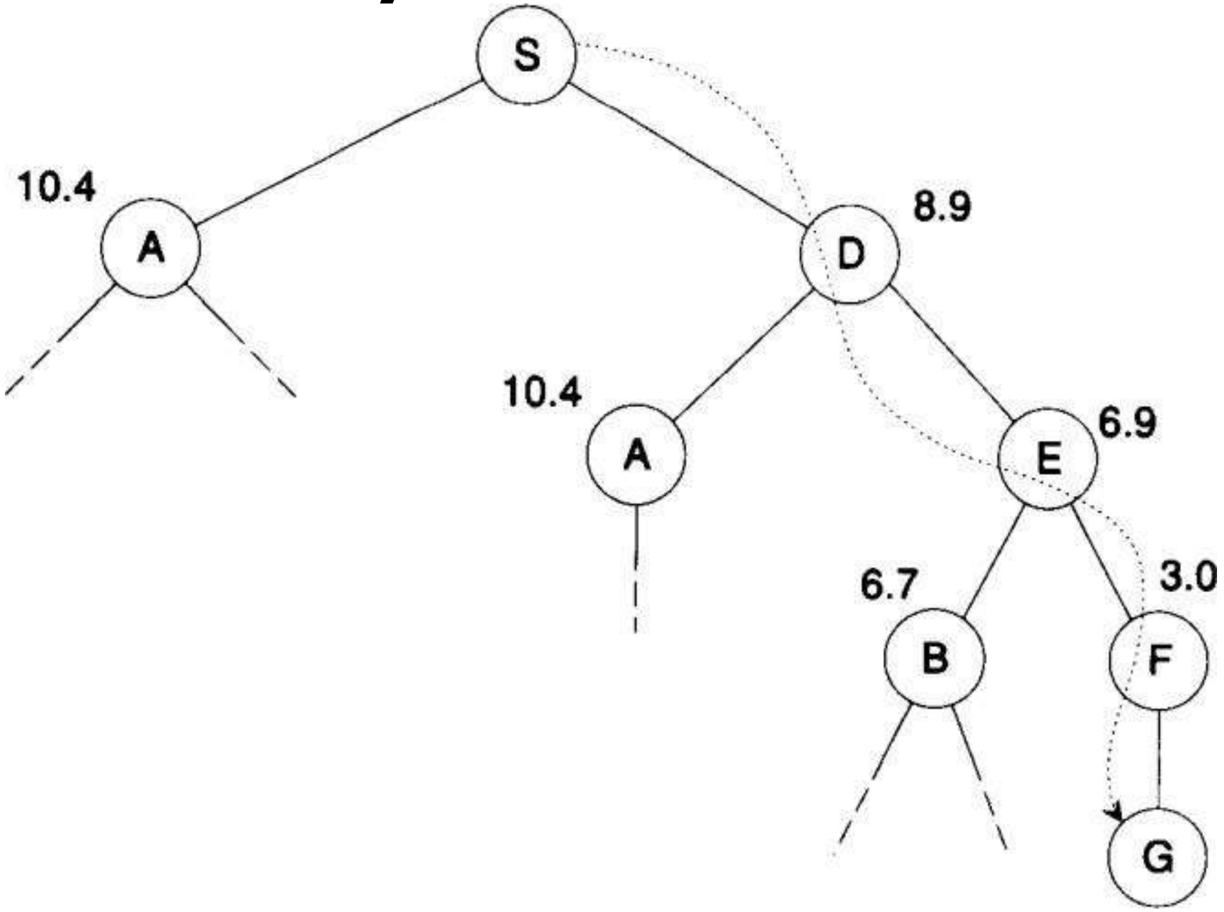
Current	Children
S	---
S	$D_{8.9}$, $A_{10.4}$
D	---
D	$E_{6.9}$, $A_{10.4}$
E	---
E	$F_{3.0}$, $B_{6.7}$

Hill Climbing Goal – Node



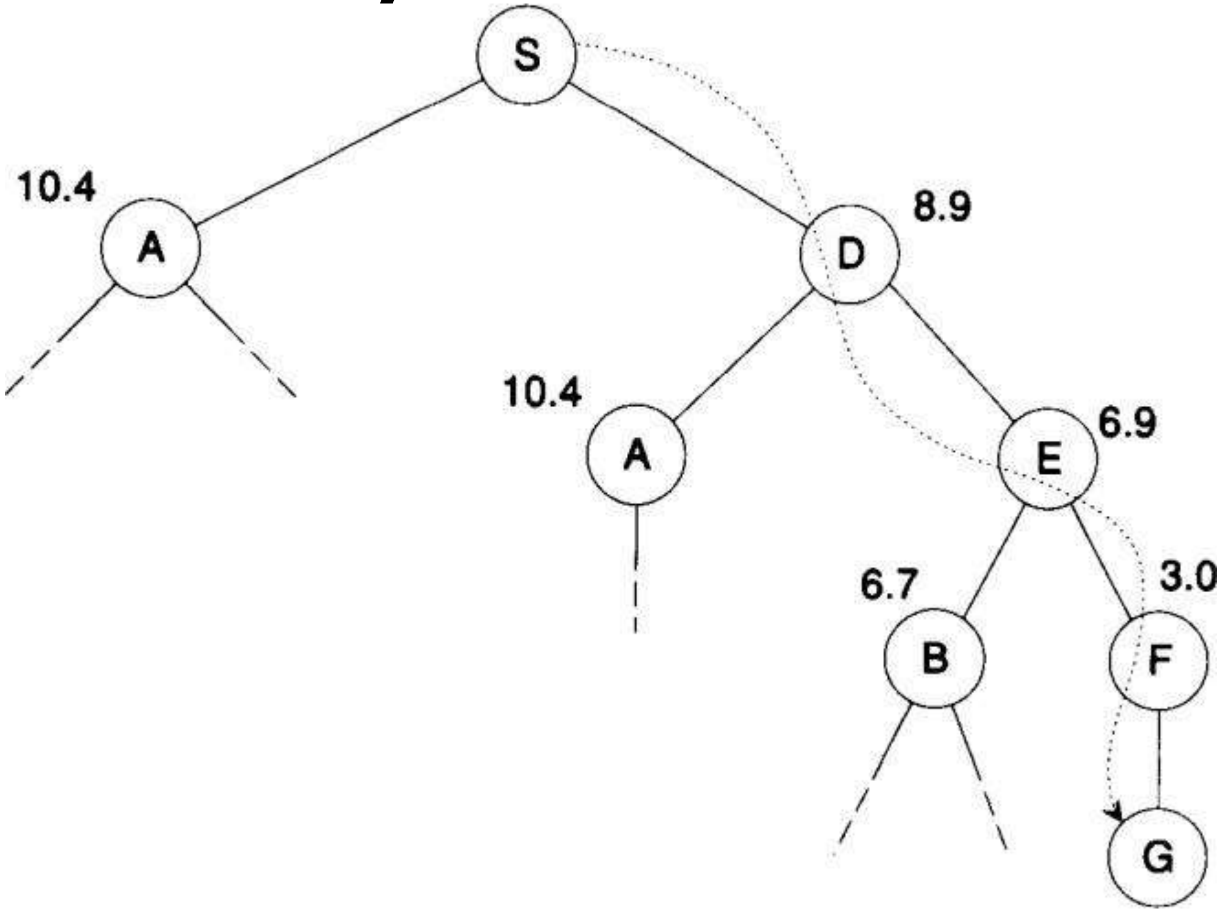
Current	Children
S	---
S	$D_{8.9}$, $A_{10.4}$
D	---
D	$E_{6.9}$, $A_{10.4}$
E	---
E	$F_{3.0}$, $B_{6.7}$
F	

Hill Climbing
Goal - Node



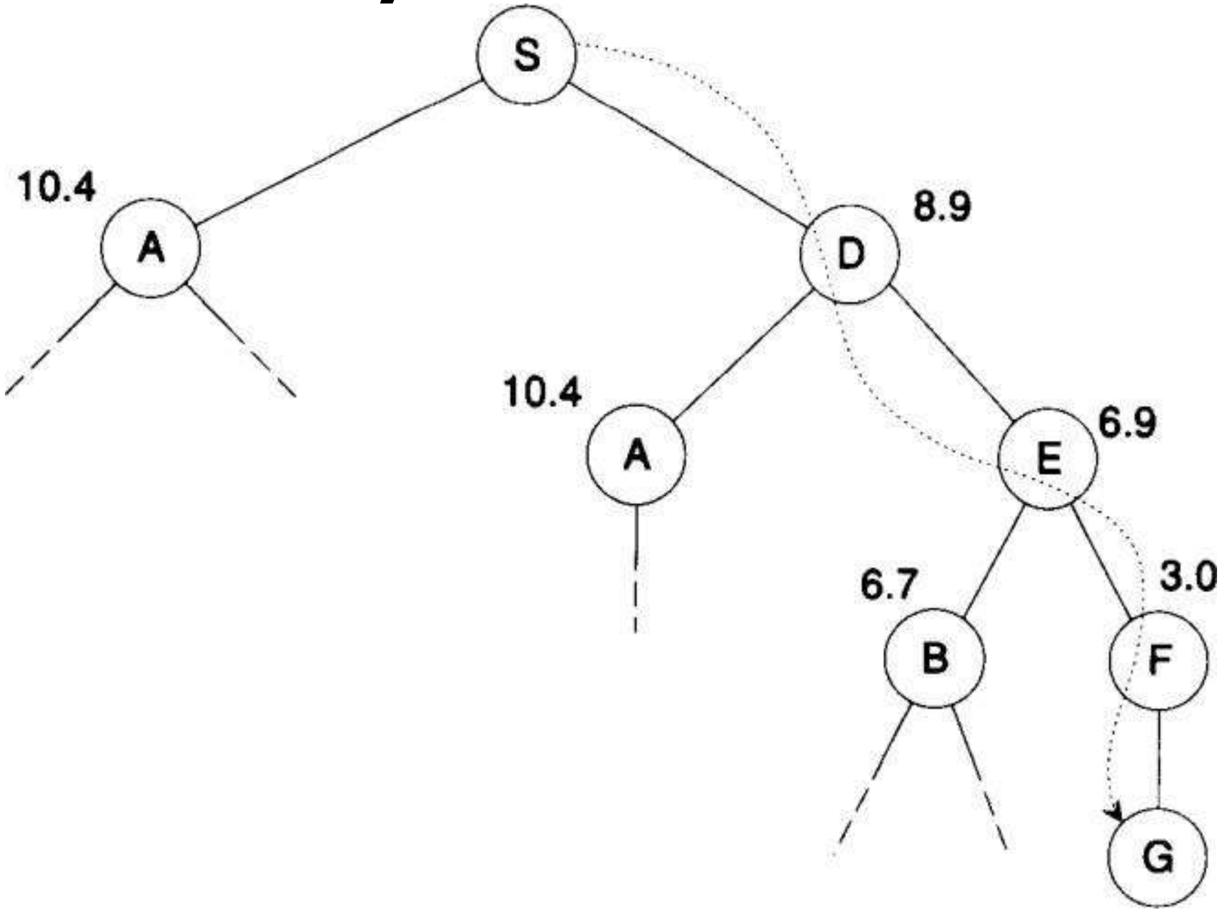
Current	Children
S	---
S	$D_{8.9}, A_{10.4}$
D	---
D	$E_{6.9}, A_{10.4}$
E	---
E	$F_{3.0}, B_{6.7}$
F	---

Hill Climbing Goal - Node



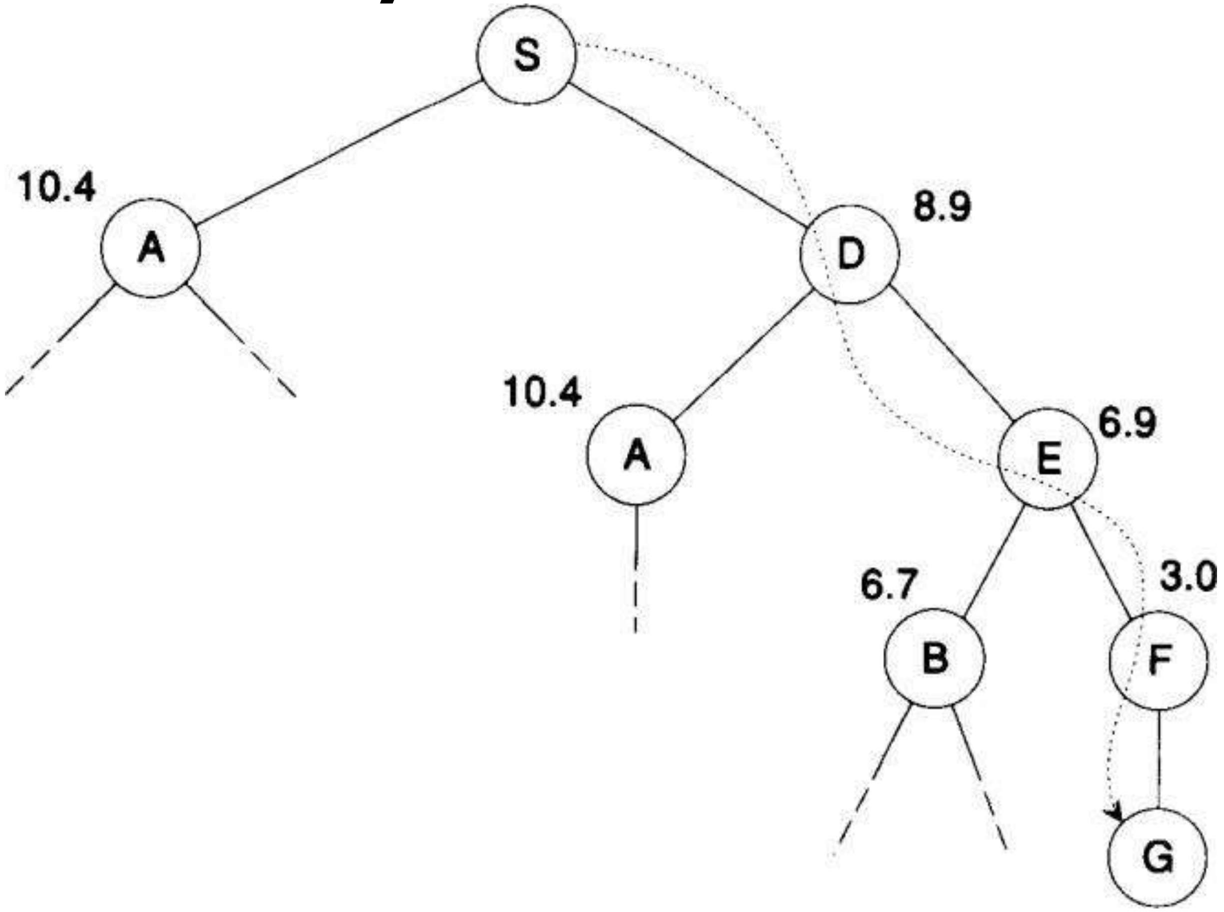
Current	Children
S	---
S	$D_{8.9}, A_{10.4}$
D	---
D	$E_{6.9}, A_{10.4}$
E	---
E	$F_{3.0}, B_{6.7}$
F	---

Hill Climbing Goal - Node



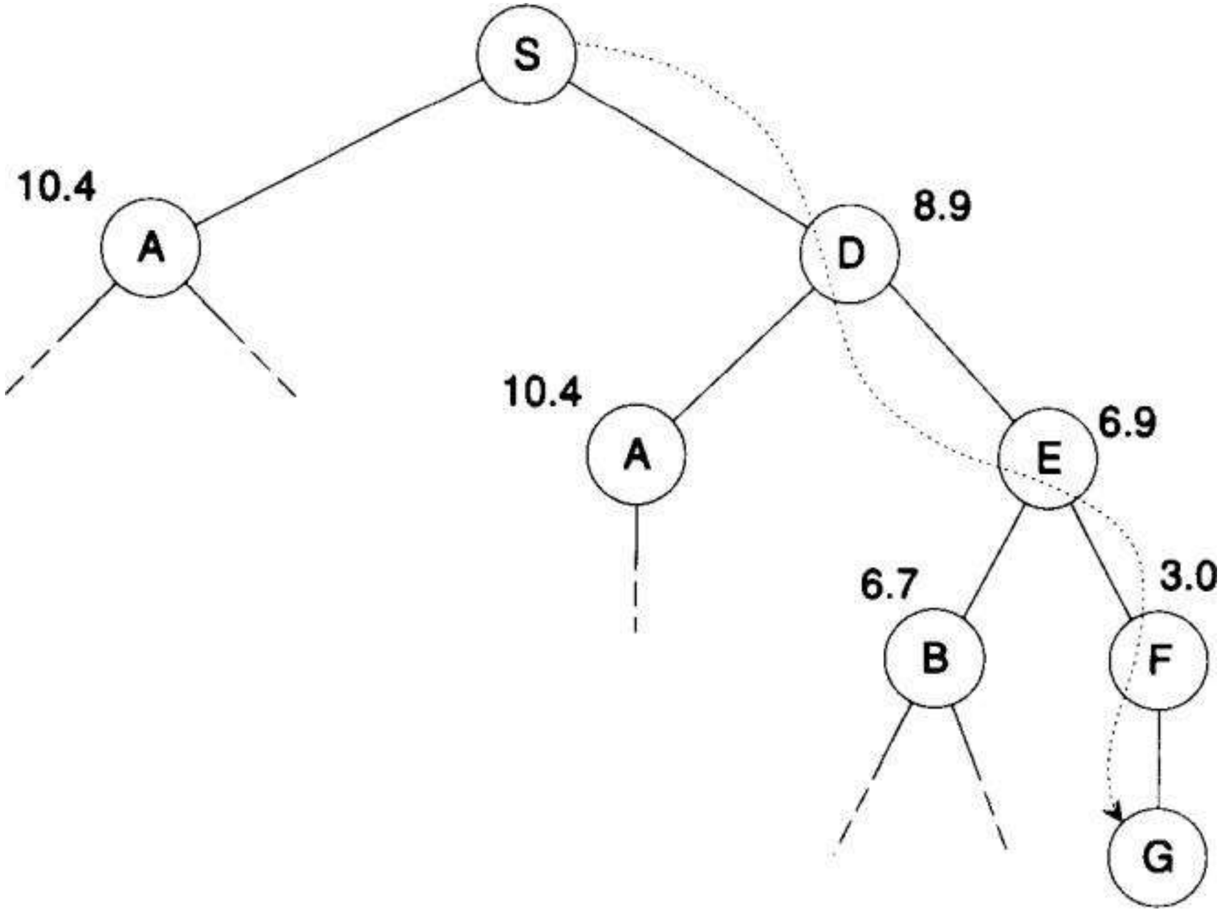
Current	Children
S	---
S	$D_{8.9}, A_{10.4}$
D	---
D	$E_{6.9}, A_{10.4}$
E	---
E	$F_{3.0}, B_{6.7}$
F	---
F	

Hill Climbing Goal - Node G



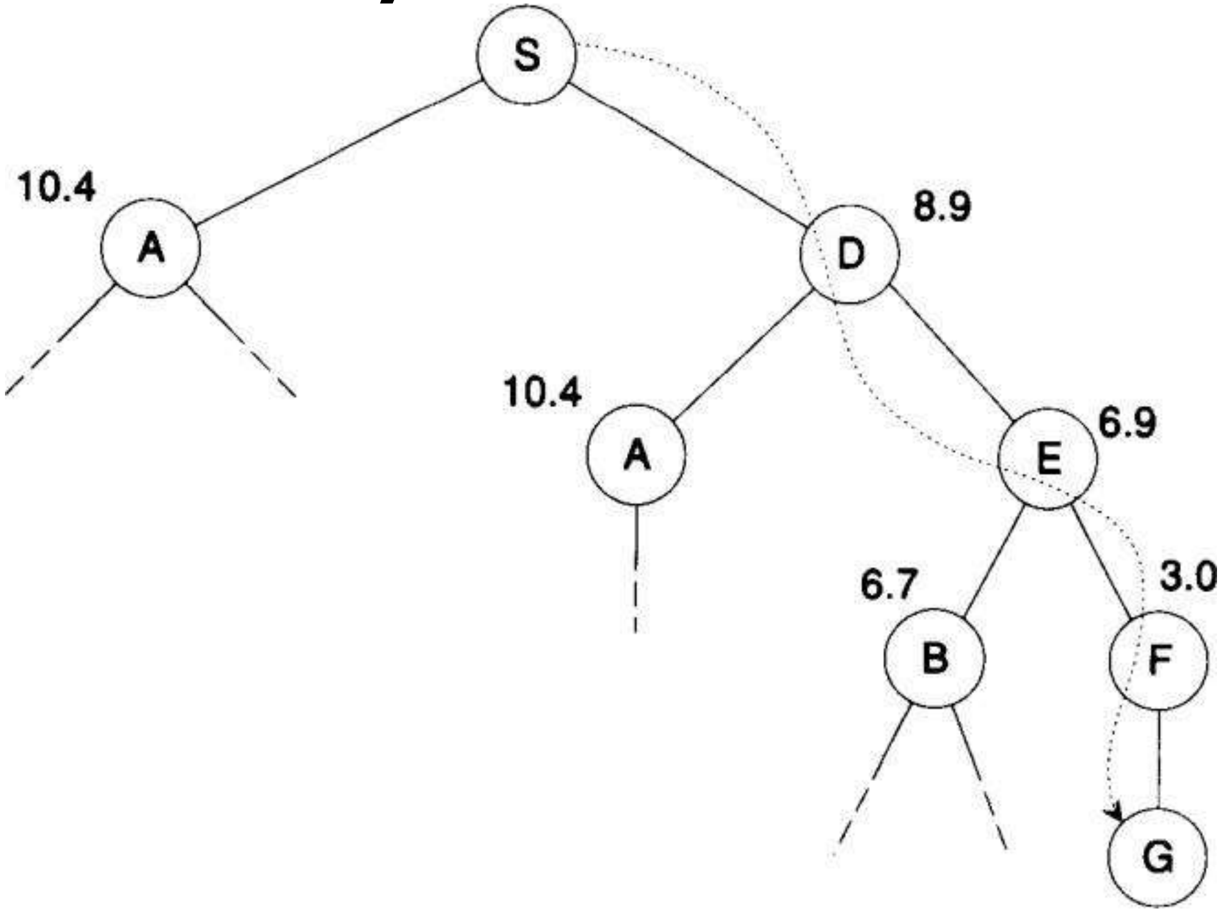
Current	Children
S	---
S	$D_{8.9}$, $A_{10.4}$
D	---
D	$E_{6.9}$, $A_{10.4}$
E	---
E	$F_{3.0}$, $B_{6.7}$
F	---
F	G

Hill Climbing
Goal - Node G



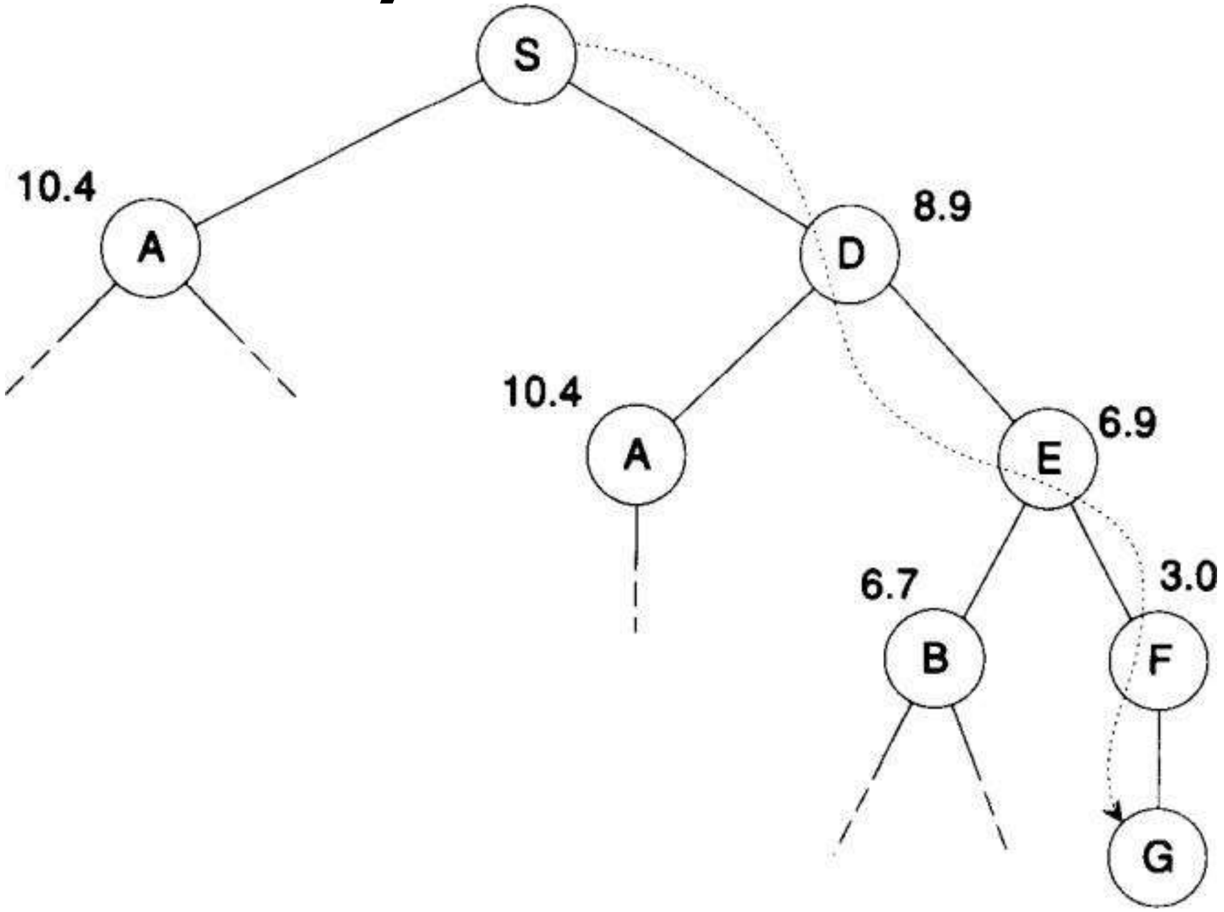
Current	Children
S	---
S	$D_{8.9}, A_{10.4}$
D	---
D	$E_{6.9}, A_{10.4}$
E	---
E	$F_{3.0}, B_{6.7}$
F	---
F	G

Hill Climbing Goal - Node



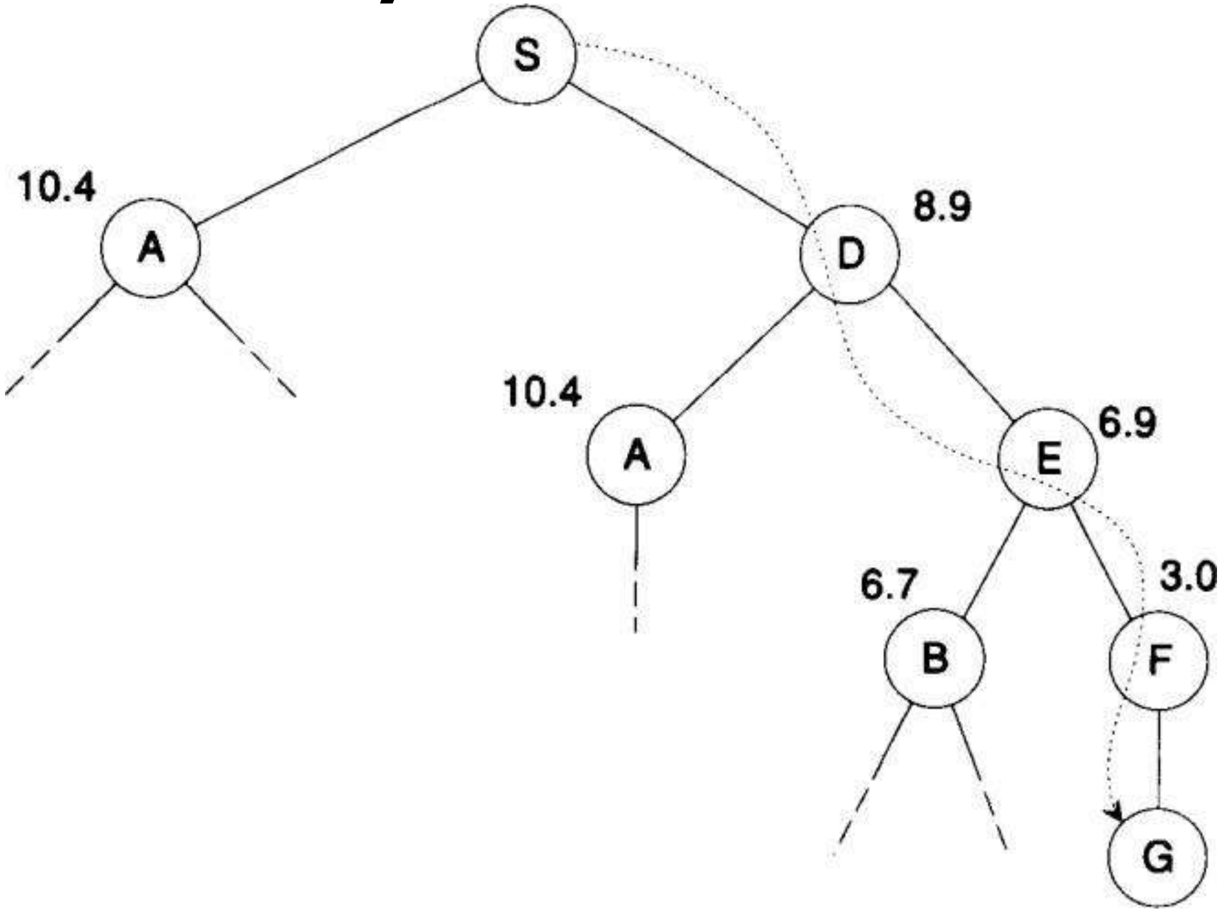
Current	Children
S	---
S	$D_{8.9}, A_{10.4}$
D	---
D	$E_{6.9}, A_{10.4}$
E	---
E	$F_{3.0}, B_{6.7}$
F	---
F	G
G	

Hill Climbing
Goal – Node



Current	Children
S	---
S	$D_{8.9}, A_{10.4}$
D	---
D	$E_{6.9}, A_{10.4}$
E	---
E	$F_{3.0}, B_{6.7}$
F	---
F	G
G	---

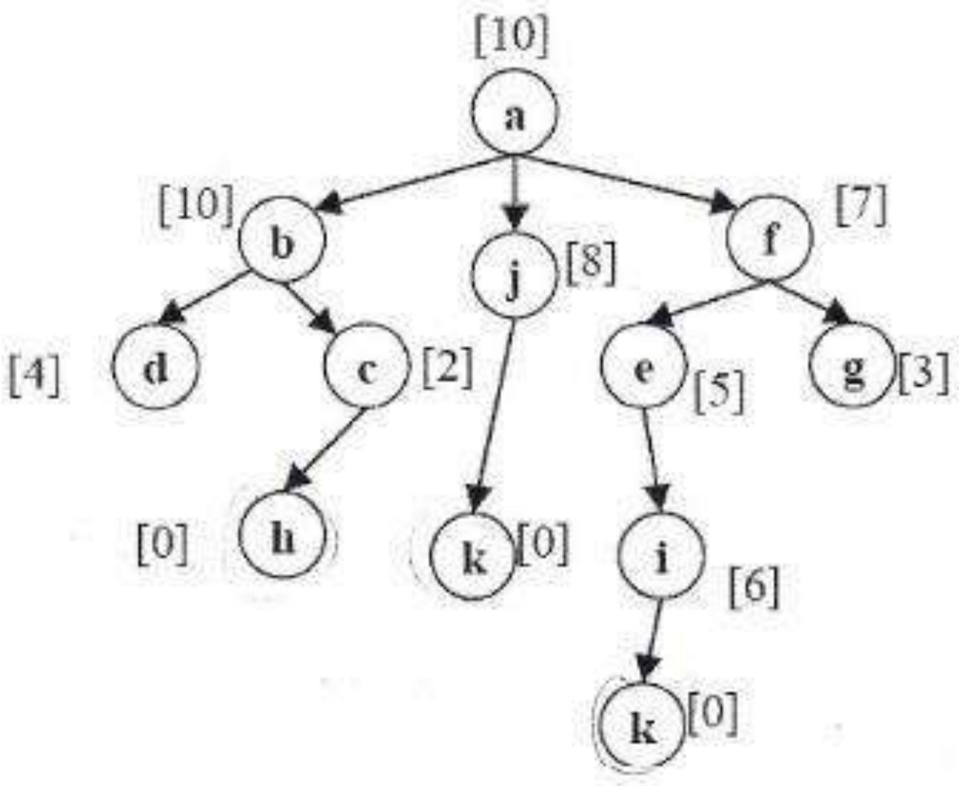
Hill Climbing Goal - Node G



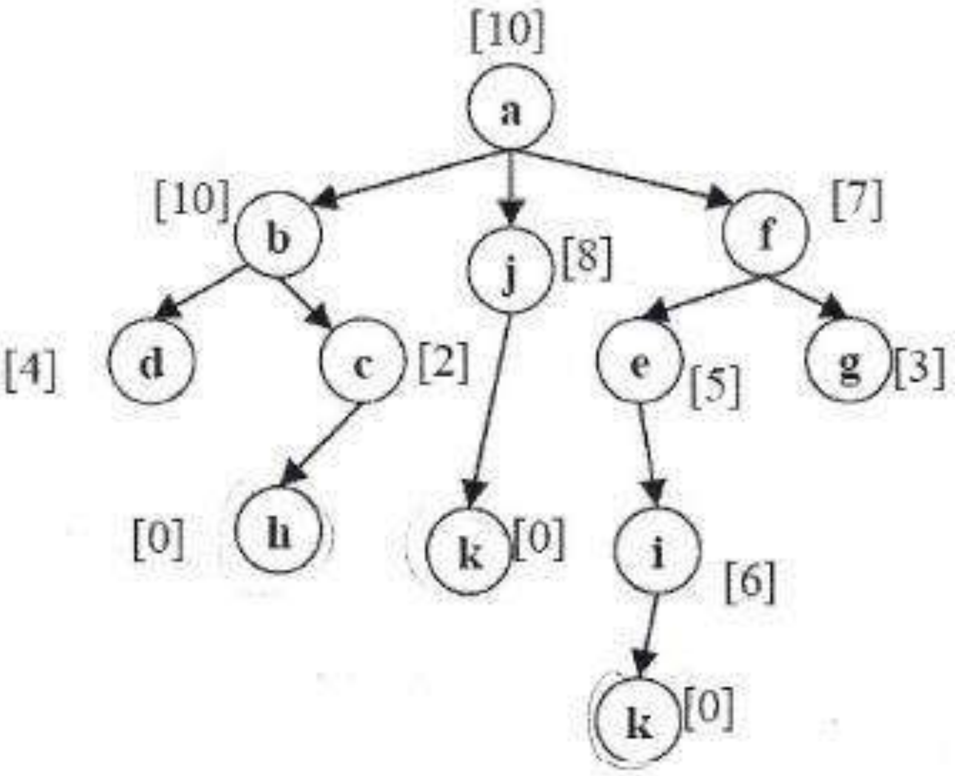
Current	Children
S	---
S	$D_{8.9}, A_{10.4}$
D	---
D	$E_{6.9}, A_{10.4}$
E	---
E	$F_{3.0}, B_{6.7}$
F	---
F	G
G	---

Hill Climbing
Goal – Node K
Local Maxima

Current	Children
a	---



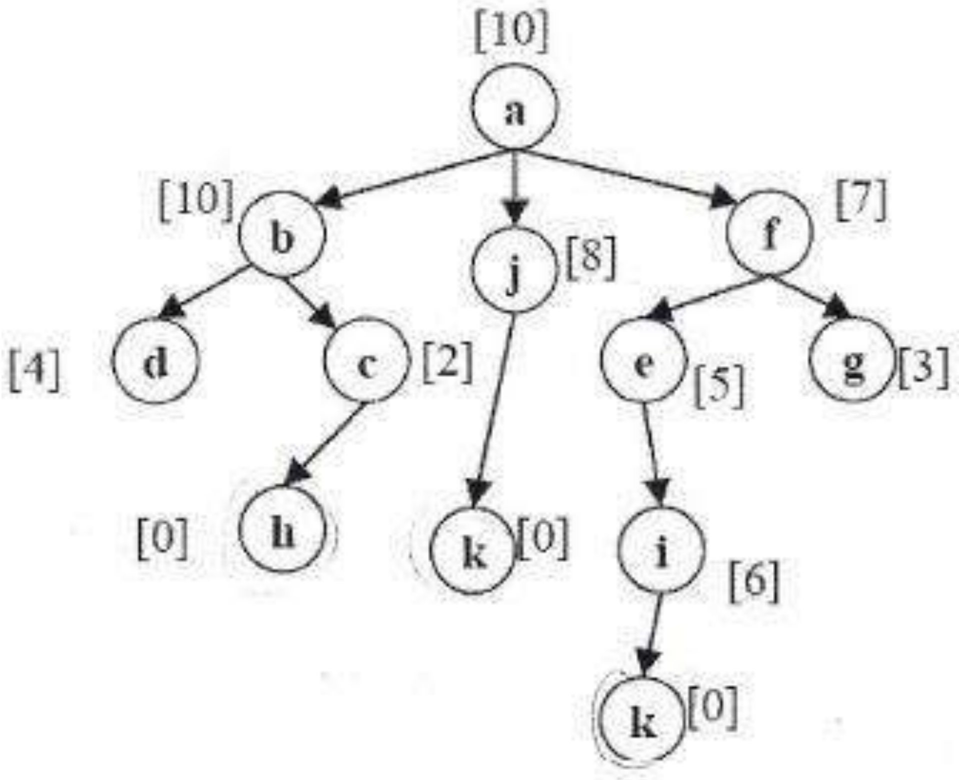
Hill Climbing Goal – Node K Local



Current	Children
a	---

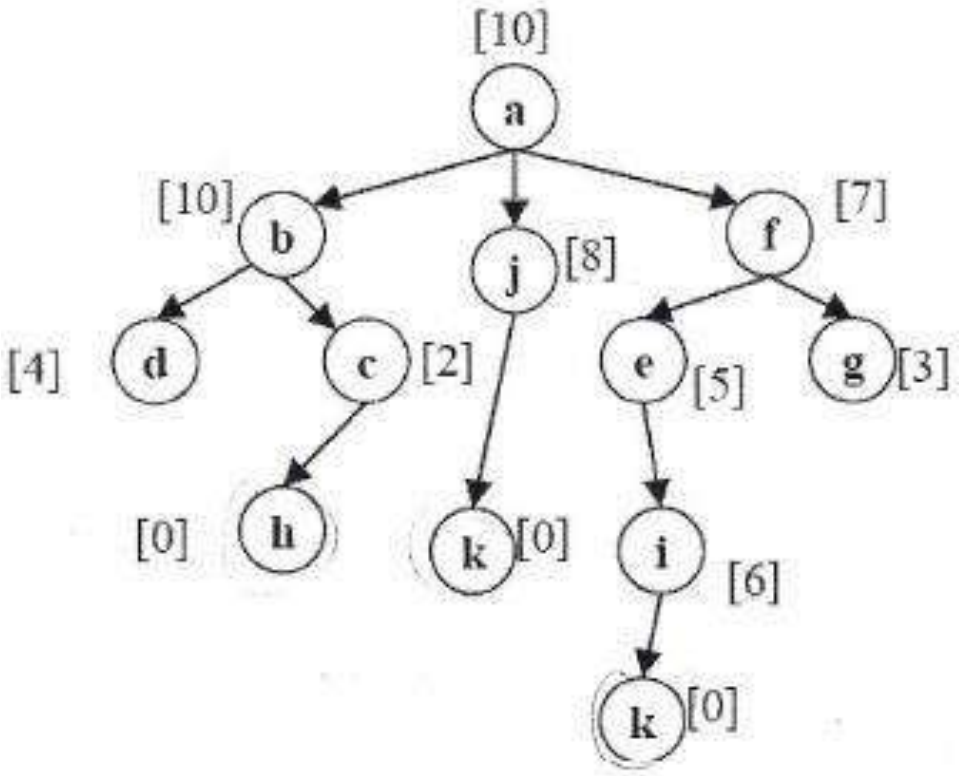
Hill Climbing Goal – Node K Local

Current	Children
a	---
a	

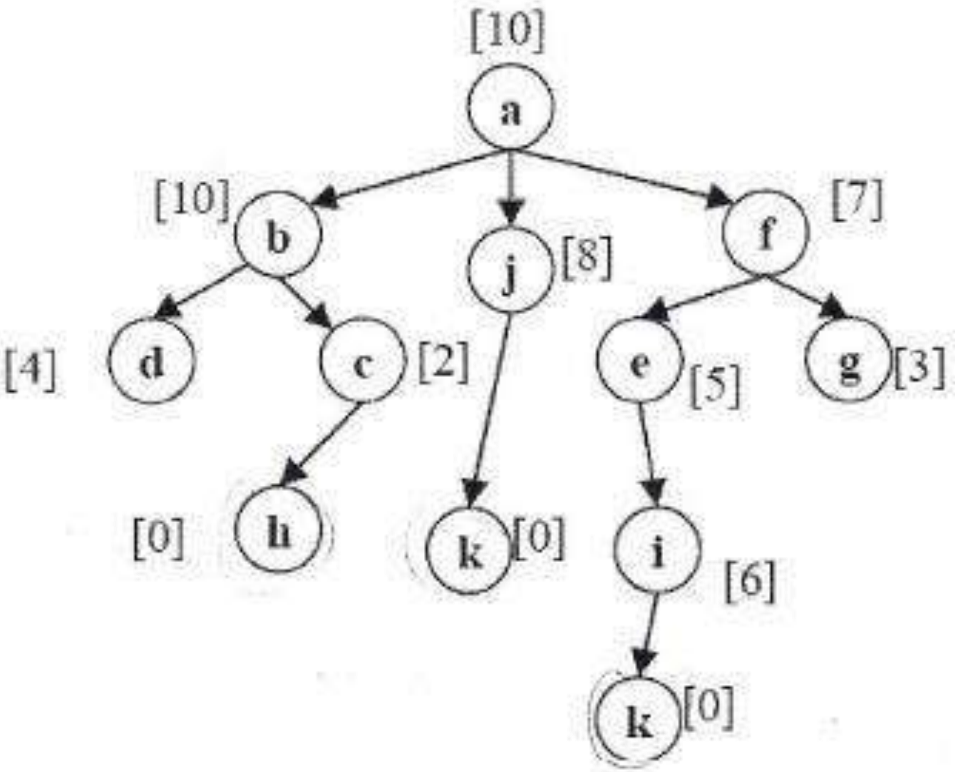


Hill Climbing Goal – Node K Local

Current	Children
a	---
a	f_7, j_8, b_{10}

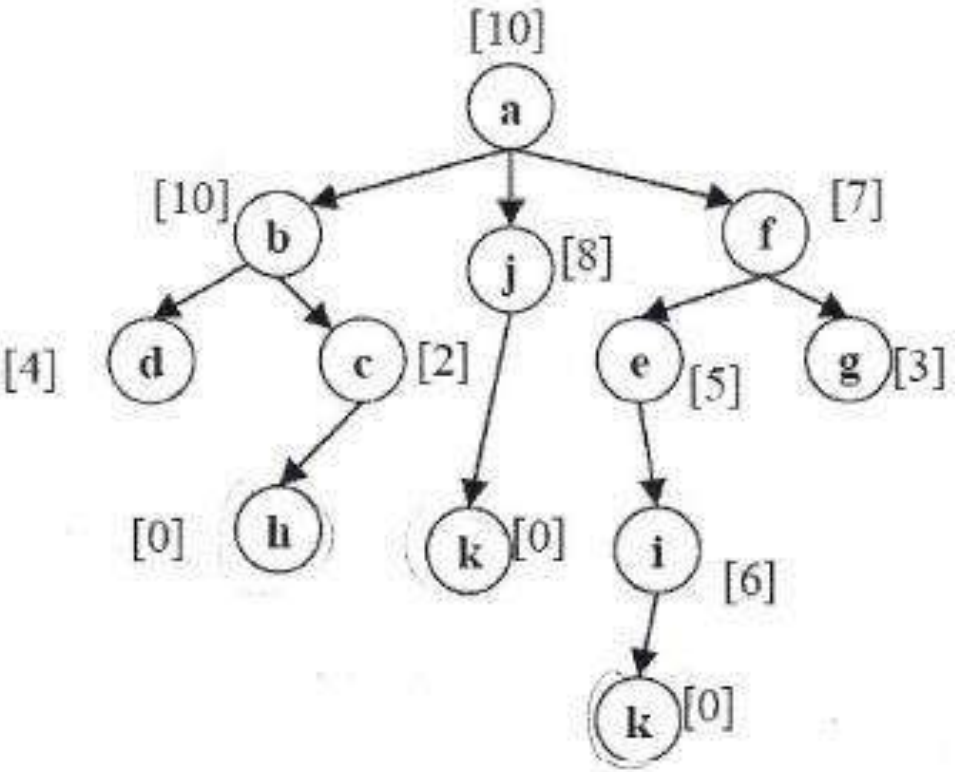


Hill Climbing Goal – Node K Local



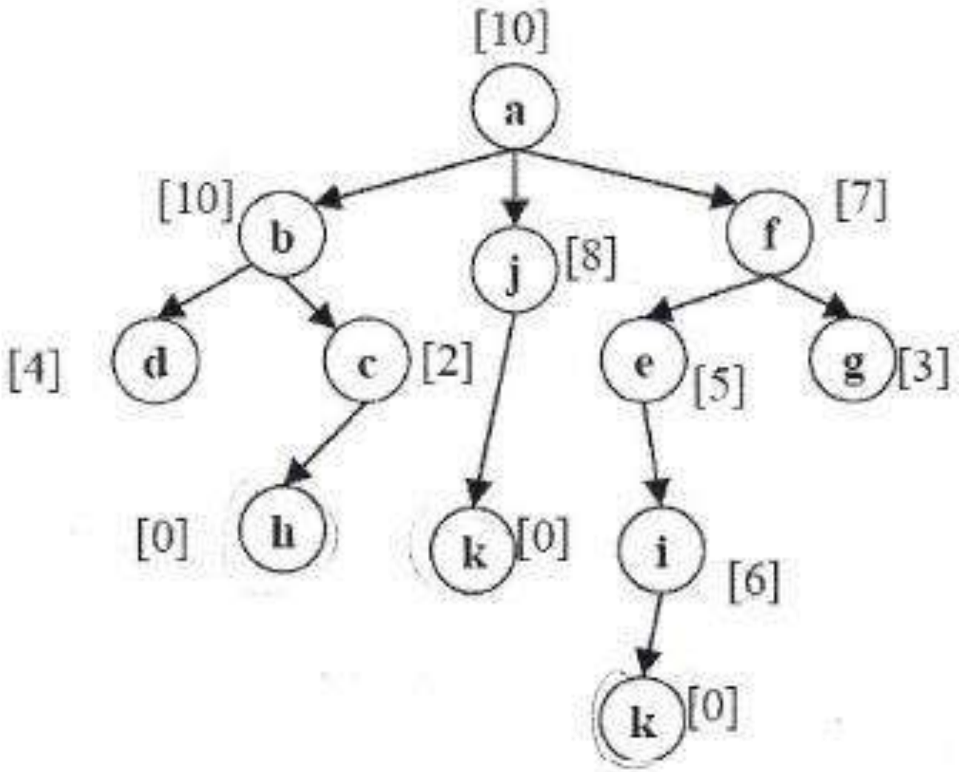
Current	Children
a	---
a	f_7, j_8, b_{10}
f	

Hill Climbing Goal – Node K Local



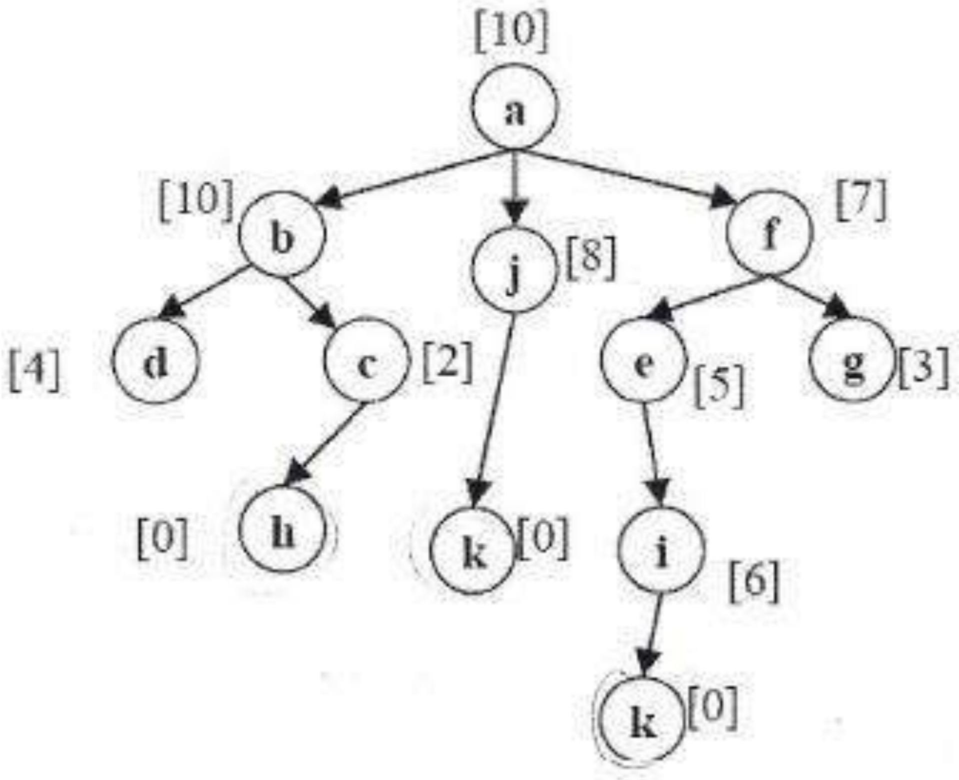
Current	Children
a	---
a	f_7, j_8, b_{10}
f	---

Hill Climbing Goal – Node K Local Maxima



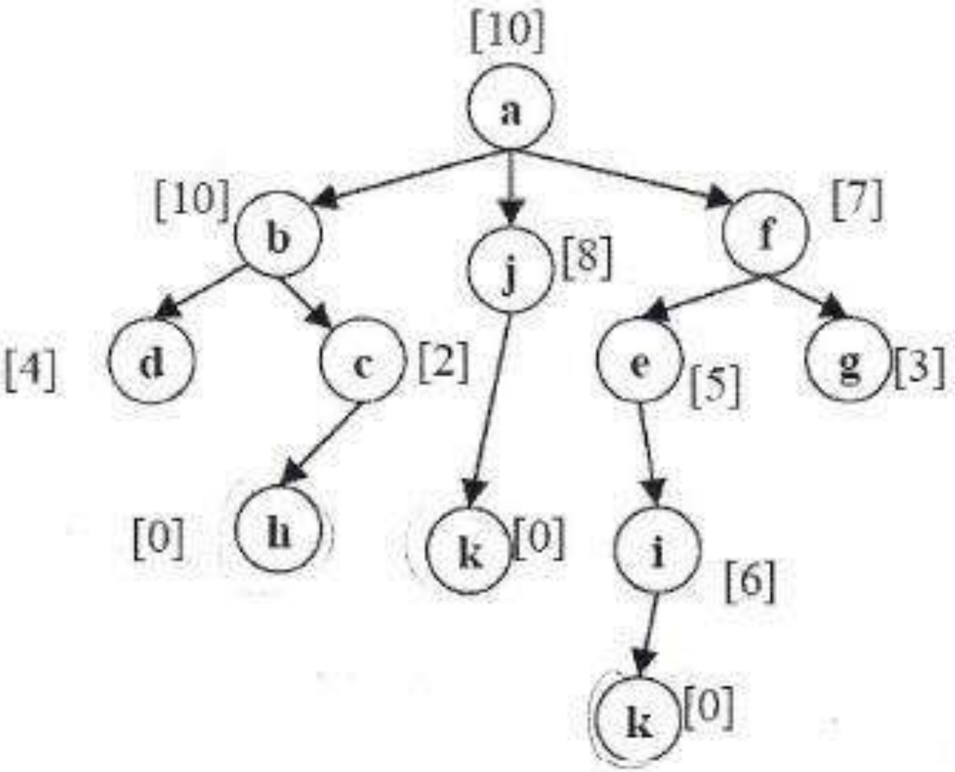
Current	Children
a	---
a	$f_7, j_8 b_{10}$
f	---

Hill Climbing Goal – Node K Local



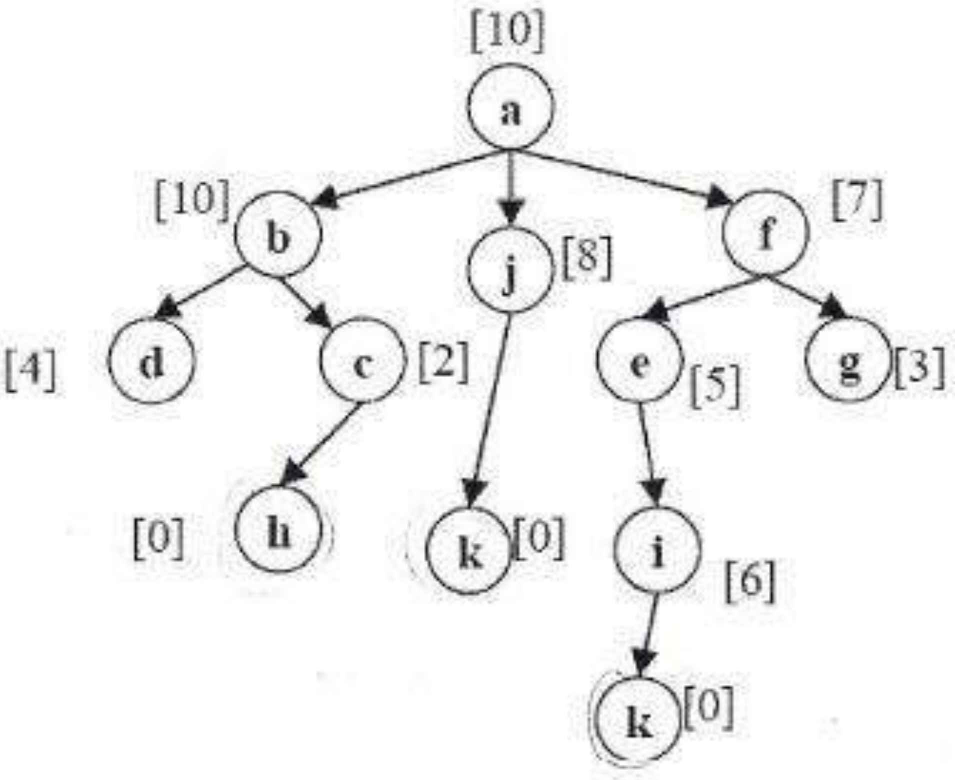
Current	Children
a	---
a	f_7, j_8, b_{10}
f	---
f	

Hill Climbing Goal – Node K Local



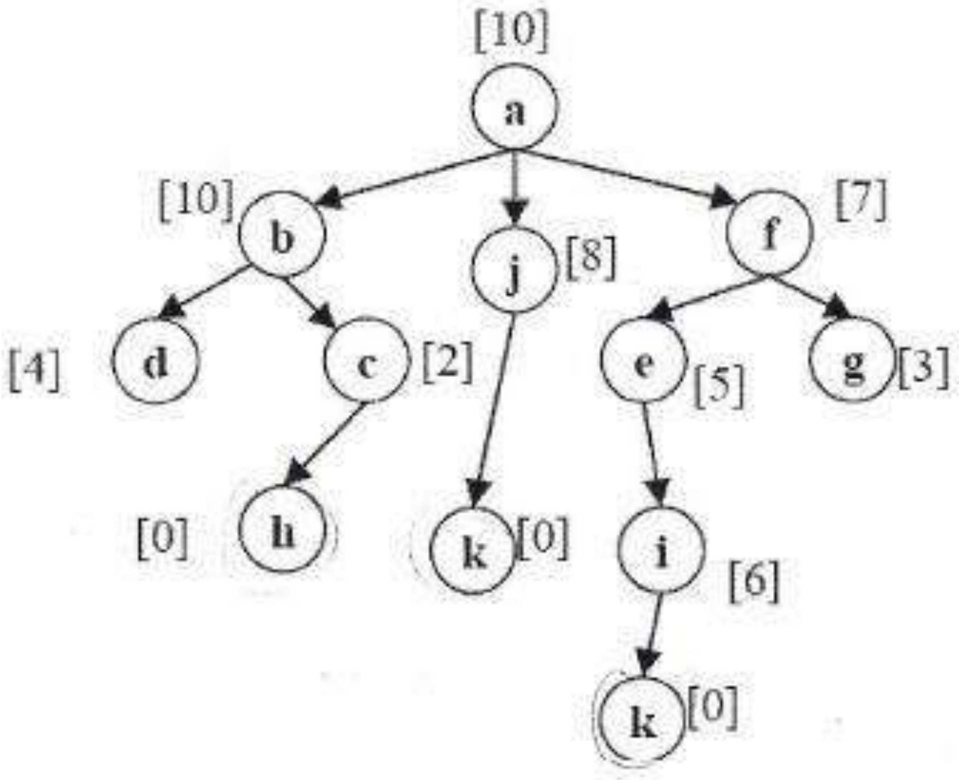
Current	Children
a	---
a	f_7, j_8, b_{10}
f	---
f	g_3, e_5

Hill Climbing Goal – Node K Local



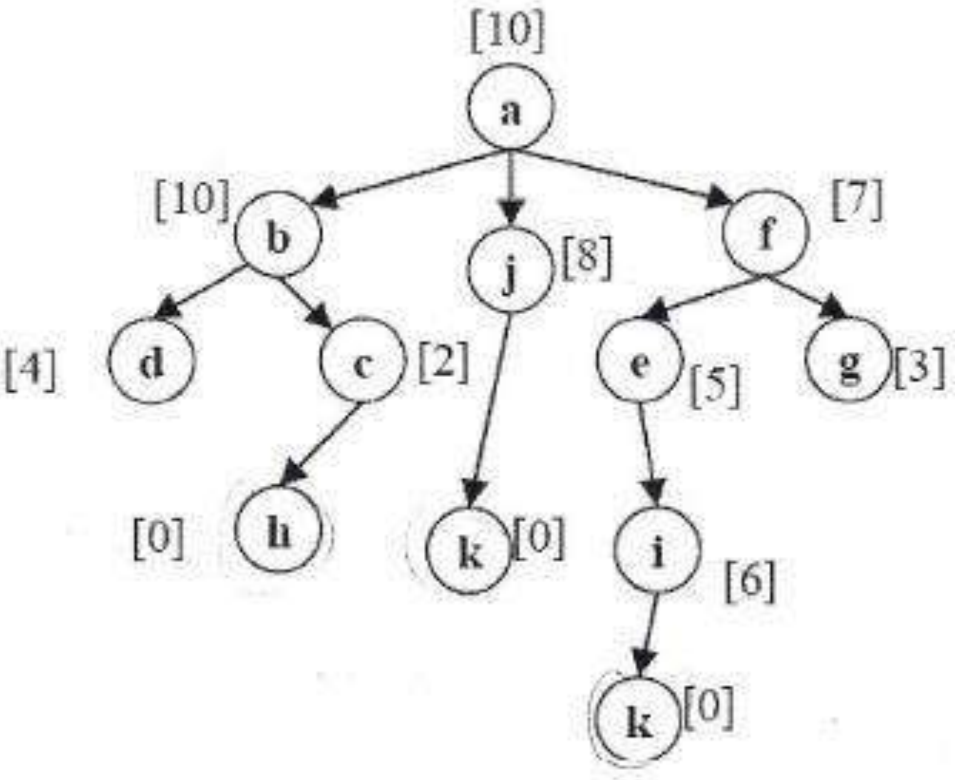
Current	Children
a	---
a	f_7, j_8, b_{10}
f	---
f	g_3, e_5
g	

Hill Climbing Goal – Node K Local



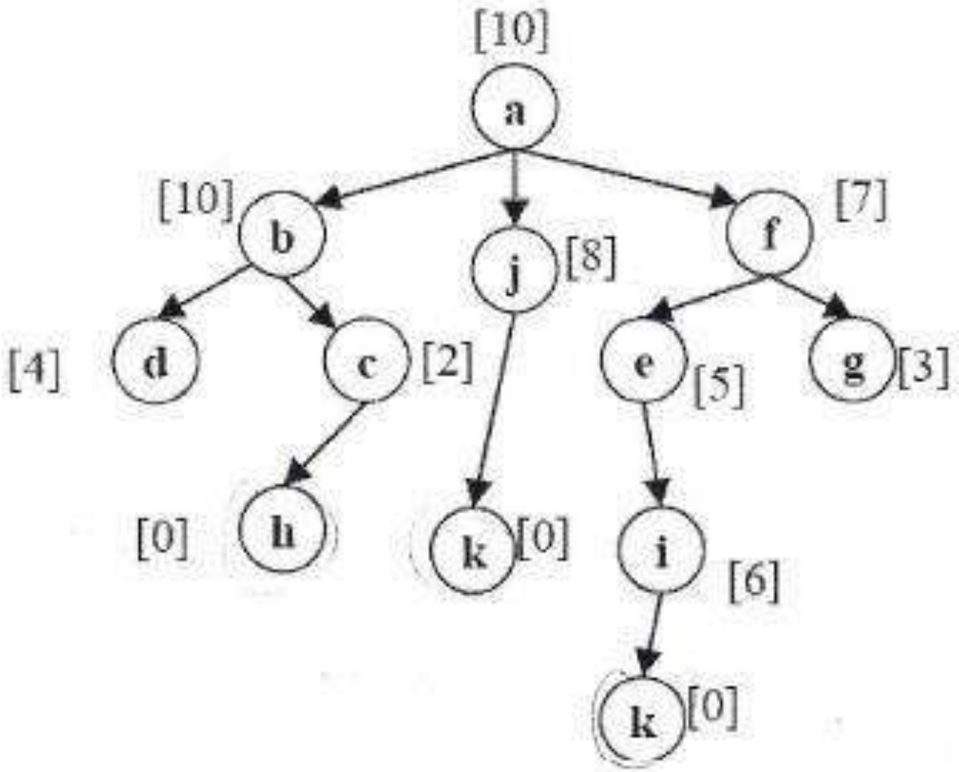
Current	Children
a	---
a	f_7, j_8, b_{10}
f	---
f	g_3, e_5
g	---

Hill Climbing Goal – Node K Local



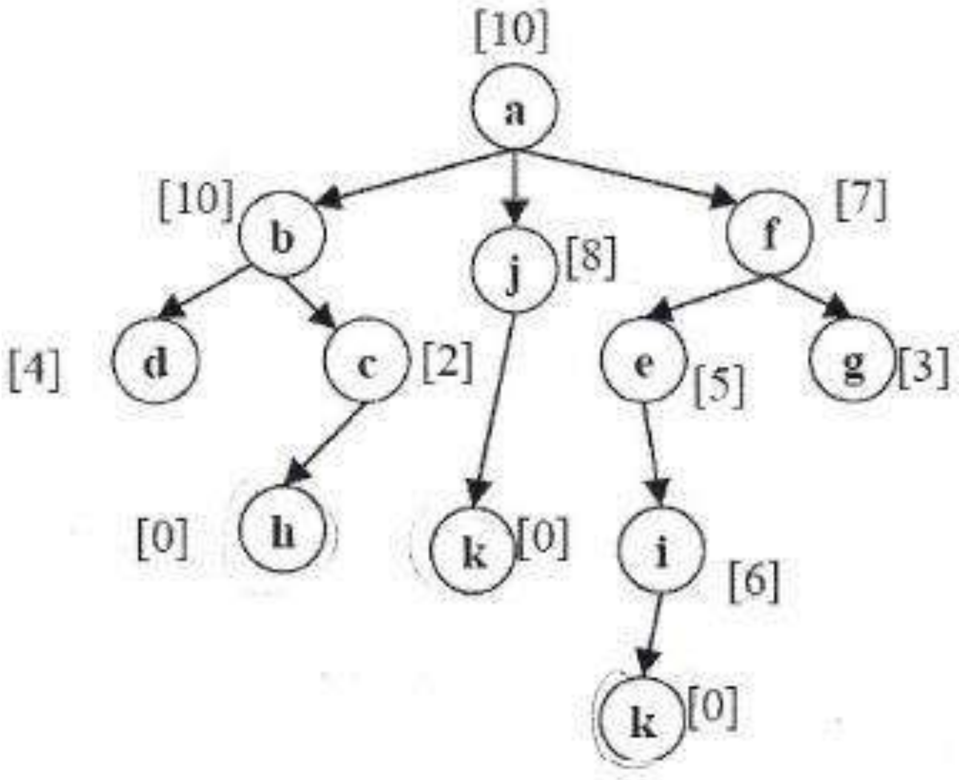
Current	Children
a	---
a	f_7, j_8, b_{10}
f	---
f	g_3, e_5
g	---

Hill Climbing Goal – Node K Local



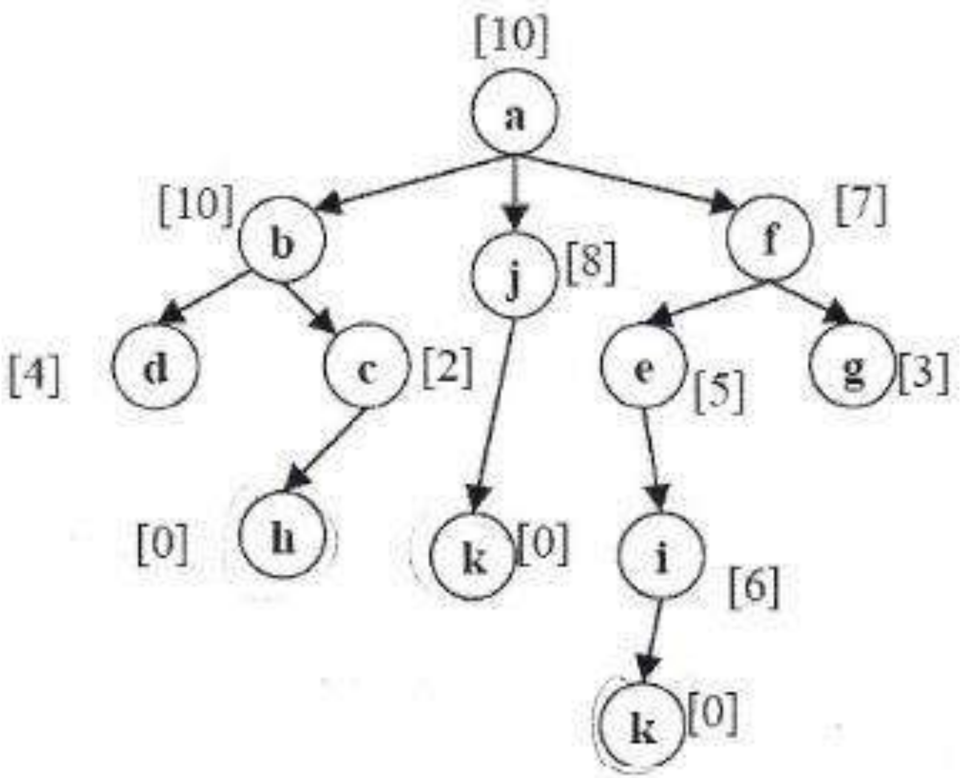
Current	Children
a	---
a	f_7, j_8, b_{10}
f	---
f	g_3, e_5
g	---
g	

Hill Climbing Goal – Node K Local



Current	Children
a	---
a	f_7, j_8, b_{10}
f	---
f	g_3, e_5
g	---
g	---

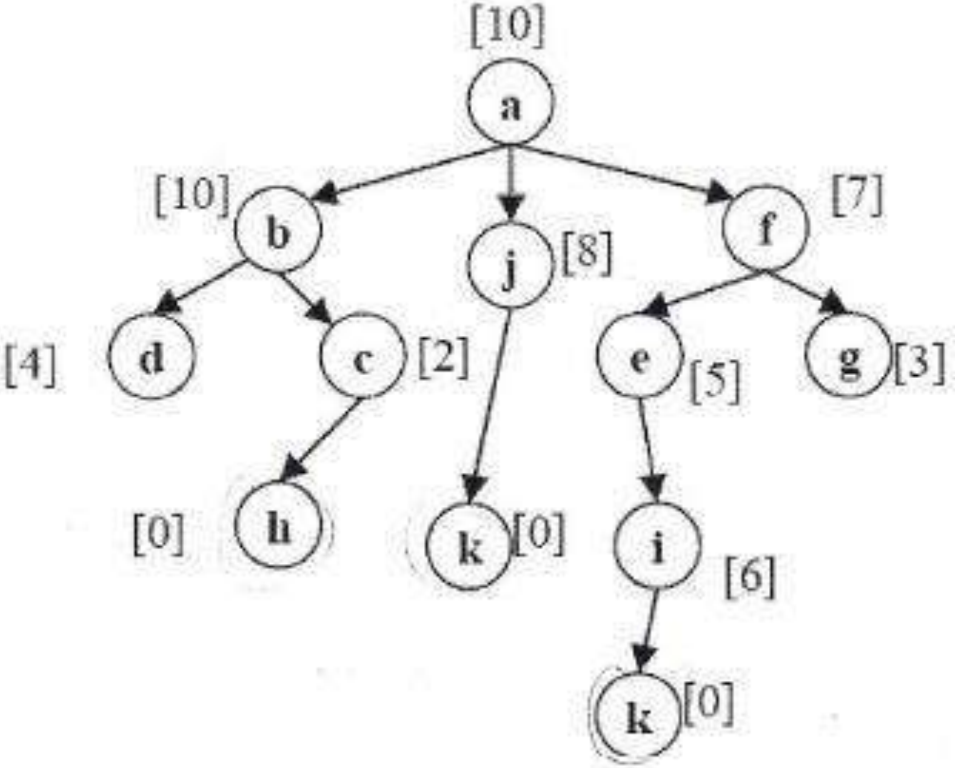
Hill Climbing
Goal – Node K
Local Maxima



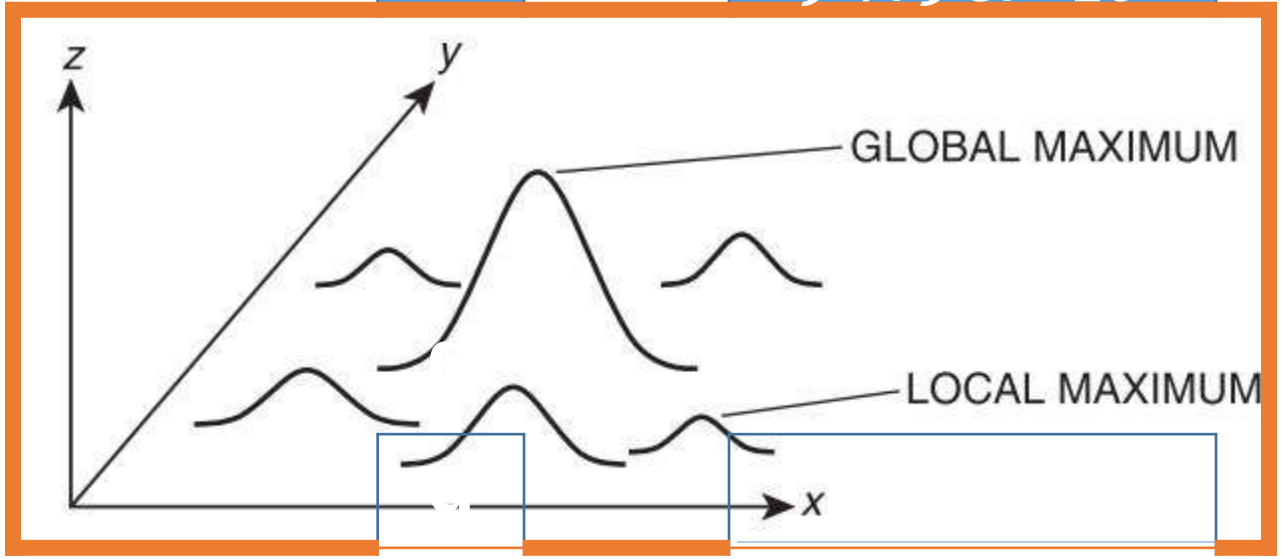
Current	Children
a	---
a	f_7, j_8, b_{10}
f	---
f	g_3, e_5
g	---
g	---

Search Finished
No GOAL

Hill Climbing Goal – Node K Local



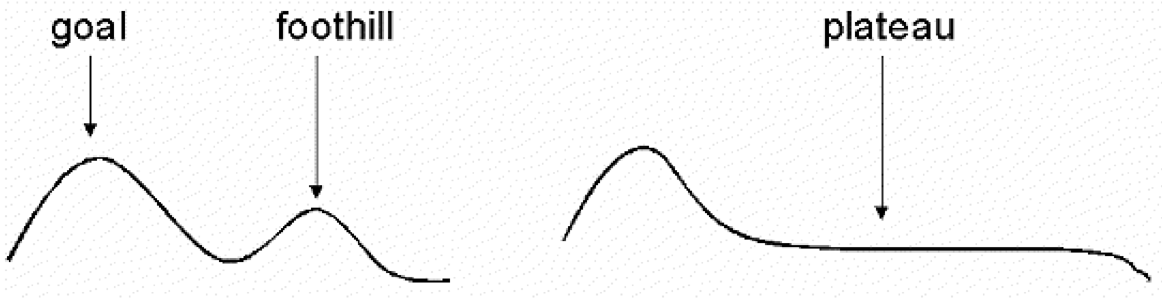
Current	Children
a	---
a	f_7, j_8, b_{10}



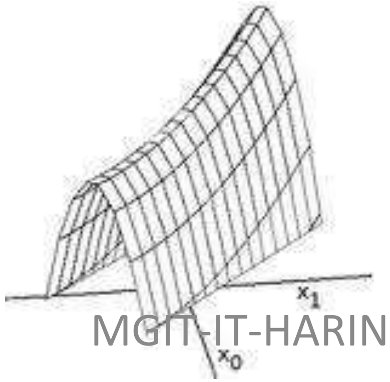
Search Finished
No GOAL

Drawbacks of Hill climbing

- **Local Maxima:** **peaks that aren't the highest point in the space**
- **Plateaus:** **the space has a broad flat region that gives the search algorithm no direction (random walk)**



- **Ridges:** **dropoffs to the sides; steps to the North, East, South and West may go down, but a step to the NW may go up.**



Variations of Hill Climbing

- **Stochastic Hill Climbing-** Chooses at random from among uphill moves
- **First Choice Hill Climbing-** Generating successors randomly until one is generated that is better than current state.
- **Random Restart Hill Climbing-** Conducts Series of Hill Climbing Searches from randomly generated initial states till goal is found

Success of Hill Climbing depends very much on the shape of the State Space LandScape

Simulated Annealing

- Variant of hill climbing (so up is good)
- Tries to explore enough of the search space early on, so that the final solution is less sensitive to the start state
- **SA hill-climbing can avoid becoming trapped at local maxima.**
- May make some downhill moves before finding a good way to move uphill.

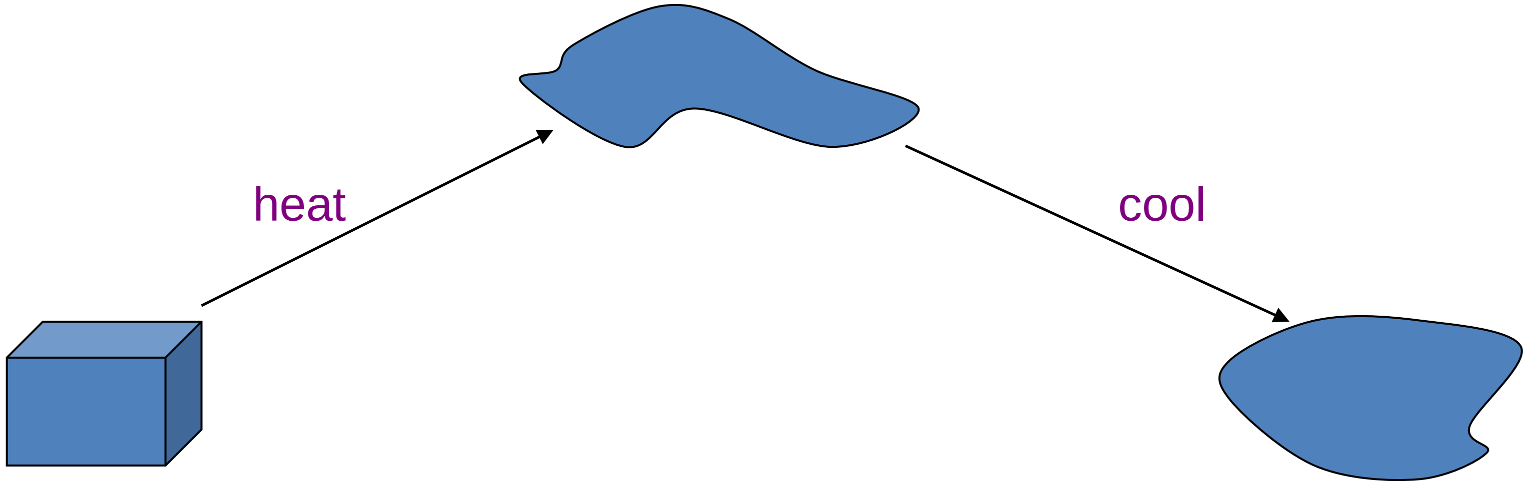
Simulated Annealing (SA) Search

- **Hill climbing:** move to a better state
  - Efficient, but incomplete (can stuck in local maxima)
- **Random walk:** move to a random successor
  - Asymptotically complete, but extremely inefficient

- **Idea:** Escape local maxima by allowing some "bad“ moves but gradually decrease their frequency.
  - More exploration at start and gradually hill-climbing become more frequently selected strategy.

Simulated Annealing

➢Comes from the physical process of annealing in which substances are raised to high energy levels (melted) and then cooled to solid state.



➢The probability of moving to a higher energy state, instead of lower is

$$p = e^{\wedge}(-\Delta E/kT)$$

where ΔE is the positive change in energy level, T is the temperature, and k is Bolzmann's constant.

Simulated Annealing

- At the beginning, the temperature is high.
- As the temperature becomes lower
  - kT becomes lower
  - $\Delta E/kT$ gets bigger
  - $(-\Delta E/kT)$ gets smaller
  - $e^{(-\Delta E/kT)}$ gets smaller
- As the process continues, the probability of a downhill move gets smaller and smaller.

For Simulated Annealing

- ΔE represents the change in the value of the objective function.
- Since the physical relationships no longer apply, drop k. So p = $e^{(-\Delta E/T)}$
- We need an annealing schedule, which is a sequence of values of T: $T_0, T_1, T_2, \ldots$

Simulated Annealing Algorithm

function SIMULATED-ANNEALING(*problem, schedule*) returns a solution state
input: *problem*, a problem
**   *schedule*, a mapping from time to "temperature"**

current $\leftarrow$ **MAKE-NODE(*problem.*INITIAL-STATE)**
for t $\leftarrow$ **1 to ∞ do**
T $\leftarrow$ ***schedule*(*t*)**
if *T = 0* then return *current*
next $\leftarrow$ **a randomly selected successor of *current***
$\Delta E \leftarrow$ ***next.*VALUE – *current.*VALUE**
if $\Delta E > 0$ **then *current*** $\leftarrow$ ***next*** **/* better than current */**
else *current* $\leftarrow$ ***next* only with probability** $e^{\Delta E / T}$

Simulated Annealing

- Inner most loop similar to Hill- Climbing
- Instead of Picking Best Move, **it picks Random Move**
- If **Move improves the situation, it is always accepted**
- The **probability decreases exponentially with the badness of the move**
- The **probability also decreases as the temperature T goes down.**
- **Bad Moves** are more likely to be allowed **at the start when T is High** and they become **unlikely as T decreases.**
- If schedule **lowers T** slowly enough, the algorithm will **find a Global Optimum with Probability approaching 1.**

Simulated Annealing Applications

Basic Problems

- Traveling salesman
- Graph partitioning
- Matching problems
- Graph coloring
- Scheduling

Engineering

- VLSI design
  - Placement
  - Routing
  - Array logic minimization
  - Layout
- Facilities layout
- Image processing
- Code design in information theory

Local search in continuous spaces

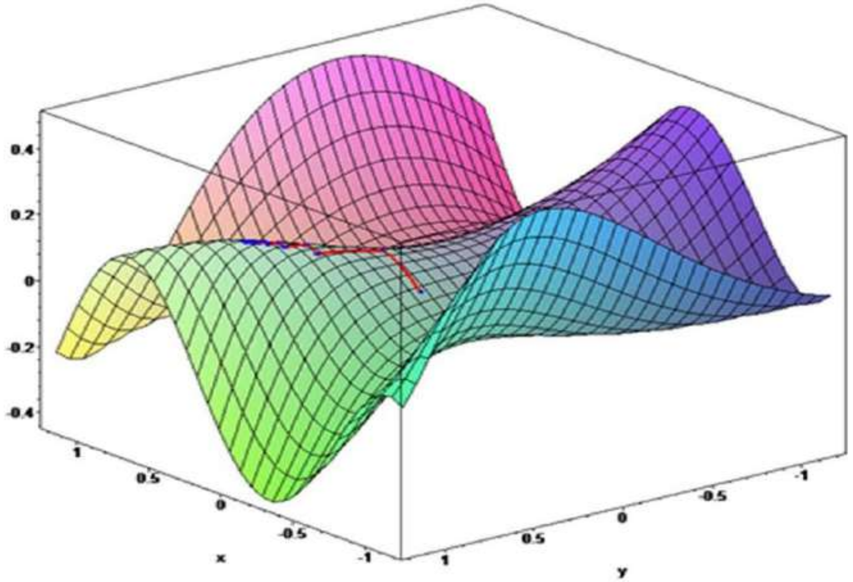
- Infinite number of successor states
  - E.g., select locations for 3 airports such that sum of squared distances from each city to its nearest airport is minimized
    - (x_1, y_1) , (x_2, y_2) , (x_3, y_3)
    - $F(x_1, y_1, x_2, y_2, x_3, y_3) = \sum_{i=1}^{3} \sum_{c \in C_i} (x_i - x_c)^2 + (y_i - y_c)^2$
- Approach 1: Discretization
  - Just change variable by $\pm\delta$
    - E.g., 6×2 actions for airport example
- Approach 2: Continuous optimization

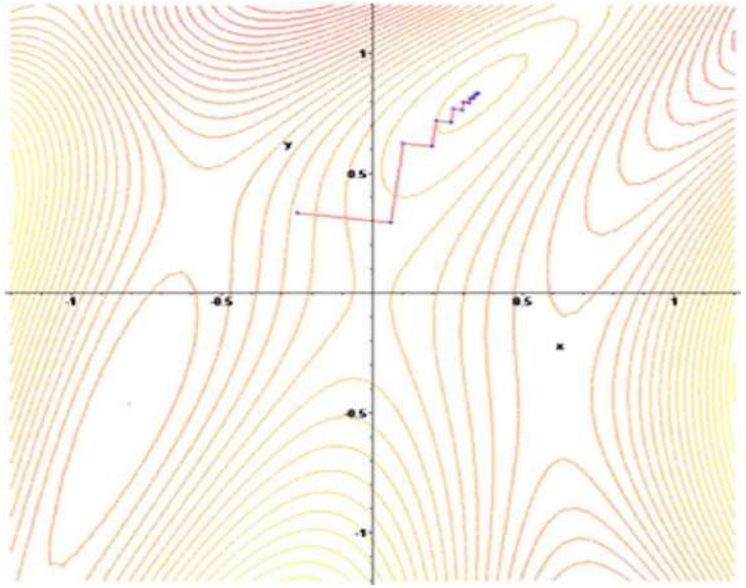
$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, + \frac{\partial f}{\partial x_d} \right)$$

  - $\nabla f = 0$ (only for simple cases)
  - Gradient ascent $\mathbf{x}^{t+1} \leftarrow \mathbf{x}^t + \alpha \nabla f(\mathbf{x}^t)$

Gradient ascent

$$\mathbf{x}^{t+1} \leftarrow \mathbf{x}^{t} + \alpha \nabla f(\mathbf{x}^{t})$$





- Local search problems also in continuous spaces
  - Random restarts and simulated annealing can be useful
  - Higher dimensions raises the rate of getting lost

Adjusting Gradient descent

- Adjusting α in gradient descent
  - Line search
  - Newton-Raphson

$$\mathbf{x}^{t+1} \leftarrow \mathbf{x}^{t} - \mathbf{H}_f^{-1}(\mathbf{x}^{t})\nabla f(\mathbf{x}^{t})$$

$$H_{ij} = \partial^2 f / \partial x_i \partial x_j$$

Searching with Nondeterministic Actions

Vacuum World (actions = {left, right, suck})

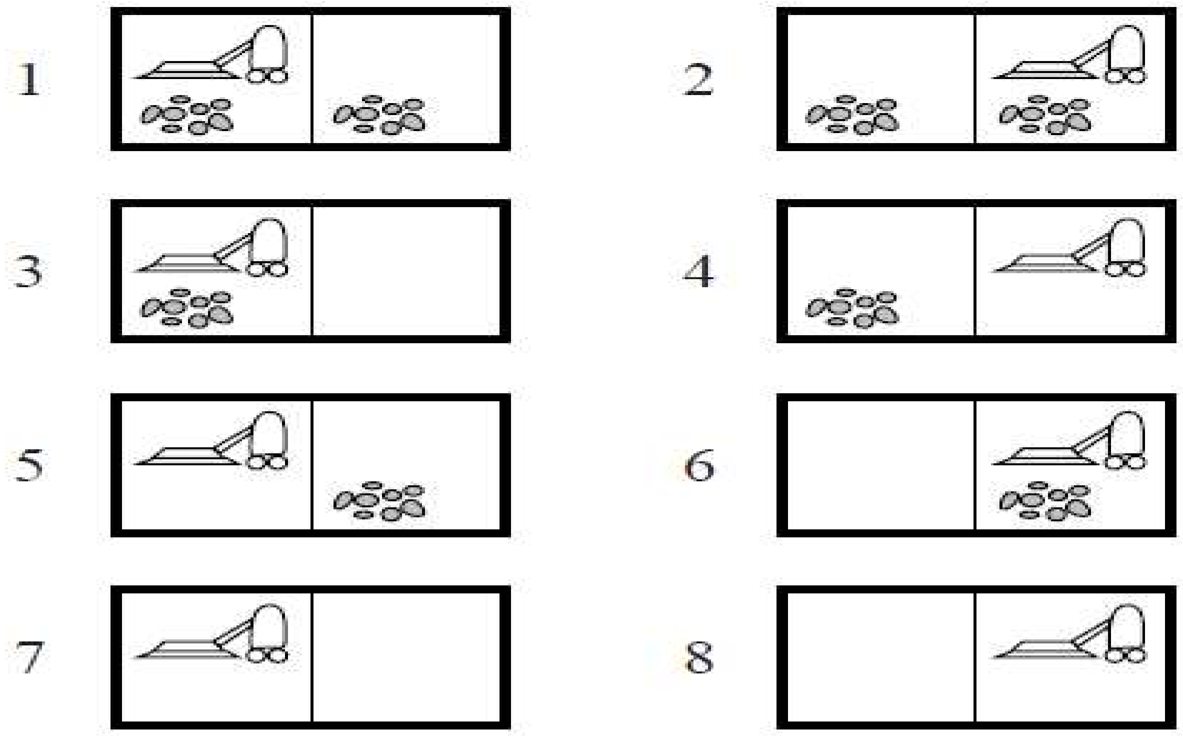


Figure 4.9 The eight possible states of the vacuum world; states 7 and 8 are goal states.

Searching with Nondeterministic Actions

- In the nondeterministic case, the result of an action can vary.

Erratic Vacuum World:

- When sucking a dirty square, it cleans it and sometimes cleans up dirt in an adjacent square.
- When sucking a clean square, it sometimes deposits dirt on the carpet.

Generalization of State-Space Model

1. Generalize the transition function to return a set of possible outcomes.

   oldf: S x A -> S   newf: S x A -> 2^S

2. Generalize the solution to a contingency plan.

   if state=s then action-set-1 else action-set-2

3. Generalize the search tree to an AND-OR tree.

AND-OR Search Tree

AND
Node

OR
Node

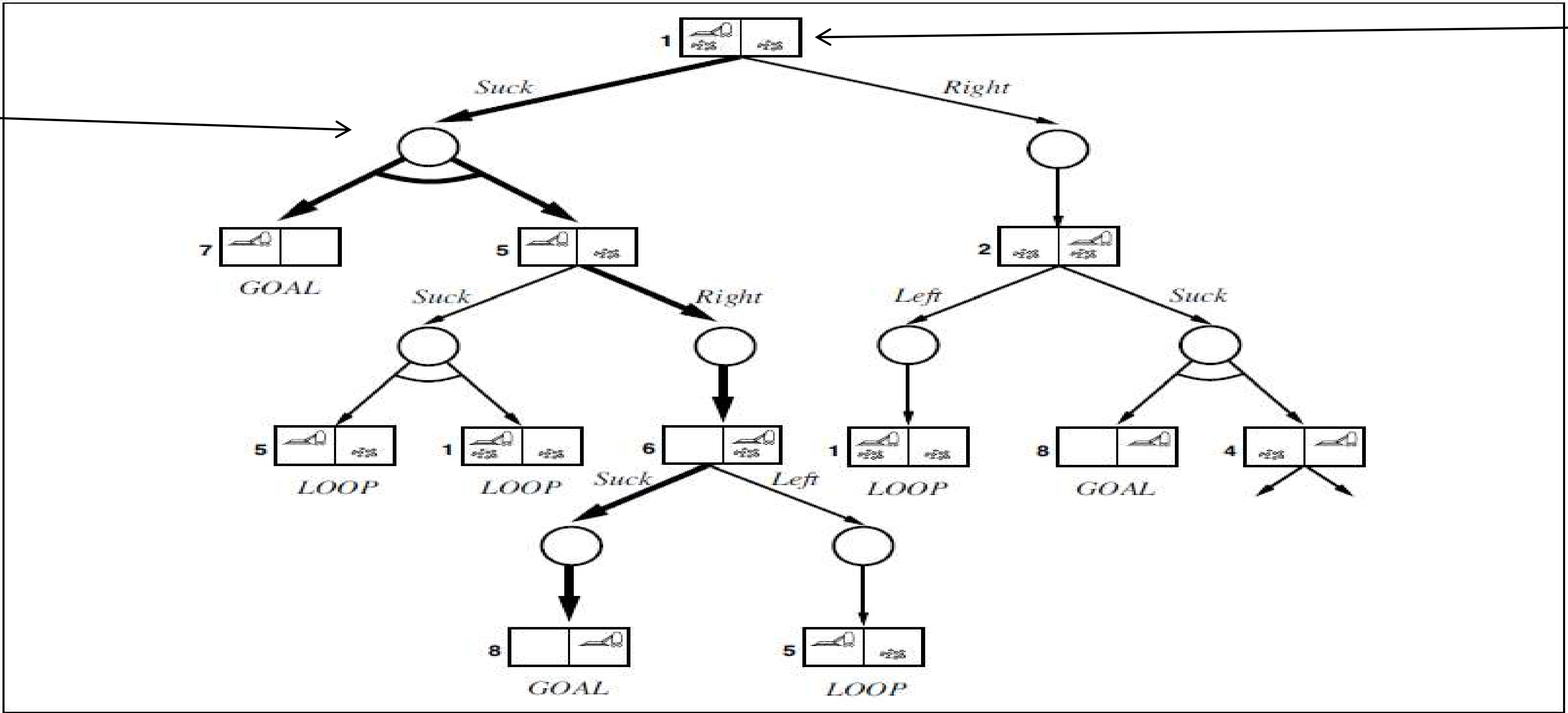


Figure 4.10 The first two levels of the search tree for the erratic vacuum world. State nodes are OR nodes where some action must be chosen. At the AND nodes, shown as circles, every outcome must be handled, as indicated by the arc linking the outgoing branches. The solution found is shown in bold lines.

Searching with Partial Observations

- The agent does not always know its state!
- Instead, it maintains a belief state: a set of possible states it might be in.
- Example: a robot can be used to build a map of a hostile environment. It will have sensors that allow it to “see” the world.

Belief State Space for Sensorless Agent

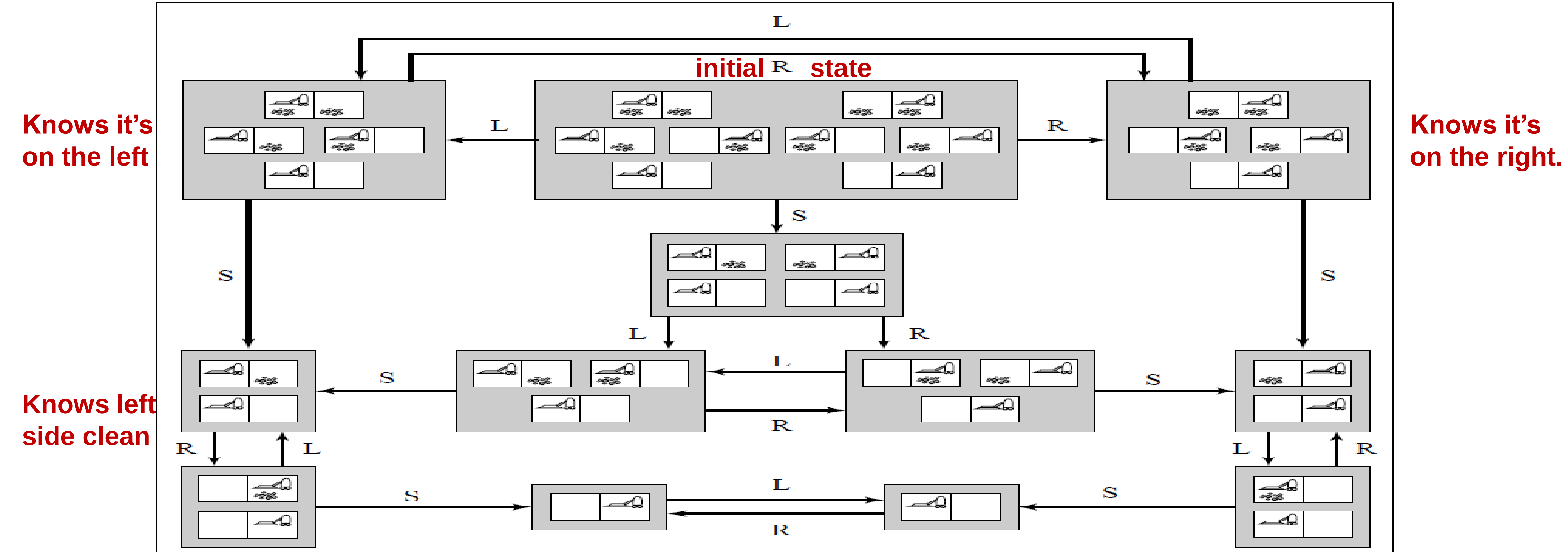


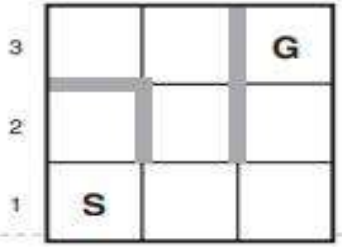
Figure 4.14 The reachable portion of the belief-state space for the deterministic, sensorless vacuum world. Each shaded box corresponds to a single belief state. At any given point, the agent is in a particular belief state but does not know which physical state it is in. The initial belief state (complete ignorance) is the top center box. Actions are represented by labeled links. Self-loops are omitted for clarity.

Online search

- Off-line Search: solution is found before the agent starts acting in the real world
- On-line search: interleaves search and acting
  - Necessary in unknown environments
  - Useful in dynamic and semi-dynamic environments
  - Saves computational resource in non-deterministic domains (focusing only on the contingencies arising during execution)
    - Tradeoff between finding a guaranteed plan (to not get stuck in an undesirable state during execution) and required time for complete planning ahead
- Examples
  - Robot in a new environment must explore to produce a map
  - New born baby
  - Autonomous vehicles

Online search problems

- Different levels of ignorance
  - E.g., an explorer robot may not know "laws of physics" about its actions
- We assume deterministic & fully observable environment here
  - Also, we assume the agent knows $ACTIONS(s)$, $c(s, a, s')$ that can be used after knowing s' as the outcome, $GOAL\_TEST(s)$
- Agent must perform an action to determine its outcome
  - $RESULTS(s, a)$ is found by actually being in s and doing a
  - By filling $RESULTS$ map table, the map of the environment is found.
- Agent may access to a heuristic function



Competitive ratio

- Online path cost: total cost of the path that the agent actually travels
- Best cost: cost of the shortest path "if it knew the search space in advance"
- Competitive ratio = Online path cost / Best path cost
  - Smaller values are more desirable
- Competitive ratio may be infinite
  - Dead-end state: no goal state is reachable from it
    - irreversible actions can lead to a dead-end state

Algorithms for online search

- Offline search: node expansion is a simulated process rather than exerting a real action
  - Can expand a node somewhere in the state space and immediately expand a node elsewhere

- Online search: can discover successors only for the physical current node
  - Expand nodes in a local order
  - Interleaving search & execution

Online search agents

- Online DFS
  - Physical backtrack (works only for reversible actions)
    - Goes back to the state from which the agent most recently entered the current state
    - Works only for state spaces with reversible actions

- Online local search: hill-climbing
  - Random walk instead of random restart
    - Randomly selecting one of available actions (preference to untried actions)
  - Adding Memory (Learning Real Time A*): more effective
    - To remember and update the costs of all visited nodes.

Definition

- A constraint satisfaction problem consists of three components,

  X, D, and C:
- X is a set of **variables,** {X1,...,Xn}.
- D is a set of **domains**, {D1,...,Dn}, one for each variable.
- C is a set of **constraints** that specify allowable combinations of values.
- Each domain Di consists of a set of allowable values, **{v1,...,vk}** for variable **Xi.**
- **Each constraint Ci consists of a pair {scope, rel}**
- **scope** is a tuple of variables that participate in the constraint
- **rel** is a relation that defines the values that those variables can take on.

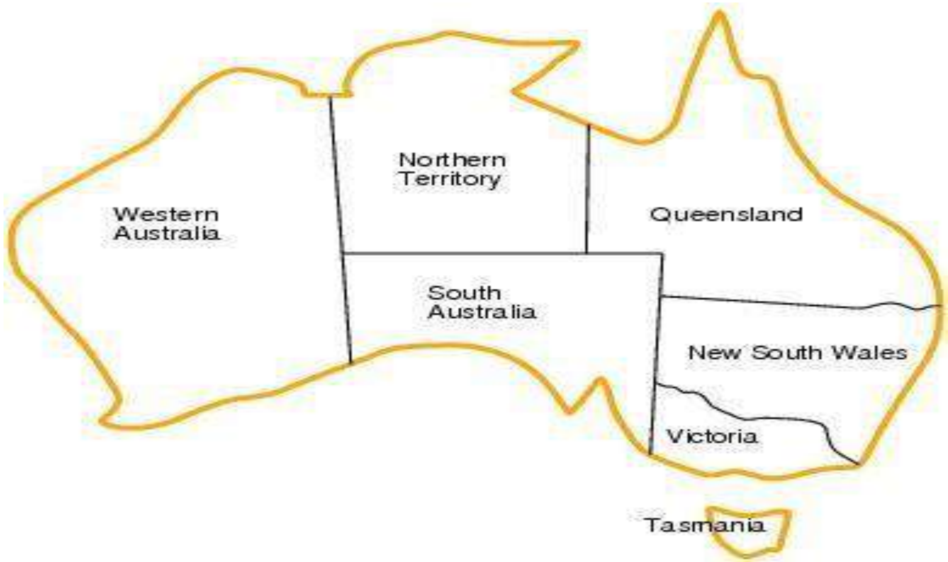
Constraint satisfaction problems

- For example, if X1 and X2 both have the domain {A,B}
- The constraint saying the two variables must have different values can be written as
- **{(X1, X2), [(A, B),(B,A)]}** or
- **{(X1, X2), X1 ≠ X2}**
- A ***state*** is defined as an ***assignment*** **of values to some or all variables.**
- ***Consistent assignment*****:** assignment does not violate the constraints.

Constraint satisfaction problems

- An assignment is ***complete*** when every value is mentioned.
- A ***solution*** to a CSP is a complete assignment that satisfies all constraints.
- Some CSPs require a solution that maximizes an ***objective function*.**
- **Applications:** Scheduling the time of observations on the Hubble Space Telescope, Floor planning, Map coloring, Cryptography

CSP example: Map Coloring

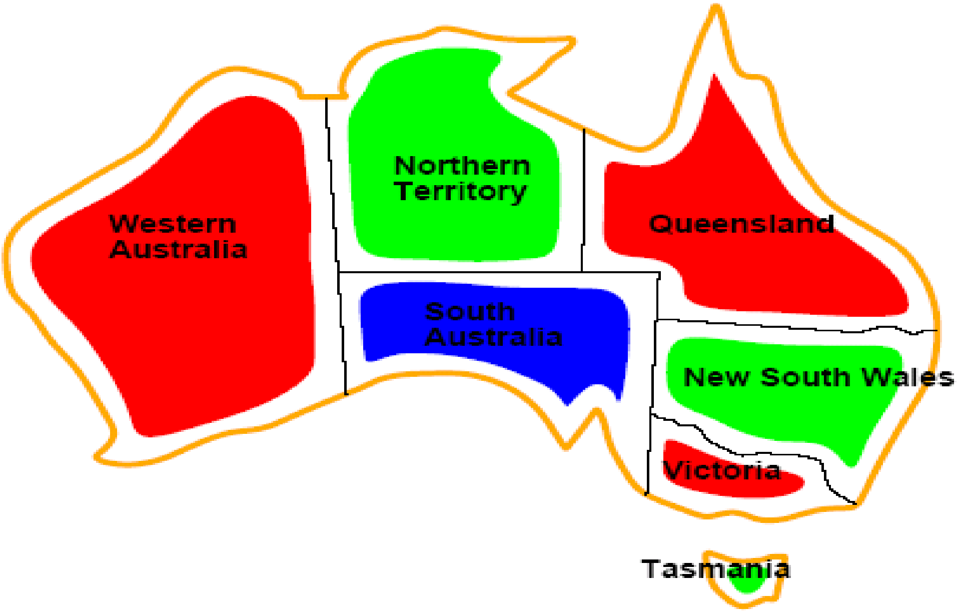


- **Variables:** *WA, NT, Q, NSW, V, SA, T*
- **Domains:** $D_i = \{red, green, blue\}$
- **Constraints:** adjacent regions must have different colors.

E.g. $WA \neq NT$ **(if the language allows this)**

E.g. $(WA,NT) = \{(red,green),(red,blue),(green,red),\ldots\}$

CSP example: Map Coloring



Solutions are assignments satisfying all constraints

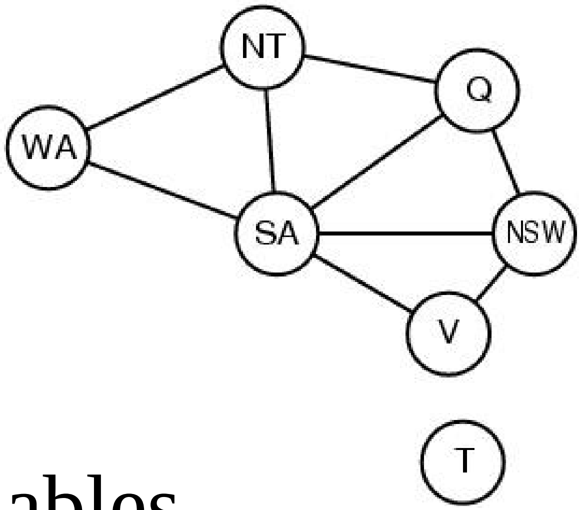
e.g.

{WA=red,NT=green,Q=red,NSW=green,V=red,SA=blue, T=green}

Constraint graph

CSP benefits

- **Standard representation pattern**
- **Generic goal and successor functions**
- **Generic heuristics (no domain specific expertise).**



Constraint graph = nodes are variables, edges show constraints.

e.g. Tasmania is an independent subproblem.

Graph can be used to simplify search.

Varieties of CSPs

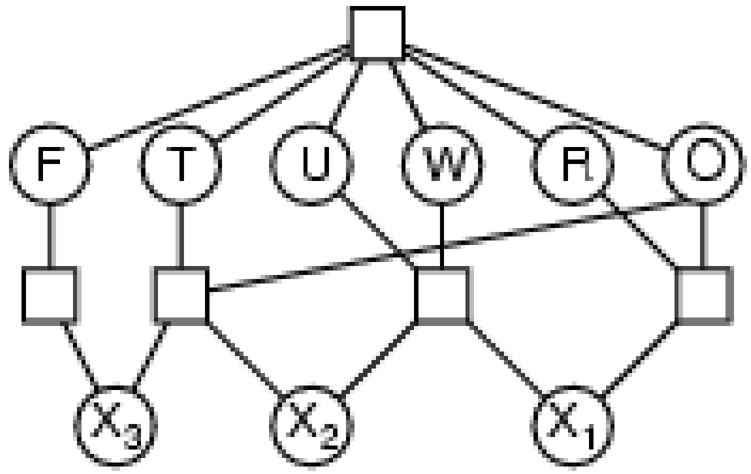
- Discrete variables
  - **Finite domains; size *d* $\Rightarrow$ *O(d ^{n})* complete assignments.**
    - E.g. Boolean CSPs,Map Coloring, Job Scheduling.
  - **Infinite domains (integers, strings, etc.)**
    - E.g. job scheduling, variables are start/end days for each job
    - Need a constraint language e.g $StartJob_1 + 5 \leq StartJob_3$.
    - Linear constraints solvable, nonlinear undecidable.
- Continuous variables
  - **e.g. start/end times for Hubble Telescope observations.**
  - **Linear constraints solvable in poly time by LP methods.**

Varieties of constraints

- **Unary** constraints involve a single variable,
  - e.g., SA ≠ green
- **Binary** constraints involve pairs of variables,
  - e.g., SA ≠ WA
- **Higher-order** constraints involve 3 or more variables,
  - e.g., SA ≠ WA ≠ NT,cryptharithmetic column constraints.

Example: Cryptharithmetic

```
    T W O
  + T W O
  -------
  F O U R
```



- **Variables**: $F\ T\ U\ W\ R\ O\ X_1\ X_2\ X_3$
- **Domains**: $\{0,1,2,3,4,5,6,7,8,9\}$
- **Constraints**: *Alldiff (F,T,U,W,R,O)*
  [redacted]
  - $X_3 = F$, $T \neq 0$, $F \neq 0$

Example: Cryptarithmetic

- **Variables**

D, E, M, N, O, R, S, Y

- **Domains**

{0, 1, 2, 3 ,4, 5, 6, 7, 8, 9 }

- **Constraints**

M ≠ 0, S ≠ 0 (unary constraints)

Y = D + E OR **Y = D + E – 10.**

D ≠E, D ≠ M, D ≠ N, etc.

```
  S E N D
+ M O R E
M O N E Y
```

Constraint Propagation

- ➢**In regular state-space search**:

  An algorithm can do only one thing: **search.**

- ➢**In CSPs there is a choice:**

  An **algorithm can search or**

  do a specific type of inference called **constraint propagation**, using the constraints to reduce the number of legal values for a variable, which in turn can reduce the legal values for another variable, and so on.

- ➢Constraint propagation may be intertwined with search, or it may be done as a preprocessing step, before search starts.

- ➢Sometimes this **preprocessing can solve the whole problem**, so no search is required at all.

Constraint Propagation

- ***V*** = variable being assigned at the current level of the search
- Set variable ***V*** to a value in *D*(***V***)
- For every variable ***V'*** connected to ***V***:
  - –Remove the values in *D*(***V'***) that are inconsistent with the assigned variables
  - –For every variable ***V''*** connected to ***V'***:
    - Remove the values in *D*(***V''***) that are no longer possible candidates
    - And do this again with the variables connected to ***V''***

–……..until no more values can be discarded

Local Consistency

- The key idea is **local consistency**.
- If we treat **each variable as a node** in a graph and each **binary constraint as an arc**.
- The process of **enforcing local consistency in each part of the graph causes inconsistent values to be eliminated** throughout the graph.

Node Consistency

- **Node consistency:**

  A single variable (corresponding to a node in the CSP network) is node-consistent **if all the values in the variable's domain satisfy the variable's unary constraints.**

Eg: The **variable SA** starts with domain **{red, green, blue},** and we can make it **node consistent** by **eliminating green**, leaving **SA with the reduced domain {red, blue}.**

- Network is node-consistent if every variable in the network is node-consistent.
- It is always possible to eliminate all the unary constraints in a CSP by running node consistency.
- It is also possible to transform all n-ary constraints into binary ones.

Arc Consistency

- **Arc consistency**:

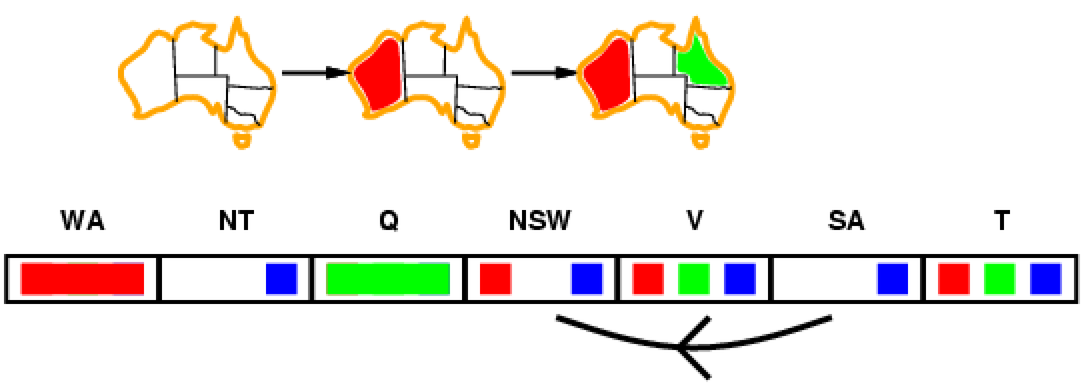
  A variable in a CSP is **arc-consistent** if **every value in its domain satisfies the variable's binary constraints.**

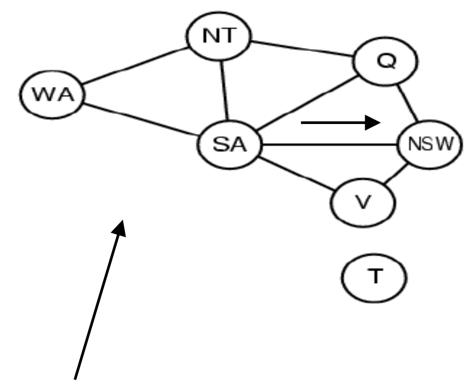
  Xi is arc-consistent with respect to another variable Xj if **for every value in the current domain Di there is some value in the domain Dj that satisfies the binary constraint on the arc (Xi, Xj).**

Arc consistency

- Simplest form of propagation makes each **arc consistent**
- *X* → *Y* is consistent iff

  for every value *x* of *X* there is some allowed *y*



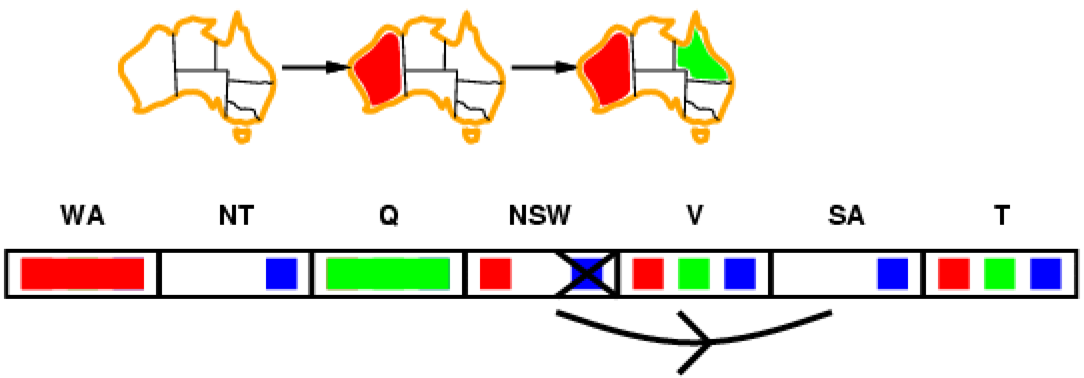


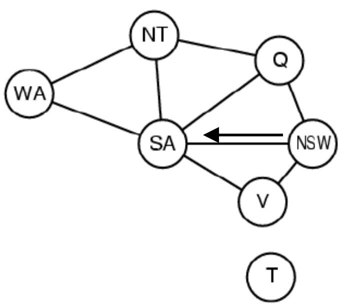
constraint propagation propagates arc consistency on the graph.

Arc consistency

- Simplest form of propagation makes each arc consistent
- $X \rightarrow Y$ is consistent iff

  for every value *x* of *X* there is some allowed *y*

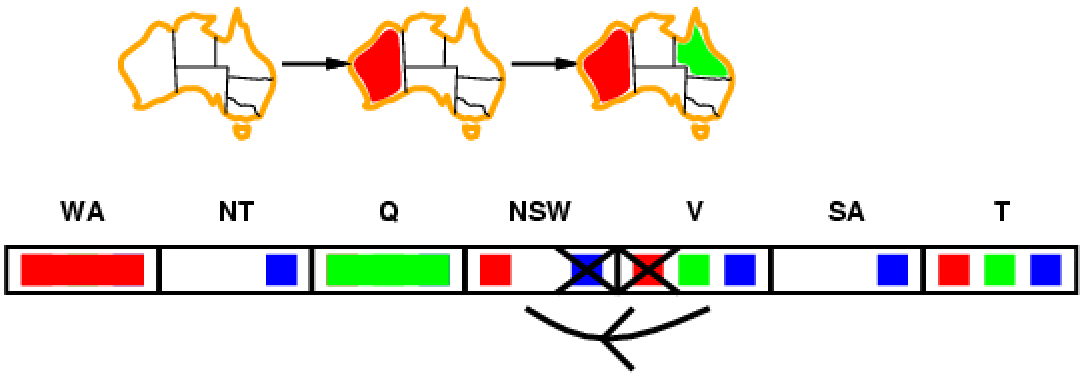


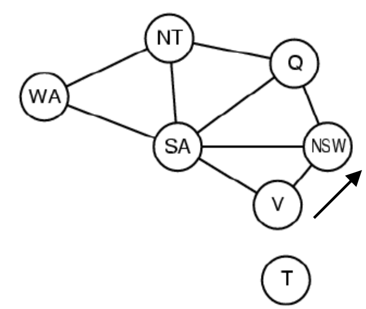


Arc consistency

- Simplest form of propagation makes each arc consistent
- *X* → *Y* is consistent iff

  for every value *x* of *X* there is some allowed *y*



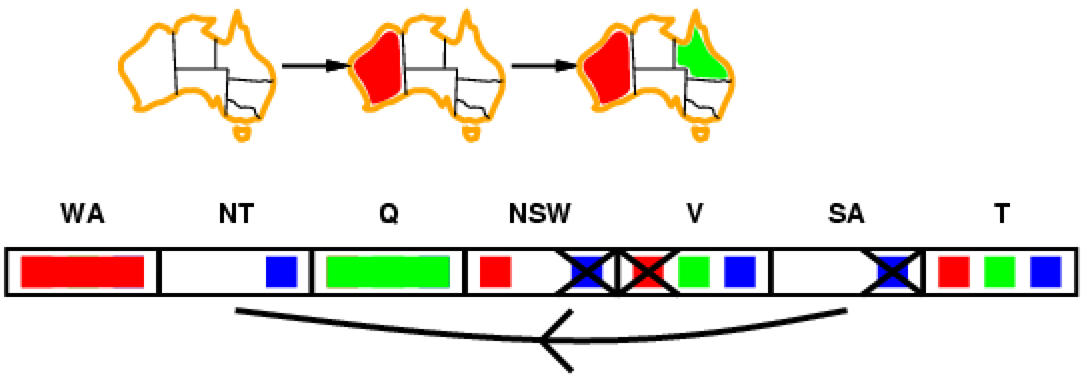


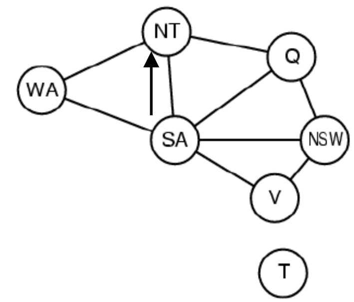
- **If *X* loses a value, neighbors of *X* need to be rechecked**

Arc consistency

- Simplest form of propagation makes each arc consistent
- $X \rightarrow Y$ is consistent iff

  for every value *x* of *X* there is some allowed *y*





- If *X* loses a value, neighbors of *X* need to be rechecked
- Arc consistency detects failure earlier than forward checking
- Can be run as a preprocessor or after each assignment

Time complexity: $O(n^2d^3)$

Path consistency

- Arc consistency reduces the domains of variables,
  - sometimes finding a solution (by reducing every domain to size 1) and
  - sometimes finding that the CSP cannot be solved (by reducing some domain to size 0).
- **Eg:** Map-coloring problem on Australia, but with only two colors allowed, red and blue.
- Arc consistency can do nothing because **every variable is already arc consistent:** each can be red with blue at the other end of the arc (or vice versa).
- But clearly there is **no solution** to the problem: because Western Australia, Northern Territory and South Australia all touch each other, we need at least three colors for them alone.

Path consistency

- Arc consistency tightens down the domains (unary constraints) using the arcs (binary constraints).
- **Path consistency tightens the binary constraints** by using implicit constraints that are inferred by looking at triples of variables.
- **A two-variable set {Xi, Xj} is path-consistent** with respect to a **third variable Xm** if, for every assignment {Xi = a, Xj = b} consistent with the constraints on {Xi, Xj}, there is an assignment to Xm that satisfies the constraints on {Xi, Xm} and {Xm, Xj}.
- This is called path consistency because one can think of it as looking at a path from Xi to Xj with Xm in the middle.

Continued…

- In this case, there are only two: **{WA = red, SA = blue}** and **{WA = blue, SA = red}.**
- With both of these assignments **NT can be neither red nor blue** (because it would conflict with either WA or SA).
- Because there is no valid choice for NT, we eliminate both assignments, and we end up with **no valid assignments for {WA, SA}.**

K-consistency

- Stronger forms of propagation can be defined with the notion of k-consistency.
- A CSP is **k-consistent if, for any set of k – 1 variables and for any consistent assignment to those variables, a consistent value can always be assigned to any kth variable.**
- **1-consistency** says that, given the empty set, we can make any set of one variable consistent: this is what we called **node consistency**.
- **2-consistency** is the same as **arc consistency**.
- For **binary constraint networks**, **3-consistency** is the same as **path consistency**.

Continued…

- A CSP is **strongly k-consistent** if it is **k-consistent** and is also **(k – 1)-consistent**, **(k – 2)-consistent,...** all the way down to **1-consistent.**

Global constraints

- Global constraint is one **involving an arbitrary number of variables.**
- **Global constraints occur frequently in real problems** and can be handled by special-purpose algorithms that are **more efficient than the general-purpose methods.**
- For example, the **Alldiff constraint** says that all the variables involved must have distinct values
- One simple form of **inconsistency detection** for Alldiff constraints works as follows:
  - if **m variables are involved in the constraint**, and if they have **n possible distinct values altogether**, and **m>n**, then the **constraint cannot be satisfied.**

Continued…

- Another important higher-order constraint is the resource constraint, sometimes called the **atmost constraint.**
- **Eg:** In a scheduling problem, let P1,...,P4 denote the numbers of personnel assigned to each of four tasks.
- The constraint that **no more than 10 personnel are assigned in total** is written as Atmost(10, P1, P2, P3, P4).
- Special propagation algorithms

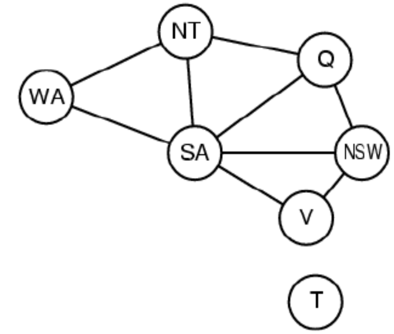
Bound propagation

  - E.g., number of people on two flight D1 = [0, 165] and D2 = [0, 385]
  - Constraint that the total number of people has to be at least 420
  - Propagating bounds constraints yields D1 = [35, 165] and D2 = [255, 385]

CSP as a standard search problem

- A CSP can easily expressed as a standard search problem.
- Incremental formulation
  - ***Initial State***: **the empty assignment {}.**
  - ***Successor function***: **Assign value to unassigned variable provided that there is not conflict.**
  - ***Goal test***: **the current assignment is complete.**
  - ***Path cost***: **as constant cost for every step.**

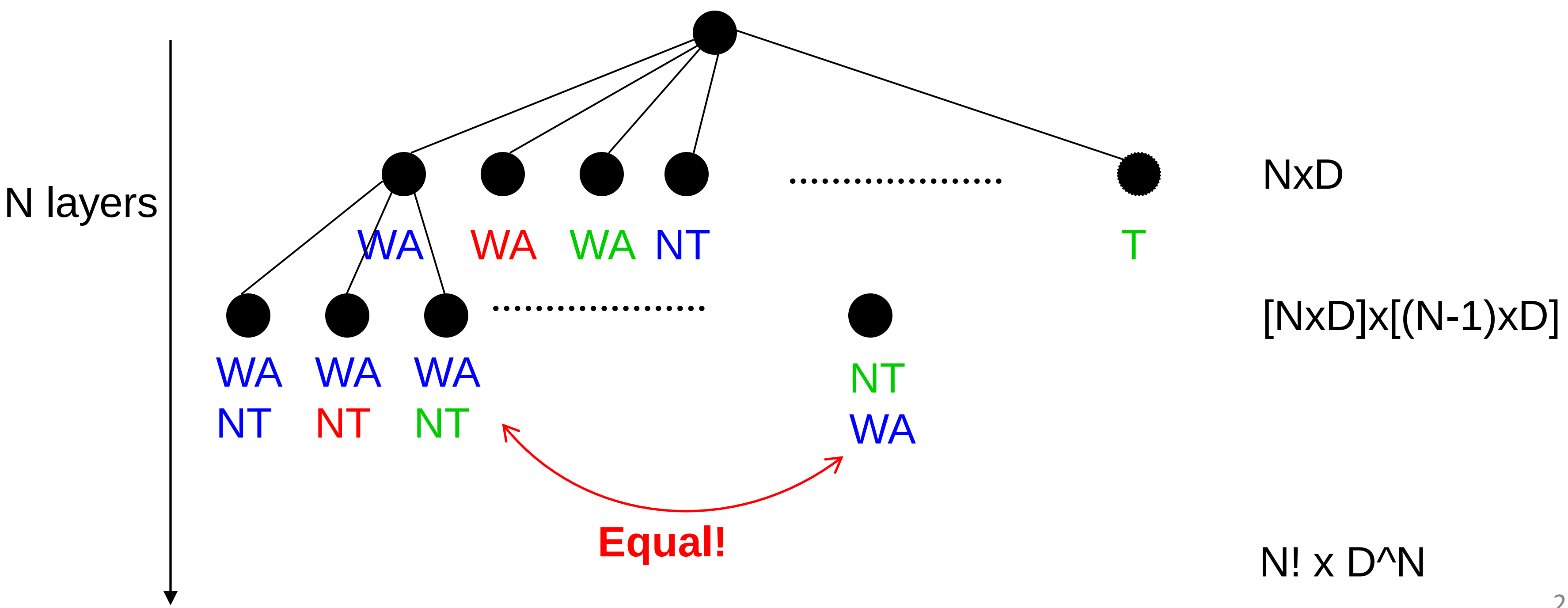
Standard search formulation



Let’s try the standard search formulation.

We need:

- Initial state**: none of the variables has a value** (color)
- Successor state: **one of the variables without a value will get some value.**
- Goal: **all variables have a value and none of the constraints is violated**.



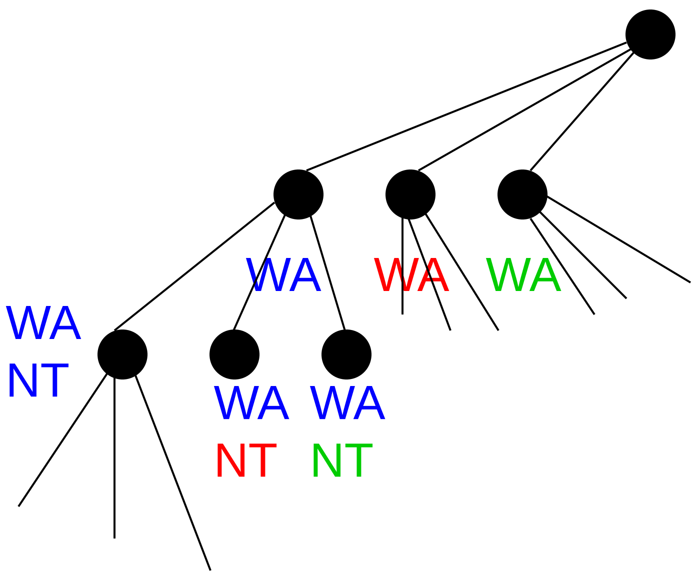
There are N! x D^N nodes in the tree but only D^N distinct states??

Backtracking (Depth-First) search

- Special property of CSPs: They **are commutative**:
  This means: the order in which we assign variables
  does not matter.

NT **WA** = **WA** **NT**

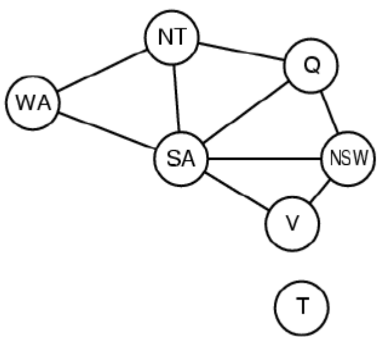
- **Better search tree: First order variables, then assign them values one-by-one.**



D

D^2

D^N



Backtracking search

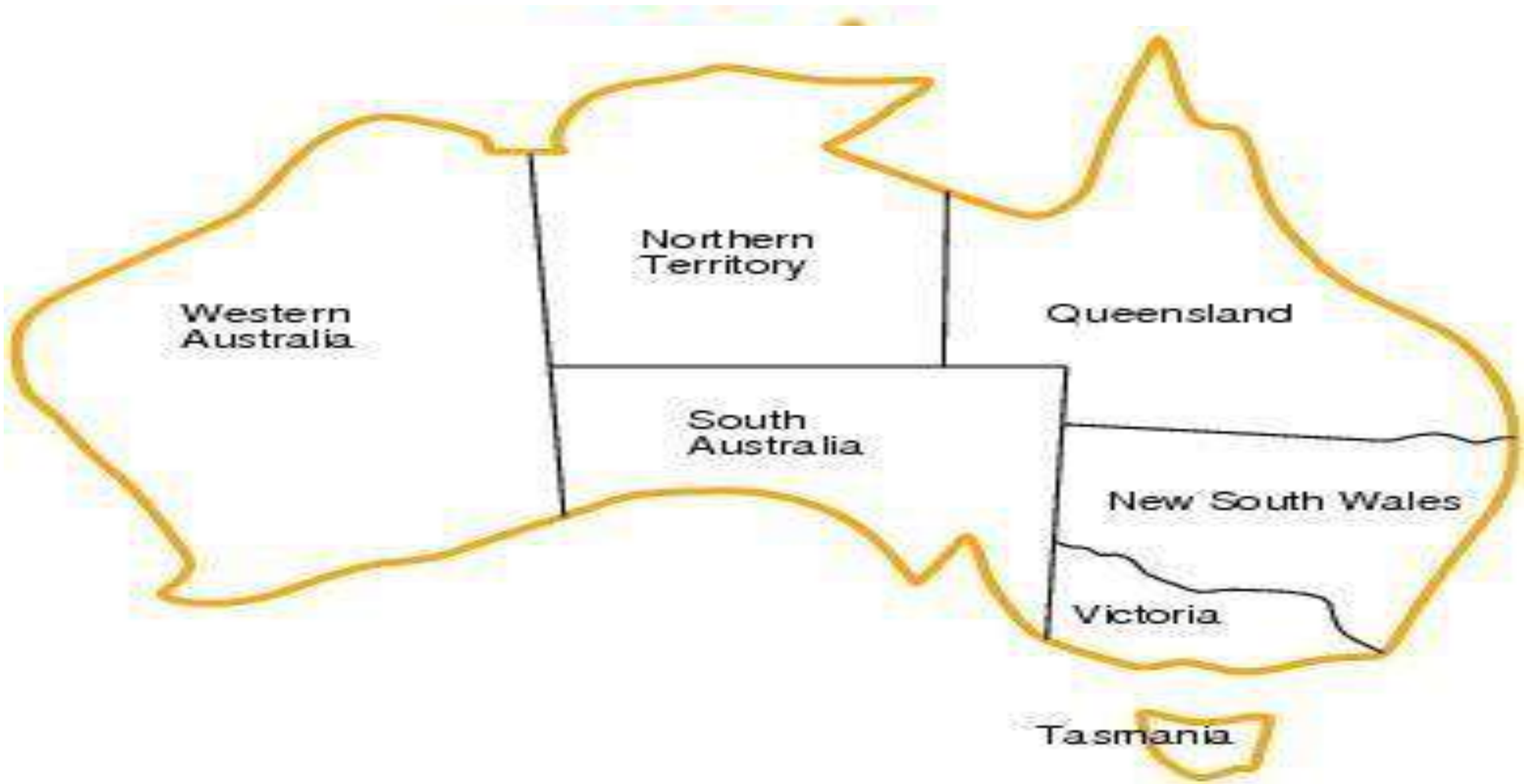
- **Depth-first search**
- **Chooses values for one variable at a time and backtracks when a variable has no legal values left to assign.**
- Uninformed algorithm
- ✓ **No good general performance**

Backtracking search

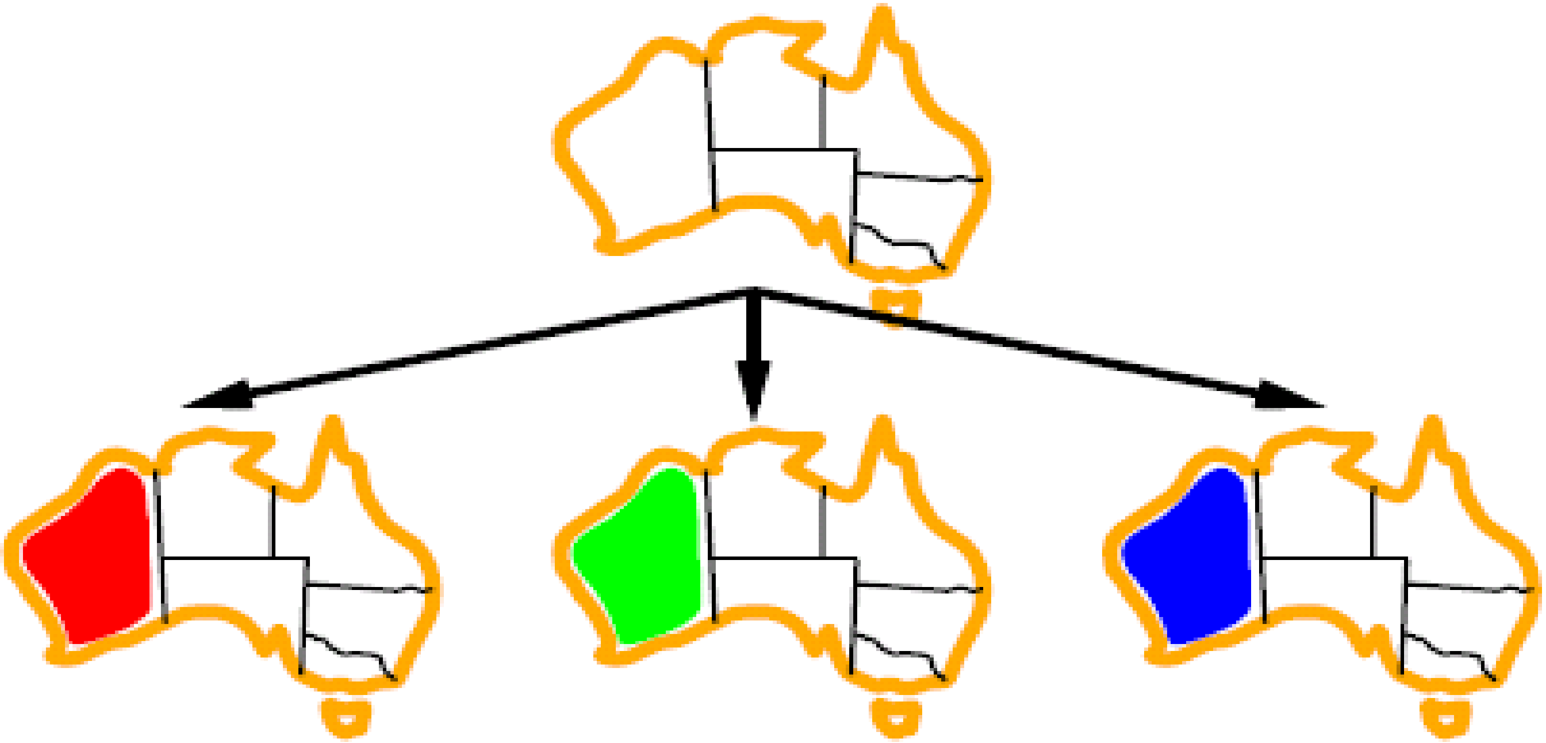
function BACKTRACKING-SEARCH(*csp*) **return** a solution or failure
return RECURSIVE-BACKTRACKING(*{}* , *csp*)

function RECURSIVE-BACKTRACKING(*assignment*, *csp*) **return** a solution or failure
if *assignment* is complete **then return** *assignment*
var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[*csp*],*assignment*,*csp*)
for each *value* **in** ORDER-DOMAIN-VALUES(*var*, *assignment*, *csp*) **do**
if *value* is consistent with *assignment* according to CONSTRAINTS[*csp*] **then**
add *{var=value}* to assignment
result ← RRECURSIVE-BACTRACKING(*assignment*, *csp*)
if *result* ≠ *failure* **then return** *result*
remove *{var=value}* from *assignment*
return *failure*

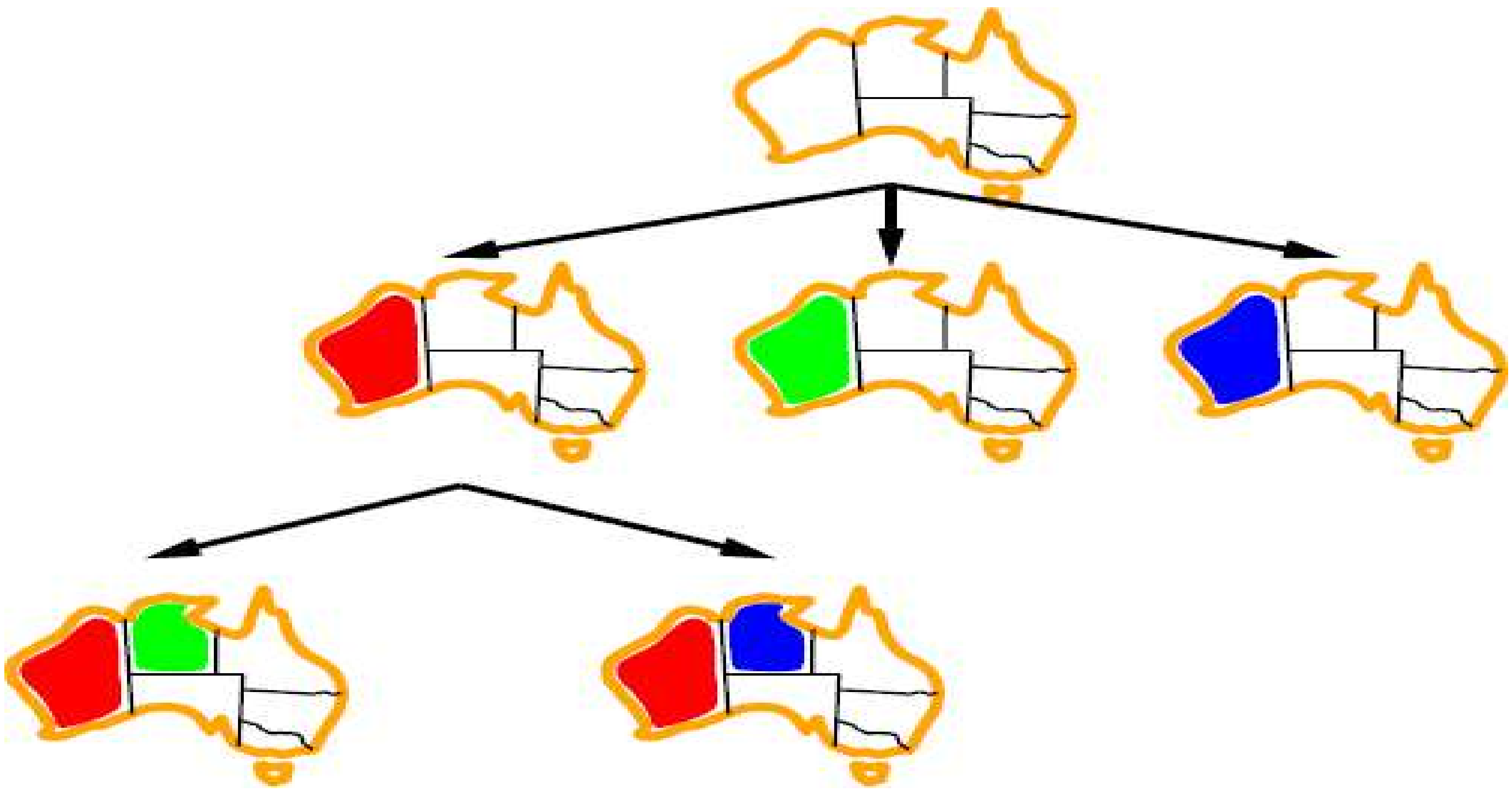
Backtracking example



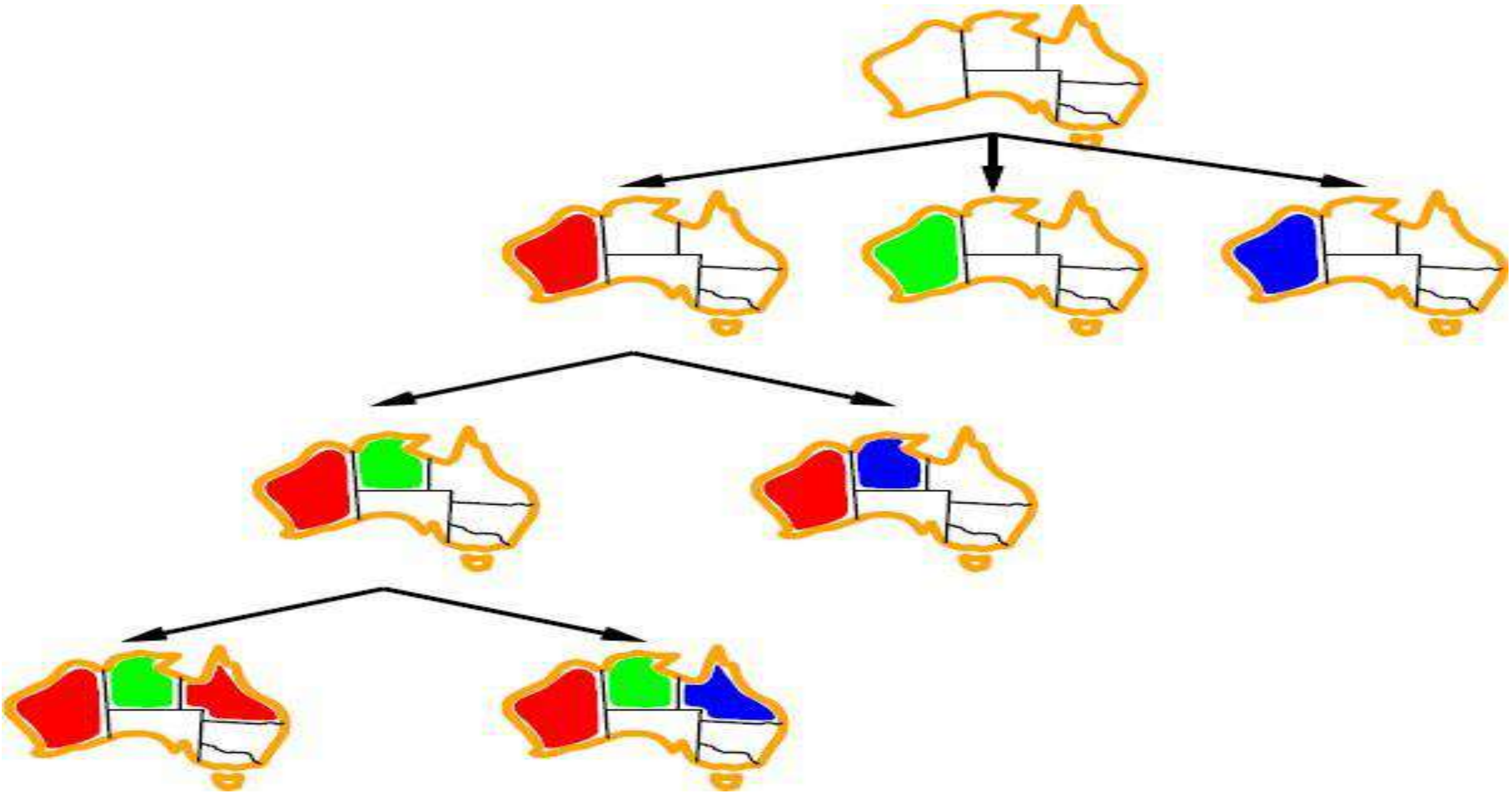
Backtracking example



Backtracking example



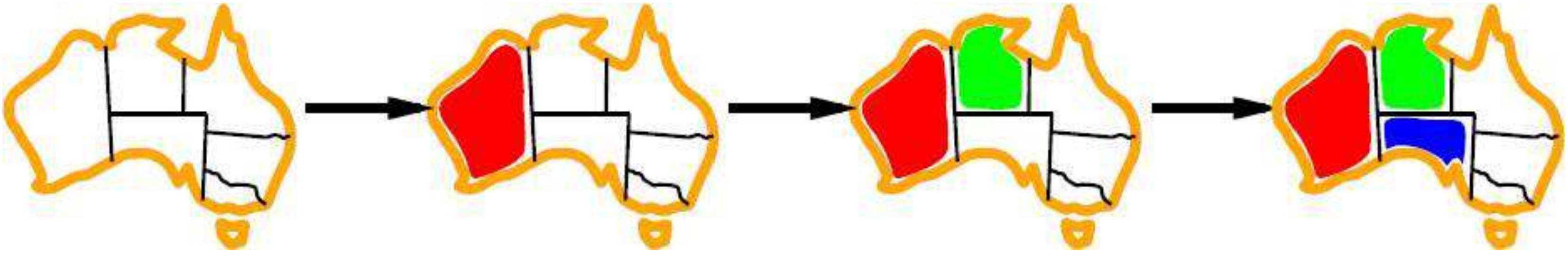
Backtracking example



Improving backtracking efficiency

- Previous improvements $\rightarrow$ introduce heuristics
- General-purpose methods can give huge gains in speed:
  - **Which variable should be assigned next?**
  - **In what order should its values be tried?**
  - **Can we detect inevitable failure early?**
  - **Can we take advantage of problem structure?**

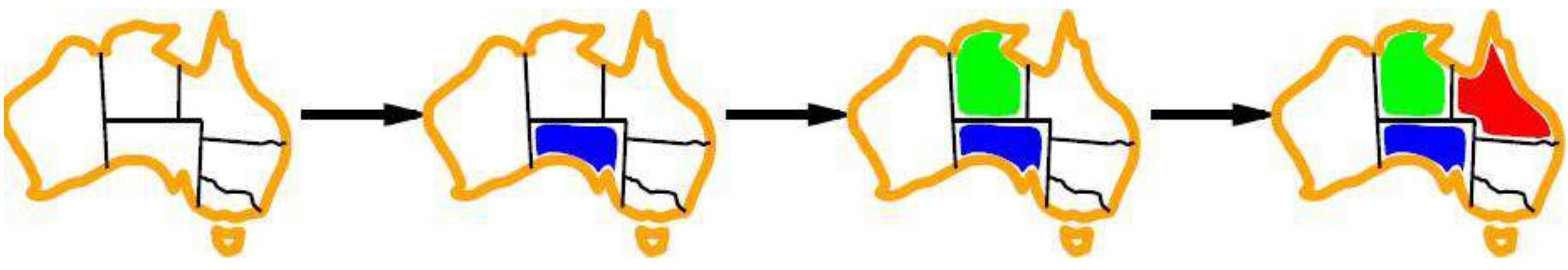
Minimum remaining values



var ← **SELECTUNASSIGNEDVARIABLE**(VARIABLES[*csp*],*assignment*,*csp*)

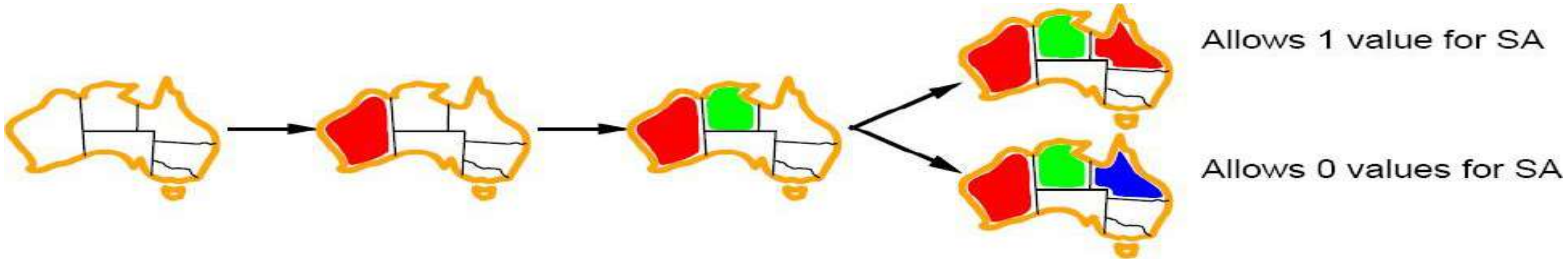
- A.k.a. most constrained variable heuristic
- ***Rule*****:** choose variable with the fewest legal moves
- ***Which variable shall we try first?***

Degree heuristic



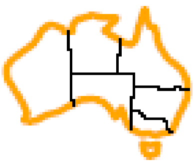
- ➢Use degree heuristic
- ➢***Rule*****:** Select variable that is involved in the largest number of constraints on other unassigned variables.
- ➢Degree heuristic is very useful as a tie breaker.
- ➢***In what order should its values be tried?***

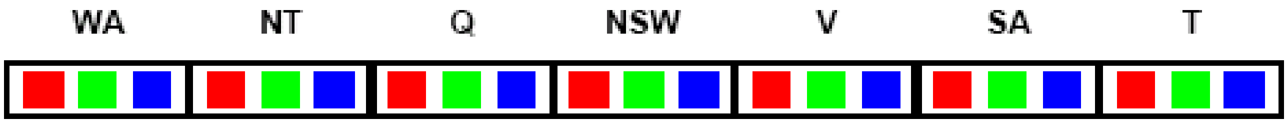
Least constraining value



- Least constraining value heuristic
- **Rule:** given a variable choose the least constraining value i.e. the one that leaves the maximum flexibility for subsequent variable assignments.

Forward checking



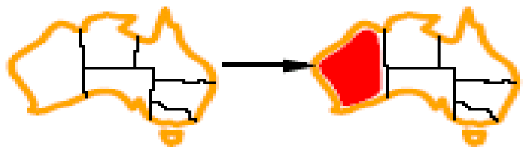


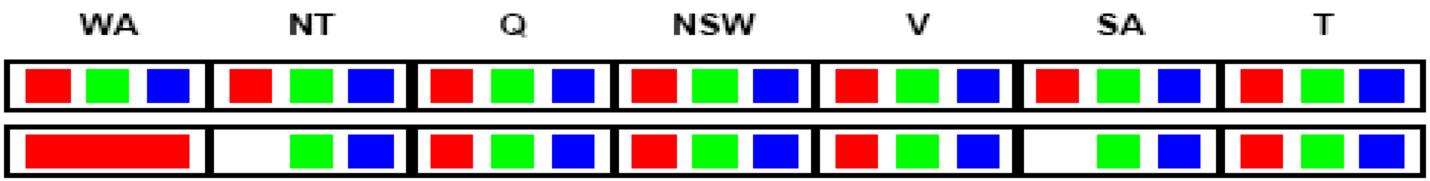
- Can we detect inevitable failure early?

  And avoid it later?

- ***Forward checking idea:*** keep track of remaining legal values for unassigned variables.
- Terminate search when any variable has no legal values.

Forward checking

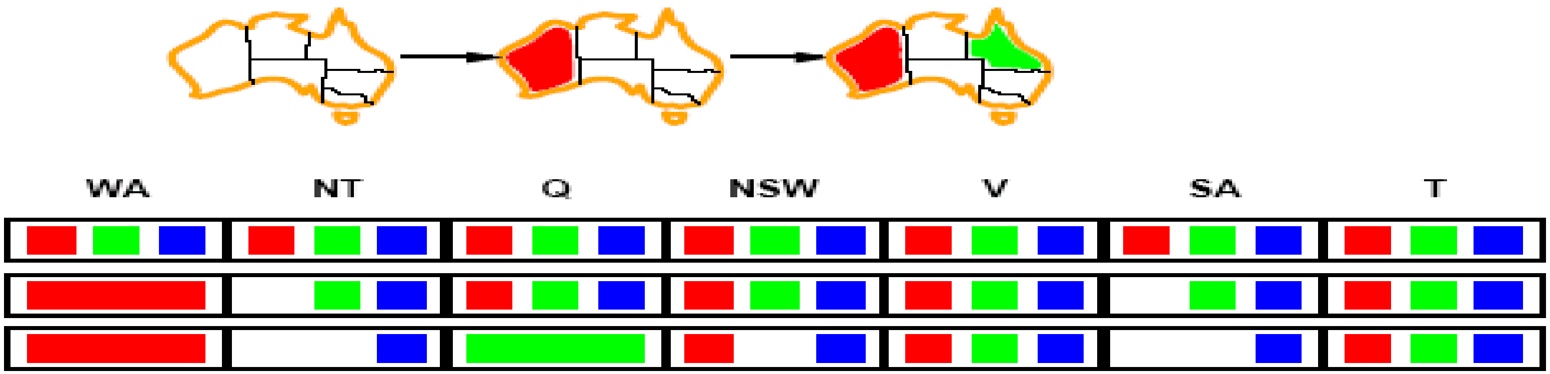




- Assign *{WA=red}*
- Effects on other variables connected by constraints with WA

NT can no longer be red
SA can no longer be red

Forward checking

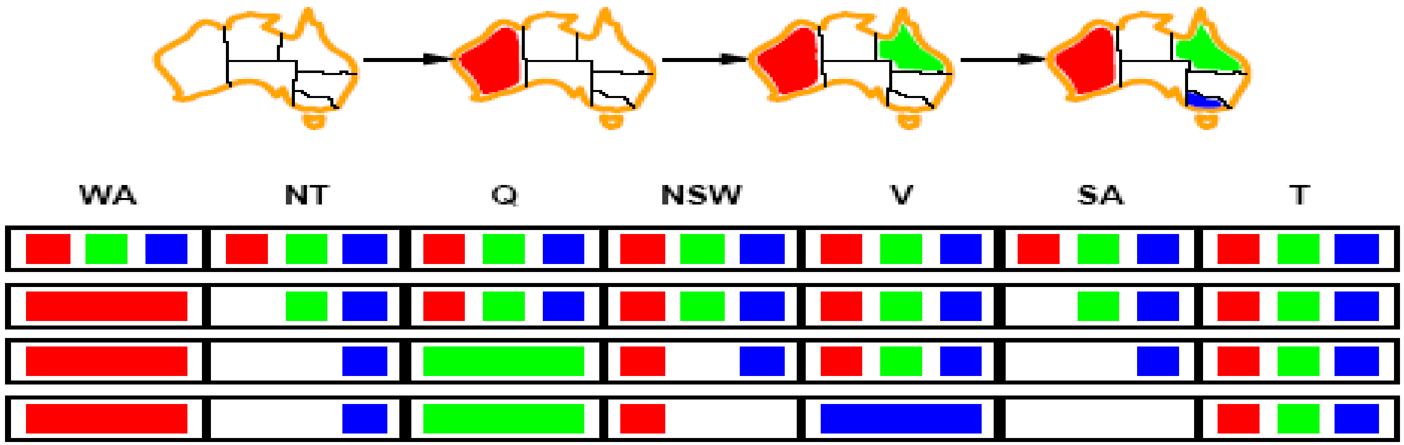


- **Assign *{Q=green}***
- Effects on other variables connected by constraints with WA

NT can no longer be green
NSW can no longer be green
SA can no longer be green

MRV heuristic will automatically select NT and SA next, why?

Forward checking



➢**If *V* is assigned *blue***

➢Effects on other variables connected by constraints with WA

SA is empty
NSW can no longer be blue

➢FC has detected that partial assignment is ***inconsistent*** with the constraints and backtracking can occur.

Local search for CSP

- Use complete-state representation
- For CSPs
  - **allow states with unsatisfied constraints**
  - **operators reassign variable values**
- **Variable selection:** randomly select any conflicted variable
- **Value selection:** *min-conflicts heuristic*
  - **Select new value that results in a minimum number of conflicts with the other variables**

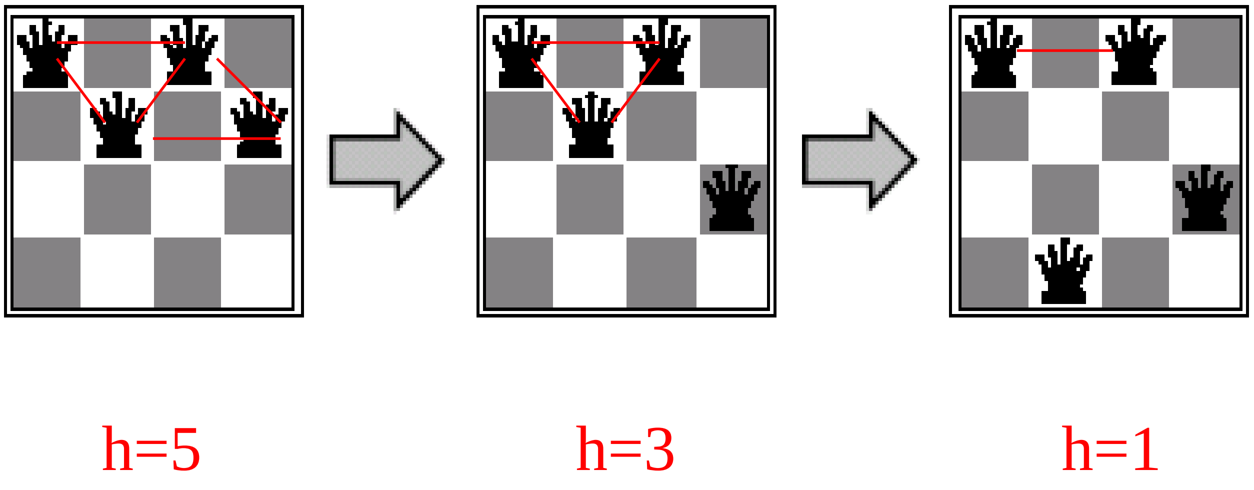
Local search for CSP

function MIN-CONFLICTS(*csp*, *max_steps*) **returns** a solution or failure
  inputs: *csp*, a constraint satisfaction problem
    max_steps, the number of steps allowed before giving up

  current ← an initial complete assignment for *csp*
  for i = 1 to *max_steps* **do**
    if *current* is a solution for *csp* **then return** *current*
    var ← a randomly chosen conflicted variable from *csp*.VARIABLES
    value ← the value v for *var* that minimizes CONFLICTS(*var*, v, *current*, *csp*)
    set *var* = *value* in *current*
  return *failure*

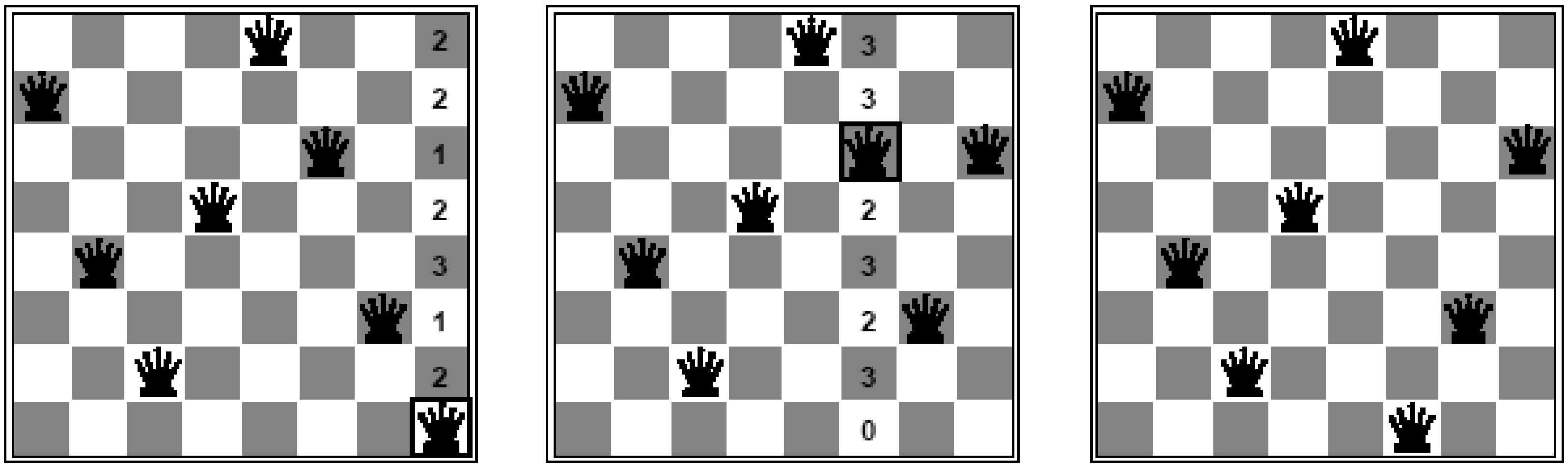
Figure 6.8 The MIN-CONFLICTS algorithm for solving CSPs by local search. The initial state may be chosen randomly or by a greedy assignment process that chooses a minimal-conflict value for each variable in turn. The CONFLICTS function counts the number of constraints violated by a particular value, given the rest of the current assignment.

Min-conflicts example 1



Use of min-conflicts heuristic in hill-climbing.

Min-conflicts example 2



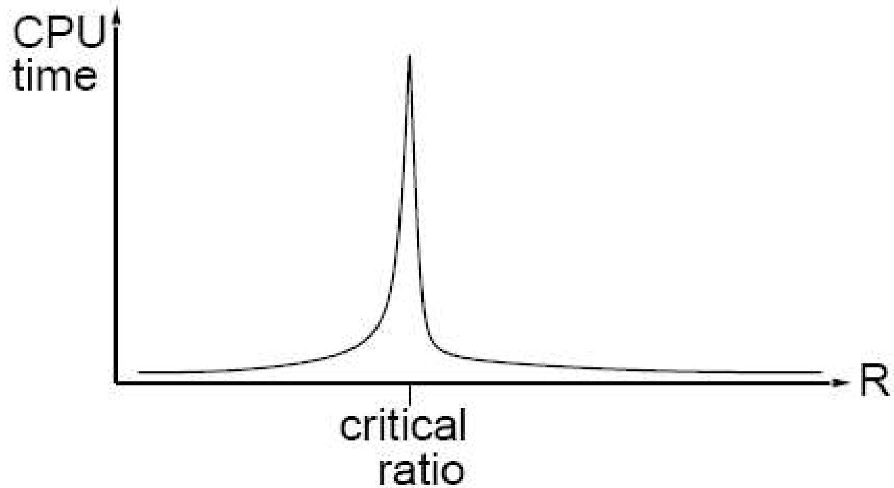
- ➢A two-step solution for an 8-queens problem using min-conflicts heuristic.
- ➢At each stage a queen is chosen for reassignment in its column.
- ➢The algorithm moves the queen to the min-conflict square breaking ties randomly.

Performance of min-conflicts

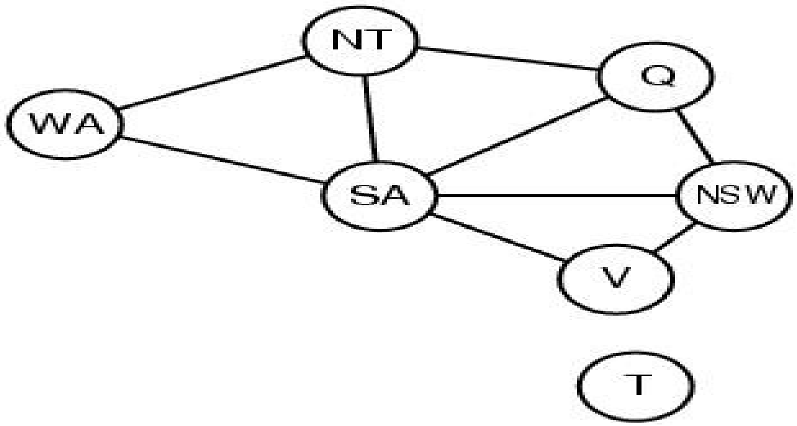
Given random initial state, can solve n-queens in almost constant time for arbitrary n with high probability (e.g., $n = 10{,}000{,}000$)

The same appears to be true for any randomly-generated CSP
except in a narrow range of the ratio

$$R = \frac{\text{number of constraints}}{\text{number of variables}}$$

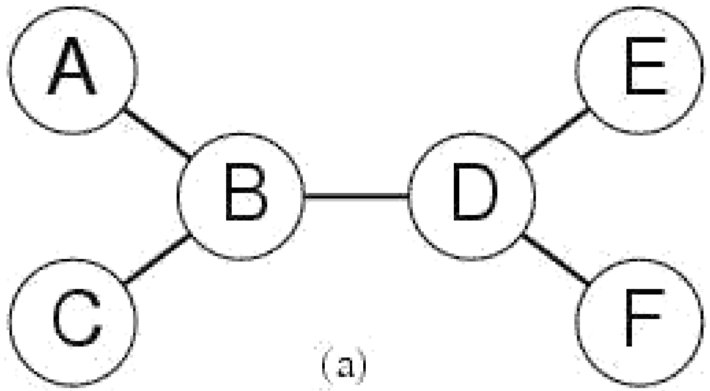


Problem structure



- ***How can the problem structure help to find a solution quickly?***
- Subproblem identification is important:
  - ✓**Coloring Tasmania and mainland are independent**
  - ✓**Subproblems Identifiable as connected components of constrained gr**
- Improves performance

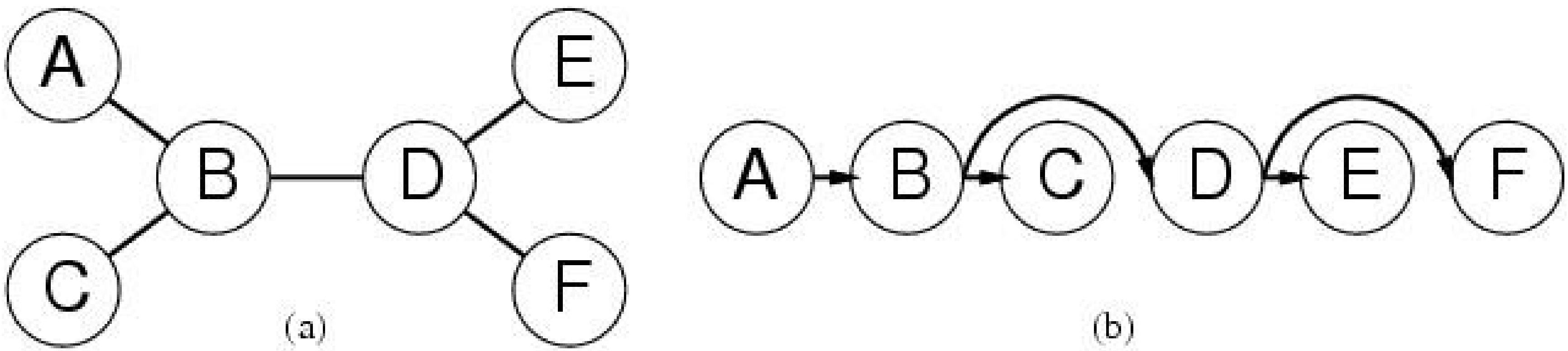
Tree-structured CSPs



➢**Theorem:** if the constraint graph has no loops then CSP can be solved in $O(nd^2)$ time

➢Compare difference with general CSP, where worst case is $O(d^n)$

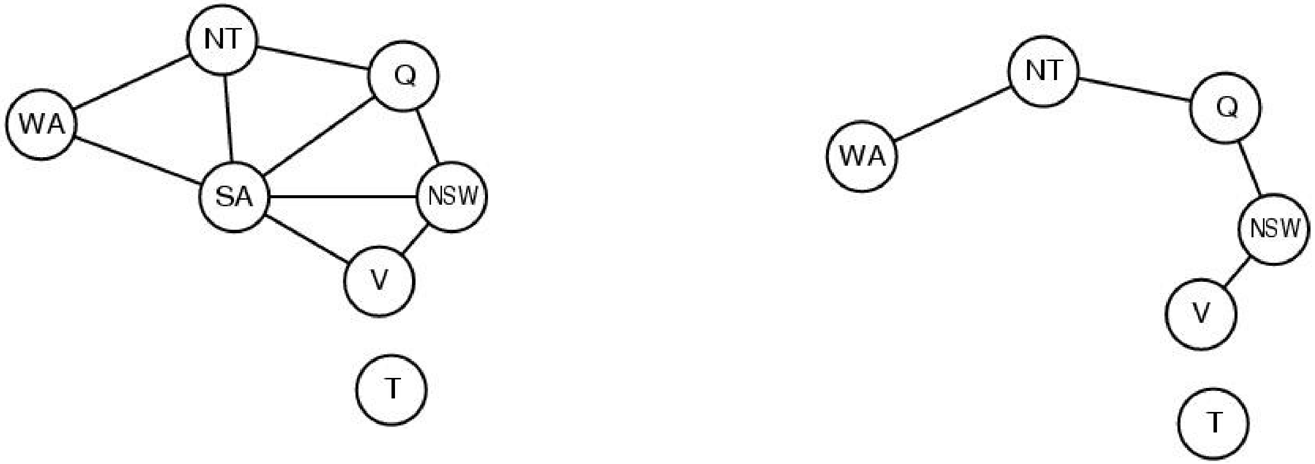
Tree-structured CSPs



- In most cases subproblems of a CSP are connected as a tree
- Any tree-structured CSP can be solved in time linear in the number of variables.

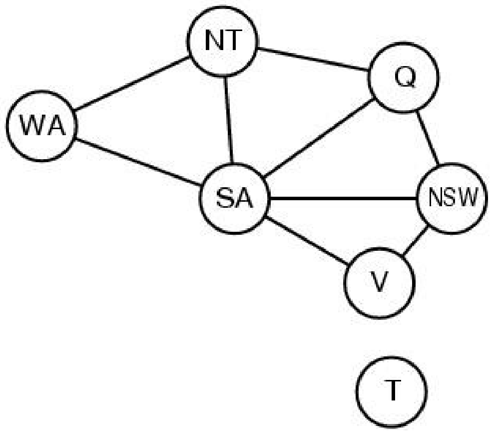
- **Choose a variable as root, order variables from root to leaves such that every node's parent precedes it in the ordering.**
- **For *j* from *n* down to 2, apply** REMOVE-INCONSISTENT-VALUES(Parent(X_j),X_j)
- **For *j* from 1 to *n* assign X_j consistently with Parent(X_j)**

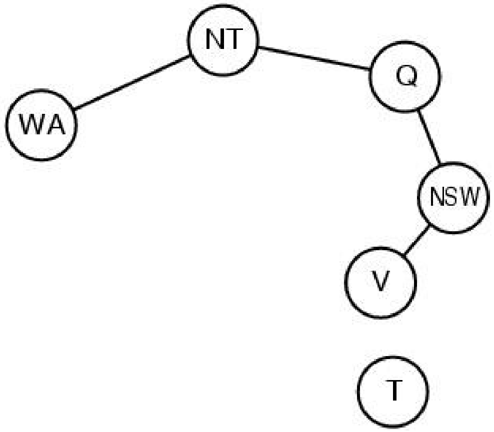
Nearly tree-structured CSPs



- ***Can more general constraint graphs be reduced to trees?***
- Two approaches:
  - ✓ **Remove certain nodes**
  - ✓ **Collapse certain nodes**

Nearly tree-structured CSPs



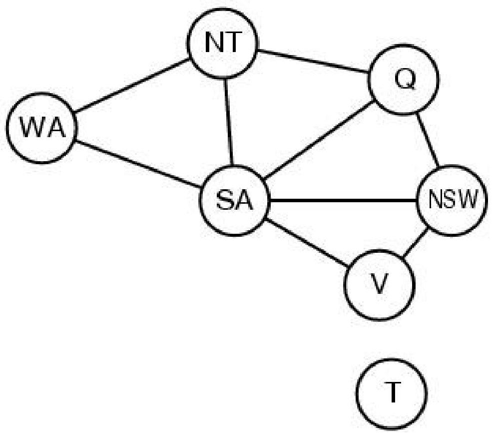


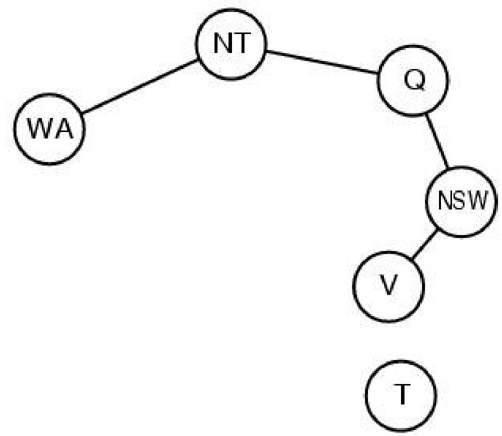
➢**Idea:** **assign values to some variables so that the remaining variables form a tree.**

Assume that we assign ***{SA=x}*** ← ***cycle cutset***

- ➢**And remove any values from the other variables that are inconsistent.**
- ➢**The selected value for SA could be the wrong one so we have to try all of them**

Nearly tree-structured CSPs



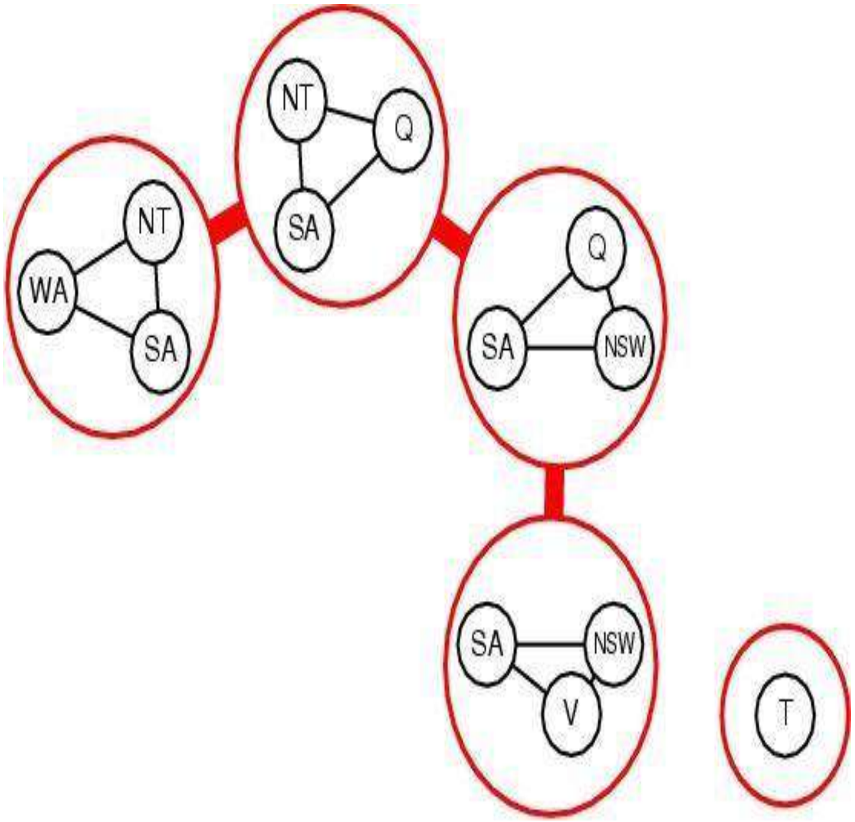


- This approach is worthwhile if cycle cutset is small.
- Finding the smallest cycle cutset is NP-hard

Approximation algorithms exist

- This approach is called ***cutset conditioning*.**

Nearly tree-structured CSPs



- ➢Tree decomposition of the constraint graph in a set of connected subproblems.
- ➢Each subproblem is solved independently
- ➢Resulting solutions are combined.
- ➢**Necessary requirements:**
  - ➢**Every variable appears in atleast one of the subproblems.**
  - ➢**If two variables are connected in the original problem, they must appear together in atleast one subproblem.**
  - ➢**If a variable appears in two subproblems, it must appear in each node on the path.**

- Constraint satisfaction problems (CSPs) represent a state with a set of variable/value pairs and represent the conditions for a solution by a set of constraints on the variables. Many important real-world problems can be described as CSPs.
- A number of inference techniques use the constraints to infer which variable/value pairs are consistent and which are not. These include node, arc, path, and k-consistency.
- Backtracking search, a form of depth-first search, is commonly used for solving CSPs. Inference can be interwoven with search.
- The minimum-remaining-values and degree heuristics are domain-independent methods for deciding which variable to choose next in a backtracking search. The leastconstraining-value heuristic helps in deciding which value to try first for a given variable. Backtracking occurs when no legal assignment can be found for a variable. Conflict-directed backjumping backtracks directly to the source of the problem.
- Local search using the min-conflicts heuristic has also been applied to constraint satisfaction problems with great success.
- The complexity of solving a CSP is strongly related to the structure of its constraint graph. Tree-structured problems can be solved in linear time. Cutset conditioning can reduce a general CSP to a tree-structured one and is quite efficient if a small cutset can be found. Tree decomposition techniques transform the CSP into a tree of subproblems and are efficient if the tree width of the constraint graph is small.

Thank You

Preposition Logic
Forward & Backward Chaining
Probability Bayes Theorem

Forward Chaining and backward chaining in AI

In artificial intelligence, forward and backward chaining is one of the important topics, but before understanding forward and backward chaining lets first understand that from where these two terms came.

Inference engine:

The inference engine is the component of the intelligent system in artificial intelligence, which applies **logical rules to the knowledge base to infer new information from known facts.** The first inference engine was part of the expert system. Inference engine commonly proceeds in two modes, which are:

1.Forward chaining (Data driven approach)

2. Backward chaining (Goal driven approach)

Horn Clause and Definite clause:
Horn clause and definite clause are the forms of sentences, which enables knowledge base to use a more restricted and efficient inference algorithm.

Logical inference algorithms use forward and backward chaining approaches, which require KB in the form of the **first-order definite clause**.

Definite clause: A clause which is a disjunction of literals with **exactly one positive literal** is known as a definite clause or strict horn clause.

Horn clause: A clause which is a disjunction of literals with **at most one positive literal** is known as horn clause. Hence all the definite clauses are horn clauses.
Example: (¬ p V ¬ q V k). It has only one positive literal k.
It is equivalent to p ∧ q → k.

A. Forward Chaining:

Forward chaining is also known as a forward deduction or forward reasoning method when using an inference engine.

Forward chaining is a form of reasoning which start with atomic sentences in the knowledge base and applies inference rules (Modus Ponens) in the forward direction to extract more data until a goal is reached.

The Forward-chaining algorithm starts from known facts, triggers all rules whose premises are satisfied, and add their conclusion to the known facts. This process repeats until the problem is solved.

Properties of Forward-Chaining:

- It is a down-up approach, as it moves from bottom to top.
- It is a process of making a conclusion based on known facts or data, by starting from the initial state and reaches the goal state.
- Forward-chaining approach is also called as data-driven as we reach to the goal using available data.
- Forward -chaining approach is commonly used in the expert system, such as CLIPS, business, and production rule systems.

Example:

"As per the law, it is a crime for an American to sell weapons to hostile nations. Country A, an enemy of America, has some missiles, and all the missiles were sold to it by Robert, who is an American citizen.“

Prove that **"Robert is criminal."**

Facts Conversion into FOL:

It is a crime for an American to sell weapons to hostile nations.
(Let's say p, q, and r are variables)

American (p) ∧ weapon(q) ∧ sells (p, q, r) ∧ hostile(r) → Criminal(p) ...(1)

Country A has some missiles.
?p Owns(A, p) ∧ Missile(p).

It can be written in two definite clauses by using Existential Instantiation, introducing new Constant T1.
Owns(A, T1) ......(2)
Missile(T1) ....(3)

All of the missiles were sold to country A by Robert.

?p Missiles(p) ∧ Owns (A, p) → Sells (Robert, p, A) ......(4)

Missiles are weapons.

Missile(p) → Weapons (p) ....(5)

Enemy of America is known as hostile.

Enemy(p, America) →Hostile(p) ....(6)

Country A is an enemy of America.

Enemy (A, America) ....(7)

Robert is American

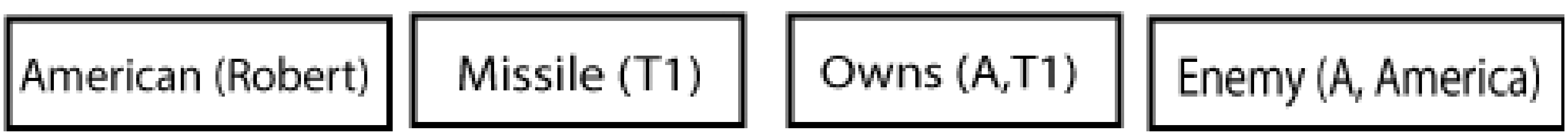
American(Robert). ....(8)

Forward chaining proof:

Step-1:
In the first step we will start with the known facts and will choose the sentences which do not have implications, such as:

American(Robert), Enemy(A, America), Owns(A, T1), and Missile(T1).

All these facts will be represented as below.



Step-2:

At the second step, we will see those facts which infer from available facts and with satisfied premises.

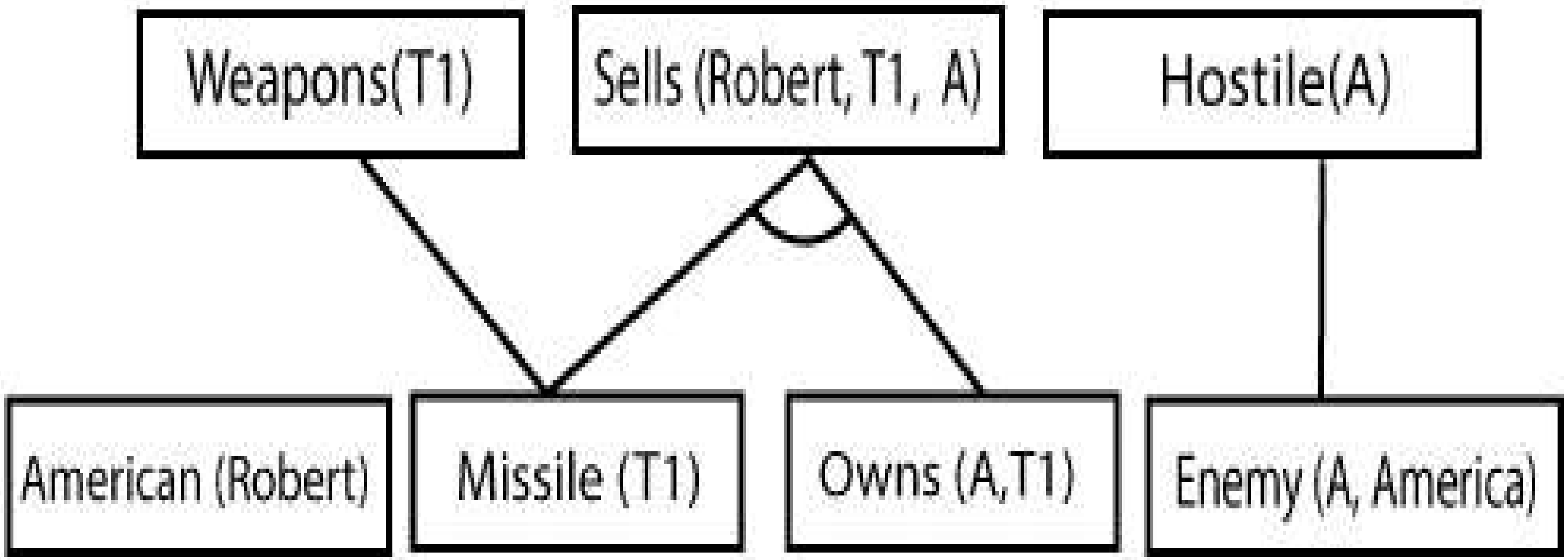
Rule-(1) does not satisfy premises, so it will not be added in the first iteration.

Rule-(2) and (3) are already added.

Rule-(4) satisfy with the substitution {p/T1},

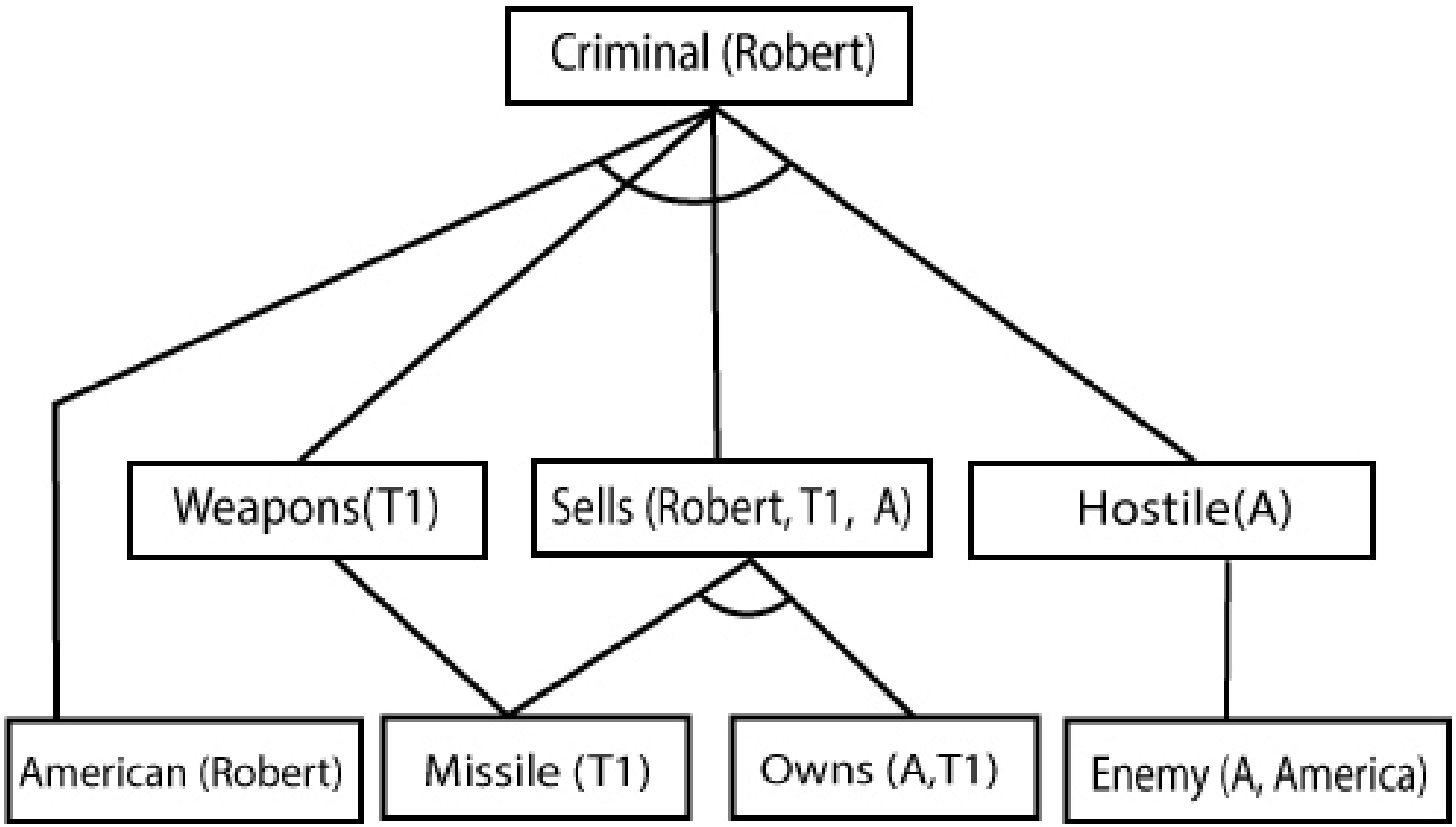
so Sells (Robert, T1, A) is added, which infers from the conjunction of Rule (2) and (3).

Rule-(6) is satisfied with the substitution(p/A), so Hostile(A) is added and which infers from Rule-(7).


Weapons(T1)
Sells (Robert, T1, A)
Hostile(A)
American (Robert)
Missile (T1)
Owns (A,T1)
Enemy (A, America)

Step-3:

At step-3, as we can check Rule-(1) is satisfied with the substitution **{p/Robert, q/T1, r/A}, so we can add Criminal(Robert)** which infers all the available facts. And hence we reached our goal statement.



B. Backward Chaining:

Backward-chaining is also known as a backward deduction or backward reasoning method when using an inference engine. A backward chaining algorithm is a form of reasoning, which starts with the goal and works backward, chaining through rules to find known facts that support the goal.

Properties of backward chaining:

- It is known as a top-down approach.
- Backward-chaining is based on modus ponens inference rule.
- In backward chaining, the goal is broken into sub-goal or sub-goals to prove the facts true.
- It is called a goal-driven approach, as a list of goals decides which rules are selected and used.
- Backward -chaining algorithm is used in game theory, automated theorem proving tools, inference engines, proof assistants, and various AI applications.
- The backward-chaining method mostly used a **depth-first search** strategy for proof.

Example:

In backward-chaining, we will use the same above example, and will rewrite all the rules.

American (p) ∧ weapon(q) ∧ sells (p, q, r) ∧ hostile(r) → Criminal(p) ...(1)

Owns(A, T1) ....(2)
Missile(T1)

?p Missiles(p) ∧ Owns (A, p) → Sells (Robert, p, A) ......(4)
Missile(p) → Weapons (p) ....(5)
Enemy(p, America) →Hostile(p) ....(6)
Enemy (A, America) ....(7)
American(Robert). ....(8)

Backward-Chaining proof:

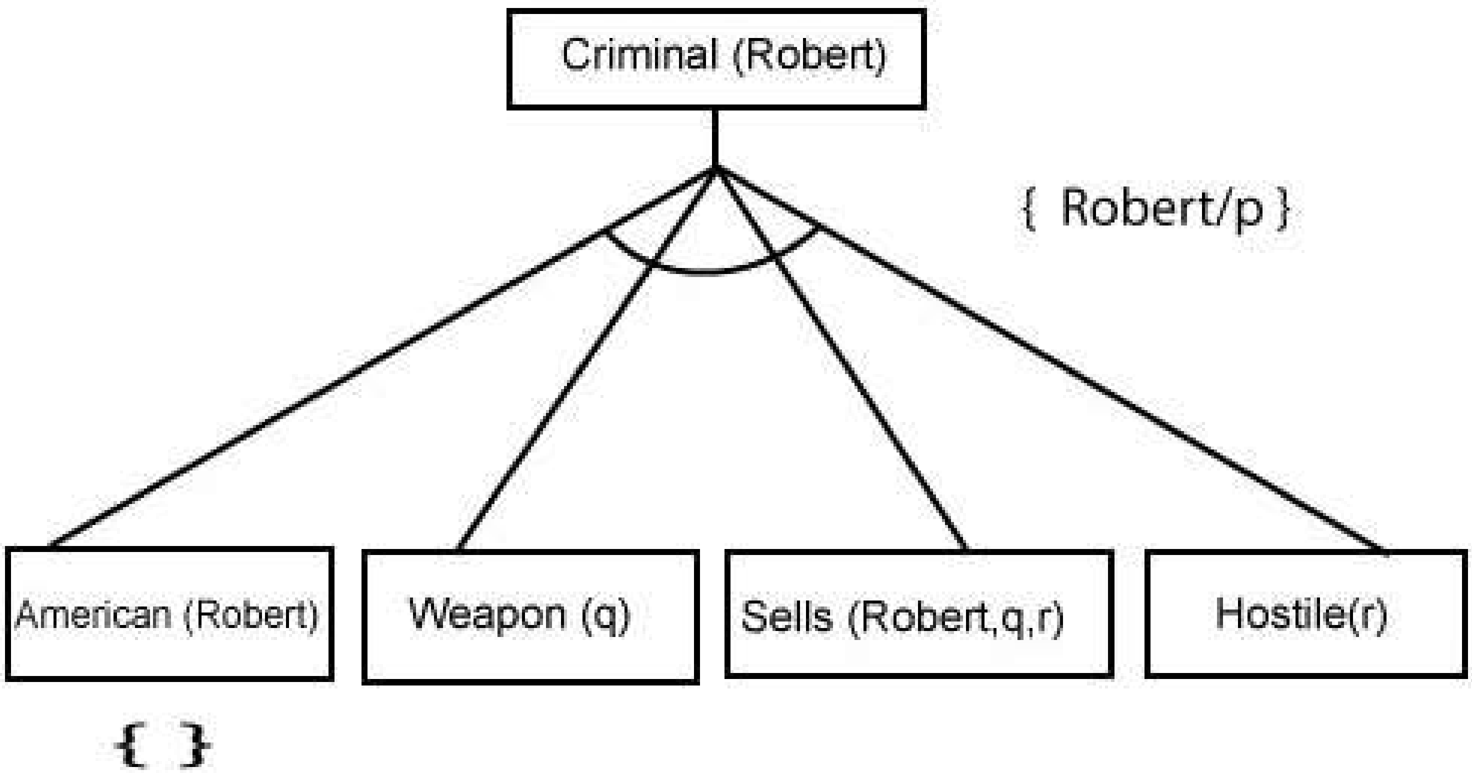
In Backward chaining, we will start with our goal predicate, which is **Criminal(Robert)**, and then infer further rules.

Step-1:
At the first step, we will take the goal fact. And from the goal fact, we will infer other facts, and at last, we will prove those facts true. So our goal fact is "Robert is Criminal," so following is the predicate of it.

Step-2:

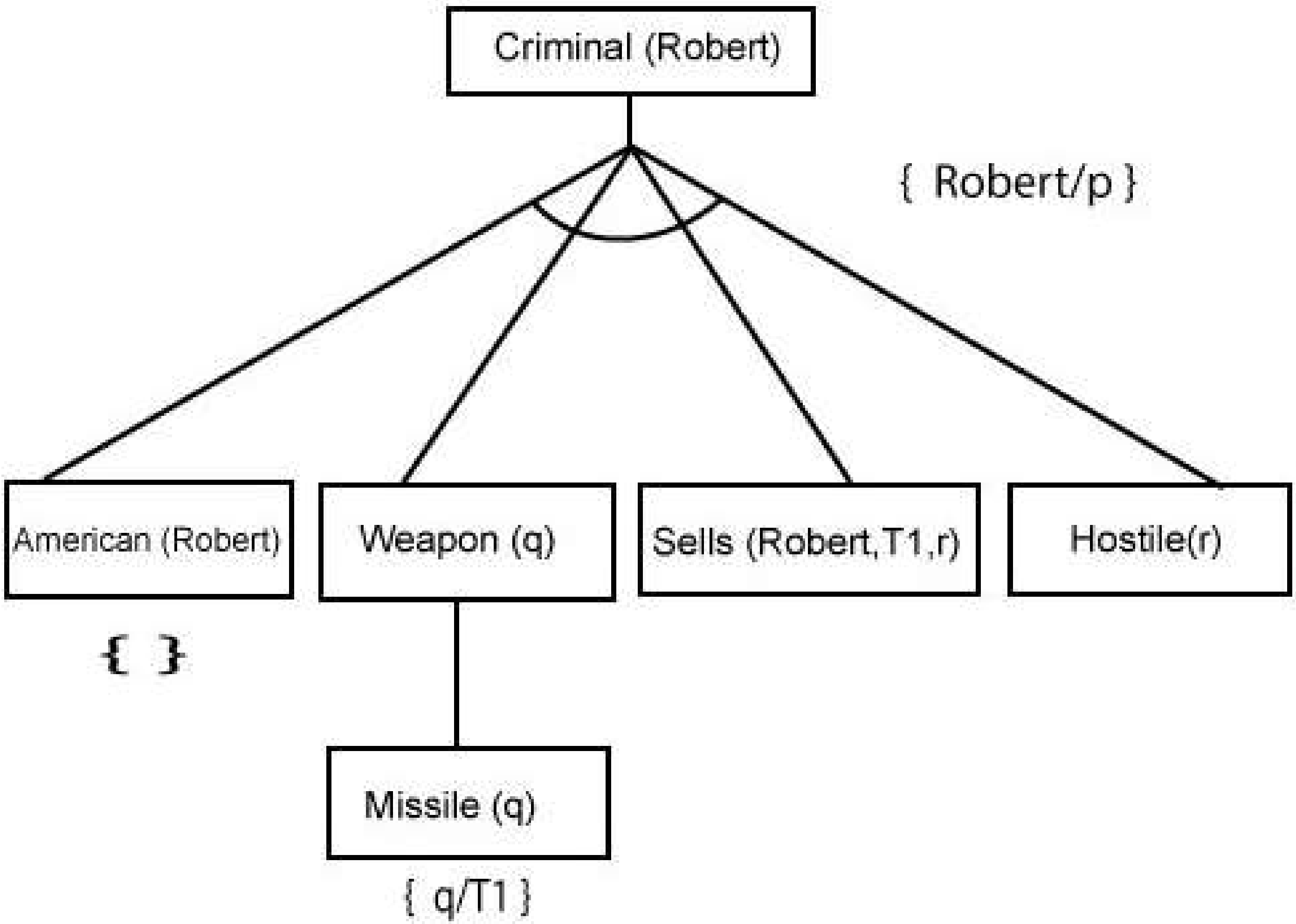
At the second step, we will infer other facts form goal fact which satisfies the rules. So as we can see in Rule-1, the goal predicate Criminal (Robert) is present with substitution {Robert/P}. So we will add all the conjunctive facts below the first level and will replace p with Robert.

Here we can see American (Robert) is a fact, so it is proved here



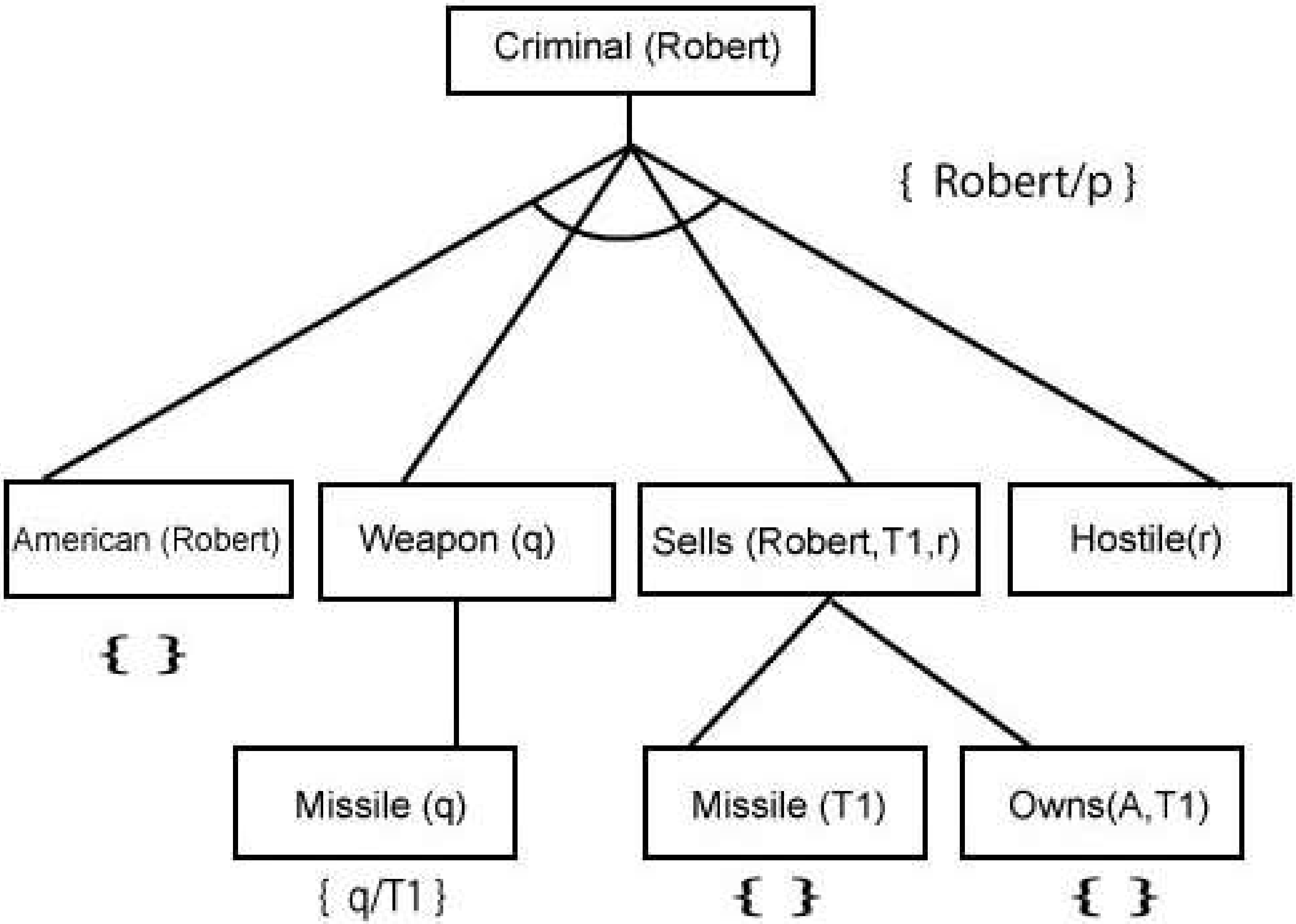
Step-3:

At step-3, we will extract further fact Missile(q) which infer from Weapon(q), as it satisfies Rule-(5). Weapon (q) is also true with the substitution of a constant T1 at q.

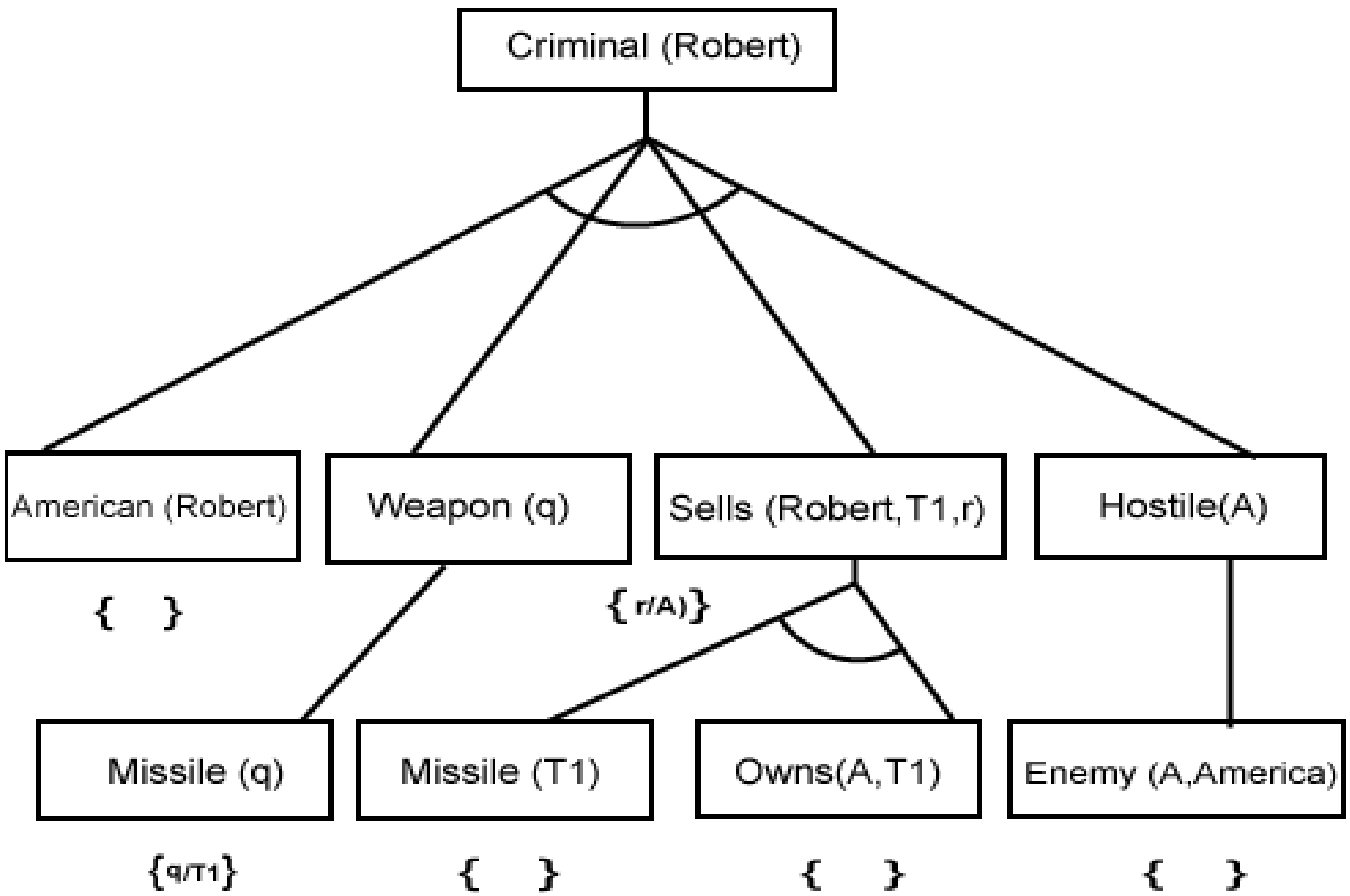


Step-4:

At step-4, we can infer facts Missile(T1) and Owns(A, T1) form Sells(Robert, T1, r) which satisfies the **Rule- 4**, with the substitution of A in place of r. So these two statements are proved here.



Step-5: we can infer the fact **Enemy(A, America)** from **Hostile(A)** which satisfies Rule- 6. And hence all the statements are proved true using backward chaining



Advantages

- It can be used to draw multiple conclusions.
- It provides a good basis for arriving at conclusions.
- It’s more flexible than backward chaining because it does not have a limitation on the data derived from it.

Disadvantages

- The process of forward chaining may be time-consuming. It may take a lot of time to eliminate and synchronize available data.
- Unlike backward chaining, the explanation of facts or observations for this type of chaining is not very clear. The former uses a goal-driven method that arrives at conclusions efficiently.

Advantages

- The result is already known, which makes it easy to deduce inferences.
- It’s a quicker method of reasoning than forward chaining because the endpoint is available.
- In this type of chaining, correct solutions can be derived effectively if pre-determined rules are met by the inference engine.

Disadvantages

- The process of reasoning can only start if the endpoint is known.
- It doesn’t deduce multiple solutions or answers.
- It only derives data that is needed, which makes it less flexible than forward chaining.

Probabilistic reasoning in Artificial intelligence

Uncertainty:

Till now, we have learned knowledge representation using first-order logic and propositional logic with certainty, which means we were sure about the predicates.

With this knowledge representation, we might write A→B, which means if A is true then B is true,

Consider a situation where we are not sure about whether A is true or not then we cannot express this statement, this situation is called uncertainty.
So to represent uncertain knowledge, where we are not sure about the predicates, we need **uncertain reasoning or probabilistic reasoning.**

Causes of uncertainty:

Following are some leading causes of uncertainty to occur in the real world.

- Information occurred from unreliable sources.
- Experimental Errors
- Equipment fault
- Temperature variation
- Climate change.

Probabilistic reasoning:

- Probabilistic reasoning is a way of knowledge representation where we apply the concept of probability to indicate the uncertainty in knowledge.
- In probabilistic reasoning, we combine probability theory with logic to handle the uncertainty.
- We use probability in probabilistic reasoning because it provides a way to handle the uncertainty that is the result of someone's laziness and ignorance.
- In the real world, there are lots of scenarios, where the certainty of something is not confirmed, such as "It will rain today," "behavior of someone for some situations," "A match between two teams or two players." These are probable sentences for which we can assume that it will happen but not sure about it, so here we use probabilistic reasoning.

Need of probabilistic reasoning in AI:

- When there are unpredictable outcomes.
- When specifications or possibilities of predicates becomes too large to handle.
- When an unknown error occurs during an experiment.

In probabilistic reasoning, there are two ways to solve problems with uncertain knowledge:

- **Bayes' rule**
- **Bayesian Statistics**

Probability:

Probability can be defined as a chance that an uncertain event will occur. It is the numerical measure of the likelihood that an event will occur. The value of probability always remains between 0 and 1 that represent ideal uncertainties.

$0 \le P(A) \le 1$, where $P(A)$ is the probability of an event A.

$P(A) = 0$, indicates total uncertainty in an event A.

$P(A) = 1$, indicates total certainty in an event A.

We can find the probability of an uncertain event by using the below formula.

$$\textbf{Probability of occurrence} = \frac{\text{Number of desired outcomes}}{\text{Total number of outcomes}}$$

- $P(\neg A)$ = probability of a not happening event.
- $P(\neg A) + P(A) = 1$.

Event: Each possible outcome of a variable is called an event.
Sample space: The collection of all possible events is called sample space.

Random variables: Random variables are used to represent the events and objects in the real world.

Prior probability: The prior probability of an event is probability computed before observing new information.

Posterior Probability: The probability that is calculated after all evidence or information has taken into account. It is a combination of prior probability and new information.

Conditional probability:

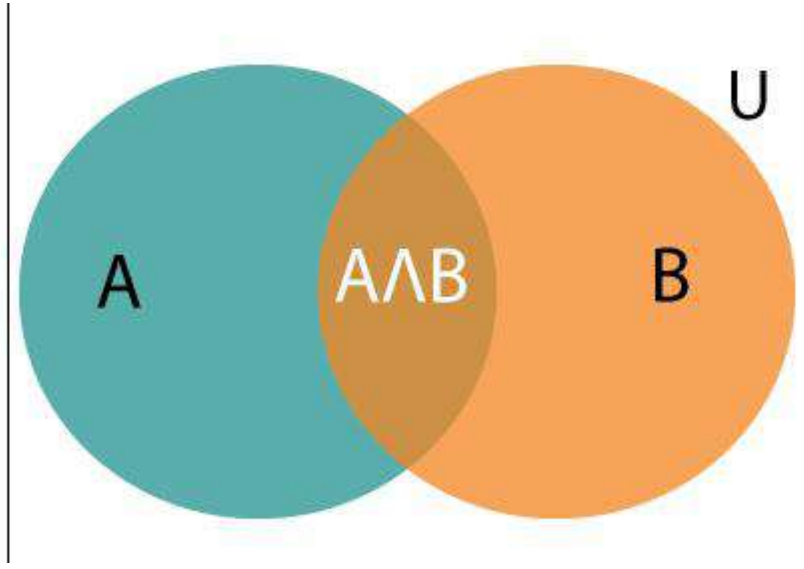
- Conditional probability is a probability of occurring an event when another event has already happened.
- Let's suppose, we want to calculate the event A when event B has already occurred, "the probability of A under the conditions of B", it can be written as: P(A/B)

Where P($A \wedge B$)= Joint probability of a and B
P(B)= Marginal probability of B.

If the probability of A is given and we need to find the probability of B, then it will be given as:

If the probability of A is given and we need to find the probability of B, then it will be given as:

It can be explained by using the below Venn diagram, where B is occurred event, so sample space will be reduced to set B, and now we can only calculate event A when event B is already occurred by dividing the probability of **P(A∧*B*) by P(B)**.



Bayes' theorem in Artificial intelligence

Bayes' theorem:

Bayes' theorem is also known as **Bayes' rule, Bayes' law**, or **Bayesian reasoning**, which determines the probability of an event with uncertain knowledge.

In probability theory, it relates the conditional probability and marginal probabilities of two random events.

Bayes' theorem was named after the British mathematician **Thomas Bayes**. The **Bayesian inference** is an application of Bayes' theorem, which is fundamental to Bayesian statistics.

It is a way to calculate the value of P(B|A) with the knowledge of P(A|B).

Bayes' theorem allows updating the probability prediction of an event by observing new information of the real world.

Example: If cancer corresponds to one's age then by using Bayes' theorem, we can determine the probability of cancer more accurately with the help of age.
Bayes' theorem can be derived using product rule and conditional probability of event A with known event B:
As from product rule we can write:

$$P(A \wedge B) = P(A|B)\,P(B) \text{ or}$$

Similarly, the probability of event B with known event A:

$$P(A \wedge B) = P(B|A)\,P(A)$$

Equating right hand side of both the equations, we will get:

$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)} \quad \text{....(a)}$$

The above equation (a) is called as **Bayes' rule** or **Bayes' theorem**. This equation is basic of most modern AI systems for **probabilistic inference**.

$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)}$$

•P(A|B) is known as **posterior**, which we need to calculate, and it will be read as Probability of hypothesis A when we have occurred an evidence B.

•P(B|A) is called the likelihood, in which we consider that hypothesis is true, then we calculate the probability of evidence.

•P(A) is called the **prior probability**, probability of hypothesis before considering the evidence

•P(B) is called **marginal probability**, pure probability of an evidence.

In the equation (a), in general, we can write P (B) = P(A)*P(B|Ai), hence the Bayes' rule can be written as:

$$P(A_i|B) = \frac{P(A_i)*P(B|A_i)}{\sum_{i=1}^{k} P(A_i)*P(B|A_i)}$$

Where A_1, A_2, A_3,........, A_n is a set of mutually exclusive and exhaustive events.

Applying Bayes' rule:

Bayes' rule allows us to compute the single term P(B|A) in terms of P(A|B), P(*B*), and P(A).

This is very useful in cases where we have a good probability of these three terms and want to determine the fourth one.

Suppose we want to perceive the effect of some unknown cause, and want to compute that cause, then the Bayes' rule becomes:

Example-1:

Question: what is the probability that a patient has diseases meningitis with a stiff neck?

Given Data:

A doctor is aware that disease meningitis causes a patient to have a stiff neck, and it occurs 80% of the time. He is also aware of some more facts, which are given as follows:

The Known probability that a patient has meningitis disease is 1/30,000.

The Known probability that a patient has a stiff neck is 2%.

Let a be the proposition that patient has stiff neck and b be the proposition that patient has meningitis. , so we can calculate the following as:

P(a|b) = 0.8

P(b) = 1/30000

P(a)= .02

$$\mathbf{P(b|a)} = \frac{P(a|b)P(b)}{P(a)} = \frac{0.8*(\frac{1}{30000})}{0.02} = 0.001333333.$$

Hence, we can assume that 1 patient out of 750 patients has meningitis disease with a stiff neck.

Unit 3

Knowledge Representation Techniques

Lecture Module - 15

Knowledge Representation

- *Knowledge representation (KR)* is an important issue in both cognitive science and artificial intelligence.
  - In cognitive science, it is concerned with the way people store and process information and
  - In artificial intelligence (AI), main focus is to store knowledge so that programs can process it and achieve human intelligence.
- There are different ways of representing knowledge e.g.
  - predicate logic,
  - semantic networks,
  - extended semantic net,
  - frames,
  - conceptual dependency etc.
- In predicate logic, knowledge is represented in the form of rules and facts as is done in Prolog.

Semantic Network

- Formalism for representing information about objects, people, concepts and specific relationship between them.
- The syntax of semantic net is simple. It is a network of labeled nodes and links.
  - It's a directed graph with nodes corresponding to concepts, facts, objects etc. and
  - arcs showing relation or association between two concepts.
- The commonly used links in semantic net are of the following types.
  - **isa** → subclass of entity (e.g., child hospital is subclass of hospital)
  - **inst** → particular instance of a class (e.g., India is an instance of country)
  - **prop** → property link (e.g., property of dog is 'bark)

Representation of Knowledge in Sem Net

"Every human, animal and bird is living thing who breathe and eat. All birds can fly. All man and woman are humans who have two legs. Cat is an animal and has a fur. All animals have skin and can move. Giraffe is an animal who is tall and has long legs. Parrot is a bird and is green in color".

Representation in Predicate Logic

- Every human, animal and bird is living thing who breathe and eat.

  $\forall$X [human(X) $\rightarrow$ living(X)]

  $\forall$X [animal(X) $\rightarrow$ living(X)]

  $\forall$X [bird(X) $\rightarrow$ living(X)]

- All birds are animal and can fly.

  $\forall$X [bird(X) $\wedge$ canfly(X)]

- Every man and woman are humans who have two legs.

  $\forall$X [man(X) $\wedge$ haslegs(X)]

  $\forall$X [woman(X) $\wedge$ haslegs(X)]

  $\forall$X [human(X) $\wedge$ has(X, legs)]

- Cat is an animal and has a fur.

  animal(cat) $\wedge$ has(cat, fur)

- All animals have skin and can move.

  $\forall$X [animal(X) $\rightarrow$ has(X, skin) $\wedge$ canmove(X)]

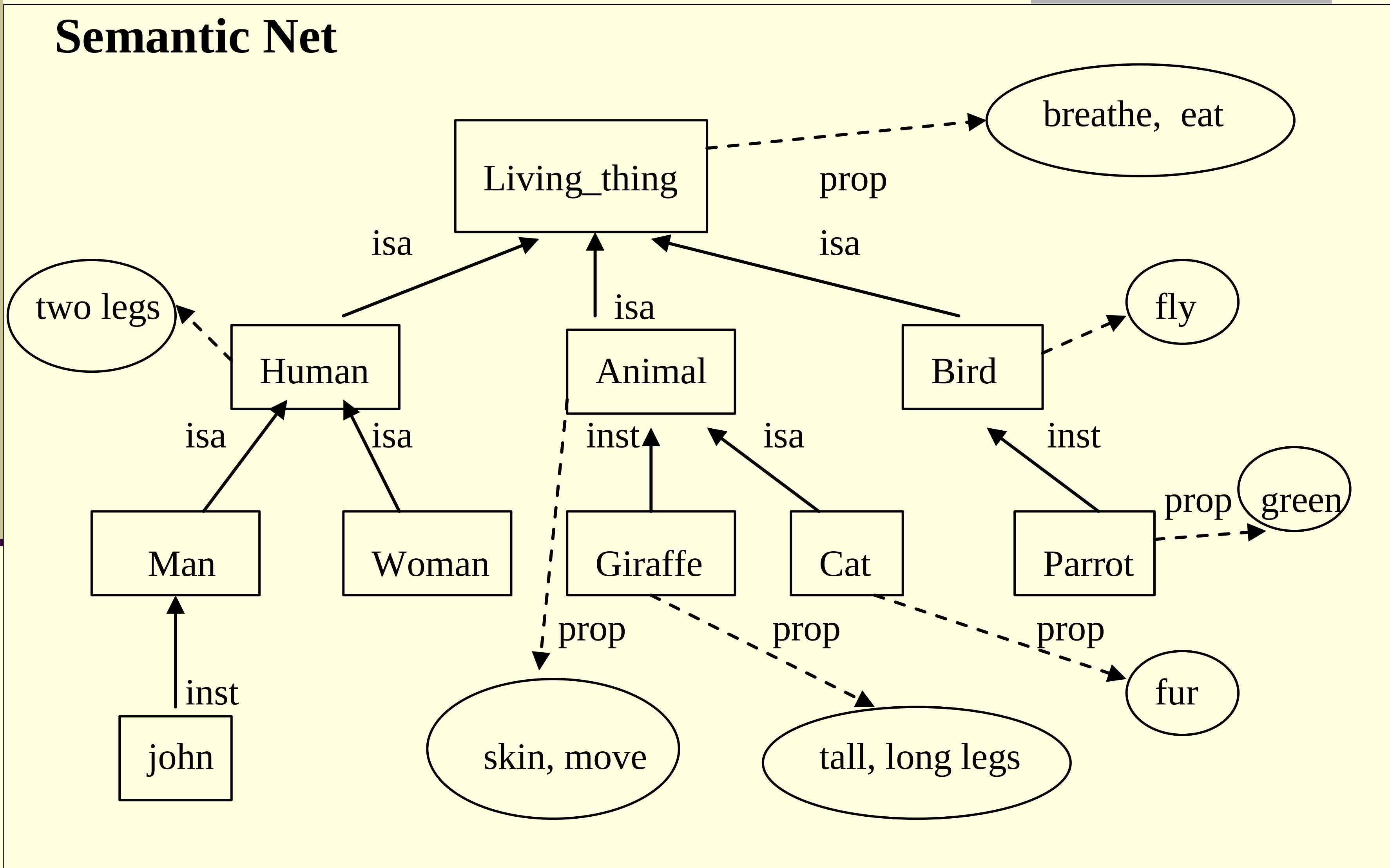
- Giraffe is an animal who is tall and has long legs.

  animal(giraffe) $\wedge$ has(giraffe, long_legs) $\wedge$ is(giraffe, tall)

- Parrot is a bird and is green in color.

  bird(parrot) $\wedge$ has(parrot, green_colour)

Representation in Semantic Net



Inheritance

- Inheritance mechanism allows knowledge to be stored at the highest possible level of abstraction which reduces the size of knowledge base.
  - It facilitates inferencing of information associated with semantic nets.
  - It is a natural tool for representing taxonomically structured information and ensures that all the members and sub-concepts of a concept share common properties.
  - It also helps us to maintain the consistency of the knowledge base by adding new concepts and members of existing ones.
- Properties attached to a particular object (class) are to be inherited by all subclasses and members of that class.

Property Inheritance Algorithm

Input: Object, and property to be found from Semantic Net;

Output: Yes, if the object has the desired property else return false;

Procedure:

- Find an object in the semantic net; Found = false;
- While {(object ≠ root) OR Found } DO

{ If there is a a property attribute attached with an object then

{ Found = true; Report ‘Yes’} else

object=inst(object, class) OR isa(object, class)

};

- If Found = False then report ‘No’; Stop

Coding of Semantic Net in Prolog

Isa facts	**Instance facts**	**Property facts**
isa(living_thing, nil).	inst(john, man).	prop(breathe, living_thing).
isa(human, living_thing).	inst(giraffe, animal).	prop(eat, living_thing).
isa(animals, living_thing).	inst(parrot, bird)	prop(two_legs, human).
isa(birds, living_thing).		prop(skin, animal).
isa(man, human).		prop(move, animal).
isa(woman, human).		prop(fur, bird).
isa(cat, animal).		prop(tall, giraffe).
		prop(long_legs, giraffe).
		prop(tall, animal).
		prop(green, parrot).

Inheritance Rules in Prolog

Instance rules:

```
instance(X, Y)      :-    inst(X, Y).
instance (X, Y)     :-    inst(X, Z), subclass(Z,Y).
```

Subclass rules:

```
subclass(X, Y)      :-    isa(X, Y).
subclass(X, Y)      :-    isa(X, Z), subclass(Z, Y) .
```

Property rules:

```
property(X, Y)      :-    prop(X, Y).
property(X, Y)      :-    instance(Y,Z), property(X, Z).
property(X, Y)      :-    subclass(Y, Z), property(X, Z).
```

Queries

- Is john human?
- Is parrot a living thing?
- Is giraffe an aimal?
- Is woman subclassof living thing
- Does parrot fly?
- Does john breathe?
- has parrot fur?
- Does cat fly?

?- instance(john, humans). Y

?- instance (parrot, living_thing). Y

?- instance (giraffe, animal). Y

?- subclass(woman, living_things). Y

?- property(fly, parrot). Y

?- property (john, breathe). Y

?- property(fur, parrot). N

?- property(fly, cat). N

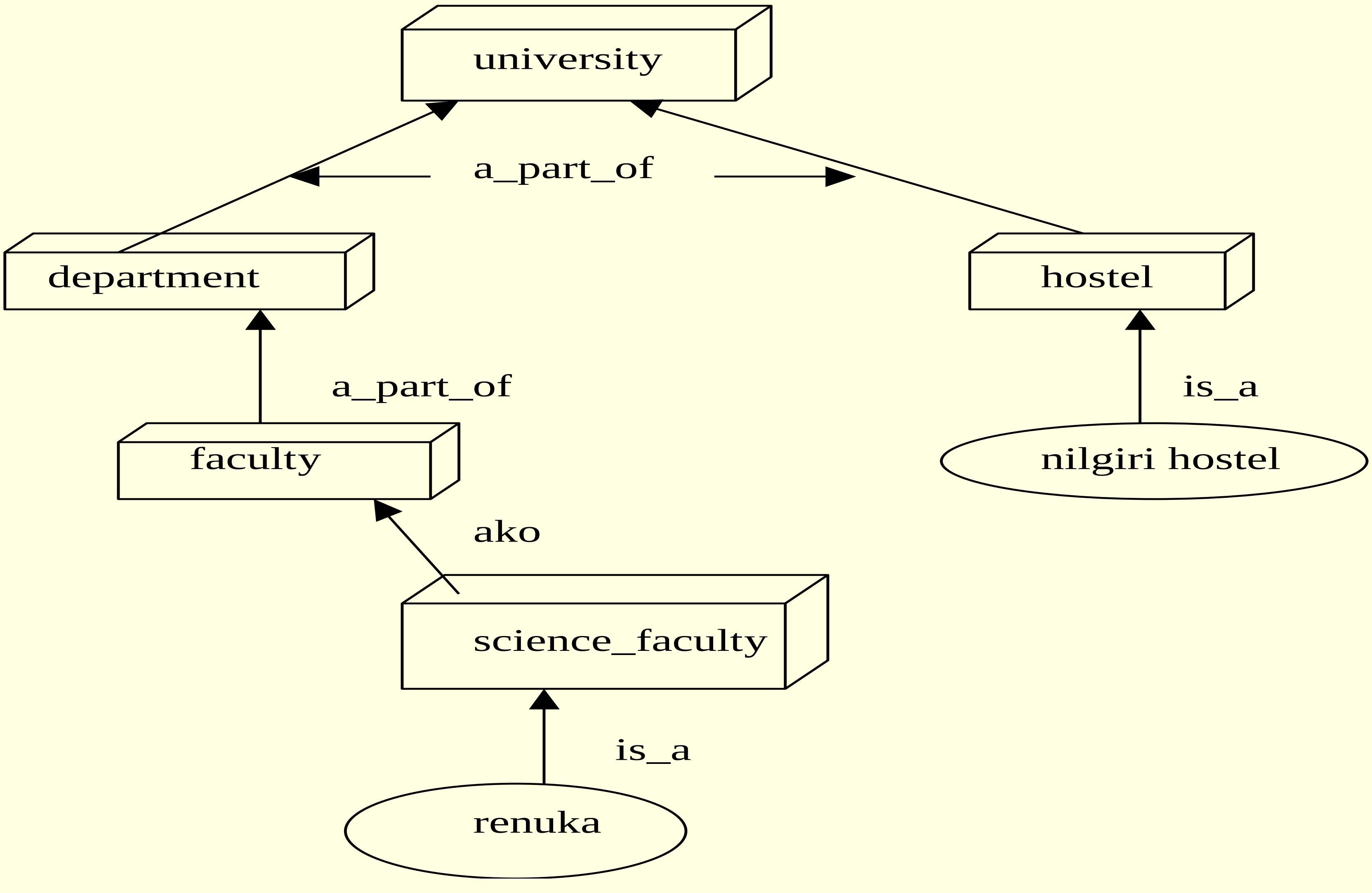
Knowledge Representation using Frames

- Frames are more structured form of packaging knowledge,
  - used for representing objects, concepts etc.
- Frames are organized into hierarchies or network of frames.
- Lower level frames can inherit information from upper level frames in network.
- Nodes are connected using links viz.,
  - **ako / subc** (links two class frames, one of which is subclass of other e.g., science_faculty class is **ako** of faculty class),
  - **is_a / inst** (connects a particular instance of a class frame e.g., Renuka **is_a** science_faculty)
  - **a_part_of** (connects two class frames one of which is contained in other e.g., faculty class **is_part_of** department class).
  - Property link of semantic net is replaced by SLOT fields.

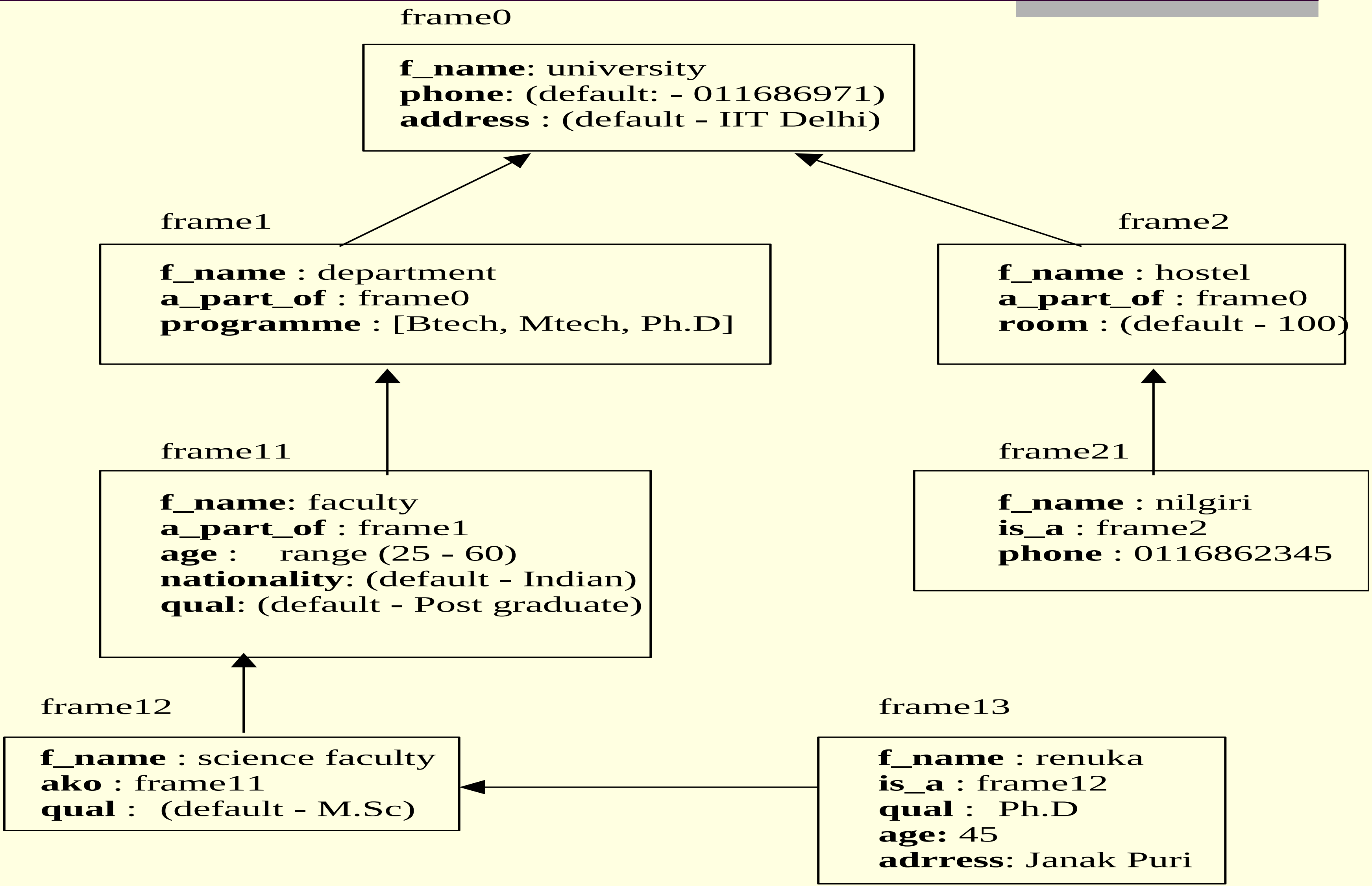
Cont…

- A frame may have any number of slots needed for describing object. e.g.,
  - faculty frame may have name, age, address, qualification etc as slot names.
- Each frame includes two basic elements : slots and facets.
  - Each slot may contain one or more **facets** (called fillers) which may take many forms such as:
    - **value** (value of the slot),
    - **default** (default value of the slot),
    - **range** (indicates the range of integer or enumerated values, a slot can have),
    - **demons** (procedural attachments such as if_needed, if_deleted, if_added etc.) and
    - **other** (may contain rules, other frames, semantic net or any type of other information).

Frame Network - Example



Detailed Representation of Frame Network



Description of Frames

- Each frame represents either a class or an instance.
- Class frame represents a general concept whereas instance frame represents a specific occurrence of the class instance.
- Class frame generally have default values which can be redefined at lower levels.
- If class frame has actual value facet then decedent frames can not modify that value.
- Value remains unchanged for subclasses and instances.

Inheritance in Frames

- Suppose we want to know nationality or phone of an instance-frame frame13 of renuka.
- These informations are not given in this frame.
- Search will start from frame13 in upward direction till we get our answer or have reached root frame.
- The frames can be easily represented in prolog by choosing predicate name as frame with two arguments.
- First argument is the name of the frame and second argument is a list of slot - facet pair.

Coding of frames in Prolog

frame(university, [phone (default, 011686971),
address (default, IIT Delhi)]).

frame(deaprtment, [a_part_of (university),
programme ([Btech, Mtech, Ph.d]))]).

frame(hostel, [a_part_of (university), room(default, 100)]).

frame(faculty, [a_part_of (department), age(range,25,60),
nationality(default, indian), qual(default, postgraduate)]).

frame(nilgiri, [is_a (hostel), phone(011686234)]).

frame(science_faculty, [ako (faculty),qual(default, M.Sc.)]).

frame(renuka, [is_a (science_faculty), qual(Ph.D.),
age(45), address(janakpuri)]).

Inheritance Program in Prolog

```
find(X, Y) :- frame(X, Z), search(Z, Y), !.
find(X, Y) :- frame(X, [is_a(Z),_]), find(Z, Y), !.
find(X, Y) :- frame(X, [ako(Z), _]), find(Z, Y), !.
find(X, Y) :- frame(X, [a_part_of(Z), _]), find(Z, Y).
```

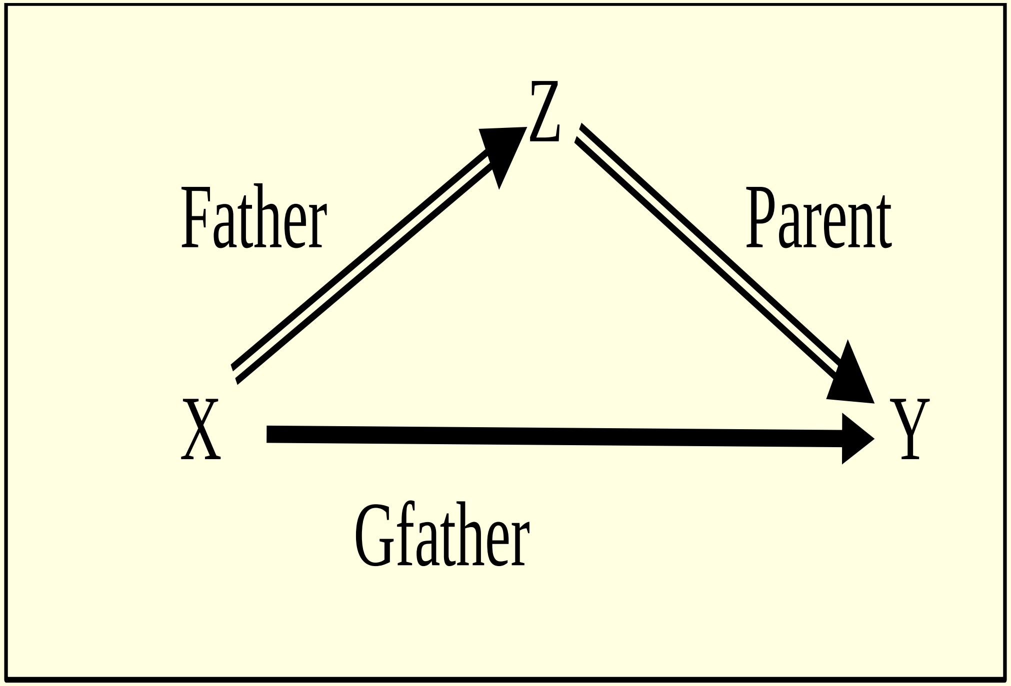
- Predicate **search** will basically retrieve the list of slots-facet pair and will try to match Y for slot.
- If match is found then its facet value is retrieved otherwise process is continued till we reach to root frame

Extended Semantic Network

- In conventional Sem Net, clausal form of logic can not be expressed.
- Extended Semantic Network (ESNet) combines the advantages of both logic and semantic network.
- In the ESNet, terms are represented by nodes similar to Sem Net.
- Binary predicate symbols in clausal logic are represented by labels on arcs of ESNet.
  - An *atom* of the form “Love(john, mary)” is an arc labeled as ‘Love’ with its two end nodes representing ‘john’ and ‘mary’.
- *Conclusions* and *conditions* in clausal form are represented by different kinds of arcs.
  - Conditions are drawn with two lines $\Longleftarrow$ and conclusions are drawn with one heavy line $\boldsymbol{\longleftarrow}$.

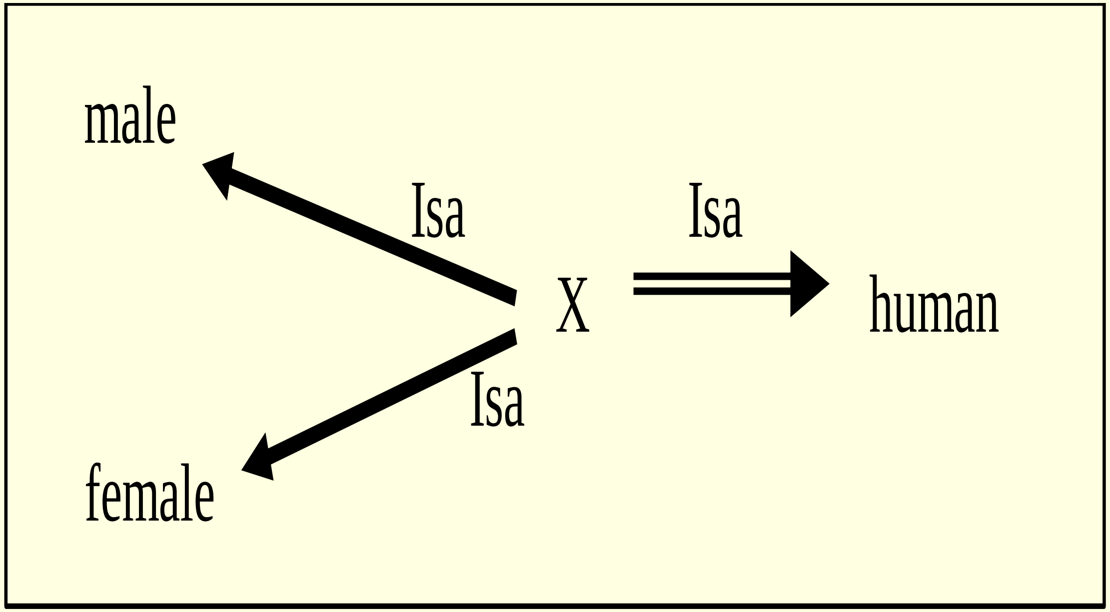
Examples

- Represent ‘grandfather’ definition
  Gfather(X, Y) ← Father(X, Z), Parent(Z, Y) in ESNet.



Cont…Example

- Represent clausal rule **“Male(X), Female(X) ← Human(X)”** using binary representation as “Isa(X, male), Isa(X, female) ← Isa(X, human)” and subsequently in ESNet as follows:

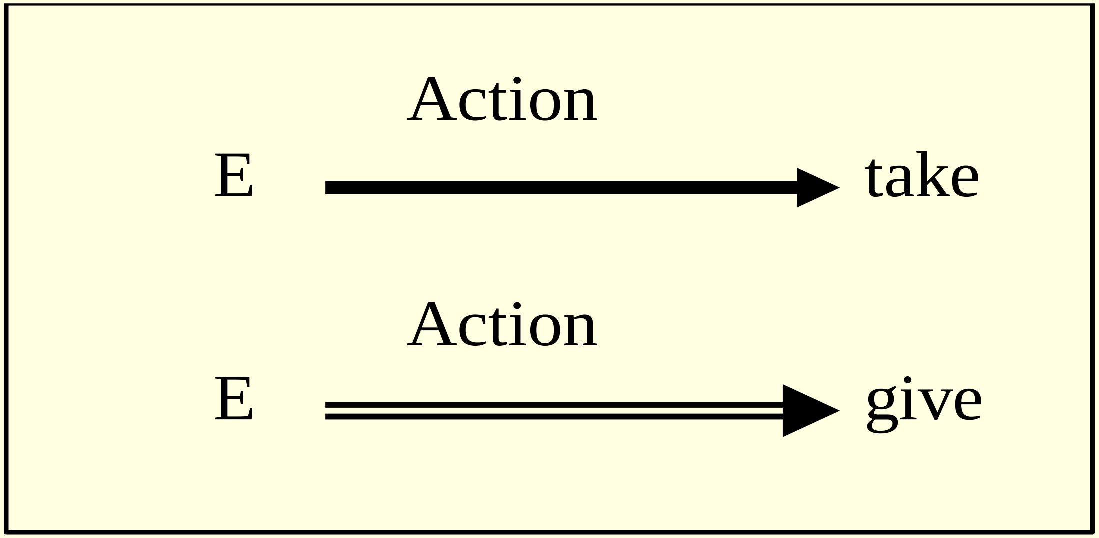


Inference Rules in ESNet

- Inference rules are embedded in the representation itself.
- The inference that "for every action of giving, there is an action of taking" in clausal logic written as

  "Action(E, take) ← Action(E, give)".

ESNet

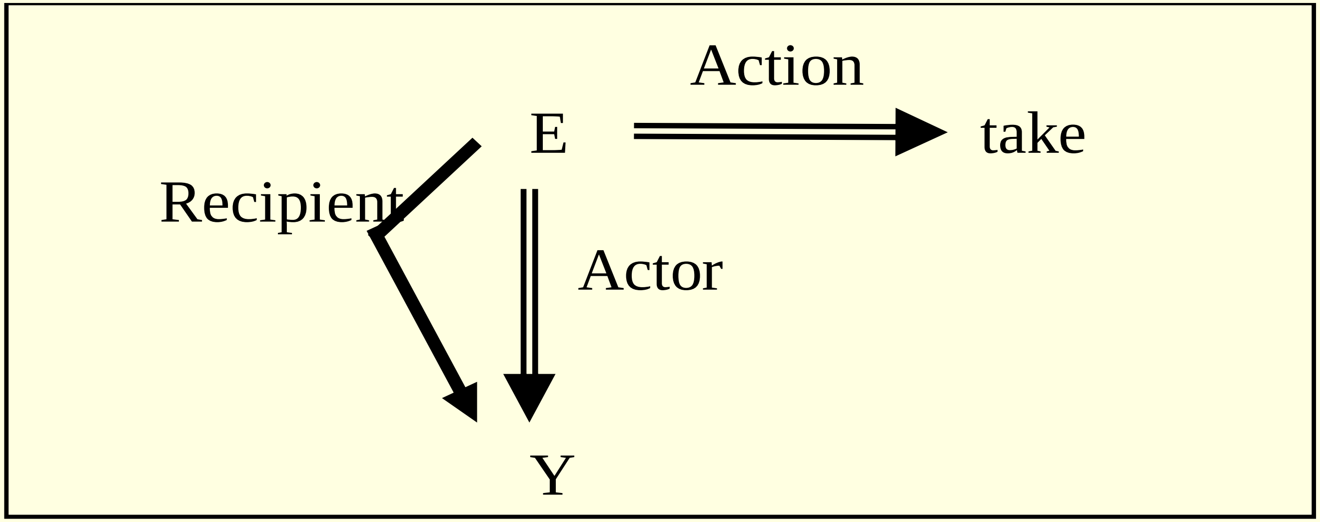


Cont…

- The inference rule such as "an actor of taking action is also the recipient of the action" can be easily represented in clausal logic as:
  - Here E is a variable representing an event where an action of taking is happening).

Recipient(E, Y) ← Acton(E, take), Actor (E, Y)

ESNet



Example

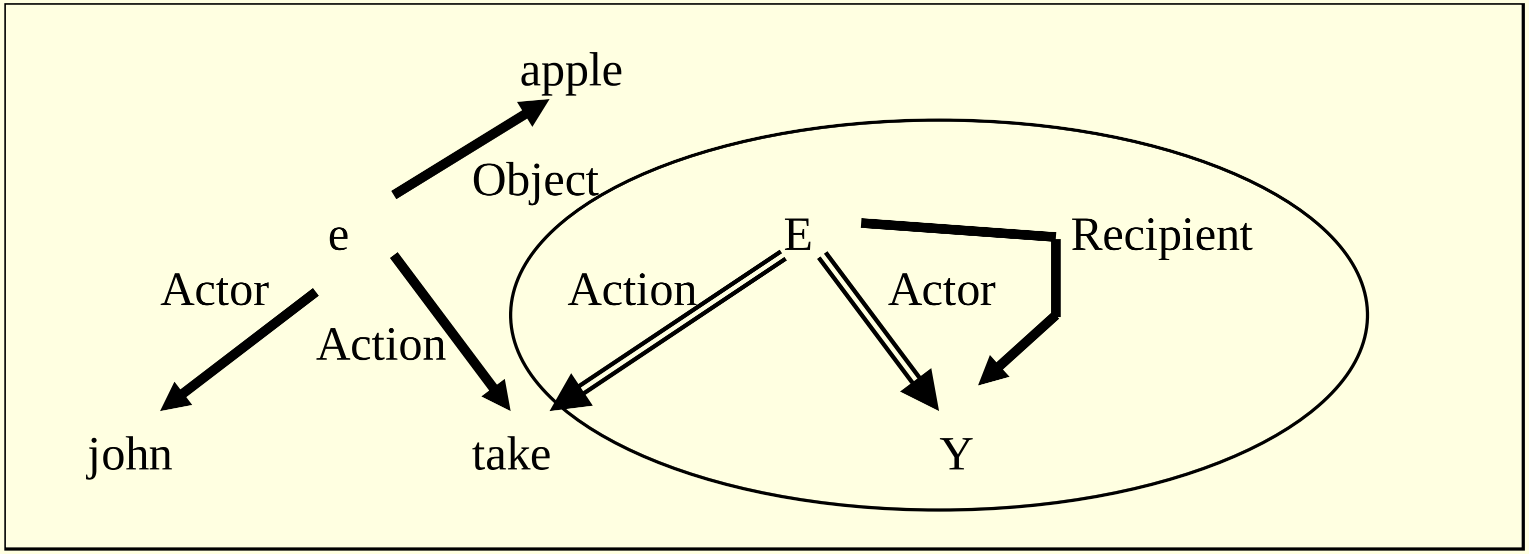
- Represent the following clauses of Logic in ESNet.

Recipient(E, Y) ← Acton(E, take), Actor (E, Y)

Object (e, apple).

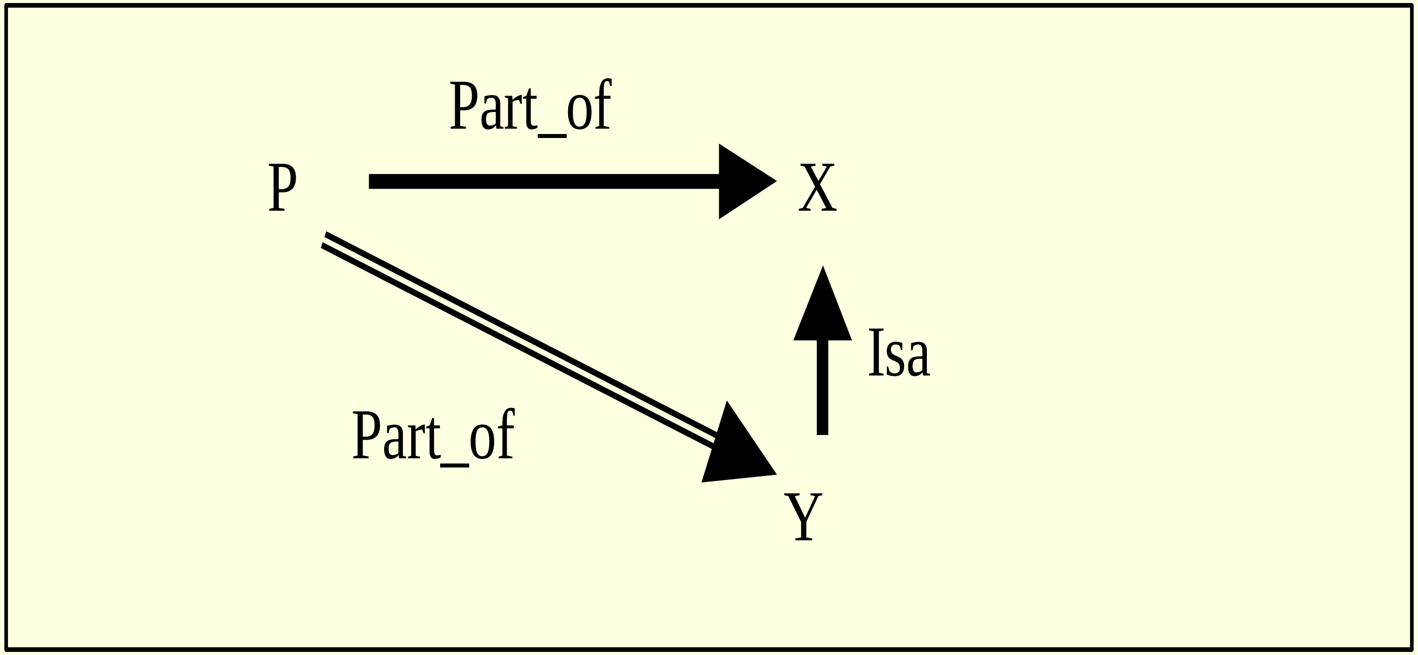
Action(e, take).

Actor (e, john) .



Contradiction

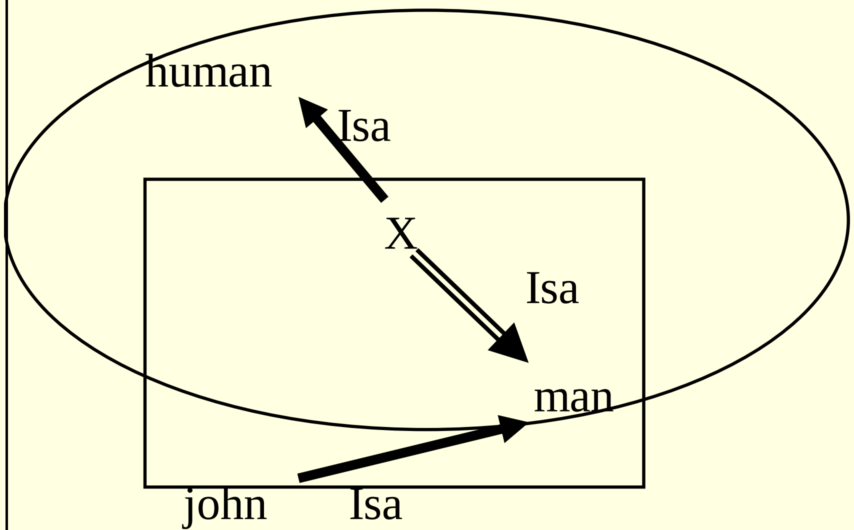
- The contradiction in the ESNet arises if we have the following situation.

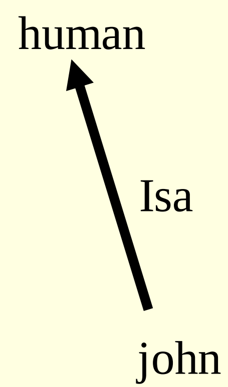


Deduction in ESNet

- Both of the following inference mechanisms are available in ESNet.
  - Forward reasoning inference (uses bottom up approach)
    - ***Bottom Up Inferencing:*** Given an ESNet, apply the following reduction (resolution) using modus ponen rule of logic ({A ← B, B} then A).
  - Backward reasoning inference (uses top down approach).
    - ***Top Down Inferencing:*** Prove a conclusion from a given ESNet by adding the denial of the conclusion to the network and show that the resulting set of clauses in the network is inconsistent.

Example: Bottom Up Inferencing

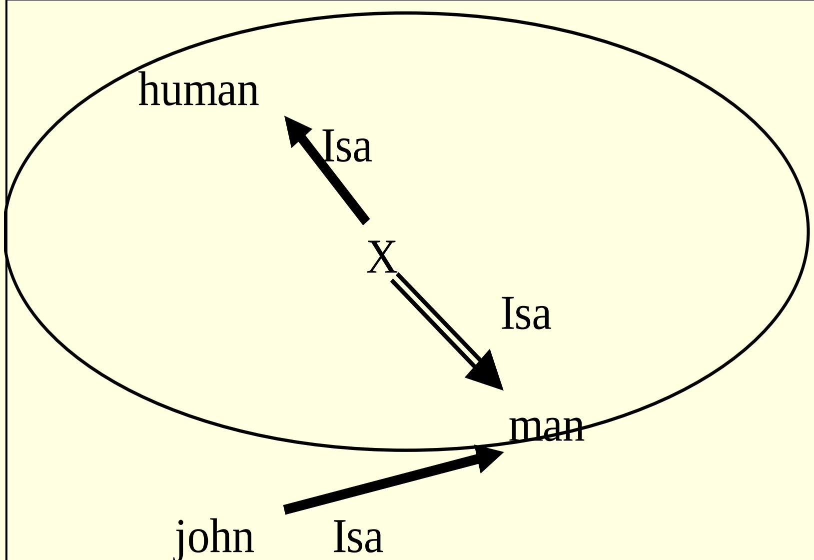
<table>
<tr><th>Given set of clauses</th><th>Inferencing</th></tr>
<tr><td>Isa(X, human) ← Isa(X, man)
Isa(john, man).</td><td>Isa(john, human)</td></tr>
<tr><td>


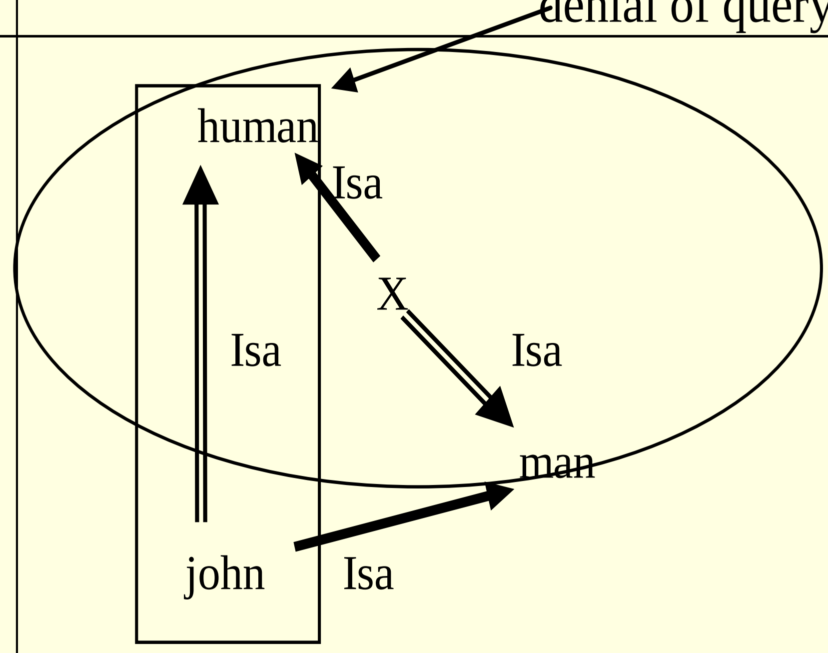
Here X is bound to john
</td><td>


</td></tr>
</table>

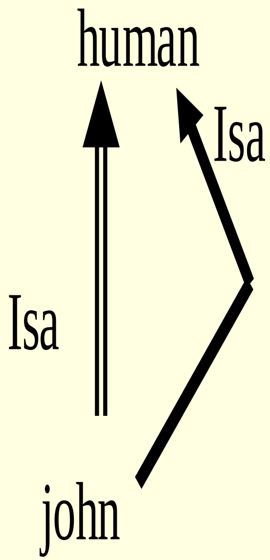
Example: Top Down Inferencing

Given set of clauses	**Prove conclusion**
Isa(X, human) ← Isa(X, man) Isa(john, man).	Query: Isa(john, human) denial of query





Cont…



X = john

Contradiction or Empty network is generated. Hence “Isa(john, human)” is proved.

Monotonic & Non-Monotonic Reasoning

Monotonic Reasoning:

In monotonic reasoning, once the conclusion is taken, then it will remain the same even if we add some other information to existing information in our knowledge base. In monotonic reasoning, adding knowledge does not decrease the set of prepositions that can be derived.

To solve monotonic problems, we can derive the valid conclusion from the available facts only, and it will not be affected by new facts.

Monotonic reasoning is used in conventional reasoning systems, and a logic-based system is monotonic.

Any theorem proving is an example of monotonic reasoning.

Example:
Earth revolves around the Sun.
It is a true fact, and it cannot be changed even if we add another sentence in knowledge base like, "The moon revolves around the earth" Or "Earth is not round," etc.

Advantages of Monotonic Reasoning:

In monotonic reasoning, each old proof will always remain valid. If we deduce some facts from available facts, then it will remain valid for always.

Disadvantages of Monotonic Reasoning:
We cannot represent the real world scenarios using Monotonic reasoning.

Hypothesis knowledge cannot be expressed with monotonic reasoning, which means facts should be true.

Since we can only derive conclusions from the old proofs, so new knowledge from the real world cannot be added.

Non-monotonic Reasoning:

In Non-monotonic reasoning, some conclusions may be invalidated if we add some more information to our knowledge base.
Logic will be said as non-monotonic if some conclusions can be invalidated by adding more knowledge into our knowledge base.
Non-monotonic reasoning deals with incomplete and uncertain models.

"Human perceptions for various things in daily life, "is a general example of non-monotonic reasoning.

Example: Let suppose the knowledge base contains the following knowledge:

Birds can fly
Penguins cannot fly
Pitty is a bird

So from the above sentences, we can conclude that **Pitty can fly**.

However, if we add one another sentence into knowledge base "**Pitty is a penguin**", which concludes "**Pitty cannot fly**", so it invalidates the above conclusion.

Advantages of Non-monotonic reasoning:
For real-world systems such as Robot navigation, we can use non-monotonic reasoning.
In Non-monotonic reasoning, we can choose probabilistic facts or can make assumptions.
Disadvantages of Non-monotonic Reasoning:
In non-monotonic reasoning, the old facts may be invalidated by adding new sentences.
It cannot be used for theorem **proving**.

Preposition Logic

Solution

1. Marcus was a man.
   - Man(Marcus).
2. Marcus was a Pompeian.
   - Pompeian(marcus)
3. All Pompeian were Romans.
   - $\forall x$: Pompeian(x) $\rightarrow$ Roman(x)
4. Caesar was a ruler.
   - Ruler(Caesar)
5. All Romans were either loyal to Caesar or hated him.
   - $\forall x$: Roman(x) $\rightarrow$ LoyalTo(x,Caesar) $\vee$ Hate(x,Caesar)

6. Everyone is loyal to someone.

$\forall x: \exists y: LoyalTo(x,y)$

7. People only try to assassinate rulers they aren't loyal to.

$\forall x: \forall y: Person(x) \wedge Ruler(y) \wedge TryAssassinate(x,y) \rightarrow \neg LoyalTo(x,y)]$

8. Marcus tried to assassinate Caesar.

TryAssassinate(Marcus, Caesar)

9. All men are people.

$\forall x: Men(x) \rightarrow People(x)$

Resolution by Refutation

Problem Statement:

1. Ravi likes all kind of food.
2. Apples and chicken are food
3. Anything anyone eats and is not killed is food
4. Ajay eats peanuts and is still alive
5. Rita eats everything that Ajay eats.

Prove by resolution that Ravi likes peanuts using resolution.

Step 1: Converting the given statements into Predicate/Propositional Logic

i. ∀x : food(x) → likes (Ravi, x)

ii. food (Apple) ^ food (chicken)

iii. ∀a : ∀b: eats (a, b) ^ killed (a) → food (b)

iv. eats (Ajay, Peanuts) ^ alive (Ajay)

v. ∀c : eats (Ajay, c) → eats (Rita, c)

vi. ∀d : alive(d) → ~killed (d)

vii. ∀e: ~killed(e) → alive(e)

Conclusion: likes (Ravi, Peanuts)

Step 2: Convert into CNF

i. ~food(x) v likes (Ravi, x)

ii. Food (apple)

iii. Food (chicken)

iv. ~ eats (a, b) v killed (a) v food (b)

v. Eats (Ajay, Peanuts)

vi. Alive (Ajay)

vii. ~eats (Ajay, c) V eats (Rita, c)

viii. ~alive (d) v ~ killed (d)

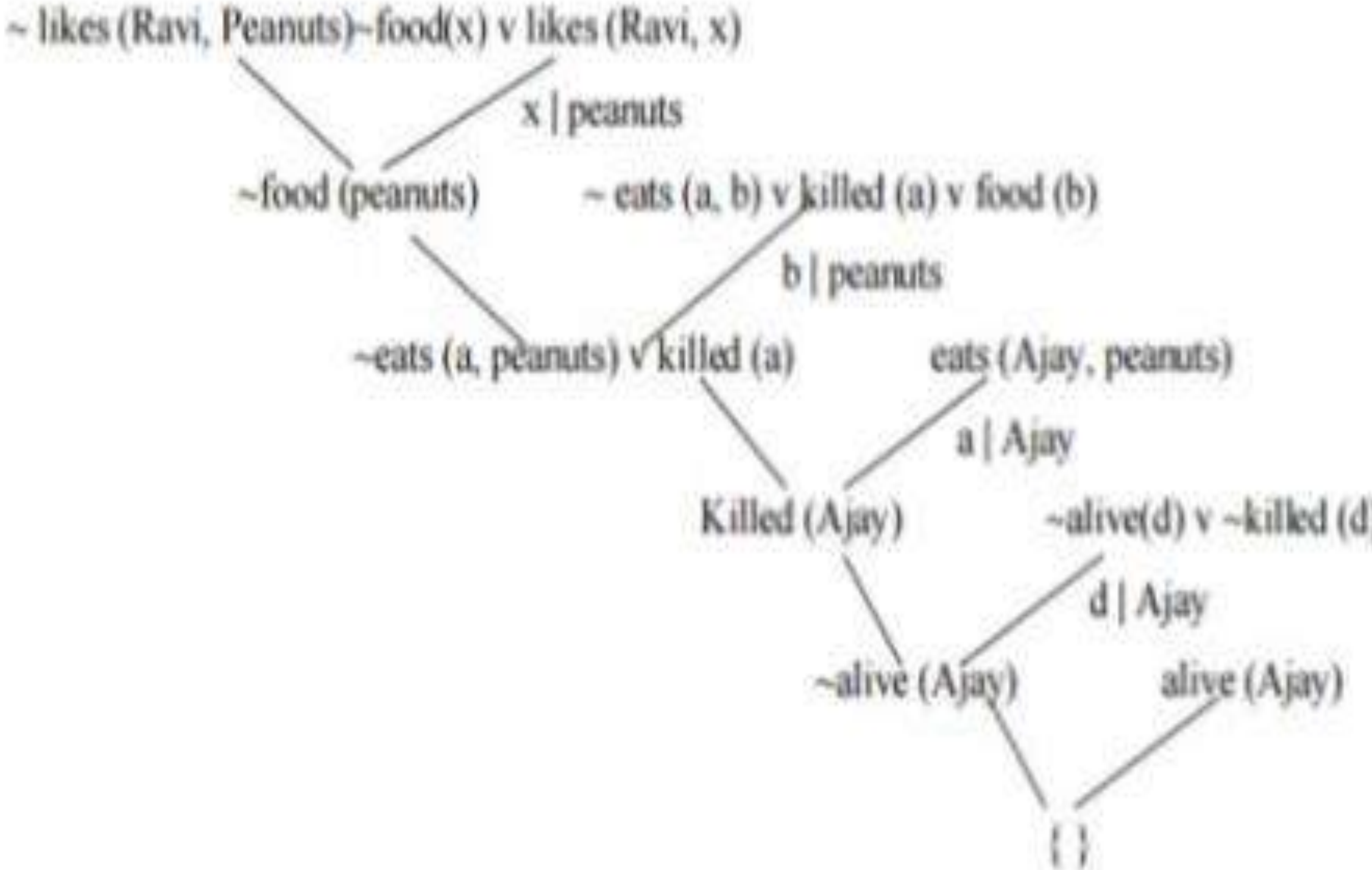
ix. Killed (e) v alive (e)

Conclusion: likes (Ravi, Peanuts)

Step 3: Negate the conclusion

~ likes (Ravi, Peanuts)

Step 4: Resolve using a resolution tree


~ likes (Ravi, Peanuts)
~food(x) v likes (Ravi, x)
x | peanuts
~food (peanuts)
~ eats (a, b) v killed (a) v food (b)
b | peanuts
~eats (a, peanuts) v killed (a)
eats (Ajay, peanuts)
a | Ajay
Killed (Ajay)
~alive(d) v ~killed (d)
d | Ajay
~alive (Ajay)
alive (Ajay)
{ }

Forward Chaining and backward chaining in AI

In artificial intelligence, forward and backward chaining is one of the important topics, but before understanding forward and backward chaining lets first understand that from where these two terms came.

Inference engine:

The inference engine is the component of the intelligent system in artificial intelligence, which applies **logical rules to the knowledge base to infer new information from known facts.** The first inference engine was part of the expert system. Inference engine commonly proceeds in two modes, which are:

1. **Forward chaining**
2. **Backward chaining**

Horn Clause and Definite clause:
Horn clause and definite clause are the forms of sentences, which enables knowledge base to use a more restricted and efficient inference algorithm.

Logical inference algorithms use forward and backward chaining approaches, which require KB in the form of the **first-order definite clause**.

Definite clause: A clause which is a disjunction of literals with **exactly one positive literal** is known as a definite clause or strict horn clause.

Horn clause: A clause which is a disjunction of literals with **at most one positive literal** is known as horn clause. Hence all the definite clauses are horn clauses.
Example: ($\neg$ p V $\neg$ q V k). It has only one positive literal k.
It is equivalent to $p \land q \rightarrow k$.

A. Forward Chaining:

Forward chaining is also known as a forward deduction or forward reasoning method when using an inference engine.

Forward chaining is a form of reasoning which start with atomic sentences in the knowledge base and applies inference rules (Modus Ponens) in the forward direction to extract more data until a goal is reached.

The Forward-chaining algorithm starts from known facts, triggers all rules whose premises are satisfied, and add their conclusion to the known facts. This process repeats until the problem is solved.

Properties of Forward-Chaining:

- It is a down-up approach, as it moves from bottom to top.
- It is a process of making a conclusion based on known facts or data, by starting from the initial state and reaches the goal state.
- Forward-chaining approach is also called as data-driven as we reach to the goal using available data.
- Forward -chaining approach is commonly used in the expert system, such as CLIPS, business, and production rule systems.

Example:

"As per the law, it is a crime for an American to sell weapons to hostile nations. Country A, an enemy of America, has some missiles, and all the missiles were sold to it by Robert, who is an American citizen."

Prove that **"Robert is criminal."**

Facts Conversion into FOL:

It is a crime for an American to sell weapons to hostile nations.
(Let's say p, q, and r are variables)

American (p) ∧ weapon(q) ∧ sells (p, q, r) ∧ hostile(r) → Criminal(p) ...(1)

Country A has some missiles.
?p Owns(A, p) ∧ Missile(p).

It can be written in two definite clauses by using Existential Instantiation, introducing new Constant T1.
Owns(A, T1) ......(2)
Missile(T1) ....(3)

All of the missiles were sold to country A by Robert.

Missiles(p) ∧ Owns (A, p) → Sells (Robert, p, A) ......(4)

Missiles are weapons.

Missile(p) → Weapons (p) ....(5)

Enemy of America is known as hostile.

Enemy(p, America) →Hostile(p) ....(6)

Country A is an enemy of America.

Enemy (A, America) ....(7)

Robert is American

American(Robert). ....(8)

Forward chaining proof:

Step-1:
In the first step we will start with the known facts and will choose the sentences which do not have implications, such as:

American(Robert), Enemy(A, America), Owns(A, T1), and Missile(T1).

All these facts will be represented as below.

American (Robert)	Missile (T1)	Owns (A,T1)	Enemy (A, America)

Step-2:

At the second step, we will see those facts which infer from available facts and with satisfied premises.

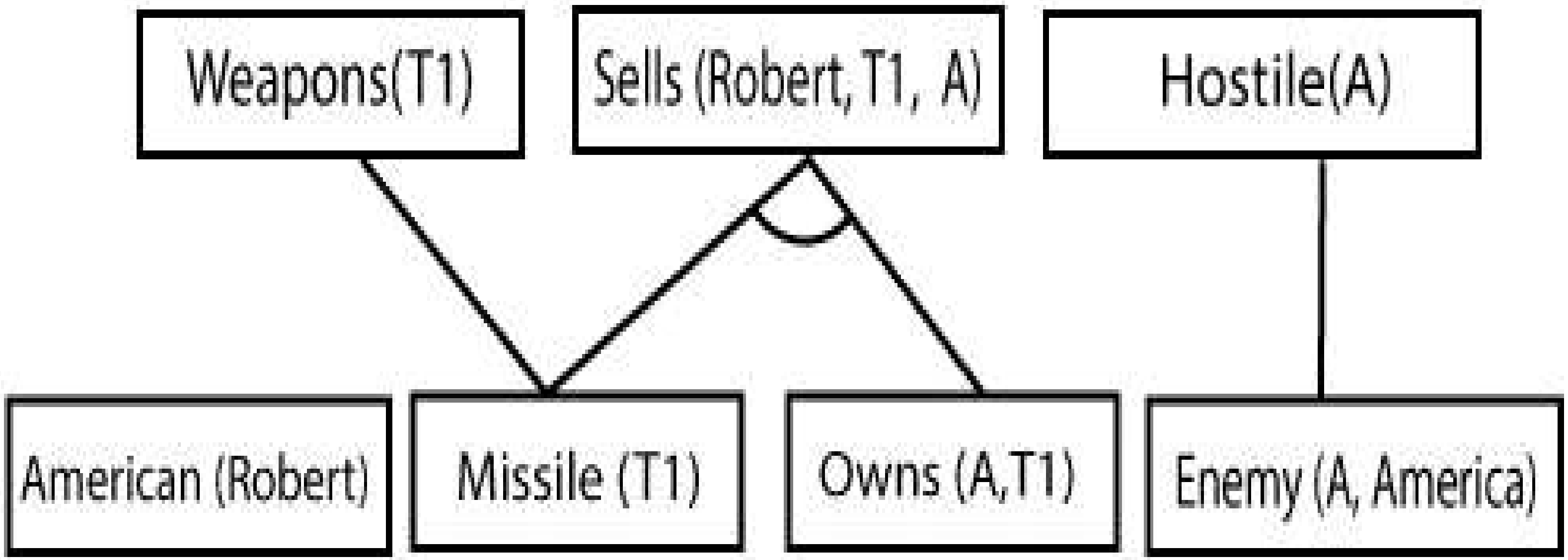
Rule-(1) does not satisfy premises, so it will not be added in the first iteration.

Rule-(2) and (3) are already added.

Rule-(4) satisfy with the substitution {p/T1},

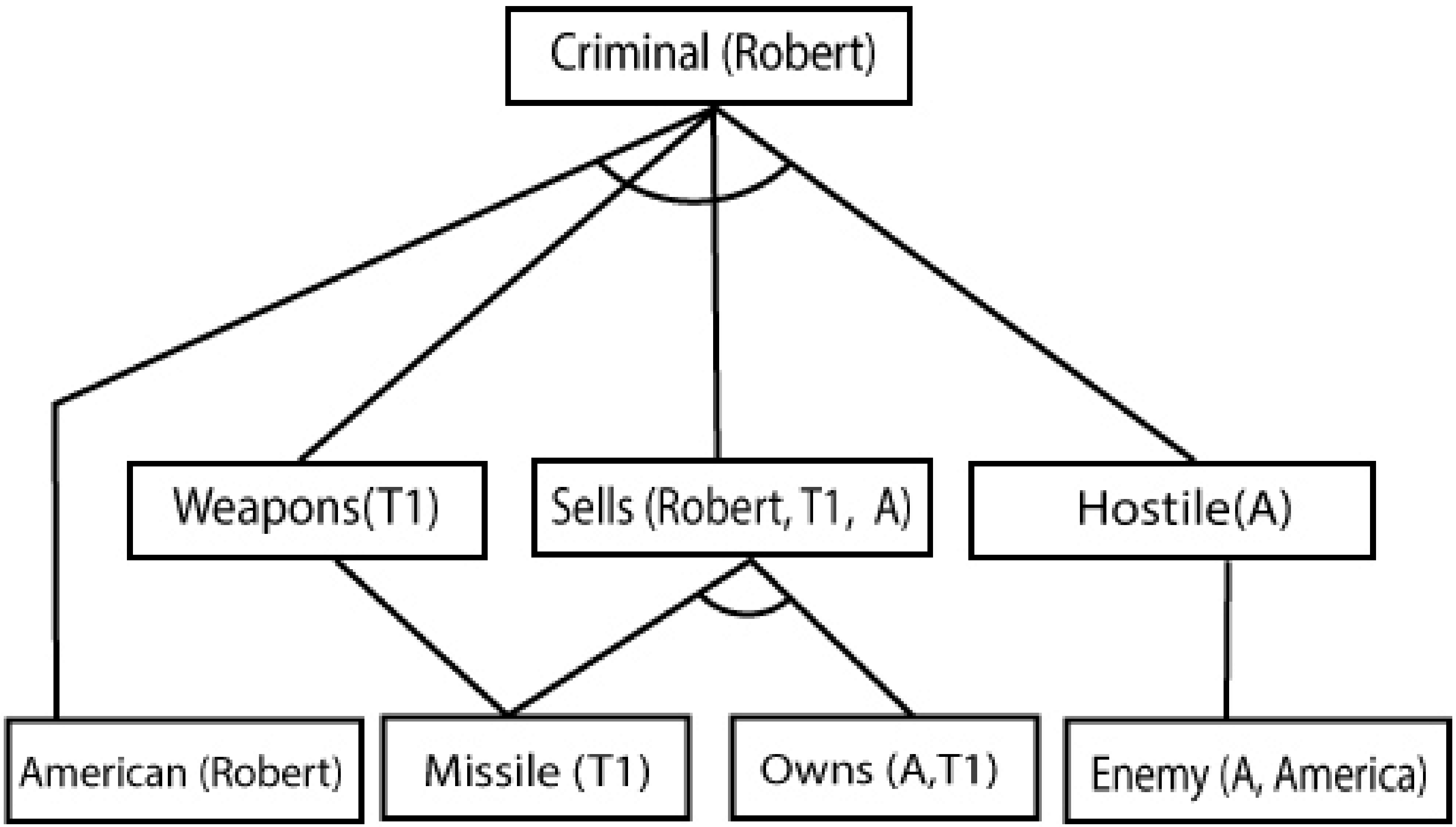
so Sells (Robert, T1, A) is added, which infers from the conjunction of Rule (2) and (3).

Rule-(6) is satisfied with the substitution(p/A), so Hostile(A) is added and which infers from Rule-(7).


Weapons(T1)
Sells (Robert, T1, A)
Hostile(A)
American (Robert)
Missile (T1)
Owns (A,T1)
Enemy (A, America)

Step-3:

At step-3, as we can check Rule-(1) is satisfied with the substitution **{p/Robert, q/T1, r/A}, so we can add Criminal(Robert)** which infers all the available facts. And hence we reached our goal statement.



B. Backward Chaining:

Backward-chaining is also known as a backward deduction or backward reasoning method when using an inference engine. A backward chaining algorithm is a form of reasoning, which starts with the goal and works backward, chaining through rules to find known facts that support the goal.

Properties of backward chaining:

- It is known as a top-down approach.
- Backward-chaining is based on modus ponens inference rule.
- In backward chaining, the goal is broken into sub-goal or sub-goals to prove the facts true.
- It is called a goal-driven approach, as a list of goals decides which rules are selected and used.
- Backward -chaining algorithm is used in game theory, automated theorem proving tools, inference engines, proof assistants, and various AI applications.
- The backward-chaining method mostly used a **depth-first search** strategy for proof.

Example:

In backward-chaining, we will use the same above example, and will rewrite all the rules.

American (p) ∧ weapon(q) ∧ sells (p, q, r) ∧ hostile(r) → Criminal(p) ...(1)

Owns(A, T1) ....(2)
Missile(T1)

?p Missiles(p) ∧ Owns (A, p) → Sells (Robert, p, A) ......(4)
Missile(p) → Weapons (p) ....(5)
Enemy(p, America) →Hostile(p) ....(6)
Enemy (A, America) ....(7)
American(Robert). ....(8)

Backward-Chaining proof:

In Backward chaining, we will start with our goal predicate, which is **Criminal(Robert)**, and then infer further rules.

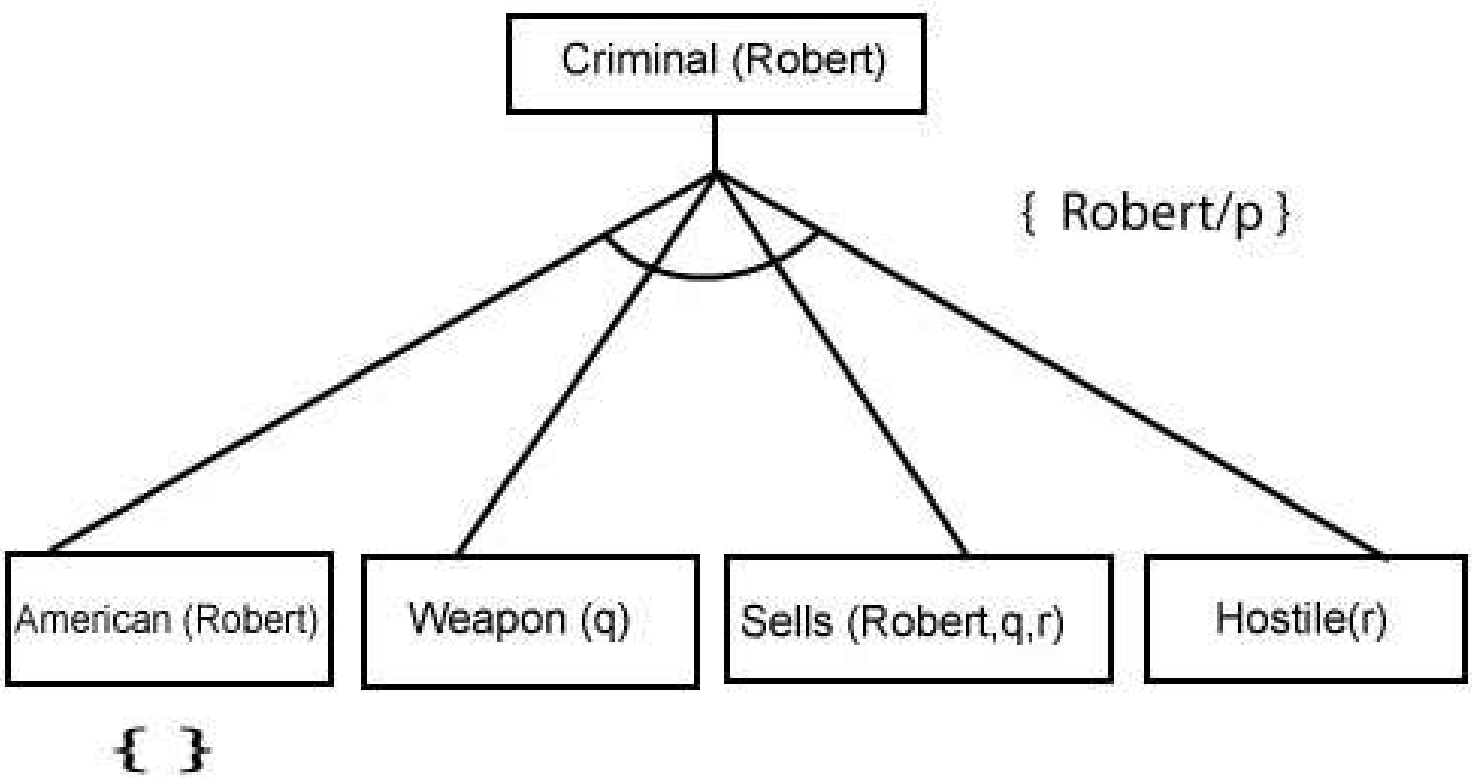
Step-1:

At the first step, we will take the goal fact. And from the goal fact, we will infer other facts, and at last, we will prove those facts true. So our goal fact is "Robert is Criminal," so following is the predicate of it.

Step-2:

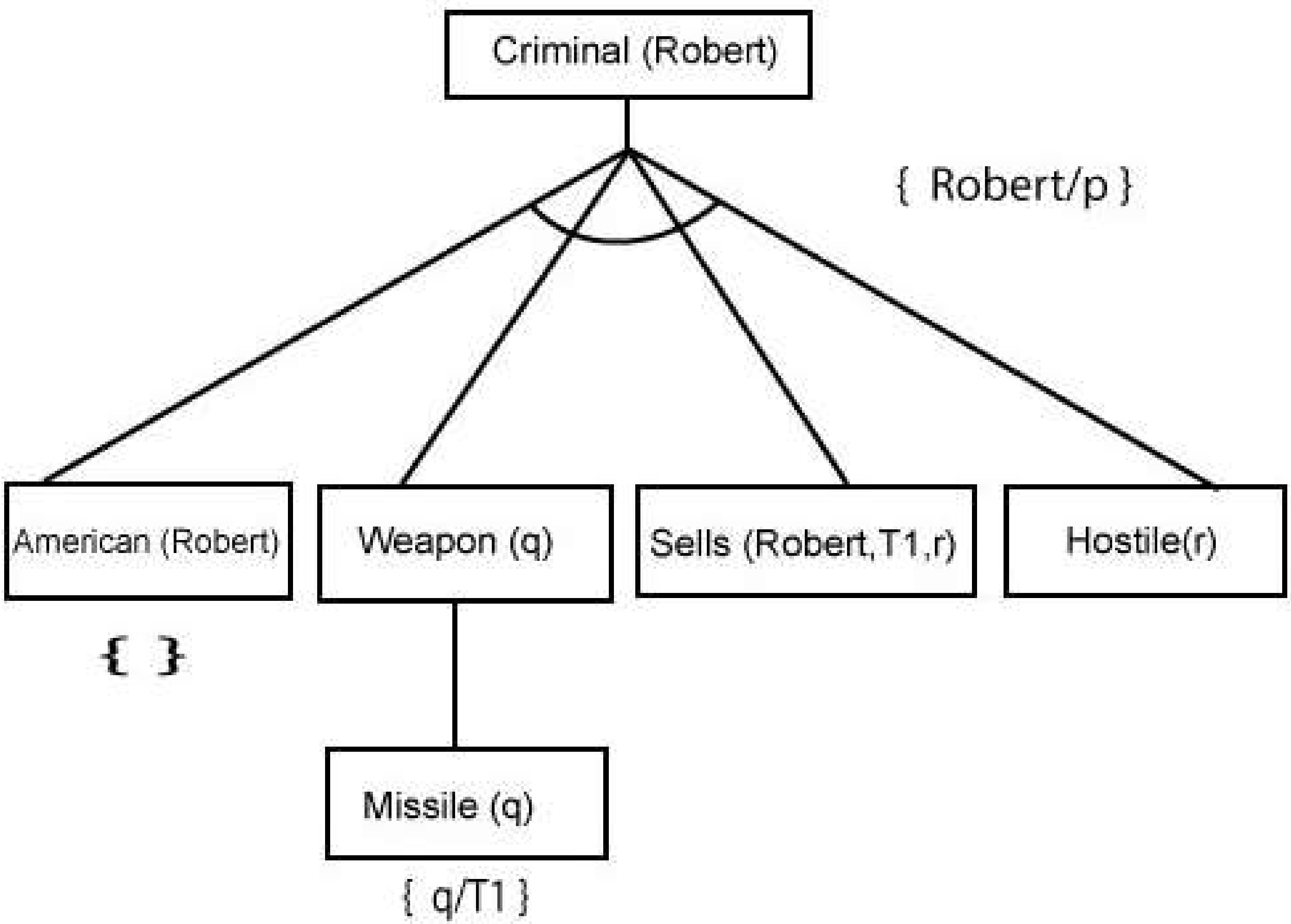
At the second step, we will infer other facts form goal fact which satisfies the rules. So as we can see in Rule-1, the goal predicate Criminal (Robert) is present with substitution {Robert/P}. So we will add all the conjunctive facts below the first level and will replace p with Robert.

Here we can see American (Robert) is a fact, so it is proved here



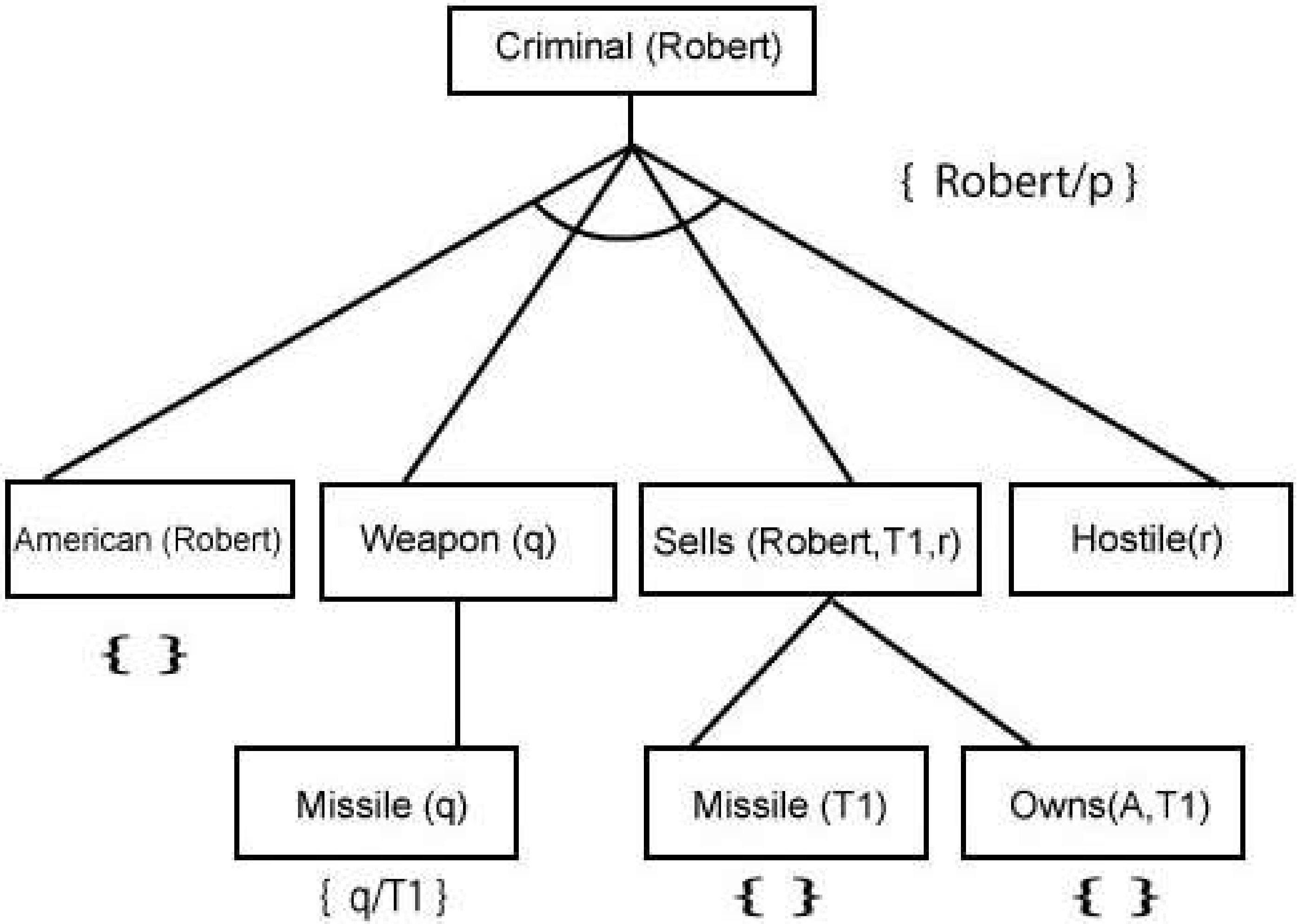
Step-3:

At step-3, we will extract further fact Missile(q) which infer from Weapon(q), as it satisfies Rule-(5). Weapon (q) is also true with the substitution of a constant T1 at q.

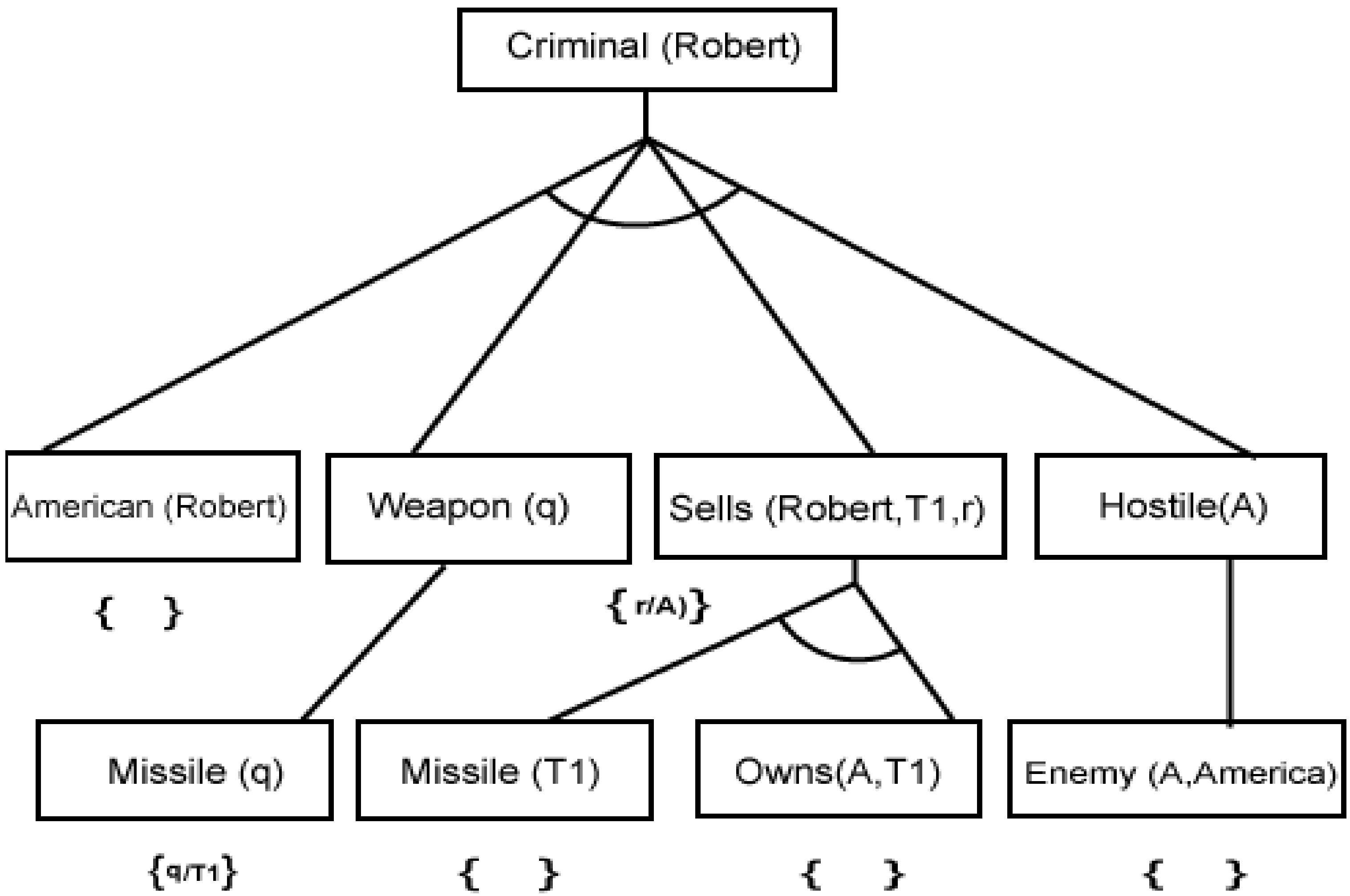


Step-4:

At step-4, we can infer facts Missile(T1) and Owns(A, T1) form Sells(Robert, T1, r) which satisfies the **Rule- 4**, with the substitution of A in place of r. So these two statements are proved here.



Step-5: we can infer the fact **Enemy(A, America)** from **Hostile(A)** which satisfies Rule- 6. And hence all the statements are proved true using backward chaining



Probabilistic reasoning in Artificial intelligence

Uncertainty:

Till now, we have learned knowledge representation using first-order logic and propositional logic with certainty, which means we were sure about the predicates.

With this knowledge representation, we might write A→B, which means if A is true then B is true,

Consider a situation where we are not sure about whether A is true or not then we cannot express this statement, this situation is called uncertainty.
So to represent uncertain knowledge, where we are not sure about the predicates, we need **uncertain reasoning or probabilistic reasoning.**

Causes of uncertainty:

Following are some leading causes of uncertainty to occur in the real world.

- Information occurred from unreliable sources.
- Experimental Errors
- Equipment fault
- Temperature variation
- Climate change.

Probabilistic reasoning:

- Probabilistic reasoning is a way of knowledge representation where we apply the concept of probability to indicate the uncertainty in knowledge.
- In probabilistic reasoning, we combine probability theory with logic to handle the uncertainty.
- We use probability in probabilistic reasoning because it provides a way to handle the uncertainty that is the result of someone's laziness and ignorance.
- In the real world, there are lots of scenarios, where the certainty of something is not confirmed, such as "It will rain today," "behavior of someone for some situations," "A match between two teams or two players." These are probable sentences for which we can assume that it will happen but not sure about it, so here we use probabilistic reasoning.

Need of probabilistic reasoning in AI:

- When there are unpredictable outcomes.
- When specifications or possibilities of predicates becomes too large to handle.
- When an unknown error occurs during an experiment.

In probabilistic reasoning, there are two ways to solve problems with uncertain knowledge:

- **Bayes' rule**
- **Bayesian Statistics**

Probability:

Probability can be defined as a chance that an uncertain event will occur. It is the numerical measure of the likelihood that an event will occur. The value of probability always remains between 0 and 1 that represent ideal uncertainties.

$0 \leq P(A) \leq 1$, where $P(A)$ is the probability of an event A.

$P(A) = 0$, indicates total uncertainty in an event A.

$P(A) = 1$, indicates total certainty in an event A.

We can find the probability of an uncertain event by using the below formula.

$$\textbf{Probability of occurrence} = \frac{\text{Number of desired outcomes}}{\text{Total number of outcomes}}$$

- $P(\neg A)$ = probability of a not happening event.
- $P(\neg A) + P(A) = 1$.

Event: Each possible outcome of a variable is called an event.

Sample space: The collection of all possible events is called sample space.

Random variables: Random variables are used to represent the events and objects in the real world.

Prior probability: The prior probability of an event is probability computed before observing new information.

Posterior Probability: The probability that is calculated after all evidence or information has taken into account. It is a combination of prior probability and new information.

Conditional probability:

- Conditional probability is a probability of occurring an event when another event has already happened.
- Let's suppose, we want to calculate the event A when event B has already occurred, "the probability of A under the conditions of B", it can be written as:

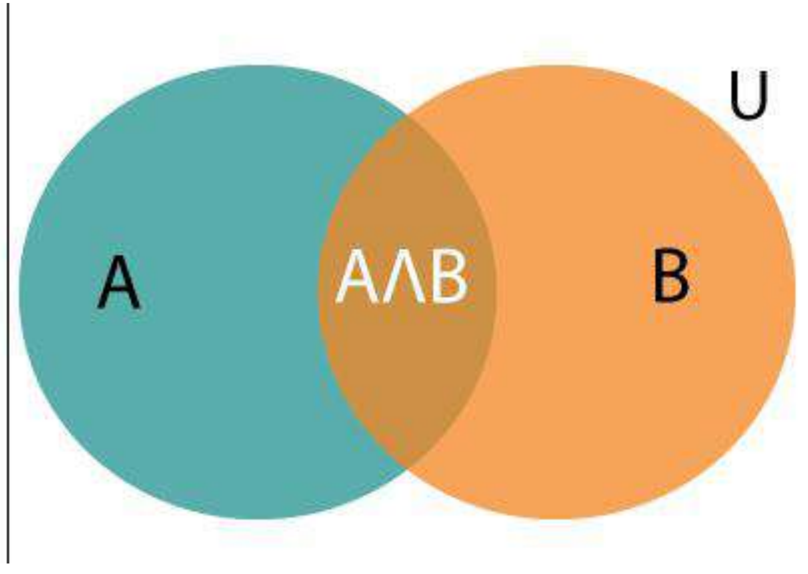
P(A/B)

Where P($A \wedge B$)= Joint probability of a and B
P(B)= Marginal probability of B.

If the probability of A is given and we need to find the probability of B, then it will be given as:

If the probability of A is given and we need to find the probability of B, then it will be given as:

It can be explained by using the below Venn diagram, where B is occurred event, so sample space will be reduced to set B, and now we can only calculate event A when event B is already occurred by dividing the probability of **P(A∧B) by P(B)**.



Bayes' theorem in Artificial intelligence

Bayes' theorem:

Bayes' theorem is also known as **Bayes' rule, Bayes' law**, or **Bayesian reasoning**, which determines the probability of an event with uncertain knowledge.

In probability theory, it relates the conditional probability and marginal probabilities of two random events.

Bayes' theorem was named after the British mathematician **Thomas Bayes**. The **Bayesian inference** is an application of Bayes' theorem, which is fundamental to Bayesian statistics.

It is a way to calculate the value of $P(B|A)$ with the knowledge of $P(A|B)$.

Bayes' theorem allows updating the probability prediction of an event by observing new information of the real world.

Example: If cancer corresponds to one's age then by using Bayes' theorem, we can determine the probability of cancer more accurately with the help of age.
Bayes' theorem can be derived using product rule and conditional probability of event A with known event B:
As from product rule we can write:

$$P(A \wedge B) = P(A|B)\, P(B) \text{ or}$$

Similarly, the probability of event B with known event A:

$$P(A \wedge B) = P(B|A)\, P(A)$$

Equating right hand side of both the equations, we will get:

$$P(A|B) = \frac{P(B|A)\, P(A)}{P(B)} \quad \text{....(a)}$$

The above equation (a) is called as **Bayes' rule** or **Bayes' theorem**. This equation is basic of most modern AI systems for **probabilistic inference**.

$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)}$$

- P(A|B) is known as **posterior**, which we need to calculate, and it will be read as Probability of hypothesis A when we have occurred an evidence B.

- P(B|A) is called the likelihood, in which we consider that hypothesis is true, then we calculate the probability of evidence.

- P(A) is called the **prior probability**, probability of hypothesis before considering the evidence

- P(B) is called **marginal probability**, pure probability of an evidence.

In the equation (a), in general, we can write P (B) = P(A)*P(B|Ai), hence the Bayes' rule can be written as:

$$P(A_i|B) = \frac{P(A_i)*P(B|A_i)}{\sum_{i=1}^{k} P(A_i)*P(B|A_i)}$$

Where A_1, A_2, A_3,........, A_n is a set of mutually exclusive and exhaustive events.

Applying Bayes' rule:

Bayes' rule allows us to compute the single term P(B|A) in terms of P(A|B), P(*B*), and P(A).

This is very useful in cases where we have a good probability of these three terms and want to determine the fourth one.

Suppose we want to perceive the effect of some unknown cause, and want to compute that cause, then the Bayes' rule becomes:

Example-1:

Question: what is the probability that a patient has diseases meningitis with a stiff neck?

Given Data:

A doctor is aware that disease meningitis causes a patient to have a stiff neck, and it occurs 80% of the time. He is also aware of some more facts, which are given as follows:

The Known probability that a patient has meningitis disease is 1/30,000.

The Known probability that a patient has a stiff neck is 2%.

Let a be the proposition that patient has stiff neck and b be the proposition that patient has meningitis. , so we can calculate the following as:

$P(a|b) = 0.8$

$P(b) = 1/30000$

$P(a)= .02$

$$\mathbf{P(b|a)} = \frac{P(a|b)P(b)}{P(a)} = \frac{0.8*(\frac{1}{30000})}{0.02} = 0.001333333.$$

Hence, we can assume that 1 patient out of 750 patients has meningitis disease with a stiff neck.

Applying Bayes' rule:

Bayes' rule allows us to compute the single term P(B|A) in terms of P(A|B), P(*B*), and P(A).

This is very useful in cases where we have a good probability of these three terms and want to determine the fourth one.

Suppose we want to perceive the effect of some unknown cause, and want to compute that cause, then the Bayes' rule becomes:

$$\mathbf{P(cause \mid effect)} = \frac{\mathrm{P(effect|cause)\ P(cause)}}{\mathrm{P(effect)}}$$

Example-2:

Question: From a standard deck of playing cards, a single card is drawn. The probability that the card is king is 4/52, then calculate posterior probability P(King|Face), which means the drawn face card is a king card.

Bayesian Belief Network in artificial intelligence

Bayesian belief network is key computer technology for dealing with probabilistic events and to solve a problem which has uncertainty. We can define a Bayesian network as:

"A Bayesian network is a probabilistic graphical model which represents a set of variables and their conditional dependencies using a directed acyclic graph."

It is also called a **Bayes network, belief network, decision network**, or **Bayesian model**.

Bayesian networks are probabilistic, because these networks are built from a **probability distribution**, and also use probability theory for prediction and anomaly detection.

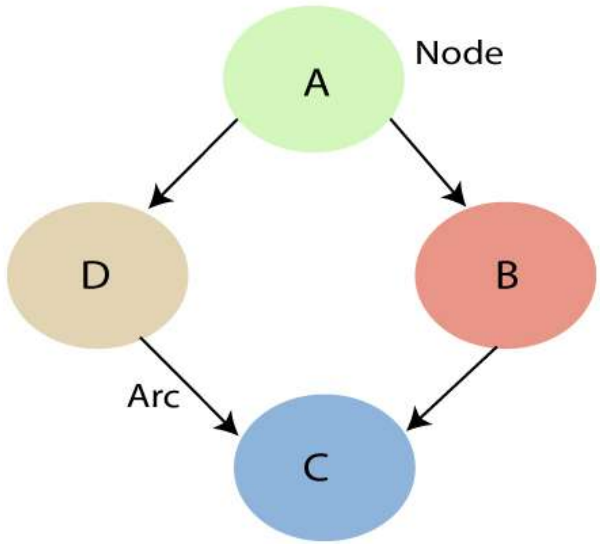
Real world applications are probabilistic in nature, and to represent the relationship between multiple events, we need a Bayesian network. It can also be used in various tasks including **prediction, anomaly detection, diagnostics, automated insight, reasoning, time series prediction**, and **decision making under uncertainty**.

Bayesian Network can be used for building models from data and experts opinions, and it consists of two parts:

Directed Acyclic Graph
Table of conditional probabilities.

The generalized form of Bayesian network that represents and solve decision problems under uncertain knowledge is known as an **Influence diagram**.

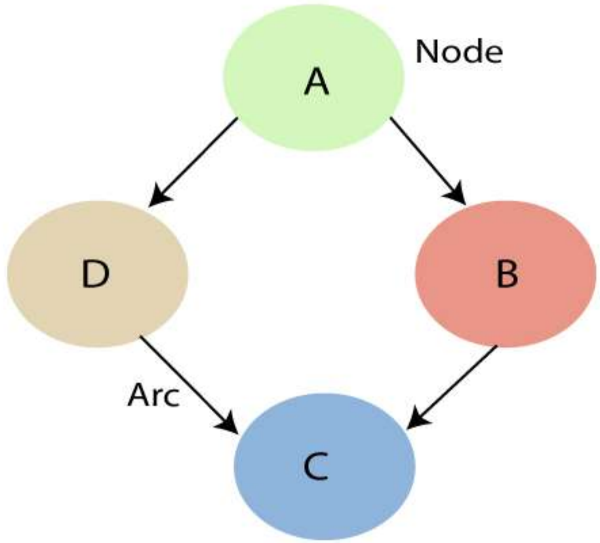
A Bayesian network graph is made up of nodes and Arcs (directed links), where:



Each **node** corresponds to the random variables, and a variable can be **continuous** or **discrete**.

Arc or directed arrows represent the causal relationship or conditional probabilities between random variables. These directed links or arrows connect the pair of nodes in the graph.

A Bayesian network graph is made up of nodes and Arcs (directed links), where:



These links represent that one node directly influence the other node, and if there is no directed link that means that nodes are independent with each other

In the above diagram, A, B, C, and D are random variables represented by the nodes of the network graph.

If we are considering node B, which is connected with node A by a directed arrow, then node A is called the parent of Node B.

Node C is independent of node A.

The Bayesian network has mainly two components:

Causal Component

Actual numbers

Each node in the Bayesian network has condition probability distribution $\mathbf{P(X_i \mid Parent(X_i))}$, which determines the effect of the parent on that node.

Bayesian network is based on Joint probability distribution and conditional probability. So let's first understand the joint probability distribution:

Joint probability distribution:

If we have variables x1, x2, x3,....., xn, then the probabilities of a different combination of x1, x2, x3.. xn, are known as Joint probability distribution.

$P[x_1, x_2, x_3,....., x_n]$, it can be written as the following way in terms of the joint probability distribution.

$= P[x_1 | x_2, x_3,....., x_n]P[x_2, x_3,....., x_n]$

$= P[x_1 | x_2, x_3,....., x_n]P[x_2 | x_3,....., x_n]....P[x_{n-1} | x_n]P[x_n]$.

In general for each variable Xi, we can write the equation as:

$$P(X_i | X_{i-1}, ..., X_1) = P(X_i | Parents(X_i))$$

Explanation of Bayesian network:
Let's understand the Bayesian network through an example by creating a directed acyclic graph:

Example: Harry installed a new burglar alarm at his home to detect burglary. The alarm reliably responds at detecting a burglary but also responds for minor earthquakes. Harry has two neighbors David and Sophia, who have taken a responsibility to inform Harry at work when they hear the alarm. David always calls Harry when he hears the alarm, but sometimes he got confused with the phone ringing and calls at that time too. On the other hand, Sophia likes to listen to high music, so sometimes she misses to hear the alarm. Here we would like to compute the probability of Burglary Alarm.

Problem:

Calculate the probability that alarm has sounded, but there is neither a burglary, nor an earthquake occurred, and David and Sophia both called the Harry.

Solution:

The Bayesian network for the above problem is given below. The network structure is showing that burglary and earthquake is the parent node of the alarm and directly affecting the probability of alarm's going off, but David and Sophia's calls depend on alarm probability.

The network is representing that our assumptions do not directly perceive the burglary and also do not notice the minor earthquake, and they also not confer before calling.

The conditional distributions for each node are given as conditional probabilities table or CPT.

Each row in the CPT must be sum to 1 because all the entries in the table represent an exhaustive set of cases for the variable.

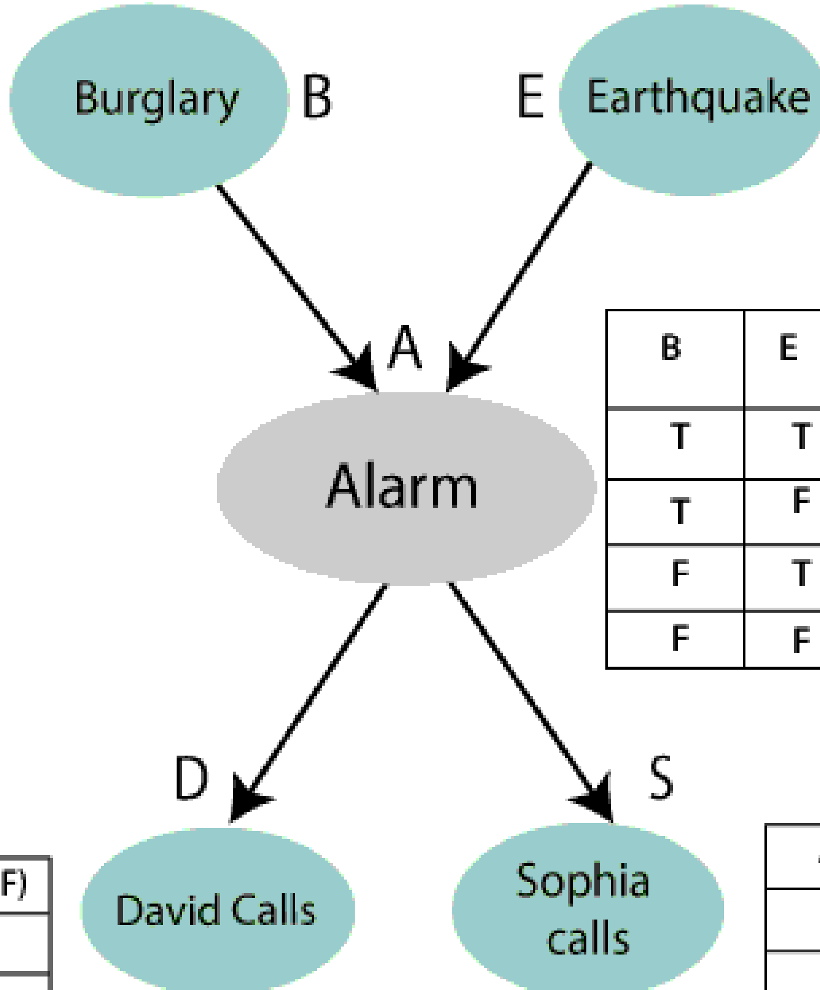
In CPT, a boolean variable with k boolean parents contains 2^K probabilities.

Hence, if there are two parents, then CPT will contain 4 probability values

List of all events occurring in this network:
Burglary (B)
Earthquake(E)
Alarm(A)
David Calls(D)
Sophia calls(S)

We can write the events of problem statement in the form of probability: **P[D, S, A, B, E]**, can rewrite the above probability statement using joint probability distribution:

P[D, S, A, B, E]= P[D | S, A, B, E]. P[S, A, B, E]
=P[D | S, A, B, E]. P[S | A, B, E]. P[A, B, E]
= P [D| A]. P [S| A, B, E]. P[A, B, E]
= P[D | A]. P[S | A]. P[A| B, E]. P[B, E]
= P[D | A]. P[S | A]. P[A| B, E]. P[B |E]. P[E]



T	0.002
F	0.998

T	0.001
F	0.999

B	E	P(A=T)	P(A=F)
T	T	0.94	0.06
T	F	0.95	0.04
F	T	0.69	0.69
F	F	0.999	0.999

A	P (D=T)	P (D=F)
T	0.91	0.09
F	0.05	0.95

A	P (S=T)	P (S=F)
T	0.75	0.25
F	0.02	0.98

Let's take the observed probability for the Burglary and earthquake component:

P(B= True) = 0.002, which is the probability of burglary.

P(B= False)= 0.998, which is the probability of no burglary.

P(E= True)= 0.001, which is the probability of a minor earthquake

P(E= False)= 0.999, Which is the probability that an earthquake not occurred.

Conditional probability table for Alarm A:

The Conditional probability of Alarm A depends on Burglar and earthquake:

B	E	P(A= True)	P(A= False)
True	True	0.94	0.06
True	False	0.95	0.04
False	True	0.31	0.69
False	False	0.001	0.999

Conditional probability table for David Calls:

The Conditional probability of David that he will call depends on the probability of Alarm.

A	P(D= True)	P(D= False)
True	0.91	0.09
False	0.05	0.95

Conditional probability table for Sophia Calls:

The Conditional probability of Sophia that she calls is depending on its Parent Node "Alarm."

A	P(S= True)	P(S= False)
True	0.75	0.25
False	0.02	0.98

Problem:
Calculate the probability that alarm has sounded, but there is neither a burglary, nor an earthquake occurred, and David and Sophia both called the Harry.

From the formula of joint distribution, we can write the problem statement in the form of probability distribution:

$P(S, D, A, \neg B, \neg E) = P(S|A) * P(D|A) * P(A|\neg B \wedge \neg E) * P(\neg B) * P(\neg E)$.

$= 0.75 * 0.91 * 0.001 * 0.998 * 0.999$

$= 0.00068045$.

Hence, a Bayesian network can answer any query about the domain by using Joint distribution.

Unit 4

Total-Order Planning

- Forward/backward state-space searches are forms of totally ordered plan search
  - explore only strictly linear sequences of actions, directly connected to the start or goal
  - cannot take advantages of problem decomposition

Partial-Order Planning (POP) - Idea:

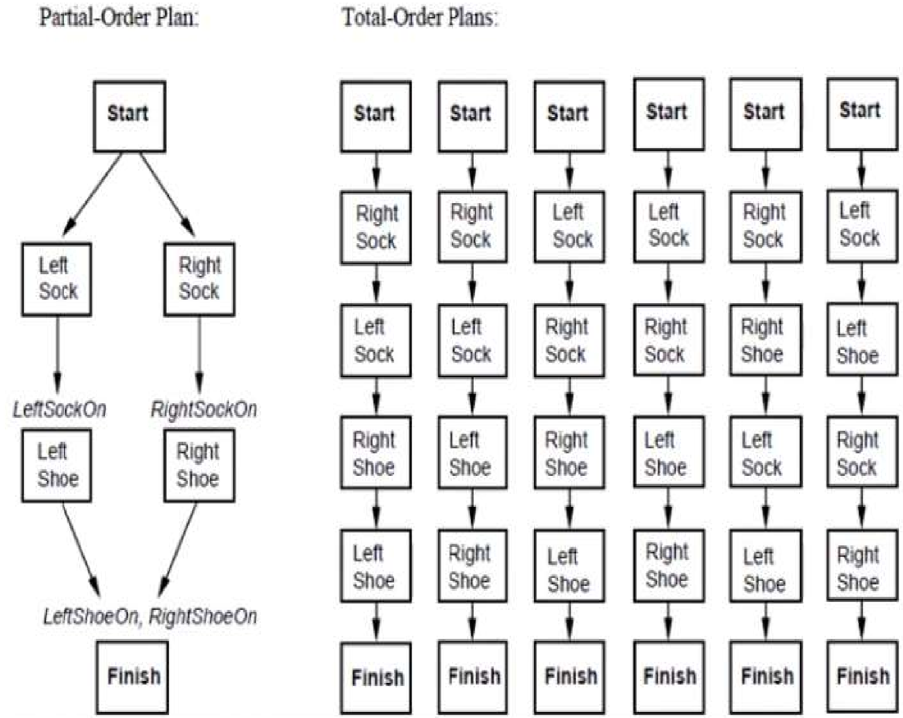
- Works on several subgoals independently
- Solves them with subplans
- Combines the subplans
- Flexibility in ordering the subplans
- Least commitment strategy:
  - delaying a choice during search
- Example, leave actions unordered, unless they must be sequential

POP Example - Putting on a pair of shoes:

- Goal(RightShoeOn ^ LeftShoeOn)
- Init()
- Action: RightShoe
  - PRECOND: RightSockOn
  - EFFECT: RightShoeOn
- Action: RightSock
  - PRECOND: None
  - EFFECT: RightSockOn
- Action:LeftShoe
  - PRECOND: LeftSockOn
  - EFFECT: LeftShoeOn
- Action: LeftSock
  - PRECOND: None
  - EFFECT: LeftSockOn

The partial-order plan - The shoes and socks problem

- A partial-order plan for putting on shoes and socks, and the six corresponding linearizations into total-order plans



How to Define Partial Order Plan?

- A set of actions, that make up the steps of the plan
- A set of ordering constrain $A \prec B$
  - A before B
- A set of causal links: $A \xrightarrow{p} B$
  - A achieves P for B $RightSock \xrightarrow{RightSockOn} RightShoe$
  - May be conflicts if C has the effect of $\neg$P and if C comes after A and before B
- A set of open preconditions:
  - A precondition is open, if it is not achieved by some action in the plan

The Initial Plan

- Initial plan contains:
- Start:
  - PRECOND: none
  - EFFECT: Add all propositions that are initially true
- Finish:
  - PRECOND: Goal state
  - EFFECT: none
- Ordering constraints: $Start \prec Finish$
- Causal links: {}
- Open preconditions:
  - {preconditions of Finish}

Next…

- Successor function
  - Arbitrarily picks one open precondition p on an action B and generates a successor plan, for every possible consistent way of choosing an action A, that achieves p
- Consistency:
  - Causal link $A \xrightarrow{p} B$ and the ordering constraint are added ( ) $(A \prec B \quad Start \prec A \quad A \prec Finish)$
  - Resolve conflict: add $B \prec C$ or $C \prec A$
- Goal test:
  - There are no open preconditions

Example: Final Plan

- The final plan has the following components:
- Actions: {RightSock, RightShoe, LeftSock, LeftShoe, Start, Finish}
- Orderings: {RightSock ‹ RightShoe, LeftSock ‹ LeftShoe}
- Open preconditions: {}
- Links:

$$RightSock \xrightarrow{RightSockOn} RightShoe$$

$$LeftSock \xrightarrow{LeftSockOn} LeftShoe$$

$$RightShoe \xrightarrow{RightShoeOn} Finish$$

$$LeftShoe \xrightarrow{LeftShoeOn} Finish$$

Example Algorithm for POP

- POP: A sound, complete partial order planner using STRIPS representation

```
function POP(initial, goal, operators) returns plan
  plan ← MAKE-MINIMAL-PLAN(initial, goal)
  loop do
    if SOLUTION?(plan) then return plan
    S_need, c ← SELECT-SUBGOAL(plan)
    CHOOSE-OPERATOR(plan, operators, S_need, c)
    RESOLVE-THREATS(plan)
  end
```

where: * c is a precondition of a step S_{need}
* RESOLVE-THREATS: orders steps as needed to ensure intermediate steps don't undo preconditions needed by other steps

POP Example - Changing a flat tire

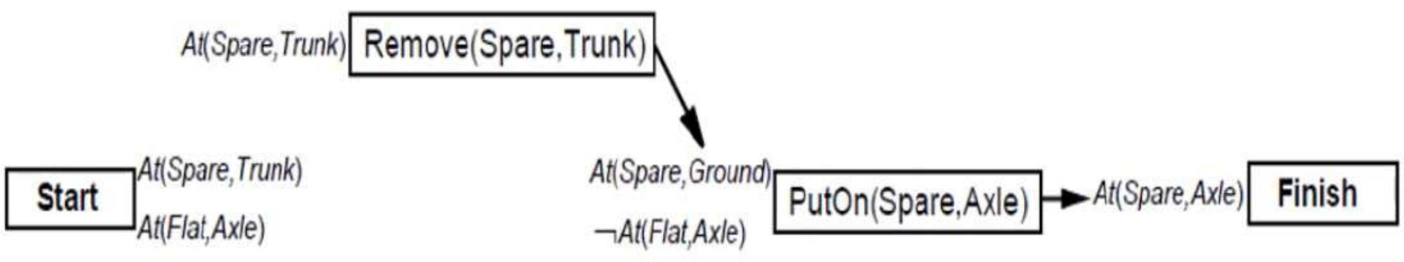
- Consider the problem of changing a flat tire.
- The goal is to have a good spare tire, properly mounted onto the car's axle,
- The initial state has a flat tire on the axle and a good spare tire in the trunk.
- There are just four actions:
  - removing the spare from the trunk,
  - removing the flat tire from the axle,
  - putting the spare on the axle, and
  - leaving the car unattended overnight.
- We assume that the car is in, particularly bad neighborhood, so that the effect of leaving it overnight is that the tires disappear.

POP Example: Flat Tire

$Init(At(Flat, Axle) \wedge At(Spare, Trunk))$
$Goal(At(Spare, Axle))$
$Action(Remove(Spare, Trunk),$
$\quad$ PRECOND: $At(Spare, Trunk)$
$\quad$ EFFECT: $\neg At(Spare, Trunk) \wedge At(Spare, Ground))$
$Action(Remove(Flat, Axle),$
$\quad$ PRECOND: $At(Flat, Axle)$
$\quad$ EFFECT: $\neg At(Flat, Axle) \wedge At(Flat, Ground))$
$Action(PutOn(Spare, Axle),$
$\quad$ PRECOND: $At(Spare, Ground) \wedge \neg At(Flat, Axle)$
$\quad$ EFFECT: $\neg At(Spare, Ground) \wedge At(Spare, Axle))$
$Action(LeaveOvernight,$
$\quad$ PRECOND:
$\quad$ EFFECT: $\neg At(Spare, Ground) \wedge \neg At(Spare, Axle) \wedge \neg At(Spare, Trunk)$
$\qquad \wedge \neg At(Flat, Ground) \wedge \neg At(Flat, Axle))$

POP Algorithm : The sequence of events

- Start : At (Spare,Trunk) ^ At (Flat, Axle) (init)
- Finish : with precondition At (Spare, Axle). (that is goal)
- Sequence of functions :
- 1. Pick the only open precondition, At (Spare, Axle) of Finish. Choose the only applicable action, PutOn(Spare, Axle).
- 2. Pick the At (Spare, Ground) precondition of PutOn(Spare, Axle). The only applicable action, to achieve it is Remove(Spare,Trunk)



• 3. Pick the : At (Flat, Axle) precondition of PutOn(Spare, Axle)

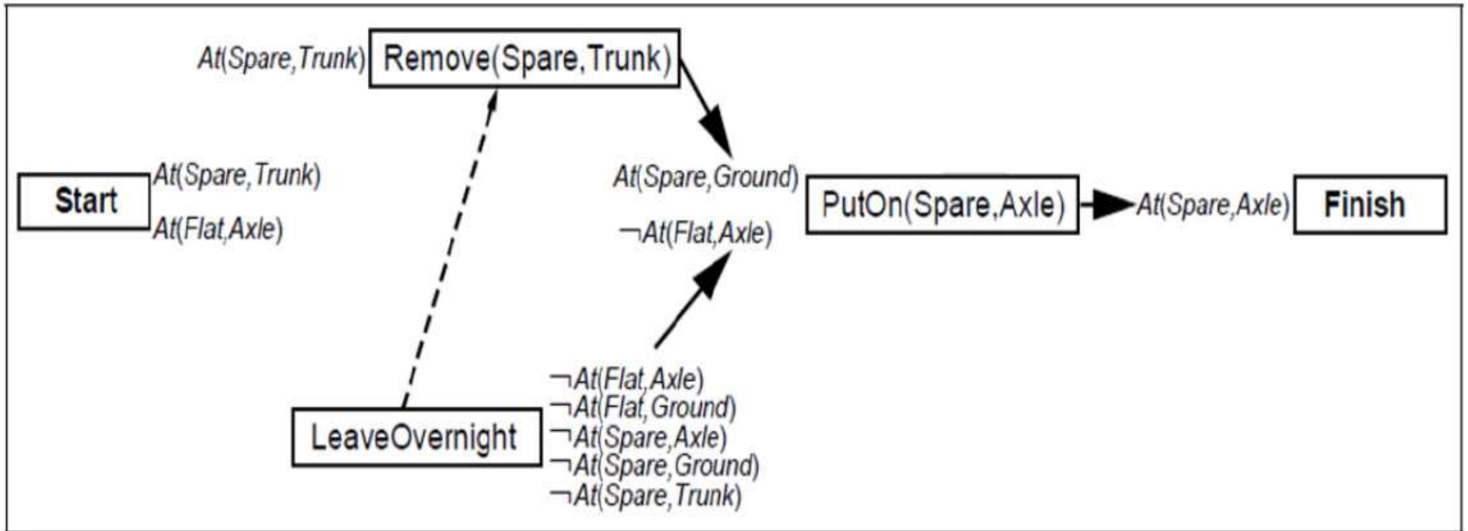


Figure 11.9 The plan after choosing *LeaveOvernight* as the action for achieving $\neg At(Flat, Axle)$. To avoid a conflict with the causal link from $Remove(Spare, Trunk)$ that protects $At(Spare, Ground)$, *LeaveOvernight* is constrained to occur before $Remove(Spare, Trunk)$, as shown by the dashed arrow.

- 4. The only remaining open precondition at this point is the At (Spare,Trunk), precondition of the action Remove(Spare,Trunk)
- 5. Consider again the : At (Flat, Axle) precondition of PutOn(Spare, Axle). This time, we choose Remove(Flat, Axle).
- 6. Once again, pick the At (Spare, Tire) precondition of Remove(Spare,Trunk) and choose Start to achieve it. This time there are no conflicts.
- 7. Pick the At (Flat, Axle) precondition of Remove(Flat, Axle), and choose Start to achieve it.

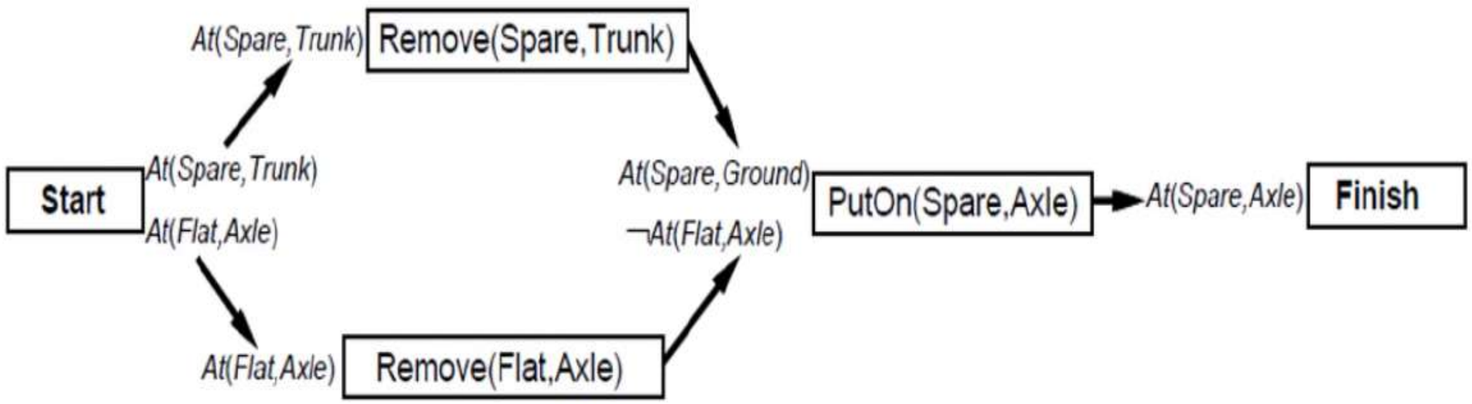


Figure 11.10 The final solution to the tire problem. Note that $Remove(Spare, Trunk)$ and $Remove(Flat, Axle)$ can be done in either order, as long as they are completed before the $PutOn(Spare, Axle)$ action.

PLANNING AND ACTING IN THE REAL WORLD:

- **Time , Schedules and Resources:**

  Time is of the essence in the general family of applications called **job shop scheduling** . Such tasks require completing a set of jobs, each of which consists of a sequence of actions , where each action has a given duration and might require some resources. The problem is to determine a **schedule** that minimizes the total time required to complete all the jobs, while respecting the resource constraints.

PLANNING AND ACTING IN THE REAL WORLD:

- **Example of job shop scheduling problem:**

This is a highly simplified automobile assembly problem. There are two jobs: assembling cars C_1 and C_2.Each job consists of three actions: **adding the engine, adding the wheels, and inspecting the results**. The engine must be put in first (because having the front wheels on would inhibit access to the engine compartment) and of course the inspection must be done last.

PLANNING AND ACTING IN THE REAL WORLD:

$$\begin{aligned}
&Init(Chassis(C_1) \wedge Chassis(C_2) \\
&\quad \wedge Engine(E_1, C_1, 30) \wedge Engine(E_2, C_2, 60) \\
&\quad \wedge Wheels(W_1, C_1, 30) \wedge Wheels(W_2, C_2, 15)) \\
&Goal(Done(C_1) \wedge Done(C_2))
\end{aligned}$$

$$\begin{aligned}
&Action(AddEngine(e, c), \\
&\quad \text{PRECOND: } Engine(e, c, d) \wedge Chassis(c) \wedge \neg EngineIn(c), \\
&\quad \text{EFFECT: } EngineIn(c) \wedge Duration(d)) \\
&Action(AddWheels(w, c), \\
&\quad \text{PRECOND: } Wheels(w, c, d) \wedge Chassis(c) \wedge EngineIn(c), \\
&\quad \text{EFFECT: } WheelsOn(c) \wedge Duration(d)) \\
&Action(Inspect(c), \text{PRECOND: } EngineIn(c) \wedge WheelsOn(c) \wedge Chassis(c), \\
&\quad \text{EFFECT: } Done(c) \wedge Duration(10))
\end{aligned}$$

Figure 12.1 A job shop scheduling problem for assembling two cars. The notation *Duration*(d) means that an action takes d minutes to execute. $Engine(E_1, C_1, 60)$ means that E_1 is an engine that fits into chassis C_1 and takes *60* minutes to install.

PLANNING AND ACTING IN THE REAL WORLD:

- Figure 12.2 shows the solution that the partial-order planner POP would come up with.
- To make this a scheduling problem rather than a planning problem, we must now determine when each action should begin and end, based on the durations of actions as well as their ordering.
- The notation Duration(d) in the effect of an action (where d must be bound to a number) means that the action takes d minutes to complete.

PLANNING AND ACTING IN THE REAL WORLD:

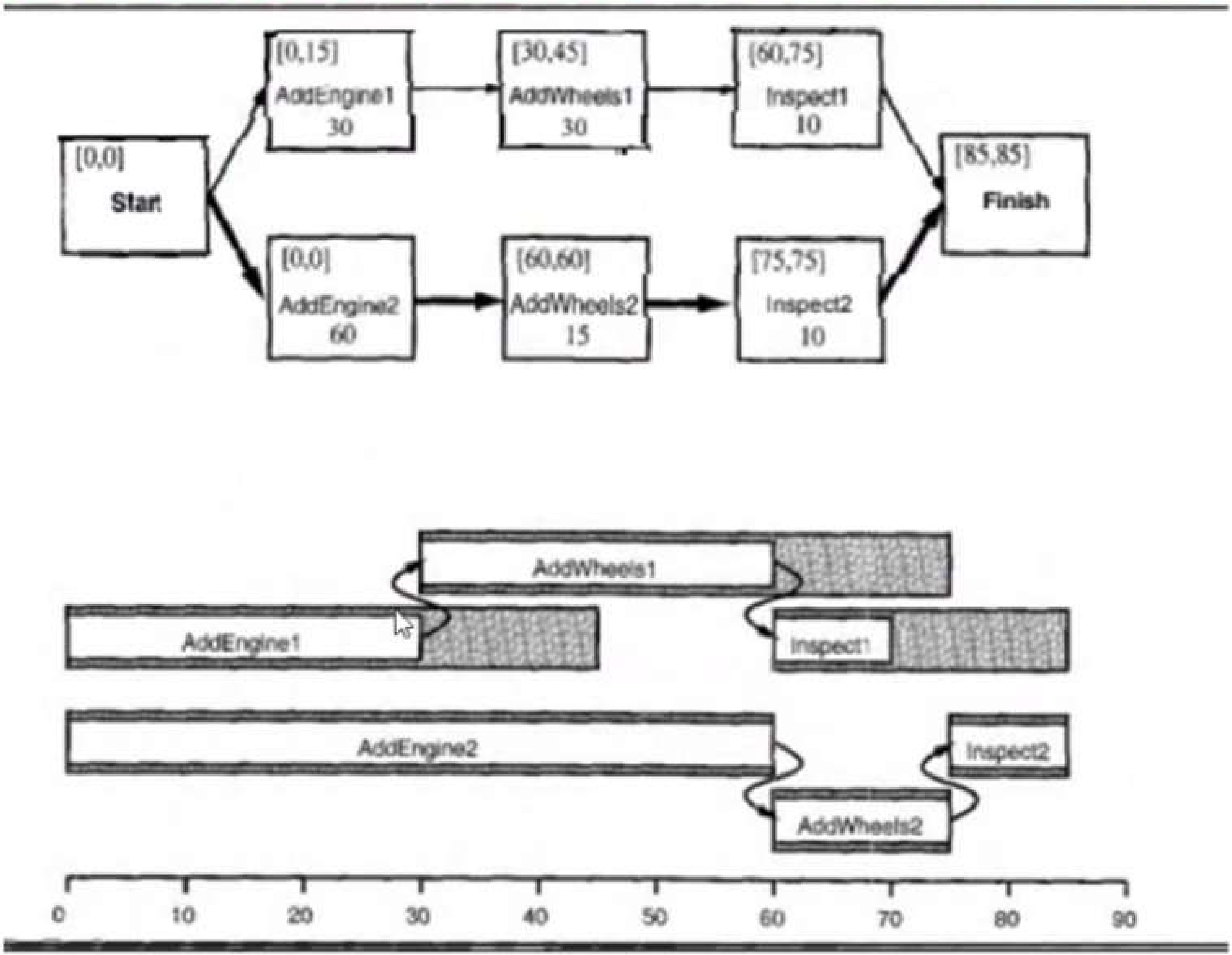


Fig 12.2

PLANNING AND ACTING IN THE REAL WORLD:

- Apply critical path method(CPM) to determine the possible start and end times of each action i.e., it determines the duration of the entire plan.
- In the before figure the critical path is shown with bold lines.
- Slack time is the difference between the earliest possible start time (ES) and latest possible start time(LS).i.e., LS – ES = slack.
- For the before figure ,the whole plan will take 85 minutes .

PLANNING AND ACTING IN THE REAL WORLD:

- **Scheduling with resource constraints:**

> Extending the engine assembly problem by including 3 resources :

1.An engine hoist for installing engines.

2.A wheel station for putting on the wheels

3.Two inspectors

> so,now this solution takes 115 minutes which is longer than time taken by a schedule without resource constraints.

> **Aggregation** groups individual objects into quantities when the objects are all indistinguishable with respect to the purpose at hand. For example, resource Inspectors(2) is represented rather than Inspector(I_1) and Inspector(I_2).

PLANNING AND ACTING IN THE REAL WORLD:

- **Example: (BuildHouse problem)**

Action(*BuyLand*, PRECOND: *Money*, EFFECT: *Land* $\land \neg$ *Money*)
Action(*GetLoan*, PRECOND: *GoodCredit*, EFFECT: *Money* $\land$ *Mortgage*)
Action(*BuildHouse*, PRECOND: *Land*, EFFECT: *House*)

Action(*GetPermit*, PRECOND: *Land*, EFFECT: *Permit*)
Action(*HireBuilder*, EFFECT: *Contract*)
Action(*Construction*, PRECOND: *Permit* $\land$ *Contract*,
EFFECT: *HouseBuilt* $\land \neg$ *Permit*)
Action(*PayBuilder*, PRECOND: *Money* $\land$ *HouseBuilt*,
EFFECT: $\neg$ *Money* $\land$ *House* $\land \neg$ *Contract*)

Decompose(*BuildHouse*,
Plan(STEPS: $\{S_1: GetPermit,\ S_2: HireBuilder,$
$S_3: Construction,\ S_4: PayBuilder\}$
ORDERINGS: $\{Start \prec S_1 \prec S_3 \prec S_4 \prec Finish,\quad Start \prec S_2 \prec S_3\}$,
LINKS: $\{Start \xrightarrow{Land} S_1,\ Start \xrightarrow{Money} S_4,$
$S_1 \xrightarrow{Permit} S_3,\ S_2 \xrightarrow{Contract} S_3,\ S_3 \xrightarrow{HouseBuilt} S_4,$
$S_4 \xrightarrow{House} Finish,\ S_4 \xrightarrow{\neg Money} Finish\}))$

Figure 12.6 Action descriptions for the house-building problem and a detailed decomposition for the *BuildHouse* action. The descriptions adopt a simplified view of money and an optimistic view of builders.

PLANNING AND ACTING IN THE REAL WORLD:

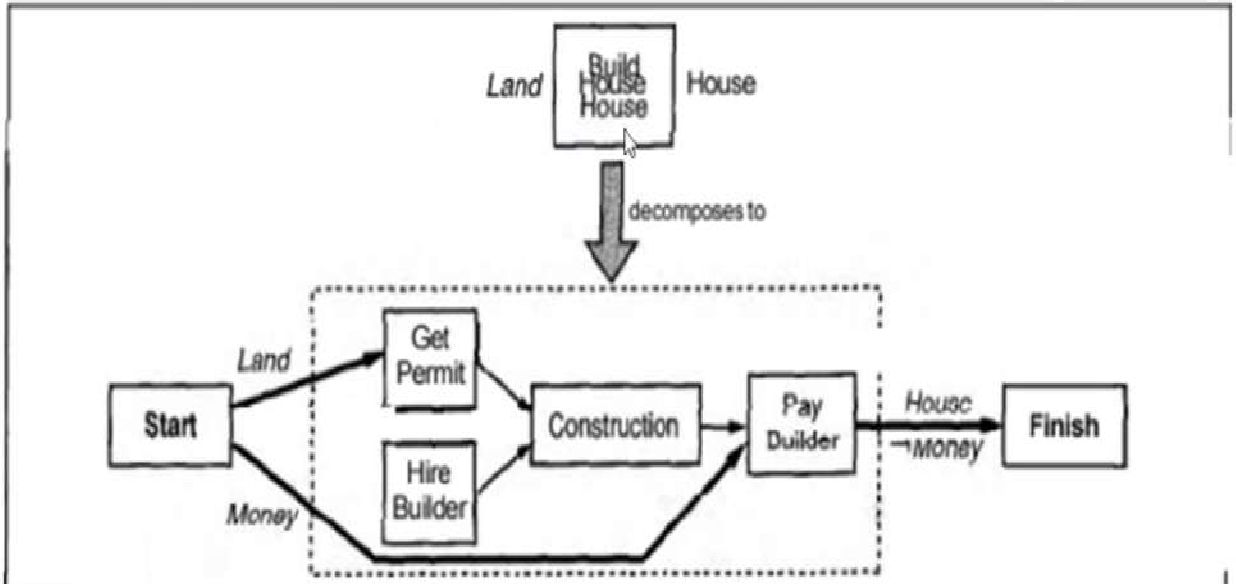


Figure 12.5 One possible decomposition for the *BuildHouse* action.

PLANNING AND ACTING IN THE REAL WORLD:

- **Properties of HTN:**

1.Decomposition should be a correct implementation of an action.

2.A decomposition is not necessarily unique.

3.Performs two other forms of information hiding:

(a).The high-level description completely ignores all internal effects of the decompositions.

(b).The high-level description does not specify the intervals "inside" the activity during which the high-level preconditions are effects must hold.

PLANNING AND ACTING IN THE REAL WORLD:

- **Modifying the planner for decompositions:**

  > For any Decompose(a, d) method from the plan library such that a and a' unify with substitution Θ, replacing a' with d' = SUBST(Θ, d).

  > The decomposition d is selected from Figure 12.5, and BuildHouse is replaced by this decomposition in Figure 12.7

PLANNING AND ACTING IN THE REAL WORLD:

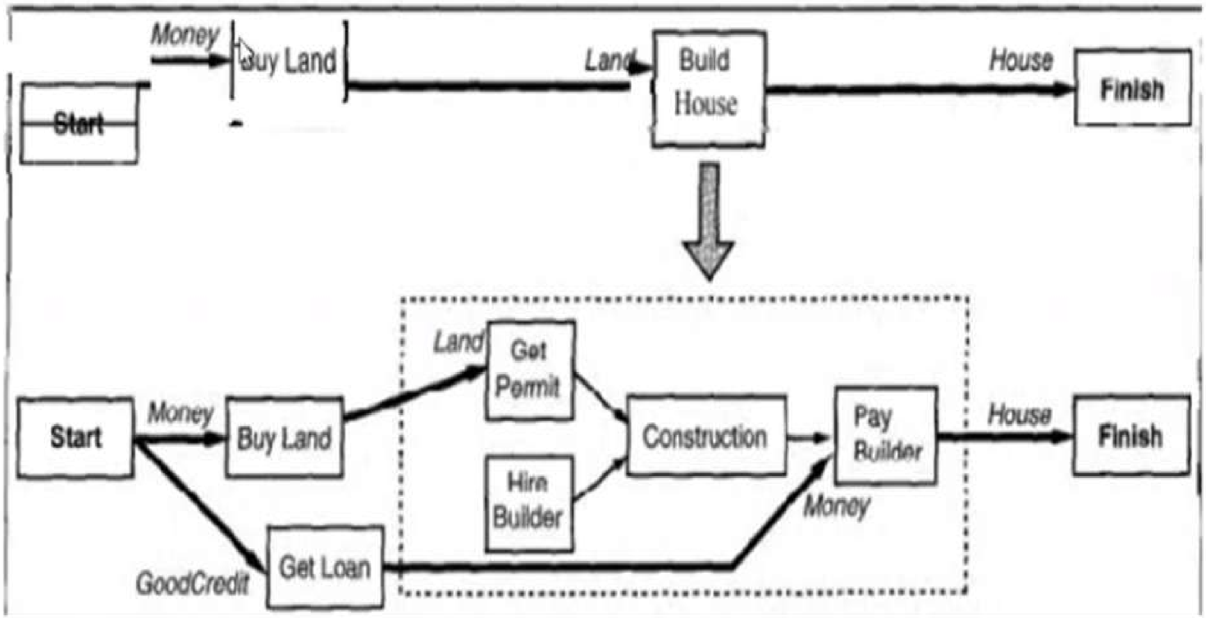


Figure 12.7 Decomposition of a high-level action within an existing plan. The *BuildHouse* action is replaced by the decomposition from Figure 12.5. The external precondition *Land* is supplied by the existing causal link from *BuyLand*. The external precondition *Money* remains open after the decomposition step, so we add a new action, *GetLoan*.

PLANNING AND ACTING IN THE REAL WORLD:

The following steps are performed in decomposition:

1. Implement **subtask sharing** i.e., action a' is removed from P.
2. Hook up the **ordering constraints** for a' in the original plan to the steps in d'.
3. Hook up **causal links**.

PLANNING AND ACTING IN THE REAL WORLD:

- **Bad news of HTN:** pure HTN planning is **undecidable** eventhough the underlined state space is finite due to recursive decomposition actions.

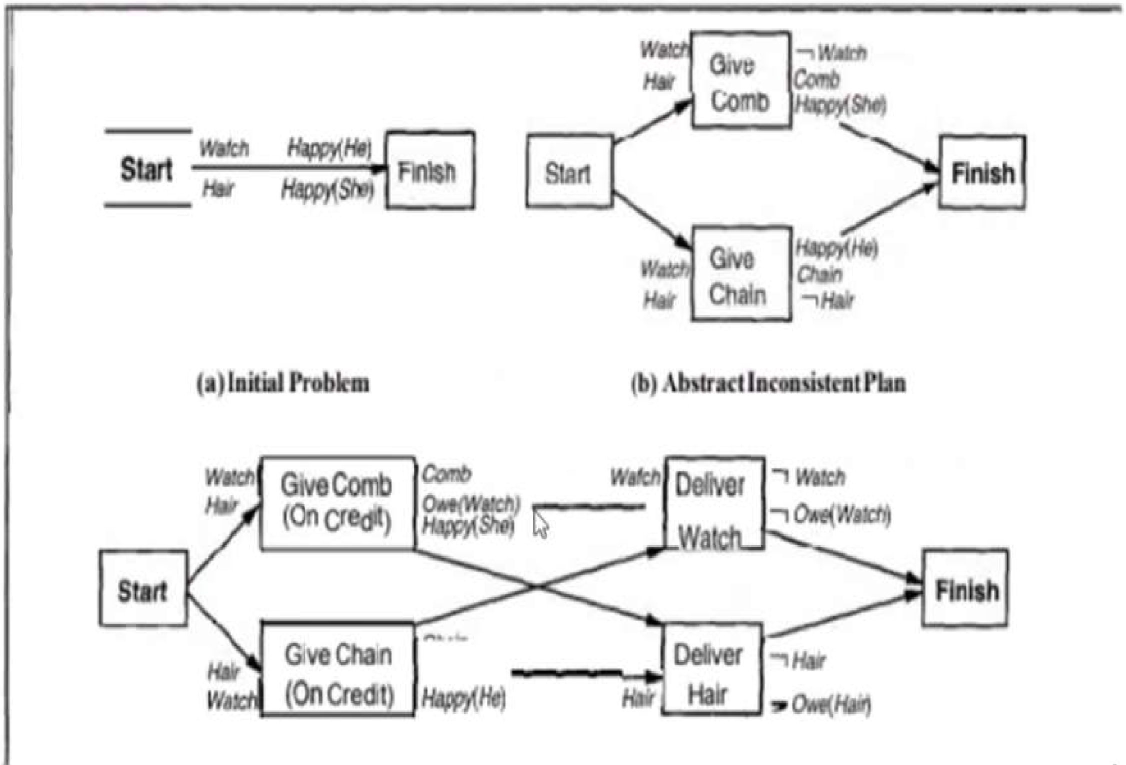
  We can resolve the recursive decomposition problem by 3 ways:

  1. Rule out recursion.
  2. Bound the length of relevant solutions.
  3. Adopt Hybrid approach that combines HTN with POP.

PLANNING AND ACTING IN THE REAL WORLD:

Example for HTN with POP planner:

(The **Gift** of the **Magi** problem)



(a) Initial Problem

(b) Abstract Inconsistent Plan

(c) Decomposition of (b) into a Consistent Solution

PLANNING AND ACTING IN THE REAL WORLD:

- **Explanation:**

  The Gift of the Magi problem,

  > Part (a) shows the problem: A poor couple has only two prized possessions-he a gold watch and she her beautiful long hair. Each plans to buy a present to make the other happy. He decides to trade his watch to buy a silver comb for her hair, and she decides to sell her hair to get a gold chain for his watch.

  > In (b) the partial plan is inconsistent, because there is no way to order the "Give Comb" and "Give Chain"abstract steps without a conflict. (We assume that the "Give Comb" action has the precondition Hair, because if the wife doesn't have her long hair, the action won't have the intended effect of making her happy, and similarly for the "Give Chain" action.)

PLANNING AND ACTING IN THE REAL WORLD:

> In (c) we decompose the "Give Comb" step with an "installment plan" method. In the first step of the decomposition, the husband takes possession of the comb and gives it to his wife, while agreeing to deliver the watch in payment at a later date.In the second step, the watch is handed over and the obligation is fulfilled. A similar method decomposes the "Give Chain" step. As long as both giving steps are ordered before the delivery steps, this decomposition solves the problem.

PLANNING AND ACTING IN THE REAL WORLD:

- **Planning and Acting in Nondeterministic Domains:**

- The classical planning domains (or) deterministic domains are fully observable & static . In it action descriptions are correct and complete . An agent can plan first and then executes the plan with its eyes closed.
- In nondeterministic domains , agents have to deal with incomplete and incorrect information.

  >Incompleteness is due to partially observable, nondeterministic or both.

  >Incorrectness is due to mismatch between real world model and actual model

PLANNING AND ACTING IN THE REAL WORLD:

- The degree of indeterminism is measured with either bounded domains or unbounded domains.
- To handle the indeterminism there are two indeterminacy bounded planning methods & two indeterminacy unbounded planning methods.
- Bounded planning methods:

  1.Sensorless planning.

  2.Conditional planning.
- Unbounded planning methods:

  1.Execution Monitoring and Re-planning.

  2.Continuous planning.

PLANNING AND ACTING IN THE REAL WORLD:

- Consider an example to clarify the differences among the various kinds of agents:The problem is,

Given an initial state with a chair,a table and some cans of paint with everything of unknown color,achieve the state where the chair and table have same color.

PLANNING AND ACTING IN THE REAL WORLD:

- As per **sensorless planning agent**,the solution is to open any can of paint and apply it to both chair and table ,thus coercing them to be the same color(even though the agent doesn't know what the color is.)
- As per **conditional planning agent**,first sense the color of the chair and table,if they are already the same then the plan is done.If not , sense the labels of the paint cans,if there is a can that is the same color as one of the furniture,then apply the paint to the other piece . Otherwise paint both pieces with any color.

PLANNING AND ACTING IN THE REAL WORLD:

- A **replanning agent** could generate the same plan as the conditional planner, or it could generate fewer branches at first and fill in the others at execution time as needed. A conditional planner would just assume that the effect has occurred once the action has been executed, but a replanning agent would check for the effect, and if it were not true (perhaps because the agent was careless and missed a spot), it could then replan to repaint the spot.
- A **continuous planning agent**, in addition to handling unexpected events, can revise its plans appropriately if, say, we add the goal of having dinner on the table, so that the painting plan must be postponed.

PLANNING AND ACTING IN THE REAL WORLD:

- **Sensorless planning:**

  Also called **conformant planning**. The sensorless planning algorithm must ensure that the plan achieves the goal in all possible circumstances , regardless of the true initial state and the actual action outcomes.Relies on coercion.

- **Example:** Vacuum world problem

PLANNING AND ACTING IN THE REAL WORLD:

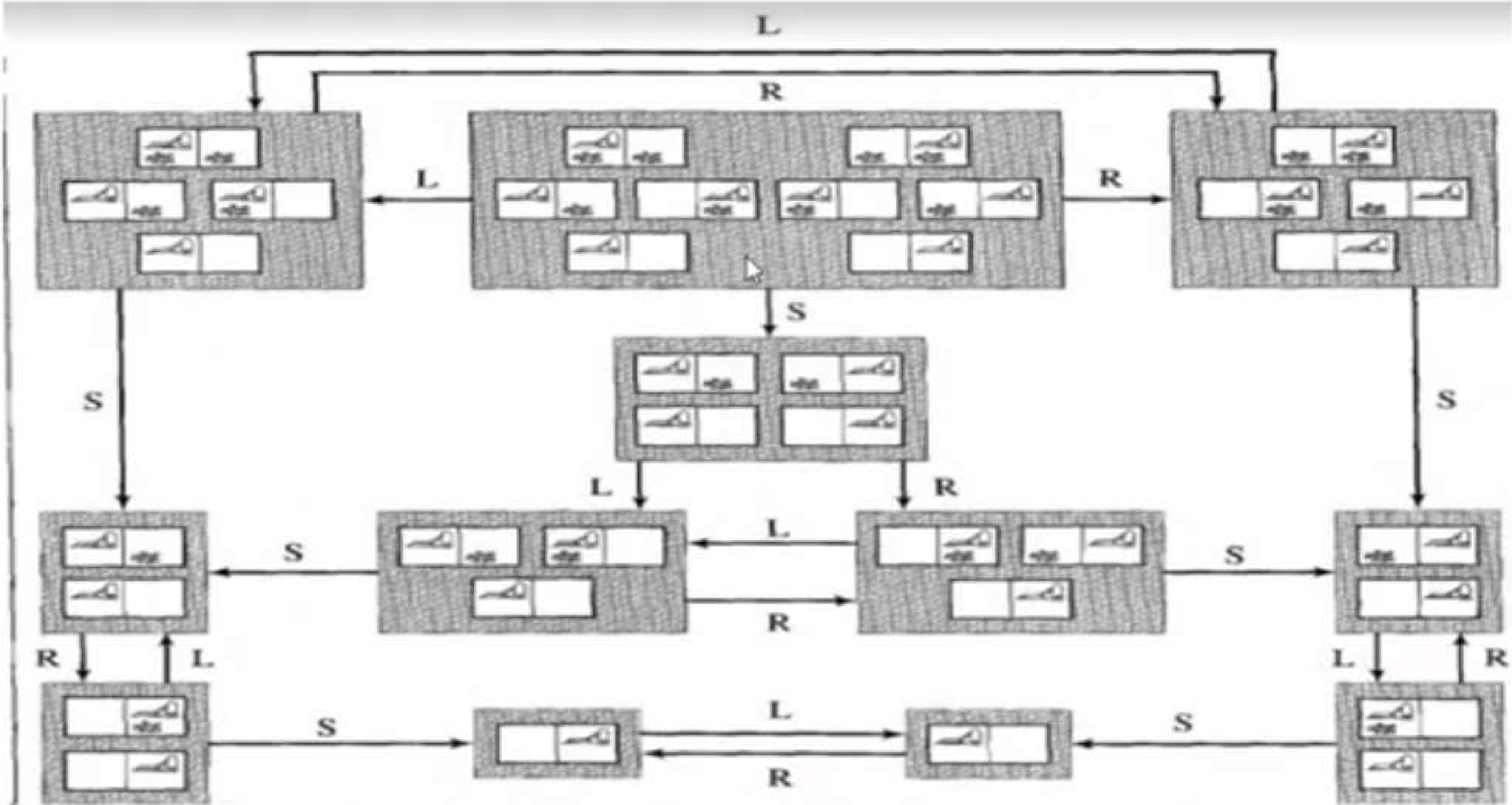


Figure 3.21 The reachable portion of the belief state space for the deterministic, sensorless vacuum world. Each shaded box corresponds to a single belief state. At any given point, the agent is in a particular belief state but does not know which physical state it is in. The initial belief state (complete ignorance) is the top center box. Actions are represented by labeled arcs. Self-loops are omitted for clarity.

PLANNING AND ACTING IN THE REAL WORLD:

- **Conditional planning:**

- Also called **contigency planning.**It deals with uncertainity by checking what is actually happening in the environment at predetermined points in the plan.
- Constructs conditional plan steps with different branches for possible contingencies.

1.conditional planning in fully observable environments.

2.conditional planning in partially observable environments.

PLANNING AND ACTING IN THE REAL WORLD:

- **Conditional planning in fully observable environments:**

  * Full observability means that the agent always knows the current state.A conditional planning agent handles nondeterminism by building into the plan conditional steps that will check the state of the environment to decide what to do next.The problem is how to construct these conditional plans.

  1.Include actions having disjunctive effects

  Action(Left, PRECOND:AtR,EFFECT:AtL V AtR)

PLANNING AND ACTING IN THE REAL WORLD:

- **Example**:Let us consider a specific example in the vacuum world. The initial state has the robot in the right square of a clean world; because the environment is fully observable, the agent knows the full state description, AtR ^CleanL ^ CleanR. The goal state has the robot in the left square of a clean world. “Double Murphy"vacuum cleaner sometimes deposits dirt when it moves to a clean destination square and sometimes deposits dirt if Suck is applied to a clean square.

PLANNING AND ACTING IN THE REAL WORLD:

* To create conditional plans,we need conditional steps.

  syntax:"if <test> then plan_A else plan_B."

  ex:-"if AtL ^ CleanL then Right else Suck."

* **Games against nature** tells to find conditional plans that work regardless of which action outcomes actually occur.

PLANNING AND ACTING IN THE REAL WORLD:

- **Example**:Let us consider a specific example in the vacuum world. The initial state has the robot in the right square of a clean world; because the environment is fully observable, the agent knows the full state description, AtR ^CleanL ^ CleanR. The goal state has the robot in the left square of a clean world. “Double Murphy"vacuum cleaner sometimes deposits dirt when it moves to a clean destination square and sometimes deposits dirt if Suck is applied to a clean square.

PLANNING AND ACTING IN THE REAL WORLD:

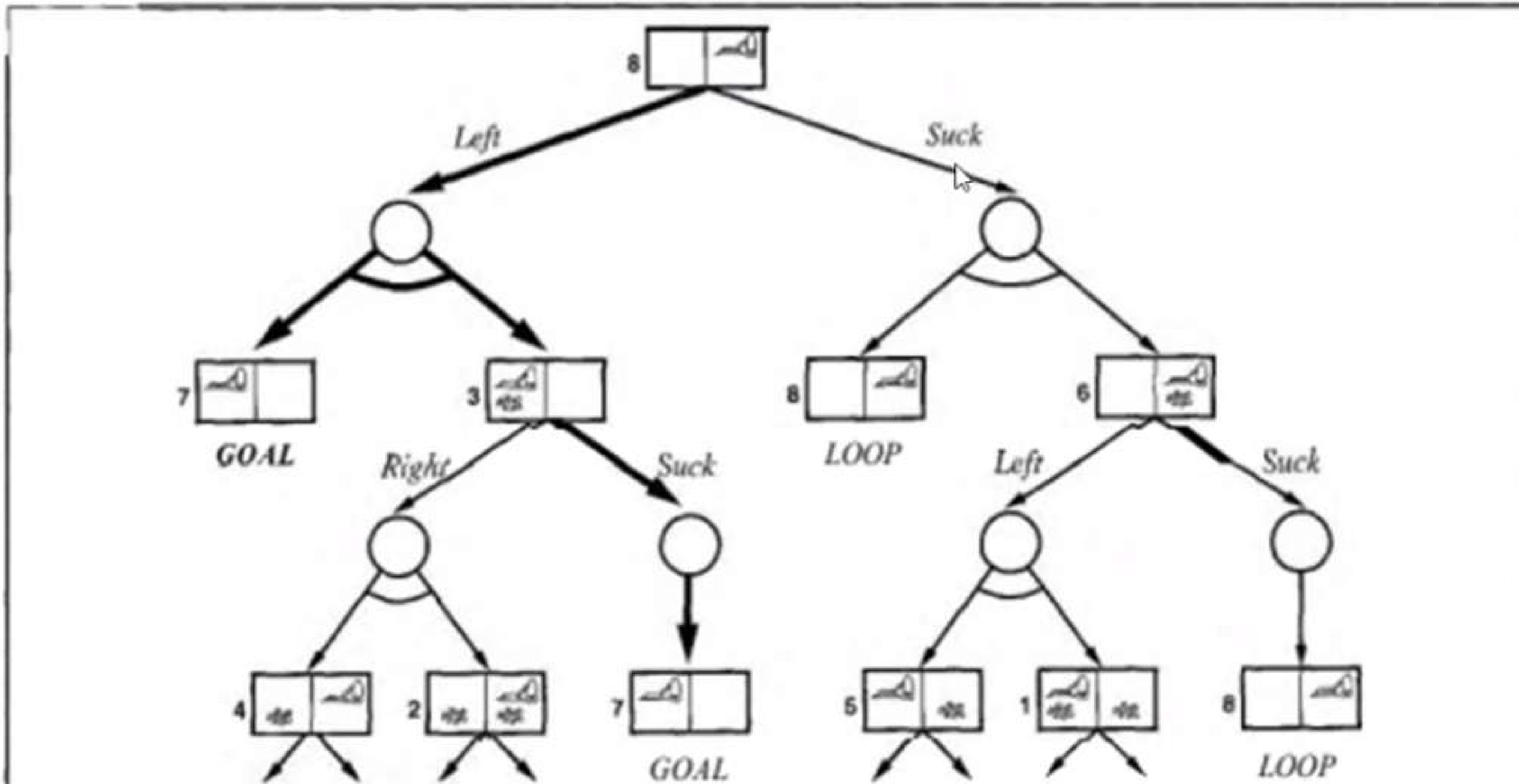


Figure 12.9 The first two levels of the search tree for the "double Murphy" vacuum world. State nodes are OR nodes where some action must be chosen. Chance nodes, shown as circles, are AND nodes where every outcome must be handled, as indicated by the arc linking the outgoing branches. The solution is shown in bold lines.

PLANNING AND ACTING IN THE REAL WORLD:

- A solution is a subtree that

  (1) has a goal node at every leaf,

  (2) specifies one action at each of its "state" nodes, and

  (3) includes every outcome branch at each of its "chance" nodes.

  The solution is shown in bold lines in the figure;
- Finally the search space is defined by AND-OR graph search.
- The double murphy AND-OR graph algorithm is a recursive DFS algorithm i.e.,if the current state is identical to a state on the path from the root then it returns with failure.

PLANNING AND ACTING IN THE REAL WORLD:

function AND-OR-GRAPH-SEARCH(*problem*) **returns** *a conditional plan, or failure*
  OR-SEARCH(INITIAL-STATE[*problem*], *problem*, [ ])

function OR-SEARCH(*state, problem, path*) **returns** *a conditional plan, or failure*
  if GOAL-TEST[*problem*](*state*) **then return** the empty plan
  if *state* is on *path* **then return** *failure*
  for each *action, state-set* **in** SUCCESSORS[*problem*](*state*) **do**
    plan ← AND-SEARCH(*state_set, problem, [state | path]*)
    if *plan* ≠ *failure* **then return** *[action | plan]*
  return *failure*

function AND-SEARCH(*state_set, problem, path*) **returns** *a conditional plan, or failure*
  for each s_i **in** *state-set* **do**
    $plan_i$ ← OR-SEARCH(s_i, *problem, path*)
    if *plan* = *failure* **then return** *failure*
  return [**if** s_1 **then** *plan*, **else if** s_2 **then** *plan*, **else** . . . **if** s_{n-1} **then** *plan*,–, **else** *plan*,]

Figure 12.10 An algorithm for searching AND–OR graphs generated *by* nondeterministic environments. We assume that *SUCCESSORS* returns a list of actions, each associated with a set of possible outcomes. The aim is to find a conditional plain that reaches a goal state in all circumstances.

PLANNING AND ACTING IN THE REAL WORLD:

- The triple murphy algorithm states that there are no longer any cyclic solutions and AND-OR-GRAPH-SEARCH would return with failure.
- It gives a cyclic solution by adding a label to denote some portion of the plan and using that label later instead of repeating the plan itself.Thus,our cyclic solution is,

$[L_1 : \text{Left, if AtR then } L_1 \text{ else if CleanL then [] else Suck}]$

PLANNING AND ACTING IN THE REAL WORLD:

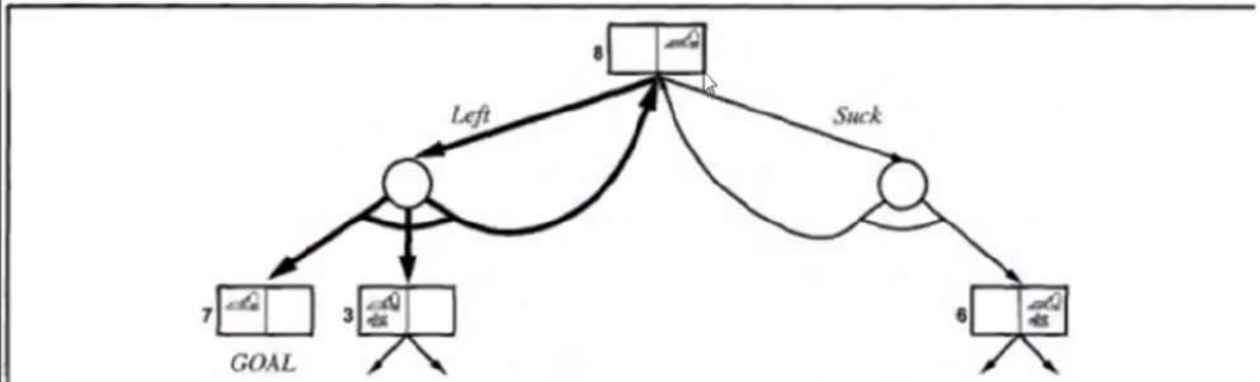


Figure 12.11 The first level of the search graph for the "triple Murphy" vacuum world, where we have shown cycles explicitly. All solutions for this problem are cyclic plans.

PLANNING AND ACTING IN THE REAL WORLD:

- **Conditional planning in partially observable environments:**

  * The agent knows only a certain amount about the actual state.This situation can be modelled by considering the initial state belongs to **state set** or **belief state**.

  *Suppose that a vacuum-world agent knows that it is in the right-hand square and that the square is clean, but it cannot sense the presence or absence of dirt in other squares. Then as far as it knows it could be in one of two states: the left-hand square might be either clean or dirty. This belief state is marked A in Figure 12.12.

  *The figure shows part of the AND-OR graph for the "alternate double Murphy" vacuum world, in which dirt can sometimes be left behind when the agent leaves a clean square.

PLANNING AND ACTING IN THE REAL WORLD:

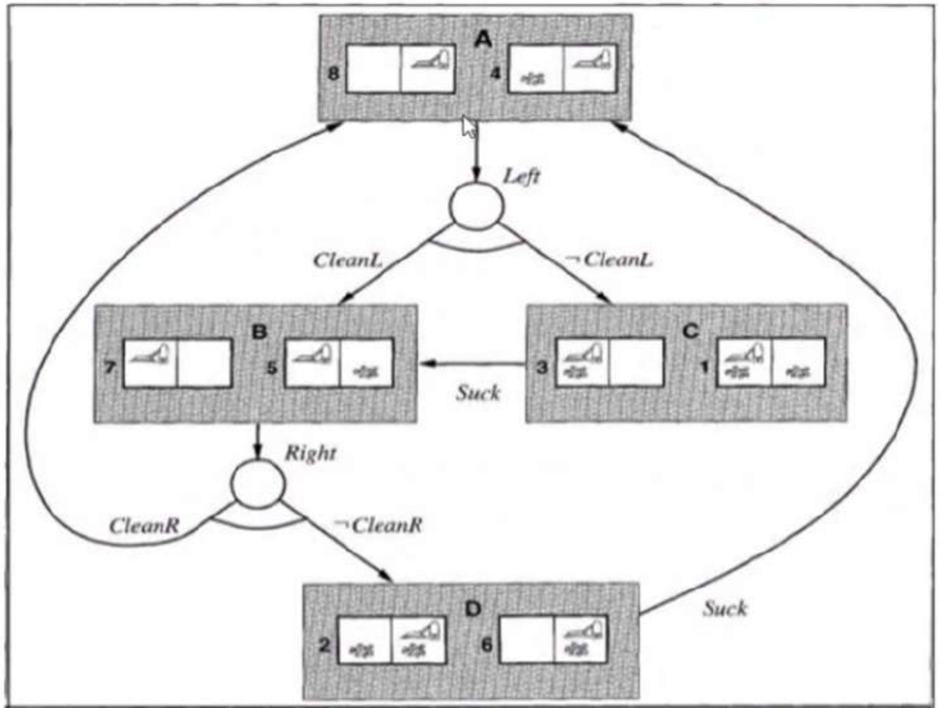


Figure 12.12 Part of the AND–OR graph for the "alternate double Murphy" vacuum world, in which dirt can sometimes be left behind when the agent leaves a clean square. The agent cannot sense dirt in other squares.

PLANNING AND ACTING IN THE REAL WORLD:

- ## Execution Monitoring and Replanning:

An execution monitoring agent checks its percepts to see whether everything is going according to the plan or not.If any unanticipated circumstances raises for which agent's action descriptions are incorrect,that problem is called as unbounded indeterminacy.There are 2 kinds of execution monitoring:

1. **Action monitoring**, whereby the agent checks the environment to verify that the next action will work, and
2. **Plan monitoring**, in which the agent verifies the entire remaining plan.

PLANNING AND ACTING IN THE REAL WORLD:

- A **replanning** agent repairs the old plan when something unexpected will happen.
- **Execution monitoring and Replanning** combinedly can be applied to both full & partially observable environments and to a state space,POP and conditional planning problems.
- **Example:**problem of achieving a chair and table of matching color, via replanning. The initial state the chair is blue, the table is green, and there is a can of blue paint and a can of red paint.

PLANNING AND ACTING IN THE REAL WORLD:

- The problem definition is:

$$
\begin{aligned}
&Init(Color(Chair, Blue) \wedge Color(Table, Green) \\
&\quad \wedge ContainsColor(BC, Blue) \wedge PaintCan(BC)) \\
&\quad \wedge ContainsColor(RC, Red) \wedge PaintCan(RC) \\
&Goal(Color(Chair, x) \wedge Color(Table, x)) \\
&Action(Paint(object, color), \\
&\quad \text{PRECOND:} HavePaint(color) \\
&\quad \text{EFFECT:} Color(object, color)) \\
&Action(Open(can), \\
&\quad \text{PRECOND:} PaintCan(can) \wedge ContainsColor(can, color \\
&\quad \text{EFFECT:} HavePaint(color)
\end{aligned}
$$

The agent's *PLANNER* should come up with the following plan:

$$[Start; Open(BC); Paint(Table, Blue); Finish]$$

PLANNING AND ACTING IN THE REAL WORLD:

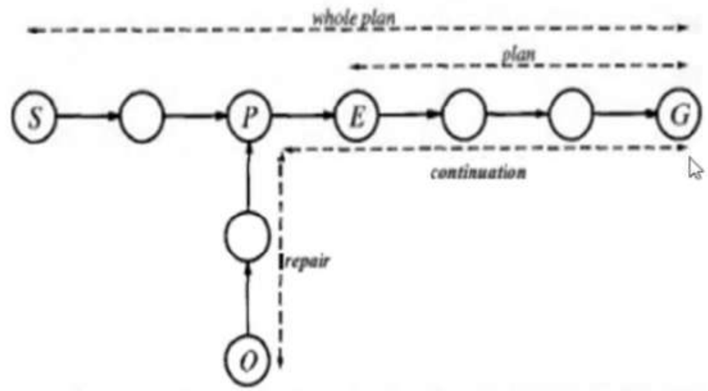


Figure 12.14 Before execution, the planner comes up with a plan, here called *whole-plan*, to get from S to G. The agent executes the plan until the point marked E. Before executing the remaining *plan*, it checks preconditions as usual and finds that it is actually in state O rather than state E. It then calls its planning algorithm to come up with *repair*, which is a plan to get from O to some point **P** on the original *whole-plan*. The new *plan* now becomes the concatenation of *repair* and *continuation* (the resumption of the original *whole-plan*).

PLANNING AND ACTING IN THE REAL WORLD:

- There are some complications in replanning for partially observable environments.They are:

  1.Things can go long without the agent's being able to detect it.

  2.checking preconditions could require the execution of sensing actions.

  The solutions for the above problems are:

  1.Choose one of the repair plan randomly from among the set of all possible repair plans.

  2.Use learning process for avoiding incorrect action descriptions.

PLANNING AND ACTING IN THE REAL WORLD:

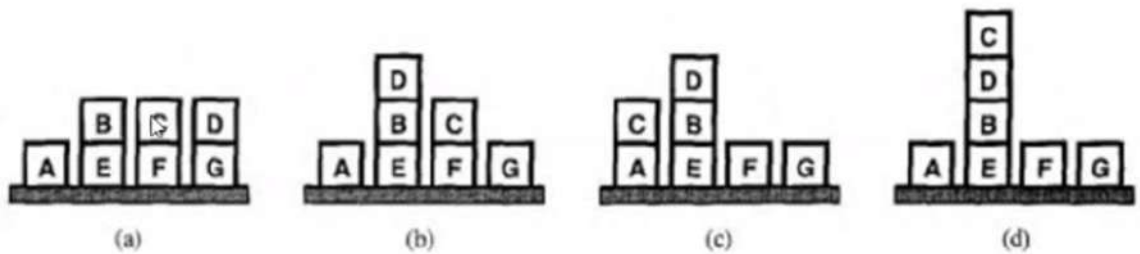


Figure 12.15 The sequence of states as the continuous planning agent tries to reach the goal state $On(C, \mathrm{D}) \wedge On(D, \mathrm{B})$, as shown in (d). The start state is (a). At (b), another agent has interfered, putting D on B. At (c), the agent has executed $Move(C, \mathrm{D})$ but has failed, dropping C on A instead. It retries $Move(C, \mathrm{D})$, reaching the goal state (d).

- Continuous planning agent builds the plan incrementally. The preconditions and ordering constraints to reach our goal state is shown in following figures:

PLANNING AND ACTING IN THE REAL WORLD:

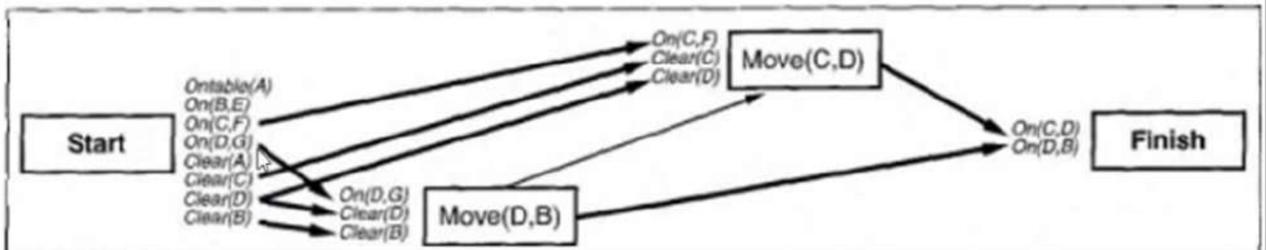


Figure 12.16 The initial plan constructed by the continuous planning agent. The plan **is** indistinguishable, so far, from that produced by a normal partial-order planner.

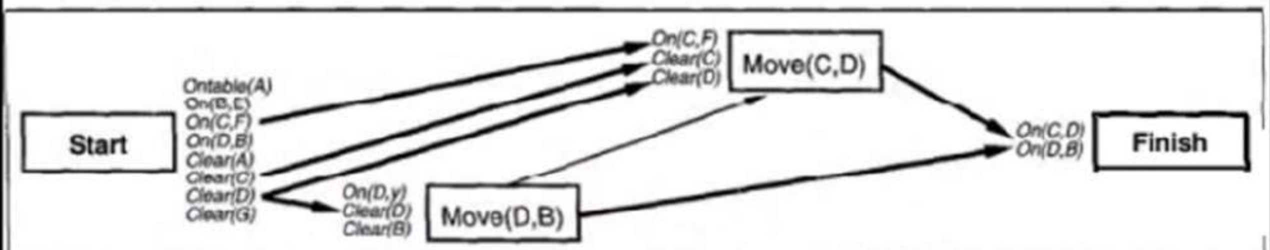


Figure 12.17 After someone else moves D onto B, the unsupported links supplying $Clear(B)$ and $On(D,G)$ are dropped, producing this plan.

PLANNING AND ACTING IN THE REAL WORLD:

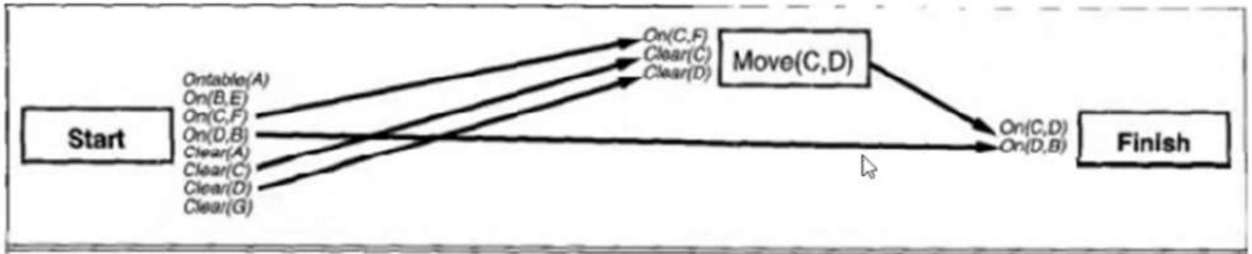


Figure 12.18 The link supplied by $Move(D, \mathrm{B})$ has been replaced by one from *Start*, and the now-redundant step $Move(D, \mathrm{B})$ has been dropped.

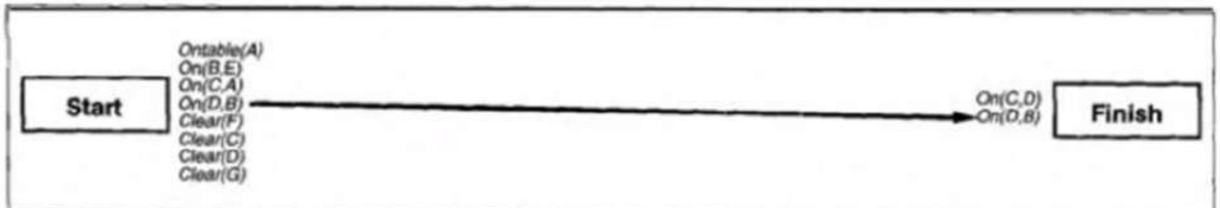


Figure 12.19 After $Move(C, \mathrm{D})$ is executed and removed from the plan, the effects of the *Start* step reflect the fact that C ended up on A instead of the intended D. The goal precondition $On(C, D)$ is still open.

PLANNING AND ACTING IN THE REAL WORLD:

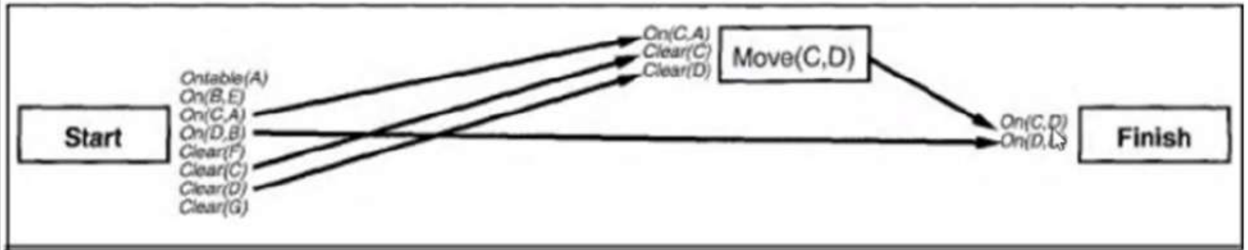


Figure 12.20 The open condition is resolved by adding $Move(C, \mathrm{D})$ back in. Notice the new bindings for the preconditions.

Figure 12.21 After $Move(C, D)$ is executed and dropped from the plan, the remaining open condition $On(C, \mathrm{D})$ is resolved by adding a causal link from the new *Start* step. The plan is now completed.

PLANNING AND ACTING IN THE REAL WORLD:

- The continuous planning agent addresses the following flaws:
  - *Missing goal:* The agent can decide to add a new goal or goals to the *Finish* state. (Under continuous planning, it might make more sense to change the name of *Finish* to *Infinity*, and of *Start* to *Current*, but we will stick with tradition.)
  - *Open precondition:* Add a causal link to an open precondition, choosing either a new or an existing action (as in POP).
  - *Causal Conflict:* Given a causal link A $\xrightarrow{p}$ B and an action C with effect $\neg p$, choose an ordering constraint or variable constraint to resolve the conflict (as in POP).
  - *Unsupported link:* If there is a causal link Start $\xrightarrow{p}$ A where p is no longer true in Start, then remove the link. (This prevents us from executing an action whose preconditions are false.)
  - *Redundant action:* If an action A supplies no causal links, remove it and its links. (This allows us to take advantage of serendipitous events.)
  - *Unexecuted action:* If an action A (other than Finish) has its preconditions satisfied in Start, has no other actions (besides Start) ordered before it, and conflicts with no causal links, then remove A and its causal links artd return it as the action to be executed.
  - *Unnecessary historical goal:* If there are no open preconditions and no actions in the plan (so that all causal links go directly from Start to Finish), then we have achieved the current goal set. Remove the goals and the links to them to allow for new goals.

PLANNING AND ACTING IN THE REAL WORLD:

```
function CONTINUOUS-POP-AGENT(percept) returns an action
  static: plan, a plan, initially with just Start, Finish

  action ← NoOp (the default)
  EFFECTS[Start] = UPDATE(EFFECTS[Start], percept)
  REMOVE-FLAW(plan) //possibly updating action
  return action
```

Figure 12.22 CONTINUOUS-POP-AGENT, a continuous partial-order planning agent. After receiving a percept, the agent removes a flaw from its constantly updated plan and then returns an action. Often it will take many steps of flaw-removal planning, during which it returns *NoOp*, before it is ready to take a real action.

PLANNING AND ACTING IN THE REAL WORLD:

- **Multiagent planning:**
- So far we discussed only single-agent environments. There may be other agents in the environment. Adding other agents into the environment leads to poor performance.
- Generally, there are two types of multiagent environments:

    1.Cooperative

    2.Competitive

- **Cooperation:** Joint goals and plans

    It is described as the act of working together for achieving a common goal.

PLANNING AND ACTING IN THE REAL WORLD:

$Agents(A, B)$
$Init(At(A, [Left, Baseline]) \land At(B, [Right, Net]) \land$
$\quad Approaching(Ball, [Right, Baseline])) \land Partner(A, B) \land Partner(B, A)$
$Goal(Returned(Ball) \land At(agent, [x, Net]))$
$Action(Hit(agent, Ball),$
$\quad \text{PRECOND:} Approaching(Ball, [x, y]) \land At(agent, [x, y]) \land$
$\quad\quad Partner(agent, partner) \land \neg At(partner, [x, y])$
$\quad \text{EFFECT:} Returned(Ball))$
$Action(Go(agent, [x, y]),$
$\quad \text{PRECOND:} At(agent, [a, b]),$
$\quad \text{EFFECT:} At(agent, [x, y]) \land \neg At(agent, [a, b]))$

Figure 12.23 The doubles tennis problem. Two agents are playing together and can be in one of four locations: *[Left,Baseline]*, *[Right,Baseline]*, *[Left,Net]*, and *[Right,Net]*. The ball can be returned if exactly one player is in the right place.

PLANNING AND ACTING IN THE REAL WORLD:

- The solution for the double tennis problem is a joint-plan consisting of actions for both agents:

plan 1: A : [Go(A, [Right, Baseline]), Hit(A, Ball)]

B : [NoOp(B), NoOp(B)].

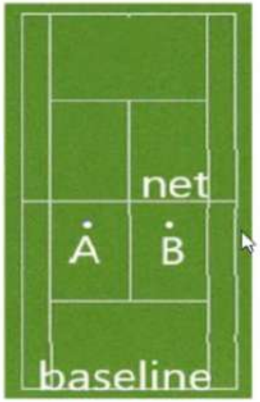
plan 2: A : [Go(A, [Left, Net]), NoOp(A)]

B : [Go(B, [Right, baseline]), Hit(B, Ball)]

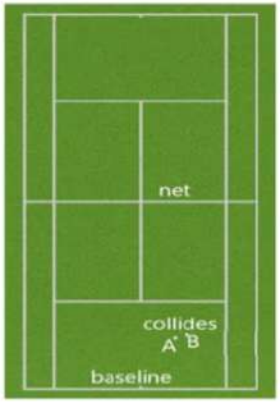
PLANNING AND ACTING IN THE REAL WORLD:

- If there is only one plan then everything would be fine . Here we have 2 plans. If A chooses plan2 and B chooses plan1,then nobody will return the ball . conversely , if A chooses plan1 and B chooses plan2, then they will collide with each other;no one returns the ball and the net may remain uncovered.

1st case



2nd case



PLANNING AND ACTING IN THE REAL WORLD:

- **Note**: Correct joint plans does not mean that goal will be achieved . There should be a same joint plan. This can be achieved by coordination.
- **Coordination** is a systematic arrangement of various elements of management so as to ensure smooth functionality.

PLANNING AND ACTING IN THE REAL WORLD:

- In coordination , the actions are synchronized i.e., performing two actions concurrently . This set of concurrent actions is called a **joint plan**.

  (Go(A, [Left, Net]), Go(B, [Right, baseline]))

  (NoOp(A), Hit(B, Ball))

Coordination mechanisms:

*The simplest method by which a group of agents can ensure agreement on a joint plan is to adapt a convention prior to engaging in joint activity.

*A convention is any constraint on the selection of joint plans.

PLANNING AND ACTING IN THE REAL WORLD:

- For eg., in double tennis problem , we can include some constraints such as,

  (i).stick to your side of the court-selects plan2.

  (ii).one player always stays at the net-selects plan1

- In the absence of applicable convention , agents can use **communication**. i.e., If the ball is at equal distance between the two partners , then one player could shout **"Mine!"** or **"Yours!"** to indicate a preferred joint plan. Or
- One player can communicate preferred joint plan by executing the first part of it. i.e.,if agent A heads for the net , then agent B is obliged to go back to baseline to hit the ball . This is called **plan recognition**.

PLANNING AND ACTING IN THE REAL WORLD:

- These conventions are domain-specific . There are some conventions , that are domain-independent. For this , consider the flocking behavior of birds:
- There are 3 rules executed by each bird agent:

1.**Separation**: Steer away from neighbors when you start to get too close.

2. **Cohesion:** Steer towards the average position of the neighbors.

3. **Alignment:** Steer towards the average orientation (heading) of the neighbors.

PLANNING AND ACTING IN THE REAL WORLD:

- **Competition:**

Not all multiagent environments involve cooperative agents . Agents with conflicting utility functions are in competition with each other.

Example: Two-player zero-sum game(chess)

Here , a chess playing agent needs to consider the opponents possible moves for several steps into the future. That is , an agent in a competitive environment must

(a) recognize that there are other agents,

(b) compute some of the other agent's possible plans,

(c) compute how the other agent's plans interact with its own plans , and

(d) decide on the best action in view of these interactions.

PLANNING AND ACTING IN THE REAL WORLD:

- Like coperation,competition requires a model of the other agent's plans but there is no commitment to a joint plan in a competitive environment.
- The conditional planning algorithm constructs plans that work under worst-case assumptions about the environment, so it can be applied in competitive situations where the agent is concerned only with success and failure.

Planning Graph

- It is an algorithm for automated planning, developed by Avrim and Merrick in 1995.
- The Graph Plan's input is planning problem, expressed in STRIPS and produces a sequence of operations for reaching a goal state.

Planning Graph...

- Convert the planning problem structure into planning graph called as GRAPHPLAN, in the increment nature.
- It gives the relation between action and states, the precondition must be satisfy the action.
- The Planning graph is a layered graph, with alternate layers of propositions and actions.
  - Layer p0
  - Layer a1
  - Layer p1

Planning Graph...

- Propositional problem will look at, what the starting state, what the objects in the domains are, and it will produce all the possible actions, and works with those actions.
- We construct the planning graph from left to right,
  - we keep inserting actions and propositions, and
  - then actions and propositions
  - until we get the goal proposition appear on the proposition layer, and
  - they are not mutually exclusive.

Planning Graph...

- There are two states in the planning graph problem
  - Construct the planning graph
  - Search for solution
- If you cannot get solution, then extend the planning graph and search for solution, and keep doing that until you get the solution.

PLANNING GRAPHS

- Planning Graph can give better heuristic estimates.
- Here we can extract a solution directly from the planning graph, using a specialized algorithm such as GRAPHPLAN
- A planning graph consists of a sequence of **levels** that correspond to time steps in the plan, where level 0 is the initial state.
- Each level contains a set of literals and a set of actions.
- The literals are true at that time step, depending on, the actions executed at preceding time steps.
- Actions *could* have their preconditions, that should be satisfied at that time step, depending on the literals actually hold.

In short

- Planning graphs are an efficient way to create a representation of a planning problem, that can be used to
  - Achieve better heuristic estimates
  - Directly construct plans
- Planning graphs only work for propositional problems.

Planning graphs

- It consists of a seq of levels that correspond to time steps in the plan.
  - Level 0 is the initial state.
  - Each level consists of a set of literals and a set of actions that represent, what might be possible at that step in the plan
  - Records only a restricted subset of possible negative interactions among actions.

Planning Graph

- Each level consists of
  - Literals = all those that could be true at that time step, depending upon the actions executed at preceding time steps.
  - Actions = all those actions have their preconditions, that satisfied at that time step, depending on which of the literals actually hold.

Example - The "have cake and eat cake too" problem.

$$Init(Have(Cake))$$
$$Goal(Have(Cake) \wedge Eaten(Cake))$$
$$Action(Eat(Cake)$$
$$\quad \text{PRECOND: } Have(Cake)$$
$$\quad \text{EFFECT: } \neg\, Have(Cake) \wedge Eaten(Cake))$$
$$Action(Bake(Cake)$$
$$\quad \text{PRECOND: } \neg\, Have(Cake)$$
$$\quad \text{EFFECT: } Have(Cake)$$

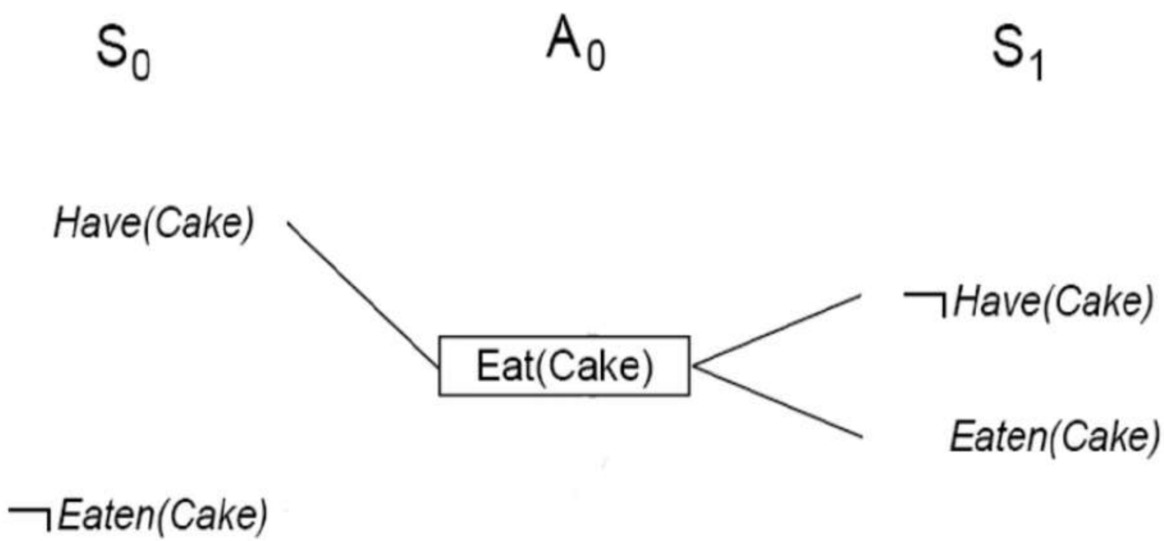
PG – example

S_0 A_0 S_1

Have(Cake)

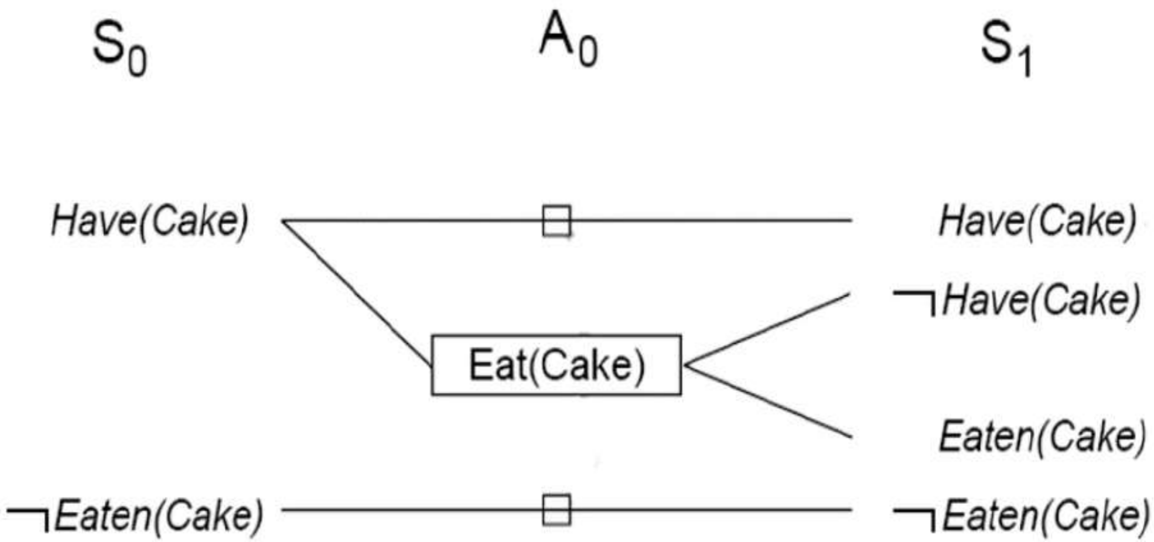
$\neg$*Eaten(Cake)*

Create level 0 from initial problem state.

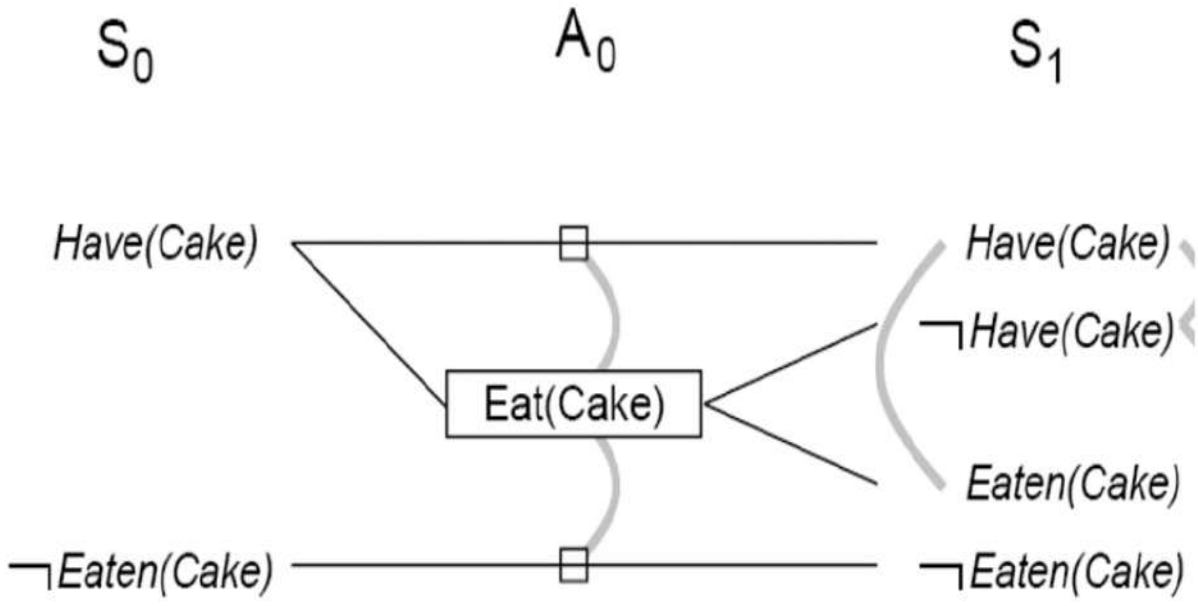


Add all applicable actions.

Add all effects to the next state.



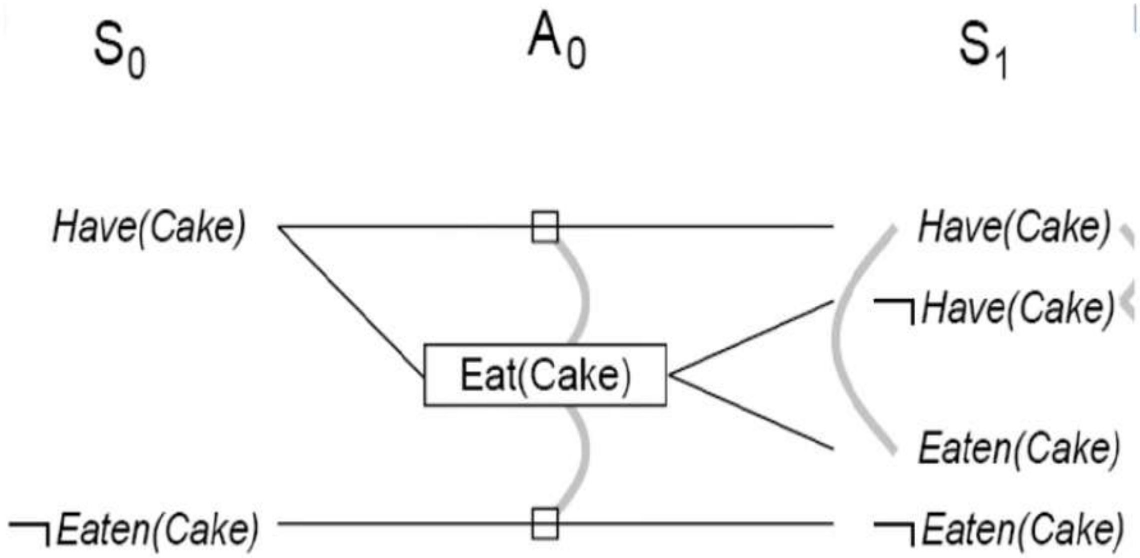
Add *persistence actions* (inaction = no-ops) to map all literals in state S_i to state S_{i+1}.



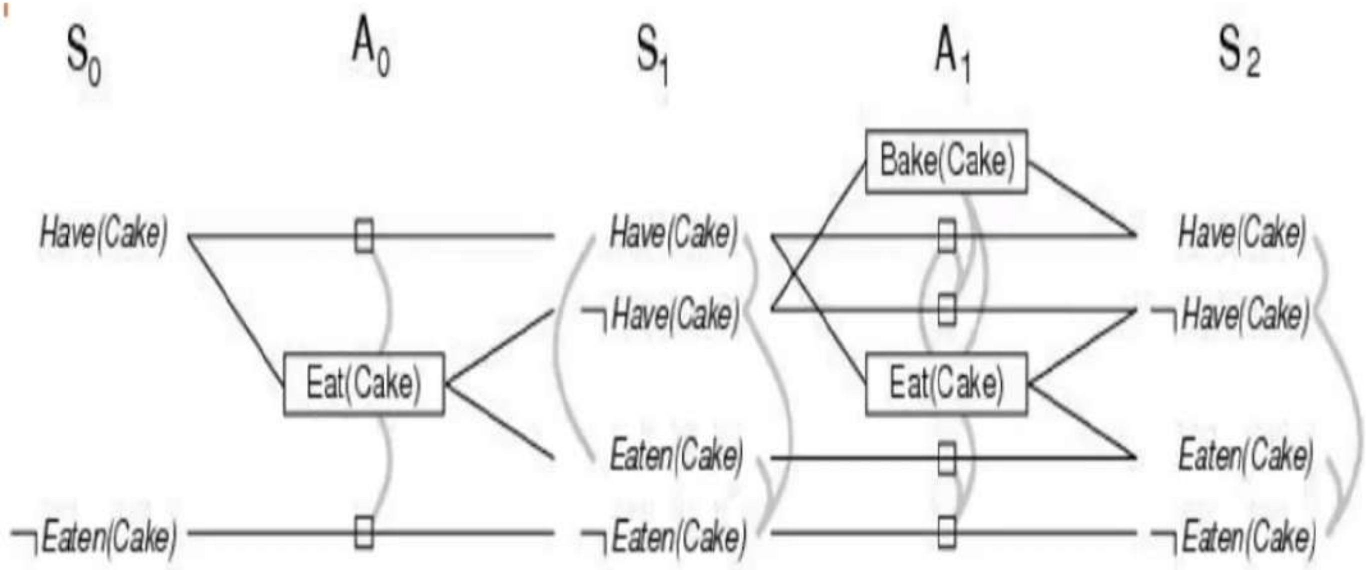
Identify *mutual exclusions* between actions and literals based on potential conflicts.

Mutual exclusion

- A mutex relation holds between two actions when:
  - Inconsistent effects: one action negates the effect of another.
  - Interference: one of the effects of one action, is the negation of a precondition of the other.
  - Competing needs: one of the preconditions of one action, is mutually exclusive with the precondition of the other.
- A mutex relation holds between two literals when:
  - one is the negation of the other
  - each possible action pair that could achieve the literals is mutex (inconsistent support).



- Level S1 contains all literals, that could result from, picking any subset of actions in A0
  - Conflicts between literals that can not occur together (as a consequence of the selection action) are represented by mutex links.
  - S1 defines multiple states, and the mutex links are the constraints, that define this set of states.



- Repeat process until graph levels off:
  - two consecutive levels are identical, or
  - contain the same amount of literals

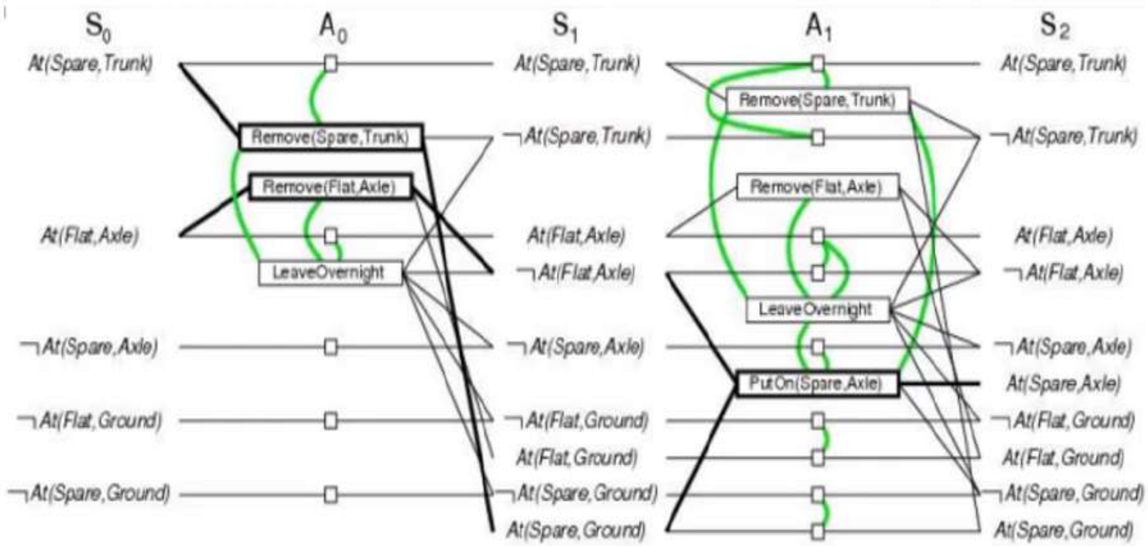
The GRAPHPLAN Algorithm

function GRAPHPLAN(*problem*) **returns** solution or failure

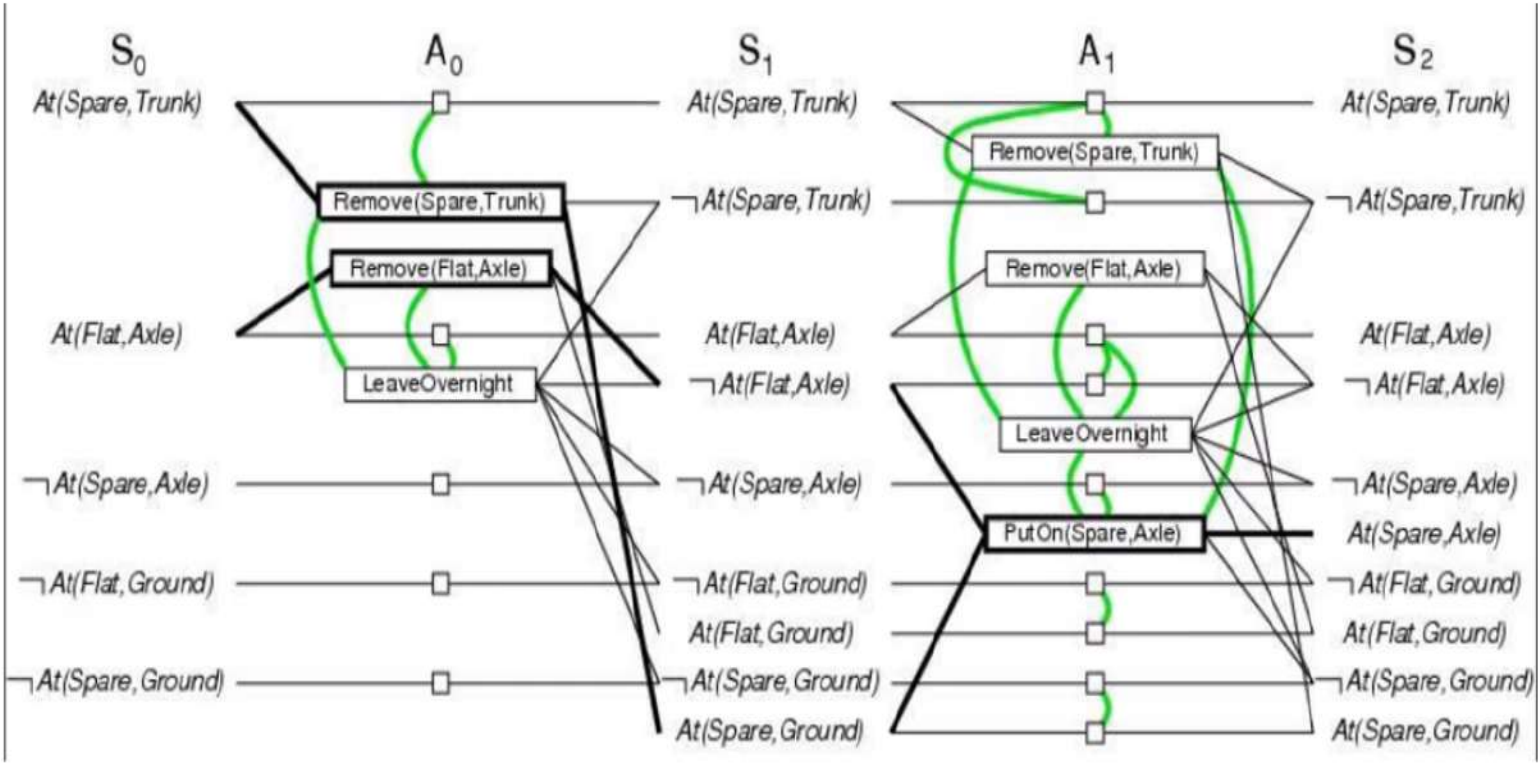
graph ← INITIAL-PLANNING-GRAPH(*problem*)
goals ← GOALS[*problem*]
loop do
    if *goals* all non-mutex in last level of *graph* **then do**
        solution ← EXTRACT-SOLUTION(*graph*, *goals*, LENGTH(*graph*))
        if *solution* $\neq$ *failure* **then return** *solution*
        else if NO-SOLUTION-POSSIBLE(*graph*) **then return** *failure*
    graph ← EXPAND-GRAPH(*graph*, *problem*)

Figure 11.13 The GRAPHPLAN algorithm. GRAPHPLAN alternates between a solution extraction step and a graph expansion step. EXTRACT-SOLUTION looks for whether a plan can be found, starting at the end and searching backwards. EXPAND-GRAPH adds the actions for the current level and the state literals for the next level.

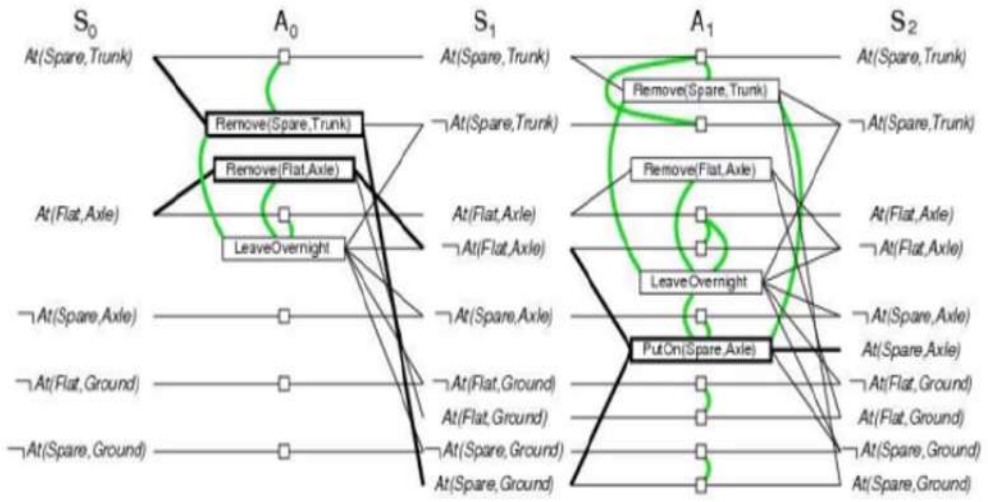
GRAPHPLAN example – Change Flat Tire



- Initially the plan consist of 5 literals from the initial state (S0).
- Add actions whose preconditions are satisfied by EXPAND-GRAPH (A0)
- Also add persistence actions and mutex relations.
- Add the effects at level S1
- Repeat until goal is in level Si

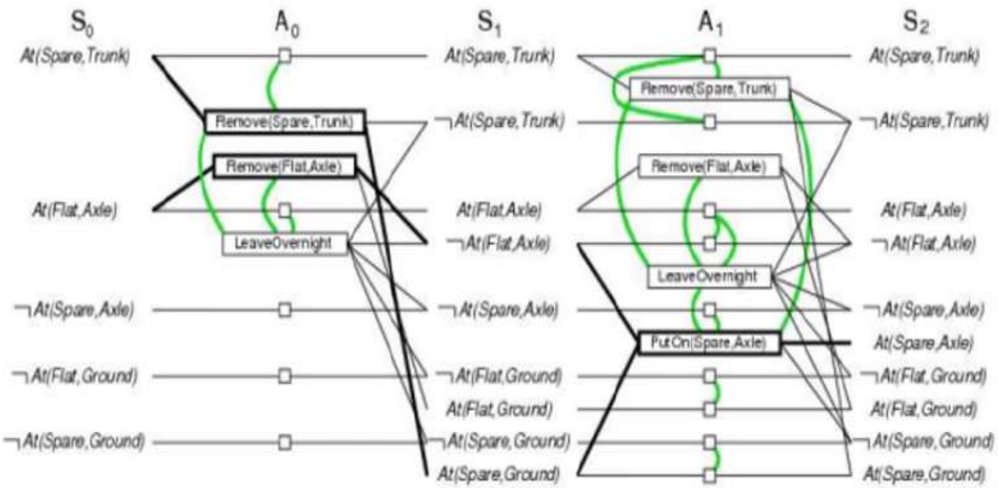


S0
A0
S1
A1
S2
At(Spare,Trunk)
Remove(Spare,Trunk)
¬At(Spare,Trunk)
Remove(Flat,Axle)
At(Flat,Axle)
LeaveOvernight
¬At(Flat,Axle)
¬At(Spare,Axle)
PutOn(Spare,Axle)
At(Spare,Axle)
¬At(Flat,Ground)
At(Flat,Ground)
¬At(Spare,Ground)
At(Spare,Ground)



EXPAND-GRAPH also looks for mutex relations

- Inconsistent effects
  - E.g. Remove(Spare, Trunk) and LeaveOverNight due to At(Spare,Ground) and **not** At(Spare, Ground)
- Interference
  - E.g. Remove(Flat, Axle) and LeaveOverNight At(Flat, Axle) as PRECOND and **not** At(Flat,Axle) as EFFECT
- Competing needs
  - E.g. PutOn(Spare,Axle) and Remove(Flat, Axle) due to At(Flat.Axle) and **not** At(Flat, Axle)
- Inconsistent support
  - E.g. in S2, At(Spare,Axle) and At(Flat,Axle)



In S2, the goal literals exist and are not mutex with any other

- Solution might exist and EXTRACT-SOLUTION will try to find it

EXTRACT-SOLUTION can use Boolean CSP to solve the problem or a search process:

- Initial state = last level of PG and goal goals of planning problem
- Actions = select any set of non-conflicting actions that cover the goals in the state
- Goal = reach level S0 such that all goals are satisfied
- Cost = 1 for each action.

GRAPHPLAN Termination

- Termination? YES
- PG are monotonically increasing or decreasing:
  - Literals increase monotonically
  - Actions increase monotonically
  - Mutexes decrease monotonically
- Because of these properties and a finite number of actions and literals, every PG will finally level off

Planning Problem

- Agent Environment States are represented as valuations of state variables
- an action can be represented as a *procedure* or a *program*
- The procedures are used to compute values of state variables
- After the execution of procedure (i.e. after the action), the environment state will be changed, towards the goal.

Planning Algorithms

- Representation of planning problems—states, actions, and goals—should make it possible for planning algorithms.
- Algorithms are nothing but logical structure of the problem.
- To define an efficient algorithm, language is very important.
- STRIPS Language – the language of classical planner.

Representation of States

- Planners decompose the agent world into logical conditions, and represent a state as a conjunction of positive literals.
- In first-order state descriptions, Literals must be **ground** and **function-free.**
  - At (x, y) or At (Father(Red), Sydney) are not allowed.
- The **closed-world assumption,** that is any conditions that are not mentioned in a state are assumed **false**

Representation of Goals.

- A goal is a partially specified state, represented as a conjunction of positive ground literals, such as
  - 1. Rich ^ Famous 2. At (P2,Delhi).
- A propositional state s, **satisfies** a goal g, if s contains all the atoms in g
- The propositional state Rich ^ Famous ^ Happy satisfies the goal state Rich ^ Famous.

Representation of Actions.

- To perform an Action, we need Precondition (how environment should be to perform this action) and Effect (how the environment will be after performing this action).
- An action for flying a plane from one location to another is:
  - Action : Fly(p, from, to)
  - PRECOND : At (p, from) ^ Plane(p) ^ Airport(from) ^ Airport(to)
  - EFFECT : At (p, from) ^ At (p, to)

Action Schema

- It represents a number of different actions can be derived
- Action schema consists of three parts
- The **action name and parameter list**—
  - for example, Fly(p, from, to)— fly is action, and p, from, to are parameters.
- The **precondition** is a conjunction of function-free positive literals, and it must be true, before the action can be executed.
  - Variables in the precondition must also appear in the action's parameter list.
- The **effect** is a conjunction of function-free literals, describing how the state changes, when the action is executed.
  - A positive literal P are declared to be true in the state, and all the negative literal P is declared to be false.
  - Variables in the effect must also appear in the action's parameter list.

Applicable Action

- An action is **applicable** in any state, iff that satisfies the precondition, otherwise, the action has no effect.
- a first-order action schema, first all the variables in precondition will be substituted
  - At (P1, JFK) ^ At (P2, SFO) ^ Plane(P1) ^ Plane(P2) ^ Airport(JFK) ^ Airport(SFO) This state satisfies the precondition
  - At (p, from) ^ Plane(p) ^ Airport(from) ^ Airport(to)
- with the substitution {p=P1, from=JFK, to=SFO}
- Thus, the concrete action Fly(P1, JFK, SFO) is applicable

Solution for Planning Problem

- An action sequence, that started from the initial state, and results in a state that satisfies the goal.
- Solutions to be partially ordered sets of actions
- Every action sequence, that respects the partial order, is a solution (i.e. every action has its own solution.)

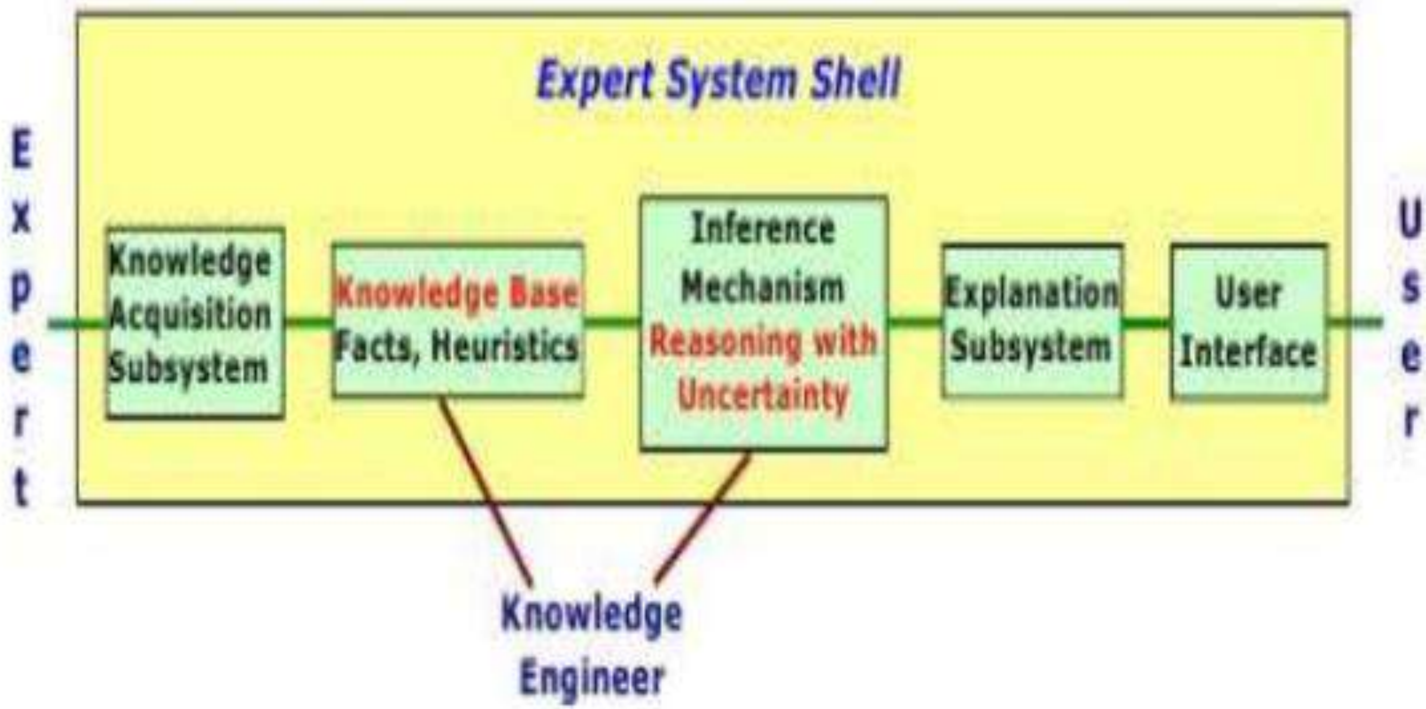
Unit 5

Expert System Shell

Expert System Shells

- An Expert system shell is a software development environment.
- It contains the basic components of expert systems.
- A shell is associated with a prescribed method for building applications by configuring and instantiating these components.

Expert System Shells



Shell components

1. knowledge acquisition,
2. knowledge Base,
3. reasoning,
4. explanation and
5. user interface

Knowledge acquisition

Clip slide

- A subsystem to help experts in build knowledge bases.
- However, collecting knowledge, needed to solve problems and build the knowledge base, is the biggest bottleneck in building expert systems.

Knowledge Base

- A store of factual and heuristic knowledge. Expert system tool provides one or more knowledge representation schemes for expressing knowledge about the application domain.
- Some tools use both Frames (objects) and IF-THEN rules.
- In PROLOG the knowledge is represented as logical statements.

Reasoning Engine

- Inference mechanisms for manipulating the symbolic information and knowledge in the knowledge base form a line of reasoning in solving a problem.
- The inference mechanism can range from simple modus ponens backward chaining of IF-THEN rules to Case-Based reasoning.

Explanation

- A subsystem that explains the system's actions.
- The explanation can range from how the final or intermediate solutions were arrived at justifying the need for additional data.

User interface

- A means of communication with the user. The user interface is generally not a part of the expert system technology.
- It was not given much attention in the past.
- However, the user interface can make a critical difference in the perceived utility of an Expert system.

Case Studies: MYCIN

MYCIN

- MYCIN is an expert system used for diagnosing bacterial infections. The major characteristics of medical domain is the uncertain and imprecise data coupled with vast quantities of medical knowledge.
- MYCIN was developed at Stanford University to help physicians in identifying what bacteria has been the cause for the infection and to suggest remedial solutions.

MYCIN: knowledge base

- MYCIN's "knowledge base" organized as set of production rules. One distinguishing features of the production rule in MYCIN is the certainty factor associated with it. Herewith, we , give a sample production rule in LISP and also the English equivalent. Sample MYCIN rule

(internal representation)

```
PREMISE : ($AND (SAME CNTXT GRAM GRAMNEG)
                (SAME CNTXT MORPH ROD)
                (SAME CNTXT AIR ANAEROBIC)
ACTION : (CONCLUDE CNTXT IDENTITY BACTERIODES TALLY .6)
```

MYCIN: knowledge base

English representation

IF the infection is primary-bacteremia

AND the site of the culture is one of the sterile sites

AND the suspected portal of entry is the gastrointestinal tract

THEN there is suggestive evidence (0.7) that infection is bacteroid.

Apart from the production rule, the system has a collection of facts. Facts are stored in the form

CONTEXT-PARAMETER-VALUE OR

OBJECT-AT TRIBUTE-VALUE TRIPLES

The OBJECT also called CONTEXT refers to such things as cultures taken from the patient, the drugs administered and so on. These OBJECTS are characterized by their ATTRIBTE or PARAMETER and the value of these are stored in VALUE. Associated with each triple is a certainty factor between -1 and +1. A CF of +1 indicates total belief, while a CF of -1 indicates total disbelief.

Ex: (IDENTITY ORGANISM -1 PSEUDOMONAS 0.8)

Which means "Pseudomonas is the identity of the organism-1 with certainty 0.8".

MYCIN: Reasoning and Problem Solving Strategy

- MYCIN could use backward chaining to find out whether a possible bacteria was to blame.
- This was augmented with "certainty factors" that allowed an assessment of the likelihood, if no one bacteria was certain.
- MYCIN's problem solving strategy was simple:
  - For each possible bacteria:
    - Using backward chaining, try to prove that it is the case, finding the certainty.
  - Find a treatment which "covers" all the bacteria above some level of certainty.

MYCIN: Problem Solving

- When trying to prove a goal through backward chaining, system could ask user certain things.
  - Certain facts are marked as "askable", so if they couldn't be proved, ask the user.
- This results in following style of dialogue:
- MYCIN: Has the patient had neurosurgery?
  USER: No.
  MYCIN: Is the patient a burn patient?
  USER: No.
  …

MYCIN: Problem Solving

- As of today, many extensions to the MYCIN approach have emerged. In order to minimize the human intervention in knowledge acquisition facility, a system called TEIRESIAS has been developed. Through TEIRESIAS, the expert can directly communicate with the knowledge base.
- Another system, called GUIDON has also been developed that uses the explanation capability of MYCIN for instructional purposes. MYCIN, as n expert system has surpassed human physicians in their reasoning capabilities.

Knowledge Acquisition stages

- Knowledge acquisition has five stages throughout the development. The stages are as following:

Identification

- This stage identifies the problems and the knowledge engineer becomes aware of the domain, its goals and selects the correct material.

Conceptualization

- This defines how the concepts or ideas and the associations between them are outlined and how experts relate them.

Formalization

- Here the knowledge engineer organizes the concepts, tasks and other information into formal and clear representation.

Knowledge Acquisition stages

Implementation

- Here the knowledge rules are put into a structured form for the expert system tool and a prototype (trial model) is created for testing out the design and the processes. The knowledge engineer has to produce a written documentation that will connect the knowledge base topics with the original data that were created earlier.

Testing

- The prototype system is tested for its efficiency and accuracy to see if it is working as required. In order to do this a small scenario or problem set is tested and the results from this system are used to alter or improve the prototype system.

LISt Processing : LISP

Introduction

- LISP was invented by John McCarthy during the late 1950s.
- It is particulary suited for AI programs becausee of its ability to process symbolic information effectively.
- Basic building blocks of LISP are the **Atom, List**, and the **String**.
- An **Atom** is a number or string of contiguous characters, including numbers and spectial characters.
- A **List** is sequence of atoms and/or other lists enclosed within parentheses.
- **A String** is a group of characters enclosed in double.

What is Prolog?

- Prolog is acronym of **PRO**gramming in **LOG**ic.
- Prolog program is sequence of rules and facts.
- Prolog Rule for grandfather is defined as:

```
gfather(X, Y) :- father(X, Z), parent(Z, Y).
```

```
father(james, robert).
```

Expert Systems

What is an Expert System?

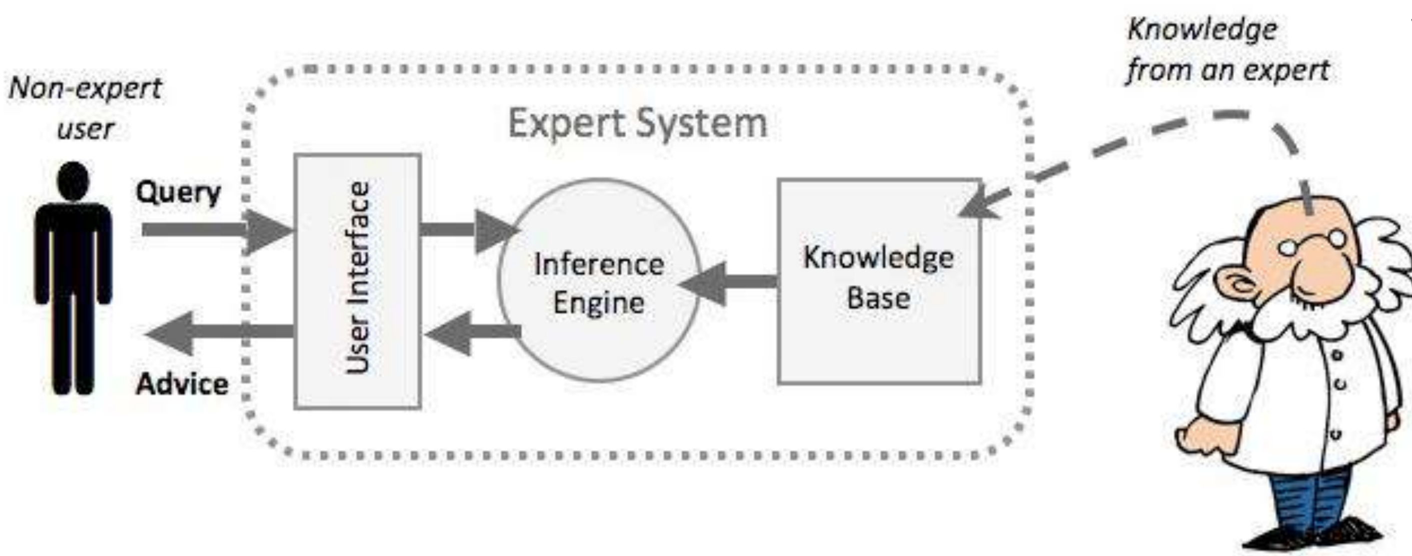
- An expert system is computer **software(program)** that exhibits **intelligent behavior and** also that attempts to **act like a human expert** on a particular **subject area**.
  It in-corporates the concepts and methods of symbolic inference reasoning and the use of knowledge for making these inferences.
  Expert systems also called as **knowledge based expert system**.
- Expert systems are often used to **advise non-experts** in situations where a human expert in unavailable (for example it may be too expensive to employ a human expert, or it might be a difficult to reach location).

How Do Expert Systems Work?

An expert system is made up of three parts:

- A **user interface** - This is the system that allows a **non-expert user** to **query** (question) the expert system, and to **receive advice**. The user-interface is designed to be a **simple** to use as possible.
- A **knowledge base** - This is a **collection of facts and rules**. The knowledge base is created from **information** provided by **human experts**
- An **inference engine** - This acts rather like a **search engine**, examining the knowledge base for information that **matches** the user's **query**

VV IMP



Where Are Expert Systems Used?

- **Medical diagnosis** (the knowledge base would contain medical information, the symptoms of the patient would be used as the query, and the advice would be a diagnose of the patient's illness)
- Playing **strategy games** like **chess** against a computer (the knowledge base would contain strategies and moves, the player's moves would be used as the query, and the output would be the computer's 'expert' moves)
- Providing **financial advice** - whether to invest in a business, etc. (the knowledge base would contain data about the performance of financial markets and businesses in the past)
- Helping to **identify items** such as plants / animals / rocks / etc. (the knowledge base would contain characteristics of every item, the details of an unknown item would be used as the query, and the advice would be a likely identification)
- Helping to **discover locations to drill for water / oil** (the knowledge base would contain characteristics of likely rock formations where oil / water could be found, the details of a particular location would be used as the query, and the advice would be the likelihood of finding oil / water there)

Phases in building Expert System

VV IMP

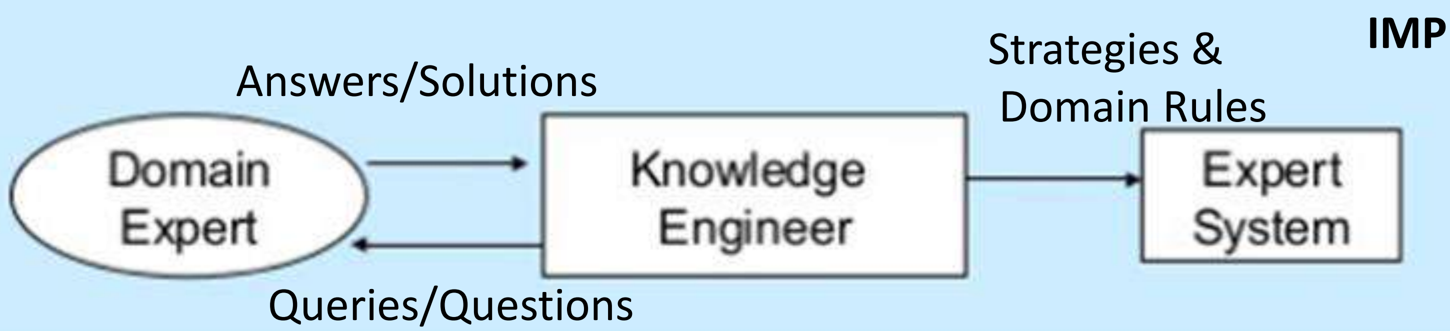
- There are different interdependent and overlapping phases in building an expert system as follows:
- ***Identification Phase***:
  - Knowledge engineer finds out important features of the problem with the help of domain expert (human).
  - He tries to determine the type and scope of the problem, the kind of resources required, goal and objective of the ES.
- ***Conceptualization Phase***:
  - In this phase, knowledge engineer and domain expert decide the concepts, relations and control mechanism needed to describe a problem solving.
- ***Formalization Phase***:
  - It involves expressing the key concepts and relations in some framework supported by ES building tools.
  - Formalized knowledge consists of data structures, inference rules, control strategies and languages for implementation.
- ***Implementation Phase***:
  - During this phase, formalized knowledge is converted to working computer program initially called prototype of the whole system.
- ***Testing Phase:***
  - It involves evaluating the performance and utility of prototype systems and revising it if need be. Domain expert evaluates the prototype system and his feedback help knowledge engineer to revise it.

Knowledge Engineering:

- The process of gathering knowledge from a domain expert and **codifying** it according to the formalism is called **knowledge engineering.**

- The tasks and responsibilities of a knowledge engineering involve the following:
  1. Ensuring that the computer has all the knowledge needed to solve a problem
  2. Choosing one or more forms to represent the required knowledge.
  3. Ensuring that the computer can use the knowledge efficiently by selecting some of the reasoning methods.

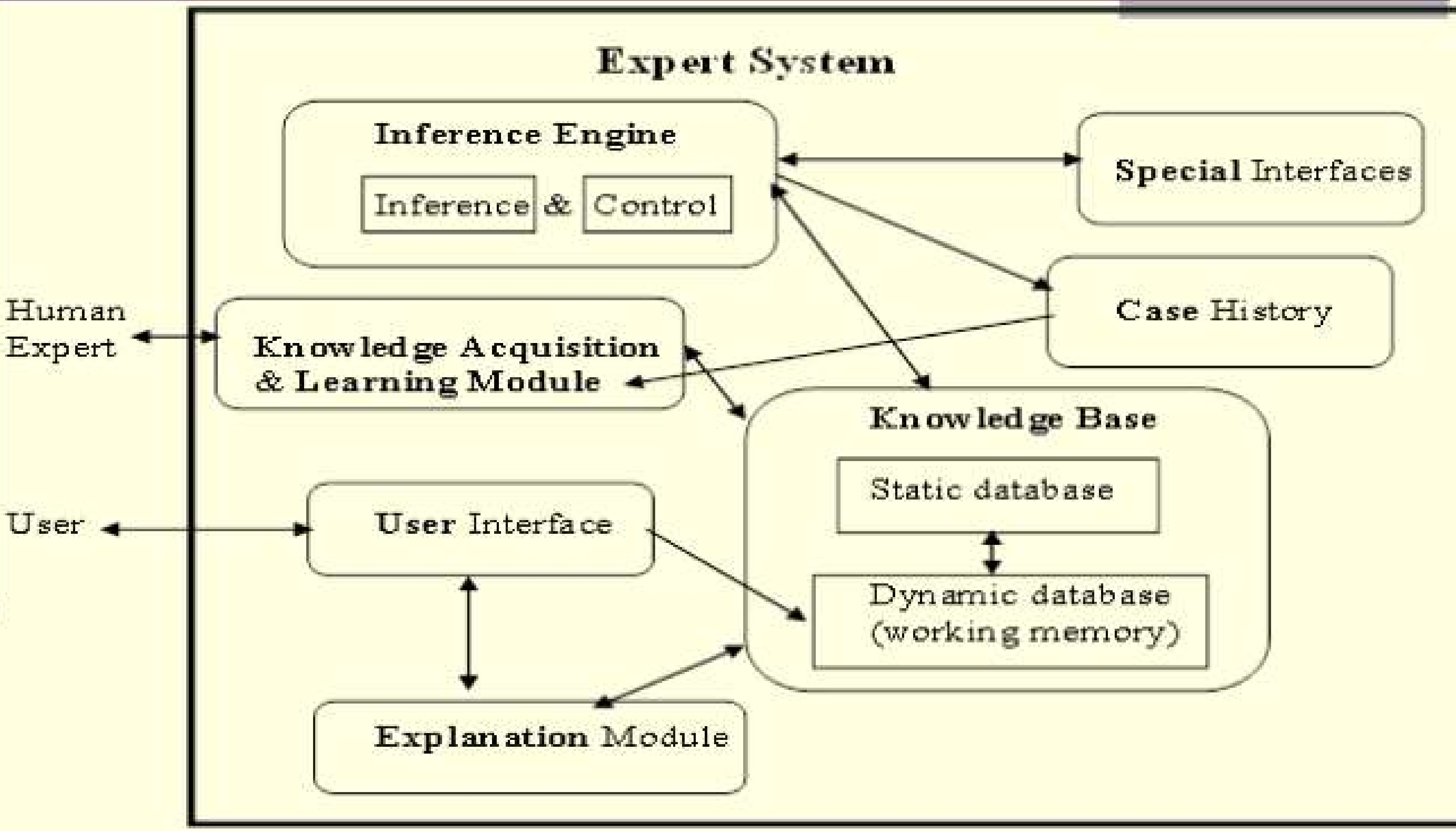
Interaction between Knowledge engineer and domain expert for creating an ES



- The main role in knowledge engineer begins only once the problem of some domain for developing an ES is decided. The job of the knowledge engineer involves close collaboration with the domain expert and end user.
- The next step of the process involves a more systematic interviewing of the expert. The knowledge engineer will then extract general rules from the discussion and interview held with expert and get them checked by the expert for correctness.
- The domain knowledge consisting of both formal, textbook knowledge and experiential knowledge is entered into the program piece by piece

Expert System Architecture

VV IMP



Knowledge Base (KB)

- KB consists of knowledge about problem domain in the form of static and dynamic databases.
- Static knowledge consists of
  - rules and facts which is complied as a part of the system and does not change during execution of the system.
- Dynamic knowledge consists of facts related to a particular consultation of the system.
  - At the beginning of the consultation, the dynamic knowledge base often called working memory is empty.
  - As a consultation progresses, dynamic knowledge base grows and is used along with static knowledge in decision making.
- Working memory is deleted at the end of consultation of the system.

Inference Engine

- It consists of inference mechanism and control strategy.
- Inference means search through knowledge base and derive new knowledge.
- It involve formal reasoning involving matching and unification similar to the one performed by human expert to solve problems in a specific area of knowledge.
- Inference operates by using modus ponen rule.
- Control strategy determines the order in which rules are applied.
- There are mainly two types of control mechanism viz., forward chaining and backward chaining.

Forward Chaining Example

Suppose we have three rules:

R1: If A and B then D

R2: If B then C

R3: If C and D then E

If facts A and B are present, we infer D from R1 and infer C from R2. With D and C inferred, we now infer E from R3.

Backward Chaining Example

The same three rules:

R1: If A and B then D

R2: If B then C

R3: If C and D then E

If E is known, then R3 implies C and D are true. R2 thus implies B is true (from C) and R1 implies A and B are true (from D).

Knowledge Acquisition

- Knowledge acquisition module allows system to acquire knowledge about the problem domain.
- Sources of Knowledge for ES
  - text books, reports, case studies,
  - empirical data and
  - domain expert experience.
- Updation of Knowledge can be done using knowledge acquisition module of the system.
  - insertion,
  - deletion and
  - updation of existing knowledge

Case History

- Case History stores the file created by inference engine using the dynamic database created at the time of consultation.
- Useful for learning module to enrich its knowledge base.
- Different cases with solutions are stored in Case Base system.
- These cases are used for solving problem using Case Base Reasoning (CBR).

Explanation module

- Most expert systems have explanation facilities that allow the user to ask the system ***why*** it asked some question, and ***how*** it reached to conclusion.
- It contains 'How' and 'Why' modules attached to it.
  - The sub-module 'How' tells the user about the process through which system has reached to a particular solution
  - 'Why' sub-module tells that why is that particular solution offered.
- It explains user about the reasoning behind any particular problem solution.
- Questions are answered by referring to the system goals, the rules being used, and any existing problem data.

Explanation Modules Responses

User	Why module
System	Do you have running nose?
User	Yes.
System	I am trying to prove that you have cold. Do you have headache?
User	Yes
System	Do you have sore throat?
User	No
System	Then you do not have cold. Do you have fever?
User	Yes
System	Now I am looking at the possibility of measles. Do you have cough?
User	Yes
System	I can infer measles using rule "If symptoms are fever, cough, running_nose, then patient has measles" measles is concluded.
User	How Module
System	Since you have fever, running_nose and cough and there is a rule "If symptoms are fever, cough, running_nose, then patient has measles". So measles is concluded for you.

User Interfaces

- Allows user to communicate with system in interactive mode and helps system to create working knowledge for the problem to be solved.

Dialogue Module (User Interface)	
System	Do you have fever?
User	Yes
System	Do you have bad throat?
User	No
System	Do you have cough?
User	Yes
System	Are you suffering from running nose?
User	Yes
System	Are you suffering from headache?
User	No

Special interfaces

- It may be used for specialized activities such as handling uncertainty in knowledge.
- This is a major area of expert systems research that involves methods for reasoning with uncertain data and uncertain knowledge.
- Knowledge is generally incomplete and uncertain.
- To deal with uncertain knowledge, a rule may have associated with it a *confidence factor* or a weight.
- The set of methods for using uncertain knowledge in combination with uncertain data in the reasoning process is called *reasoning with uncertainty*.

Traditional System(Conventional) vs Expert Systems

Conventional Systems	Expert Systems
Information and processing combined in a single sequential program	Knowledge base separate from the mechanism processing (inference)
The program is never wrong	The program could have made a mistake
Need all the input data	Not necessarily need all inputs data or facts
Changes to the program inconvenient	Changes in the rules can be made with ease
The system works if it is complete	The system can work only with the rules a little
Efficiency is the main objective	Effectiveness is the main objective
quantitative data	qualitative data
Representation of data in numerical	Representation symbols

Characteristics Of Expert Systems

- The Highest level of expertise
- Right on time reaction
- Accepting the incorrect reasoning
- Good reliability
- Easily understood
- Flexible
- Symbolic reasoning
- Heuristic reasoning
- Making mistakes
- Expanding with tolerable difficulties

Advantages of Expert System:

- Helps in preservation scarce expertise
- Provides consistent answers for repetitive decisions process and tasks.
- Fastens the pace of human professional or semi-professional work
- Holds and maintains significant levels of information.
- Provides improved quality of decision making
- Domain experts are not always able to explain their logic and reasoning unlike ES
- Encourages organizations to clarify the logic of decision making
- Leads to major internal cost savings within companies
- Causes introduction of new products
- Never forgets to ask questions, unlike human.

DISAdvantages of Expert System:

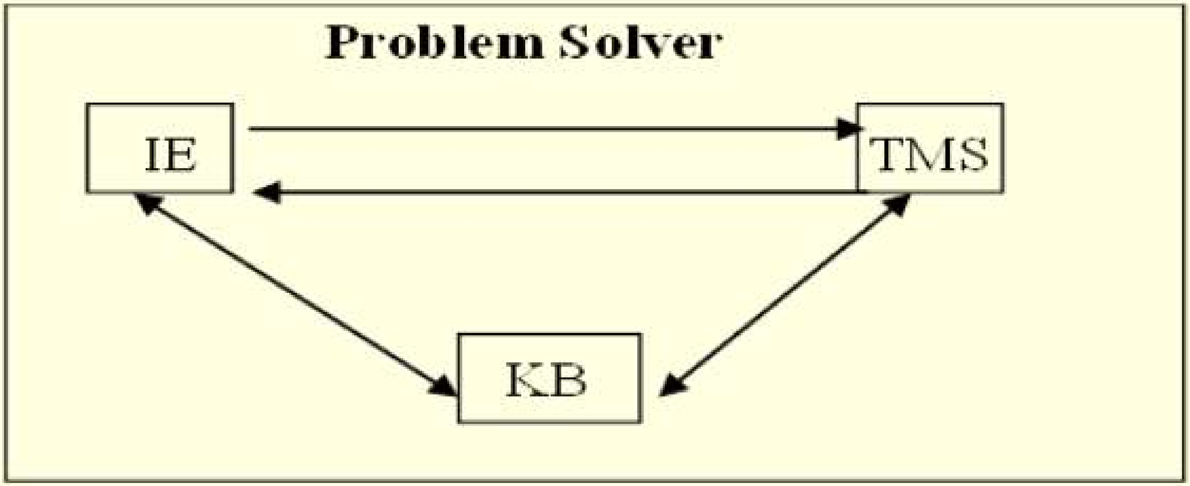
- Unable to make creative response as human experts would in unusual circumstances.
- Lacks common sense needed in some decision making.
- May cause errors in the knowledge base, and lead to wrong decisions.
- Cannot adapt to changing environments. Unless knowledge base in changed

Languages used in Expert System:

LISP (List Processing), Prolog(Programming in Logic), C, C++, JAVA etc.

Truth Maintenance System (TMS)

- Truth maintenance system (TMS) works with inference engines for solving problems within large search spaces.
- The TMS and inference engine both put together can solve problems where algorithmic solutions do not exist.
- TMS maintains the beliefs for general problem solving systems.



TMS - Cont

- TMS can be used to implement monotonic or non-monotonic systems.
- In monotonic system, once a fact or piece of knowledge is stored in KB, it can not change.
  - In monotonic reasoning, the world of axioms continually increases in size and keeps on expending.
  - Predicate logic is an example of monotonic form of reasoning. It is a deductive reasoning system where new facts are derived from the known facts.
- Non-monotonic system allows retraction of truths that are present in the system whenever contradictions arise.
  - So number of axioms can both increase and decrease and depending upon the changes in KB, it can be updated.

Monotonic TMS:

- The most practical applications of monotonic systems using TMS are qualitative simulation, fault diagnosis and search applications.
- A monotonic TMS is general facility for manipulating Boolean constraints on proposition symbols. The constraint has the form P→Q where P and Q are proposition symbols that an outside observer can interpret as representation of the statements.

Functionality of a Monotonic TMS

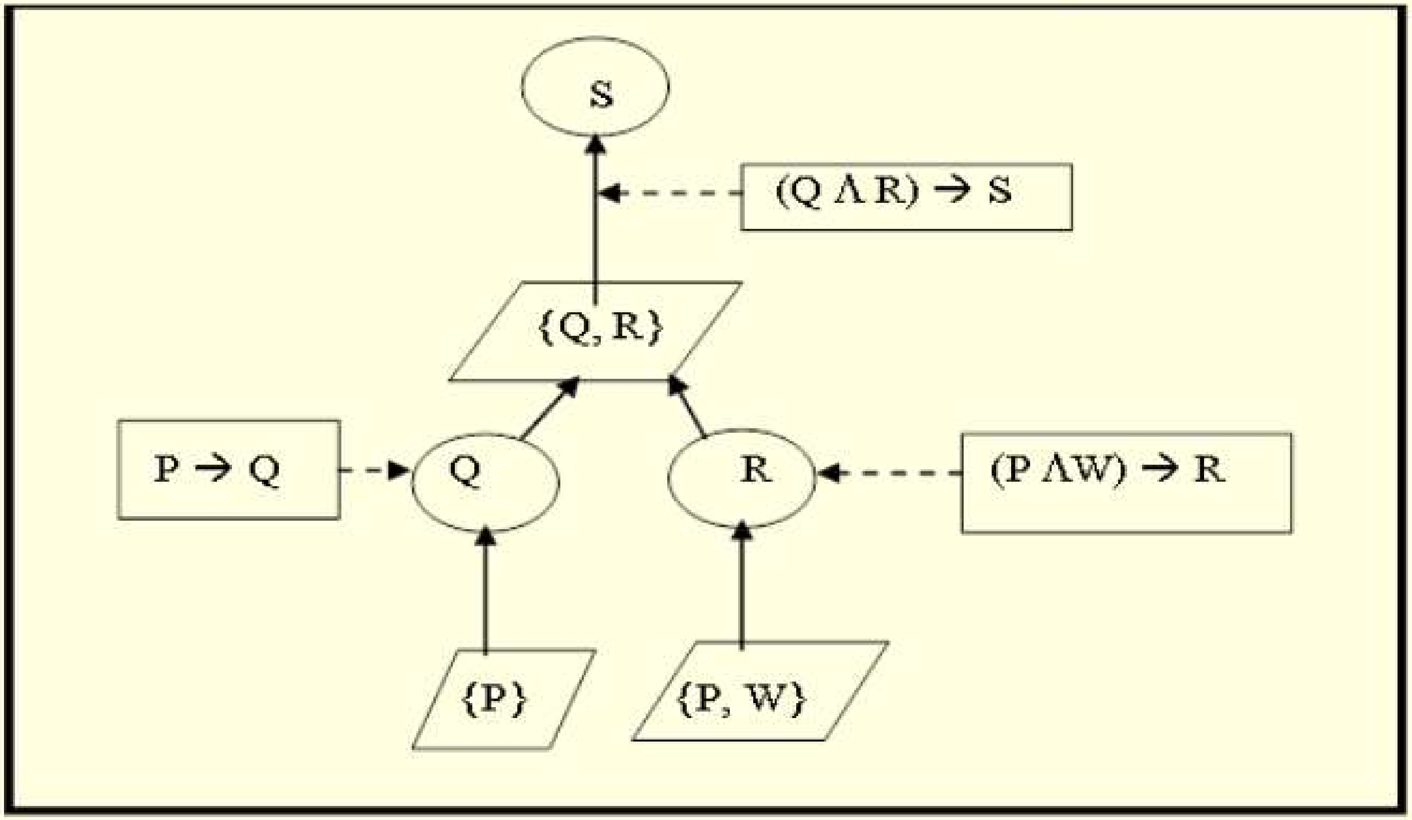
- A TMS stores a set of Boolean constraints, Boolean formulas(premises) and assings truth values to literals that satisfy this stored set of constraints.
- A TMS generally consist of the following generic interface funstions:
  - Add_constraint
  - Follow_Form
  - Interface funtions

Example – Monotonic TMS

- Suppose we are given the premise set $\Sigma = \{P, W\}$ and the internal constraint set

  $\{P \rightarrow Q, (P \wedge W) \rightarrow R, (Q \wedge R) \rightarrow S\}$.
- TMS are able to derive S from these constraints and the premise set Σ.
- TMS should provide the justifications of deriving S from constraints and premises.
- Therefore, for any given set of internal constraints and premise set Σ, if a formula S can be derived from these, then justification functions generate a justification tree for S.

Justification Tree



Non-Monotonic TMS

- TMS basically operates with two kinds of objects
  - 'Propositions' declaring different beliefs and
  - 'Justifications' related to individual propositions for backing up the belief or disbelief expressed by the proposition.
- For every TMS, there are two kinds of justifications required namely 'Support list' and 'Conditional proof'.

Support list (SL) :

- It is defined as "SL(IN-node)(OUT-node)", where IN-node is a list of all IN-nodes (propositions) that support the considered node as true.
  - Here IN means that the belief is true.
  - OUT-node is a list of all OUT nodes for the considered node to be true. OUT means that belief is not true.

Example

Node number	Facts/assertions	Justification (justified belief)
1	It is sunny	SL(3) (2,4)
2	It rains	SL() ()
3	It is warm	SL(1) (2)
4	It is night time	SL() (1)

Conditional Proof

- A belief may be justified on the basis of several other beliefs, by the conditional proof on one belief relative to other beliefs, or by the lack of belief in some fact.
- These are justifications which support belief if a specified belief follows from a set of other beliefs.
- Truth maintenance processing is required when new justifications change previously existing beliefs.
- In such cases, the status of all beliefs depending on the changed beliefs must be re determined.
- Dependency-directed backtracking is a powerful technique based on the representations of the truth maintenance system.

List of Shells and Tools

- **Acquire:** It is primarily a knowledge-acquisition system and ES shell. Which provides a complete development environment for the building and maintenance of knowledge-based application.

- **MYCIN: MYCIN** was an early backward chaining expert system that used artificial intelligence to identify bacteria causing severe infections, such as bacteremia and meningitis, and to recommend antibiotics, with the dosage adjusted for patient's body weight — the name derived from the antibiotics themselves, as many antibiotics have the suffix "-mycin". The Mycin system was also used for the diagnosis of blood clotting diseases. MYCIN was developed over five or six years in the early 1970s at Stanford University. It was written in Lispas the doctoral dissertation of Edward Shortliffe under the direction of Bruce G. Buchanan, Stanley N. Cohen and others.

- **K-Vision**: It is a knowledge acquisition and visualization tool. It runs on windows dos etc.
- **MailBot**: IT is personal e-mail agent that reasd an e-mail message on standard input and creates an e-mail reply to be sent to the sender of the original message. It provides filtering, forwarding notification and automatic question-answering capabilities