RAMAIAH
Institute of Technology

Ramaiah Institute of Technology
(Autonomous Institute, Affiliated to VTU)
Department of AI and ML/AI and DS

Course Name: Introduction to Artificial Intelligence
Course Code: AI35/AD35
Credits: 3:0:0

Unit I

Introduction: (Chapter 1 of Text Book 1)

- What is AI?

Intelligent Agents: (Chapter 2 of Text Book 1)

- Agents and Environments
- Rationality
- The Nature of Environments
- The Structure of Agents

Solving Problems by search: (Chapter 3 of Text Book 1)

- Problem-Solving Agents
- Example Problems
- Search Algorithms
- Best- first search
- Uniformed Search Strategies- Breadth-first search
- Dijkstra‘s algorithm or uniform-cost search
- Informed (Heuristic) Search Strategies- Greedy best-first search
- A* search.

Text Books:

1. Stuart J Russel and Peter Norvig: “Artificial Intelligence - A Modern Approach”, 4th Edition, Pearson Education, 2021.
2. Tom M Mitchell, “Machine Learning”, McGraw-Hill Education (Indian Edition), 2013.
3. Elaine Rich, Kevin Knight, Shivashankar B Nair: Artificial Intelligence, 3rd Edition, Tata McGraw Hill, 2011.

Introduction

- **Why** consider artificial intelligence to be a subject most worthy of study?
- **What** exactly it is?
- This being a good thing to decide before embarking.
- Our intelligence is very important to us.
- For thousands of years, we have tried to u**nderstand how we think and act**?- perceive, understand, predict, and manipulate a world far larger and more complicated than itself

Introduction…

- Artificial intelligence (AI) is the study of ideas that enable **computers to be intelligent**.

- The goals of the field of AI are
  - To make **computers** more useful
  - To understand the **principles** that make intelligence possible.
  - **Building intelligent entities**—machines that can compute how to act effectively and safely in a wide variety of novel situations.

Introduction…

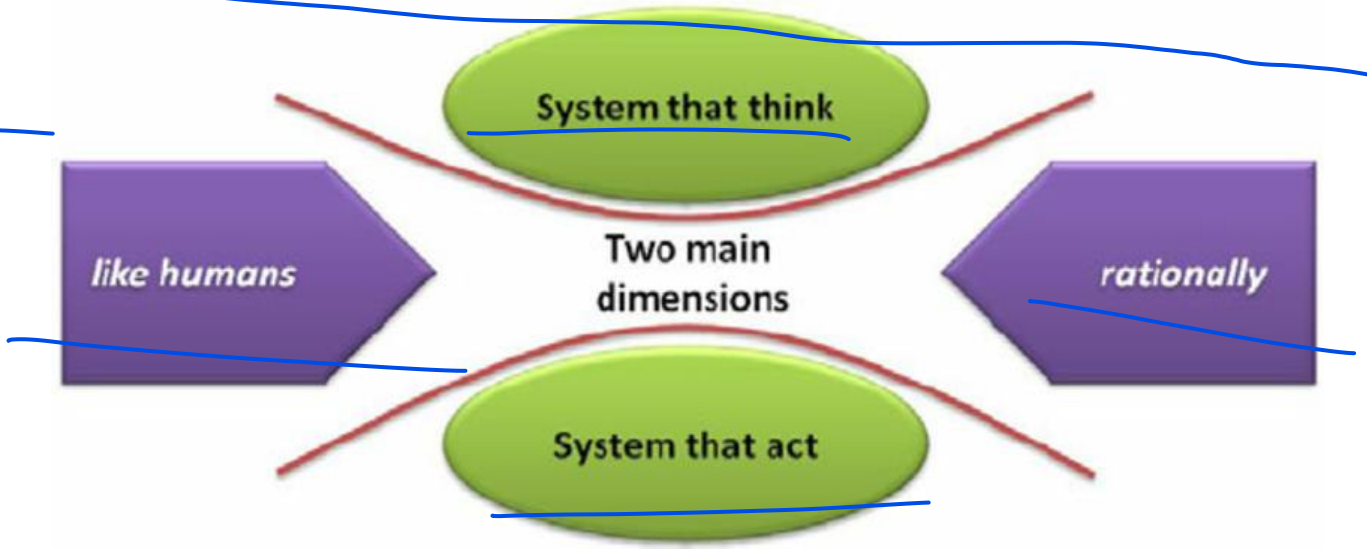
- **Surveys Rank AI as:**
  - One of the most interesting and **fastest-growing fields**
  - Generating over a trillion dollars a year in **revenue**.
  - AI expert Kai-Fu Lee predicts that its impact will be **"more than anything in the history of mankind"**
  - Subfields ranging from the **general** (learning, reasoning, perception, and so on) to the **specific**, such as playing chess, proving mathematical theorems, writing poetry, driving a car, or diagnosing diseases.
  - It is truly a universal field.

What is Artificial Intelligence?

- Some have defined intelligence as:
  - Fidelity to **human performance**
  - An abstract, formal definition of intelligence called **rationality**—loosely speaking, doing the "right thing."
  - Property of **internal thought processes and reasoning,**
  - Focus on **intelligent behavior,** an external characterization.

What is Artificial Intelligence?

- AI definitions can be viewed into two dimensions.
- **In the first dimension, AI can be regarded as a system that think like humans or that thinks rationally.**
- **In second dimension, AI is viewed as a system that acts like humans or that act rationally.**



What is Artificial Intelligence?

- From the two dimensions—
  - **Human vs. Rational and Thought vs. Behavior**
  - There are four possible combinations.
  - **The activity of human-like intelligence** : related to psychology, involving observations and hypotheses about actual human behavior and thought processes;
  - **The rationalist approach** : a combination of mathematics and engineering, and connects to statistics, control theory, and economics.

What is Artificial Intelligence?

Four Main Approaches to Artificial Intelligence

	Humanly	Rationally
Thinking	Think Humanly (e.g., cognitive modeling)	Think Rationally (e.g., logic)
Acting	Act Humanly (e.g., Turing test)	Act Rationally (e.g., engineering)

What is Artificial Intelligence?

Four Main Approaches to Artificial Intelligence

- TH and TR — Thought Process
- AH and AR —Behaviour
- TH and AH — Human Performance
- TR and AR — Ideal Performance (Rationally)

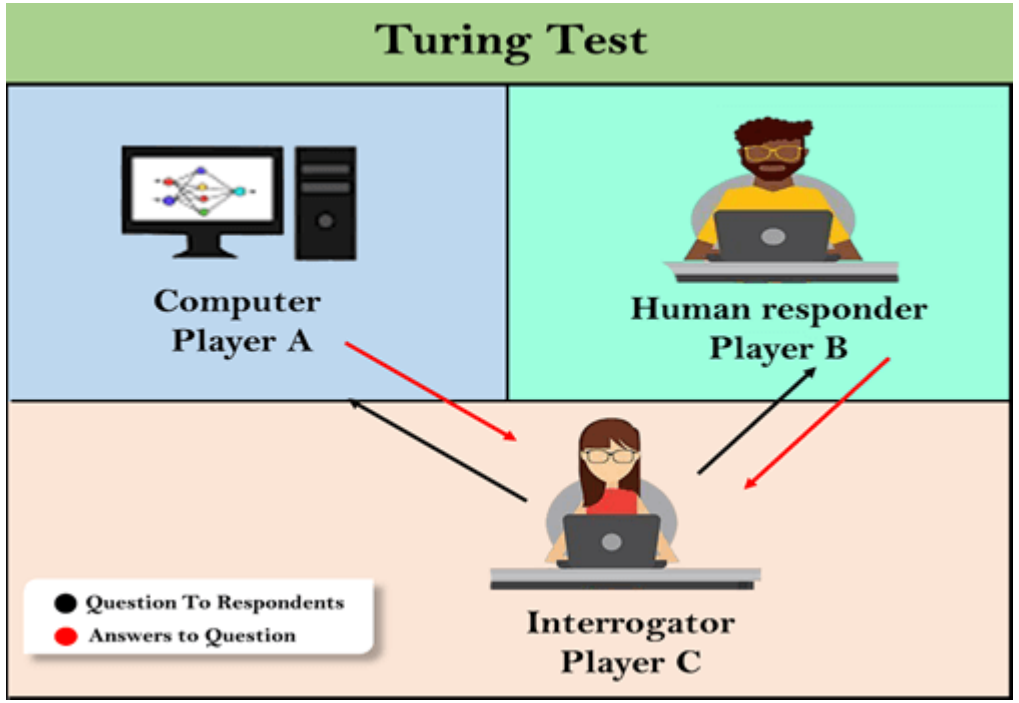
What is Artificial Intelligence? (contd..)

- Acting humanly: The Turing Test approach
- The Turing Test, proposed by Alan Turing (1950), was designed as a thought experiment that would check on "Can a machine think?"

Alan Turing (1912-1954)

What is Artificial Intelligence? (contd..)

- Acting humanly: The Turing Test approach
- A human interrogates the computer and another human via a terminal simultaneously.
- After a reasonable period, a computer passes the test if a human interrogator, after posing some written questions, and cannot tell whether the written responses come from a person or from a computer.

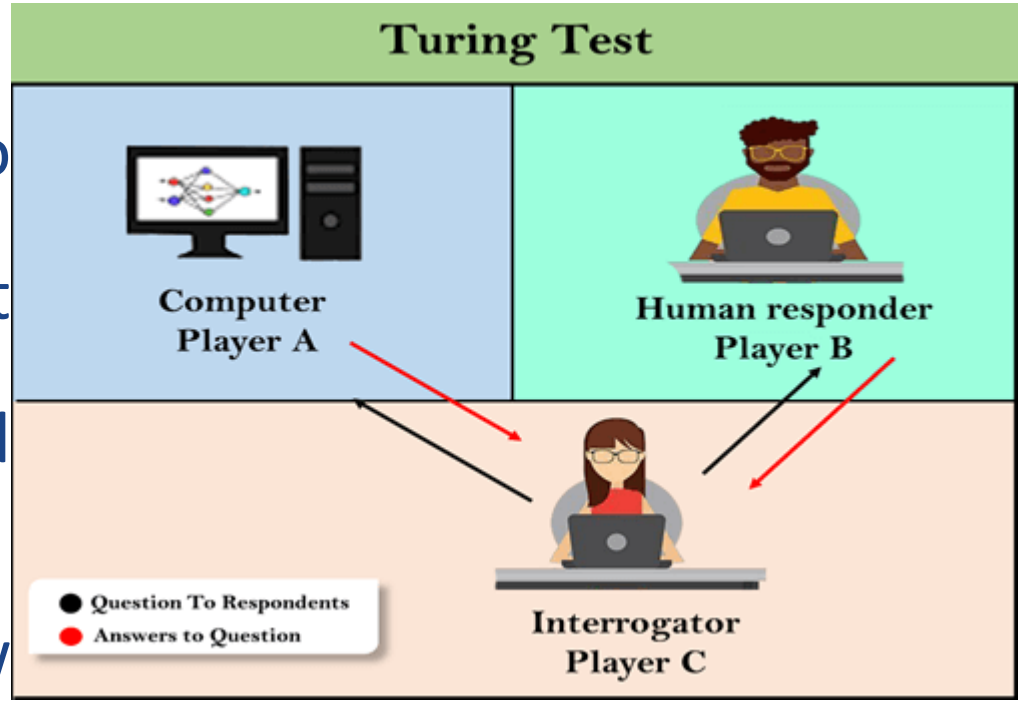


What is Artificial Intelligence? (contd..)

- Acting humanly: The Turing Test approach

The computer would need to possess the following capabilities:

- **Natural Language Processing** to enable it to communicate successfully in English;
- **Knowledge Representation** to store what it knows or hears;
- **Automated Reasoning** to use the stored information to answer questions and to draw new conclusions;
- **Machine Learning** to adapt to new circumstances and to detect and extrapolate patterns



What is Artificial Intelligence? (contd..)

- Acting humanly: The Turing Test approach

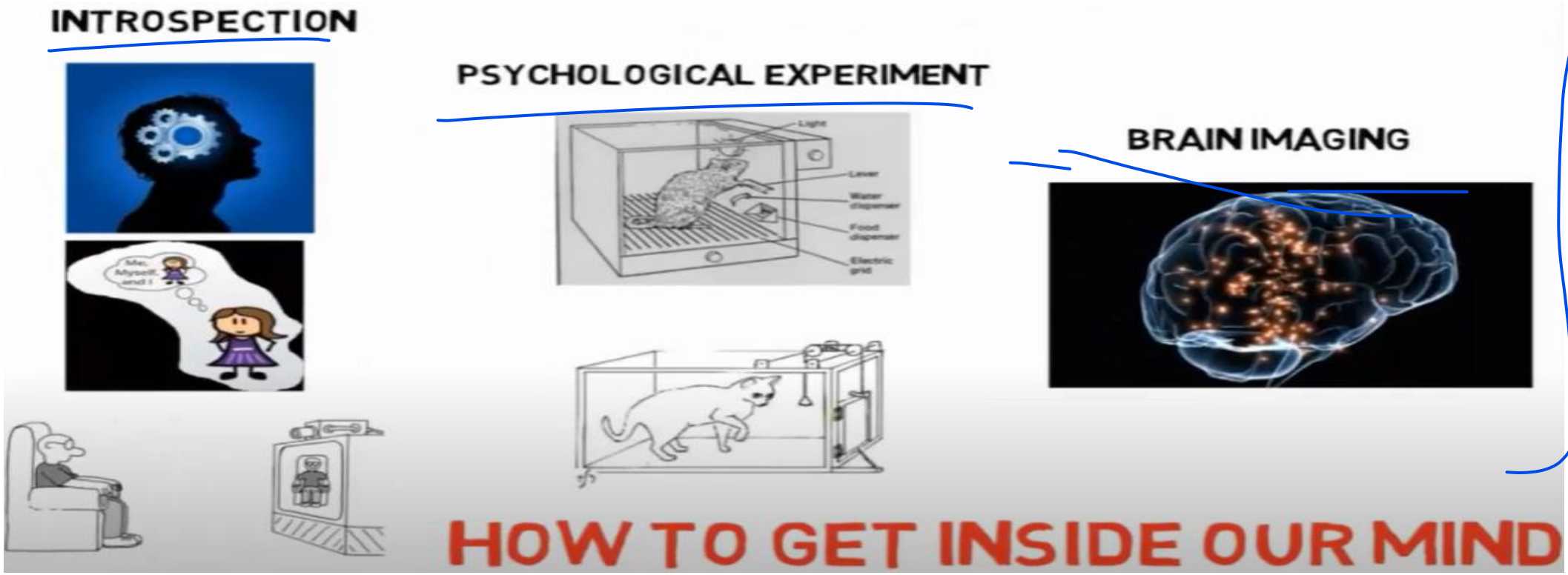
- Turing viewed the physical simulation of a person as **unnecessary** to demonstrate intelligence.
- Other researchers have proposed a **total Turing test**, which requires **interaction** with objects and people in the real world.
- To pass the Total Turing test, a **robot** will need:
  - Computer vision to perceive objects,
  - Robotics to manipulate objects and move about.

What is Artificial Intelligence? (contd..)

- **Thinking humanly: The cognitive modeling approach**
  - Program thinks like a human
  - Determining how humans think.
  - Get inside the actual workings of human minds.
  - There are three ways to do this:
    - Introspection—trying to catch our own thoughts as they go by;
    - Psychological experiments—observing a person in action; and
    - Brain imaging—observing the brain in action.

What is Artificial Intelligence? (contd..)

- Thinking humanly: The cognitive modeling approach



What is Artificial Intelligence? (contd..)

- Thinking humanly: The cognitive modeling approach
  - **Cognitive science** is the study of the human mind and brain, focusing on how the mind represents and manipulates knowledge and how mental representations and processes are realized in the brain.
  - The **interdisciplinary field** of cognitive science brings together computer models from AI and experimental techniques from psychology to construct precise and testable theories of the human mind.

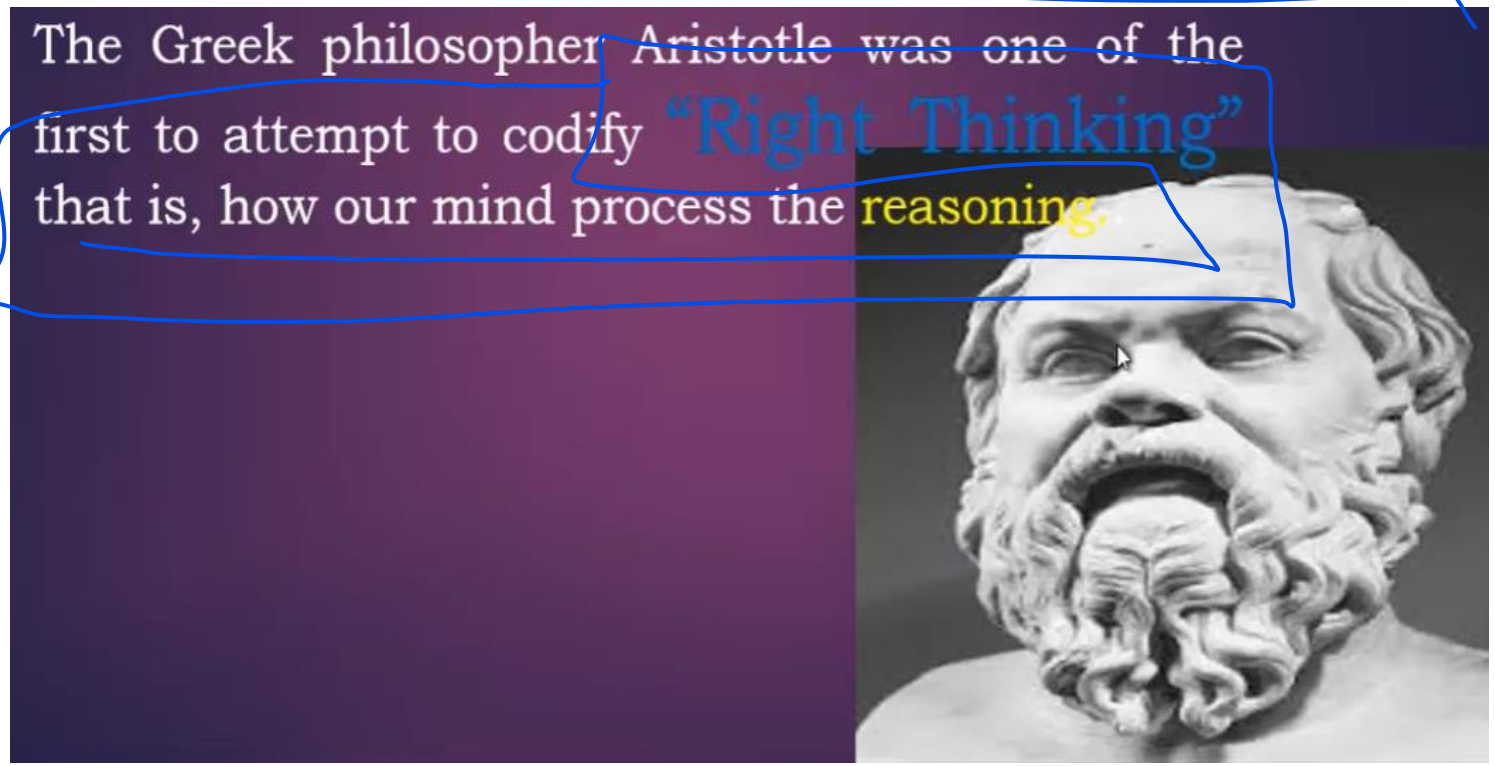
What is Artificial Intelligence? (contd..)

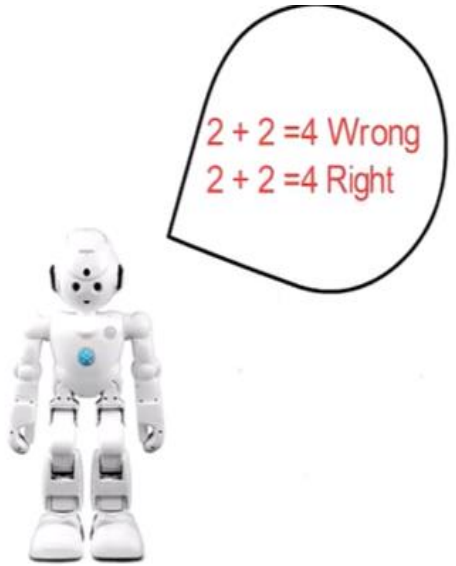
- Thinking rationally: The "laws of thought" approach

  - Rational thinking is a **process**.
  - It refers to the ability **to think with reason**.
  - It encompasses the ability to draw sensible **conclusions from facts, logic and data**.
  - If your thoughts are based on **facts and not emotions**, it is called rational thinking.
    Rational thinking focuses on resolving problems and achieving goals.

What is Artificial Intelligence? (contd..)

- Thinking rationally: The “laws of thought” approach





What is Artificial Intelligence? (contd..)

- Thinking rationally: The "laws of thought" approach
- **Syllogisms** provided patterns for argument structures that always yielded correct conclusions when given correct premises. A syllogism is a type of logical reasoning where the conclusion is yielded from two linked premises.

- Example, "Socrates is a man; all men are mortal; therefore, Socrates is mortal."
- These laws of thought were supposed to govern the operation of the mind; these study initiated the field called **logic**.

What is Artificial Intelligence? (contd..)

Thinking rationally: The "laws of thought" approach

- Logicians in the 19th century developed a precise notation for statements about objects in the world and the relations among them.
- Logic as conventionally understood and requires knowledge of the world that is **certain—a** condition that, in reality, is seldom achieved.

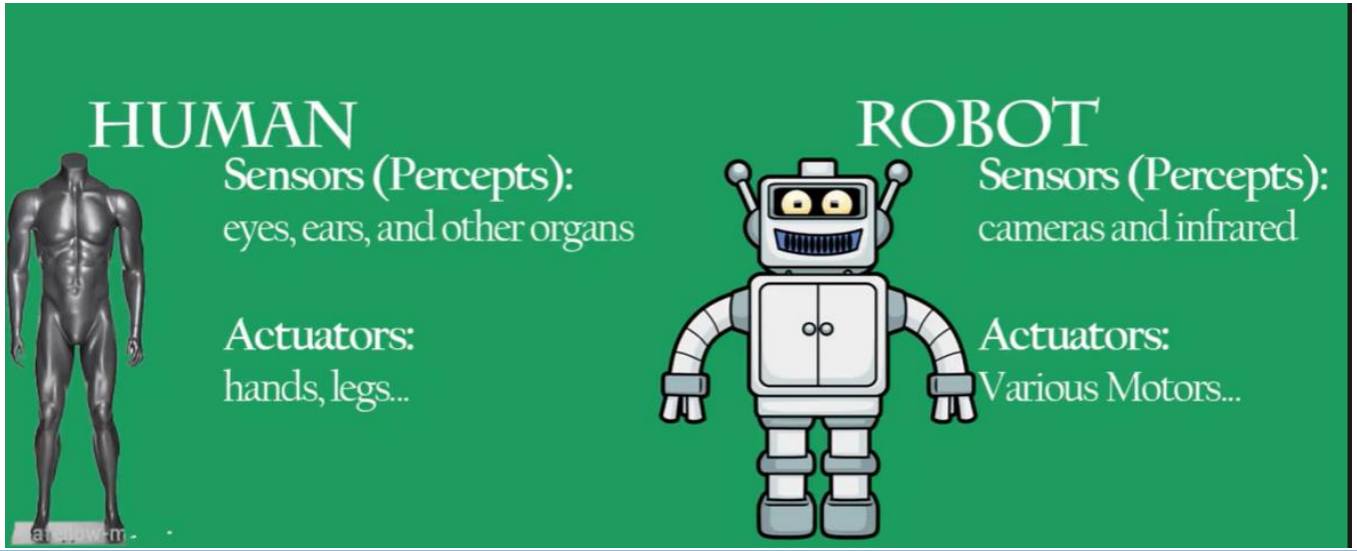
What is Artificial Intelligence? (contd..)

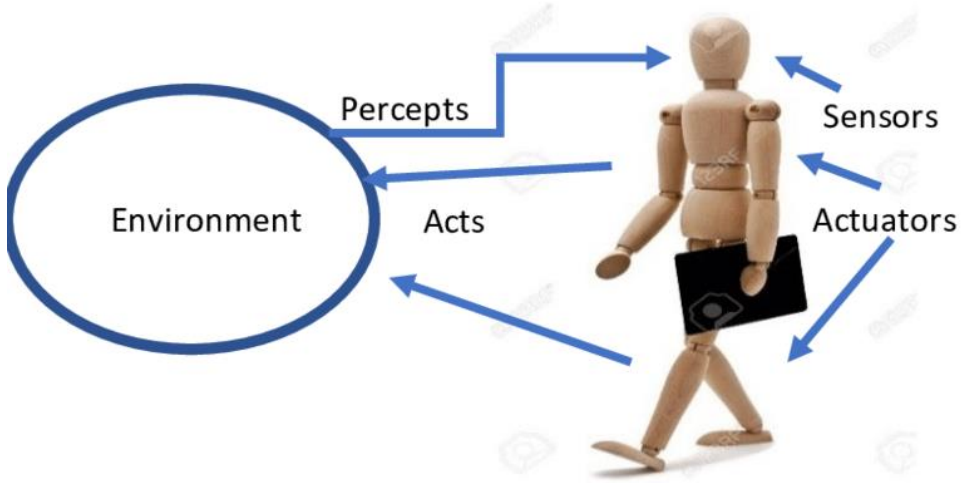
Thinking rationally: The "laws of thought" approach

- The theory of **probability** fills the gap, allowing rigorous reasoning with uncertain information.
- In principle, it allows the construction of a comprehensive model of rational thought, leading from raw perceptual information to an understanding of how the world works to **predictions about the future**.

What is Artificial Intelligence? (contd..)

- Acting rationally: The rational agent approach
- An agent is just something that **acts** (Latin agree- to do).
- An agent can be viewed as perceiving its environment through sensors and acting upon that environment through actuators





What is Artificial Intelligence? (contd..)

- Acting rationally: The rational agent approach

- All computer programs do something, but **computer agents** are expected to do more: operate autonomously, perceive their environment, persist over a prolonged time period, adapt to change, and create and pursue goals.
  - A rational agent is one that acts so as to achieve the best outcome or, when there is uncertainty, the best expected outcome.

What is Artificial Intelligence? (contd..)

- Acting rationally: The rational agent approach
- In the “laws of thought” approach to AI, the emphasis was on correct inferences.
- Making correct inferences is sometimes **part** of being a rational agent, because one way to act rationally is to deduce that a given action is best and then to act on that conclusion.

What is Artificial Intelligence? (contd..)

- Acting rationally: The rational agent approach
- *In Conclusion:* ***"AI has focused on the study and construction of agents that do the right thing. "***
- What counts as the right thing is defined by the **objective** that we provide to the agent.
- This general paradigm we call it the **standard model.**

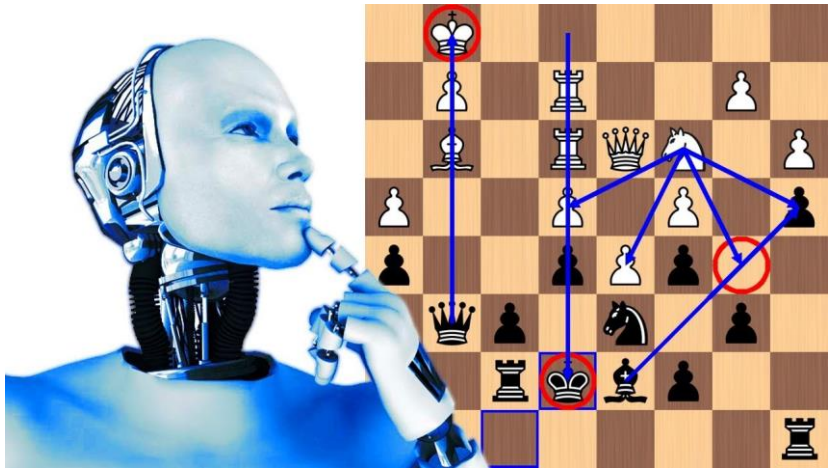
RAMAIAH
Institute of Technology

What is Artificial Intelligence? (contd..)

- Acting rationally: The rational agent approach
- **Need one important refinement to standard model:**
  - **Perfect rationality**—always taking the exactly optimal action—is not feasible in complex environments. The computational demands are just too high.
  - **Limited rationality**—acting appropriately when there is not enough time to do all the computations one might like.

What is Artificial Intelligence? (contd..)

- Beneficial machines
  - The standard model probably is not the right model in the long run.
  - As it assumes a fully specified objective will be supplied to the machine.
  - Applicable : Chess
  - Not Applicable : Real time self driving car.



What is Artificial Intelligence? (contd..)

- Beneficial machines
  - The problem of achieving agreement between our **true preferences** and the **objective** are put into the machine is called the **value alignment problem.**
  - AI system in the **lab or in a simulator**—easy fix for an incorrectly specified objective: reset the system, fix the objective, and try again.
  - Intelligent systems in the **real world**: This approach is no longer viable.
  - A system deployed with an **incorrect objective** will have negative consequences.

RAMAIAH
Institute of Technology

What is Artificial Intelligence? (contd..)

Thinking Humanly	**Thinking Rationally**
"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)
Acting Humanly	**Acting Rationally**
"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)	"Computational Intelligence is the study of the design of intelligent agents." (Poole *et al.*, 1998) "AI . . . is concerned with intelligent behavior in artifacts." (Nilsson, 1998)

Some definitions of artificial intelligence, organized into four categories.

Eight definitions of AI, laid out along two dimensions.

Top - thought processes and reasoning,
Bottom - address behavior.

Left measure success - Human performance
Right ideal performance measure, called rationality

Unit I

Introduction: (Chapter 1 of Text Book 1)

- What is AI?

Intelligent Agents: (Chapter 2 of Text Book 1)

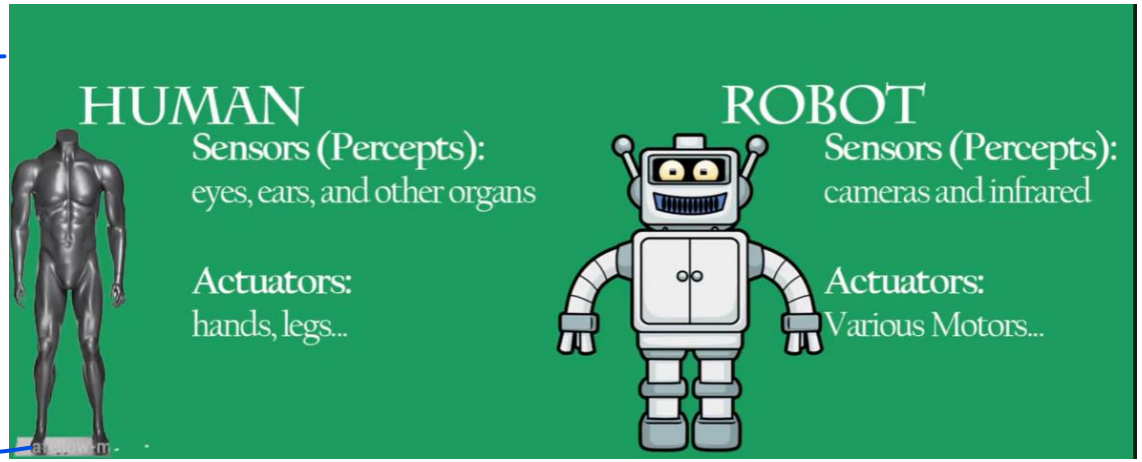
- **Agents and Environments**
- Rationality
- The Nature of Environments
- The Structure of Agents

Solving Problems by search: (Chapter 3 of Text Book 1)

- Problem-Solving Agents
- Example Problems
- Search Algorithms
- Best- first search
- Uniformed Search Strategies- Breadth-first search
- Dijkstra‘s algorithm or uniform-cost search
- Informed (Heuristic) Search Strategies- Greedy best-first search
- A* search.

Agents and Environments

- Rational Agents are considered as **approach to artificial intelligence**
- An **agent** is anything that can be viewed as perceiving its **environment** through **sensors** and **acting** upon that environment through **actuators.**
- **Example: Human, Robot**
- **Software Agents :** A software agent receives file contents, network packets, and human input (keyboard/mouse/touchscreen/voice) as sensory inputs and acts on the environment by writing files, sending network packets, and displaying information or generating sounds.



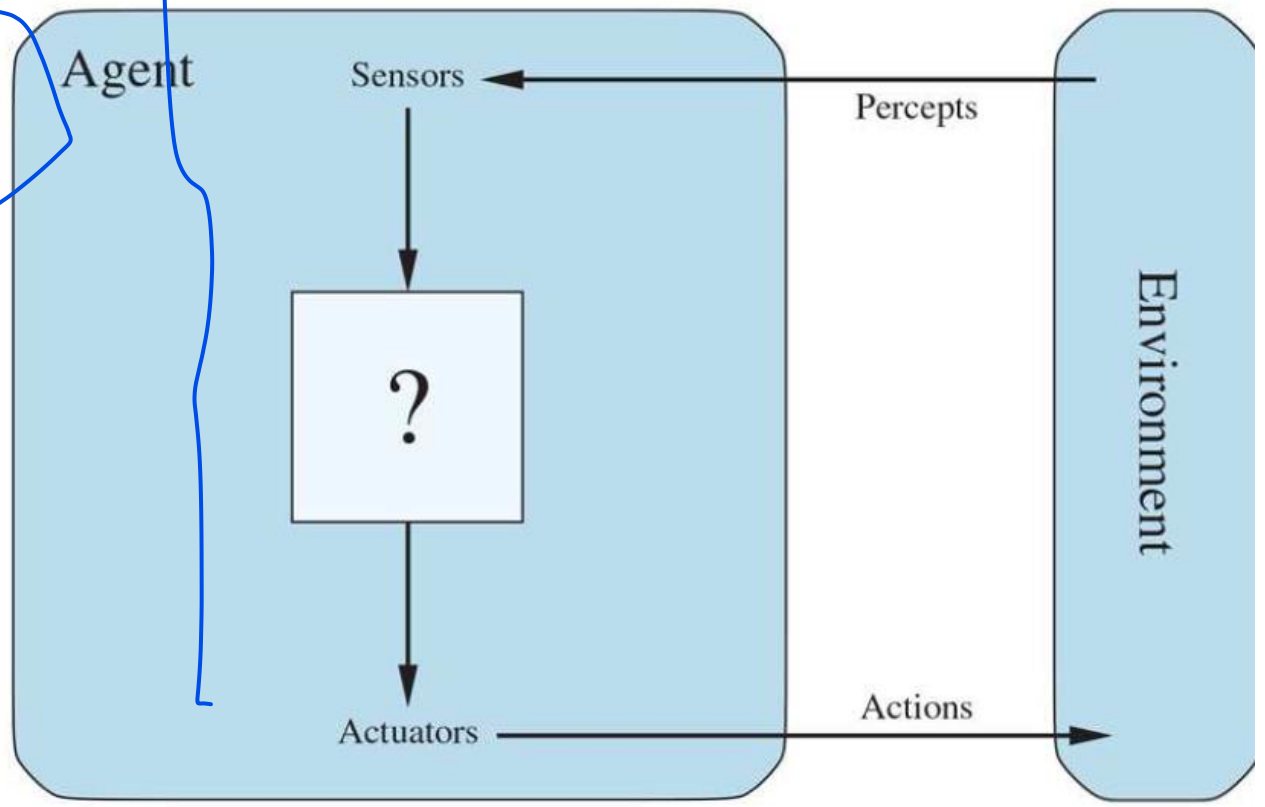
Agents and Environments

- The **environment** could be everything—the entire universe!
- In practice it is just that part of the universe whose state we care about when designing this agent
- The part that affects what the agent perceives and that is affected by the agent's actions.

Agents and Environments

- Perceive the environment through **sensors** (→Percepts)
- Act upon the environment through **actuators** (→Actions)

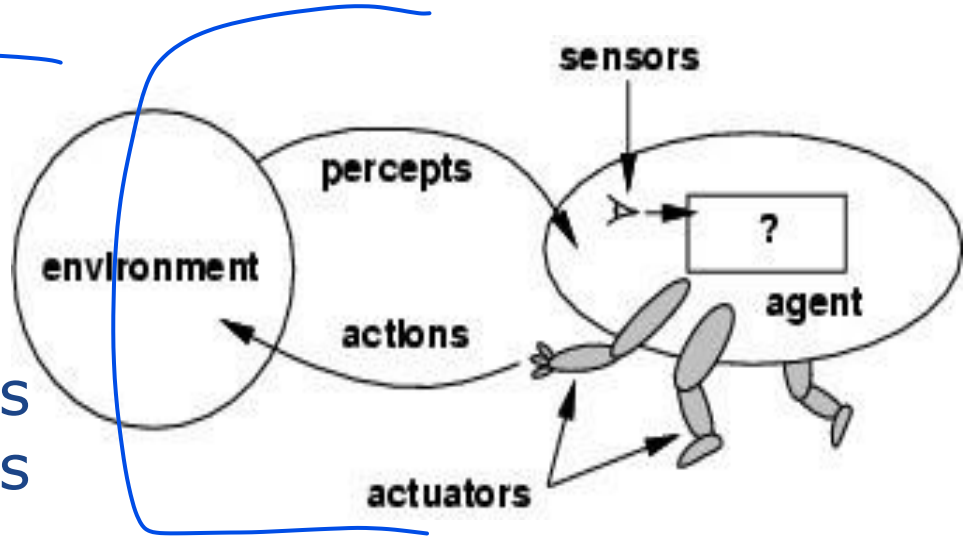
Figure 2.1



Agents interact with environments through sensors and actuators.

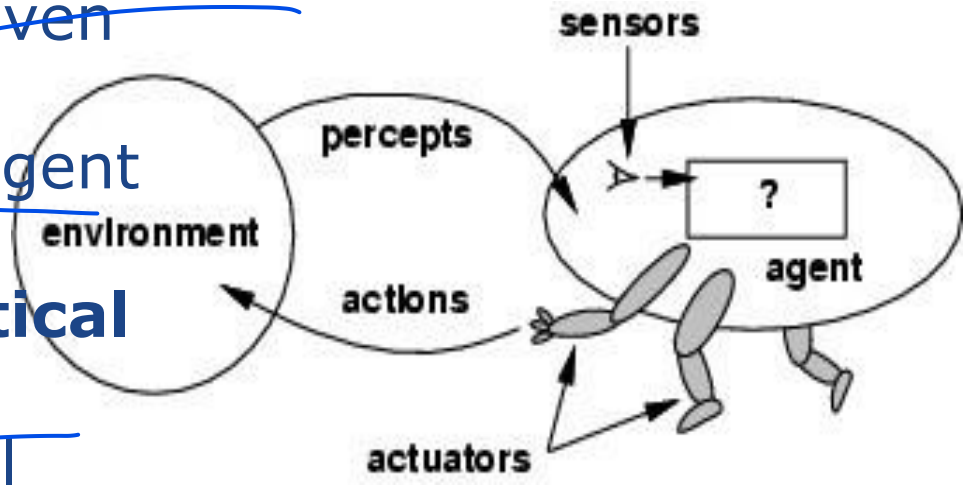
Agents

- Percept: The agent's perceptual inputs at any given instant
- Percept sequence: The complete history of everything the agent has perceived
- An **agent's choice of action**
- Built-in knowledge
- Entire percept sequence observed to date
- Mathematically, an agent's behavior is described by the **agent function** that maps any given percept sequence to an action.



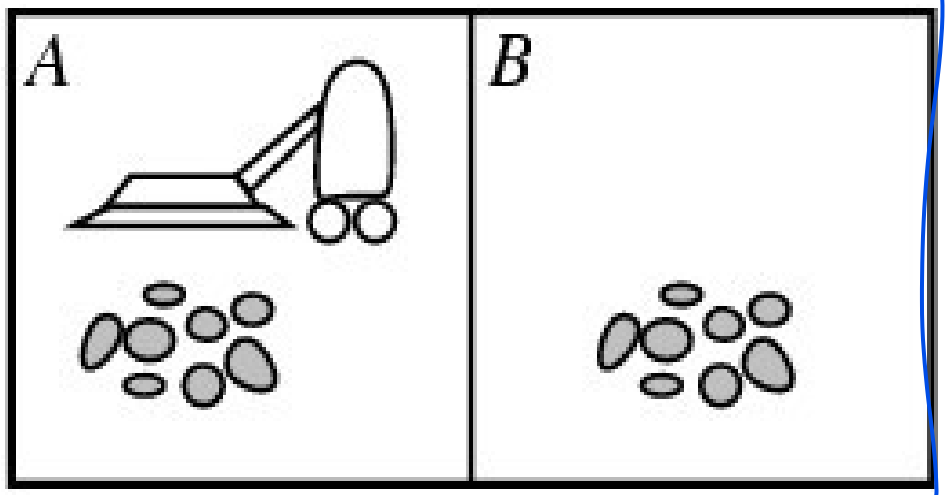
Agents

- **Tabulating** the agent function describes any given agent.
- Internally, the agent function for an artificial agent will be implemented by an **agent program**.
- The **agent function** is an **abstract mathematical** description; the agent program is a **concrete implementation**, running within some physical system.



Agents -> Example : Vacuum-Cleaner World

- Agent : Vaccum Cleaner
- Environment:Area of squares either clean or dirty
- Percepts: Perceive Location(Square), Clean or Dirty e.g., [A, dirty]
- Actions: Move Left, Right, Suck, NoOp



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
⋮	⋮
[A, Clean], [A, Clean], [A, Clean]	Right
[A, Clean], [A, Clean], [A, Dirty]	Suck
⋮	⋮

Partial tabulation of a simple agent function for the vacuum-cleaner world

Rationality

- A rational agent is one that does the right thing.
- What does it mean to do the right thing?
- Performance measure: Use notion called **consequentialism**:
  - Evaluate an agent's behavior by its **consequences**.
  - An agent is in an environment, it **generates a sequence** of actions according to the **percepts it receives**.
  - This sequence of actions causes the environment to go through a **sequence of states.**
  - If the sequence is **desirable**, then the agent has **performed well**.
  - This notion of desirability is captured by a **performance measure** that **evaluates** any given sequence of **environment states.**

Unit I

Introduction: (Chapter 1 of Text Book 1)

- What is AI?

Intelligent Agents: (Chapter 2 of Text Book 1)

- Agents and Environments
- **Rationality**
- The Nature of Environments
- The Structure of Agents

Solving Problems by search: (Chapter 3 of Text Book 1)

- Problem-Solving Agents
- Example Problems
- Search Algorithms
- Best- first search
- Uniformed Search Strategies- Breadth-first search
- Dijkstra‘s algorithm or uniform-cost search
- Informed (Heuristic) Search Strategies- Greedy best-first search
- A* search.

Rationality- Rational Agent

What is rational at any given time depends on four things:

- The performance measure that defines the criterion of success.
- The agent's prior knowledge of the environment.
- The actions that the agent can perform.
- The agent's percept sequence to date.
- **Definition of 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 –Vacuum Cleaner Example

- A simple agent that cleans a square if it is dirty and moves to the other square if not.
- Is it rational?
- Assumption:
  - **Performance measure**: Awards one point for each clean square at each time step, over a "lifetime" of 1000 time steps.
  - The "**geography**" of the environment is known a priori but the dirt distribution and the initial location of the agent are not.
  - **Actions** = {left, right, suck, no-op}
  - Agent is able to **perceive** the location and dirt in that location

Rationality –Vacuum Cleaner Example

- Is it irrational?
- Assumption:
  - Once all the dirt is cleaned up, the agent will **oscillate** needlessly back and forth;
  - A better agent for this case would do **nothing** once it is sure that all the squares are clean.
  - If clean squares can become dirty again, the agent should occasionally **check and re-clean** them if needed.
  - If the geography of the **environment is unknown**, the agent will need to **explore** it.

Unit I

Introduction: (Chapter 1 of Text Book 1)

- What is AI?

Intelligent Agents: (Chapter 2 of Text Book 1)

- Agents and Environments
- Rationality
- **The Nature of Environments**
- The Structure of Agents

Solving Problems by search: (Chapter 3 of Text Book 1)

- Problem-Solving Agents
- Example Problems
- Search Algorithms
- Best- first search
- Uniformed Search Strategies- Breadth-first search
- Dijkstra‘s algorithm or uniform-cost search
- Informed (Heuristic) Search Strategies- Greedy best-first search
- A* search.

The Nature of Environments

- Task environments, are essentially the "**problems**" to which rational agents are the "**solutions**."
- Task environments come in a variety of **flavors**.
- The **nature** of the task environment directly affects the appropriate design for the **agent program**.
- In Ex: Simple vacuum-cleaner agent: Specified the performance measure, the environment, and the agent's actuators and sensors.
- Group all these under the heading of the **task environment**

The Nature of Environments

- Specifying the task environment is always the first step in designing agent
- PEAS: **P**erformance, **E**nvironment, **A**ctuators, **S**ensors

P	**Performance** – how we measure the system's achievements
E	**Environment** – what the agent is interacting with
A	**Actuators** – what produces the outputs of the system
S	**Sensors** – what provides the inputs to the system

In **designing** an agent, the first step must always be to specify the task environment as fully as possible.

Taxi Driver Example

Agent Type	Performance Measure	Environment	Actuators	Sensors
Taxi driver	Safe, fast, legal, comfortable trip, maximize profits, minimize impact on other road users	Roads, other traffic, police, pedestrians, customers, weather	Steering, accelerator, brake, signal, horn, display, speech	Cameras, radar, speedometer, GPS, engine sensors, accelerometer, microphones, touchscreen

Mushroom-Picking Robot

Performance Measure	Environment	Actuators	Sensors
Percentage of good mushrooms in correct bins	Conveyor belt with mushrooms, bins	Jointed arm and hand	camera, joint angle sensors

Agent Type	Performance Measure	Environment	Actuators	Sensors
Medical diagnosis system	Healthy patient, reduced costs	Patient, hospital, staff	Display of questions, tests, diagnoses, treatments, referrals	Keyboard entry of symptoms, findings, patient's answers
Satellite image analysis system	Correct image categorization	Downlink from orbiting satellite	Display of scene categorization	Color pixel arrays
Part-picking robot	Percentage of parts in correct bins	Conveyor belt with parts; bins	Jointed arm and hand	Camera, joint angle sensors
Refinery controller	Purity, yield, safety	Refinery, operators	Valves, pumps, heaters, displays	Temperature, pressure, chemical sensors
Interactive English tutor	Student's score on test	Set of students, testing agency	Display of exercises, suggestions, corrections	Keyboard entry

Examples of agent types and their PEAS descriptions.

Properties of Task Environments

- The range of task environments that might arise in AI is **vast**.
- Identify a small number of **dimensions** along which task environments can be categorized.
- These dimensions determine, to a large extent, the appropriate agent design.

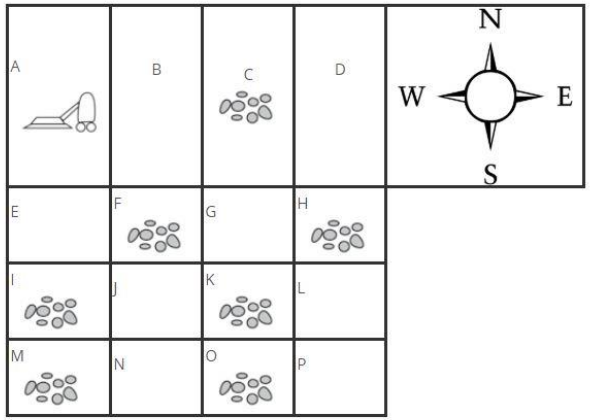
1. Fully observable vs partially observable
2. Deterministic vs stochastic
3. Episodic vs sequential
4. Static vs dynamic
5. Discrete vs continuous
6. Single agent vs multiagent

Properties of Task Environments

- **Fully observable and Partially Observable**
- **Fully Observable :** Sensors give it access to the complete state of the environment at each point in time.
- Sensors detect all aspects that are relevant to the choice of action.
- Convenient

Properties of Task Environments

- **Partially Observable :** Environment partially observable because of noisy, missing and inaccurate sensors.
- Example, Vacuum agent , Automated Taxi (cannot see what other agents are thinking)
- If the agent has no sensors at all then the environment is **unobservable**.
  - Example: Industrial Robots

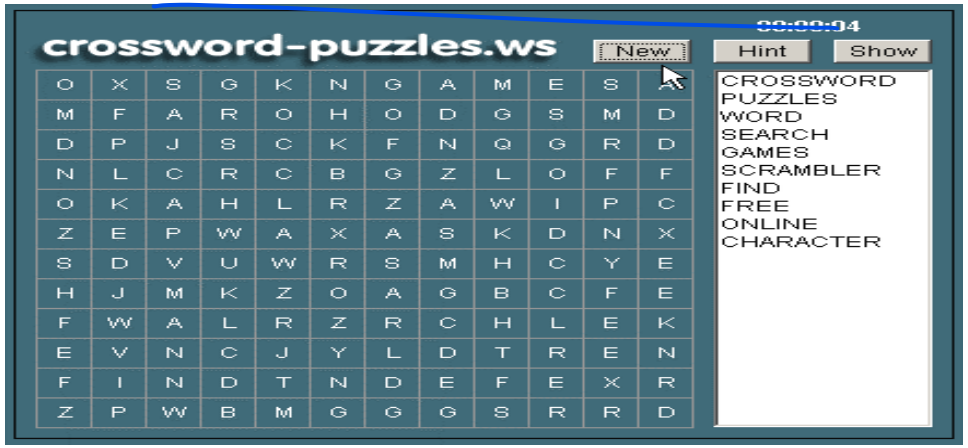


Single agent v/s multi-agent

An agent solving a crossword puzzle by itself is clearly in a single-agent environment

An agent playing chess is in a two-agent environment. which entities must be viewed

Which entities must be viewed



Single agent v/s multi-agent

Chess: The opponent entity is trying to maximize its performance measure, which, by the rules of chess, minimizes agent 's performance measure. Thus, chess is a **competitive multiagent environment.**

Taxi-driving environment: Avoiding collisions maximizes the performance measure of all agents, so it is a partially **cooperative multiagent environment**

Single agent v/s multi-agent

Single-agent

- If only one agent is involved in an environment, and operating by itself then such an environment is called single agent environment.
- **Example**: maze

Multi-agent

- if multiple agents are operating in an environment, then such an environment is called a multi-agent environment.
- **Example**: football

Deterministic agent v/s Nondeterministic agent

- Deterministic : If the next state of the environment is completely determined by the current state and the action executed by the agent(s), then the environment is deterministic;
- There is no uncertainty in a **fully observable**, deterministic environment
- **Example**: Vacuum Cleaning Agent
- Nondeterministic: If the environment is **partially observable**, then it could appear to be nondeterministic.
- Example: Taxi Driving

Deterministic agent v/s Nondeterministic agent

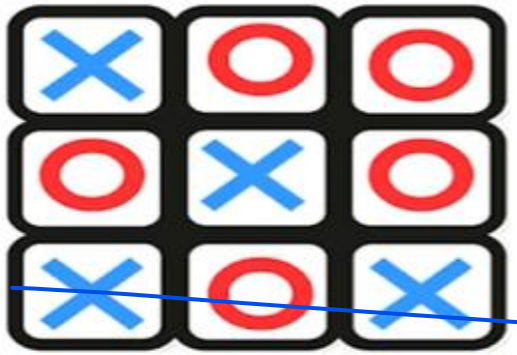
- **Stochastic** is used by some as a synonym for "nondeterministic,"
- A model of the environment is **stochastic** if it explicitly deals with probabilities (e.g., "there's a 25% chance of rain tomorrow") .
- "nondeterministic" if the possibilities are listed without being quantified (e.g., "there's a chance of rain tomorrow").

RAMAIAH
Institute of Technology

Deterministic v/s stochastic

Deterministic

- An agent's current state and selected action can completely determine the next state of the environment.
- **Example:** Tic tac toe



Stochastic

- A stochastic environment is random in nature and cannot be determined completely by an agent.
- **Example**: Ludo (Any games that involve dice)

RAMAIAH
Institute of Technology

Episodic v/s sequential

Episodic

- Only the current percept is required for the action
- Every episode is independent of each other
- **Example**: part picking robot

Sequential

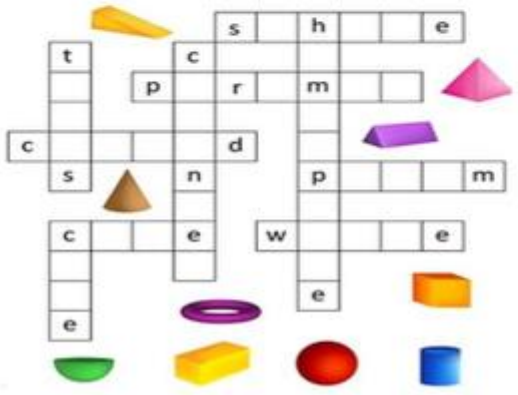
- An agent requires memory of past actions to determine the next best actions.
- The current decision could affect all future decisions.
- **Example**: Chess

RAMAIAH
Institute of Technology

Static v/s dynamic

Static

- The environment does not change while an agent is acting
- **Example**: Crossword puzzles



Dynamic

- The environment may change over time
- **Example**: roller coaster ride

  Taxi driving

RAMAIAH
Institute of Technology

Static v/s dynamic

If the environment itself does not change with the passage of time but the agent's performance score does, then we say the environment is **semidynamic**.

Example: Chess, when played with a clock, is semidynamic.

RAMAIAH
Institute of Technology

Discrete v/s continuous

- The discrete/continuous distinction applies to the state of the environment, to the way time is handled, and to the percepts and actions of the agent

Discrete

- The environment consists of a finite number of actions that can be deliberated in the environment to obtain the output.
- **Example**: chess

Continuous

- The environment in which the actions performed cannot be numbered ie.,is not discrete, is said to be continuous.
- **Example**: Self-driving cars

RAMAIAH
Institute of Technology

Discrete v/s continuous

The discrete/continuous distinction applies to the *state* of the environment, to the way *time* is handled, and to the *percepts* and *actions* of the agent.

Discrete- Cross Word Puzzle, Poker, Input to Digital camera

Continuous- Taxi Driving(Steering)

Known v/s Unknown

This distinction refers not to the environment itself but to the **agent's (or designer's) state of knowledge** about the "laws of physics" of the environment.

In a known environment, the outcomes (or outcome probabilities if the environment is nondeterministic) for all actions are given.

In a unknown environment is the agent will have to learn how it works in order to make good decisions.

RAMAIAH
Institute of Technology

Known v/s Unknown

Known environment can be partially observable—for example, in solitaire card games.

Unknown environment can be fully observable—in a new video game

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
English tutor	Partially	Multi	Stochastic	Sequential	Dynamic	Discrete

sequential, dynamic, continuous, multi-agent

Unit I

Introduction: (Chapter 1 of Text Book 1)

- What is AI?

Intelligent Agents: (Chapter 2 of Text Book 1)

- Agents and Environments
- Rationality
- The Nature of Environments
- **The Structure of Agents**

Solving Problems by search: (Chapter 3 of Text Book 1)

- Problem-Solving Agents
- Example Problems
- Search Algorithms
- Best- first search
- Uniformed Search Strategies- Breadth-first search
- Dijkstra‘s algorithm or uniform-cost search
- Informed (Heuristic) Search Strategies- Greedy best-first search
- A* search.

The Structure of Agents

- Till now we discussed about agents by describing behavior—the action that is performed after any given sequence of percepts.
- Job of AI is to design an **agent program** that implements the **agent function**—the mapping from percepts to actions.
- **Assumption**: This program runs on some sort of computing device with physical sensors and actuators— called as the agent architecture.

- An intelligent agent is a combination of Agent Program and Architecture.
  - Intelligent Agent = Agent Program + Architecture

The Structure of Agents

- Program chosen has to be appropriate for the architecture.
- *The architecture makes the precepts from the sensors available to the program, runs the program, and feeds the program's action choices to the actuators as they are generated.*
- Architecture is a computing device used to run the agent program.

RAMAIAH
Institute of Technology

Agent Programs

- Agent Program take the current percept as input from the sensors and return an action to the actuators.

  Understand : Agent program vs Agent Function

- Agent program takes the current percept as input
  - Nothing is available from the environment

- Agent function takes the entire percept history
  - If the agent's actions need to depend on the entire percept sequence, the agent will have to remember the percepts.

RAMAIAH
Institute of Technology

Agent Programs

Agent Function: Table-Driven Agent

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*
⋮	⋮
$[A, Clean], [A, Clean], [A, Clean]$	*Right*
$[A, Clean], [A, Clean], [A, Dirty]$	*Suck*
⋮	⋮

Agent Programs

Table-Driven Agent

Agent program that keeps track of the percept sequence and then uses it to index into a table of actions to decide what to do.

```
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
```

RAMAIAH Institute of Technology

Agent Programs

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*

Implementation Issue:

```
percepts = []
table = {}

def table_agent (percept):
    action = True
    percepts.append(percept)
    action = lookup(percepts, table)
    return action
```

```
# create a list(array) of possible percepts
# crate (key, value) pairs of percept   sequence

# define a function with parameter percept
#Boolean variable action
# append new percept in the percepts list defined before
# use a function name lookup() to return action
```

Agent Programs

Table-Driven Agent

- Designer needs to construct a table that contains the appropriate action for every possible percept sequence

- Drawbacks?
  - Huge table
  - Take a long time to construct such a table
  - Even with learning, need a long time to learn the table entries

Agent programs

The **key challenge for AI** is to find out how to write programs that, to the extent possible, produce rational behavior from a smallish program rather than from a vast table.

- Arranged in order of increasing generality:
  - Simple reflex agents
  - Model-based reflex agents
  - Goal-based agents
  - Utility-based agents; and
  - Learning agents

RAMAIAH
Institute of Technology

Simple Reflex Agent

- It is a simplest kind of agent.
- These agents select actions on the basis of the current percept, ignoring the rest of the percept history.
- Example: Vacuum World: An agent program for this agent is:

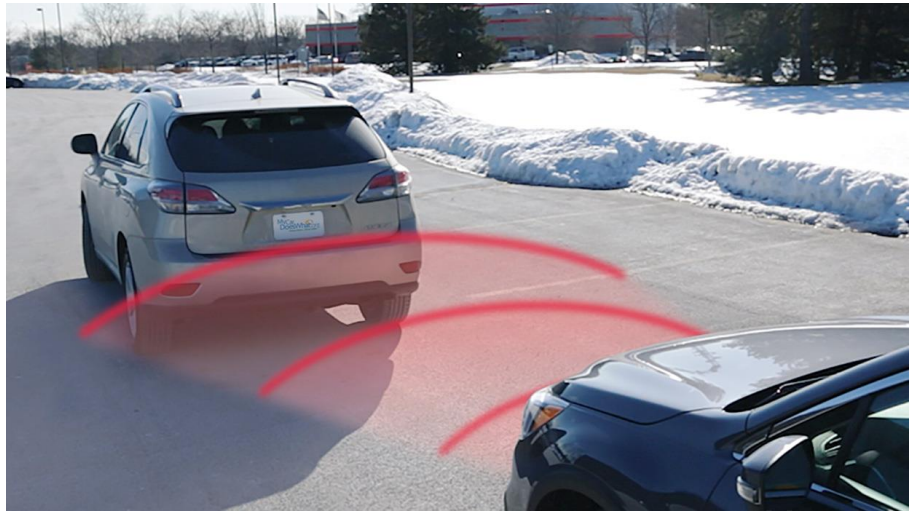
```
function REFLEX-VACUUM-AGENT([location,status]) returns an action
    if status = Dirty then return Suck
    else if location = A then return Right
    else if location = B then return Left
```

Simple Reflex Agent

- It is simple environment.
- Reduction due to :
  - Ignoring the percept history,
  - Reduces number of relevant percept sequences.
  - When the current square is dirty, the action does not depend on the location.

Simple Reflex Agent

- In complex environment as Automated Taxi
  - "The car in front is braking."
  - Such a connection are called condition–action rule, written as
  - if car-in-front-is-braking then initiate-braking.

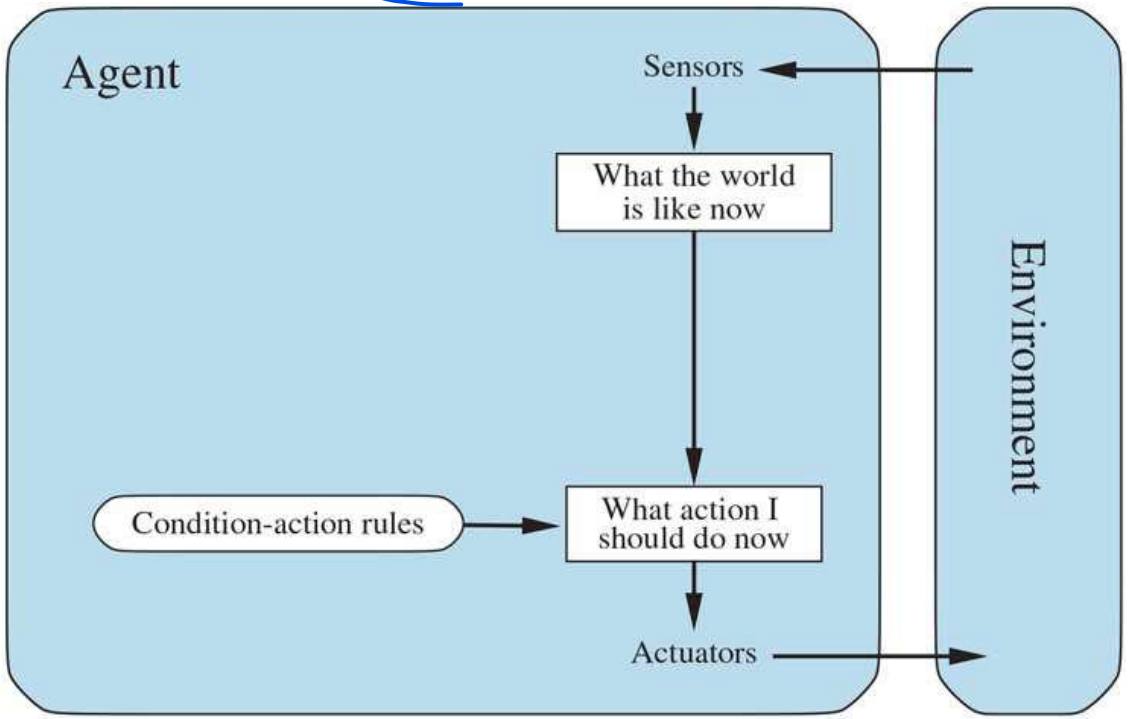


Simple Reflex Agent

- The schematic form, showing how the condition–action rules allow the agent to make the connection from percept to action

Schematic diagram of a simple reflex agent.

- Rectangles to denote the current internal state of the agent's decision process,
- Ovals to represent the background information used in the process.



RAMAIAH
Institute of Technology

Simple Reflex Agent

- A more general and flexible approach is first to build a general-purpose interpreter for condition–action rules and then to create rule sets for specific task environments.

```
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
```

INTERRUPT-INPUT – function generates an abstracted description of the current state from the percept.
RULE-MATCH – function returns the first rule in the set of rules that matches the given state description.
RULE-ACTION – the selected rule is executed as action of the given percept.

Simple Reflex Agent

- Simple reflex agents are **simple** and of **limited intelligence**.
- Will work only if environment is **fully observable.**
- Even a little bit of unobservability can cause serious trouble.
- For example, the braking rule assumes that the condition car-in-front-is-braking can be determined from the
  - Current percept—a single frame of video.
  - Car in front has a centrally mounted brake light.
  - Older models have different configurations of taillights, brake lights, and turn-signal lights.
- A simple reflex agent driving behind such a car would either brake continuously and unnecessarily, or, worse, never brake at all.

RAMAIAH
Institute of Technology

Simple Reflex Agent

- **Infinite loops** are often unavoidable for simple reflex agent operating in partially observable environments
  - No location sensor
- Use Randomize its actions.
- For example, if the vacuum agent perceives , it might flip a coin to choose between Right and Left .

Model-based reflex agents

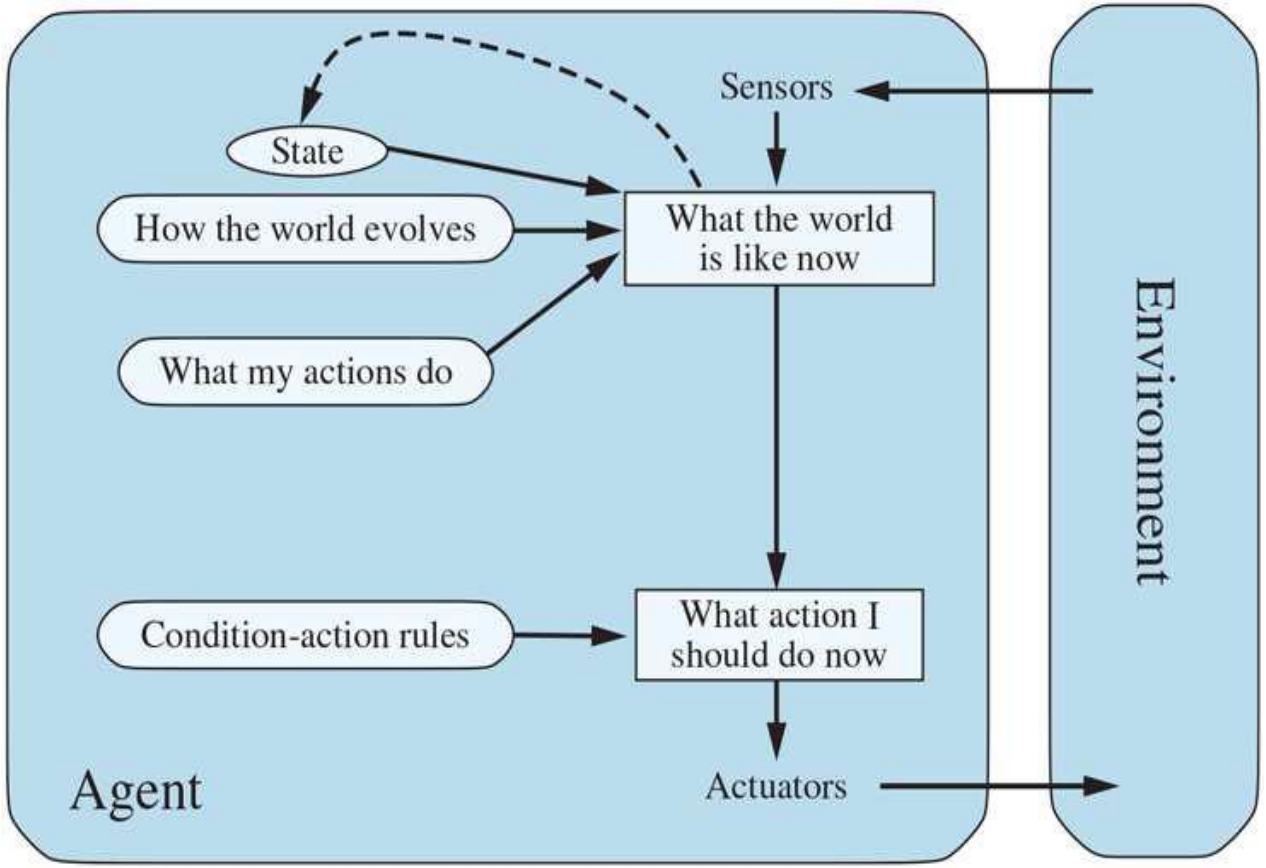
- **Model-based reflex agents** is and intelligent agent that uses percept history and internal memory to make decisions about the model of the world around it.
- This reflects at least some of the unobserved aspects of the current state.
- Agent can handle partial observability to keep track of the part of the world it can't see now.

RAMAIAH
Institute of Technology

Model-based reflex agents

Updating this internal state information requires two kinds of **knowledge**

a. How the world changes over time,
   i. The effects of the agent's actions
   ii. How the world evolves independently of the agent.
b. How the state of the world is reflected in the agent's percepts.

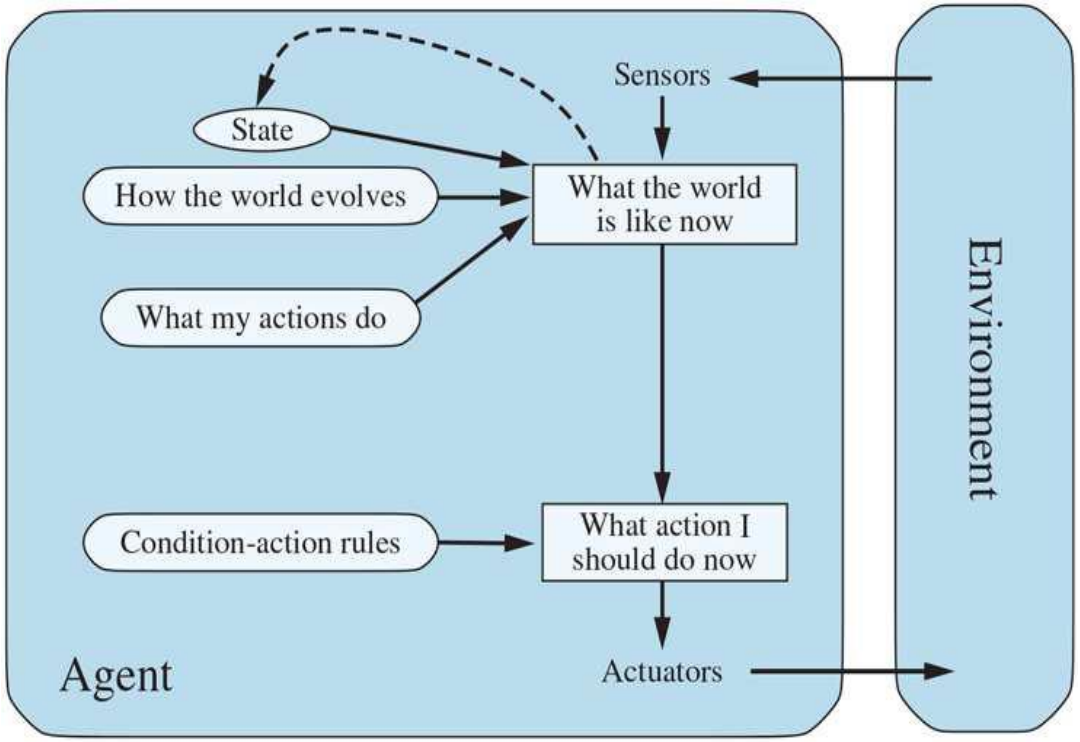


RAMAIAH
Institute of Technology

Model-based reflex agents

1. How the world changes over time,
   i. The effects of the agent's actions
   ii. How the world evolves independently of the agent.

Ex: Agent turns the steering wheel clockwise, the car turns to the right. When it's raining the car's cameras can get wet. This knowledge about "how the world works"—whether implemented in simple Boolean circuits or in complete scientific theories—is called a **transition model** of the world.



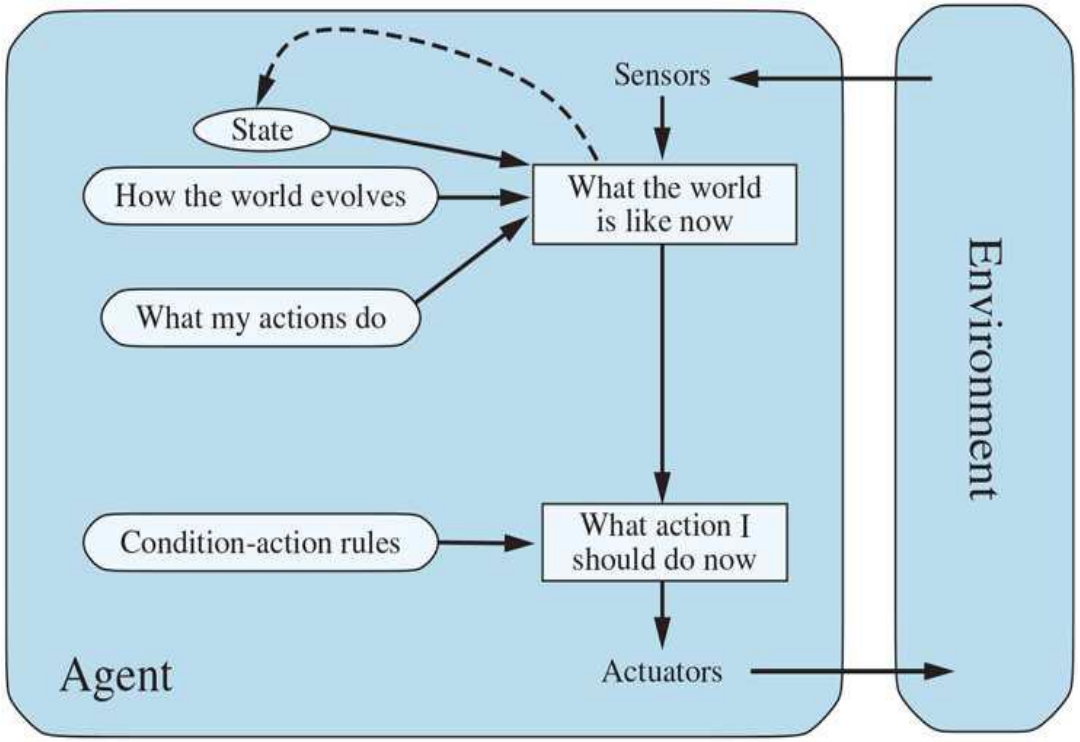
RAMAIAH
Institute of Technology

Model-based reflex agents

2. How the state of the world is reflected in the agent's percepts.

Ex: when the car in front initiates braking, one or more illuminated red regions appear in the forward-facing camera image, and, when the camera gets wet, droplet-shaped objects appear in the image partially obscuring the road.

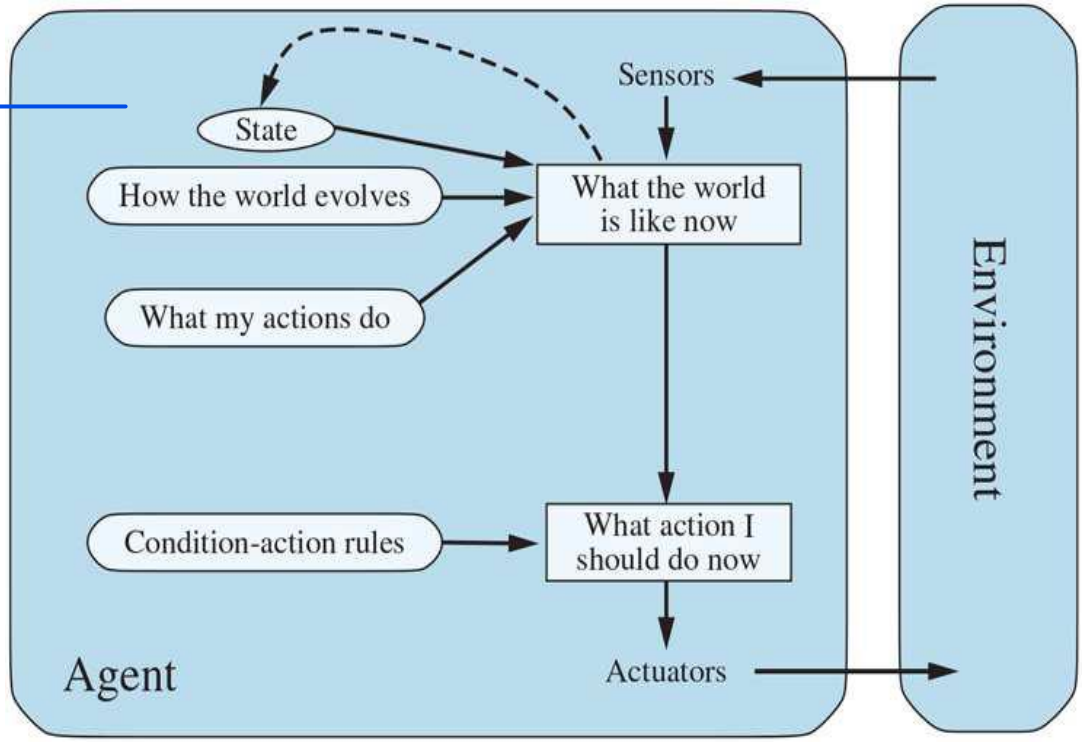
This kind of knowledge is called a **sensor model.**



RAMAIAH
Institute of Technology

Model-based reflex agents

The transition model and sensor model allow an agent to keep track of the state of the world—to the extent possible given the limitations of the agent's sensors. An agent that uses such models is called a **model-based agent.**



Pseudo-Code

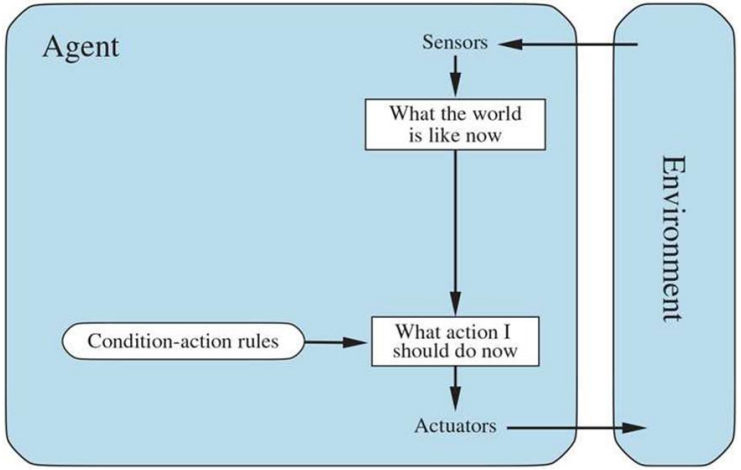
```
function MODEL-BASED-REFLEX-AGENT(percept) returns an action
    persistent: state, the agent's current conception of the world state
                transition_model, a description of how the next state depends on
                    the current state and action
                sensor_model, a description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

    state ← UPDATE-STATE(state, action, percept, transition_model, sensor_model)
    rule ← RULE-MATCH(state, rules)
    action ← rule.ACTION
    return action
```

RAMAIAH Institute of Technology

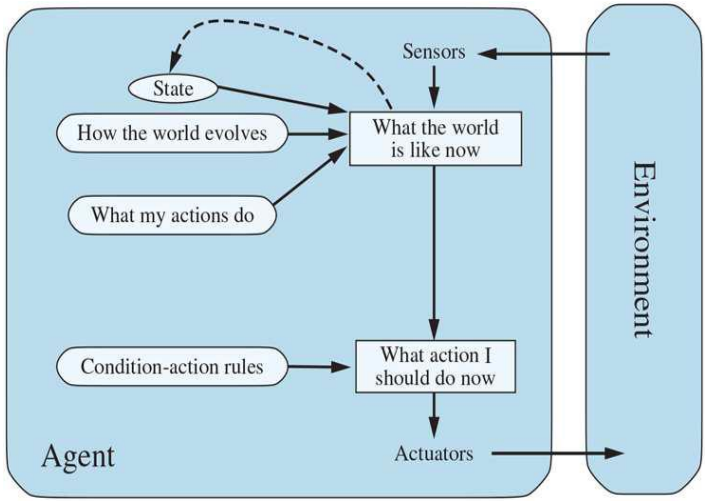
Model-based reflex agents

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
⋮	⋮
[A, Clean], [A, Clean], [A, Clean]	Right
[A, Clean], [A, Clean], [A, Dirty]	Suck
⋮	⋮

Table Driven



Simple Reflex Agent



Model Based Reflex Agent

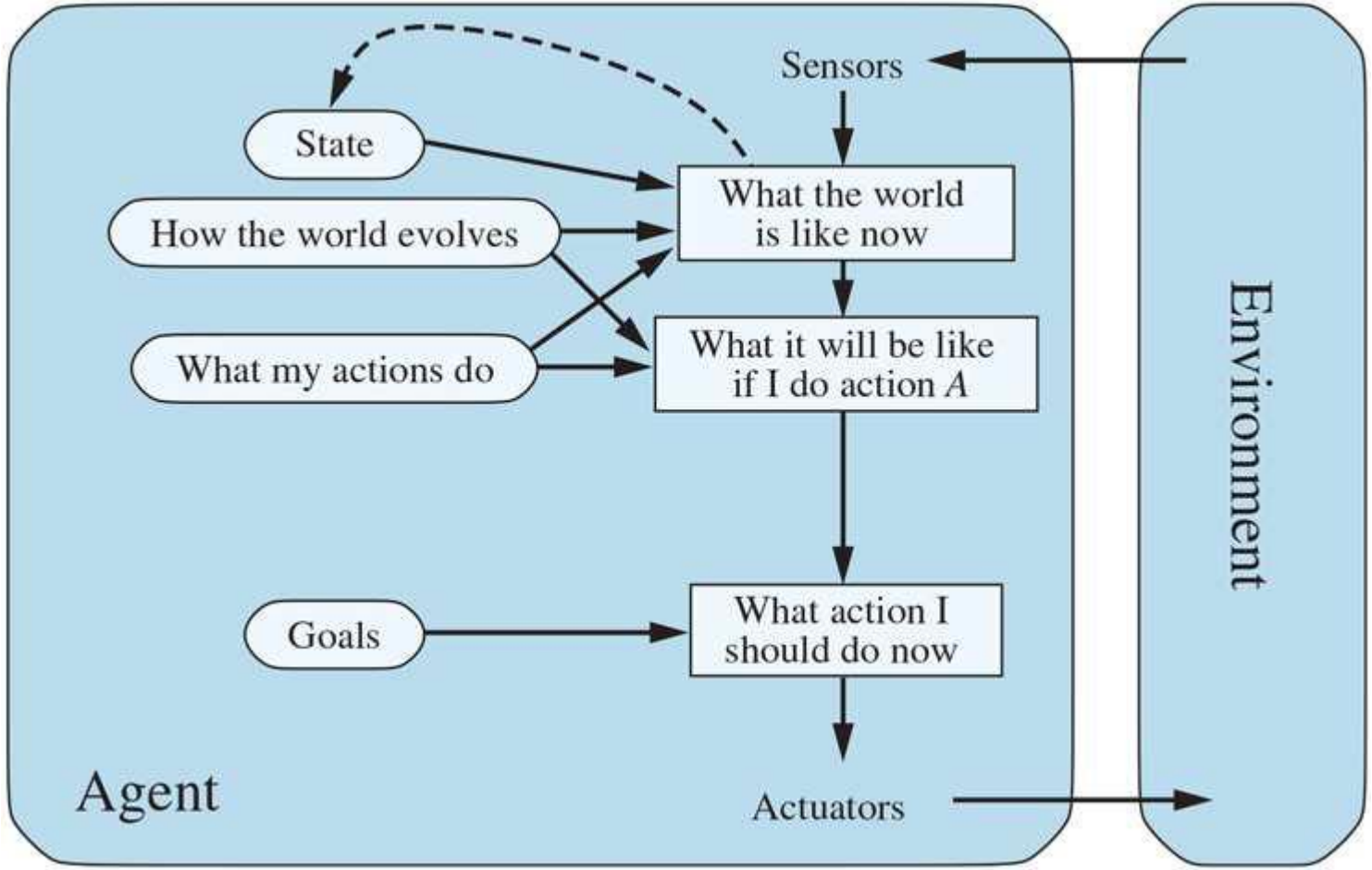
Rectangles to denote the current internal state of the agent's decision process, and ovals to represent the background information used in the process

Goal-based agents

- Knowing something about the **current state** of the environment is not always enough to decide what to do in a situation
- The agent needs some sort of **goal information** that describes situations that are desirable.
- The agent program can **combine this with the model** to choose actions that achieve the goal.

Goal-based agents

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.



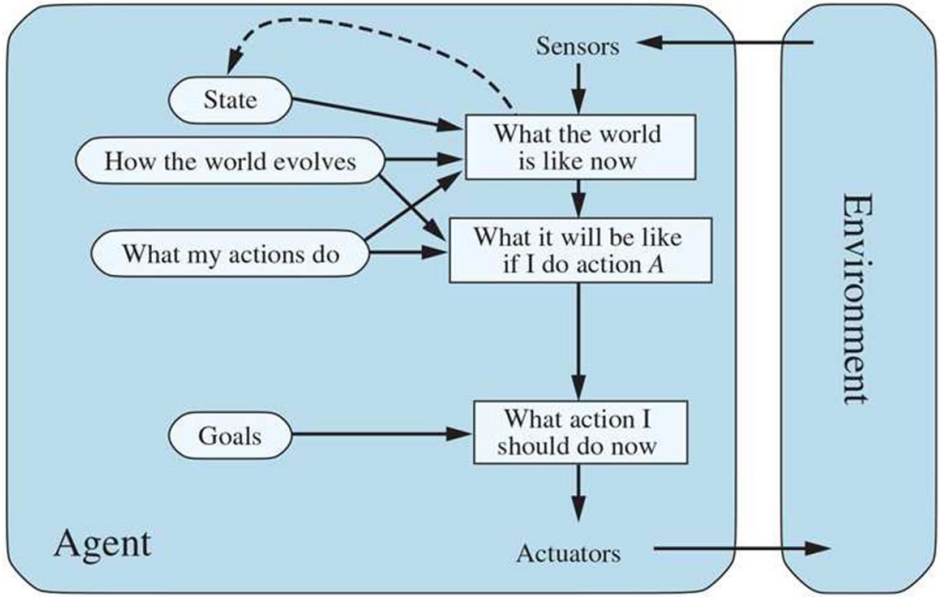
Goal-based agents

- Goal-based action selection is **straightforward** —for example, when goal satisfaction results immediately from a single action.
- **Tricky**—for example, when the agent has to consider long sequences of twists and turns in order to find a way to achieve the goal.
- **Search** and **planning** are the **subfields of AI devoted** to finding action sequences that achieve the agent's goals.

Goal-based agents

- **Decision making** is different from the condition– action rules.
- Goal Based involves consideration of the **future**.
- Both "What will happen if I do such-and-such?" and "Will that make me happy?

- In the reflex agent designs, involves built-in rules map directly from percepts to actions.
- Ex: Car Breaks.

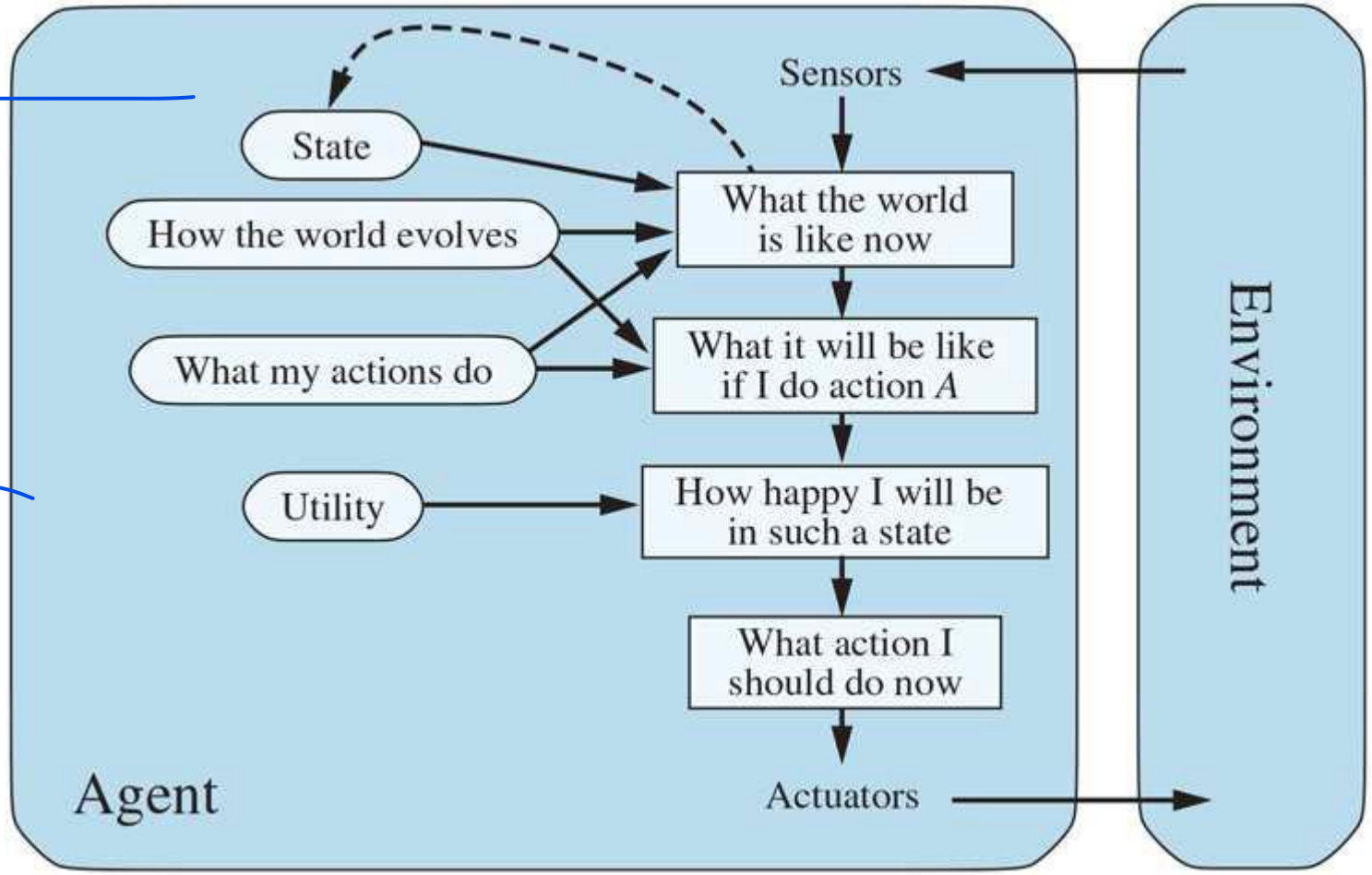


Utility-based agents

- Goals alone are not enough to generate **high-quality behavior** in most environments.
- Utility based agents is an agent that acts for the best way to reach that goal.
- **Ex: Taxi** to its destination (but some are quicker, safer, more reliable, or cheaper than others.)
- Goals just provide a crude binary distinction between “happy” and “unhappy” states.
- Because “happy” does not sound very scientific, economists and computer scientists use the term **utility** instead. Word “utility” here refers to **“the quality of being useful,”**

Utility-based agents

- 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.



Utility-based agents

- **Performance measure** assigns a score to any given sequence of environment states.
- Provided that the **internal utility function** and the **external performance measure** are in agreement, an agent that **chooses actions to maximize** its utility will be rational according to the external performance measure.

Utility-based agents

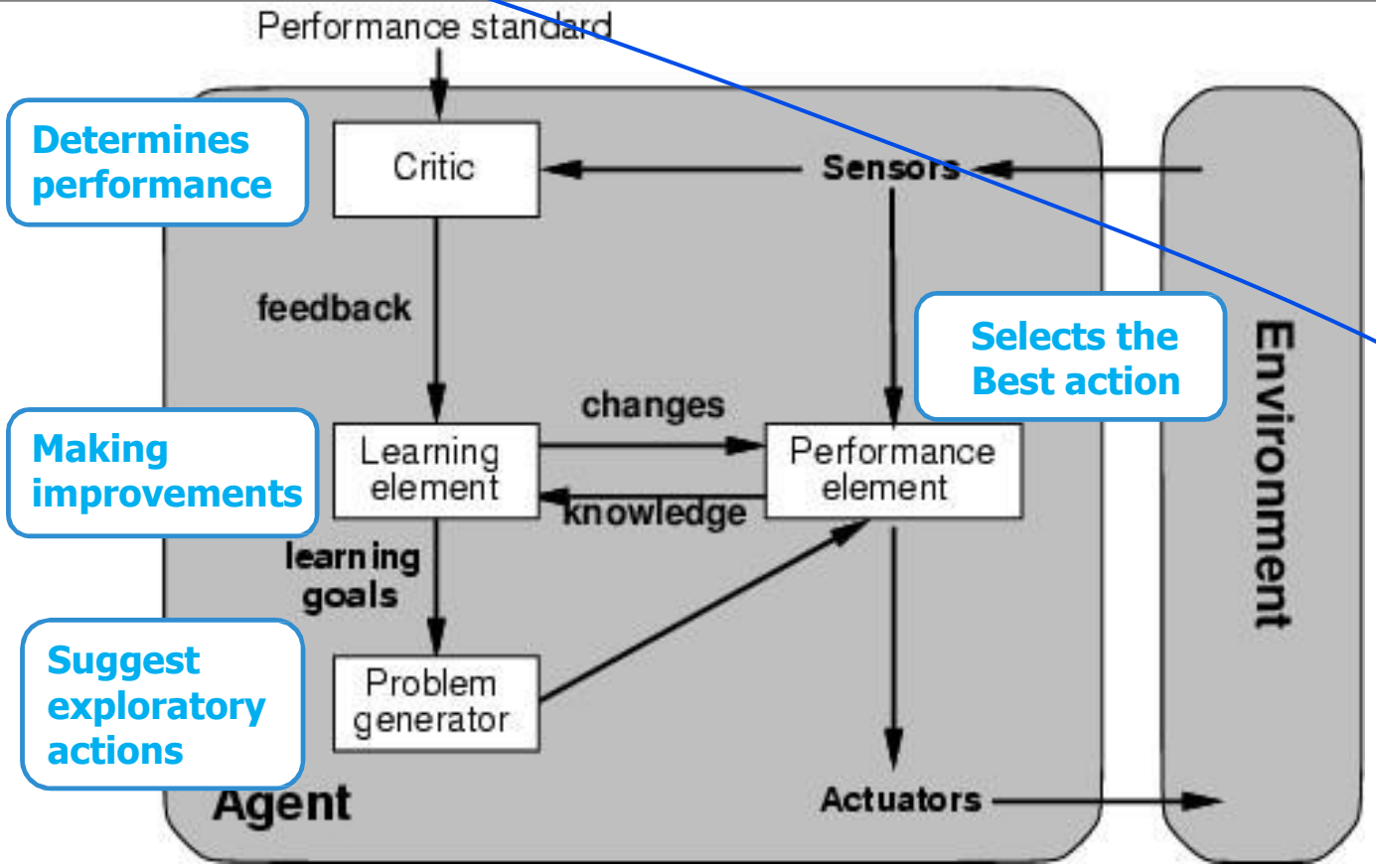
- A utility-based agent has many advantages in terms of flexibility and learning.
- Goals can be inadequate but a utility-based agent can still make rational decisions.
  - First, conflicting goals, only one can be achieved (for example, speed and safety), the utility function specifies the appropriate tradeoff.
  - Second, several goals that the agent can aim for, none of which can be achieved with certainty,
- Utility provides a way in which the likelihood of success can be weighed against the importance of the goals.

Learning Agent

- A learning agent in AI is the type of agent which can learn from its **past experiences** or it has learning capabilities.
- It starts to act with basic knowledge and then able to act and adapt automatically through learning.
- **Four main components:**
- **Performance element**: Selects external action :Takes in percepts and decides on actions
- **Learning element** :It is responsible for making improvements by observing performance
- **Critic**: Learning element takes feedback from critic which describes how well the agent is doing with respect to a fixed performance standard.
- **Problem Generator**: This component is responsible for suggesting actions that will lead to new and informative experiences.

RAMAIAH
Institute of Technology

Learning Agent



A general learning agent. The "performance element" box represents what we have previously considered to be the whole agent program. Now, the "learning element" box gets to modify that program to improve its performance.

Learning Agent

- The **critic** tells the learning element how well the agent is doing with respect to a fixed performance standard.
- The critic is necessary because the percepts themselves provide no indication of the agent's success.
- For example, a chess program could receive a percept indicating that it has checkmated its opponent, but it needs a performance standard to know that this is a good thing; the percept itself does not say so. It is important that the performance standard be fixed.

RAMAIAH
Institute of Technology

Learning Agent

- **Problem generator** is responsible for suggesting actions that will lead to new and informative experiences.
- The problem generator's job is to suggest these exploratory actions

Unit I

Introduction: (Chapter 1 of Text Book 1)

- What is AI?

Intelligent Agents: (Chapter 2 of Text Book 1)

- Agents and Environments
- Rationality
- The Nature of Environments
- The Structure of Agents

Solving Problems by search: (Chapter 3 of Text Book 1)

- **Problem-Solving Agents**
- Example Problems
- Search Algorithms
- Best- first search
- Uniformed Search Strategies- Breadth-first search
- Dijkstra‘s algorithm or uniform-cost search
- Informed (Heuristic) Search Strategies- Greedy best-first search
- A* search.

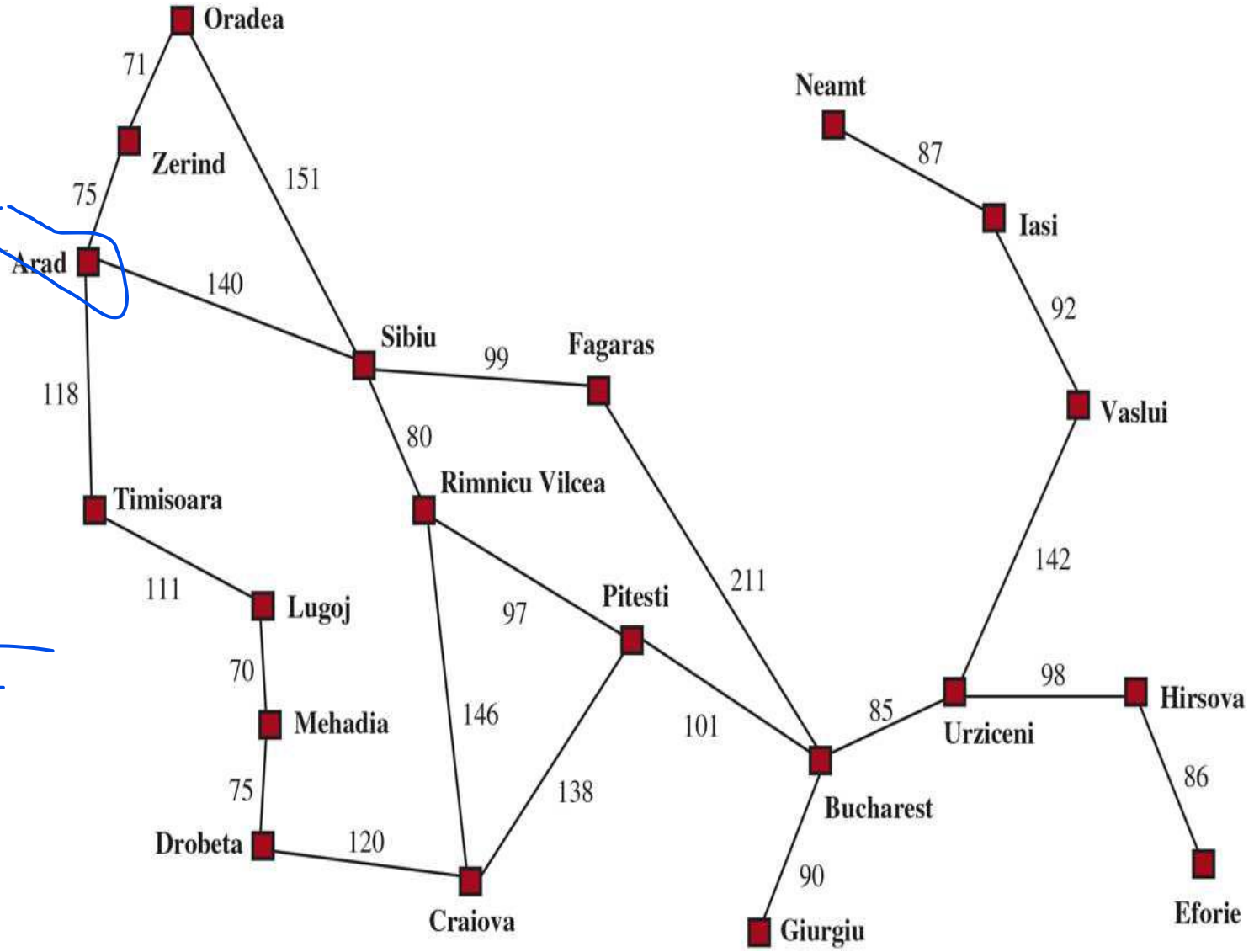
Problem Solving by Search

- When the correct action to take is not immediately obvious, an agent may need to **plan ahead.**
- Considering a **sequence of actions** that form a path to a goal state. Such an agent is called a **problem-solving agent**, and the computational process it undertakes is called **search**.
- Study several **search algorithms.**
- Consider only the **simplest environments:** episodic, single agent, fully observable, deterministic, static, discrete, and known.
- **Informed algorithms,** in which the agent can estimate how far it is from the goal,
- **Uninformed algorithms**, where no such estimate is available.

Problem-Solving Agents

- **Example**: Agent enjoying a touring vacation in Romania. Current Location : Arad

- If the environment is unknown—then the agent can do no better than to execute one of the actions at random.
- Assume our agents always have access to information about the world, such as the map.



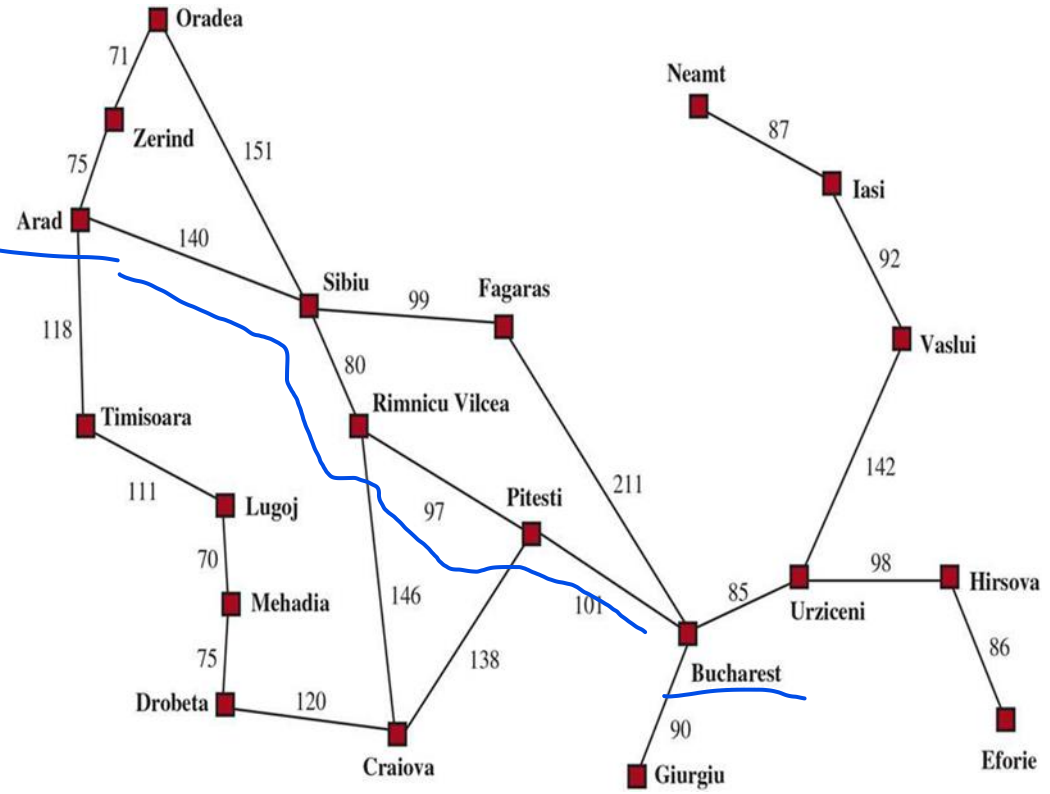
Problem-Solving Agents

- **The Agent can follow Four-phase problem-solving process:**

1. **GOAL FORMULATION:** The agent adopts the goal of reaching Bucharest. Goals **organize behavior** by limiting the **objectives** and hence the **actions** to be considered.
2. **PROBLEM FORMULATION:** The **agent devises a description of the states and actions necessary to reach the goal** an abstract model of the relevant part of the world.

   Ex: **Actions**: Traveling from one city to an adjacent city

   State of the world : Change in current city.

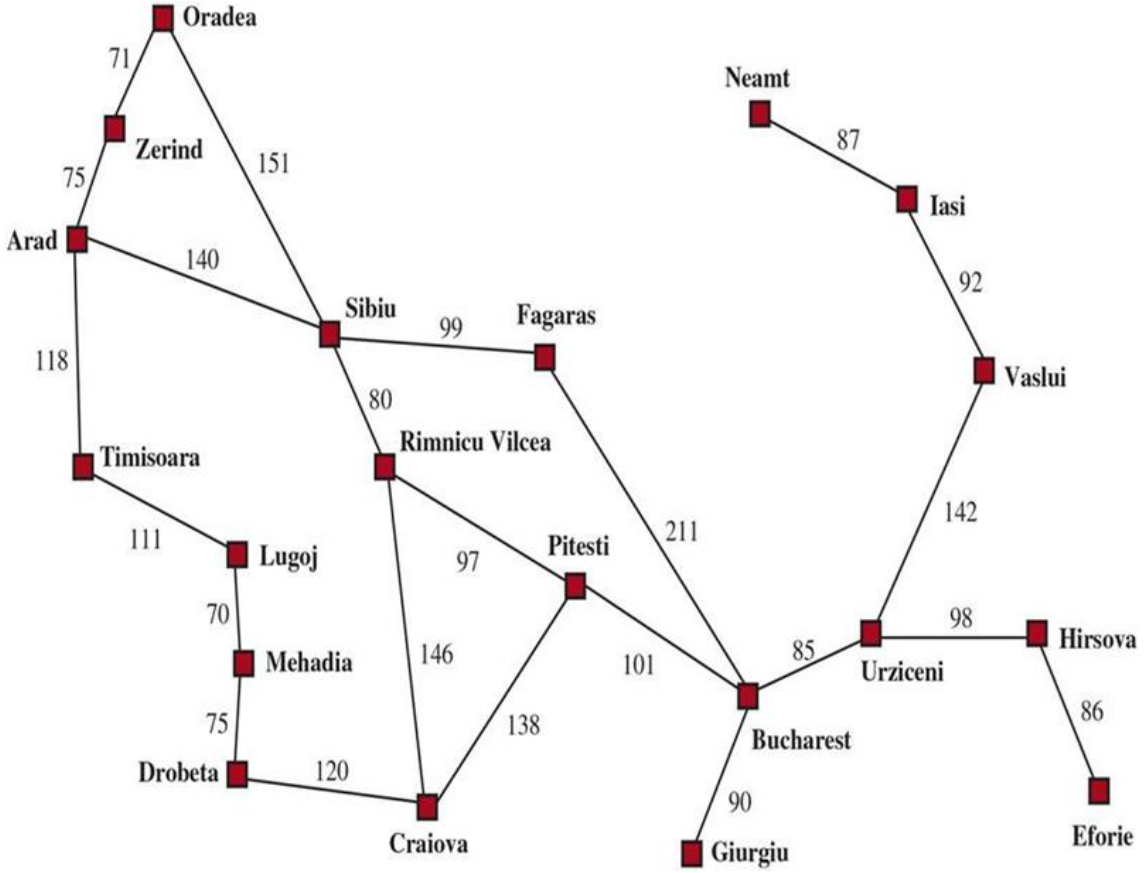


Problem-Solving Agents

3. SEARCH: Before taking any action in the real world, the agent **simulates sequences of actions in its model**, searching until it finds a sequence of actions that reaches the goal. Such a sequence is called a **solution**.

The agent might have to simulate **multiple sequences** that do not reach the goal, or it will find that no solution is possible(Ex: Arad to Sibiu to Fagaras to Bucharest)

4. **EXECUTION:** The agent can now **execute the actions** in the solution, one at a time.

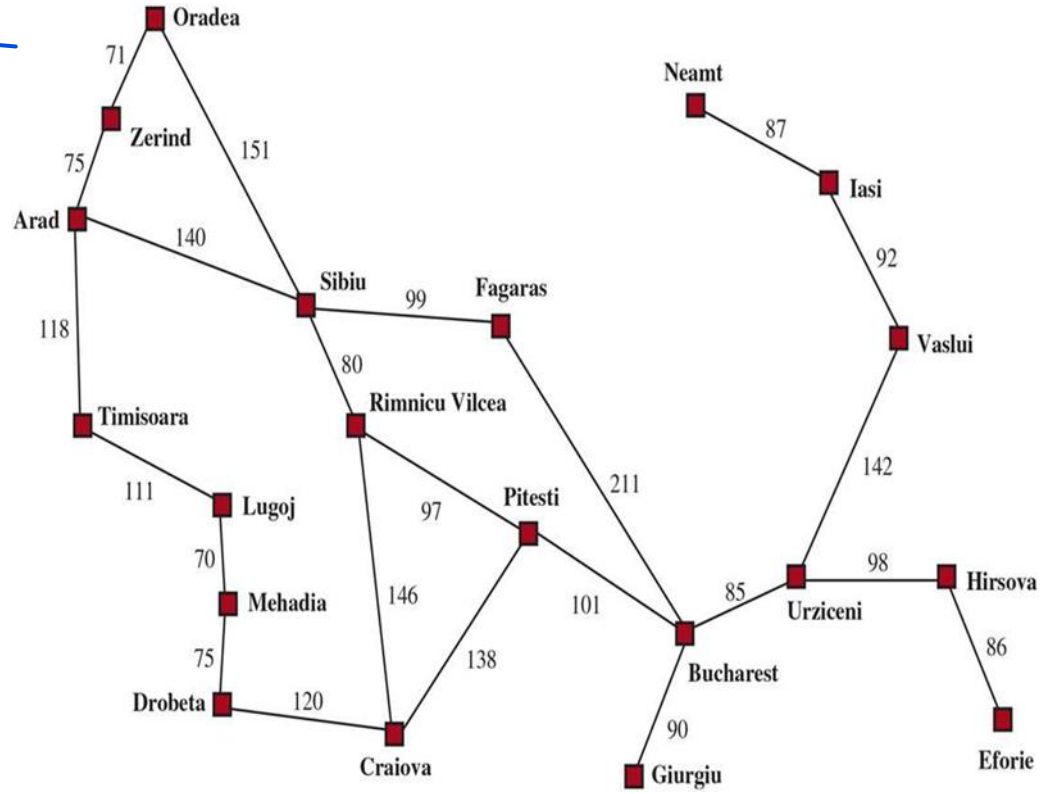


Problem-Solving Agents

Environment : Is fully observable, deterministic, known environment, the solution to any problem is a fixed sequence of actions.

Ex: Drive to Sibiu, then Fagaras, then Bucharest.

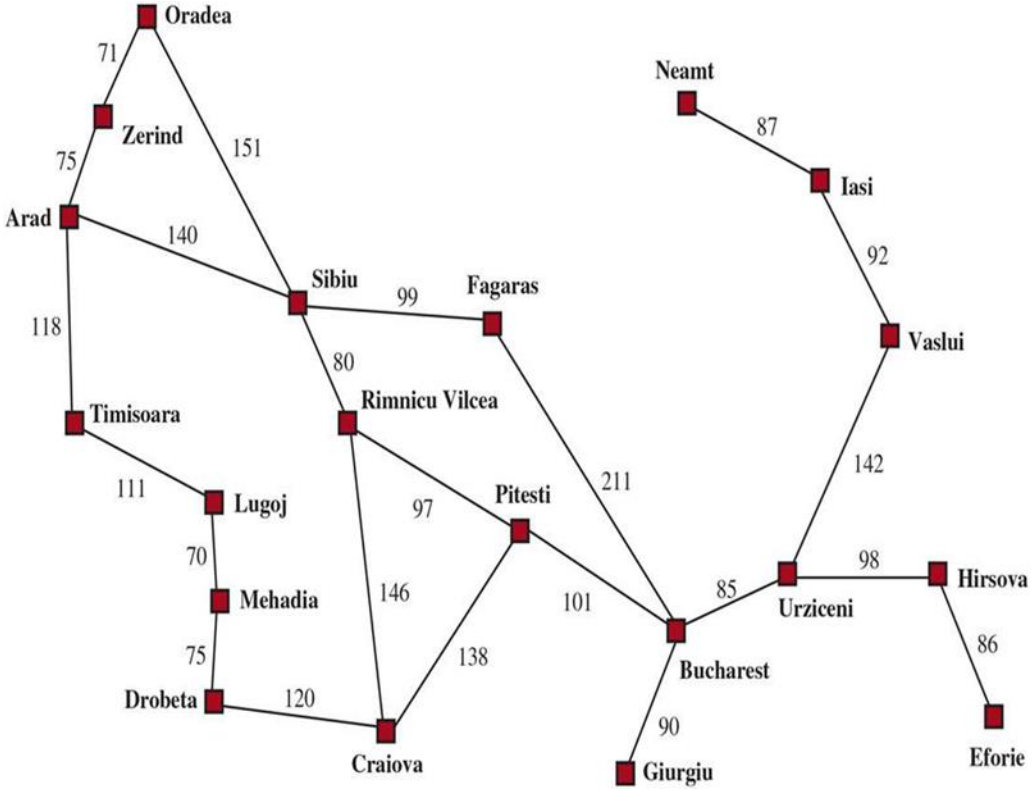
If the **model is correct** then use **Open-loop system:** ignoring the percepts breaks the loop between agent and environment.



Problem-Solving Agents

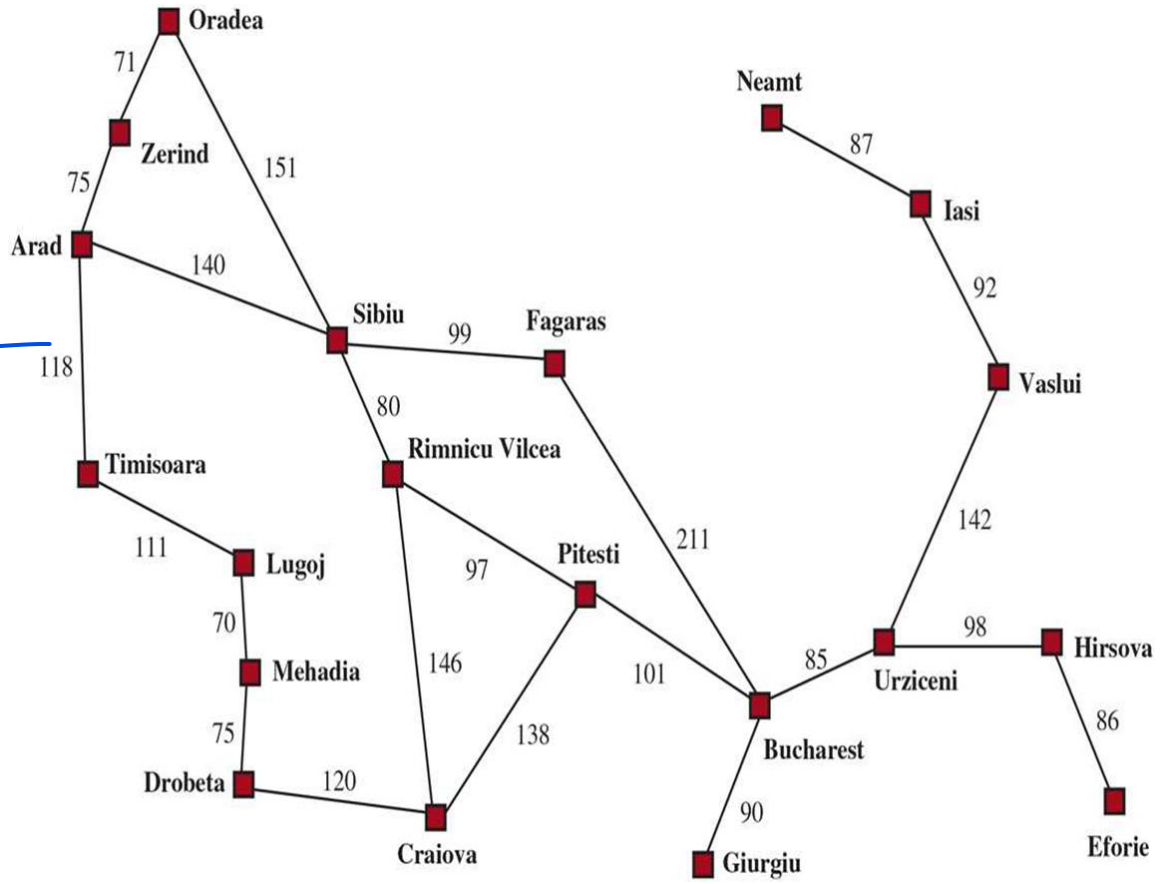
Model is incorrect, or the environment is **nondeterministic**, then the agent would be safer using a **closed-loop approach** that monitors the percepts

Ex: The agent might plan to drive from Arad to Sibiu but might need a contingency plan in case it arrives to a sign saying **Road Closed**.



Search problems and solutions

- A **search problem** can be **defined formally** as follows:
- A set of possible states that the environment can be in called as **state space**.
- The **state space** can be represented as a **graph** in which the **vertices are states** and the **directed edges between them are actions**.
- The **initial state** that the agent starts in. **Ex:** Arad.
- A set of one or more ~~**goal sta**~~**states.**
  - One goal state (e.g., Bucharest),
  - Small set goal states



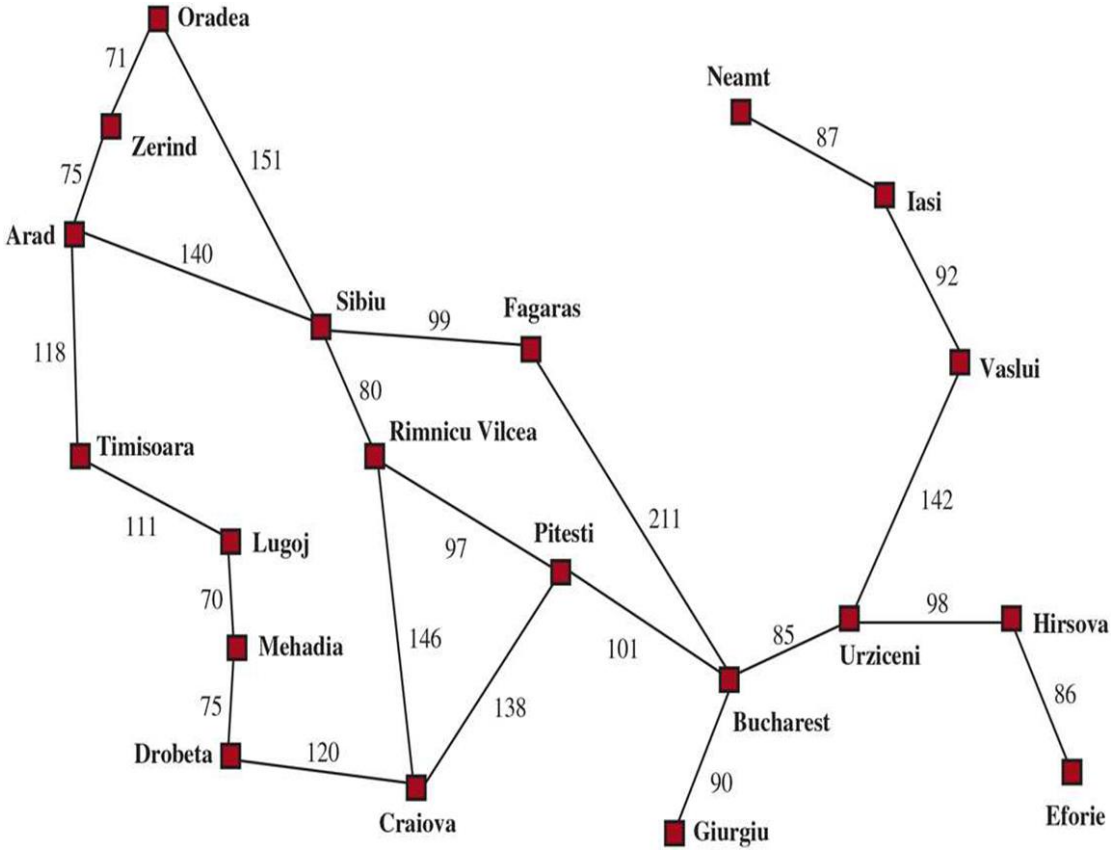
Search problems and solutions

- The **actions** available to the agent. Given a state ACTIONS returns a finite set of actions that can be executed in **s**. Each of these actions is applicable in **s**

  Ex: ACTIONS(Arad) = {ToSibiu, ToTimisoara, ToZerind}.

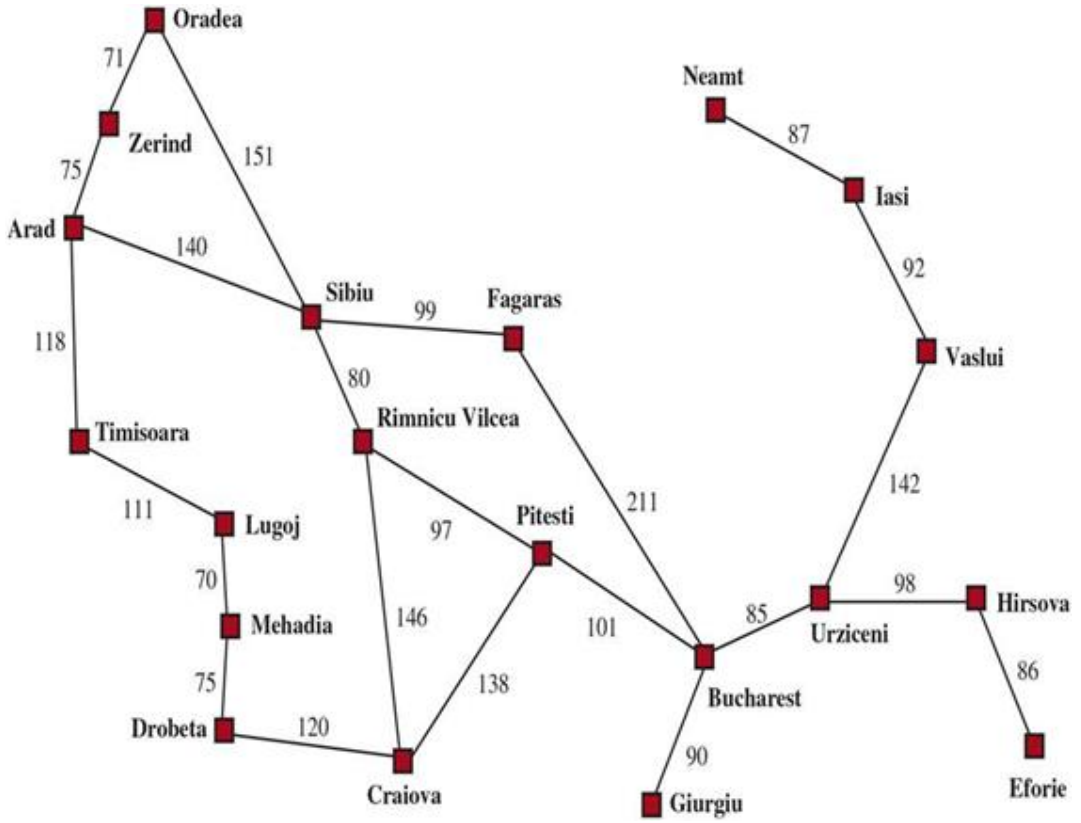
- A **transition model** describes what each action does. **RESULT (s,a)** returns the state that results from doing action in state

  Ex: RESULT(Arad, ToZerind) = Zerind.



Search problems and solutions

- An action cost function, denoted by **ACTION-COST(s,a,s')** gives the numeric cost of applying action **a** in state **s** to reach state **s'**.
- A problem-solving agent should use a cost function that reflects its own **performance measure.**
- **Ex:** For route-finding agents: The cost of an action might be the length in miles or it might be the time it takes to complete the action.



Search problems and solutions

- A sequence of actions forms a **path**
- A **solution** is a path from the initial state to a goal state.
- Action costs are **additive**; the total cost of a path is the **sum** of the individual action costs.
- An **optimal solution** has the lowest path cost among all solutions.
- Assume that all action costs will be **positive**, to avoid certain complications.

Formulating problems

- Formulation of the problem of **getting to Goal is a model**—an abstract mathematical description—and not the real thing.
- In actual **cross-country trip,** the state of the world includes many things: the traveling companions, the current radio program, the scenery out of the window, the proximity of law enforcement officers, the distance to the next rest stop, the condition of the road, the weather, the traffic, and so on. **All these considerations are left out of model**.
- The process of removing detail from a representation is called **abstraction**

Unit I

Introduction: (Chapter 1 of Text Book 1)

- What is AI?

Intelligent Agents: (Chapter 2 of Text Book 1)

- Agents and Environments
- Rationality
- The Nature of Environments
- The Structure of Agents

Solving Problems by search: (Chapter 3 of Text Book 1)

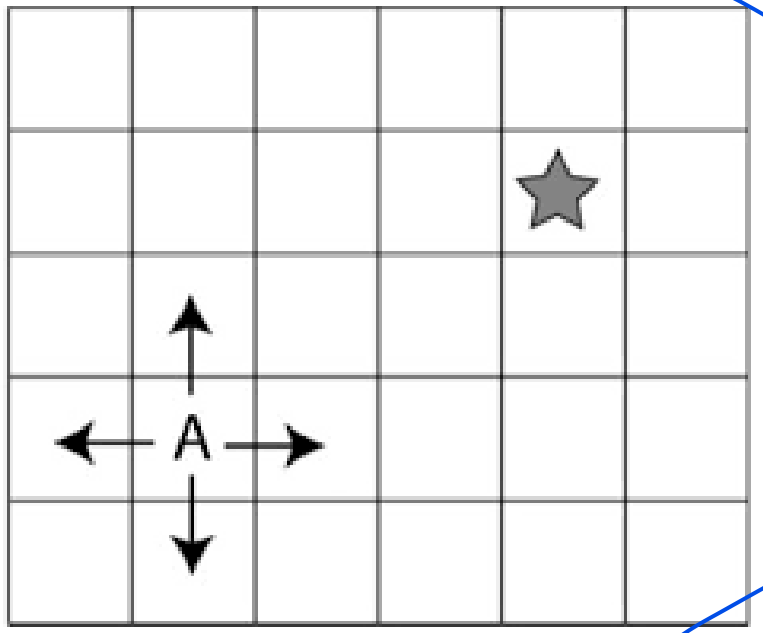
- Problem-Solving Agents
- **Example Problems**
- Search Algorithms
- Best- first search
- Uniformed Search Strategies- Breadth-first search
- Dijkstra‘s algorithm or uniform-cost search
- Informed (Heuristic) Search Strategies- Greedy best-first search
- A* search.

Example Problems

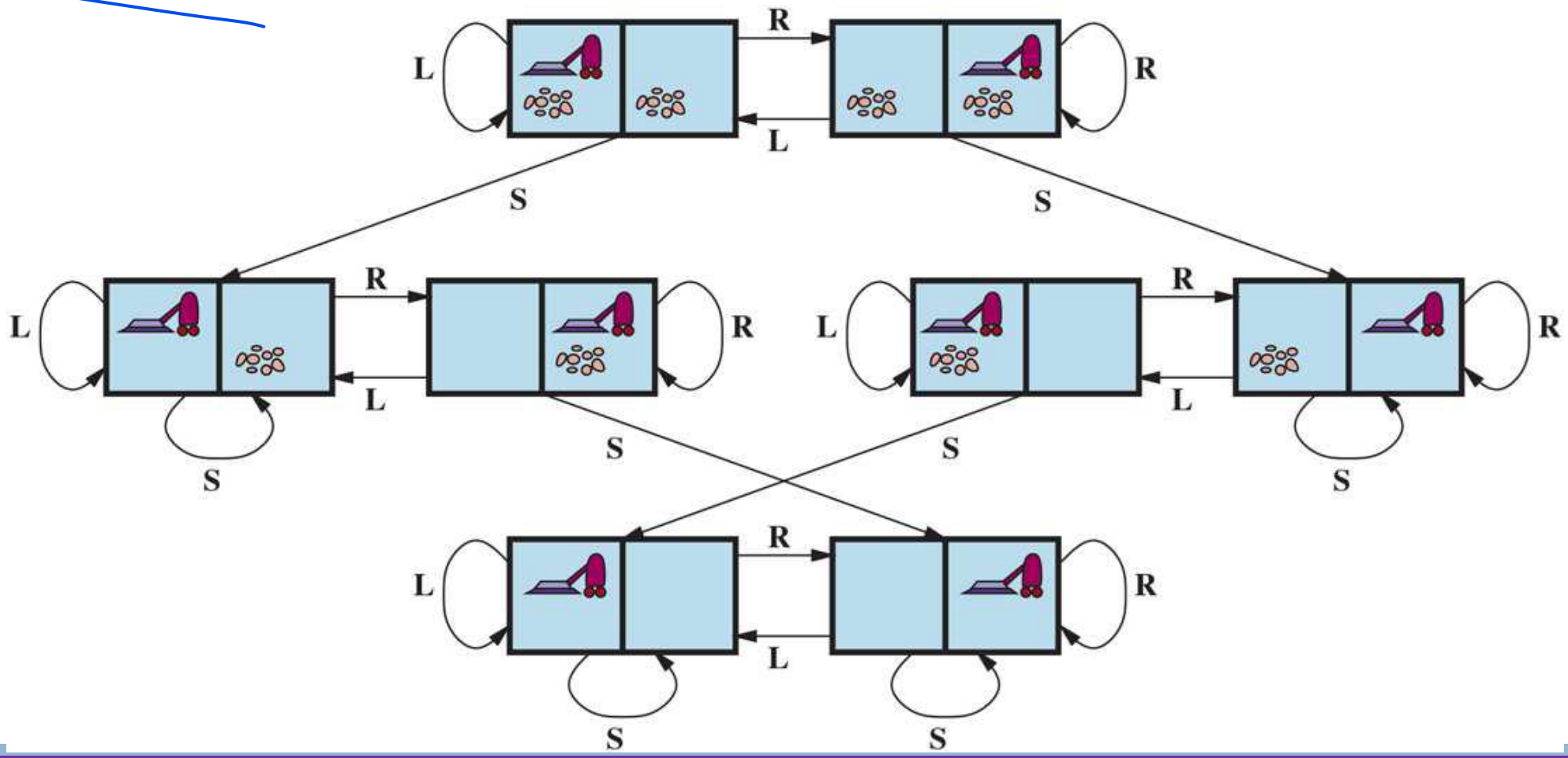
- A **standardized problem** is intended to illustrate or exercise various problem solving methods. It can be given a **concise**, **exact description** and hence is **suitable as a benchmark** for researchers to compare the performance of algorithms.
- A **real-world problem**, such as **robot navigation**, is one whose solutions **people actually use**, and whose formulation is, **not standardized**.
  - Ex: Each robot has different sensors that produce different data.

Standardized problems: Grid World

- A **grid world** problem is a two-dimensional rectangular array of square cells in which agents can move from cell to cell.
- The agent can move to any obstacle-free adjacent cell, horizontally or vertically and in some problems diagonally.
- Cells can contain objects, that can be pick up, push, or otherwise act upon; a wall or other impassible obstacle in a cell prevents an agent from moving into that cell.
- Ex: The vacuum world formulated as a grid world problem as follows:

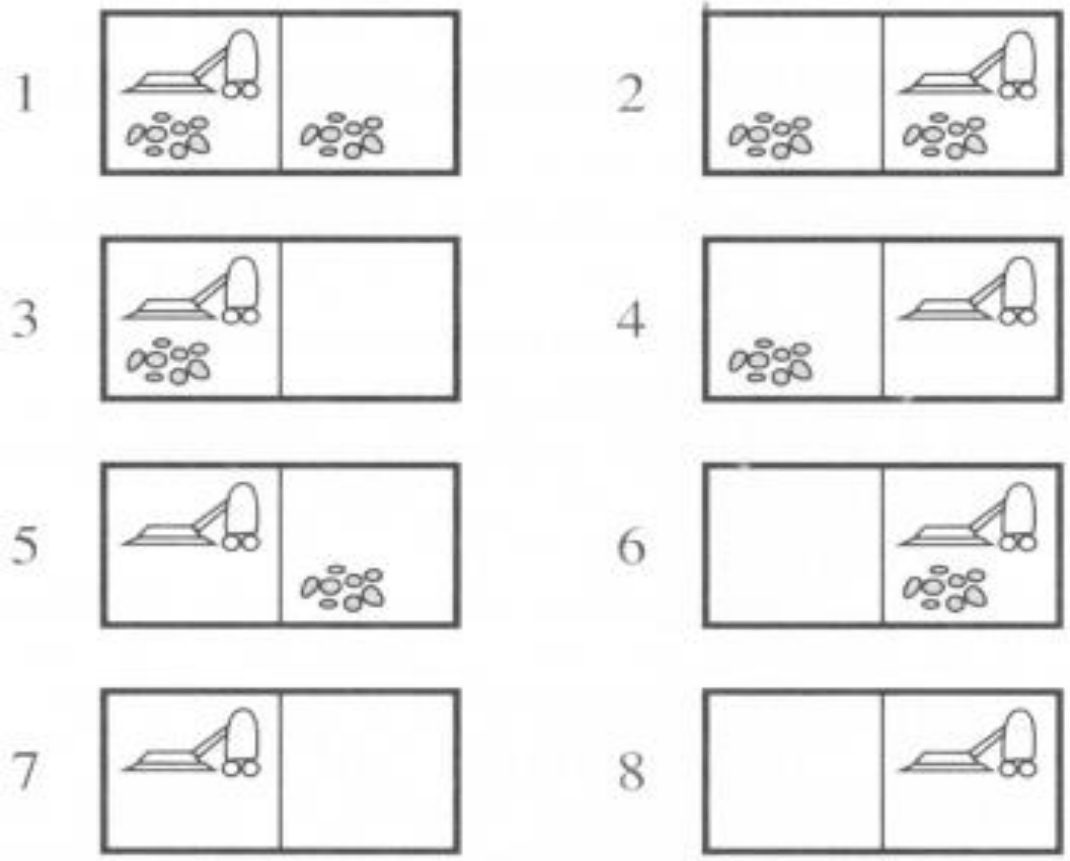


Standardized problems: Grid World: Vacuum World



RAMAIAH Institute of Technology

Standardized problems: Grid World : Vacuum World

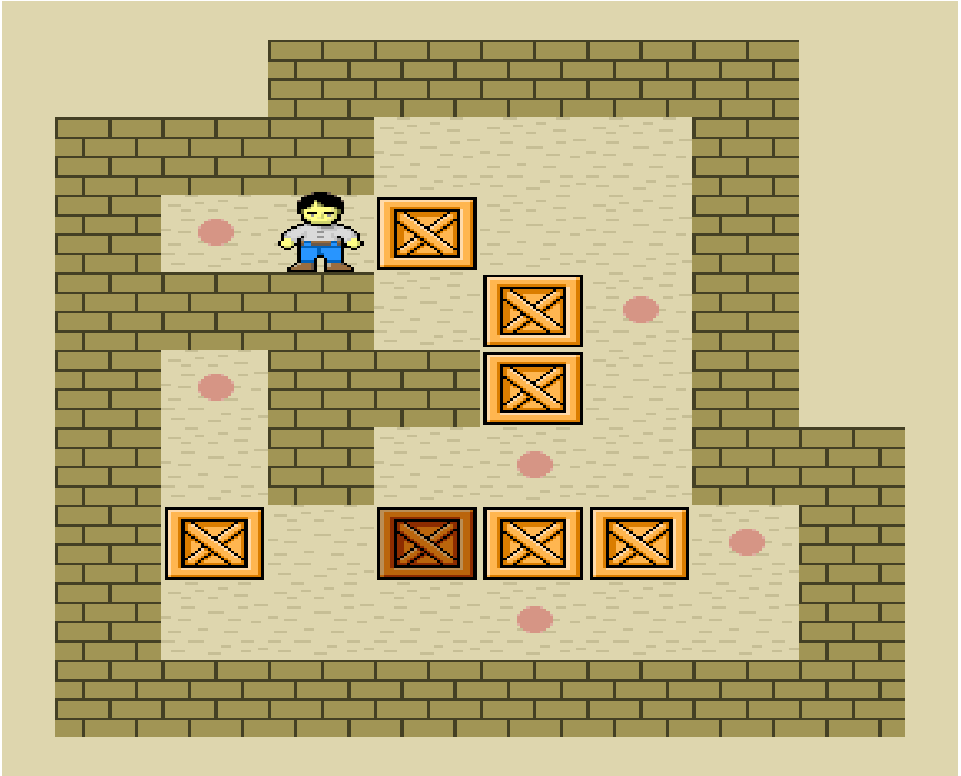


- **STATES:** A state of the world says which objects are in which cells.
  Number of states: $8 = n.2^n$, n is no. of cells
- **INITIAL STATE**: Any
- **ACTIONS:** 3 : left, right, suck
- **TRANSITION MODEL**:
  - Suck removes any dirt
  - Forward moves till it hits a wall: action has no effect.
  - Backward moves in the opposite
  - TurnRight and TurnLeft change the direction
- **GOAL**: clean up all dirt : Goal states: {7, 8}
- **Path Cost**: Each step costs 1

RAMAIAH
Institute of Technology

Standardized problems: Grid World: sokoban puzzle

- Agent's goal is to push a number of boxes, scattered about the grid, to designated storage locations.
- There can be at most one box per cell.
- The agent can't push a box into another box or a wall.



RAMAIAH
Institute of Technology

Standardized problems: Sliding Tile Puzzle

Sliding-tile puzzle: Number of tiles/ blocks/pieces are arranged in a grid with one or more blank spaces so that some of the tiles can slide into the blank space.

Initial configuration

1	2	3
5	6	
7	8	4

Final configuration

1	2	3
5	8	6
	7	4

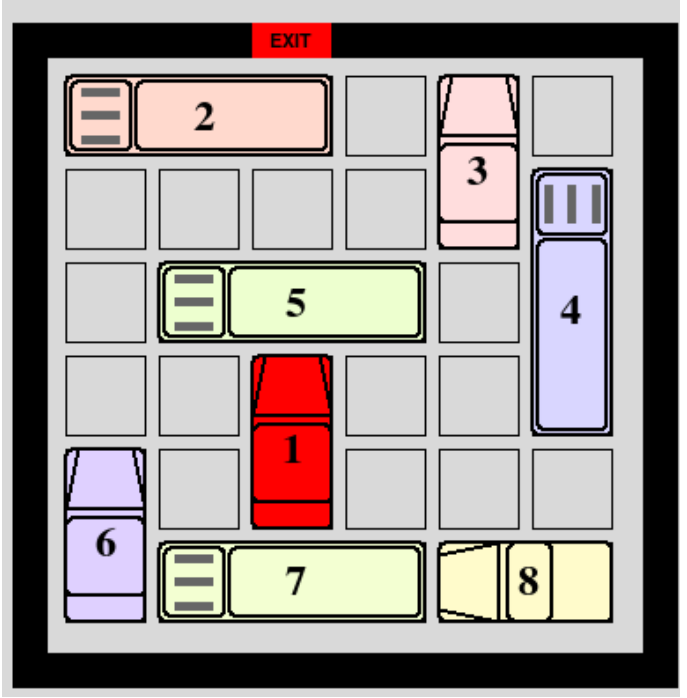
Initial Configration

2	1	3	4
5	6	7	8
9	10	11	12
13	14	15	X

Final Configuration

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	X

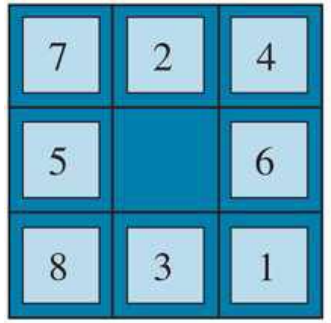
Rush Hour puzzle, in which cars and trucks slide around a grid in an attempt to free a car from the traffic jam



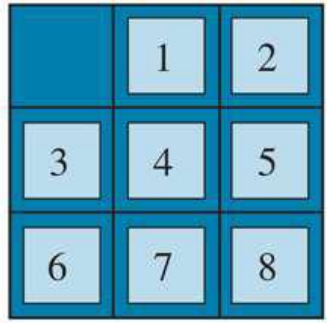
RAMAIAH
Institute of Technology

Standardized problems: Sliding Tile Puzzle

- **The standard formulation of the 8 puzzle is as follows:**
- **STATES**: A state description specifies the location of each of the tiles.
- **INITIAL STATE**: Any state can be designated as the initial state.
- **ACTIONS**: Physical world: Tile slides.
  Describing an action is to think of the blank space moving Left, Right, Up, or Down.
- **TRANSITION MODEL:** Maps a state and action to a resulting state; **Ex**: Apply Left to the start state, the resulting state has the 5 and the blank switched.
- **GOAL STATE:** Numbers in order .
- **ACTION COST:** Each action costs 1.



Start State



Goal State

Unit I

Introduction: (Chapter 1 of Text Book 1)

- What is AI?

Intelligent Agents: (Chapter 2 of Text Book 1)

- Agents and Environments
- Rationality
- The Nature of Environments
- The Structure of Agents

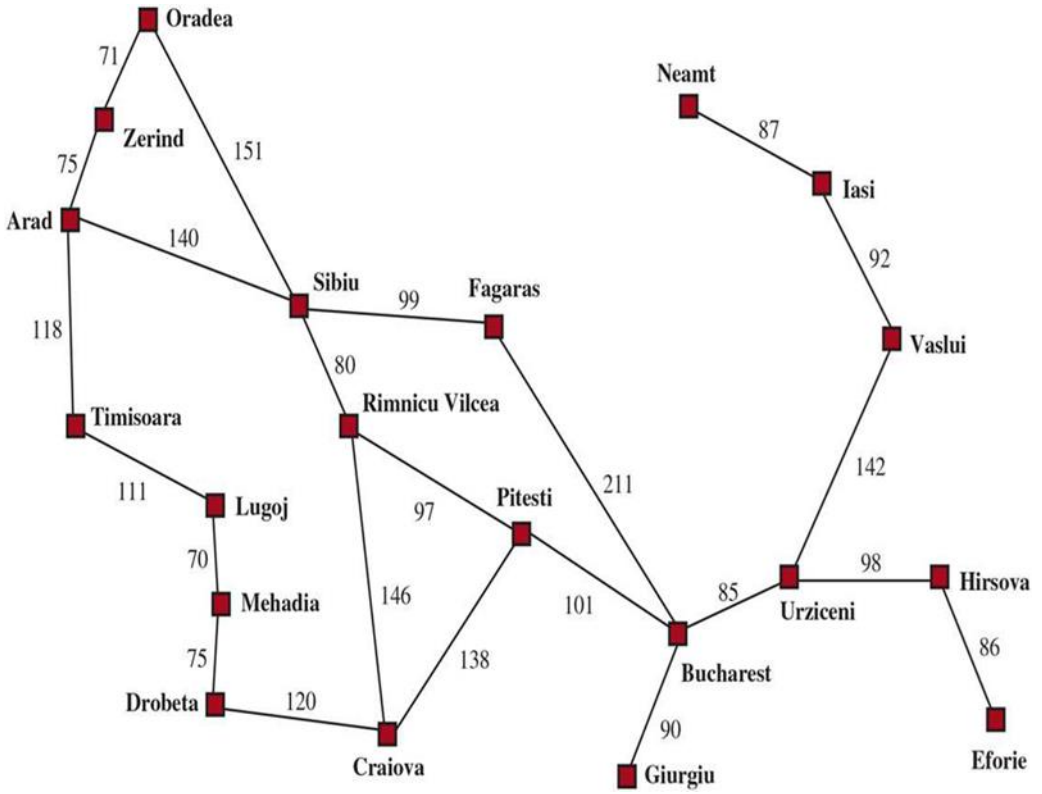
Solving Problems by search: (Chapter 3 of Text Book 1)

- Problem-Solving Agents
- **Example Problems**
- Search Algorithms
- Best- first search
- Uniformed Search Strategies- Breadth-first search
- Dijkstra‘s algorithm or uniform-cost search
- Informed (Heuristic) Search Strategies- Greedy best-first search
- A* search.

RAMAIAH
Institute of Technology

Real-world problems

- **Route-finding problem** is defined in terms of specified locations and transitions along edges between them.
- Route-finding algorithms are used in a variety of applications.
  - Ex: Web sites
  - In-car systems that provide driving directions,
  - Routing video streams in computer networks
  - Military operations planning
  - Airline travel-planning systems



Real-world problems

Example problems

1. Airline travel problems
2. Touring problems
3. Traveling salesperson problem
4. A VLSI layout
5. Robot navigation
6. Automatic assembly sequencing

Real-world problems

Airline travel problems

STATES: Each state obviously includes a location (e.g., an airport) and the current time.

INITIAL STATE: The user's home airport.

ACTIONS: Take any flight from the current location, in any seat class, leaving after the current time, leaving enough time for within-airport transfer if needed.

TRANSITION MODEL: The state resulting from taking a flight will have the flight's destination as the new location and the flight's arrival time as the new time.

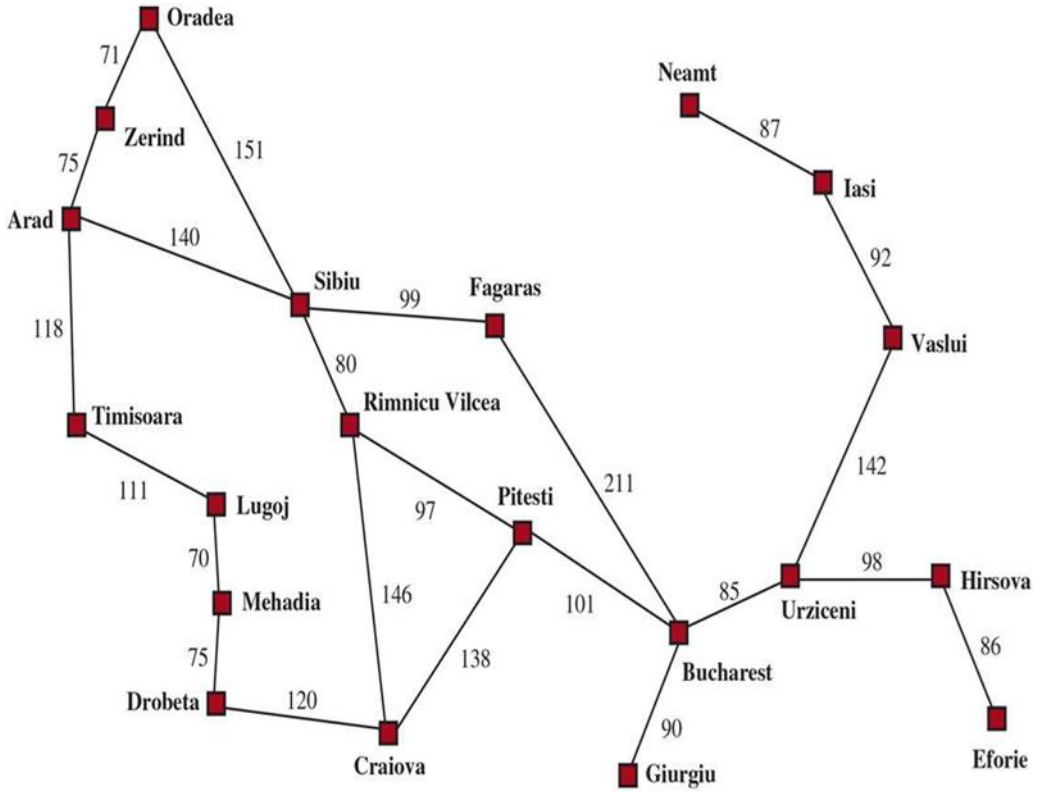
GOAL STATE: A destination city.

ACTION COST: A combination of monetary cost, waiting time, flight time, customs and immigration procedures, seat quality, time of day, type of airplane, frequent-flyer reward points, and so on.

RAMAIAH
Institute of Technology

Real-world problems

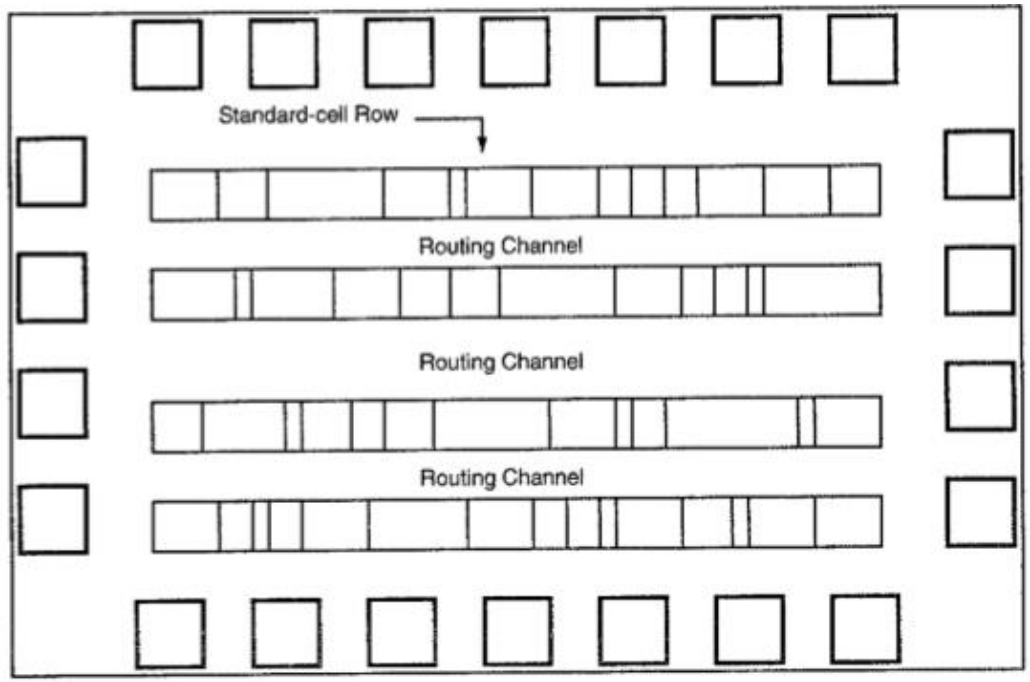
- Touring problems [Traveling salesperson problem (TSP)]



RAMAIAH
Institute of Technology

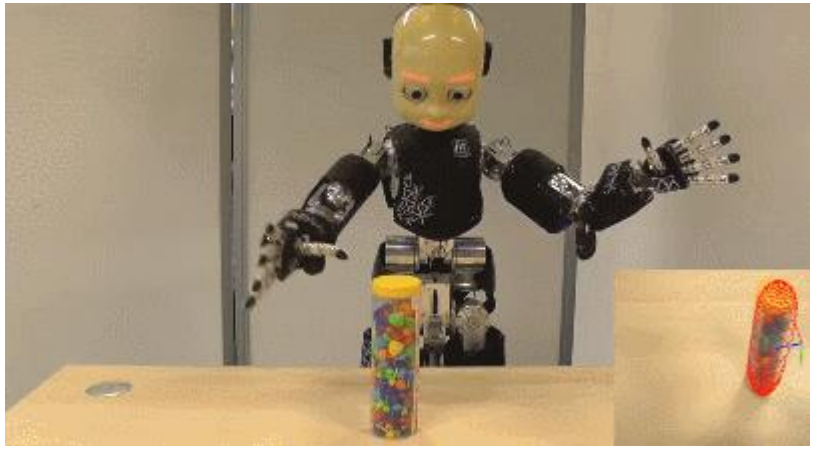
Real-world problems

- **VLSI layout**
- Position millions of components and connections on a chip to minimize area, minimize circuit delays, minimize stray capacitances, and maximize manufacturing yield.
- The layout phase is split into two parts:
  - Cell layout
  - Channel routing.
- The aim is to place the cells on the chip so that they do not overlap
- There is room for the connecting wires to be placed between the cells.
- Channel routing finds a specific route for each wire through the gaps between the cells.
- These search problems are extremely complex, but definitely worth solving.



Real-world problems

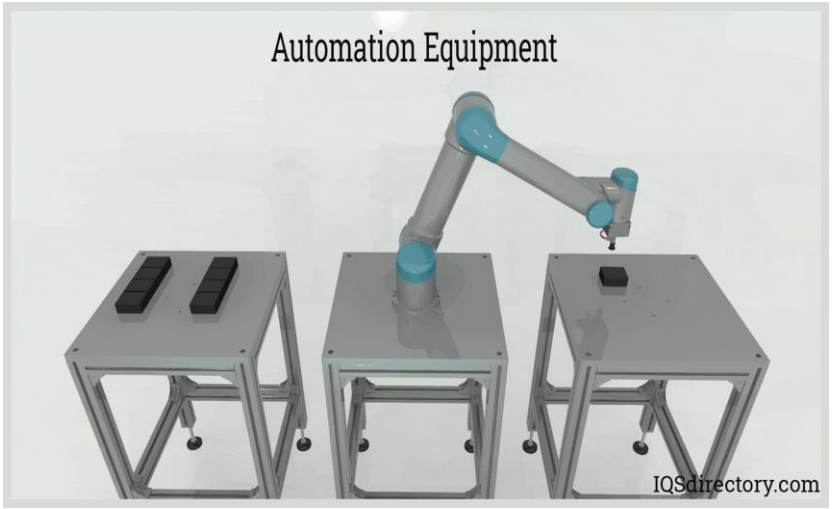
- **Robot Navigation**
- A robot can roam around, in effect making its own paths.
- **Circular robot** moving on a flat surface, the space is essentially two-dimensional.
- **Robot with arms and legs also be controlled:**
- The search space becomes many-dimensional. One dimension for each joint angle.
- In addition to the complexity of the problem, real robots must also deal:
  - Errors in their sensor readings and motor controls,
  - Partial observability, and
  - Other agents that might alter the environment.



RAMAIAH
Institute of Technology

Real-world problems

- **Automatic assembly sequencing of complex objects (such as electric motors)**
- Algorithms first find a feasible assembly sequence and then work to optimize the process.
- In assembly problems, the aim is to find an order in which to assemble the parts of some object.
- If the wrong order is chosen, there will be no way to add some part later in the sequence without undoing some of the work already done.
- Thus, the generation of legal actions is the expensive part of assembly sequencing.



Unit I

Introduction: (Chapter 1 of Text Book 1)

- What is AI?

Intelligent Agents: (Chapter 2 of Text Book 1)

- Agents and Environments
- Rationality
- The Nature of Environments
- The Structure of Agents

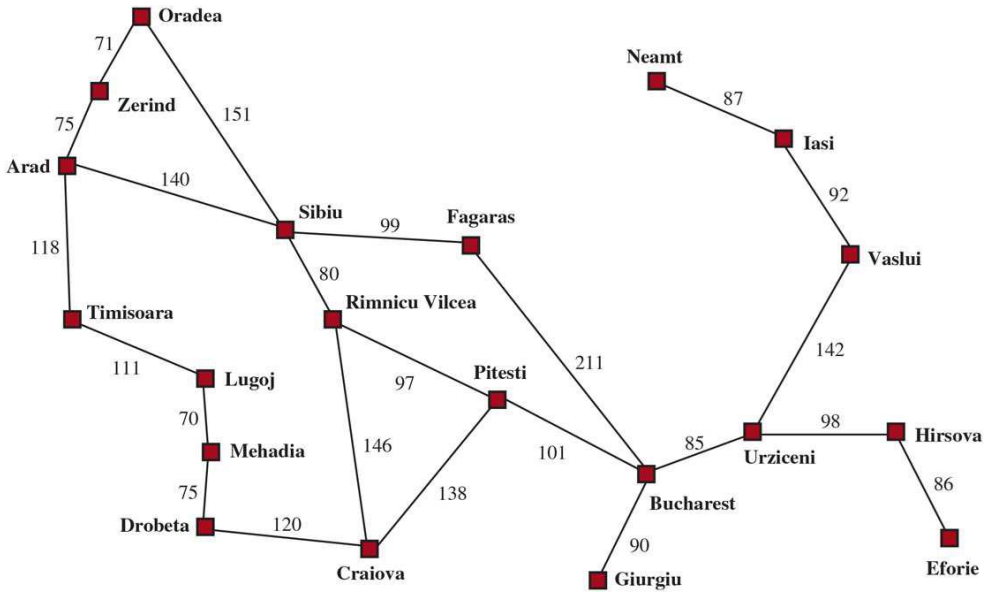
Solving Problems by search: (Chapter 3 of Text Book 1)

- Problem-Solving Agents
- Example Problems
- **Search Algorithms**
- Best- first search
- Uniformed Search Strategies- Breadth-first search
- Dijkstra‘s algorithm or uniform-cost search
- Informed (Heuristic) Search Strategies- Greedy best-first search
- A* search.

RAMAIAH
Institute of Technology

Search Algorithms

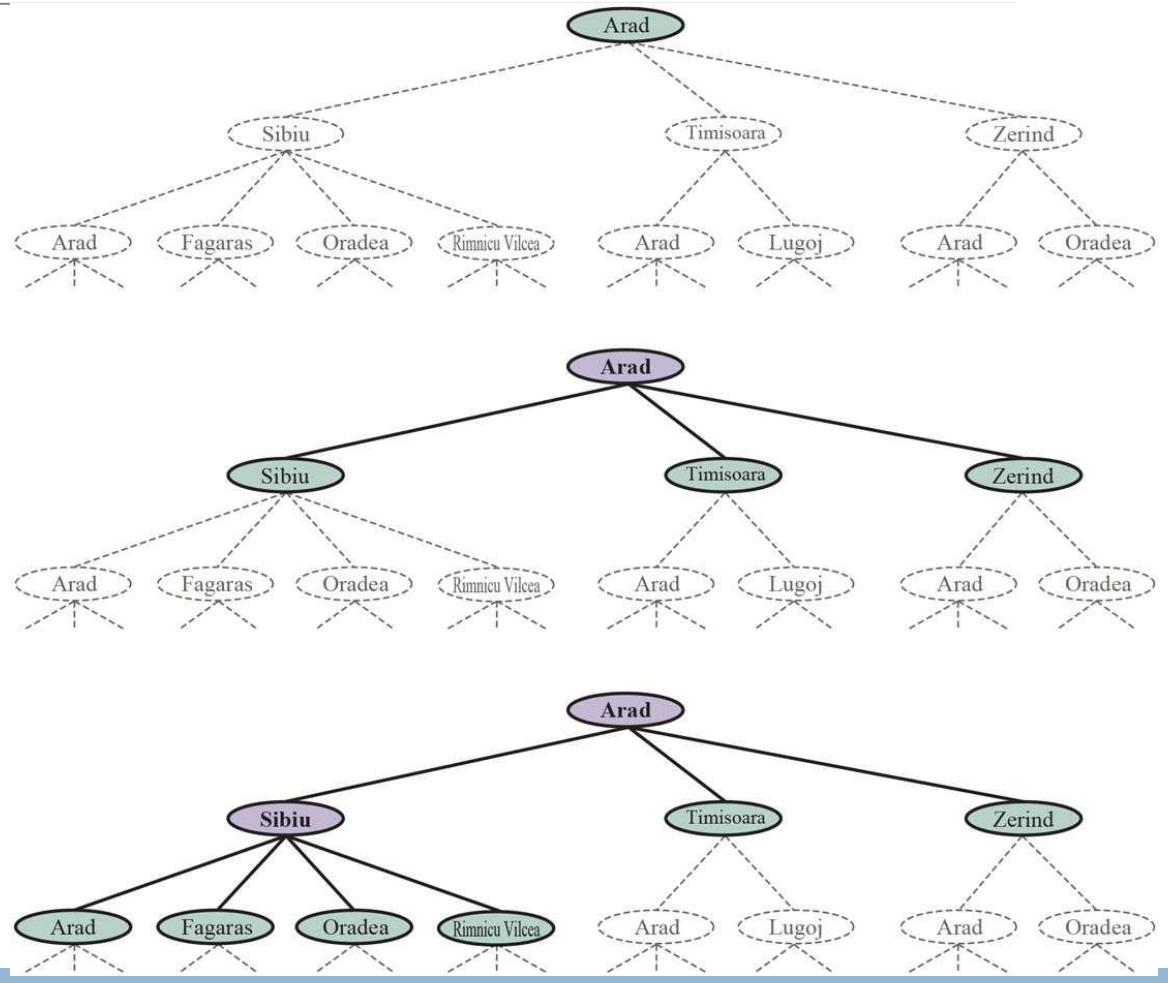
- A search algorithm
  - takes a search problem as input
  - returns a solution, or an indication of failure.
- In a search tree
  - Node => State in the state space
  - Edges =>actions.
  - root => initial state of the problem.
- The **state space** describes the (possibly infinite) set of states in the world, and the actions that allow transitions from one state to another.
- The search tree may have **multiple paths** to (and thus multiple nodes for) any given state, but each node in the tree has a unique path back to the root (as in all trees).



RAMAIAH
Institute of Technology

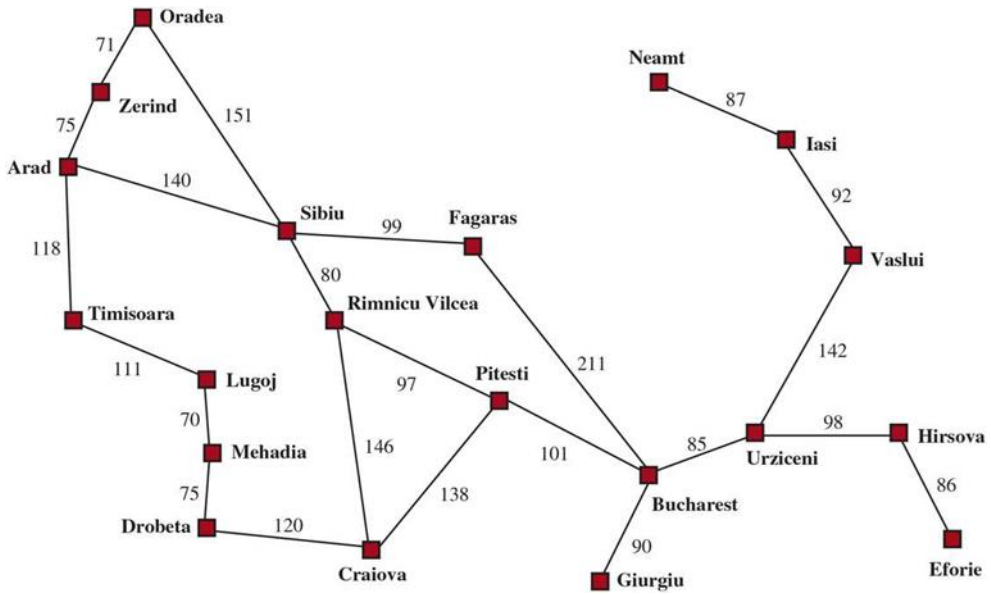
Search Algorithms

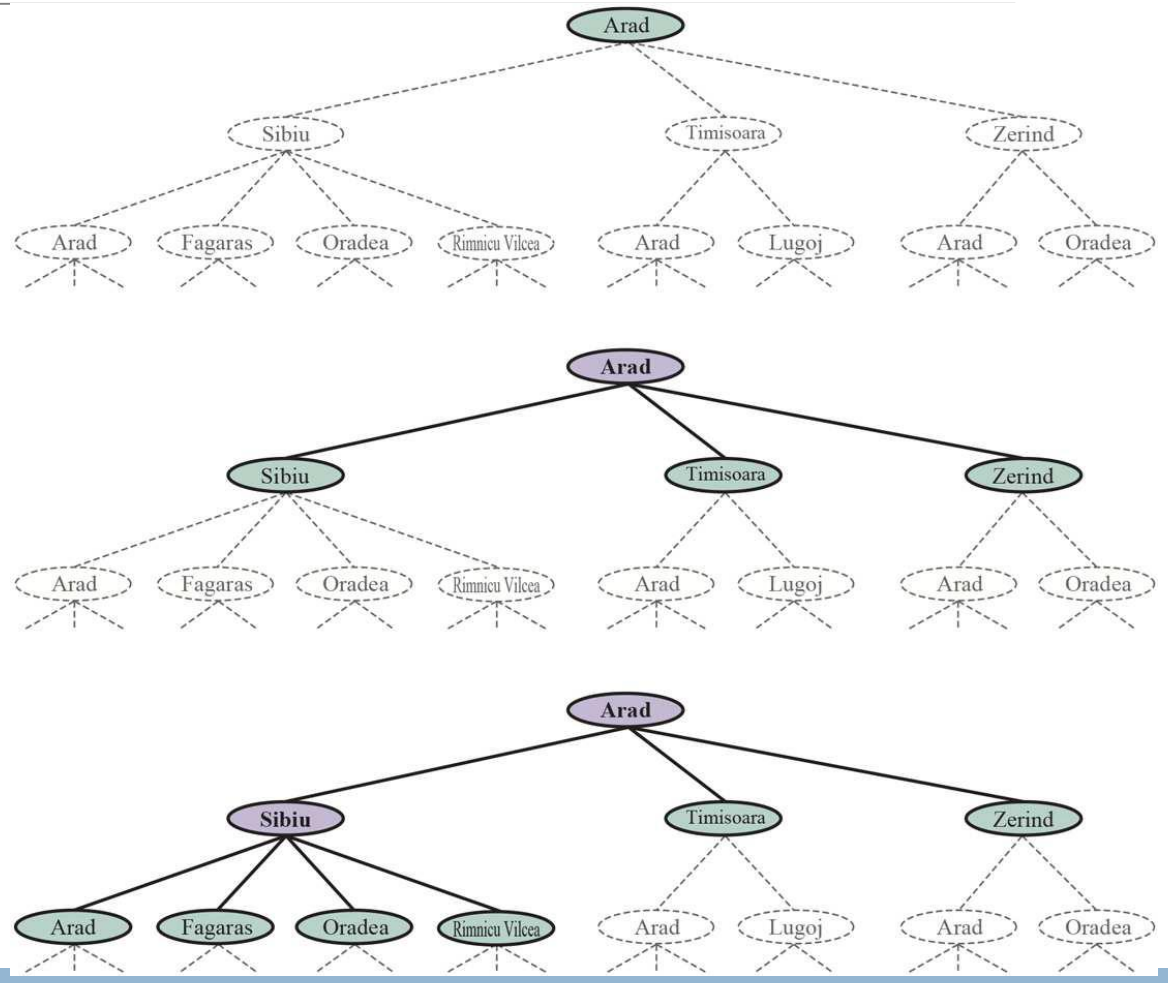
- Steps in finding a path from Arad to Bucharest.
- **Initial State:**The root node of the search tree is at the initial state, Arad.
- **Expand**: the node, considering the available **ACTIONS** for that state.
- **RESULT** function to see where those actions lead to, and generating a new node (called a child node or successor node) for each of the resulting states.
- Each child node has Arad as its parent node.



RAMAIAH
Institute of Technology

Search Algorithms

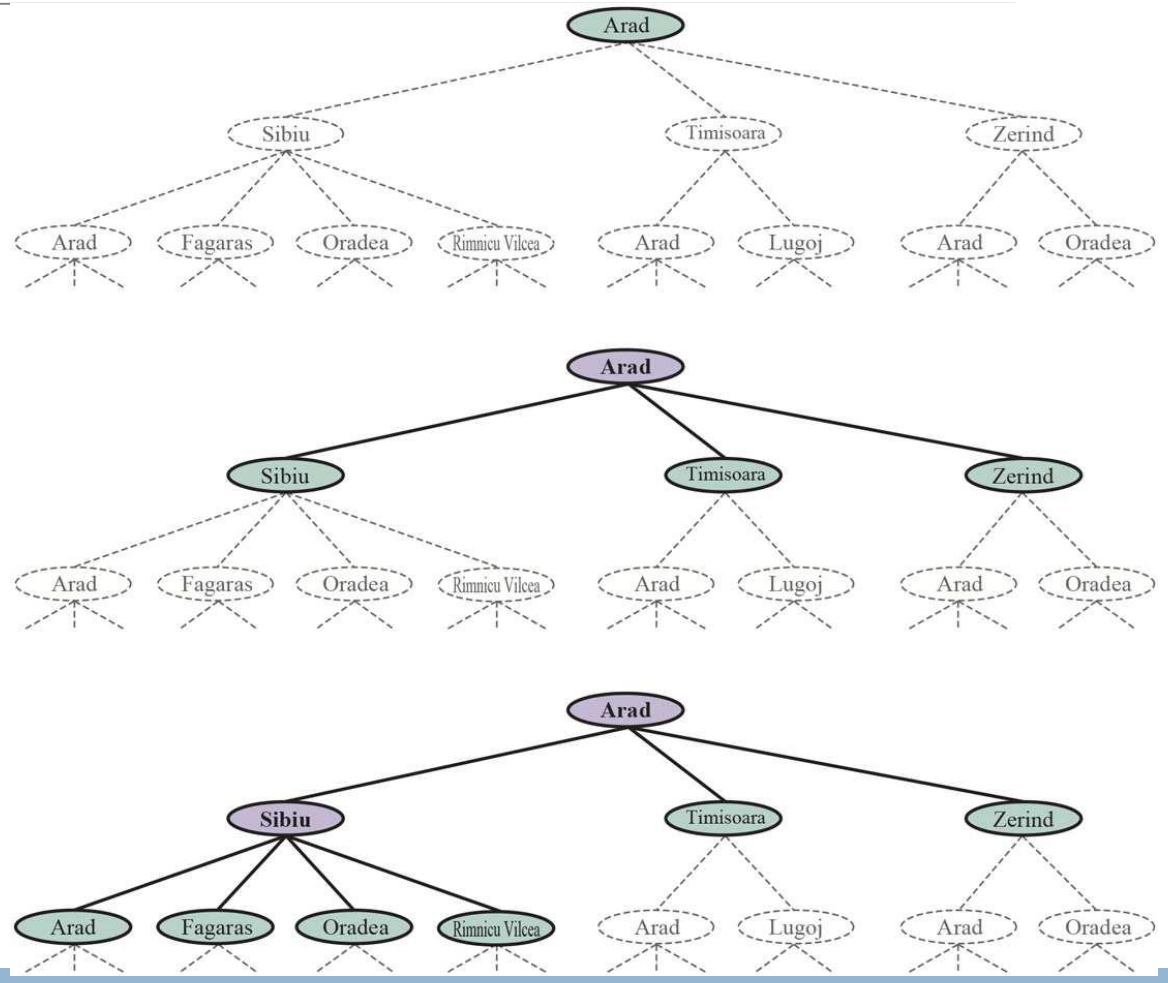




RAMAIAH
Institute of Technology

Search Algorithms

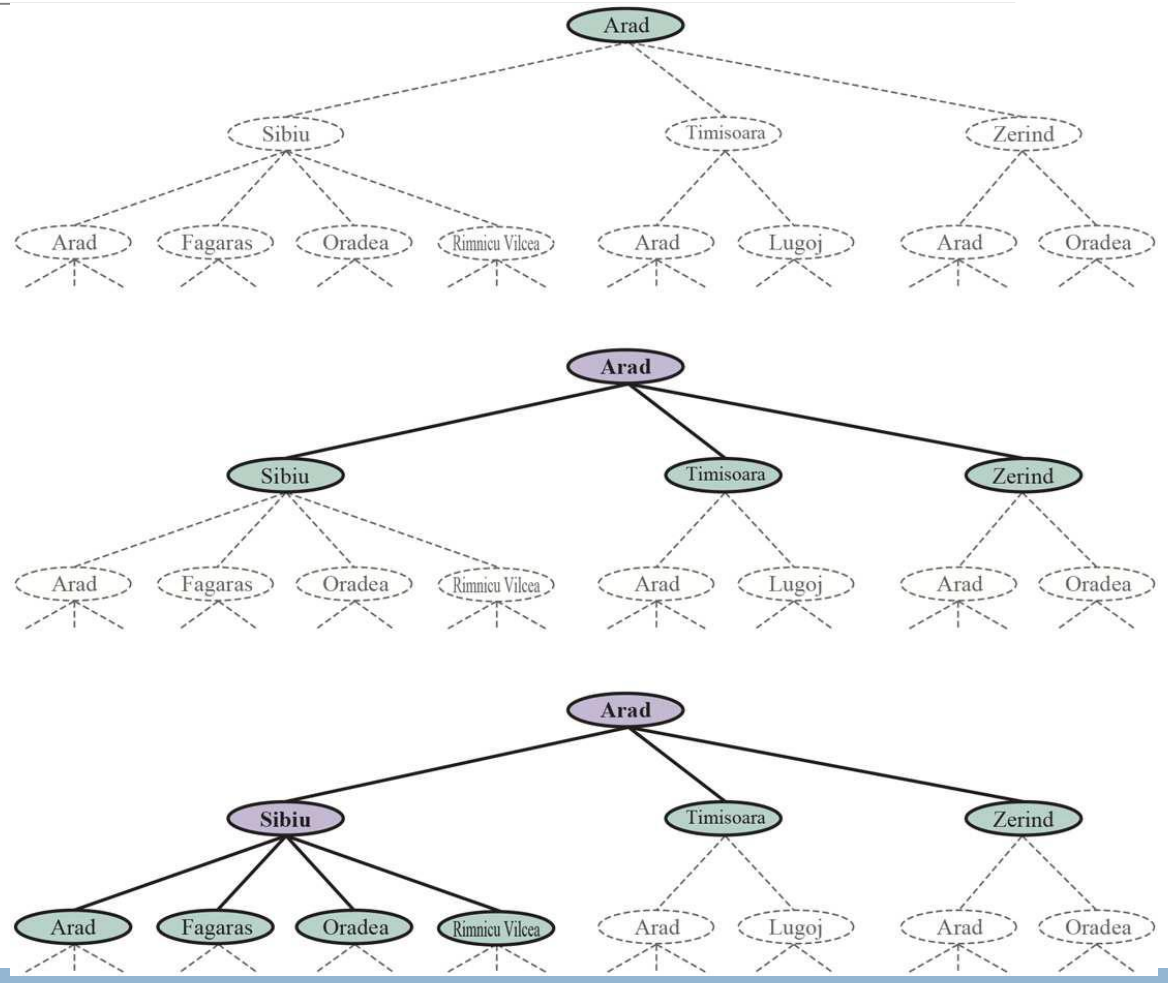
- **Three partial search** trees for finding a route from Arad to Bucharest.
- **Expanded nodes :** Lavender with bold letters; **Frontier nodes:** Generated not yet expanded are in green; the set of states corresponding to these two types of nodes are said to have been **reached**.
- Nodes that could be **generated next** are shown in faint dashed lines.
- **Note**: Bottom tree there is a **cycle** from Arad to Sibiu to Arad; that can't be an optimal path, so search should not continue from there.



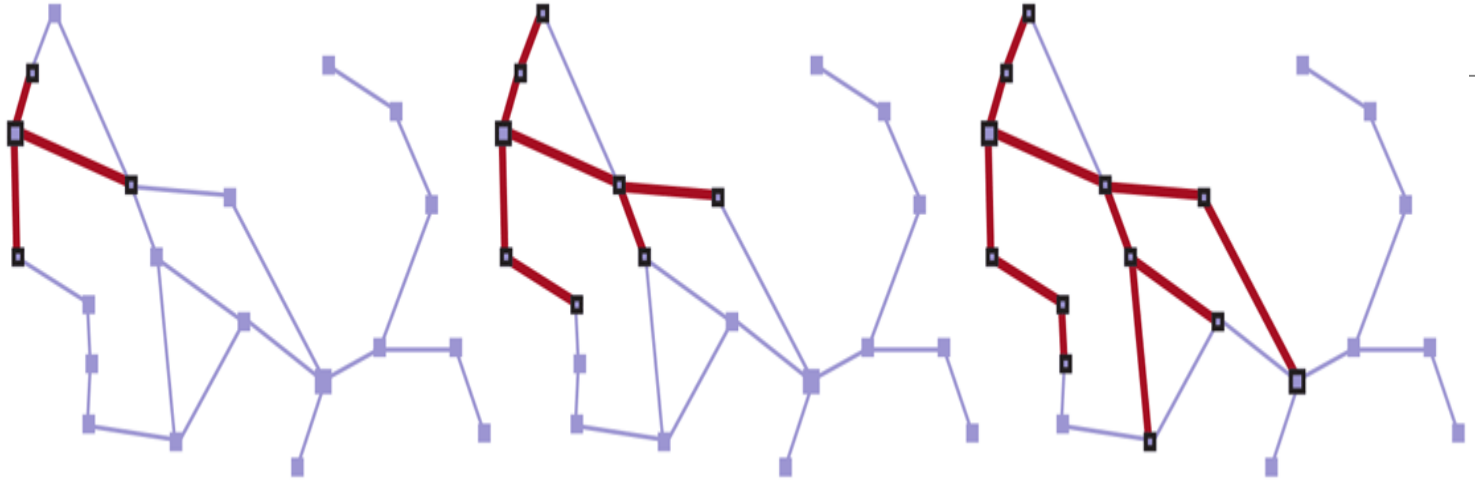
RAMAIAH
Institute of Technology

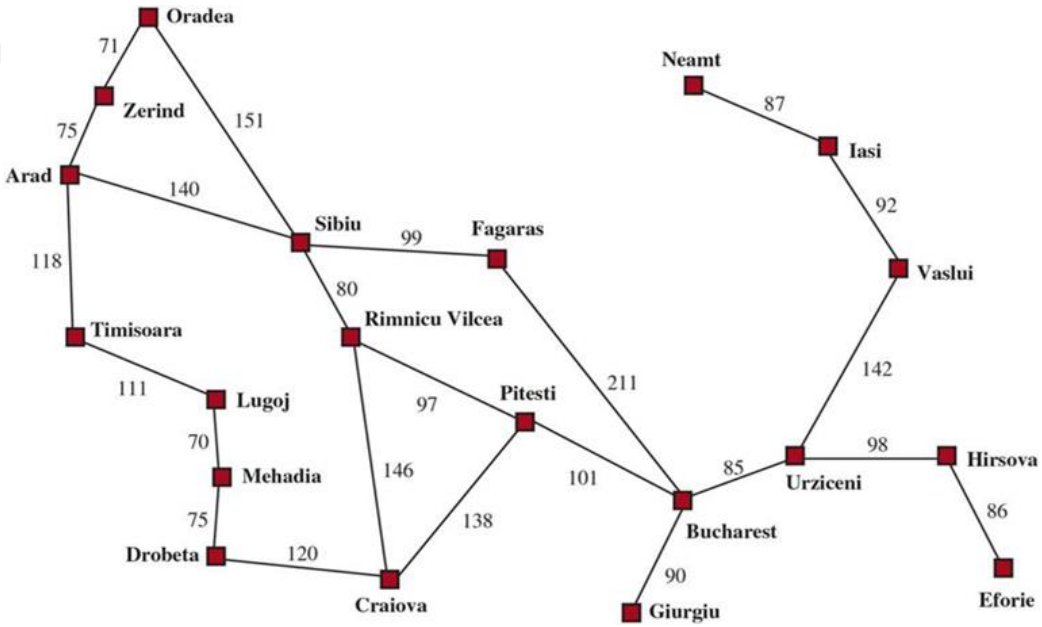
Search Algorithms

- Choose one of these three child nodes to consider next.
- Following up one option now and putting the others aside for later.
- Ex: **Choose to expand Sibiu first**.
- set of 6 unexpanded nodes (outlined in bold). These are called frontier of the search tree. expanded).



- A sequence of search trees generated by a graph search on the Romania problem

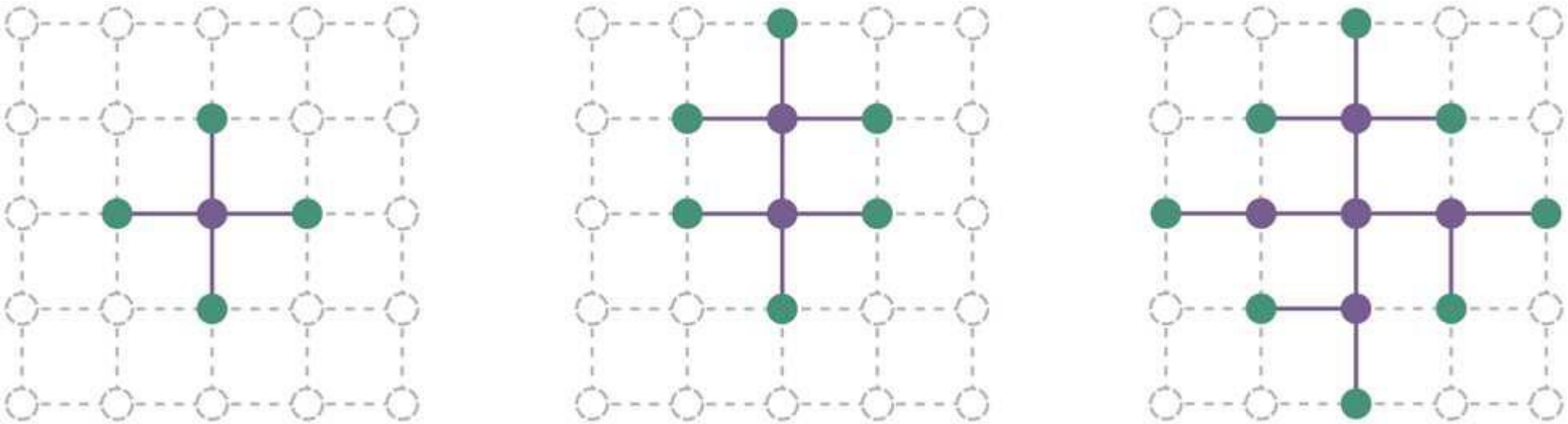




RAMAIAH
Institute of Technology

Search Algorithms

- **Frontier** separates two regions of the state-space graph:
  - an **interior region** where every state has been expanded,
  - an **exterior region** of states that have not yet been reached.



(a) (b) (c)

Best First Search Algorithms

It uses Heuristic/informed search process

Priority queue is used to store cost of nodes

Algorithm:

Priority Queue 'PQ' containing initial states

Loop

if PQ =empty Return FAIL

else

Node<- Remove_First(PQ)

if Node=Goal

Return path from initial state to Node

Else

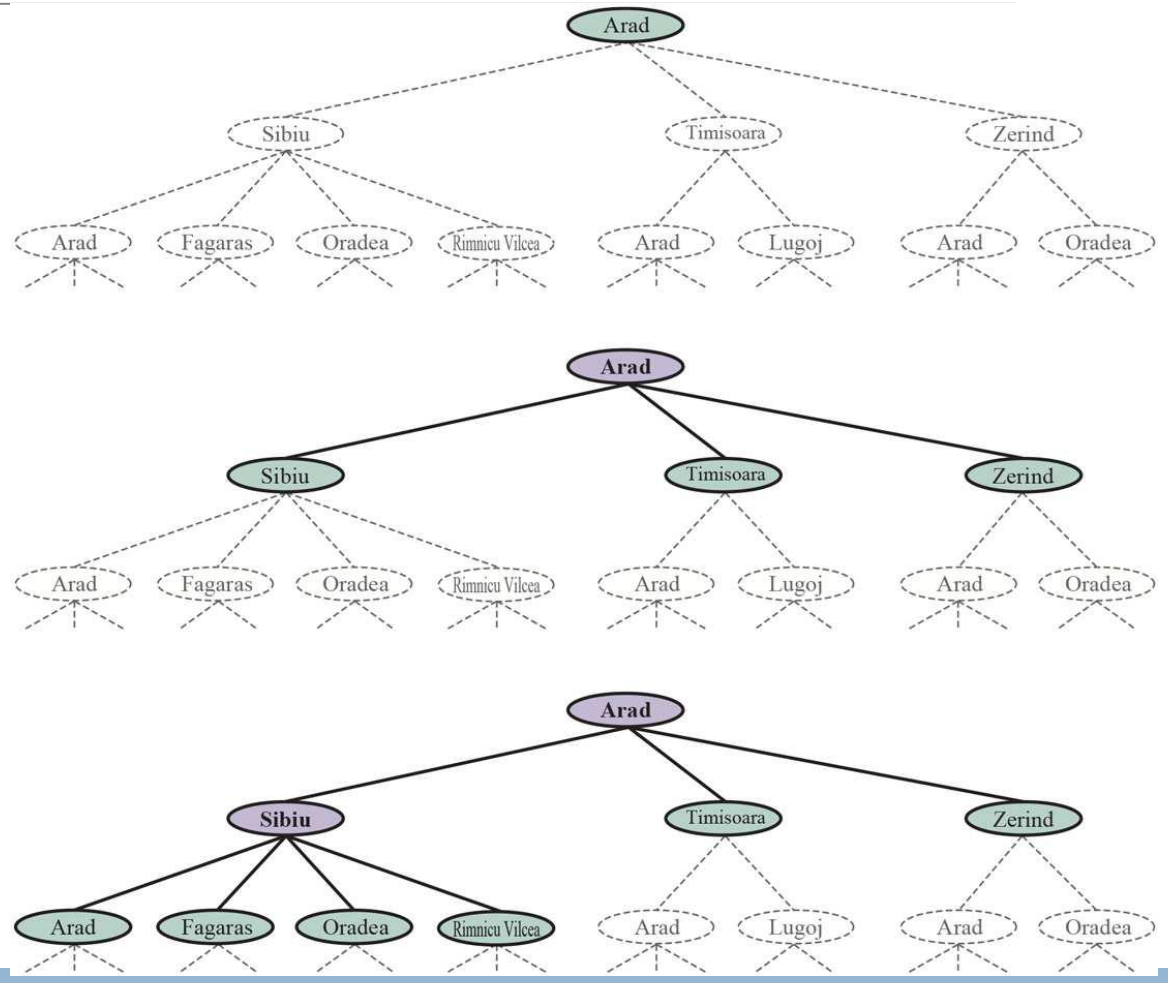
Generate all successors of Node & insert newly generated Node into PQ according to cost values

End Loop

RAMAIAH
Institute of Technology

Best-first search

- Best-first search is used to decide which **node from the frontier to expand next?**
- A node, with minimum value of some **f(n) evaluation function is used**
- On each iteration
  - Choose a node on the frontier with minimum value f(n), return it if its state is a goal state, and otherwise apply EXPAND to generate child nodes.
  - Each child node is added to the frontier if it has not been reached before, or is re-added if it is now being reached with a path that has a lower path cost than any previous path.
  - The algorithm returns either an indication of failure, or a node that represents a path to a goal.



Best-first search

Three kinds of queues are used in search algorithms:

1. A priority queue first pops the node with the minimum cost according to some evaluation function f, It is used in best-first search.
2. A FIFO queue or first-in-first-out queue first pops the node that was added to the queue first; we shall see it is used in breadth-first search.
3. A LIFO queue or last-in-first-out queue (also known as a stack) pops first the most recently added node; we shall see it is used in depth-first search

The reached states can be stored as a lookup table (e.g. a hash table) where each key is a state and each value is the node for that state..

RAMAIAH
Institute of Technology

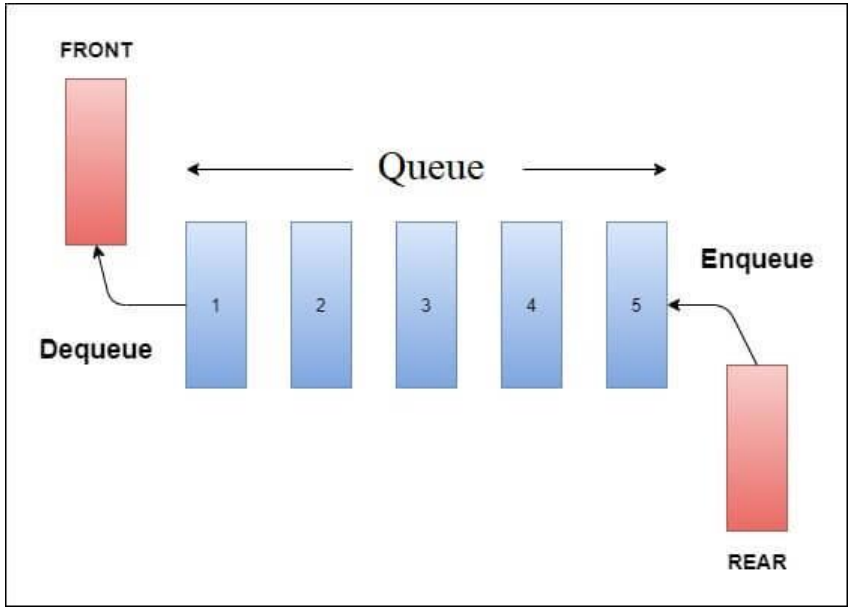
Data Structure : Queue(Additional Information)

We need to put Nodes somewhere.

The frontier needs to be stored in such a way that the search algorithm can easily choose the next node to expand according to its preferred strategy.

The appropriate data structure for this is a queue.

The operations on a queue are as follows:

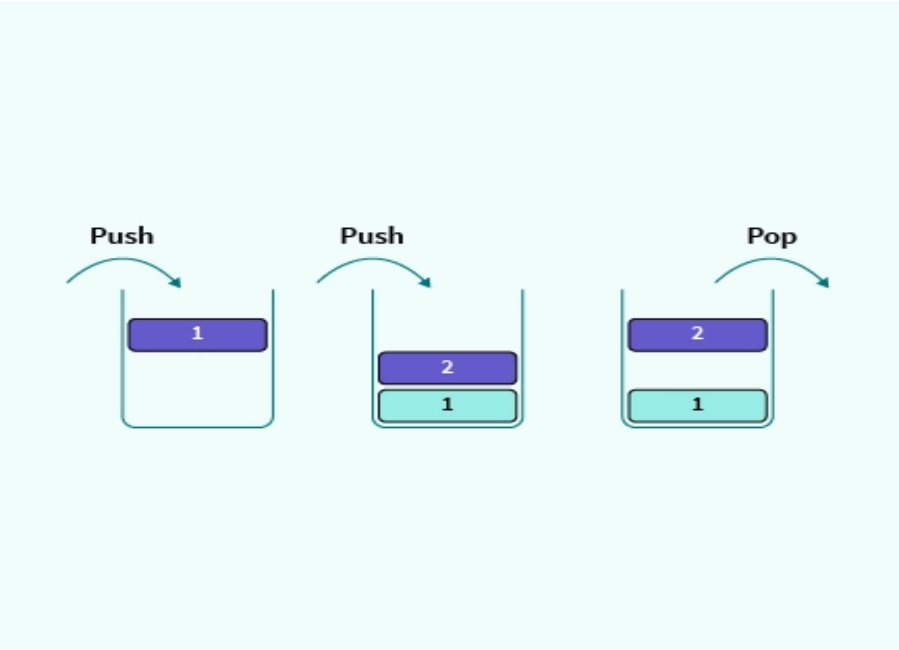


- EMPTY?(*queue*) returns true only if there are no more elements in the queue.
- POP(*queue*) removes the first element of the queue and returns it.
- INSERT(*element*, *queue*) inserts an element and returns the resulting queue.

RAMAIAH
Institute of Technology

Data Structure : Stack Queue(Additional Information)

- Last-in First-out(LIFO) : pops the newest ele
  queue

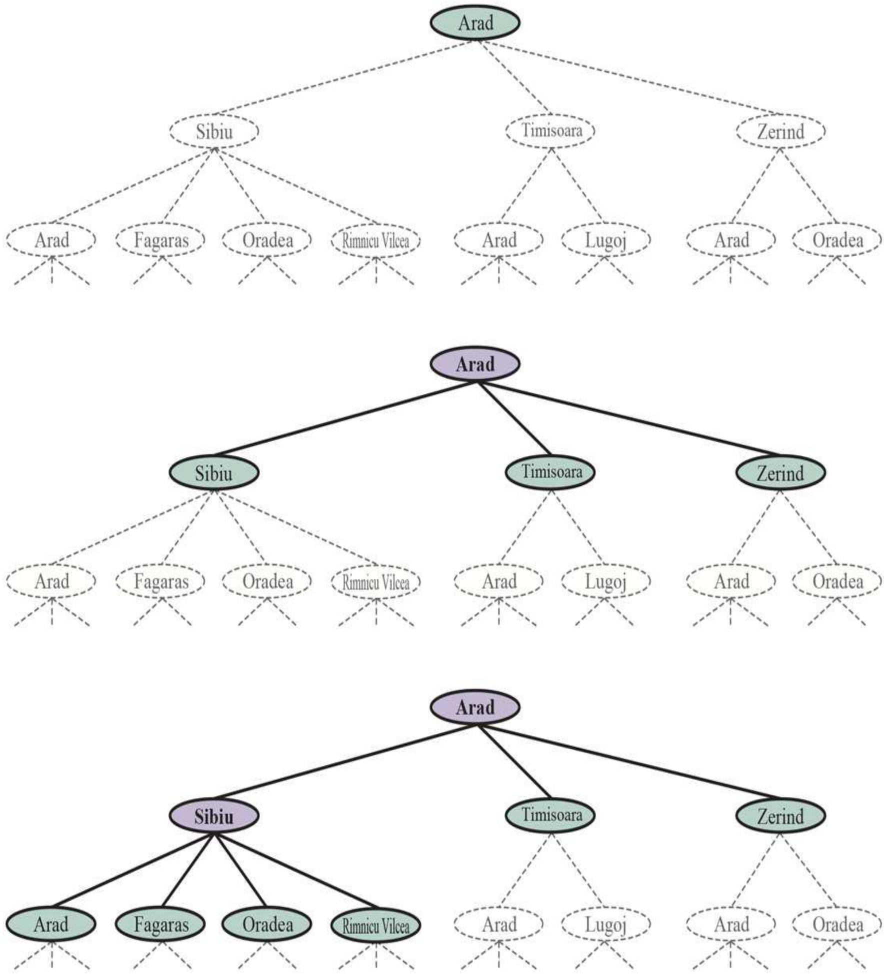


RAMAIAH
Institute of Technology

Data Structure : Priority Queue

Operation	Priority Queue	Return Value
Push(1)	1	
Push(4)	1 4	
Push(2)	1 4 2	
Pop()	1 2	4
Push(3)	1 2 3	
Pop()	1 4 2	3

```
function BEST-FIRST-SEARCH(problem, f) returns a solution node or failure
  node ← NODE(STATE=problem.INITIAL)
  frontier ← a priority queue ordered by f, with node as an element
  reached ← a lookup table, with one entry with key problem.INITIAL and value node
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if s is not in reached or child.PATH-COST < reached[s].PATH-COST then
        reached[s] ← child
        add child to frontier
  return failure
```




```
function EXPAND(problem, node) yields nodes
  s ← node.STATE
  for each action in problem.ACTIONS(s) do
    s′ ← problem.RESULT(s, action)
    cost ← node.PATH-COST + problem.ACTION-COST(s, action, s′)
    yield NODE(STATE=s′, PARENT=node, ACTION=action, PATH-COST=cost)
```

RAMAIAH
Institute of Technology

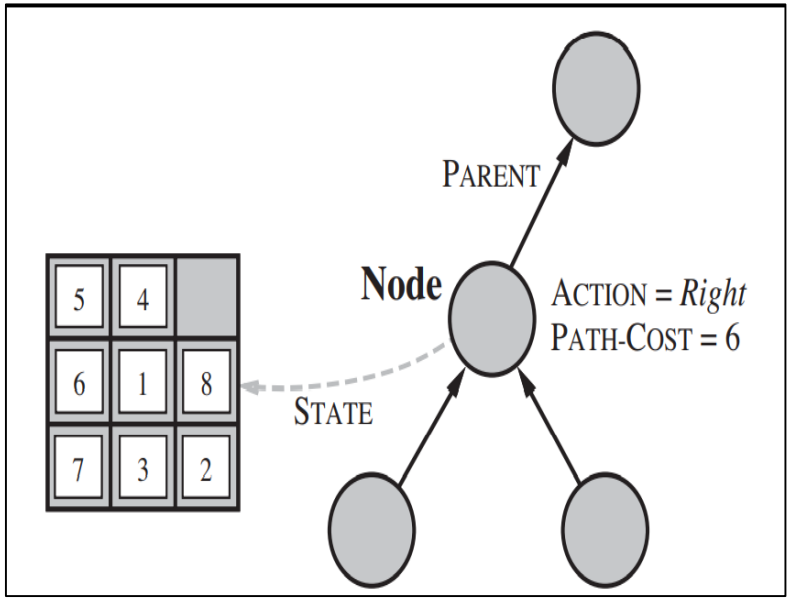
Best-first search

Search data structures:

Search algorithms require a data structure to keep track of the search tree.

A node in the tree is represented by a data structure with four components:

1. node.STATE: The state to which the node corresponds;
2. node.PARENT: The node in the tree that generated this node;
3. node.ACTION: The action that was applied to the parent's state to generate this node;
4. node.PATH-COST: The total cost of the path from the initial state to this node.



Best-first search

Data structure to store the frontier.

The appropriate choice is a **queue**, because the **operations** on a frontier are:

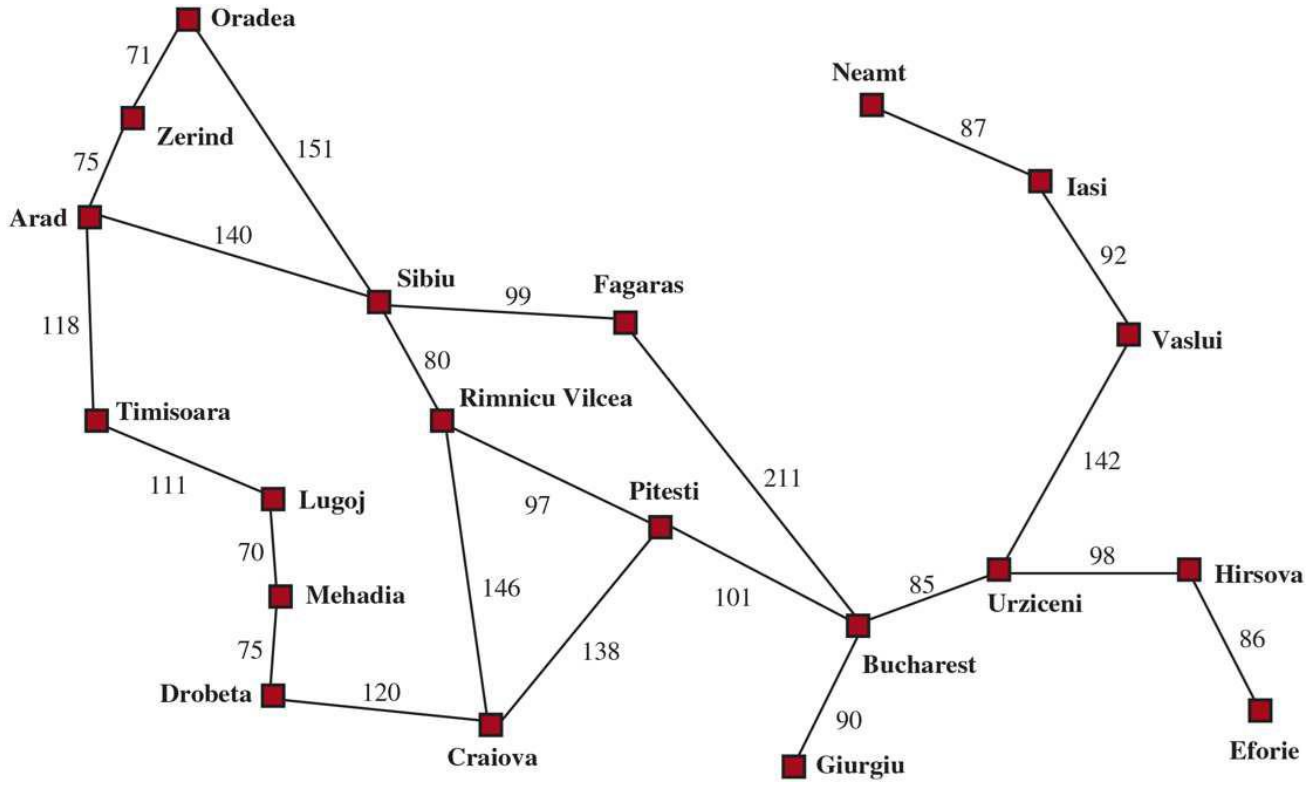
1. IS-EMPTY(frontier) returns true only if there are no nodes in the frontier.
2. POP(frontier) removes the top node from the frontier and returns it.
3. TOP(frontier) returns (but does not remove) the top node of the frontier.
4. ADD(node, frontier) inserts node into its proper place in the queue.

RAMAIAH
Institute of Technology

Best-first search

Redundant paths:

- The search tree includes a path from Arad to Sibiu and back to Arad again.
- Arad is a repeated state in the search tree, generated in this case by a cycle (also known as a loopy path).
- State space has only 20 states, the complete search tree is infinite because there is no limit to how often one can traverse a loop.



Measuring problem-solving performance

In AI, complexity is expressed with 3 quantities

- **b**, branching factor, maximum number of successors of any node
- **d**, the depth of the shallowest goal node.
- **m**, the maximum length of any path in the state space

Time and Space is measured in

- Time - number of nodes generated during the search
- Space -maximum number of nodes stored in memory

Measuring problem-solving performance

- **There are 4 criteria's used to choose among search algorithms.**
- **Algorithm's performance can be evaluated in four ways:**
- The evaluation of a search strategy in 4 ways:
  - **Completeness:**
    - is the strategy guaranteed to find a solution when there is one?
  - **Optimality**:
    - does the strategy find the highest-quality solution when there are several different solutions?
  - **Time complexity**:
    - how long does it take to find a solution?
  - **Space complexity**:
    - how much memory is needed to perform the search?

Measuring problem-solving performance

- Graph is an explicit data structure that is input to the search program.
- The typical measure is the size of the state space graph, |V | + |E|
- where V is the set of vertices (nodes) of the graph
- E is the set of edges (links).
- In AI, the graph is often represented implicitly by the initial state, actions, and transition model

RAMAIAH
Institute of Technology

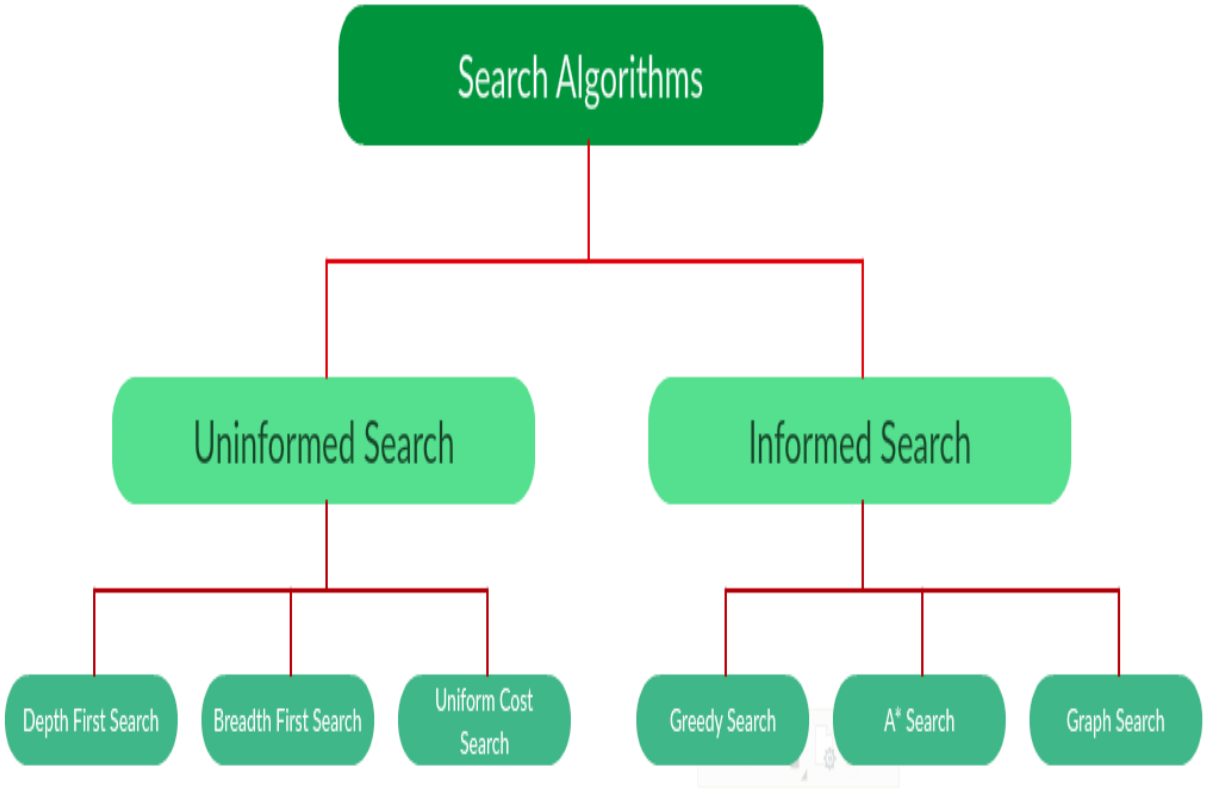
Search Strategies

Uninformed search (Blind search)

- No information about the number of steps or the path cost from the current state to the goal
- Search the state space **blindly**

Informed search, or heuristic search

- A cleverer strategy that searches toward the goal, based on the information from the current state so far



RAMAIAH
Institute of Technology

Uninformed Search Strategies

1. Breadth-first search
2. Uniform-cost search
3. Depth-first search

RAMAIAH
Institute of Technology

Breadth-first search

- Breadth-first search is the most common search strategy for traversing a tree or graph. This algorithm searches breadthwise in a tree or graph, so it is called breadth-first search.
- BFS algorithm starts searching from the root node of the tree and expands all successor node at the current level before moving to nodes of next level.
- Breadth-first search implemented using FIFO queue data structure.

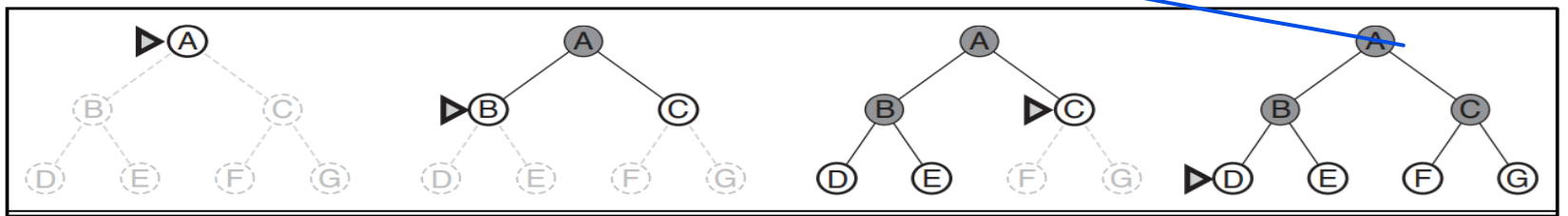
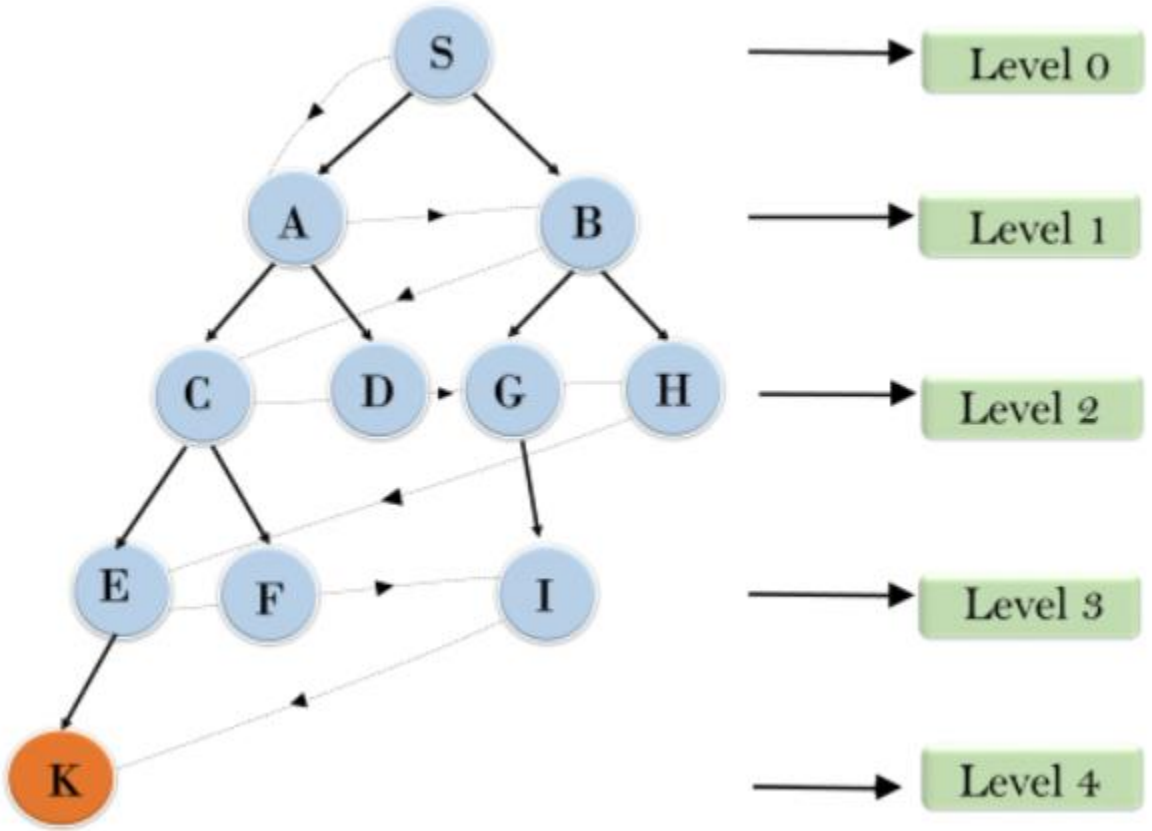


Figure 3.12 Breadth-first search on a simple binary tree. At each stage, the node to be expanded next is indicated by a marker.

RAMAIAH
Institute of Technology

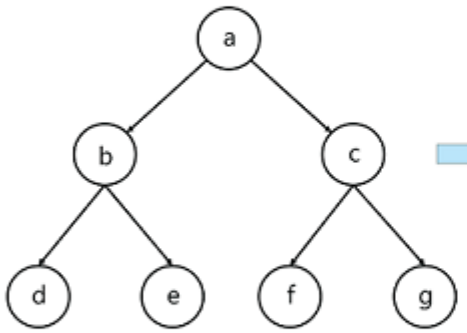
Example:Breadth-first search

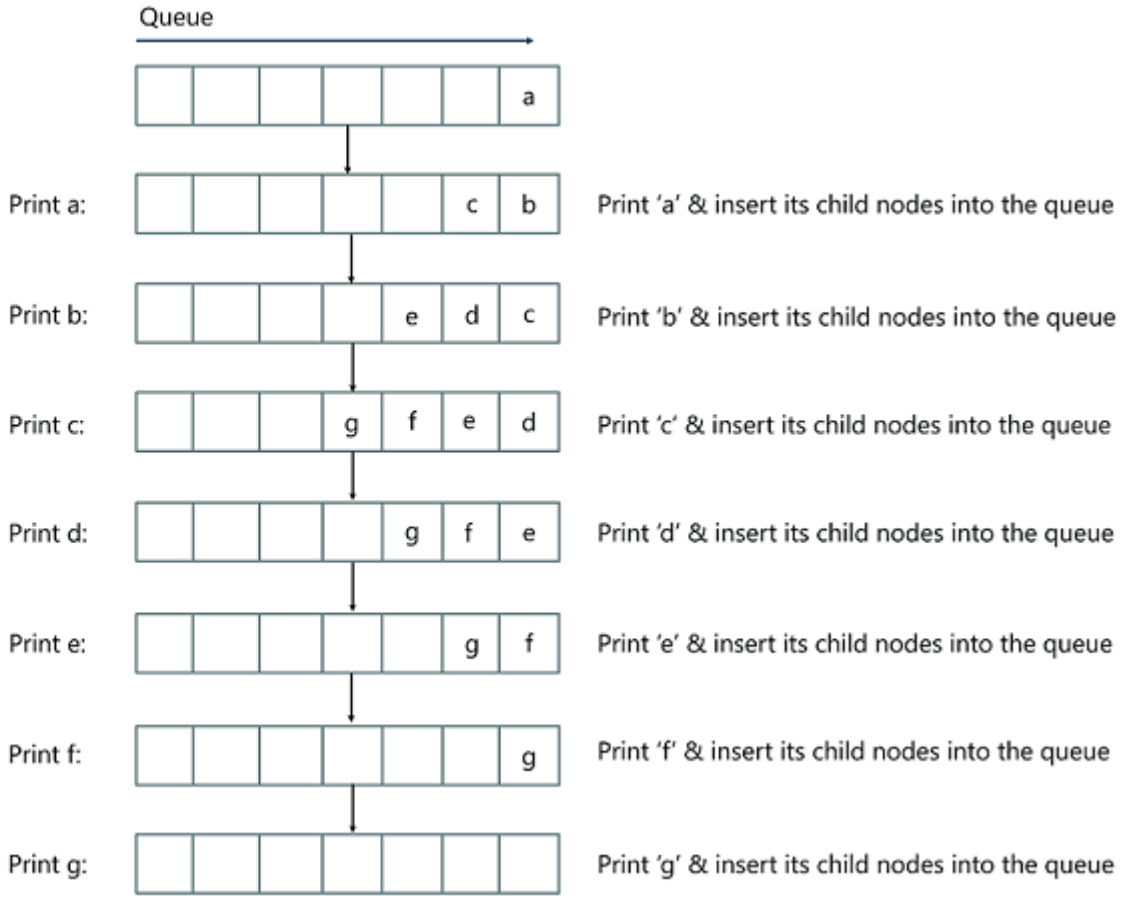


S---> A--->B---->C--->D---->G--->H--->E---->F---->I---->K

RAMAIAH
Institute of Technology

Example:Breadth-first search





Breadth-first search

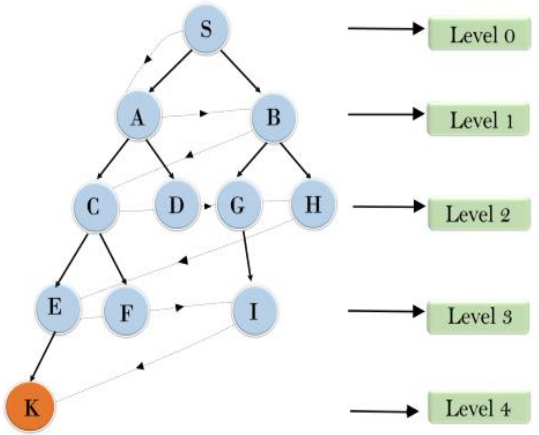
```
function BREADTH-FIRST-SEARCH(problem) returns a solution node or failure
  node ← NODE(problem.INITIAL)
  if problem.IS-GOAL(node.STATE) then return node
  frontier ← a FIFO queue, with node as an element
  reached ← {problem.INITIAL}
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if problem.IS-GOAL(s) then return child
      if s is not in reached then
        add s to reached
        add child to frontier
  return failure
```

RAMAIAH Institute of Technology

Properties of breadth-first search

- **Completeness**: BFS is complete, which means if the shallowest goal noc is at some finite depth, then BFS will find a solution.
- **Optimality** : BFS is optimal if path cost is a non-decreasing function of tl depth of the node.
- **Time and Space complexity**: Searching a uniform tree where every state has b successors. The root of the search tree generates b nodes, each of which generates b more nodes, for a total of b^2 at the second level. Each of these generates b more nodes, yielding b^3nodes at the third level, and so on. Now suppose that the solution is at depth d. Then the total number of nodes generated
- **Space complexity**: Memory size to store every level of the tree is $O(b^d)$

$$b + b^2 + b^3 + \cdots + b^d = O(b^d) .$$



The memory requirements are a bigger problem for breadth-first search than the execution time.

RAMAIAH
Institute of Technology

Breadth-first search

Advantages:

- BFS will provide a solution if any solution exists.
- If there are more than one solutions for a given problem, then BFS will provide the minimal solution which requires the least number of steps.

Disadvantage

- It requires lots of memory since each level of the tree must be saved into memory to expand the next level.
- BFS needs lots of time if the solution is far away from the root node.

RAMAIAH
Institute of Technology

Dijkstra's algorithm or Uniform cost search

- When actions have different costs, a choice is to use best-first search where the evaluation function is the cost of the path from the root to the current node.
- This is called Dijkstra's algorithm by the theoretical **computer science community**, and uniform-cost search by the **AI community.**
- Breadth-first search spreads out in waves of uniform depth—first depth 1, then depth 2, and so on—uniform-cost search spreads out in waves of uniform path-cost

Dijkstra's algorithm or Uniform cost search

Breadth-first finds the shallowest goal state

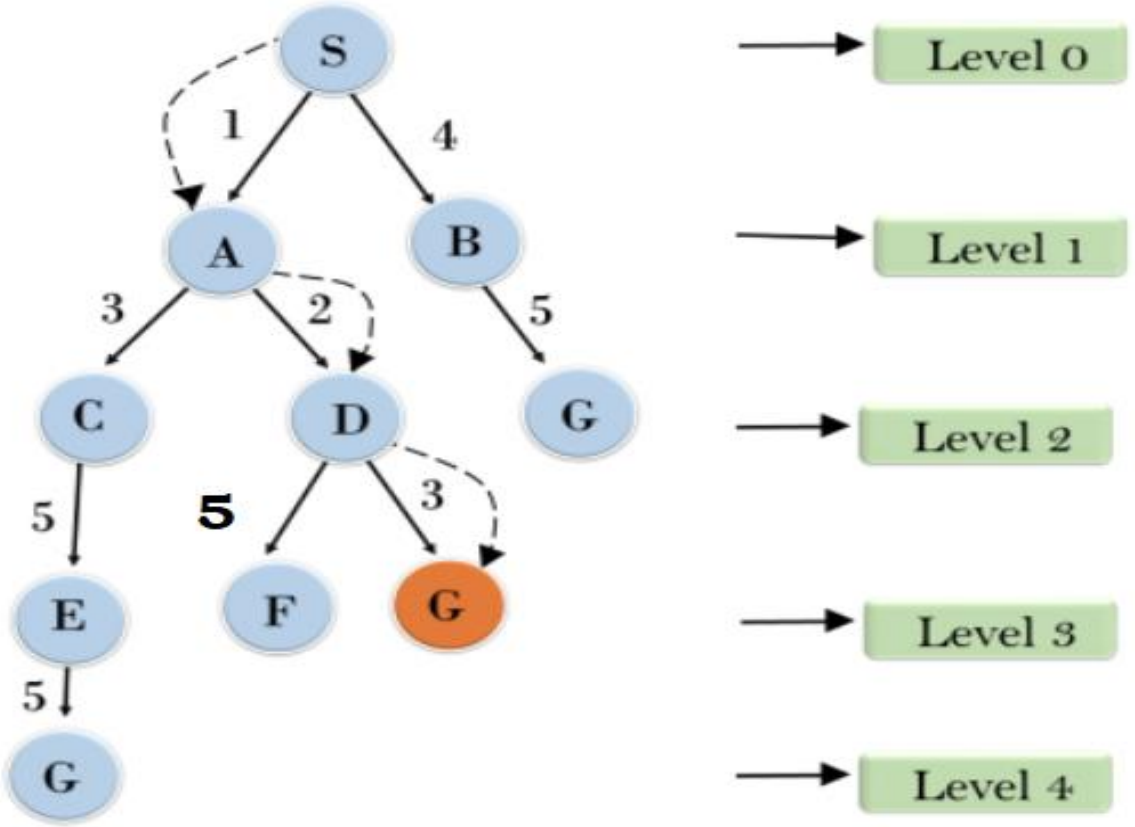
- but not necessarily be the least-cost solution
- work only if all step costs are equal

Uniform cost search

- modifies breadth-first strategy by always expanding the lowest-cost node
- The lowest-cost node is measured by the path cost *g(n)*

RAMAIAH
Institute of Technology

Example: Uniform cost search



RAMAIAH
Institute of Technology

Uniform cost search

```
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
```

RAMAIAH
Institute of Technology

Uniform cost search

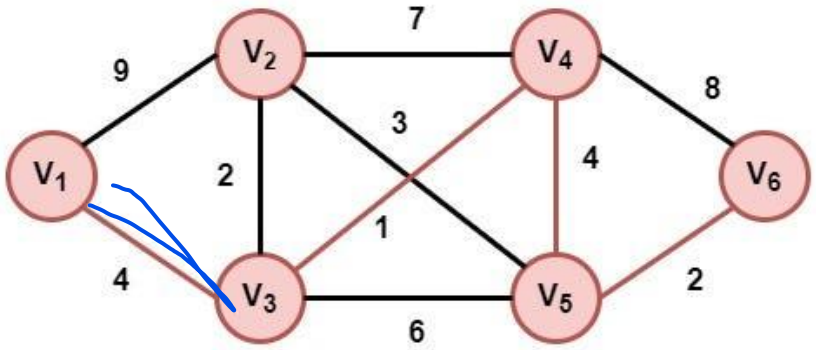
Advantages:

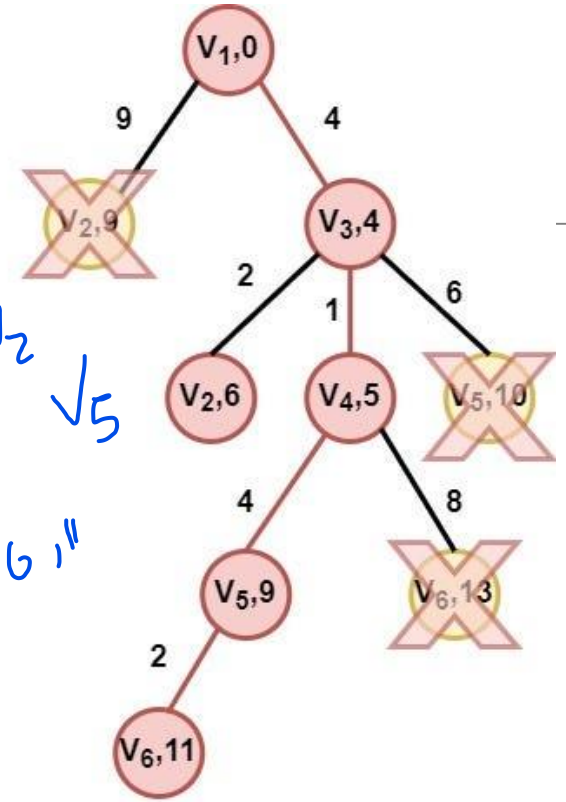
- Uniform cost search is optimal because at every state the path with the least cost is chosen.

Disadvantages:

- It does not care about the number of steps involve in searching and only concerned about path cost. Due to which this algorithm may be stuck in an infinite loop.

RAMAIAH
Institute of Technology





Pace	Opened	Closed
0	{(V_1,0)}	{-}
1	{(V_2,9),(V_3,4)}	{(V_1,0)}
2	{(V_2,6),(V_4,5),(V_5,10)}	{(V_1,0),(V_3,4)}
3	{(V_2,6),(V_6,13),(V_5,9)}	{(V_1,0),(V_3,4),(V_4,5)}
4	{(V_6,13),(V_5,9)}	{(V_1,0),(V_3,4),(V_4,5),(V_2,6)}
5	{(V_6,11)}	{(V_1,0),(V_3,4),(V_4,5),(V_2,6),(V_5,9)}
6	{-}	{(V_1,0),(V_3,4),(V_4,5),(V_2,6),(V_5,9)}

Path: V_1 - V_3 - V_4 - V_5 - V_6

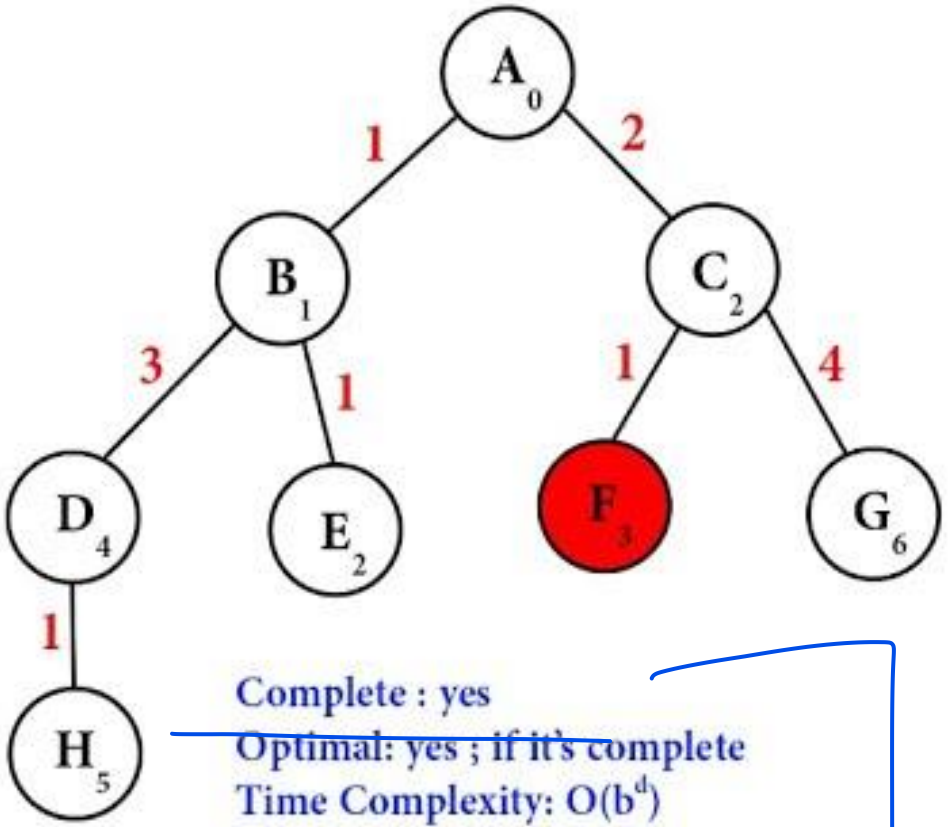
Total Cost: 11

RAMAIAH Institute of Technology

Unifo

Uniform Cost Search (UCS)

—	A_0	
A_0	B_1 C_2	
B_1	C_2 D_4 E_2	C_2 E_2 D_4
C_2	E_2 D_4 F_3 G_6	E_2 F_3 D_4 G_6
E_2	F_3 D_4 G_6	
F_3	Goal	



Complete : yes
Optimal: yes ; if it's complete
Time Complexity: $O(b^d)$
Space Complexity: $O(b^d)$

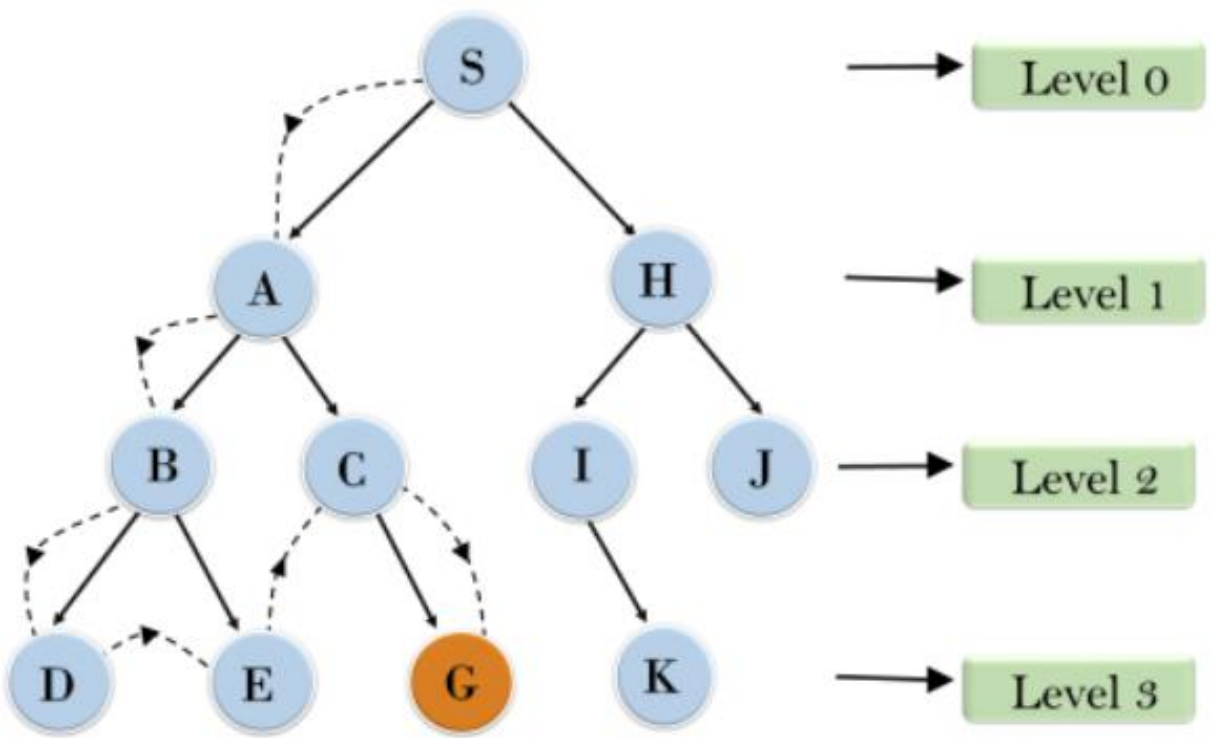
RAMAIAH
Institute of Technology

Depth-first search

- Depth-first search is a recursive algorithm for traversing a tree or graph data structure.
- It is called the depth-first search because it starts from the root node and follows each path to its greatest depth node before moving to the next path.
- DFS uses a STACK.

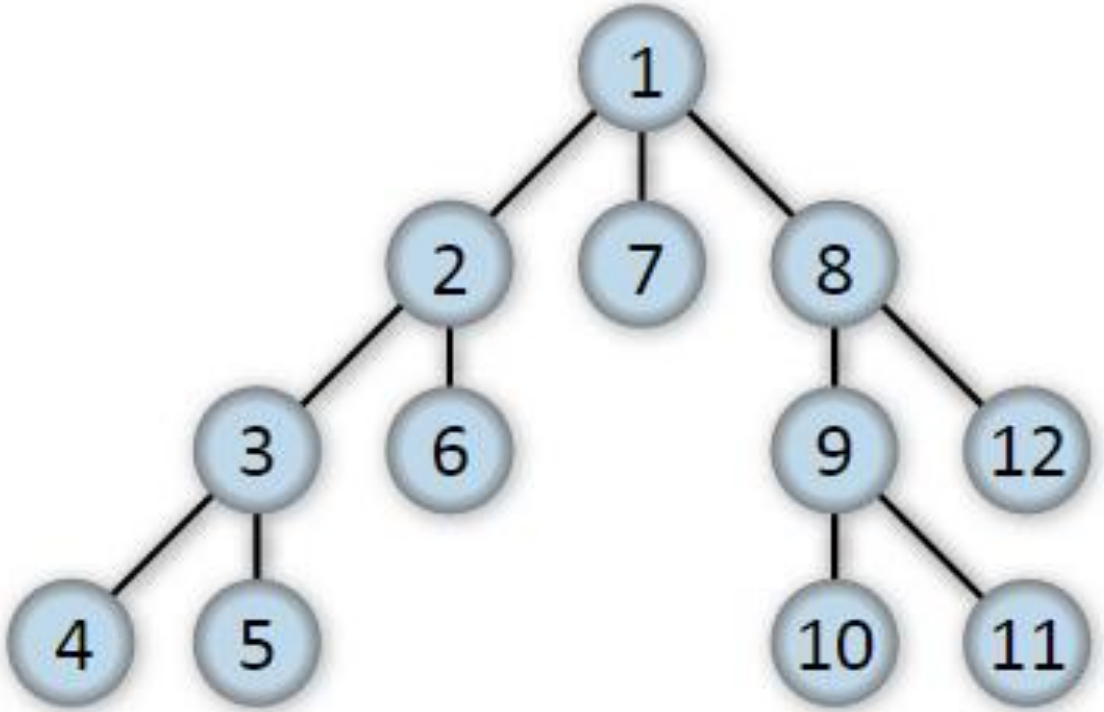
RAMAIAH
Institute of Technology

Example:Depth-first search



RAMAIAH
Institute of Technology

Example:Depth-first search



RAMAIAH
Institute of Technology

Depth-first search

- **Completeness**: Is complete within finite state space as it will expand every node within a limited search tree.
- **Optimal**: Is non-optimal, as it may generate a large number of steps or high cost to reach to the goal node.
- **Time Complexity**: Time complexity of DFS will be equivalent to the node traversed by the algorithm. It is given by: $O(b^m)$

  Where, m= maximum depth of any node and this can be much larger than d
- **Space Complexity**: DFS algorithm needs to store only single path from the root node, hence space complexity of DFS is equivalent to the O(bm).

RAMAIAH
Institute of Technology

Depth-first search

Advantage:

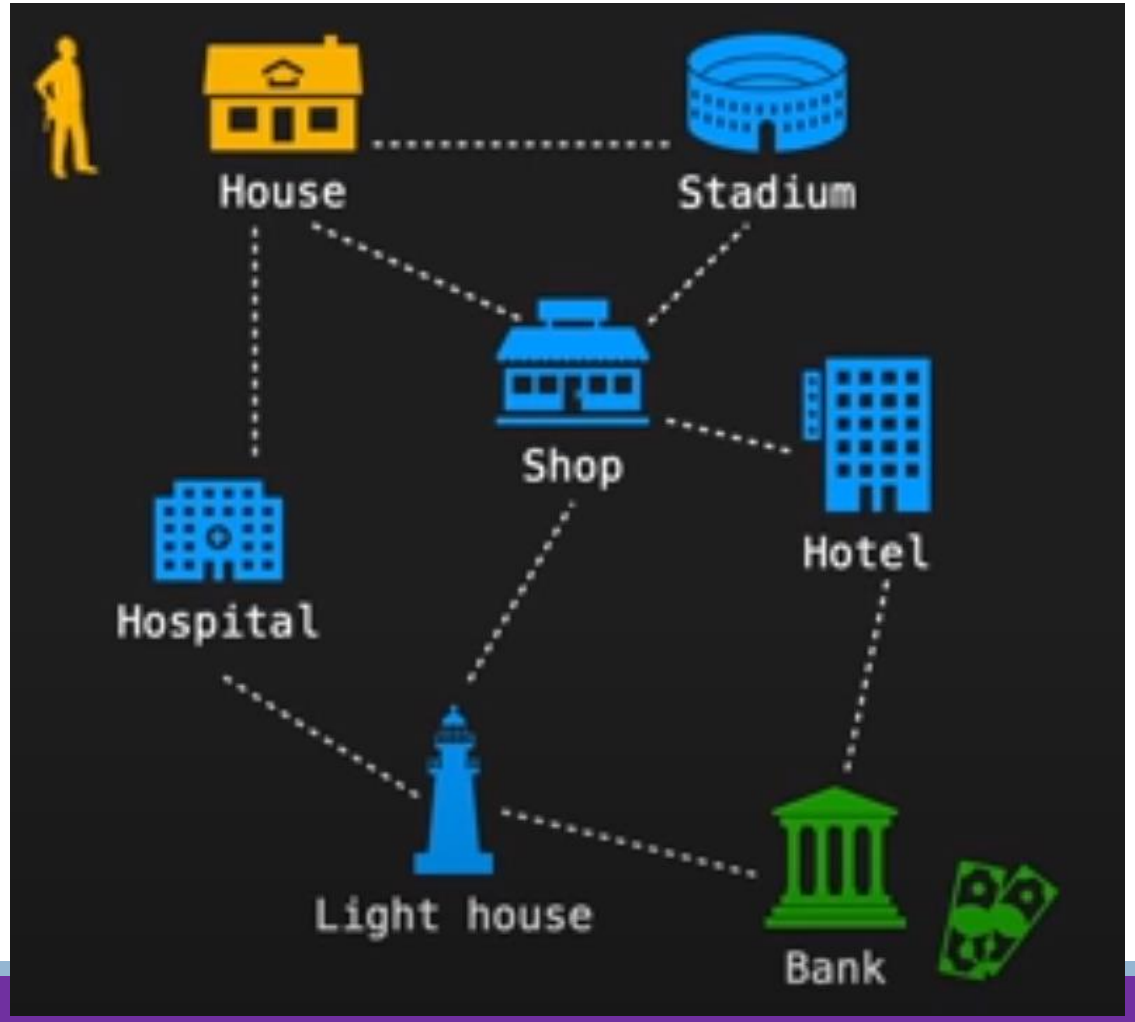
- DFS requires very less memory as it only needs to store a stack of the nodes on the path from root node to the current node.
- It takes less time to reach to the goal node than BFS algorithm (if it traverses in the right path).

Disadvantage:

- There is the possibility that many states keep re-occurring, and there is no guarantee of finding the solution.
- DFS algorithm goes for deep down searching and sometime it may go to the infinite loop

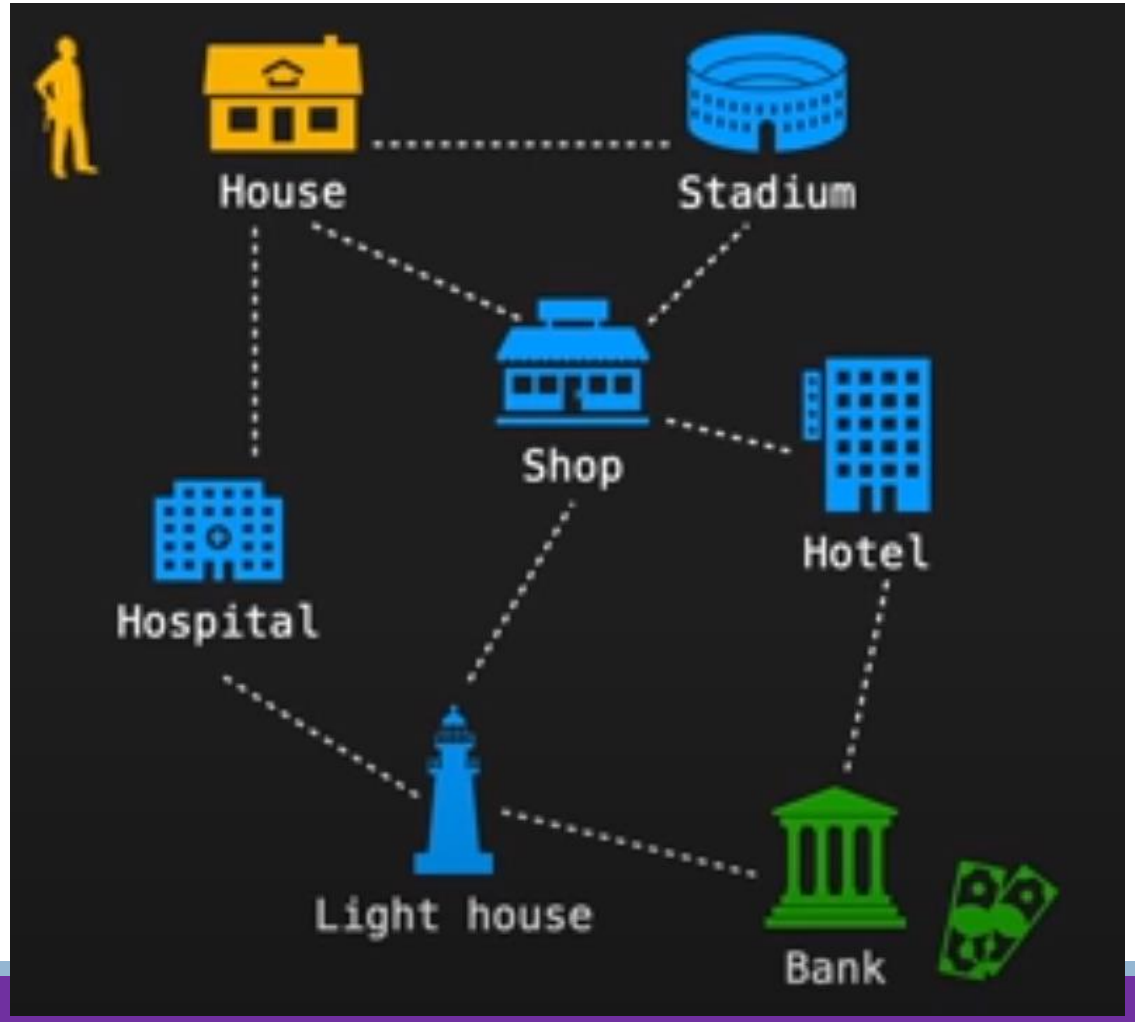
RAMAIAH
Institute of Technology

Uninformed search(Additional)



RAMAIAH
Institute of Technology

Informed search



Location	Estimated Time to reach the Bank
House	25 minutes
Hospital	15 minutes
Light house	5 minutes
Stadium	30 minutes
Shop	20 minutes
Hotel	12 minutes

Informed search strategies

- The informed search algorithm is more useful for **large search space.**
- It uses **domain-specific hints** about the location of goals—can find solutions more efficiently than an uninformed strategy.
- The hints come in the form of a **heuristic function,**
- **h(n) = estimated cost of the cheapest path from the state at node n to a goal state.**
- Ex: Route-finding problems, estimate the distance from the current state to a goal by computing the straight-line distance on the map between the two points.

Greedy best-first search

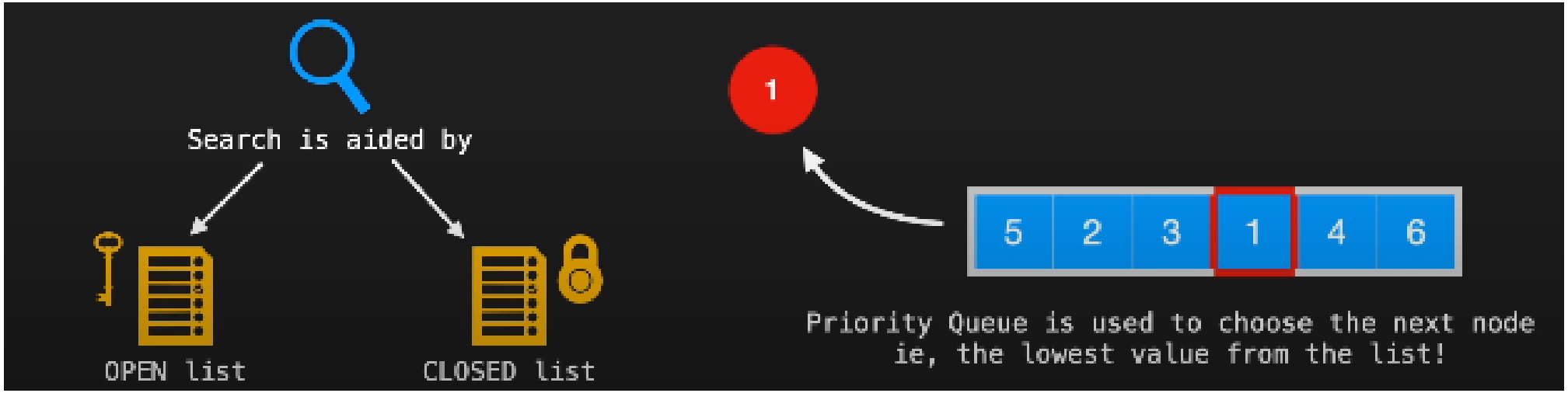
Greedy best-first search is a form of best-first search that expands first the node with the lowest value—the node that appears to be closest to the goal—on the grounds that this is likely to lead to a solution quickly. So the evaluation function

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

$$f(n) = h(n).$$

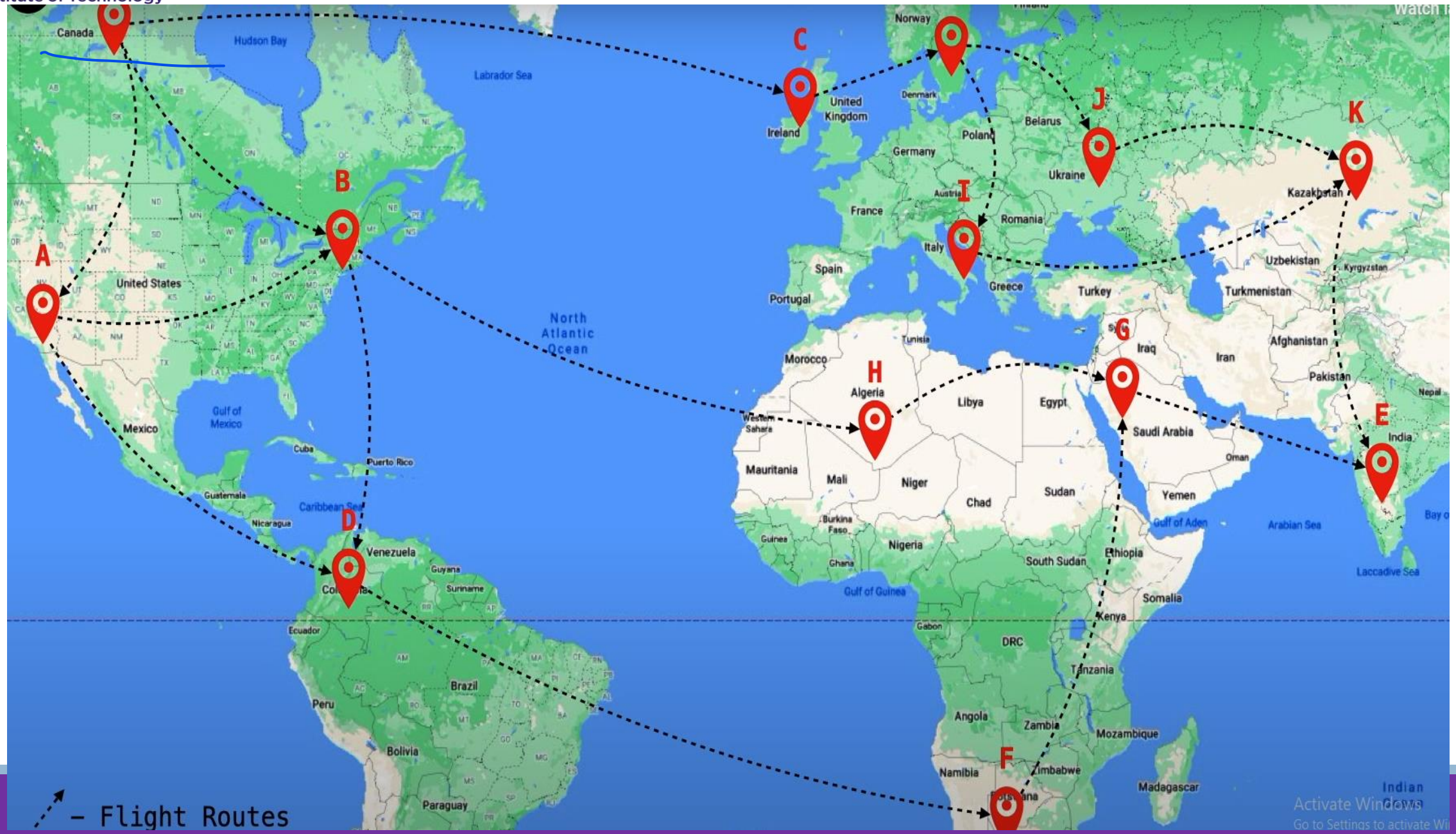
RAMAIAH
Institute of Technology

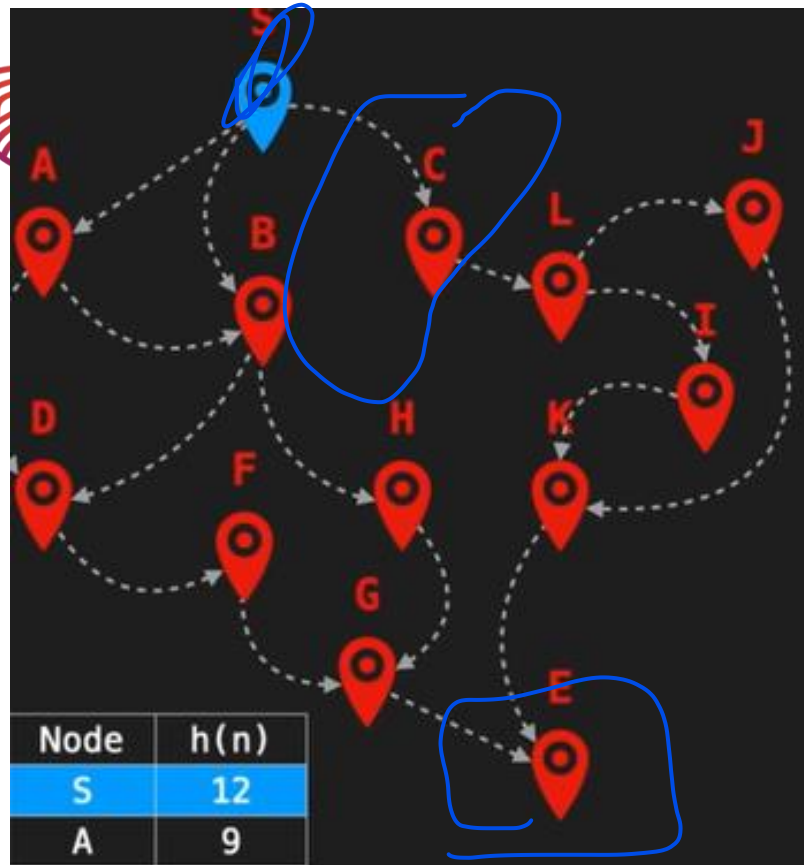
Greedy best-first search



RAMAIAH Institute of Technology

Greedy best-first search Ex: Canada to India





S

S C L

Node	h(n)
S	12
A	9
B	8
C	7
D	6
E	0
F	4
G	3
H	7
I	9
J	9
K	4
L	6

Nodes	ø

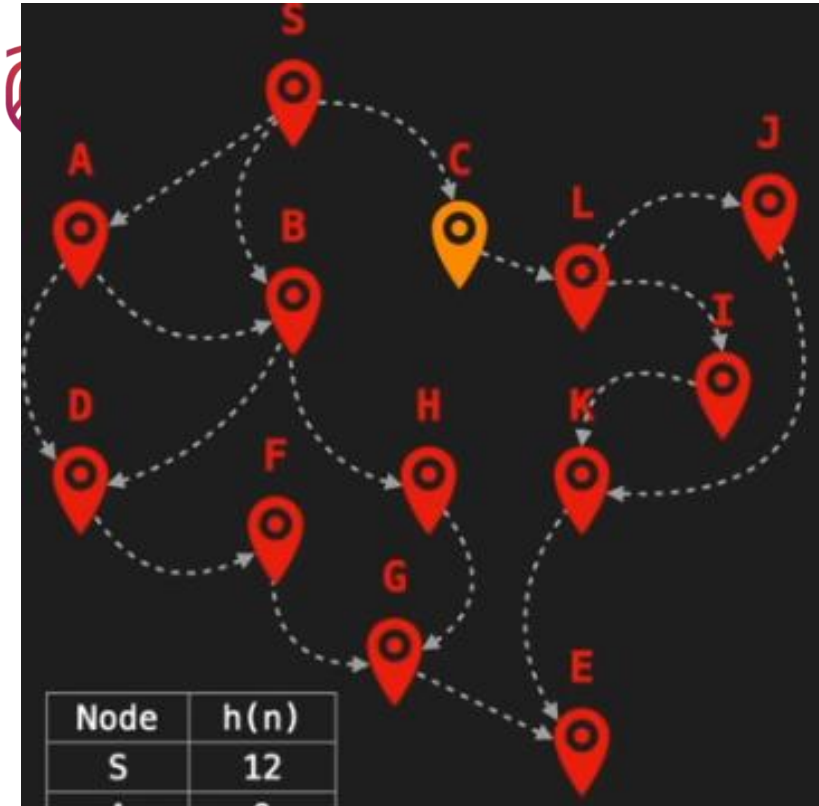
Closed List

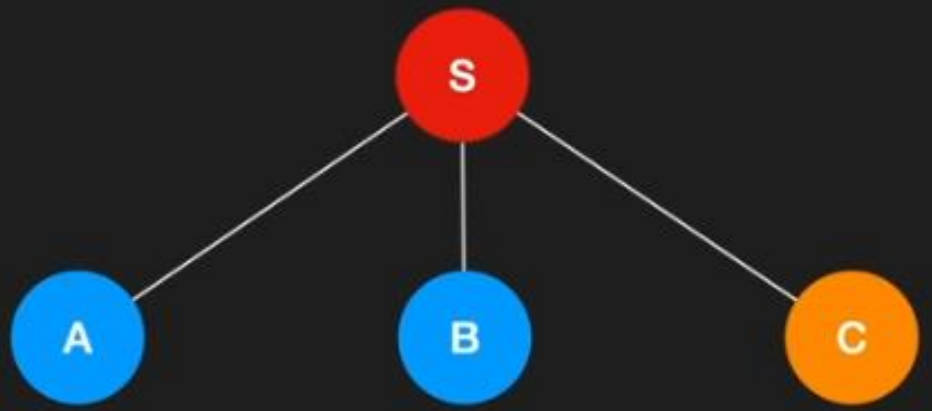
Nodes	S
f(n)	12

Open List

Heurestic Values of $h_{SLD\ is}$ straight-line distances to India.

S is added to open list.





Node	h(n)
S	12
A	9
B	8
C	7
D	6
E	0
F	4
G	3
H	7
I	9
J	9
K	4
L	6

Nodes	S

Closed List

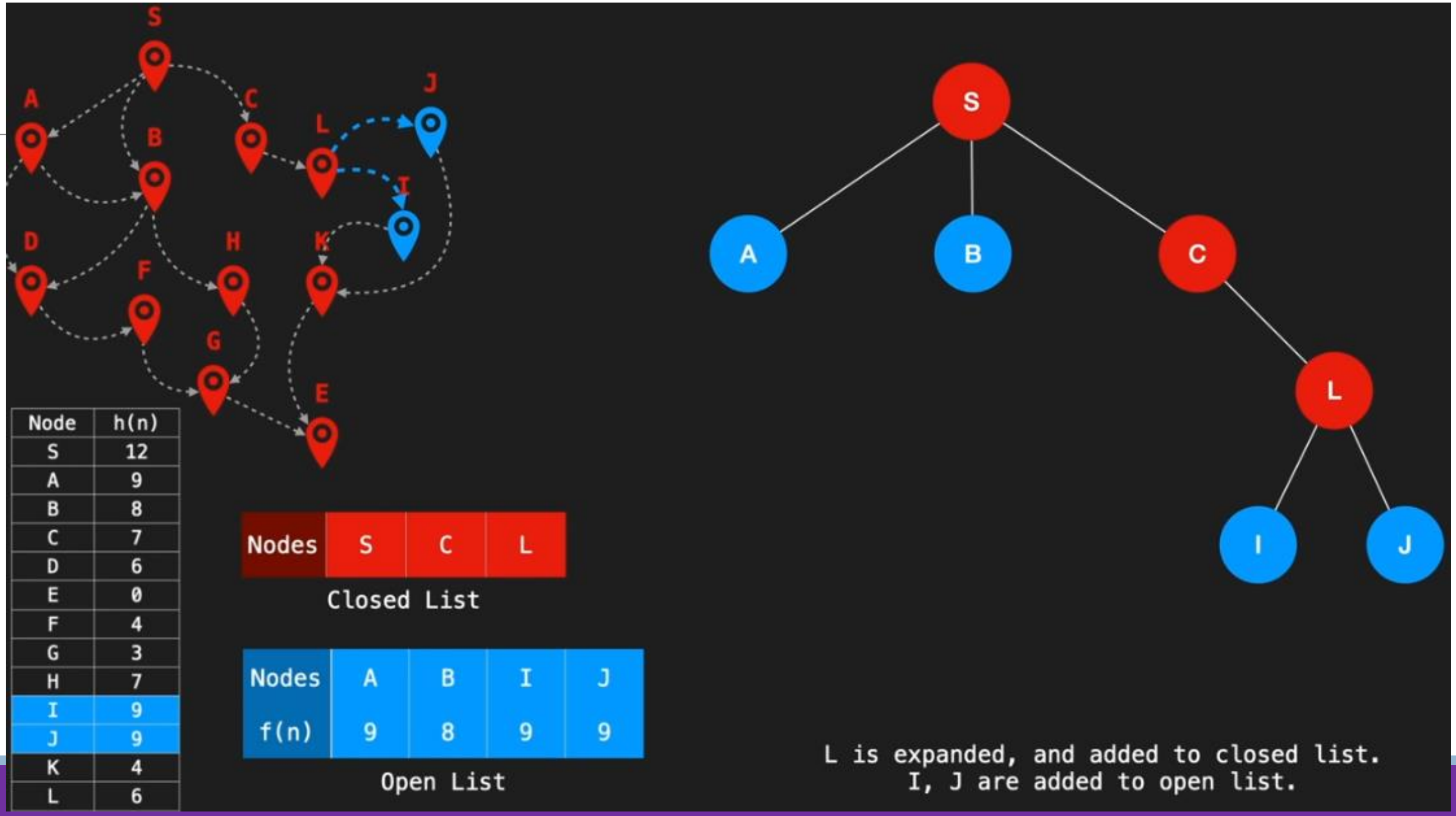
Nodes	A	B	C
f(n)	9	8	7

Open List

C is chosen next
since it has the lowest f(n) value!

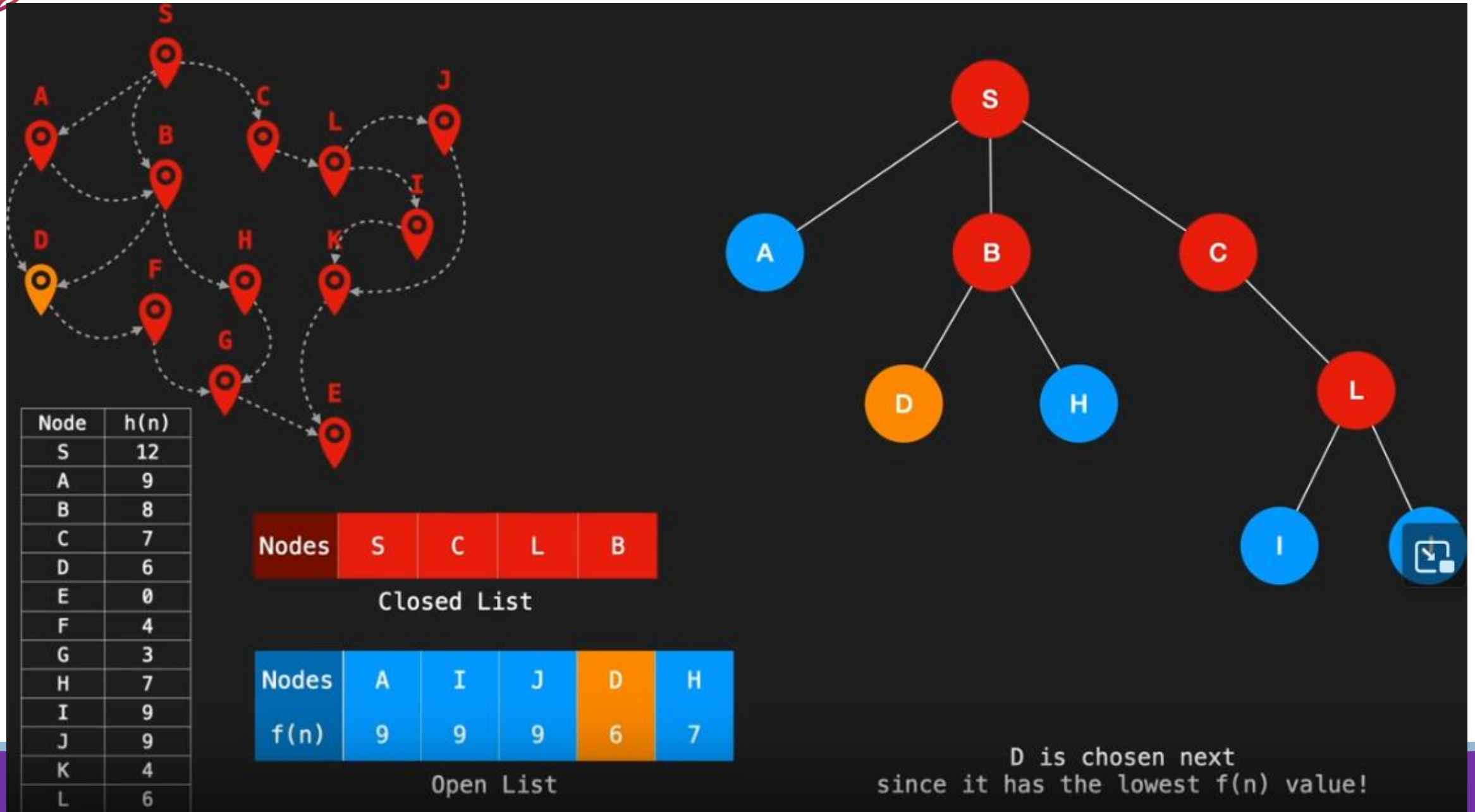
RAMAIAH Institute of Technology

Greedy best-first search Ex: Canada to India



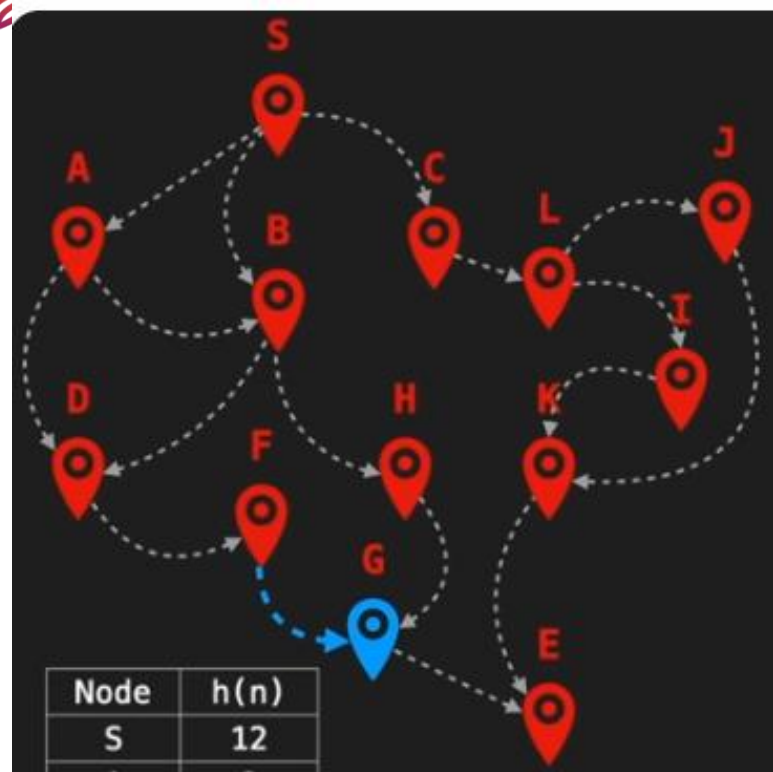
RAMAIAH Institute of Technology

Greedy best-first search Ex: Canada to India



RAMAIAH Institute of Technology

Greedy best-first search Ex: Canada to India



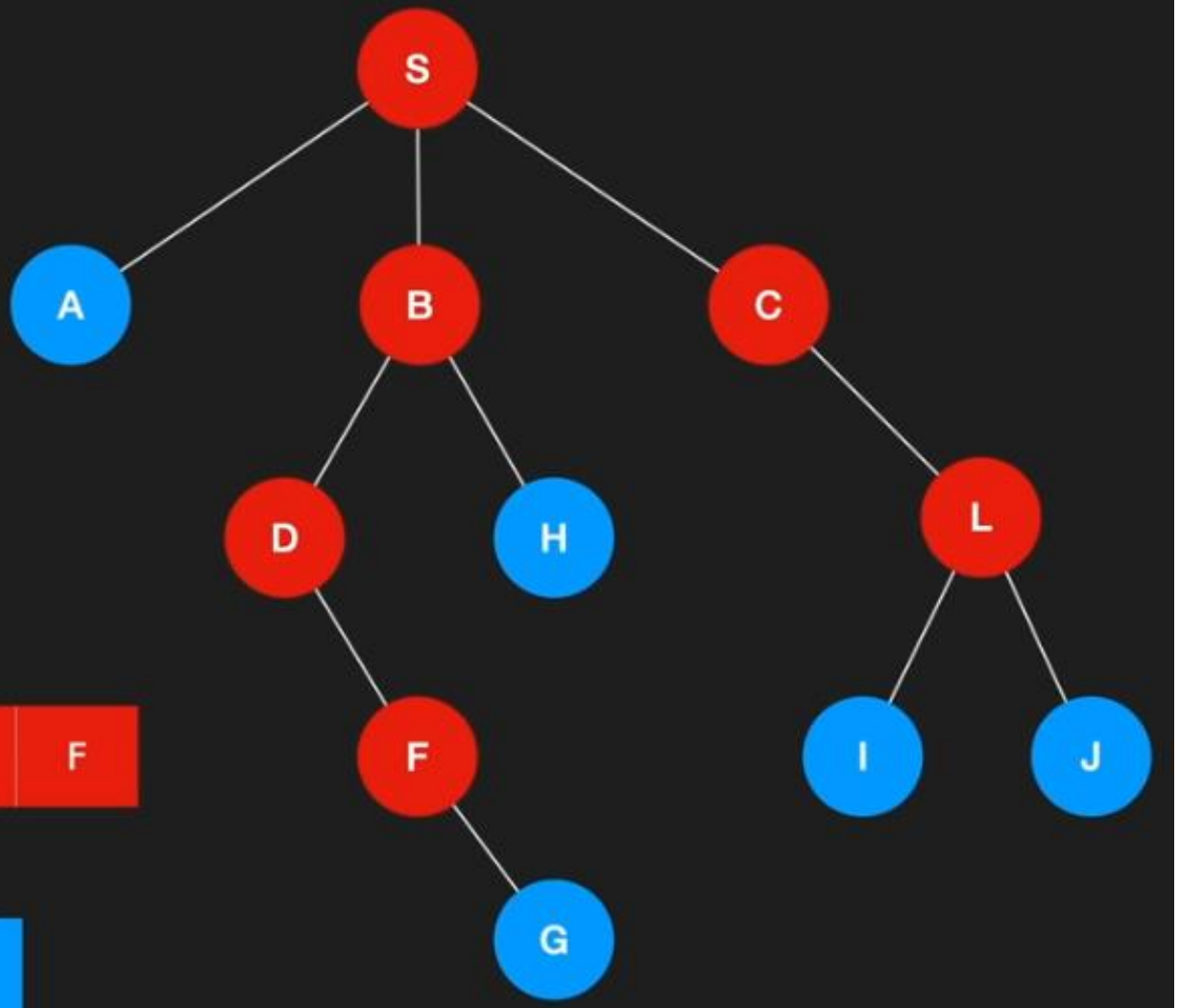
Node	h(n)
S	12
A	9
B	8
C	7
D	6
E	0
F	4
G	3
H	7
I	9
J	9
K	4
L	6

Nodes	S	C	L	B	D	F

Closed List

Nodes	A	I	J	H	G
f(n)	9	9	9	7	3

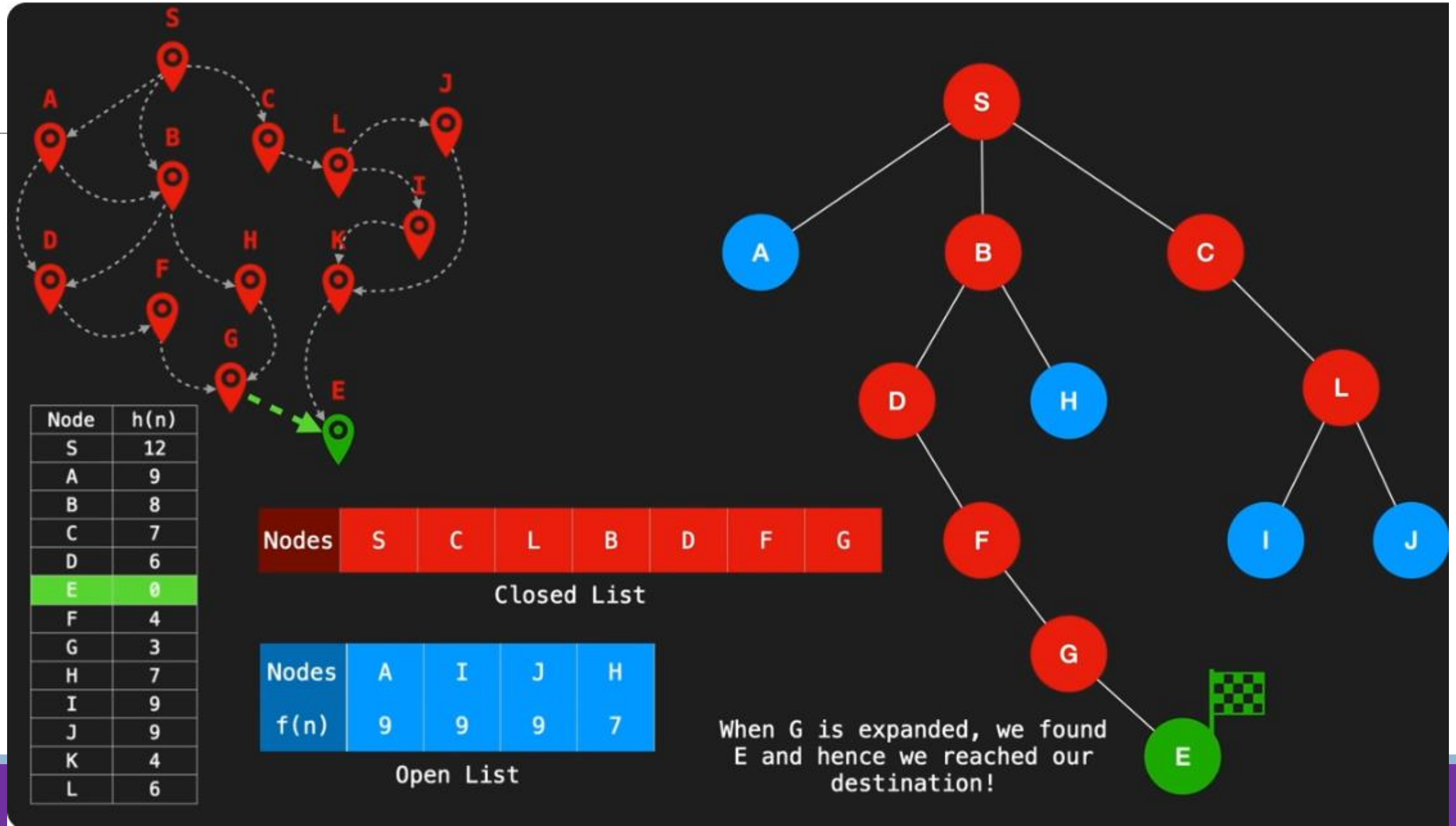
Open List



F is expanded, and added to closed list.
G is added to open list.

RAMAIAH Institute of Technology

Greedy best-first search Ex: Canada to India



RAMAIAH
Institute of Technology

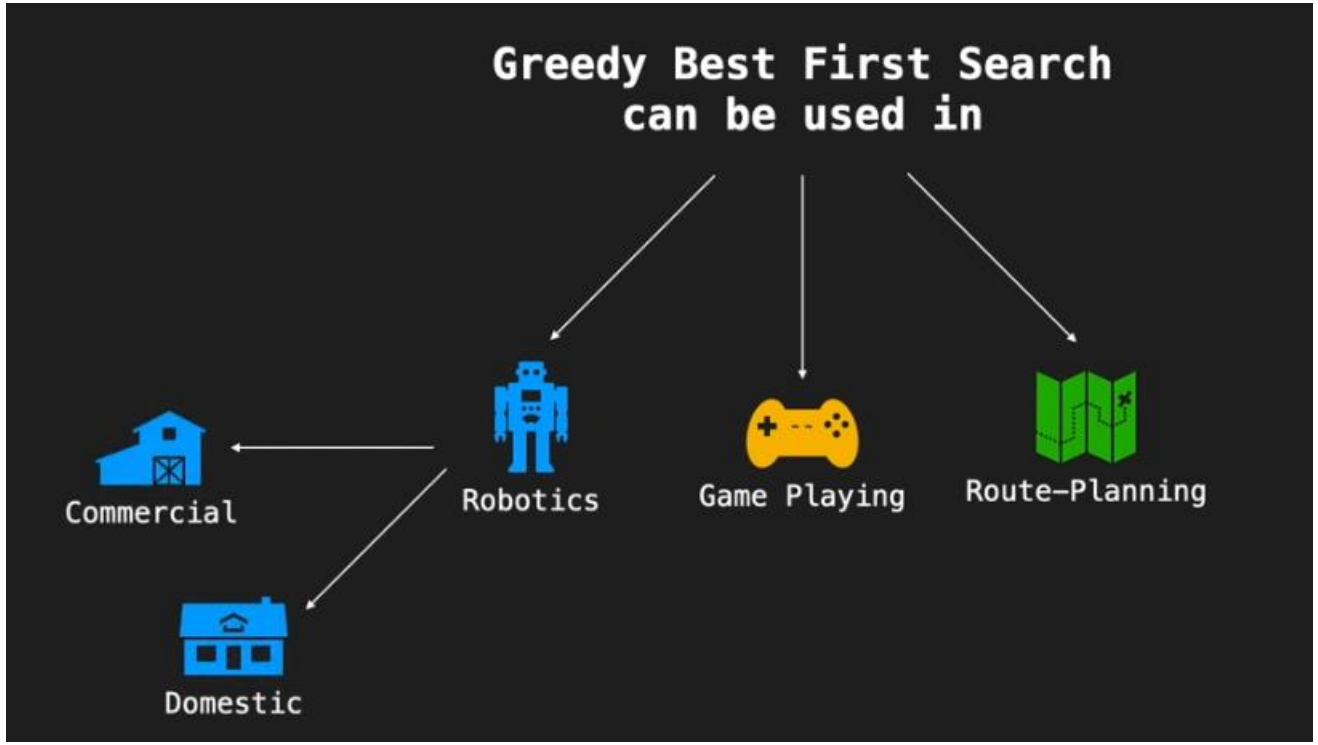
Greedy best-first search Algorithm

Algorithm

- Step 1: Place the starting node into the OPEN list.
- Step 2: If the OPEN list is empty, Stop and return failure.
- Step 3: Remove the node n from the OPEN list which has the lowest value of h(n), and place it in the CLOSED list.
- Step 4: Expand the node n and generate the successors of node n.
- Step 5: Check each successor of node n and find whether any node is a goal node or not. If any successor node is goal node, then return success and terminate the search, else proceed to Step 6.
- Step 6: For each successor node, algorithm checks for evaluation function f(n), and then check if the node has been in either OPEN or CLOSED list. If the node has not been in both list, then add it to the OPEN list.
- Step 7: Return to Step 2.

RAMAIAH
Institute of Technology

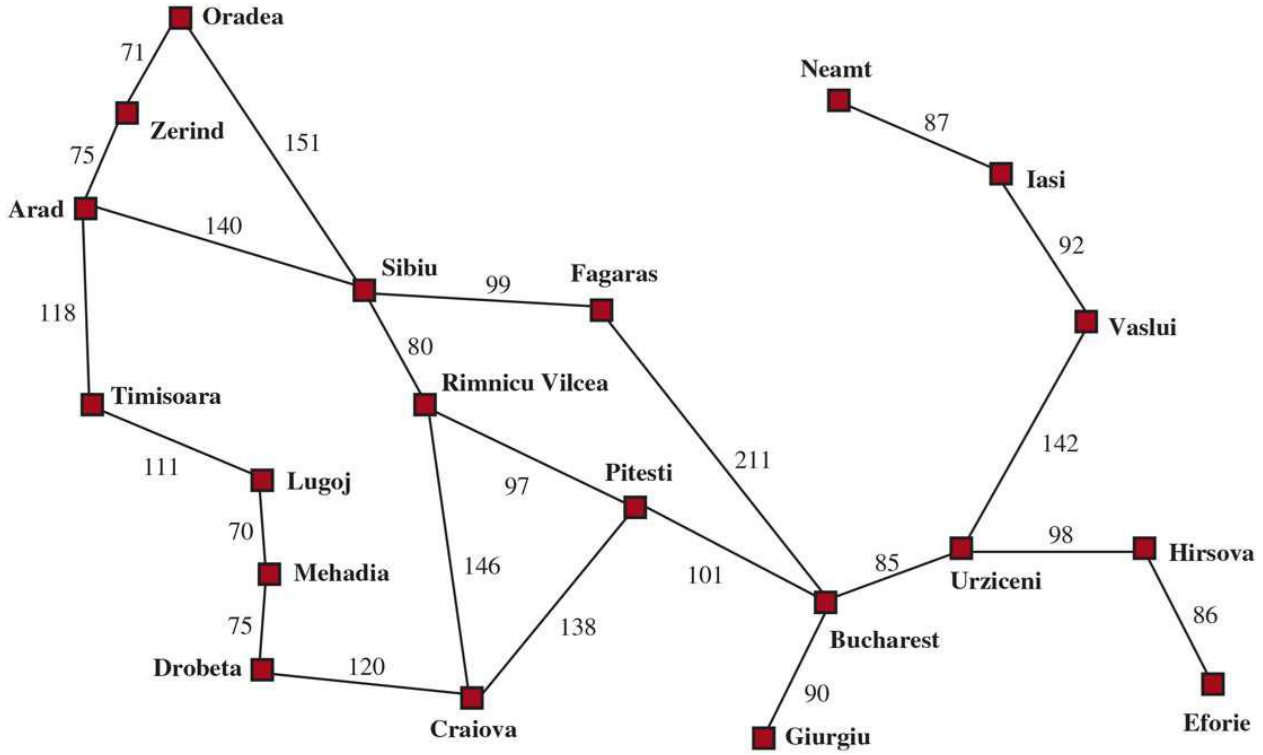
Greedy best-first search Applications



RAMAIAH
Institute of Technology

Greedy best-first search

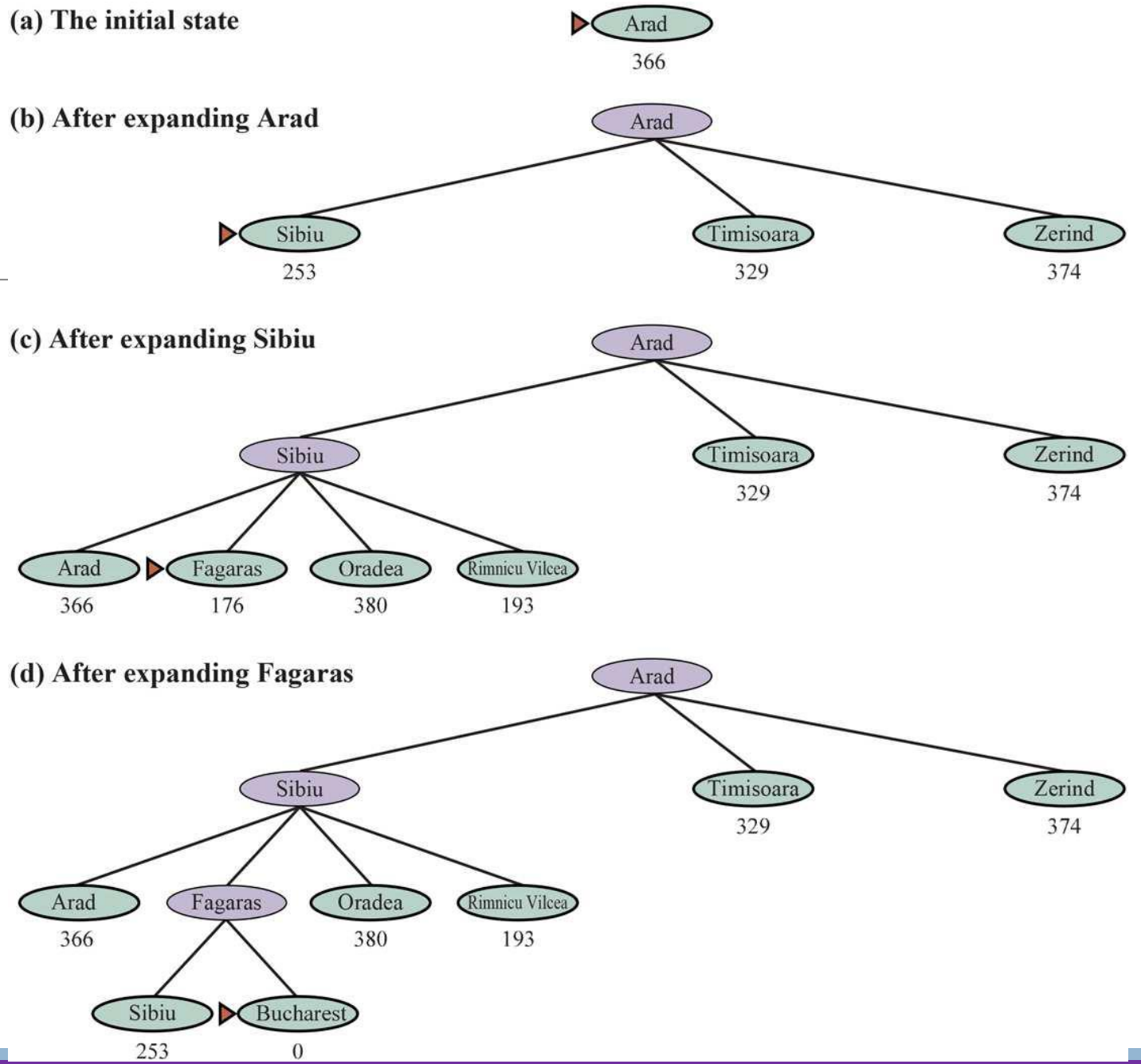
Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374



Values of h_{SLD} straight-line distances to Bucharest.

RAMAIAH
Institute of Technology

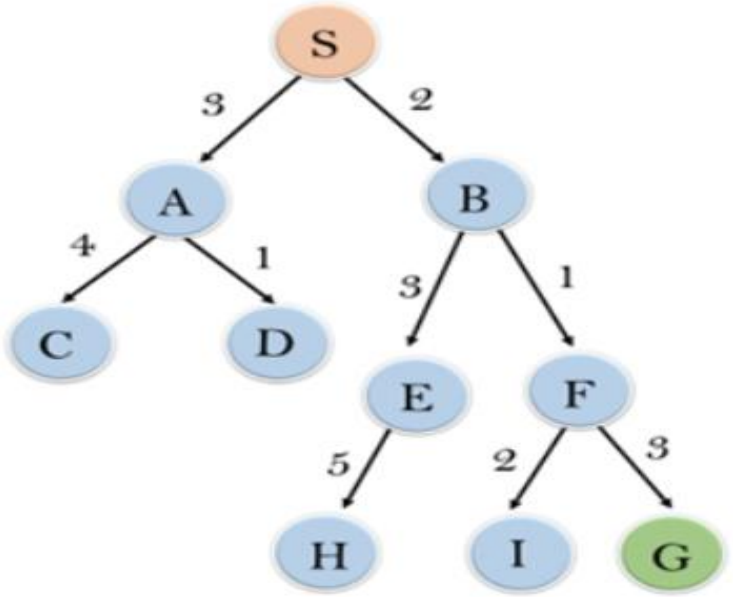
Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374



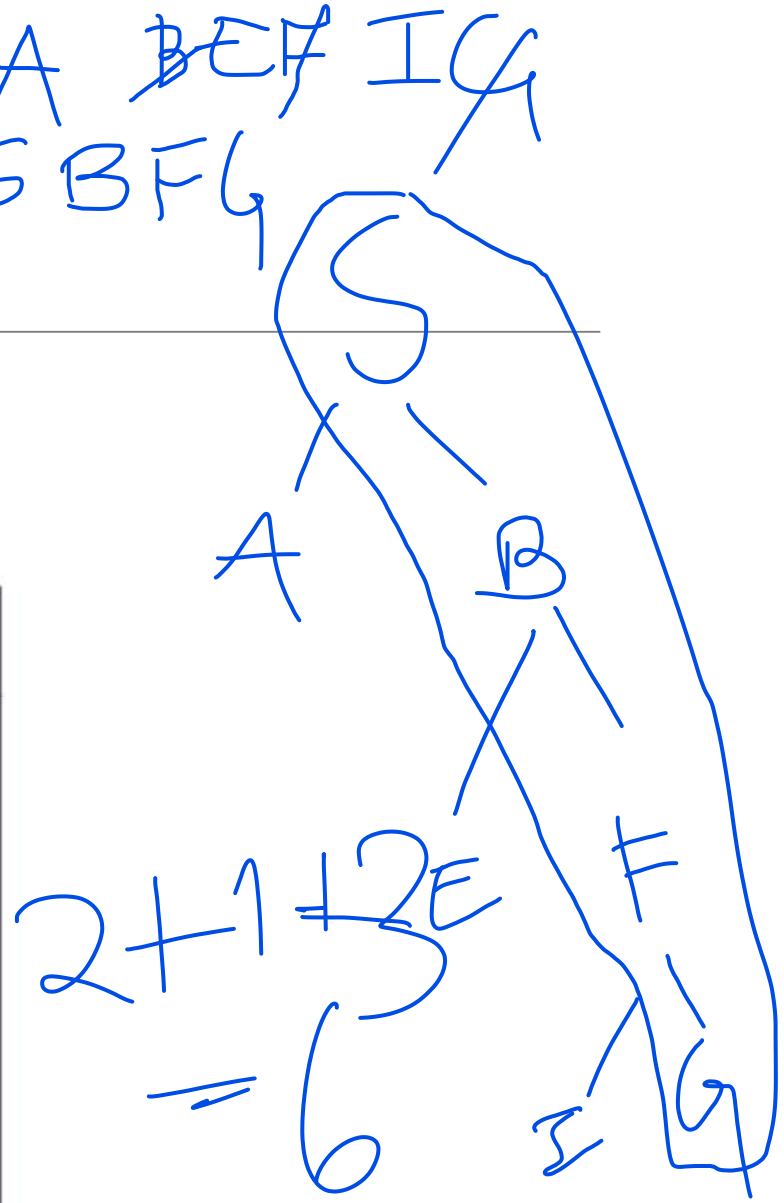
RAMAIAH
Institute of Technology

Greedy best-first search

Final solution path will be **S----> B----->F----> G = 2+1+3 =8**



node	H (n)
A	12
B	4
C	7
D	3
E	8
F	2
H	4
I	9
S	13
G	0



RAMAIAH
Institute of Technology

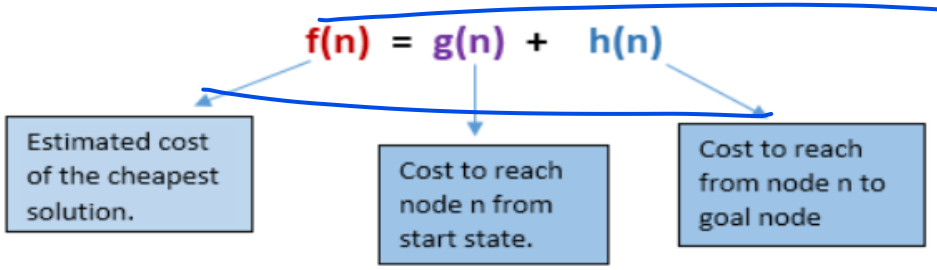
Properties of Greedy best-first search

- **Time Complexity**: The worst case time complexity of Greedy best first search is $O(b^m)$.
- **Space Complexity**: The worst case space complexity of Greedy best first search is $O(b^m)$. Where, m is the maximum depth of the search space.
- **Complete**: Greedy best-first graph search is complete in finite state spaces, but not in infinite ones.
- **Optimal**: Greedy best first search algorithm is not optimal.

RAMAIAH
Institute of Technology

A* search

- A* search is the most commonly known form of best-first search.
- It uses heuristic function h(n), and path cost to reach the node n from the start state g(n).
- A* search algorithm finds the shortest path through the search space using the heuristic function.
- A* search algorithm uses search heuristic as well as the cost to reach the node.



f(n) = estimated cost of the best path that continues from n to a goal.

RAMAIAH
Institute of Technology

A* search

f (n) - function that gives an evaluation of the state

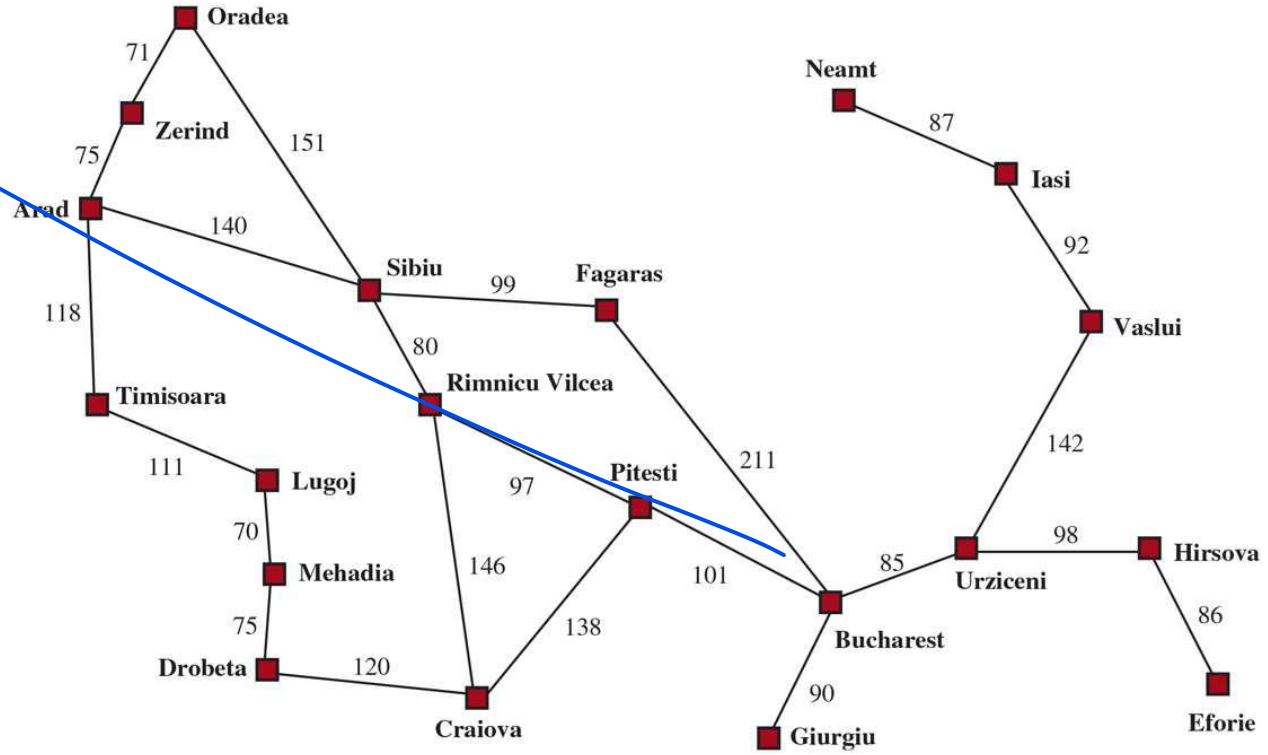
g(n) - the cost of getting from the initial state to the current state

h(n) - the cost of getting from the current state to a goal state

RAMAIAH
Institute of Technology

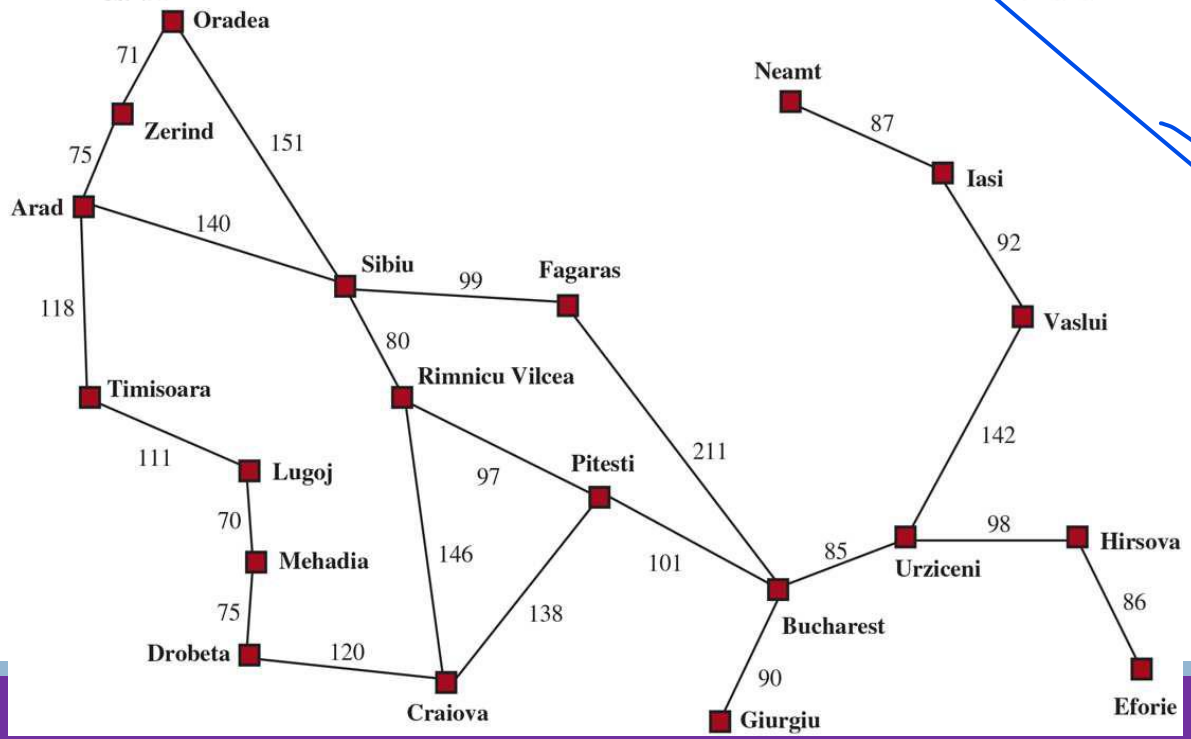
Greedy best-first search

Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

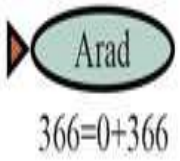


Values of hSLDstraight-line distances to Bucharest.

Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

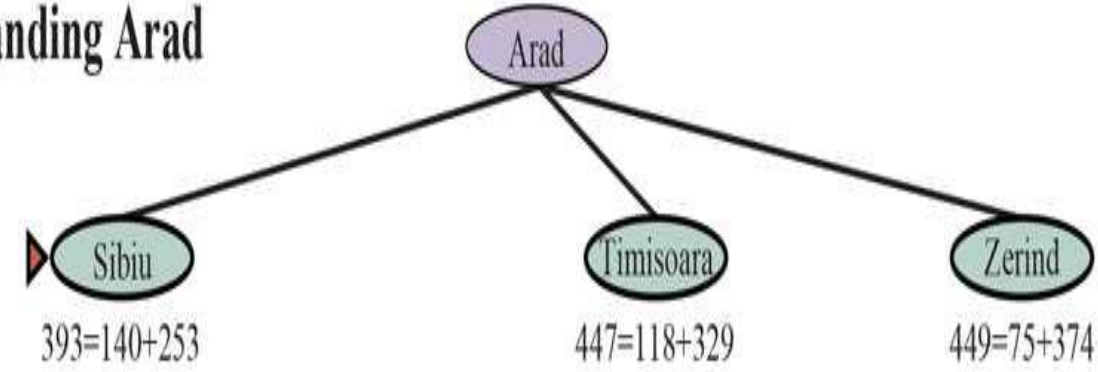


(a) The initial state

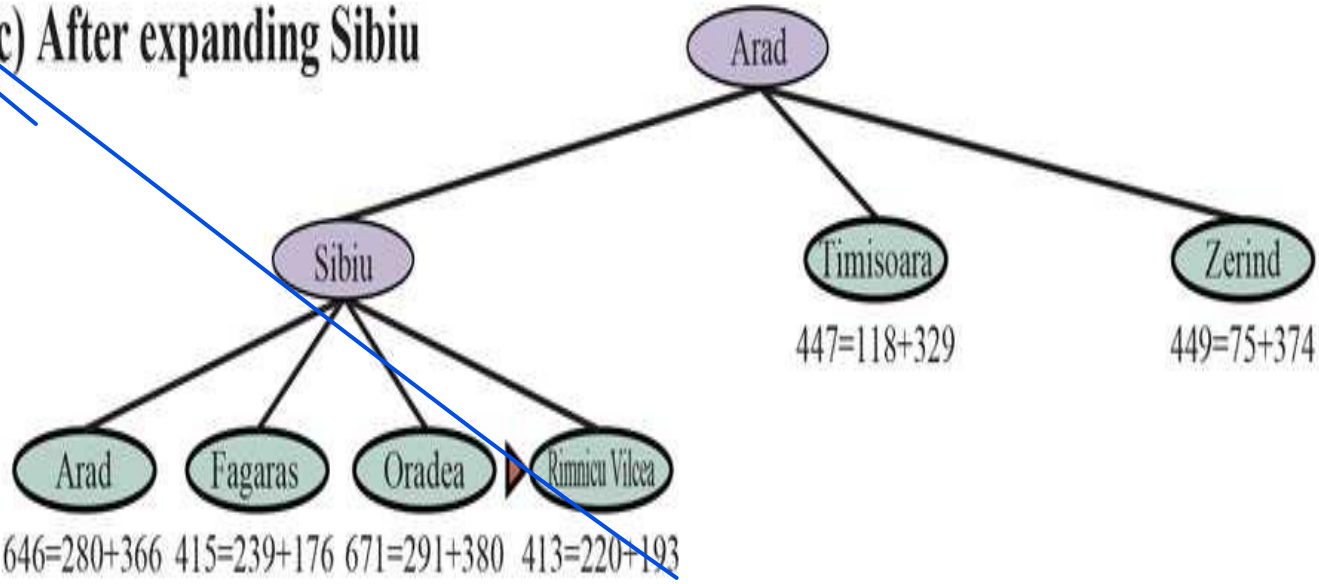


366=0+366

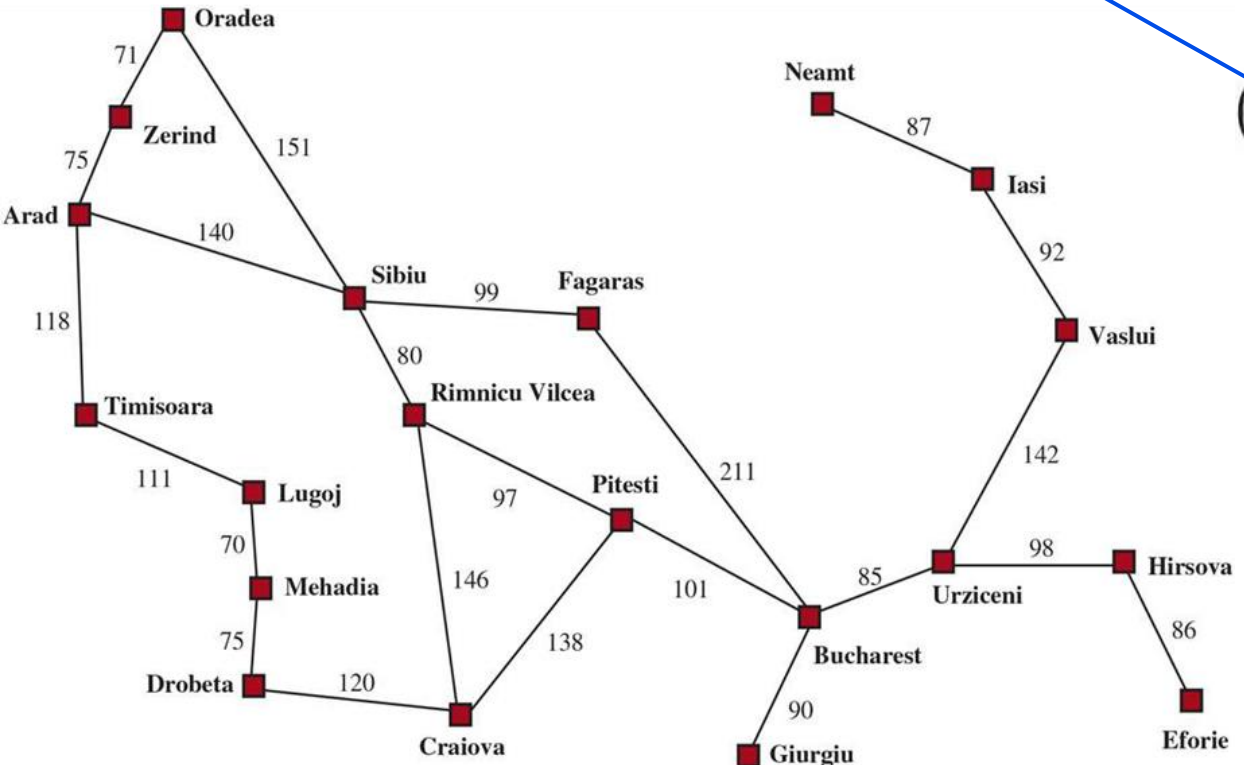
(b) After expanding Arad



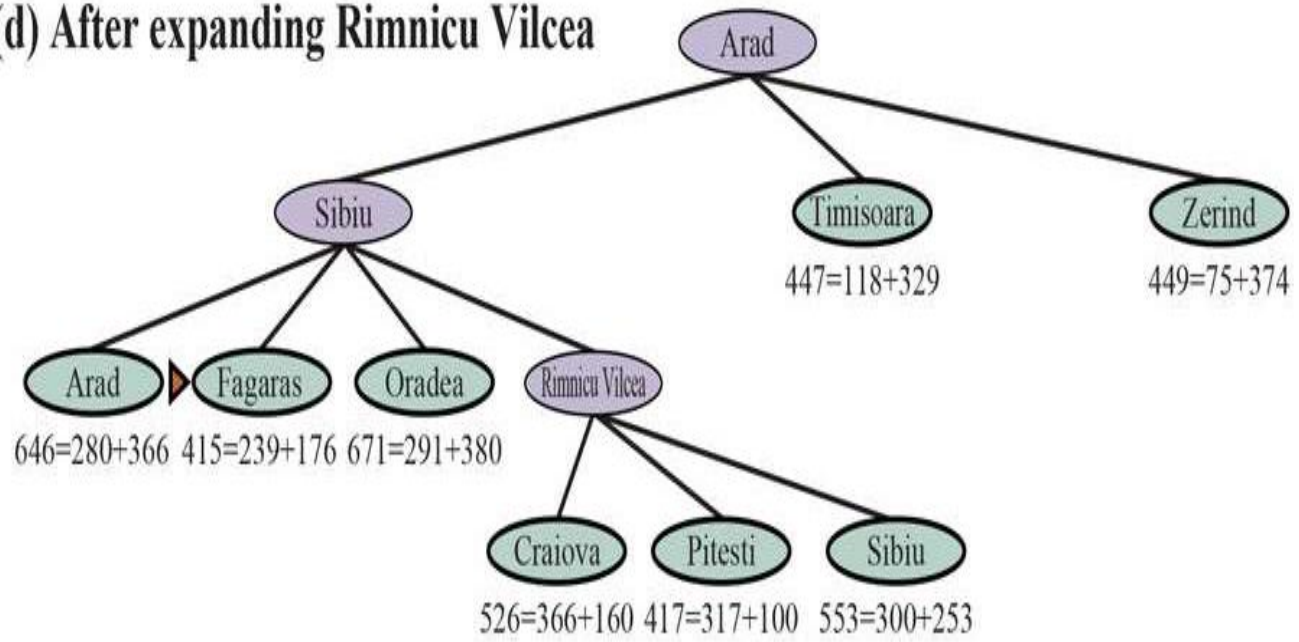
(c) After expanding Sibiu



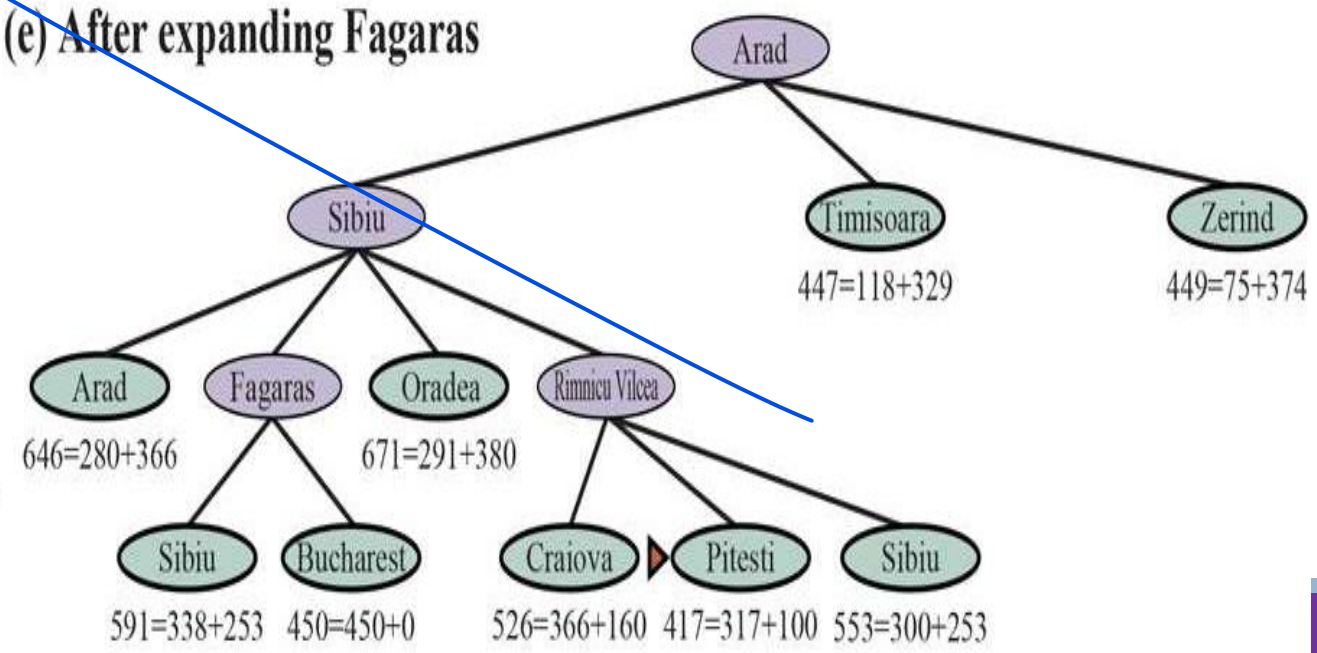
Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374



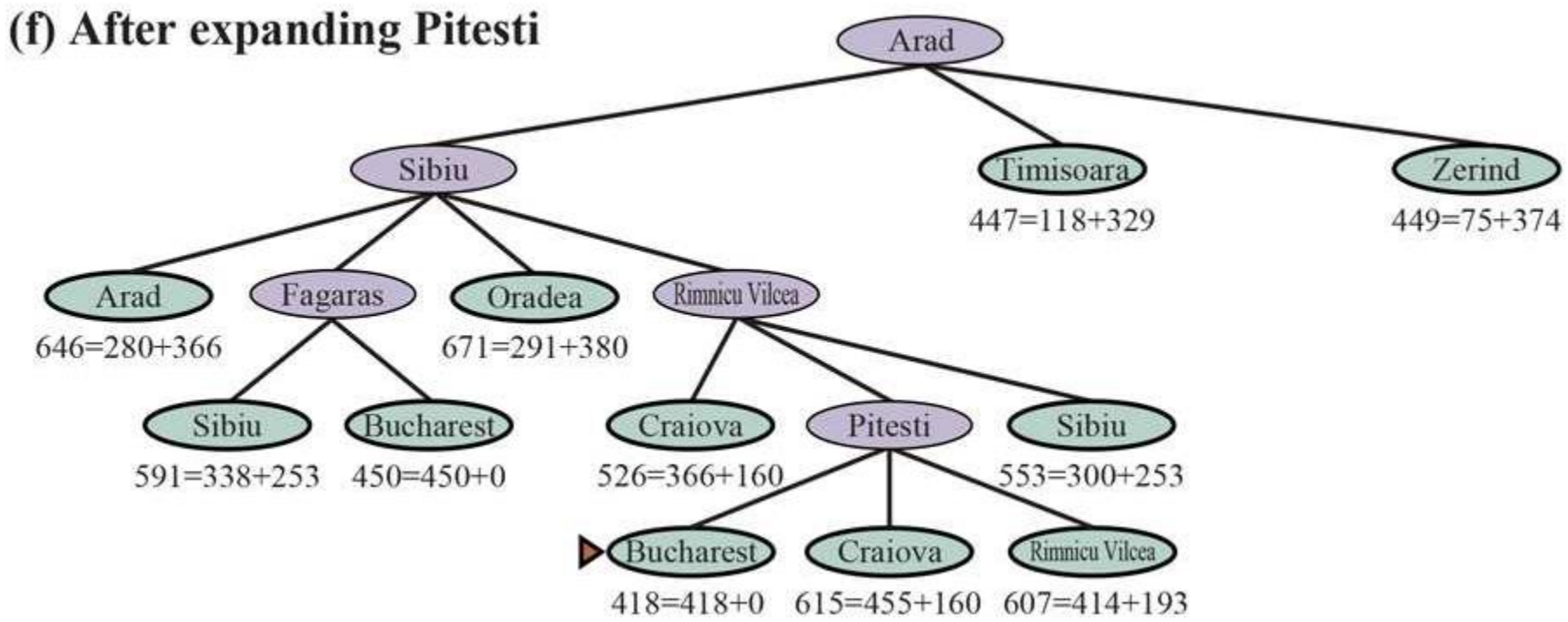
(d) After expanding Rimnicu Vilcea



(e) After expanding Fagaras



A* search

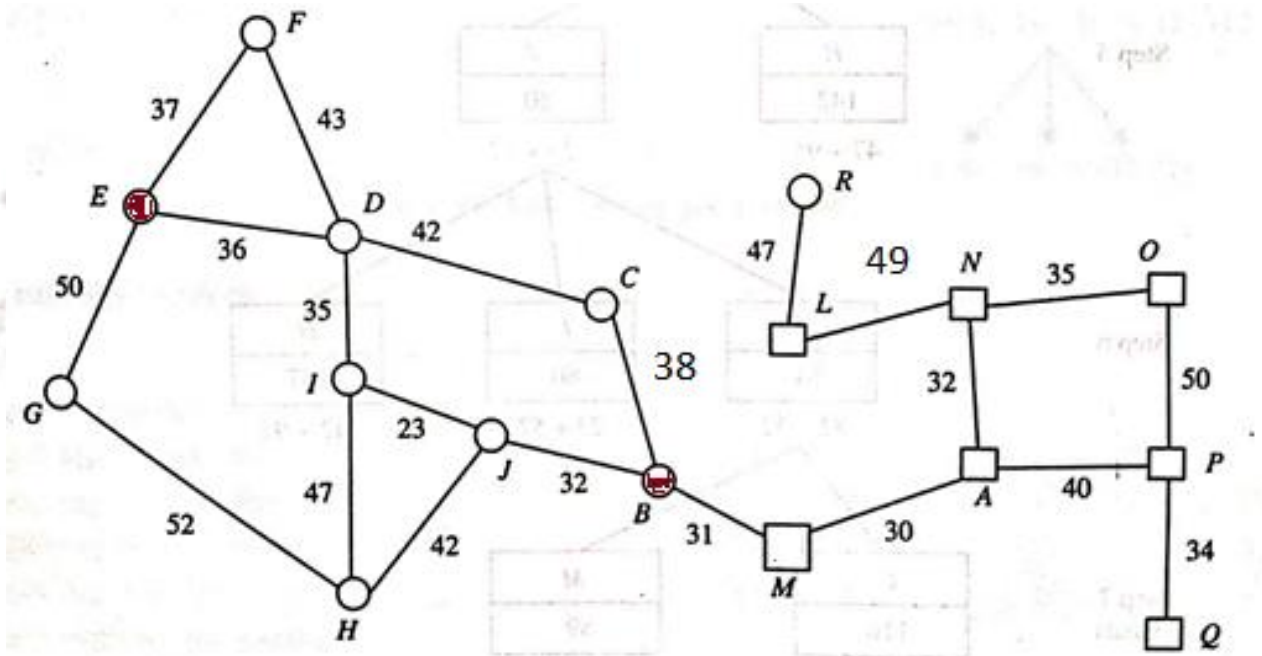

(f) After expanding Pitesti
Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
Oradea
671=291+380
Rimnicu Vilcea
Sibiu
591=338+253
Bucharest
450=450+0
Craiova
526=366+160
Pitesti
Sibiu
553=300+253
Bucharest
418=418+0
Craiova
615=455+160
Rimnicu Vilcea
607=414+193

RAMAIAH
Institute of Technology

Example: A* search

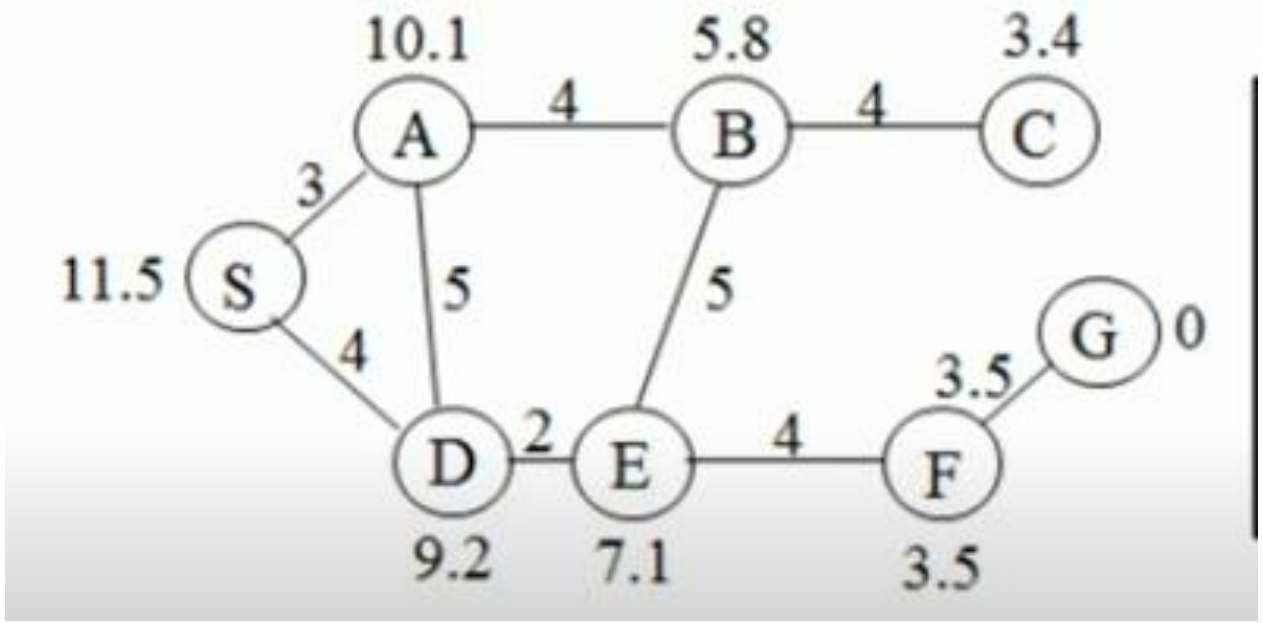
Heuristic Table

A 24	F 145	K 108	P 65
B 28	G 123	L 51	Q 11
C 70	H 95	N 58	
D 95	I 77	O 10	
E 118	J 57	M 0	



RAMAIAH
Institute of Technology

Example: A* search



RAMAIAH
Institute of Technology

Properties of A* search

Complete: A* search is complete.

Whether A* is cost-optimal depends on certain properties of the heuristic.

Optimal: A* search algorithm is optimal if it follows below two conditions:

Admissible: an admissible heuristic is one that never overestimates the cost to reach a goal

Consistency: A heuristic h(n) is consistent if, for every node n and every successor n′ of n generated by an action a we have:

$h(n) \leq c(n, a, n') + h(n')$.

Time Complexity: O(b^d), where b is the branching factor.

Space Complexity: O(b^d)

A* search

Advantages

- A* search algorithm is the best algorithm than other search algorithms.
- A* search algorithm is optimal and complete.
- This algorithm can solve very complex problems.

Disadvantages

- It does not always produce the shortest path as it mostly based on heuristics and approximation.
- A* search algorithm has some complexity issues.
- The main drawback of A* is memory requirement as it keeps all generated nodes in the memory, so it is not practical for various large-scale problems.