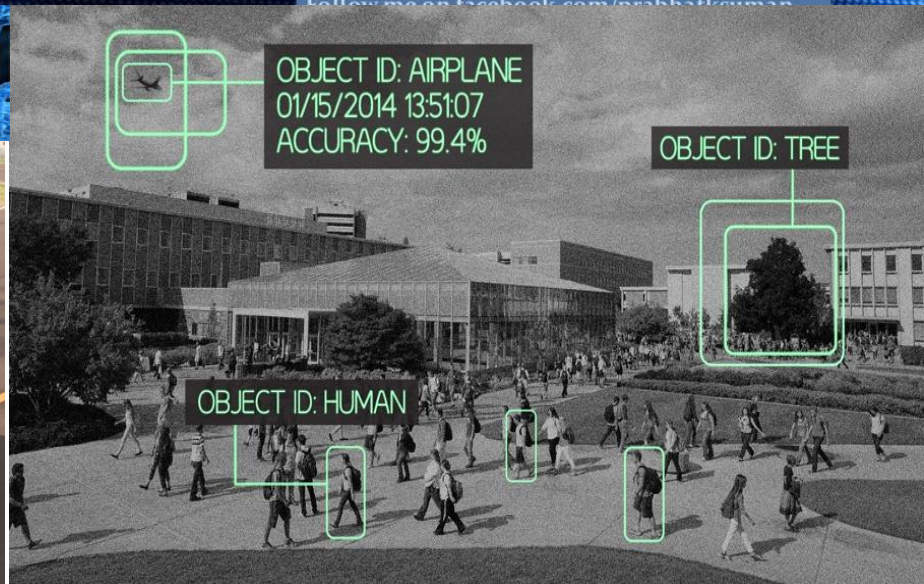
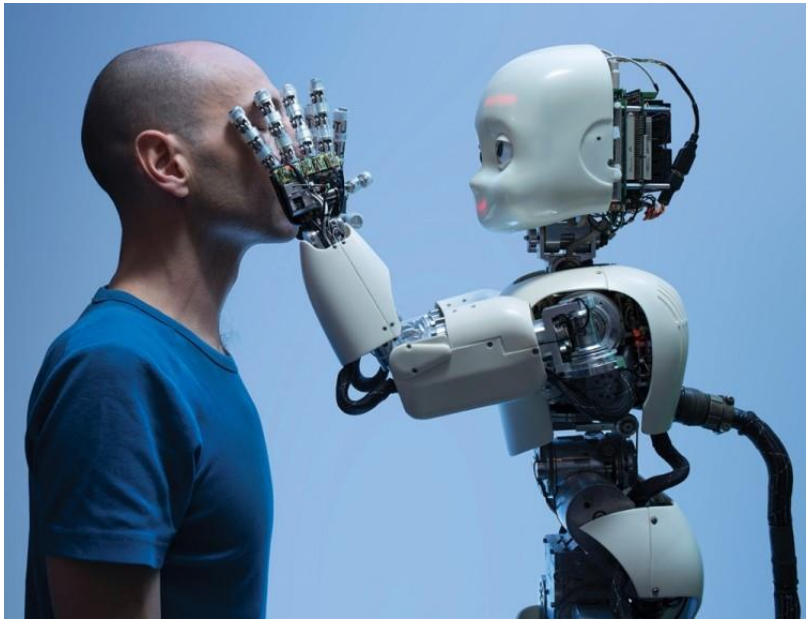
Artificial Intelligence



LOVELY PROFESSIONAL UNIVERSITY

Unit-1

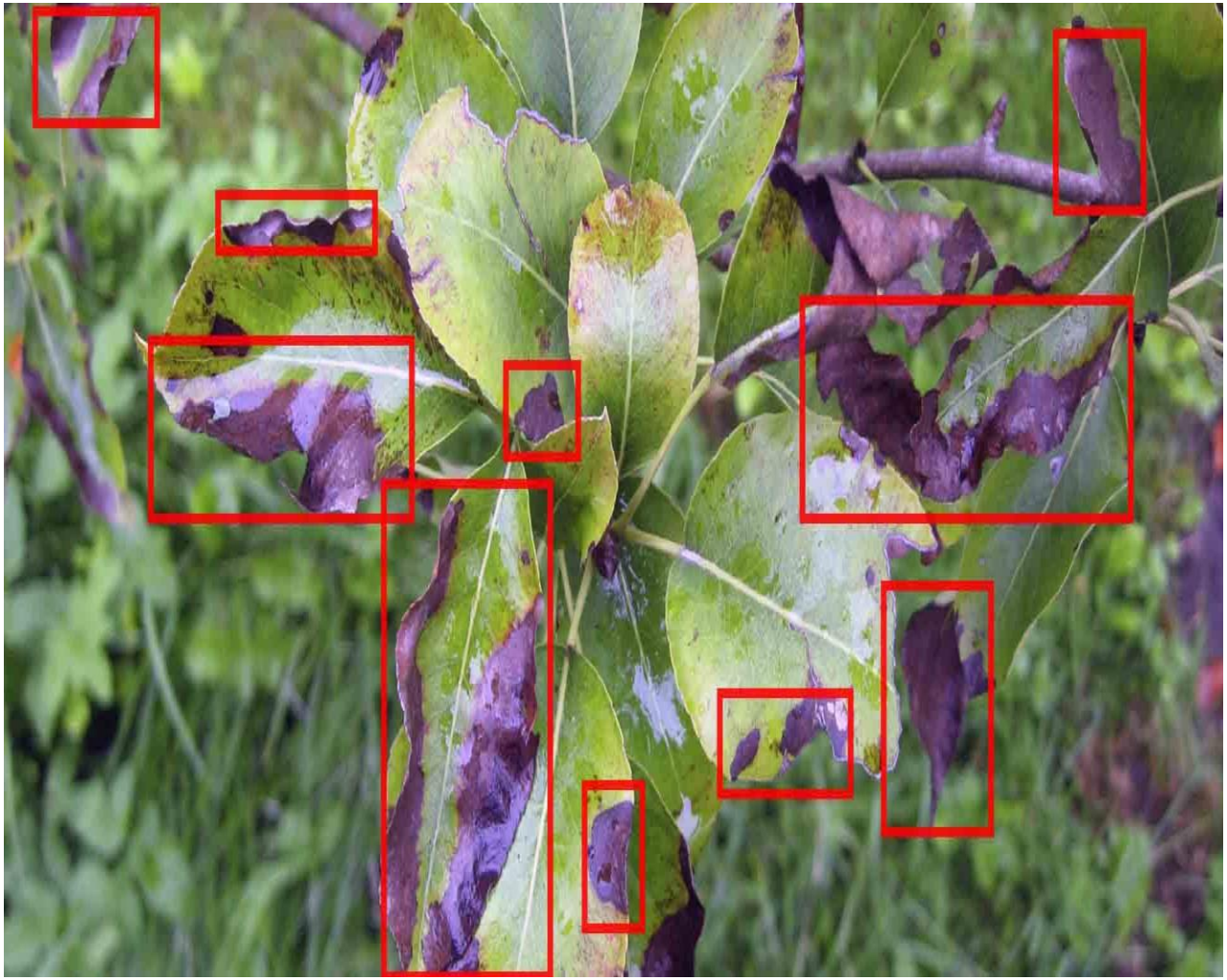
Basic Preliminaries of AI



Applications of AI

- Healthcare
- Automobile
- Finance
- Surveillance
- Social Media
- Entertainment
- Education
- Space Exploration
- Gaming
- Robotics
- Agriculture
- E-commerce

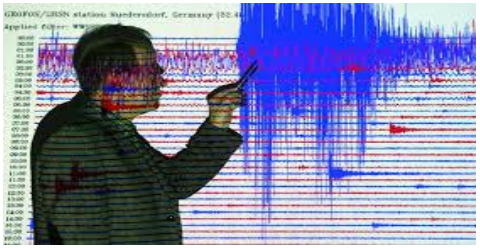
Case Study 1

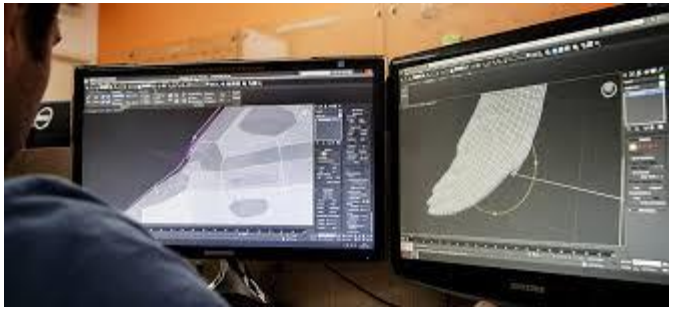


Case Study 2

Pillars of Intelligence

- Perceive
- Understand
- Prediction
- Manipulate





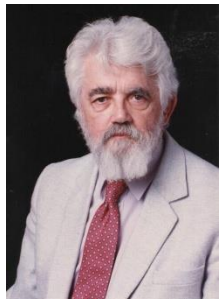
Artificial Intelligence

- **Artificial intelligence** (**AI**) is the intelligence exhibited by machines or software, and it is the branch of computer science that develops machines and software with human-like intelligence.
- It is defined as the study and design of intelligent agents", where an intelligent agent is a system that perceives its environment and takes actions that maximize its chances of success
- AI currently encompasses a huge variety of subfields, ranging from the general (learning and perception) to the specific, such as playing chess, proving mathematical theorems, writing poetry, driving a car on a crowded street, and diagnosing diseases.

Machine learning

- It is a branch of artificial intelligence based on the idea that machines should be able to learn and adapt through experience.
- examples of machine learning applications you may be familiar with:
- The heavily hyped, self-driving Google car? The essence of machine learning.
- Online recommendation offers such as those from Amazon and Netflix? Machine learning applications for everyday life.
- Fraud detection? One of the more obvious, important uses in our world today.

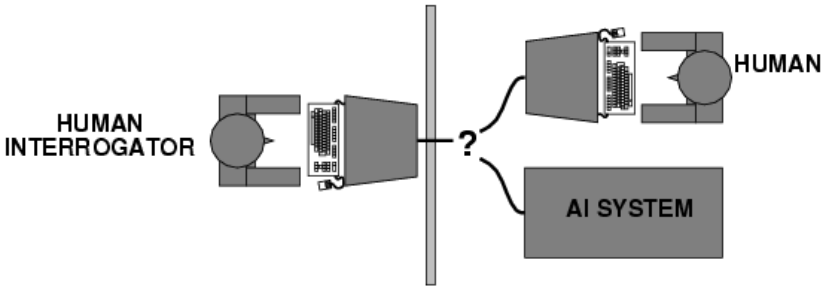
A brief history of artificial intelligence



- Work started in earnest soon after World War II, and the name itself was coined in 1956.
- The effort to finalize the language of logic continued with Leibniz and Newton. George Boole developed Boolean algebra in the nineteenth century that laid the foundation of computer circuits.
- However, the main idea of a thinking machine came from Alan Turing, who proposed the Turing test. The term “artificial intelligence” was first coined by John McCarthy in 1956.

Acting humanly: Turing test

- Turing (1950) "Computing machinery and intelligence“
- "Can machines think?" → "Can machines behave intelligently?“
- Operational test



- Suggests major components required for AI:
  - knowledge representation
  - reasoning,
  - language/image understanding,
  - learning

* Question: is it important that an intelligent system act like a human?

AI Paradigm

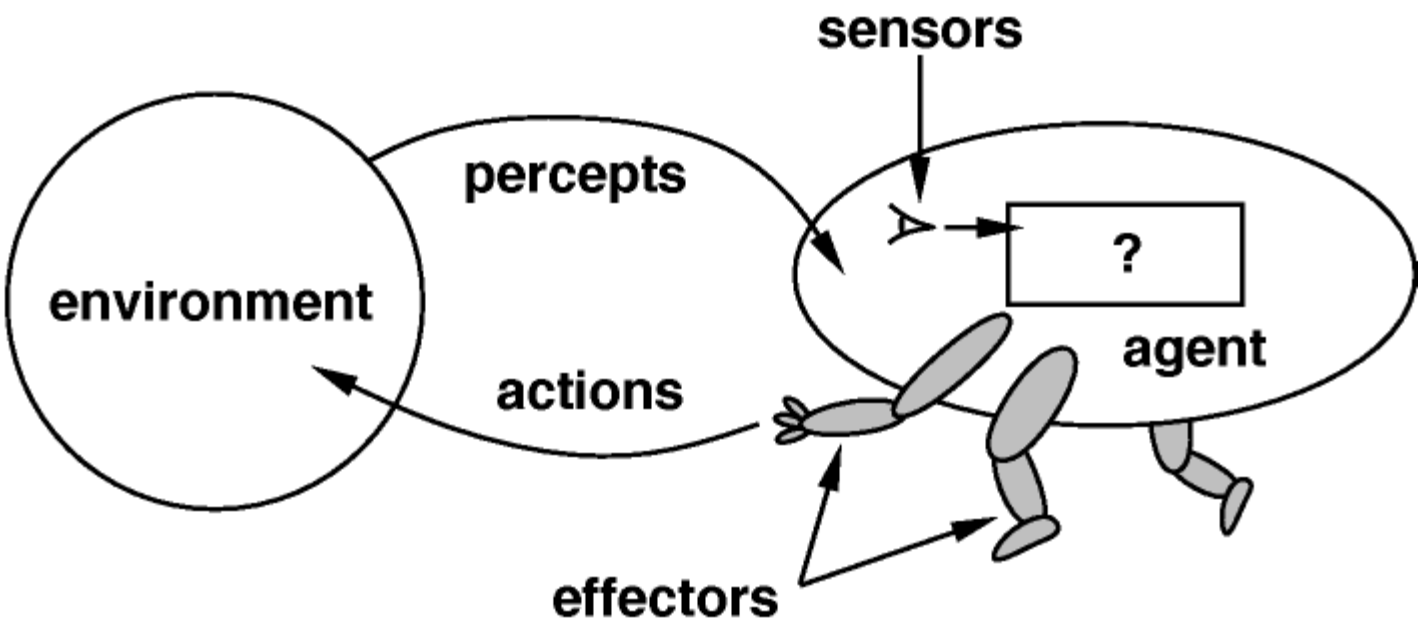
Thinking Humanly	**Thinking Rationally**
"The exciting new effort to make computers think . . . *machines with minds*, in the full and literal sense." (Haugeland, 1985) "[The automation of] activities that we associate with human thinking, activities such as decision-making, problem solving, learning . . ." (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)

Figure 1.1 Some definitions of artificial intelligence, organized into four categories.

Examples of AI and Intelligence Behaviour

- Driverless Car
- Chatbots
- Games(Chess,
- Extrapolate*
- Autopilot mode
- Google translator

Who will bring intelligency , goals…… ????



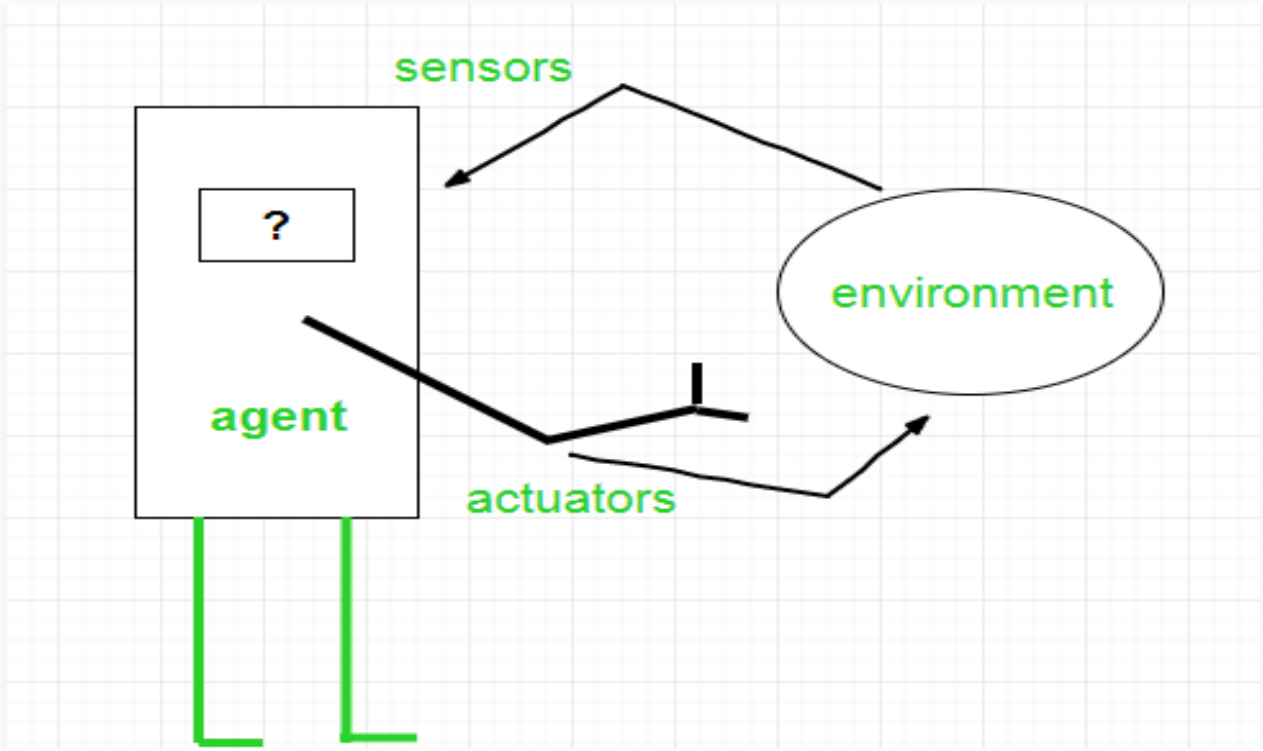
Agents in Artificial Intelligence

Artificial intelligence is defined as study of rational agents. A rational agent could be anything which makes decisions, like a person, firm, machine, or software. It carries out an action with the best outcome after considering past and current percepts(agent's perceptual inputs at a given instance).

An AI system is composed of an **agent and its environment**. The agents act in their environment. The environment may contain other agents. An agent is anything that can be viewed as :

- perceiving its environment through **sensors** and
- acting upon that environment through **actuators**

Note : Every agent can perceive its own actions (but not always the effects)



To understand the structure of Intelligent Agents, we should be familiar with *Architecture* and *Agent Program*. **Architecture** is the machinery that the agent executes on. It is a device with sensors and actuators, for example : a robotic car, a camera, a PC. **Agent program** is an implementation of an agent function. An **agent function** is a map from the percept sequence(history of all that an agent has perceived till date) to an action.

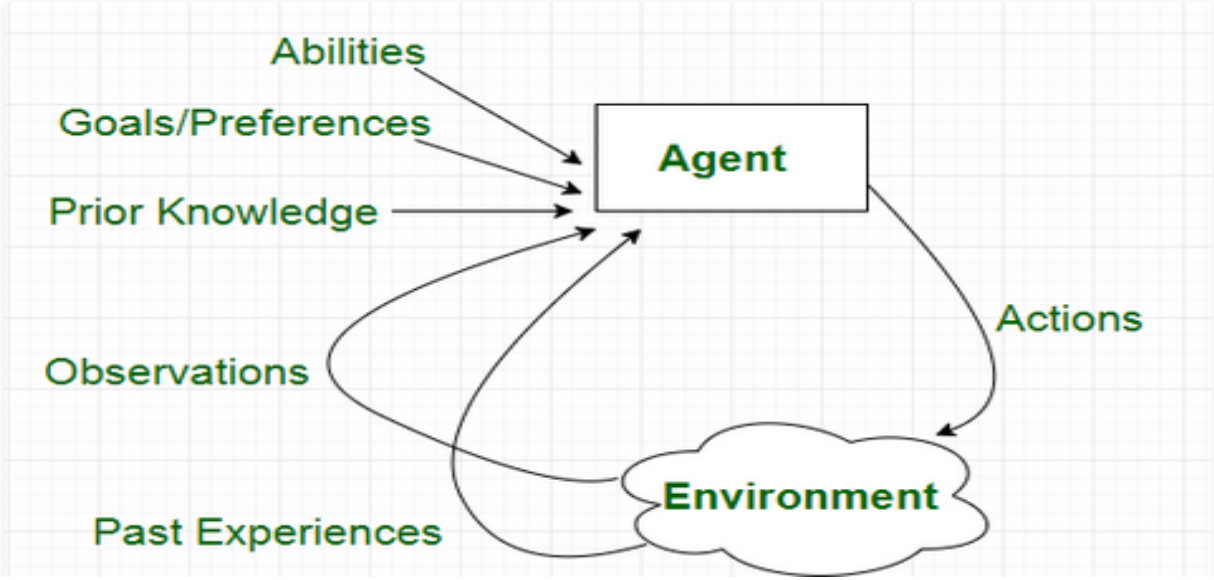
> *Agent = Architecture + Agent Program*

Examples of Agent:-

A **software agent** has Keystrokes, file contents, received network packages which act as sensors and displays on the screen, files, sent network packets acting as actuators.

A **Human agent** has eyes, ears, and other organs which act as sensors and hands, legs, mouth, and other body parts acting as actuators.

A **Robotic agent** has Cameras and infrared range finders which act as sensors and various motors acting as actuators.



Agent

- Sensors for perception
- Have goals
- Implements mapping from percept sequence to actions with the use of programs.
- Autonomous agent decide autonomously which action to take in the current situation to maximize progress towards its goals.
- Performance measure: A subjective measure to characterize how successfully an agent is(Examples :
- Softbots

→ Askjeevans.com

. Expert system

-> Cardiologist

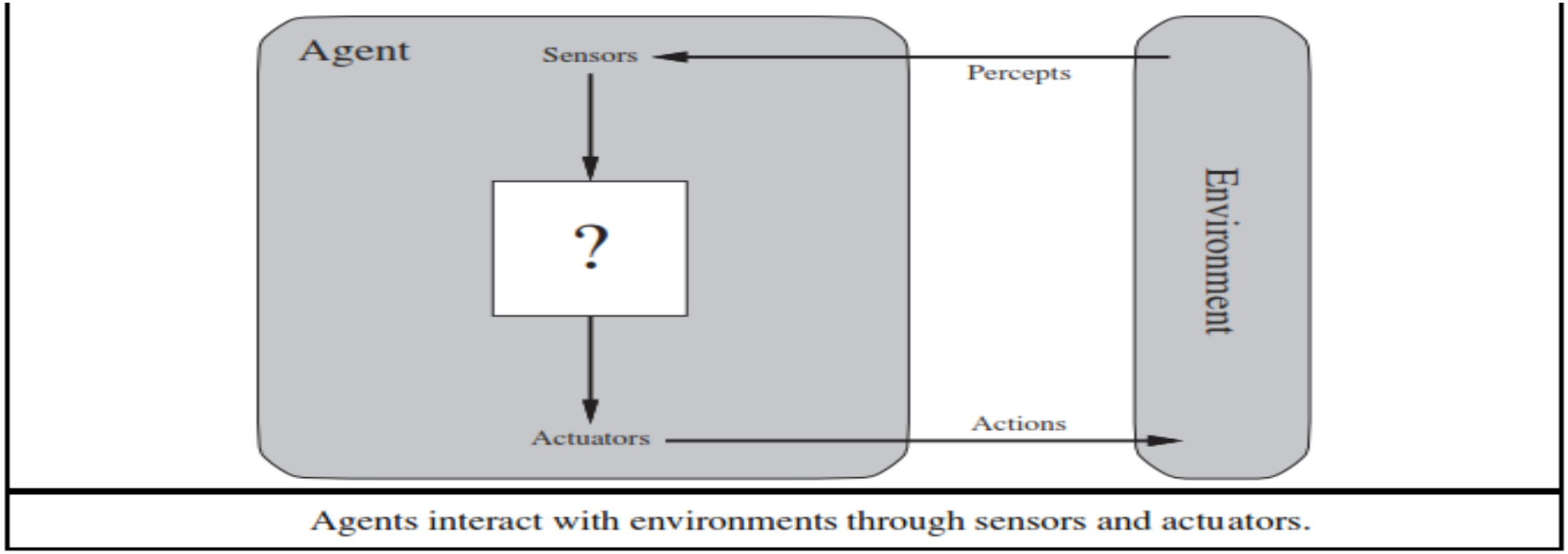
. Autonomous Spacecraft

. Intelligent buildings

Features

- Acting
- Sensing
- Understanding
- Reasoning
- Learning
- Must be rational
- Should have goals
- Provide solutions to various environment which are based on **Performance**, **Environment**, **Actuators** and **Sensors**.PEAS(e.g Speed, Power usage , accuracy, money etc.)

Intelligent Agent



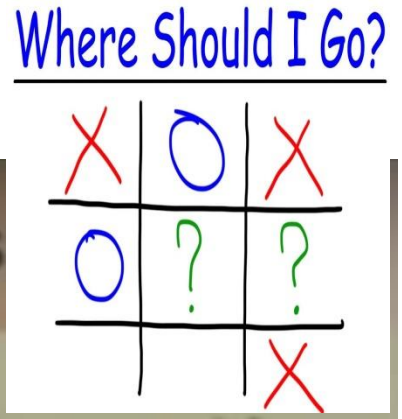
Agents interact with environments through sensors and actuators.

- An **agent is anything that can be viewed as perceiving its environment through sensors and** acting upon that environment through **actuators.**
- A human agent has eyes, ears, and other organs for sensors and hands, legs, vocal tract, and soon for actuators.
- A robotic agent might have cameras and infrared range finders for sensors and various motors for actuators.
- A software agent receives keystrokes, file contents, and network packets as sensory inputs and acts on the environment by displaying on the screen, writing files, and sending network packets.

Structure of an agent

- The job of AI is to design an **agent program that implements the agent function** the mapping from percepts to actions.
- *agent = architecture + program .*
- Types of agent

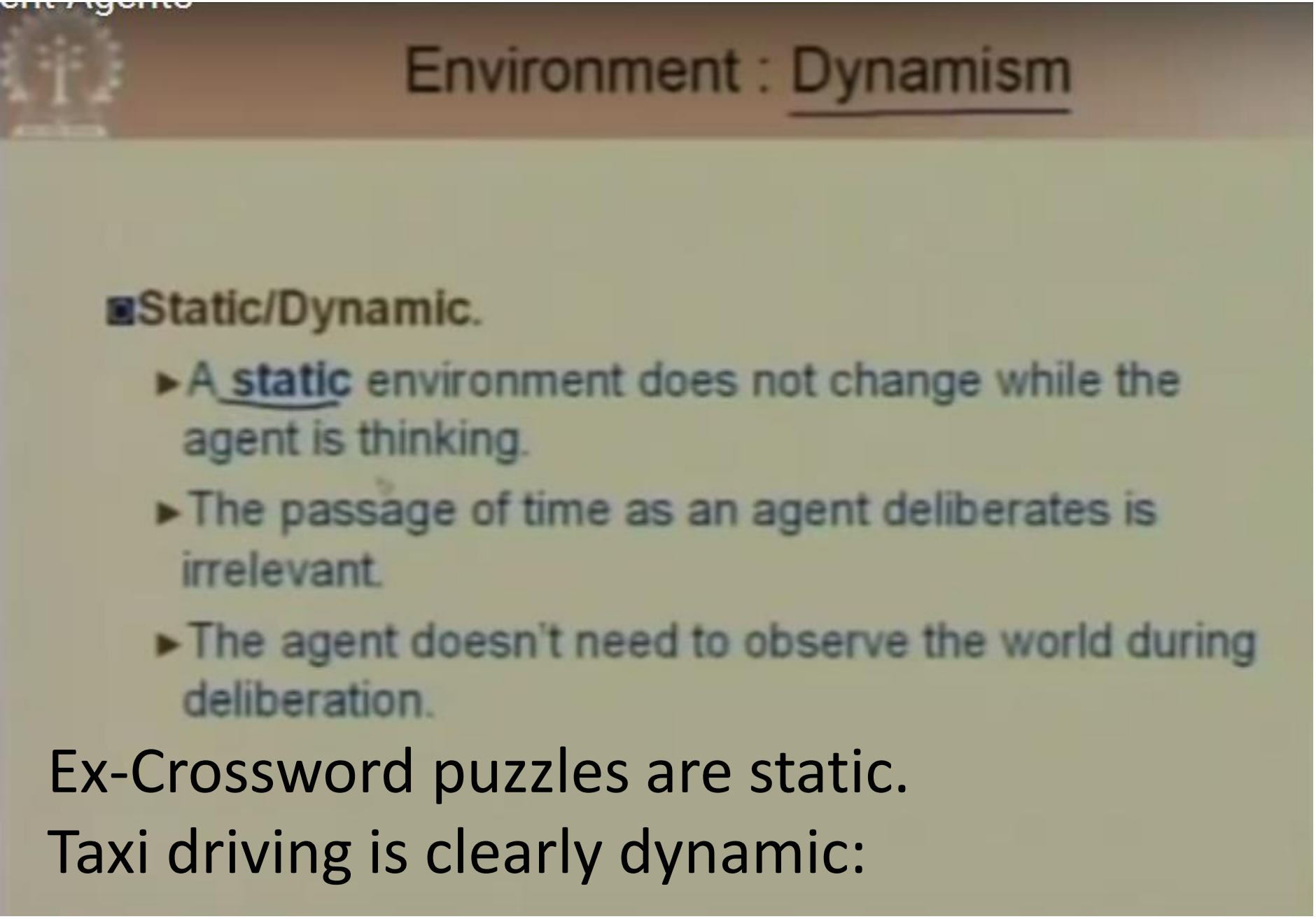
- Simple reflex agents;
- Model-based reflex agents;
- Goal-based agents; and
- Utility-based agents.



Environment: Determinis

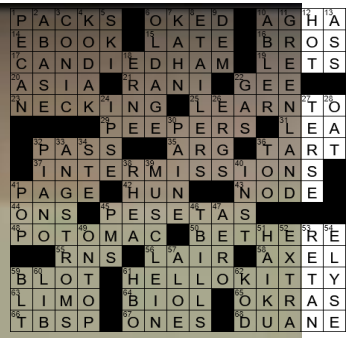
- Deterministic: The next state of the environment is completely described by the current state and the agent's action. Image analysis
- Stochastic: If an element of interference or uncertainty occurs then the environment is stochastic. Note that a deterministic yet partially observable environment will *appear* to be stochastic to the agent. Ludo

Strategic: environment state wholly determined by the preceding state and the actions of *multiple* agents is called strategic. Chess



Ex-Crossword puzzles are static.

Taxi driving is clearly dynamic:



Environments: other agents

- **Single agent/Multi-agent.**
  - If the environment contains other intelligent agents, the agent needs to be concerned about strategic, game-theoretic aspects of the environment (for either cooperative *or* competitive agents)
  - Most engineering environments don't have multi-agent properties, whereas most social and economic systems get their complexity from the interactions of (more or less) rational agents.

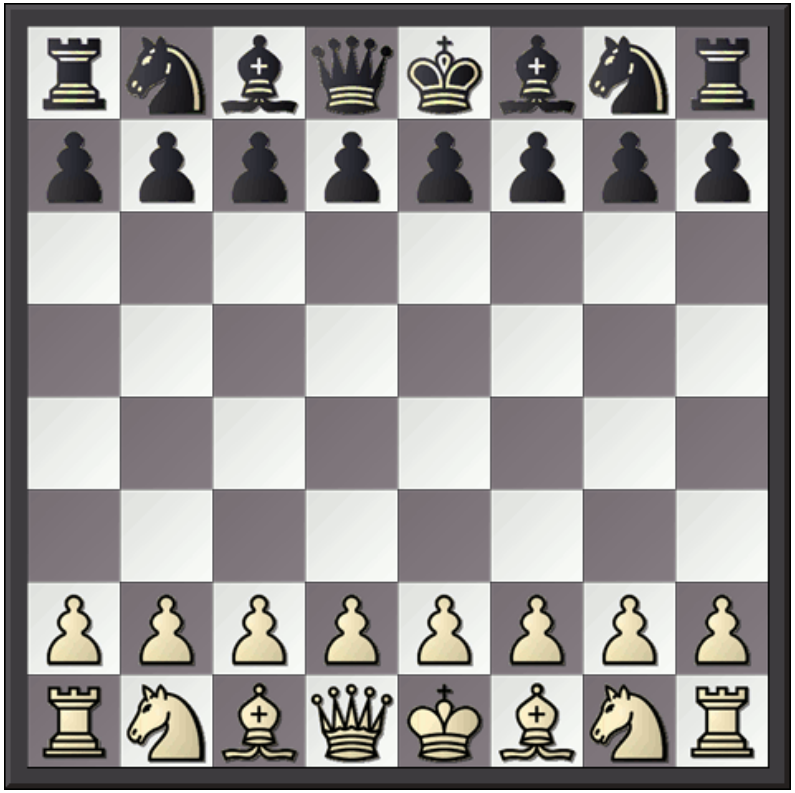
Ex-crossword puzzle (Single agent **)**

Ex- Chess (Multi-agent)

Environmental observation

- **Fully observable** : All of the environment relevant to the action being considered is observable.
- If an agent's sensors give it access to the complete state of the environment at each point in time, then we say that the task environment is fully observable.

- For Ex- Chess
- : All of the environment relevant to the action being considered is partial observable.
- Ex-poker **Partially observable**
- **Un-observable**: If the agent has no sensors at all then the environment is **unobserv**able



Episodic vs. Sequential

- Episodic : In an episodic task environment, the agent's experience is divided into atomic episodes. In each episode the agent receives a percept and then performs a single action. Crucially, the next episode does not depend on the actions taken in previous episodes.
- For example, an agent that has to spot defective parts on an assembly line bases each decision on the current part, regardless of previous decisions; moreover, the current decision doesn't affect whether the next part is defective.
- In sequential environments, on the other hand, the current decision could affect all future decisions.
- Chess and taxi driving are sequential: in both cases, short-term actions can have long-term consequences.
- Episodic environments are much simpler than sequential environments because the agent does not need to think ahead

Static vs. dynamic

- If the environment can change while an agent is deliberating, then we say the environment is dynamic for that agent; otherwise, it is static.
- Static environments are easy to deal with because the agent need not keep looking at the world while it is deciding on an action, nor need it worry about the passage of time.
- Dynamic environments, on the other hand, are continuously asking the agent what it wants to do; if it hasn’t decided yet, that counts as deciding to do nothing.
- Crossword puzzles are static. Taxi driving is clearly dynamic:

Discrete vs. continuous:

- The discrete/continuous distinction applies to the *state of the e*nvironment, to the way *time is handled, and to the percepts and actions of the agent.*
- *For* example, the chess environment has a finite number of distinct states (excluding the clock). Chess also has a discrete set of percepts and actions.
- Taxi driving is a continuous-state and continuous-time problem: the speed and location of the taxi and of the other vehicles sweep through a range of continuous values and do so smoothly over time.

Known vs. unknown:

- In a known environment, the outcomes (or outcome probabilities if the environment is stochastic) for all actions are given.
- Ex-Chess
- In a unknown environment, the outcomes (or outcome probabilities if the environment is stochastic) for all actions are not given.
- Ex-card game

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

Figure 2.5 Examples of agent types and their PEAS descriptions.

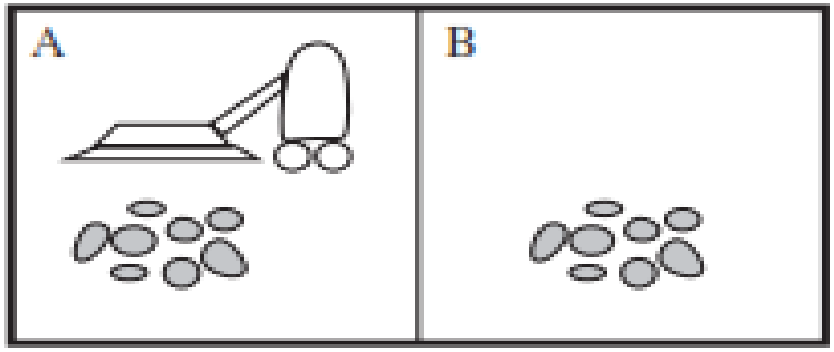


Figure 2.2 A vacuum-cleaner world with just two locations.

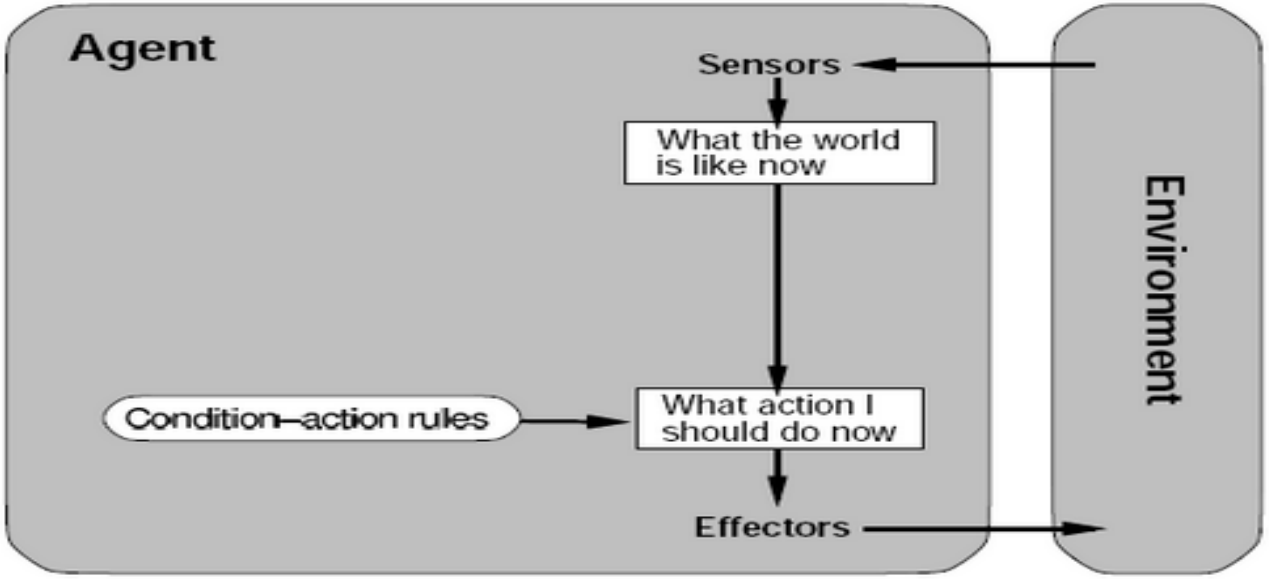
Percept sequence	Action
$[A, Clean]$	$Right$
$[A, Dirty]$	$Suck$
$[B, Clean]$	$Left$
$[B, Dirty]$	$Suck$
$[A, Clean], [A, Clean]$	$Right$
$[A, Clean], [A, Dirty]$	$Suck$
⋮	⋮
$[A, Clean], [A, Clean], [A, Clean]$	$Right$
$[A, Clean], [A, Clean], [A, Dirty]$	$Suck$
⋮	⋮

Figure 2.3 Partial tabulation of a simple agent function for the vacuum-cleaner world shown in Figure 2.2.

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

Figure 2.6 Examples of task environments and their characteristics.

Simple Reflex Agents



This agent selects actions based on the agents current perception or the world and not based on past perceptions.

For example if a mars lander found a rock in a specific place it needed to collect then it would collect it, if it was a simple reflex agent then if it found the same rock in a different place it would still pick it up as it doesn't take into account that it already picked it up.

This is useful for when a quick automated response is needed, humans have a very similar reaction to fire for example, our brain pulls our hand away without thinking about any possibility that there could be danger in the path of your arm. We call these reflex actions.

This kind of connection where only one possibility is acted upon is called a condition-action rule, written as:

if hand is in fire **then** pull away hand

The simple reflex agent has a library of such rules so that if a certain situation should arise and it is in the set of Condition-action rules the agent will know how to react with minimal reasoning.

These agents are simple to work with but have very limited intelligence, such as picking up 2 rock samples.

Model based reflex agents

Model-based reflex agents are made to deal with partial accessibility; they do this by keeping track of the part of the world it can see now. It does this by keeping an internal state that depends on what it has seen before so it holds information on the unobserved aspects of the current state.

This time out mars Lander after picking up its first sample, it stores this in the internal state of the world around it so when it come across the second same sample it passes it by and saves space for other samples.

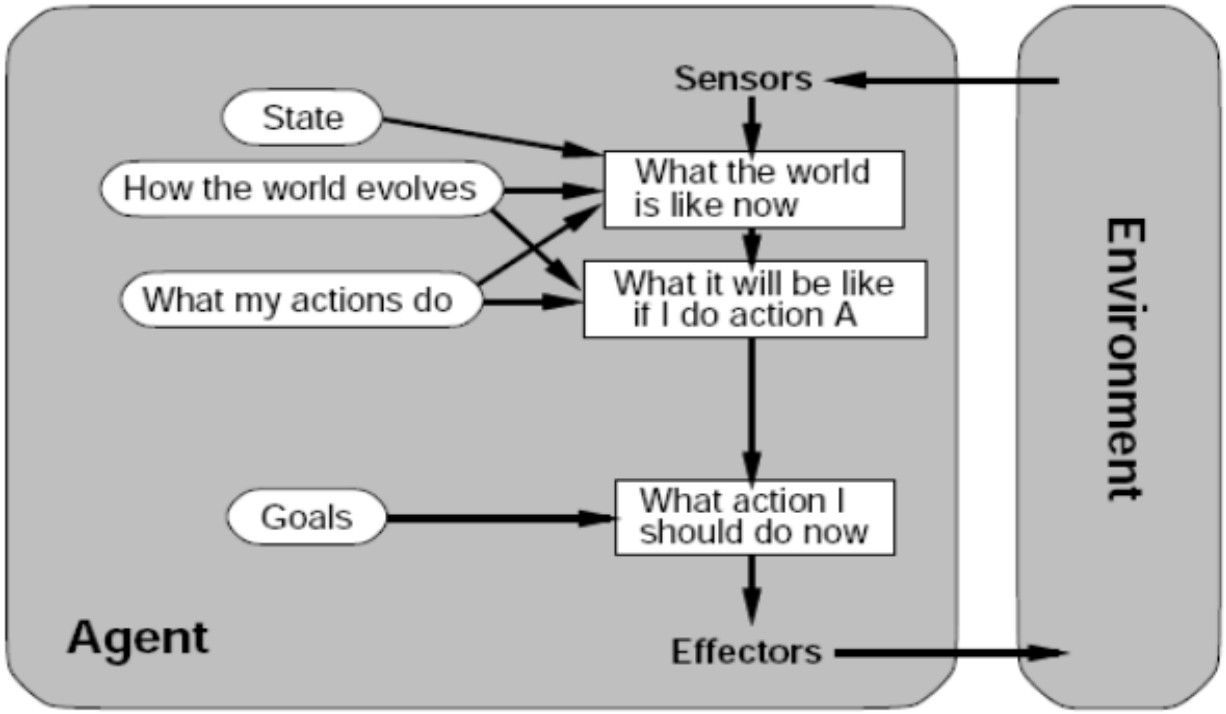
While reading this you are keeping track of where you have got to somewhere internally in your brain just in case you lose your place.

But in order to update this internal store we need 2 things:

1. Information on how the world evolves on its own.
   e.g. If our mars Lander picked up the rock next to the one it was going to the world around it would carry on as normal
2. How the world is affected by the agents actions.
   E.g. If our mars Lander took a sample under a precarious ledge it could displace a rock and it could be crushed.

We can predict how the world will react with facts like if you remove a supporting rock under a ledge the ledge will fall, such facts are called models, hence the name model-based agent.

Goal based agents



In life, in order to get things done we set goals for us to achieve, this pushes us to make the right decisions when we need to. A simple example would be the shopping list; our goal is to pick up every thing on that list. This makes it easier to decide if you need to choose between milk and orange juice because you can only afford one. As milk is a goal on our shopping list and the orange juice is not we chose the milk.

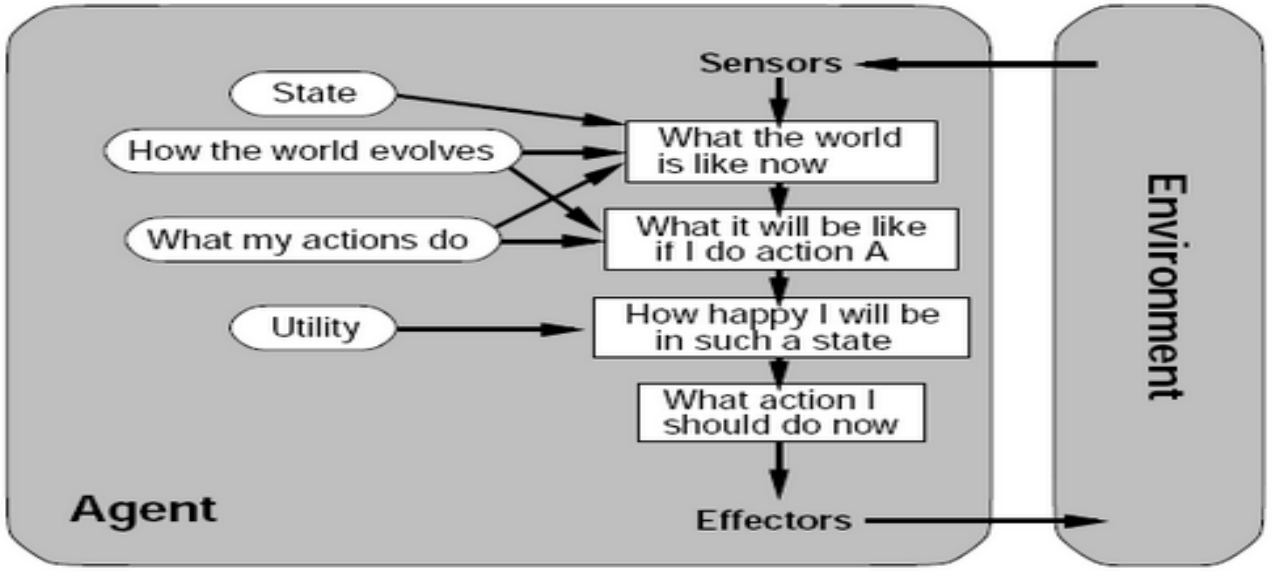
So in an intelligent agent having a set of goals with desirable situations are needed. The agent can use these goals with a set of actions and their predicted outcomes to see which action(s) achieve our goal(s).

Achieving the goals can take 1 action or many actions. Search and planning are two subfields in AI devoted to finding sequences of actions to achieve an agents goals.

Unlike the previous reflex agents before acting this agent reviews many actions and chooses the one which come closest to achieving its goals, whereas the reflex agents just have an automated response for certain situations.

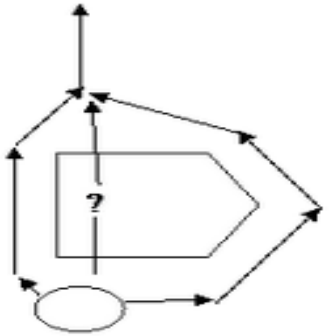
Although the goal-based agent does a lot more work that the reflex agent this makes it much more flexible because the knowledge used for decision making is is represented explicitly and can be modified. For example if our mars Lander needed to get up a hill the agent can update it's knowledge on how much power to put into the wheels to gain certain speeds, through this all relevant behaviors will now automatically follow the new knowledge on moving. However in a reflex agent many condition-action rules would have to be re-written.

Utility based agents



Just having goals isn't good enough because often we may have several actions which all satisfy our goal so we need some way of working out the most efficient one. A utility function maps each state after each action to a real number representing how efficiently each action achieves the goal. This is useful when we either have many actions all solving the same goal or when we have many goals that can be satisfied and we need to choose an action to perform.

For example let's show our mars Lander on the surface of mars with an obstacle in its way. In a goal based agent it is uncertain which path will be taken by the agent and some a re clearly not as efficient as others but in a utility based agent the best path will have the best output from the utility function and that path will be chosen.



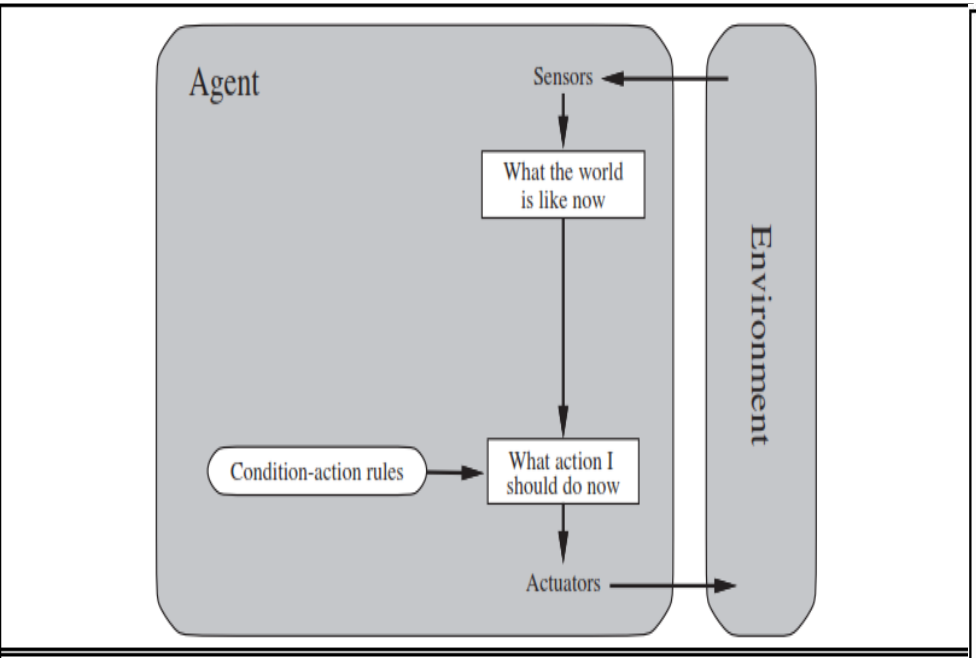


Figure 2.9 Schematic diagram of a simple reflex agent.

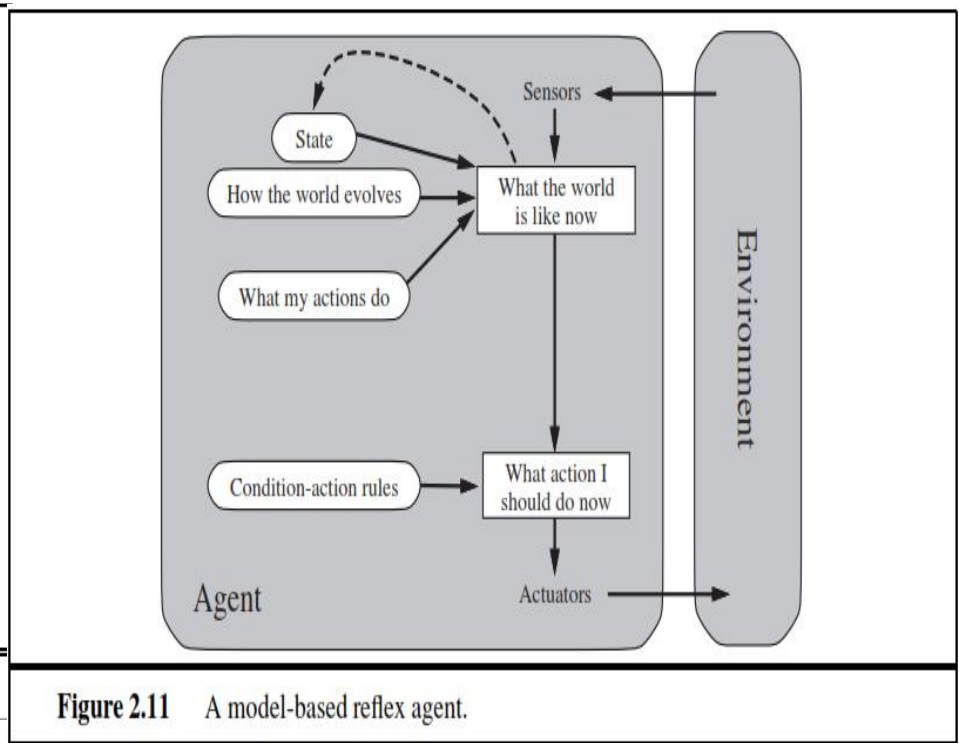
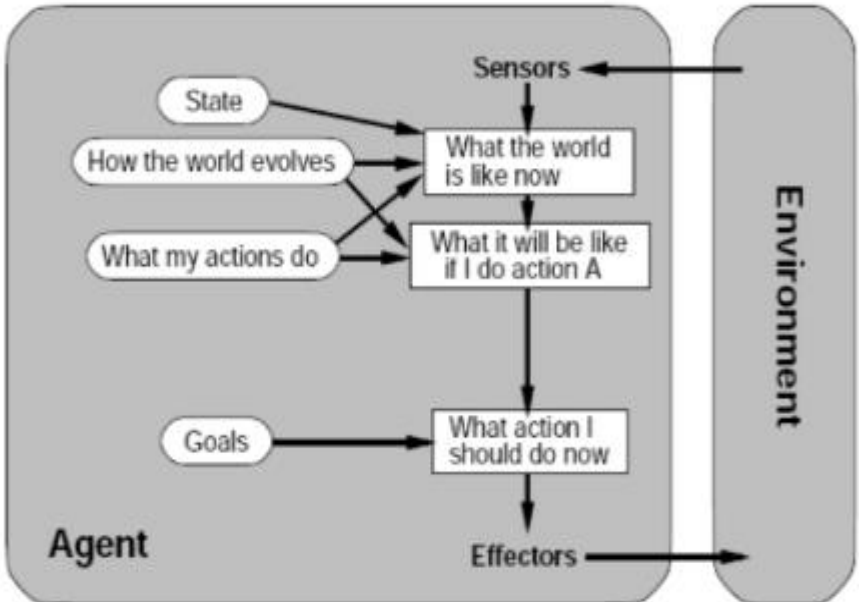
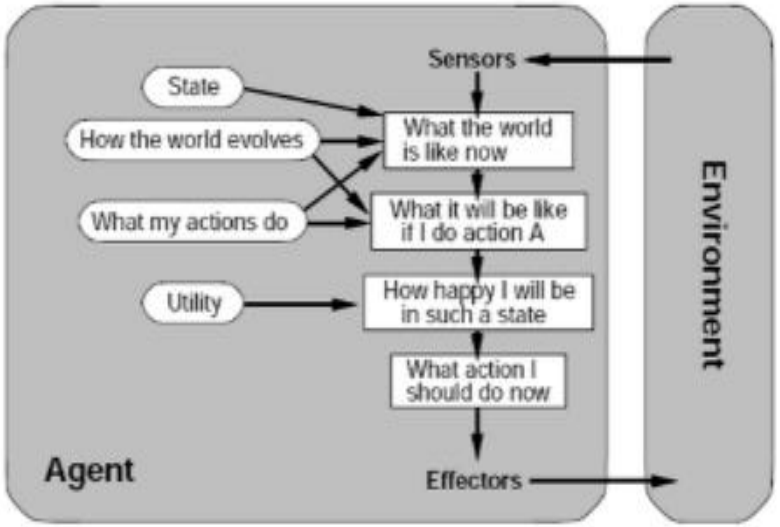


Figure 2.11 A model-based reflex agent.

oal based agents



Utility based agents



Simple Reflex agents

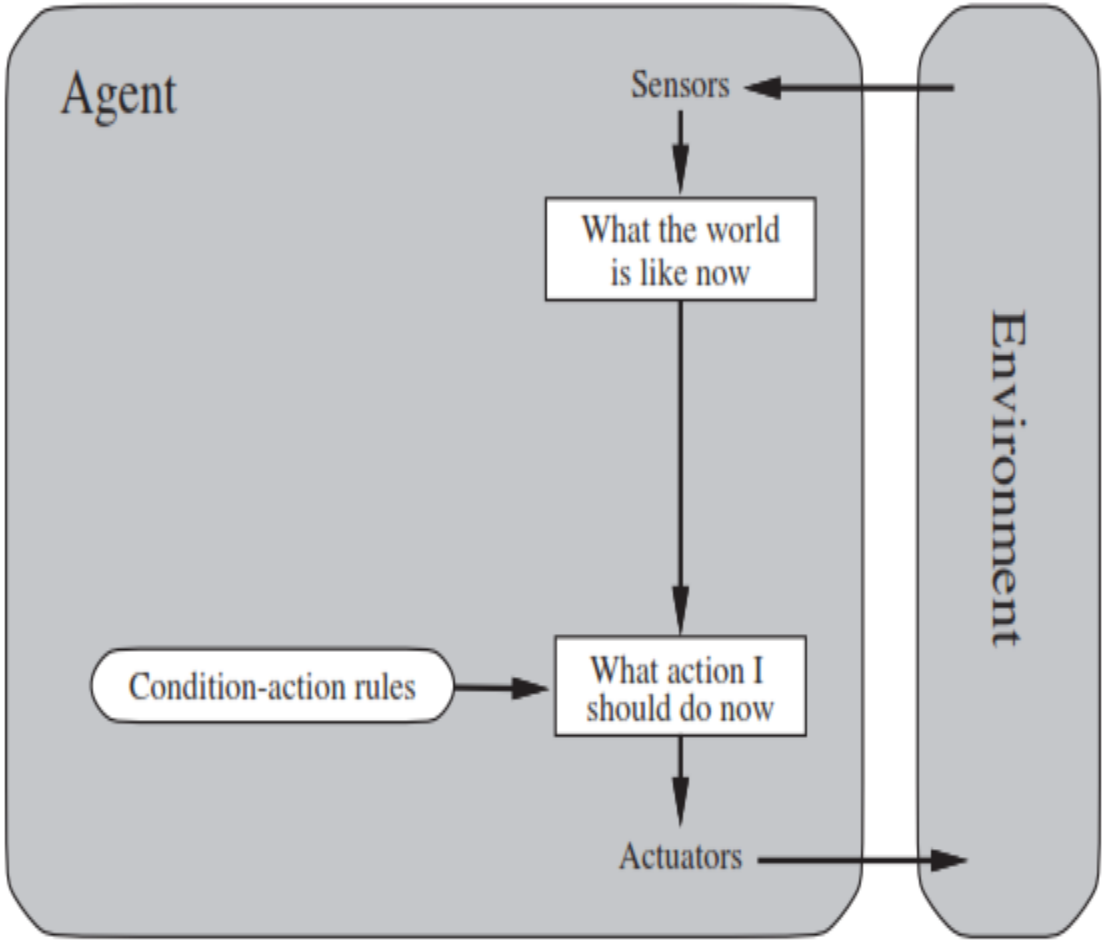


Figure 2.9 Schematic diagram of a simple reflex agent.

- These agents select actions on the basis of the *current percept, ignoring the rest of the percept history.*
- **if *car-in-front-is-braking then initiate-braking.***
- Imagine yourself as the driver of the automated taxi. If the car in front brakes and its brake lights come on, then you should notice this and initiate braking.
- Ex – alarm clock

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

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

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

Model-based reflex agents

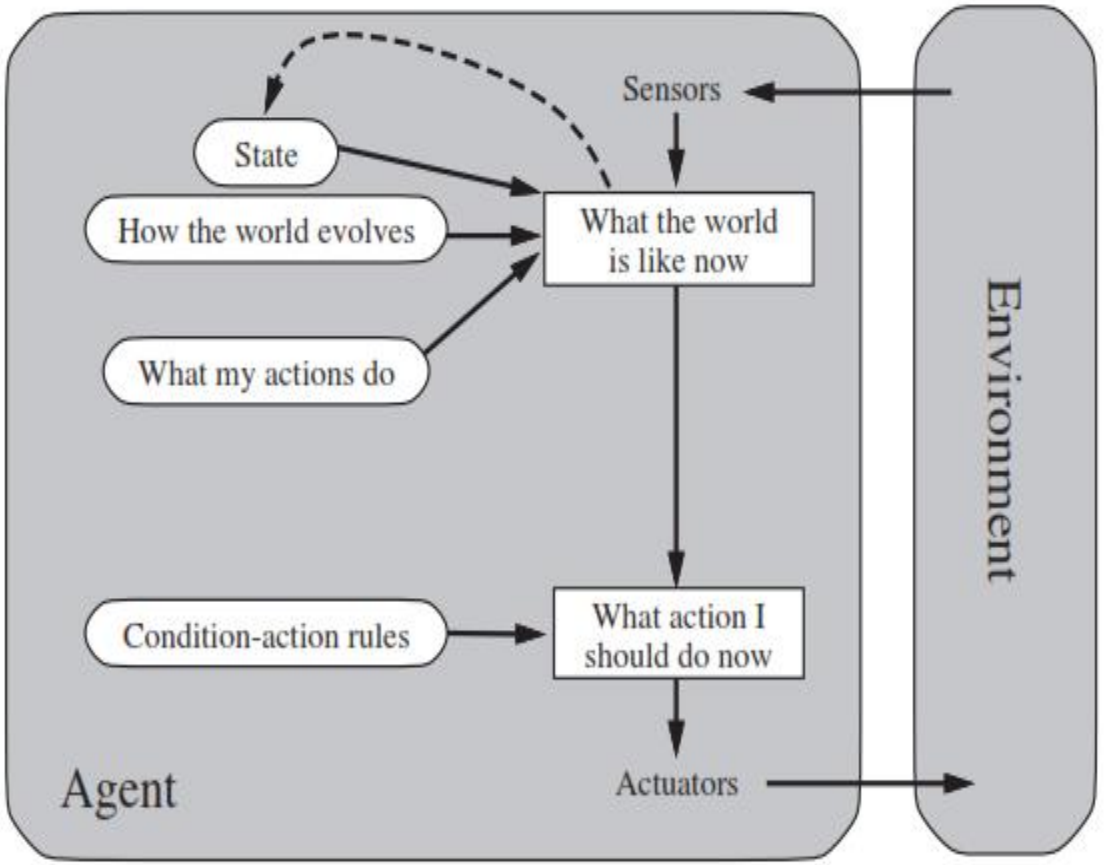


Figure 2.11 A model-based reflex agent.

- The most effective way to handle partial observability is for the agent to *keep track of the part of the world it can't see now.*
- For the braking problem, the internal state is not too extensive—just the previous frame from the camera, allowing the agent to detect when two red lights at
- the edge of the vehicle go on or off simultaneously. For other driving tasks such as changing
- lanes, the agent needs to keep track of where the other cars are if it can't see them all at once.
- And for any driving to be possible at all, the agent needs t
- Ex-Telephone operator/answering machine

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

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

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

Goal-based agents

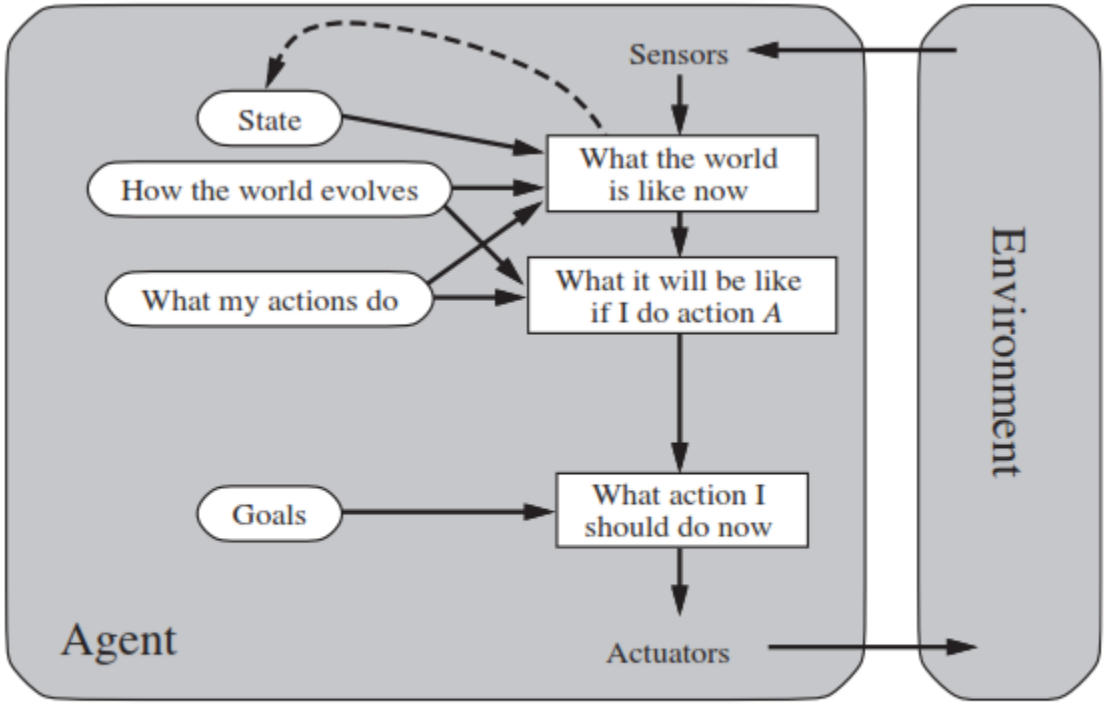


Figure 2.13 A model-based, goal-based agent. It keeps track of the world state as well as a set of goals it is trying to achieve, and chooses an action that will (eventually) lead to the achievement of its goals.

- In other words, as well as a current state description, the agent needs some sort of **goal information that describes** GOAL situations that are desirable.
- For example, at a road junction, the taxi can turn left, turn right, or go straight on.
- Ex-GPS system finding path to certain destination.

Utility-based agents

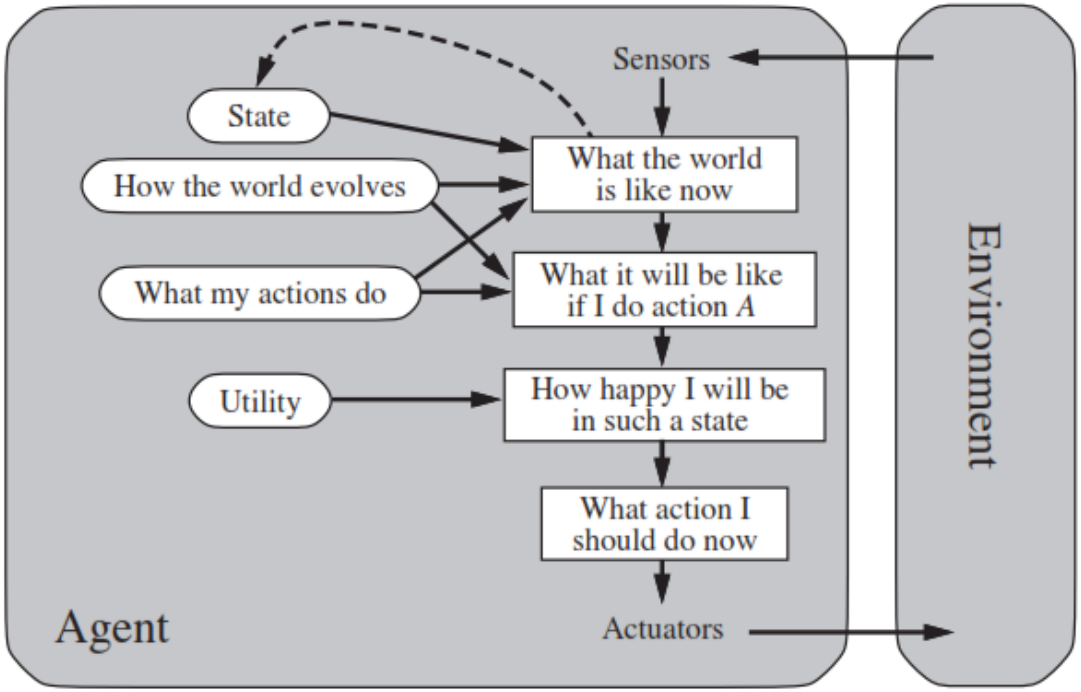


Figure 2.14 A model-based, utility-based agent. It uses a model of the world, along with a utility function that measures its preferences among states of the world. Then it chooses the action that leads to the best expected utility, where expected utility is computed by averaging over all possible outcome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

- Goals alone are not enough to generate high-quality behavior in most environments.
- For example, many action sequences will get the taxi to its destination (thereby achieving the goal) but some are quicker, safer, more reliable, or cheaper than others. Goals just provide a crude binary distinction between “happy” and “unhappy” states.
- An agent’s **utility function is essentially an internalization**

  UTILITY FUNCTION of the performance measure.

  If the internal utility function and the external performance

  measure are in agreement, then an agent that chooses actions to maximize its utility will be rational according to the external performance measure.

Ex-GPS system finding shortest /safer /fastest path to certain destination.

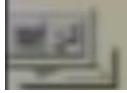
Goal directed Agent

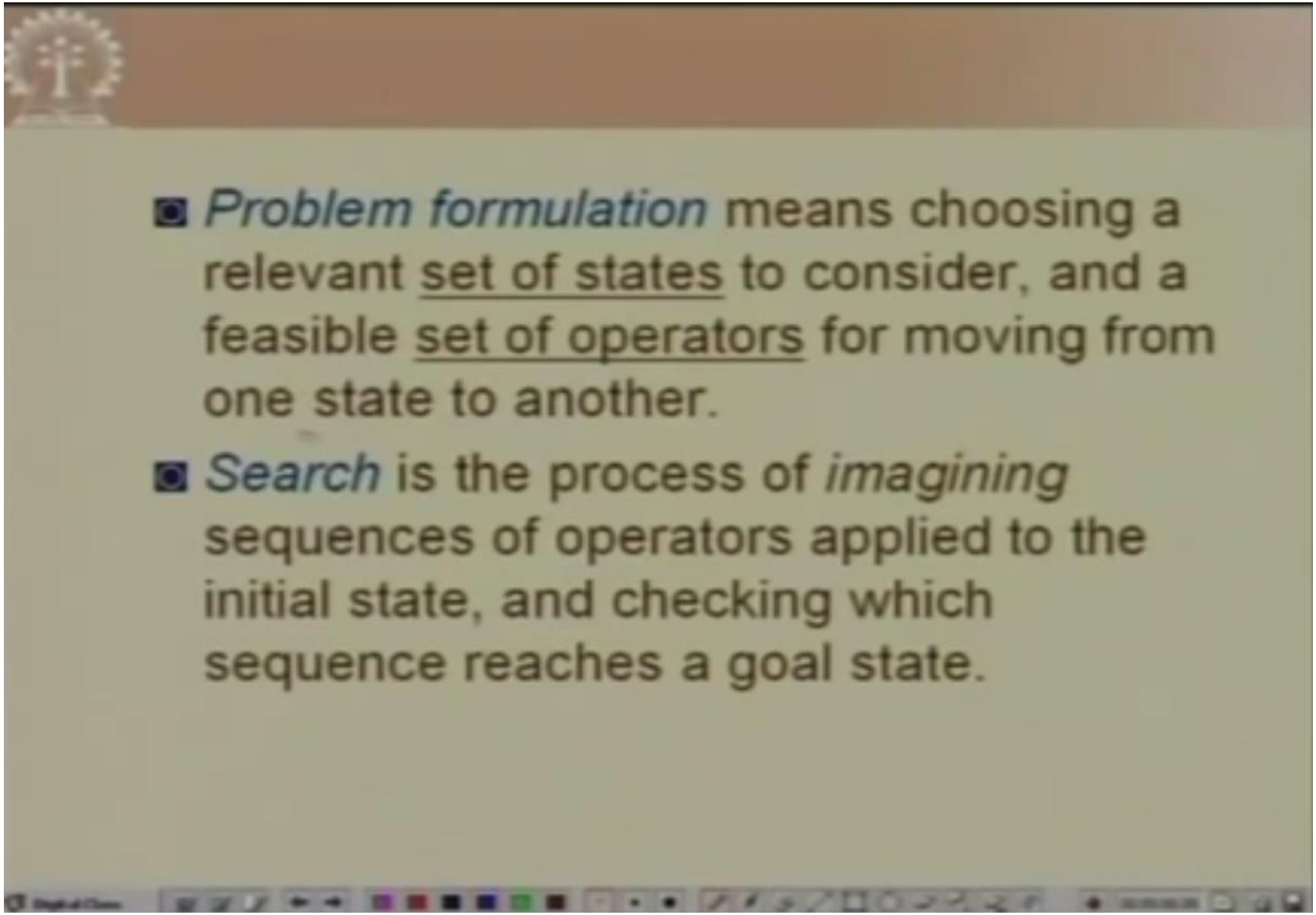
- A goal directed agent needs to achieve certain goals.
- Many problems can be represented as a set of states and a set of rules of how one state is transformed to another
- The agent must choose a sequence of actions to achieve the desired goal.

Each state is an abstract representation of the agent's environment. It is an abstraction that denotes a configuration of the agent.

- Initial state : The description of the starting configuration of the agent
- An action/ operator takes the agent from one state to another state. A state can have a number of successor states.
- A *plan* is a sequence of actions.

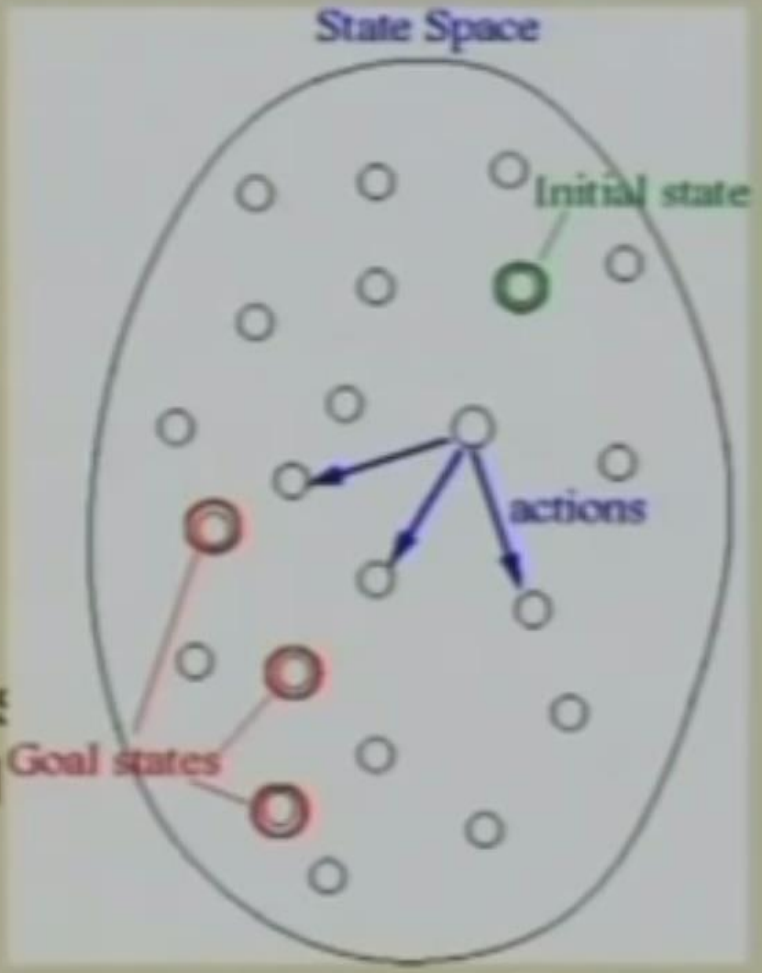
- A *goal* is a description of a set of desirable states of the world. Goal states are often specified by a goal test which any goal state must satisfy.
- Path cost : path ⟶ positive number
  Usually path cost = sum of step costs




Problem formulation means choosing a relevant set of states to consider, and a feasible set of operators for moving from one state to another.
Search is the process of imagining sequences of operators applied to the initial state, and checking which sequence reaches a goal state.

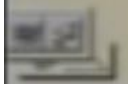
Search problem

- S: the full set of states
- s_0 : the initial state
- A:S→S set of operators
- G : the set of final states. $G \subseteq S$
- Search problem : Find a sequence of actions which transforms the agent from the initial state to a goal state $g \in G$.



Search problem

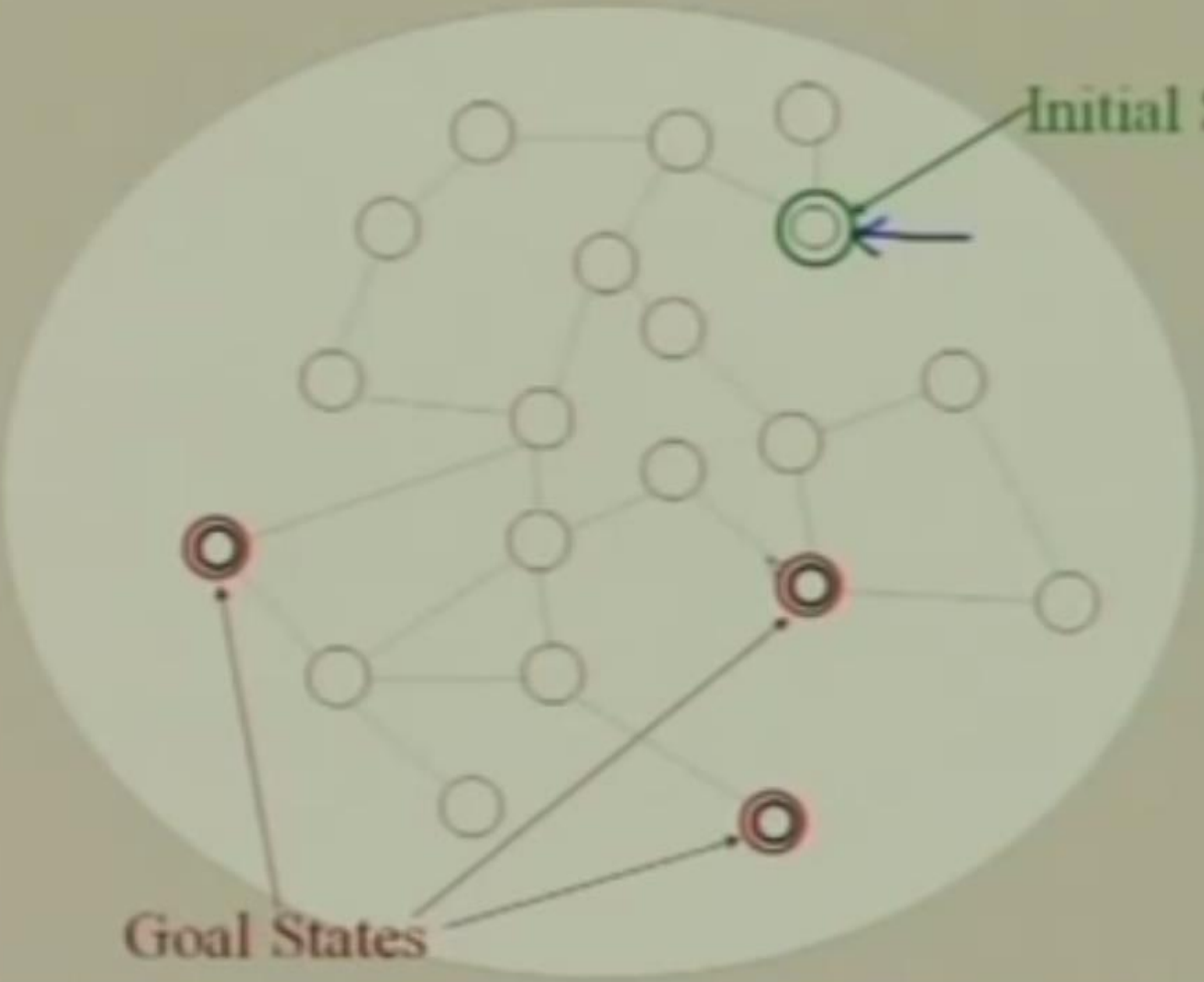
- The search problem consists of finding a solution plan, which is a path from the current state to the goal state.
- Representing search problems
  - A search problem is represented using a directed graph.
    - The states are represented as nodes.
    - The allowed actions are represented as arcs.



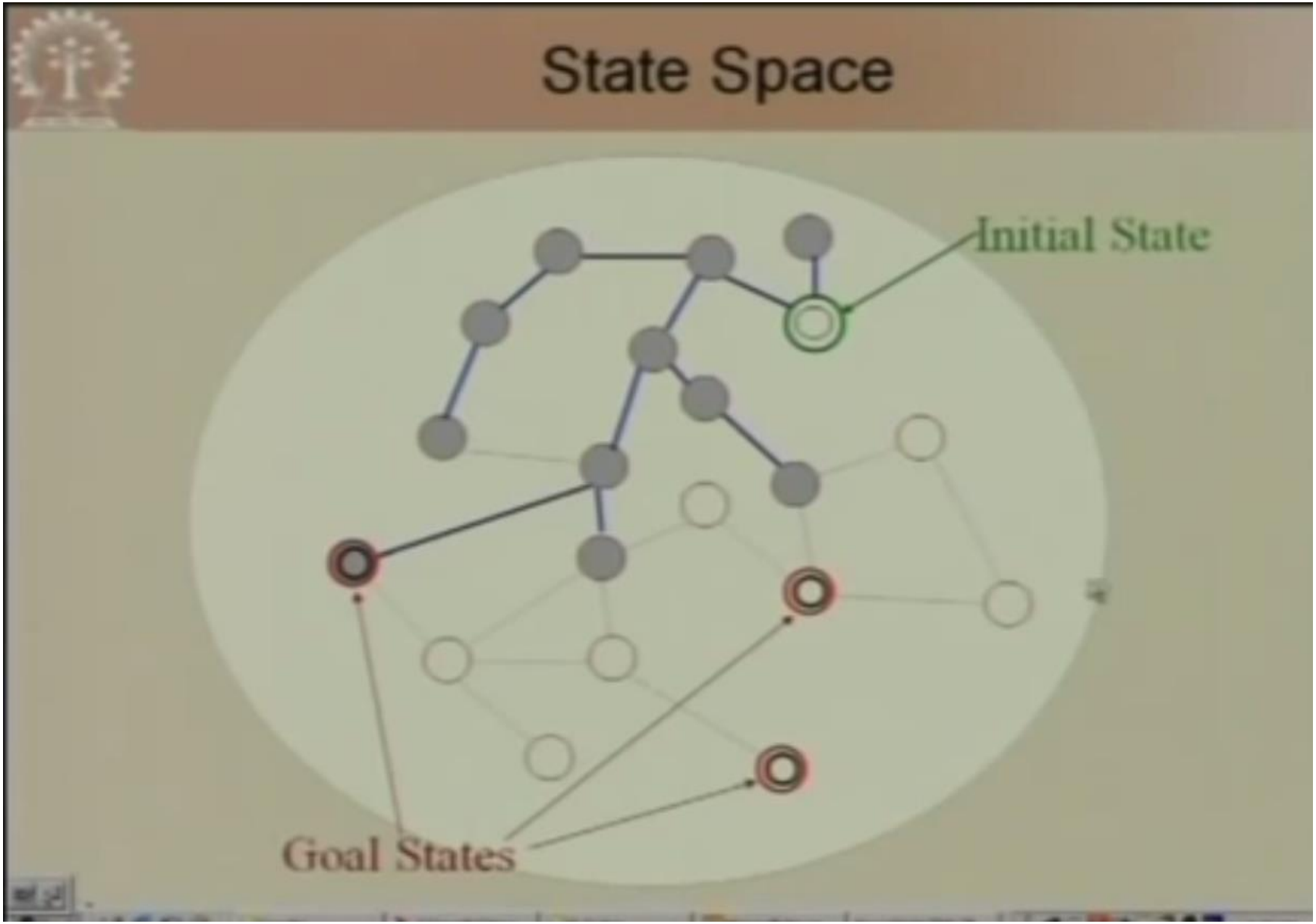
Searching process

- Check the current state
- Execute allowable actions to move to the next state
- Check if the new state is a solution state
  - If it is not, the new state becomes the current state and the process is repeated until a solution is found or the state space is exhausted.

State Space



State Space

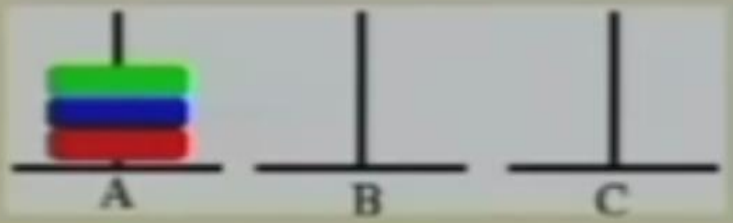


3 Peg problem

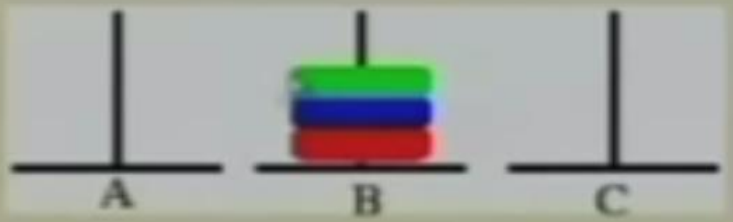
Pegs and Disks

- Consider the following problem. We have 3 pegs and 3 disks.
- Operators: one may move the topmost disk on any needle to the topmost position to any other needle

Initial state



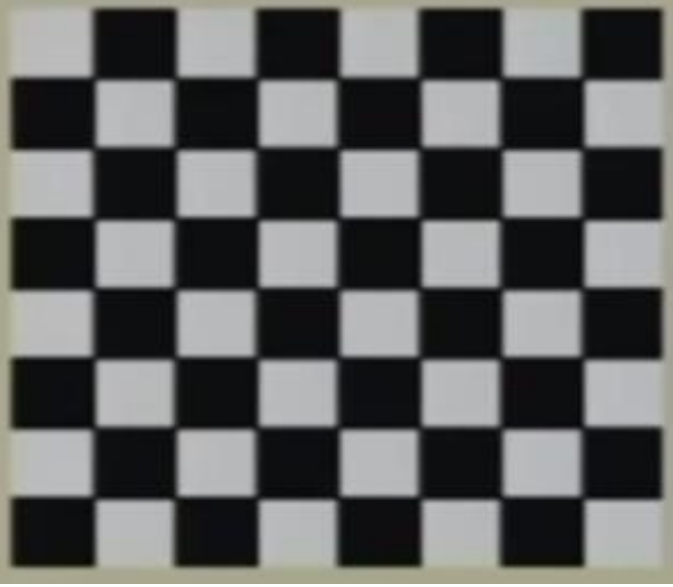
Goal configuration

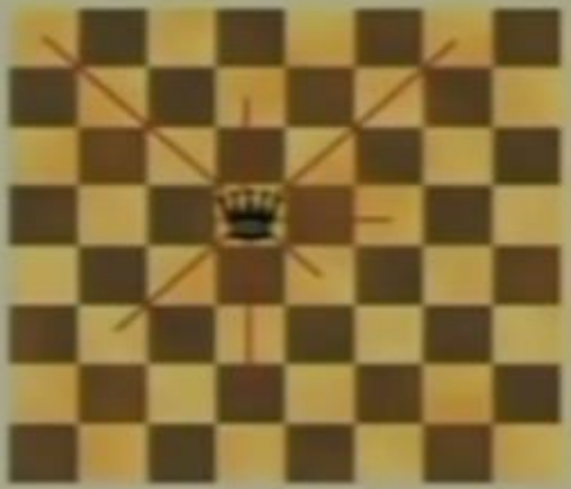


8 queen problem

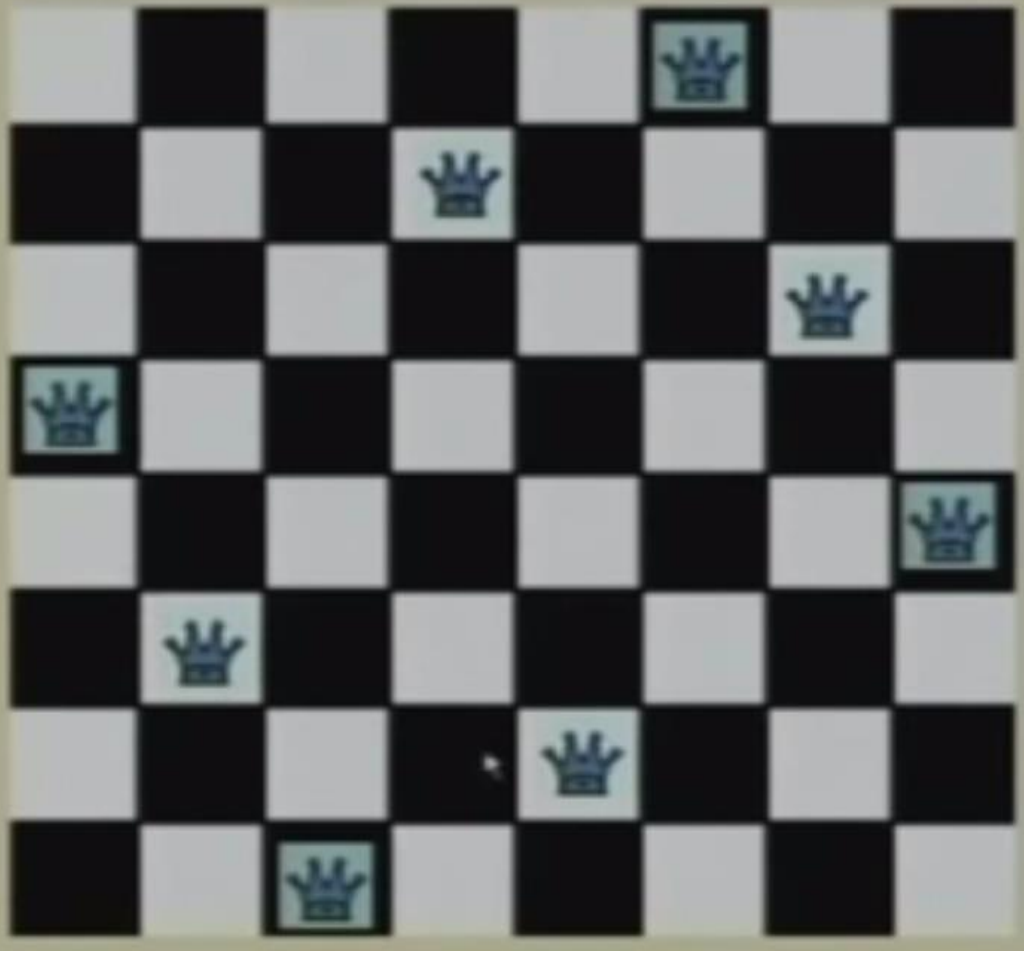
8 queens

- Place 8 queens on a chessboard so that no two queens are in the same row, column or diagonal

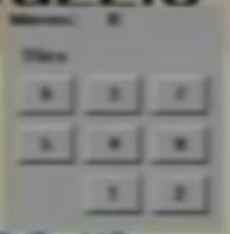




8 queens solution



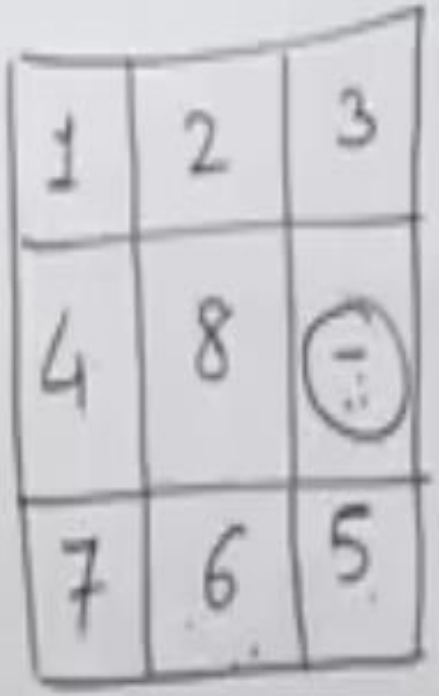
Problem Definition - Example, 8 puzzle



- States
  - A description of each of the eight tiles in each location that it can occupy.
- Operators/Action
  - The blank moves left, right, up or down
- Goal Test
  - The current state matches a certain state (e.g. one of the ones shown on previous slide)
- Path Cost
  - Each move of the blank costs 1

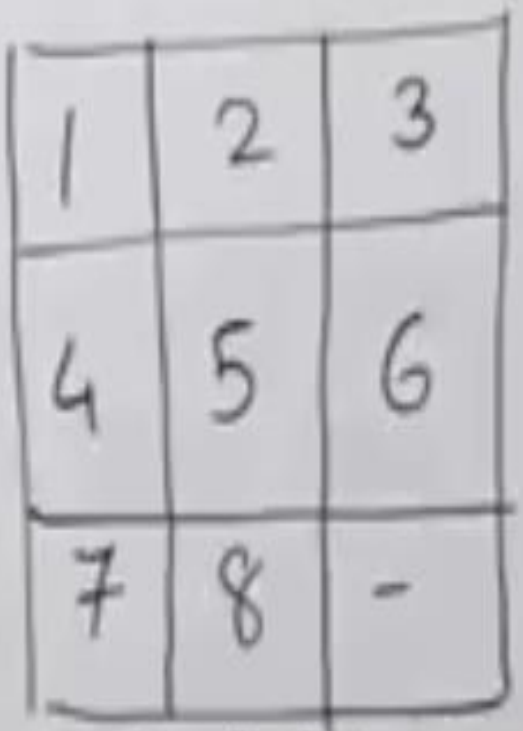
8-puzzle problem

"down"
"up"
"left"
"right"



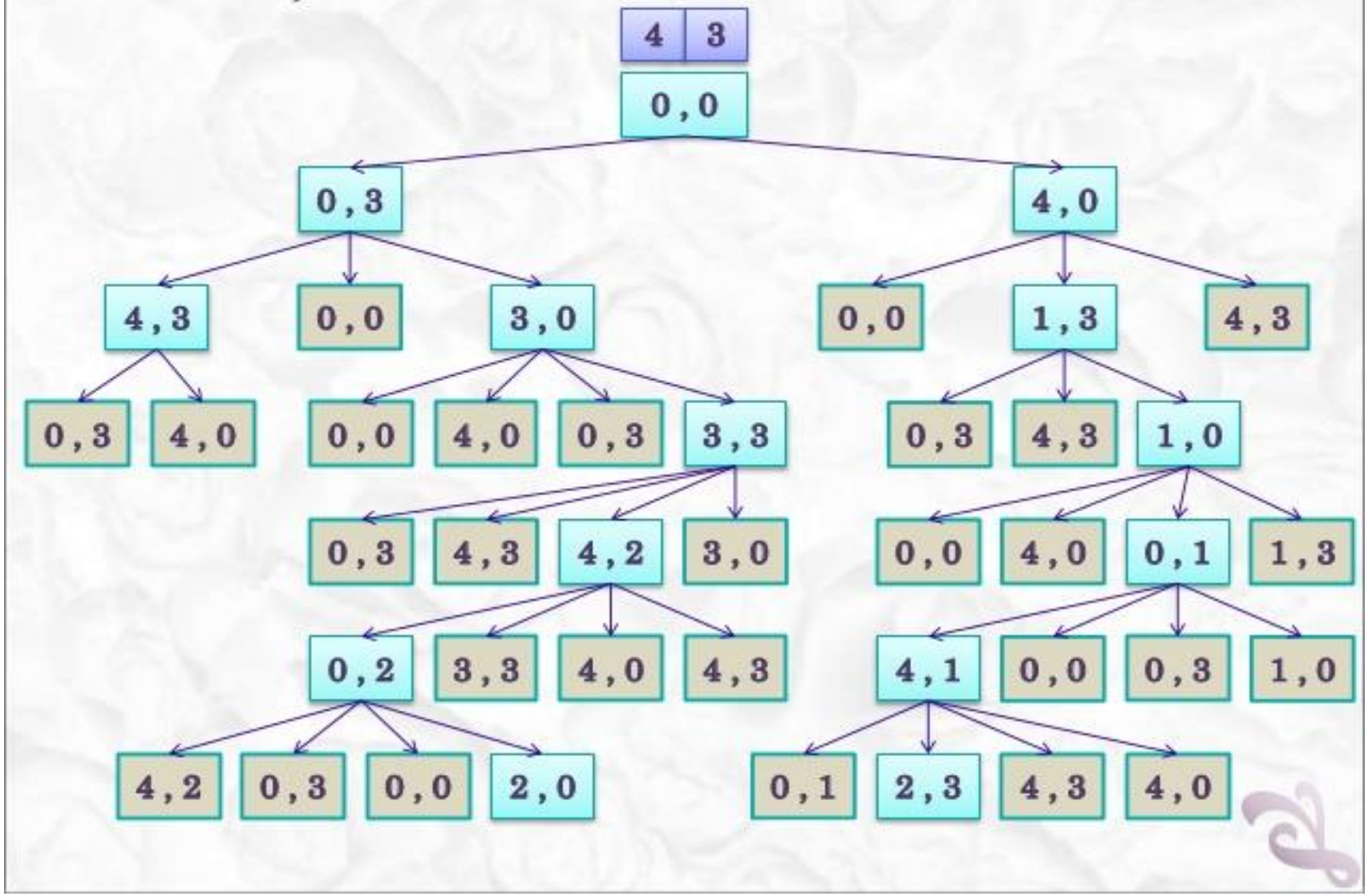
Start state

⇒



Goal state

THE WATER JUGS PROBLEM – SEARCH TREE

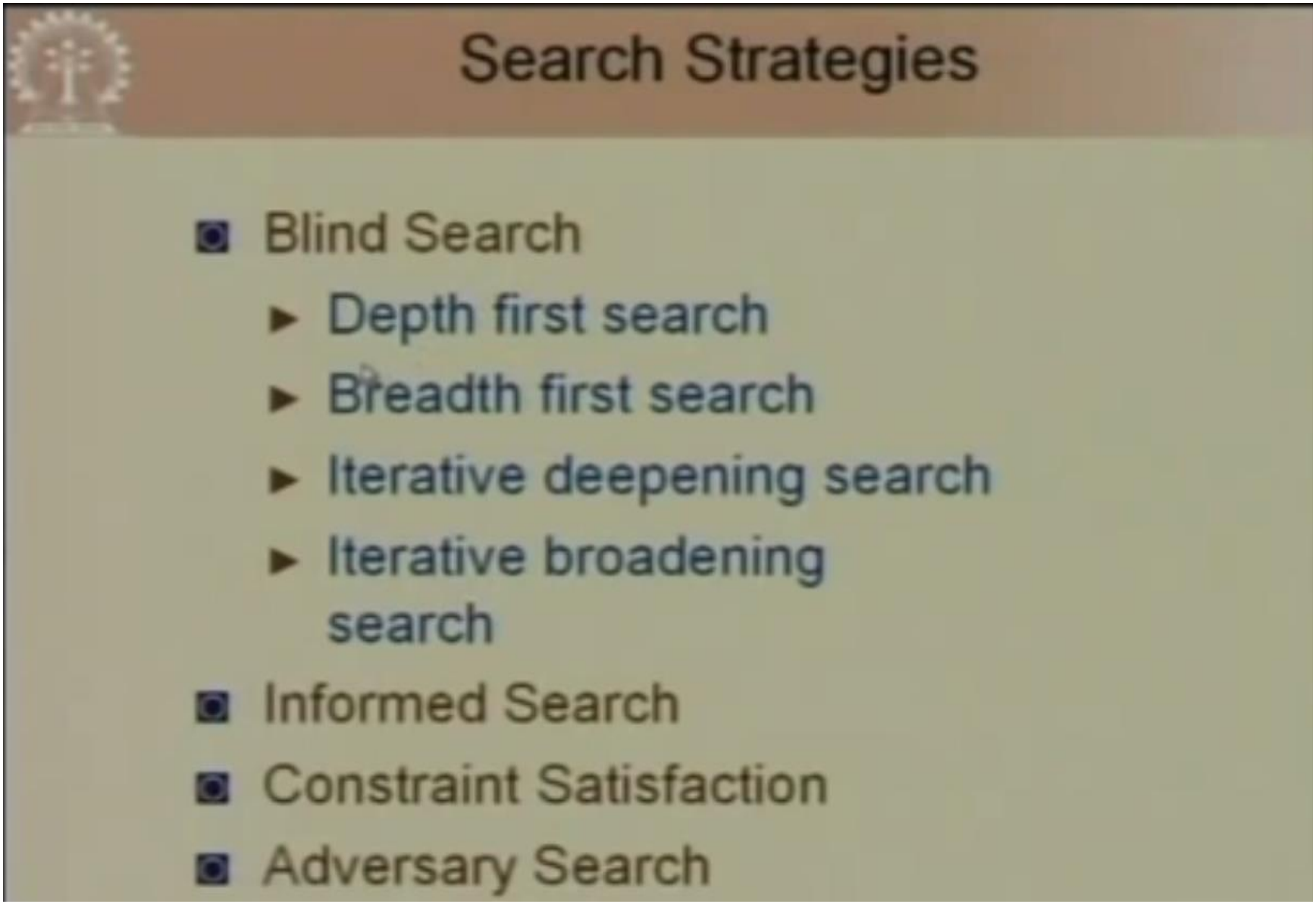


Search through a state space

- Input:
  - Set of states
  - Operators [and costs]
  - Start state
  - Goal state [test]
- Output:
  - Path: start ⇒ a state satisfying goal test
  - [May require shortest path]

Search strategy

- Measuring problem solving performance:
  - Completeness: Is the strategy guaranteed to find a solution if one exists ?
  - Optimality: Does the solution have low cost or the minimal cost ?
  - What is the search cost associated with the time and memory required to find a solution?


Search Strategies
Blind Search
Depth first search
Breadth first search
Iterative deepening search
Iterative broadening search
Informed Search
Constraint Satisfaction
Adversary Search

Search Techniques

- **Uninformed search** algorithms—algorithms that are given no information about the problem other than its definition.
- It is also known as blind or brute force search.
- Not so efficient.
- **Informed search algorithms**, on the other hand can do quite well given some guidance on where to look for solutions.

- Tree is a non-linear data structure which organizes data in hierarchical structure and this is a recursive definition.
- http://btechsmartclass.com/DS/U3_T1.html

Graph Representation

TUTORIAL PROBLEMS

Graphs are mathematical structures that represent pairwise relationships between objects. A graph is a flow structure that represents the relationship between various objects. It can be visualized by using the following two basic components:

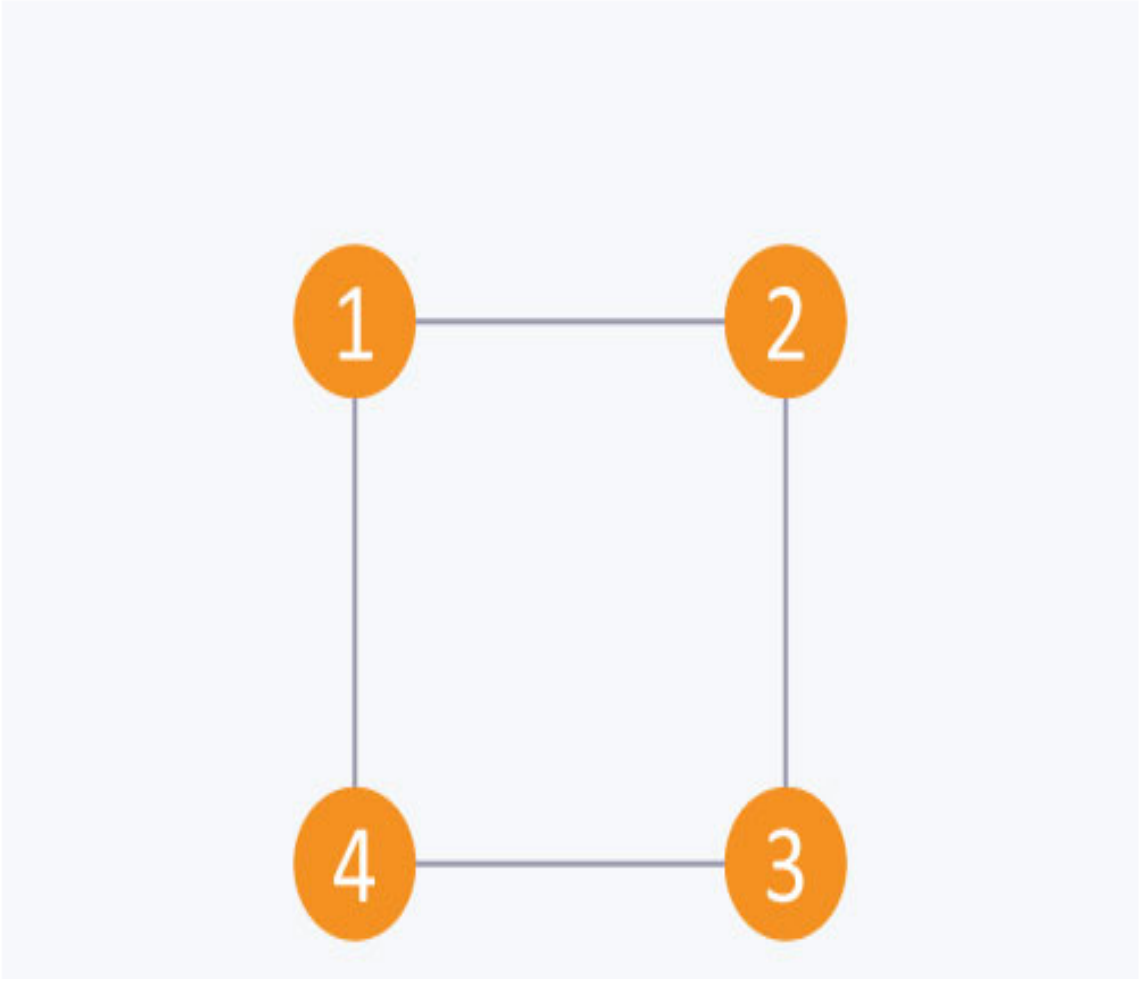
- Nodes: These are the most important components in any graph. Nodes are entities whose relationships are expressed using edges. If a graph comprises 2 nodes A and B and an undirected edge between them, then it expresses a bi-directional relationship between the nodes and edge.
- Edges: Edges are the components that are used to represent the relationships between various nodes in a graph. An edge between two nodes expresses a one-way or two-way relationship between the nodes.

Types of nodes

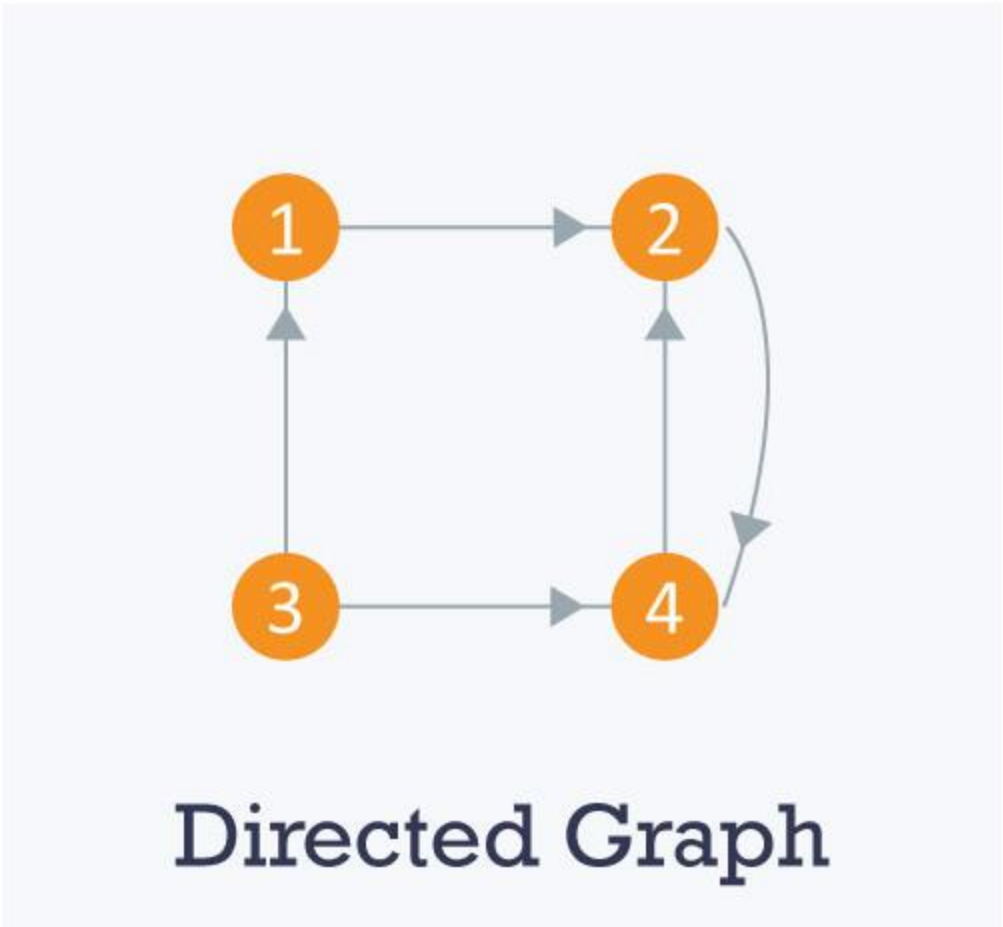
- Root node: The root node is the ancestor of all other nodes in a graph. It does not have any ancestor. Each graph consists of exactly one root node. Generally, you must start traversing a graph from the root node.
- Leaf nodes: In a graph, leaf nodes represent the nodes that do not have any successors. These nodes only have ancestor nodes. They can have any number of incoming edges but they will not have any outgoing edges.

Types of graphs

- Undirected: An undirected graph is a graph in which all the edges are bi-directional i.e. the edges do not point in any specific direction.

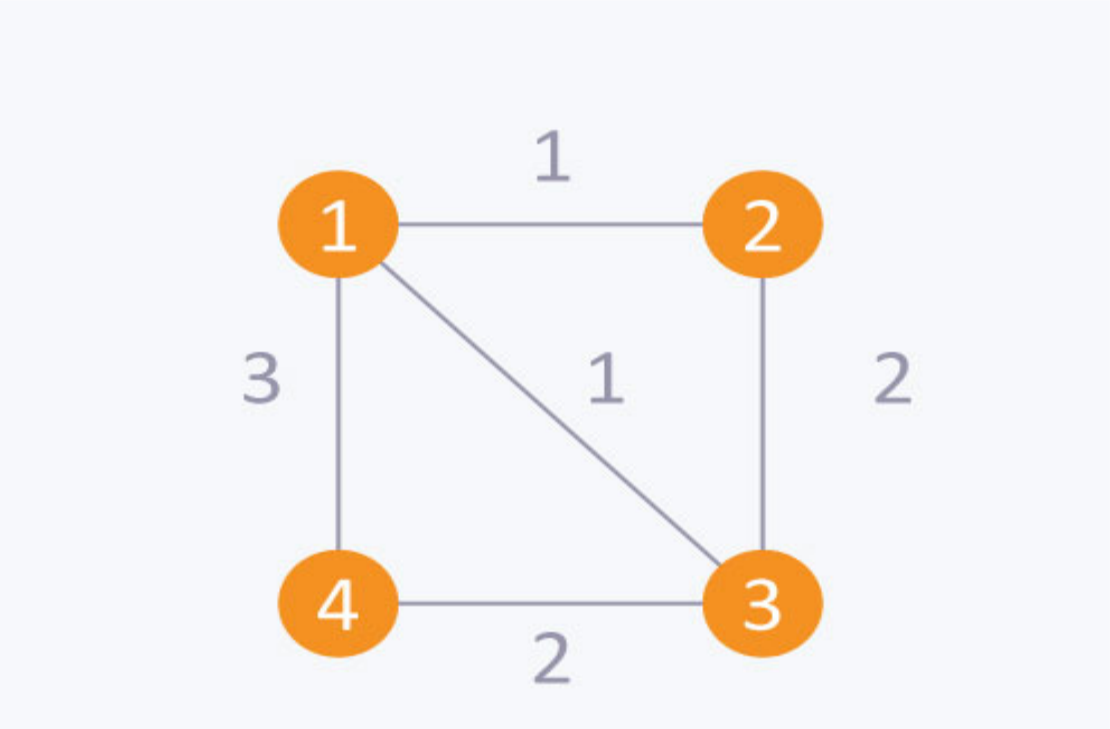


- Directed: A directed graph is a graph in which all the edges are uni-directional i.e. the edges point in a single direction.



- Weighted: In a weighted graph, each edge is assigned a weight or cost. Consider a graph of 4 nodes as in the diagram below. As you can see each edge has a weight/cost assigned to it. If you want to go from vertex 1 to vertex 3, you can take one of the following 3 paths:
    - 1 -> 2 -> 3
    - 1 -> 3
    - 1 -> 4 -> 3

  Therefore the total cost of each path will be as follows: - The total cost of 1 -> 2 -> 3 will be (1 + 2) i.e. 3 units - The total cost of 1 -> 3 will be 1 unit - The total cost of 1 -> 4 -> 3 will be (3 + 2) i.e. 5 units

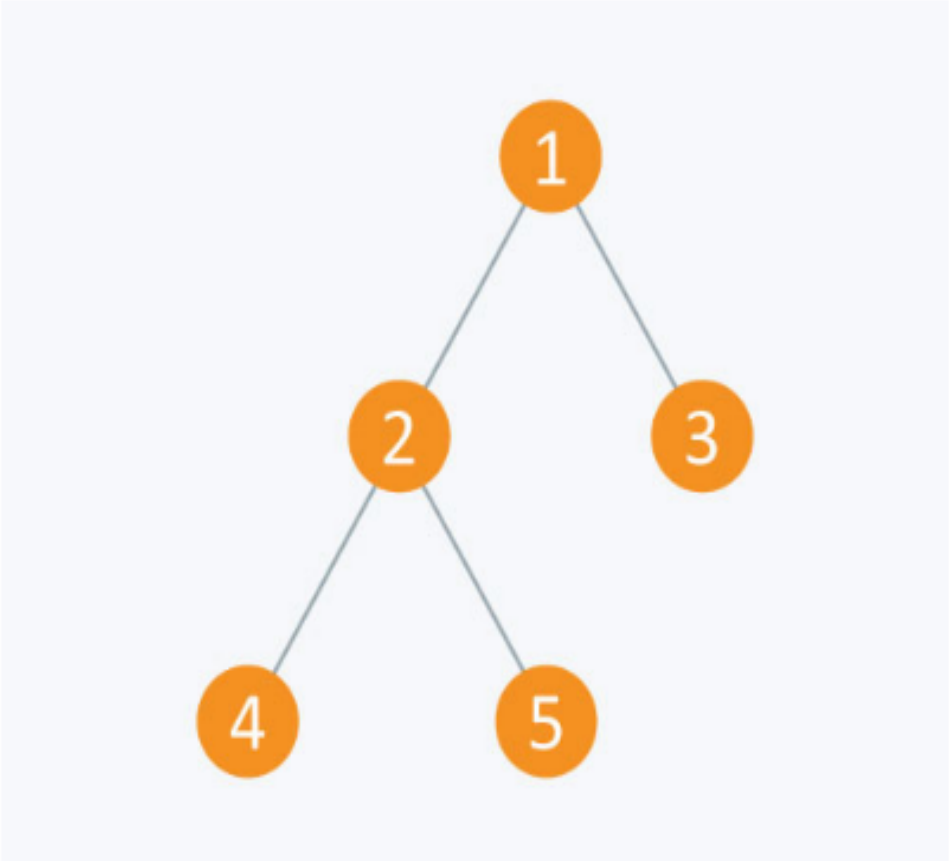


- Cyclic: A graph is cyclic if the graph comprises a path that starts from a vertex and ends at the same vertex. That path is called a cycle. An acyclic graph is a graph that has no cycle.

  A **tree** is an undirected graph in which any two vertices are connected by only one path. A tree is an acyclic graph and has N - 1 edges where N is the number of vertices. Each node in a graph may have one or multiple parent nodes. However, in a tree, each node (except the root node) comprises exactly one parent node.

  Note: A root node has no parent.

  A tree cannot contain any cycles or self loops, however, the same does not apply to graphs.



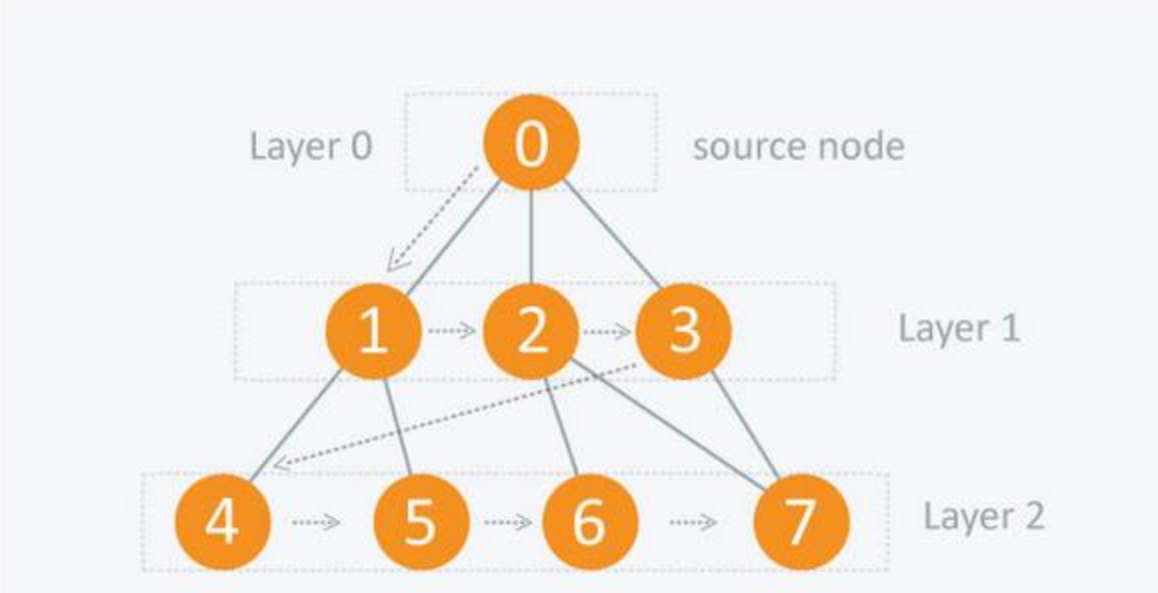
Breadth First Search (BFS)

There are many ways to traverse graphs. BFS is the most commonly used approach.

BFS is a traversing algorithm where you should start traversing from a selected node (source or starting node) and traverse the graph layerwise thus exploring the neighbour nodes (nodes which are directly connected to source node). You must then move towards the next-level neighbour nodes.

As the name BFS suggests, you are required to traverse the graph breadthwise as follows:

1. First move horizontally and visit all the nodes of the current layer
2. Move to the next layer



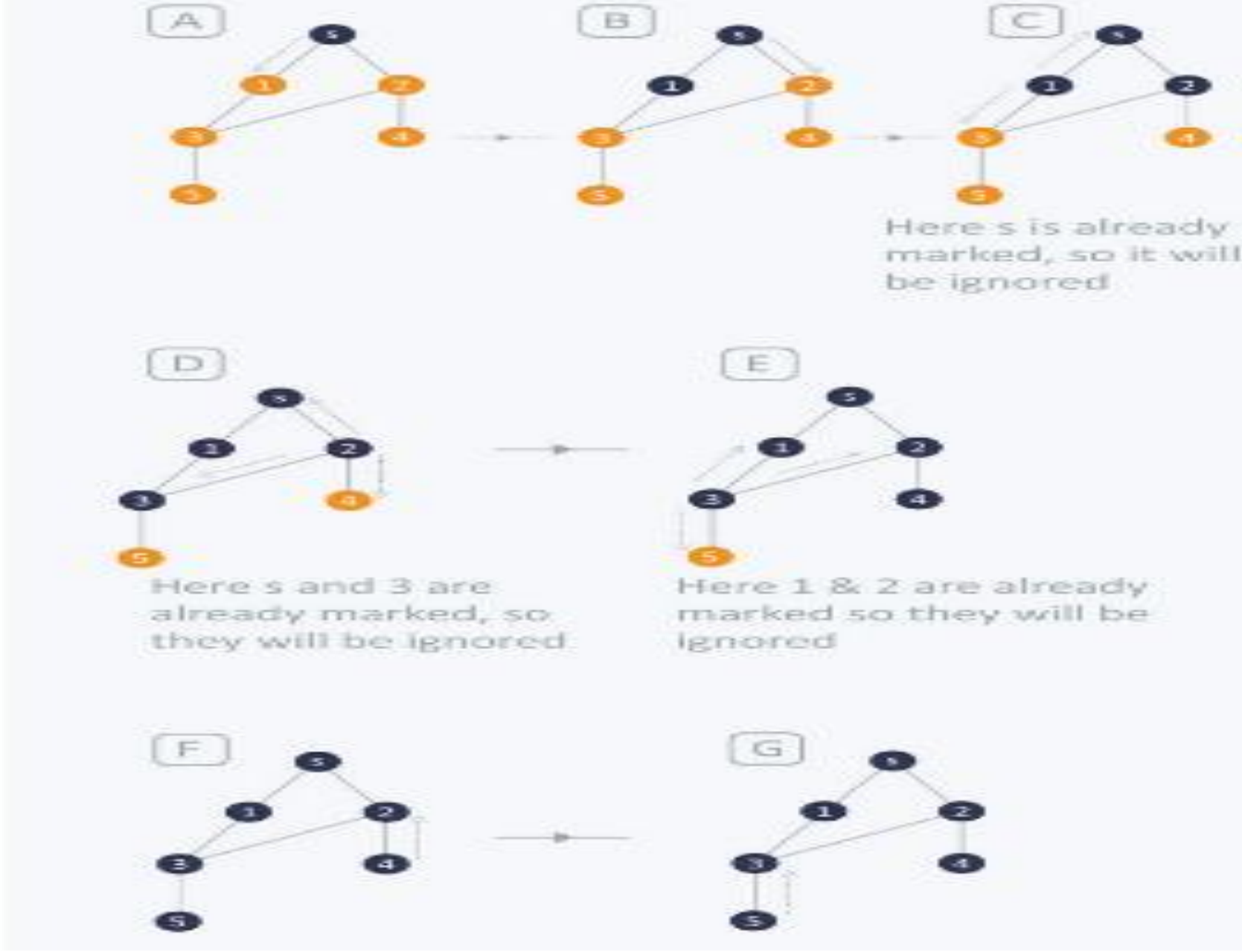
Traversing child nodes

A graph can contain cycles, which may bring you to the same node again while traversing the graph. To avoid processing of same node again, use a boolean array which marks the node after it is processed. While visiting the nodes in the layer of a graph, store them in a manner such that you can traverse the corresponding child nodes in a similar order.

In the earlier diagram, start traversing from 0 and visit its child nodes 1, 2, and 3. Store them in the order in which they are visited. This will allow you to visit the child nodes of 1 first (i.e. 4 and 5), then of 2 (i.e. 6 and 7), and then of 3 (i.e. 7) etc.

To make this process easy, use a queue to store the node and mark it as 'visited' until all its neighbours (vertices that are directly connected to it) are marked. The queue follows the First In First Out (FIFO) queuing method, and therefore, the neigbors of the node will be visited in the order in which they were inserted in the node i.e. the node that was inserted first will be visited first, and so on.

Traversing process



First iteration

- s will be popped from the queue
- Neighbors of s i.e. 1 and 2 will be traversed
- 1 and 2, which have not been traversed earlier, are traversed. They will be:
    - Pushed in the queue
    - 1 and 2 will be marked as visited

Second iteration

- 1 is popped from the queue
- Neighbors of 1 i.e. s and 3 are traversed
- s is ignored because it is marked as 'visited'
- 3, which has not been traversed earlier, is traversed. It is:
    - Pushed in the queue
    - Marked as visited

Third iteration

- 2 is popped from the queue
- Neighbors of 2 i.e. s, 3, and 4 are traversed
- 3 and s are ignored because they are marked as 'visited'
- 4, which has not been traversed earlier, is traversed. It is:
    - Pushed in the queue
    - Marked as visited

Fourth iteration

- 3 is popped from the queue
- Neighbors of 3 i.e. 1, 2, and 5 are traversed
- 1 and 2 are ignored because they are marked as 'visited'
- 5, which has not been traversed earlier, is traversed. It is:
    - Pushed in the queue
    - Marked as visited

Breadth First Search

- **Breadth-first search is a simple strategy in which the root node is expanded first, then all the** successors of the root node are expanded next, then *their successors, and so on.*
- It uses a FIFO queue.
- Time complexity=
- Space complexity=

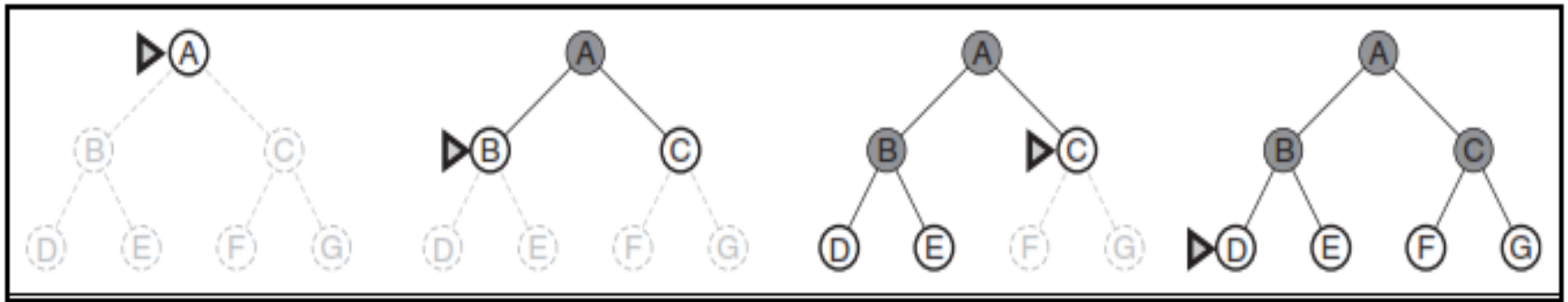


Figure 3.12 Breadth-first search on a simple binary tree. At each stage, the node to be expanded next is indicated by a marker.

Applications :

1)Social Networking Websites
2) In Garbage Collection
3) GPS Navigation systems
4) Localization (Path Finding We can either use Breadth First or Depth First Traversal to find if there is a path between two vertices.)

Advantage and Disadvantages

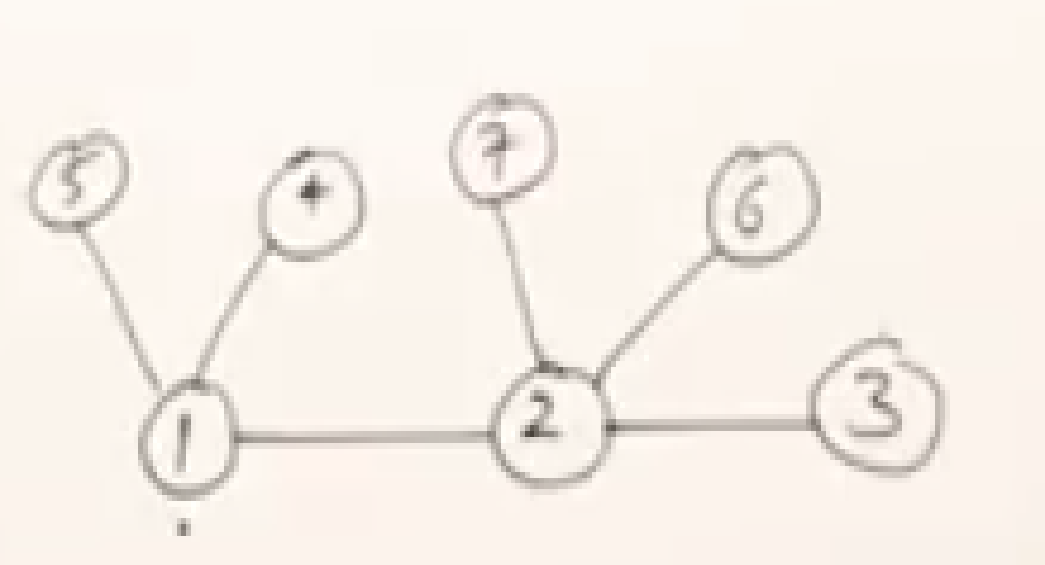
ADVANTAGES OF BREADTH-FIRST SEARCH

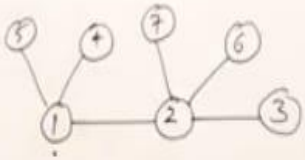
1. Breadth first search will never get trapped exploring the useless path forever.
2. If there is a solution, BFS will definitely find it out.
3. If there is more than one solution then BFS can find the minimal one that requires less number of steps.

DISADVANTAGES OF BREADTH-FIRST SEARCH

1. The main drawback of Breadth first search is its memory requirement. Since each level of the tree must be saved in order to generate the next level, and the amount of memory is proportional to the number of nodes stored, the space complexity of BFS is $O(b^d)$. As a result, BFS is severely space-bound in practice so will exhaust the memory available on typical computers in a matter of minutes.
2. If the solution is farther away from the root, breath first search will consume lot of time.

a)

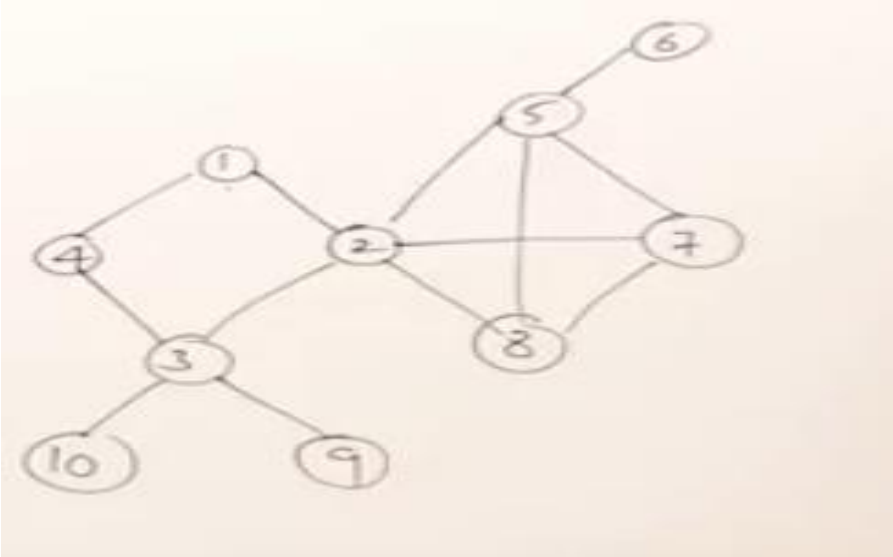




BFS: 1,2,4,5,7,3,6

DFS: 1,2,3,6,7,4,5

B)



BFS: 1,4,2,3,5,8,7,10,9,6

1) 1,2,4,8,5,7,3,6,10,9

2) 5,2,8,7,6,3,1,9,10,4

1) 1,2,8,7,5,6,3,9,10,4,

2) 3,4,1,2,5,6,7,8,10,9

Depth-first search

- **Depth-first search always expands the *deepest node in the current frontier of the search tree.***
- The search proceeds immediately to the deepest level of the search tree, where the nodes have no successors. As those nodes are expanded, they are dropped from the frontier, so then the search “backs up” to the next deepest node that still has unexplored successors.
- depth-first search uses a LIFO queue.
- A LIFO queue means that the most recently generated node is chosen for expansion. This must be the deepest unexpanded node because it is one deeper than its parent—which, in turn,was the deepest unexpanded node when it was selected.

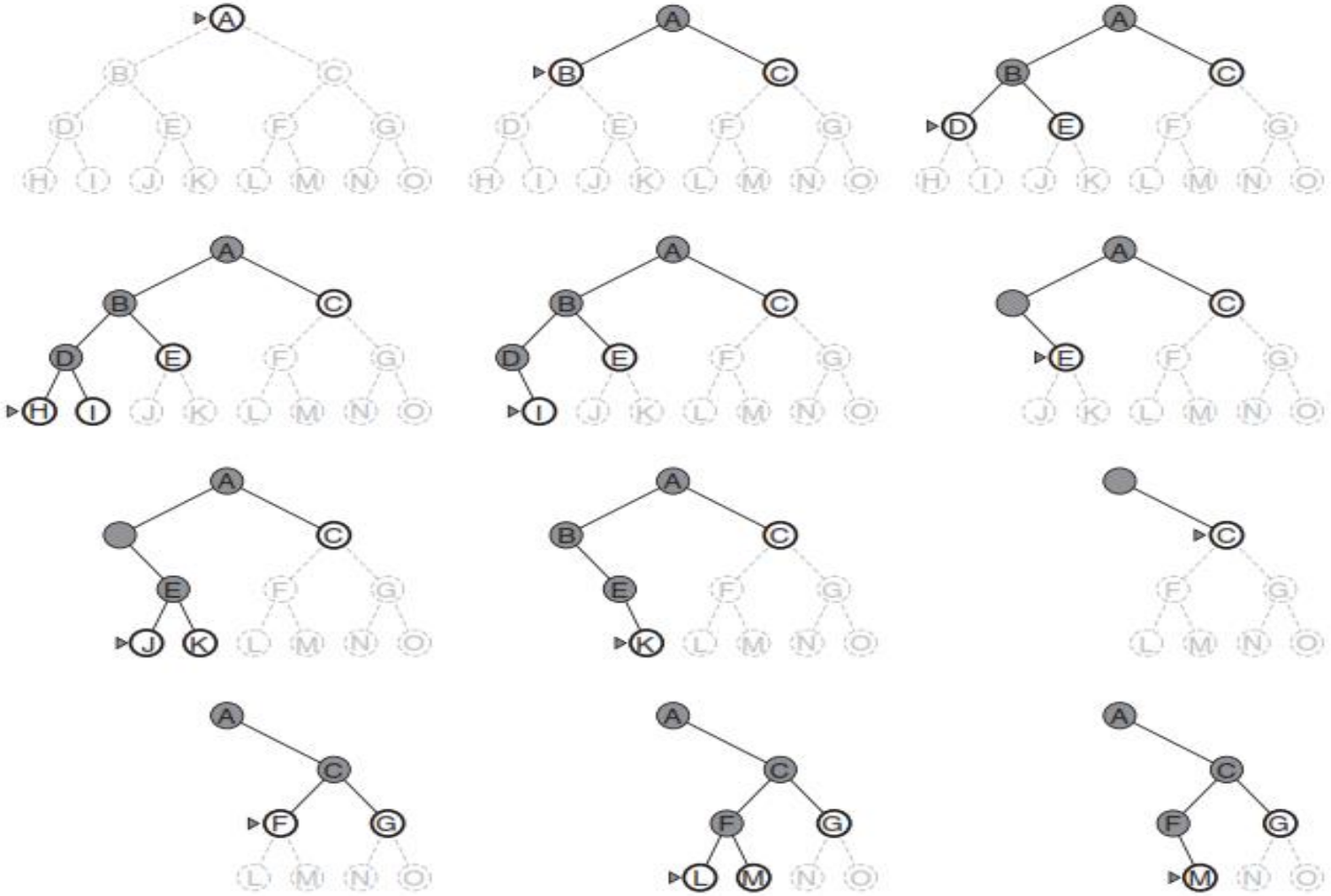


Figure 3.16 Depth-first search on a binary tree. The unexplored region is shown in light gray. Explored nodes with no descendants in the frontier are removed from memory. Nodes at depth 3 have no successors and M is the only goal node.

Examples with adv and dis adv

ALGORITHM: DEPTH FIRST SEARCH

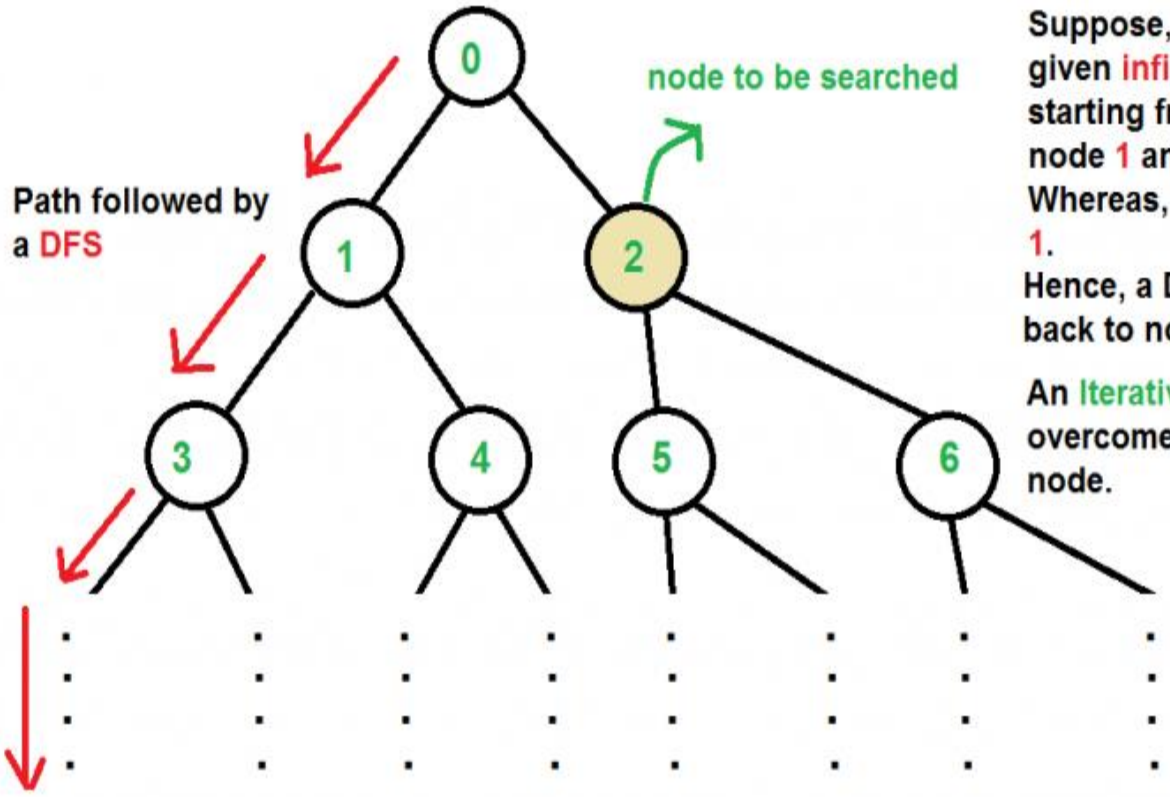
1.If the initial state is a goal state, quit and return success.
2.Otherwise, loop until success or failure is signaled.
a) Generate a state, say E, and let it be the successor of the initial state. If there is no successor, signal failure.
b) Call Depth-First Search with E as the initial state.
c) If success is returned, signal success. Otherwise continue in this loop.

ADVANTAGES OF DEPTH-FIRST SEARCH

- The advantage of depth-first Search is that memory requirement is only linear with respect to the search graph. This is in contrast with breadth-first search which requires more space. The reason is that the algorithm only needs to store a stack of nodes on the path from the root to the current node.

- The time complexity of a depth-first Search to depth d is O(b^d) since it generates the same set of nodes as breadth-first search, but simply in a different order. Thus practically depth-first search is time-limited rather than space-limited.

- If depth-first search finds solution without exploring much in a path then the time and space it takes will be very less.



Suppose, we want to find node- '2' of the given infinite undirected graph/tree. A DFS starting from node- 0 will dive left, towards node 1 and so on.
Whereas, the node 2 is just adjacent to node 1.
Hence, a DFS wastes a lot of time in coming back to node 2.

An Iterative Deepening Depth First Search overcomes this and quickly find the required node.

Iterative deepening search

BFS

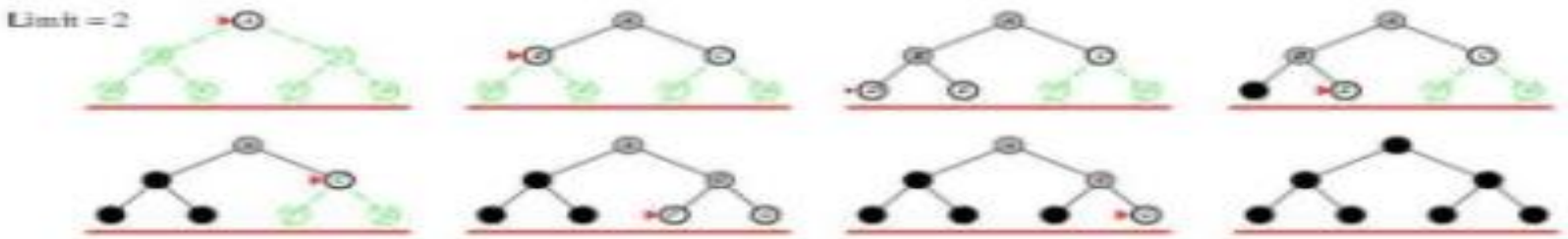
- Adv: It will find goal node at any point
- Dis adv:It consume very large memory.

DFS

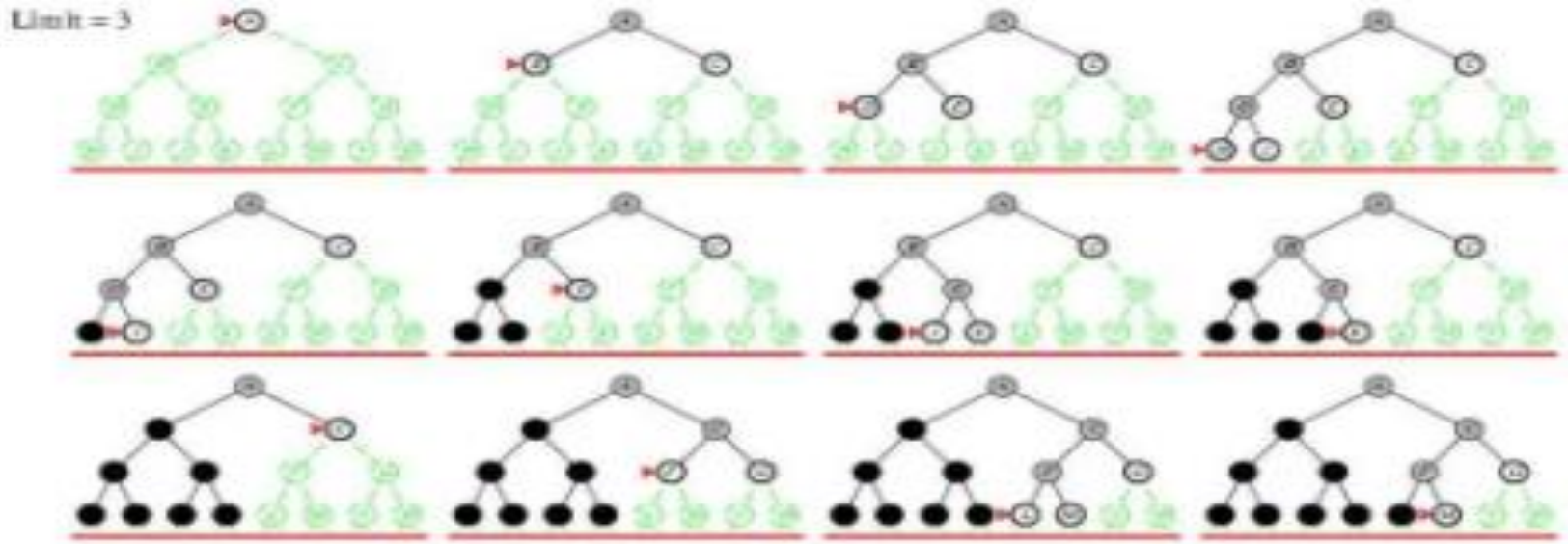
- Dis Adv: It is not sure that goal node at any point will be found.
- adv:It consume less memory.

ITERATIVE DEEPENING SEARCH *L* =1

ITERATIVE DEEPENING SEARCH *L* =2



ITERATIVE DEEPENING SEARCH *L* =3



Advantages

- IDDFS gives us the hope to find the solution if it exists in the tree.
- When the solutions are found at the lower depths say n, then the algorithm proves to be efficient and in time.
- The great advantage of IDDFS is found in-game tree searching where the IDDFS search operation tries to improve the depth definition, heuristics, and scores of searching nodes so as to enable efficiency in the search algorithm.
- Another major advantage of the IDDFS algorithm is its quick responsiveness. The early results indications are a plus point in this algorithm. This followed up with multiple refinements after the individual iteration is completed.
- Though the work is done here is more yet the performance of IDDFS is better than single BFS and DFS operating exclusively.
- Space and time complexities are expressed as: O(d) and here d is defined as goal depth.
- Let us consider the run time of IDDFS. Let say b>l where b is branching factor and l is the depth limit. Then next we search the goal node under the bound k. On the depth k, we say there may be b^k nodes that are only generated once. Similarly, the nodes at the depth limit k-1 is twice and thrice for k-2 depth. Thus the node

Disadvantages

- The time taken is exponential to reach the goal node.
- The main problem with IDDFS is the time and wasted calculations that take place at each depth.
- The IDDFS might fail when the BFS fails. When we are to find multiple answers from the IDDFS, it gives back the success nodes and its path once even if it needs to be found again after multiple iterations. To stop the depth bound is not increased further.

PROPERTIES OF ITERATIVE DEEPENING SEARCH

- Complete- Yes
- Time- (b^d)
- Space- (b^d)
- Optimal- Yes, if step cost = 1

EXAMPLE 1:- SEARCH TREE FOR THE 8 PUZZLE PROBLEM

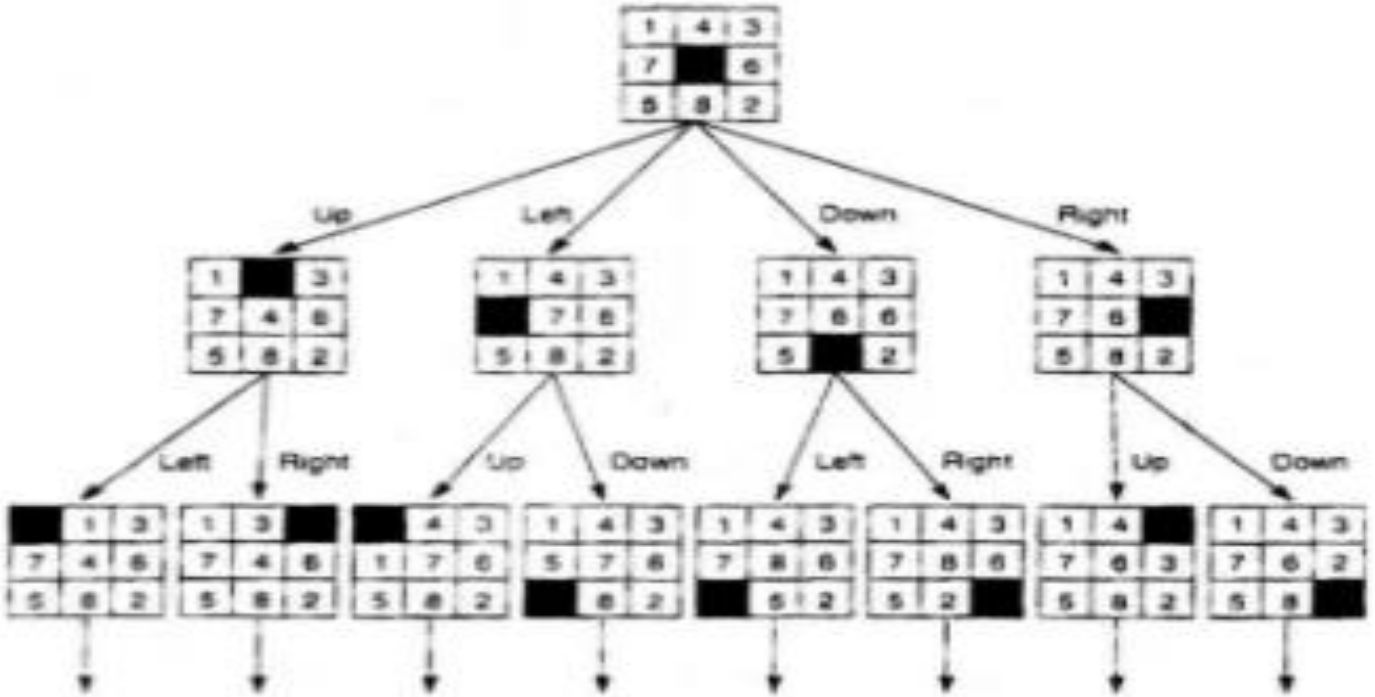
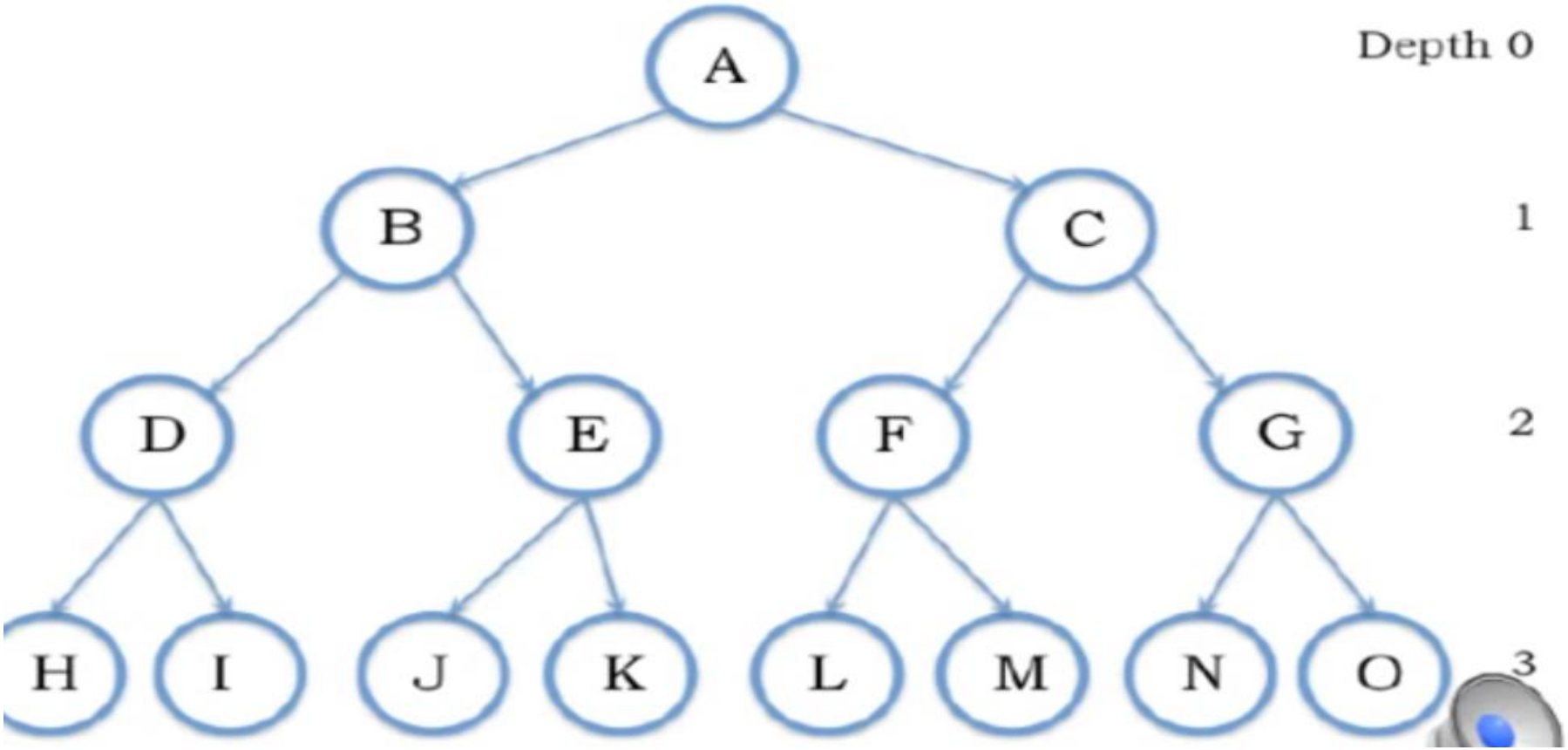


Figure 3.6 State space of the 8-puzzle generated by "move blank" operations.

Over ALL the iterations, from depth bound 0 to 3, the order in which nodes removed from the frontier is:

A A B C A B D E C F G A B D H I E J K C F L M G N O



Bidirectional Search

Searching a graph is quite famous problem and have a lot of practical use. We have already discussed here how to search for a goal vertex starting from a source vertex using BFS. In normal graph search using BFS/DFS we begin our search in one direction usually from source vertex toward the goal vertex, **but what if we start search form both direction simultaneously.**

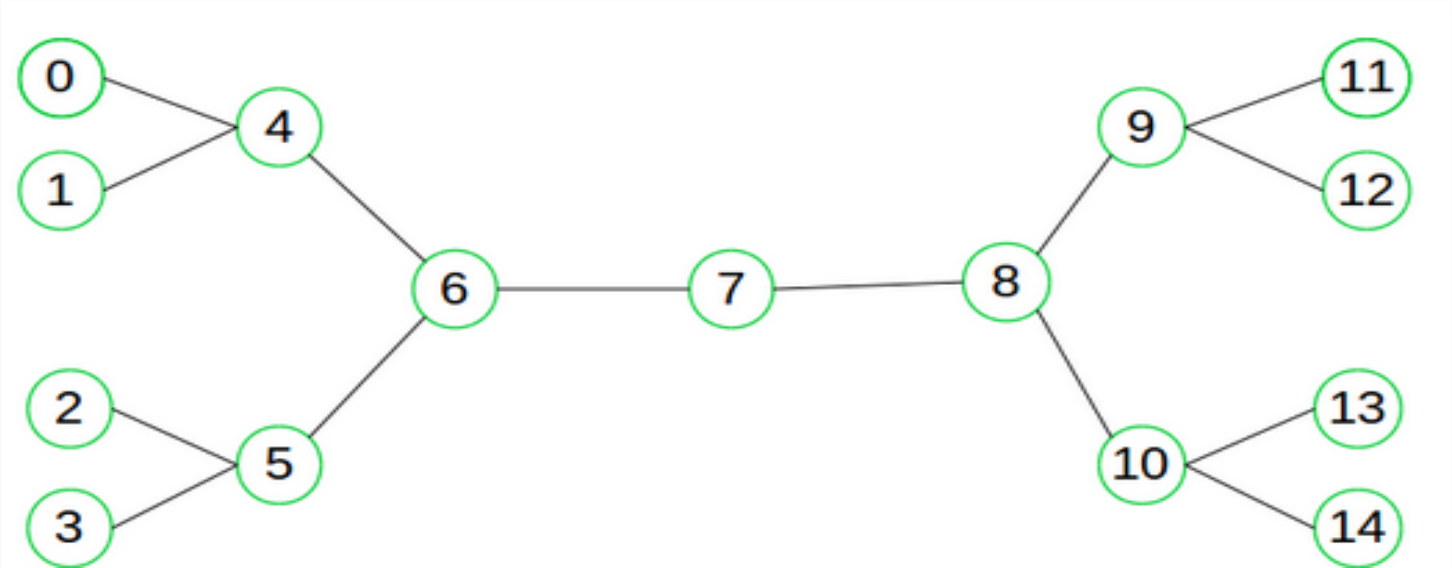
Bidirectional search is a graph search algorithm which find smallest path form source to goal vertex. It runs two simultaneous search –

1. Forward search form source/initial vertex toward goal vertex
2. Backward search form goal/target vertex toward source vertex

Bidirectional search replaces single search graph(which is likely to grow exponentially) with two smaller sub graphs – one starting from initial vertex and other starting from goal vertex. **The search terminates when two graphs intersect.**

Just like A* algorithm, bidirectional search can be guided by a heuristic estimate of remaining distance from source to goal and vice versa for finding shortest path possible.

Consider following simple example-



Suppose we want to find if there exists a path from vertex 0 to vertex 14. Here we can execute two searches, one from vertex 0 and other from vertex 14. When both forward and backward search meet at vertex 7, we know that we have found a path from node 0 to 14 and search can be terminated now. We can clearly see that we have successfully avoided unnecessary exploration.

Why bidirectional approach?

Because in many cases it is faster, it dramatically reduce the amount of required exploration.

Suppose if branching factor of tree is **b** and distance of goal vertex from source is **d**, then the normal BFS/DFS searching complexity would be $O(b^d)$. On the other hand, if we execute two search operation then the complexity would be $O(b^{d/2})$ for each search and total complexity would be $O(b^{d/2} + b^{d/2})$ which is far less than $O(b^d)$.

When to use bidirectional approach?

We can consider bidirectional approach when-

1. Both initial and goal states are unique and completely defined.
2. The branching factor is exactly the same in both directions.

Performance measures

- Completeness : Bidirectional search is complete if BFS is used in both searches.
- Optimality : It is optimal if BFS is used for search and paths have uniform cost.
- Time and Space Complexity : Time and space complexity is $O(b^{d/2})$

Advantages:

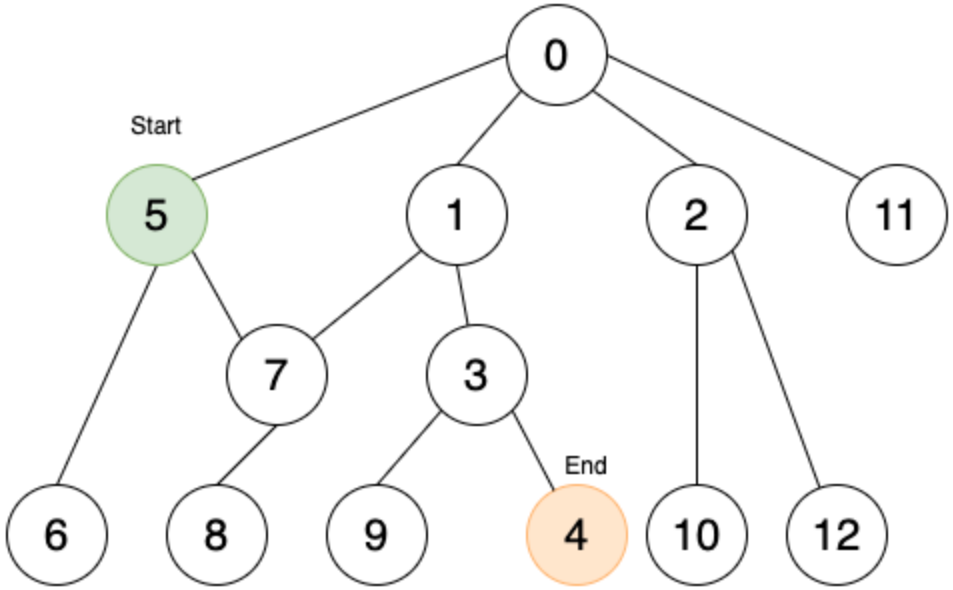
→ Bidirectional search is fast

→ Bidirectional search requires less m

Disadvantages:

→ Implementation of the bidirectional difficult.

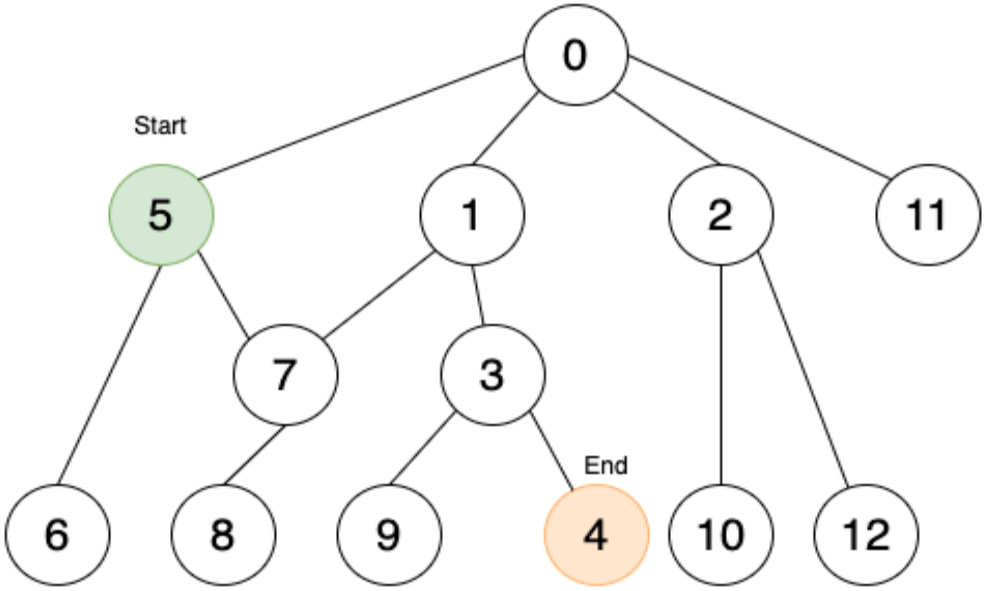
→ In bidirectional search, one should state in advance.



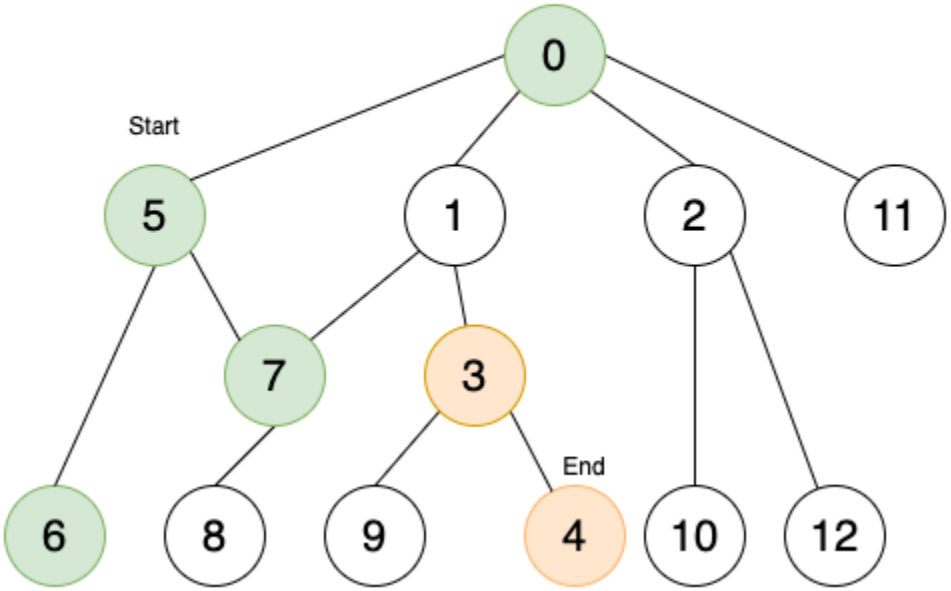
The start node is 5 and the end node is 4.

Aim: To find the shortest path from 5 to 4 using bidirectional search.

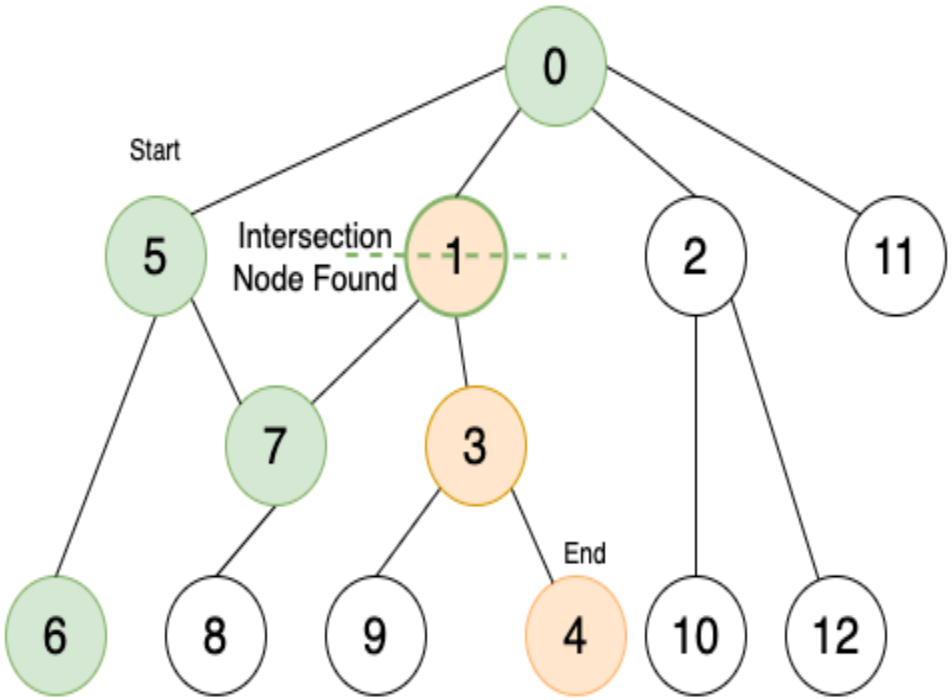
1. Start moving forward from start node (Green) and backwards from end node (Orange).



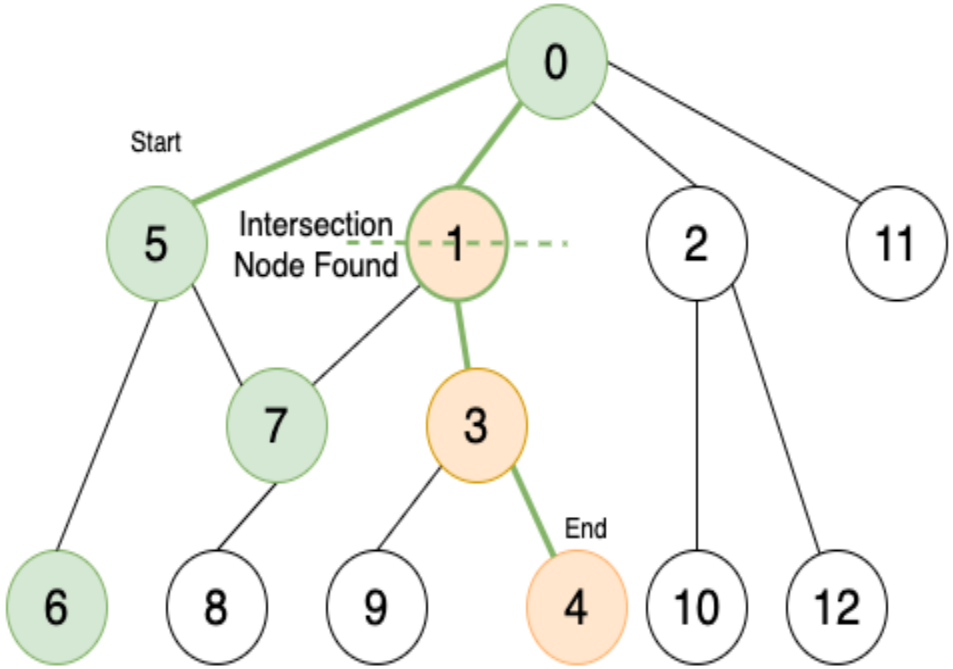
2. Similar to BFS, at every point explore the next level of nodes till you find an intersecting node.



3. Stop on finding the intersecting node.



4. Trace back to find the path



- Path ?

Comparing results from BFS and Bidirectional Search: