AIE111 :

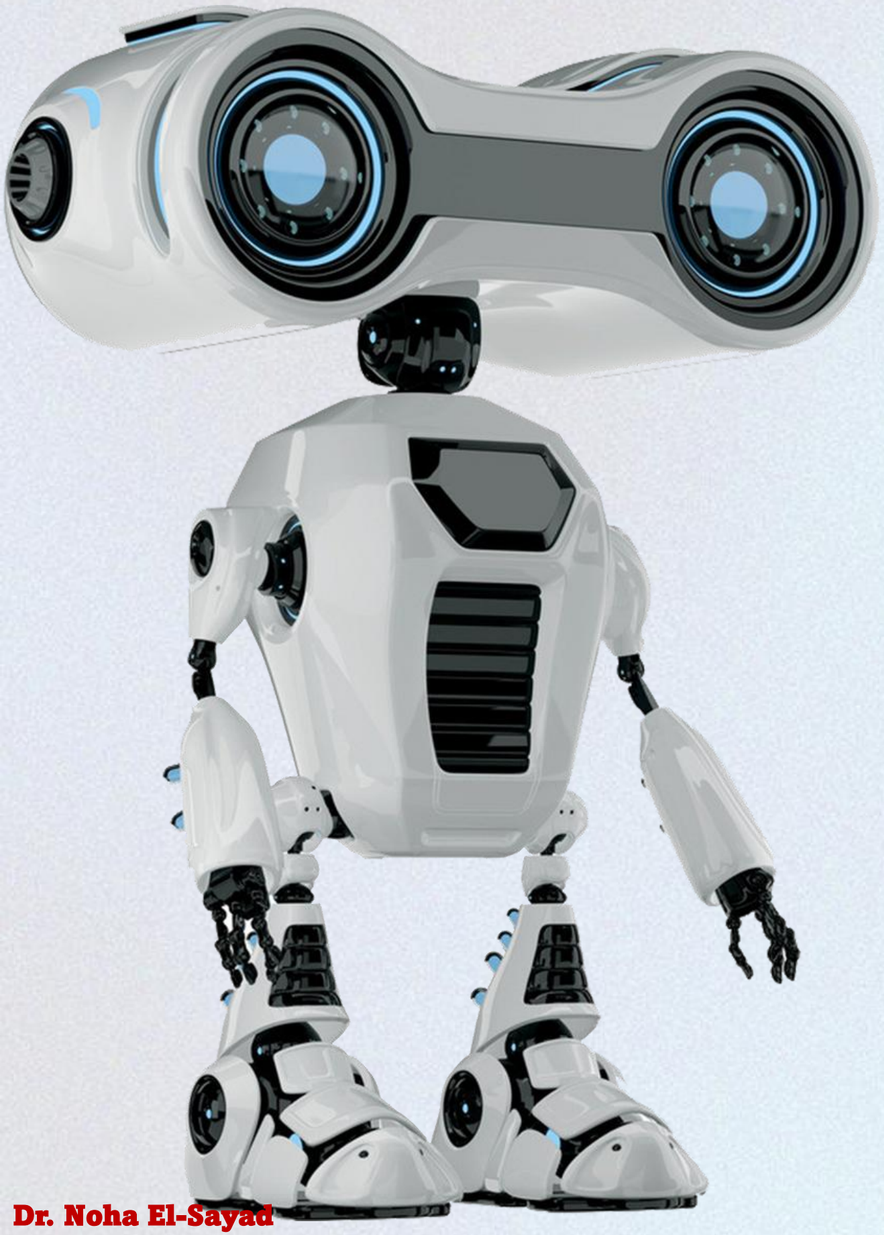
Artificial Intelligence

Dr. Noha El-Sayad

Lecture 3:

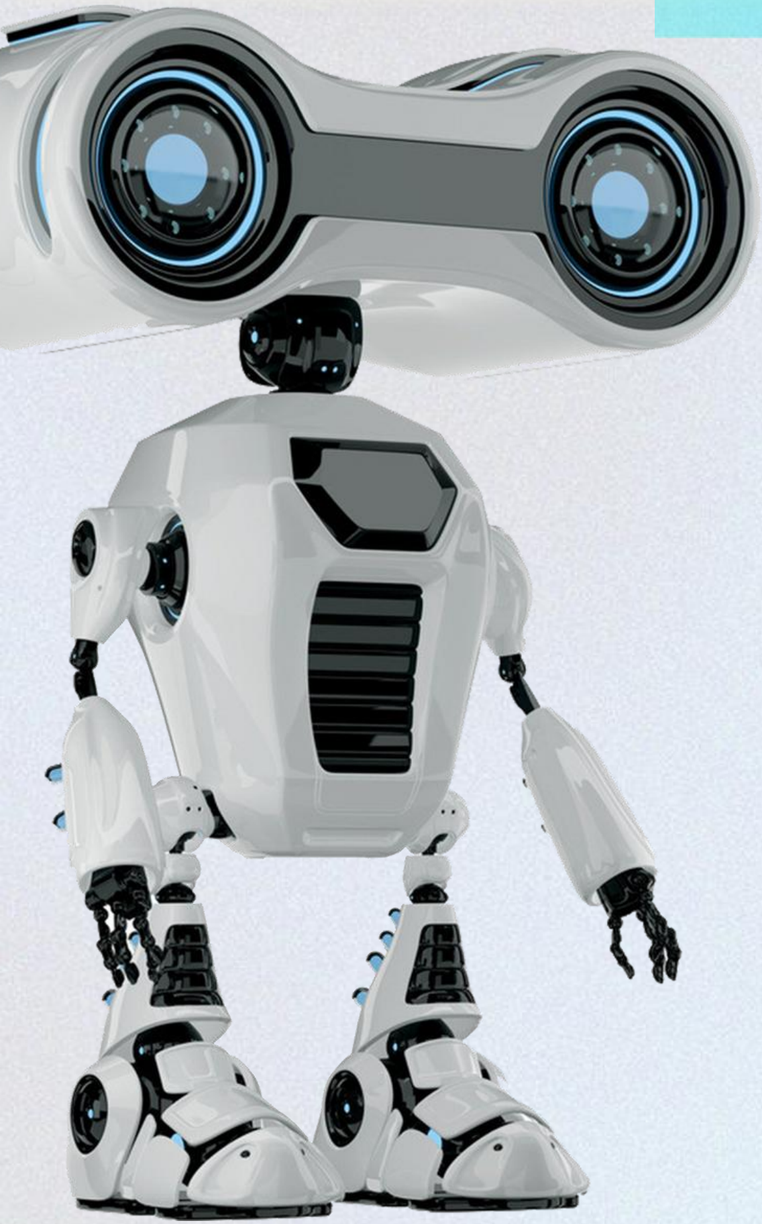
Solving Problems by Searching Algorithm

What is Search in AI?



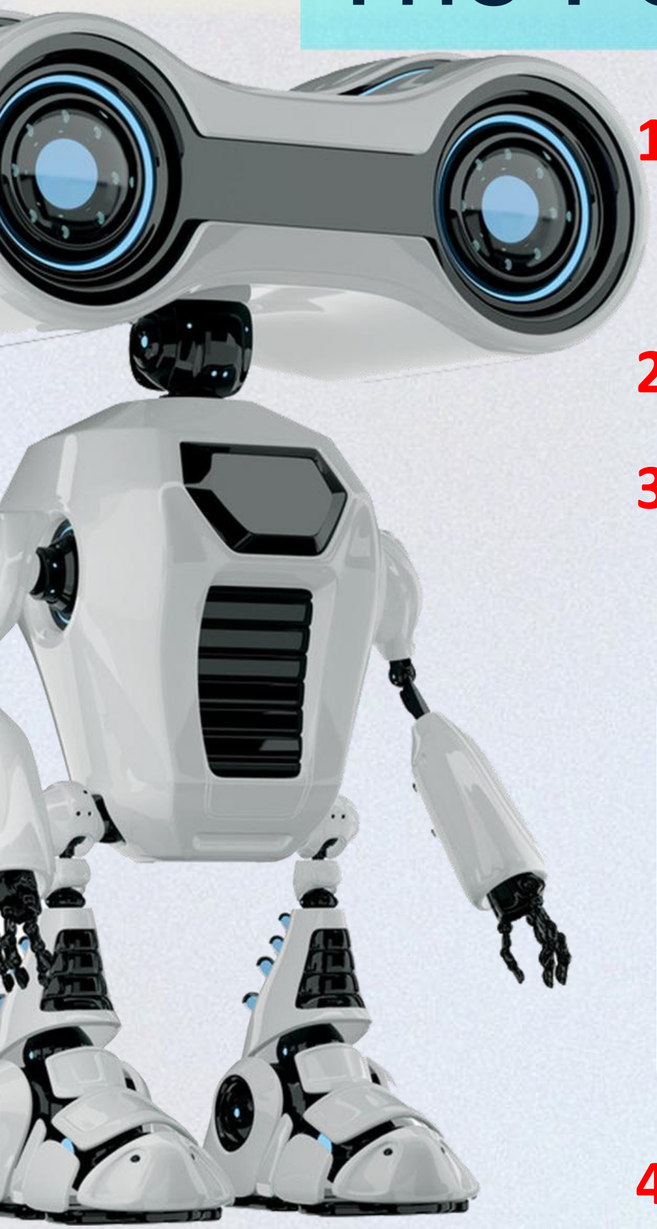
- Search is a problem-solving technique.
- It explores different states to find a solution.
- Search Space: Includes all possible states leading to a solution.

What is Problem-Solving in AI?



- AI looks ahead to find the best sequence of actions to achieve a goal.
- **A problem-solving agent** plans and searches for solutions before taking action.
- The **process** of finding solutions is called searching.
- There are two types of search algorithms:
  1. **Uninformed Search** – No prior knowledge, explores blindly.
  2. **Informed Search** – Uses heuristics (guidance) to improve efficiency.

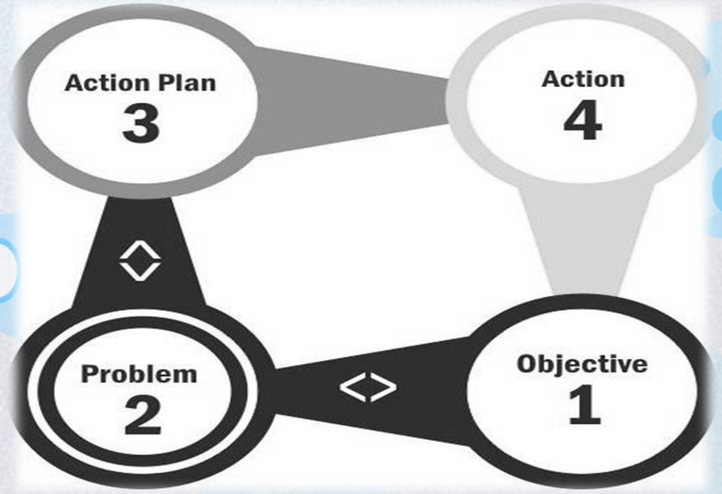
The Four-Phase Problem-Solving Process



1. **Goal Formulation** – Define the target (e.g., AI wants to reach Bucharest).
2. **Problem Formulation** – Describe the problem as states and actions.
3. **Search** – Simulate different action sequences to find a path to the goal.

Before acting in the real world, the agent **simulates action sequences** within its model to find a path to the goal.

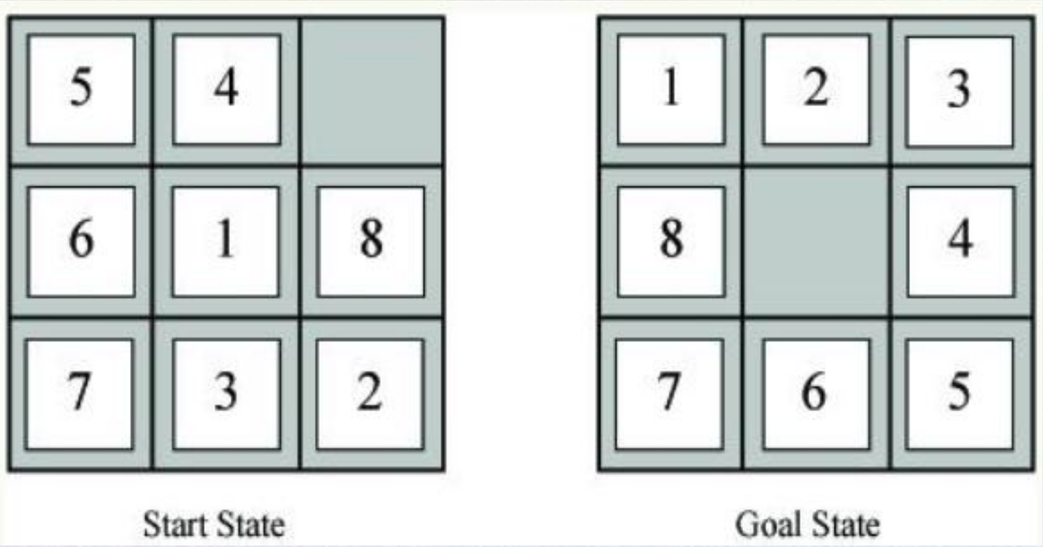
- A successful sequence is called a **solution**.
- The agent may test **multiple unsuccessful sequences** before finding a solution or determining that no solution exists.



4. **Execution** – Perform the chosen actions to solve the problem.

The Four-Phase Problem-Solving Process

Example Problems – Eight puzzle applet



States: tile (square) locations

Initial state: one specific tile configuration

Operators: move blank tile left, right, up, or down

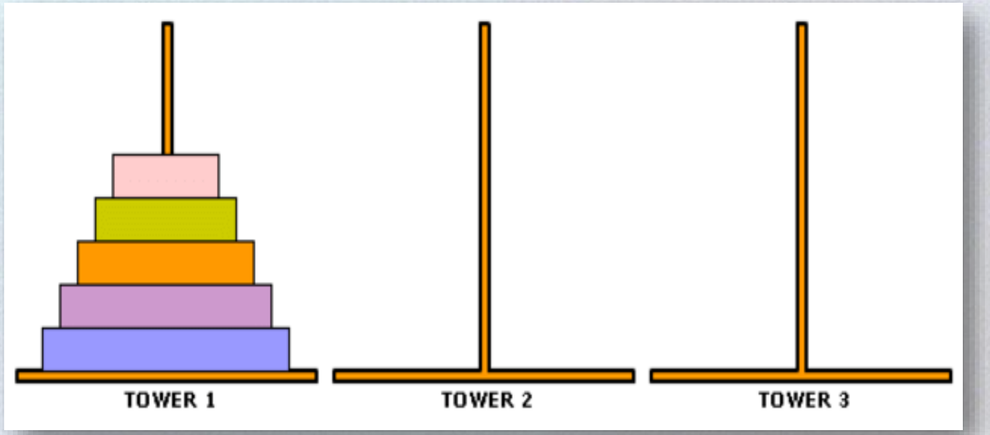
Goal: tiles are numbered from one to eight around the square

Path cost: cost of 1 per move

(solution cost same as number of most or path length)

The Four-Phase Problem-Solving Process

Example Problems – Towers of Hanoi



States:

- A state represents the current arrangement of disks on the three poles.
- Each state can be described as a set of configurations of disks on the poles.
- The initial state starts with all disks on the first pole, stacked from largest (at the bottom) to smallest (at the top).

Operators: The only operation allowed is moving a disk from one pole to another, subject to the following constraints:

1. A larger disk cannot be placed on top of a smaller disk.
2. A disk can only be moved if no other disks are on top of it.

Goal test: The goal state is achieved when all disks are transferred to the target pole (e.g., C) in the correct order

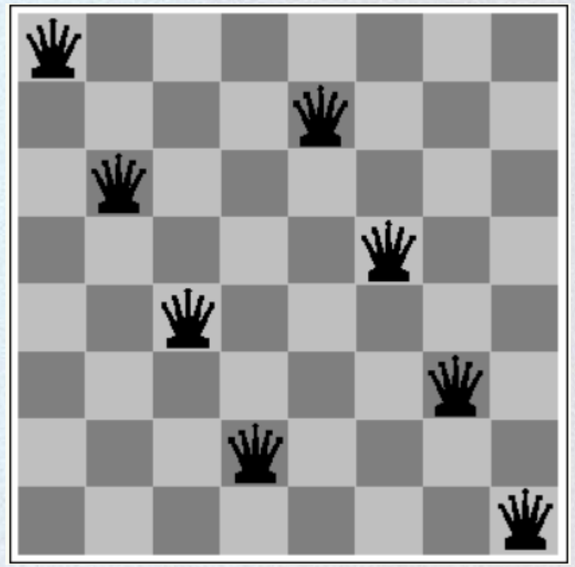
(largest at the bottom, smallest at the top).

Path cost: Each move has a cost of 1.

The minimum number of moves required to solve the puzzle for nn disks is:2^n-1

Examples:1 disk → 1 move, 2 disks → 3 moves, 3 disks → 7, moves4 disks → 15 moves

The Four-Phase Problem-Solving Process



Example Problems – Eight queens applet

States: locations of 8 queens on chess board

Initial state: one specific queens

configuration

Operators: move queen x to row y and column z

Goal: no queen can attack another (cannot be in same row, column, or diagonal)

Path cost: 0 per move

The Four-Phase Problem-Solving Process

Example Problems – Route Finding Problem

States

– locations

Initial state

– starting point

Successor function (operators)

– move from one location to another

Goal test

– arrive at a certain location

Path cost

– may be quite complex • money, time, travel comfort, scenery

Car Navigation

Airline travel planning

Routing in computer Networks

The Four-Phase Problem-Solving Process

Example Problems – Automatic Assembly Sequencing

States

– location of components

- **Initial state**

– no components assembled

- **Successor function (operators)**

– place component

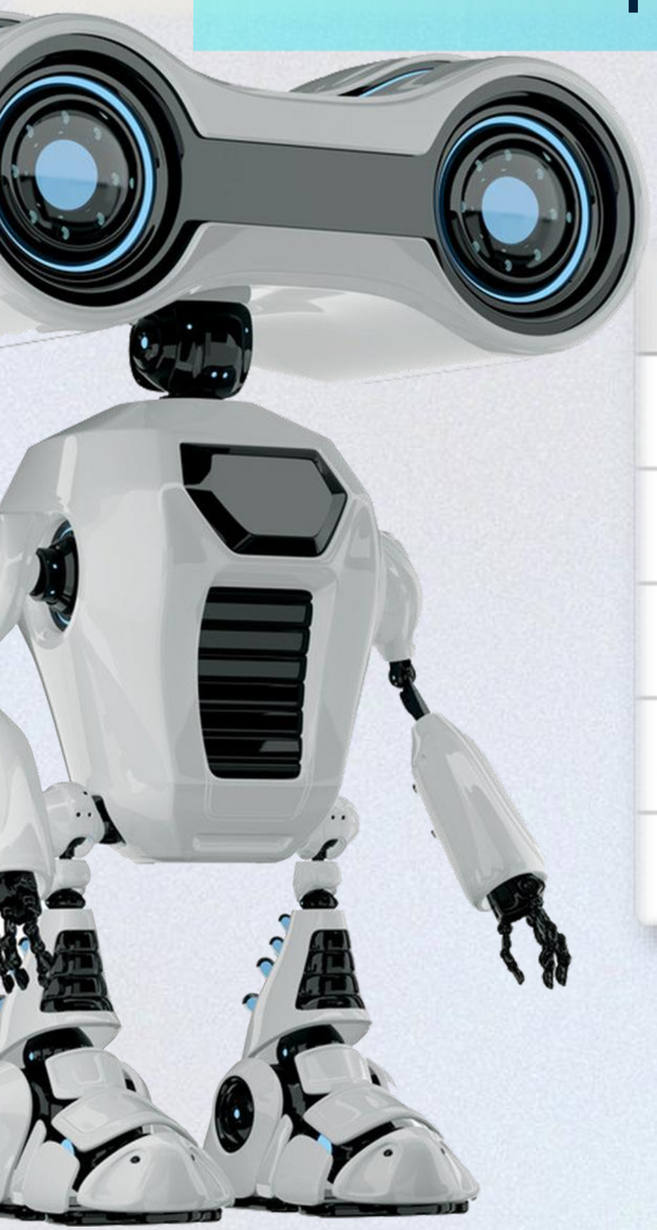
- **Goal test**

– system fully assembled

- **Path cost**

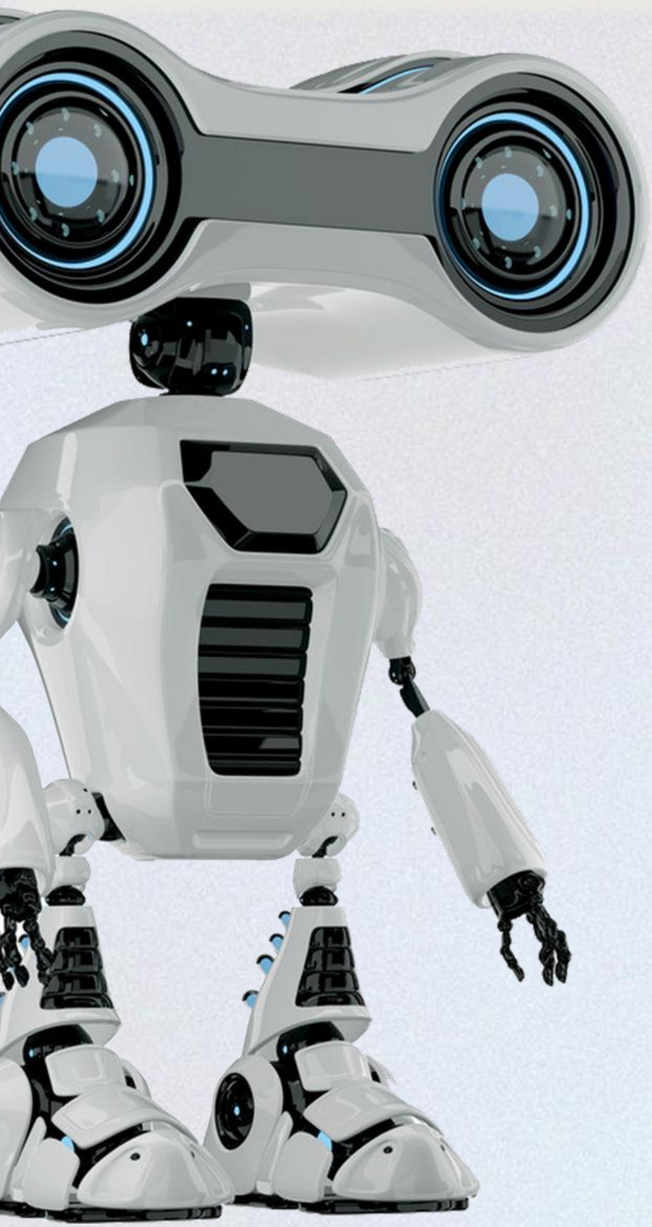
– number of movess

Search Space and Definitions (Key Terms in Searching)



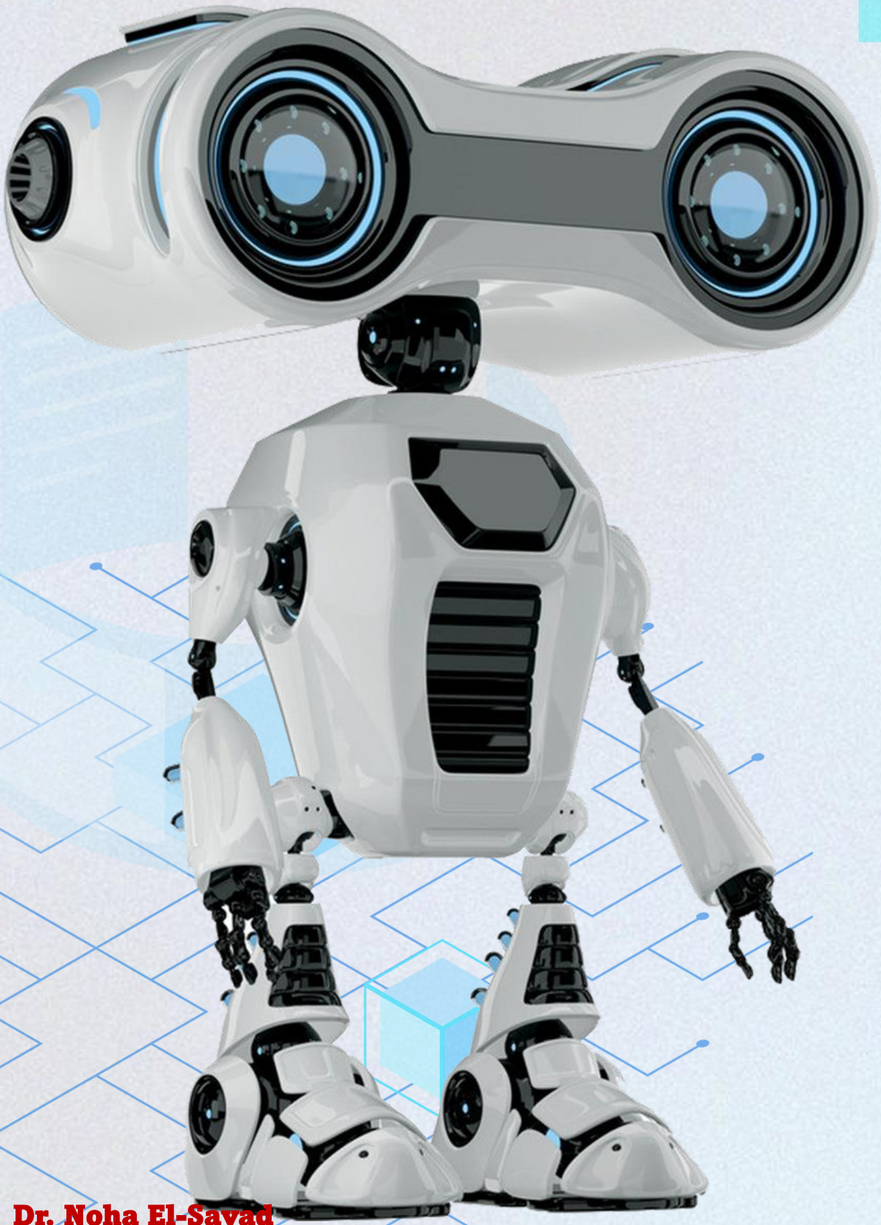
Term	Definition	Example
State	A description of a world situation.	The positions of pieces in a chess game.
Initial State	The starting condition of the problem.	The beginning of a puzzle game.
Goal State	A condition that satisfies the problem's solution.	Reaching Bucharest in a navigation problem.
Action	A move that transitions one state to another.	Moving from one city to another in a map.
Solution	The sequence of actions that reaches the goal.	A set of driving directions.

Example Problems and AI Solutions

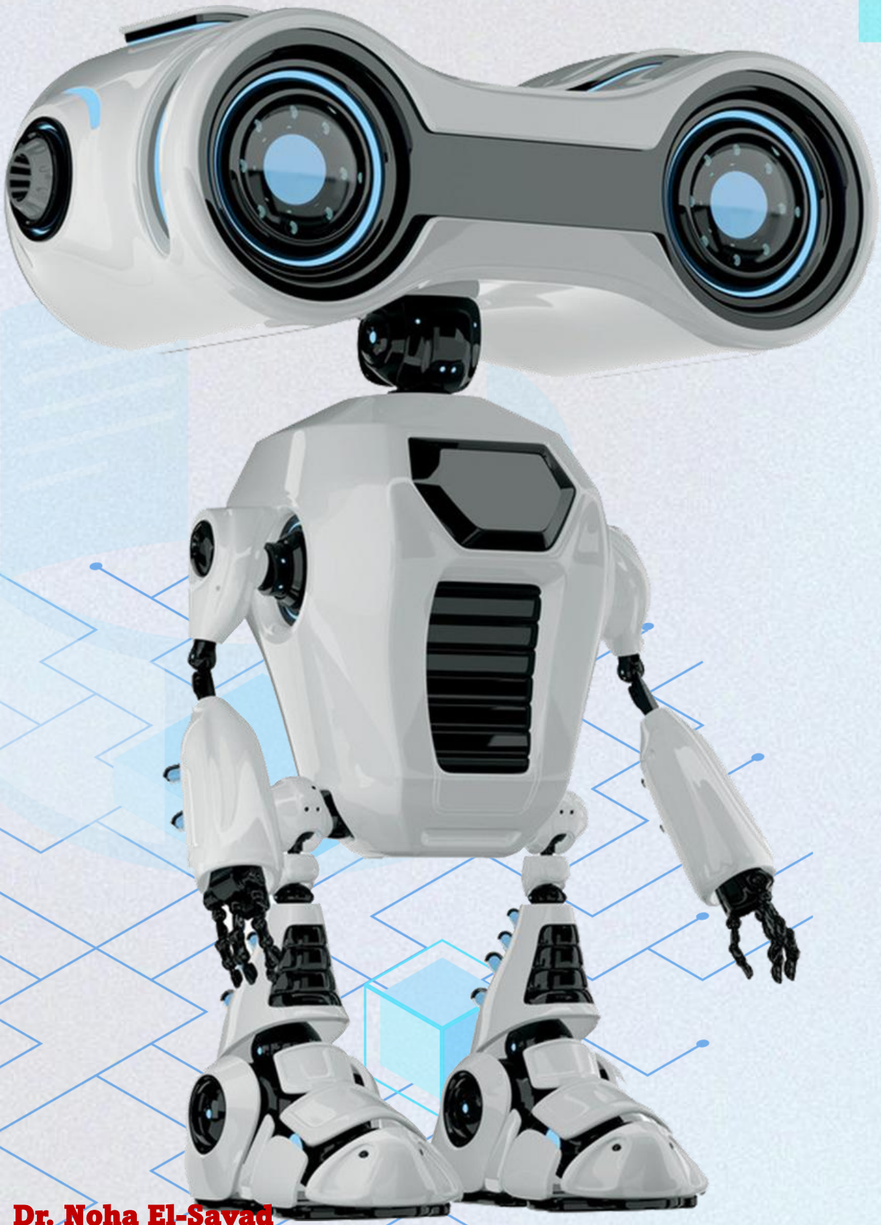


Problem	States	Operators (Actions)	Goal
8-Puzzle	Tile positions	Move tile left, right, up, down	Arrange tiles in order
Towers of Hanoi	Disk positions on poles	Move disk from one pole to another	Stack disks correctly
8-Queens Problem	Positions of queens	Move queen to a new position	No queens attack each other
Route Planning	Locations on a map	Move from one city to another	Reach the destination
Robot Assembly	Parts of a system	Attach parts together	Fully assembled system

AI Search Algorithm Terminologies?

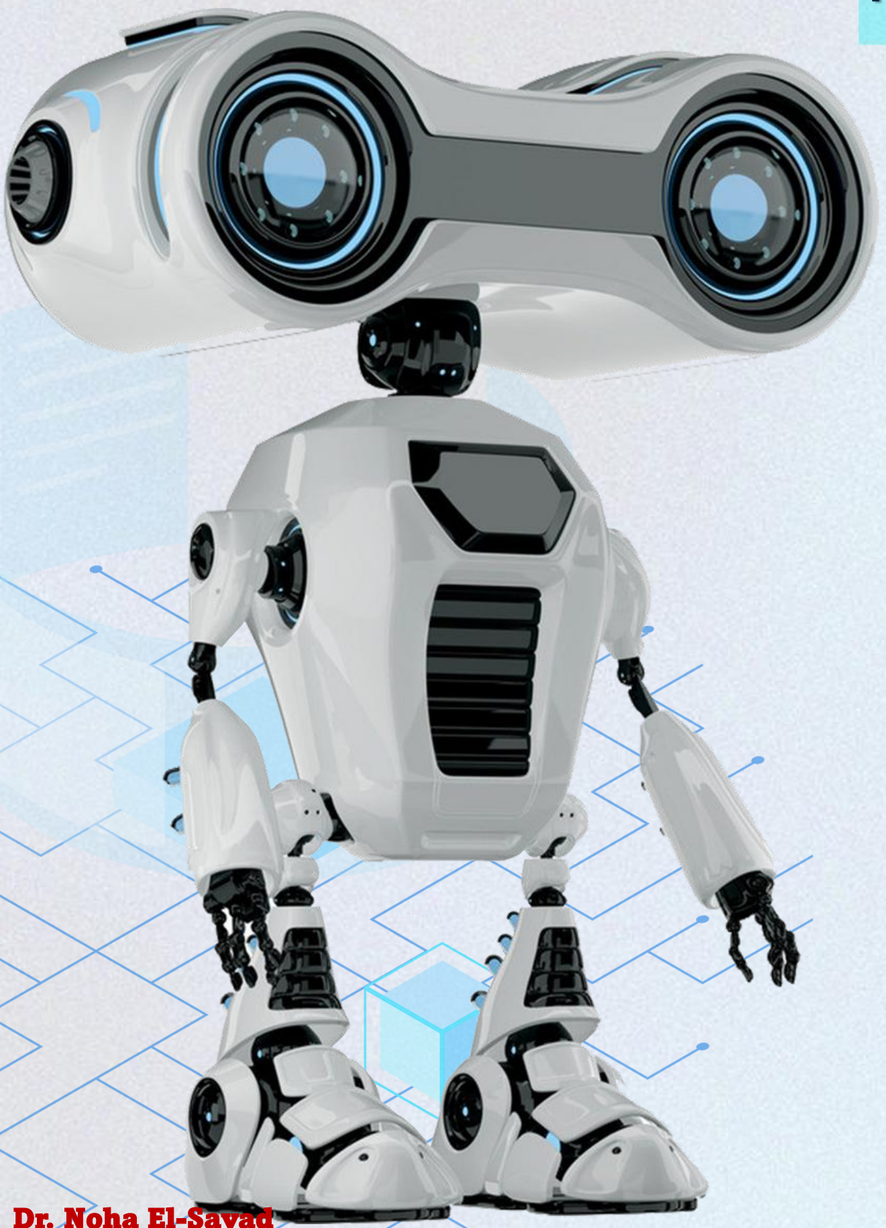


2. **Search tree** - A Search tree is a tree representation of a search issue. The node at the root of the search tree corresponds to the initial condition.
3. **Actions** - It describes all the steps, activities, or operations accessible to the agent.
4. **Transition model** - It can be used to convey a description of what each action does.
5. **Path Cost** - It is a function that gives a cost to each path.
6. **Solution** - An action sequence connects the start node to the target node.
7. **Optimal Solution** - If a solution has the lowest cost among all solutions, it is said to be the optimal answer.



Properties of Search Algorithms

1. **Completeness -** A search algorithm is said to be complete if it guarantees to yield a solution for any random input if at least one solution exists.
2. **Optimality -** A solution discovered for an algorithm is considered optimal if it is assumed to be the best solution (lowest path cost) among all other solutions.
3. **Time complexity -** It measures how long an algorithm takes to complete its job.
4. **Space Complexity -** The maximum storage space required during the search, as determined by the problem's complexity.



Importance of Search Algorithms in Artificial Intelligence

1. **Problem Solving:** AI search algorithms enhance problem-solving by logically exploring search spaces, as seen in route planning applications like Google Maps.
2. **Search Programming:** Many AI tasks can be formulated as search problems, improving problem-solving efficiency.
3. **Goal-Based Agents:** These agents use search algorithms to find the best course of action for optimal problem resolution.
4. **Production Systems:** AI search helps production systems apply rules and methods to execute actions effectively.
5. **Neural Networks:** Search algorithms optimize connection weights in neural networks, improving AI performance in tasks like input-output mapping.

Types of Search Problems

1. Deterministic & Fully Observable
2. Non-Observable (Sensorless)
3. Partially Observable & Nondeterministic

Search Algorithms: Complete vs. Incomplete

Complete Search: Guaranteed to find a solution.

Incomplete Search: May not always find a solution but is more efficient.

Search Problem consist of:

- **Search Space:** Set of all possible states where you can be.
- **Start State:** The state from where the search begins, also called root & source.
- **Goal State:** The state from where the search ends, also called destination.
- **Objective Function:** A function that looks at the current state and returns whether or not it is the goal state.

State Space Representation

- **Using Graphs**: Nodes (states), Arcs (steps).
- **Using Trees:** A special graph with a single path between nodes.
- **Root node:** represents the starting point.

Searching Algorithm in AI: Types

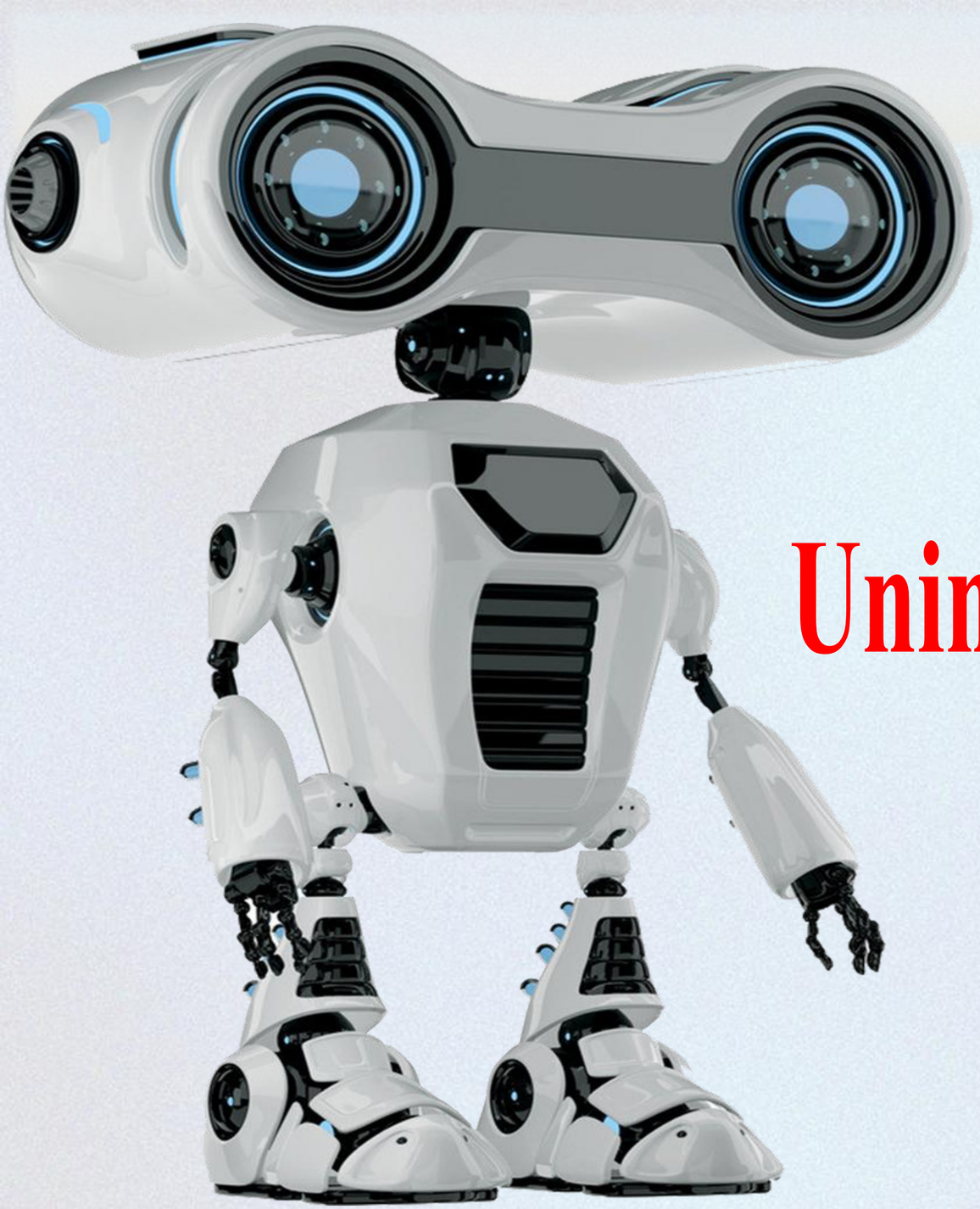
Uninformed Search Algorithms:

•Uninformed search algorithms **do not have additional information** about the goal or search space **other than how to traverse the tree**, so it is also called **blind search**.

Informed Search Algorithms:

•Informed Search Algorithms have **information about the goal state**, which helps in more efficient searching. This information is obtained by something called a **heuristic**.

Uninformed Search	Informed Search
– breadth-first – uniform-cost search – depth-first – depth-limited search – iterative deepening – bi-directional search	– best-first search – search with heuristics – memory-bounded search – iterative improvement search

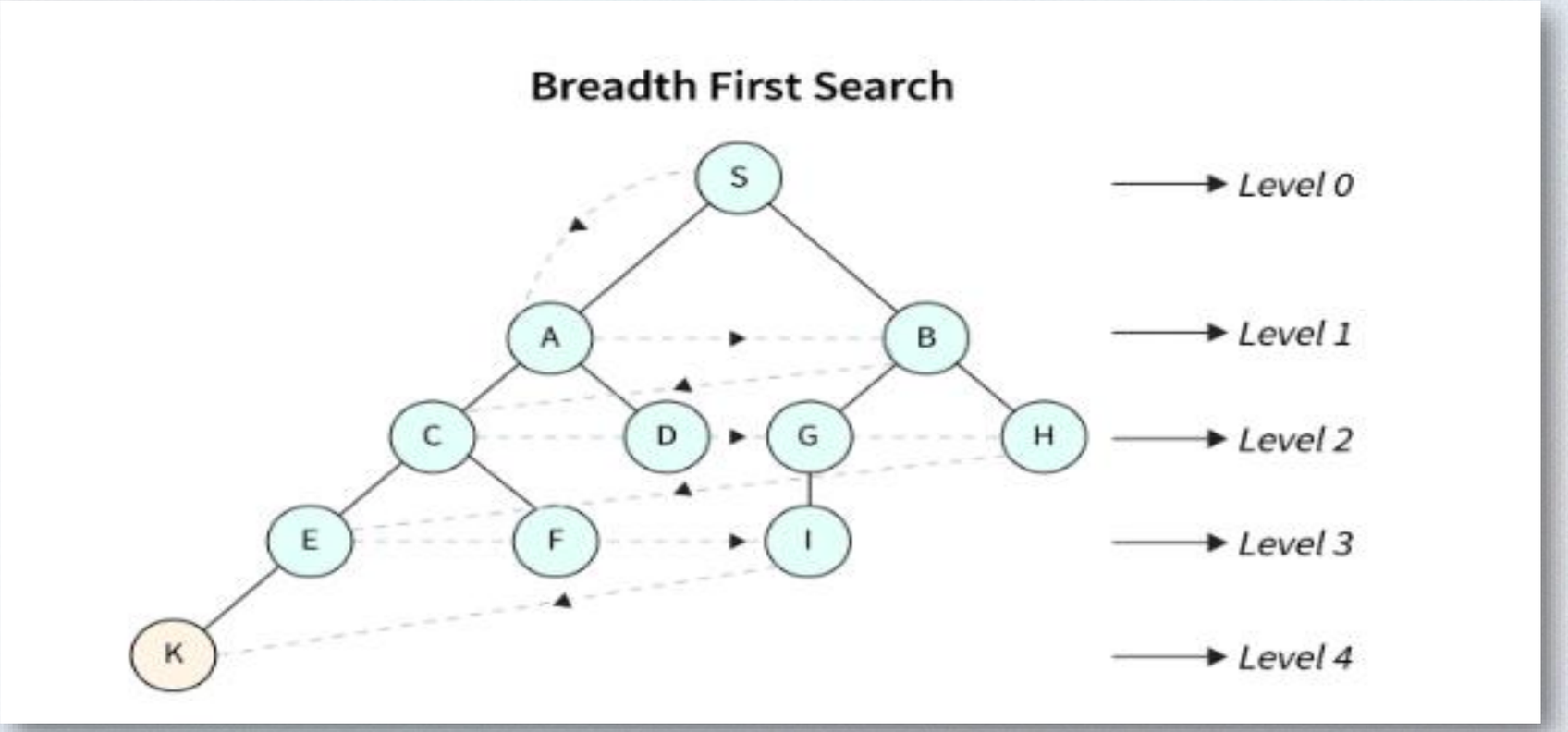


Uninformed Search Strategies

1. Breadth-first search (BFS)

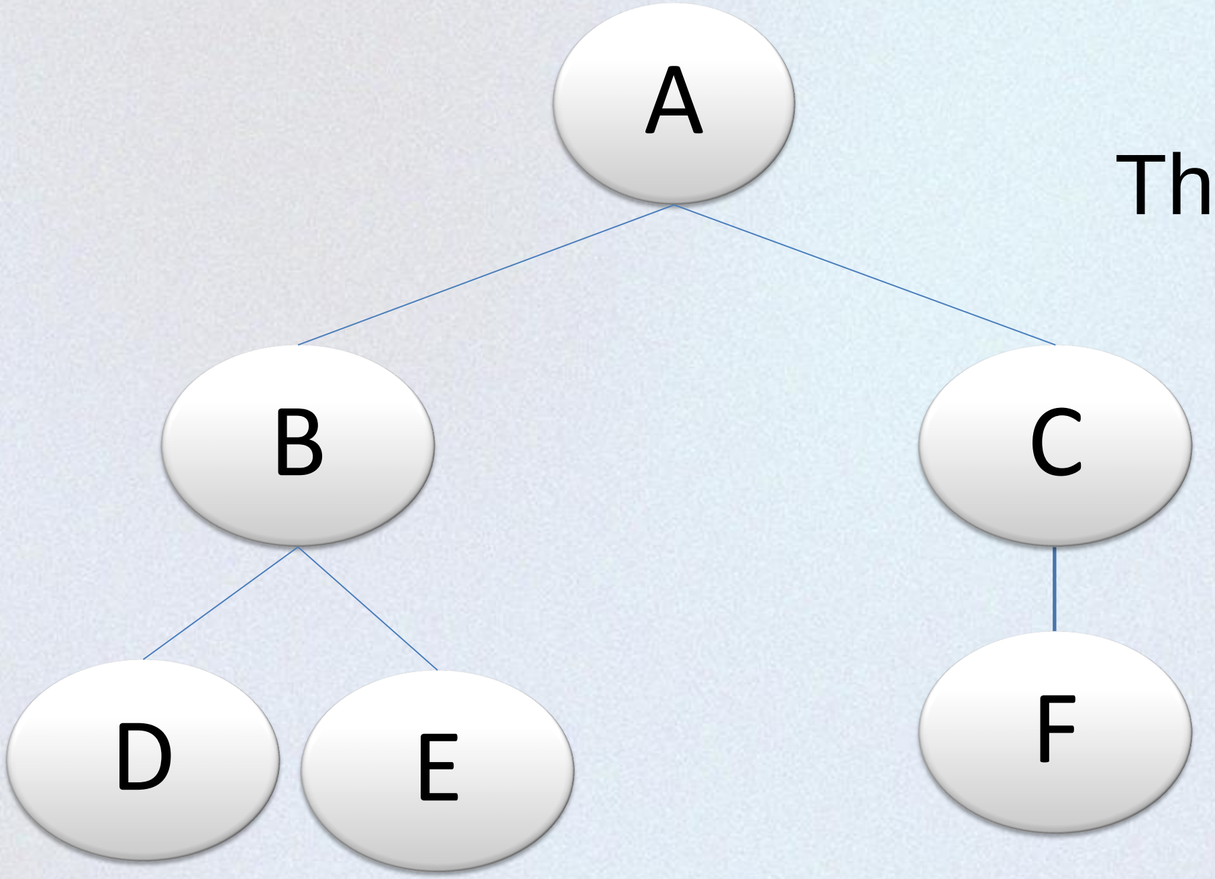
Breadth-First Search (BFS): A graph or tree search method that starts at the root and explores all nodes at the current depth before moving to the next level.

It uses a **FIFO queue** and is a **complete algorithm**, guaranteeing a solution if one exists.



1. Breadth-first search (BFS)

Example

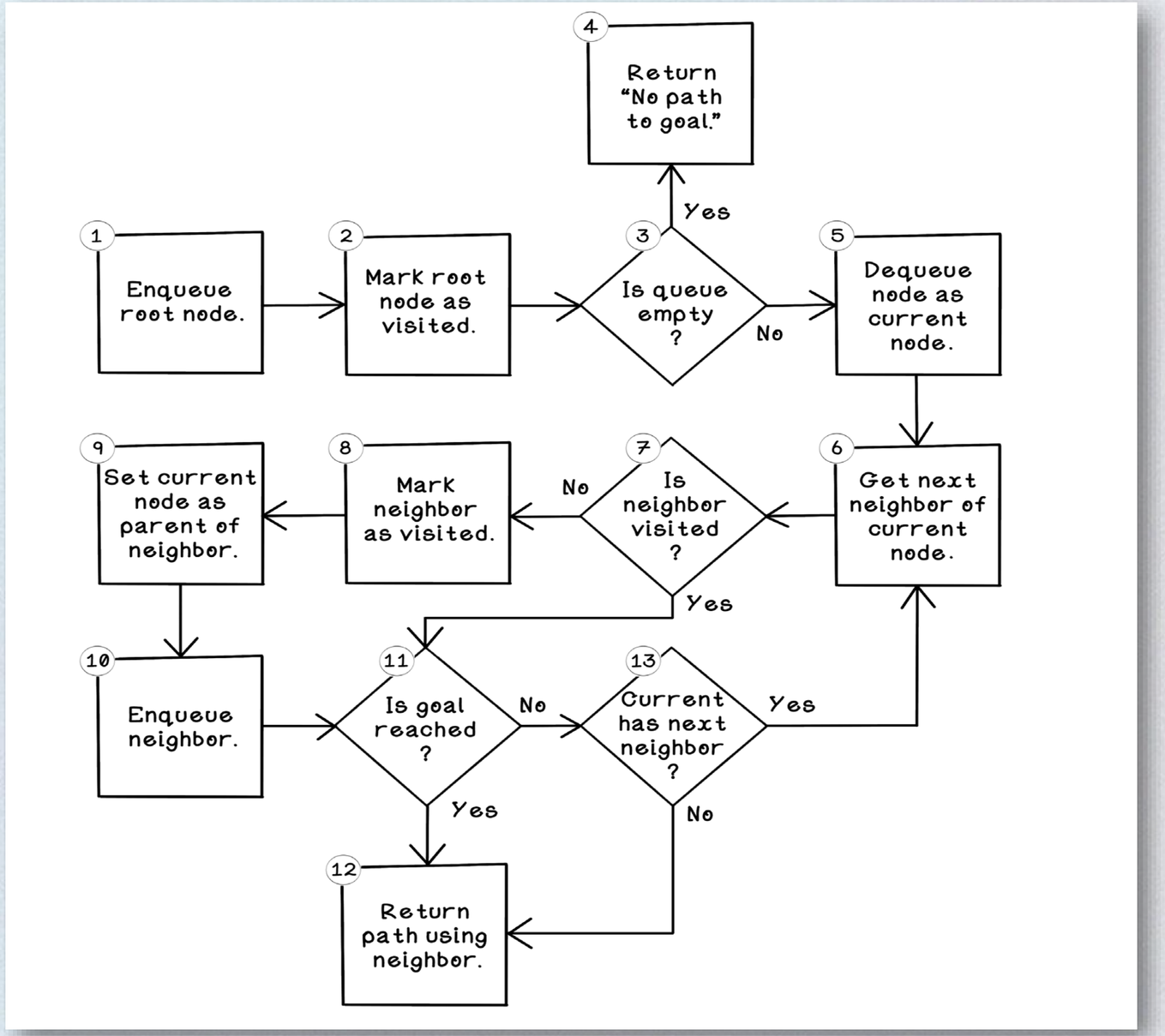


The **BFS** starting from node **A**

will visit: **A** → **B** → **C** → **D** → **E** → **F**

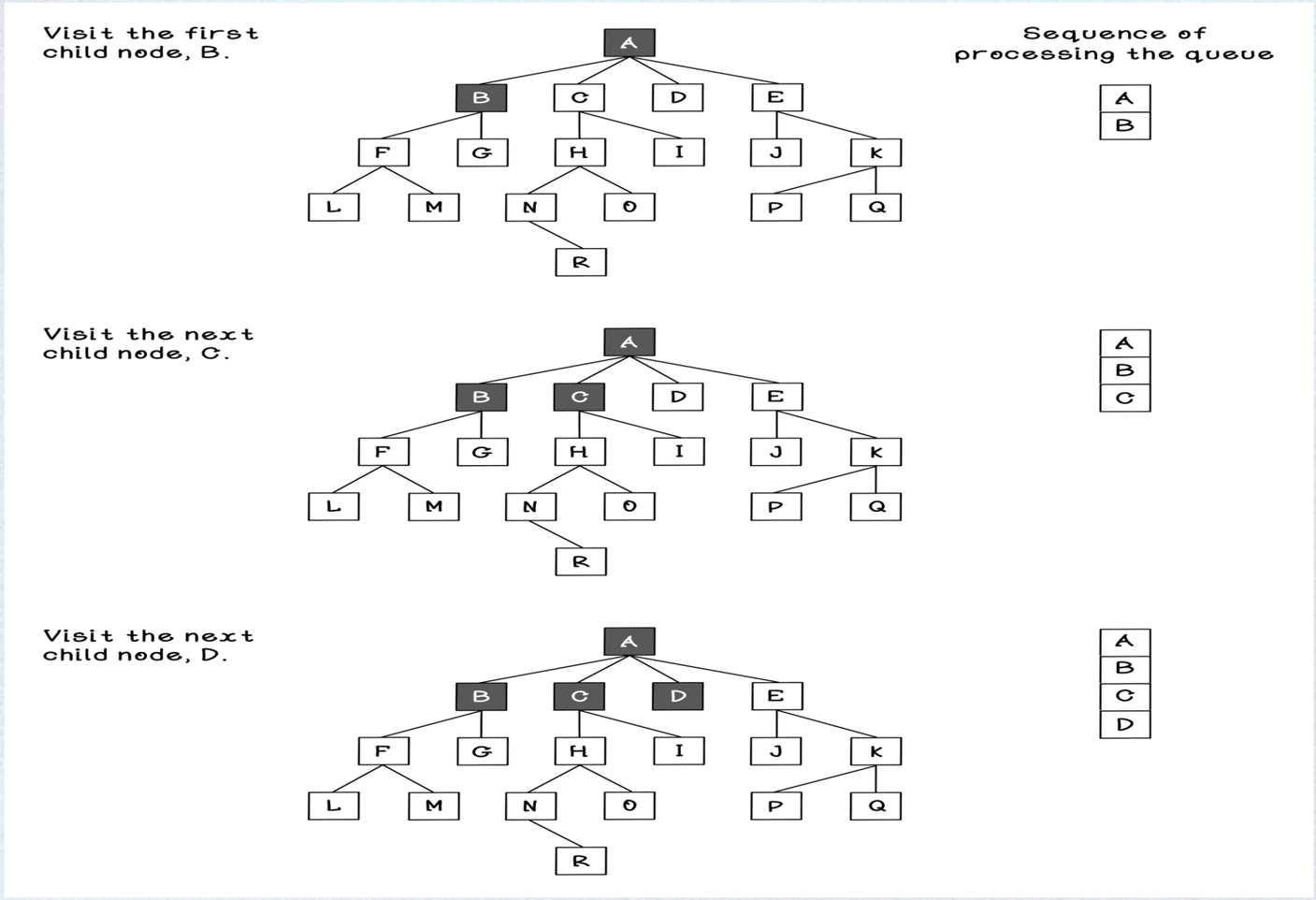
1. Breadth-first search (BFS)

Flow Of The BFS Algorithm



1. Breadth-first search (BFS)

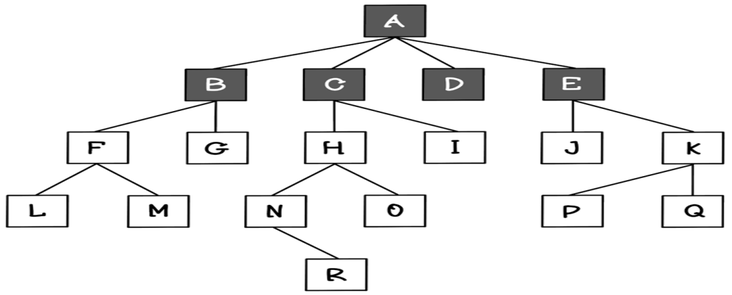
The sequence of tree processing using breadth-first search



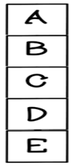
1. Breadth-first search (BFS)

The sequence of tree processing using breadth-first search

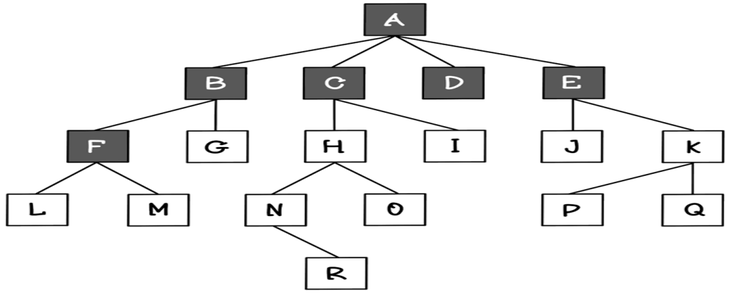
Visit the final node on this depth, E.

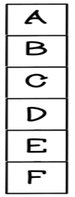


Sequence of processing the queue

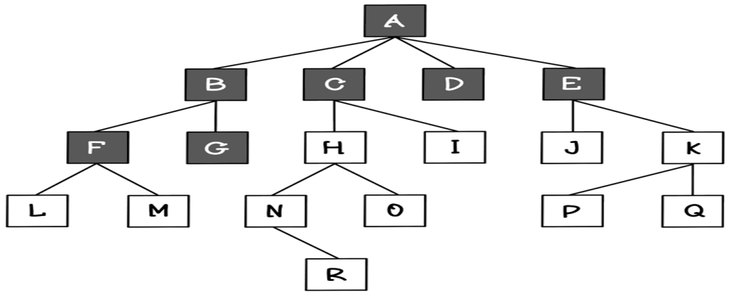


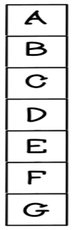
Visit the first child of the first neighbor of A. This is F, first child of B.





Visit the next child of B. This is G.

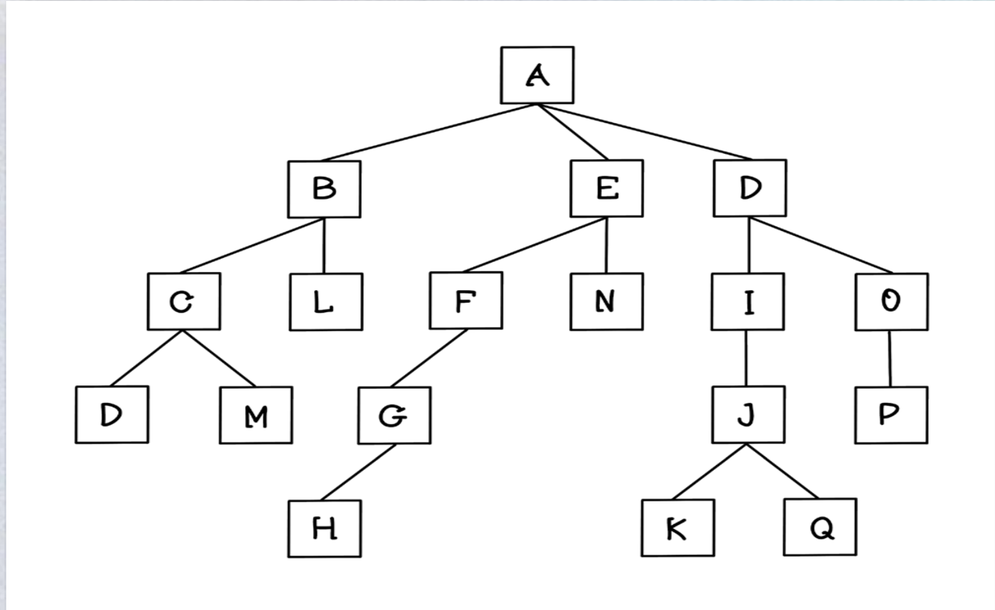




1. Breadth-first search (BFS)

Exercise: Determine the path to the solution

- What would be the order of visits using breadth-first search for the following tree?

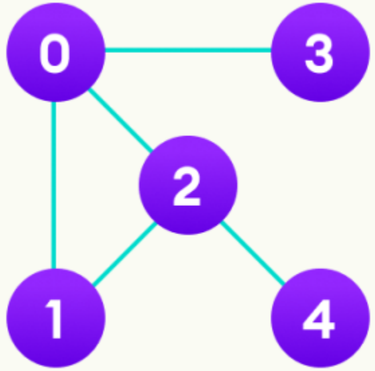


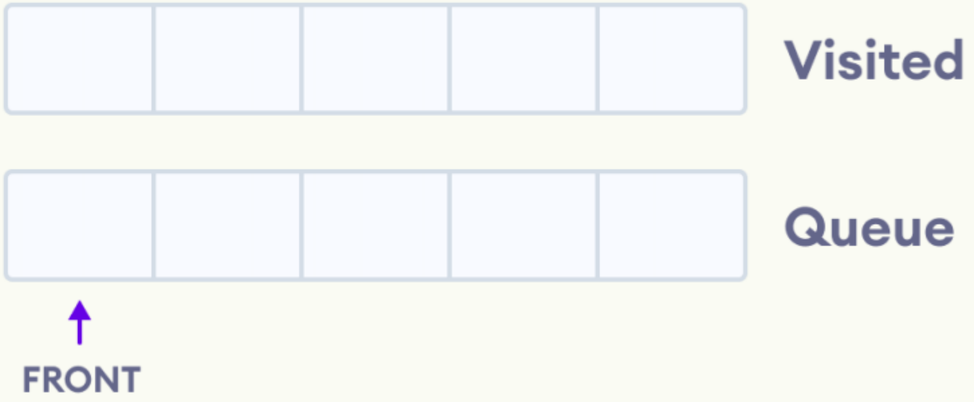
Breadth-first search order:
A, B, E, D, C, L, F, N, I, O, D, M, G, J, P, H, K, Q

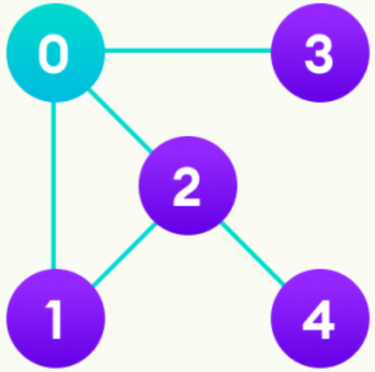
1. Breadth-first search (BFS)

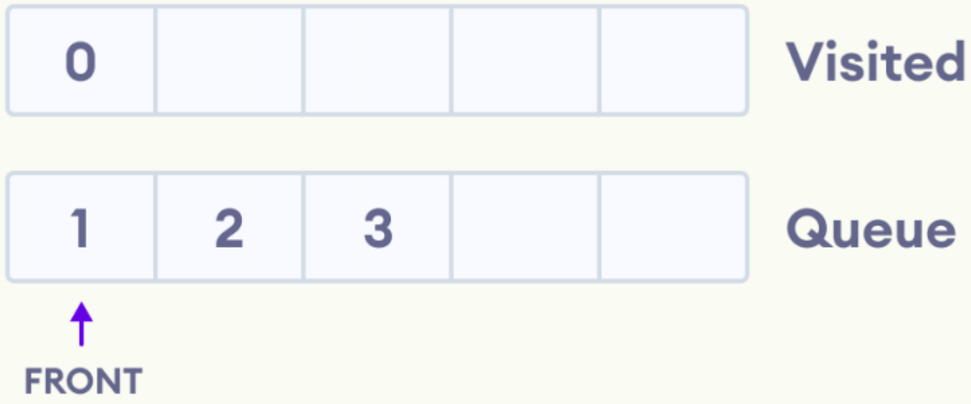
Example

The **BFS** starting from node **0**



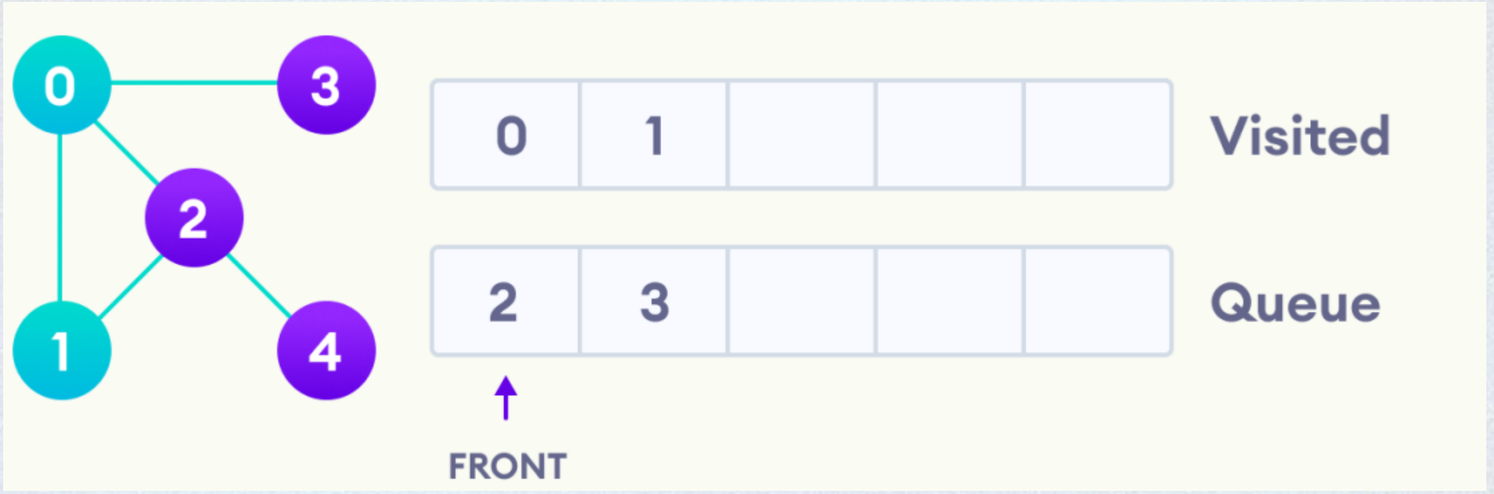


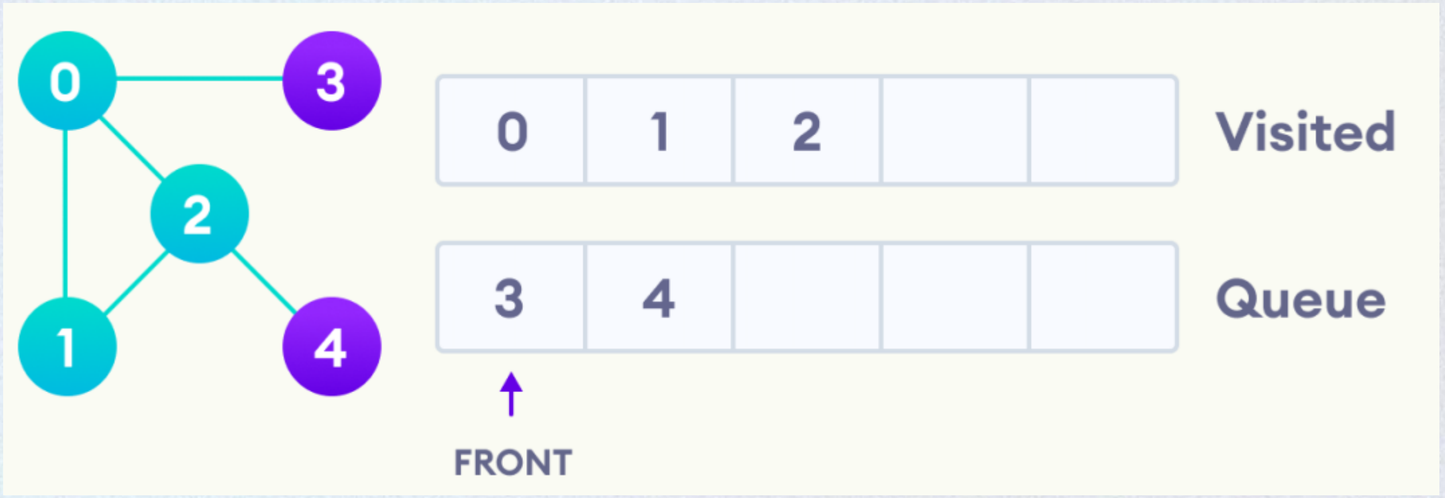




1. Breadth-first search (BFS)

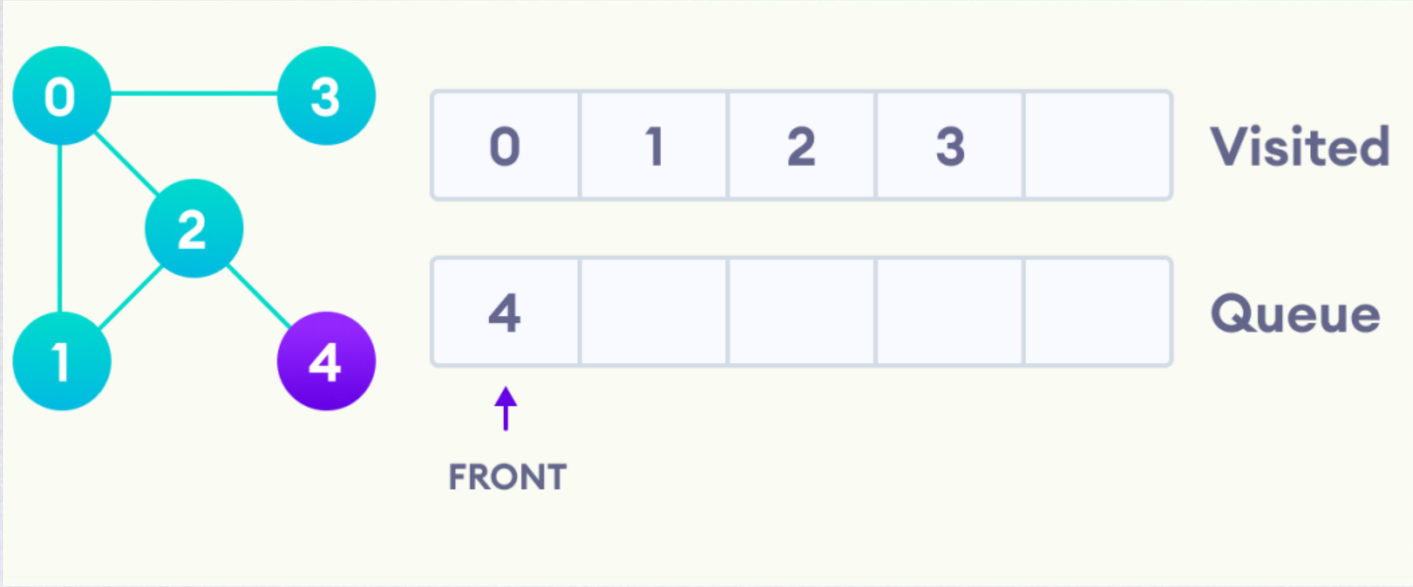
Example

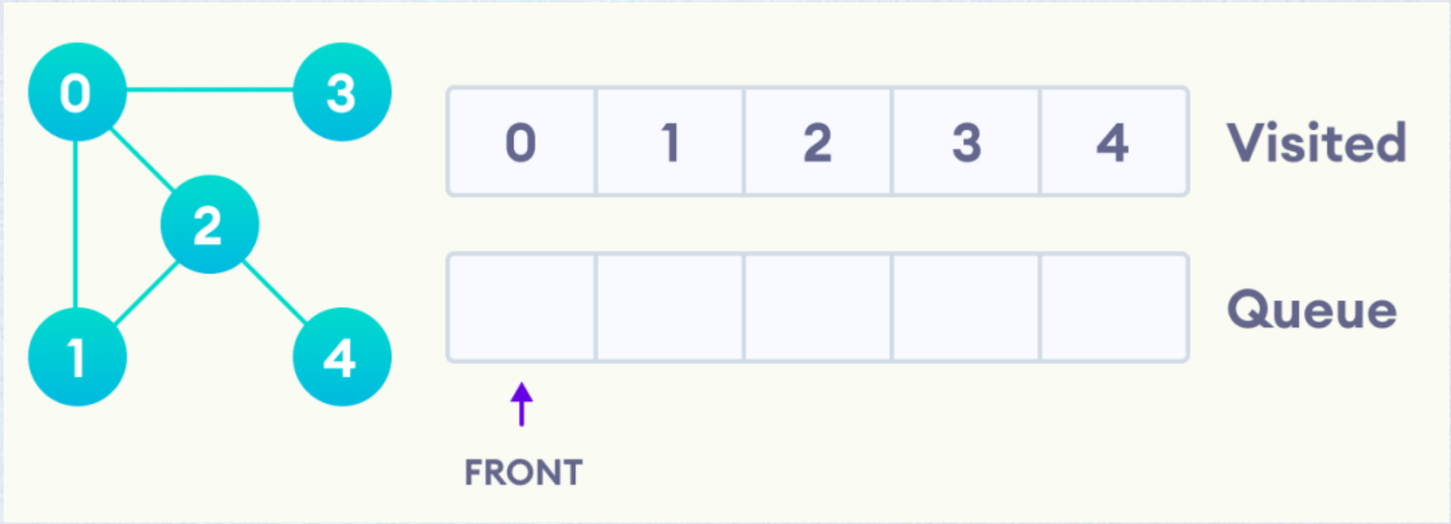




1. Breadth-first search (BFS)

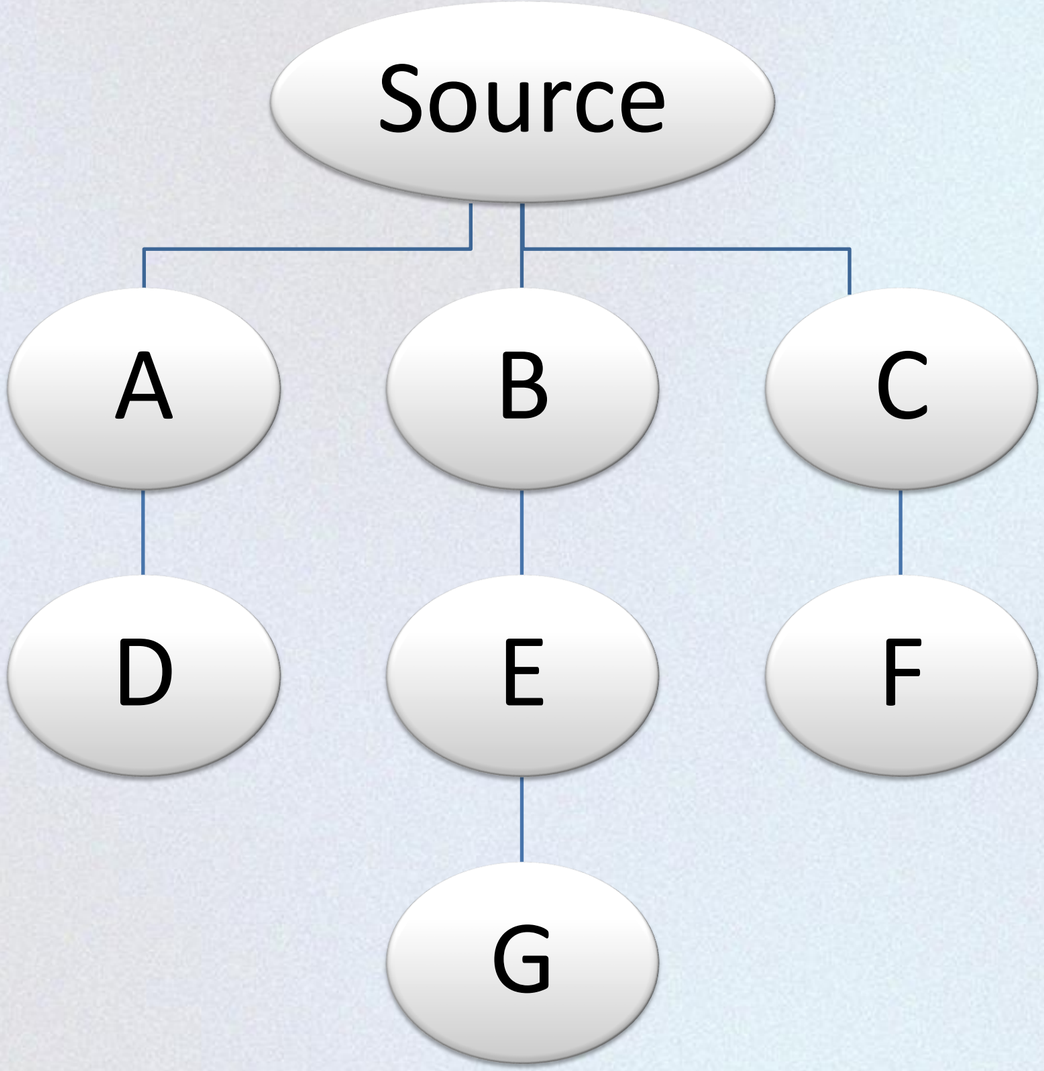
Example





1. Breadth-first search (BFS)

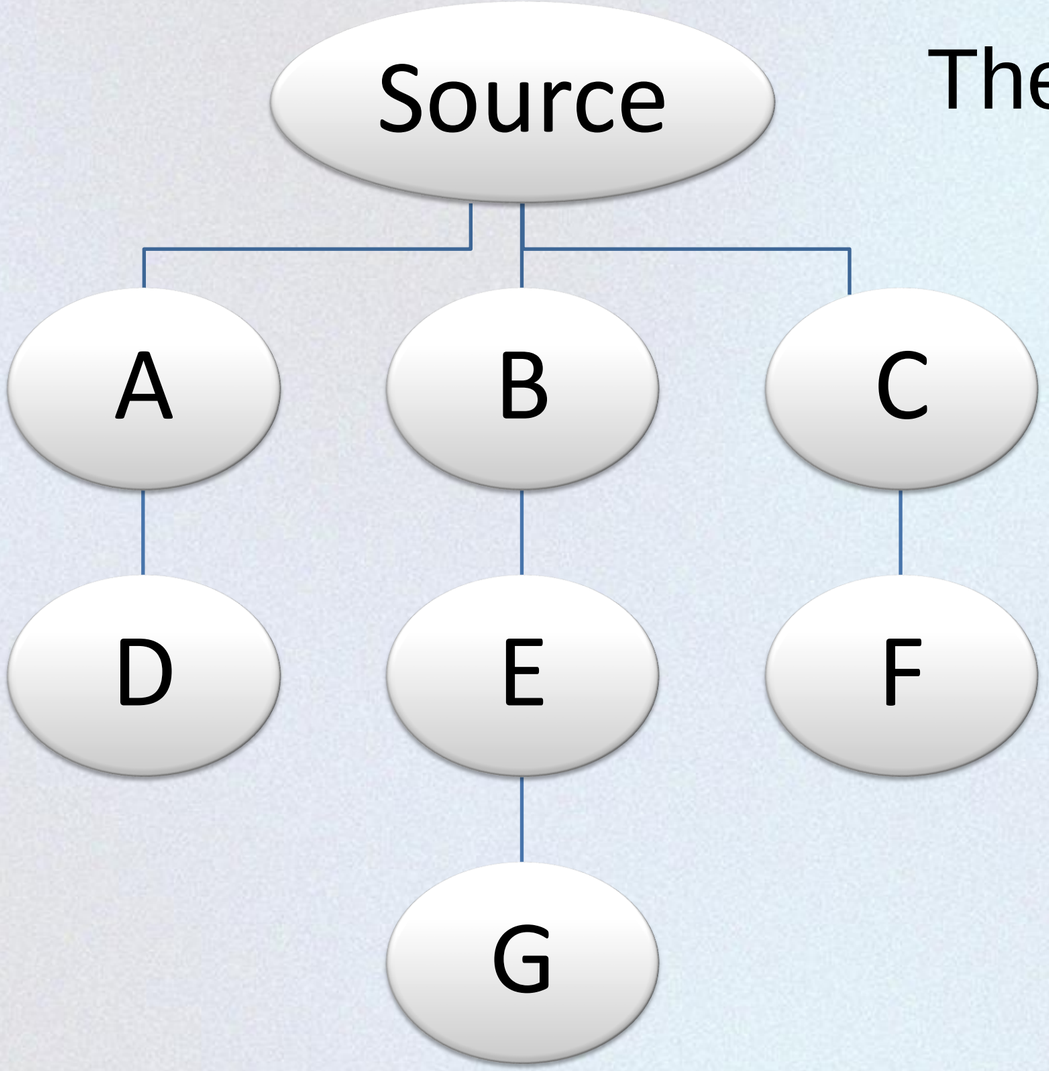
Example1: Where F is Destination The **BFS** starting from node **source**



1. Start at node Source:
   Queue: [Source]
   Visit Source, and enqueue its neighbors A, B and C.
   Queue becomes: [A, B, C]
2. Visit node A (first element in the queue):
   Queue: [B, C]
3. Visit node B (first element in the queue):
   Queue: [C]
4. Visit C, and enqueue its neighbors D, E and F.
   Queue becomes: [D, E, F]
5. Visit node D: (no neighbors to enqueue).
   Queue becomes: [E, F]
6. Visit node E:
   Queue becomes: [G, F]
7. Visit node f: this goal
   Queue becomes: [G]

1. Breadth-first search (BFS)

Example1: Where F is Destination



The **BFS** starting from node **source**

Visited Nodes:

Source	A	B	C	D	E	F

Queue

G		

1. Breadth-first search (BFS)

Advantages of BFS:

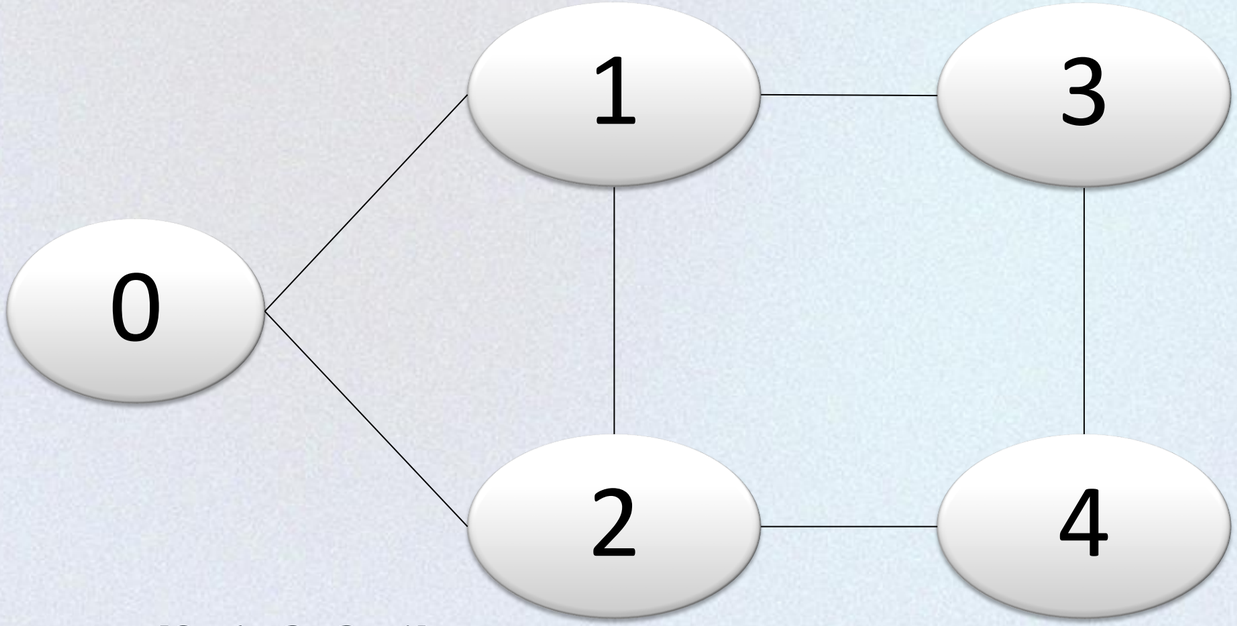
- Guarantees the shortest path in an unweighted graph.
- Suitable for searching in levels or layers.

Disadvantages of BFS:

- Can consume a lot of memory, especially for large graphs, because it uses a queue to store all the nodes at a given level.
- Not ideal for very deep trees or graphs.

1. Breadth-first search (BFS)

Example 2: *Input: adj = [[1,2], [0,2,3], [0,4], [1,4], [2,3]]*




```
Nodes adjacent to 0 is 1,2
Nodes adjacent to 1 is 0,2,3
Nodes adjacent to 2 is 0,4
Nodes adjacent to 3 is 1,4
Nodes adjacent to 4 is 2,3
```

Output: *[0, 1, 2, 3, 4]*

The BFS *Starting from 0, the BFS traversal will follow these steps:*

Visit 0 → Output: [0]

Visit 1 (first neighbor of 0) → Output: [0, 1]

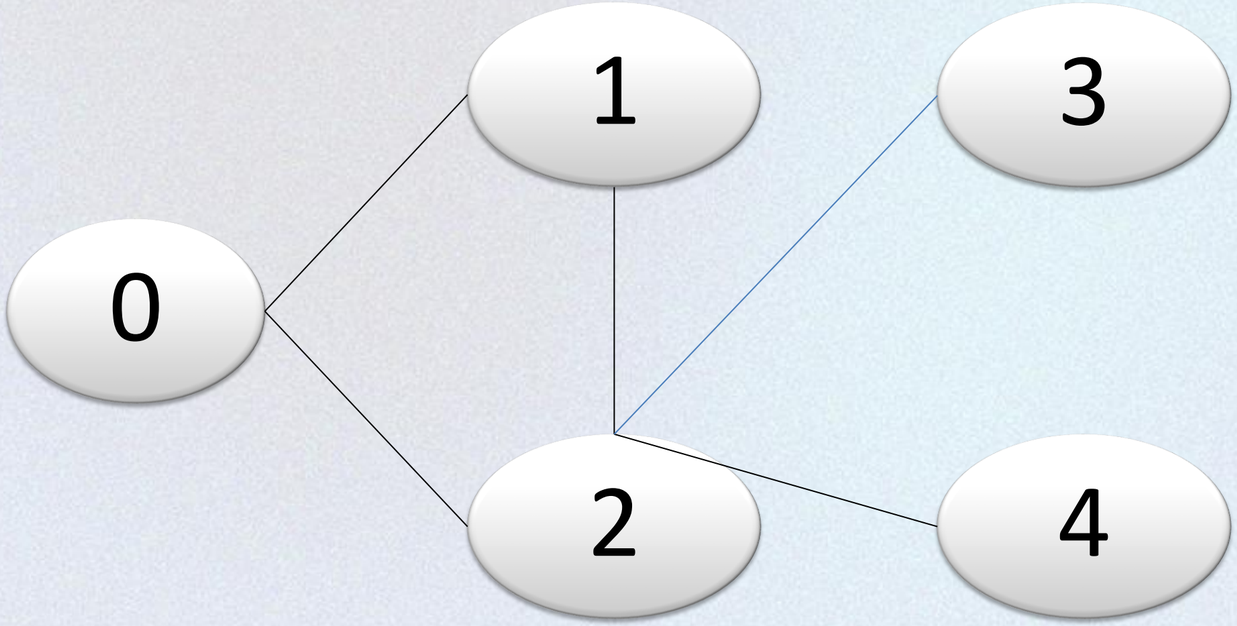
Visit 2 (next neighbor of 0) → Output: [0, 1, 2]

Visit 3 (next neighbor of 1) → Output: [0, 1, 2,3]

Visit 4 (neighbor of 2) → Final Output: [0, 1, 2, 3, 4]

1. Breadth-first search (BFS)

Example 2: *Input: adj = [[1, 2], [0, 2], [0, 1, 3, 4], [2], [2]]*




```
Nodes adjacent to 0 is 1,2
Nodes adjacent to 1 is 0,2
Nodes adjacent to 2 is 0,1,3,4
Nodes adjacent to 3 is 2
Nodes adjacent to 4 is 2
```

Output: *[0, 1, 2, 3, 4]*

The BFS *Starting from 0, the BFS traversal will follow these steps:*

Visit 0 → Output: [0]

Visit 1 (first neighbor of 0) → Output: [0, 1]

Visit 2 (next neighbor of 0) → Output: [0, 1, 2]

Visit 3 (the first neighbor of 2 that hasn't been visited yet) → Output: [0, 1, 2,3]

Visit 4 (the next neighbor of 2) → Final Output: [0, 1, 2, 3, 4]

THANK YOU!