Introduction to AI

Lecture 2: Problem Solving as Search (Blind/Uninformed Strategies)

Dr. Dalia Ezzat

Assistant Professor of Information Technology

Agenda

- **Goal-based agents**
- **Problem Solving as Search**
- **Search Strategies**

Excerpts from the previous lecture

- Agent types:

  ✓ Simple reflex agents

  ✓ Model-based reflex agents

  ✓ **Goal-based agents**

  ✓ Utility-based agents

- All of which can be generalized into learning agents that can improve their performance and generate better actions.

Excerpts from the previous lecture

- Goal-based agents: problem solving agents or planning agents.

- Agents that work towards a goal.

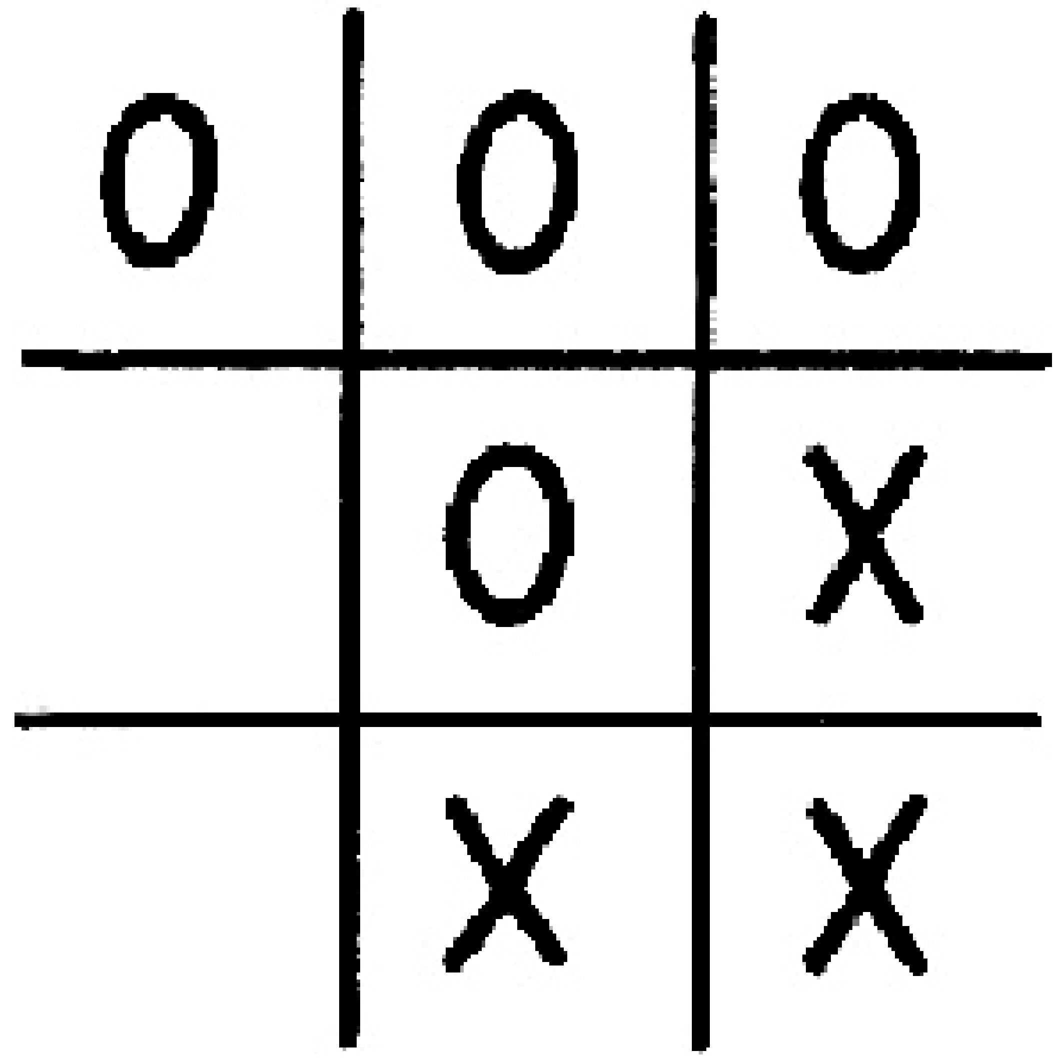
- Agents consider the impact of actions on future states.

- Agent’s job is to identify the action or series of actions that lead to the goal.

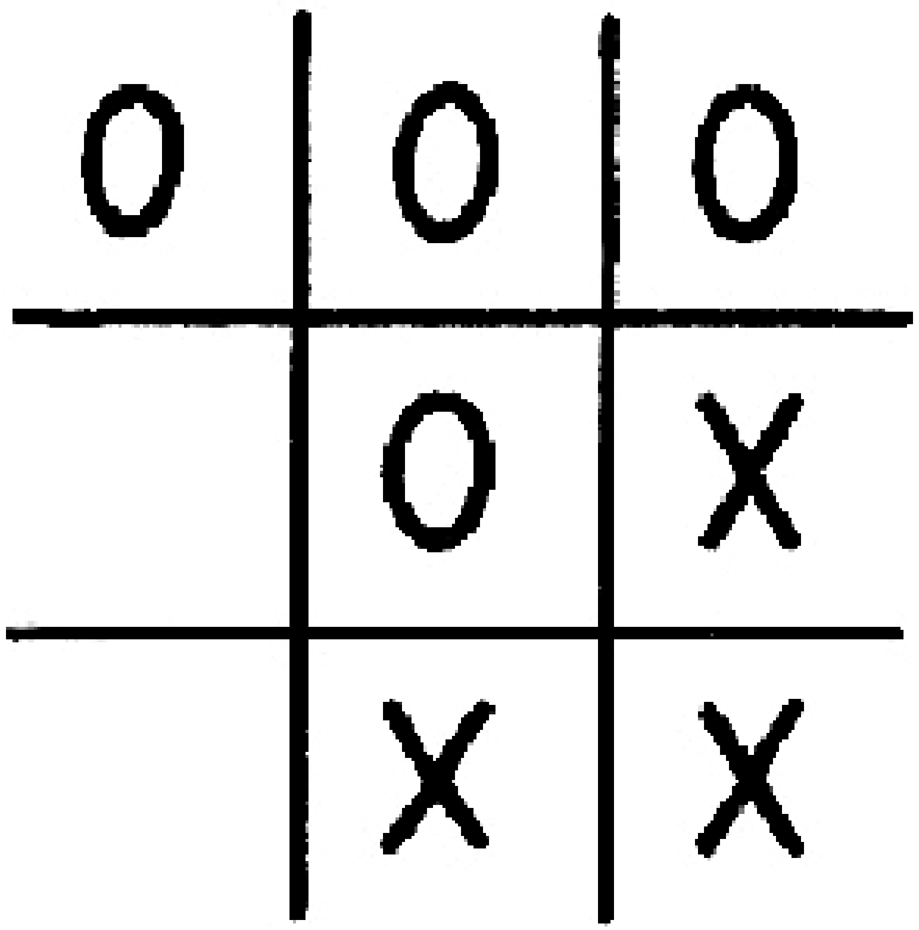
- Formalized as a search through possible solutions.

Real World Problem

Real World Problem



How to Make AI Solve Real World Problems



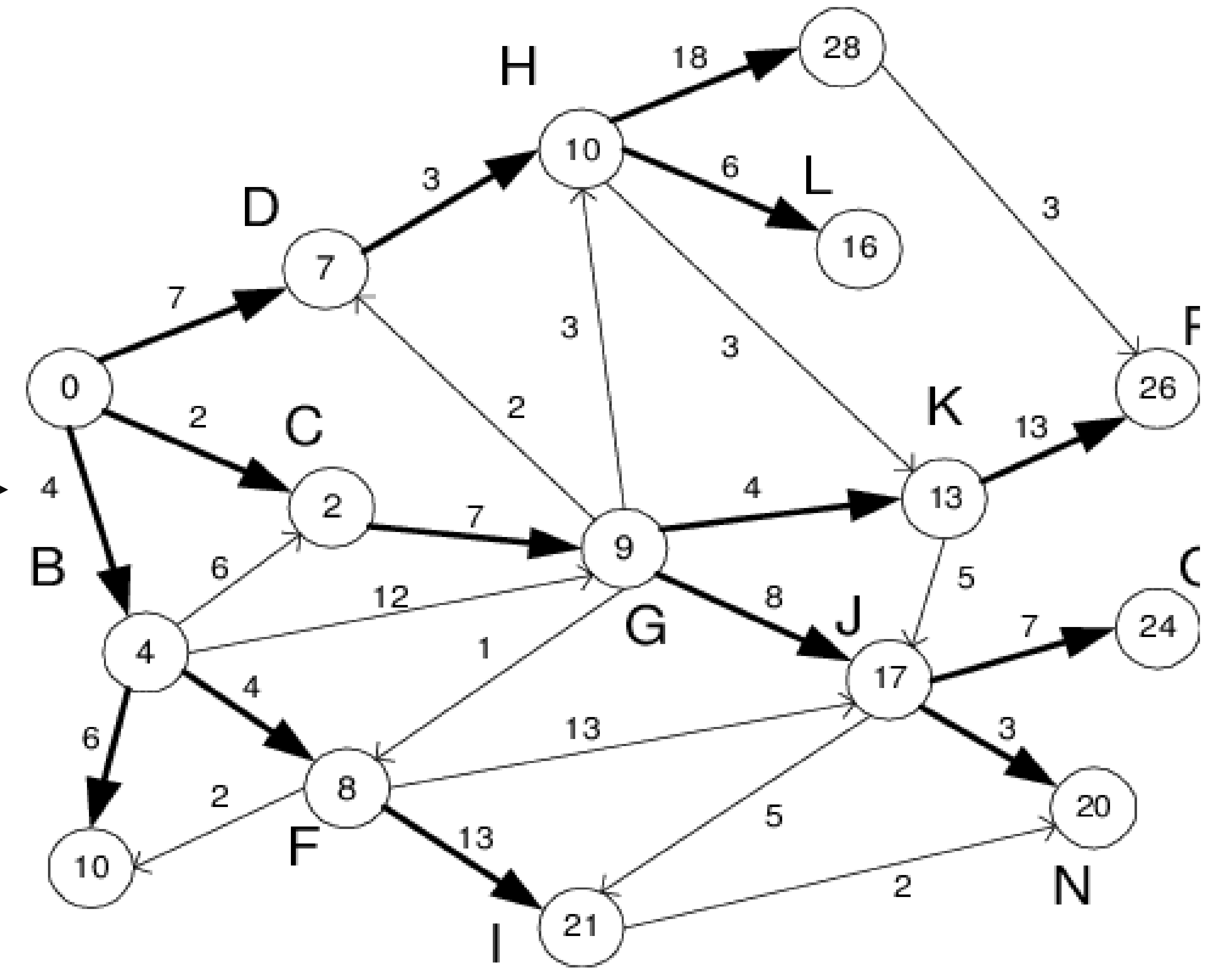
Modeling Problems

- Modeling a problem is a critical first step in solving it, especially in artificial intelligence and computer science.

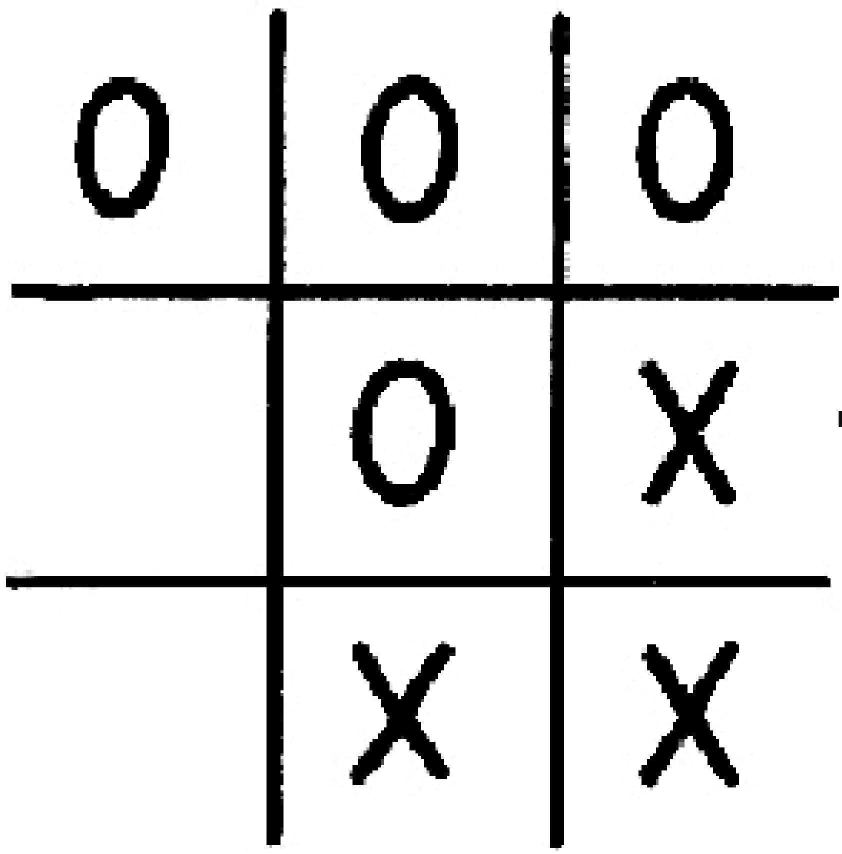
- Modeling involves abstracting the real-world problem into a structured representation that can be analyzed and solved using computational methods.

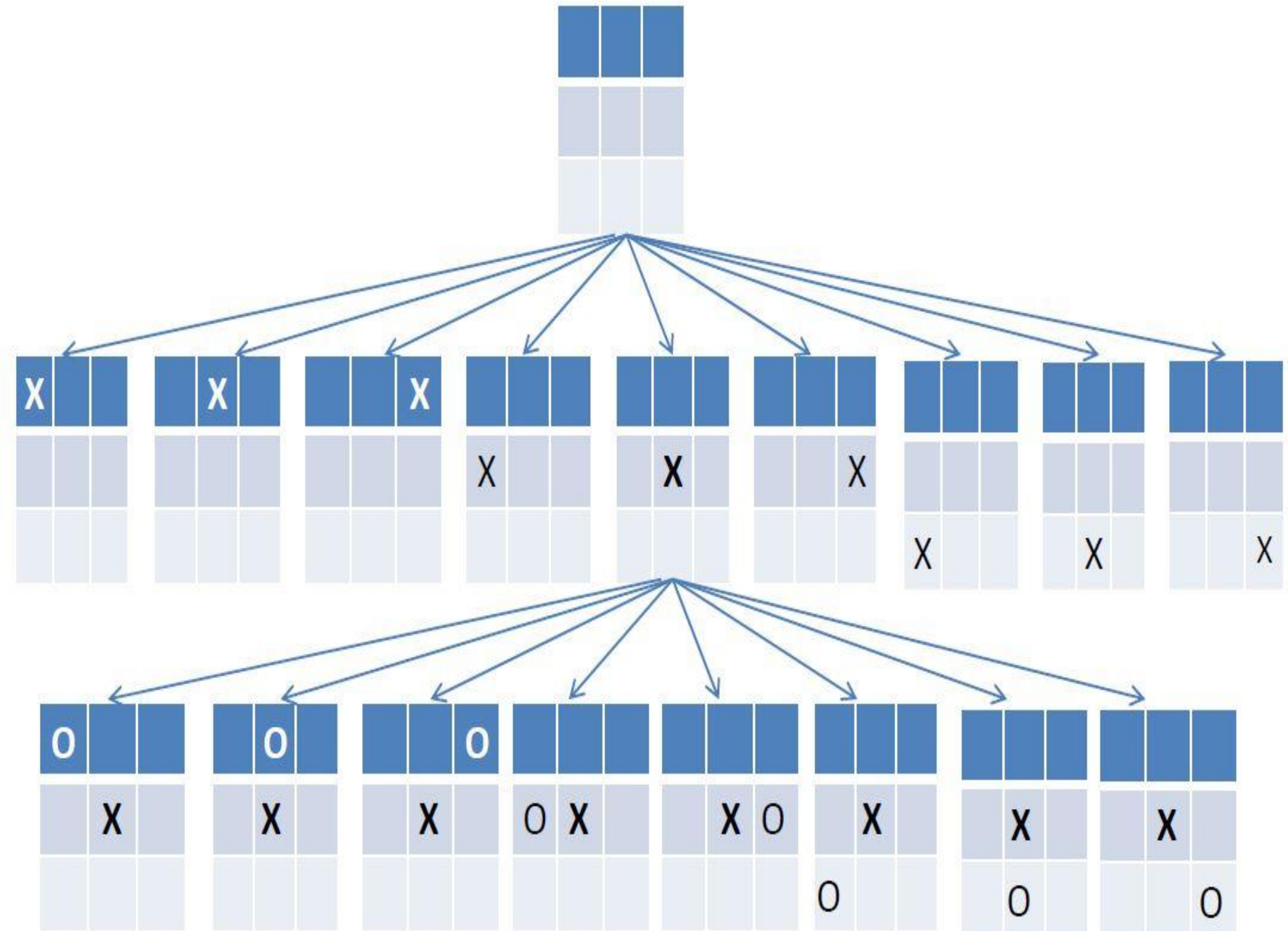
- For problems that involve searching for solutions (e.g., finding the shortest path, solving puzzles, or optimizing resources), modeling the problem as a search problem is a common and effective approach.

Modeling Problems



Modeling Problems





Problem Solving as Search

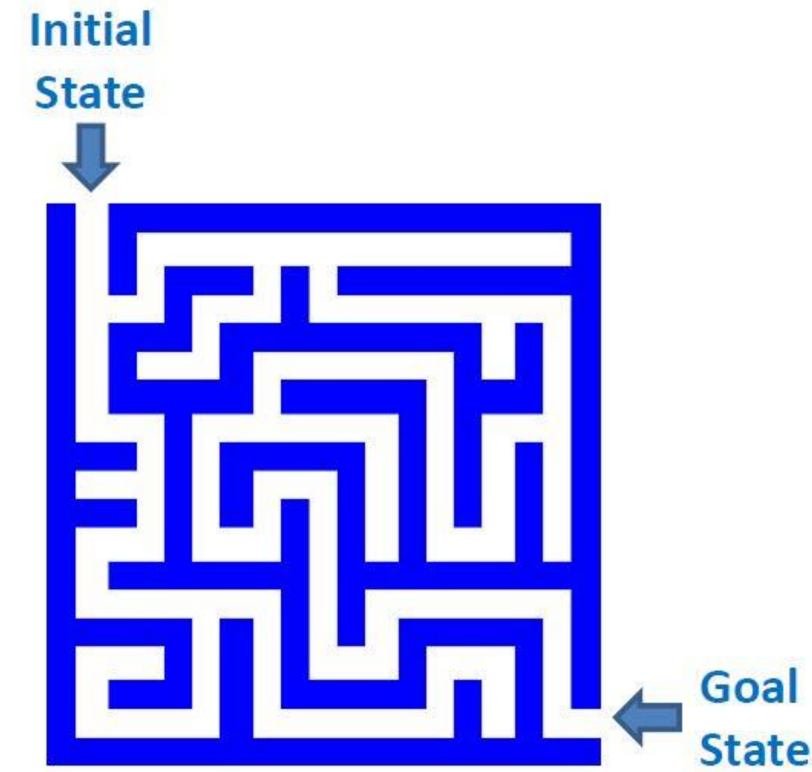
1. Define the problem through:
   - ✓ Goal formulation
   - ✓ Problem formulation
2. Solving the problem as a 2-stage process:
   - ✓ Search: exploration of several possibilities
   - ✓ Execute the solution found

Search Problem Components

Initial state: the state in which the agent starts

States: All states reachable from the initial state by any sequence of actions **(State space).**

Actions: possible actions available to the agent. At a state s, Actions(s) returns the set of actions that can be executed in state s **(Action space).**

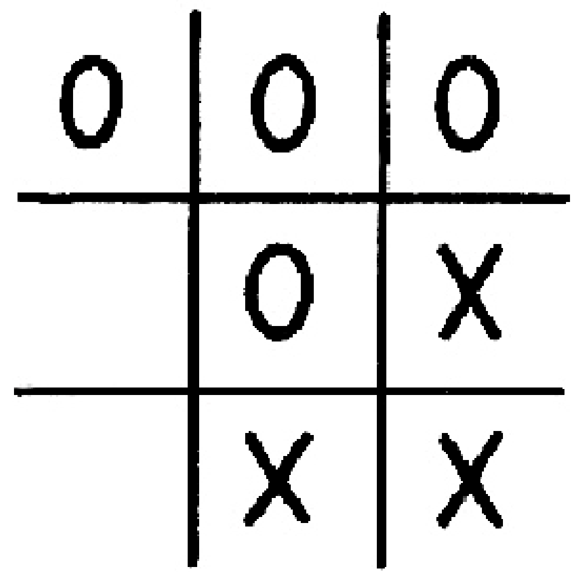


Transition model: A description of what each action does Results(s, a)

Goal test: determines if a given state is a goal state

Path cost: function that assigns a numeric cost to a path w.r.t. performance measure

Example: Tic-Tac-Toe

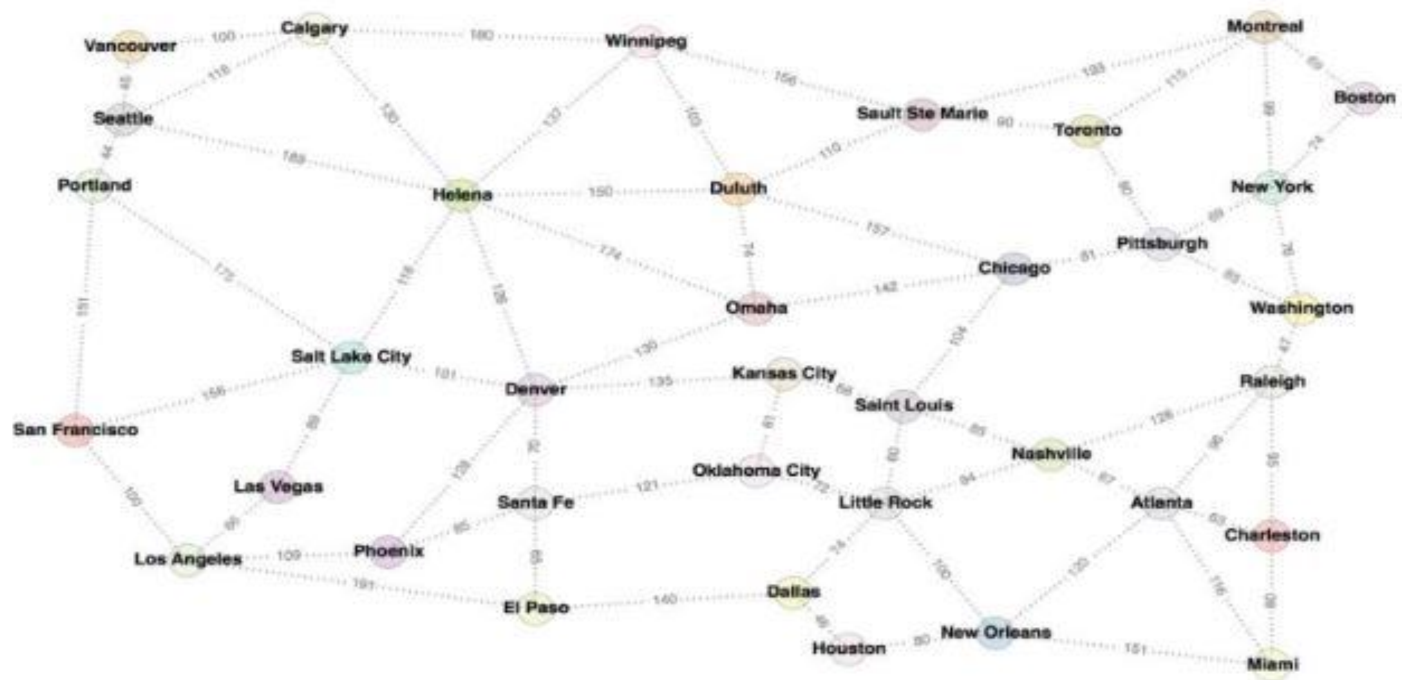


- Initial state: an empty board
- States: The state space consists of all possible configurations of the 3x3 grid, where each cell can be empty, contain an X, or contain an O.
- Actions: The actions are the possible moves a player can make. For a given state, an action is placing an X or an O in an empty cell.
- Transition model: Given a state and an action, returns resulting state
- Goal test: a board state having three Xs in a row, column, or diagonal
- Path cost: total moves, each move costs 1

Example: 8 puzzles

- Initial state: Any state
- States: Location of each of the 8 tiles in the 3x3 grid
- Actions: Move Left, Right, Up or Down
- Transition model: Given a state and an action, returns resulting state
- Goal test: state matches the goal state?
- Path cost: total moves, each move costs 1

Example: Map Search



- Initial state: The starting location on the map (for example: in Boston
- States: In City where City ∈{Los Angeles, San Francisco, Denver,...}
- Actions: Go New York, etc.
- Transition model: Results (In (Boston), Go (New York)) = In(New York)
- Goal test: In(Denver)
- Path cost: path length in kilometers

Search

- Given:
  - ✓ Initial state
  - ✓ Actions
  - ✓ Transition model
  - ✓ Goal state
  - ✓ Path cost

- How do we find the optimal solution?

Search

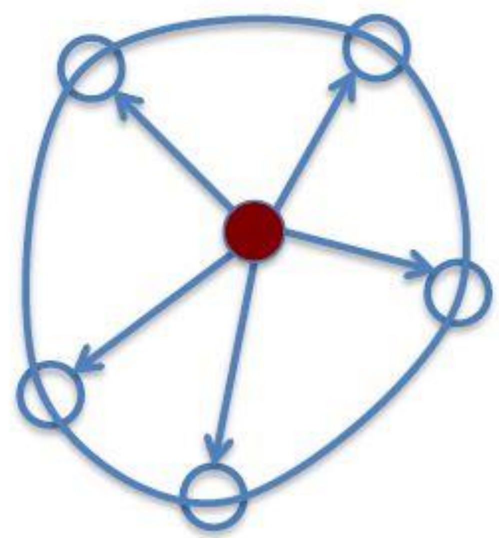
- The search space is divided into three regions:

  - ✓ Explored (Closed List, Visited Set)
  - ✓ Frontier (Open List)
  - ✓ Unexplored.

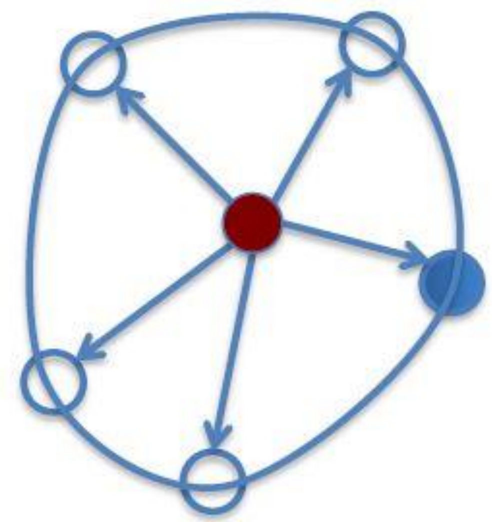
- The essence of search is moving nodes from regions (3) to (2) to (1), and the essence of search strategy is deciding the order of such moves.

Search

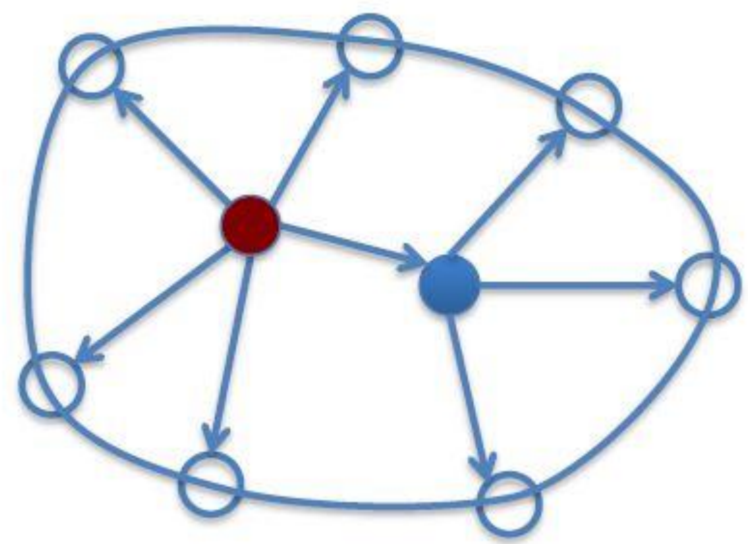
Search



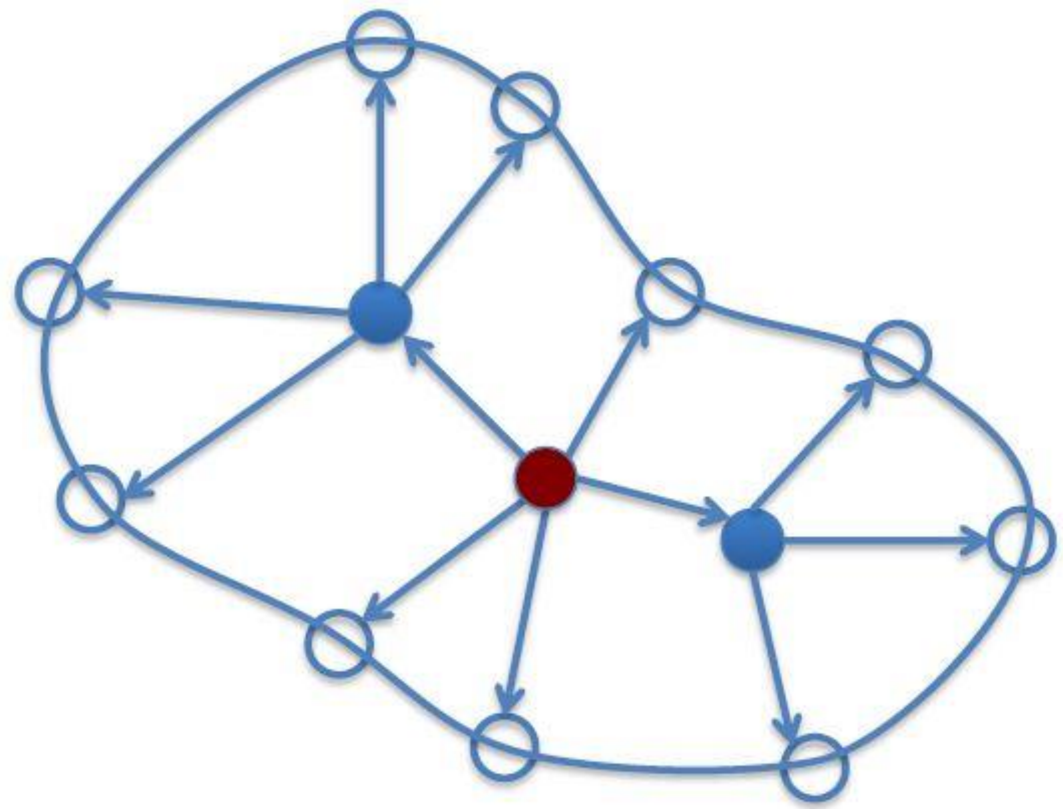
Search



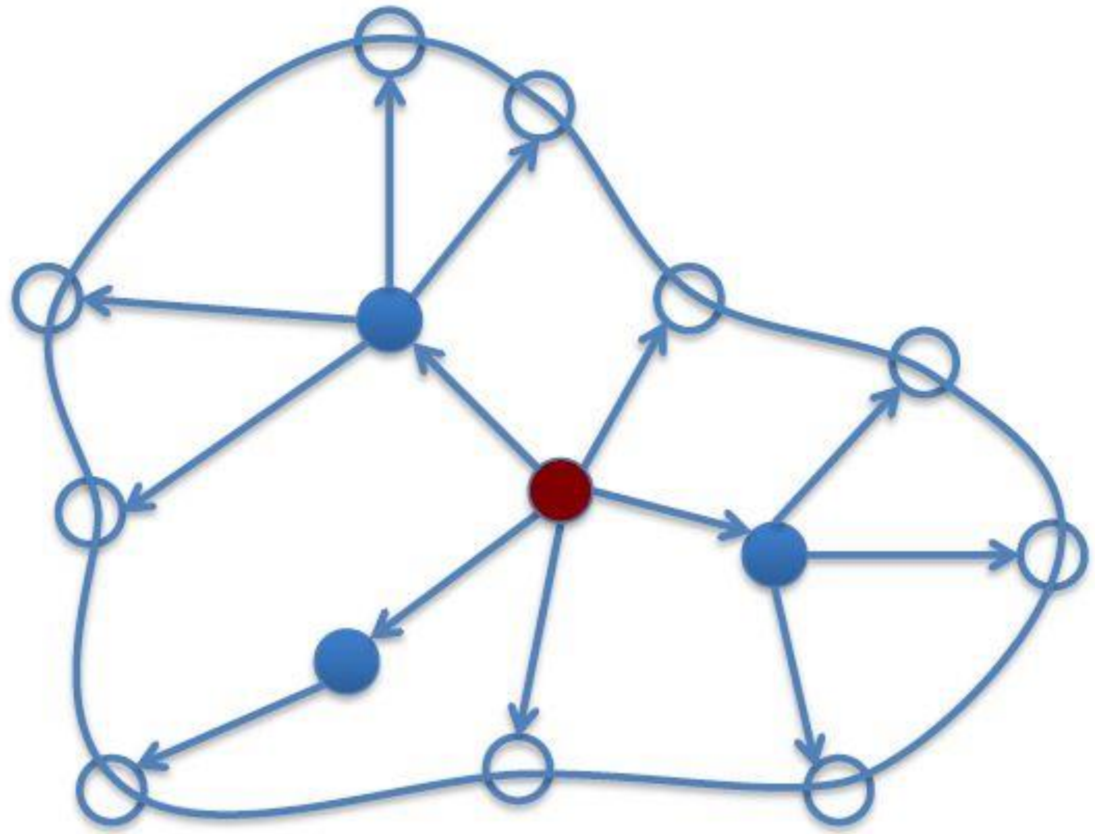
Search



Search



Search



Search

- Start with the initial state (or node) and expand it by making a list of all possible successor states.
- Maintain a frontier or a list of newly discovered nodes that have not yet been expanded.
- At each step, pick a state from the frontier to expand.
- Keep going until you reach a goal state.
- Try to expand as few states as possible.

Search

- Handling Repeated States
  - ✓ Every time you expand a node according to search strategy, add that state to the explored set do not put explored states on the frontier again
  - ✓ Every time you add a node to the frontier, check whether it already exists with a higher path cost, and if yes, replace that node with the new one

Search Strategies

- **A strategy is defined by picking the order of node expansion**

- **Strategies are evaluated along the following dimensions:**
  - ✓ **Completeness**
    - ➢ **Does it always find a solution if one exists?**
  - ✓ **Time complexity**
    - ➢ **Number of nodes generated/expanded**
  - ✓ **Space complexity**
    - ➢ **Maximum number of nodes in memory**
  - ✓ **Optimality**
    - ➢ **Does it always find a least-cost solution?**

Search Strategies

There are two kinds of search

- **Uniformed (Blind, Exhaustive or Brute Force) search**

- **Informed (or Heuristic) search**

Uniformed Search

- It is a type of search algorithm used in artificial intelligence to solve problems where the agent has no additional information about the problem other than its definition.

- It explores the state space systematically without any preference for paths that are more likely to lead to the goal.

Informed Search

- It is a type of search algorithm in artificial intelligence that uses problem-specific knowledge or heuristics to guide the search process.

- Informed search algorithms use heuristic functions to estimate the cost or potential of reaching the goal from a given state. This allows them to prioritize paths that are more likely to lead to the goal, making them more efficient for many problems.

Today Uninformed

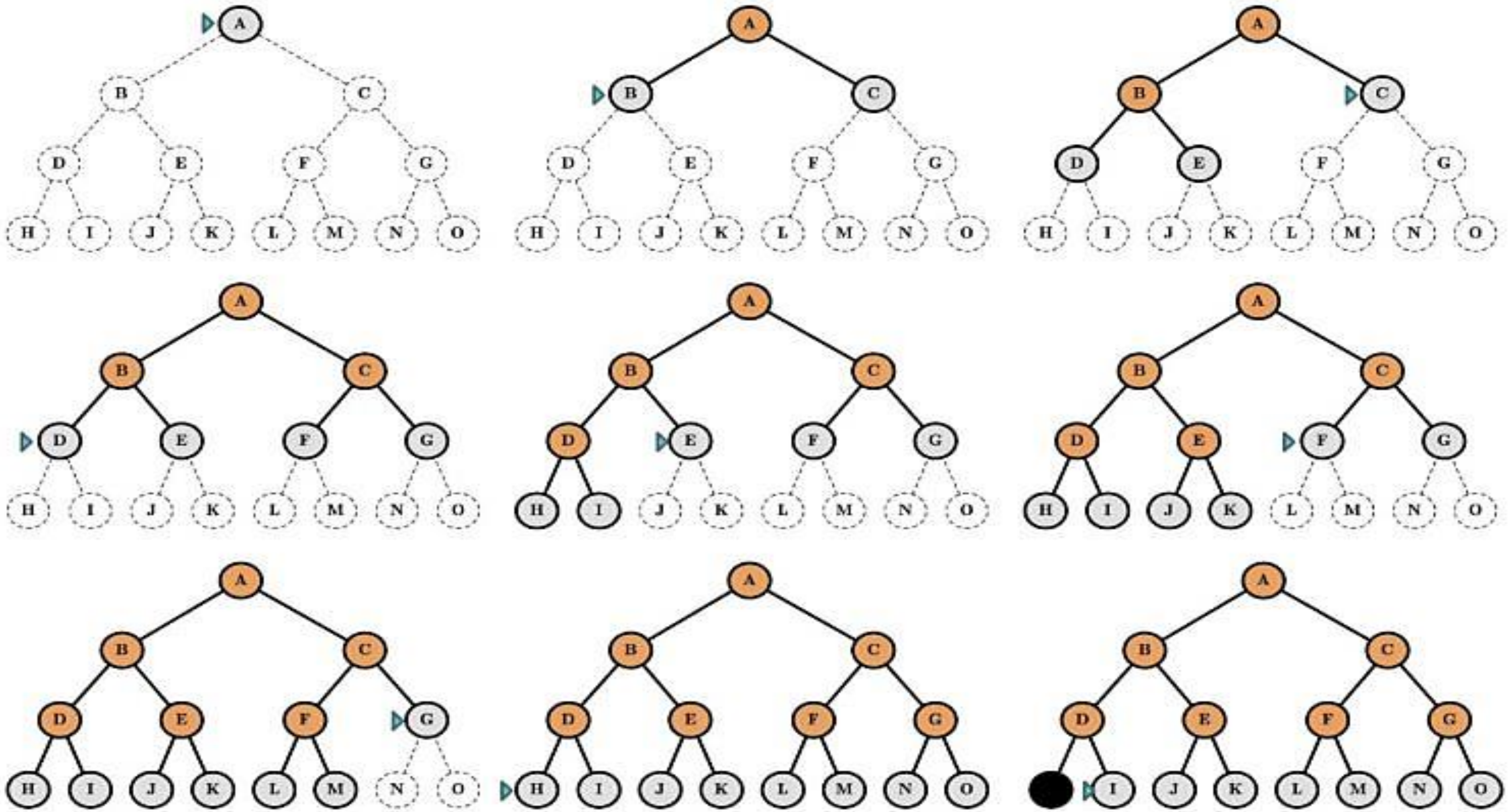
Search will be covered

Uniformed Search Algorithms

- There are several common blind search algorithms, each with its own strategy for exploring the state space:

  - ✓ Breadth-First Search (BFS)
  - ✓ Depth-First Search (DFS)
  - ✓ Depth-Limited Search (DLS)
  - ✓ Uniform-Cost Search (UCS)

Breadth-First Search (BFS)

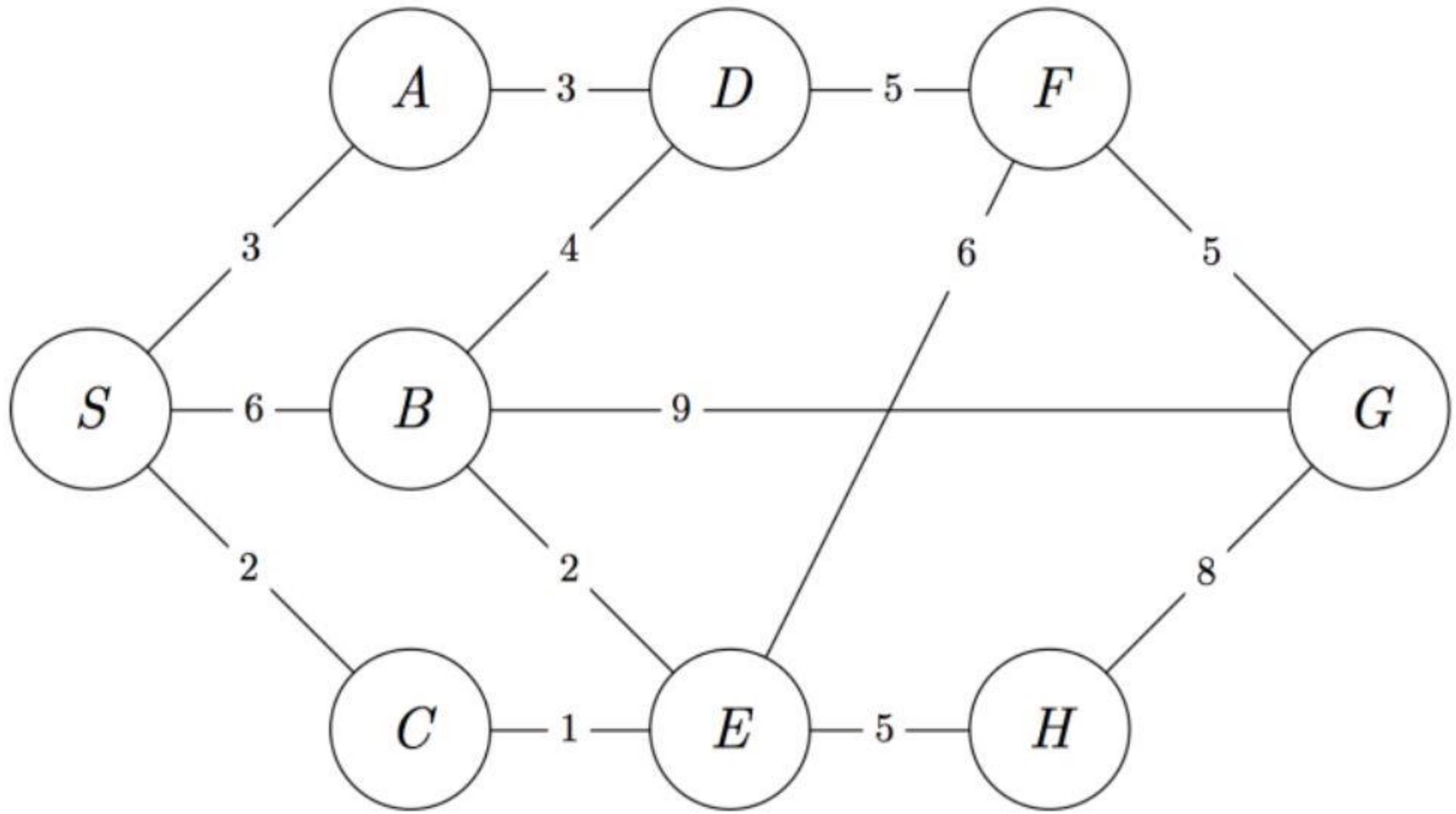


New nodes are inserted at the end of the FRINGE: First input first output FIFO (Queue)

Breadth-First Search (BFS)

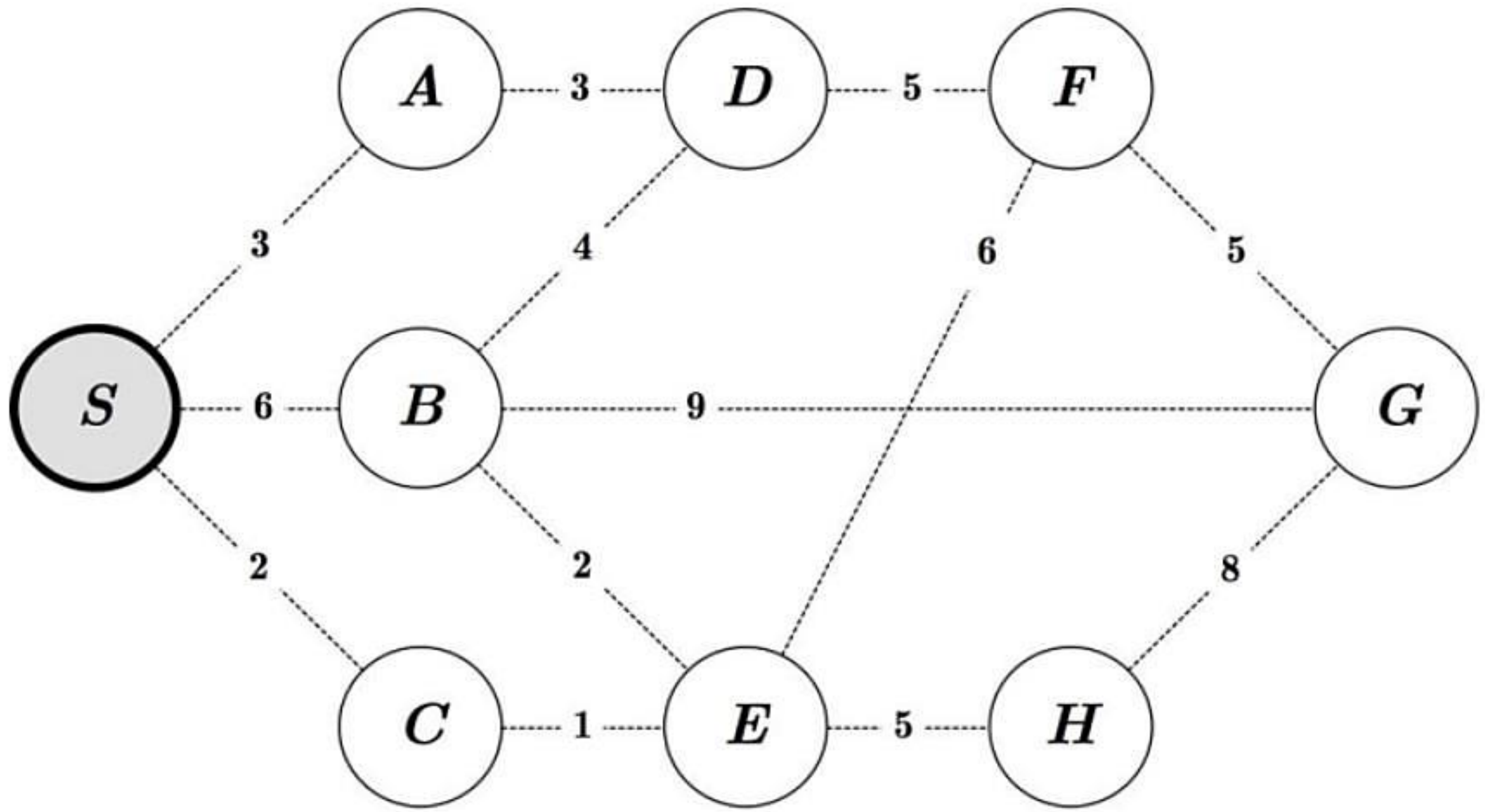
- Avoiding Repeated States:
  - ✓ Requires comparing state descriptions.
  - ✓ For example, Keep track of all generated states. If the state of a new node already exists, then discard the node.

Breadth-First Search (BFS)- Example



Question: What is the order of visits of the nodes and the path returned by BFS? Start node S and goal node G

Breadth-First Search (BFS)- Example

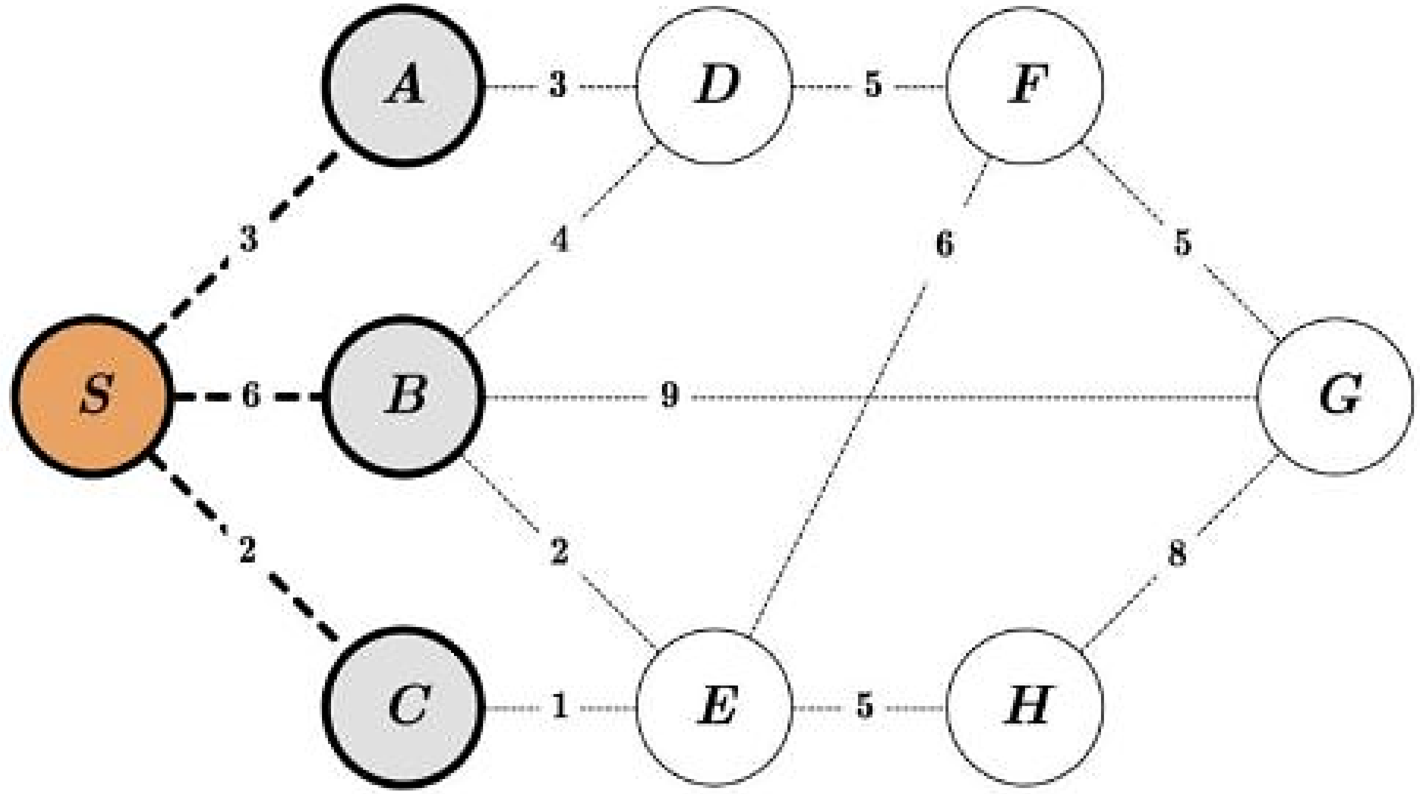


Queue:

S

Order of Visit:

Breadth-First Search (BFS)- Example



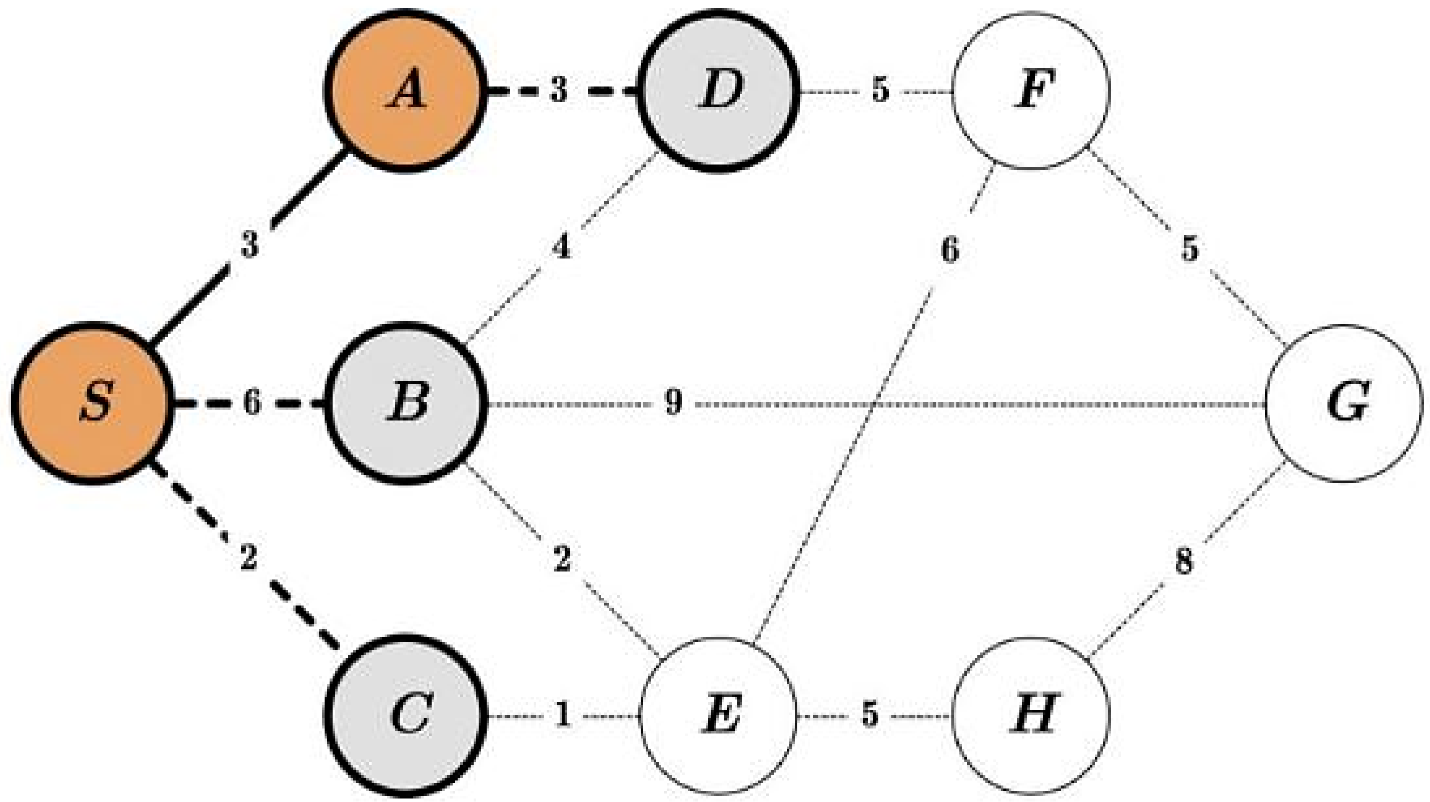
Queue:

S	A	B	C

Order of Visit:

S

Breadth-First Search (BFS)- Example



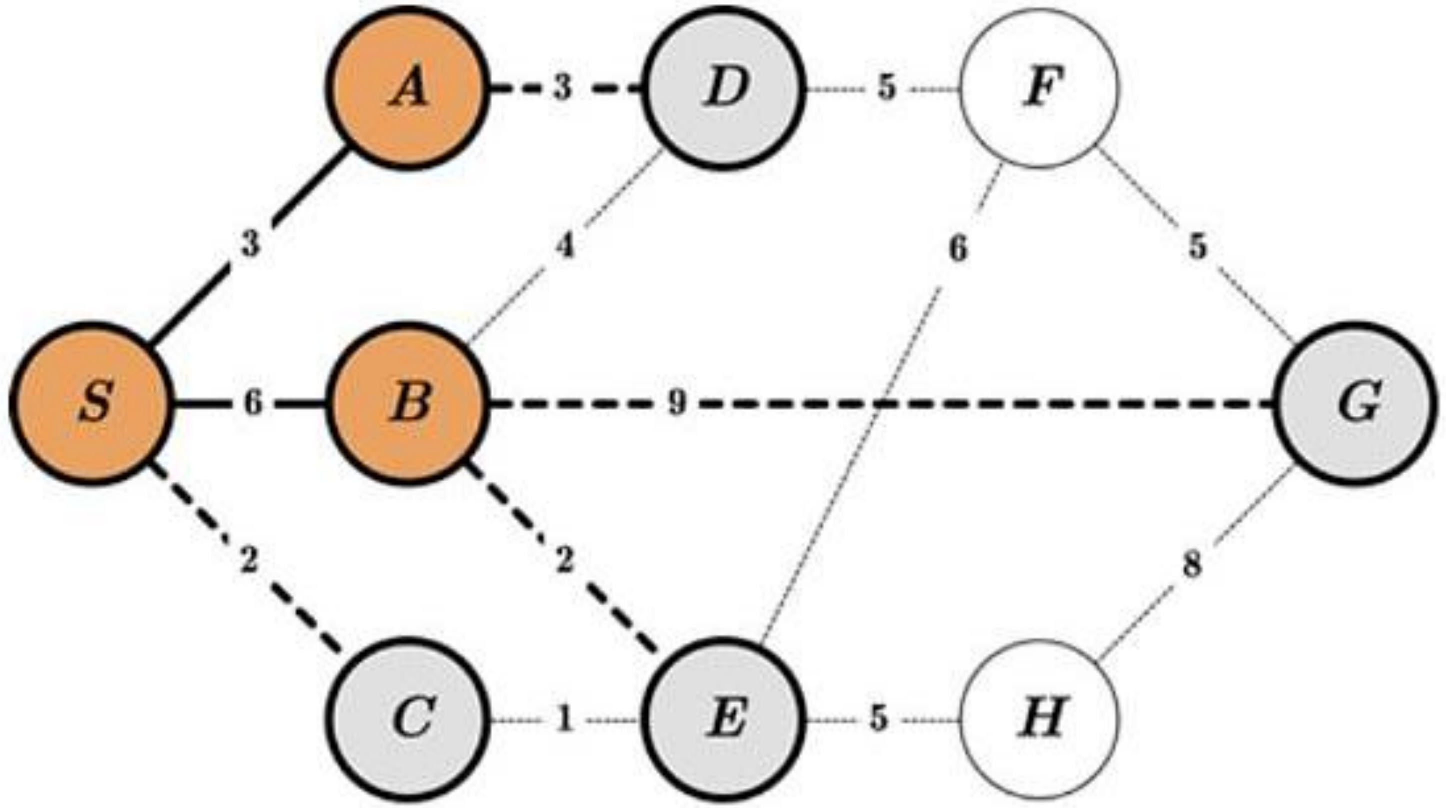
Queue:

S	*A*	*B*	*C*	*D*

Order of Visit:

S *A*

Breadth-First Search (BFS)- Example



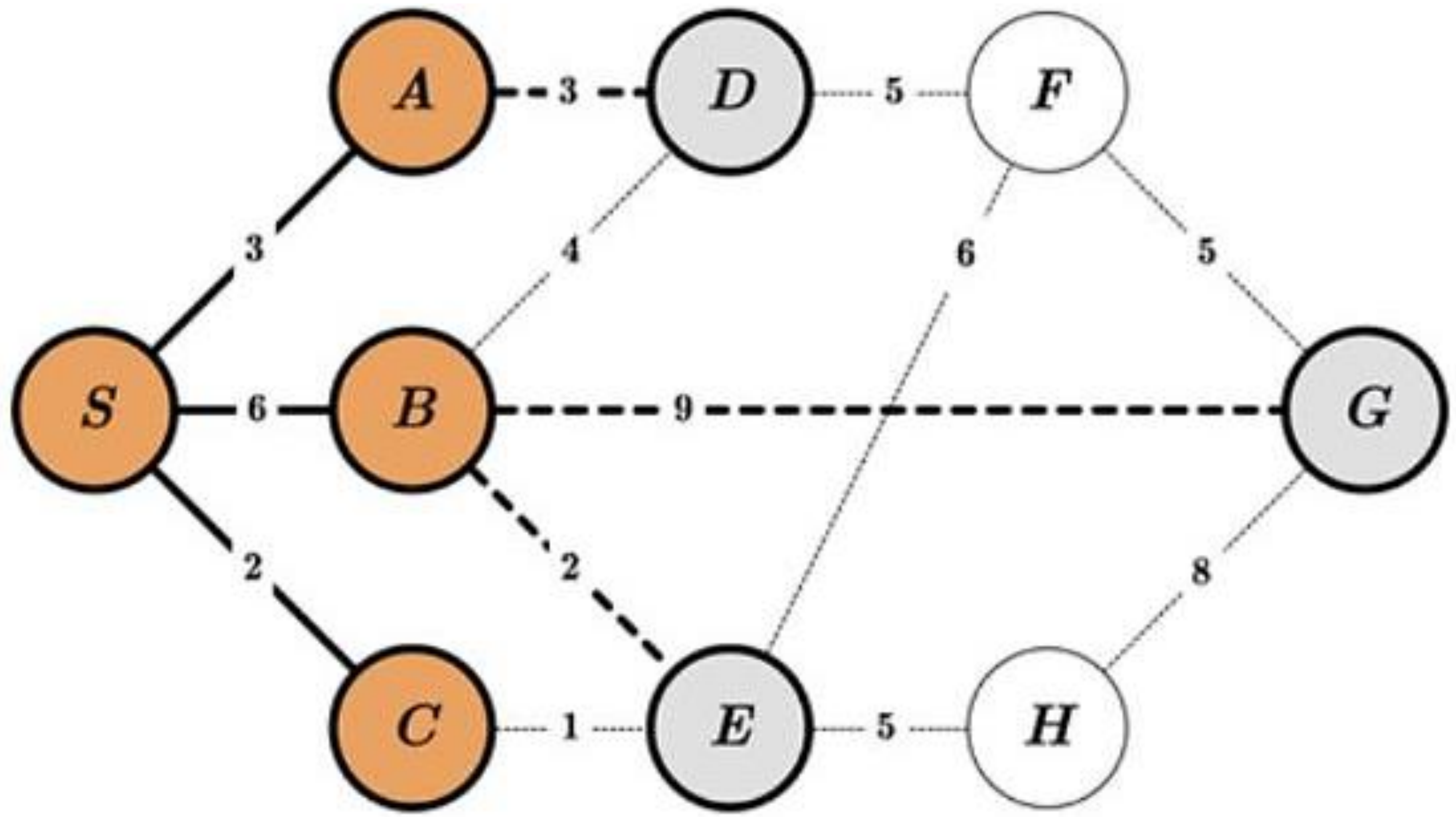
Queue:

S	*A*	*B*	*C*	*D*	G	E

Order of Visit:

S *A* *B*

Breadth-First Search (BFS)- Example



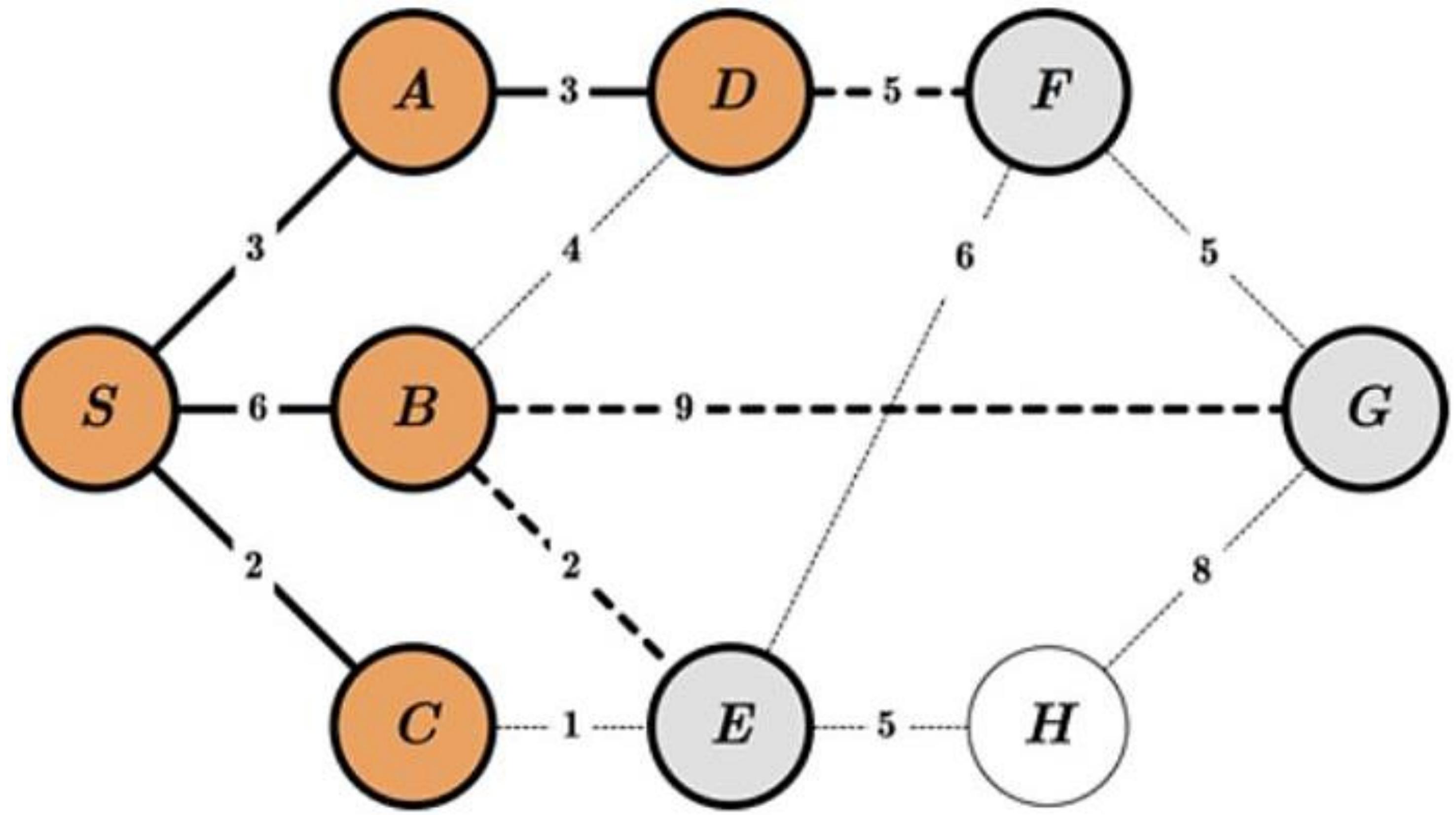
Queue:

S	*A*	*B*	*C*	*D*	G	E

Order of Visit:

S *A* *B* *C*

Breadth-First Search (BFS)- Example



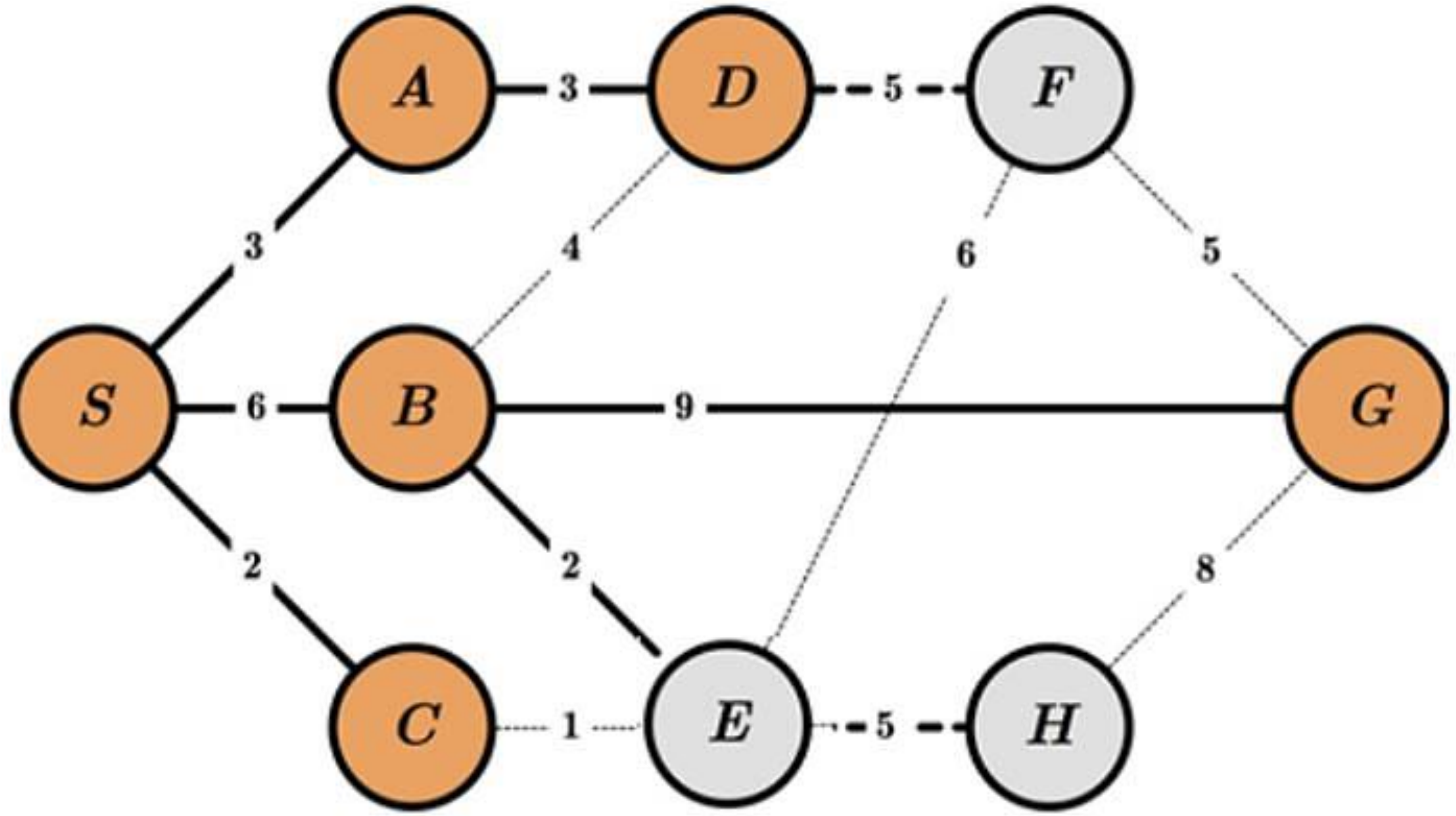
Queue:

S	*A*	*B*	*C*	*D*	G	E	*F*

Order of Visit:

S *A* *B* *C* *D*

Breadth-First Search (BFS)- Example



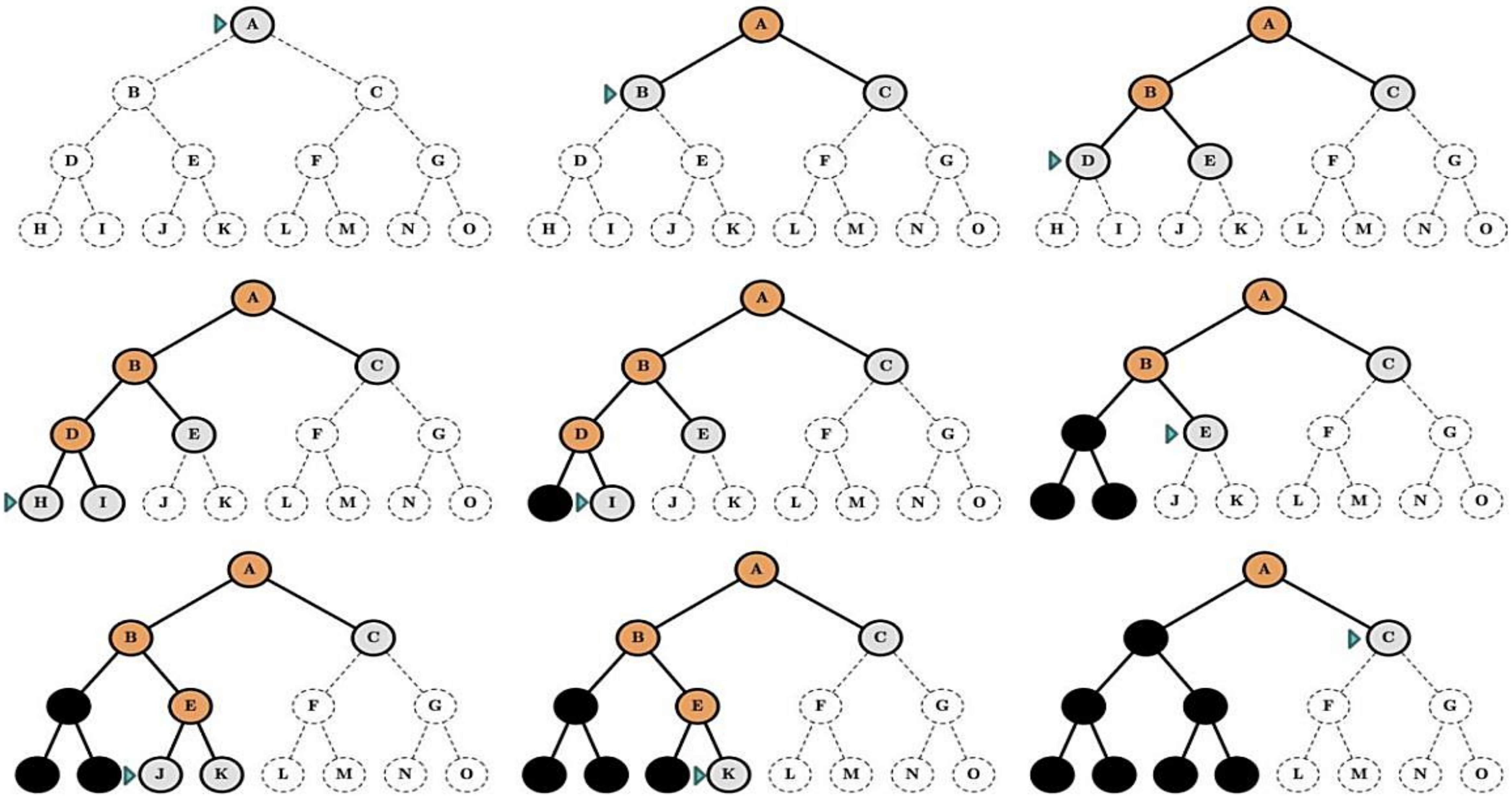
Queue:

S	*A*	*B*	*C*	*D*	*G*	*E*	*F*

Order of Visit:

S *A* *B* *C* *D* *G*

Depth-First Search (DFS)



New nodes are inserted at the front of the FRINGE: Last input first output LIFO (Stack)

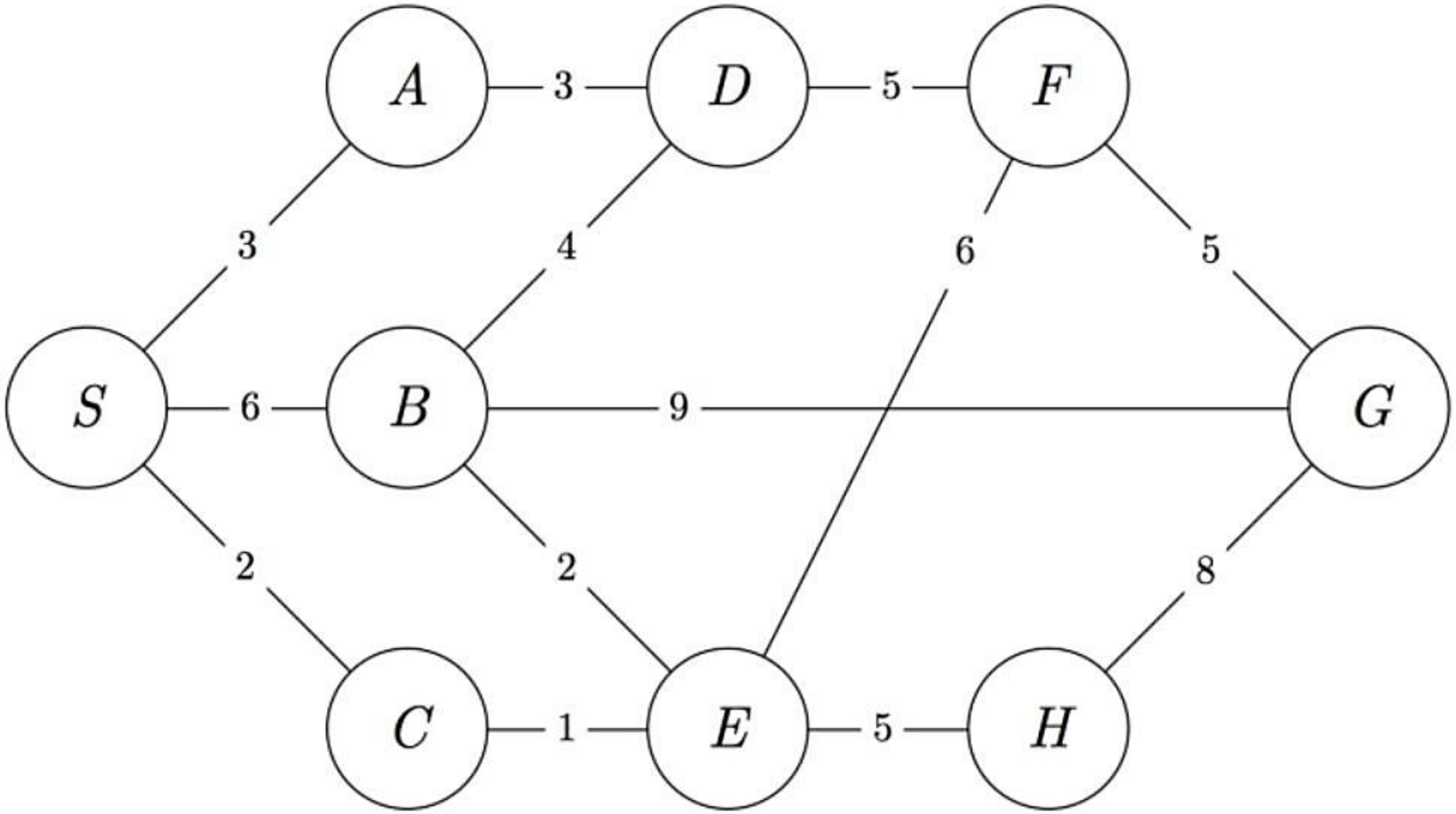
Depth-First Search (DFS)

- Avoiding Repeated States:

  ✓ Requires comparing state descriptions.

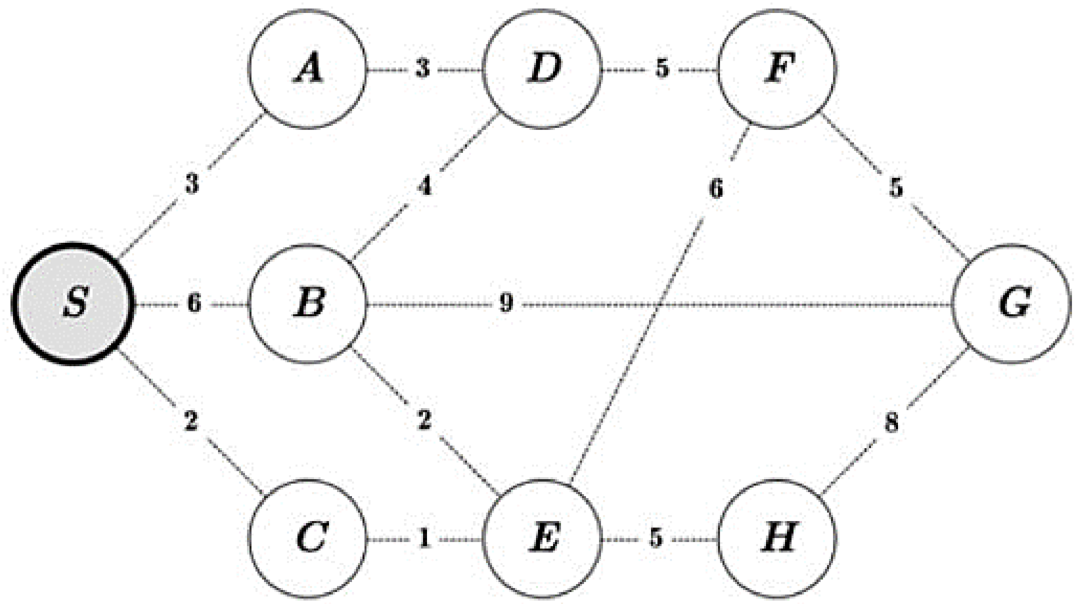
  ✓ For example, Keep track of all generated states. If the state of a new node already exists, then discard the node.

Depth-First Search (DFS)- Example



Question: What is the order of visits of the nodes and the path returned by DFS? Start node S and goal node G

Depth-First Search (DFS)- Example



Stack	S	C	B	A	D	F	E	G
Order of visit	S	A	D	F	G			

Another Solution:

Stack	S	A	B	C	E	F	H	G
Order of visit	S	C	E	H	G			

Depth-Limited Search (DLS)

- Depth-Limited Search (DLS) is a variation of the Depth-First Search (DFS) algorithm that addresses one of DFS's major drawbacks: the possibility of getting stuck in infinite loops or exploring excessively deep paths in infinite state spaces. DLS imposes a depth limit

- Limit (L) on the search, meaning it will not explore nodes beyond a specified depth. This makes it more practical for problems where the depth of the solution is known or can be estimated.

Depth-Limited Search (DLS)

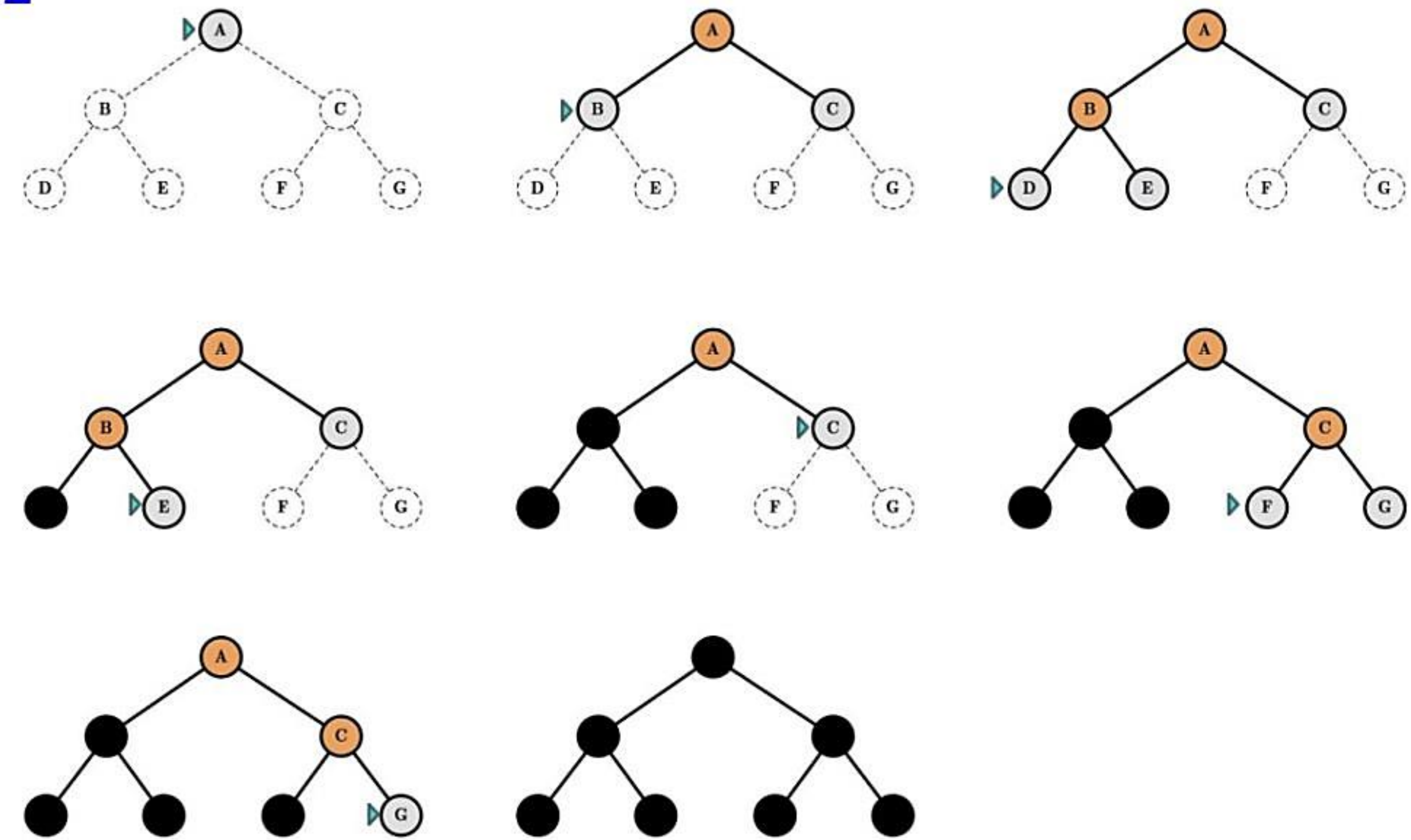
- Avoiding Repeated States:

  ✓ Requires comparing state descriptions.

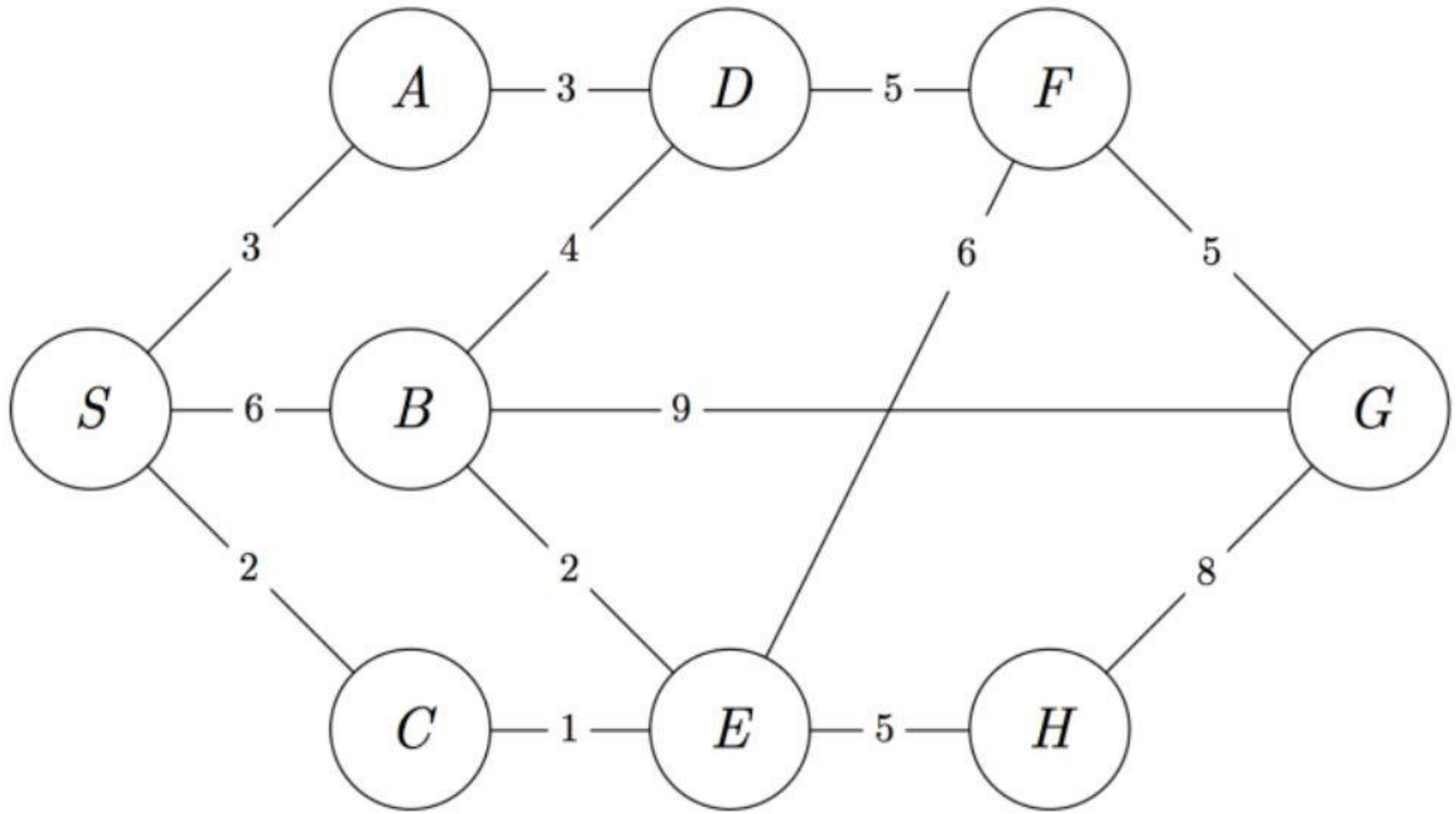
  ✓ For example, Keep track of all generated states. If the state of a new node already exists, then discard the node.

Depth-Limited Search (DLS)

Limit = 2



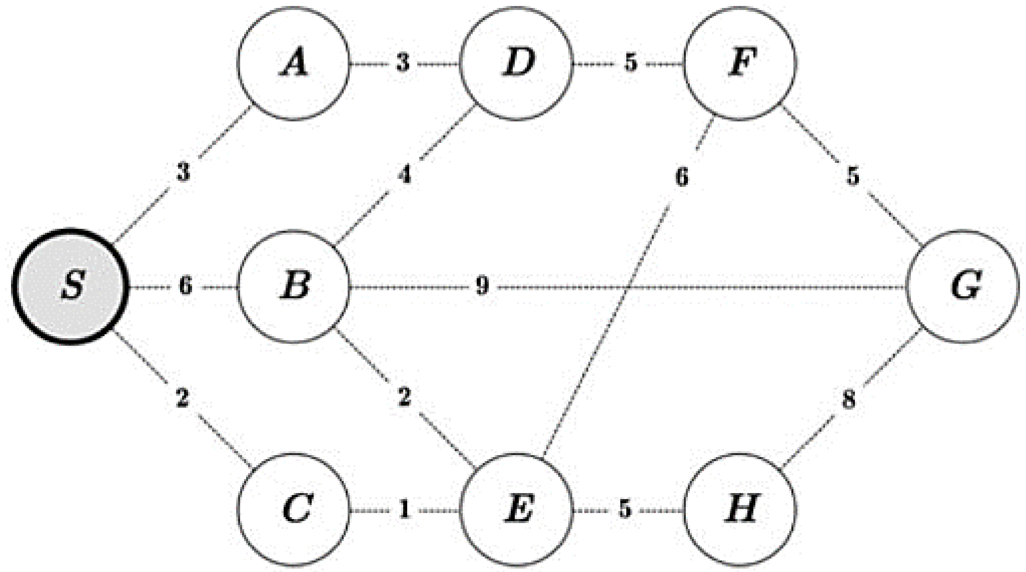
Depth-Limited Search (DLS)



Question: What is the order of visits of the nodes and the path returned by DLS with L=2? Start node S and goal node G

Depth-Limited Search (DLS)

L=2

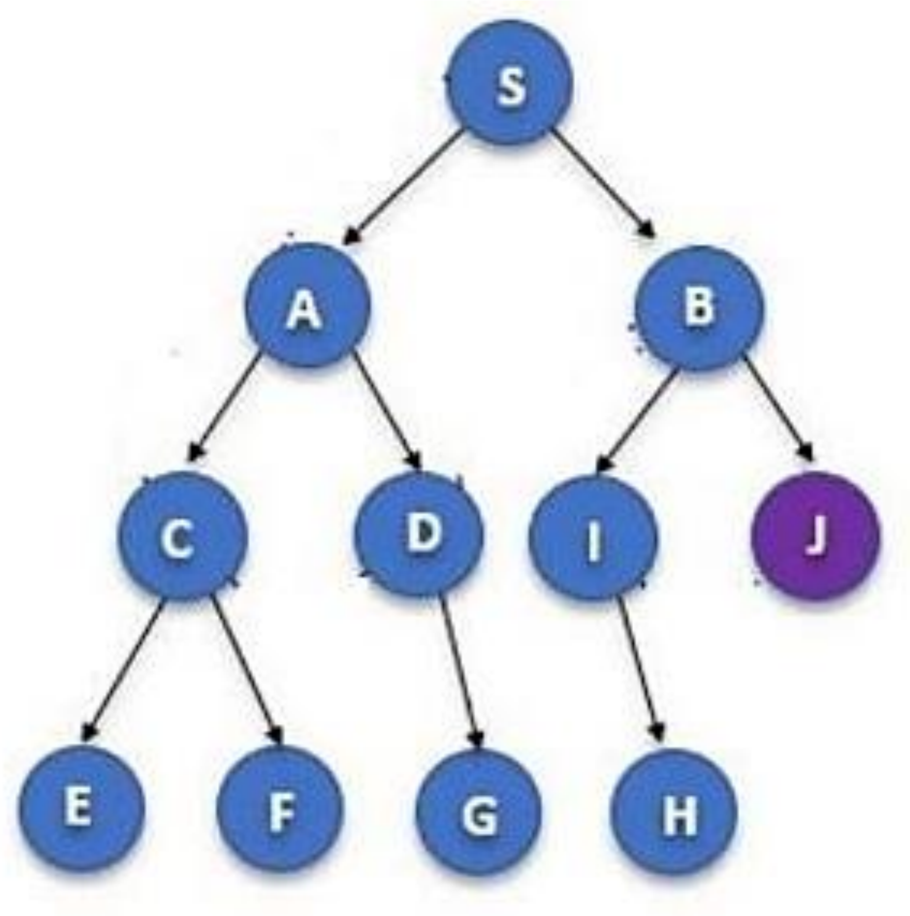


Level	0	1	1	1	2	3	2	2
Stack	S	C	B	A	D	F	E	G
Order of visit	S	A	D	B	G			

Another Solution:

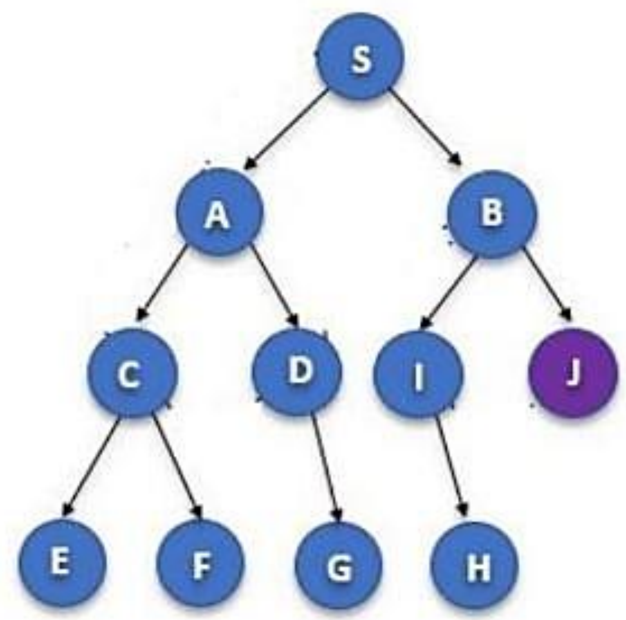
Level	0	1	1	1	2	3	3	2	2
Stack	S	A	B	C	E	F	H	D	G
Order of visit	S	C	E	B	G				

Example



Question: What is the order of visits of the nodes and the path returned by DFS and DLS with L=2? Start node S and goal node j

Answer- DFS



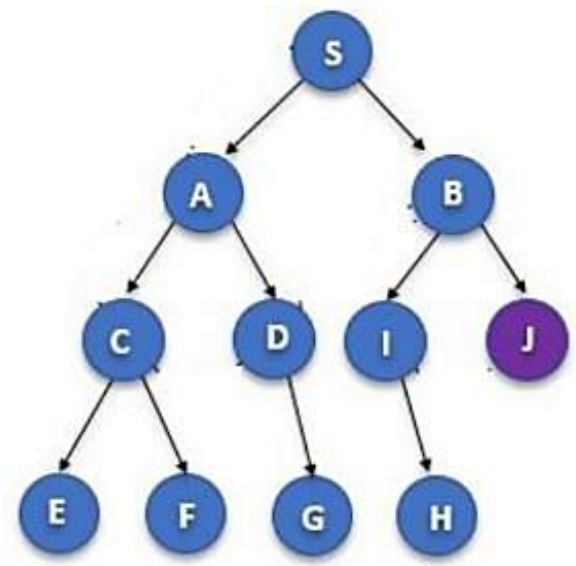
Stack	S	A	B	I	J			
Order of visit	S	B	J					

Another Solution:

Stack	S	B	A	D	C	F	E	G	J	I	H
Order of visit	S	A	C	E	F	D	G	B	I	H	J

Answer- DLS

L=2



Level	0	1	1	2	2			
Stack	S	A	B	I	J			
Order of visit	S	B	J					

Another Solution:

Level	0	1	1	2	2	3	3	3	2	2	3
Stack	S	B	A	D	C	E	F	G	J	I	H
Order of visit	S	A	C	D	B	I	J				

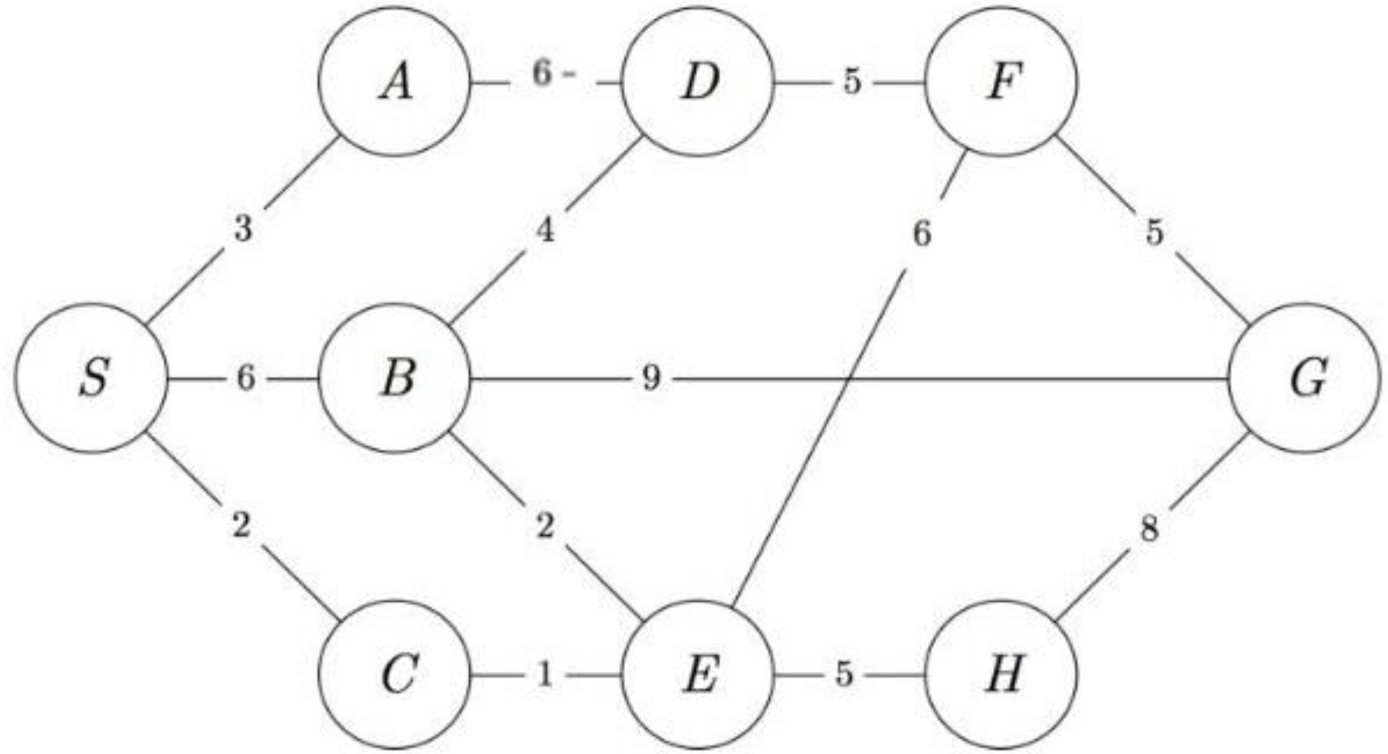
Uniform-Cost Search (UCS)

- It is a blind search algorithm used in artificial intelligence to find the lowest-cost path from an initial state to a goal state.

- Unlike algorithms like Breadth-First Search (BFS) or Depth-First Search (DFS), UCS considers the cost of each path and prioritizes exploring the path with the lowest cumulative cost.

- It is particularly useful for problems where actions have varying costs, such as finding the cheapest route in a weighted graph.

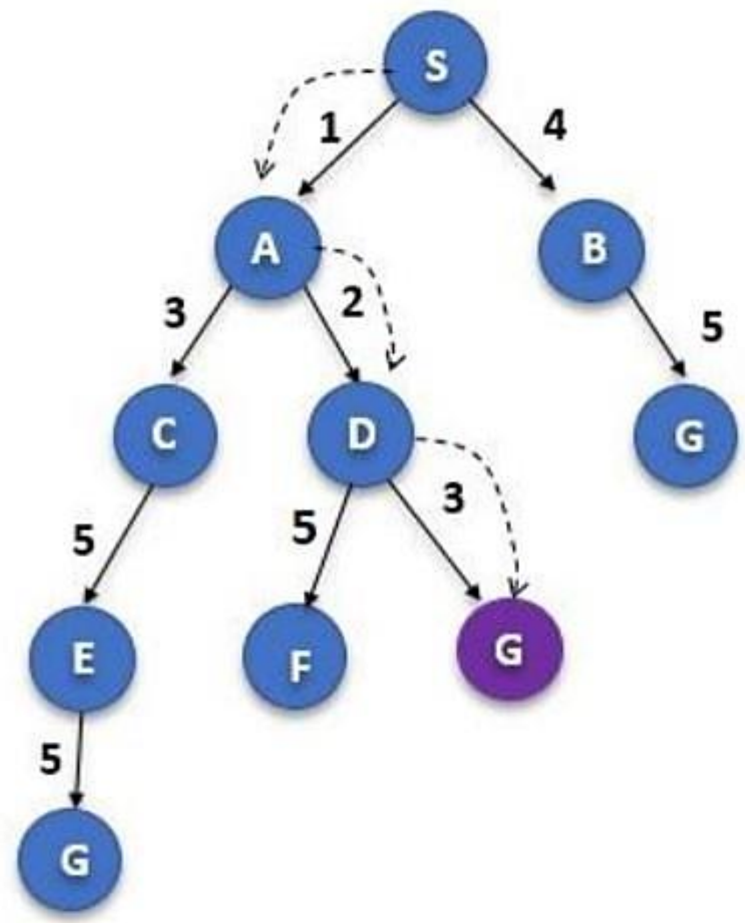
Uniform-Cost Search (UCS)- Example



Question: What is the order of visits of the nodes and the path returned by UCS? Start node S and goal node G

Priority Queue	S	A	B	C	E	F	H	D	G
Order of visit	S	C	E	B	D	F	G		

Uniform-Cost Search (UCS)- Example



Priority Queue	S	A	B	C	D	F	G		
Order of visit	S	A	D	G					

When to use uniformed Search

- Uniformed or Blind search algorithms are useful when:
  - ✓ No additional information about the problem is available (e.g., no heuristic function).
  - ✓ The state space is small or manageable.


Any Questions ?