Chapter 3

Solving Problems by Searching

Agents

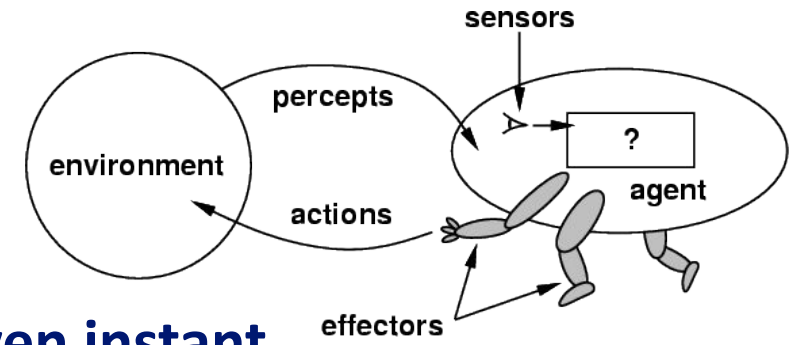
In artificial intelligence, an agent is a **computer program** or system that is designed to **perceive its environment**, **make decisions** and **take actions** to achieve a specific **goal or set of goals**. The agent operates **autonomously**, meaning it is not directly controlled by a human operator.

An agent anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators.

A human agent
- Sensors eyes, ears, and other organs
- Actuators hands, legs, and vocal tract

A robotic agent
- Sensors cameras and infrared range finders
- Actuators various motors

Agents and Environments



- Percept the agent's **perceptual** inputs at **any given instant**.
- An agent's percept sequence the complete **history** of everything the agent has ever perceived.
  - An agent's choice of action **depends** on the entire percept sequence observed to date.
- Agent function a **mathematical description** of the agent's behavior that **maps** any given **percept** sequence to an **action**.

Agent = architecture + program

- Agent program the **implementation** of the agent function.
- Architecture some sort of computing device with physical sensors and actuators

Problem-Solving Agent

- Goal-based agents consider **future actions** and the **desirability** of their **outcomes**.

- Intelligent agents can solve problems by searching a state-space.
  - Problem-solving agent an agent that **plans ahead** to consider a sequence of actions that form a **path** to a goal state.
  - This computational process is called **SEARCH**.

Problem-Solving Agent

A problem-solving agent typically works by:

Defining the problem: This includes specifying the initial state, a goal state, and a set of possible actions the agent can take.

Searching for a solution: The agent explores different sequences of actions (or paths) from the initial state to reach the goal state.

Evaluating actions: The agent determines which path is optimal, based on criteria such as cost or time.

Problem-Solving Agent

Search Process

- The **process of searching is the exploration of a state-space** where each **node** in the state-space **represents** a possible **state** the agent **can reach**.
- The agent **expands** these nodes by performing **actions**, **transitioning** from one state to another.

Common search algorithms include:

- **Uninformed search**: These algorithms, like Breadth-First Search (BFS) and Depth-First Search (DFS), **do not have any information** other than knowing the rules of the environment.
- **Informed search**: Algorithms like A* and Greedy Search use **heuristics** to **estimate** the cost or distance to the goal, **guiding** the search more efficiently.

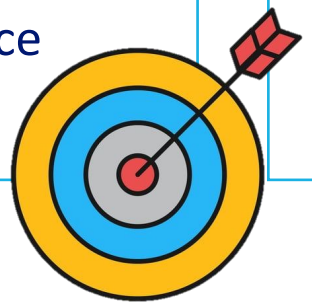
Problem-Solving Agent

- **State-space Model** □ the agent's **model of the world** or agent's abstract **representation of the world**, consisting of
  - **States** usually a set of discrete or continuous states
    - discrete: e.g., in driving, the states in the model could be towns/cities.
    - continuous: e.g., a robot's position in a room.
  - **State-space**: The set of **all possible states the agent can be in**, based on the rules of the environment.
- **Goal State(s)** □ A goal state is the desired state (or set of states) that the agent aims to reach.
  - **Single** goal state: When the goal is a specific and unique state. Example: Driving to a specific city, like Amman.
  - **Multiple** goal states: When the goal is more **generalized**, allowing for **multiple states that can satisfy it**. Example: Driving to any town with a ski resort. There could be multiple towns that fit this goal, and the agent must find one of them.
- **Operators (actions)** □ l**egal actions** which the agent can take to **move** from one state to another

Problem-Solving Process

Goal Formulation

- Goals organize behavior by limiting the objectives and hence the actions to be considered.
- □based on the current situation and the agent's performance measure.



Problem Formulation

- The process of deciding what actions and states to consider, given a goal.

Search

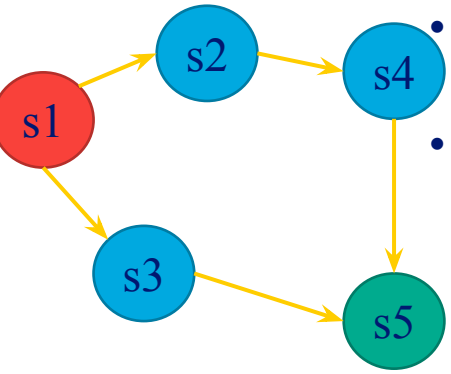
- The process of looking for a sequence of actions that reaches the goal.
- A search algorithm takes a problem as input and returns a solution in the form of an action sequence.

Execution

- Execute the actions in the solution

Search Problem

- A problem can be defined formally by five components:
  1. The initial state that the agent starts in.
  2. A description of the possible actions available to the agent.
     - Given a particular state *s*, ACTIONS(*s*) returns the set of actions that can be executed in *s*.
  3. A description of what each action does (Transition Model).
     - A function RESULT(*s*, *a*) that returns the state that results from doing action *a* in state *s*.
     - Successor □ any state reachable from a given state by a single action.
     - The state space forms a directed network or graph in which the nodes are states and the links between nodes are actions.
     - A path in the state space is a sequence of states connected by a sequence of actions.

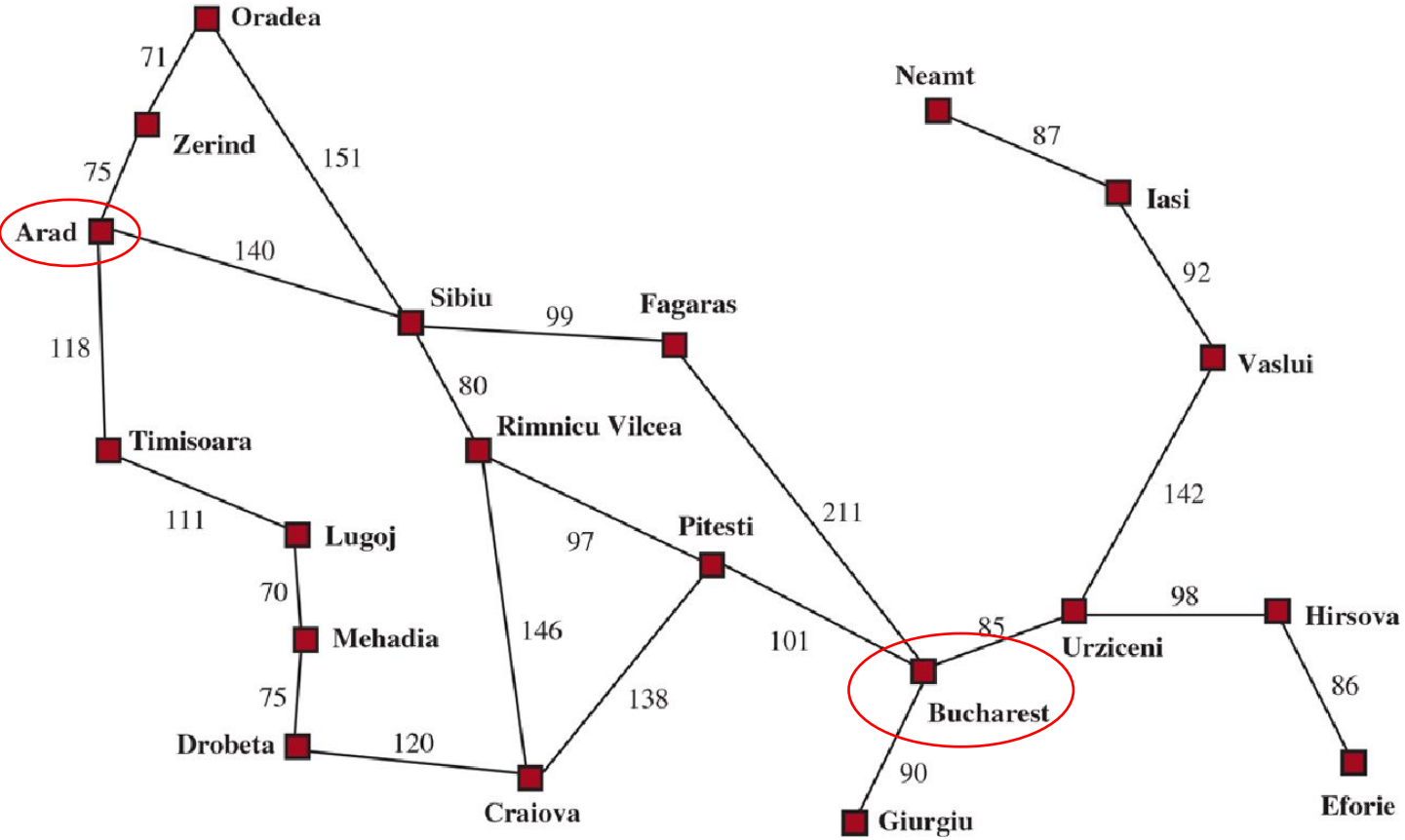


Search Problem

4. The goal test, which determines whether a given state is a goal state.
   - Explicit goal state vs a goal with an abstract property.

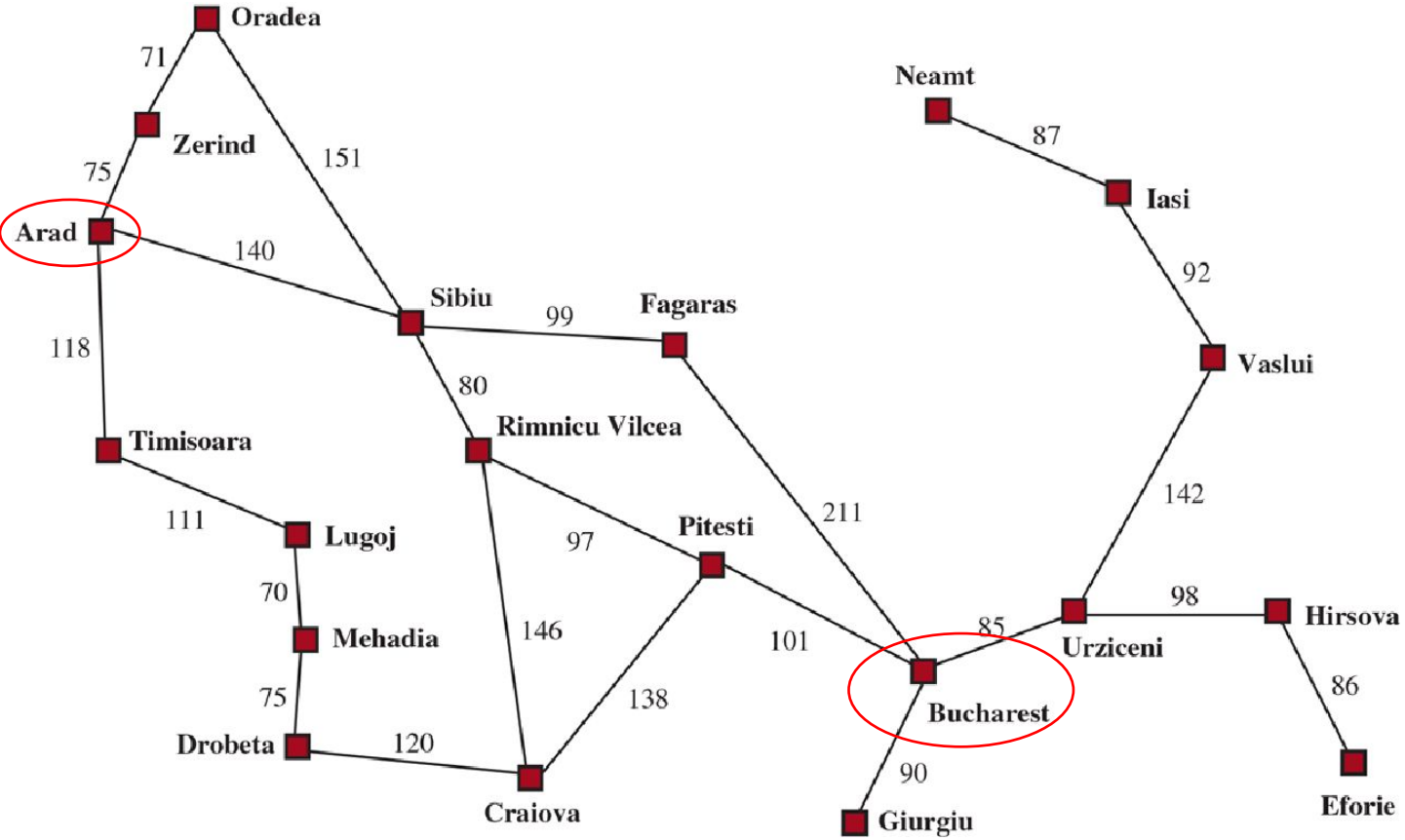
5. A path cost function that assigns a numeric cost to each path.
   - The cost of a path can be described as the sum of the costs of the individual actions along the path.
   - The step cost of taking action *a* in state *s1* to reach state *s2* is denoted by c(*s1*, *a*, *s2*).
   - Solution quality is measured by the path cost function, and an optimal solution has the lowest/highest path cost among all solutions.

Example – Traveling in Romania



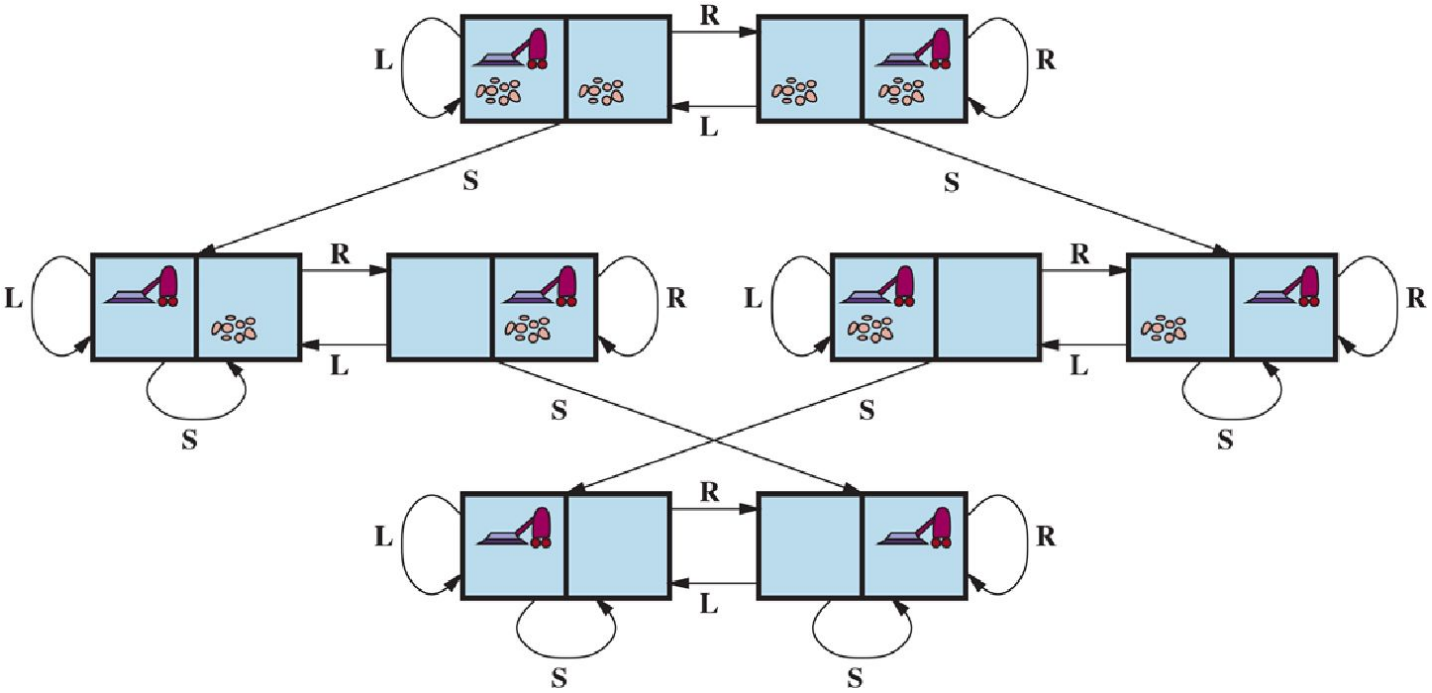
- State space:
- Actions:
- Initial state:
- Path Cost:
- Goal test:
- Solution?

Example – Traveling in Romania



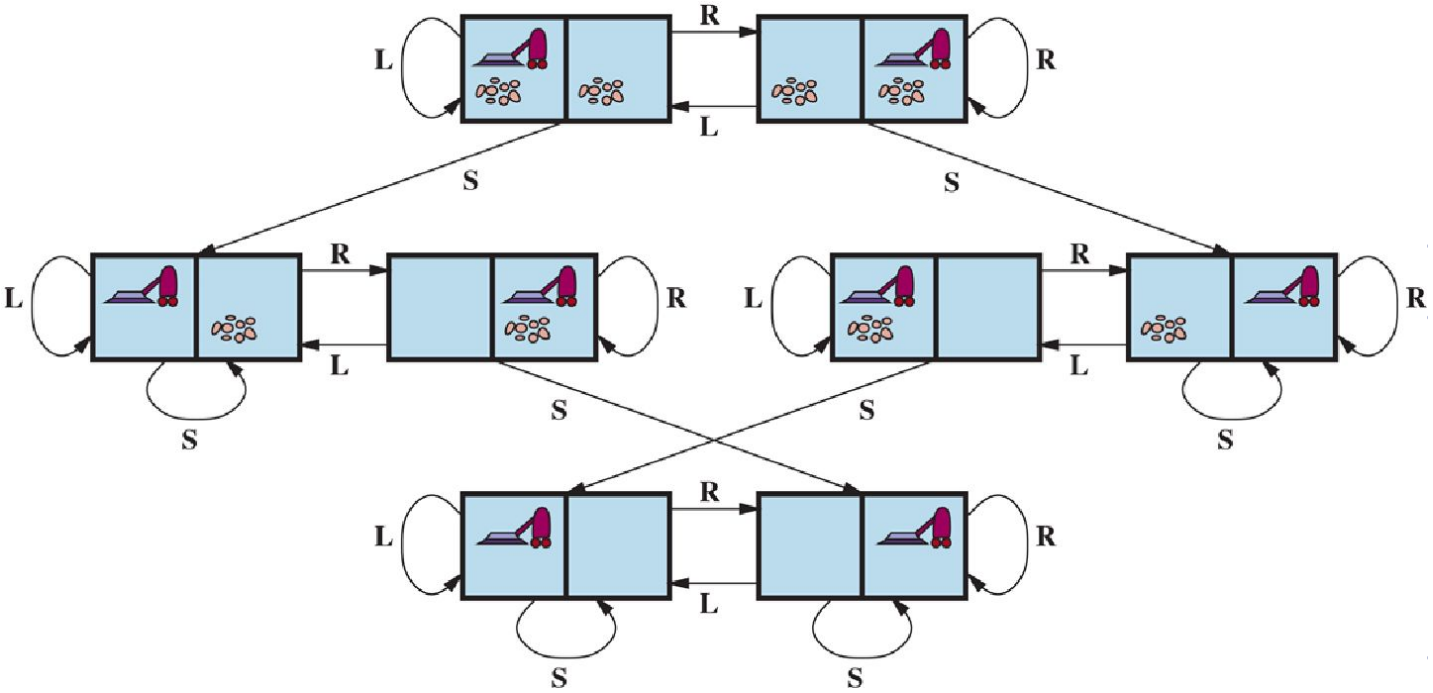
- State space:
  - Cities
- Actions:
  - Go to adjacent city
- Initial state:
  - Arad
- Path Cost:
  - cost = distance
- Goal test:
  - Is state == Bucharest?
- Solution?

Example – Vacuum World



- Initial state:
- States:
- Actions:
- Transition model:
- Path Cost:
- Goal test:

Example – Vacuum World



- Initial state: Any state.
- States: The state is determined by both the agent location and the dirt locations. The agent is in one of two locations, each of which might or might not contain dirt. Thus, there are $2 \times 2^2 = 8$ possible world states.

  Actions: Left, Right, and Suck.

  Transition model: The actions have their expected effects, which include (move right, move left, and suck) except that moving Left in the leftmost square, moving Right in the rightmost square, and Sucking in a clean square has no effect.

  Path Cost: The number of steps in the path.
- Goal test: This checks whether all the squares are clean.

Example – Vacuum World

States space: all possible configurations of the environment, which includes:

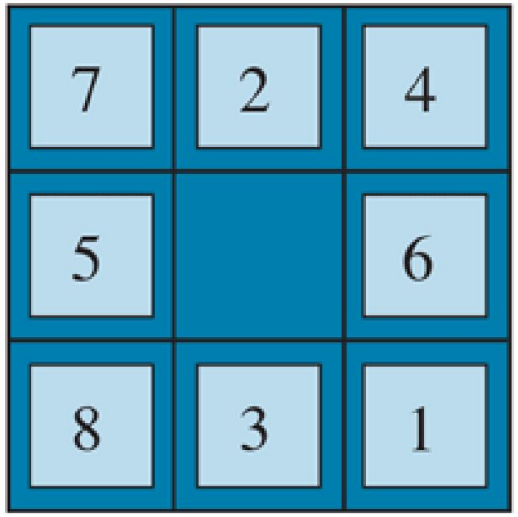
- The location of the vacuum (in one room or the other).
- The cleanliness status of each room (clean or dirty).

For example, if the environment has two rooms (let's say "Room A" and "Room B"), a simple model of the world might define the state-space as combinations of:

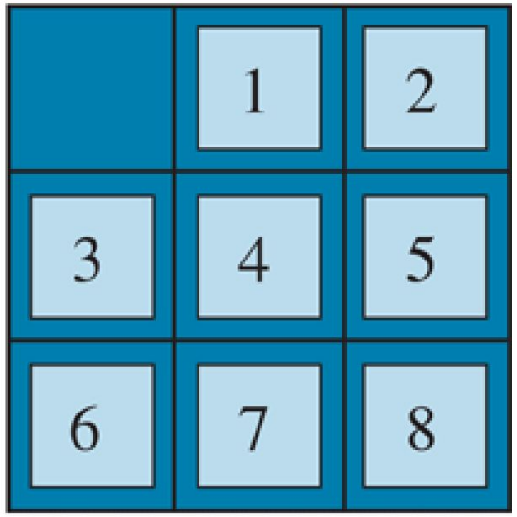
- Vacuum's location: In Room A or Room B.
- Cleanliness: Each room could be clean or dirty.

Thus, there are 2 (locations) x 2 (clean/dirty status for Room A) x 2 (clean/dirty status for Room B) = 8 possible states in this simple two-room environment.

Example – 8 Puzzle



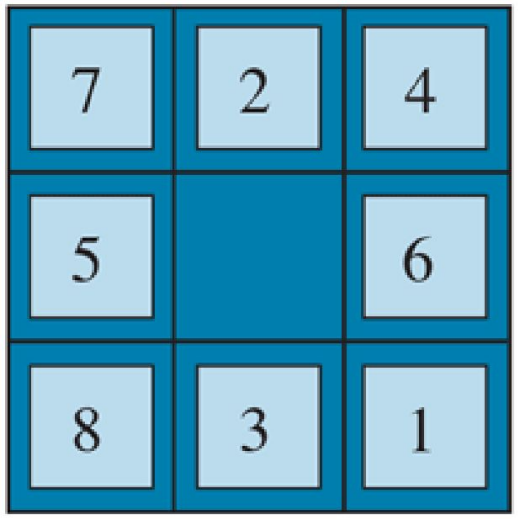
Start State



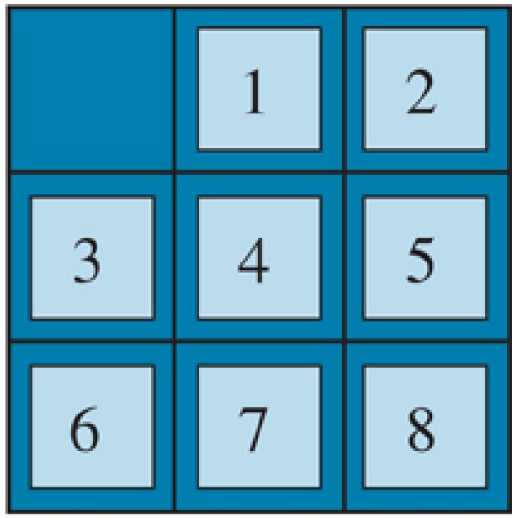
Goal State

- Initial state:
- States:
- Actions:
- Transition model:
- Path Cost:
- Goal test:

Example – 8 Puzzle



Start State



Goal State

- Initial state: Any state.
- States: A state description specifies the location of each of the eight tiles and the blank in one of the nine squares.
- Actions: The movements of the blank space Left, Right, Up, or Down.
- Transition model: Given a state and action, this returns the resulting state; for example, if we apply Left to the start state in Fig., the resulting state has the 5, and the blank is switched.
- Path Cost: The number of steps in the path.
- Goal test: This checks whether the state matches the goal configuration shown in Fig.

Abstraction

- Model a formulation of the problem of getting to a goal state in terms of the initial state, actions, transition model, goal test, and path cost.
  - An abstract mathematical description.

- Abstraction the process of removing irrelevant detail to create an abstract representation (“high-level” ignores irrelevant details).

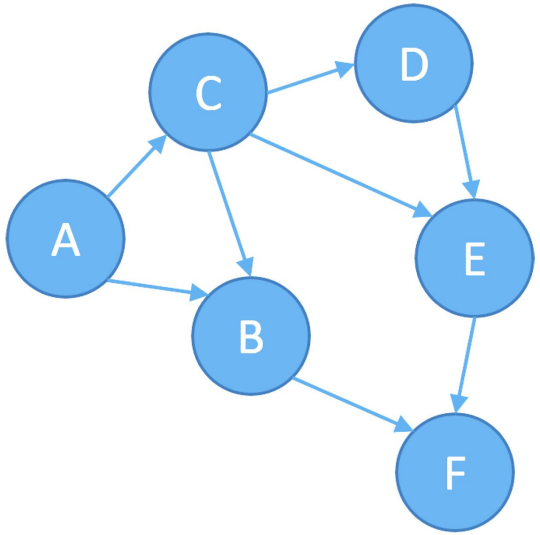
- Abstraction is critical for automated problem-solving.
  - must create an approximate, simplified, model of the world for the computer to deal with: real-world is too detailed to model exactly
  - good abstractions retain all important details

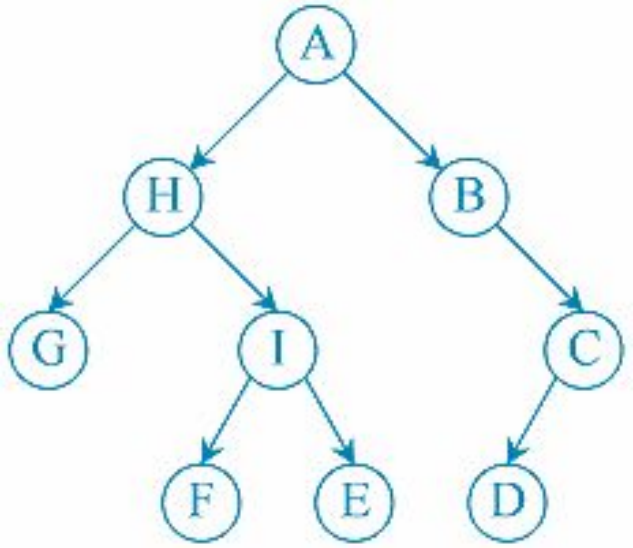
Searching For Solutions

- A solution is an action sequence, so search algorithms work by considering various possible action sequences.

How to represent the search problem mathematically?

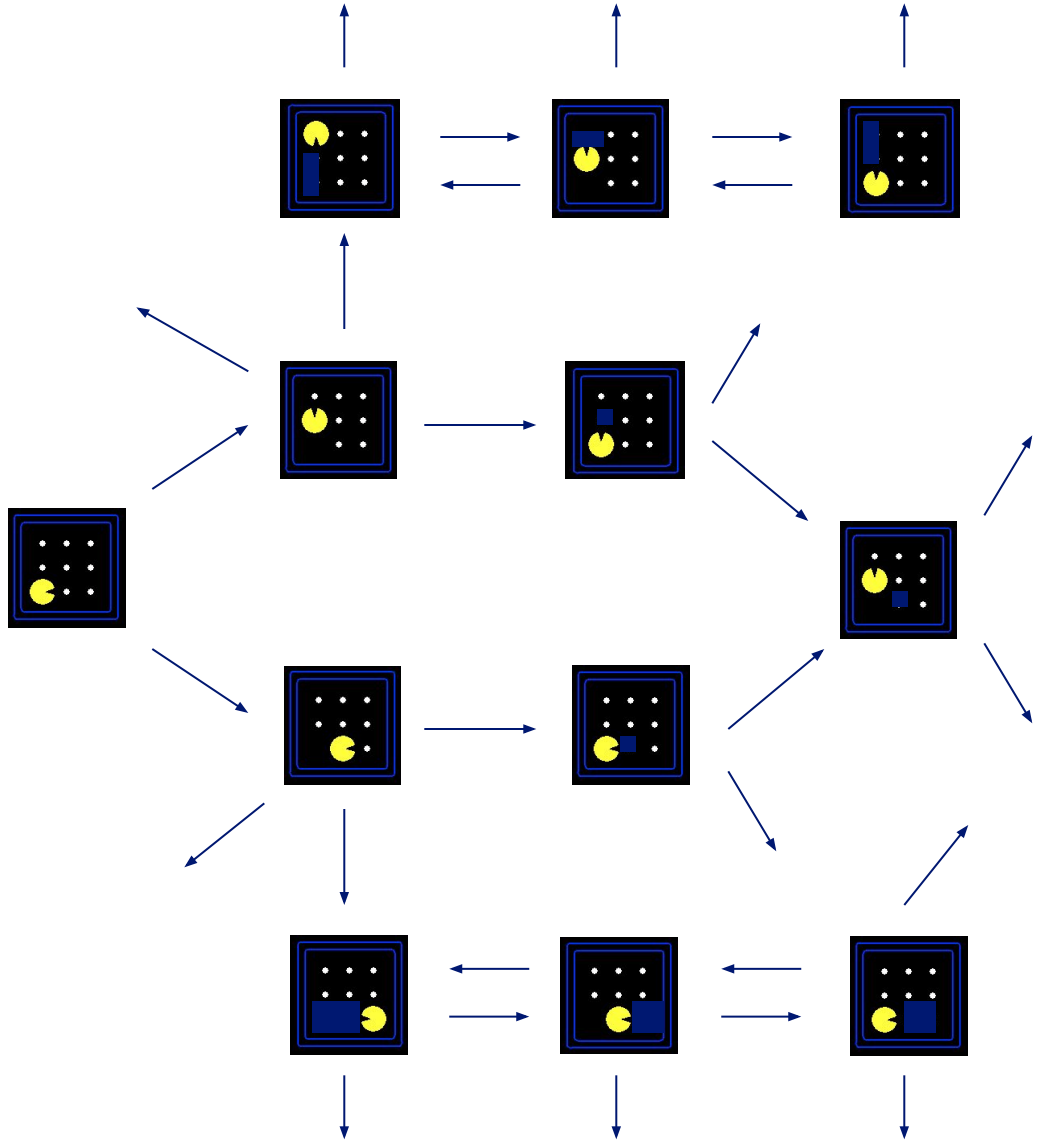
- State space graph
- Search Tree





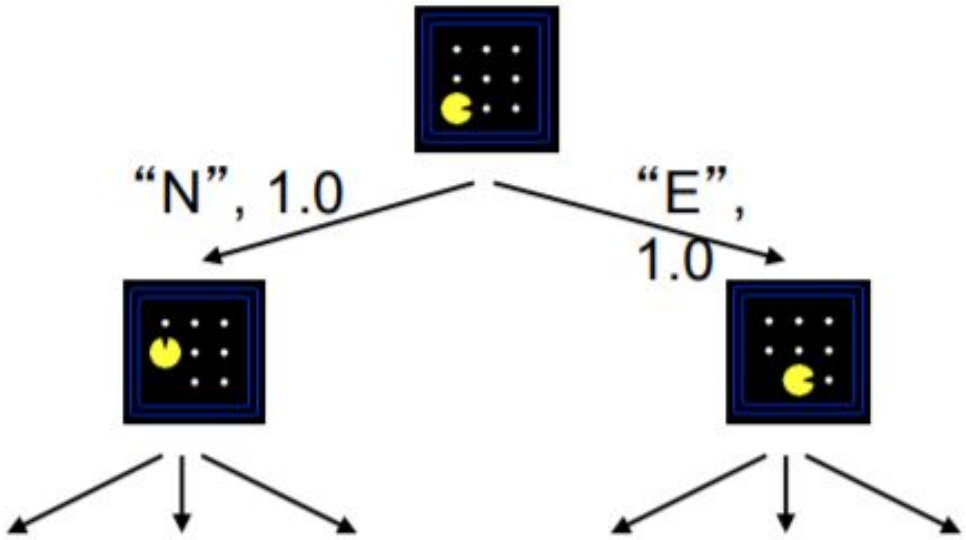
State Space Graphs

- State space graph:
  - Nodes are (abstracted) world configurations.
  - Directed links (edges) are the actions.
  - The goal test is a set of goal nodes (maybe only one).
- In a state space graph, each state occurs only once!
- We can rarely build this full graph in memory (it's too big), but it's a useful idea



Source: http://ai.berkeley.edu

Search Tree

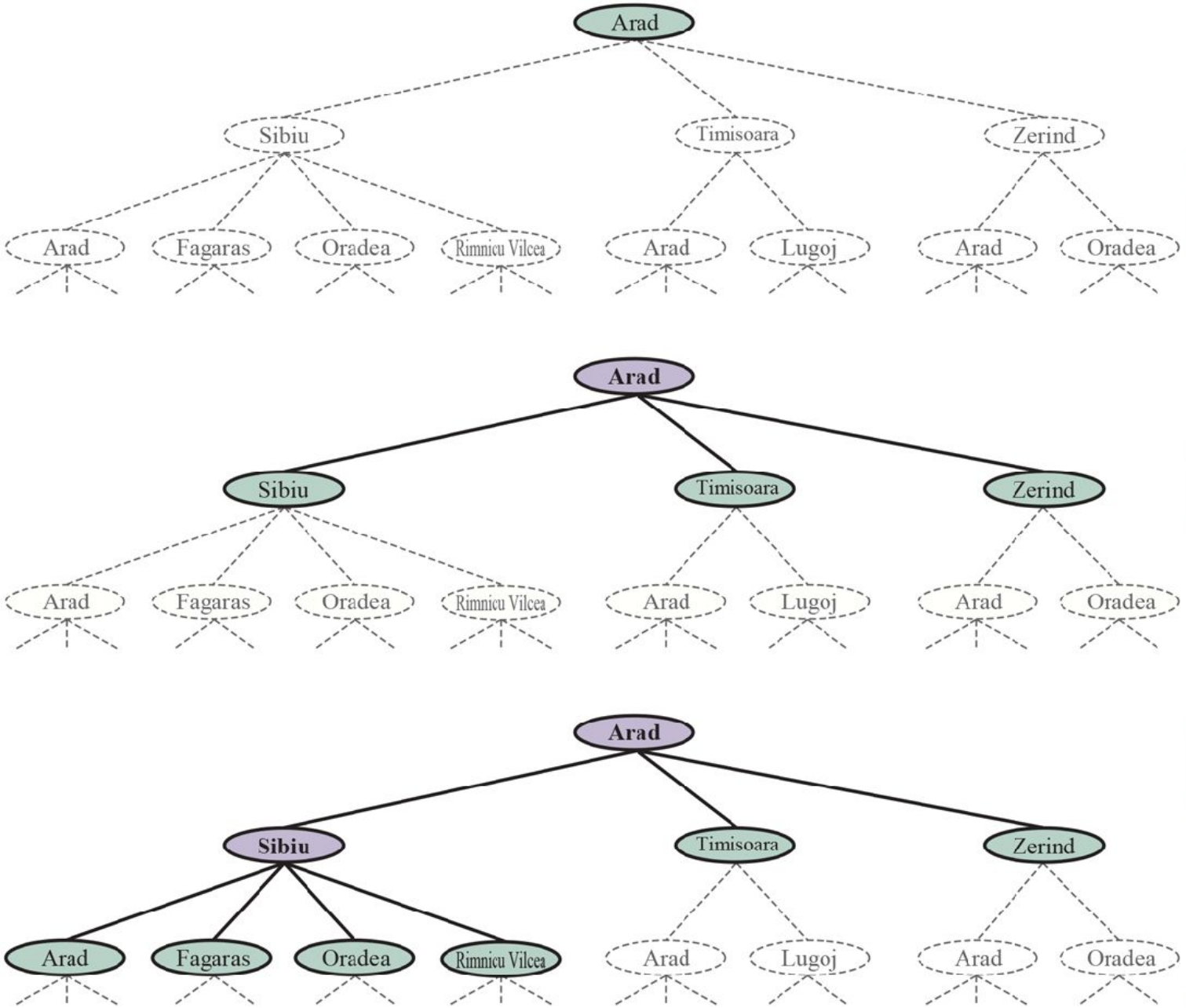


- Search Tree:
  - Root node is the initial state.
  - Nodes are the states.
  - Children nodes correspond to successors.
  - Leaf node is a node with no children in the tree.
- This is a "what if" tree of plans and outcomes.
- For most problems, we can never actually build the whole tree.

Search Tree

- Search strategy how they choose which state to expand next.

- The process of expanding nodes until either a solution is found or there are no more states to expand.



Measuring Problem-Solving Performance

- The performance of search algorithms is evaluated based on:
  - Completeness: Is the algorithm guaranteed to find a solution when there is one?
  - Optimality: Does the strategy find the optimal solution?
  - Time complexity: How long does it take to find a solution?
  - Space complexity: How much memory is needed to perform the search?

Search Strategies

- Uninformed search strategies (blind, exhaustive, brute-force) do not guide the search with any additional information about the problem.
  - All we know is how to generate new states and recognize a goal state.

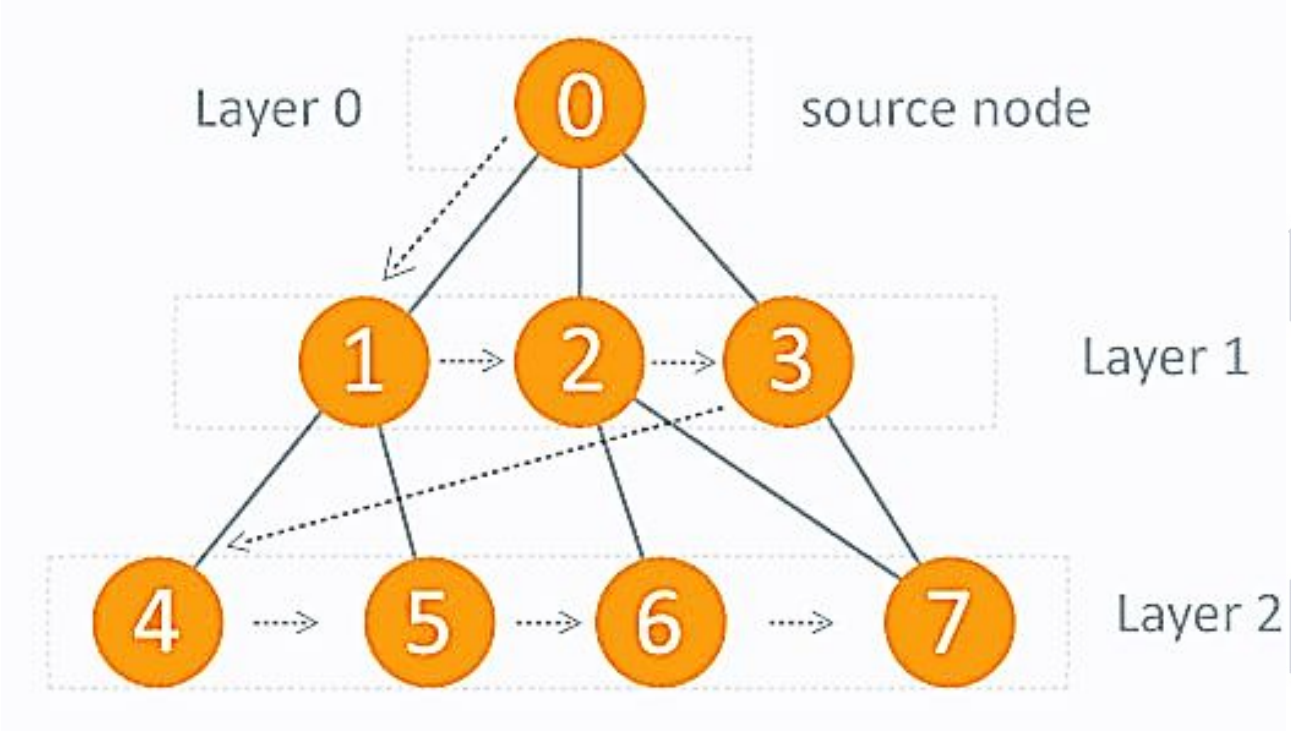
- Informed search strategies (heuristic, intelligent) use information about the problem (estimated distance from a state to the goal) to guide the search.
  - The estimate is not perfect (otherwise no search is needed!) but can help prune the search space considerably.

Uninformed Search Strategies

- Breadth-first search.
- Depth-first search.
- Uniform-cost search.
- Depth-limited search.

Breadth-First Search (BFS)

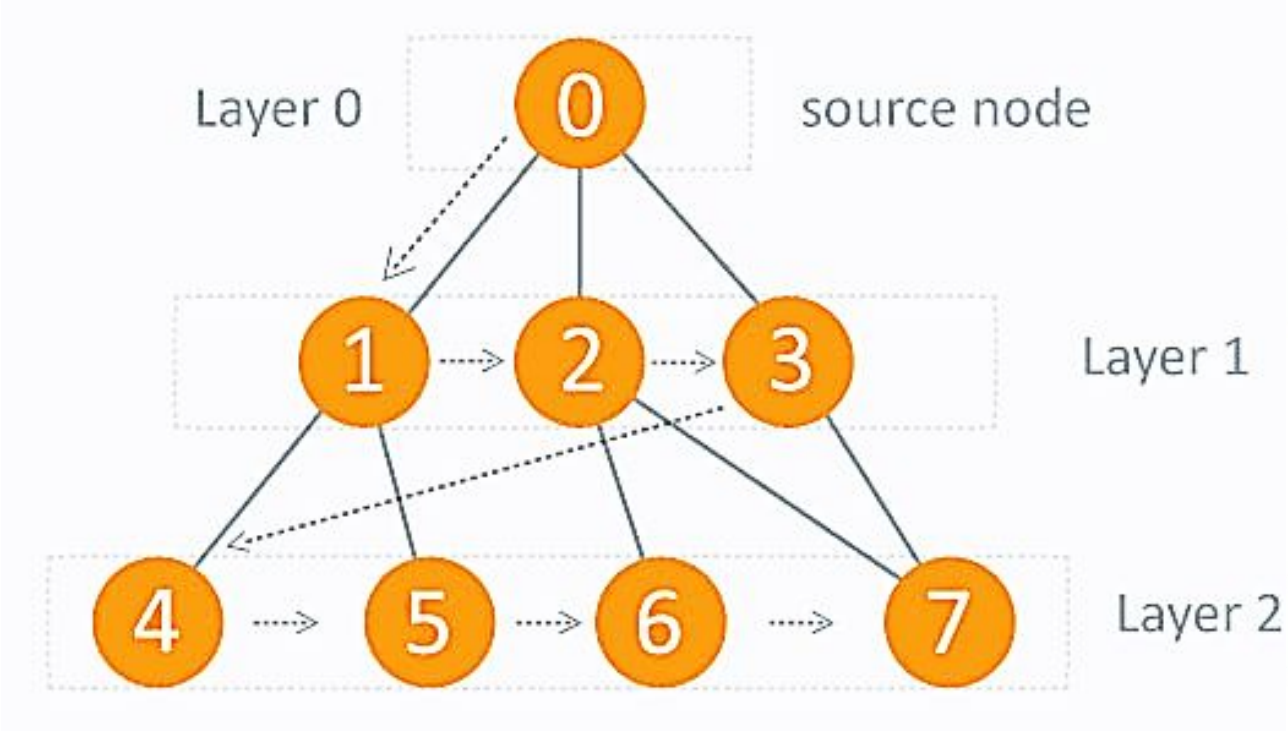
- Breadth-first search is a simple strategy in which the root node is expanded first, then all the successors of the root node are expanded next, then their successors, and so on.
  - All the nodes are expanded at a given depth in the search tree before any nodes at the next level are expanded.



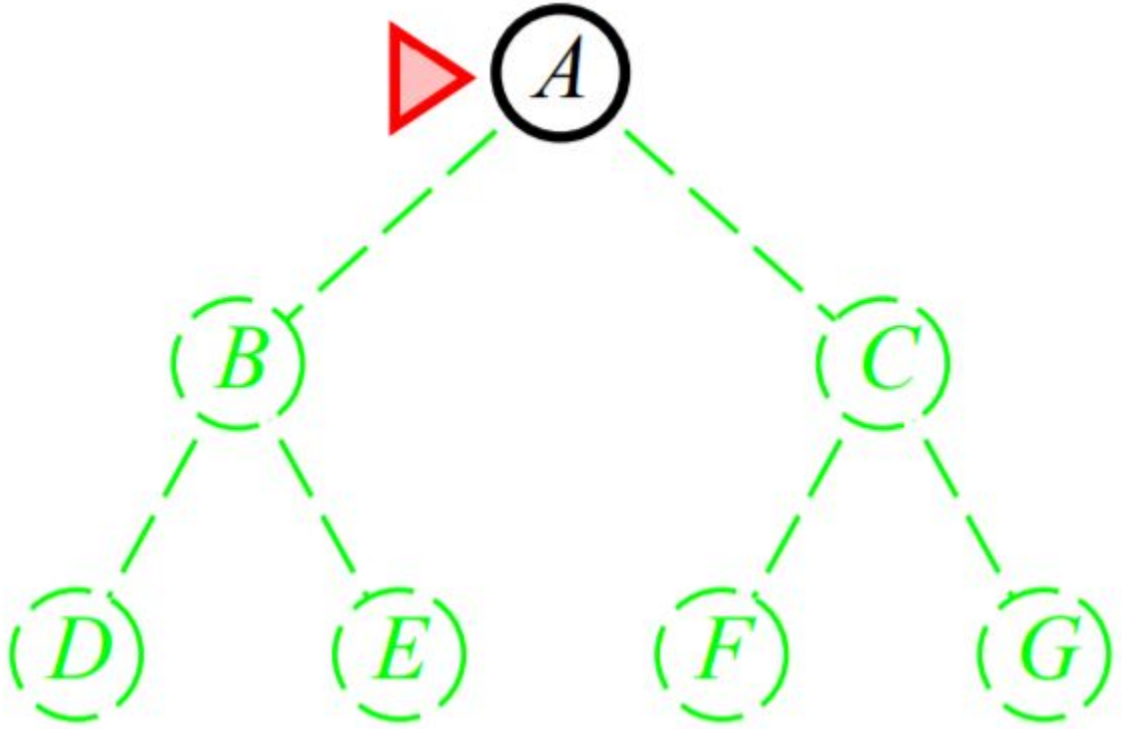
Breadth-First Search (BFS)

Strategy Expand shallowest unexpanded node

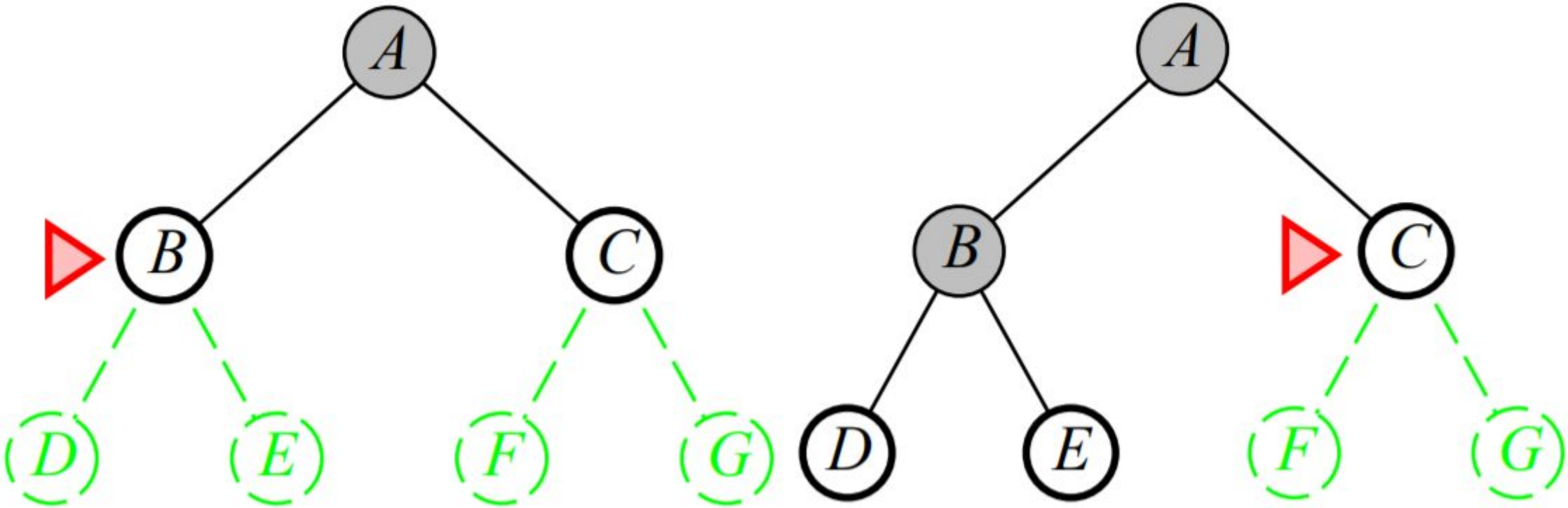
Implementation the current set of unexpanded nodes, the fringe (or frontier), is processed as FIFO queue, i.e., new successors go at end



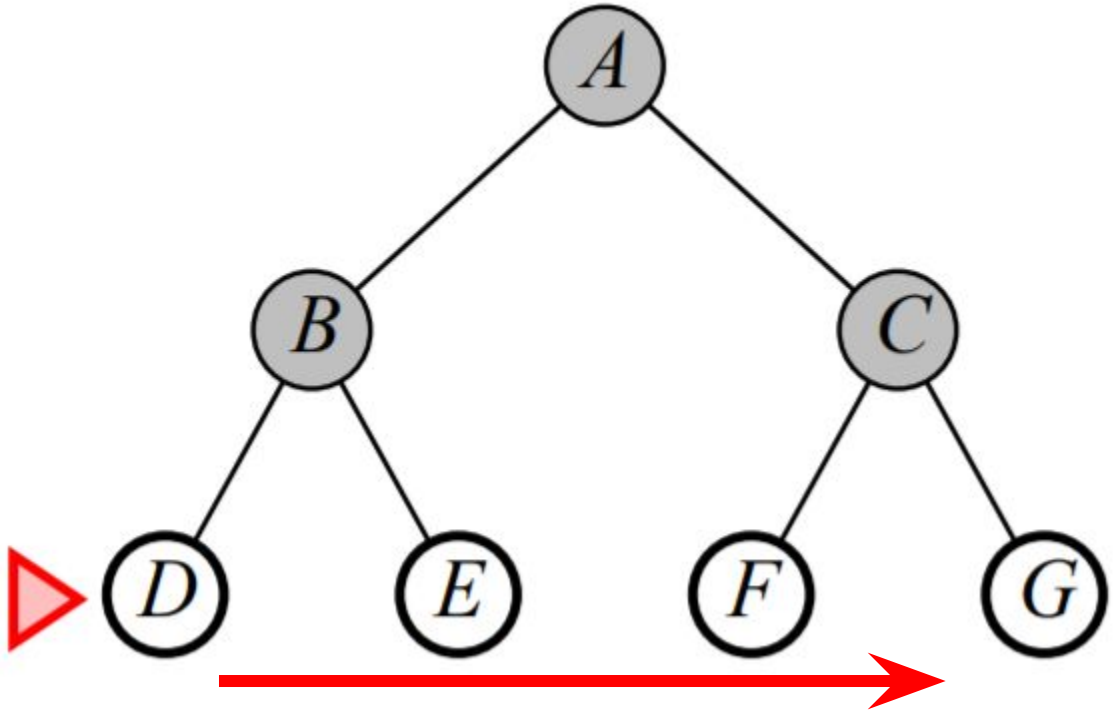
Breadth-First Search (BFS)



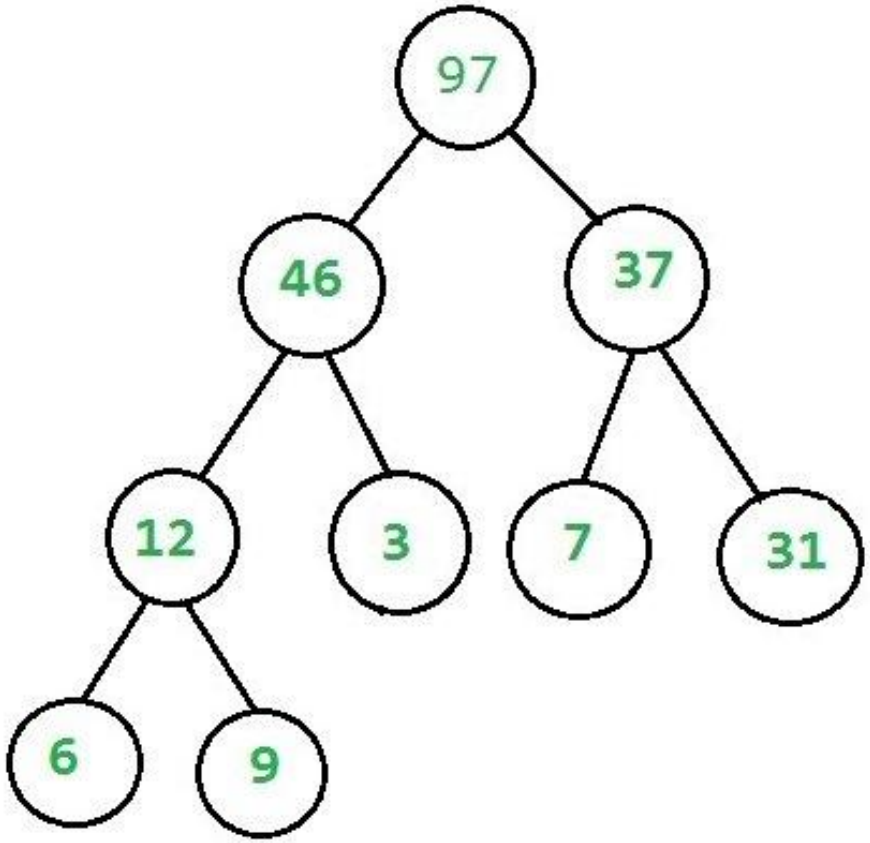
Breadth-First Search (BFS)



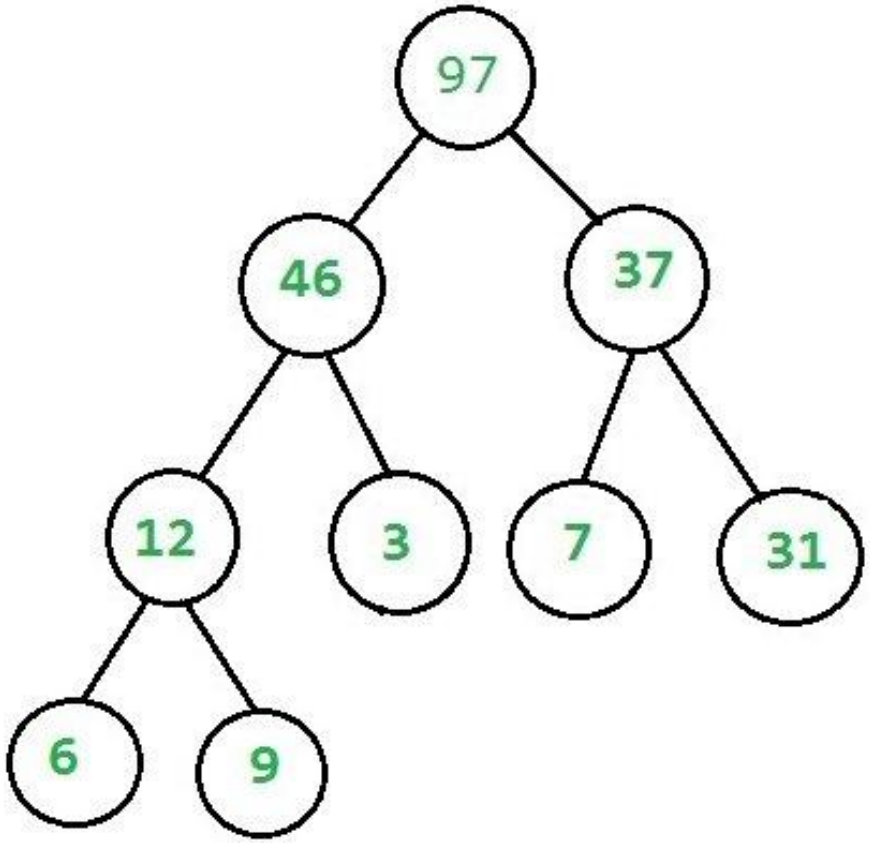
Breadth-First Search (BFS)



Breadth-First Search (BFS)

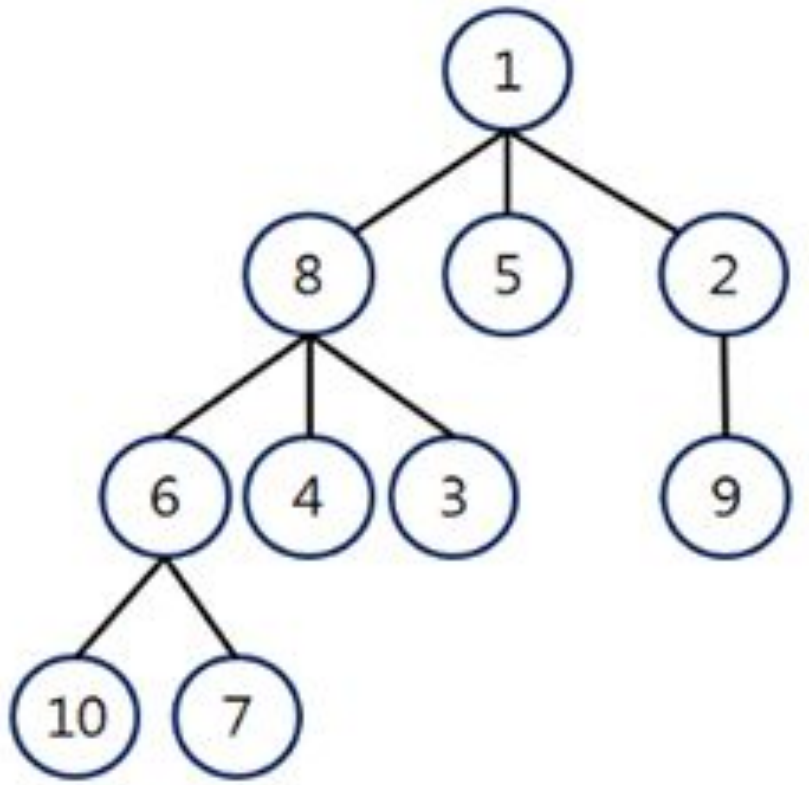


Breadth-First Search (BFS)



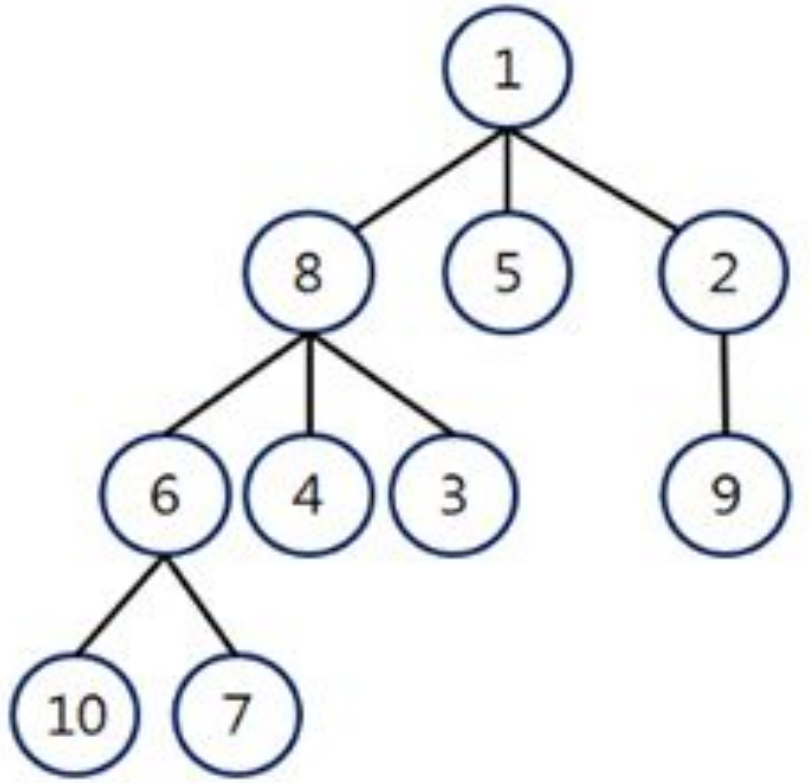
97, 46, 37, 12, 3, 7, 31, 6, 9

Breadth-First Search (BFS)

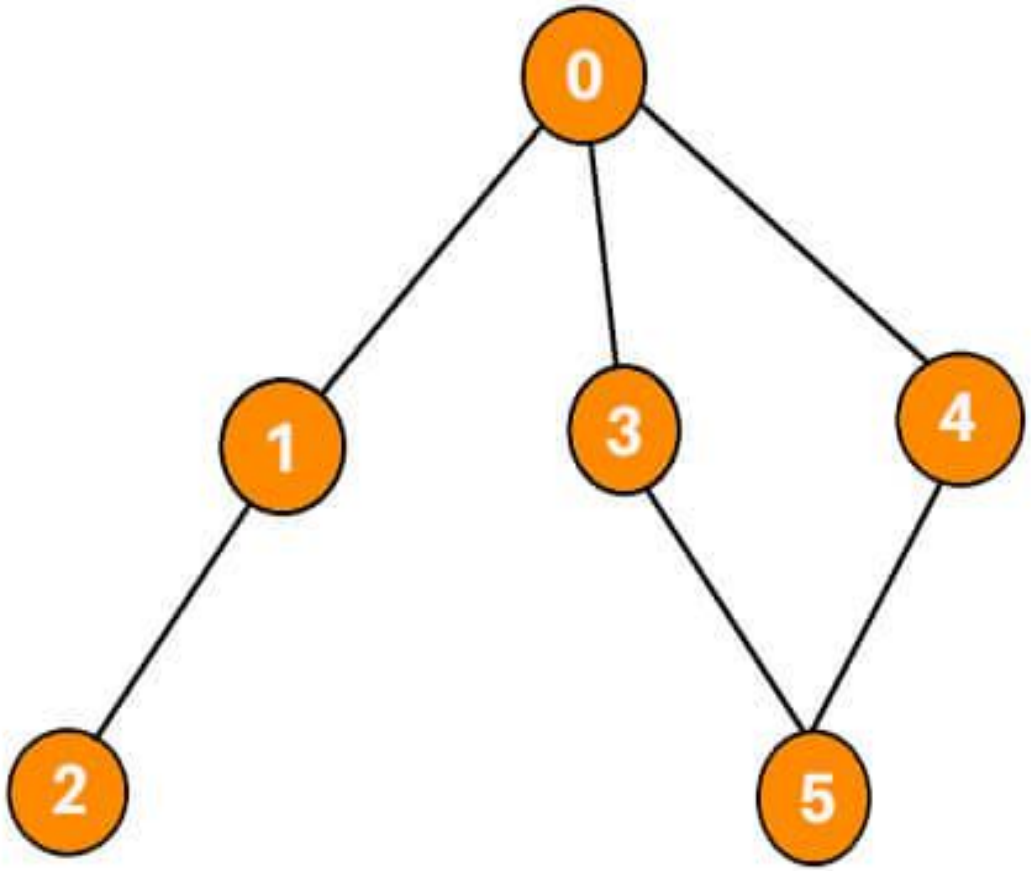


Breadth-First Search (BFS)

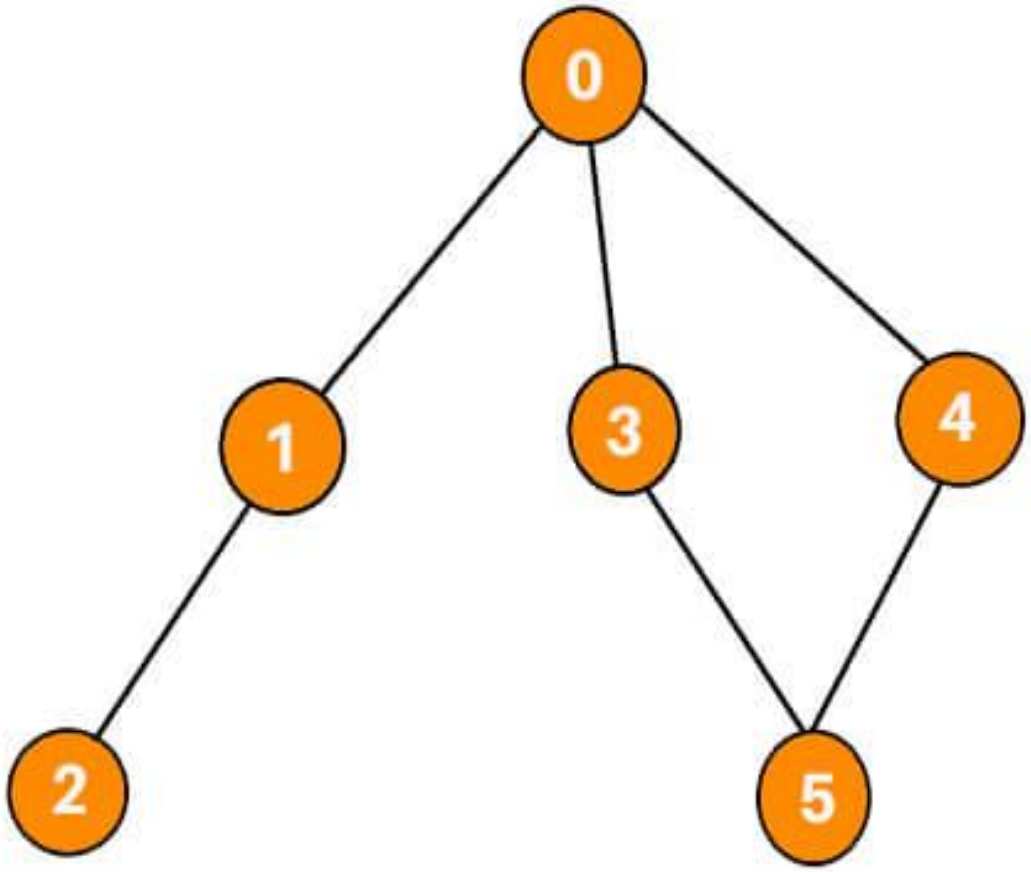
1, 8, 5, 2, 6, 4, 3, 9, 10, 7



Breadth-First Search (BFS)



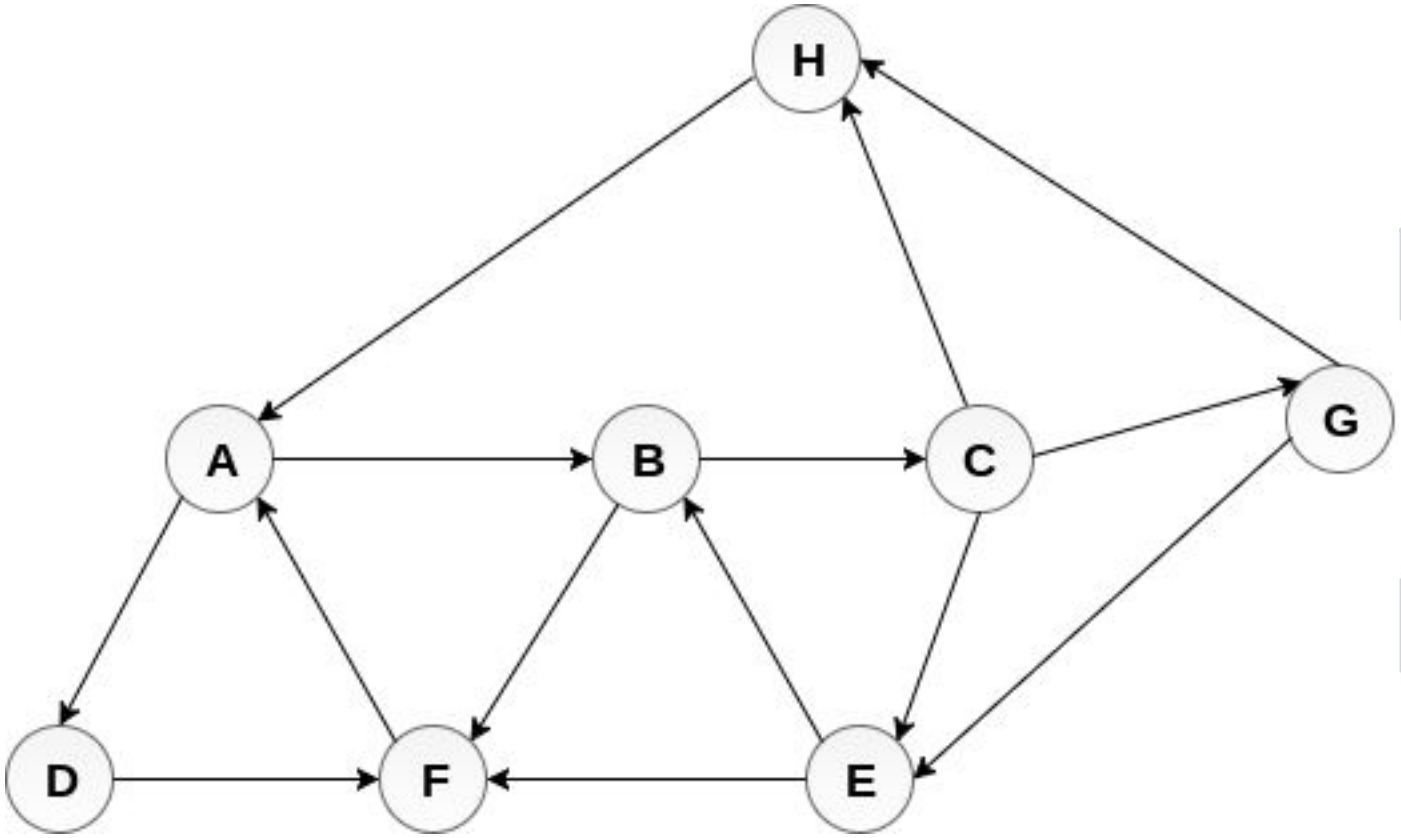
Breadth-First Search (BFS)



0, 1, 3, 4, 2, 5

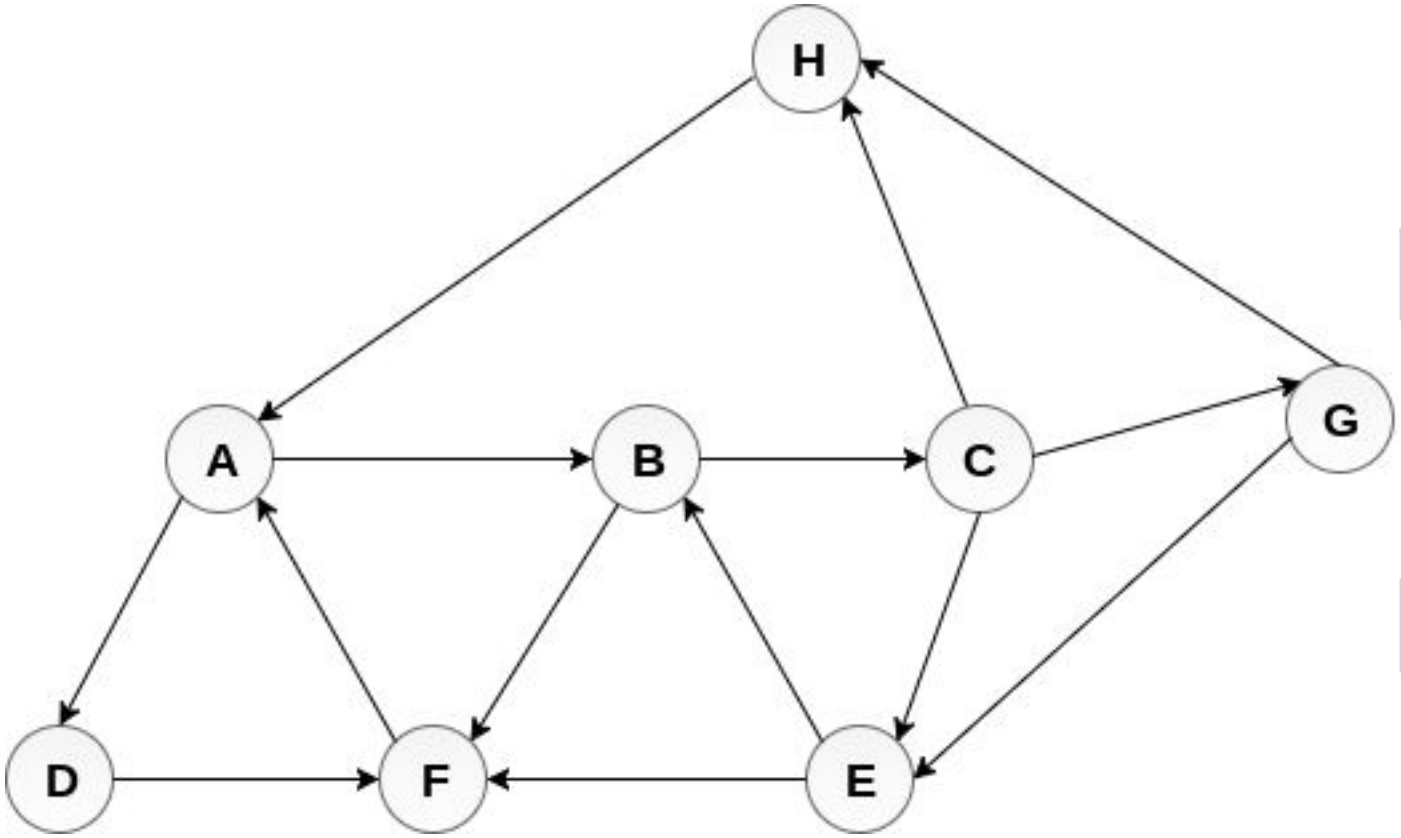
Breadth-First Search (BFS)

Initial state or start node is A
leftmost node first



Breadth-First Search (BFS)

Initial state or start node is A
leftmost node first



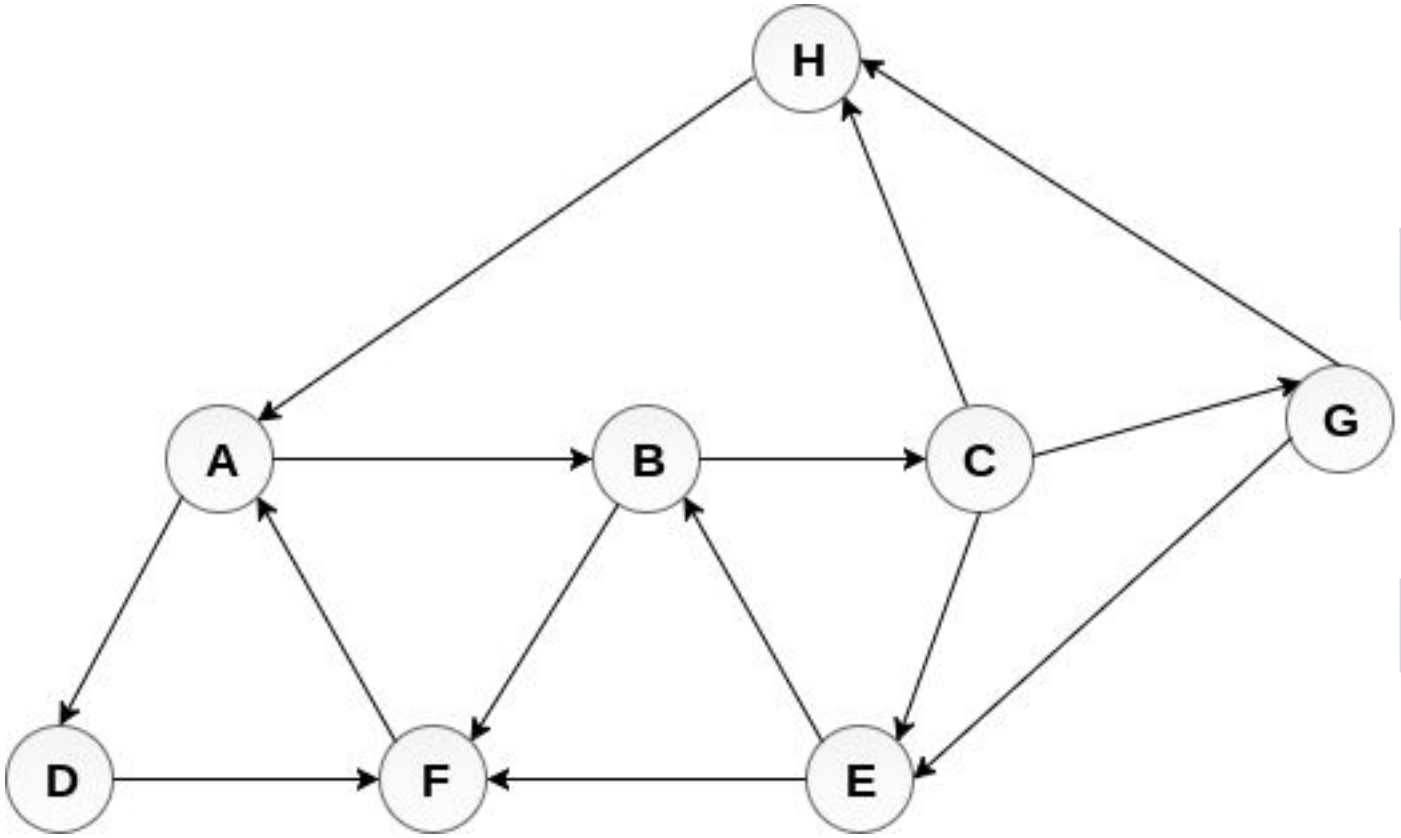
A, D, B, F, C, E, G, H

Breadth-First Search (BFS)

Initial state or start node is A

What the result if

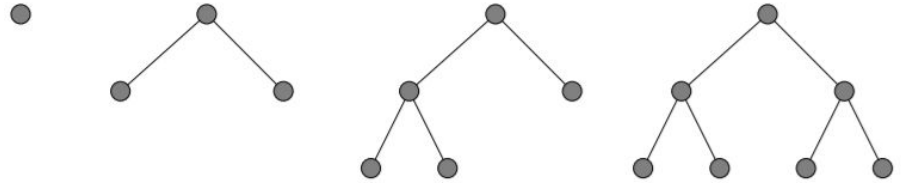
1. rightmost node first
2. alphabetical order first
3. Show the search tree



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

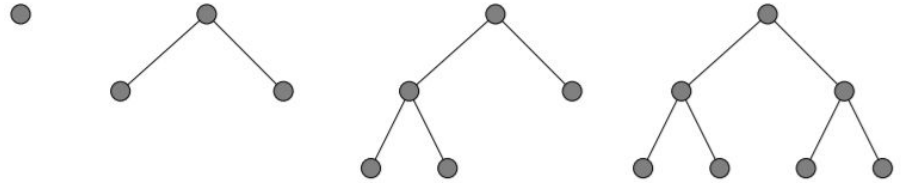
Figure 3.11 Breadth-first search on a graph.

Breadth-First Search (BFS)



- Is it complete?
- Is it optimal?
- Time complexity?
- Space complexity?

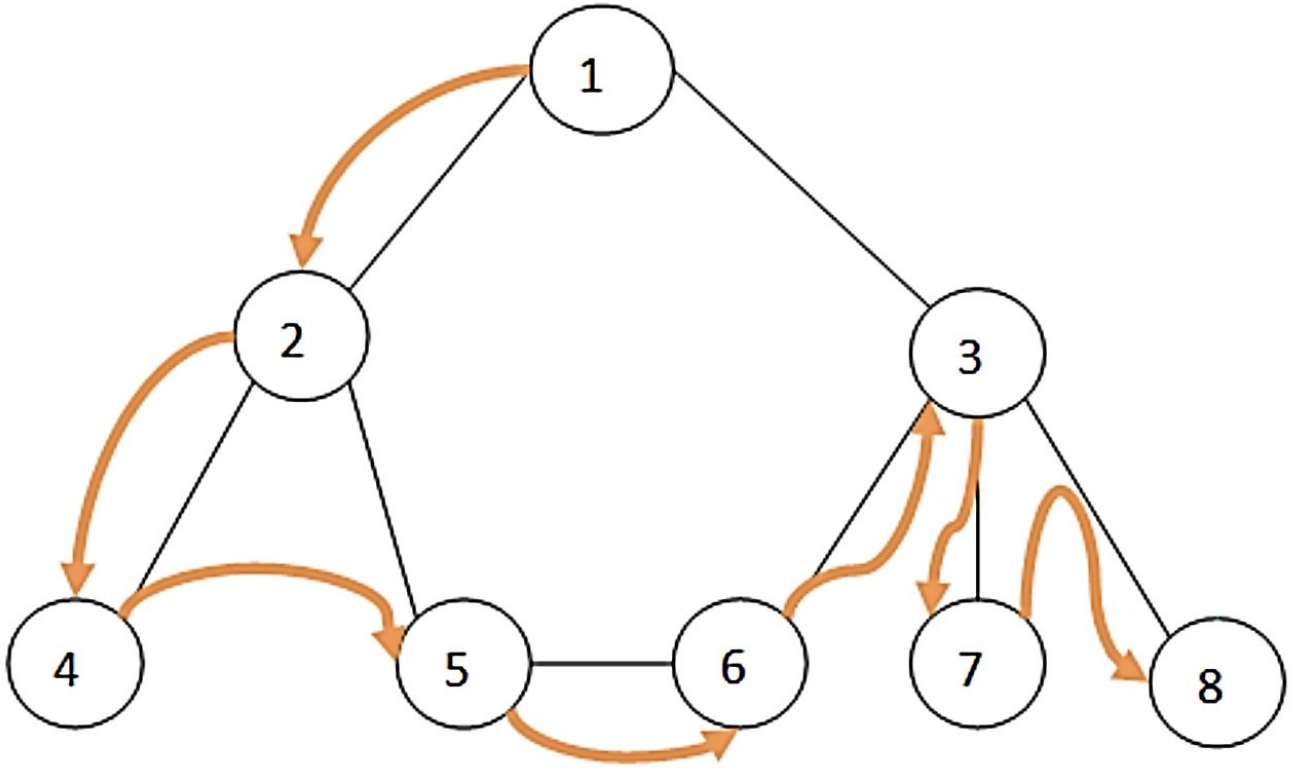
Breadth-First Search (BFS)



- Is it complete?
  - Yes.
- Is it optimal?
  - Only if costs are all 1.
- Time complexity?
  - Number of node expansions. Worst Case → All nodes must be expanded to find a goal state.
  - $O(b^d)$, b → maximum branching factor and d → depth of shallowest goal state.
  - Space complexity?
  - Every node generated remains in memory.
  - $O(b^d)$.

Depth-First Search (DFS)

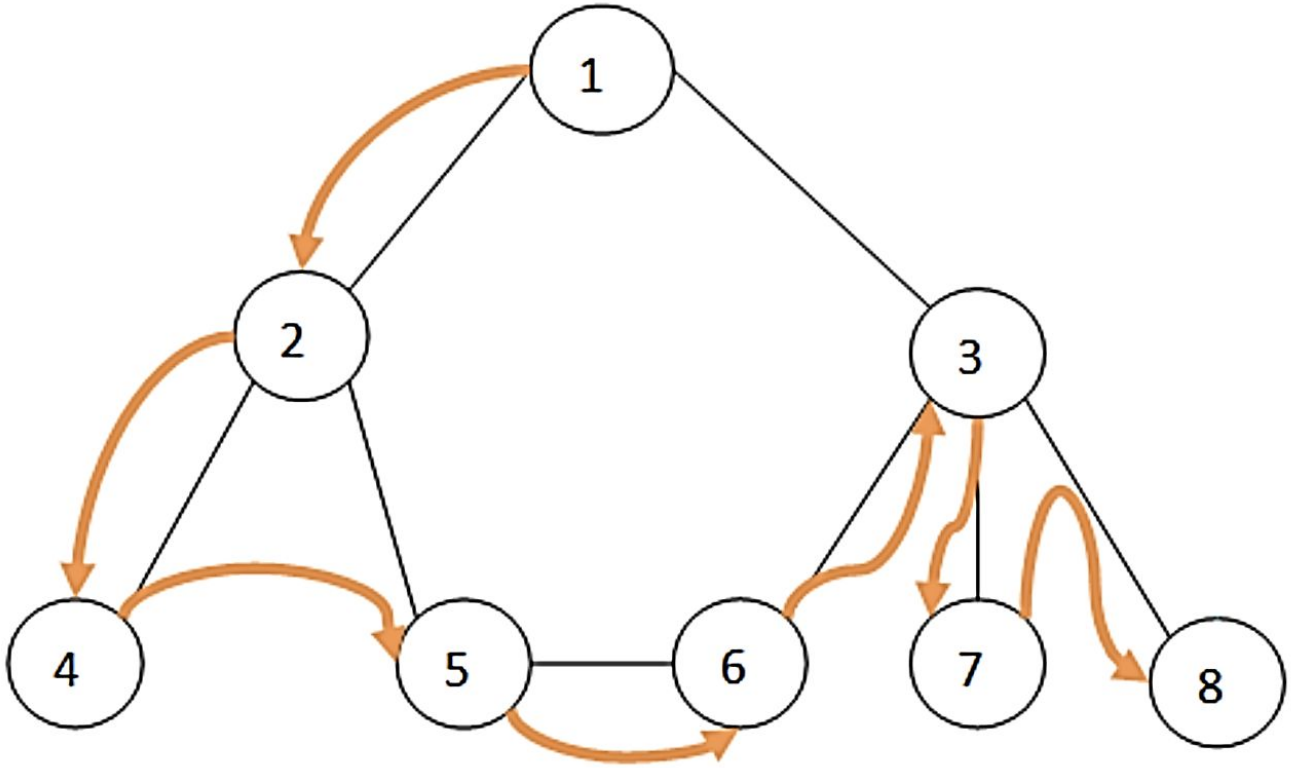
- Depth-first search always expands the deepest node in the current frontier of the search tree.
  - The search proceeds immediately to the deepest level of the search tree, where the nodes have no successors.



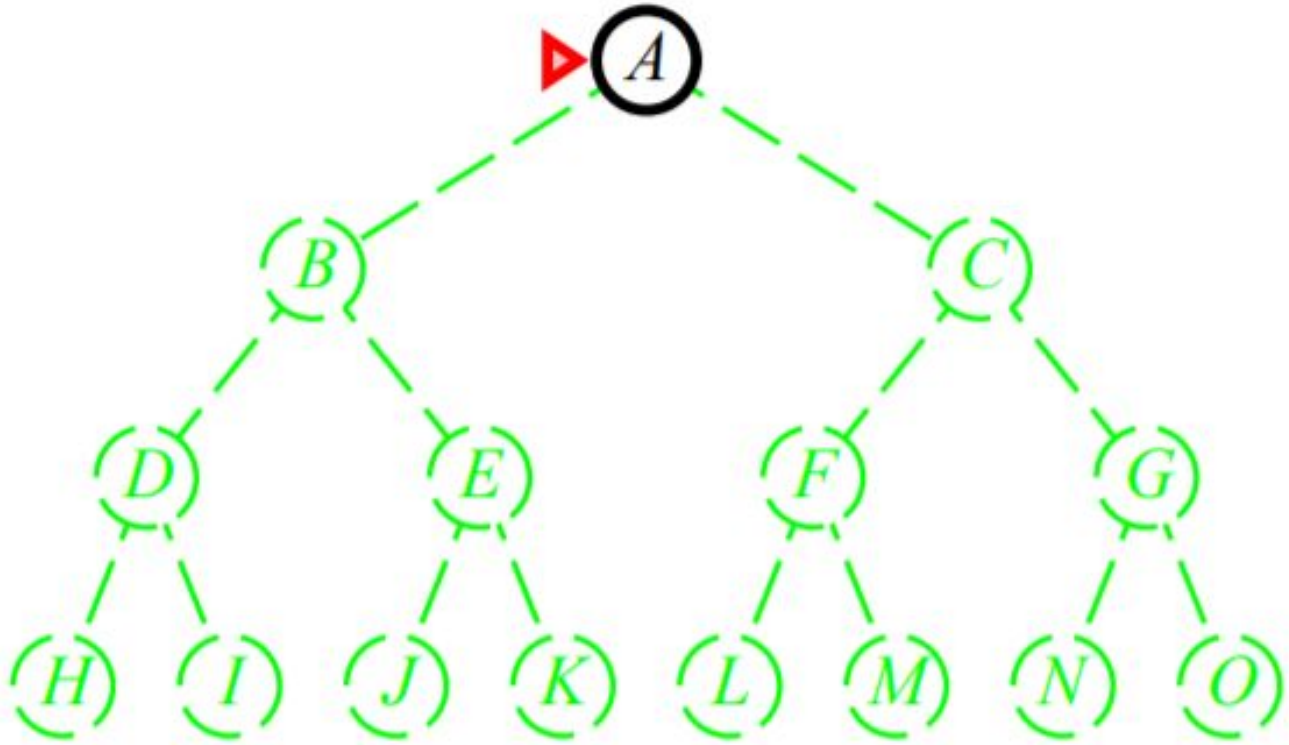
Depth-First Search (DFS)

- Strategy □ Expand deepest unexpanded node

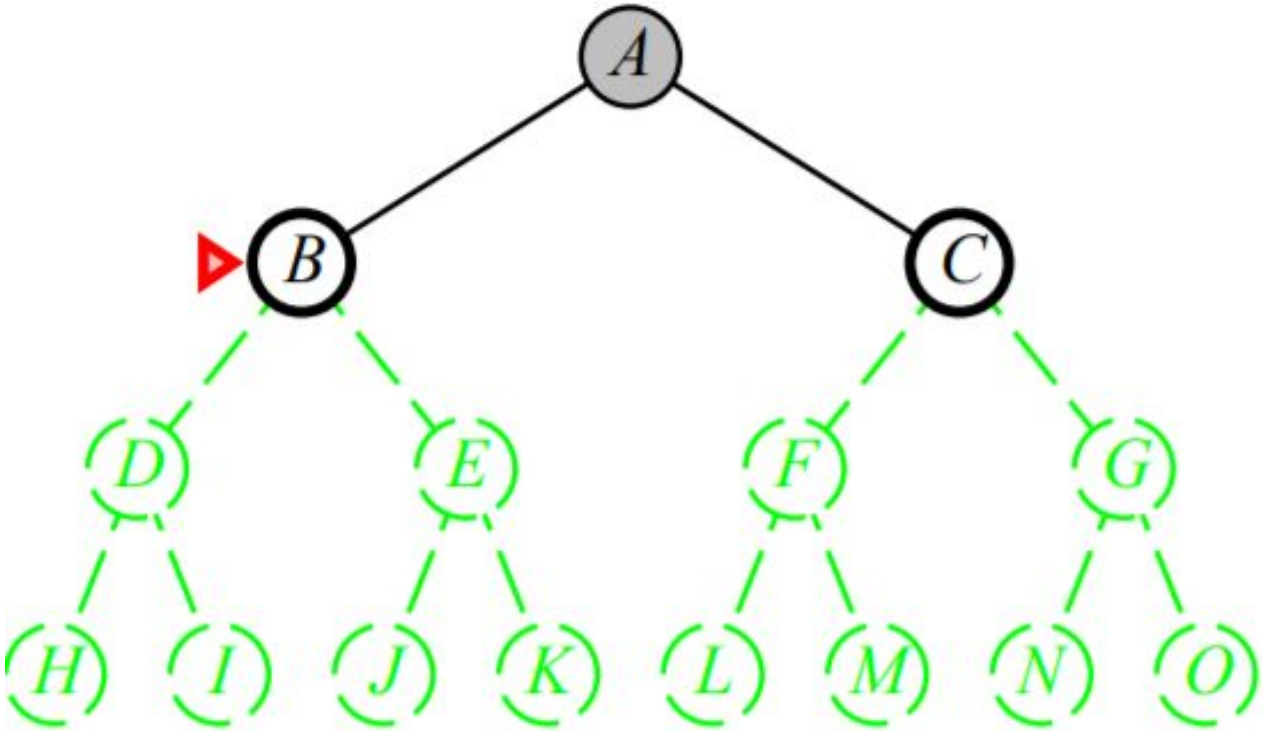
- Implementation □ fringe = LIFO queue, i.e., put successors at front



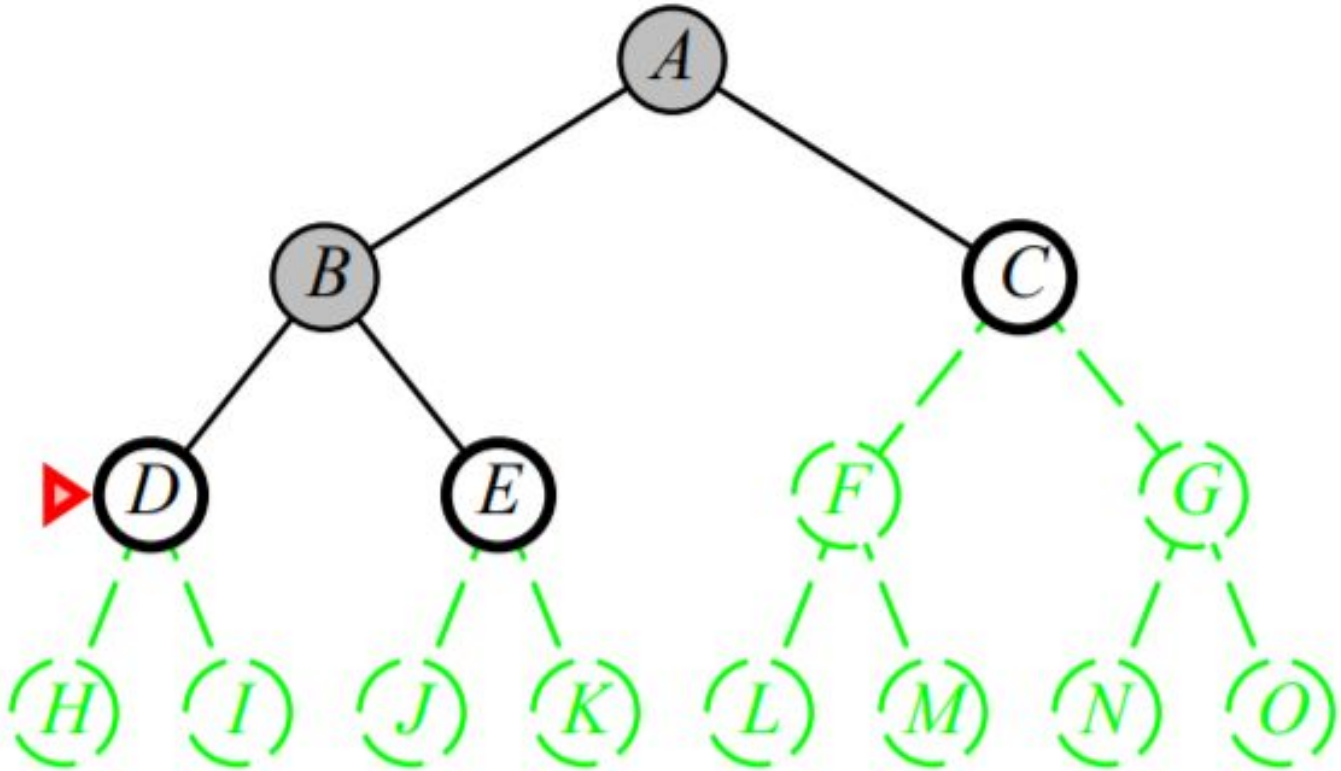
Depth-First Search (DFS)



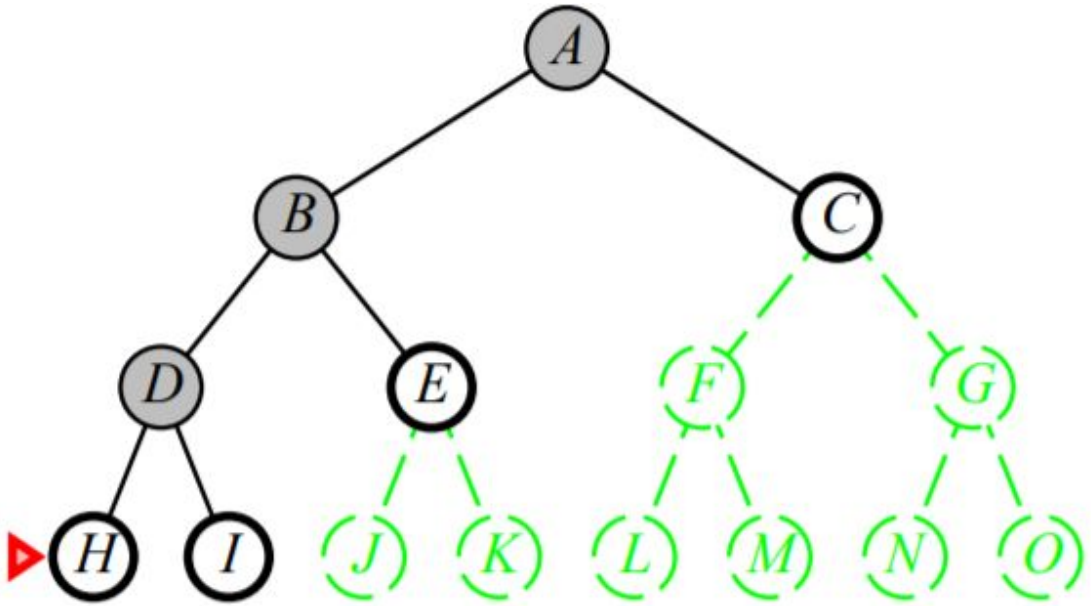
Depth-First Search (DFS)

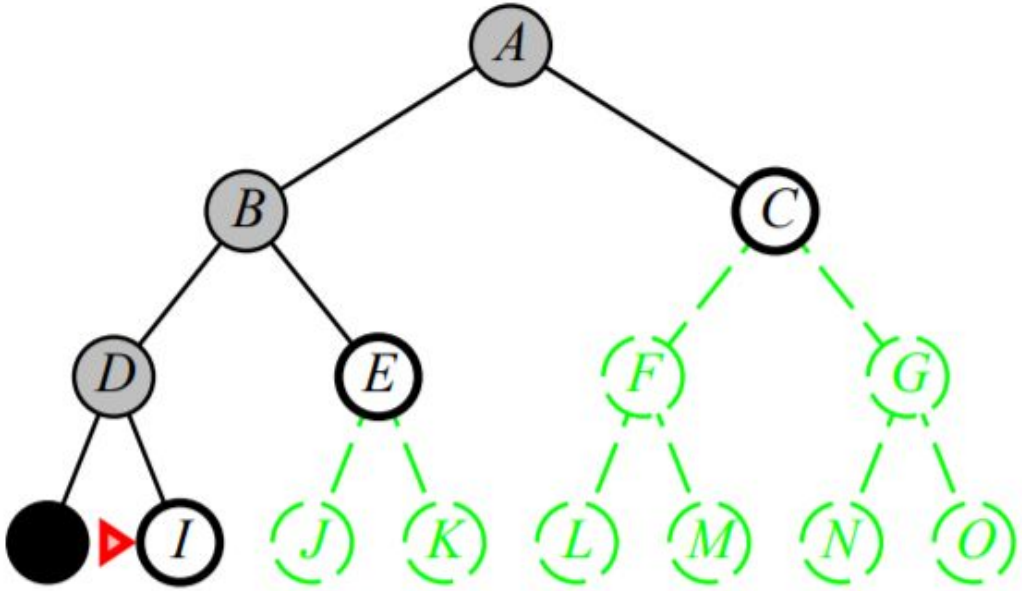


Depth-First Search (DFS)

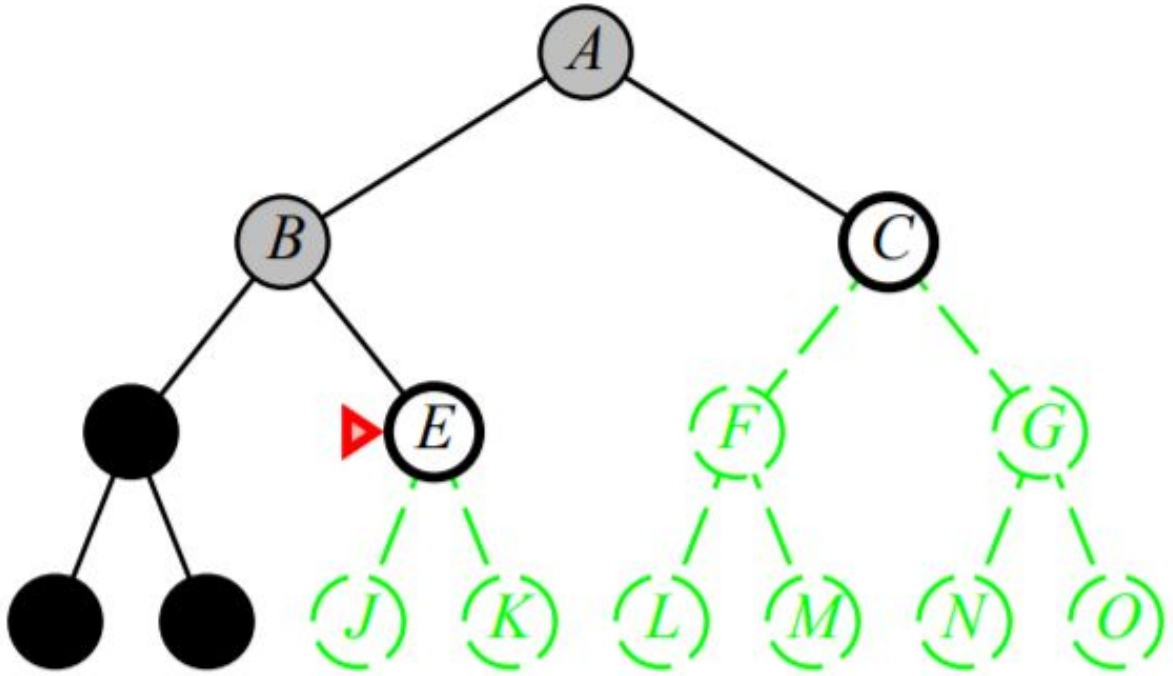


Depth-First Search (DFS)

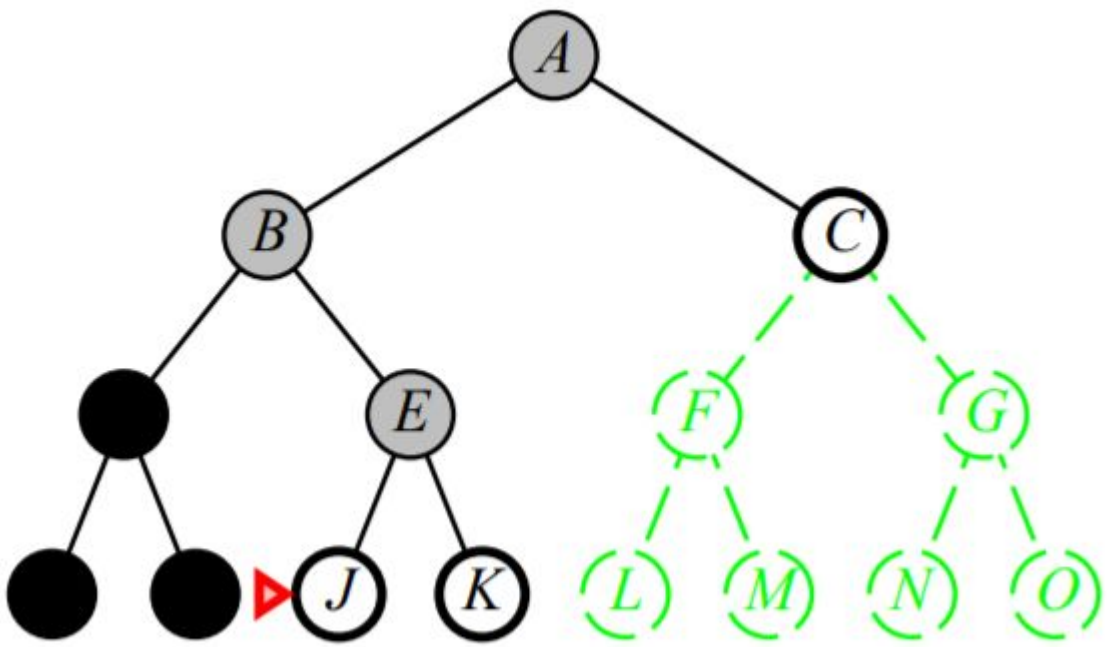


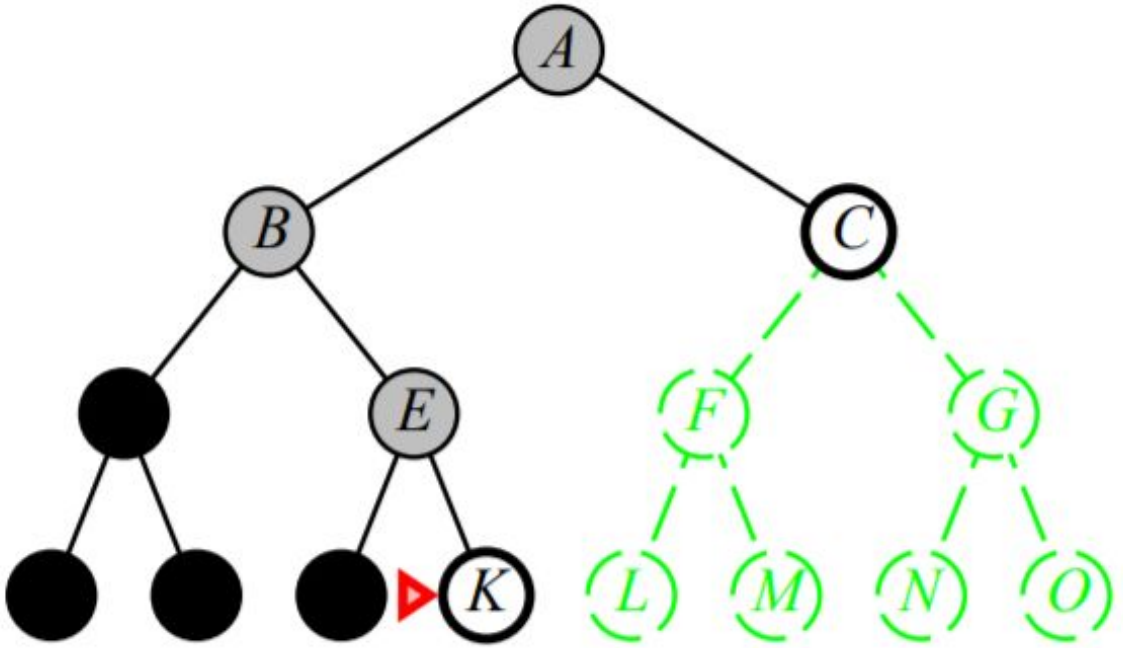


Depth-First Search (DFS)

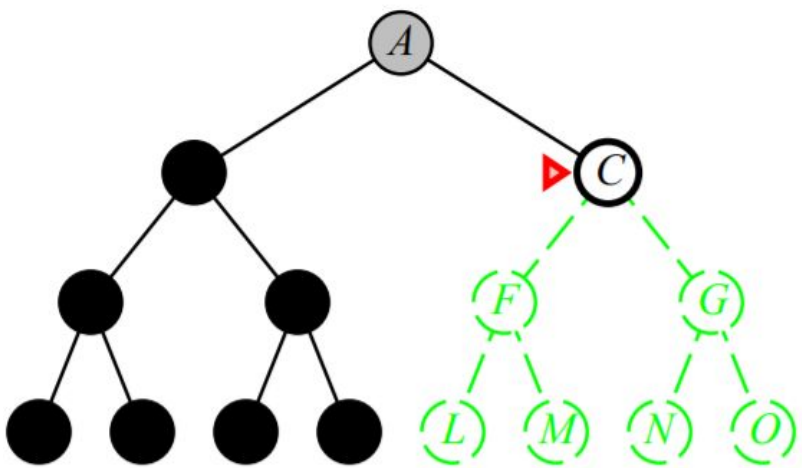


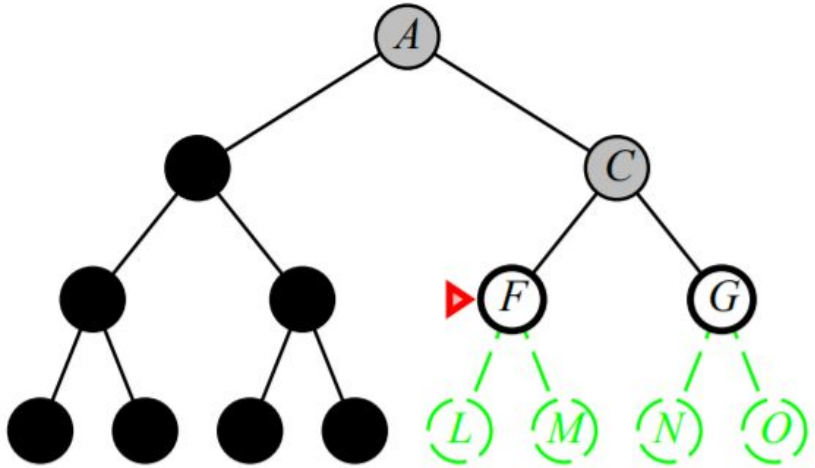
Depth-First Search (DFS)

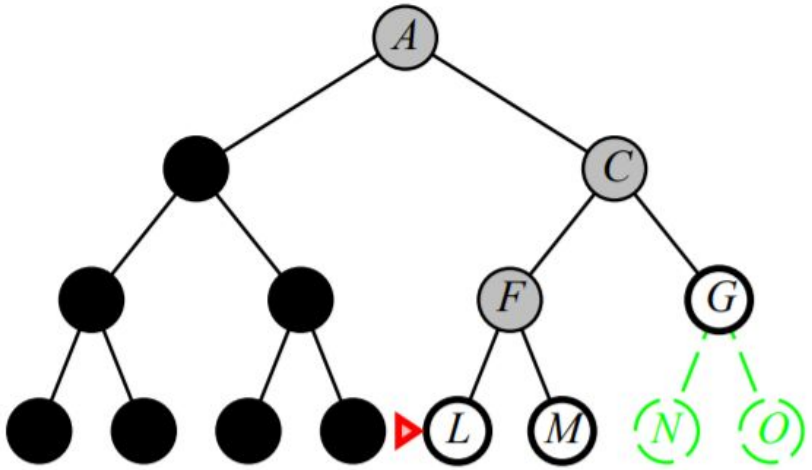


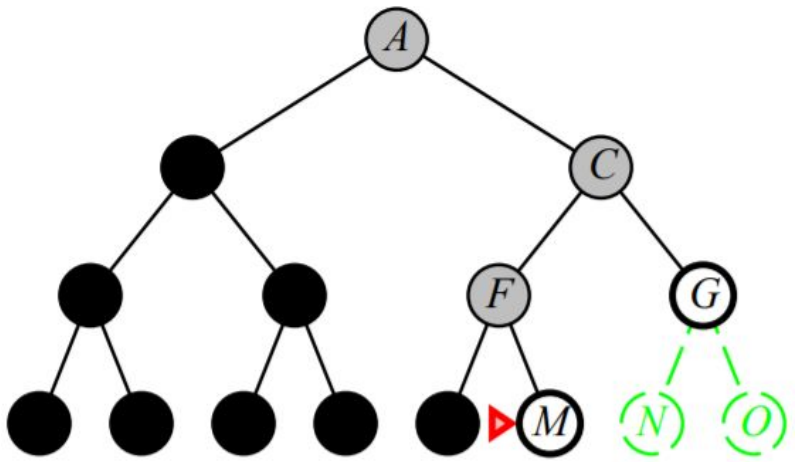


Depth-First Search (DFS)

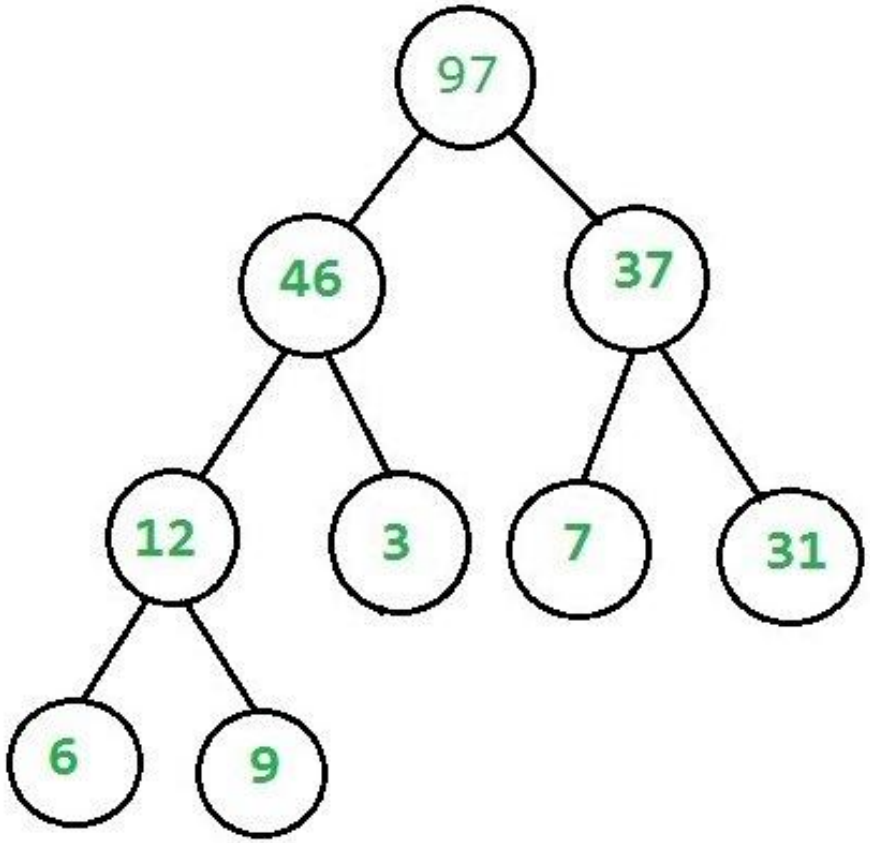




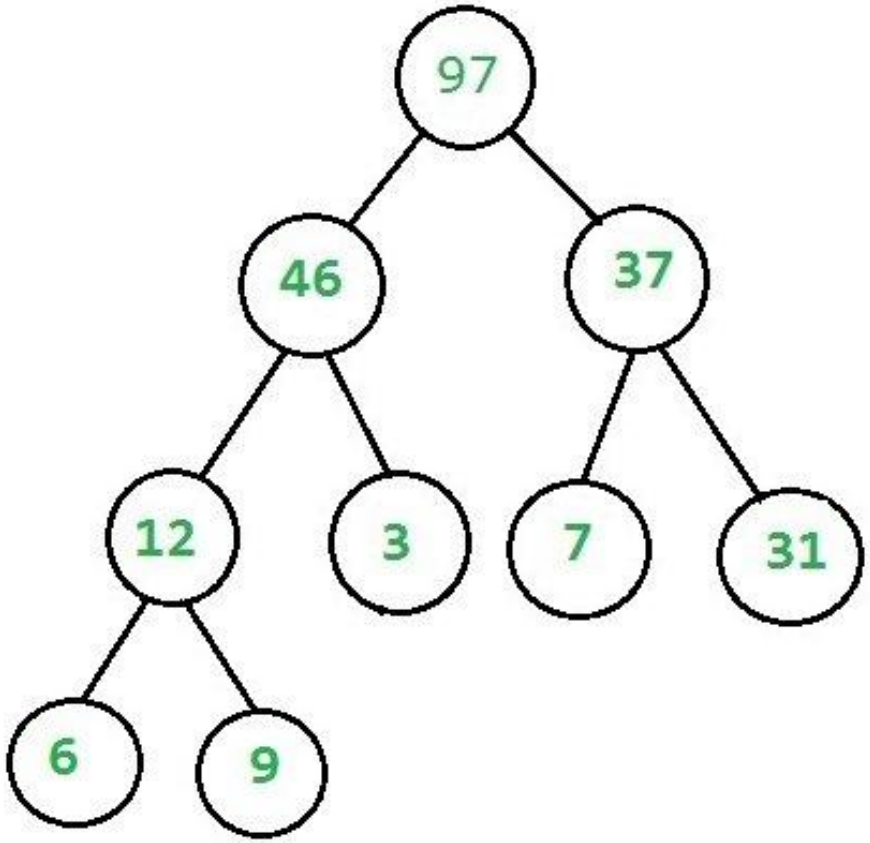




Depth-First Search (DFS)

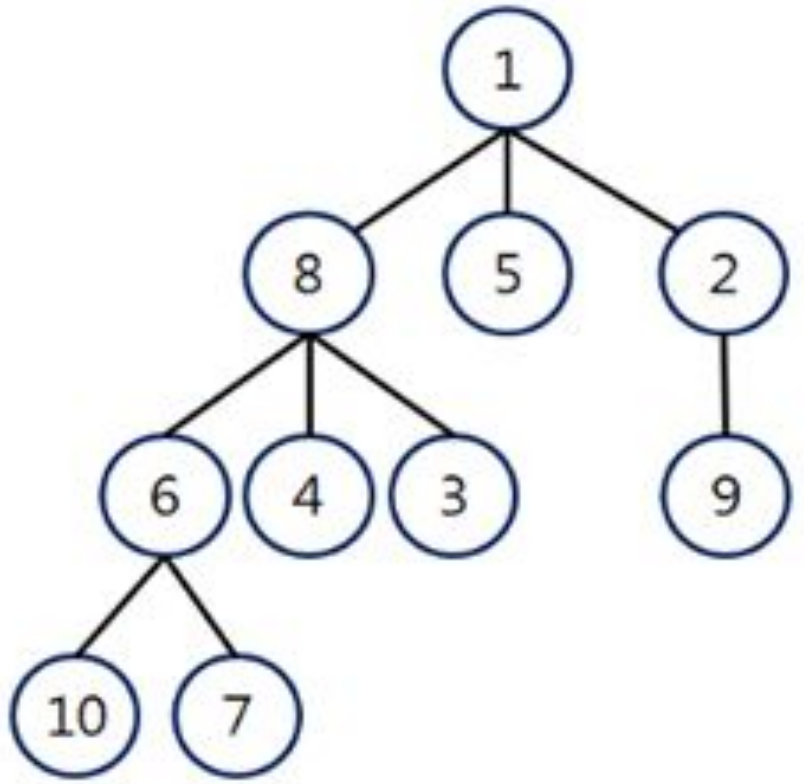


Depth-First Search (DFS)



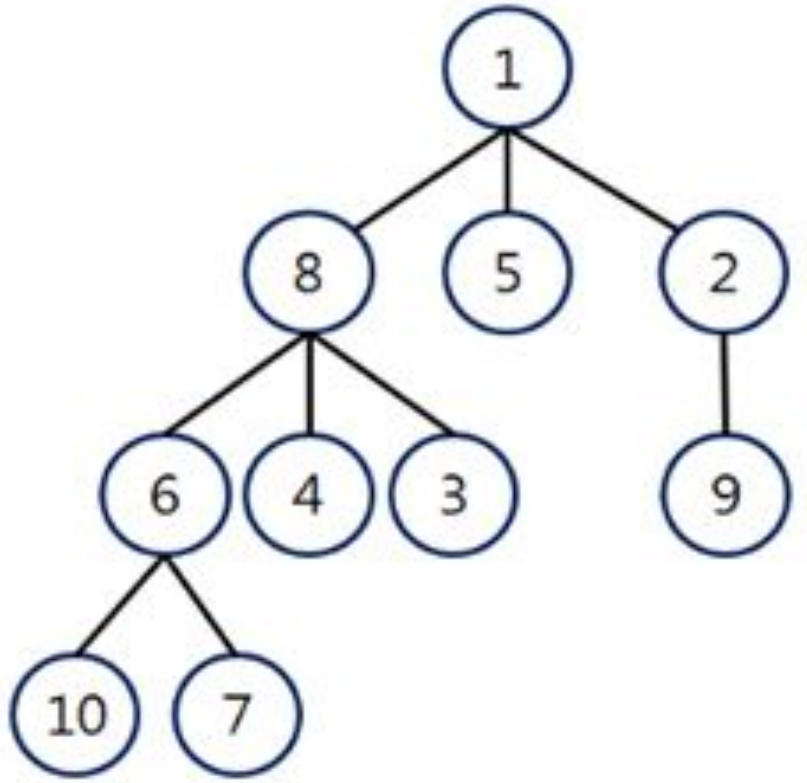
97, 46, 12, 6, 9, 3, 37, 7, 31

Depth-First Search (DFS)

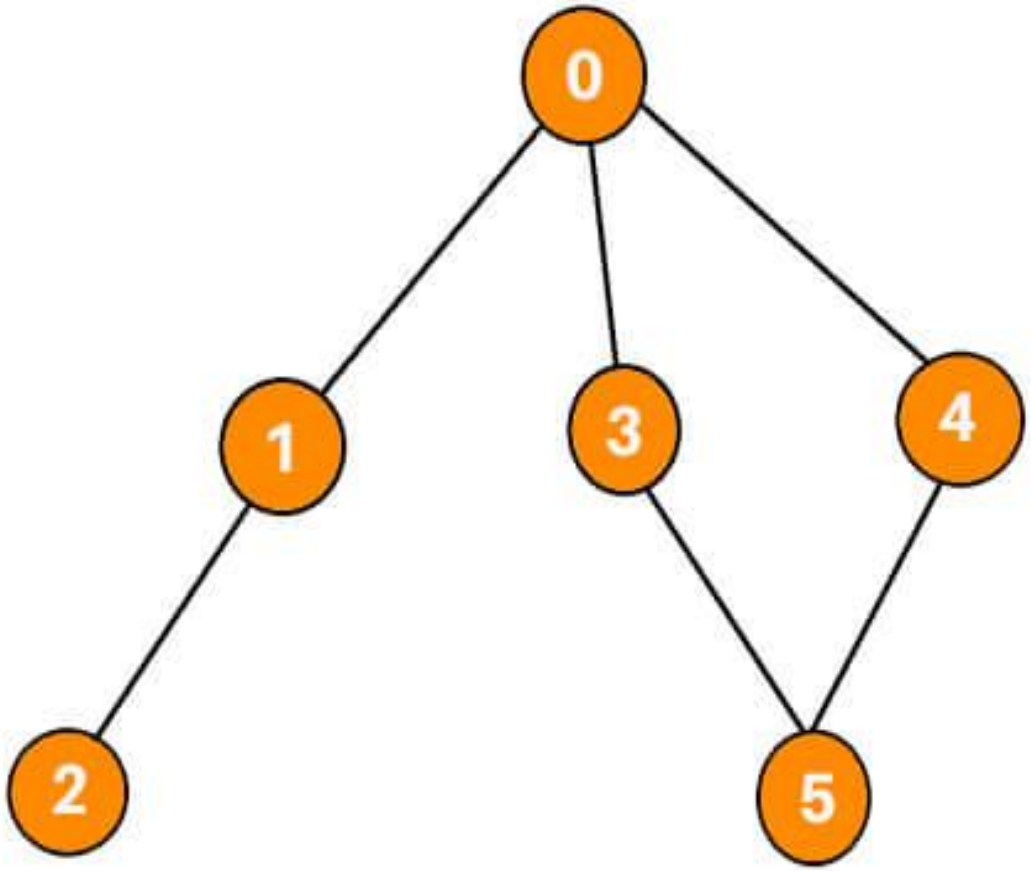


Depth-First Search (DFS)

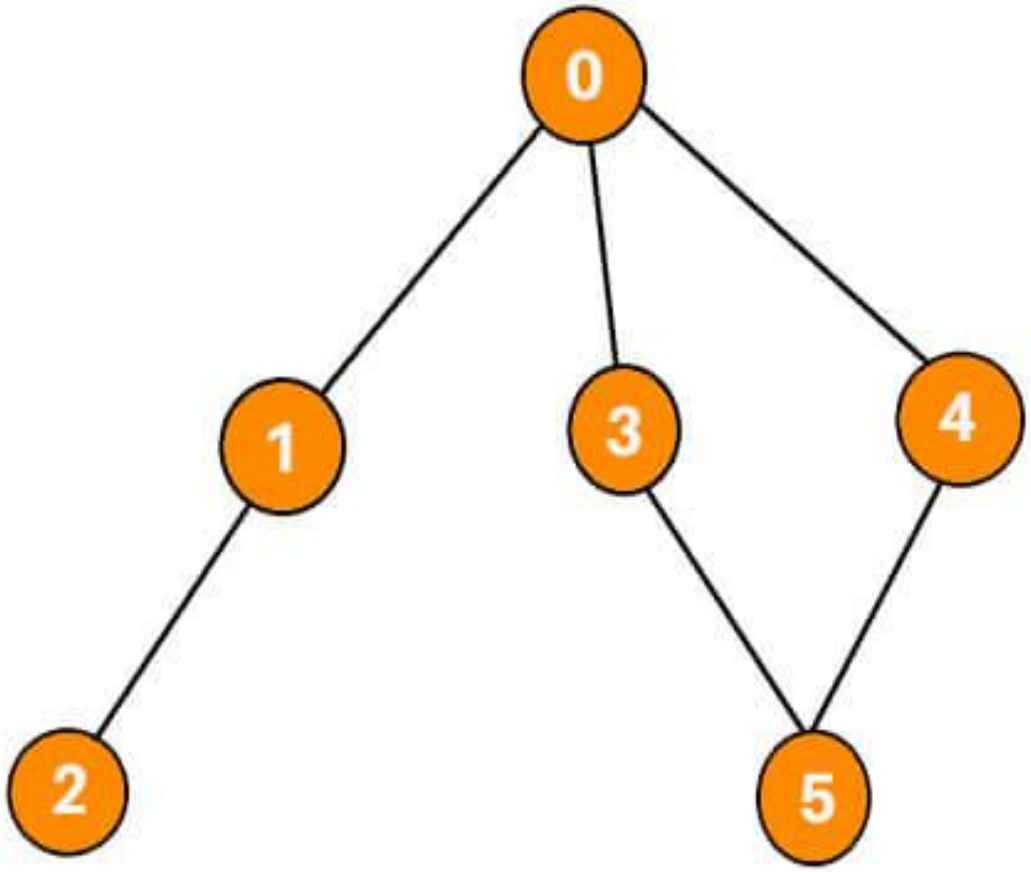
1, 8, 6, 10, 7, 4, 3, 5, 2, 9



Depth-First Search (DFS)



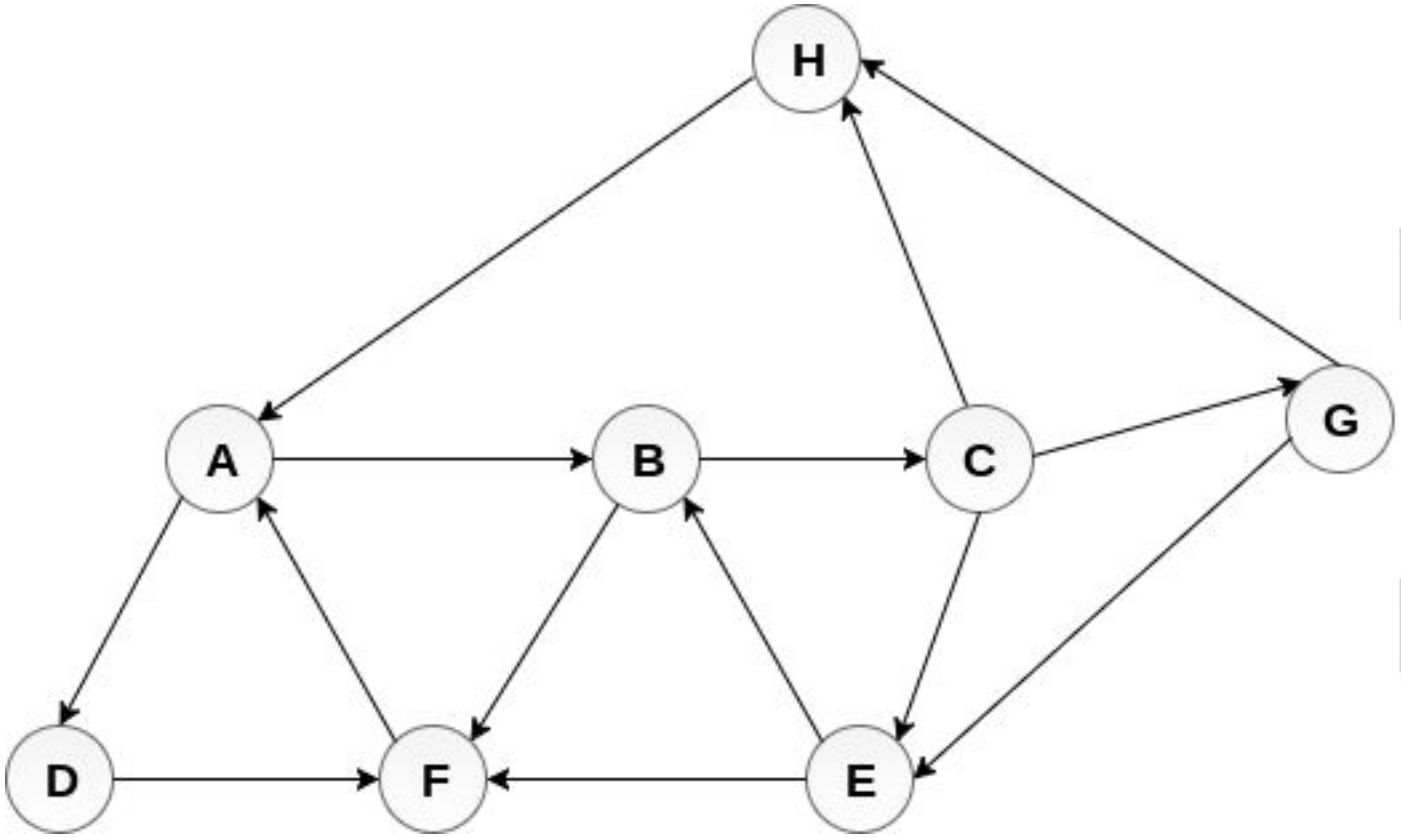
Depth-First Search (DFS)



0, 1, 2, 3, 5, 4

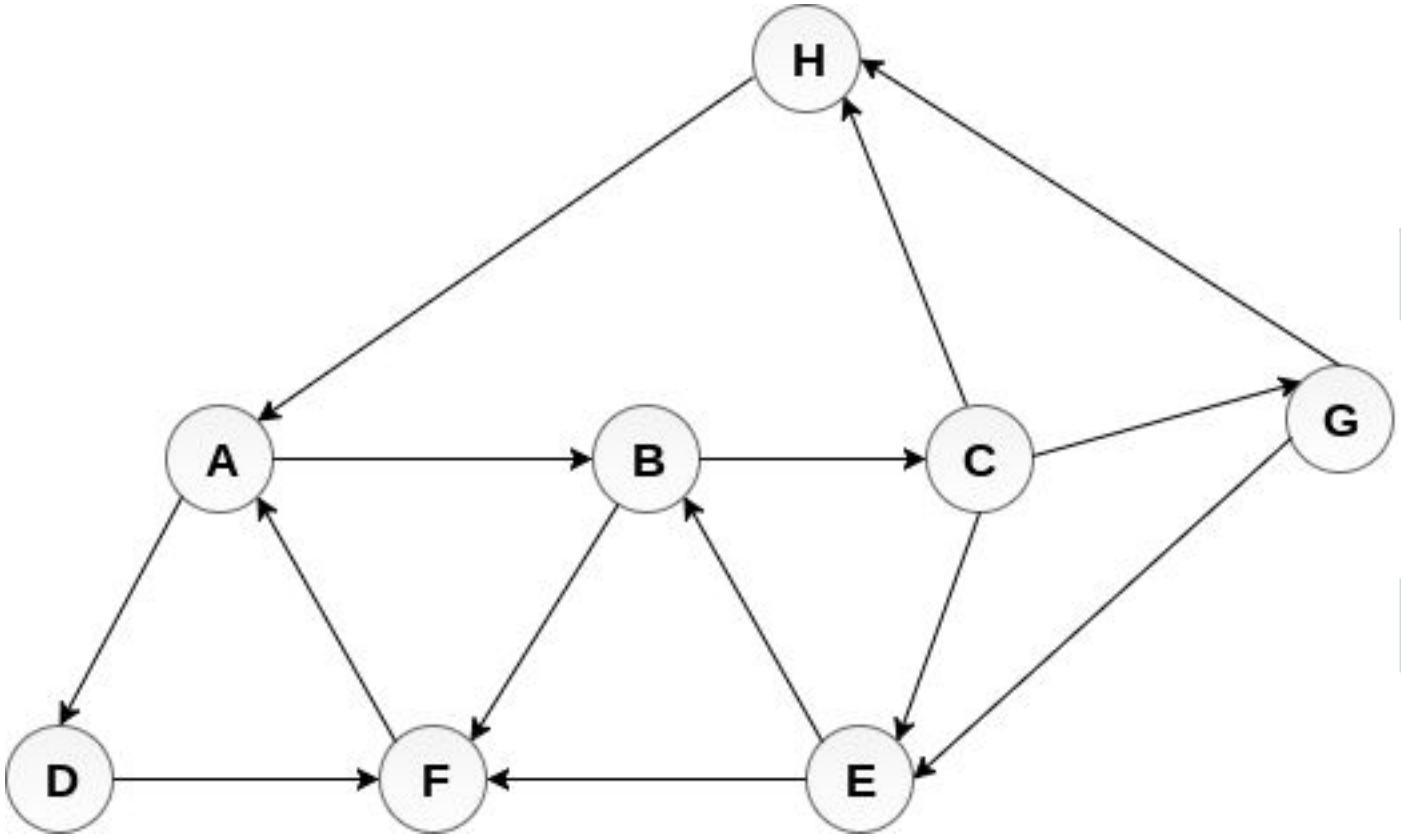
Depth-First Search (DFS)

Initial state or start node is A
leftmost node first



Depth-First Search (DFS)

Initial state or start node is A
leftmost node first



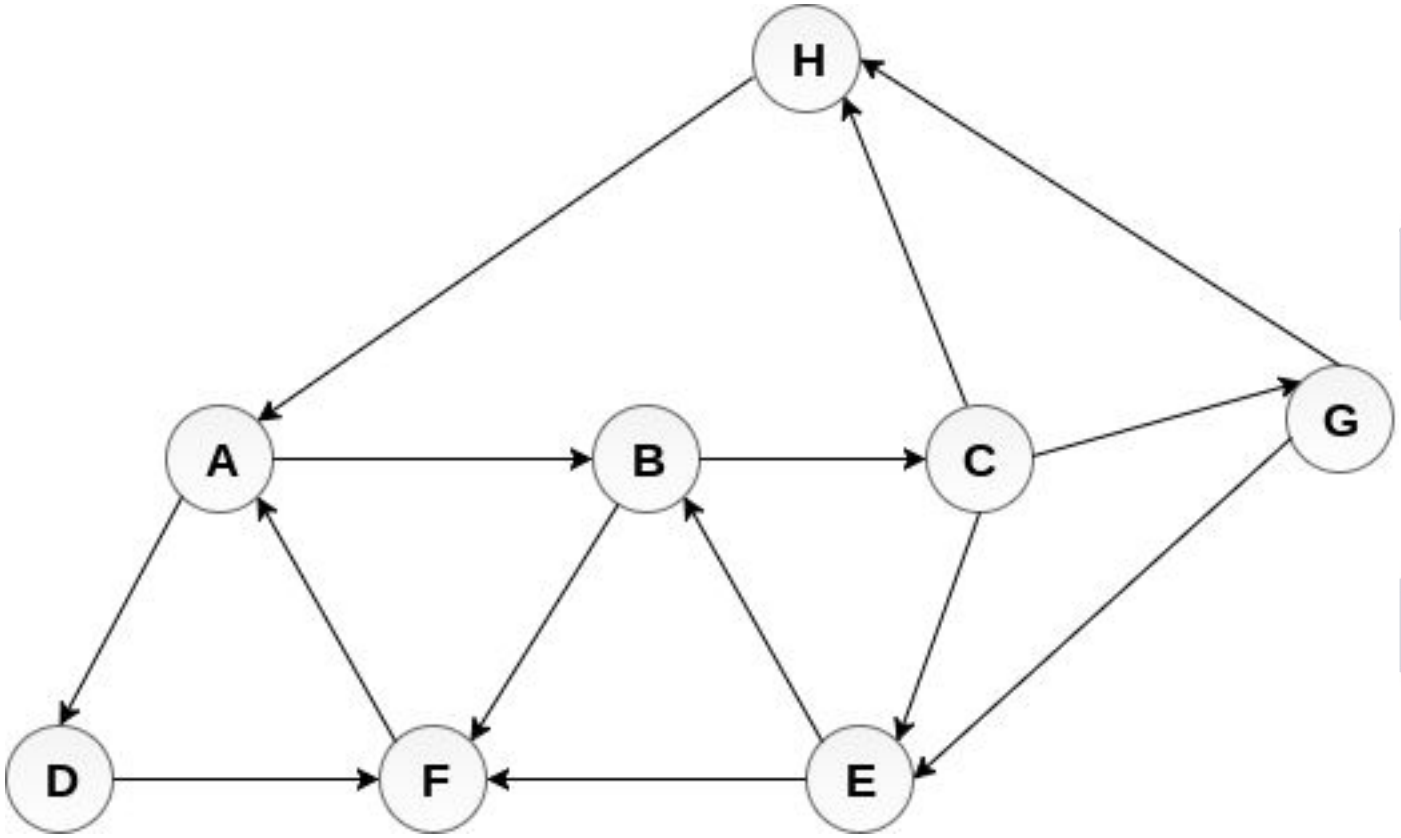
A, D, F, B, C, E, G, H

Depth-First Search (DFS)

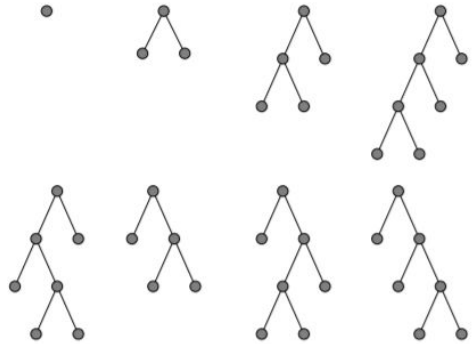
Initial state or start node is A

What the result if

1. rightmost node first
2. alphabetical order first
3. Show the search tree

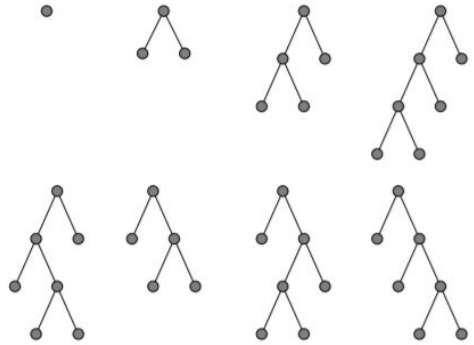


Depth-First Search (DFS)



- Is it complete?
- Is it optimal?
- Time complexity?
- Space complexity?

Depth-First Search (DFS)



- Is it complete?
  - Search tree → Not complete because of infinite loops. Ex. A-B
    - Solution → check new states against those on the path from the root to the current node
  - Graph version → Complete
- Is it optimal?
  - No, it finds the "leftmost" solution, regardless of depth or cost. Ex. DFS will explore the entire left subtree even if node H is the optimal solution. If node F were also a solution, then DFS would return it as a solution instead of the optimal solution.
- Time complexity?
  - $O(b^m)$, m → maximum depth of any node.
- Space complexity?
  - Store only a single path from the root to a leaf node.
  - $O(bm)$.

Uniform-Cost Search

- When all step costs are equal, BFS is optimal because it always expands the shallowest unexpanded node.

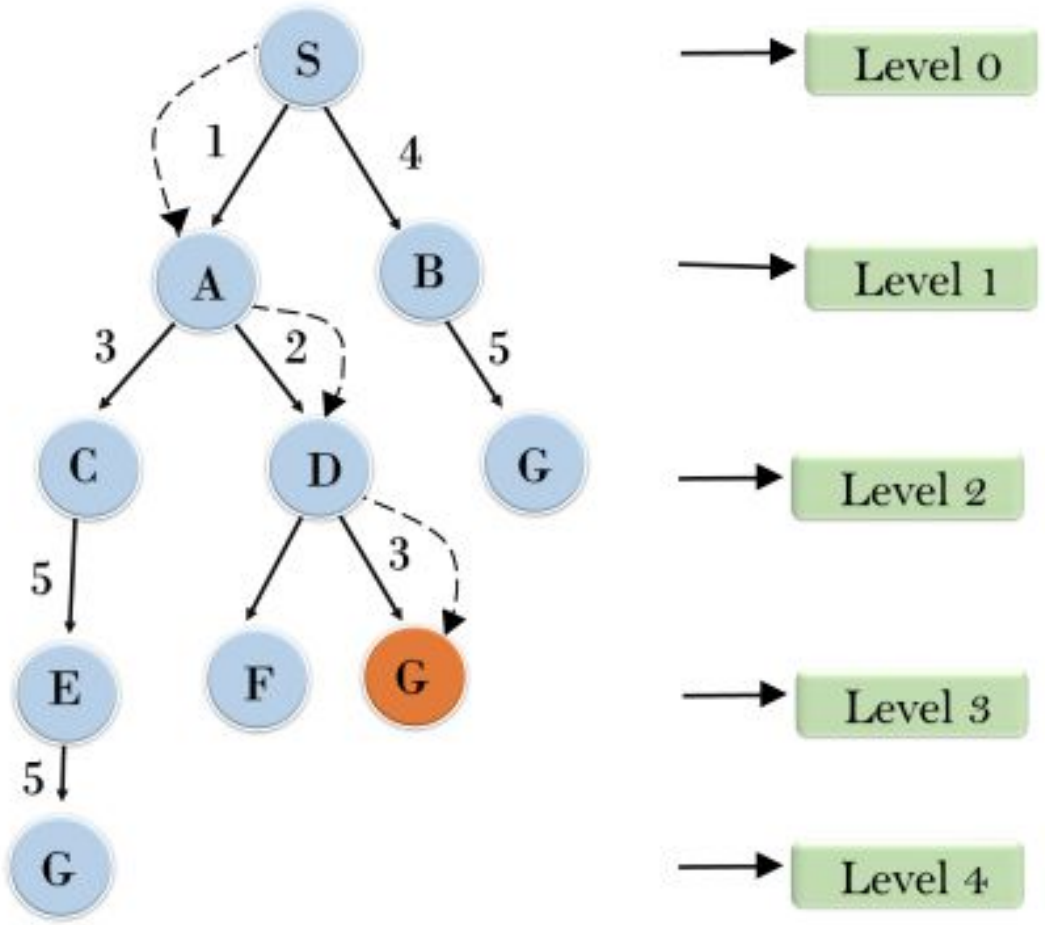
What if each step has a different cost ?

- Uniform-cost search expands the node *n* with the lowest path cost *g(n)*.
  - Find the optimal solution with any step-cost function.

Uniform-Cost Search

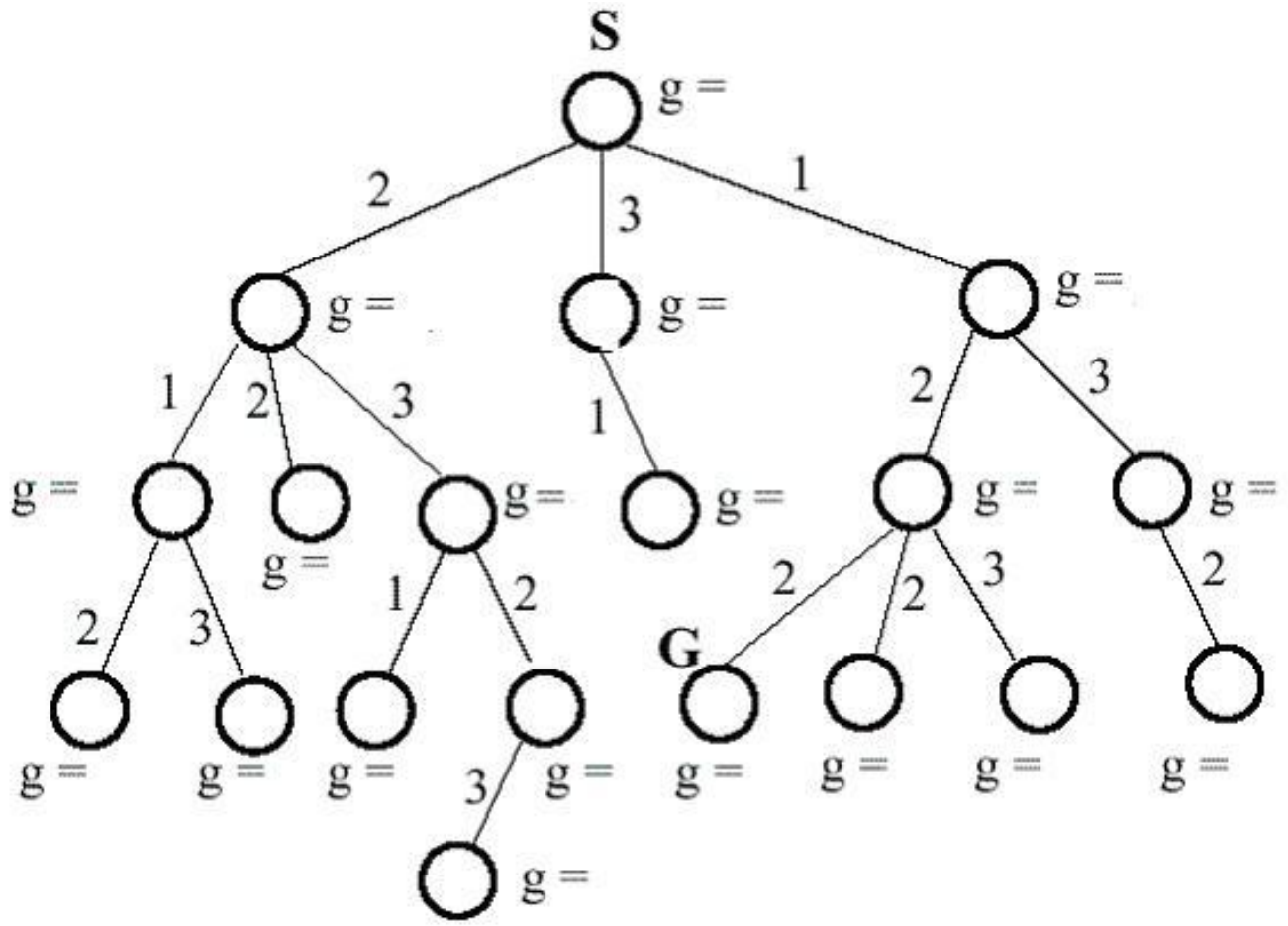
- Strategy □ Expand least-cost unexpanded node

- Implementation □ fringe = priority queue ordered by path cost



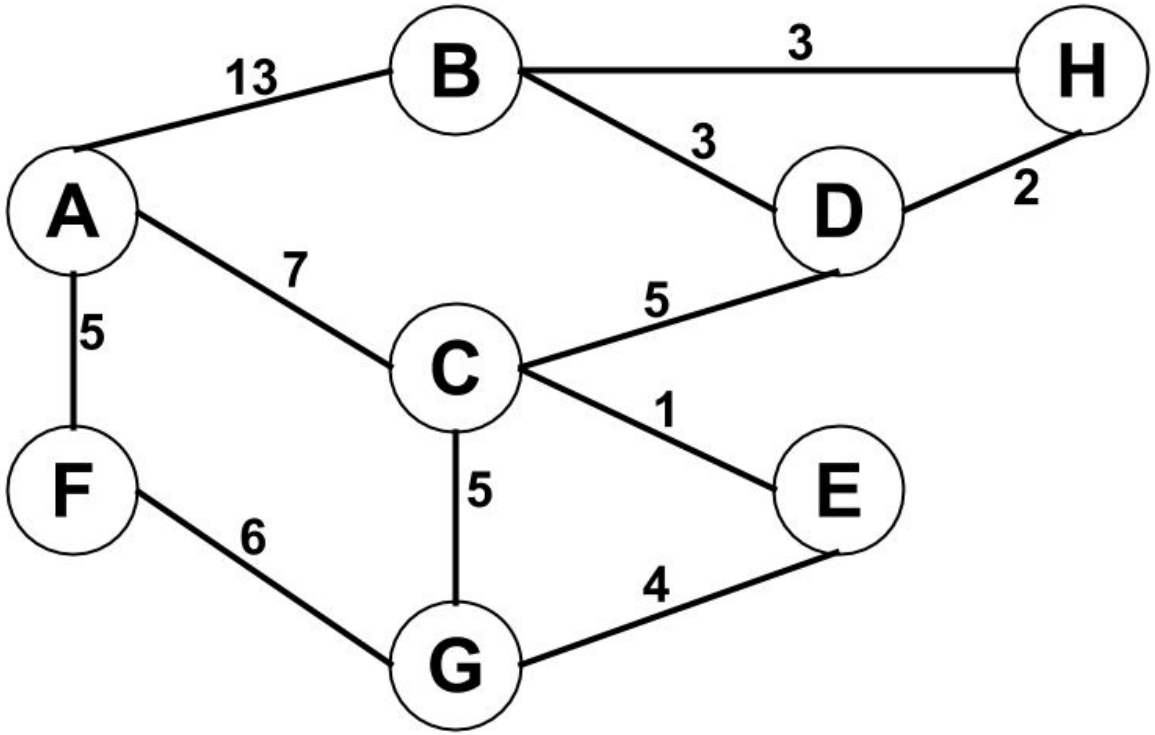
```
function UNIFORM-COST-SEARCH(problem) returns a solution, or failure

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

Uniform-Cost Search



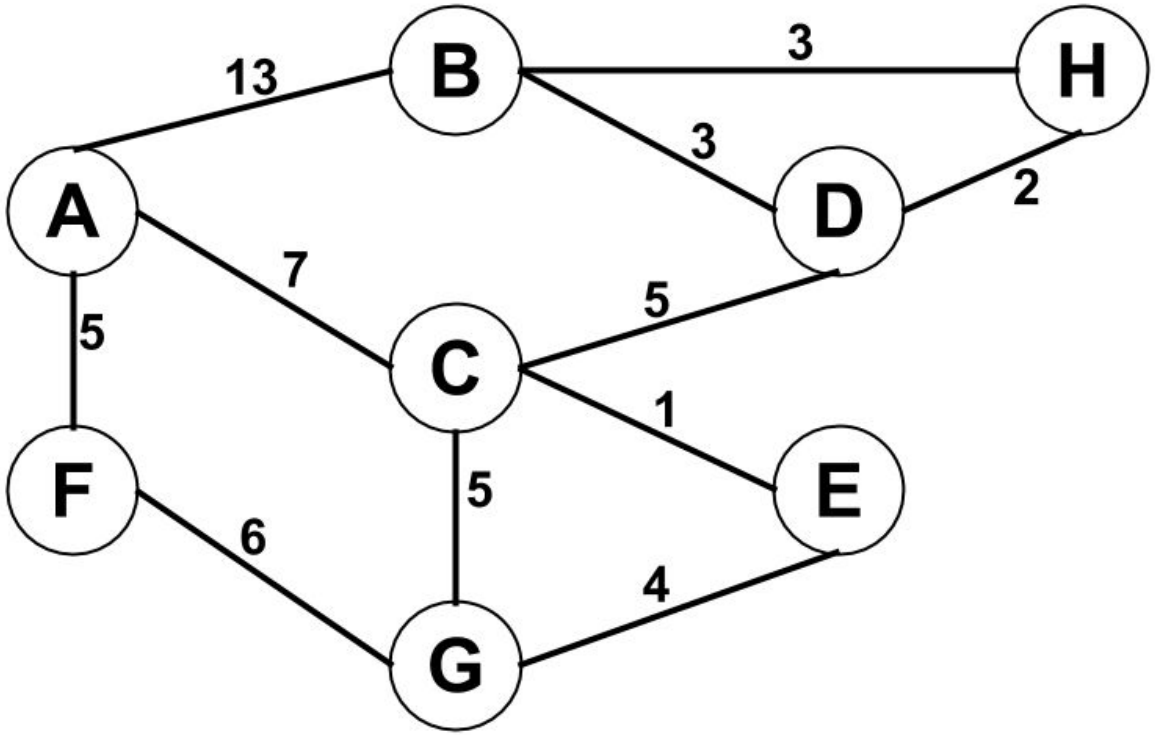
Uniform-Cost Search

Find the pat from A to H



Uniform-Cost Search

Find the pat from A to H

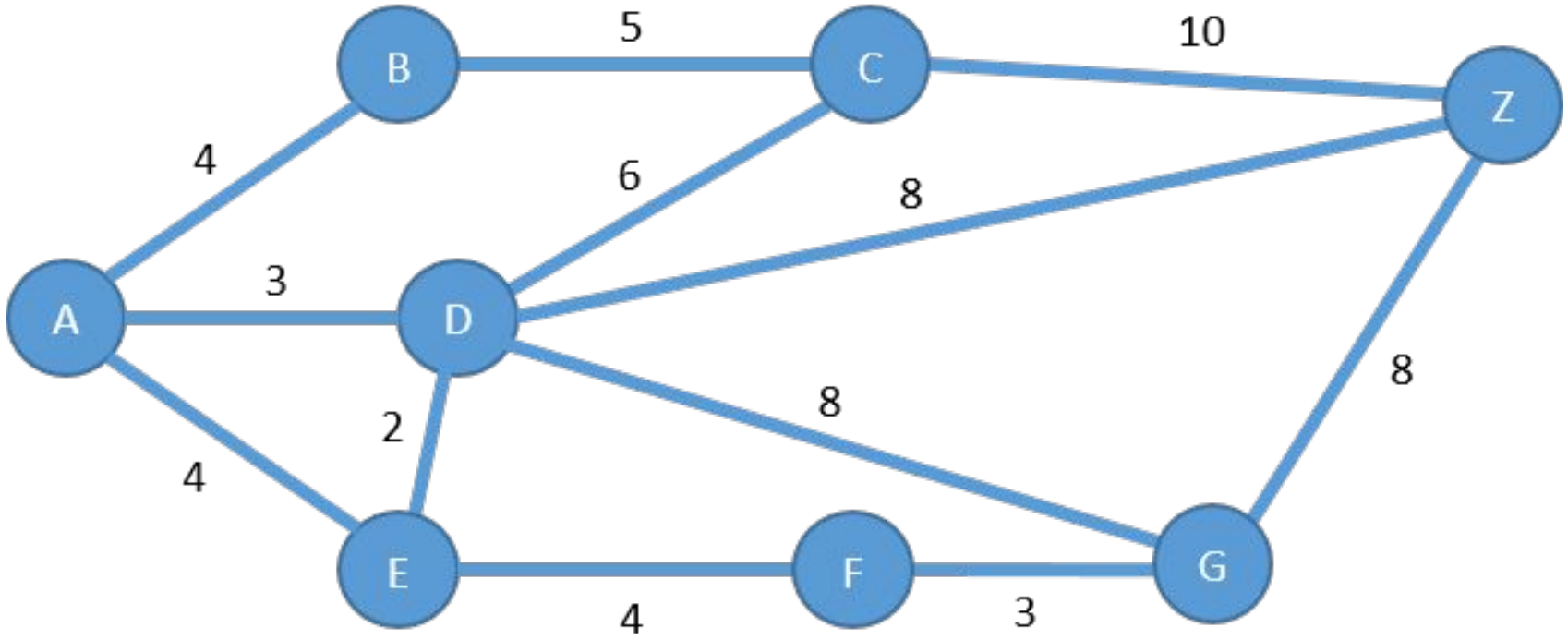


Path: A, C, D, H

Path Cost: 14

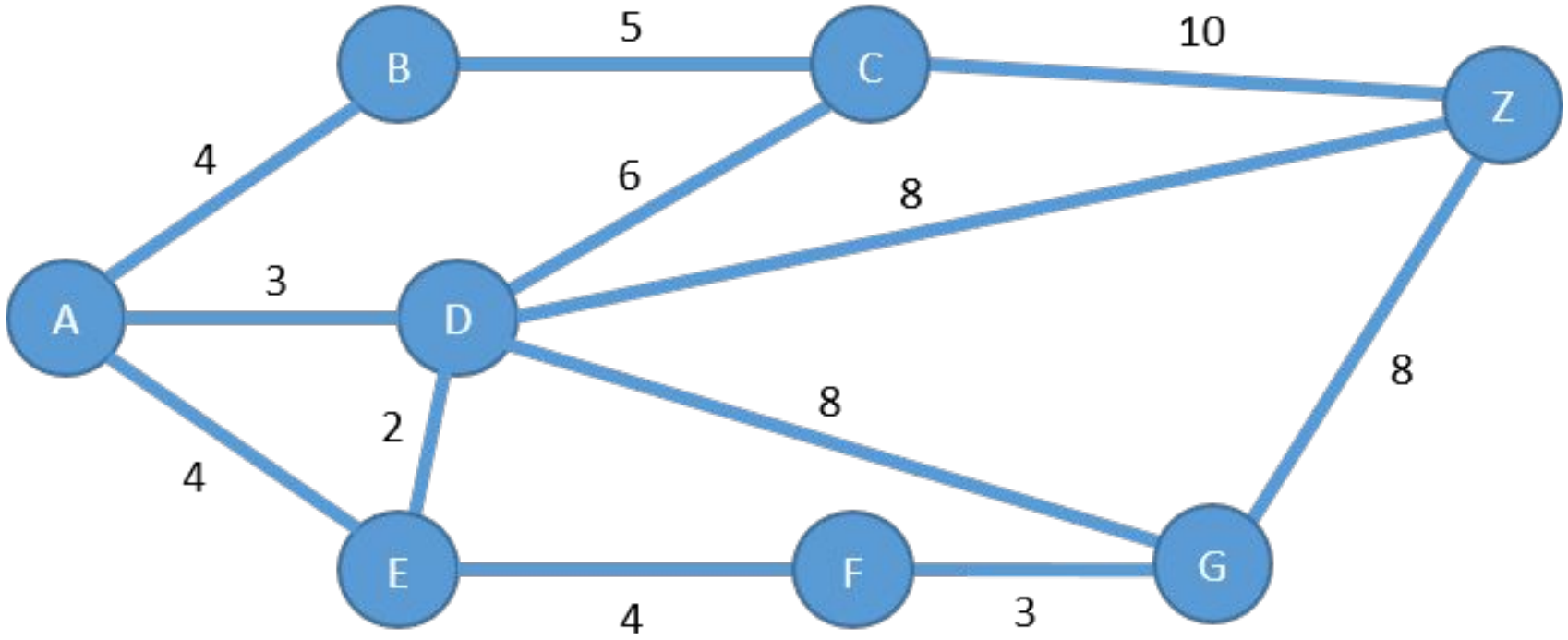
Uniform-Cost Search

Find the pat from A to G



Uniform-Cost Search

Find the pat from A to G



Path: A, D, G

Path Cost: 11

Uniform-Cost Search

- Is it complete?
- Is it optimal?
- Time complexity?
- Space complexity?

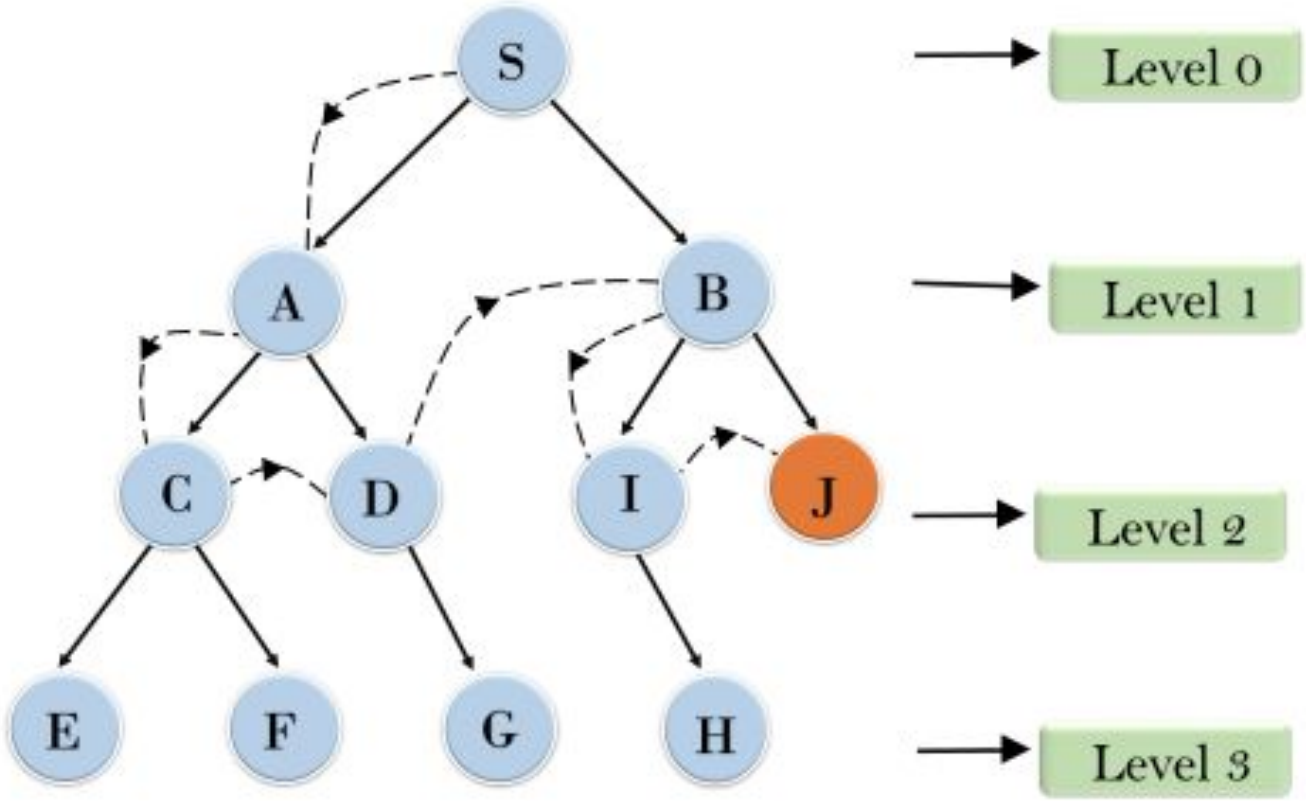
Uniform-Cost Search

- Is it complete?
  - It will get stuck in an infinite loop if there's a path with an infinite sequence of zero-cost actions.
  - Complete only if the cost of every step is some positive number.
- Is it optimal?
  - In general, optimal.
- Time complexity?
  - $O\left(b^{(C^*/\varepsilon)}\right)$, C^* → the optimal path cost of the solution, ε → the least step cost.
- Space complexity?
  - $O\left(b^{(C^*/\varepsilon)}\right)$.

Depth-Limited Search

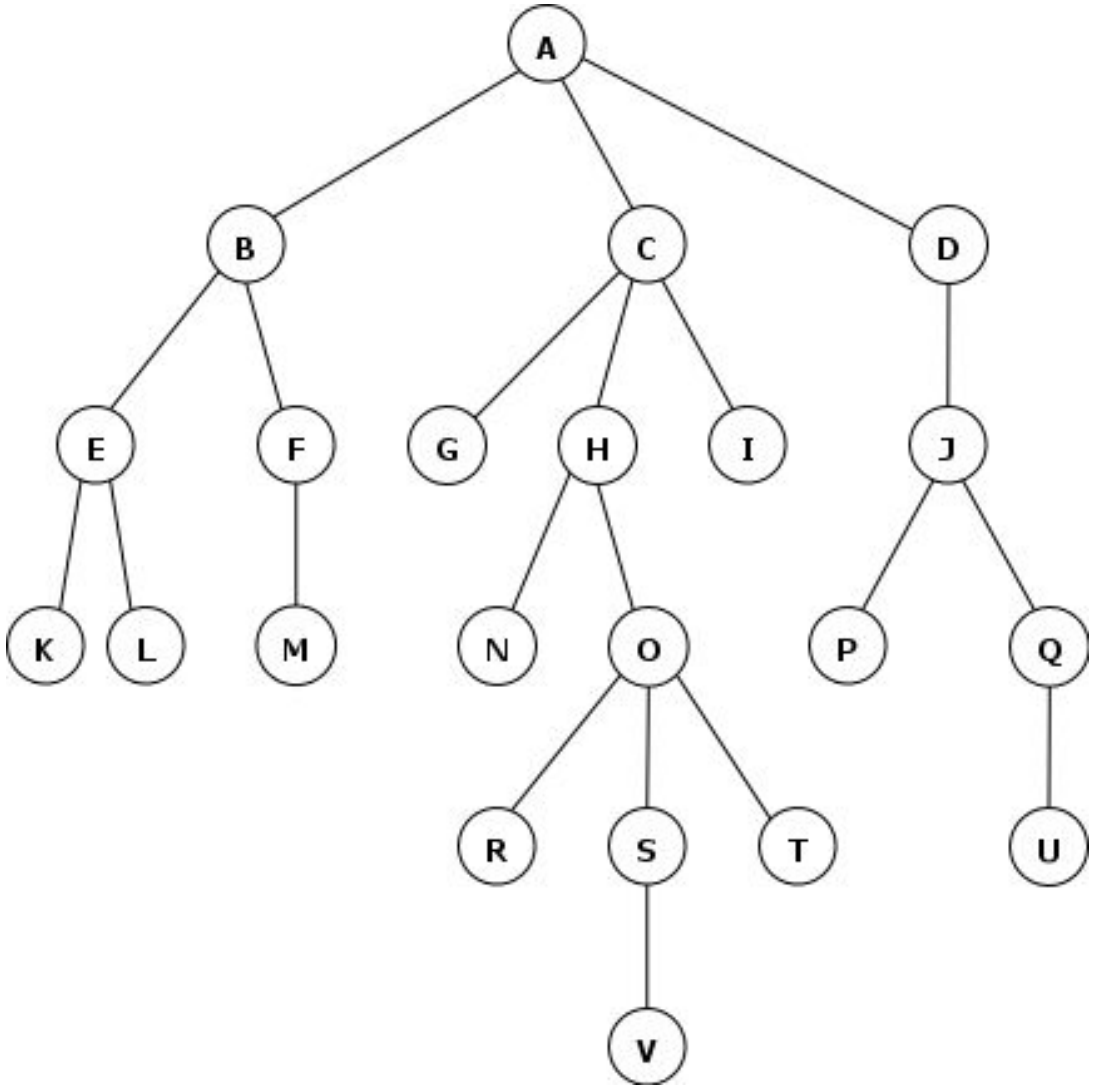
- DFS fails in infinite state spaces can be alleviated by supplying DFS with a predetermined depth limit.

- Depth-limited search → DFS with depth limit (l).



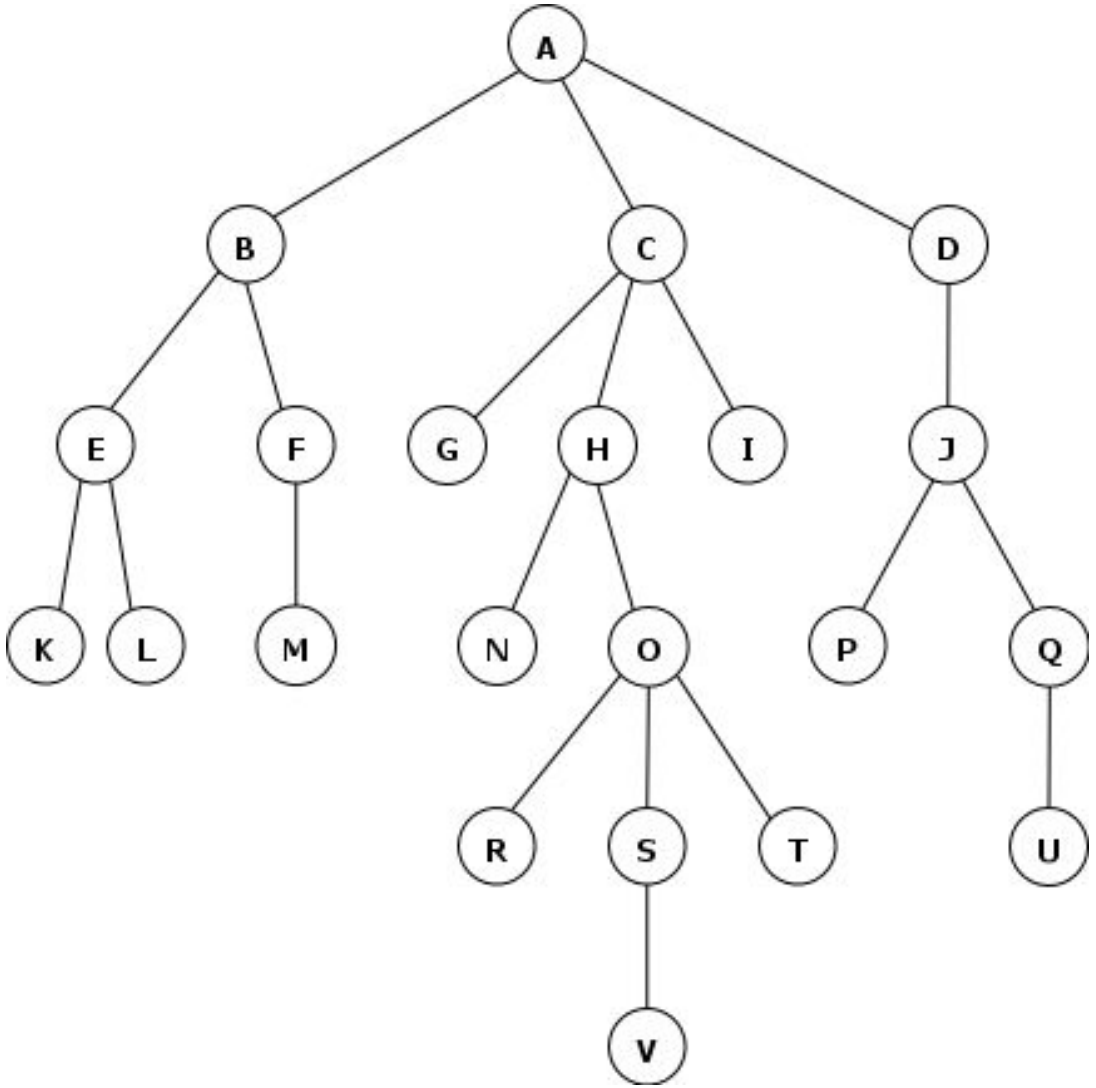
Depth-Limited Search

- $l = 3$
- Goal state: O



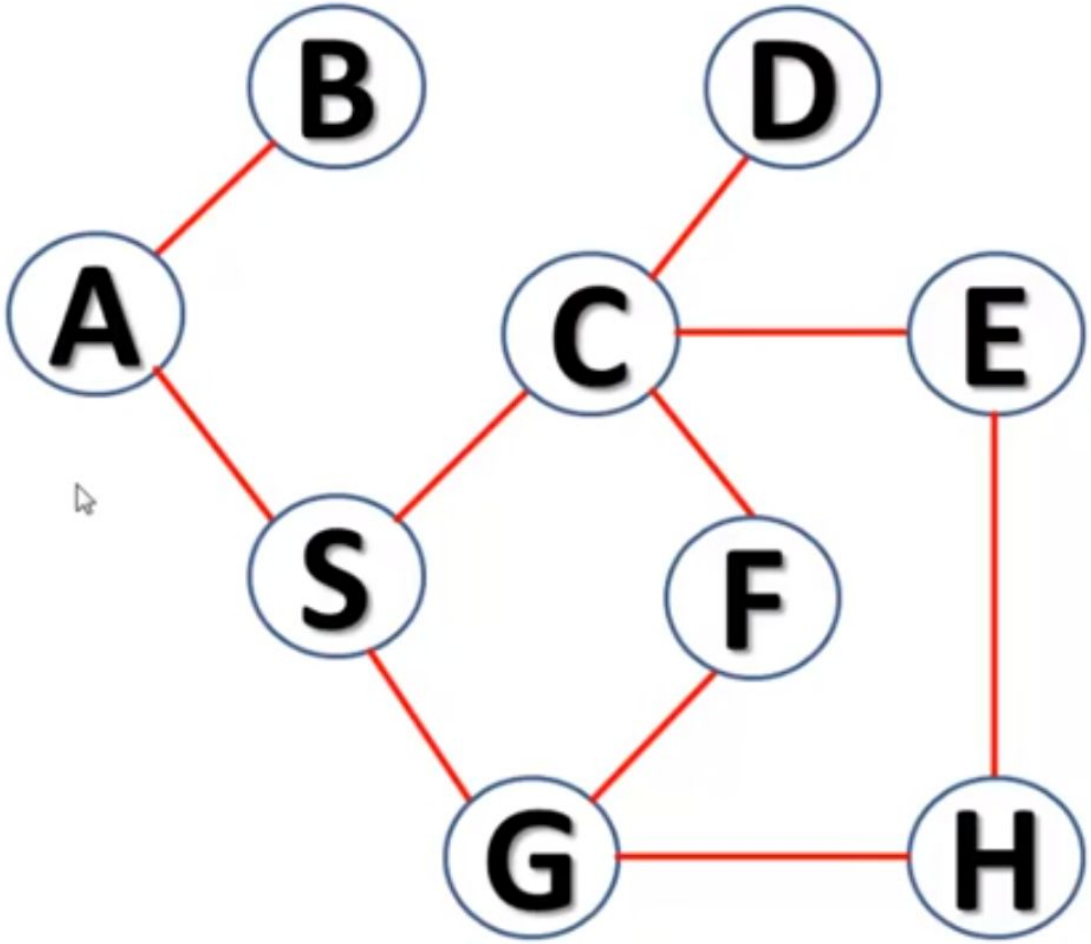
Depth-Limited Search

- $l = 3$
- Goal state: S



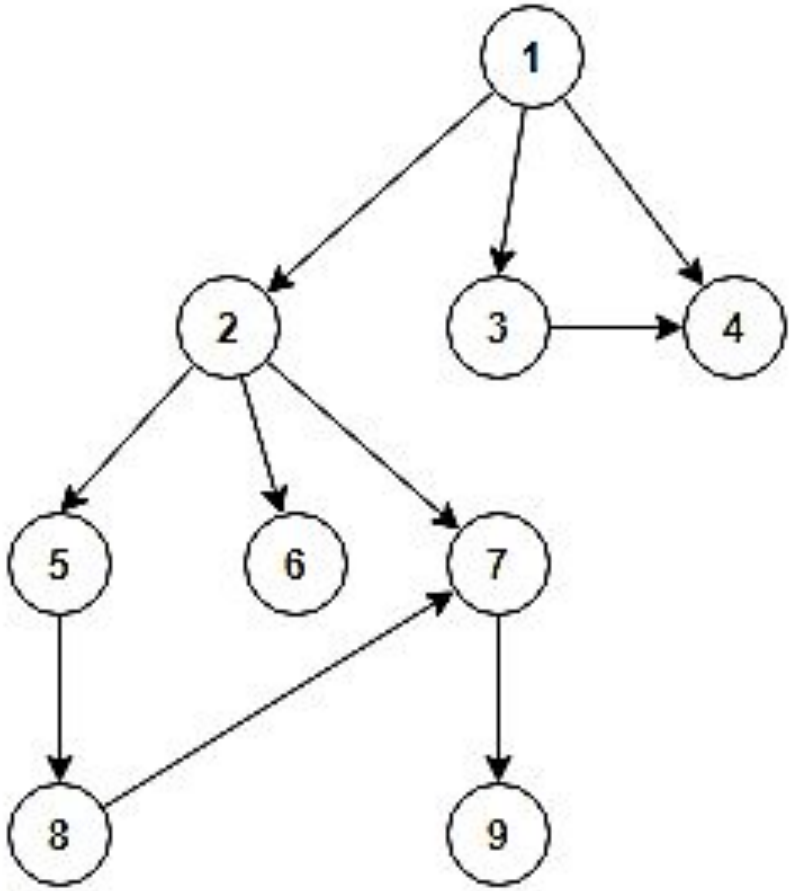
Depth-Limited Search

- $l = 2$
- Goal state: E



Depth-Limited Search

- $l = 2$
- Goal state: 8



Depth-Limited Search

- Is it complete?
- Is it optimal?
- Time complexity?
- Space complexity?

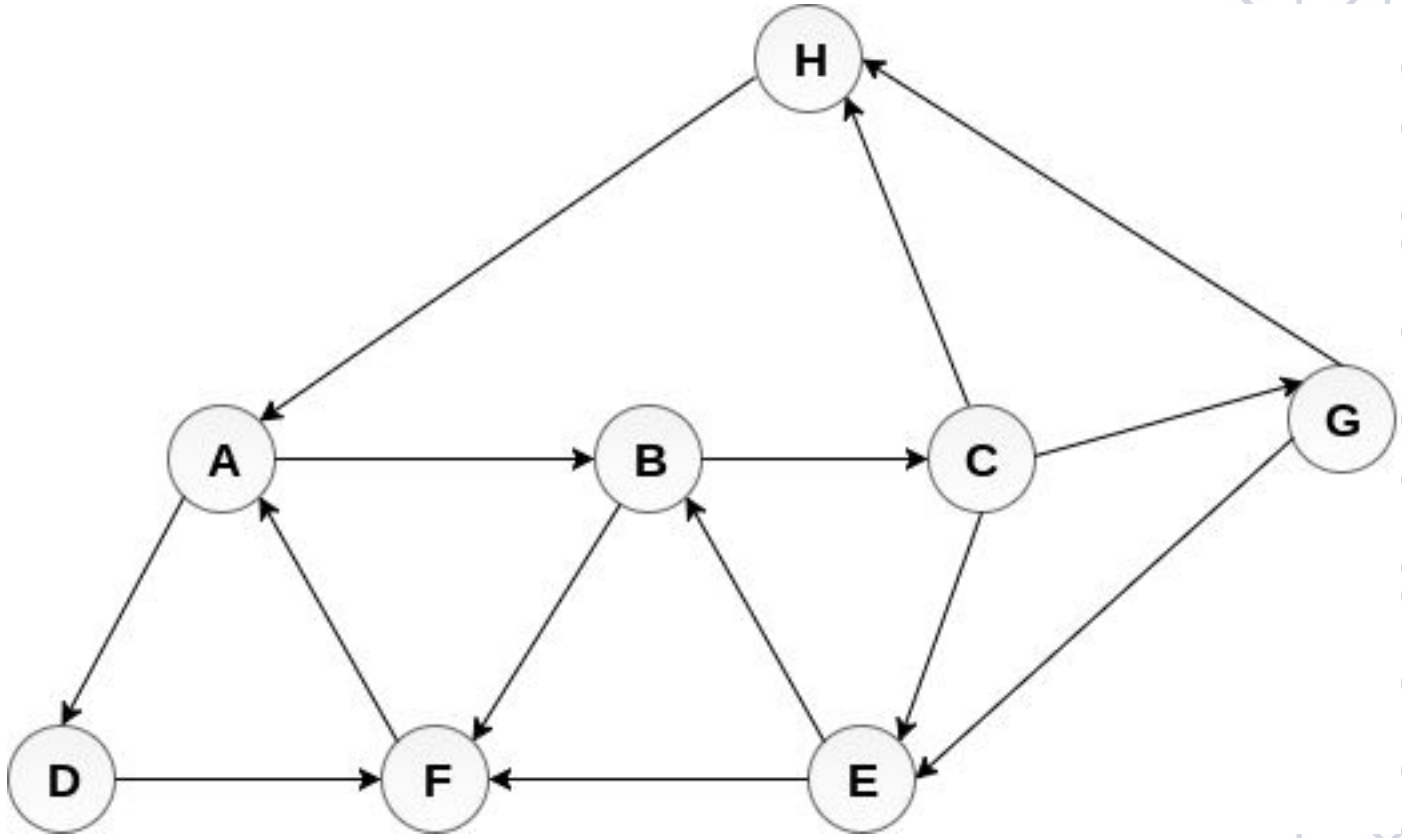
Depth-Limited Search

- Is it complete?
  - Complete in general except the case when the goal node is the shallowest node.
- Is it optimal?
  - Not optimal.
- Time complexity?
  - $O(b^l)$, $l \rightarrow$ depth limit.
- Space complexity?
  - $O(bl)$.

Breadth-First Search (BFS)

Initial state or start node is A and the Goal is E
use leftmost node first

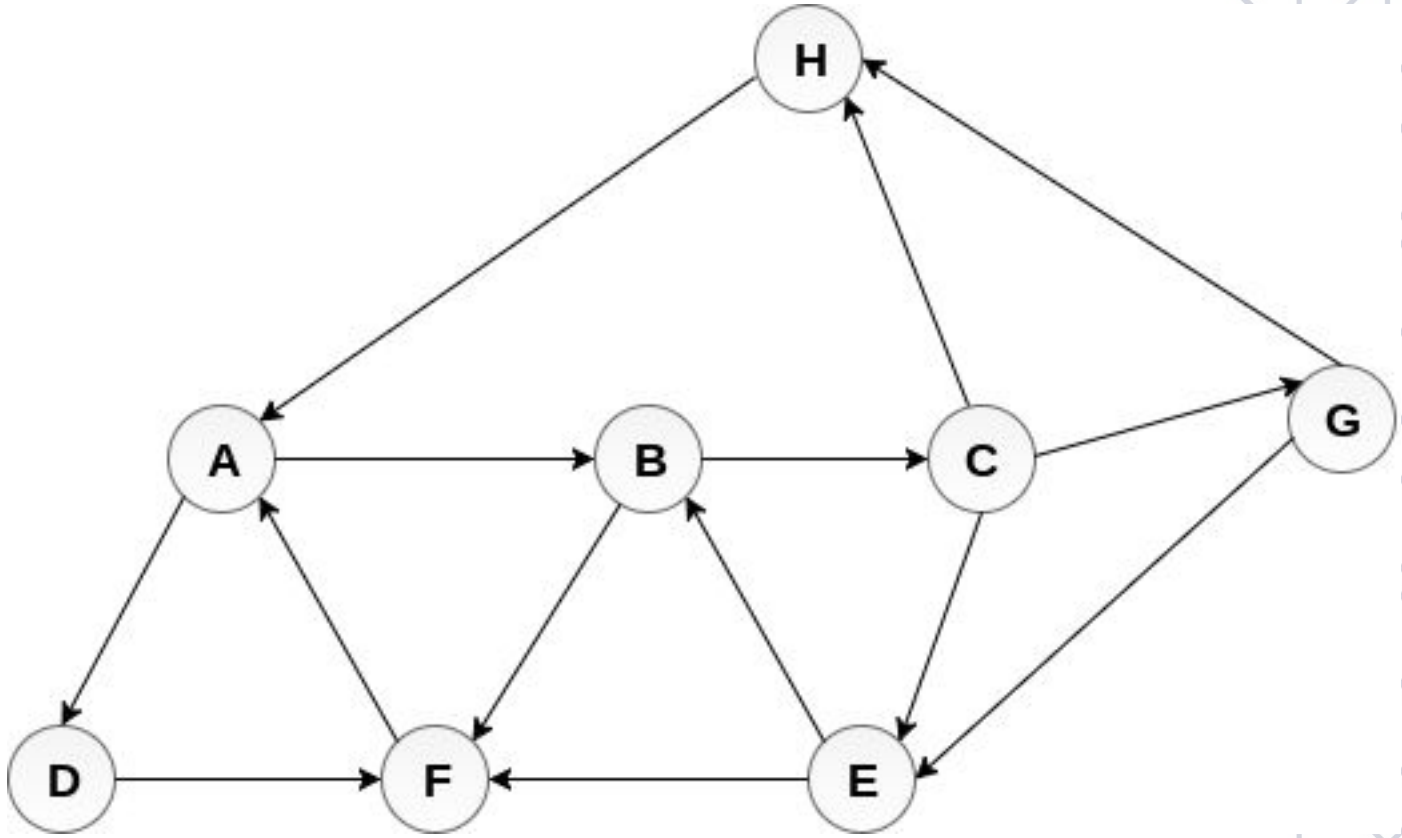
1) show the search tree
2) what is the path and its cost?



Depth-First Search (DFS)

Initial state or start node is A and the Goal is E
use leftmost node first

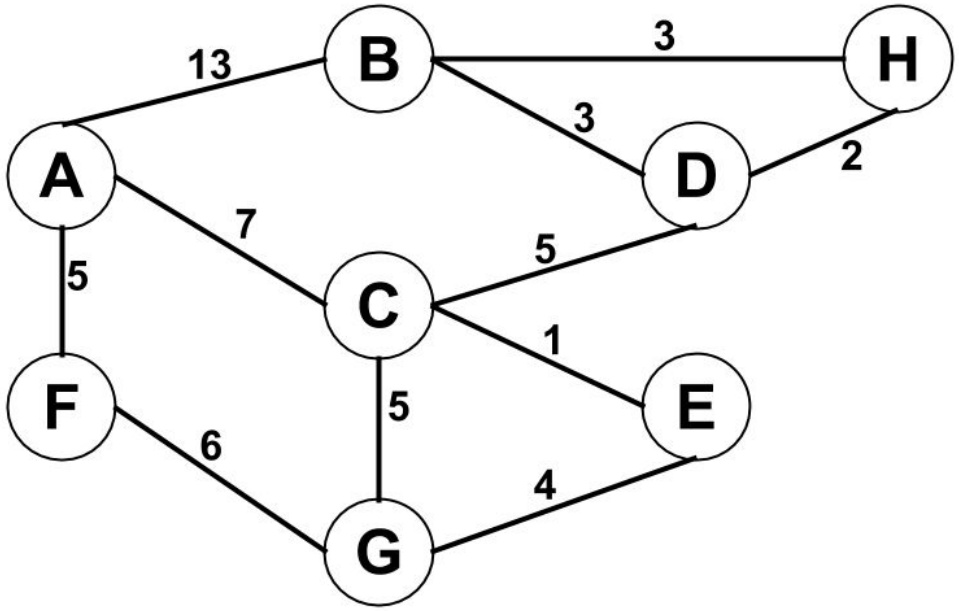
1) show the search tree
2) what is the path and its cost?



Uniform-Cost Search

a) Find the path from A to H
b) Fill below table
c) What is the path and its cost?

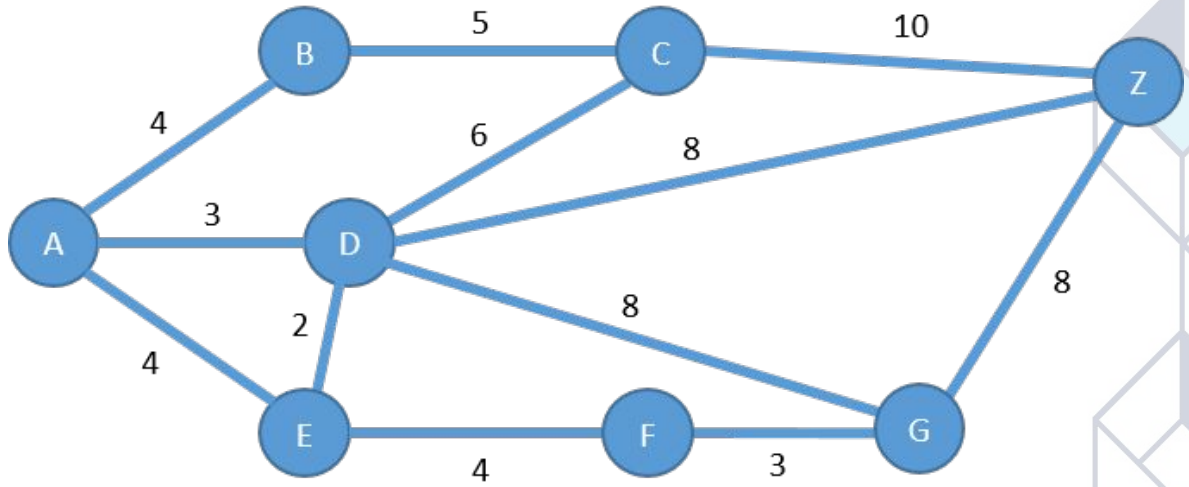
N: node	g(n): cost	p(n): parent



Uniform-Cost Search

a) Find the path from A to G
b) Fill below table
c) What is the path and its cost?

N: node	g(n): cost	p(n): parent

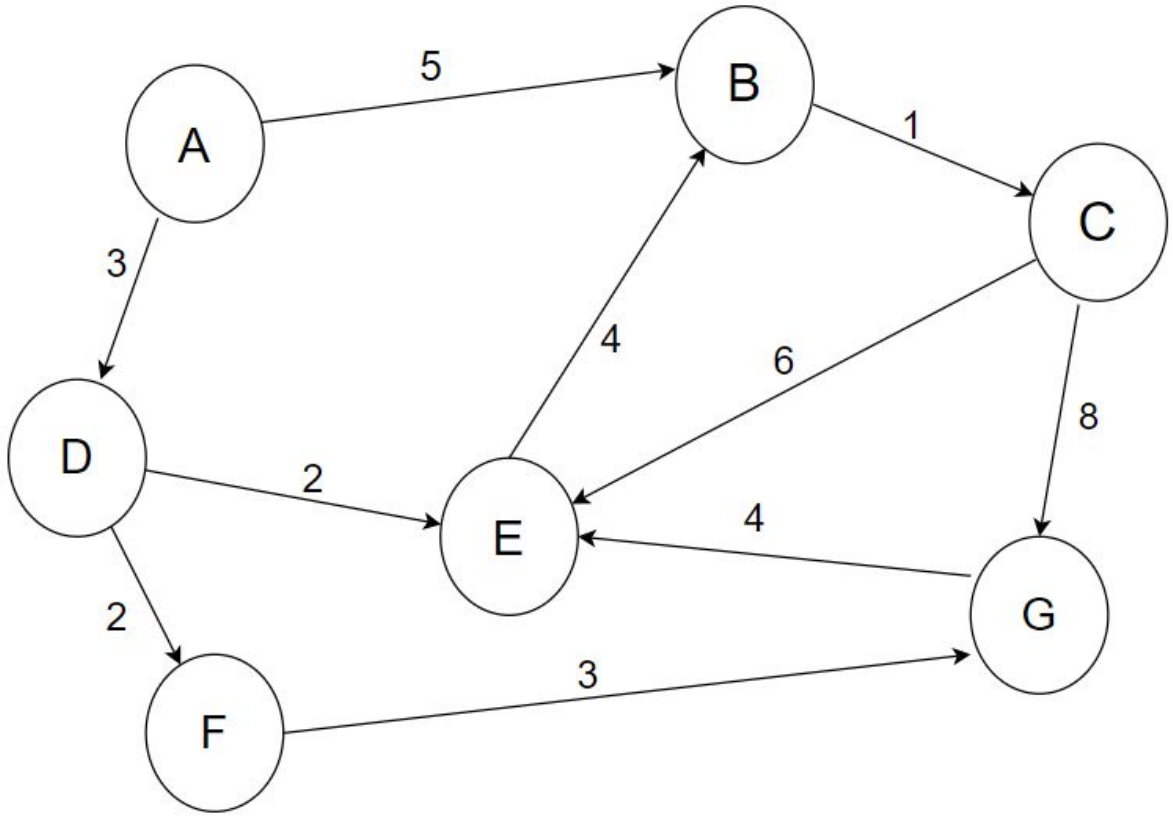


Example

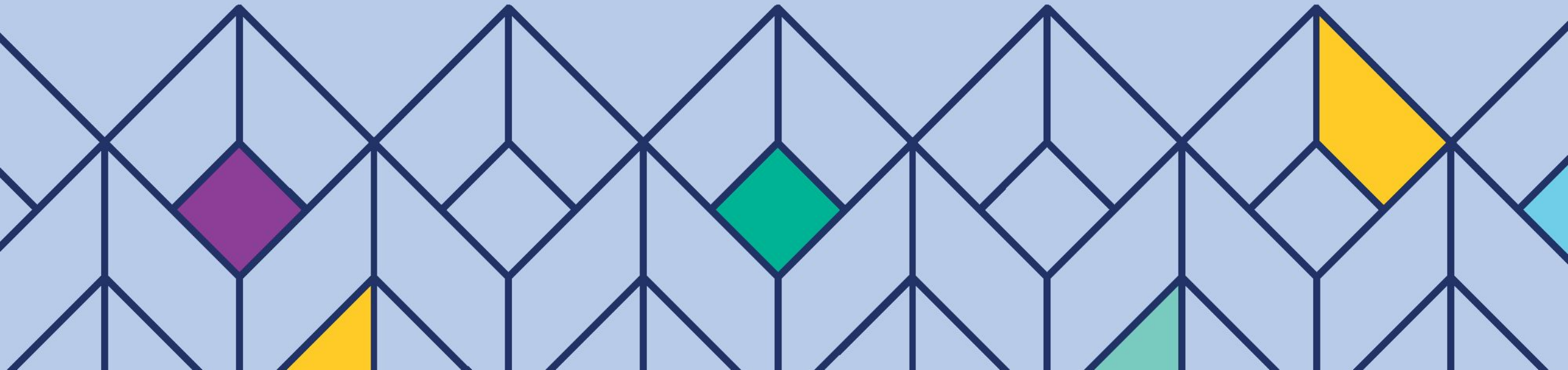
what are the path and the cost from A to G using

1. BFS (leftmost, alphabetical order)?
2. DFS (leftmost, alphabetical order)?
3. UCS?

Show the search tree.



Any Question



Princess Sumaya University for Technology
جامعة الأميرة سميّة للتكنولوجيا

Excellence in Education, since 1991
التميّز في التعليم منذ عام 1991

www.psut.edu.jo

Call: (+962) 6-5359 949
Fax: (+962) 6-5347 295
Email: info@psut.edu.jo

Princess Sumaya University for Technology
Amman 11941 Jordan
P.o.Box 1438 Al-Jubaiha