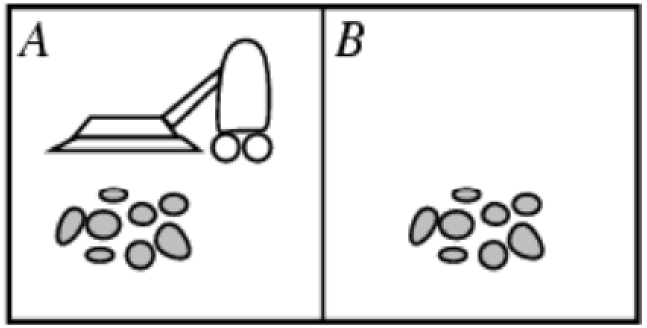
Artificial Intelligence (BCSE306L)

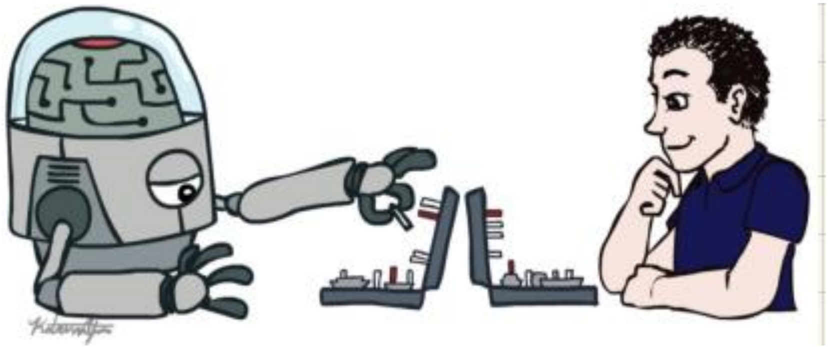
Module 02 – Problem Solving based on Search

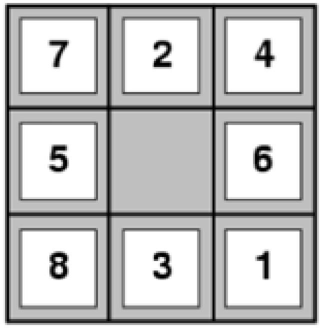
- Uninformed Search

o BFS o DFS o Depth Limited Search (DLS)

o Iterative Deepening DLS o Uniform Cost Search (UCS)







Start State

Dr. Durgesh Kumar

Assistant Professor (Senior),
SCOPE VIT Vellore

Lecture Outline

- Search Algorithm and Search Tree
  - Search tree, node, parent, successor, predecessor, fringe
  - Search data structures
  - Different types of Queue – Priority queue, FIFO, LIFO
  - Measuring Problem solving performance

- Uninformed Search
  - Breadth First Search (BFS)
  - Depth First Search (DFS)
  - Uniform cost Search (Dijkstra)

Previous Lecture Recap

- Introduction to Problems solving based on search
  - Problem solving agent vs Planning Agent
  - Informed search vs Uniformed Search
  - Problem solving agent – Cleaning Robot
- 4 phase of Problem-solving process
- Search problem formulation and its solution
  - 2 cell vacuum cleaner
  - Sokoban puzzle (8 tile puzzle)

Learning goals and Outcomes

- Course Outcomes (CO)
  - CO2 : Apply **basic principles of AI** in solutions that require problem-solving, inference, perception, knowledge representation, and learning.
  - Problem solving agent vs Planning Agent
  - Informed search vs Uniformed Search
  - Problem solving agent – Cleaning Robot
- 4 phase of Problem-solving process
- Search problem formulation and its solution
  - 2 cell vacuum cleaner
  - Sokoban puzzle (8 tile puzzle)

Search Algorithms

Un-informed Search

- **Agent can not estimate how far it is from Goal states.**
  - Breadth First Search (BFS)
  - Depth First Search (DFS)
  - Uniform cost search (Dijkstra's algo)
  - Depth-limited and iterative deepening DFS
  - $F(n) = g(n)$
    - *$h(n)$ can't be estimated*

Informed Search

- **Agent can estimate how far it is from Goal states.**
  - Greedy Best First Search
  - A* Search
  - $F(n) = g(n) + h(n)$

Uninformed Search

- $f(n) = g(n) + h\ (n)\ ;\ h(n) = 0$ *(Unknown)*
- Also known as *"**Blind Search**."*
- Search strategies that explore a problem space without using domain-specific knowledge to guide them.
- They rely only on the *problem's structure:*

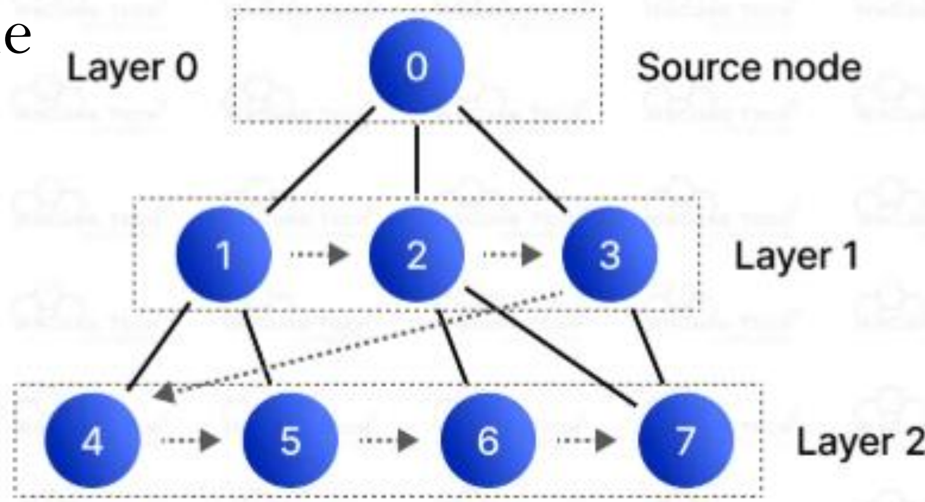
  i) ***Initial State*** *ii)* ***Operators*** *(Moves)* *iii)* ***Goal Test***

- *Key Algorithms We'll Cover: BFS, DFS, DLS, IDS, and UCS.*

Breadth First Search (BFS)

- **How it Works**:

Explores layer by layer, visiting all neighbors at the current depth before moving to the next level.



- **Key Feature**: Finds the shallowest solution. Guaranteed to be the shortest path if all step costs are equal.

- **Use Cases**:
  - Social Networks: Finding "**friends** of **friends**."“
  - GPS: Finding a path with the **fewest** number of stops.
  - Web Crawlers: Indexing web pages level by level from a source.
  - Peer-to-Peer Networks: Locating the nearest peers in a network like BitTorrent.

Depth First Search (DFS)

- **How it Works**:

Explores as deeply as possible along one branch before backtracking.

- **Key Feature**: Very **memory-efficient** compared to BFS. Only stores the current path.

- **Use Cases**:
  - Pathfinding & Mazes: Finding a path from a start to an end point.
  - Topological Sorting: Scheduling tasks with dependencies (e.g., compiling code).
  - Cycle Detection: Identifying loops in a graph (critical for dependency management).

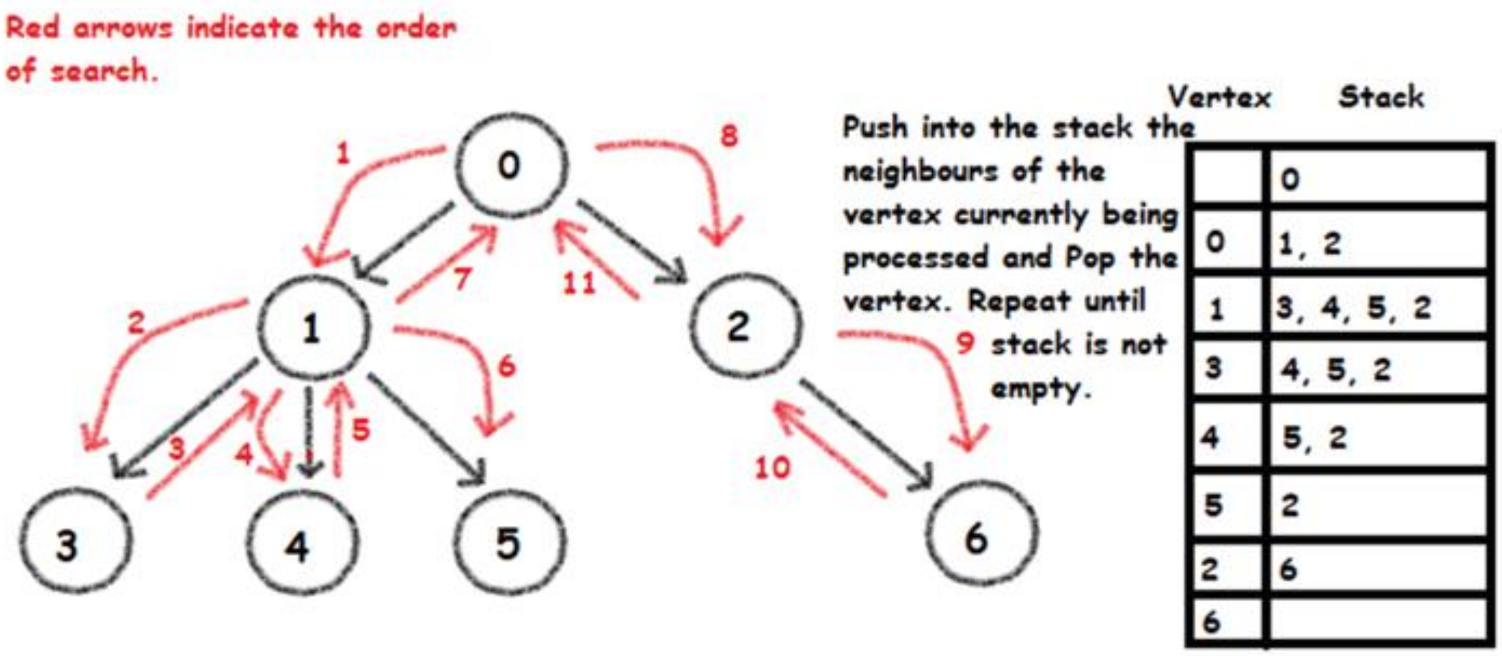
Depth First Search (DFS)

- **How it Works**:

Explores as deeply as possible alc
before backtracking.

- **Key Feature**: Very **memory-e**
  to BFS. Only stores the curren

- **Use Cases**:
  - Pathfinding & Mazes: Fin
    start to an end point.
  - Topological Sorting: Sche
    dependencies (e.g., compil
  - Cycle Detection: Identifying loops in a graph
    (critical for dependency management).



Depth First Search

Depth Limited Search (DLS)

- **How it Works**:

A standard DFS that stops searching once it reaches a pre-defined depth limit.

- **Key Feature**: Solves the infinite-path problem of DFS in graphs with cycles.

- **Use Cases**:
  - Game AI: Exploring a game tree up to a certain number of moves ahead (e.g., a chess engine looking 5 moves deep). ♟
  - Puzzle Solving: Constraining the search space in complex puzzles where a solution is likely within a known depth.

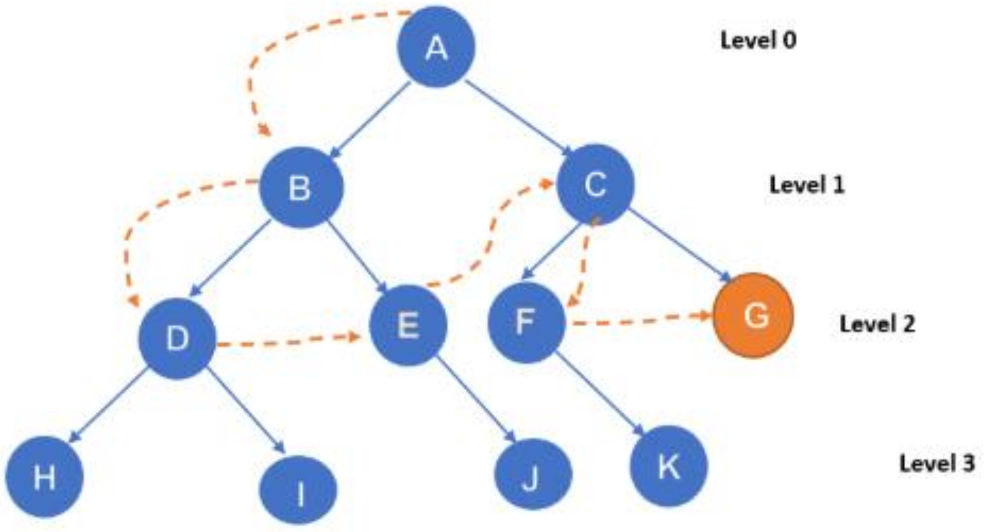


Fig: DLS (with depth limit =2)

Iterative Deepening Search (IDS)

- **How it Works**:

It performs repeated Depth-Limited Searches, increasing the depth limit with each iteration (1, 2, 3...). Combines BFS and DFS.

- **Key Feature**: The "best of both worlds"—finds the optima solution like BFS while using the low memory of DFS.

- **Use Cases**:
  - Large Search Spaces: The ideal choice when the searc space is **huge** and the solution depth is unknown.
  - Game AI: Finds the optimal (shortest) sequence of moves without the massive memory cost of BFS.

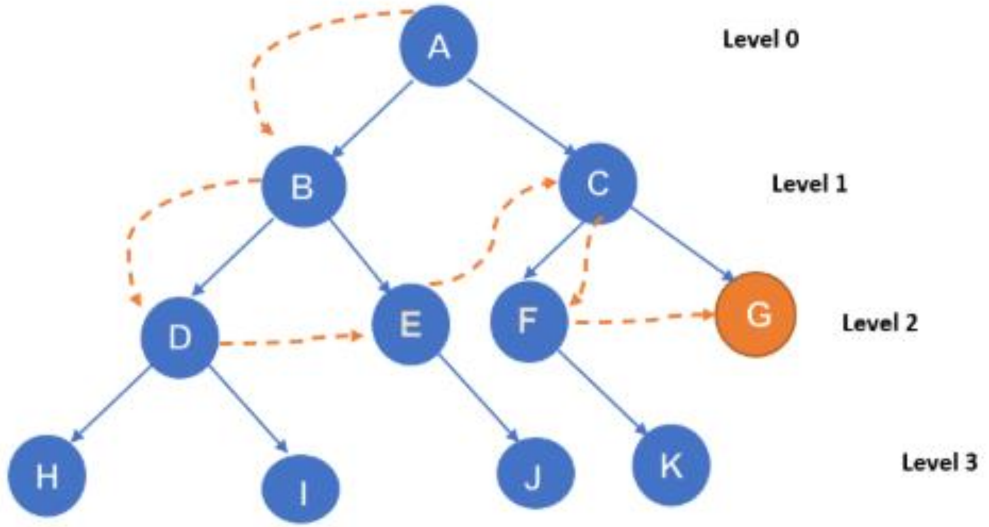
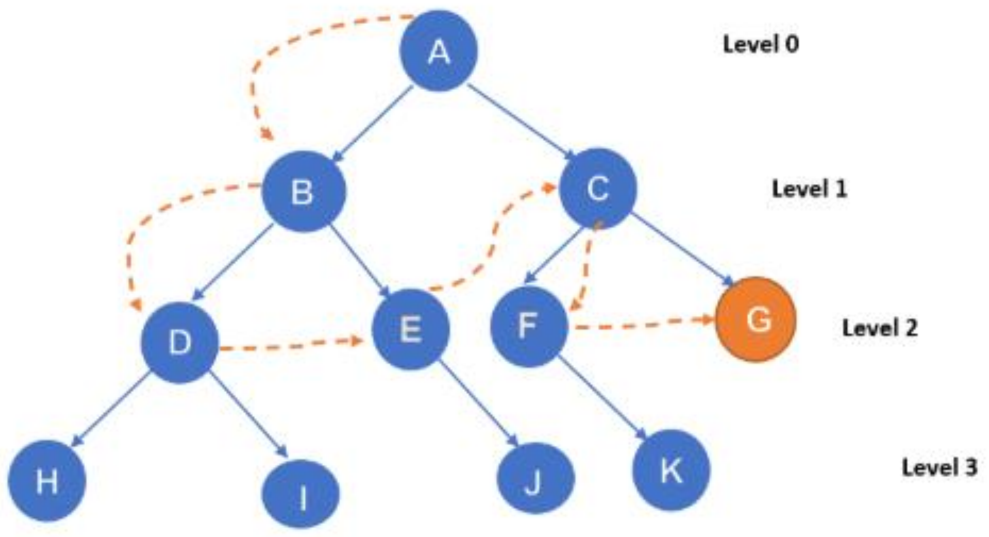


Fig: IDS

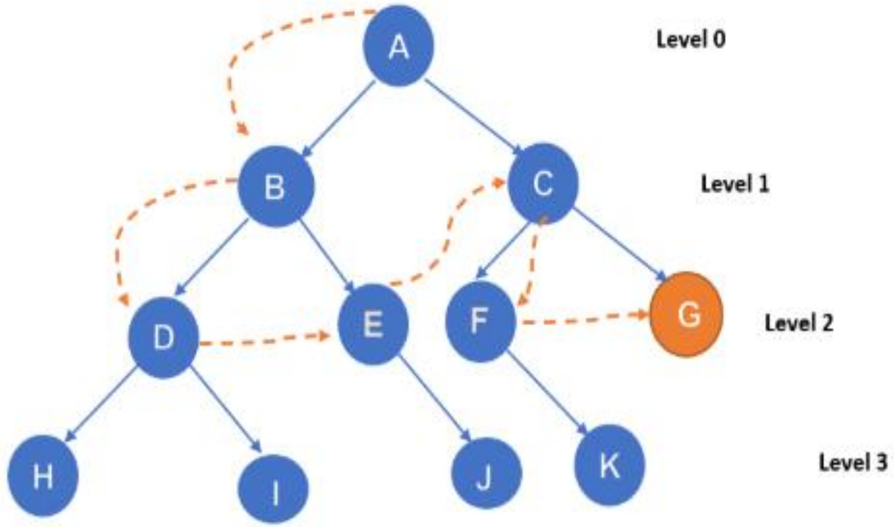
Search Process

- **A search** algorithm takes a search problem as input and returns a solution, or an indication of failure.
- **Each node** in a search tree over state-space search graph forms various paths from initial state.
- **Each node** in a search tree correspond to a state and the edges correspond to an action.
- frontier node
- interior region vs exterior region



Search Data structure

- **node.STATE** : the state which the node corresponds
- **node.PARENT** : the node that generated this node du
search process.
- **node.ACTION** : the action that was applied to paren
state to generate this node.
- **node.PATH_COST** : the total cost of the path from
initial state to this node, also denoted by *g(node)*



Search Data structure

- **Priority queue:** pops the node with minimum coset according to some evaluation function f. Used in best-first search

- **FIFO queue**: QUEUE pops the node which was inserted first, used in breadth-first search.

- **LIFO queue** : STACK pops the most recently added node, used in depth-first search.

Measuring Search algorithm performance

- **C.O.S.T**
- **Completeness:** Is the algorithm guaranteed to find a solution when there is one, and to correctly report failure when there is not ?
  - Clear communication about **solution** or **failure**
- **Optimality**: Does it find the a solution with lowest cost of all solutions ?
- **Space Complexity :** How much memory is needed to perform search?
- **Time Complexity** : How long it takes to find a solution?

Time and space complexity are measured in terms of

- *b:* maximum of the search tree
- *d:* depth of the least-cost branching factor solution.
- *m*: maximum depth of the state space (may be ∞)

BFS algo

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

BFS Example 1

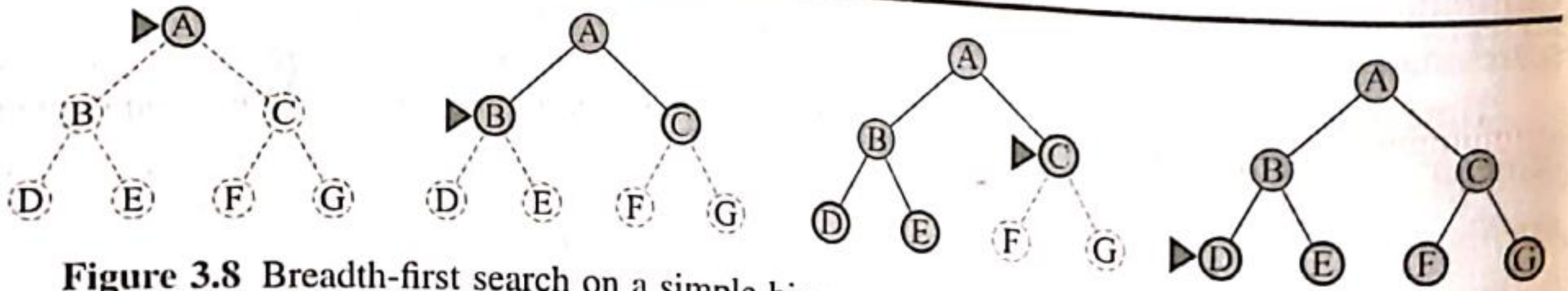


Figure 3.8 Breadth-first search on a simple binary tree. At each stage, the node to be expanded next is indicated by the triangular marker.

BFS Properties

- Complete ?
- Time ?
- Space ?
- Optimal ?

.- where b is the branching factor for each node and

- d is the maximum depth at which goal state is found

BFS Properties

- Complete? Yes (if b is finite)
- Time? O(b^d)

  [1+ b + b^2 + …+ b^d] = O(b^d)

- Space? O(b^d) (keeps every node in memory)

- Optimal? Yes (if cost = 1 per step)

  .- where b is the branching factor for each node and

- d is the maximum depth at which goal state is found

- BFS **always finds a solution** with minimum number of actions

BFS Properties

- Complete? Yes (if b is finite)
- Time? O(b^d)

  [1+ b + b^2 + …+ b^d] = O(b^d)

- Space? O(b^d) (keeps every node in memory)

- Optimal? Yes (if cost = 1 per step)

  .- where b is the branching factor for each node and

- d is the maximum depth at which goal state is found

- BFS **always finds a solution** with minimum number of actions

BFS Drawback

- When b=10, and d=10, BFS would take 3 hrs and 10 TB of memory.
- With d=14, search would take 3.5 years even with infinite memory.
- Exponential complexity search problems can not be solved by uninformed search for any but smallest instance.

BFS algo

- function BREADTH-FIRST-SEARCH(problem) returns a solution node or failure

  node ← NODE(problem.INITIAL)

  if problem.IS-GOAL(node.STATE) then return node

  frontier ← a FIFO queue, with node as an element

  reached ← {problem.INITIAL}

  while not IS-EMPTY(frontier) do

node ← POP(frontier)

for each child in EXPAND(problem, node) do

s ← child.STATE

if problem.IS-GOAL(s) then return child

- if s is not in reached then
- add s to reached
- add child to frontier
- return failure

Practice Problem -1

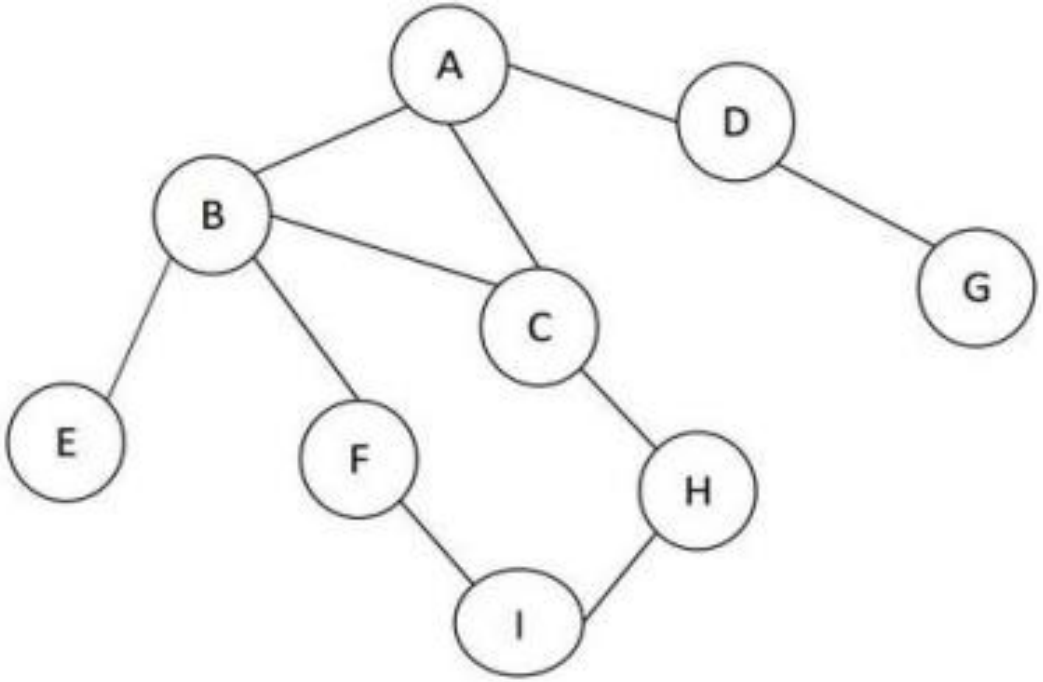
Q1. Apply the following search algorithms to find a path from the node A to G in the given graph:

i) Depth First Search (DFS)

ii) Depth-limited search (L=2)

iii) Iterative Deepening Search (IDS)

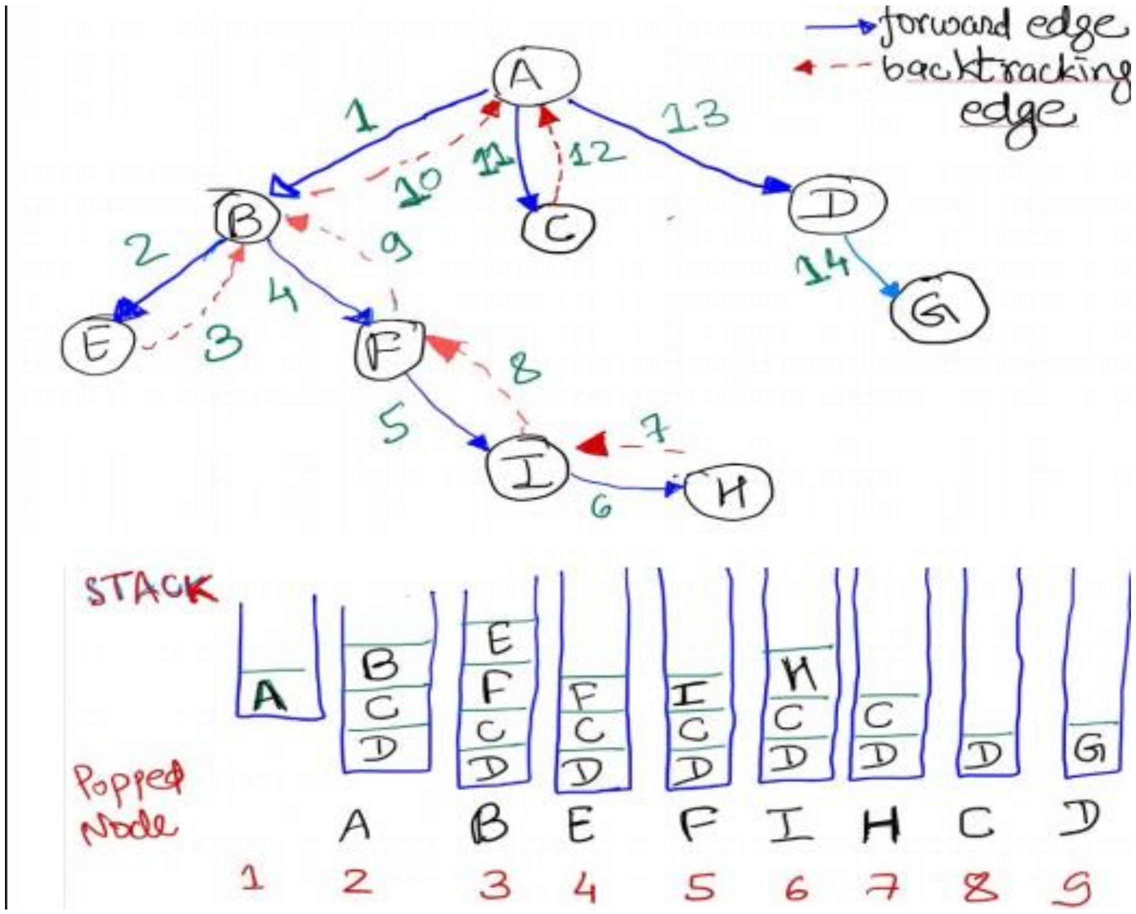
Note: Process the nodes alphabetically for the nodes at the same level.

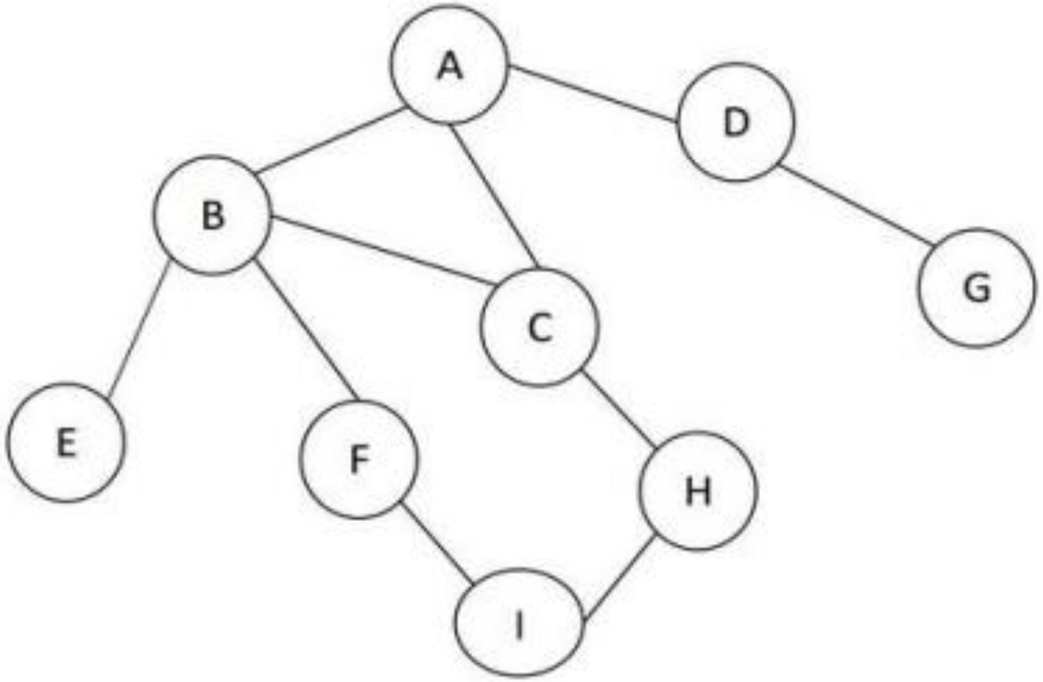


b) Analyze the above algorithms regarding completeness, optimality, time complexity, and space complexity.

Solution - Practice Problem -1

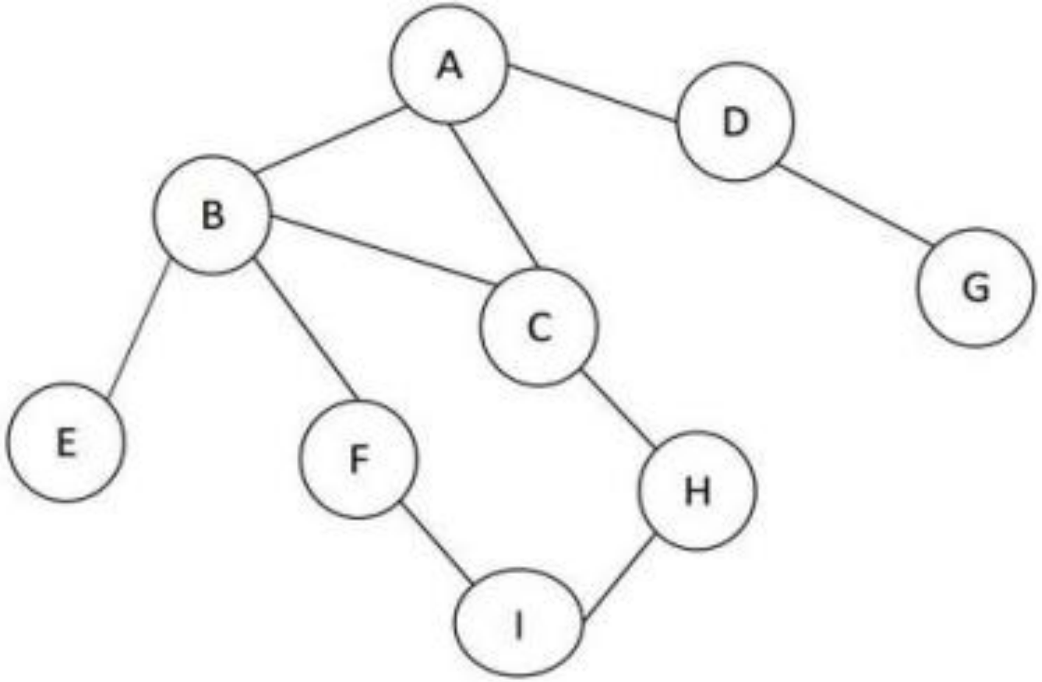
Solution Q1. i) DFS

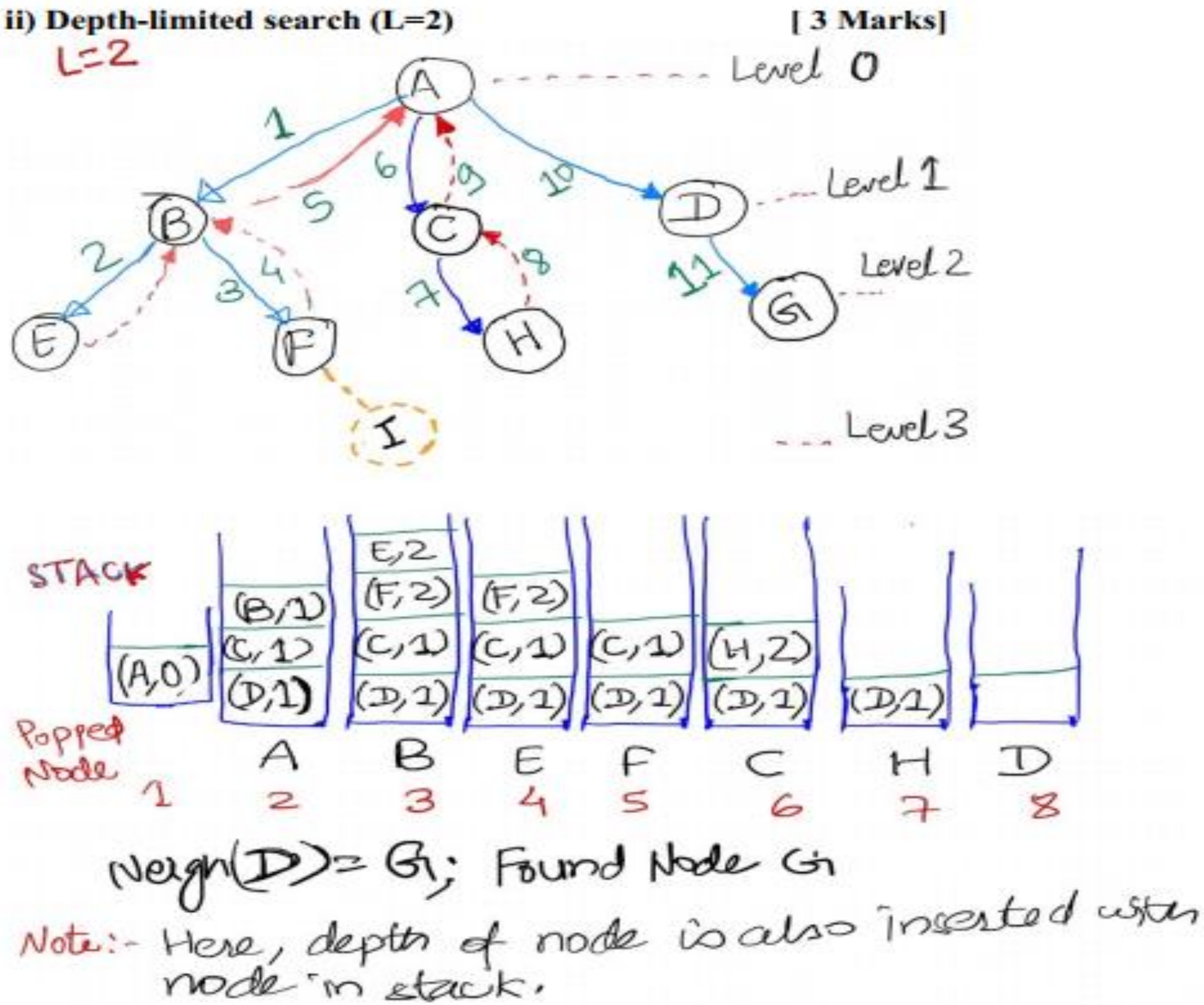




Solution - Practice Problem -1

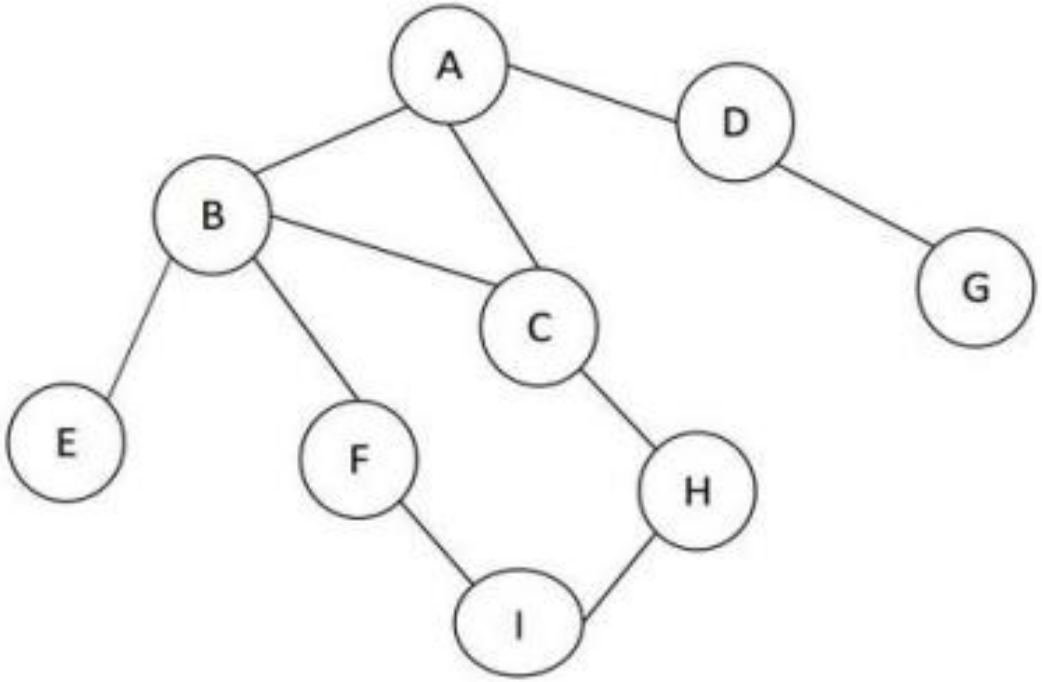
Solution Q1. ii) DLS

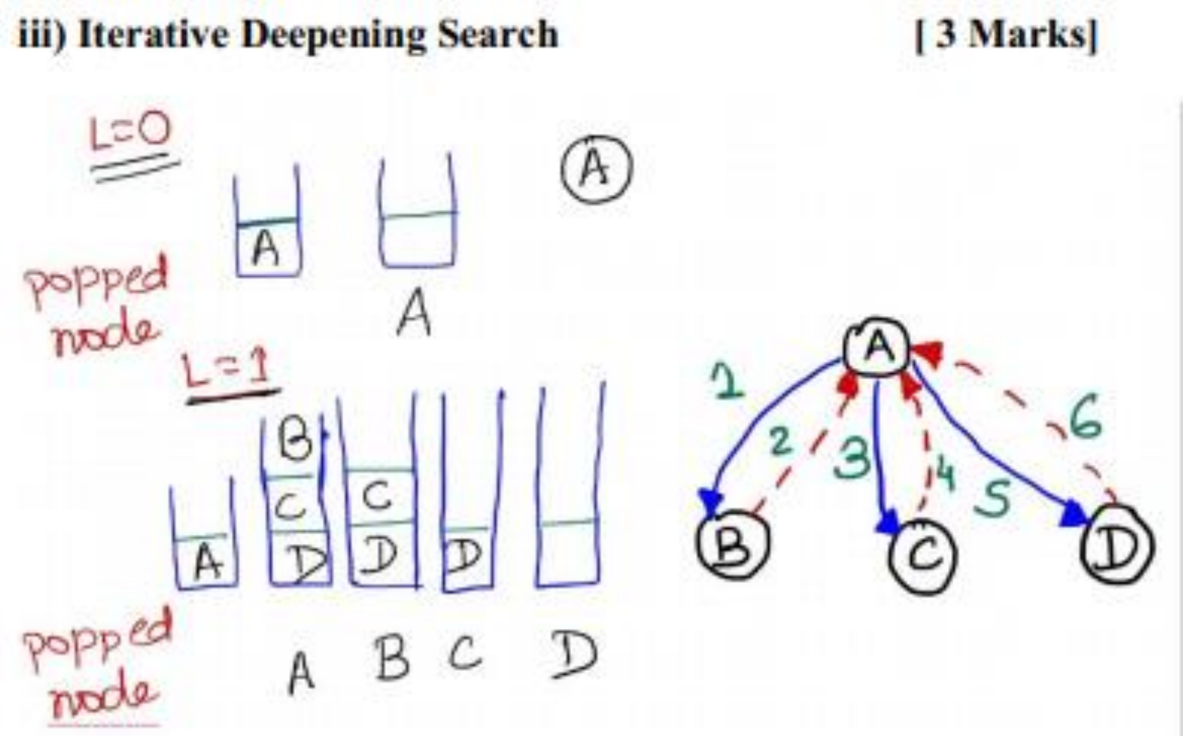




Solution - Practice Problem -1

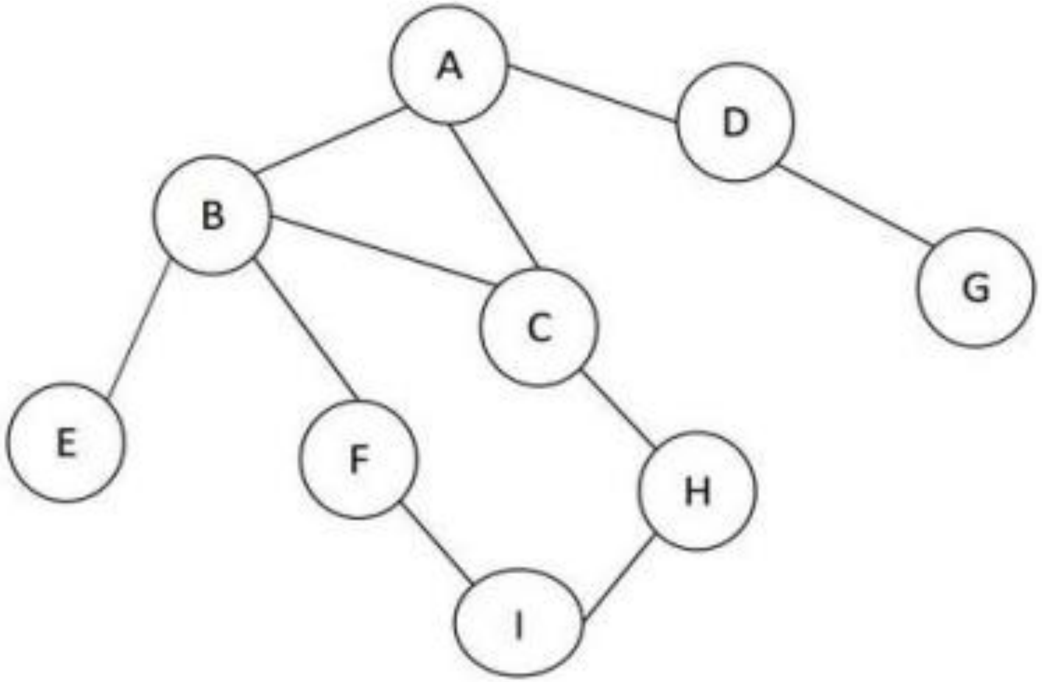
Solution Q1. iii) IDS

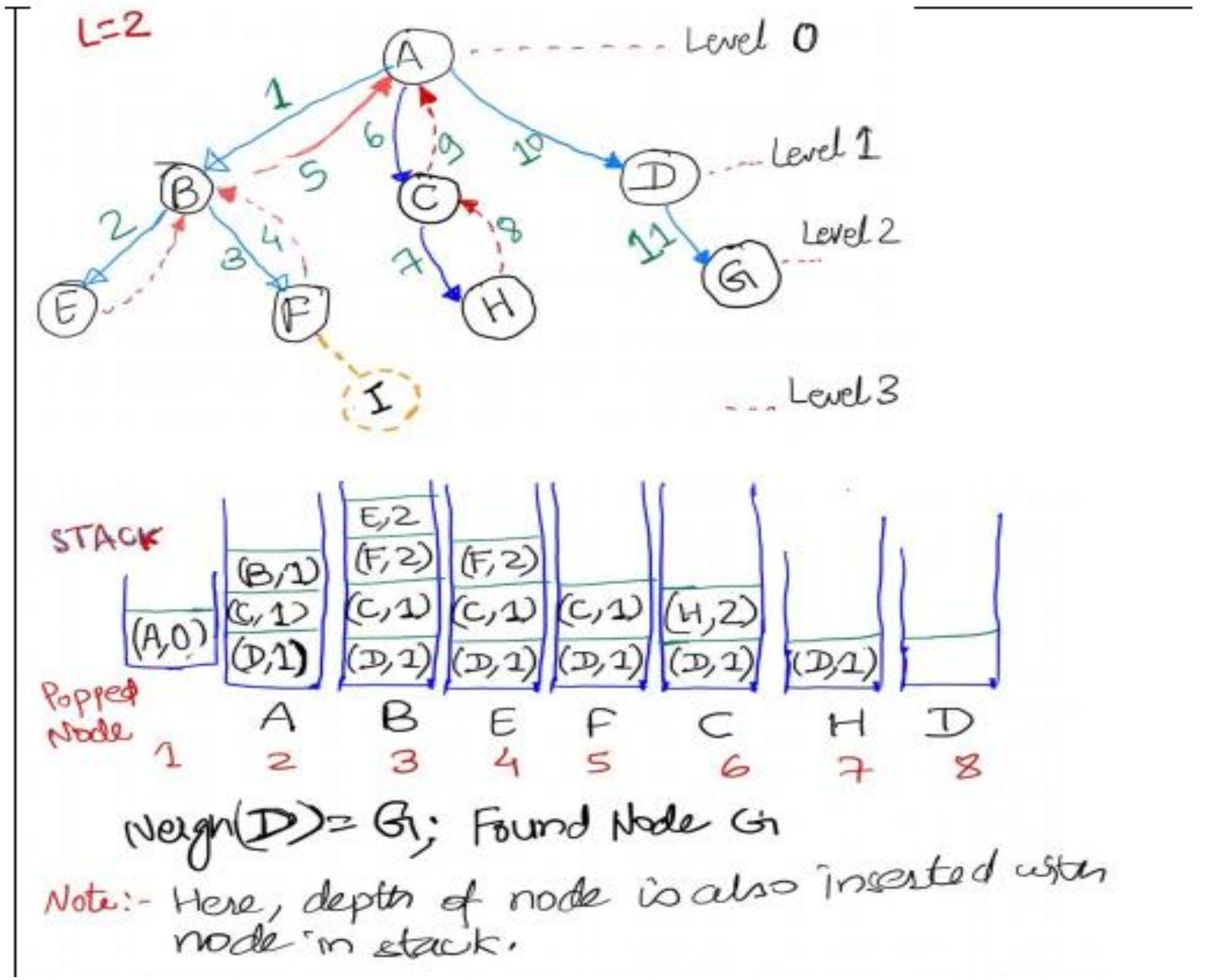




Solution - Practice Problem -1

Solution Q1. iii) IDS





Solution - Practice Problem -1

Solutn Q1.

b) Anlaysis of Algorithm

Algorithm Name	Completenes	Optimality	Time Complexity	Space Complexity
Depth First Search (DFS)	No	No	$O(bm)$	$O(bm)$
Depth Limited DFS	No	No	$O(b^l)$	$O(bl)$
Iterative Deepening DFS	Yes	Yes	$O(bd)$ when there is solution $O(bm)$ when there is no solution	$O(bd)$ when there is solution $O(bm)$ when there is no solution

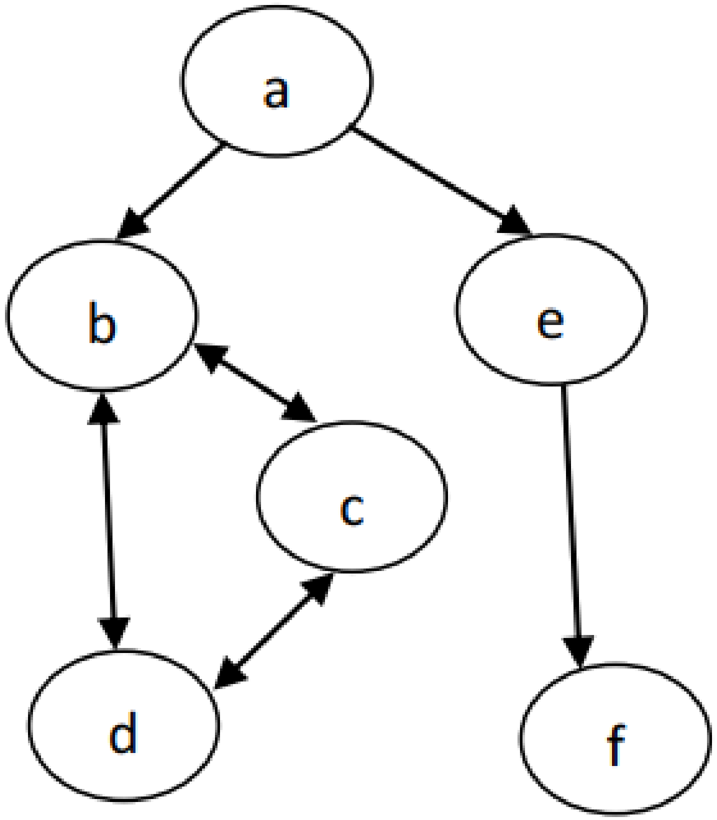
where
b is branching factor
m is the maximum depth of tree
d is the depth where the goal node v is found.

Practice Problem -2

Q2. Which sequences of paths are explored by BFS and DFS in this problem? Show the complete intermediate state space for DFS and BFS with a neat sketch.

- Would you prefer DFS or BFS for this problem? Justify?

Note: Nodes are revisited as per the direction mentioned.



Practice Problem -3

Q3. For the given directed graph (start node is S), perform the Uniform Cost Search (UCS). Expand the node based on the alphabetic order if the nodes have the same cumulative cost.

A. Show the priority queue and visited nodes for each step (4M)

B. Draw the tree for each step (3M)

C. Path and cost to reach the goal states (G1, G2, G3) (2M)

D. Find the goal node which has the minimum cost. (1M)

Note: Graph is given in the next slide due to space constraint.

Practice Problem -3

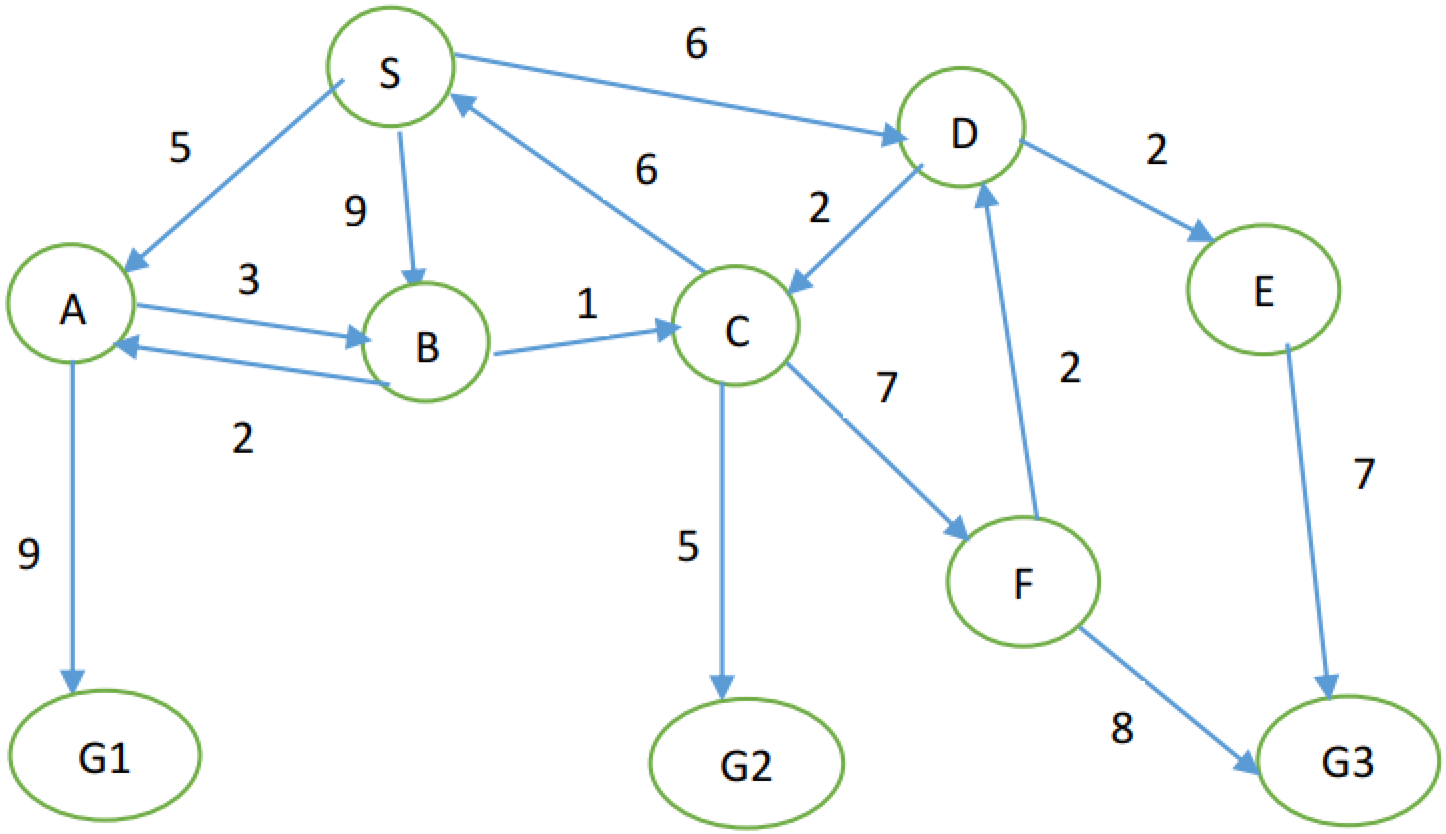
Q3. For the given directed graph (start node is S), perform the **Uniform Cost Search (**UCS). Expand the node based on the alphabetic order if the nodes have the same cumulative cost.

A. Show the priority queue and visited nodes for each step (4M)

B. Draw the tree for each step (3M)

C. Path and cost to reach the goal states (G1, G2, G3) (2M)

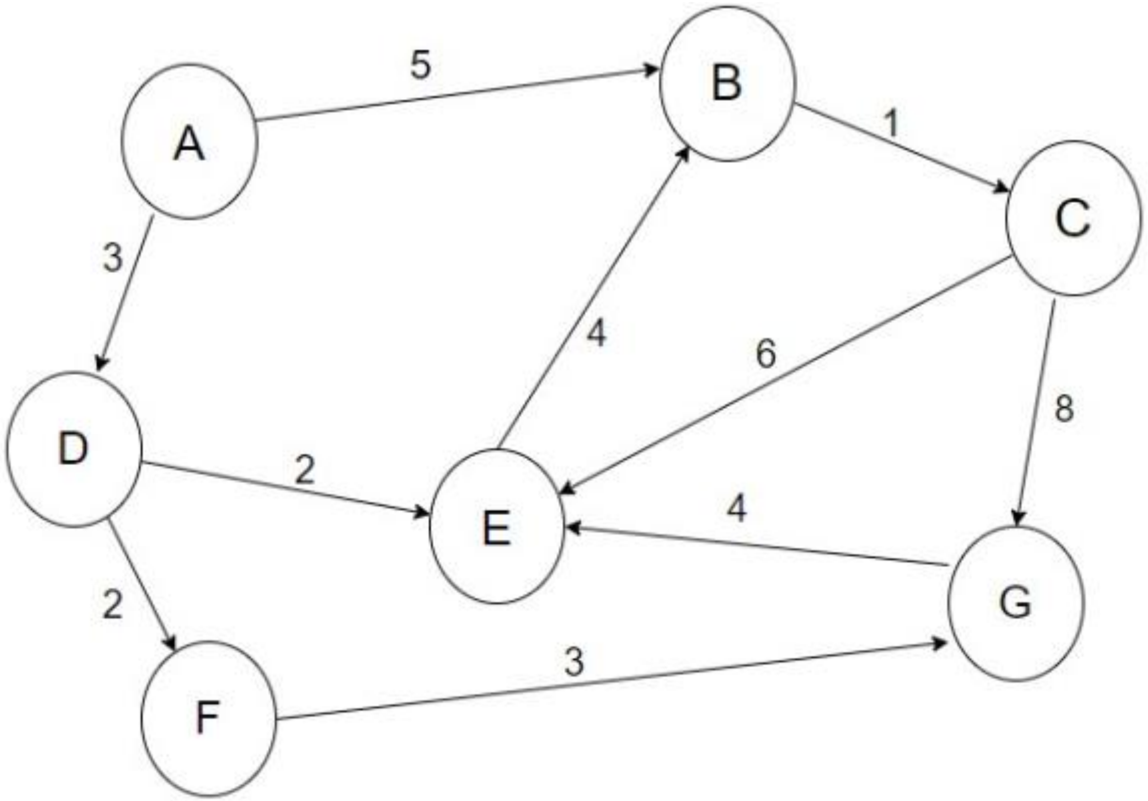
D. Find the goal node which has the minimum cost. (1M)



Practice Problem -4

Q4. Consider the following graph. The starting node is A and the goal node is G. Find the actual and traversed paths from A to G using

uniform cost search along with the algorithm's performance measures.



Practice Problem -5

Q5. A.

i. Calculate the number of nodes generated in a depth limited search to depth l with branching factor n.

ii. Calculate the number of nodes generated in an iterative deepening search to depth l with branching factor b.

iii. For n = 5, l = 10 suggest which one of the above two

algorithms performs better. Justify the answer.

Practice Problem -5

Q5. B.

i. Calculate the time required to search a node in iterative deepening search to depth d with branching factor b.

ii. Calculate the time required to search a node in breadth first search to depth d with branching factor b.

iii. For b = 10, d = 5 suggest which one of the above two algorithms performs better. Justify the answer.

Practice Problem - 6

Q6. A. Consider the initial and the final state for an 8-puzzle problem given below.

i. Generate all the states for the given problem,

ii. Find out the path to final state using Breadth First Search (BFS).

iii. Find out the path to final state using Depth First Search (DFS)

Initial State

2	8	3
	6	4
7	1	5

Final State

1	2	3
8		4
7	6	5

Practice Problem - 6

Q6. B. Consider the initial and the final state for an 8-puzzle problem given below.

i. Generate all the states for the given problem,

ii. Find out the path to final state using Breadth First Search (BFS).

iii. Find out the path to final state using Depth First Search (DFS)

Initial State

7		5
1	6	4
2	8	3

Final State

7	6	5
8		4
1	2	3

Thank you

Questions ?