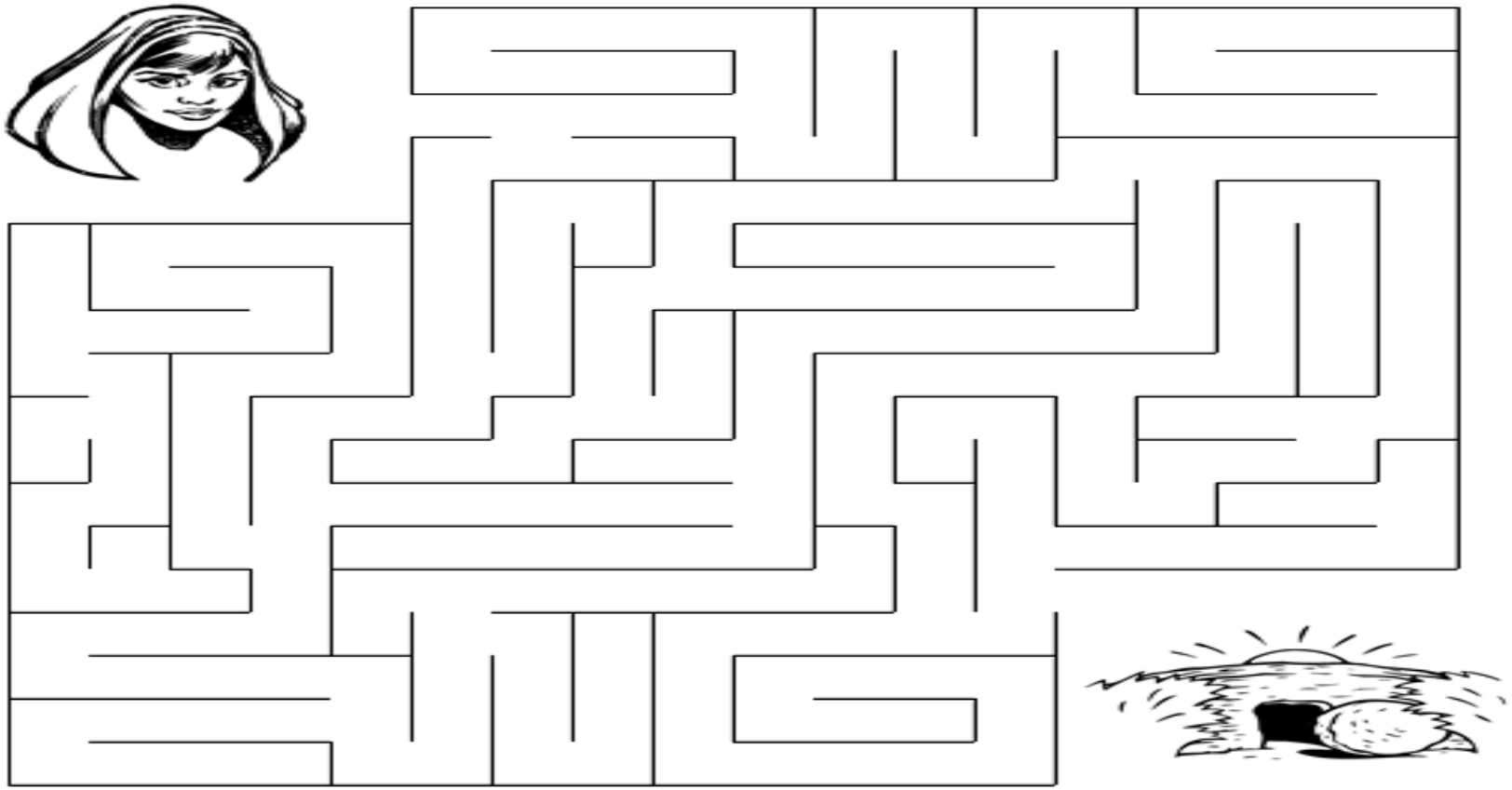
Chapter 2

Solving Problems by Searching

Problem Solving by Searching

- Problem is a goal and a means for achieving the goal.
- The process of exploring what the means can do is search.
- Searching is the process of considering various possible sequences of operators applied to the initial state, and finding out a sequence which culminates in a goal sate.
- The goal specifies the state of affairs we want to bring about and the means specifies the operations we can perform in an attempt to bring about the goal.
- Solution will be a sequence of operations (actions) leading from initial state to goal state (plan)

Exampl
e



The states in the space can correspond to any kind of configuration. For example, the possible settings of a device, positions in a game or (more abstract still) a set of assignments to variables. Paths in state space correspond to possible sequences of transitions between states

States

- A problem is defined by its elements and their relations.
- In each instant of the resolution of a problem, those elements have specific descriptors (How to select them?) and relations.
- A state is a representation of those elements in a given moment.
- Two special states are defined:
  - Initial state (starting point)
  - Final state (goal state)

State space

- The state space is the set of all states reachable from the initial state.
- It forms a graph (or map) in which the nodes are states and the arcs between nodes are actions.
- A path in the state space is a sequence of states connected by a sequence of actions.
- The solution of the problem is part of the map formed by the state space.

Problem Solution

- A solution in the state space is a path from the initial state to a goal state or, sometimes, just a goal state.
- Path/solution cost: Function that assigns a numeric cost to each path, the cost of applying the operators to the states
- Solution Quality is measured by the path cost function, and an optimal solution has the lowest path cost among all solutions.
- Solutions: any, an optimal one, all. Cost is important depending on the problem and the type of solution required.

Problem-solving agents

- In Artificial Intelligence, Searching techniques are universal problem-solving methods.
- **Rational Agents** or **Problem-solving agents** in AI mostly used these search strategies or algorithms to solve a specific problem and provide the best result.
- Problem-solving agents are the goal-based agents and use atomic representation. In this topic, we will learn various problem-solving search algorithms.

Cont..

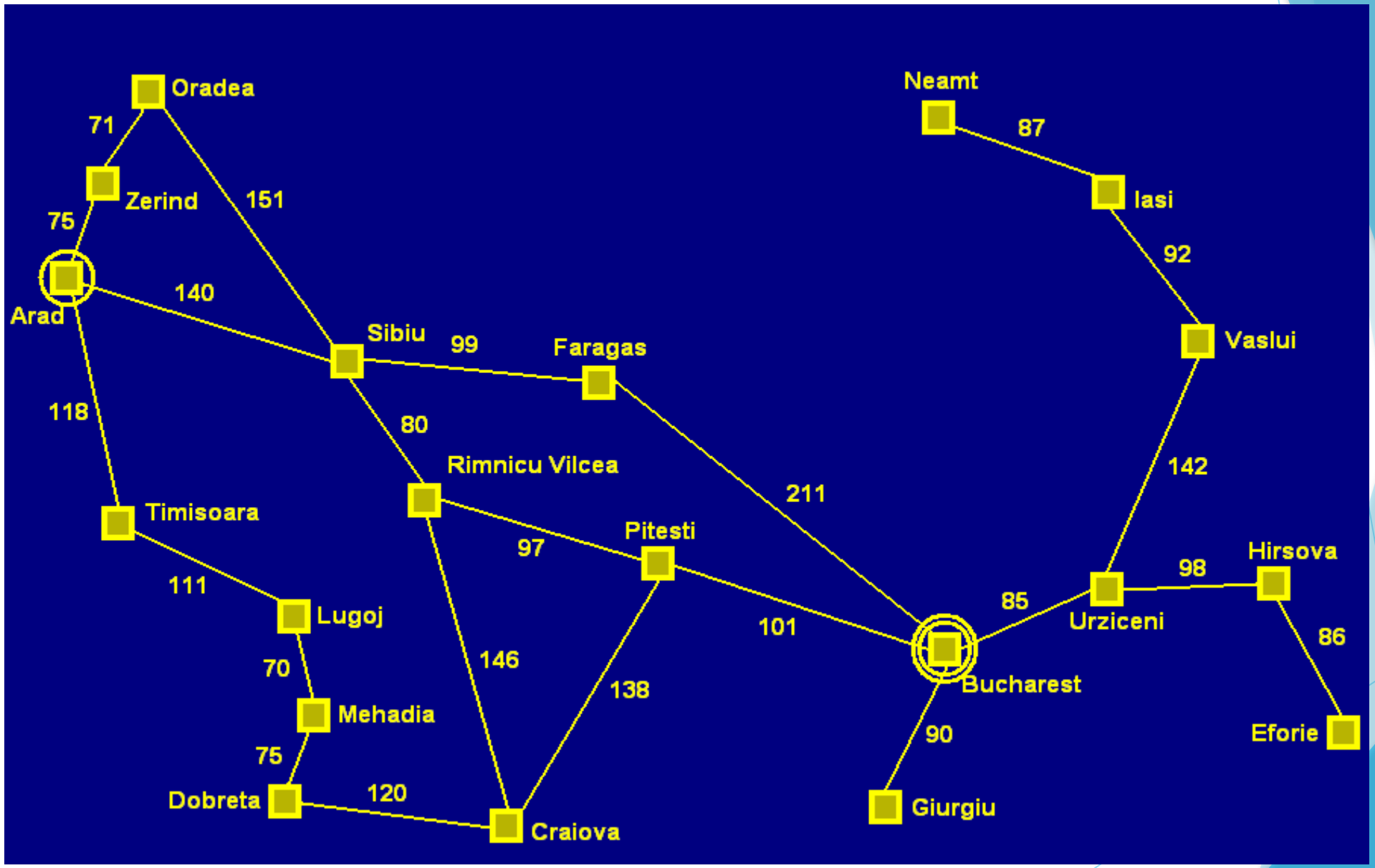
- Problem solving agents are goal-directed agents:
  1. Goal Formulation: Set of one or more (desirable) world states (e.g. checkmate in chess).
  2. Problem Formulation: What actions and states to consider given a goal and an initial state.
  3. Search for solution: Given the problem, search for a solution --- a sequence of actions to achieve the goal starting from the initial state.
  4. Execution of the solution

Problem formulation

- A well-defined problem can be described by:
  a) Initial state: The initial state that the agent starts in.
  b) Action: A description of the possible actions available to the agent.
  c) Transition model: A description of what each action does; the formal name for this is the transition model.
  d) Path cost: functions that assigns a cost to a path
  e) Goal test: test to determine the goal state

Example A

- The initial state is In(Arad)
- From the state In(Arad), the applicable **actions** are {Go(Sibiu), Go(Timisoara), Go(Zerind)}.
- Transition Model: RESULT(In(Arad), Go(Zerind)) = In(Zerind)
- **Goal test** :{In(Bucharest)}
- **Path cost:** length in kilometers


Oradea
71
Zerind
151
75
Arad
140
Sibiu
99
Faragas
118
80
Rimnicu Vilcea
211
Timisoara
Pitesti
97
111
Lugoj
101
146
138
70
Mehadia
75
Dobreta
120
Craiova
Neamt
87
Iasi
92
Vaslui
142
Hirsova
98
85
Urziceni
86
Bucharest
90
Eforie
Giurgiu

Example B : The 8-puzzle

S: start state

2	8	3
1	6	4
7		5

G: Goal state

1	2	3
8		4
7	6	5

State:

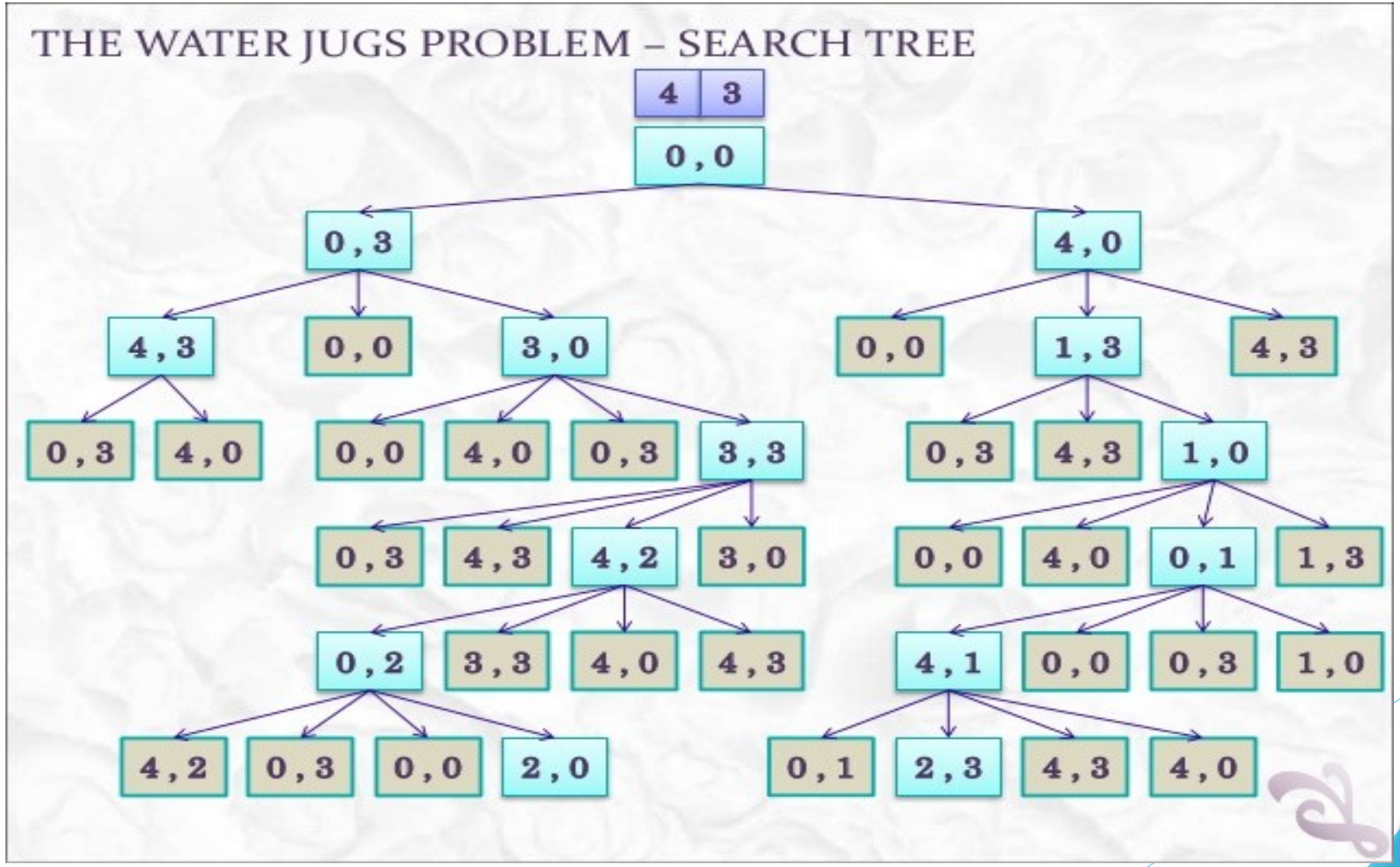
The boards, i.e., Location of blank, integer location of tile Initial sate: any sate can be initial state

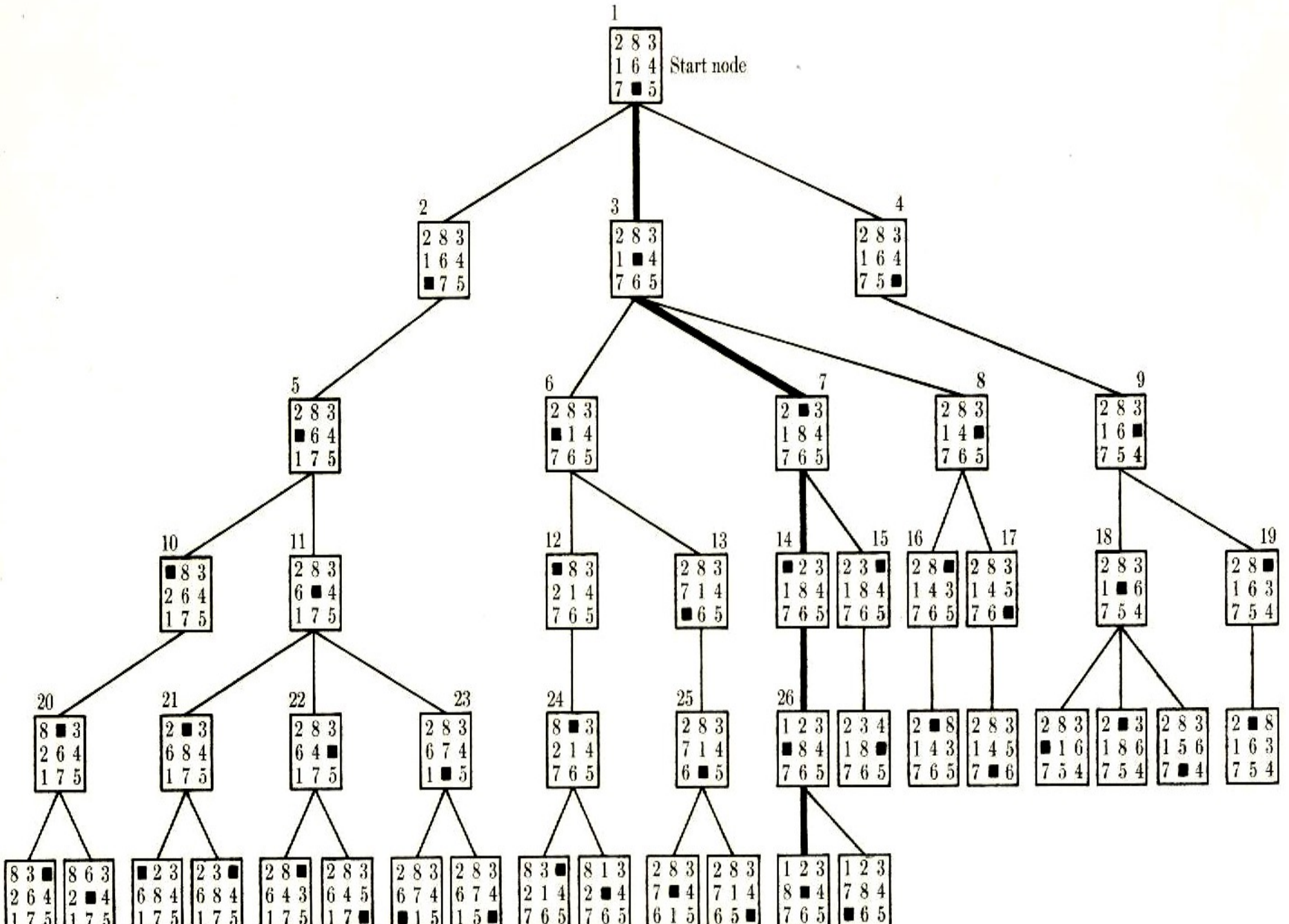
Operators/ Actions: Blank moves left, right, up, and down

Goal state: Match G

Path Cost: Each step costs 1 so cost length of path to reach goal

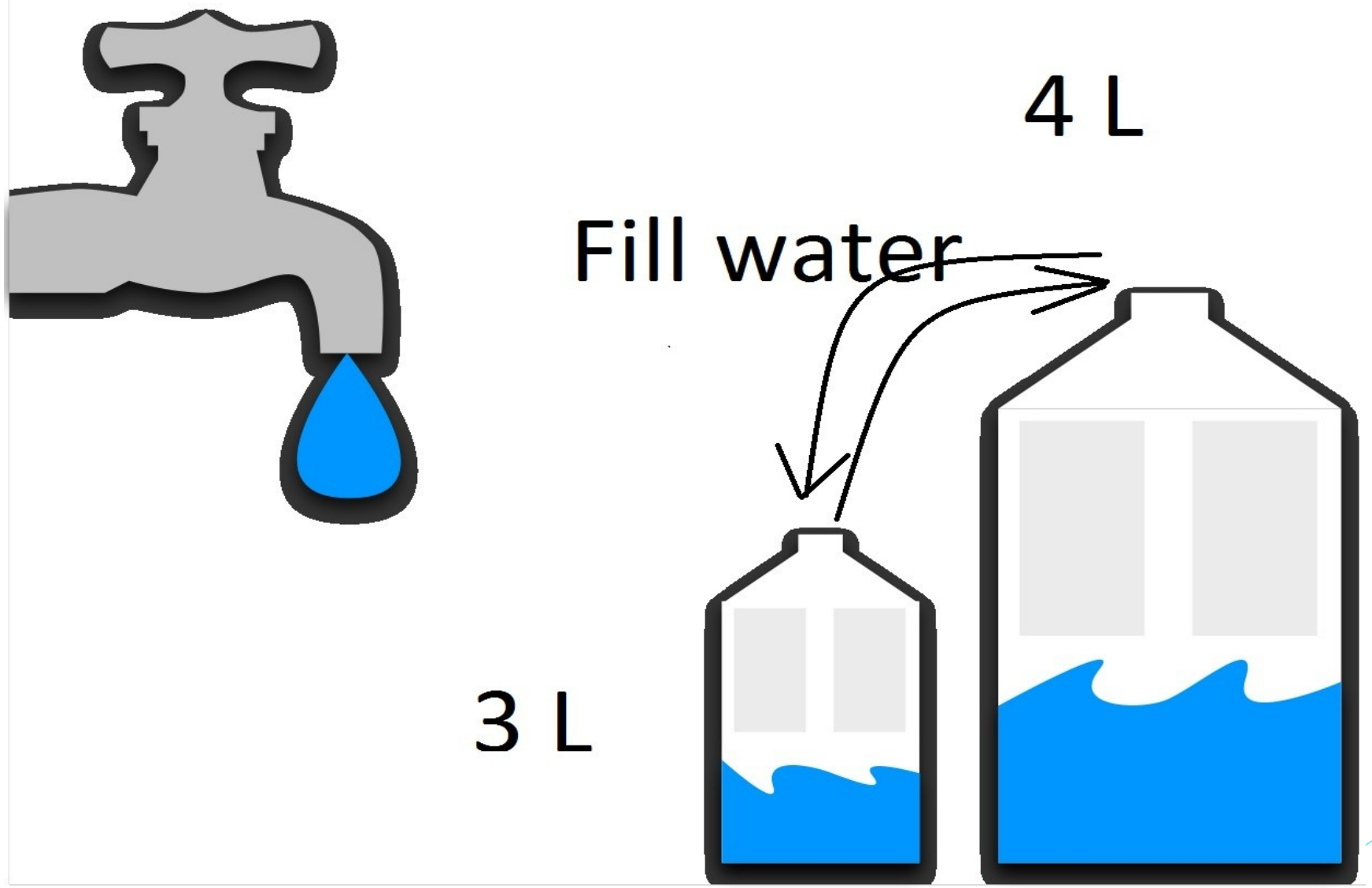
Search tree


THE WATER JUGS PROBLEM – SEARCH TREE
4 3
0 , 0
0 , 3
4 , 0
4 , 3
0 , 0
3 , 0
0 , 0
1 , 3
4 , 3
0 , 3
4 , 0
0 , 0
4 , 0
0 , 3
3 , 3
0 , 3
4 , 3
1 , 0
0 , 3
4 , 3
4 , 2
3 , 0
0 , 0
4 , 0
0 , 1
1 , 3
0 , 2
3 , 3
4 , 0
4 , 3
4 , 1
0 , 0
0 , 3
1 , 0
4 , 2
0 , 3
0 , 0
2 , 0
0 , 1
2 , 3
4 , 3
4 , 0


1
Start node
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26

Example C

- You are given two jugs, a 4-gallon and a 3-gallon one. Neither has any measuring marks on it. There is a tap that can be used to fill the jugs with water. How can you get exactly 2 gallons of water into the 4-gallon jugs.
- Specify the initial state, the goal state , all the possible operators to reach from the start state to the goal state.
- Solution:


4 L
Fill water
3 L

Steps to solve the problem

- There are many possible ways to formulate the problem as search.
- 1st step:
  - State description the integers (x,y) {x:0,1,2,3,4}, {y:0,1,2,3,4}
- 2nd step:
  - Describe the initial and goal state
    - The initial state is {0,0}
    - The goal state is {2,x} where x can take any value.
- 3rd step:
  - List all the action in the production rule(as rules or operators as follows)

Cont..

- Fill the 4-gallon jug {x,y}➔{4,y}, if x<4
- Fill the 3-gallon jug {x,y} ➔{x,3}, if y<3
- Empty 4-gallon jug {x,y}➔{0,y},if x>0
- Empty 3-gallon jug{x,y}➔{x,0}, if y>0
- Empty the 4-gallon jug into the 3-gallon one {x,y} ➔ {0,x+y}(if x+y<=3 and y>0)
- Empty the 3-gallon into the 4-gallon one{x,y} ➔ {x+y,0} (if x+y<=4)
- Fill the 4-gallon jug from the 3-gallon until it become full {x,y} ➔{4,y-(4-x)}or {4,x+y-4} (if x+y >4)
- Fill the 3-gallon jug from the 4-gallon until it become full {x,y} ➔{x-(3-y) or x+y-3 ,3}(if x+y >3)

Possible answers

- **Option 1**
  - **(0,3)---rule2**
  - **(3,0)---rule4**
  - **(3,3)---rule2**
  - **(4,2)---rule5**
  - **(0,2)---rule 8**
  - **(2,0)---rule4**
- **Option2**
  - **(4,0)---rule1**
  - **(1,3)---rule6**
  - **(1,0)---rule4**
  - **(0,1)---rule6**
  - **(4,1)---rule1**
  - **(2,3)---rule6**
  - **(2,0)---rule7**

Example D: The wolf-goat-cabbage problem

Cont..

A farmer has a goat, a wolf and a cabbage on the west side of the river. He wants to get all of his animals and his cabbage across the river onto the cost side. The farmer has a boat but he only has enough room for himself and one other thing.

Case 1: The goat will eat the cabbage if they are left together alone.

Case 2: The wolf will eat the goat if they are left alone.

How can the farmer get everything on the other side?

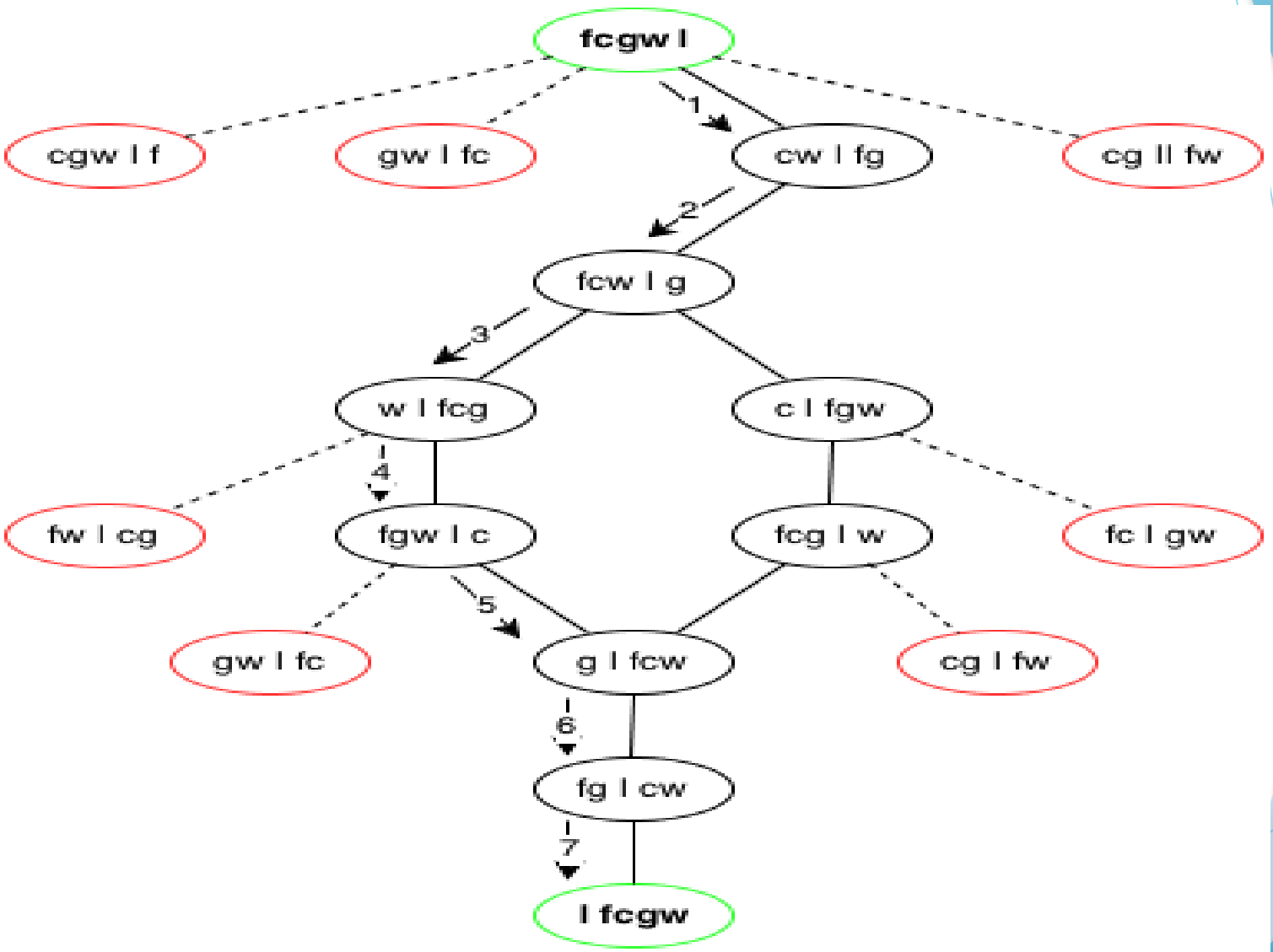
Possible solution

- State Space Representation:
  - We can represent the states of the problem with two sets W and E. We can also can have representations for the elements in the two sets as f, g ,w, c representing the farmer, goat, wolf, and cabbage.
  - Actions :
    - Move f from the E to W and vice versa
    - Move f and one of g,c,w from E to W and vice versa.
  - Start state:
    - W={f, g, c, w}, E={}
  - Goal state:
    - W={},E={f, g,c,w}

One possible Solution:

- Farmer takes goat across the river, W={w,c},E={f,g}
- Farmer comes back alone, W={f,c,w,},E={g}
- Farmer takes wolf across the river, W={c},E={f,g,w}
- Farmer comes back with goat, W=={f,g,c},E={w}
- Farmer takes cabbage across the river, W={g},E={f,w,c}
- Farmer comes back alone, W={f, g}, E={w, c}
- Farmer takes goat across the river, W={},E={f, g,w,c}

Cont..



Simple Depth First Search on State Space Graph

Quiz

1. The **missionaries and cannibals** problem is usually stated as follows. Three missionaries and three cannibals are on one side of a river, along with a boat that can hold one or two people. Find a way to get everyone to the other side without ever leaving a group of missionaries in one place outnumbered by the cannibals in that place.

A. Formulate the problem precisely, making only those distinctions necessary to ensure a valid solution.

2. For each of the following activities, give a PEAS description of the task environment

   - Shopping for used AI books on the Internet.
   - Playing soccer

Search Algorithm Terminologies

- **Search:** Searching is a step by step procedure to solve a search-problem in a given search space. A search problem can have three main factors:
  - **Search Space:** it represents a set of possible solutions, which a system may have.
  - **Start State:** It is a state from where agent begins the **search**.
  - **Goal test:** It is a function which observe the current state and returns whether the goal state is achieved or not.
- **Search tree:** A tree representation of search problem is called Search tree. The root of the search tree is the root node which is corresponding to the initial state.
- **Path Cost:** It is a function which assigns a numeric cost to each path.
- **Solution:** It is an action sequence which leads from the start node to the goal node.
- **Optimal Solution:** If a solution has the lowest cost among all solutions.

Properties of Search Algorithms

- The following are the four essential properties of search algorithms to compare the efficiency of these algorithms
- **Completeness:** A search algorithm is said to be complete if it guarantees to return a solution if at least any solution exists for any random input.
- **Optimality:** If a solution found for an algorithm is guaranteed to be the best solution (lowest path cost) among all other solutions, then such a solution for is said to be an optimal solution.

Cont..

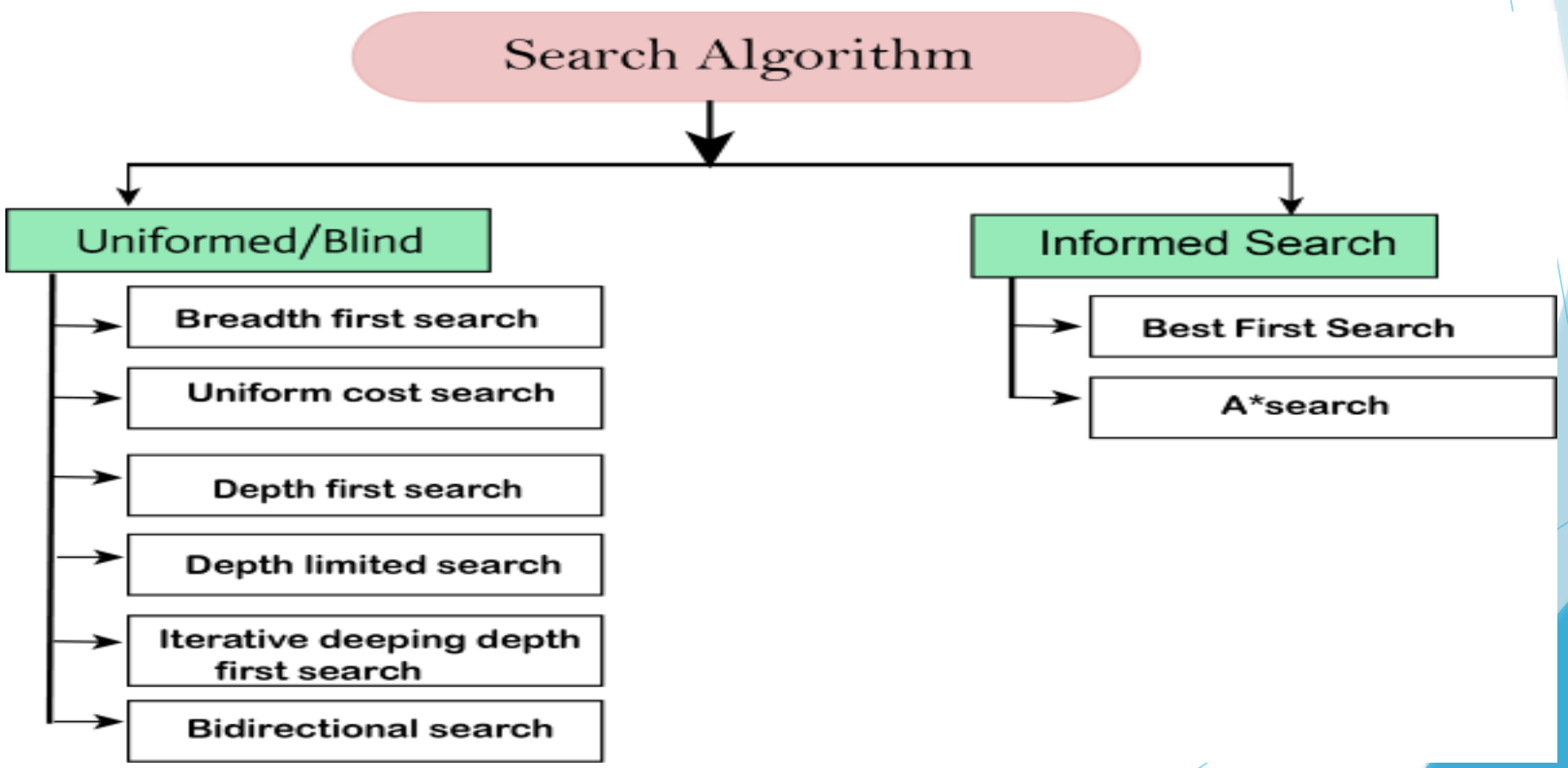
- **Time Complexity:** is a measure of time for an algorithm to complete its task.
- **Space Complexity:** It is the maximum storage space required at any point during the search, as the complexity of the problem.

Cont..

- Different search strategies are evaluated along completeness, time complexity, space complexity and optimality. The time and space complexity are measured in terms of:
- b: Maximum branching factor of the search tree (actions per state).
- d: Depth of the solution
- m: Maximum depth of the state space (may be ∞) (also noted sometimes D).

Types of Searching Algorithms

- Based on the search problems, we can classify the search algorithms into uninformed (**Blind** search) and informed search (**Heuristic** search) algorithms.



Uninformed Searching

1) Search without Information
2) No knowledge
3) Time Consuming
4) More Complexity (Time, Space)
5) DFS, BFS etc.

Informed Searching

1) Search with information
2) Use knowledge to find steps to solution
3) Quick solution
4) Less Complexity (Time, Space)
5) A*, Heuristic DFS, Best first Search

Uninformed/Blind Search

- The uninformed search **does not contain** any domain knowledge such as **closeness**, the **location** of the goal.
- It operates in a **brute-force** way as it only includes information about how to traverse the tree and how to identify **leaf** and **goal** nodes.
- Uninformed search applies a way in which search tree is searched **without any information** about the search space like initial state operators, so it is also called **blind** search.

Cont..

- It examines each node of the tree until it achieves the goal node.
- **It can be divided into six main types:**
  - Breadth-first Search(BFS)
  - Depth-first Search(DFS)
  - Depth-Limited Search(DLS)
  - Iterative Deepening Depth-first Search(IDDS)
  - Uniform Cost Search(UCS)
  - Bidirectional Search(BDS)

Breadth-first Search

- **Breadth-first search** is the most common search strategy for traversing a tree or graph. This algorithm searches breadthwise in a tree or graph, so it is called breadth-first search.
- BFS algorithm starts searching from the root node of the tree and expands all successor nodes at the current level before moving to nodes of next level.
- The **breadth-first search** algorithm is an example of a general-graph search algorithm.

Cont..

- Breadth-first search implemented using FIFO queue data structure.

Advantages:

- BFS will provide a solution if any solution exists.
- If there are more than one solutions for a given problem, then BFS will provide the minimal solution which requires the least number of steps.

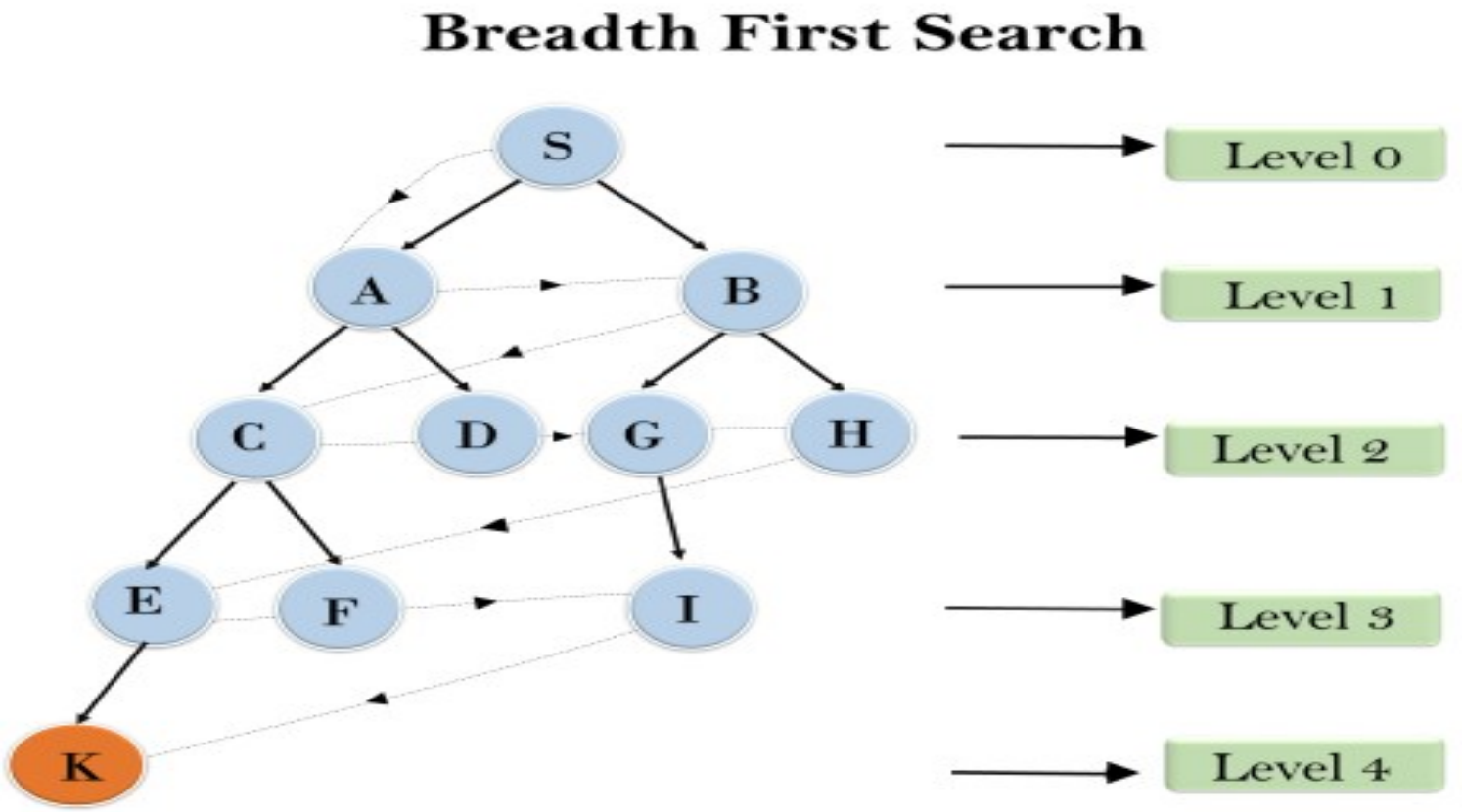
Cont..

Disadvantages:

- It requires lots of memory since each level of the tree must be saved into memory to expand the next level.
- BFS needs lots of time if the solution is far away from the root node.
- **Example:**
- In the below tree structure, we have shown the traversing of the tree using BFS algorithm from the root node S to goal node K.

BFS algorithm traverse in layers, so it will follow the path which is shown by the dotted arrow, and the traversed path will be:

S---> A--->B---->C--->D---->G--->H--->E---->F---->I---->K



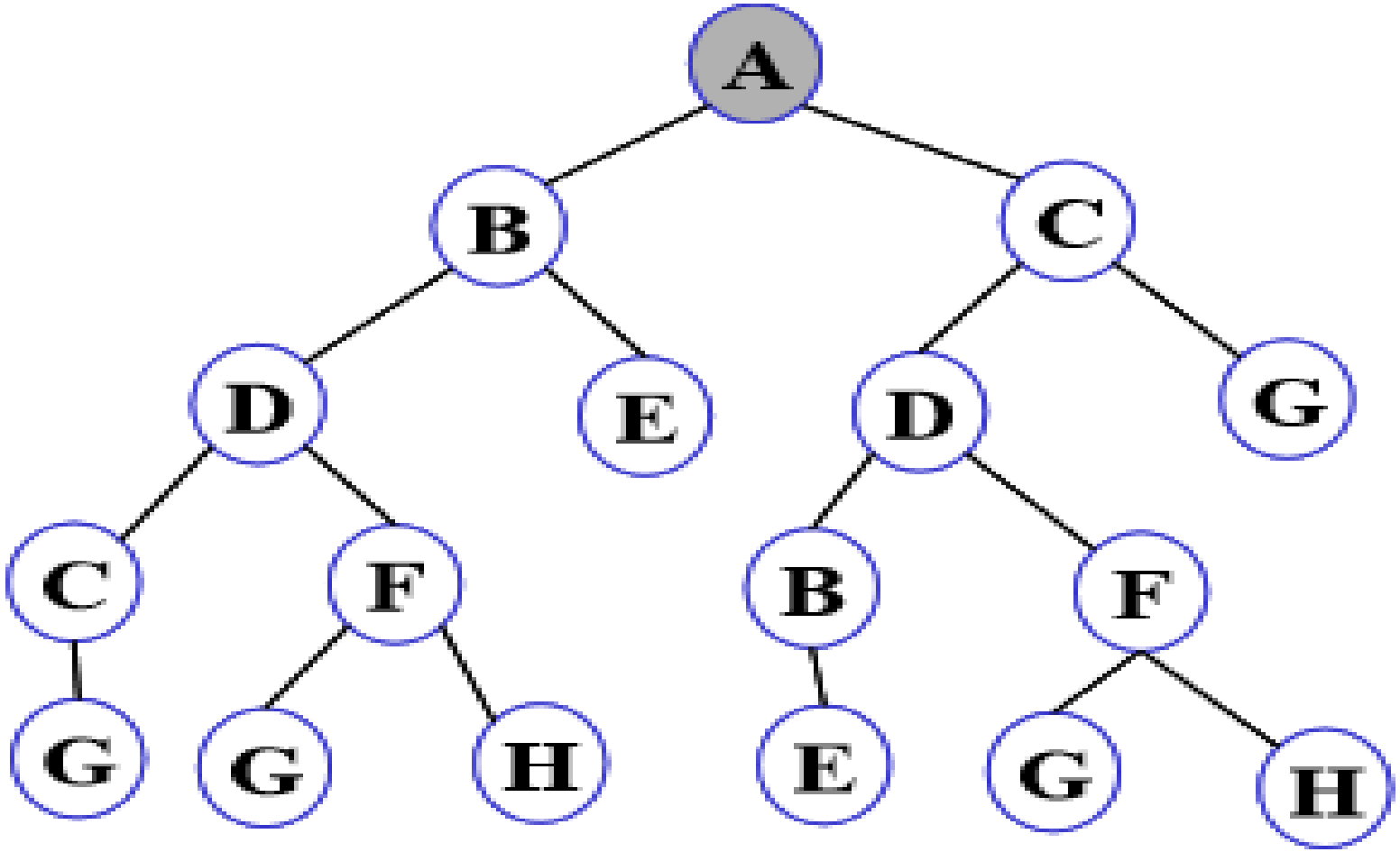
Cont..

- To simplify the search algorithms, it is often convenient to logically and programmatically **represent a problem space as a tree**.
- A tree is a graph in which any two vertices are connected exactly one path. Alternatively, any connected graph with no cycles is a tree.

Cont..

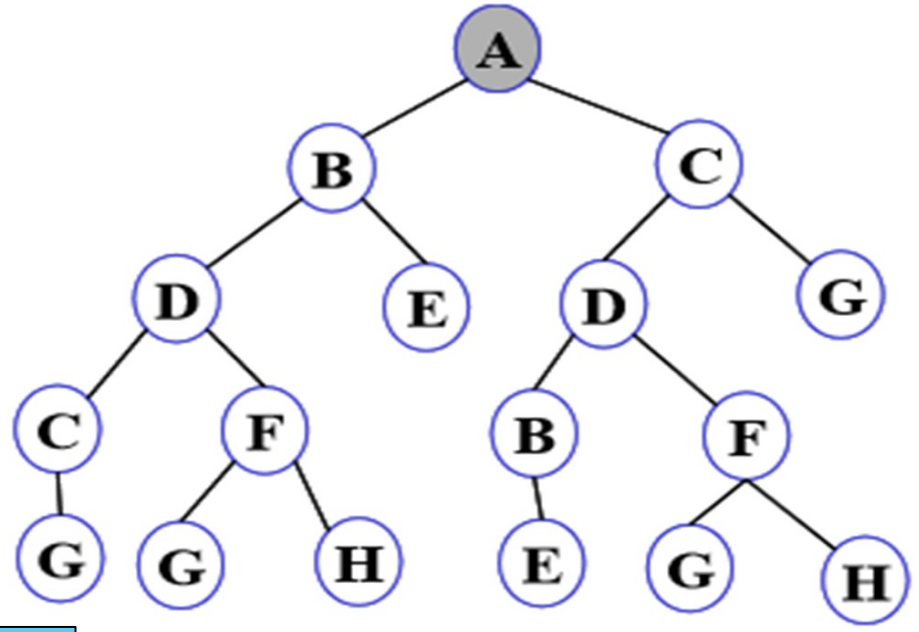
- The process constructs a search tree is as follows, where
  - root is the initial state and
  - leaf nodes are nodes
    - not yet expanded (i.e., in fringe/agenda)
    - having no successors (i.e., “dead-ends”)
- Search tree may be infinite because of loops even if state space is small
- The search problem will return as a solution a path to a goal node.
- Treat agenda as queue
  - Expansion: put children at the end of the queue
  - Get new nodes from the front of the queue

Search tree



BFS illustrated

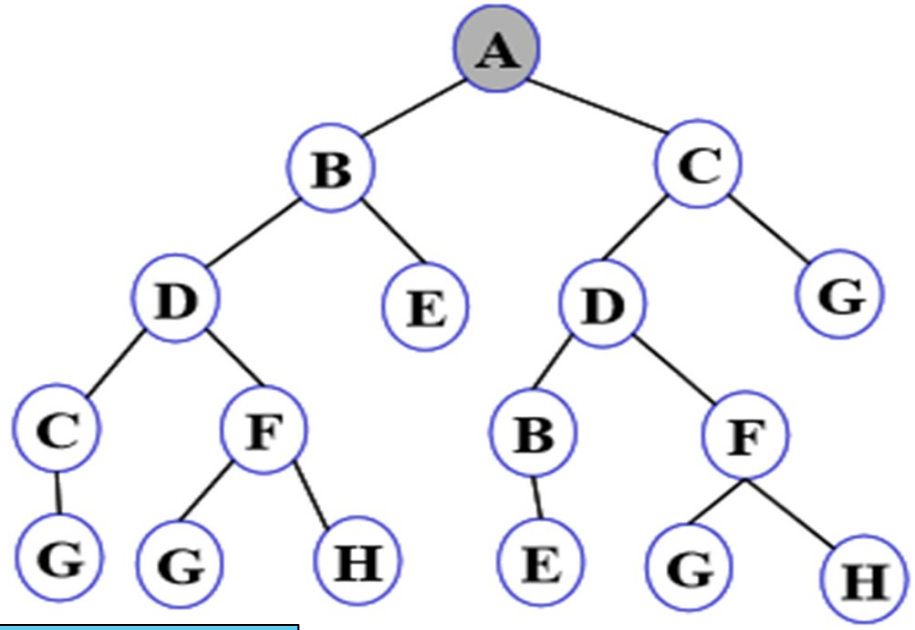
Step 1: Initially fringe contains only one node corresponding to the source state A.



Fringe: A

BFS illustrated

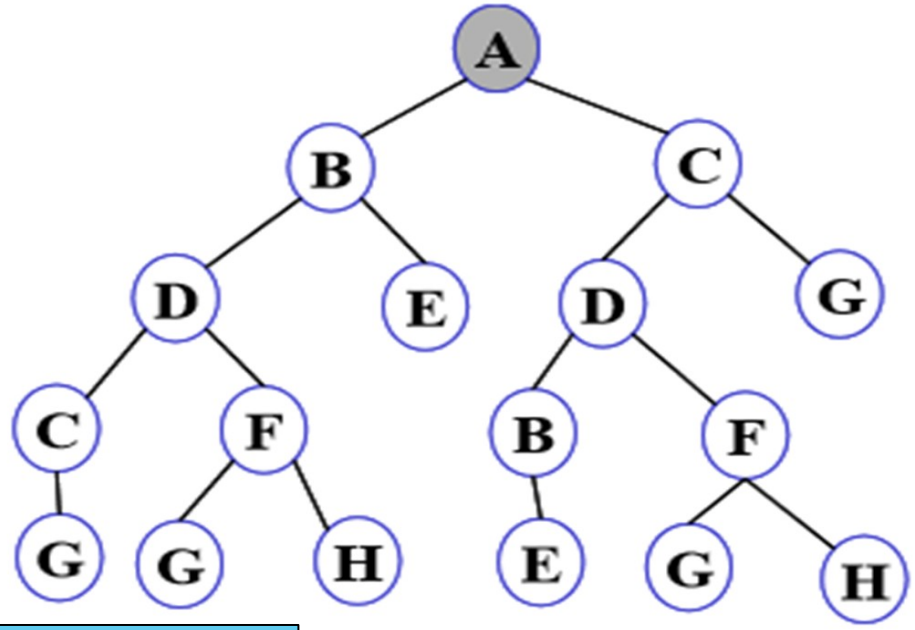
Step 2: A is removed from fringe. The node is expanded, and its children B and C are generated. They are placed at the back of fringe.



Fringe: BC

BFS illustrated

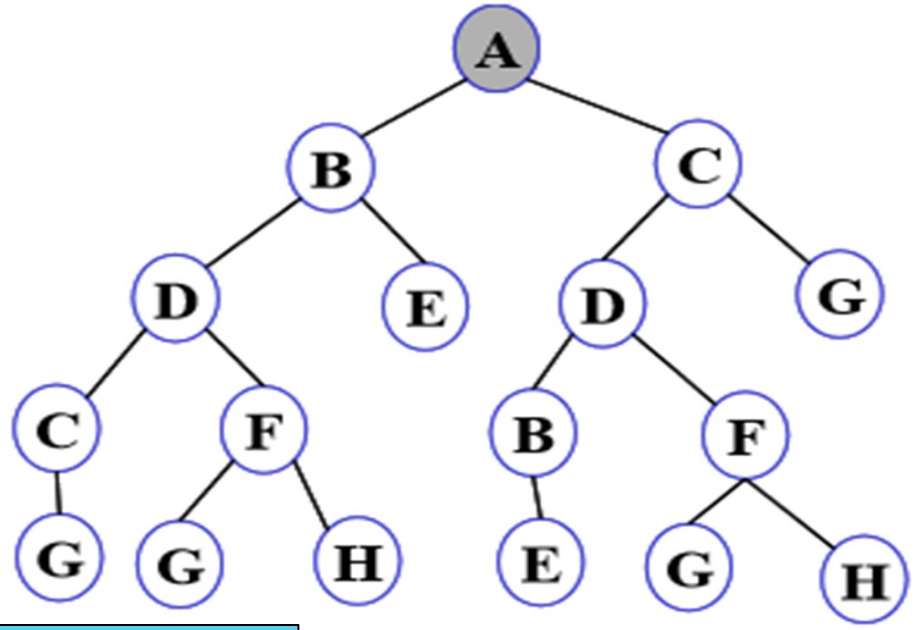
Step 3: Node B is removed from fringe and is expanded. Its children D, E are generated and put at the back of fringe.



Fringe: CDE

BFS illustrated

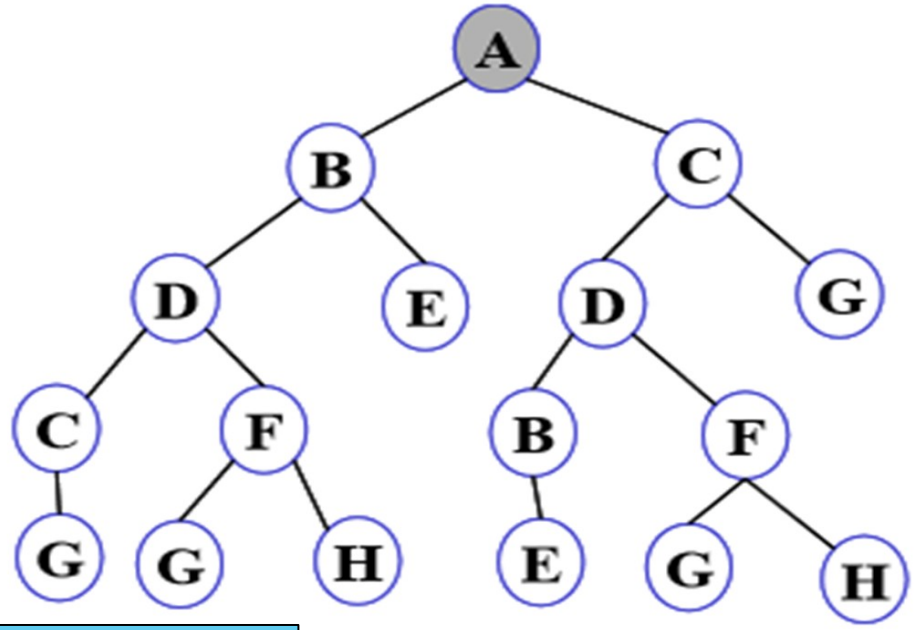
Step 4: Node C is removed from fringe and is expanded. Its children D and G are added to the back of fringe.



Fringe: DEDG

BFS illustrated

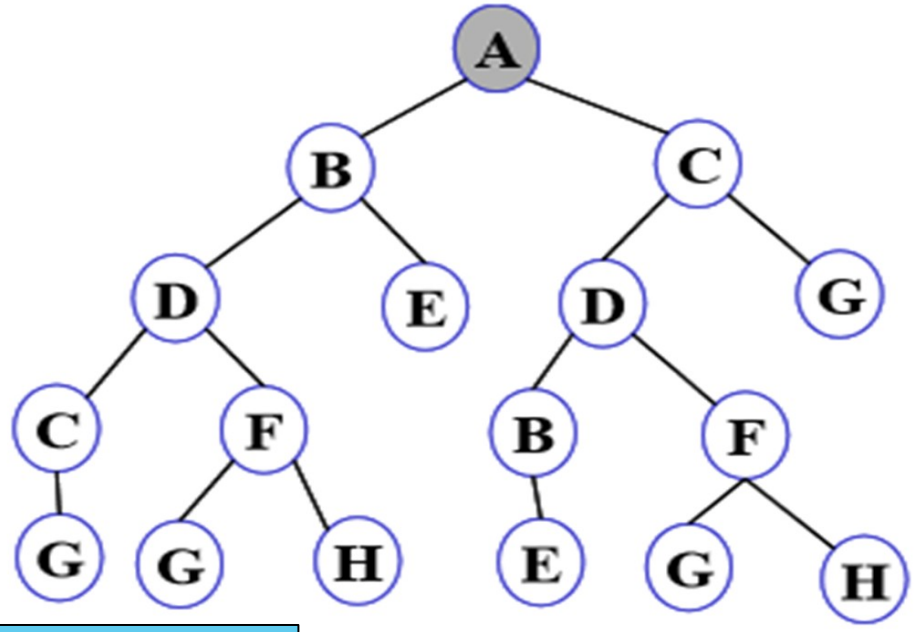
Step 5: Node D is removed from fringe. Its children C and F are generated and added to the back of fringe.



Fringe: EDGCF

BFS illustrated

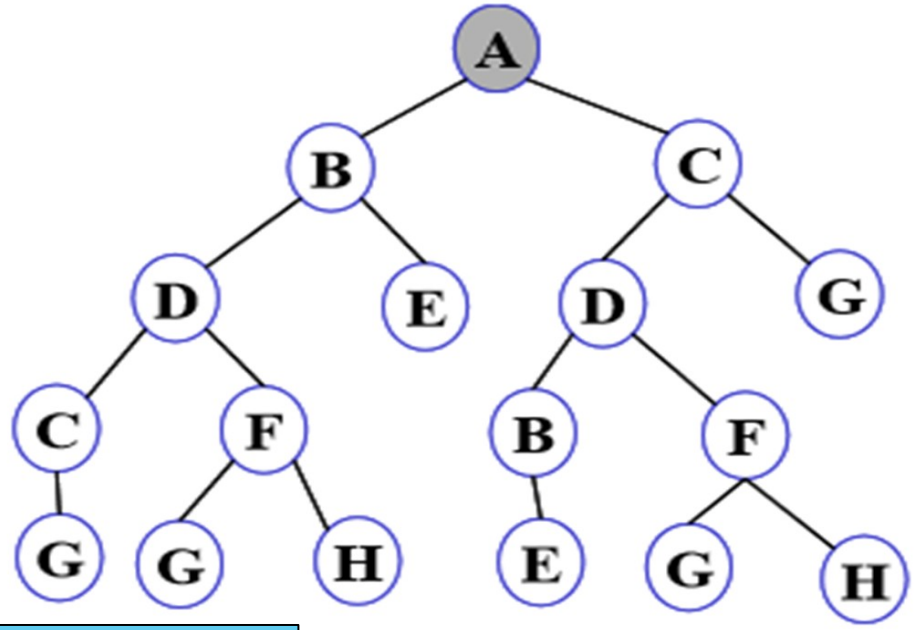
Step 6: Node E is removed from fringe. It has no children.



Fringe: DGCF

BFS illustrated

Step 7: D is expanded, B and F are put in OPEN.

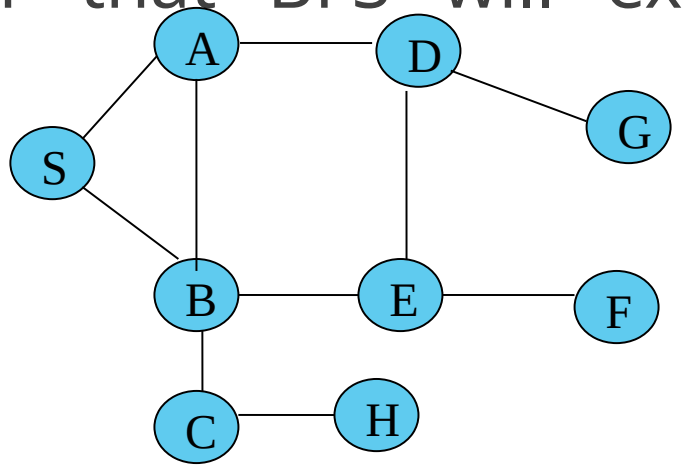


Fringe: GCFBF

cont..

Step 8: G is selected for expansion. It is found to be a goal node. So the algorithm returns the path A C G by following the parent pointers of the node corresponding to G. The algorithm terminates.

- Exercise: Consider the following graph representing the state space and operators of a navigation problem: What is the order that BFS will expand the nodes



- S is the start state and G is the goal state. When placing expanded child nodes on the queue, assume that the child nodes are placed in the alphabetical order(if node S is expanded the queue will be A,B). Assume that we never generate child nodes that appear as ancestors of the current node in

Cont..

- **Time Complexity:** Time Complexity of BFS algorithm can be obtained by the number of nodes traversed in BFS until the shallowest Node. Where d= depth of shallowest solution and b is a node at every state. $T(b) = 1+b2+b3+.....+ b^d= O (b^d)$
- **Space Complexity:** Space complexity of BFS algorithm is given by the Memory size of frontier which is $O(b^d)$.
- **Completeness:** BFS is complete, which means if the shallowest goal node is at some finite depth, then BFS will find a solution.
- **Optimality:** BFS is optimal if path cost is a non-decreasing function of the depth of the node.

Depth-first Search(DFS)

- Depth-first search is a recursive algorithm for traversing a tree or graph data structure.
- It is called the depth-first search because it starts from the root node and follows each path to its greatest depth node before moving to the next path.
- DFS uses a stack data structure for its implementation.
- The process of the DFS algorithm is similar to the BFS algorithm.

Cont..

Advantages:

- DFS requires very less memory as it only needs to store a stack of the nodes on the path from root node to the current node.
- It takes less time to reach to the goal node than BFS algorithm (if it traverses in the right path).

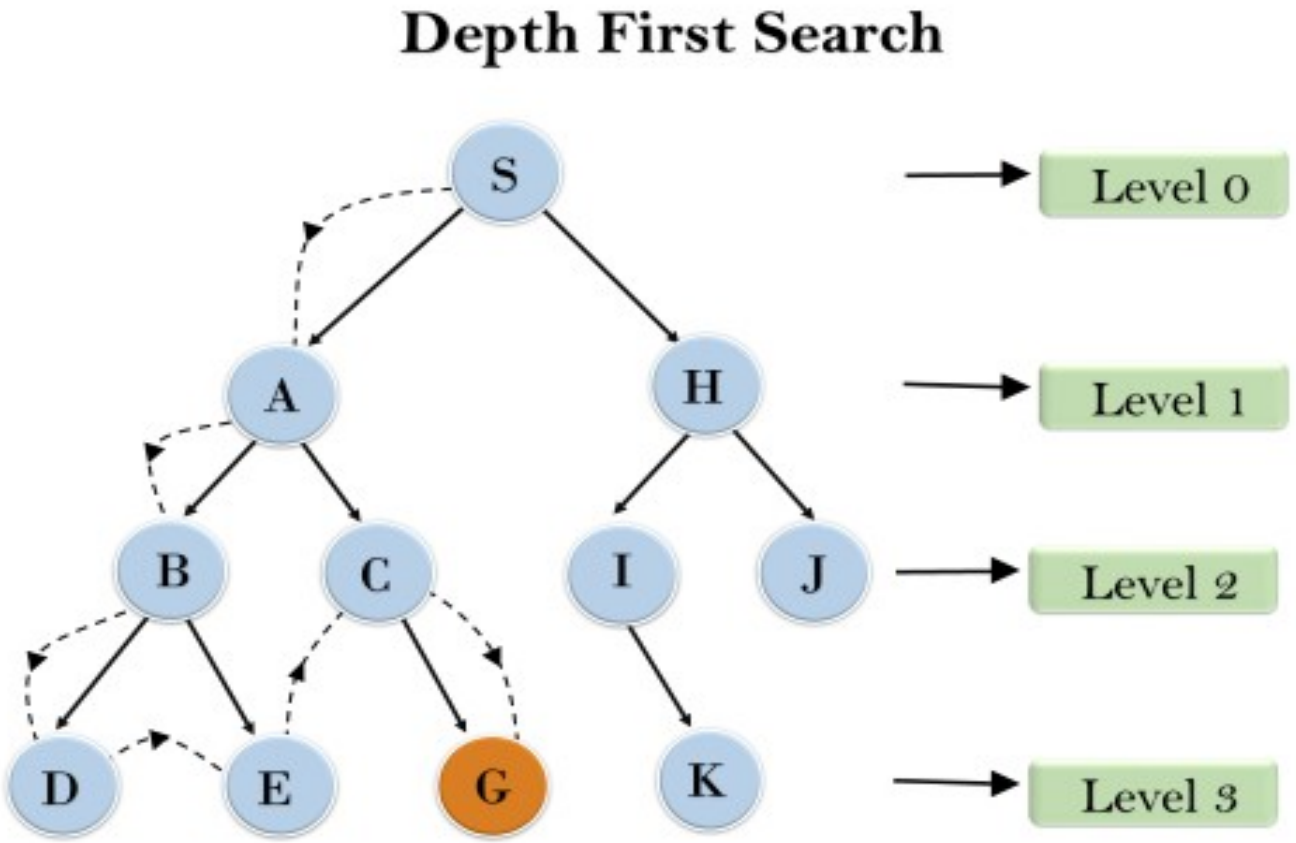
Disadvantages:

- There is the possibility that many states keep re-occurring, and there is no guarantee of finding the solution.
- DFS algorithm goes for deep down searching and sometime it may go to the infinite loop.

Cont..

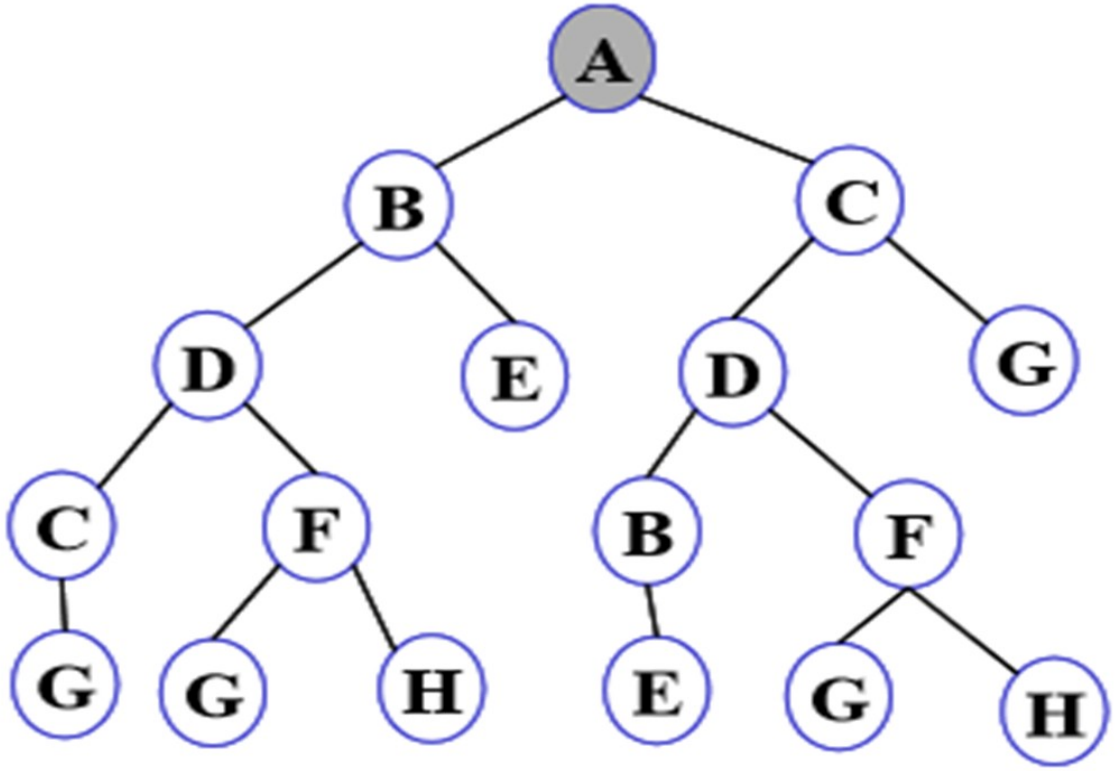
- Example:
- In the below search tree, we have shown the flow of depth-first search, and it will follow the order as:
- Root node--->Left node ----> right node.
- It will start searching from root node S, and traverse A, then B, then D and E, after traversing E, it will backtrack the tree as E has no other successor and still goal node is not found. After backtracking it will traverse node C and then G, and here it will terminate as it found goal node.

Cont..


Depth First Search
S
A
H
B
C
I
J
D
E
G
K
Level 0
Level 1
Level 2
Level 3

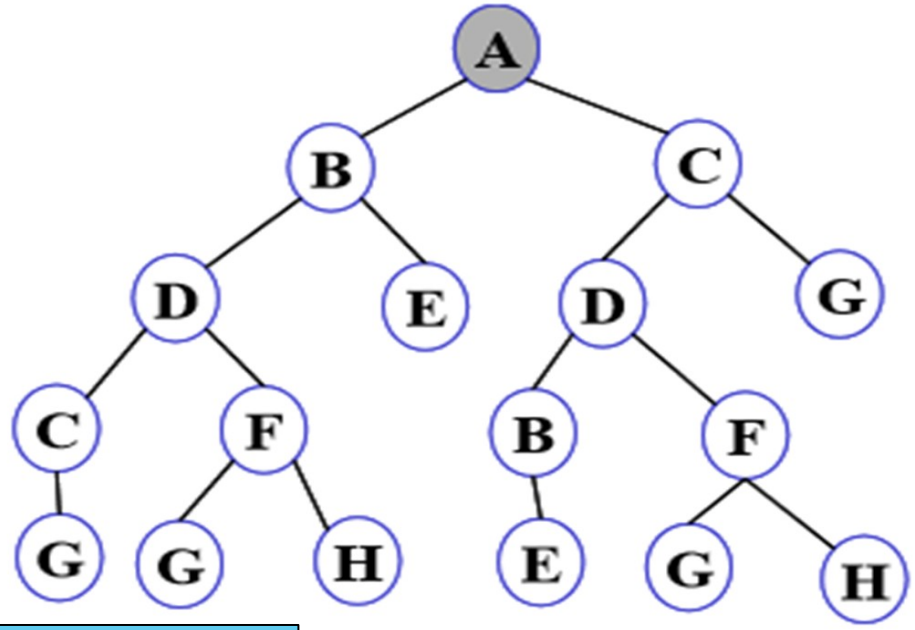
Search tree for the state space

Let us now run Depth First Search on the search space given in Figure below, and trace its progress



DFS illustrated

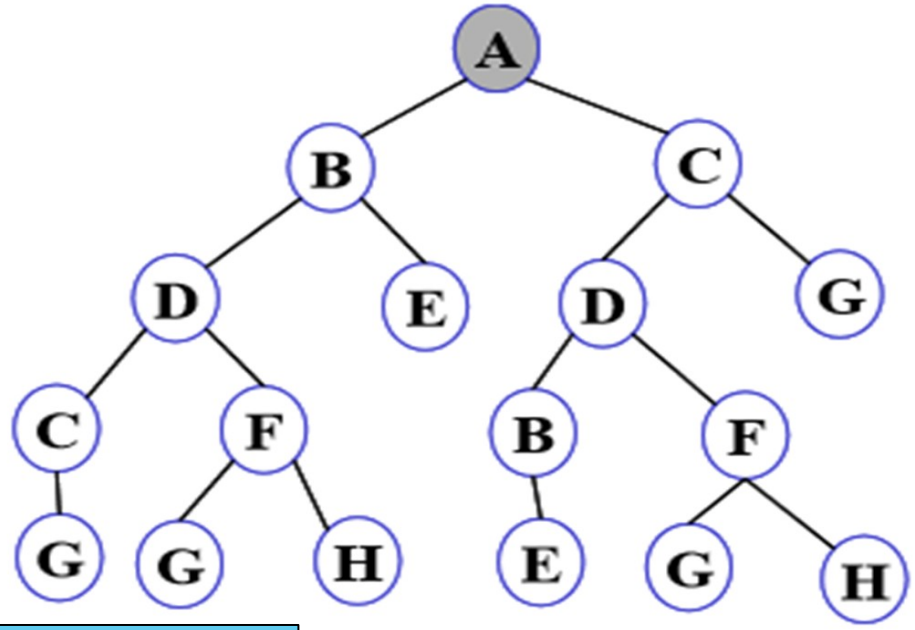
Step 1: Initially fringe contains only the node for A.



Fringe: A

DFS illustrated

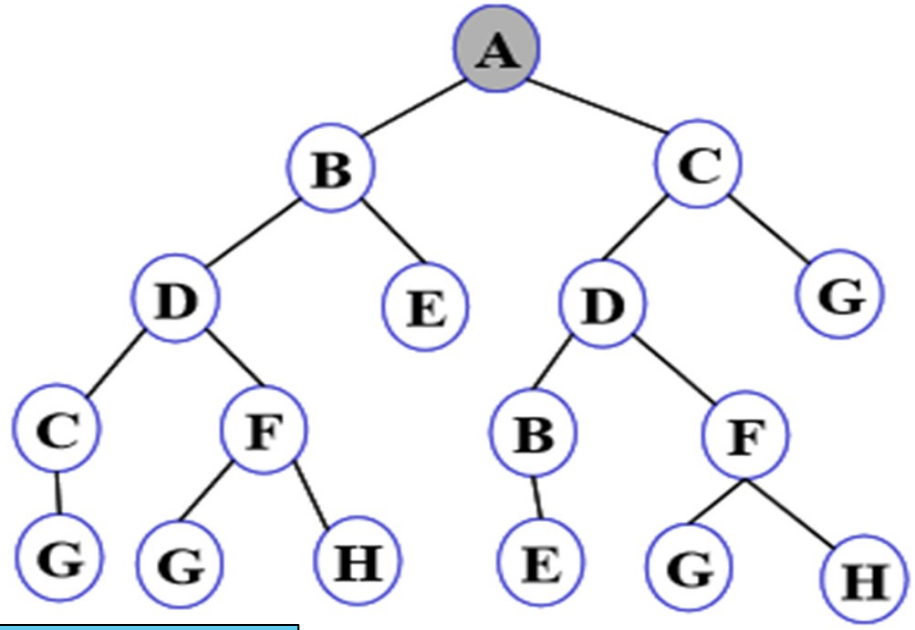
Step 2: A is removed from fringe. A is expanded and its children B and C are put in front of fringe



Fringe: BC

DFS illustrated

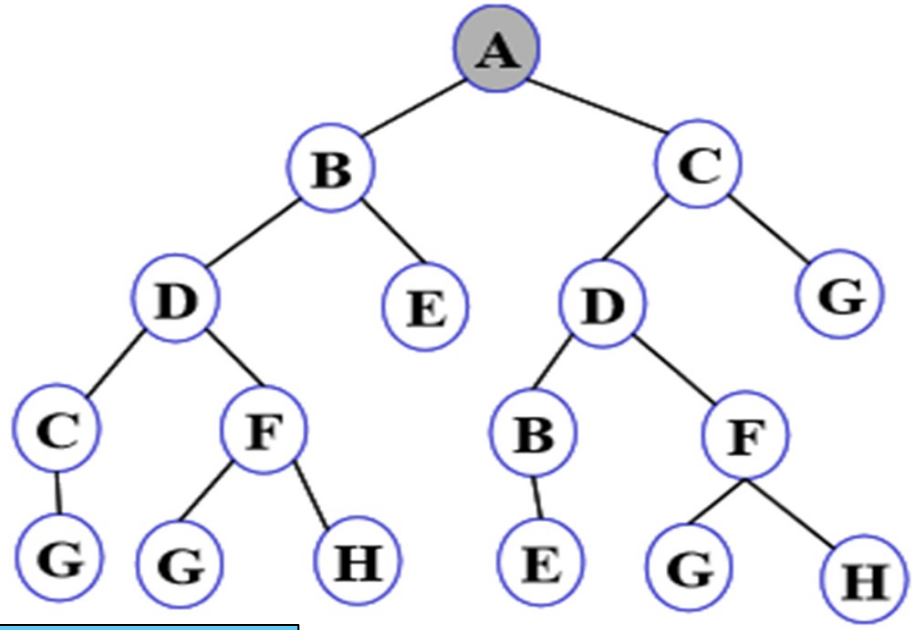
Step 3: Node B is removed from fringe, and its children D and E are pushed in front of fringe.



Fringe: DEC

DFS illustrated

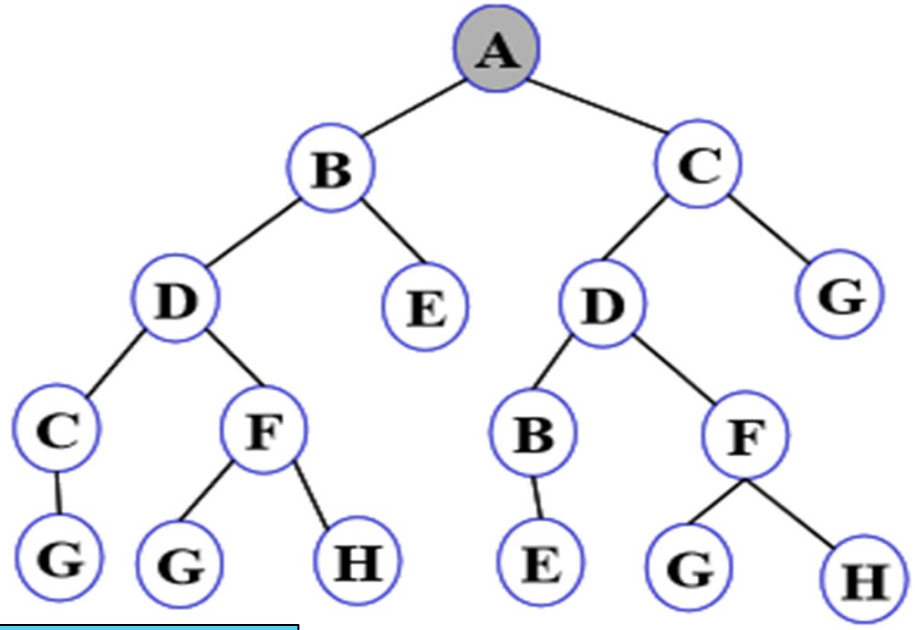
Step 4: Node D is removed from fringe. C and F are pushed in front of fringe.



Fringe: CFEC

DFS illustrated

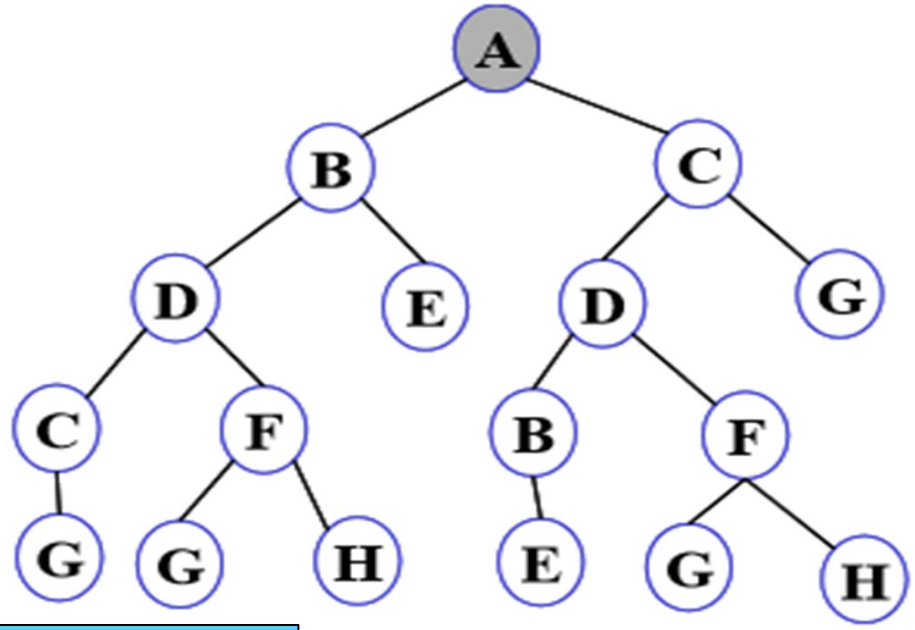
Step 5: Node C is removed from fringe. Its child G is pushed in front of fringe.



Fringe: GFEC

DFS illustrated

Step 6: Node G is expanded and found to be a goal node. The solution path A-B-D-C-G is returned and the algorithm terminates.



Goal.

Fringe: GFEC

Cont..

- **Completeness:** DFS search algorithm is complete within finite state space as it will expand every node within a limited search tree.
- **Time Complexity:** Time complexity of DFS will be equivalent to the node traversed by the algorithm. It is given by: T(n)= 1+ n2+ n3 +.....+ nm=O(nm)
- Where, m= maximum depth of any node and this can be much larger than d (Shallowest solution depth)
- **Space Complexity:** DFS algorithm needs to store only single path from the root node, hence space complexity of DFS is equivalent to the size of the fringe set which is O(bm).
- **Optimal:** DFS search algorithm is non-optimal, as it may generate a large number of steps or high cost to reach to the goal node.

Depth-Limited Search Algorithm

- A depth-limited search algorithm is similar to depth-first search with a predetermined limit.
- Depth-limited search can solve the drawback of the infinite path in the Depth-first search.
- In this algorithm, the node at the depth limit will treat as it has no successor nodes further.
- Depth-limited search can be terminated with two Conditions of failure:
- Standard failure value: It indicates that problem does not have any solution.

Cont..

- Cutoff failure value: It defines no solution for the problem within a given depth limit.

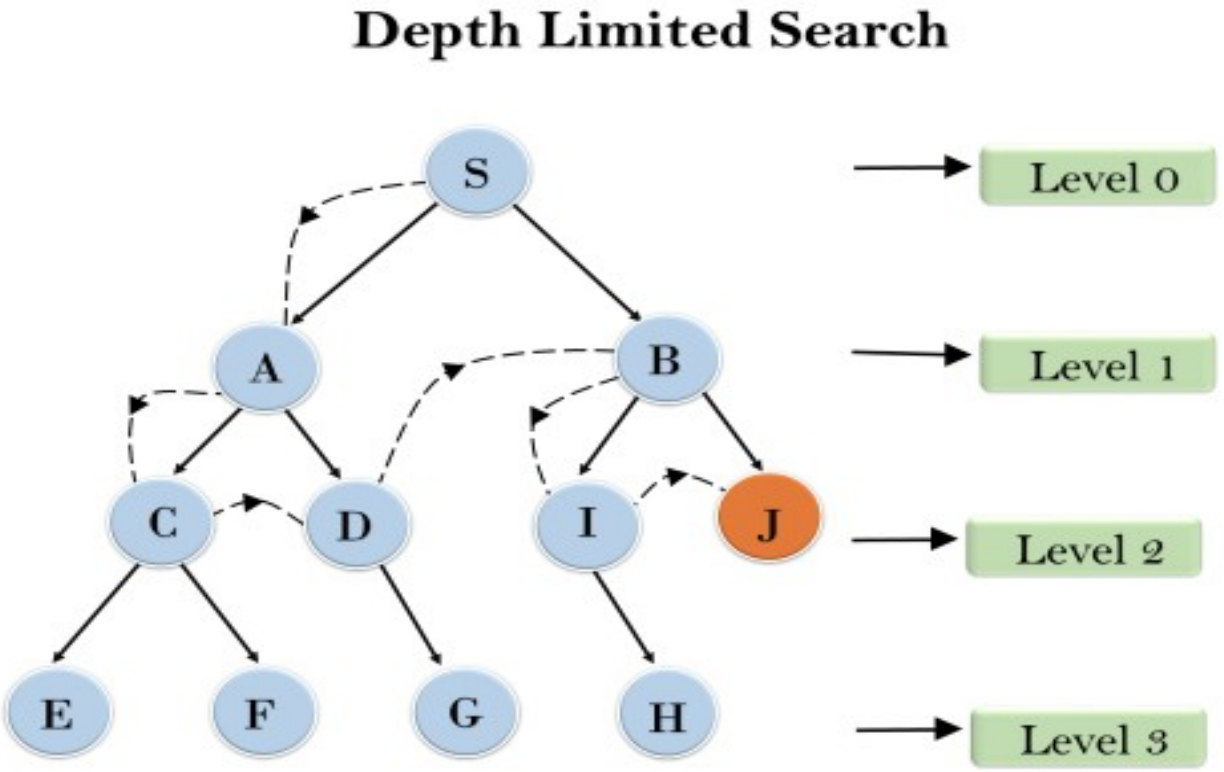
Advantages:

- Depth-limited search is Memory efficient.

Disadvantages:

- Depth-limited search also has a disadvantage of incompleteness.
- It may not be optimal if the problem has more than one solution.

Cont..


Depth Limited Search
S
A
B
C
D
I
J
E
F
G
H
Level 0
Level 1
Level 2
Level 3

Cont..

- **Completeness:** DLS search algorithm is complete if the solution is above the depth-limit.
- **Optimal:** Depth-limited search can be viewed as a special case of DFS, and it is also not optimal.
- **Time Complexity:** Time complexity of DLS algorithm is O(bℓ).
- **Space Complexity:** Space complexity of DLS algorithm is O(b×ℓ).

Uniform-cost Search Algorithm

- Uniform-Cost Search(UCS) is a searching algorithm used for traversing a weighted tree or graph.
- This algorithm comes into play when a different cost is available for each edge.
- The primary goal of the uniform-cost search is to find a path to the goal node which has the lowest cumulative cost.
- Uniform-cost search expands nodes according to their path costs from the root node.
- It can be used to solve any graph/tree where the optimal cost is in demand.
- A uniform-cost search algorithm is implemented by the priority queue.

Cont..

- It gives maximum priority to the lowest cumulative cost.
- Uniform cost search is equivalent to BFS algorithm if the path cost of all edges is the same.

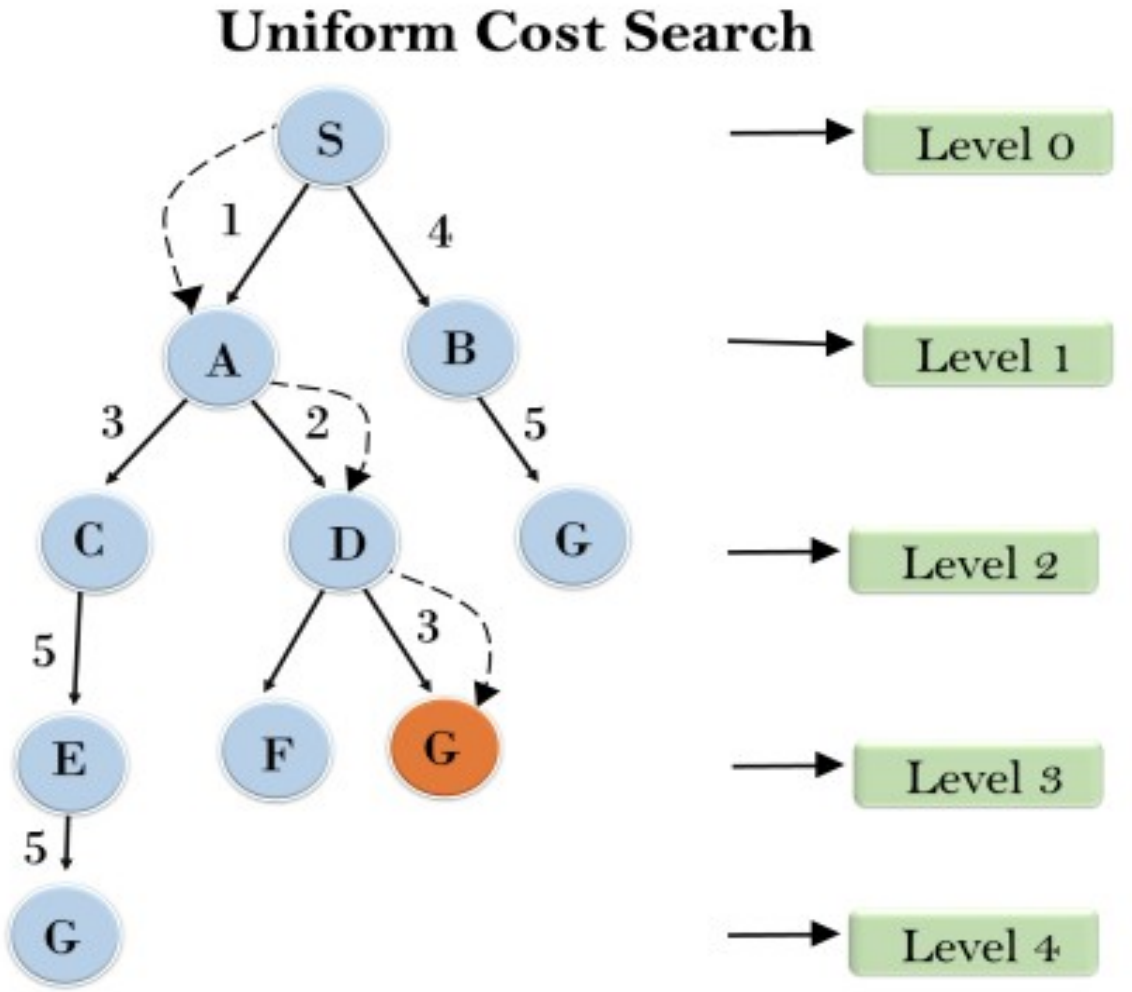
Advantages:

- Uniform cost search is optimal because at every state the path with the least cost is chosen.

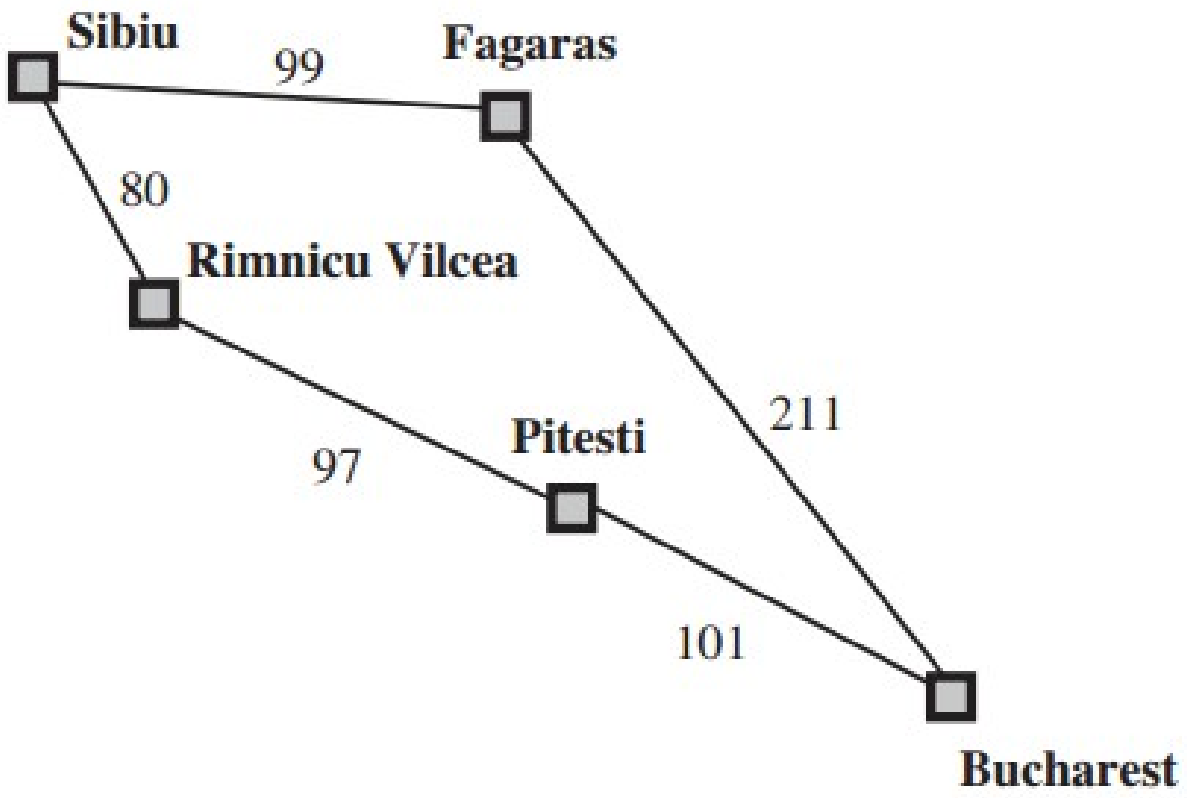
Disadvantages:

- It does not care about the number of steps involve in searching and only concerned about path cost. Due to which this algorithm may be stuck in an infinite loop.

Cont..


Uniform Cost Search
S
A
B
C
D
G
E
F
G
G
1
4
3
2
5
5
3
5
Level 0
Level 1
Level 2
Level 3
Level 4

Cont..


Sibiu
99
Fagaras
80
Rimnicu Vilcea
211
Pitesti
97
101
Bucharest

Cont..

- The successors of Sibiu are Riminculus Vilcea (with cost of 80) and Fagaras (cost = 99). The least-cost node is Riminculus Vilcea, which is expanded next to get Pitesti whose path cost from Sibiu is now 80 + 97 =177. The least-cost node is now Fagaras, which is then expanded to get Bucharest with path cost 99 + 211 = 310.
- Now, we have generated the goal node, but the search still continues. What if the path through Pitesti reaches Bucharest with a lesser cost? Turns out this is the case in this situation. The Pitesti is expanded next, to give Bucharest with path cost 177 + 101 = 278. Bucharest, now with path cost 278, is chosen for expansion, and the solution is returned.

Cont..

Completeness:

- Uniform-cost search is complete such as if there is a solution, UCS will find it.

Time Complexity:

- Let **C* is Cost of the optimal solution**, and **ε** is each step to get closer to the goal node. Then, the number of steps is = **C*/ε**+1. Here we have taken +1, as we start from state 0 and end to **C*/ε**.
- Hence, the worst-case time complexity of Uniform-cost search is $\mathbf{O(b^{1 + [C^*/\varepsilon]})/}$.

Cont..

Space Complexity:

- The same logic is for space complexity. So, the worst-case space complexity of Uniform-cost search is $\mathbf{O(b^{1 + [C^*/\varepsilon]})}$.

Optimal:

- Uniform-cost search is always optimal as it only selects a path with the lowest path cost.

- The iterative deepening algorithm is a combination of DFS and BFS algorithms. This search algorithm finds out the best depth limit and does it by gradually increasing the limit until a goal is found.
- This algorithm performs depth-first search up to a certain "depth limit", and it keeps increasing the depth limit after each iteration until the goal node is found.
- This Search algorithm combines the benefits of Breadth-first search's fast search and depth-first search's memory efficiency.
- The iterative search algorithm is useful uninformed search when search space is large, and depth of goal node is unknown.

Cont..

Advantages:

- It combines the benefits of BFS and DFS search algorithm in terms of fast search and memory efficiency.

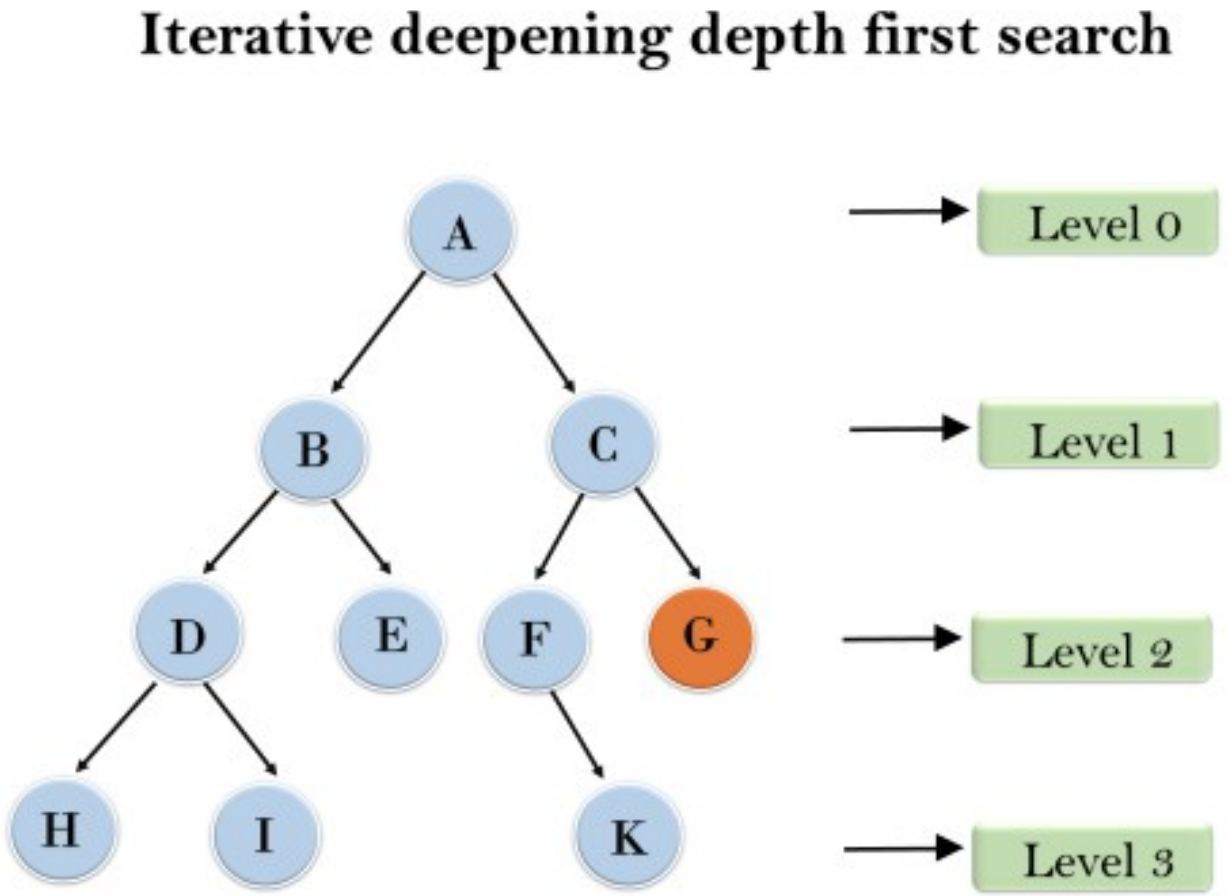
Disadvantages:

- The main drawback of IDDFS is that it repeats all the work of the previous phase.

Example:

- Following tree structure is showing the iterative deepening depth-first search. IDDFS algorithm performs various iterations until it does not find the goal node. The iteration performed by the algorithm is given as:

Cont..


Iterative deepening depth first search
A
B
C
D
E
F
G
H
I
K
Level 0
Level 1
Level 2
Level 3

Cont..

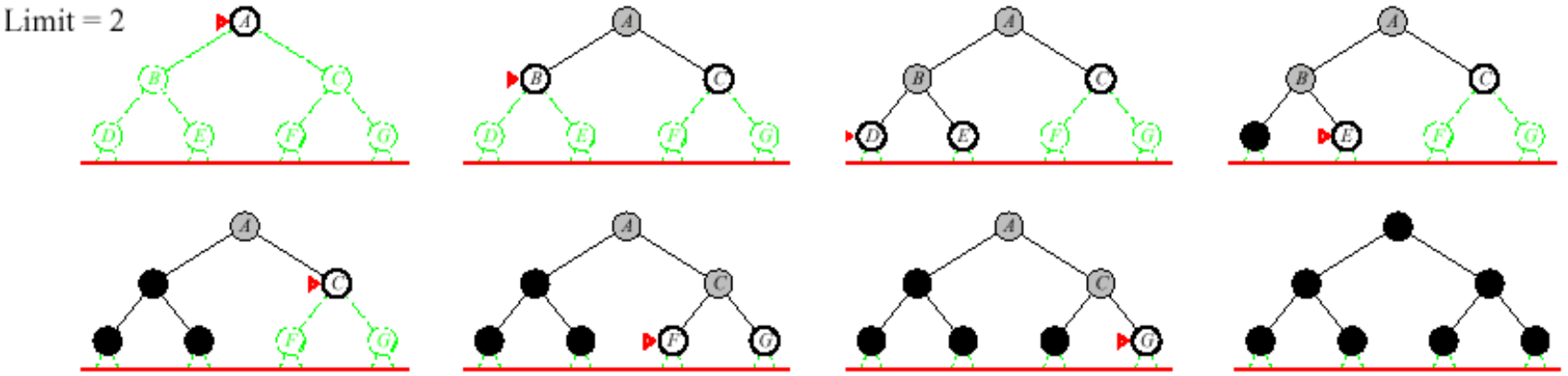
- 1'st Iteration-----> A
  2'nd Iteration----> A, B, C
  3'rd Iteration------>A, B, D, E, C, F, G
  4'th Iteration------>A, B, D, H, I, E, C, F, K, G
  In the fourth iteration, the algorithm will find the goal node.

Iterative deepening search $l = 1$

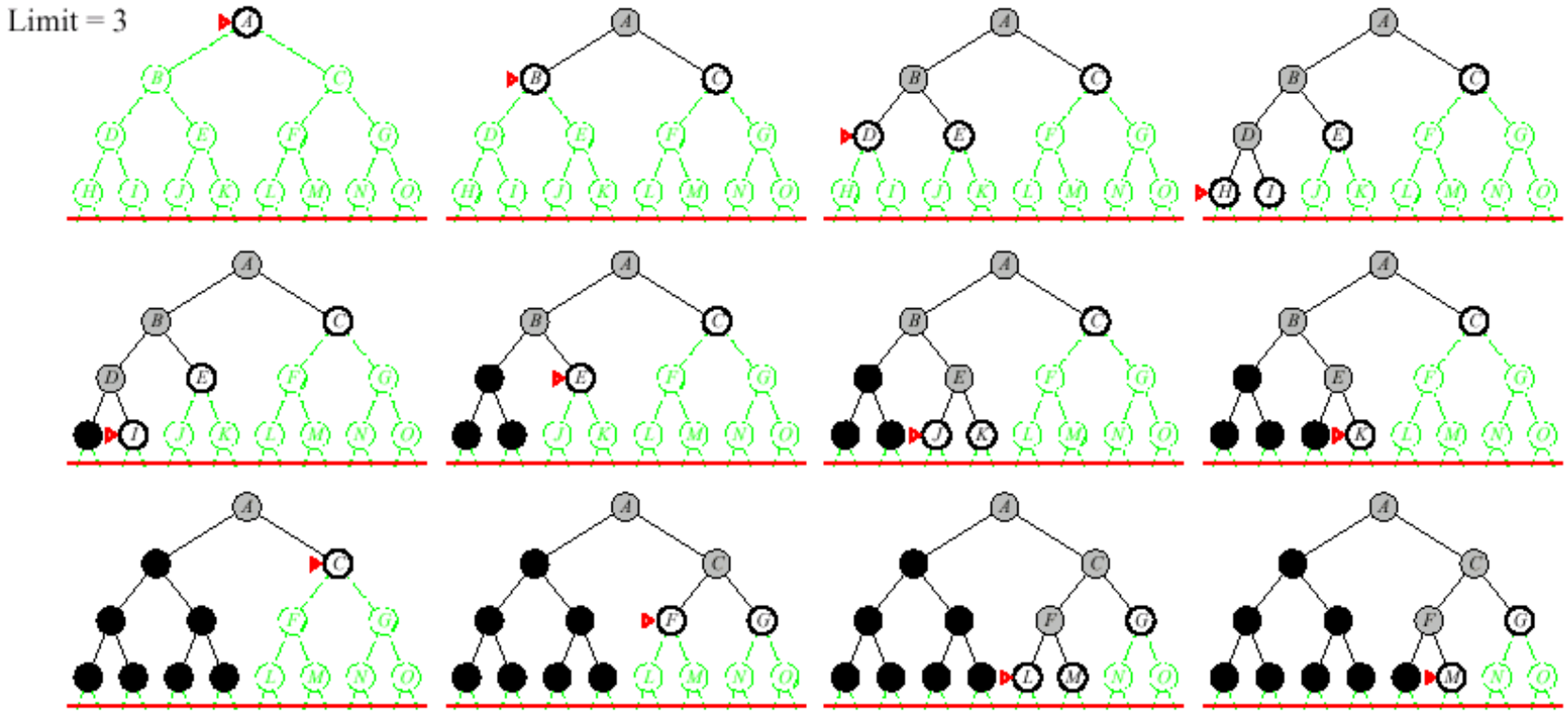
Limit = 1

Iterative deepening search $l = 2$

Limit = 2



Iterative deepening search $l = 3$



Cont..

Completeness:

- This algorithm is complete if the branching factor is finite.

Time Complexity:

- Suppose b is the branching factor and depth is d then the worst-case time complexity is **$O(b^d)$**.

Space Complexity:

- The space complexity of IDDFS will be **O(bd)**.

Optimal:

- IDDFS algorithm is optimal if path cost is a non decreasing function of the depth of the node.

Bidirectional Search Algorithm

- Bidirectional search algorithm runs two simultaneous searches, one from initial state called as forward-search and the other from goal node called as backward-search, to find the goal node.
- Bidirectional search replaces one single search graph with two small subgraphs in which one starts the search from an initial vertex and other starts from goal vertex.
- The search stops when these two graphs intersect each other.
- Bidirectional search can use search techniques such as BFS, DFS, DLS, etc.

Cont..

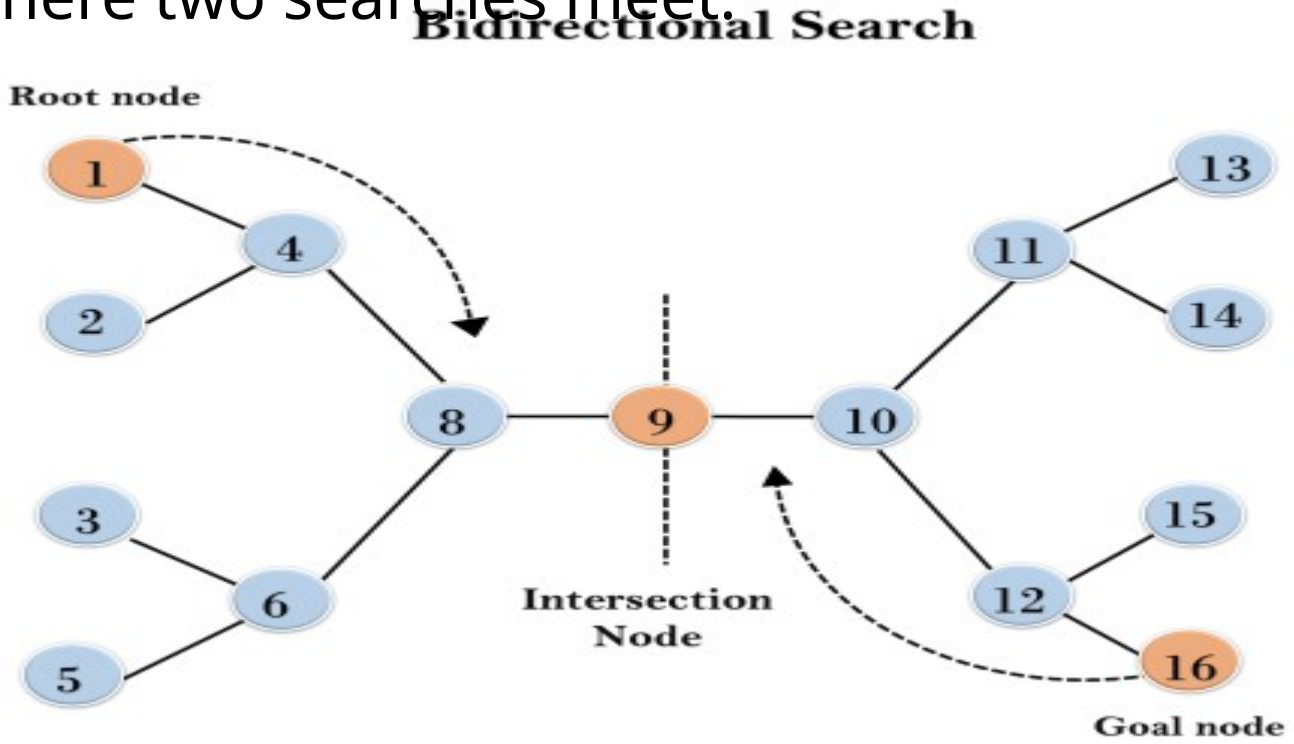
Advantages:

- Bidirectional search is fast.
- Bidirectional search requires less memory

Disadvantages:

- Implementation of the bidirectional search tree is difficult.

Example:
In the below search tree, bidirectional search algorithm is applied. This algorithm divides one graph/tree into two sub-graphs. It starts traversing from node 1 in the forward direction and starts from goal
node 16 in the backward direction. The algorithm terminates at node 9 where two searches meet.



Cont..

- **Completeness:** Bidirectional Search is complete if we use BFS in both searches.
- **Time Complexity:** Time complexity of bidirectional search using BFS is **$O(b^d)$**.
- **Space Complexity:** Space complexity of bidirectional search is **$O(b^d)$**.
- **Optimal:** Bidirectional search is Optimal.

Informed Search Algorithms

- So far we have talked about the uninformed search algorithms which looked through search space for all possible solutions of the problem without having any additional knowledge about search space.
- But informed search algorithm contains an array of knowledge such as how far we are from the goal, path cost, how to reach to goal node, etc.
- This knowledge help agents to explore less to the search space and find more efficiently the goal node.
- The informed search algorithm is more useful for large search space.
- Informed search algorithm uses the idea of heuristic, so it is also called Heuristic search.

Cont..

- **Heuristics Function:** Heuristic is a function which is used in Informed Search, and it finds the most promising path.
- It takes the current state of the agent as its input and produces the estimation of how close agent is from the goal.
- The heuristic method, however, might not always give the best solution, but it guaranteed to find a good solution in reasonable time.
- Heuristic function estimates how close a state is to the goal.
- It is represented by h(n), and it calculates the cost of an optimal path between the pair of states.
- The value of the heuristic function is always positive.

Pure Heuristic Search

- Pure heuristic search is the simplest form of heuristic search algorithms.
- It expands nodes based on their heuristic value h(n).
- It maintains two lists: OPEN and CLOSED lists.
- In the CLOSED list, it places those nodes which have already expanded and in the OPEN list, it places nodes which have yet not been expanded.

Cont..

- On each iteration, each node n with the lowest heuristic value is expanded and generates all its successors and n is placed to the closed list.
- The algorithm continues unitl a goal state is found.
- In the informed search we will discuss two main algorithms which are given below:
- **Best First Search Algorithm(Greedy search)**
- **A* Search Algorithm**

Best-first Search Algorithm (Greedy Search)

- Greedy best-first search algorithm always selects the path which appears best at that moment.
- It is the combination of depth-first search and breadth-first search algorithms.
- It uses the heuristic function and search.
- Best-first search allows us to take the advantages of both algorithms.
- With the help of best-first search, at each step, we can choose the most promising node.

Cont..

- In the best first search algorithm, we expand the node which is closest to the goal node and the closest cost is estimated by heuristic function.
- The greedy best first algorithm is implemented by the priority queue.

Cont..

- Best first search algorithm:
- **Step 1:** Place the starting node into the OPEN list.
- **Step 2:** If the OPEN list is empty, Stop and return failure.
- **Step 3:** Remove the node n, from the OPEN list which has the lowest value of h(n), and places it in the CLOSED list.
- **Step 4:** Expand the node n, and generate the successors of node n.
- **Step 5:** Check each successor of node n, and find whether any node is a goal node or not. If any successor node is goal node, then return success and terminate the search, else proceed to Step 6.
- **Step 6:** For each successor node, algorithm checks for evaluation function f(n), and then check if the node has been in either OPEN or CLOSED list. If the node has not been in both list, then add it to the OPEN list.
- **Step 7:** Return to Step 2.

Cont..

Advantages:

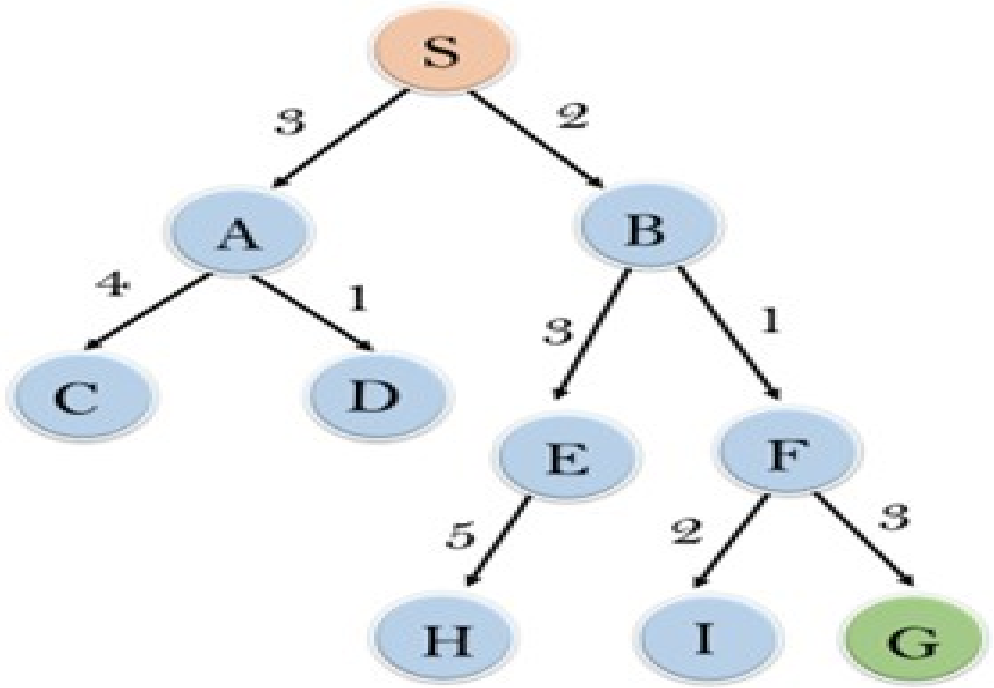
- Best first search can switch between BFS and DFS by gaining the advantages of both the algorithms.
- This algorithm is more efficient than BFS and DFS algorithms.

Disadvantages:

- It can behave as an unguided depth-first search in the worst case scenario.
- It can get stuck in a loop as DFS.
- This algorithm is not optimal.

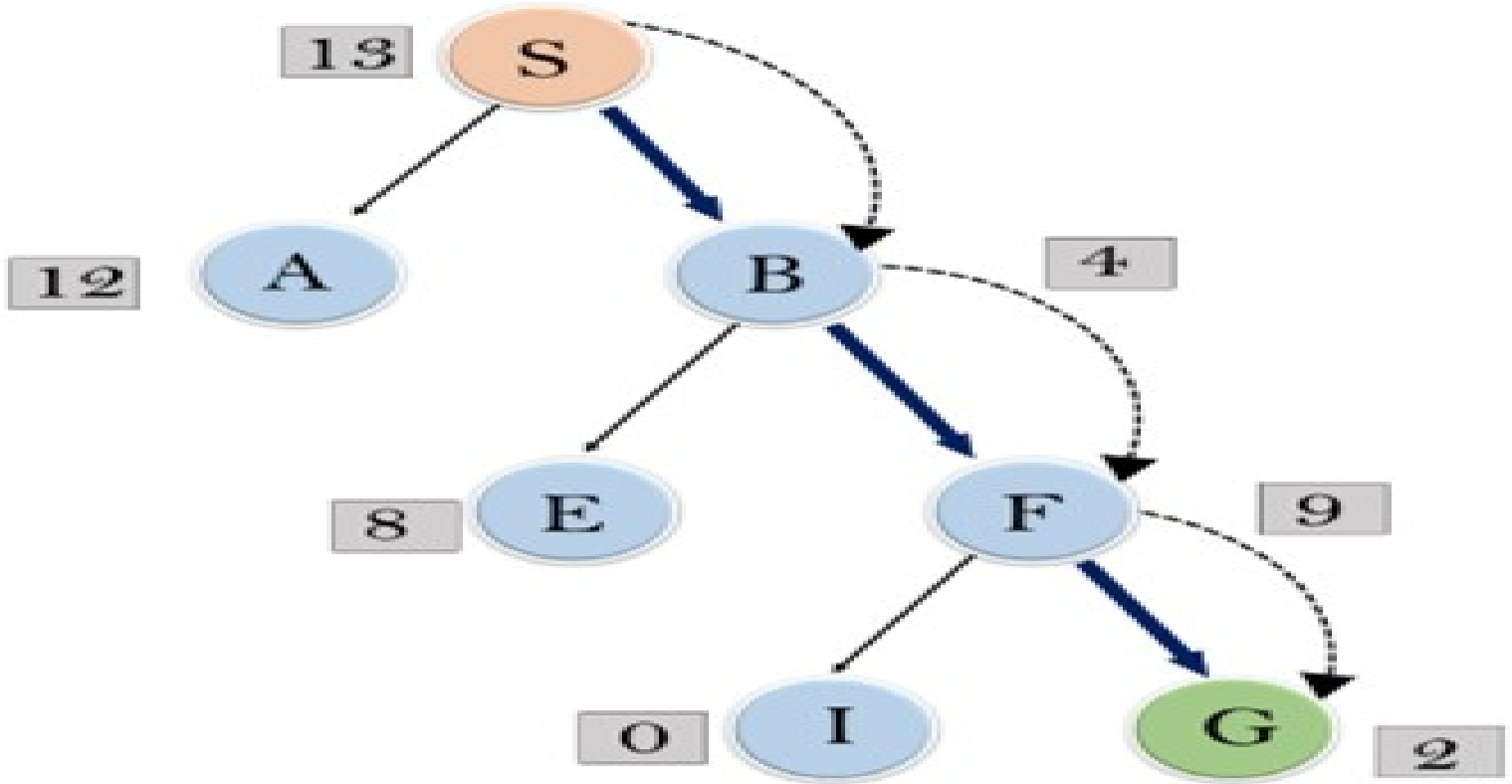
Example: Consider the below search problem, and you will traverse it using greedy best-first search. At each iteration, each node is expanded using evaluation function f(n)=h(n) which is given in the below table.

Cont..



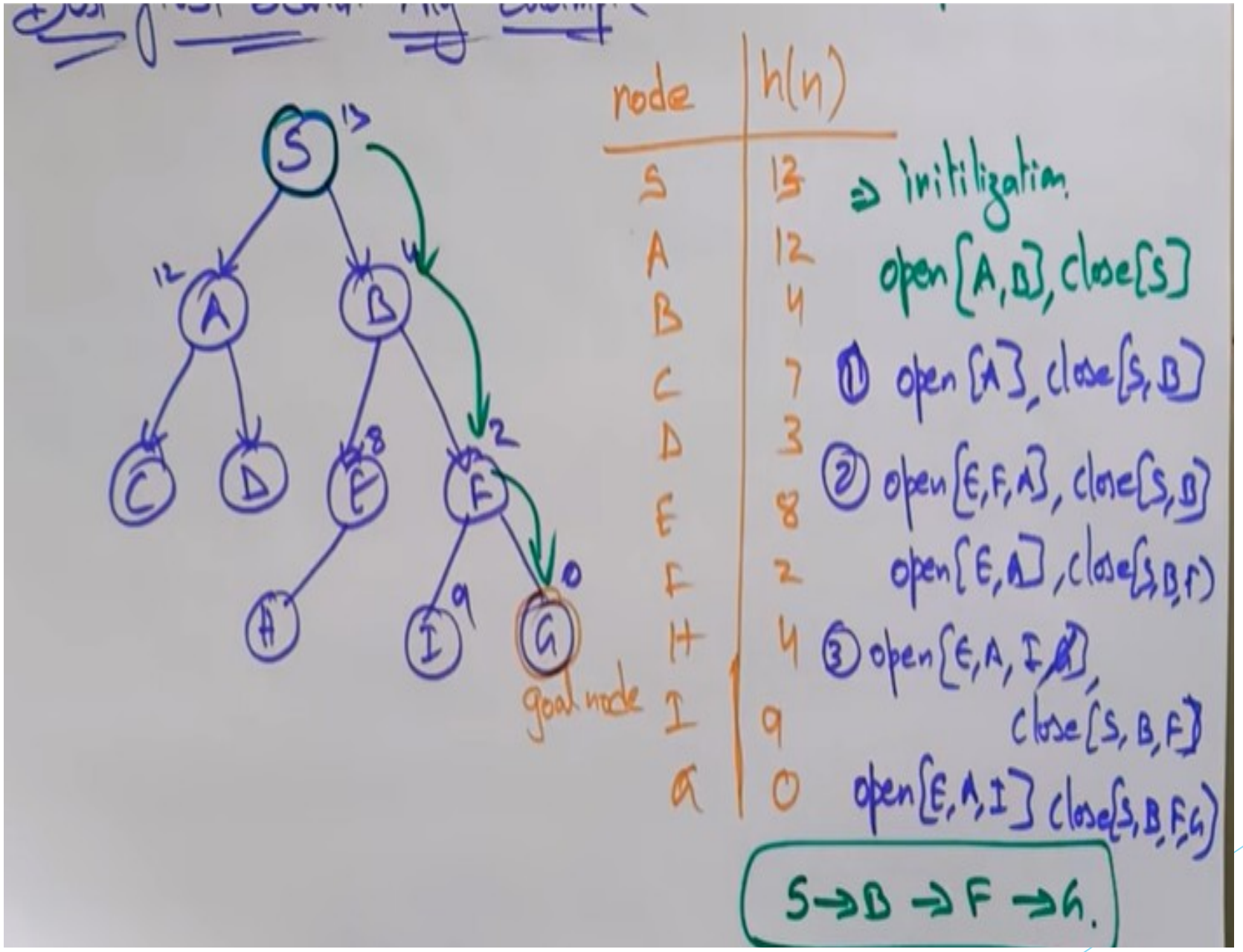
node	H (n)
A	12
B	4
C	7
D	3
E	8
F	2
H	4
I	9
S	13
G	0

- In this search example, we are using two lists which are **OPEN** and **CLOSED** Lists. Following are the iteration for traversing the above example.



Cont..

- **Expand the nodes of S and put in the CLOSED list**
- **Initialization:** Open [A, B], Closed [S]
- **Iteration 1:** Open [A], Closed [S, B]
- **Iteration 2:** Open [E, F, A], Closed [S, B]
  : Open [E, A], Closed [S, B, F]
- **Iteration 3:** Open [I, G, E, A], Closed [S, B, F]
  : Open [I, E, A], Closed [S, B, F, G]
- Hence the final solution path will be: **S----> B----->F----> G**
- **Time Complexity:** The worst case time complexity of Greedy best first search is $O(b^m)$.
- **Space Complexity:** The worst case space complexity of Greedy best first search is $O(b^m)$. Where, m is the maximum depth of the search space.
- **Complete:** Greedy best-first search is also incomplete, even if the given state space is finite.
- **Optimal:** Greedy best first search algorithm is not optimal.

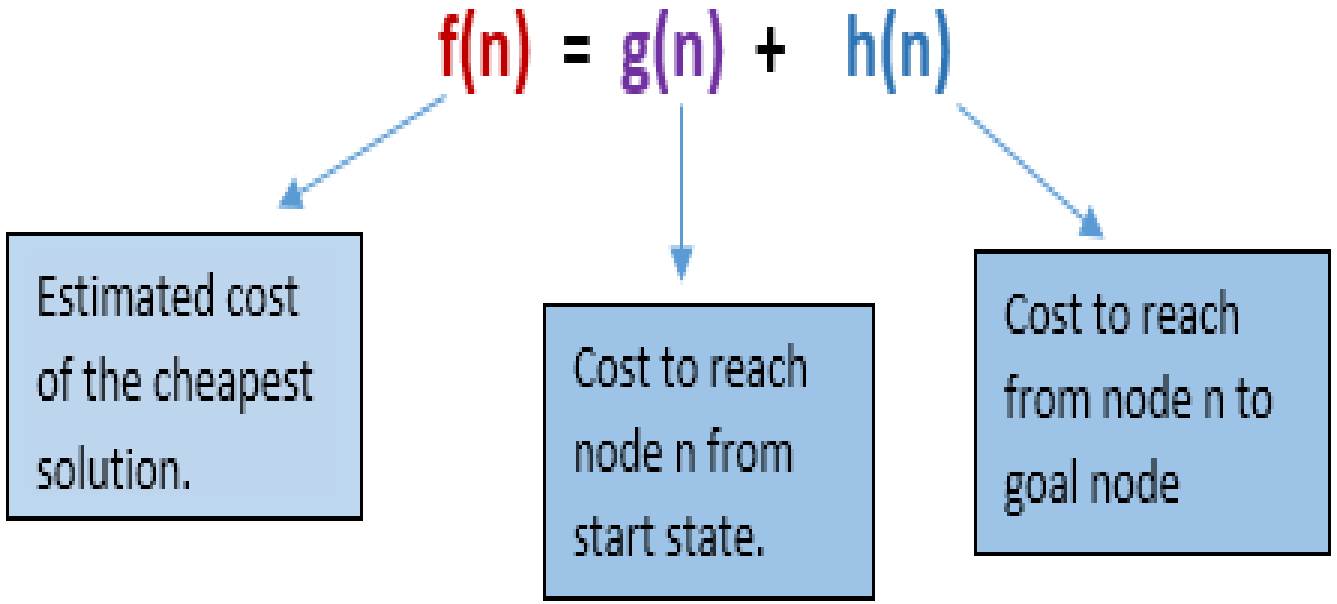

S
A
B
C
D
E
F
H
I
G
13
12
4
8
2
9
0
goal node
node
h(n)
S 13 ⇒ initilization
A 12 open [A, B], close [S]
B 4
C 7 ① open [A], close [S, B]
D 3
E 8 ② open [E, F, A], close [S, B]
F 2 open [E, A], close [S, B, F]
H 4 ③ open [E, A, I],
I 9 close [S, B, F]
G 0 open [E, A, I] close [S, B, F, G]
S → B → F → G.

A* Search Algorithm

- A* Search is the most commonly known form of best-first search.
- It uses heuristic function h(n), and cost to reach the node n from the start state g(n).
- It has combined features of UCS and greedy best-first search, by which it solve the problem efficiently.
- A* search algorithm finds the shortest path through the search space using the heuristic function.

Cont..

- This search algorithm expands less search tree and provides optimal result faster. A* algorithm is similar to UCS except that it uses g(n)+h(n) instead of g(n).
- In A* search algorithm, you use search heuristic as well as the cost to reach the node. Hence, you can combine both costs as following, and this sum is called as a **fitness number**.



- At each point in the search space, only those node is expanded which have the lowest value of f(n), and the algorithm terminates when the goal node is found.

Cont..

- Algorithm of A* search:
- **Step1:** Place the starting node in the OPEN list.
- **Step 2:** Check if the OPEN List is empty or not, if the list is empty then return failure and stops.
- **Step 3:** Select the node from the OPEN list which has the smallest value of evaluation function (g+h), if node n is goal node then return success and stop, otherwise

Cont..

- **Step 4:** Expand node n and generate all of its successors, and put n into the closed list. For each successor n', check whether n' is already in the OPEN or CLOSED list, if not then compute evaluation function for n' and place into Open list.
- **Step 5:** Else if node n' is already in OPEN and CLOSED, then it should be attached to the back pointer which reflects the lowest g(n') value.
- **Step 6:** Return to **Step 2**.

Cont..

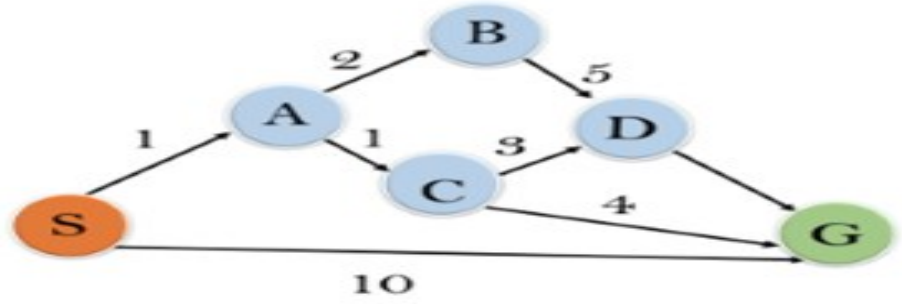
Advantages:

- A* search algorithm is the best algorithm than other search algorithms.
- A* search algorithm is optimal and complete.
- This algorithm can solve very complex problems.

Disadvantages:

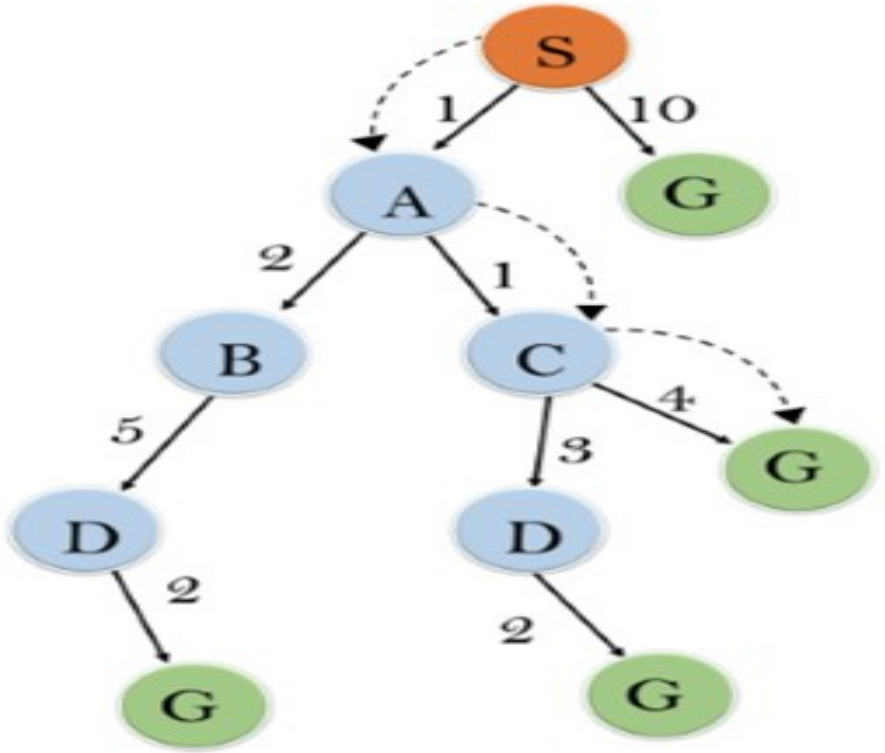
- It does not always produce the shortest path as it mostly based on heuristics and approximation.
- A* search algorithm has some complexity issues.
- The main drawback of A* is memory requirement as it keeps all generated nodes in the memory, so it is not practical for various large-scale problems.

Example: In this example, we will traverse the given graph using the A* algorithm. The heuristic value of all states is given in the below table so we will calculate the f(n) of each state using the formula f(n)= g(n) + h(n), where g(n) is the cost to reach any node from start state. Here we will use OPEN and CLOSED list.



State	h(n)
S	5
A	3
B	4
C	2
D	6
G	0

Solution



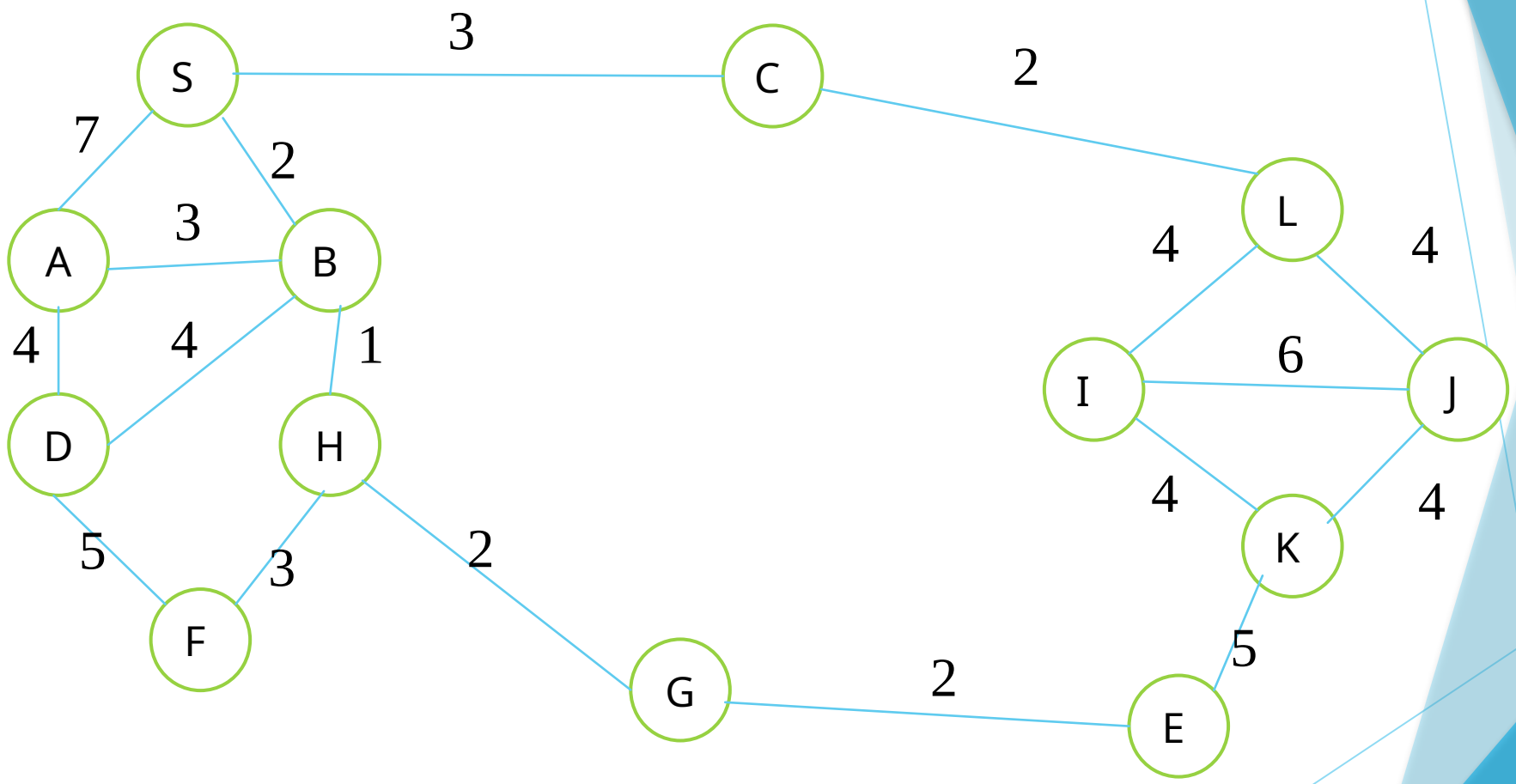
Cont..

- **Initialization:** {(S, 5)}
- **Iteration1:** {(S--> A, 4), (S-->G, 10)}
- **Iteration2:** {(S--> A-->C, 4), (S--> A-->B, 7), (S-->G, 10)}
- **Iteration3:** {(S--> A-->C--->G, 6), (S--> A-->C--->D, 11), (S--> A-->B, 7), (S-->G, 10)}
- **Iteration 4:** will give the final result, as **S--->A--->C--->G** it provides the optimal path with cost 6.

Points to Remember:

- A* algorithm returns the path which occurred first, and it does not search for all remaining paths.
- The efficiency of A* algorithm depends on the quality of heuristic.
- **Complete:** A* algorithm is complete as long as:
  - Branching factor is finite.
  - Cost at every action is fixed.

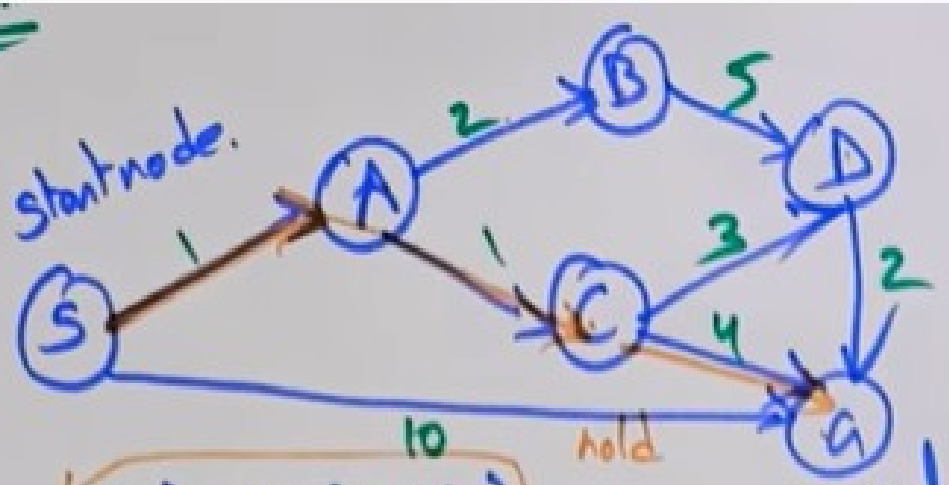
Exercise



A search

Example 1

State	h(n)
S	5
A	3
B	4
C	2
D	6
G	0



① $S \to A \Rightarrow f(n) = g(n) + h(n)$
$= 1 + 3 = 4$ ✓

$S \to G = f(n) = 10 + 0 = 10$ hold ✗

$S \to A \to C \to G$
Cost = 6

② $S \to A \to B \Rightarrow f(n) = 3 + 4 = 7$ hold ✗

$S \to A \to C \Rightarrow f(n) = 2 + 2 = 4$ ✓

③ $S \to A \to C \to D \Rightarrow f(n) = 5 + 6 = 11$ hold ✗

$S \to A \to C \to G \Rightarrow f(n) = 6 + 0 = 6$ ✓

Node	H(n)
S	10
A	9
B	7
C	8
D	8
E	0
F	6
G	3
H	6
I	4
J	4
K	3

- S=0+10=10, then expand S, then A,B,C
- A=9+7,B=2+7,C=3+8
- Prioritize them BCA
- Then expand the smallest value which is B
- D=6+8, H=3+6
- Expand H
- Ans= SBHGE

Cont..

- **Optimal:** A* search algorithm is optimal if it follows below two conditions:
- **Admissible:** the first condition requires for optimality is that h(n) should be an admissible heuristic for A* tree search. An admissible heuristic is optimistic in nature.
- **Consistency:** Second required condition is consistency for only A* graph-search.
- If the heuristic function is admissible, then A* tree search will always find the least cost path.
- **Time Complexity:** The time complexity of A* search algorithm depends on heuristic function, and the number of nodes expanded is exponential to the depth of solution d. So the time complexity is O(b^d), where b is the branching factor.
- **Space Complexity:** The space complexity of A* search algorithm is **O(b^d)**

Hill Climbing Algorithm in Artificial Intelligence

- Hill climbing algorithm is a local search algorithm which continuously moves in the direction of increasing elevation/value to find the peak of the mountain or best solution to the problem.
- It terminates when it reaches a peak value where no neighbor has a higher value.
- Hill climbing algorithm is a technique which is used for optimizing the mathematical problems.
- One of the widely discussed examples of Hill climbing algorithm is Traveling-salesman Problem in which we need to minimize the distance traveled by the salesman.

Cont..

- It is also called greedy local search as it only looks to its good immediate neighbor state and not beyond that.
- A node of hill climbing algorithm has two components which are state and value.
- Hill Climbing is mostly used when a good heuristic is available.
- In this algorithm, we don't need to maintain and handle the search tree or graph as it only keeps a single current state.

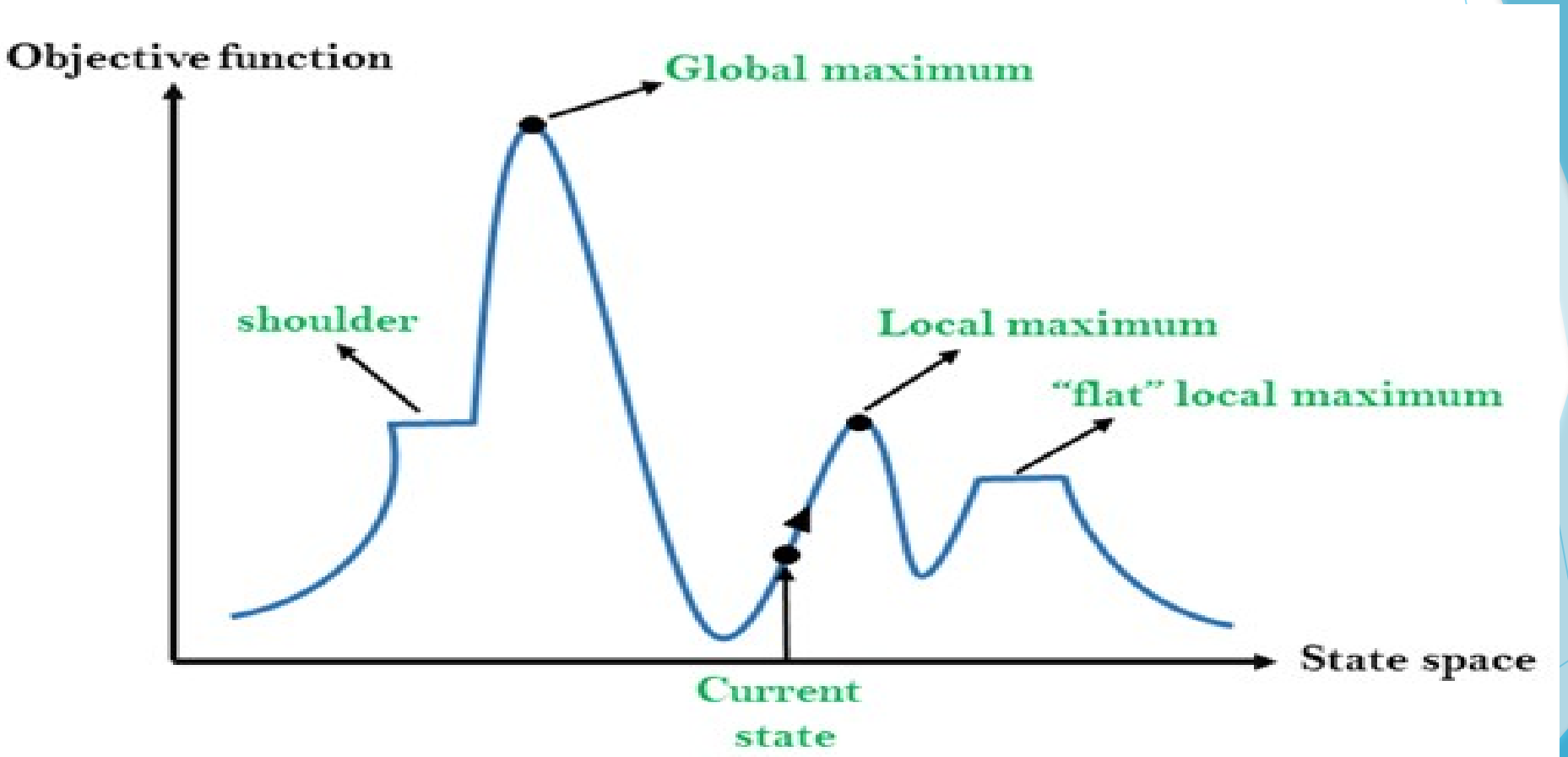
Features of Hill Climbing

- Following are some main features of Hill Climbing Algorithm:
- **Generate and Test variant:** Hill Climbing is the variant of Generate and Test method. The Generate and Test method produce feedback which helps to decide which direction to move in the search space.
- **Greedy Approach:** Hill-climbing algorithm search moves in the direction which optimizes the cost.
- **No backtracking:** It does not backtrack the search space, as it does not remember the previous states.

State-space Diagram for Hill Climbing

- The state-space landscape is a graphical representation of the hill-climbing algorithm which is showing a graph between various states of algorithm and Objective function/Cost.
- On Y-axis we have taken the function which can be an objective function or cost function, and state-space on the x-axis.
- If the function on Y-axis is cost then, the goal of search is to find the global minimum and local minimum.
- If the function of Y-axis is Objective function, then the goal of the search is to find the global maximum and local maximum.

Cont..


Objective function
Global maximum
shoulder
Local maximum
"flat" local maximum
State space
Current
state

Different regions in the state space landscape

- **Local Maximum:** Local maximum is a state which is better than its neighbor states, but there is also another state which is higher than it.
- **Global Maximum:** Global maximum is the best possible state of state space landscape. It has the highest value of objective function.
- **Current state:** It is a state in a landscape diagram where an agent is currently present.
- **Flat local maximum:** It is a flat space in the landscape where all the neighbor states of current states have the same value.
- **Shoulder:** It is a plateau region which has an uphill edge.

Types of Hill Climbing Algorithm

- **Simple hill Climbing:**
- **Steepest-Ascent hill-climbing:**
- **Stochastic hill Climbing:**

Simple Hill Climbing

- Simple hill climbing is the simplest way to implement a hill climbing algorithm. **It only evaluates the neighbor node state at a time and selects the first one which optimizes current cost and set it as a current state**. It only checks it's one successor state, and if it finds better than the current state, then move else be in the same state. This algorithm has the following features:
- Less time consuming
- Less optimal solution and the solution is not guaranteed
- Algorithm for Simple Hill Climbing:
- **Step 1:** Evaluate the initial state, if it is goal state then return success and Stop.
- **Step 2:** Loop Until a solution is found or there is no new operator left to apply.
- **Step 3:** Select and apply an operator to the current state.
- **Step 4:** Check new state:
  - If it is goal state, then return success and quit.
  - Else if it is better than the current state then assign new state as a current state.
  - Else if not better than the current state, then return to step2.
- **Step 5:** Exit.

Steepest-Ascent hill climbing

The steepest-Ascent algorithm is a variation of simple hill climbing algorithm. This algorithm examines all the neighboring nodes of the current state and selects one neighbor node which is closest to the goal state. This algorithm consumes more time as it searches for multiple neighbors

- Algorithm for Steepest-Ascent hill climbing:
- **Step 1:** Evaluate the initial state, if it is goal state then return success and stop, else make current state as initial state.
- **Step 2:** Loop until a solution is found or the current state does not change.
  - Let SUCC be a state such that any successor of the current state will be better than it.
  - For each operator that applies to the current state:
    - Apply the new operator and generate a new state.
    - Evaluate the new state.
    - If it is goal state, then return it and quit, else compare it to the SUCC.
    - If it is better than SUCC, then set new state as SUCC.
    - If the SUCC is better than the current state, then set current state to SUCC.
- **Step 5:** Exit.

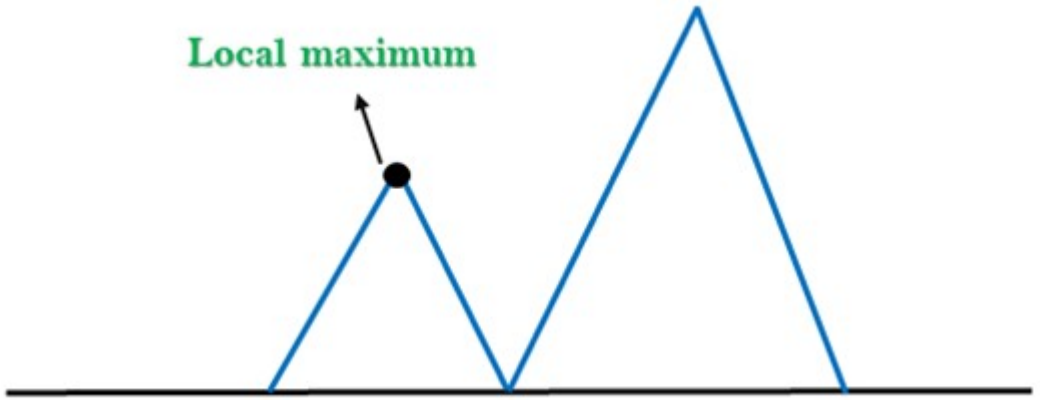
Stochastic hill climbing

- Stochastic hill climbing does not examine for all its neighbor before moving. Rather, this search algorithm selects one neighbor node at random and decides whether to choose it as a current state or examine another state.

Problems in Hill Climbing Algorithm

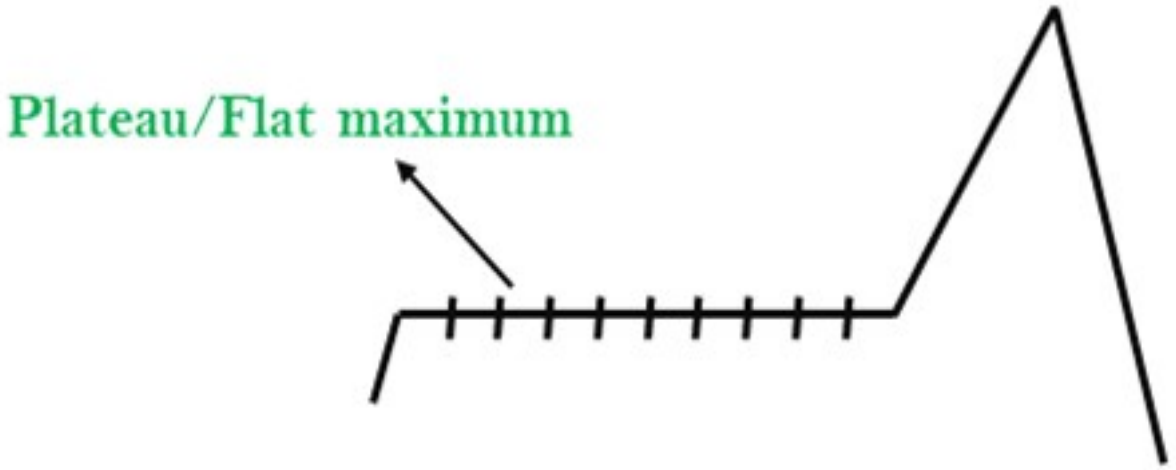
- **1. Local Maximum:** A local maximum is a peak state in the landscape which is better than each of its neighboring states, but there is another state also present which is higher than the local maximum.
- **Solution:** Backtracking technique can be a solution of the local maximum in state space landscape. Create a list of the promising path so that the algorithm can backtrack the search space and explore other paths as well.

Cont..



Cont..

- **2. Plateau:** A plateau is the flat area of the search space in which all the neighbor states of the current state contains the same value, because of this algorithm does not find any best direction to move. A hill-climbing search might be lost in the plateau area.
- **Solution:** The solution for the plateau is to take big steps or very little steps while searching, to solve the problem. Randomly select a state which is far away from the current state so it is possible that the algorithm could find non-plateau region.


Plateau/Flat maximum

Cont..

- **Ridges:** A ridge is a special form of the local maximum. It has an area which is higher than its surrounding areas, but itself has a slope, and cannot be reached in a single move.
- **Solution:** With the use of bidirectional search, or by moving in different directions, we can improve this problem.

Simulated Annealing

- A hill-climbing algorithm which never makes a move towards a lower value guaranteed to be incomplete because it can get stuck on a local maximum.
- And if algorithm applies a random walk, by moving a successor, then it may complete but not efficient.
- **Simulated Annealing** is an algorithm which yields both efficiency and completeness.

Cont..

- In mechanical term **Annealing** is a process of hardening a metal or glass to a high temperature then cooling gradually, so this allows the metal to reach a low-energy crystalline state.
- The same process is used in simulated annealing in which the algorithm picks a random move, instead of picking the best move.
- If the random move improves the state, then it follows the same path.
- Otherwise, the algorithm follows the path which has a probability of less than 1 or it moves downhill and chooses another path.

Means-Ends Analysis in Artificial Intelligence

- We have studied the strategies which can reason either in forward or backward, but a mixture of the two directions is appropriate for solving a complex and large problem.
- Such a mixed strategy, make it possible that first to solve the major part of a problem and then go back and solve the small problems arise during combining the big parts of the problem.
- Such a technique is called **Means-Ends Analysis**.

Cont..

- Means-Ends Analysis is problem-solving techniques used in Artificial intelligence for limiting search in AI programs.
- It is a mixture of Backward and forward search technique.
- The MEA technique was first introduced in 1961 by Allen Newell, and Herbert A. Simon in their problem-solving computer program, which was named as General Problem Solver (GPS).
- The MEA analysis process centered on the evaluation of the difference between the current state and goal state.

How means-ends analysis Works

- The means-ends analysis process can be applied recursively for a problem. It is a strategy to control search in problem-solving. Following are the main Steps which describes the working of MEA technique for solving a problem.
- First, evaluate the difference between Initial State and final State.
- Select the various operators which can be applied for each difference.
- Apply the operator at each difference, which reduces the difference between the current state and goal state.

Operator Subgoaling

- In the MEA process, we detect the differences between the current state and goal state.
- Once these differences occur, then we can apply an operator to reduce the differences.
- But sometimes it is possible that an operator cannot be applied to the current state.
- So we create the subproblem of the current state, in which operator can be applied, such type of backward chaining in which operators are selected, and then sub goals are set up to establish the preconditions of the operator is called **Operator Subgoaling**.

Algorithm for Means-Ends Analysis

- Let's we take Current state as CURRENT and Goal State as GOAL, then following are the steps for the MEA algorithm.
- **Step 1:** Compare CURRENT to GOAL, if there are no differences between both then return Success and Exit.
- **Step 2:** Else, select the most significant difference and reduce it by doing the following steps until the success or failure occurs.
  - Select a new operator O which is applicable for the current difference, and if there is no such operator, then signal failure.
  - Attempt to apply operator O to CURRENT. Make a description of two states.
    i) O-Start, a state in which O?s preconditions are satisfied.
    ii) O-Result, the state that would result if O were applied In O-start.
  - If
    (First-Part <------ MEA (CURRENT, O-START)
    And
    (LAST-Part <----- MEA (O-Result, GOAL), are successful, then signal Success and return the result of combining FIRST-PART, O, and LAST-PART.
- The above-discussed algorithm is more suitable for a simple problem and not adequate for solving complex problems.

Adversarial Search

- Adversarial search is a search, where we examine the problem which arises when we try to plan ahead of the world and other agents are planning against us.
- In previous topics, we have studied the search strategies which are only associated with a single agent that aims to find the solution which often expressed in the form of a sequence of actions.
- But, there might be some situations where more than one agent is searching for the solution in the same search space, and this situation usually occurs in game playing.

Cont..

- The environment with more than one agent is termed as **multi-agent environment**, in which each agent is an opponent of other agent and playing against each other. Each agent needs to consider the action of other agent and effect of that action on their performance.
- So, **Searches in which two or more players with conflicting goals are trying to explore the same search space for the solution, are called adversarial searches, often known as Games**.
- Games are modeled as a Search problem and heuristic evaluation function, and these are the two main factors which help to model and solve games in AI.

Types of Games in AI

Deterministic	Chance Moves	
Perfect information	Chess, Checkers, go, Othello	Backgammon, monopoly
Imperfect information	Battleships, blind, tic-tac-toe	Bridge, poker, scrabble, nuclear war

Cont..

- **Perfect information:** A game with the perfect information is that in which agents can look into the complete board.
- Agents have all the information about the game, and they can see each other moves also. Examples are Chess, Checkers, Go, etc.
- **Imperfect information:** If in a game agents do not have all information about the game and not aware with what's going on, such type of games are called the game with imperfect information, such as tic-tac-toe, Battleship, blind, Bridge, etc.

Cont..

- **Deterministic games:** Deterministic games are those games which follow a strict pattern and set of rules for the games, and there is no randomness associated with them. Examples are chess, Checkers, Go, tic-tac-toe, etc.
- **Non-deterministic games:** Non-deterministic are those games which have various unpredictable events and has a factor of chance or luck.
- This factor of chance or luck is introduced by either dice or cards. These are random, and each action response is not fixed. Such games are also called as stochastic games. Example: Backgammon, Monopoly, Poker, etc.

Zero-Sum Game

- Zero-sum games are adversarial search which involves pure competition.
- In Zero-sum game each agent's gain or loss of utility is exactly balanced by the losses or gains of utility of another agent.
- One player of the game try to maximize one single value, while other player tries to minimize it.
- Each move by one player in the game is called as ply.
- Chess and tic-tac-toe are examples of a Zero-sum game.

Zero-sum game: Embedded thinking

- The Zero-sum game involved embedded thinking in which one agent or player is trying to figure out:
- What to do.
- How to decide the move
- Needs to think about his opponent as well
- The opponent also thinks what to do
- Each of the players is trying to find out the response of his opponent to their actions. This requires embedded thinking or backward reasoning to solve the game problems in AI.

Formalization of the problem

- **A game can be defined as a type of search in AI which can be formalized of the following elements:**
- **Initial state:** It specifies how the game is set up at the start.
- **Player(s):** It specifies which player has moved in the state space.
- **Action(s):** It returns the set of legal moves in state space.
- **Result(s, a):** It is the transition model, which specifies the result of moves in the state space.

Cont..

- **Terminal-Test(s):** Terminal test is true if the game is over, else it is false at any case. The state where the game ends is called terminal states.
- **Utility(s, p):** A utility function gives the final numeric value for a game that ends in terminal states s for player p. It is also called payoff function.
- For Chess, the outcomes are a win, loss, or draw and its payoff values are +1, 0, ½. And for tic-tac-toe, utility values are +1, -1, and 0.

END