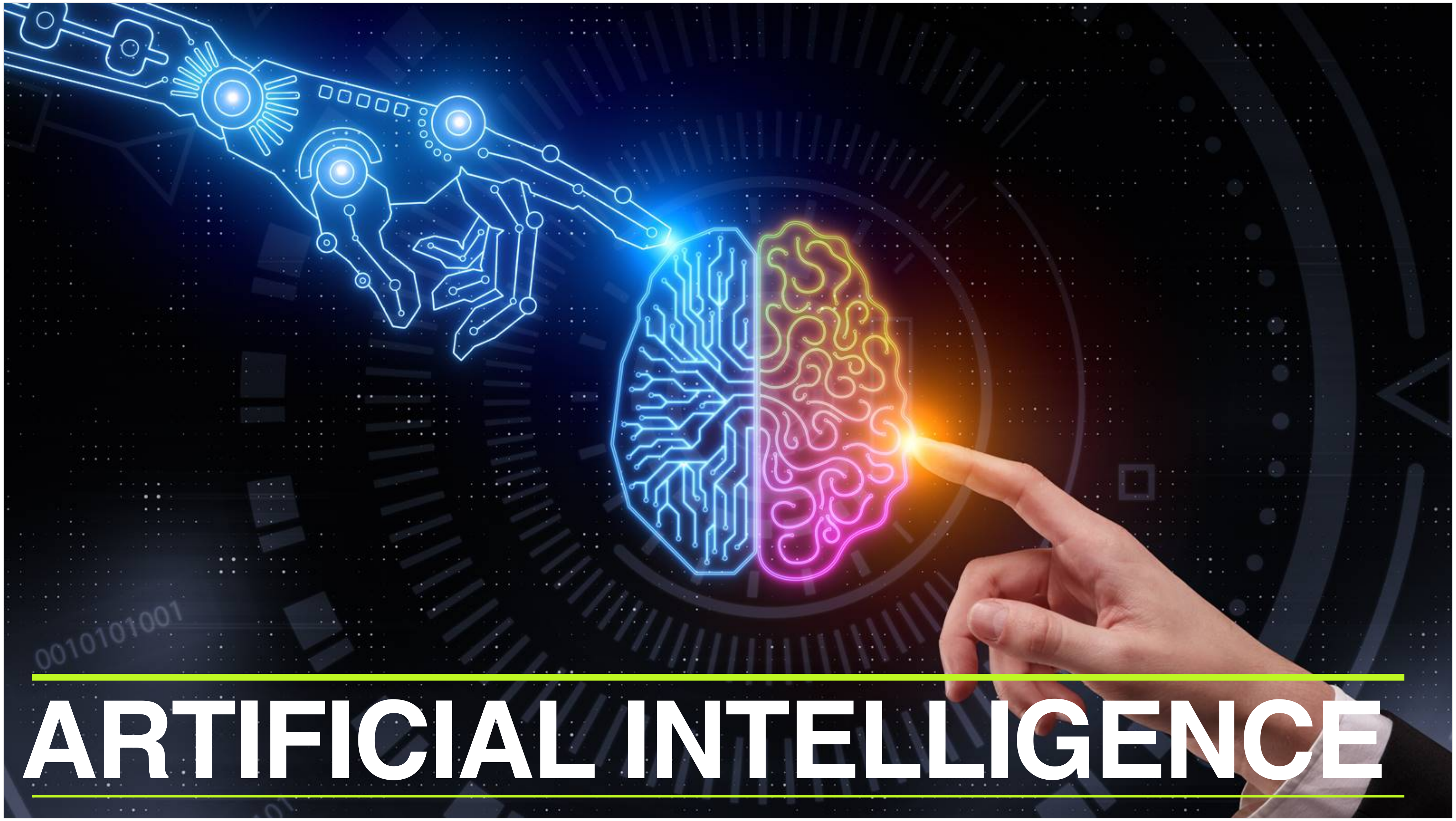


ARTIFICIAL INTELLIGENCE

Problem Solving

- **Define the problem precisely** by including specification of initial situation, and final situation constituting the solution of the problem.
- **Analyse the problem** to find a few important features for appropriateness of the solution technique.
- **Isolate and represent the knowledge** that is necessary for the solution
- **Select the best** problem solving technique.

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:
    1. **Search Space:** Search space represents a set of possible solutions, which a system may have.
    2. **Start State:** It is a state from where agent begins the search.
    3. **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.
- **Actions:** It gives the description of all the available actions to the agent.
- **Transition model:** A description of what each action do, can be represented as a transition model.
- **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 :

- **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.
- **Time Complexity:** 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.

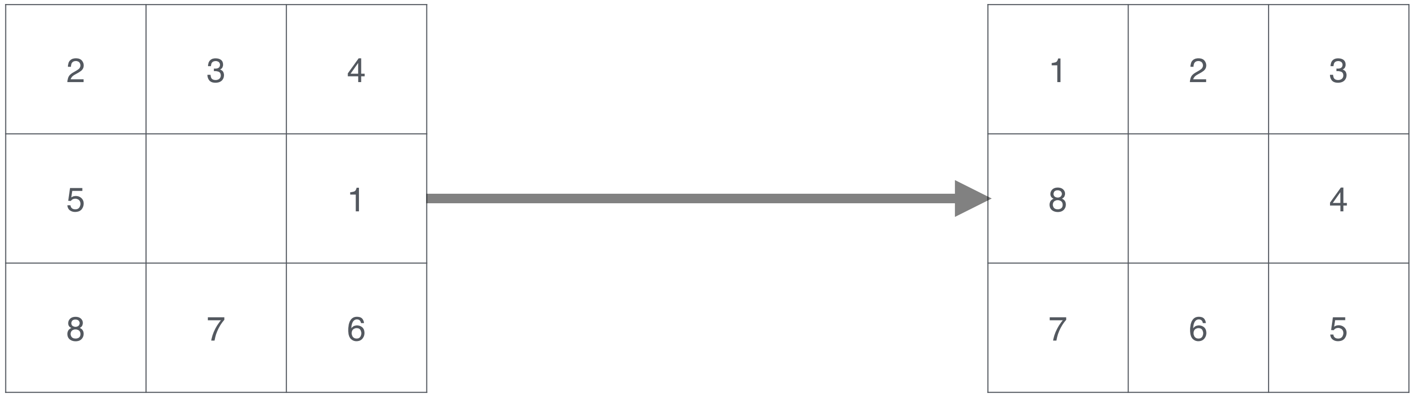
State Space

- Number of state in which problem can go.

$$\boxed{S : \{S, A, Action(s), Result(s,a), Cost(s,a)\}}$$

- S = no. of states (start, intermediate, goal)
- A = all possible set of actions

Example : 8 puzzle problem



2	3	4
5		1
8	7	6

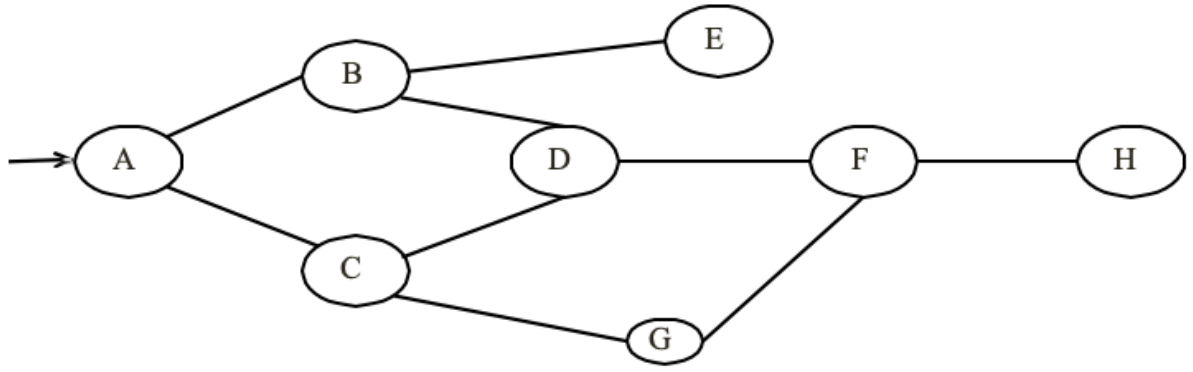
Start State

1	2	3
8		4
7	6	5

Goal State

Uninformed Search

- Do not have additional information about the states beyond problem definition.
- The plans to reach the goal state from the start state differ only by the order and/or length of actions.
- Also called blind search or Brute-force.
- Each of these algorithms will have:
    - A problem **graph,** containing the start node s and the goal nod G.
    - A **strategy,** describing the manner in which the graph will be traversed to get to G.
    - A **fringe,** which is a data structure used to store all the possible states (nodes) that you can go from the current states.
    - A **tree,** that results while traversing to the goal node.
    - A solution **plan,** which the sequence of nodes from S to G.



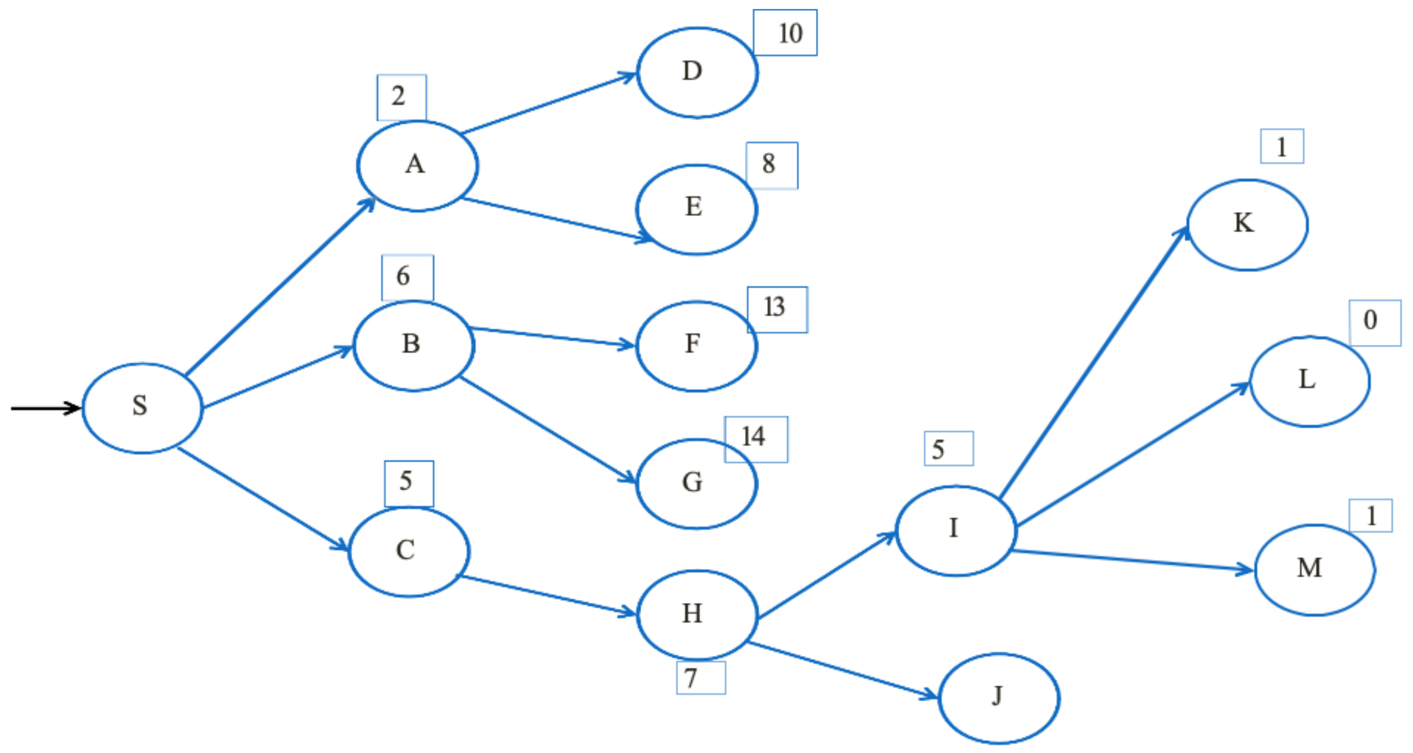
State Space without any extra information associated with each state

Informed Search

- Some info about problem space(heuristic) is used to compute preference among the children for exploration and expansion.
- Heuristic function : maps each state to a numerical value which depicts goodness of a node.

$$\boxed{H(n)=value}$$

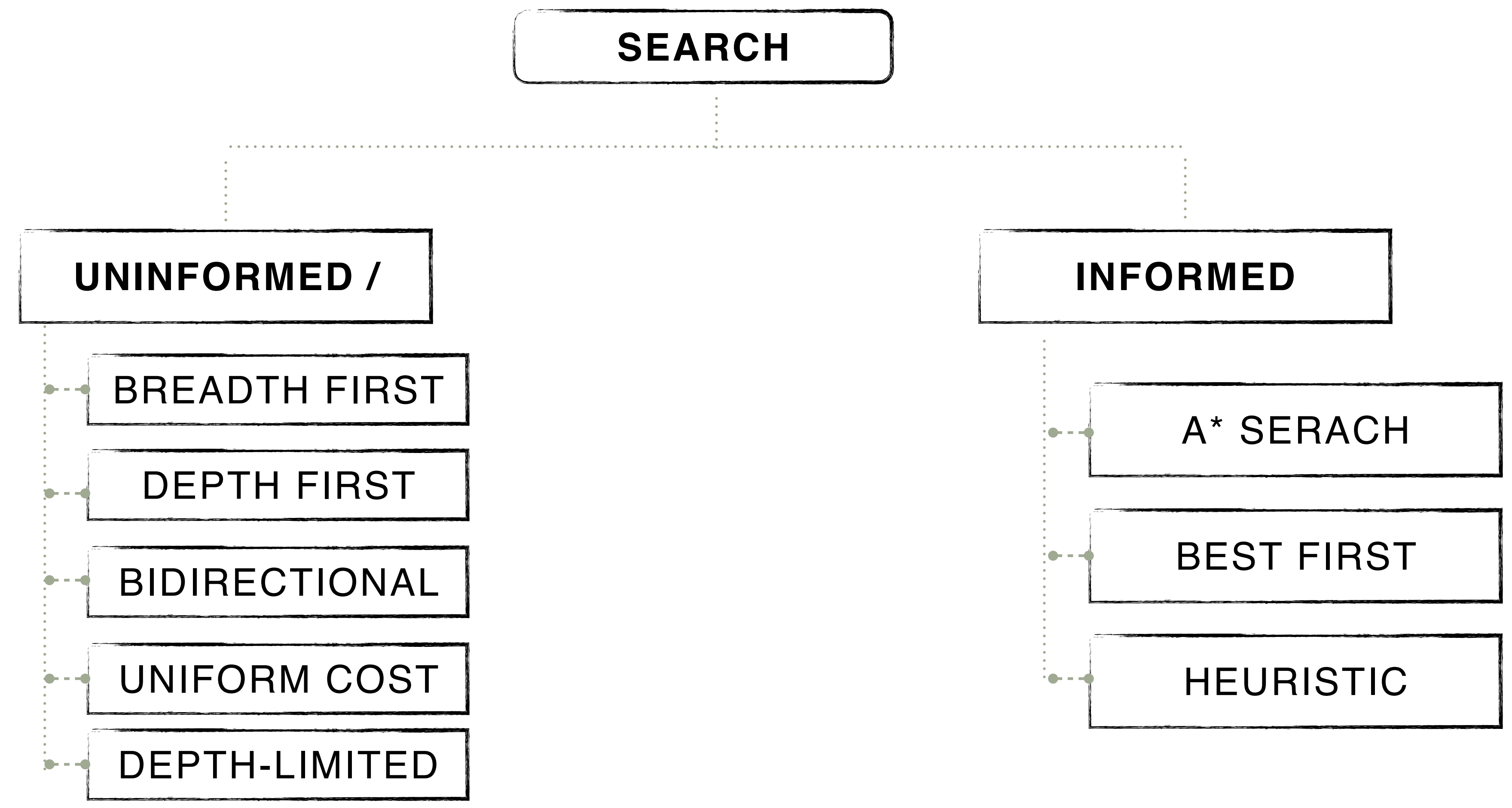
H() — heuristic function , n — current state



Example of the state space with heuristic values associated with each state

Uniformed vs Informed Searching

	Uninformed (Blind / Brute-force)	Informed
1	Search without information	Search with information
2	No knowledge	Use knowledge to find solution
3	Time consuming	Quick solution
4	More complexity (time,space)	Less complexity(time,space)
5	DFS, BFS etc	A* , Heuristic


SEARCH
UNINFORMED /
BREADTH FIRST
DEPTH FIRST
BIDIRECTIONAL
UNIFORM COST
DEPTH-LIMITED
INFORMED
A* SERACH
BEST FIRST
HEURISTIC

Water Jug Problem

- **Problem:** There are two jugs of **volume A litre** and **B litre.** Neither has any **measuring mark** on it.There is a pump that can be used to fill the jugs with water.How can you get exactly **x litre** of water into the **A litre jug.**Assuming that we have unlimited supply of water.

Note:Let's assume we have A=4 litre and B= 3 litre jugs. And we want exactly 2 Litre water into jug A (i.e 4 litre jug) how we will do this.

- **Solution:**

The state space for this problem can be described as the set of ordered **pairs of integers (x,y)**

Where, **x** represents the quantity of water in the **4-gallon** jug and **y** represents the quantity of water in **3-gallon** jug.

- **Start State**: (0,0) and **Goal State:** (2,0)
- We basically perform three operations to achieve the goal :

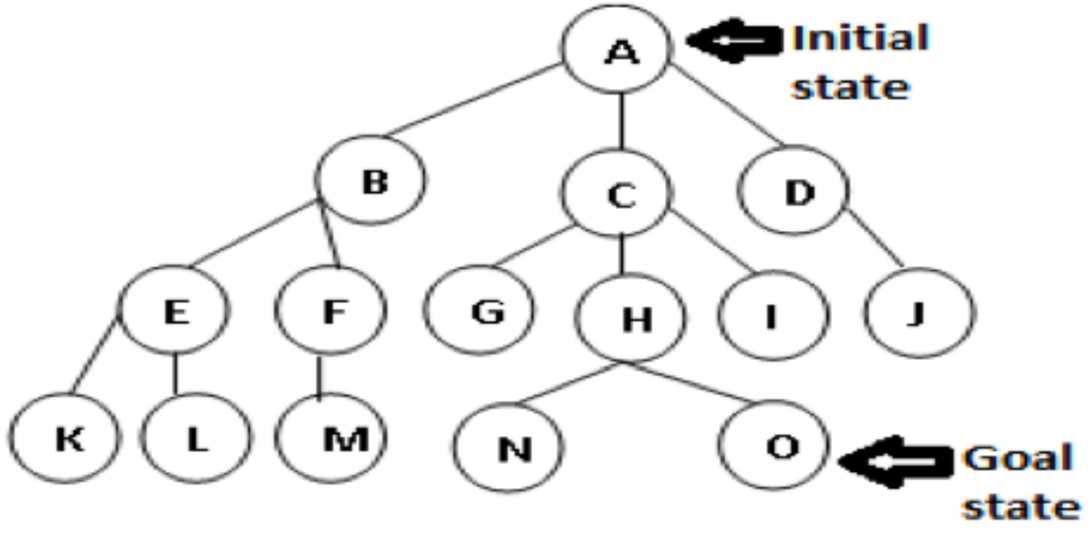
1. Fill water jug.
2. Empty water jug
3. Transfer water jug

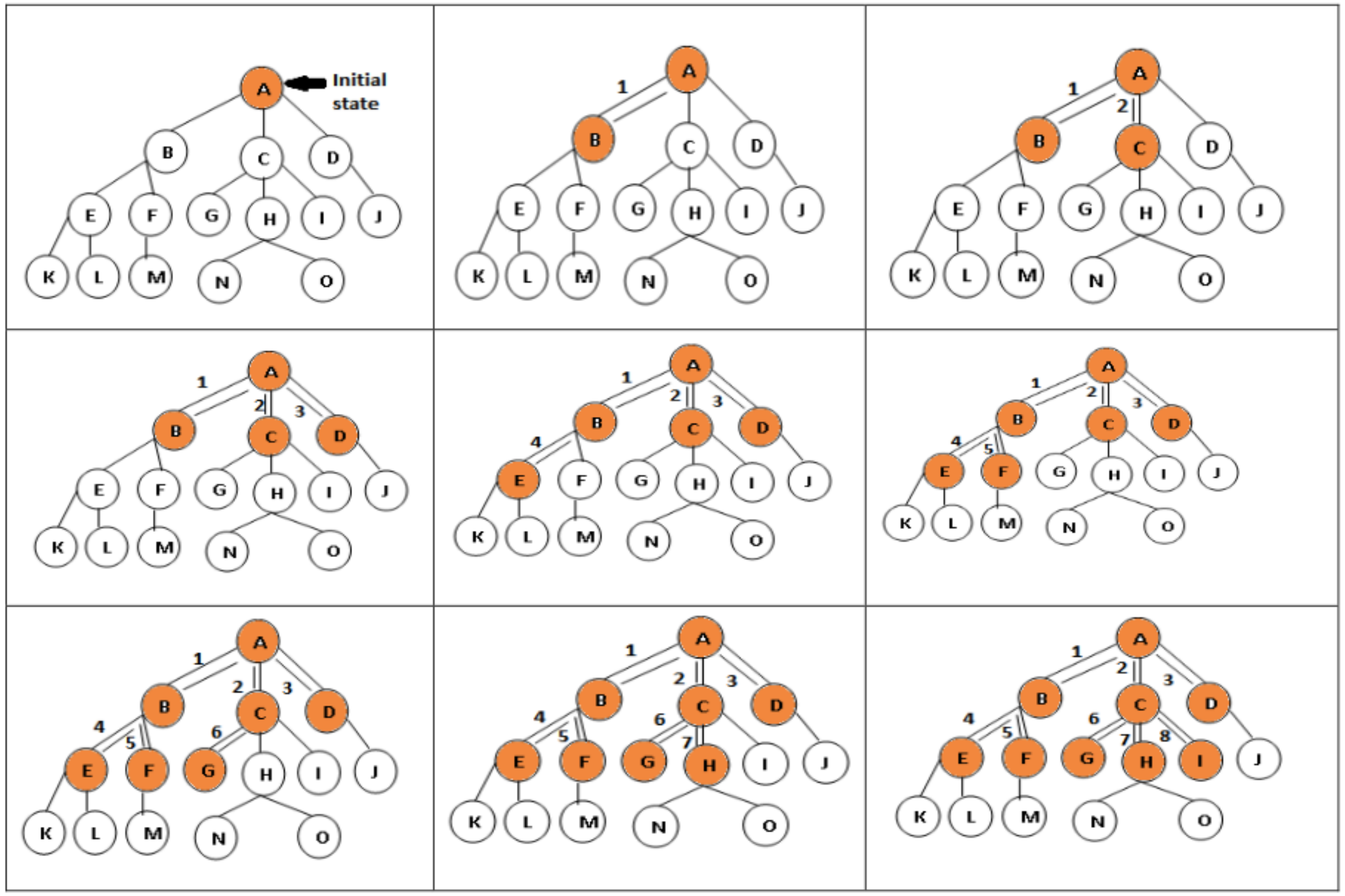
Rule	State	Process
1	(X,Y | X<4)	(4,Y) {Fill 4-gallon jug}
2	(X,Y |Y<3)	(X,3) {Fill 3-gallon jug}
3	(X,Y |X>0)	(0,Y) {Empty 4-gallon jug}
4	(X,Y | Y>0)	(X,0) {Empty 3-gallon jug}
5	(X,Y | X+Y>=4 ^ Y>0)	(4,Y-(4-X)) {Pour water from 3-gallon jug into 4-gallon jug until 4-gallon jug is full}
6	(X,Y | X+Y>=3 ^X>0)	(X-(3-Y),3) {Pour water from 4-gallon jug into 3-gallon jug until 3-gallon jug is full}
7	(X,Y | X+Y<=4 ^Y>0)	(X+Y,0) {Pour all water from 3-gallon jug into 4-gallon jug}
8	(X,Y | X+Y <=3^ X>0)	(0,X+Y) {Pour all water from 4-gallon jug into 3-gallon jug}
9	(0,2)	(2,0) {Pour 2 gallon water from 3 gallon jug into 4 gallon jug}

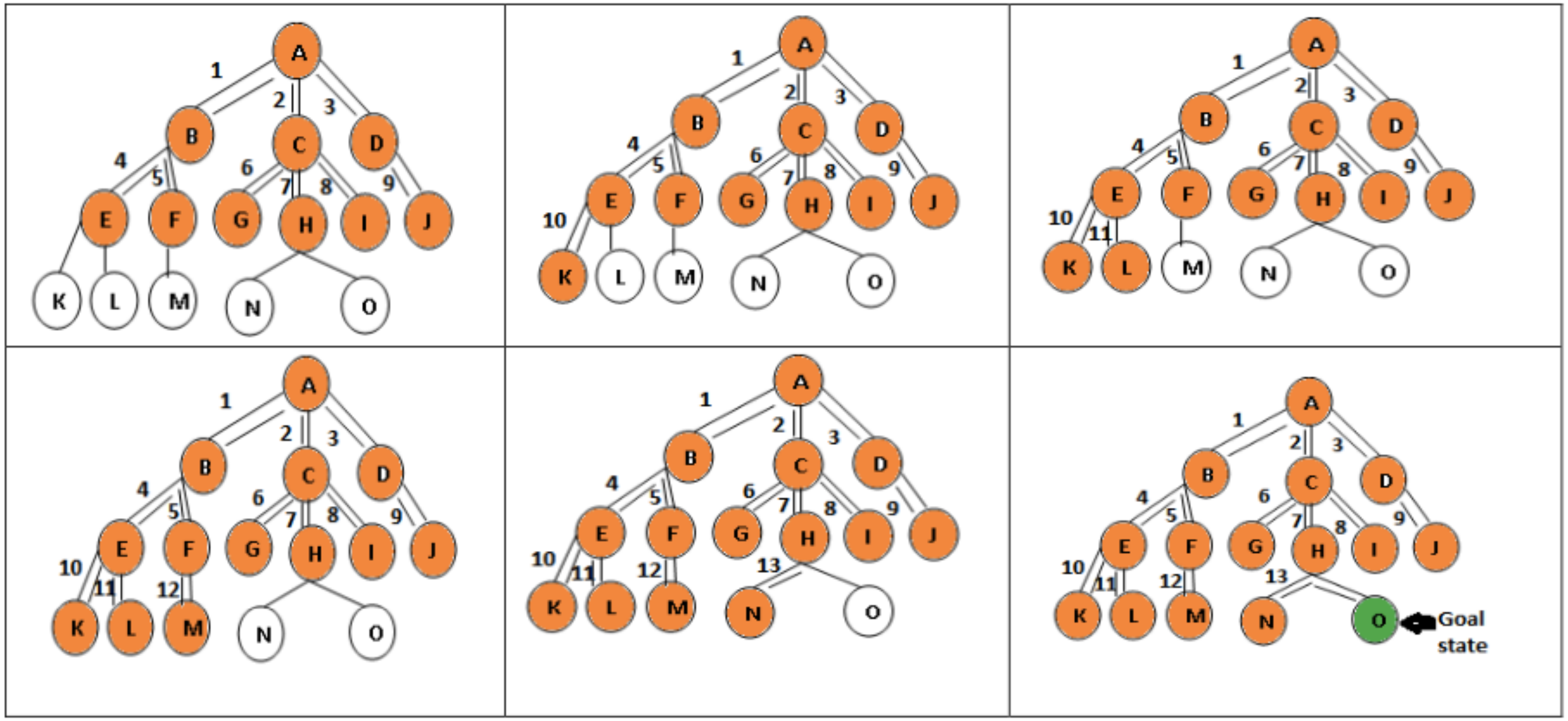
Breadth First Search

- 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 node 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.
- Breadth-first search implemented using FIFO queue data structure.
- This method provides shortest path to the solution.

Example




Initial state


Goal
state

Open list (Unexplored nodes)	Close list (Visited nodes)
A	A
B,C,D	B
C,D,E,F	C
D,E,F,G,H,I	D
E,F,G,H,I,J	E
F,G,H,I,J,K,L	F
G,H,I,J,K,L,M	G
H,I,J,K,L,M	H
I,J,K,L,M,N,O	I
J,K,L,M,N,O	J
K,L,M,N,O	K
L,M,N,O	L
M,N,O	M
N,O	N
O ← Goal state	-

- If **branching factor** (average number of child nodes for a given node) = b and depth = d, then number of nodes at level d = b^d
- **Time Complexity:** Time Complexity of BFS algorithm can be obtained by the number of nodes traversed in BFS until the shallowest Node. Where the d= depth of shallowest solution and b is a node at every state. **$T(b) = 1+b^2+b^3+.......+ b^d= O(b^d)$**
- **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.

Rules :

1. A queue (FIFO-First in First Out) data structure is used by BFS.
2. You mark any node in the graph as root and start traversing the data from it.
3. BFS traverses all the nodes in the graph and keeps dropping them as completed.
4. BFS visits an adjacent unvisited node, marks it as done, and inserts it into a queue.Removes the previous vertex from the queue in case no adjacent vertex is found.
5. BFS algorithm iterates until all the vertices in the graph are successfully traversed and marked as completed.
6. There are no loops caused by BFS during the traversing of data from any node.

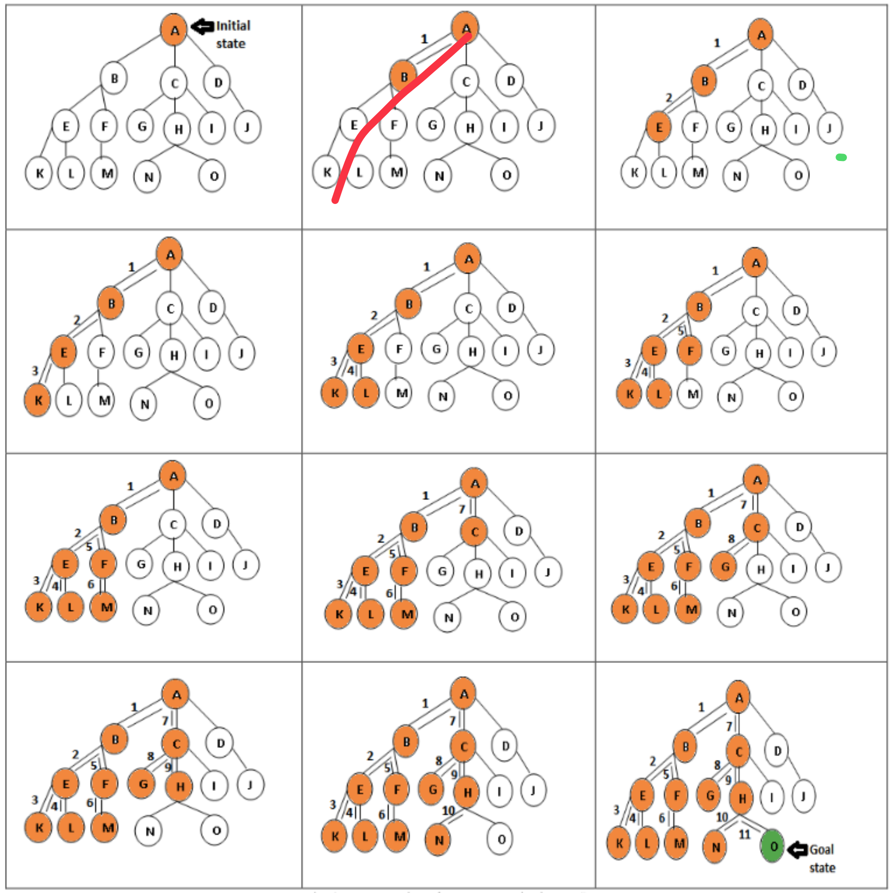
Applications:

- **Un-weighted Graphs**: BFS algorithm can easily create the shortest path and a minimum spanning tree to visit all the vertices of the graph in the shortest time possible with high accuracy.
- **P2P Networks**: BFS can be implemented to locate all the nearest or neighbouring nodes in a peer to peer network. This will find the required data faster.
- **Web Crawlers**: Search engines or web crawlers can easily build multiple levels of indexes by employing BFS. BFS implementation starts from the source, which is the web page, and then it visits all the links from that source.
- **Navigation Systems:** BFS can help find all the neighboring locations from the main or source location.
- **Network Broadcasting**: A broadcasted packet is guided by the BFS algorithm to find and reach all the nodes it has the address for.

- **Advantages :**
    - 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.
- **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.

Depth First Search

- Depth-first search isa 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.



Open list (Unexplored nodes)	Close list (Visited nodes)
A	A
B,C,D	B
E,F,C,D	E
K,L,F,C,D	K
L,F,C,D	L
F,C,D	F
M,C,D	M
C,D	C
G,H,I,D	G
H,I,D	H
N,O,I,D	N
O,I,D ↑ Goal state	-

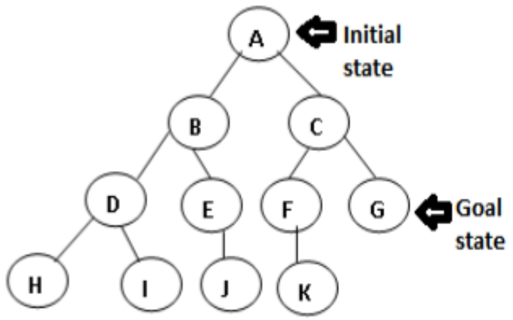
Properties :

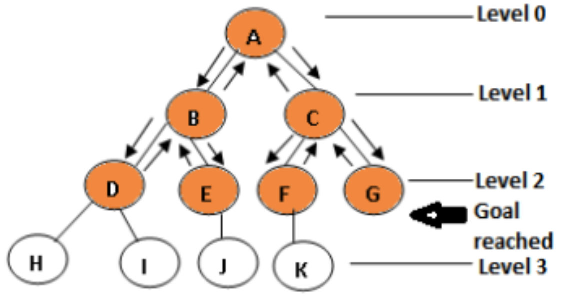
1. Complete : No
2. Optimal : No
3. Time Complexity: $O(b^m)$; where *b* is branching factor, and *m* is maximum depth of the tree
4. Space Complexity: $O(bm)$

- **Advantage:**
    - 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).
- **Disadvantage:**
    - 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.

Depth Limited Search

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





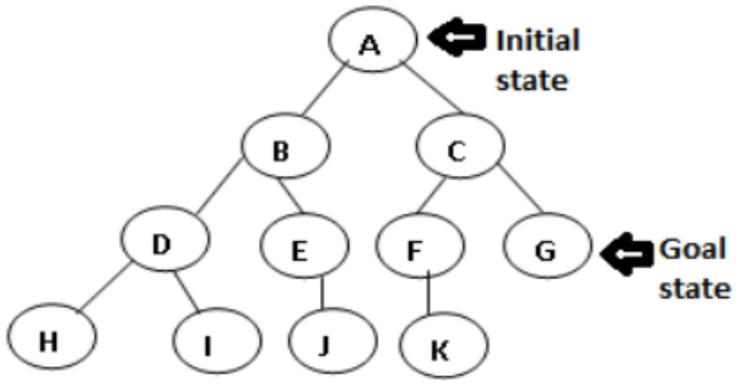
Open list (Unexplored nodes)	Close list (Visited nodes)
Depth bound **(*l* = 2)**	
Open=[A]	Close=[ ]
Open=[B,C]	Close=[A]
Open=[D,E,C]	Close=[A,B]
Open=[E,C]	Close=[A,B,D]
Open=[C]	Close=[A,B,D,E]
Open=[F,G]	Close=[A,B,D,E,C]
Open=[G]	Close=[A,B,D,E,C,F]
Open=[ ]	Close=[A,B,D,E,C,F,G]

Properties :

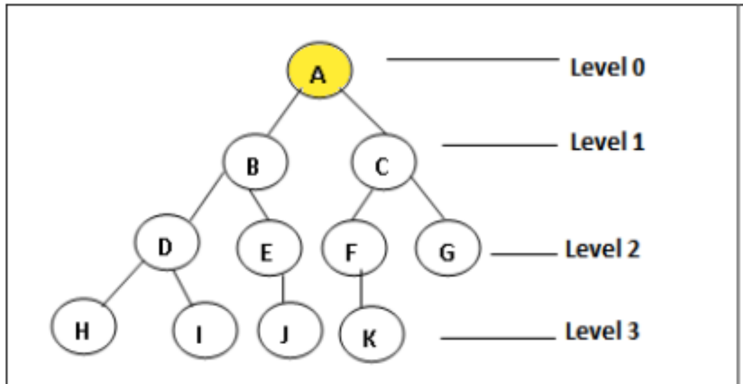
1. Complete : No
2. Optimal : No
3. Time Complexity: $O(b^l)$; where b is branching factor, and l is depth limit
4. Space Complexity: $O(bl)$; where d is depth of the solution.

- **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.

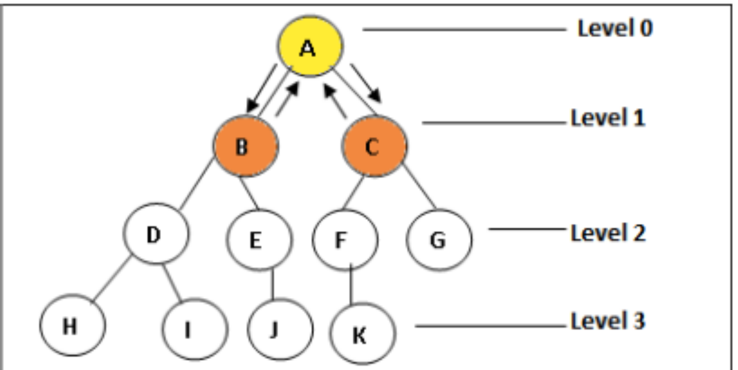
Iterative Deepening Search



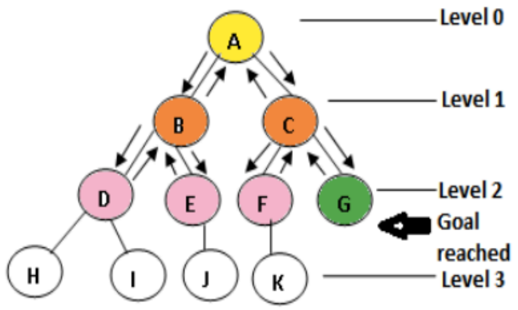
- This search method is also known as iterative deepening depth-first search.
- The method is a combination of two search techniques: the breadth-first search and the depth-first search. The limit of the depth is changed iteratively to best fit the problem definition and the goal sought.



IDS: Depth bound 0 (Level – 0)



IDS: Depth bound 1 (Level – 1)



IDS: Depth bound 0 (Level – 0)

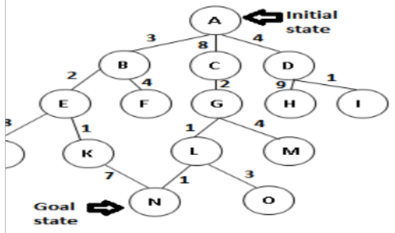
Depth (Level)	Iterative Deepening Search
Level – 0	A
Level – 1	A,B,C
Level – 2	A,B,C,D,E,F,G

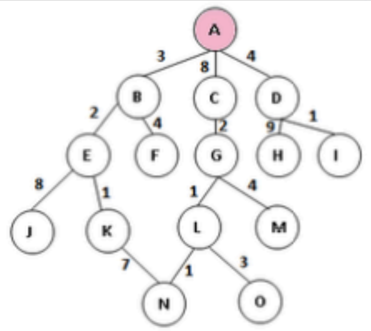
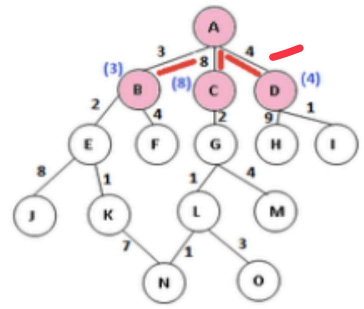
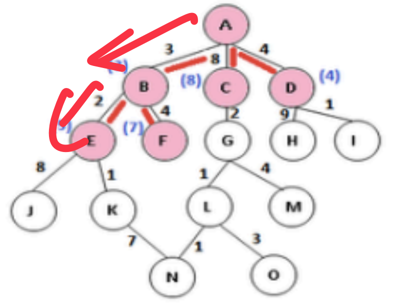
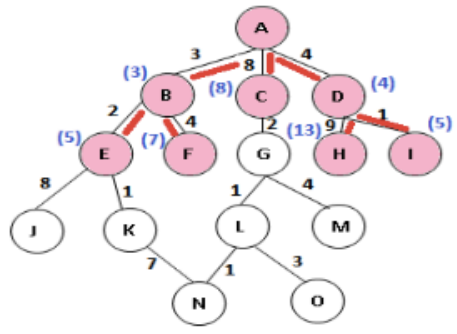
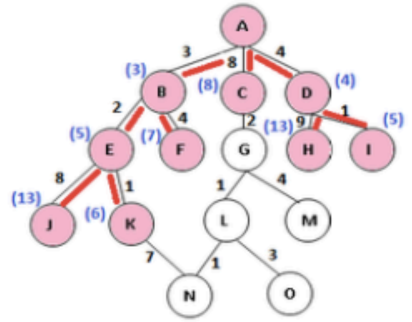
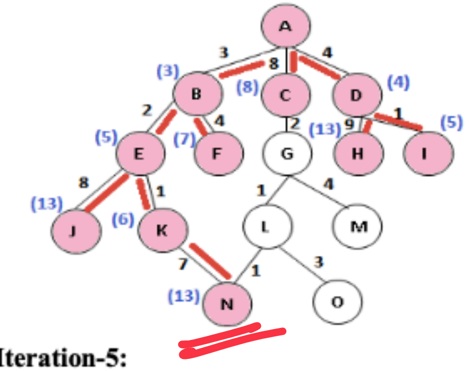
Properties :

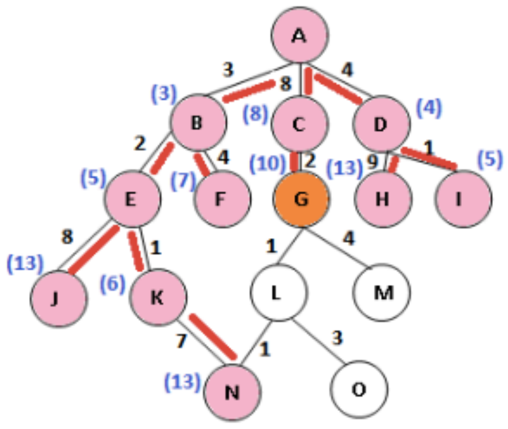
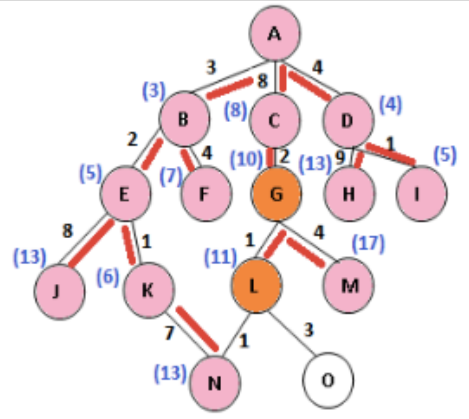
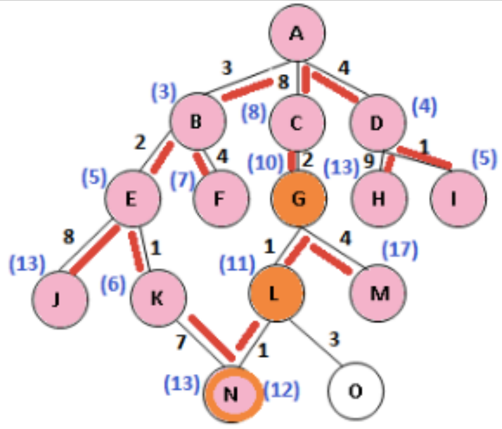
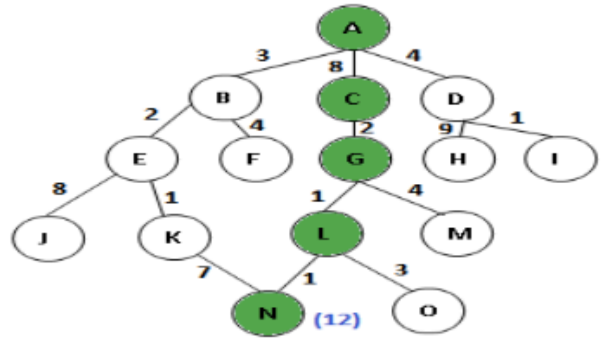
- Complete : Yes
- Optimal : yes
- Time Complexity: $O(b^d)$; where *b* is branching factor and *d* is depth of the solution
- Space Complexity: $O(bd)$

Uniform Cost Search

- Uniform-cost search 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 form 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. 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.



 Initial State: **A**	 **Iteration-1:** {A→B,3},{A→C,8},{A→D,4}	 **Iteration-2:** {A→B→E,5},{A→B→F,7}
 **Iteration-3:** {A→D→H,13},{A→D→I,5}	 **Iteration-4:** {A→B→E→J,13}, {A→B→E→K,6}	 **Iteration-5:** {A→B→E→K→N,13} (Goal state reached with cost 13)

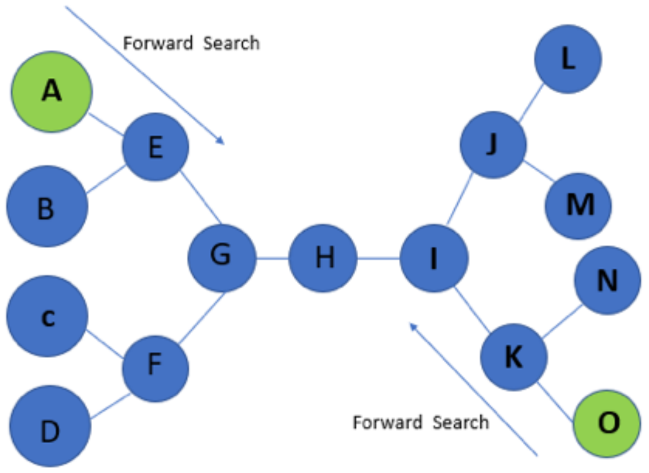
 **Iteration-6:** {A→C→G,10}	 **Iteration-7:** {A→C→G→L,11}, {A→C→G→M,17}	 **Iteration-8:** {A→C→G→L→N,12} (Goal state reached with cost 12)
 Minimum cost from A to N is **{A→C→G→L→N, 12}** .		

Properties :

- Complete : Yes
- Optimal : yes
- Time Complexity: O(b^m) ; where b is branching factor and m is maximum depth of the tree;
- Space Complexity: O(b^d) ; where d is depth of the solution.

- **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.

Bidirectional Search



- In this method the search has a running technique in two directions. Indeed, two searches run simultaneously. One of them is done in a forward direction from the initial state, and another one is done backward from the goal state. The aim is that these two searches should join in the midway.

Properties :

1. Complete: Yes (may be or may be not)

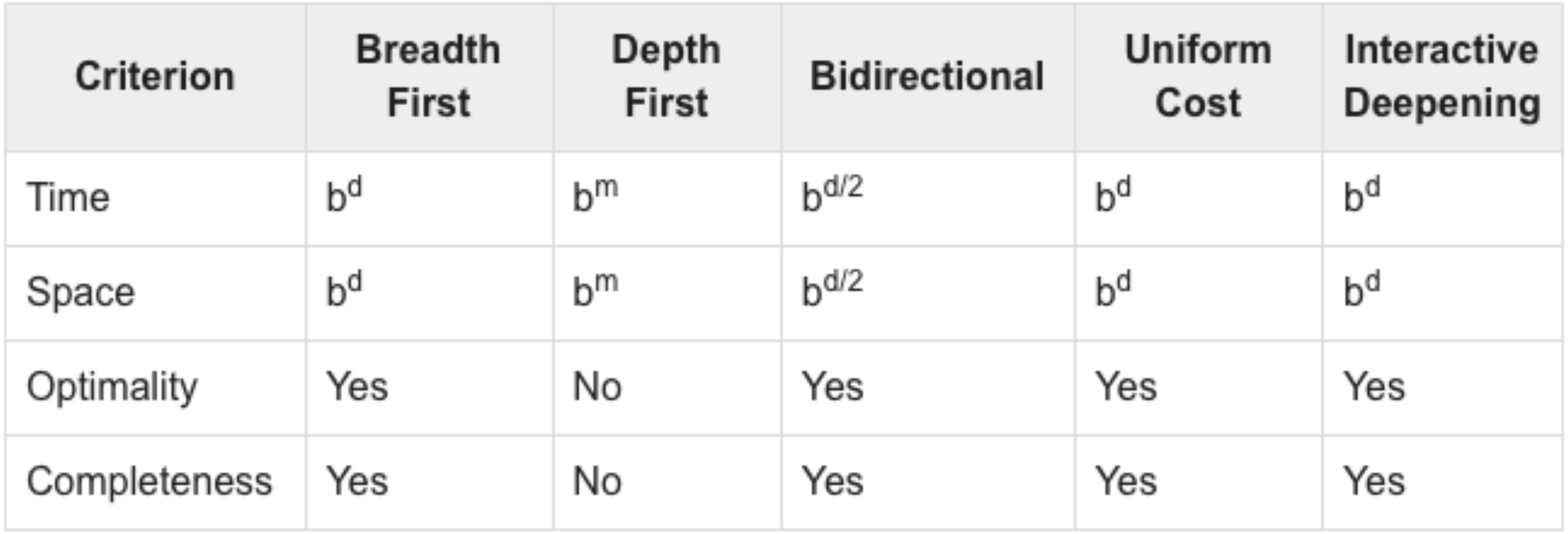
2. Optimal: Yes

3. Time Complexity: $b^{d/2}$; where b is branching factor and d is depth of the solution

4. Space Complexity: $b^{d/2}$.

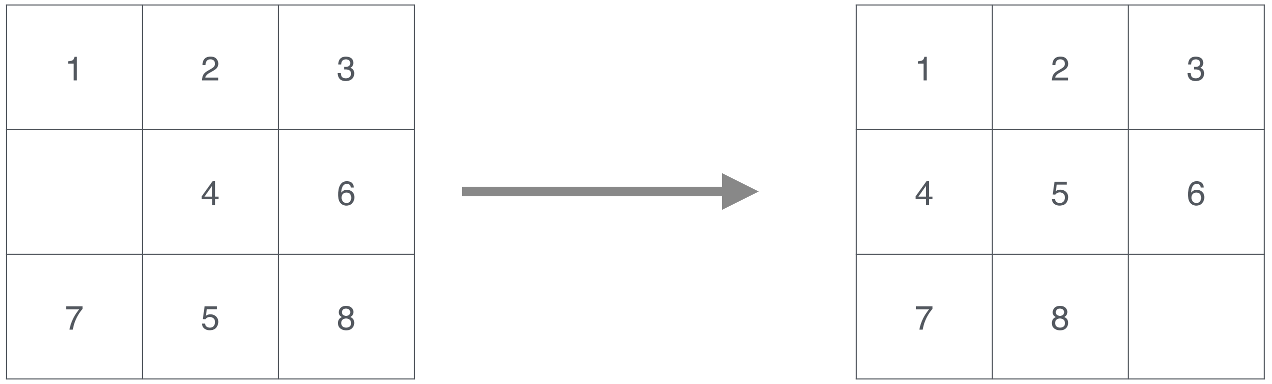
- For example, for the branching factor $_b$= 8 and depth $_d$= 4, the breadth-first search needs time which is proportional to 8^4= 4096 whereas the bidirectional search is more effective because of time which is proportional to $2*8^2$= 128 to reach the goal.

- **Advantages :**
    - One of the main advantages of bidirectional searches is the speed at which we get the desired results.
    - It drastically reduces the time taken by the search by having simultaneous searches.
    - It also saves resources for users as it requires less memory capacity to store all the searches.
- **Disadvantages** :
    - The fundamental issue with bidirectional search is that the user should be aware of the goal state to use bidirectional search and thereby to decrease its use cases drastically.
    - The implementation is another challenge as additional code and instructions are needed to implement this algorithm, and also care has to be taken as each node and step to implement such searches.
    - The algorithm must be robust enough to understand the intersection when the search should come to an end or else there’s a possibility of an infinite loop.
    - It is also not possible to search backwards through all states.



Criterion	Breadth First	Depth First	Bidirectional	Uniform Cost	Interactive Deepening
Time	b^d	b^m	$b^{d/2}$	b^d	b^d
Space	b^d	b^m	$b^{d/2}$	b^d	b^d
Optimality	Yes	No	Yes	Yes	Yes
Completeness	Yes	No	Yes	Yes	Yes

8 Puzzle Problem

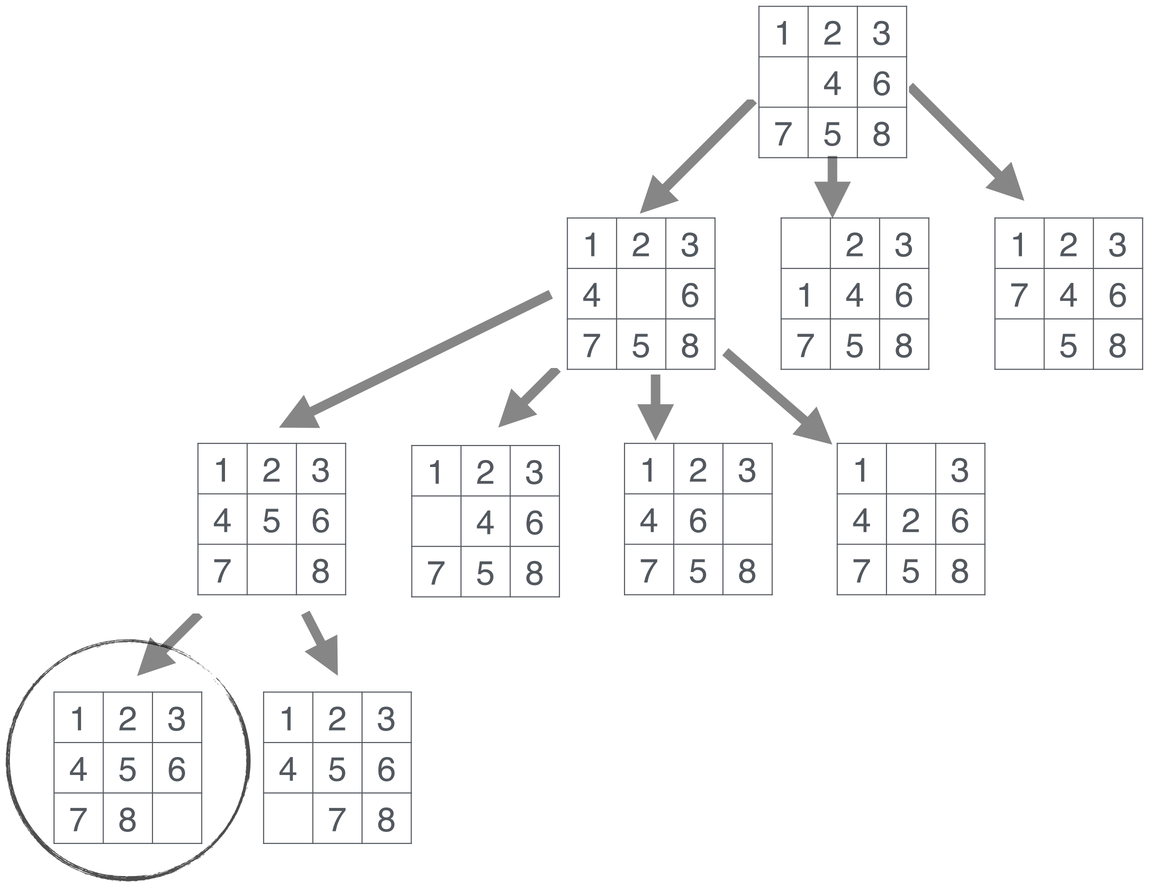


1	2	3
	4	6
7	5	8

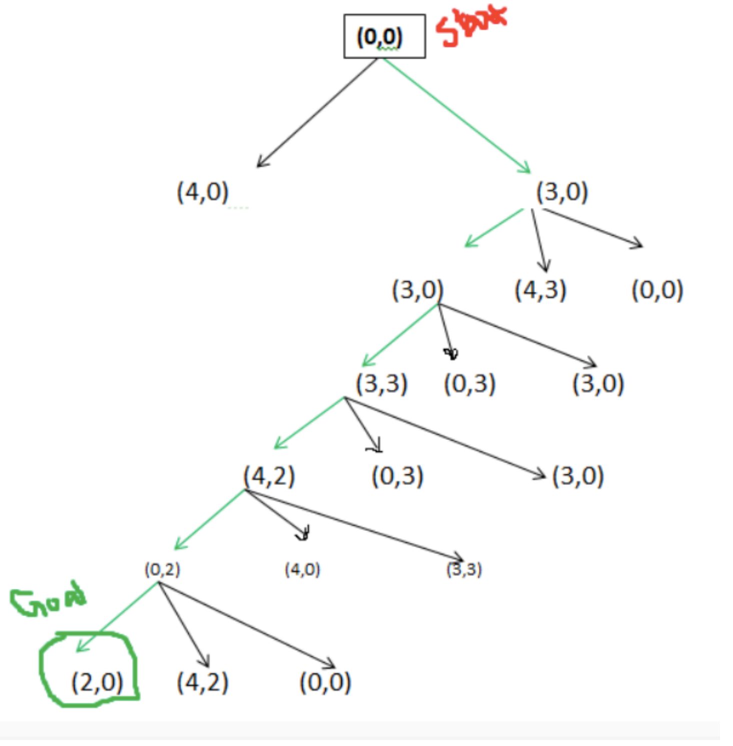
Start state

1	2	3
4	5	6
7	8	

Goal State


1 2 3
4 6
7 5 8

Water Jug problem using DFS



(0,0)
Start
(4,0)
(3,0)
(3,0)
(4,3)
(0,0)
(3,3)
(0,3)
(3,0)
(4,2)
(0,3)
(3,0)
(0,2)
(4,0)
(3,3)
Goal
(2,0)
(4,2)
(0,0)