Blind Search of Problems:

1st: Depth First Search (DFS):

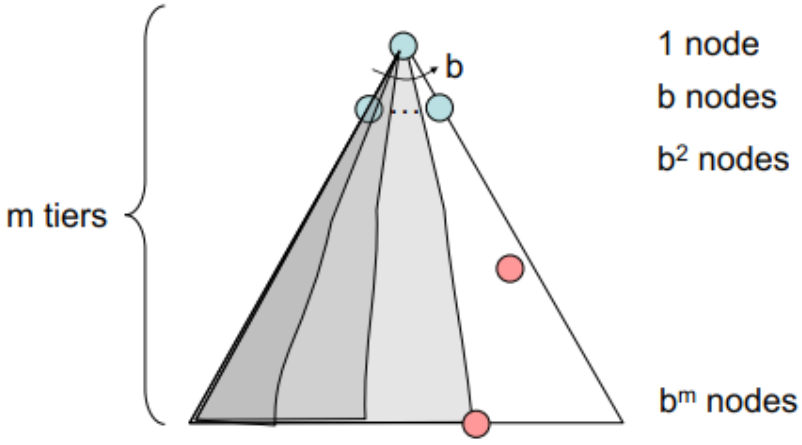
DFS is an algorithm that dives as deep as possible down one path of a search tree before looking anywhere else. It prioritizes expanding the deepest node in its current path.

- Begins at the root node. Travels down the tree (usually taking the left path first) until it hits a "leaf" node (a dead end with no children).
- **Backtrack:** If the deepest isn't the goal, it drops that node and "backs up" to the nearest previous node that still has unexplored branches.
- **Repeat:** It repeats this deep-dive and backtrack process until the goal is found or the whole tree is checked.

Key Features

- **Blind Search:** It searches without a map or guide. It just blindly explores and checks if the current node is the target.
- **Stack Data Structure:** It operates on a **Last-In, First-Out (LIFO)** principle, just like a stack.

Instead of going down forever, you can set a depth limit (e.g., Level 2). If the search hits Level 2 without finding the goal, it will stop going deeper and immediately backtrack to check other paths within that limit. This is called **Depth-Limited Search (DLS).**



DFS Algorithm(1):

Step No. 1:- Form a list consisting of the root node.

Step No. 2:- Until the path is empty or the goal is found, determine if the first element in the list is a goal.

a- If it is a goal, then exit with solution.

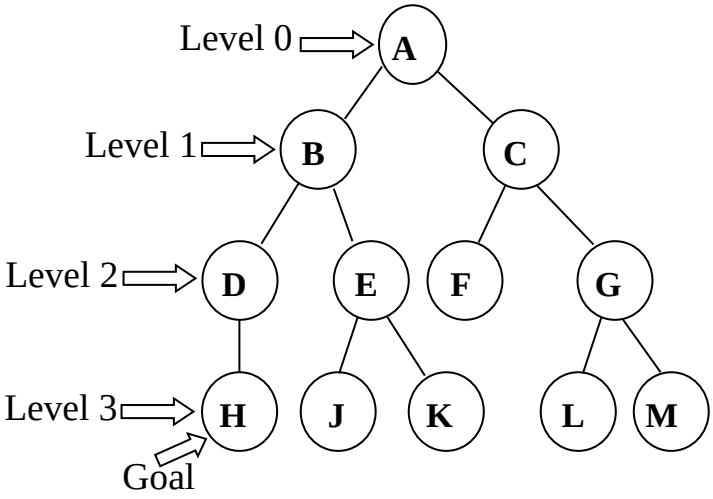
b- Otherwise, remove it from the list and add its children to front of the list.

Step No. 3:- If a goal is found then declare success, otherwise declare failure.

DFS Algorithm(2):

```
initialize: open=[start state], close=[];
while open<>[] do
    remove(discard) the first state from the left of open call it X;
    if X is the goal then return(path);
    else
       begin
           generate all children of X;
           put X in close;
           remove(discard) any child of X already in open or close;
           add the remaining children of X to the left of open;
       end
end
return(fail);
end
```

Example 1: Find the goal (H) using DFS algorithm to the following tree:



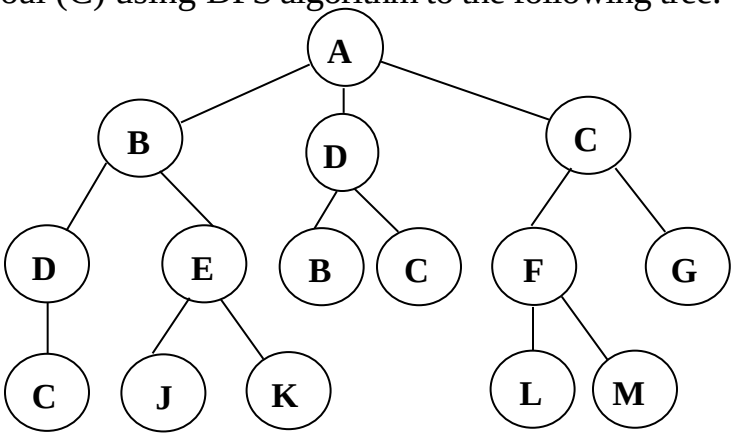
Solution of 1:

First: The (Open, Close) Method:

State Sol.	Open	Closed	Path
0	[A]	[]	A
1	[B] , [C]	[A]	B
2	[D] , [E] , [C]	[B] , [A]	D
3	[H] , [E] , [C]	[D] , [B] , [A]	H
4	[E] , [C]	[H] , [D] , [B] , [A]	

Solution Path(shortest path to reach the solution): A → B → D → H
Search Space(nodes visited during the solution): A --- B --- D --- H
State Space(all nodes in the graph): A --- B--- C --- D--- E --- H
Note: We must get the desired goal (H) in the (Closed) list in order to stop.

***H.W.*:** Find the goal (C) using DFS algorithm to the following tree:

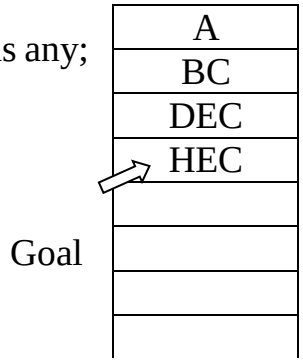


Solution Path: ? Search Space: ? State Space: ?

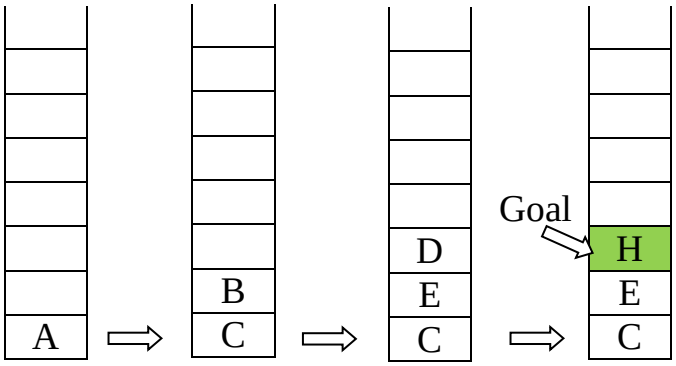
Secondly: The (Matrix) Method

In this method, child nodes are generated from a node if it has any; otherwise, the node is removed from the matrix.

Solution Path: A → B → D → H
Search Space: A --- B --- D --- H
State Space: A --- B--- C --- D--- E --- H

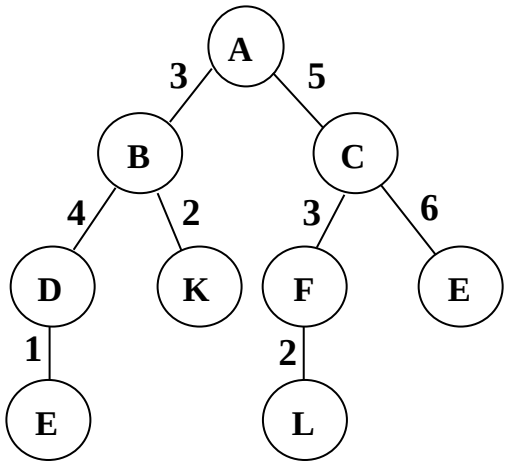


Third: The Stack method:



The solution in this method depends on the (LIFO) principle.

Example 2: Use DFS algorithm to finding the minimum cost for reaching the goal(E) to the following tree:

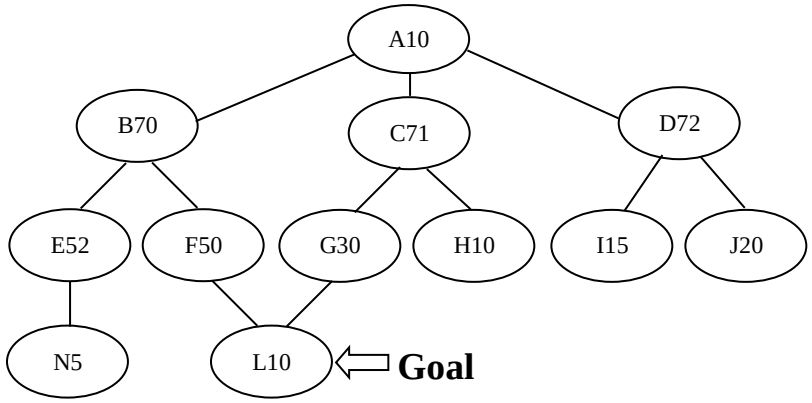


Solution of 2:

State Sol.	Open	Closed	Path
0	[A]	[]	A
1	[B] , [C]	[A]	B
2	[D] , [K] , [C]	[B] , [A]	D
3	[E] , [K] , [C]	[D] , [B] , [A]	E
4	[K] , [C]	[E] , [D] , [B] , [A]	K
5	[C]	[K] , [E] , [D] , [B] , [A]	C
6	[F] , [E]	[C] , [K] , [E] , [D] , [B] , [A]	F
7	[L] , [E]	[F] , [C] , [K] , [E] , [D] , [B] , [A]	L
8	[E]	[L] , [F] , [C] , [K] , [E] , [D] , [B] , [A]	E
9	[]	[E] , [L] , [F] , [C] , [K] , [E] , [D] , [B] , [A]	

Solution Paths: Path one: (A - B - D - E) = 3 + 4 + 1 = 8
Path two: (A - C - E) = 5 + 6 = 11
Therefore, the first path (Path one) is the least costly.
Solution Path: A → B → D → E & A → C → E
Search Space: A --- B --- D --- E --- K --- C --- F --- L --- E
State Space: All the nodes of the graph.

***H.W.*:** Use DFS algorithm (by the three methods above) to find the minimum cost for reaching the goal (L) to the following tree:



The DFS Advantages

- Guaranteeing reaching the (Goal) when it is present.
- The search process takes path after path until it reaches the (Goal), so it is considered efficient for searching in many branches as it does not keep every (Node) in the given level in the (Open) set.

The DFS Disadvantages

- Not guaranteed in finding the shortest path to the solution, for when the path is long and does not contain the (Goal) we will be forced to enter it and waste time, especially when the level is not determined, therefore it has a high (Complex Degree). And this algorithm is preferred when the (Goal) is far (deep) from the (Root).
- Because of the (Back Tracking) a waste of time will happen.

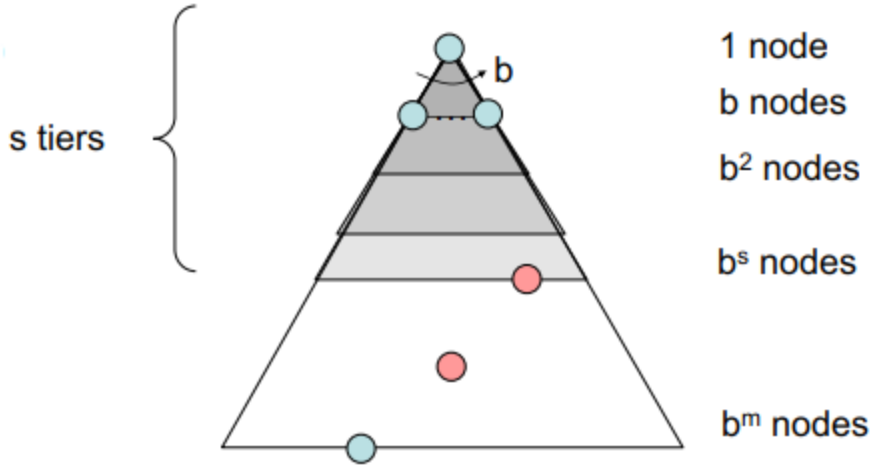
2nd: Breadth First Search(BFS):

In this algorithm all nodes on one level are examined before any of the nodes on the next level. عملية البحث مستوى بعد مستوى

This algorithm is guaranteed to find a solution of one exists, and this method does not only provide a solution but also find the shortest path of the nodes between the start and the goal states.

- In **DFS**, nodes are added from the left side, whereas in **BFS**, nodes are added from the right side.
- In **DFS**, the old duplicated node will be deleted, whereas in **BFS**, the newly duplicated node will be deleted.
- When a stopping level (depth limit) is specified, you must stop upon reaching it, even if the **Goal** has not been reached.
- If a level limit is set (for example, the fifth level) and the **Goal** is reached at the fourth level, the search will stop and will not continue to the fifth level specified in the question.

- The operating principle of this algorithm is **First-In, First-Out (FIFO)**, just like a **Queue**.



BFS Algorithm(1):

Step No. 1:- Form a list consisting of the root node.

Step No. 2:- Until the path is empty or the goal is found, determine if the first element in the list is a goal.

a- If it is a goal, then exit with solution.

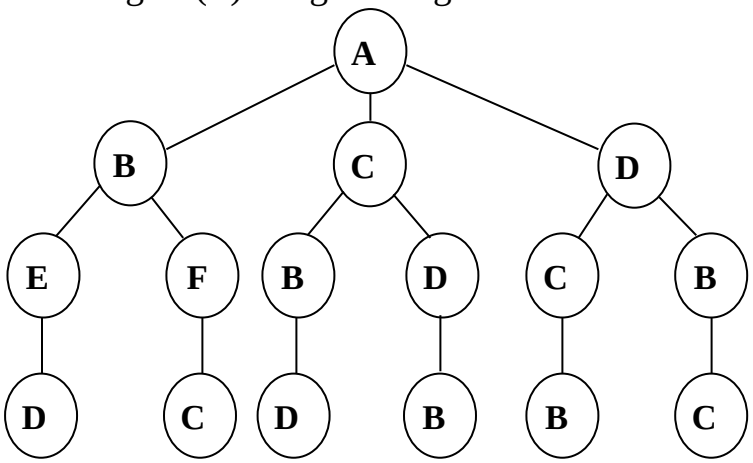
b- Otherwise, remove it from the list and add its children to the front of the list.

Step No. 3:- If a goal is found then declare success, otherwise declare failure.

BFS Algorithm(2):

```
initialize: open=[start state], close=[];
while open<>[] do
    remove(discard) the first state from the left of open call it X;
    if X is the goal then return(path);
    else
       begin
           generate all children of X;
           put X in close;
           remove(discard) any child of X already in open or close;
           add the remaining children of X to the right of open;
       end
end
return(fail);
end
```

Example 1: Find the goal (C) using BFS algorithm to the following tree:



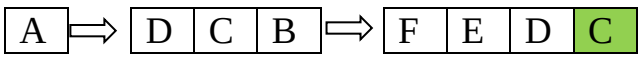
Solution:

First: The (Open, Close) Method:

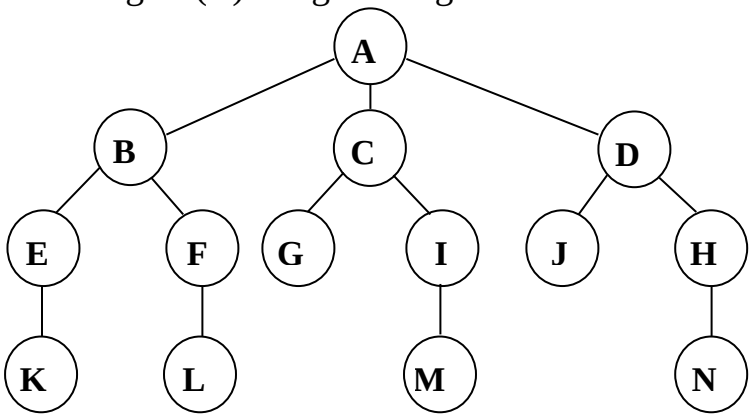
State Sol.	Open	Closed	Path
0	[A]	[]	A
1	[B] , [C] , [D]	[A]	B
2	[C] , [D], [E], [F]	[B] , [A]	C
3	[D], [E], [F]	[C] , [B] , [A]	

Solution Path: A → C
Search Space: A --- B --- C
State Space: All the nodes of the graph.

Second: The Queue method:



Example 2: Find the goal (G) using BFS algorithm to the following tree:



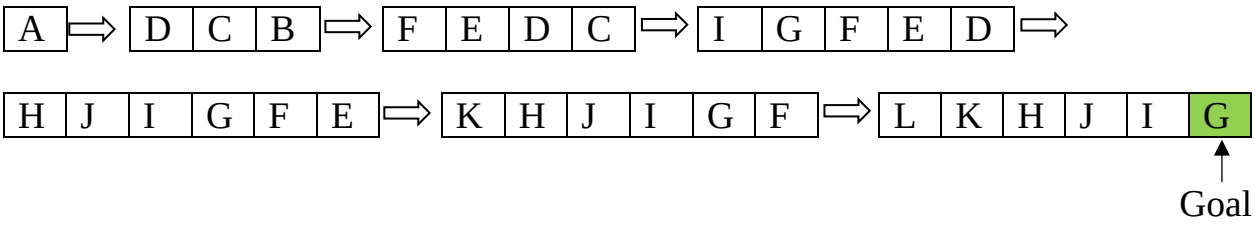
Solution:

First: The (Open, Close) Method:

State Sol.	Open	Closed	Path
0	[A]	[]	A
1	[B], [C], [D]	[A]	B
2	[C], [D], [E], [F]	[B], [A]	C
3	[D], [E], [F], [G], [I]	[C], [B], [A]	D
4	[E], [F], [G], [I], [J], [H]	[D], [C], [B], [A]	E
5	[F], [G], [I], [J], [H], [K]	[E], [D], [C], [B], [A]	F
6	[G], [I], [J], [H], [K], [L]	[F], [E], [D], [C], [B], [A]	G
7	[I], [J], [H], [K], [L]	[G], [F], [E], [D], [C], [B], [A]	

Solution Path: A → C → G
Search Space: A --- B --- C --- D --- E --- F --- G
State Space: All the nodes of the graph.

Second: The Queue method:



H.W.: Find the goal (D) to the same example(2) above by using BFS.

The BFS Advantages

- **It finds the shortest path of nodes.** It guarantees reaching the target by finding the shortest path between the nodes.
- **No backtracking.** It is faster than DFS in getting the Goal because it does not need to return to the starting point. It examines all nodes at every level before moving to the next level.
- **It is good when the number of alternatives at the choosing points is not too large.** This algorithm is preferred when the Goal is not deep (far) from the Root.

The BFS Disadvantage

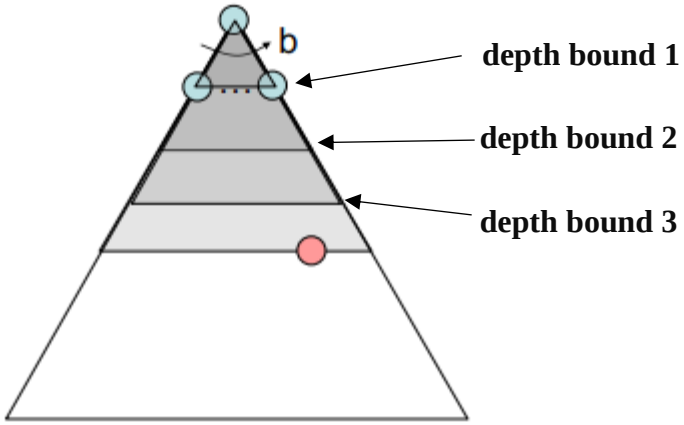
- It requires a lot of memory, the number of nodes increases exponentially with the level number.
- It requires a lot of work if the shortest path is quit long, the number of nodes increases exponentially with the length of the path.
- In situation in which there are many paths that lead to solutions, but each of them is quit long, then DFS is more suitable to apply than BFS.

Comparison between DFS and BFS:

DFS	**BFS**
1. Depend on stack(LIFO).	1. Depend on Queue(FIFO).
2. Used when the goal is far from the root or start state.	2. Used when the goal is near from the root or start state.
3. The search is vertically in depth from left until we reach the goal.	3. The search is horizontally for all levels from left until we reach the goal.
4. Wasting time because it contains backtracking.	4. No backtracking, so no waste time.
5. Not Optimal: It finds "a" path, but rarely the shortest one.	5. Optimal: Guaranteed to find the shortest path in unweighted graphs.
6. Not always guaranteed.	6. Always guaranteed.
7. It is Efficient, it doesn't store all the nodes, so it needs less memory.	7. It is Expensive, because it needs a lot of memory.

3rd: Hyper First Search(HFS):

In this type, the search level is determined, and it is a combination of the two previous types: DFS and BFS.



HFS Algorithm:

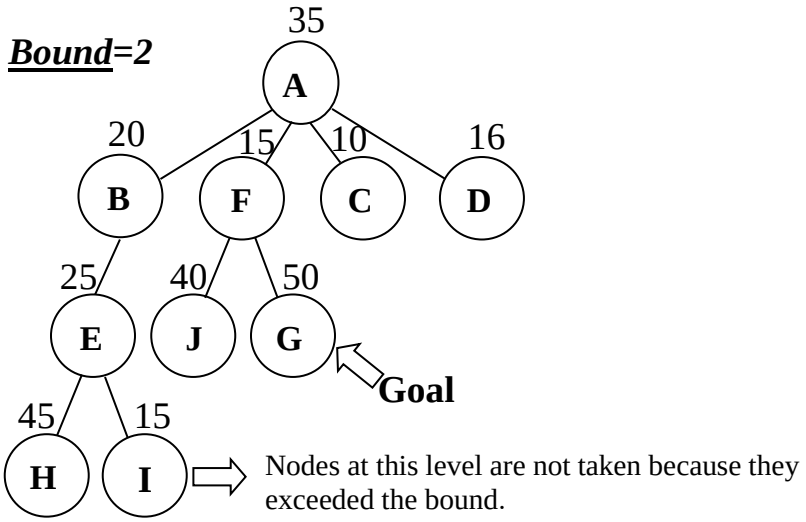
Step No. 1:- Set current depth bound=1;
Step No. 2:- Put the initial node in the open;
Step No. 3:- While open<>empty and depth within the depth bound:

```
begin
    remove left most state from open call it X;
    if X is the goal then return(success);
    else
      begin
        generate children of X;
        put X in close;
         eliminate children of X already in open or close;
        put remaining children to the left of open;
      end
end
```

Step No. 4:- Increment depth bound by 1 and repeat through 2;

```
end
```

Example 1: Find the goal (G) using HFS algorithm to the following tree:

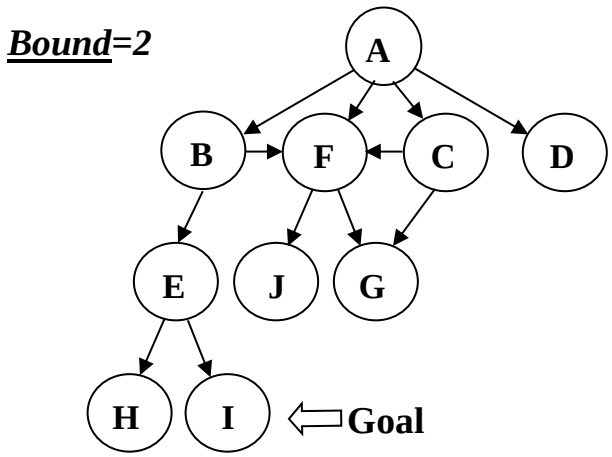


Solution of 1:

#	Open	Close
0	[A,0]	[]
1	[B,A] , [F,A] , [C,A] , [D,A]	[A,0]
2	[E,B] , [F,A] , [C,A] , [D,A]	[B,A] , [A,0]
3	[F,A] , [C,A] , [D,A]	[E,B] , [B,A] , [A,0]
4	[J,F] , [G,F] , [C,A] , [D,A]	[F,A] , [E,B] , [B,A] , [A,0]
5	[G,F] , [C,A] , [D,A]	[J,F] , [F,A] , [E,B] , [B,A] , [A,0]
6	[C,A] , [D,A]	[G,F] , [J,F] , [F,A] , [E,B] , [B,A] , [A,0]

Solution Path: A --- F --- G = 35+15+50=100
Search Space: A --- B --- E --- F --- J --- G
State Space: A --- B --- F --- C --- D --- E --- J --- G

Example 2: Find the goal (I) using HFS algorithm to the following graph:



Solution of 2:

#	Open	Close
0	[A,0]	[]
1	[B,A] , [F,A] , [C,A] , [D,A]	[A,0]
2	[E,B] , ~~[F,B]~~ , [F,A] , [C,A] , [D,A]	[B,A] , [A,0]
3	[F,A] , [C,A] , [D,A]	[E,B] , [B,A] , [A,0]
4	[J,F] , [G,F] , [C,A] , [D,A]	[F,A] , [E,B] , [B,A] , [A,0]
5	[G,F] , [C,A] , [D,A]	[J,F] , [F,A] , [E,B] , [B,A] , [A,0]
6	[C,A] , [D,A]	[G,F] , [J,F] , [F,A] , [E,B] , [B,A] , [A,0]
7	~~[F,C]~~ , ~~[G,C]~~ , [D,A]	[C,A] , [G,F] , [J,F] , [F,A] , [E,B] , [B,A] , [A,0]
8	[D,A]	[C,A] , [G,F] , [J,F] , [F,A] , [E,B] , [B,A] , [A,0]
9	[] It became empty, and we did not get the solution.	[D,A] , [C,A] , [G,F] , [J,F] , [F,A] , [E,B] , [B,A] , [A,0]
10	[A,0]	[]
11	[B,A] , [F,A] , [C,A] , [D,A]	[A,0]
12	[E,B] , ~~[F,B]~~ , [F,A] , [C,A] , [D,A]	[B,A] , [A,0]
13	[H,E] , [I,E] , [F,A] , [C,A] , [D,A]	[E,B] , [B,A] , [A,0]
14	[I,E] , [F,A] , [C,A] , [D,A]	[H,E] , [E,B] , [B,A] , [A,0]
15	[F,A] , [C,A] , [D,A]	[I,E] , [H,E] , [E,B] , [B,A] , [A,0]

Solution Path: ?
Search Space: ?
State Space: ?

-In this example, we do not obtain the (Goal) because the question statement limits it to (Bound=2). However, when the requirement is to reach the (Goal) by any means, we will increase the (Depth Bound) by (1) to become (Bound=3) and increase the (Iterative) to become (10), and start solving from the (Root) again until we obtain the (Goal).

***H.W.*:** undoing the example(2) above by using BFS, where(Bound=3).