BFS (Breadth First Search)

BFS stands for Breadth First Search is a type of uninformed state-pace searching algorithm and a general graph-searching algorithm.

Q) Why it is uniformed search algorithm?

Ans: This search algorithm does not know when it will reach its goal state. It is also known as unguided search or blind search. It works on basis of Brute Force technique.

Brute Force Technique: It checks each elements along its way to get the solution.

There are two type of states obtained 1) non-final state and 2) final state.

In Non-final state when goal state is not reached but at each state it obtain a new state and proceed to get the final goal state.

In Final State when goal state is reached.

Some terminologies:

1. **Start State: This is also known as goal-driven agent . This is the state in which the program starts to find the solution. It is the initial condition of the agent.**

2. **Goal state: This the output of the search.**

3. **State space: This refers to all the states that can be reached from the start state by a series of actions. It can also be thought of as all the states in which the agent can exits.**

4. **Action space: This refers to the list of all possible actions that can be executed by the agent. More specifically, it tells us about all the actions that can be executed in a particular state.**

Note: "Each state can move to another state by transition, how that transition occurs is called action."

5. **Transition model: This gives us a description of the outcomes of the actions performed in a state.**

6. **Goal Test: This is the method to check whether the current state is goal state or not.**

7. **Path cost: This is a function that is used in assigning vales to a path, which can be thought of as cost with respect to performance.**

8. **Path: The route along which we can reach goal state.**

Q) Why Breadth First Search algorithm is called Graph Searching Algorithm?

Ans: In graph searching technique, also known as traversal technique visits every node exactly one in a systematic fashion. Same technique used in BFS or breadth first search.

Q) How Breadth First Algorithm works?

Procedure:

1) **Here we first expand the root node and then we go for expanding the successor which are also known as 'neighbours' , then their successors are expanded next and so on.**

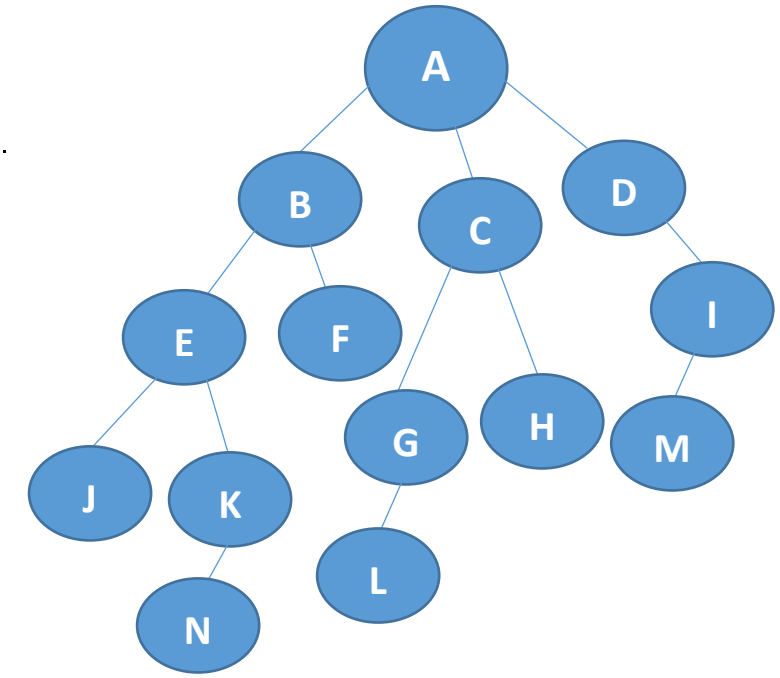
2) **It is an instance of general graph search algorithm in which the shallowest unexpanded node is chosen for expansion.**

"Q) What is shallower node and deeper nodes?

Ans:

Shallow means having little depth.

Consider a graph:



B,C,D nodes are deeper nodes than A and A is the shallowest node .

Similarly E, F nodes are deeper nodes than B and B is shallower node of E,F and similarly G, H are deeper node of C and C is shallower node of G,H etc.

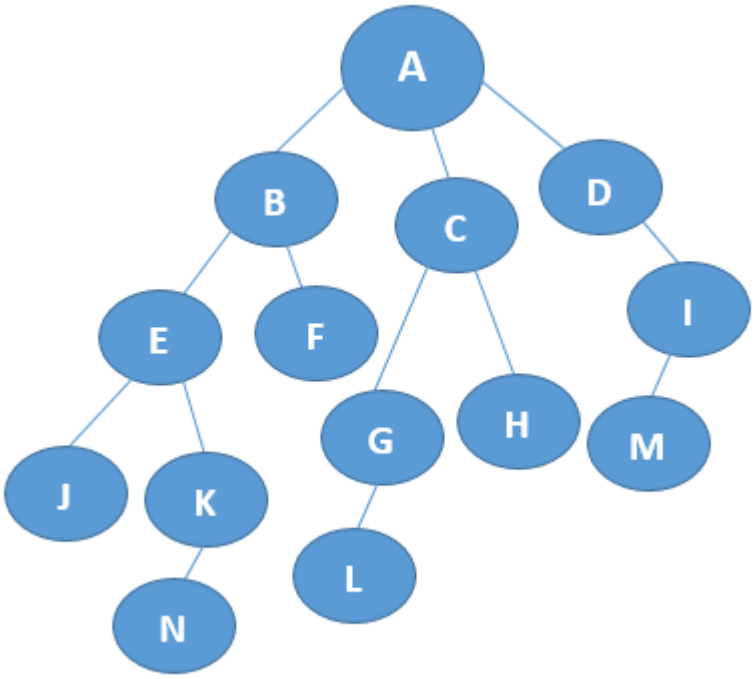
Most simpler way: BFS always starts and processes all nodes that are direct neighbours of starting node. Then it processes all direct neighbours of direct neighbours of starting nodes (excluding the ones already processed) and so on.

The last nodes to be processed are the ones with the longest distance from starting node.

The starting node is shallowest node and last leaf node processed is the deepest node. "

3) **We use a FIFO (First in First Out) Queue for achieving this functionality. That is "ENQUEUE"(Addition of Element) will occur at rear(end) of Queue and "DEQUEUE"(Deletion of Element) will occur at front of queue.**

Again, recall the above graph:

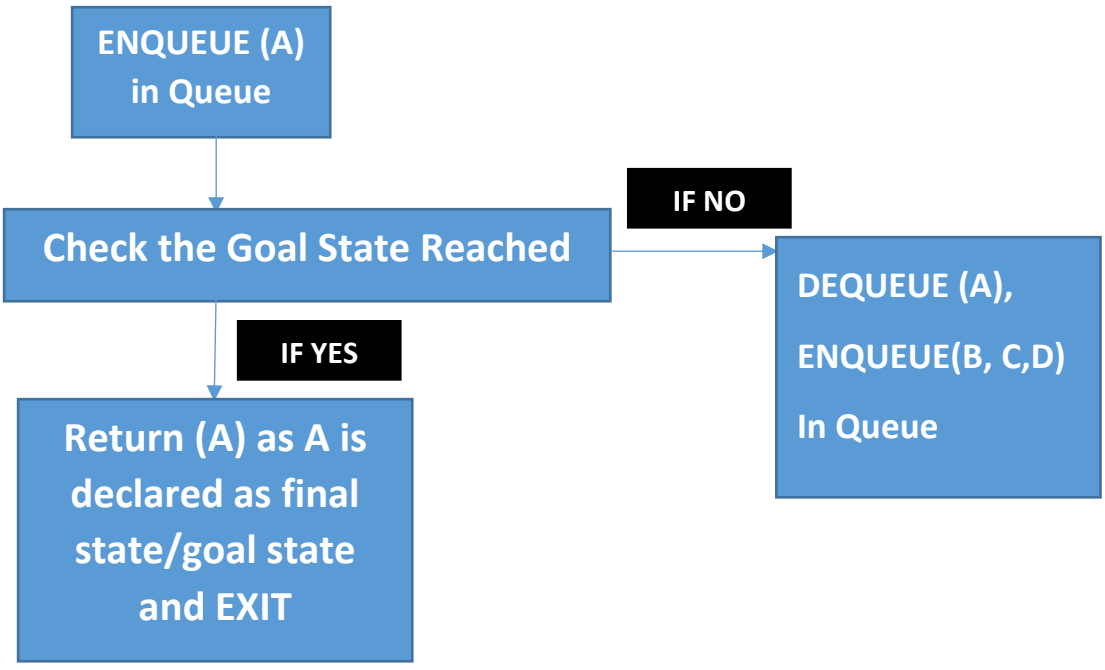


1. **First search process will start in A , so A will be in Queue:**

Queue

A

Now,



Suppose, A is not the goal state ,

DEQUEUE(A)

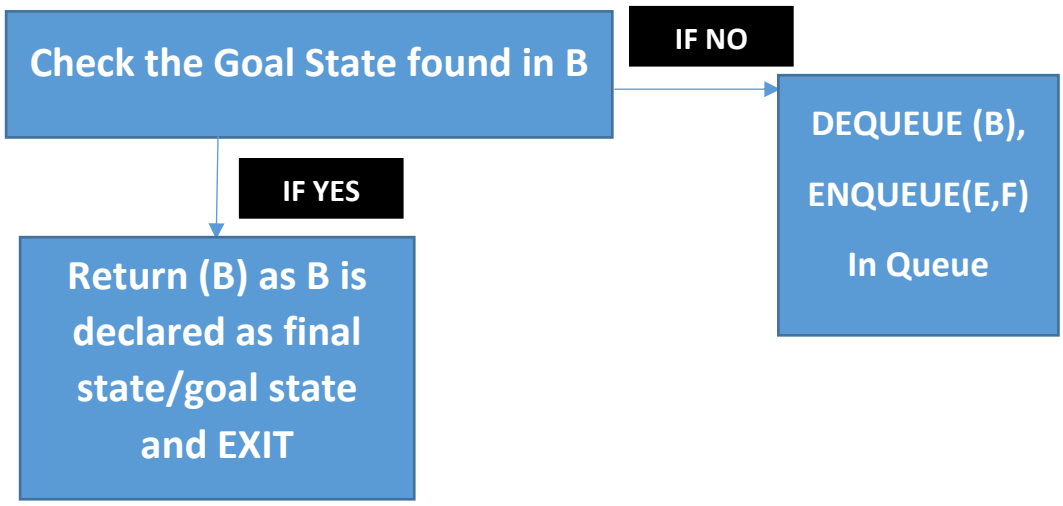
ENQUEUE (B,C,D) – A's neighbours or successors in Queue, -B,C,D are the new states added at the end of the Queue.

Note B,C,D can be inserted and placed in order in $3_{P_3} = 3!$ ways in the queue.

Lets it being inserted as:

B	**C**	**D**

Now, goal state is checked according to the order it have been inserted in to the Queue.



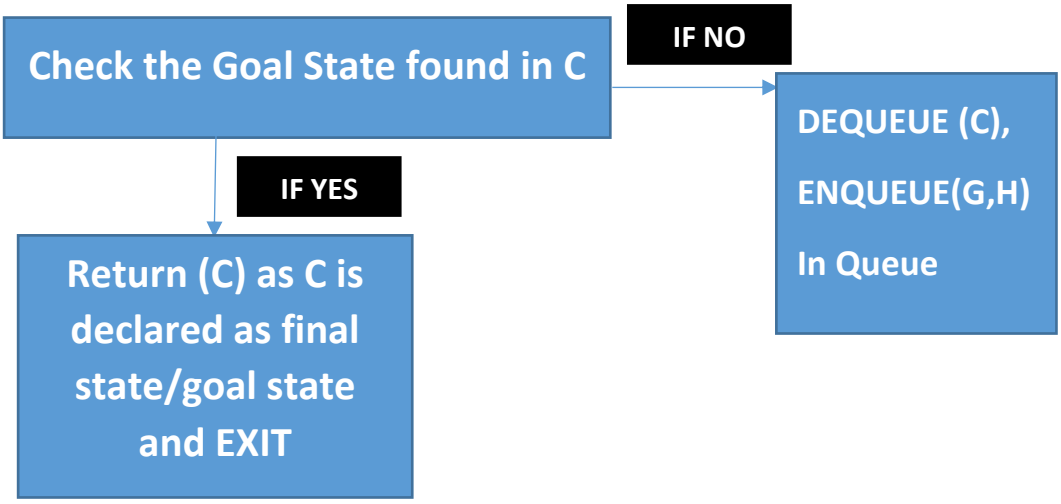
As, E,F is successor of B.

Note , E,F can be inserted and placed in Queue in any order i.e. in 2P2 = 2! ways And E,F are the new states added at the end of the queue.

Lets it being inserted as:

C	D	E	F

Next if B is not the goal state then,



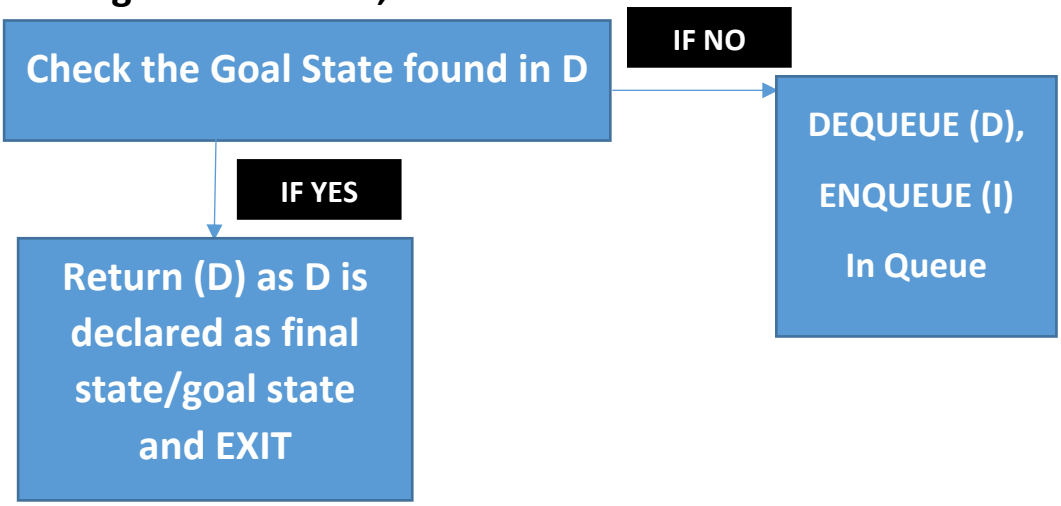
As, G,H is successor of C.

Note , G,H can be inserted in Queue in any order in 2P2 = 2! ways. And G,H are the new states added at the end of the queue.

Lets it being inserted as:

D	E	F	G	H

Next if C is not the goal state then,



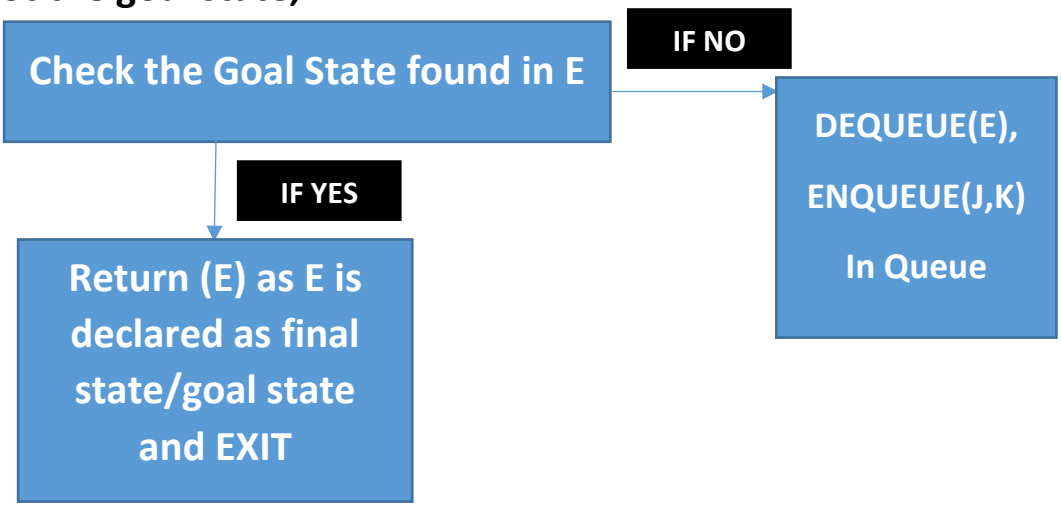
As 'I' is successor of D.

Note , 'I' can be inserted in Queue in any order in 1P1 = 1! ways i.e. 1 way only. And I is the new state added at the end of the Queue.

Therefore,

E	F	G	H	I

Next if D is not the goal state,



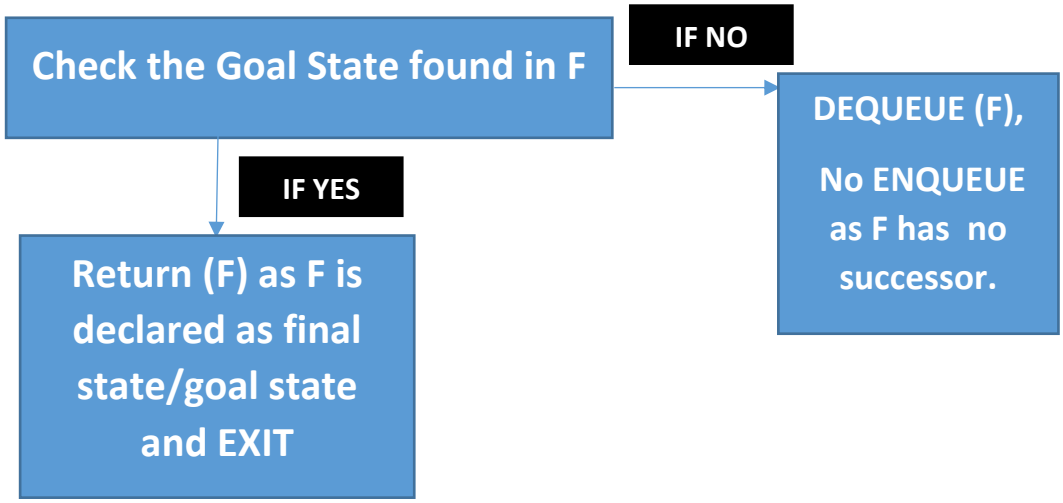
As, J,K is successor of E.

Note , J,K can be inserted in Queue in any order in 2P2 = 2! ways. And J , K are the new states added at the end of the Queue.

Lets it being inserted as:

F	G	H	I	J	K

Next According to the queue structure present in front of us , searching will occur, if 'E' is not the goal state :

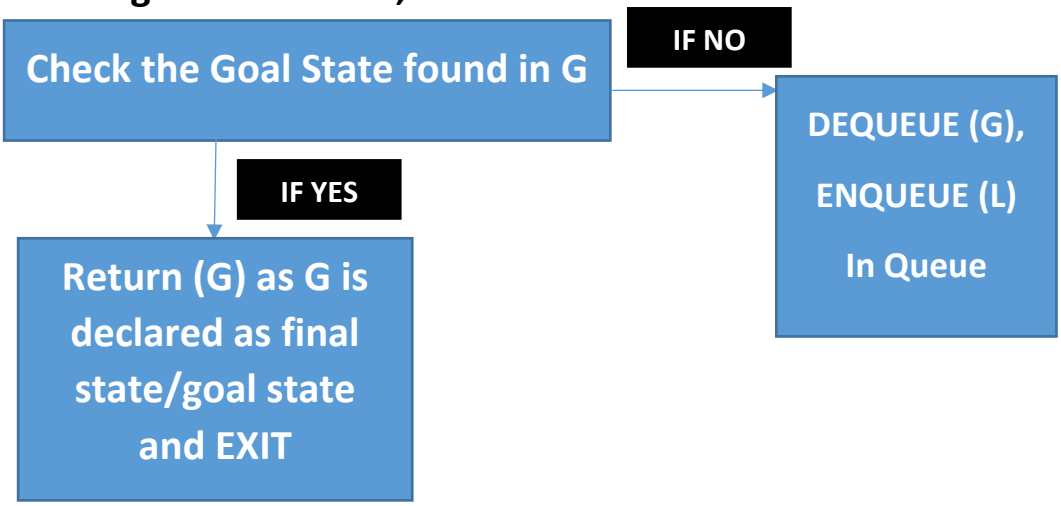


Note : F has no successor, hence no new states added at the end of the queue.

Now, queue we have:

G	H	I	J	K

Next if 'F' is not the goal state then,

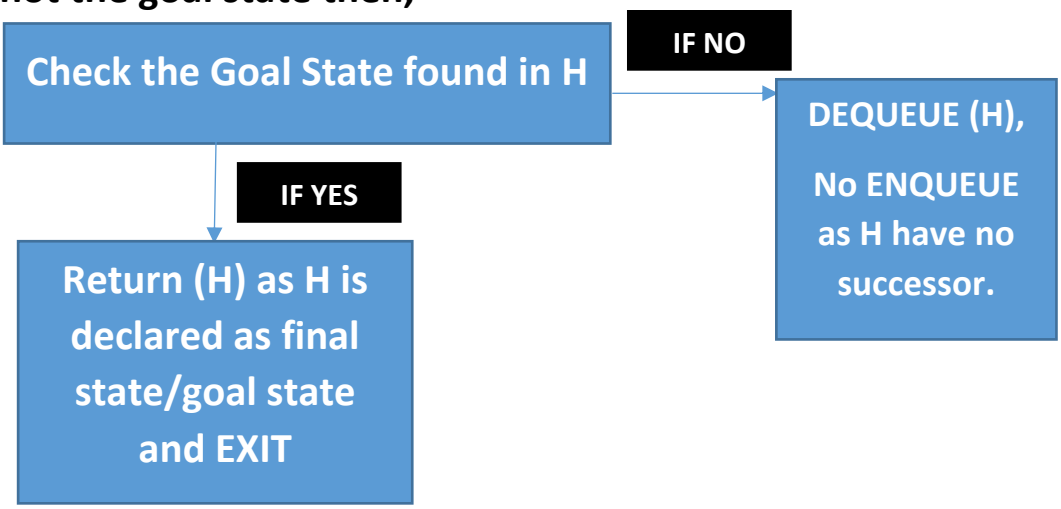


As 'L' is the successor of 'G' and can be inserted in 1P1 or 1! or 1 ways in queue. L is the new state added at the end of the queue.

Now, queue we have:

H	I	J	K	L

Next if 'G' is not the goal state then,

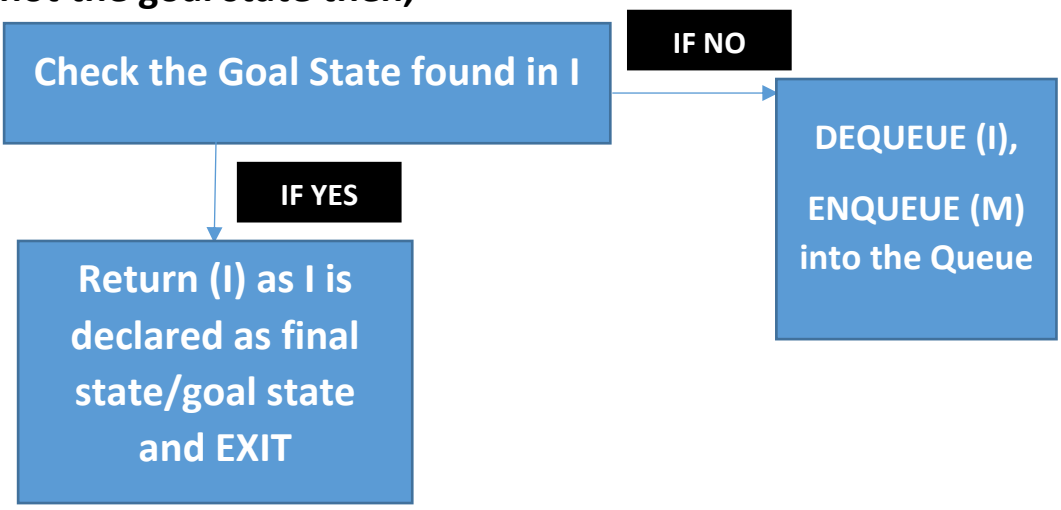


Note : H has no successor, hence no new state added at the end of the queue.

Now, queue we have:

I	J	K	L

Next if 'H' is not the goal state then,

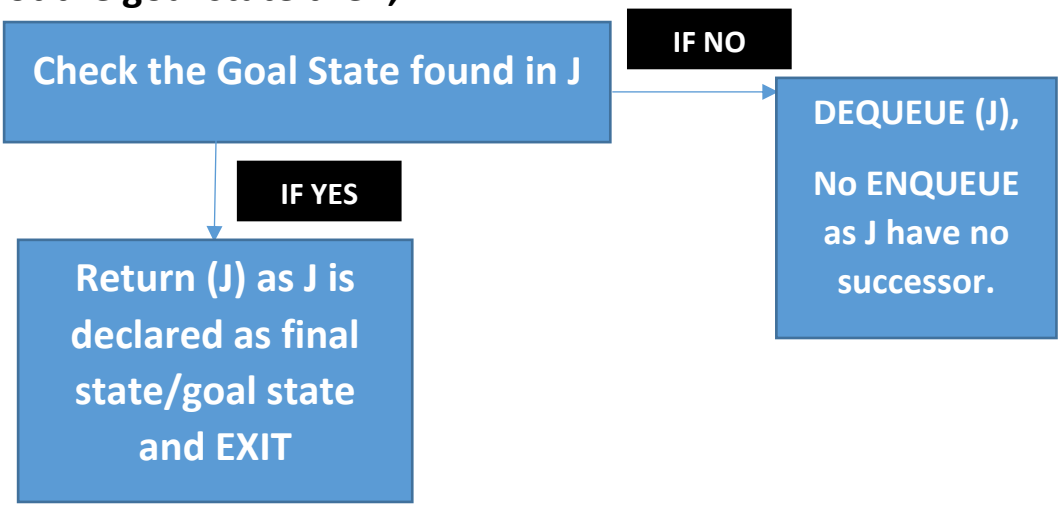


As 'M' is the successor of 'I' and can be inserted in 1P1 or 1! or 1 ways in queue. 'M' is the new state added at the end of the queue.

Now, queue we have:

J	K	L	M

Next if 'I' is not the goal state then,

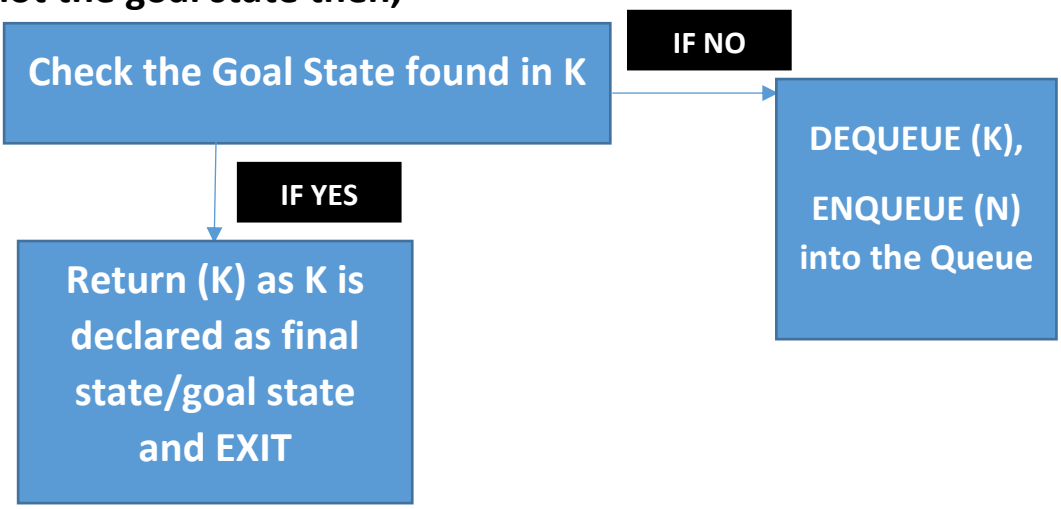


Note : J has no successor, hence no new states were added.

Now, queue we have:

K	L	M

Next if 'J' is not the goal state then,



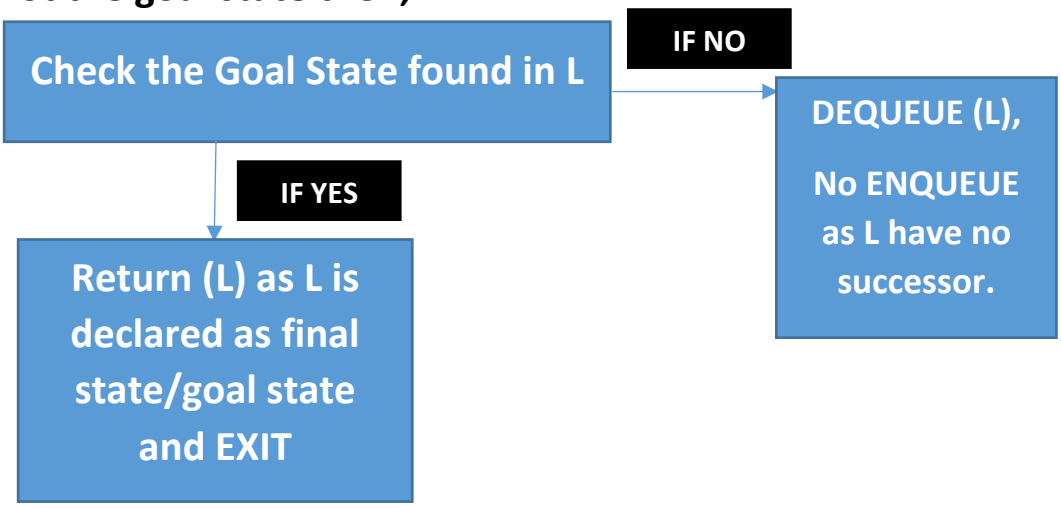
As ‘N’ is the successor of ‘N’ and can be inserted in 1P1 or 1! or 1 ways in queue. N is the new state added at the end of the queue.

Now, queue we have:

L	M	N

Now, L,M,N have no successors/neighbours hence no Enqueue into the Queue take place. Hence no new states were added further.

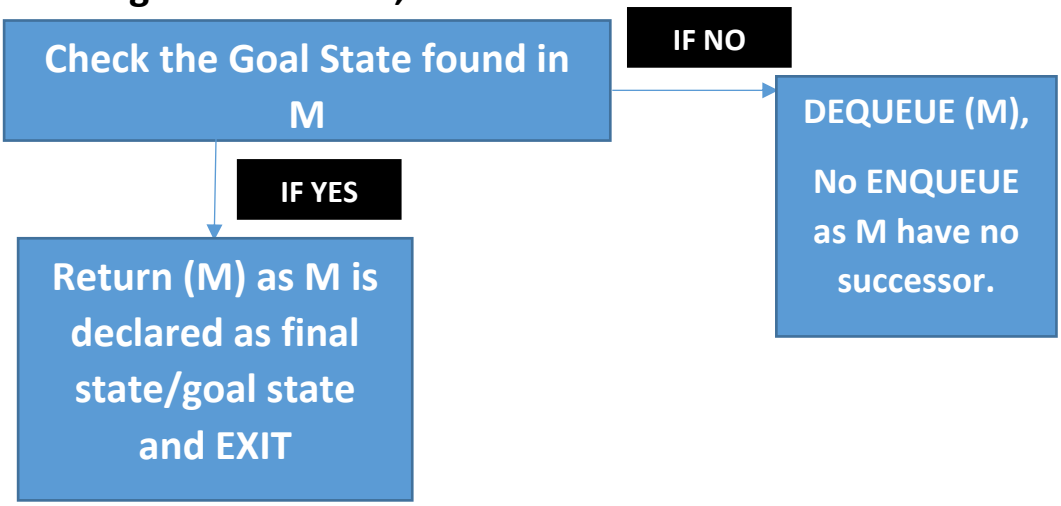
Next if ‘K’ is not the goal state then,



Now, queue we have:

M	N

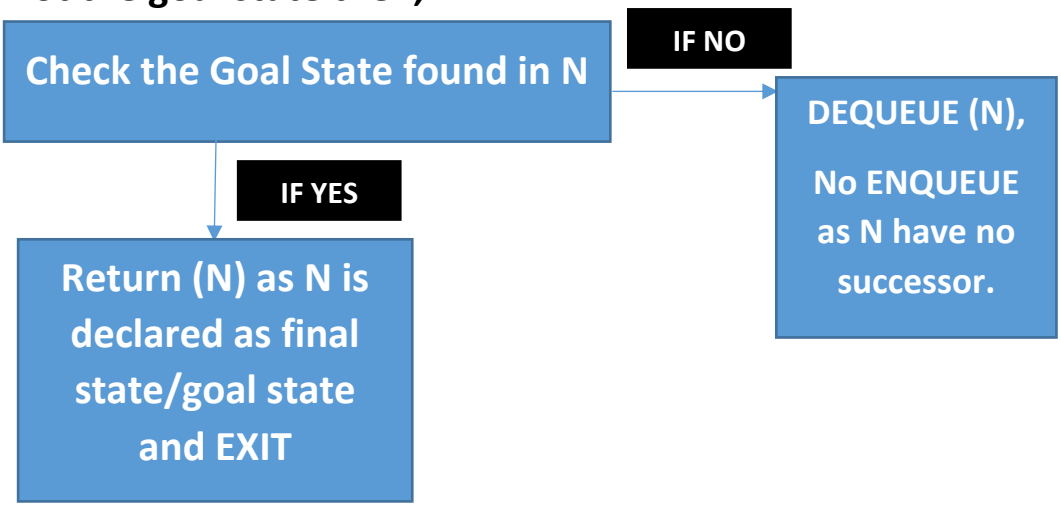
Next if 'L' is not the goal state then,



Now, queue we have:

N

Next if 'M' is not the goal state then,



Now we have empty queue. If goal is not found then the item we have searched is not there in the Queue.

Then it's a failure.

If BFS done in term of Linked List then:

Nodes are added at the end of the Queue. These Nodes become the new states.

ALGORITHM

Insert the root nodes in the FIFO Queue.

Loop through the queue till the queue is empty and do the following steps:

1. **Remove the front element of the queue.**
2. **Check the front element if it is the goal state.**
3. **If the goal state:**
   a. **Return this node.**
4. **Else:**
   a. **"ENQUEUE" all the neighbours of the "DEQUEUE "element in the queue.**

Alternative,

1. **Create a variable called 'NODE-LIST' and set it to the initial state.**
2. **Until a goal state is found or 'NODE-LIST' is empty:**
   a. **REMOVE first element from NODE-LIST and call it E.**
      If 'NODE-LIST' was empty, quit.

b. For each way that each rule can match the state described in E do:
   i. Apply to generate a new state.
   ii. If the new state is goal state, quit and return this state.
   iii. Otherwise, add the new state to the end of the 'NODE-LIST'.

Advantage of BFS (Breadth First Search) over DFS (Depth First Search):

1. Breadth first search Algorithm will not be trapped exploring a blind alley. This contrasts with depth-first searching , which may follow a single, unfruitful path for a very long time, perhaps forever, before the path actually terminates in a state that has no successors. This is a particular problem with depth-first search if there are loops(i.e. a state has a successor that is also one of its ancestors) unless special care is expended to test for such a situation.

2. If there is solution, then breadth-first search is guaranteed to find it. Furthermore, if there are multiple solutions, then a minimal (i.e. one requires the minimum number of steps) will be found.

   This is guaranteed by the fact that longer paths are never explored until shorter ones have already been examined. This contrasts with depth first search which may find a long to a

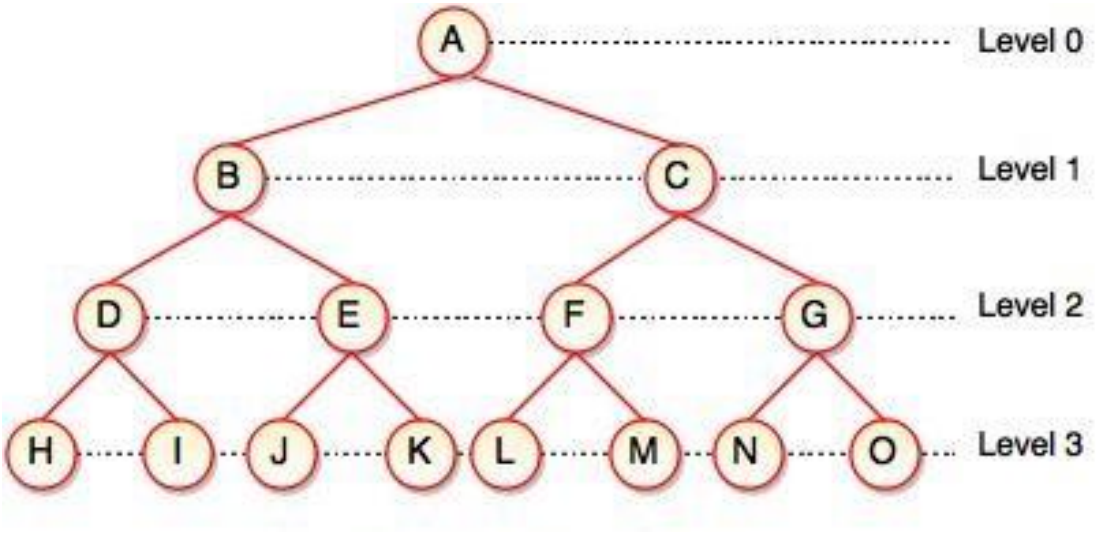
solution in one part of the tree, when a shorter path exists in some other , unexplored part of the tree.

Hence, Breadth First Search is a Complete Search.

Time Complexity of BFS

Note: BFS (Breadth First Search) Algorithm give shortest result and it is optimal.

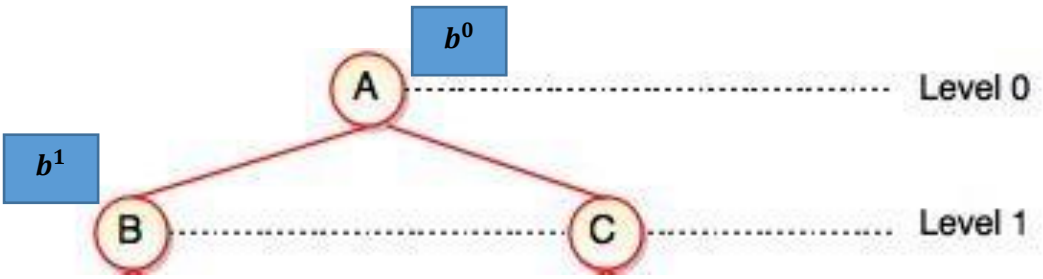
Lets divide BFS into several levels:



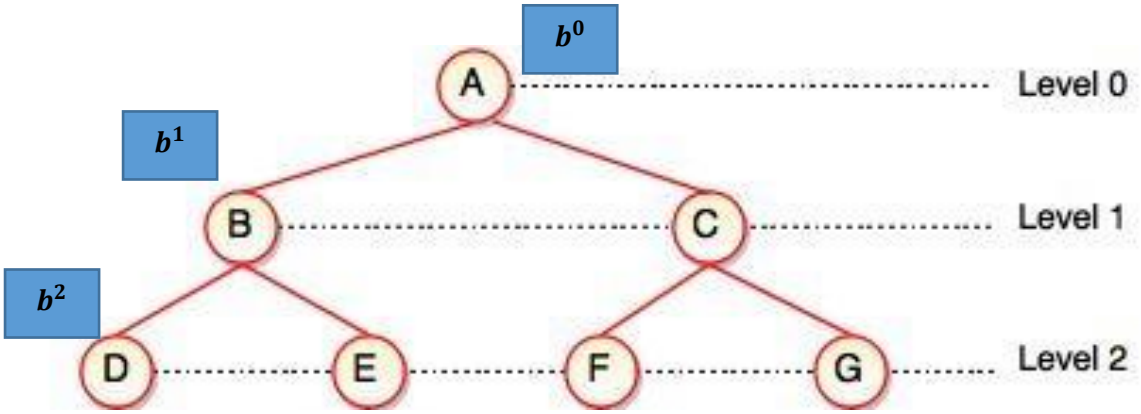
Searching goes through different levels Level 0, Level 1, Level 2, Level 3 and so on.

Therefore most shallowest node is level 'A' , Now shallowest node is not necessarily optimal one. That is we may not find out desired goal at LEVEL 0.

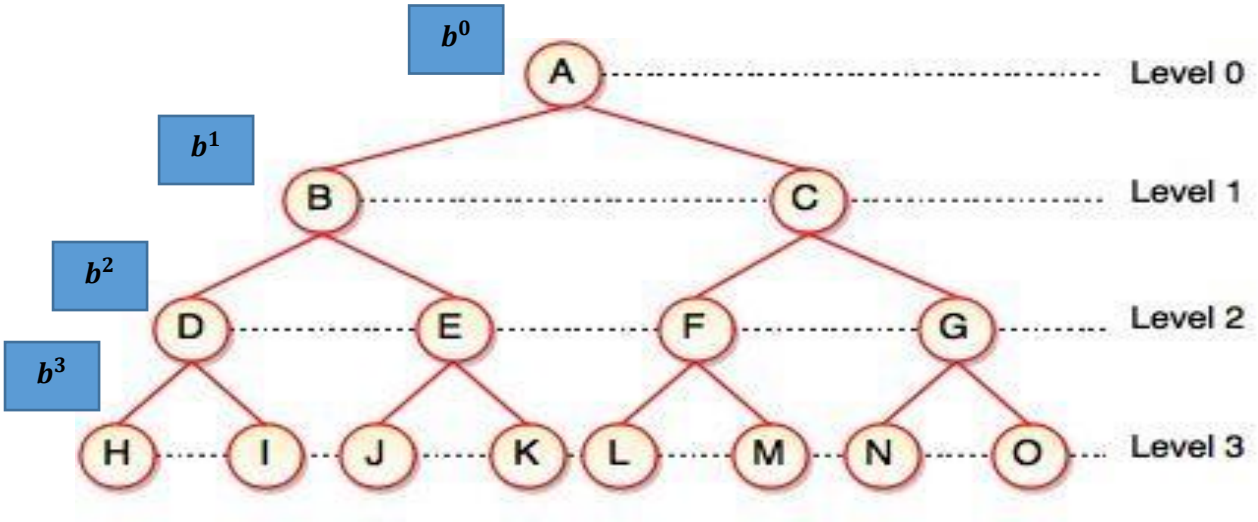
When we move to next level 1 that 'b' successors i.e. b no. of branches connected to nodes, hence b is also known as branch factor. There 2 individual successor that have b^1 time complexity where depth is 1.



Then, for 2nd level it is b^2 where depth is 2 .

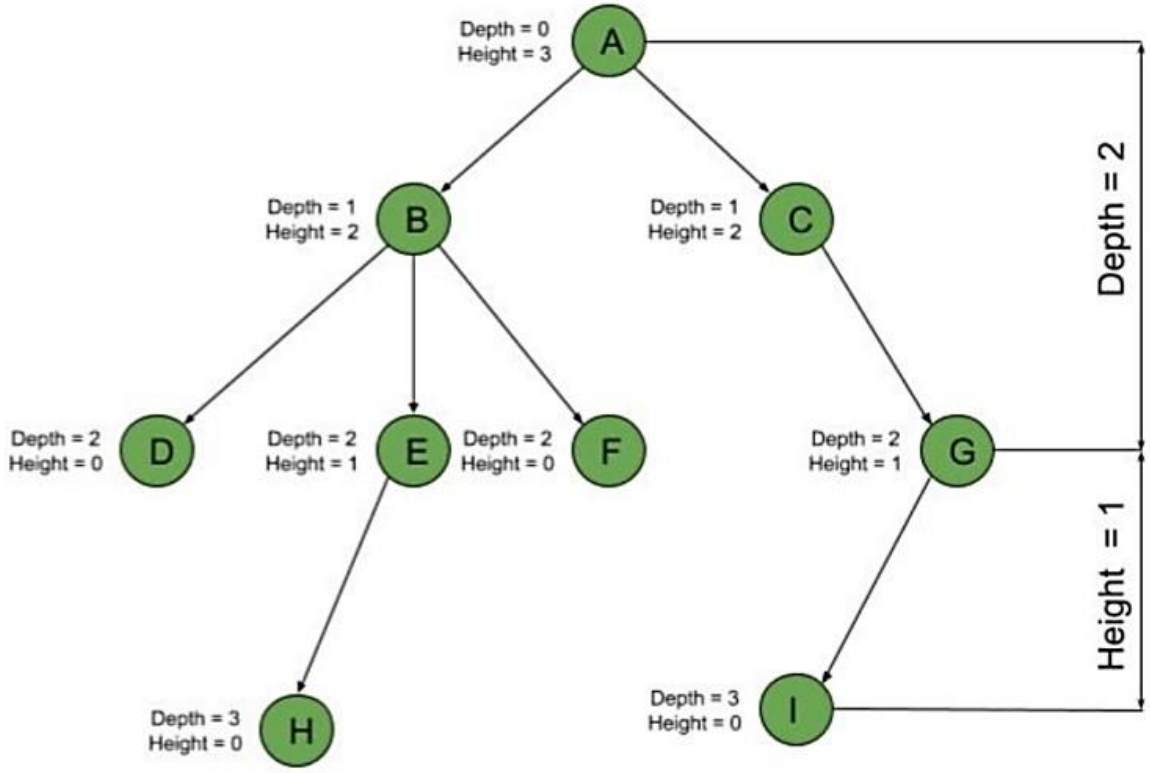


Then, for 3rd level it is b^3 where depth is 3.



if we observe closely , depth of a Tree is determined by no. of edges / branches connected to the vertex.

Consider the below graph:



Node : A have no edge hence Depth is 0 and also no branches/edges hence Level 0 has b^0 time complexity.

Similarly ,B and C have a single edge traversal to root A , hence Depth is 1, hence at this Level 1 has b^1 time complexity.

Similarly ,D,E and F till root A each of it have 2 edges traversal to root A, hence Depth is 2, also G have 2 edges traversal to root A , G's Depth is also 2 hence at this Level 2 has b^2 time complexity.

Similarly, H and I have 3 edges traversal to root A, hence Depth of H and I is 3 and at this level 3 has b^3 time complexity.

Hence total number of nodes generated is:

$$b^0 + b^1 + b^2 + b^3 + \cdots + b^d = O(b^d)$$

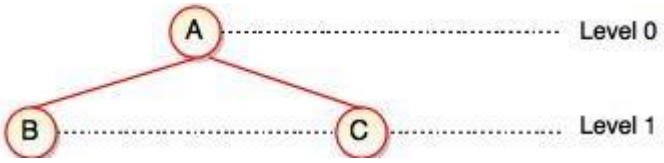
Where,

b *is branch factor*,

d *is depth*

Note: If the time complexity is applied during goal test , then depth will be expanded at the rate of $d + 1$ before the goal was detected and the time complexity would be $O(b^{d+1})$.

Lets say C is the goal,



Before C, B is searched remembering C to be searched if the goal is not reached in B.

Up to B search is not completed also the level is not yet fully completed but due to depth of B is 1 time complexity is b^1 i.e. depth have already expanded before search is completed.

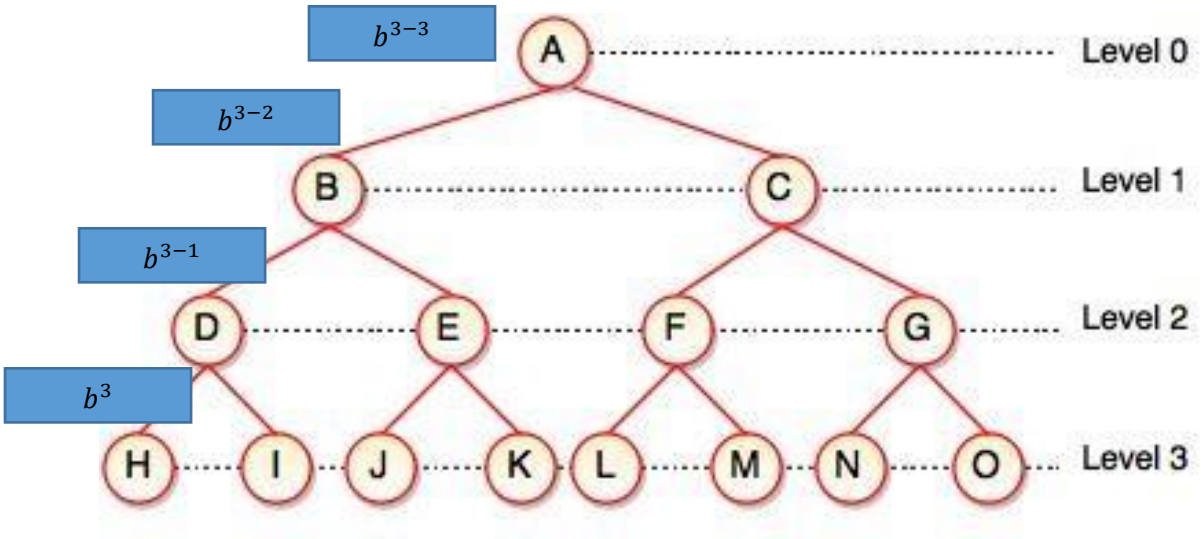
Hence in such a case time complexity of $O(b^{d+1})$ is maintained, as search and level is not yet completed remembering the next node to be searched to obtain the goal state.

Space Complexity

In space complexity, it is very simple:

$$b^0 + b^1 + b^2 + b^3 + \cdots b^{d-2} + b^{d-1} + b^d = O(b^d)$$

i.e. As depth decreases space is also decreases .



i.e. At b^3 more space is consumed for searching as more number of nodes exists.

At b^2 less space needed than b^3

At b^1 less space needed than b^2

At b^1 less space needed than b^0

But at b^3 ultimate space is consumed for searching if goal state is not reached.

Hence , giving the equation:

$$b^0 + b^1 + b^2 + b^3 + \cdots b^{d-2} + b^{d-1} + b^d = O(b^d)$$