Breadth first search

Introduction

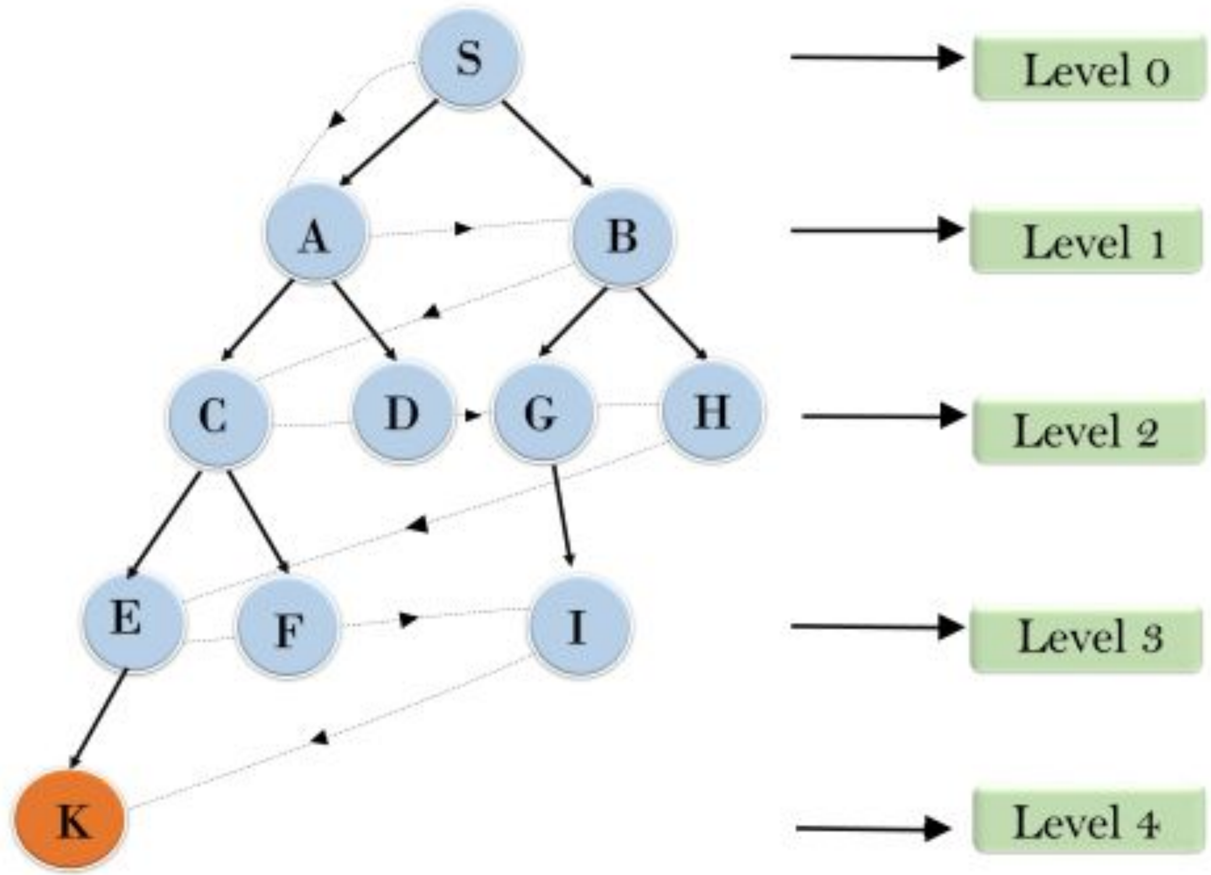
- Breadth-first search is the most common search strategy for traversing a tree or graph. This algorithm searches breadthwise in a tree or graph, so it is called breadth-first search.
- BFS algorithm starts searching from the root node of the tree and expands all successor node at the current level before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

- **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.
- **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 search 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

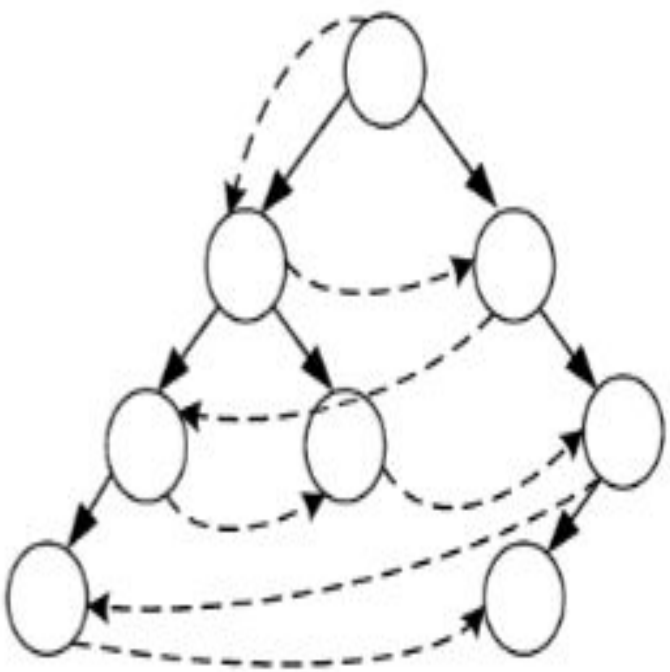
Breadth First Search



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

- **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.

Example(2)



ALGORITHM

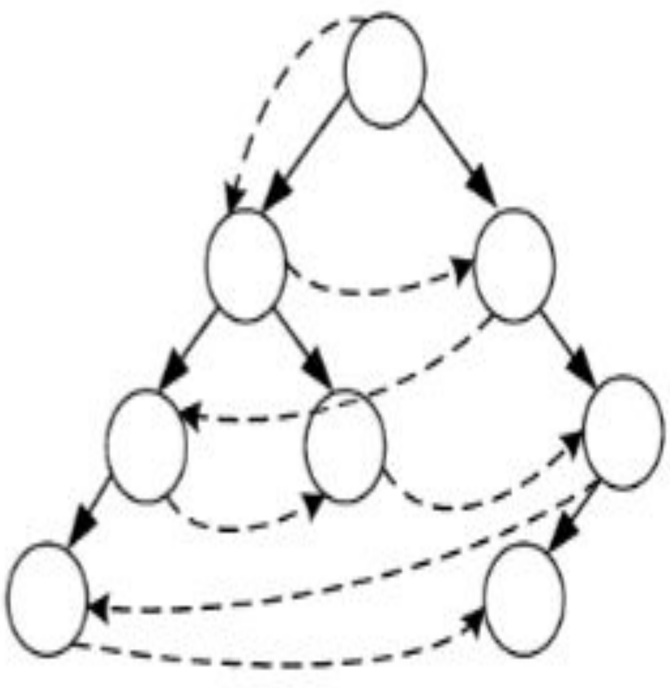
- Create a node list (Queue) that initially contains the first node.

- Do till a goal state is found
    - Remove X from node-list is empty. Then exit.

•Check if this is goal state.

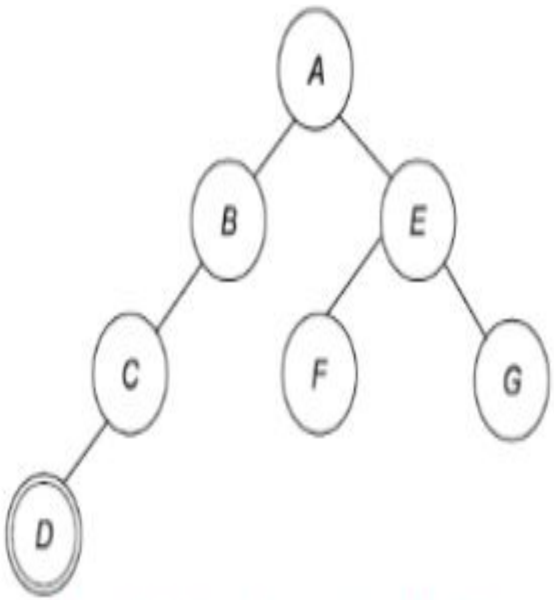
•If yes, return the state.

•If not, get the next level nodes, i.e. node reachable from the parent and add to node-list.



Let
A = Start State
D = Goal state

Now Solving by the queue method



Queue	*Check*
A	—
	Removed *A*, is it goal? No, add children. So, *B* and *E* are added.
BE	
E	Removed *B*, is it goal? No, add children. So, *C* is added at the end of queue.
EC	
C	Removed *E*, is it goal? No, add children. So, *F* and *G* are added.
CFG	
FG	Removed *C*, is it goal? No, add children. So, *D* is added.
FGD	
GD	Removed *F*, is it goal? No, add children. Cannot be added, leaf node, so remove the next node.
D	Removed *G*, is it goal? No, add children. Cannot be added, leaf node, so remove the next node.
Empty	Removed *D*, is it goal? YES!