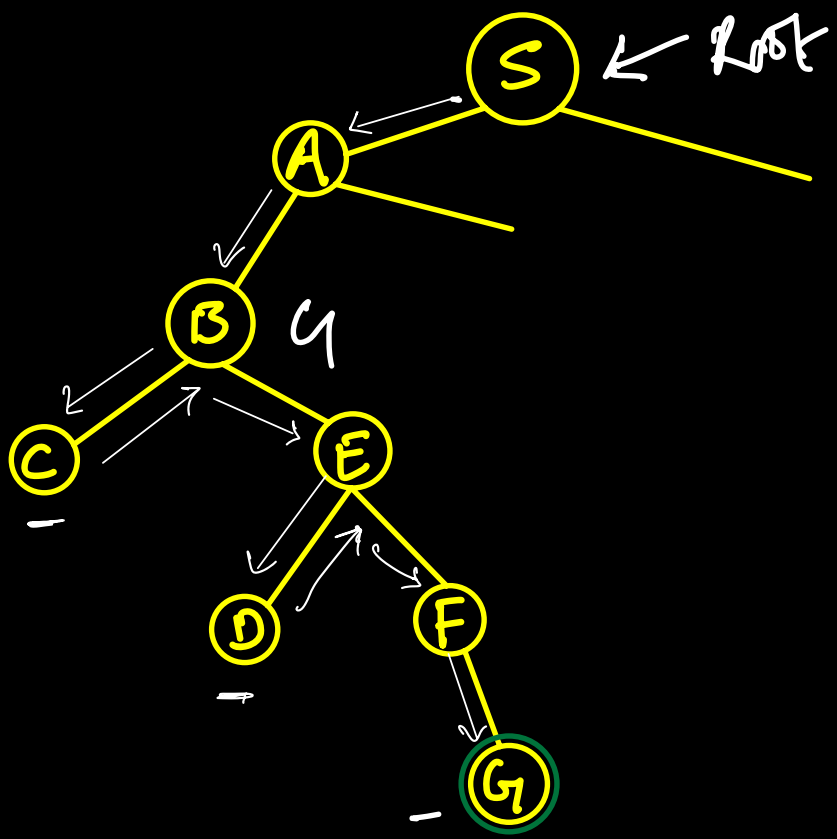
# Depth First Search:

- Looks for the goal state/node among all the successor/children of the current node before moving towards the siblings of the node.

- Fringe is implemented as LIFO.



## Evaluation:

Complete: Does it always find sol$^{n}$ if one exist?

→ NO, it may not find the solution in case if search space is infinite & contains loops.

optimal / admissible:

No, it may not be optimal because:

- It expand deepest node first.
- If it expand entire sub-tree even right contains goal node at some depth 'd'.

## Time complexity:

- let d be the max depth of search tree.
- Root has b successor and each node at each level has again b successor.
- Worst case expand all the node except node at depth 'd'
- Total node generated:

$$b + b^2 + b^3 + b^4 + \cdots + b^d = O(b^d)$$

## Space complexity:

- DFS need to store only single path from root to leaf node

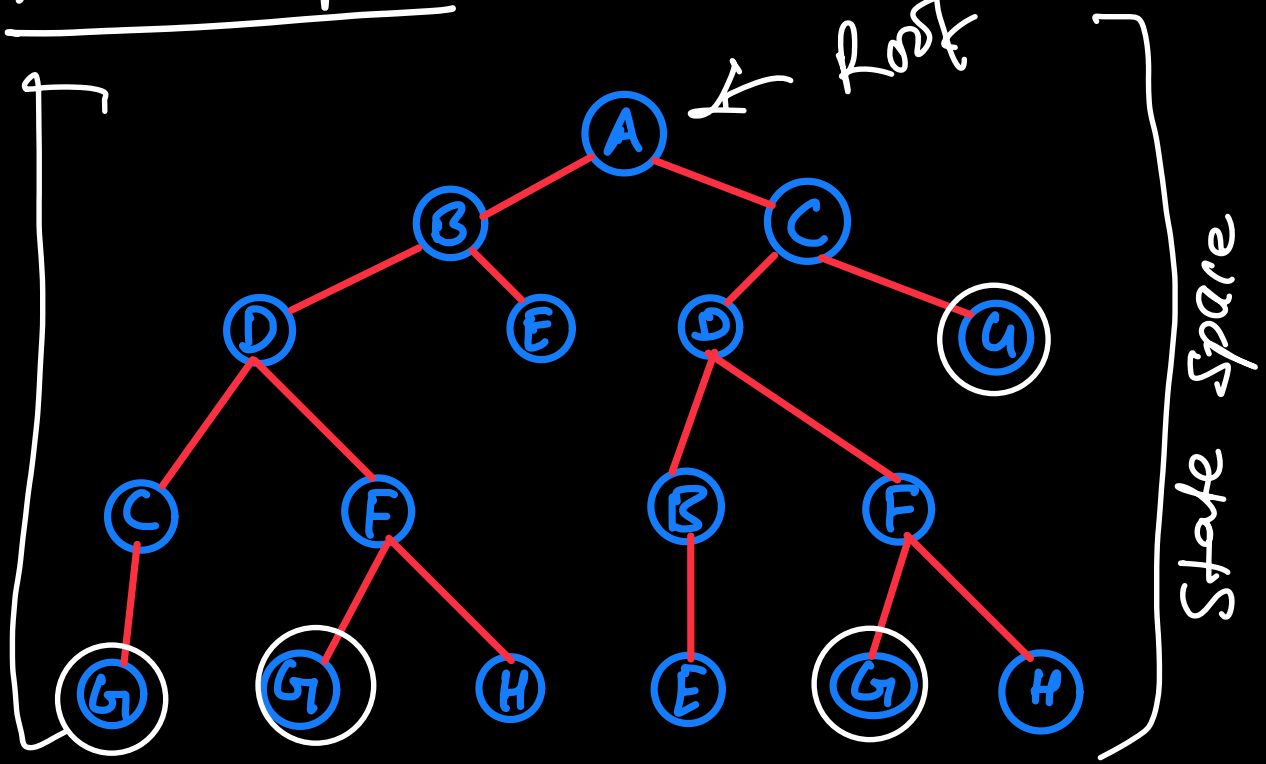
- It also stores remaining unexpanded sibling nodes for each node on the path.

- Once a node has been expanded it can be remove from a memory as soon as its descendents have been fully explored.

- So, for state space with branching factor 'b' & depth 'd', it stores of only:

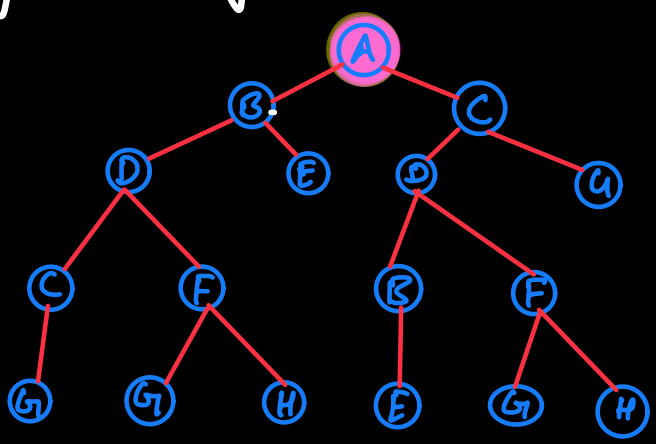
$O(bd)$ nodes

For example:




Step 1:

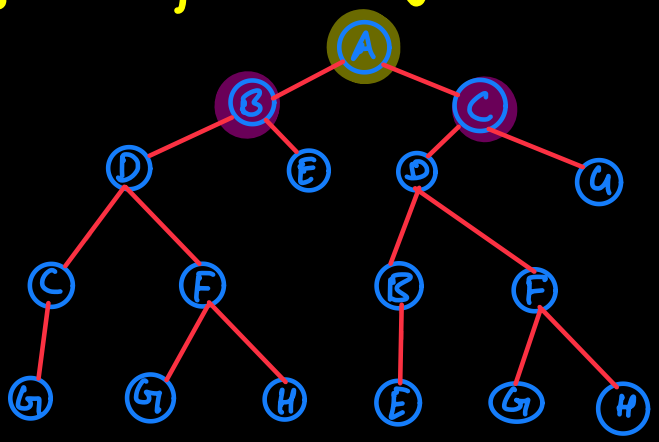
Initially, Fringe contains only node A.



Fringe | A

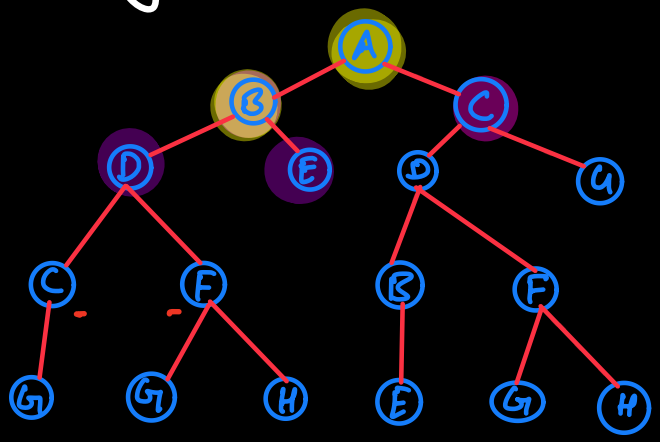
LIFO Approach

Step 2: Remove A from fringe.
Expand A & put its successor in font of fringe.



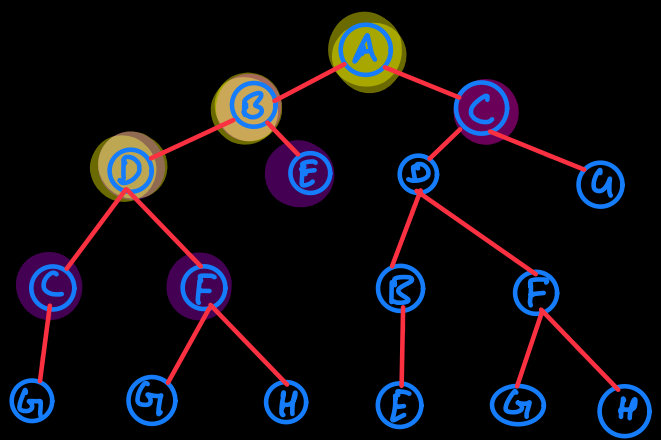

Fringe | B C |

Step 3: Node B is removed & its successor are placed in font of fringe.



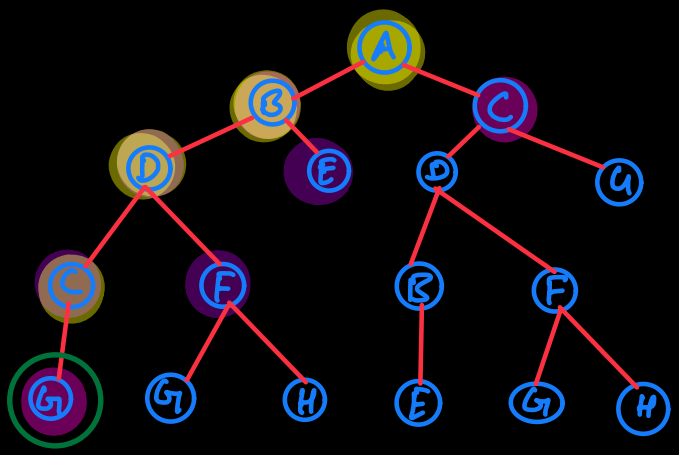
Fringe | D E C

Step 4: Now node D is removed from the fringe & its successor are push in front of fringe.




Fringe | C F E C

Step 5: Remove node/state C from the fringe and its child U is pushed in front of fringe.




Fringe | G | F | E | C |

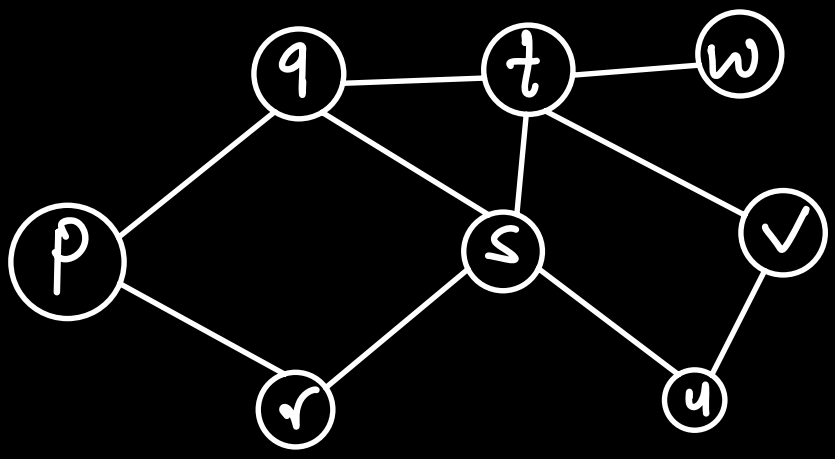
Step 6:

Here, Node G is expanded and found to be goal node. Finally, algorithm terminates reporting solution path

A B D C G

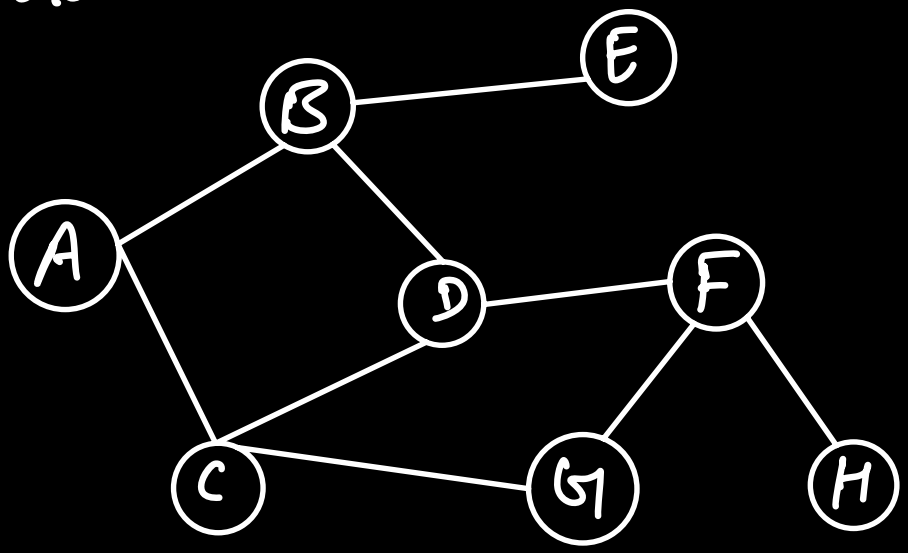
# Practice:

1.




start state: P &
goal state: V

2.




start node: A
goal node: G1