

**Uninformed search** does not use additional information to guide the search process. Instead, these algorithms explore the search space in a systematic, but blind, manner without considering the cost of reaching the goal or the likelihood of finding a solution.

Fringe: nodes that need to be explored.

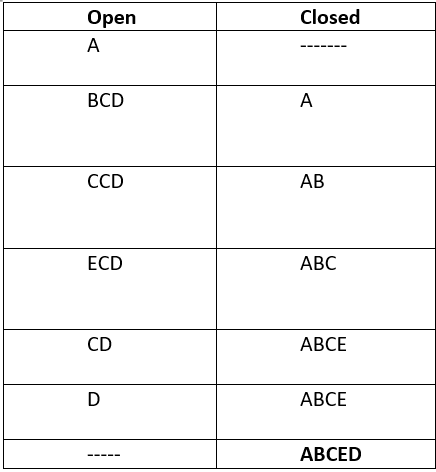
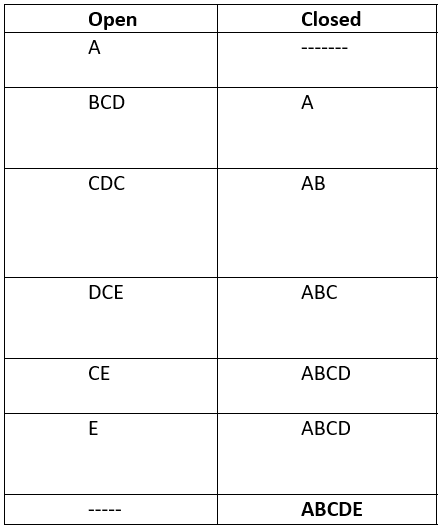
**General Algo for Graph Search:-**

1. Initialize "fringe" as a list containing the initial state.

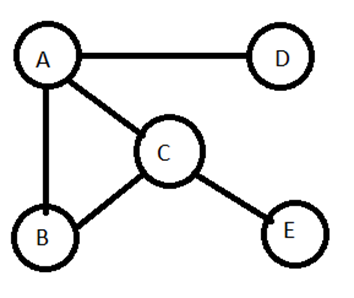
2. Initialize "closed" as an empty set.

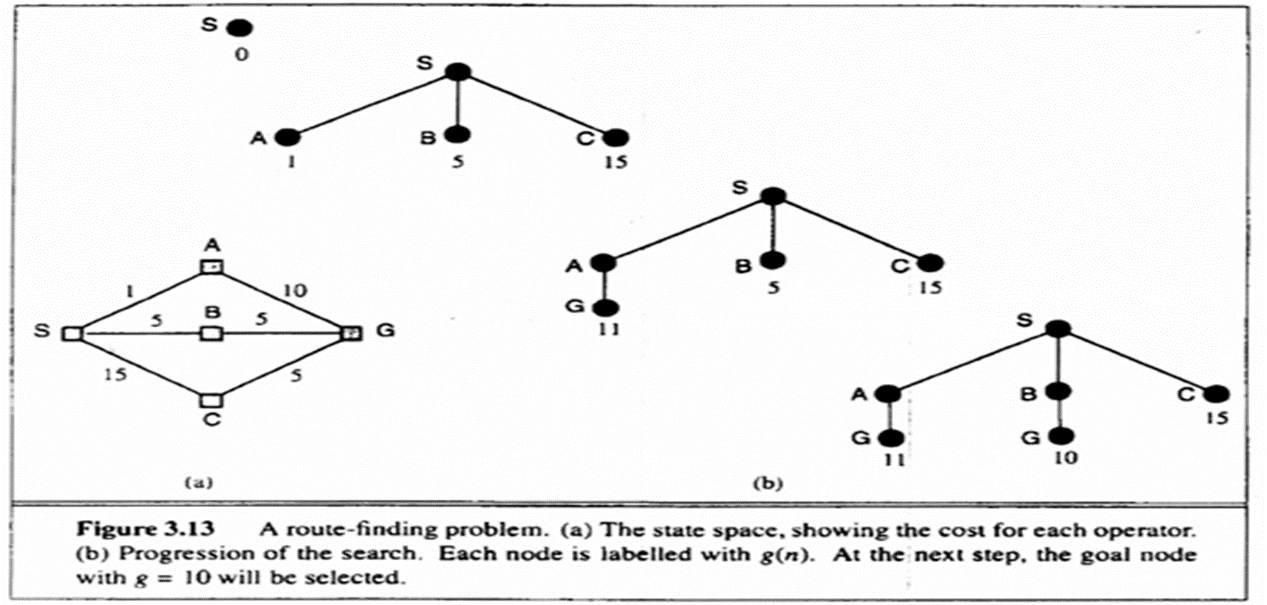
3. Loop until a solution is found or the "fringe" is empty:

* If the "fringe" is empty, return failure.
* Take the first node from the "fringe" and assign it to "Node."
* If "Node" is the goal state, return the path from the initial state to "Node."
* Otherwise, add "Node" to the set of closed nodes.
* Generate all successor nodes of "Node".
* For each successor node "m" in the set of successors:
  + If "m" is not in the set of closed nodes, add it to the fringe.

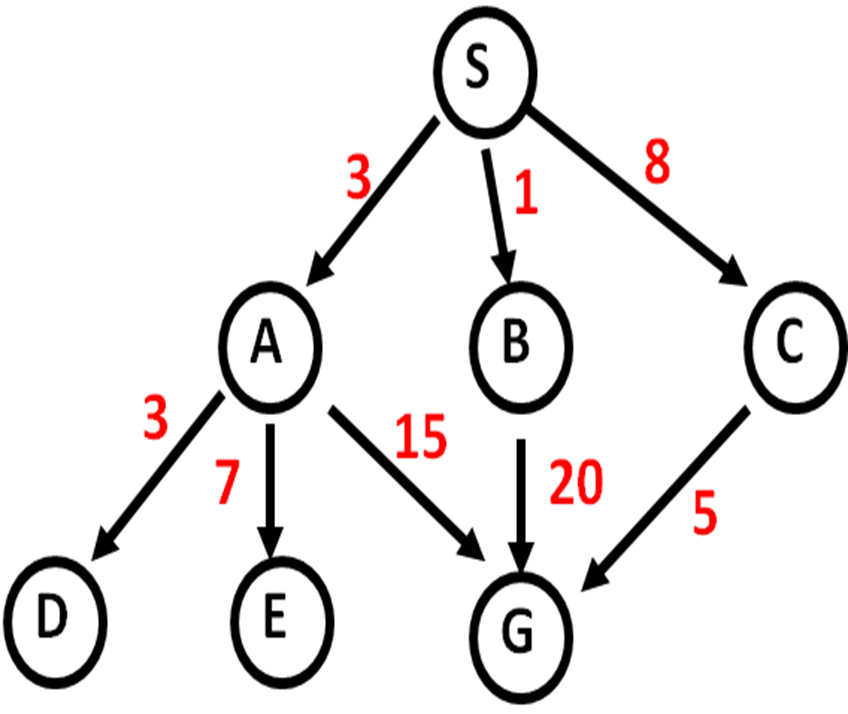
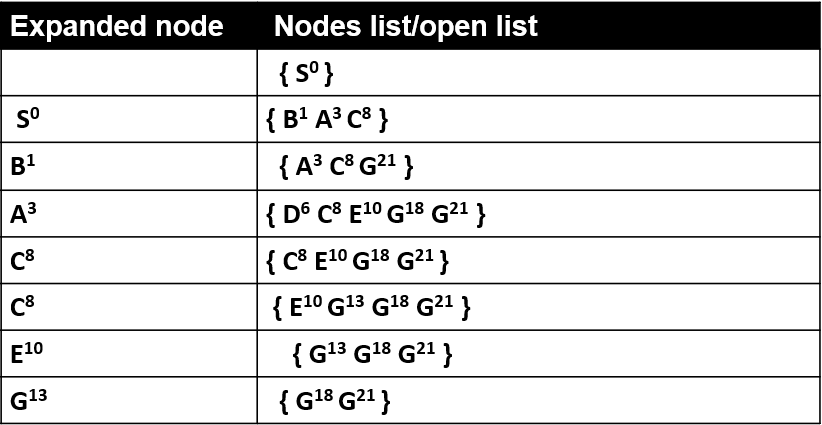
4. Loop End

**1. BFS:** explores nodes level by level

**2. DFS:** explores nodes depth by depth.

**Uniform Cost Search:-**

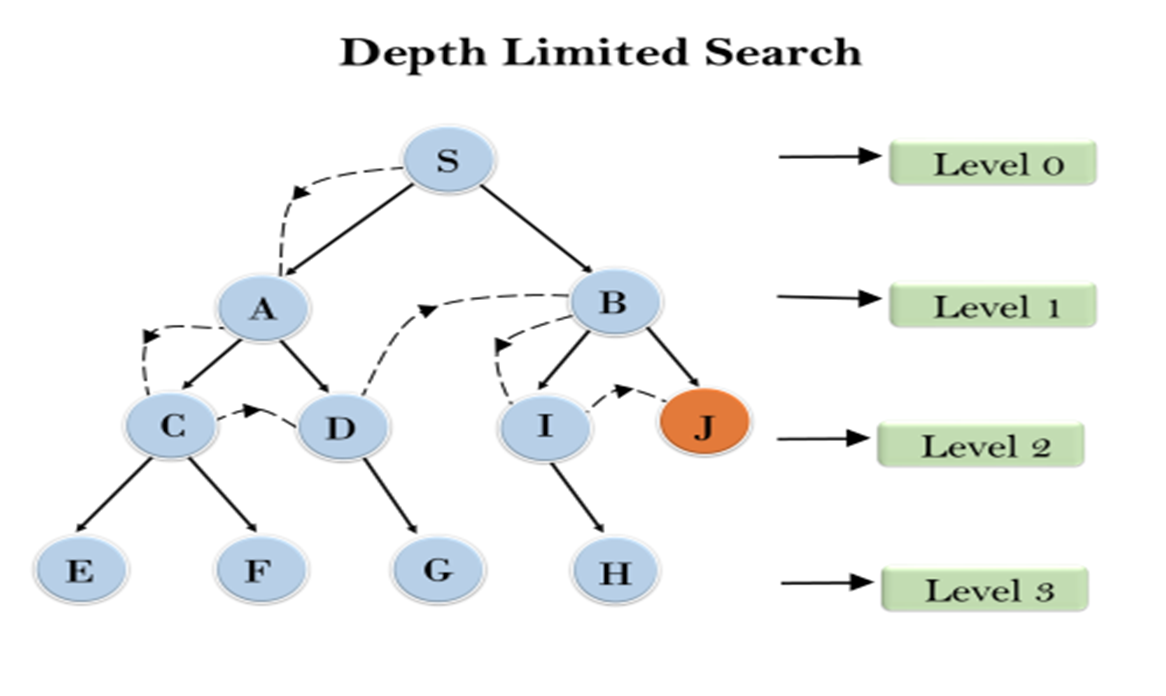
Find the path to the goal node with the lowest cumulative cost.

Uses priority queue to explore nodes based on their cumulative path costs. Expands lowest cost node from the fringe, ensuring lower-cost paths are explored first. It ends when a goal node is selected for expansion, having found the lowest cost path.

Solution path found is S C G, cost 13  
Number of nodes expanded (including goal node) = 7

UCS is effective for finding optimal solutions but can be inefficient in terms of time and space complexity, especially in scenarios with high branching factors and large solution costs.

**Depth Limited Search:-**

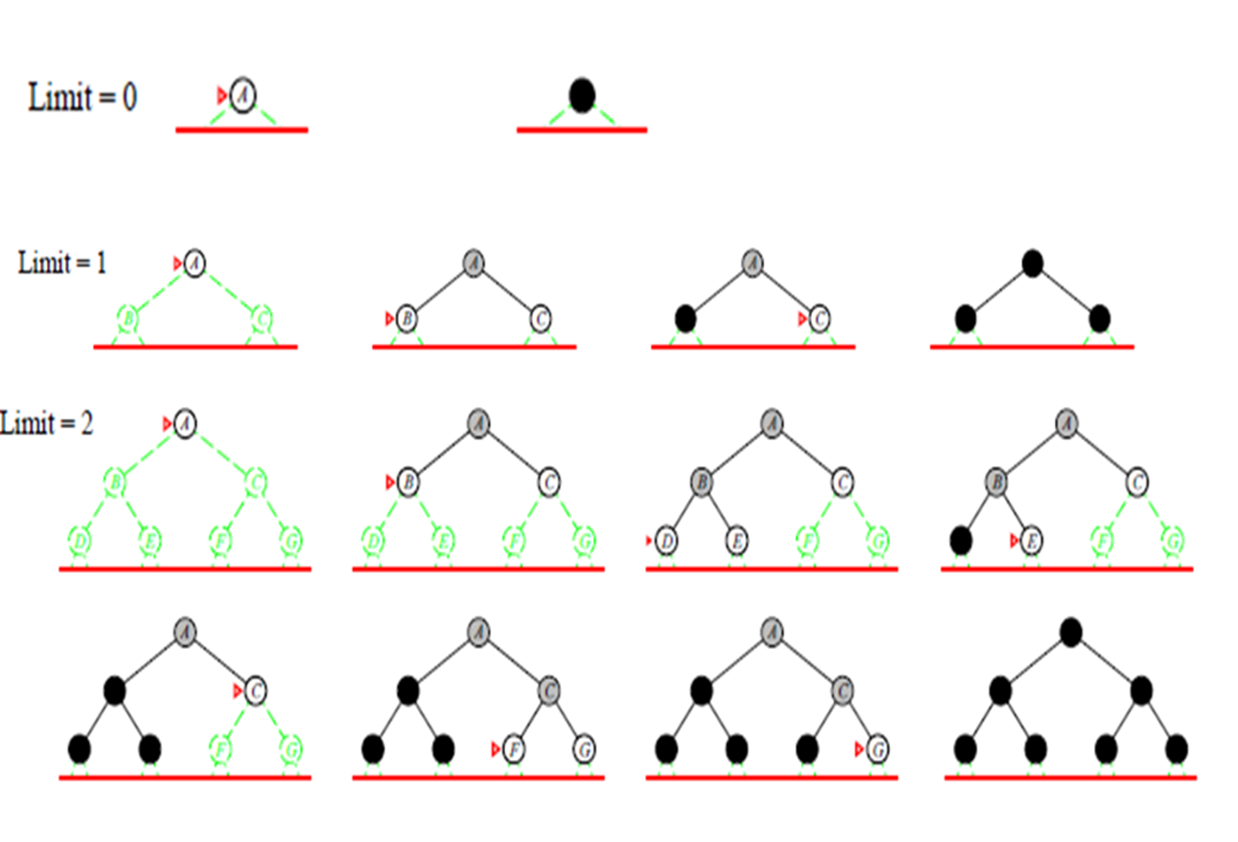
Depth-limited search is a variant of depth-first search that addresses the issue of infinite paths by setting a maximum depth limit on the search. Nodes beyond the depth limit are treated as having no successor nodes, preventing the algorithm from exploring infinitely deep paths.

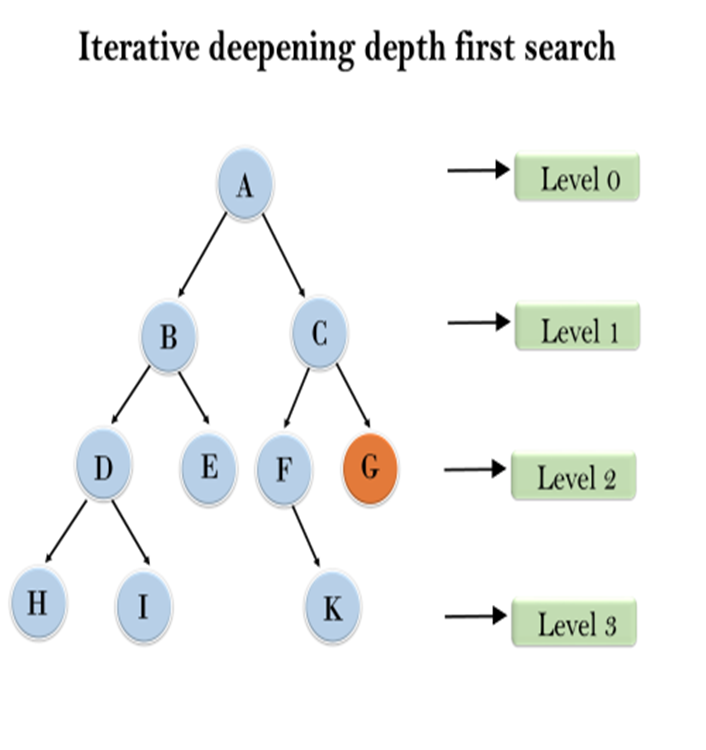
It provides a balance between memory efficiency and search completeness. However, it sacrifices completeness & optimality compared to other search algorithms, making it suitable for certain problem domains where memory constraints outweigh the need for optimality.

<- Depth: level 2

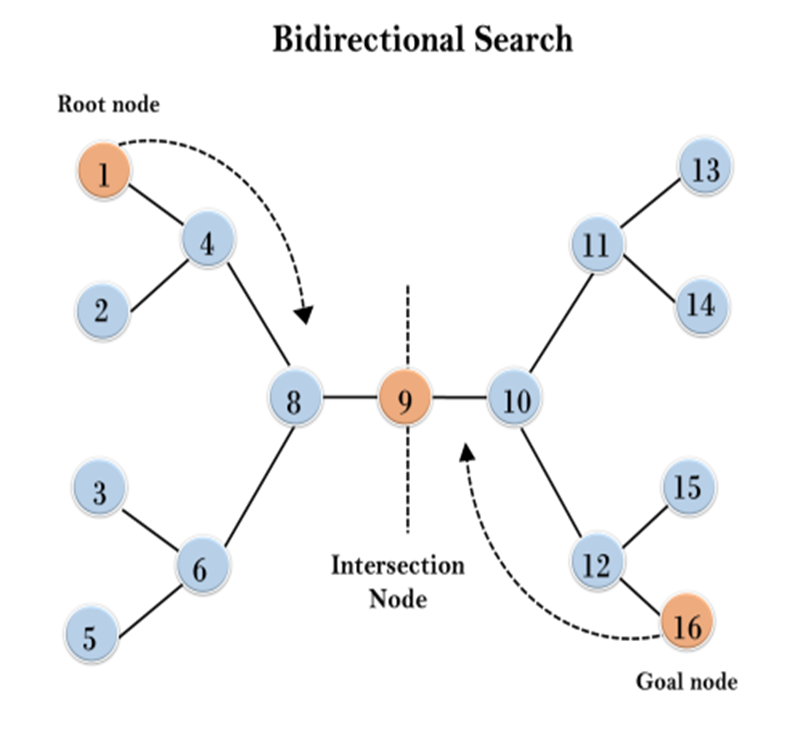
**Iterative Deepening:-**

Its complete because it explores all nodes within the depth limit and iteratively increases the depth limit until the goal is found.(Iteration 1 for lvl 1, Itr 2 for lvl 2, etc.)





1'st Iteration-----> A  
2'nd Iteration----> A, B, C  
3'rd Iteration------>A, B, D, E, C, F, G  
In 3rd itr, the algo will find goal node.



**Bidirectional Search:-**  
It involves running two simultaneous searches, one from the initial state (forward-search) and the other from the goal node (backward-search), with the aim of meeting in the middle.

The search stops when the two graphs intersect each other.

Bidirectional search is suitable when both the initial and goal states are unique and completely defined, and the branching factor is the same in both directions.  
Powerful search algo that can efficiently find optimal solutions by simultaneously exploring from both the initial and goal states. It may consume significant memory resources.

