Uniform Cost Search

- Uniform-cost search is a searching algorithm used for traversing a weighted tree or graph.
- This algorithm comes into play when a different cost is available for each edge.
- The primary goal of the uniform-cost search is to find a path to the goal node which has the lowest cumulative cost.
- Uniform-cost search expands nodes according to their path costs form the root node.
- It can be used to solve any graph/tree where the optimal cost is in demand.
- A uniform-cost search algorithm is implemented by the priority queue.
- It gives maximum priority to the lowest cumulative cost.
- Uniform cost search is equivalent to BFS algorithm if the path cost of all edges is the same.

Uniform Cost Search

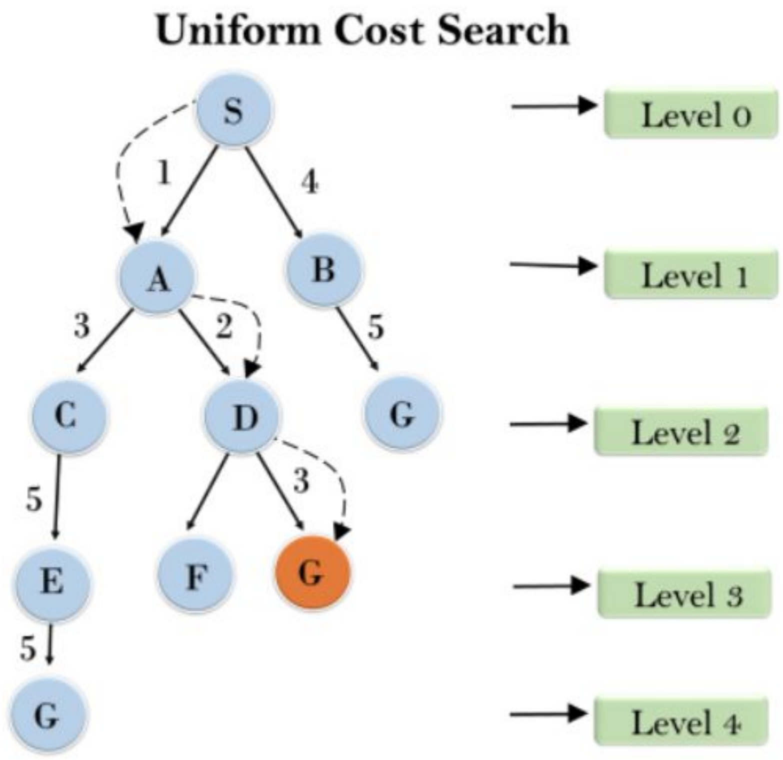
Advantages:

- Uniform cost search is optimal because at every state the path with the least cost is chosen.

Disadvantages:

- It does not care about the number of steps involved in searching and only concerned about path cost. Due to which this algorithm may be stuck in an infinite loop.

Uniform Cost Search



Completeness: Uniform-cost search is complete

Time Complexity: O (b^d)

Space Complexity: O (b^d)

Optimal:

Uniform-cost search is always optimal as it only selects a path with the lowest path cost.

Iterative Deepening DFS

- The iterative deepening algorithm is a combination of DFS and BFS algorithms.
- Finds out the best depth limit and does it by gradually increasing the limit until a goal is found.
- Performs depth-first search up to a certain "depth limit", and it keeps increasing the depth limit after each iteration until the goal node is found.
- Combines the benefits of Breadth-first search's fast search and depth-first search's memory efficiency.
- The iterative search algorithm is useful uninformed search when search space is large, and depth of goal node is unknown.

Iterative Deepening DFS

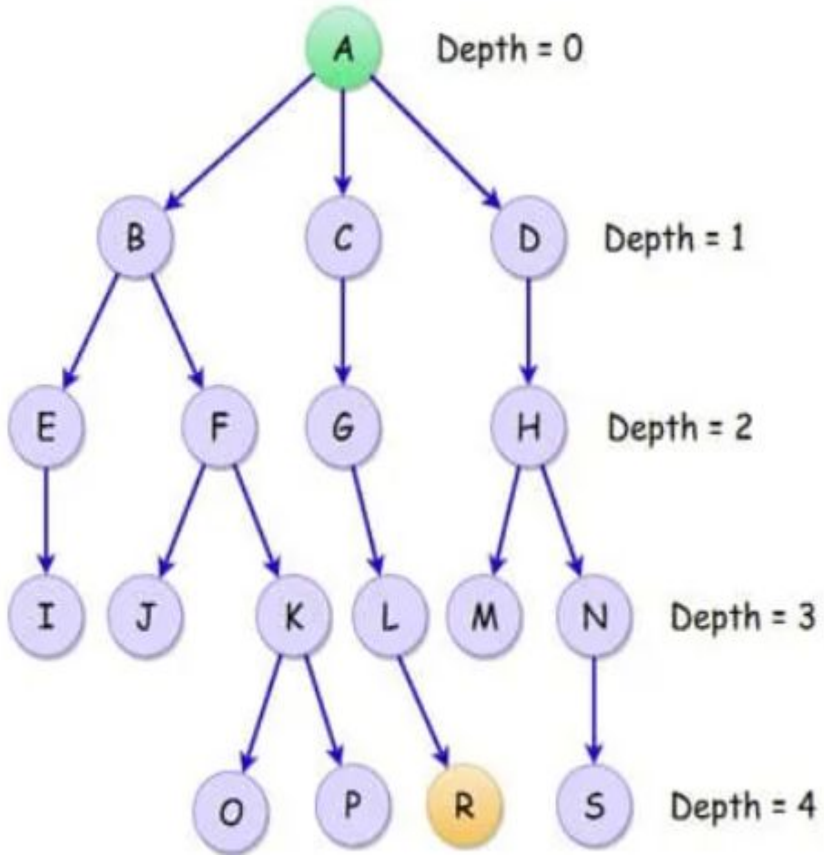
Advantages:

- It combines the benefits of BFS and DFS search algorithm in terms of fast search and memory efficiency.

Disadvantages:

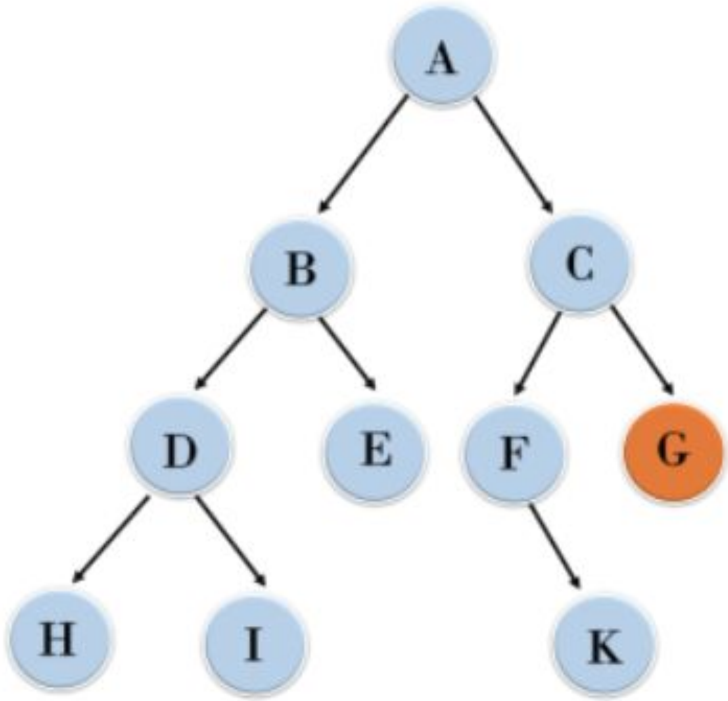
- The main drawback of IDDFS is that it repeats all the work of the previous phase.

Iterative Deepening DFS



DEPTH LIMITS	IDDFS
0	A
1	A B C D
2	A B E F C G D H
3	A B E I F J K C G L D H M N
4	A B E I F J K O P C G L R D H M N S

Iterative Deepening DFS



1'st Iteration-----> A

2'nd Iteration----> A, B, C

3'rd Iteration------>A, B, D, E, C, F, G

4'th Iteration------>A, B, D, H, I, E, C, F, K, G

In the fourth iteration, the algorithm will find the goal node.

Iterative Deepening DFS

Completeness:

This algorithm is complete is if the branching factor is finite.

Time Complexity:

Let's suppose b is the branching factor and depth is d then the worst-case time complexity is **$O(b^d)$**

Space Complexity:

The space complexity of IDDFS will be **$O(bd)$**.

Optimal:

IDDFS algorithm is optimal if path cost is a non- decreasing function of the depth of the node.

	Time Complexity	Space Complexity	When to Use ?
DFS	$O(b^d)$	$O(d)$	=> Don't care if the answer is closest to the starting vertex/root. => When graph/tree is not very big/infinite.
BFS	$O(b^d)$	$O(b^d)$	=> When space is not an issue => When we do care/want the closest answer to the root.
IDDFS	$O(b^d)$	$O(bd)$	=> You want a BFS, you don't have enough memory, and somewhat slower performance is accepted. In short, you want a BFS + DFS.

Bidirectional Search Algorithm

- Bidirectional search algorithm runs two simultaneous searches
- One form initial state called as forward-search
- Other from goal node called as backward-search, to find the goal node.
- Bidirectional search replaces one single search graph with two small subgraphs in which one starts the search from an initial vertex and other starts from goal vertex.
- The search stops when these two graphs intersect each other.

Bidirectional Search Algorithm

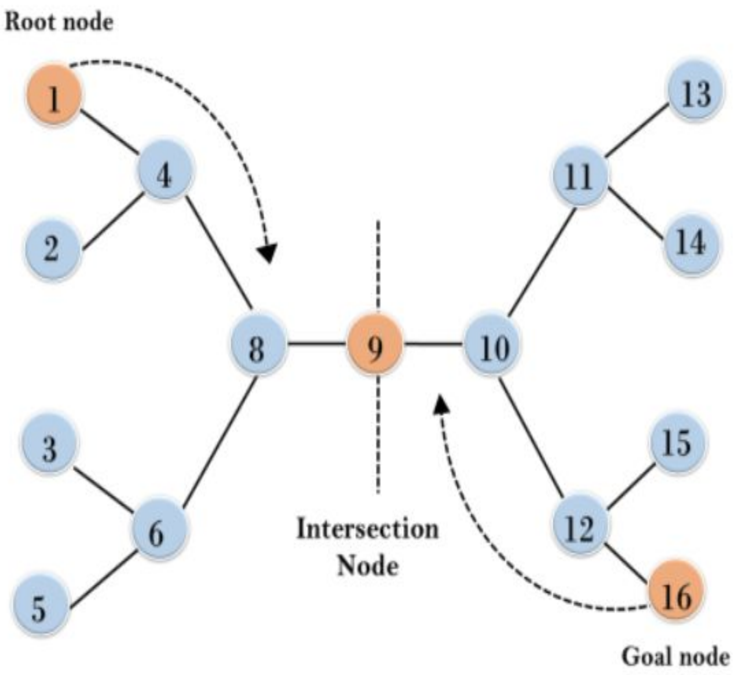
Advantages:

- Bidirectional search is fast.
- Bidirectional search requires less memory

Disadvantages:

- Implementation of the bidirectional search tree is difficult.
- In bidirectional search, one should know the goal state in advance.

Bidirectional Search Algorithm



- This algorithm divides one graph/tree into two sub-graphs.
- It starts traversing from node 1 in the forward direction and
- Starts from goal node 16 in the backward direction.
- The algorithm terminates at node 9 where two searches meet.

Bidirectional Search Algorithm

Completeness: Bidirectional Search is complete.

Time Complexity: Time complexity of bidirectional search using BFS is $\mathbf{O(b^{d/2})}$.

Space Complexity: Space complexity of bidirectional search is $\mathbf{O(b^{d/2})}$.

Optimal: Bidirectional search is Optimal.

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Time	b^d	b^d	b^m	b^l	b^d	$b^{d/2}$
Space	b^d	b^d	bm	bl	bd	$b^{d/2}$
Optimal?	Yes	Yes	No	No	Yes	Yes
Complete?	Yes	Yes	No	Yes, if $l > d$	Yes	Yes

Figure 3.18 Evaluation of search strategies. b is the branching factor; d is the depth of solution; m is the maximum depth of the search tree; l is the depth limit.