Uninformed Search Algorithms

Introduction:

Uninformed search is one in which the search systems do not use any clues about the suitable area but it depend on the random nature of search. Nevertheless, they begins the exploration of search space (all possible solutions) synchronously,. The search operation begins from the initial state and providing all possible next steps arrangement until goal is reached. These are mostly the simplest search strategies, but they may not be suitable for complex paths which involve in irrelevant or even irrelevant components. These algorithms are necessary for solving basic tasks or providing simple processing before passing on the data to more advanced search algorithms that incorporate prioritized information.

Following are the various types of uninformed search algorithms:

1. Breadth-first Search
2. Depth-first Search
3. Depth-limited Search
4. Iterative deepening depth-first search
5. Uniform cost search
6. Bidirectional Search

1. Breadth-first Search:

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.

It also helps in finding the shortest path in goal state, since it needs all nodes at the same hierarchical level before making a move to nodes at lower levels.

It is also very easy to comprehend with the help of this we can assign the higher rank among path types.

Disadvantages:

It requires lots of memory since each level of the tree must be saved into memory to expand the next level.

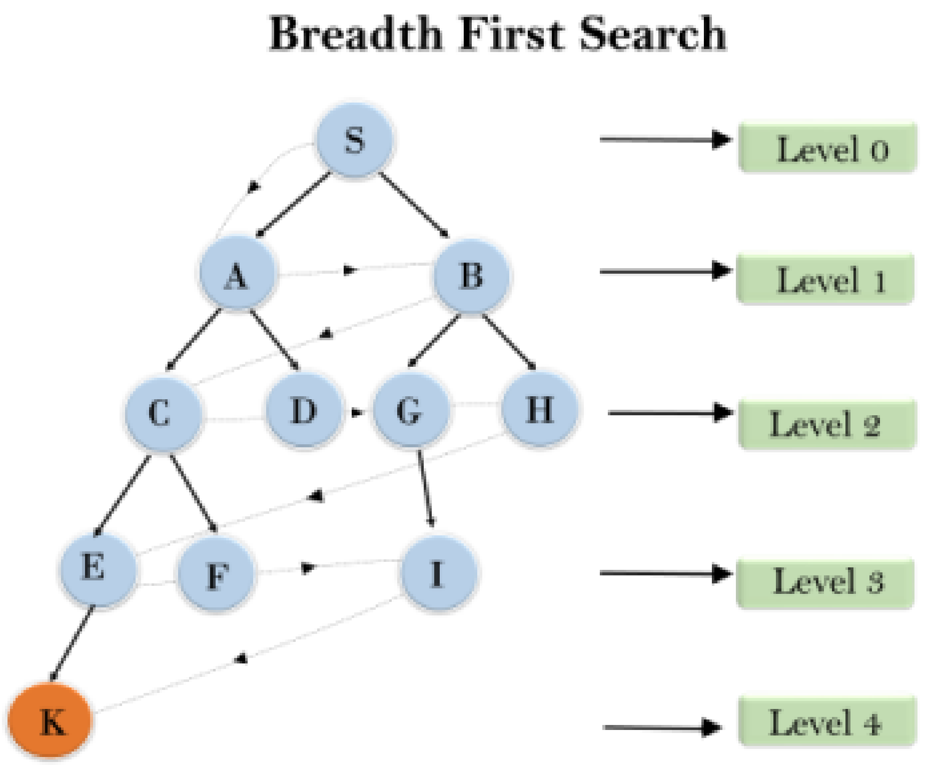
BFS needs lots of time if the solution is far away from the root node.

It can be very inefficient approach for searching through deeply layered spaces, as it needs to thoroughly explore all nodes at each level before moving on to the next

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



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.

$$T(b) = 1+b^2+b^3+\ldots\ldots+b^d = O(b^d)$$

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.

2. Depth-first Search

Depth-first search isa recursive algorithm for traversing a tree or graph data structure.

It is called the depth-first search because it starts from the root node and follows each path to its greatest depth node before moving to the next path.

DFS uses a stack data structure for its implementation.

The process of the DFS algorithm is similar to the BFS algorithm.
Note: Backtracking is an algorithm technique for finding all possible solutions using recursion.

Advantage:

DFS requires very less memory as it only needs to store a stack of the nodes on the path from root node to the current node.

It takes less time to reach to the goal node than BFS algorithm (if it traverses in the right path).

With the help of this we can stores the route which is being tracked in memory to save time as it only needs to keep one at a particular time.

Disadvantage:

There is the possibility that many states keep re-occurring, and there is no guarantee of finding the solution.

DFS algorithm goes for deep down searching and sometime it may go to the infinite loop. The

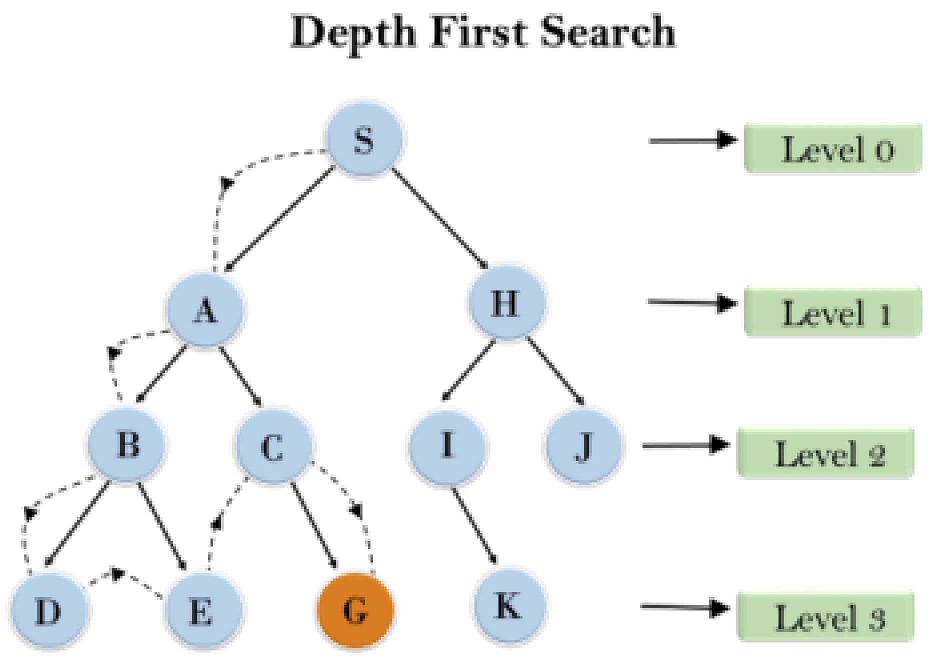
de�pth-first search (DFS) algorithm does not always find the shorte�st path to a solution.

Example:

In the below search tree, we have shown the flow of depth-first search, and it will follow the order

as: Root node--->Left node ----> right node.

It will start searching from root node S, and traverse A, then B, then D and E, after traversing E, it will backtrack the tree as E has no other successor and still goal node is not found. After backtracking it will traverse node C and then G, and here it will terminate as it found goal node.



Completeness: DFS search algorithm is complete within finite state space as it will expand every node within a limited search tree.

Time Complexity: Time complexity of DFS will be equivalent to the node traversed by the algorithm. It is given by:

$T(n)= 1+ n^2 + n^3 +........+ n^m =O(n^m)$

Where, m= maximum depth of any node and this can be much larger than d (Shallowest solution depth)

Space Complexity: DFS algorithm needs to store only single path from the root node, hence space complexity of DFS is equivalent to the size of the fringe set, which is **O(bm)**.

Optimal: DFS search algorithm is non-optimal, as it may generate a large number of steps or high cost to reach to the goal node.

3. Depth-Limited Search Algorithm:

A depth-limited search algorithm is similar to depth-first search with a predetermined limit. Depth-limited search can solve the drawback of the infinite path in the Depth-first search. In this algorithm, the node at the depth limit will treat as it has no successor nodes further.

Depth-limited search can be terminated with two Conditions of failure:

- **Standard failure value:** It indicates that problem does not have any solution.
- **Cutoff failure value:** It defines no solution for the problem within a given depth limit.

Advantages:

Depth-Limited Search will restrict the search depth of the tree, thus, the algorithm will require fewer memory resources than the straight BFS (Breadth-First Search) and IDDFS (Iterative Deepening Depth-First Search). After all, this implies automatic selection of more segments of the search space and the consequent why consumption of the resources. Due to the depth restriction, DLS omits a predicament of holding the entire search tree within memory which contemplatively leaves room for a more memory-efficient vice for solving a particular kind of problems.

When there is a leaf node depth which is as large as the highest level allowed, do not describe its children, and then discard it from the stack.

Depth-Limited Search does not explain the infinite loops which can arise in classical when there are cycles in graph of cities.

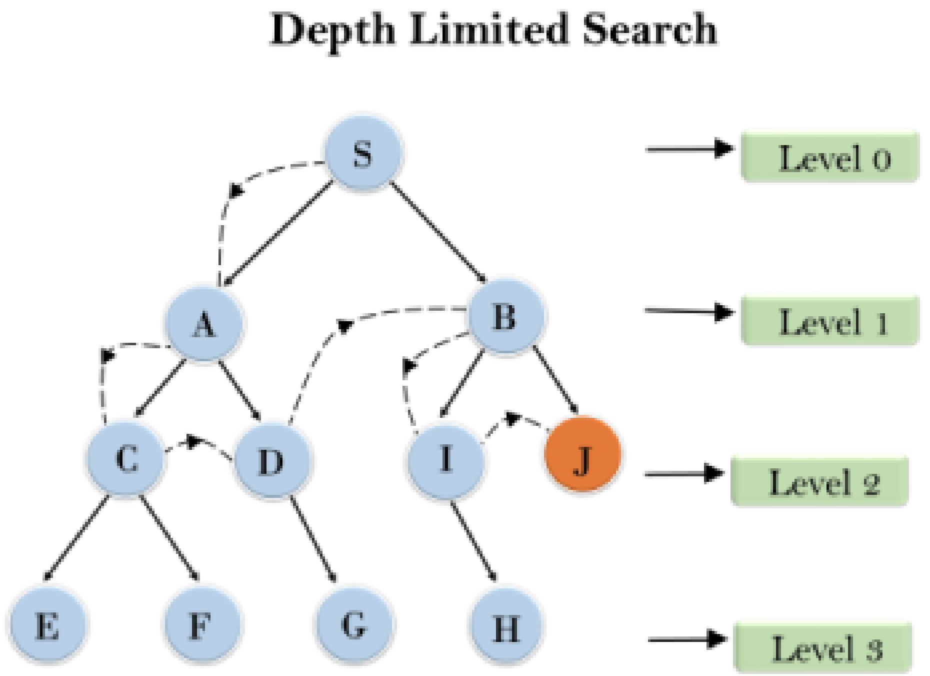
Disadvantages:

Depth-limited search also has a disadvantage of incompleteness.

It may not be optimal if the problem has more than one solution.

The effectiveness of the Depth-Limited Search (DLS) algorithm is largely dependent on the depth limit specified. If the depth limit is set too low, the algorithm may fail to find the solution altogether.

Example:



Completeness: DLS search algorithm is complete if the solution is above the depth-limit.

Time Complexity: Time complexity of DLS algorithm is **$O(b^{\ell})$** where b is the branching factor of the search tree, and l is the depth limit.

Space Complexity: Space complexity of DLS algorithm is O**(b×ℓ)** where **b** is the branching factor of the search tree, and **l** is the depth limit.

Optimal: Depth-limited search can be viewed as a special case of DFS, and it is also not optimal even if ℓ>d.

4. Uniform-cost Search Algorithm:

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.

Advantages:

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

It is an efficient when the edge weights are small, as it explores the paths in an order that ensures that the shortest path is found early.

It's a fundamental search method that is not overly complex, making it accessible for many users.

It is a type of comprehensive algorithm that will find a solution if one exists. This means the algorithm is complete, ensuring it can locate a solution whenever a viable one is available. The algorithm covers all the necessary steps to arrive at a resolution.

Disadvantages:

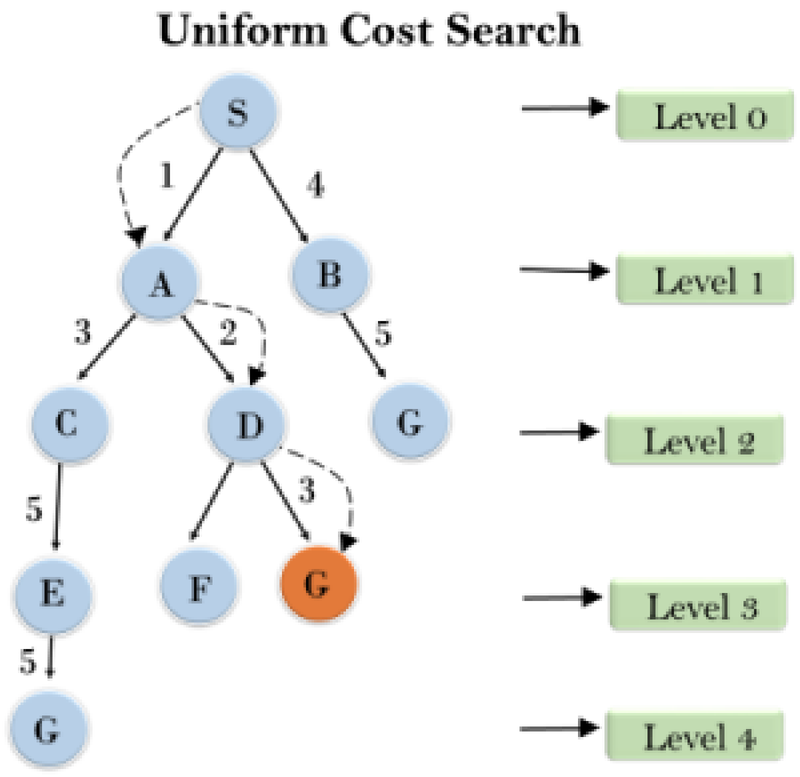
It does not care about the number of steps involve in searching and only concerned about path cost. Due to which this algorithm may be stuck in an infinite loop.

When in operation, UCS shall know all the edge weights to start off the search.

This search holds constant the list of the nodes that it has already discovered in a priority queue. Such is a much weightier thing if you have a large graph. Algorithm allocates the memory by storing the path sequence of prioritizes, which can be memory intensive as the graph gets larger.With the help of Uniform cost search we can end up with the problem if the graph has edge's cycles with smaller cost than that of the shortest path.

The Uniform cost search will keep deploying priority queue so that the paths explored can be stored in any case as the graph size can be even bigger that can eventually result in too much memory being used.

Example:



Completeness:

Uniform-cost search is complete, such as if there is a solution, UCS will find it.

Time Complexity:

Let C* **is Cost of the optimal solution**, and **ε** is each step to get closer to the goal node. Then the number of steps is = C*/ε+1. Here we have taken +1, as we start from state 0 and end to C*/ε.

Hence, the worst-case time complexity of Uniform-cost search is**$O(b^{1 + [C^*/ε]})$/**.

Space Complexity:

The same logic is for space complexity so, the worst-case space complexity of Uniform-cost search is **$O(b^{1 + [C^*/ε]})$**.

Optimal:

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

cost.

5. Iterative deepeningdepth-first Search:

The iterative deepening algorithm is a combination of DFS and BFS algorithms. This search algorithm finds out the best depth limit and does it by gradually increasing the limit until a goal is found.

This algorithm 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.

This Search algorithm 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.

Here are the steps for Iterative deepening depth first search algorithm:

Set the depth limit to 0.

Perform DFS to the depth limit.

If the goal state is found, return it.

If the goal state is not found and the maximum depth has not been reached, increment the depth limit and repeat steps 2-4.

If the goal state is not found and the maximum depth has been reached, terminate the search and return failure.

Advantages:

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

It is a type of straightforward which is used to put into practice since it builds upon the conventional depth-first search algorithm.

It is a type ofsearch algorithm which provides guarantees to find the optimal solution, as long as the cost of each edge in the search space is the same.

It is a type ofcomplete algorithm, and the meaning of this is it will always find a solution if one exists.

The Iterative Deepening Depth-First Search (IDDFS) algorithm uses less memory compared to Breadth-First Search (BFS) because it only stores the current path in memory, rather than the entire search tree.

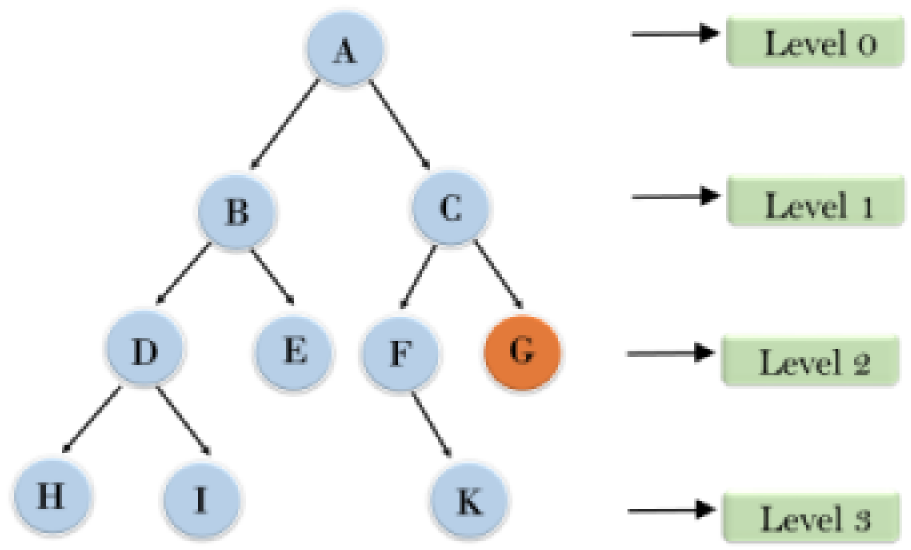
Disadvantages:

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

Example:

Following tree structure is showing the iterative deepening depth-first search. IDDFS algorithm performs various iterations until it does not find the goal node. The iteration performed by the algorithm is given as:

Iterative deepening depth first search



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.

Completeness:

This algorithm is complete is ifthe 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.

6. Bidirectional Search Algorithm:

Bidirectional search algorithm runs two simultaneous searches, one form initial state called as forward-search and 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 can use search techniques such as BFS, DFS, DLS, etc.

Advantages:

Bidirectional search is fast.

Bidirectional search requires less memory

The graph can be extremely helpful when it is very large in size and there is no way to make it smaller. In such cases, using this tool becomes particularly useful.

The cost of expanding nodes can be high in certain cases. In such scenarios, using this approach can help reduce the number of nodes that need to be expanded.

Disadvantages:

Implementation of the bidirectional search tree is difficult.

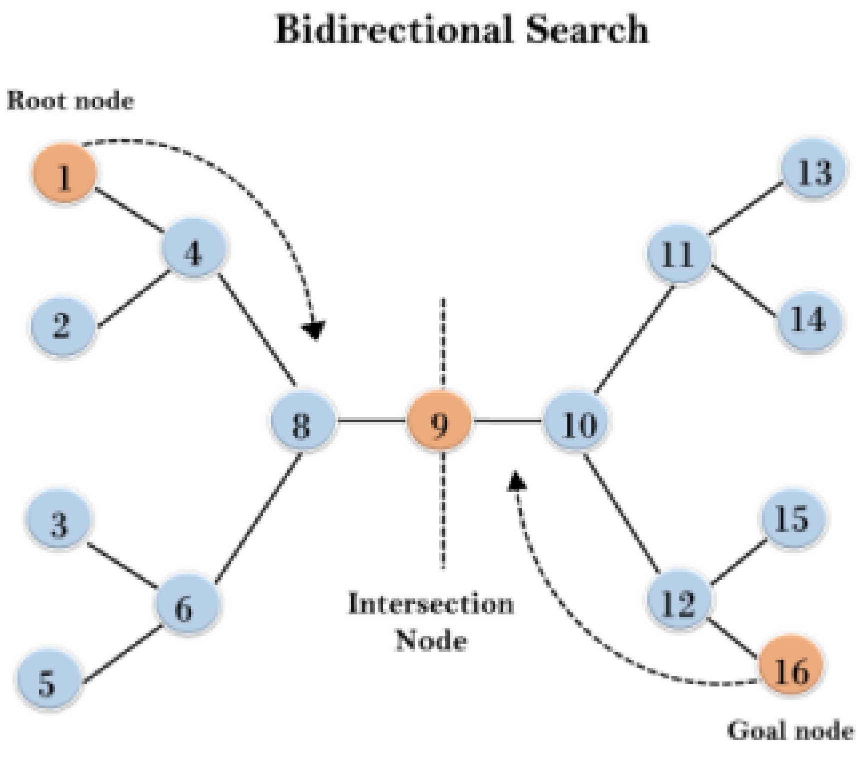
In bidirectional search, one should know the goal state in advance.

Finding an efficient way to check if a match exists between search trees can be tricky, which can increase the time it takes to complete the task.

Example:

In the below search tree, bidirectional search algorithm is applied. 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.



Completeness: Bidirectional Search is complete if we use BFS in both searches.

d

Time Complexity: Time complexity of bidirectional search using BFS is **O(b)**.

Space Complexity: Space complexity of bidirectional search is **$O(b^{d})$**.
Optimal: Bidirectional search is Optimal.