|  |
| --- |
| Close-up image showing the leaf-sides of two oversized books side-by-side on a bookshelf, with additional books in soft focus background |
| ASSIGNMENT 1  122012173027  Bsc (Artifical Intelligence) – II YEAR |
| |  |  |  | | --- | --- | --- | | SUBASH R | 9/4/23 | XAI302 | |

**TABLE OF CONTENTS**

[Depth First Search (DFS) 2](#_Toc144481747)

[Breadth-First Search (BFS) 3](#_Toc144481748)

[Uniform Cost Search Algorithm (UCS) 4](#_Toc144481749)

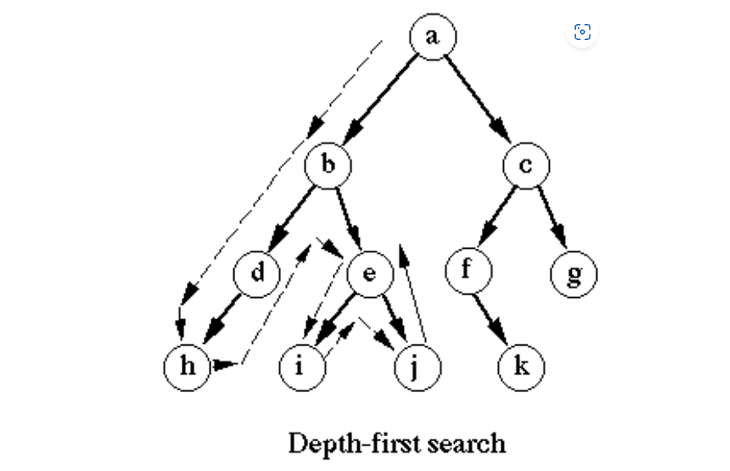
[Depth Limited Search (DLS) 6](#_Toc144481750)

[Iterative Deepening Depth First Search (IDDFS) 10](#_Toc144481751)

[Bidirectional Search (BS) 11](#_Toc144481752)

# Depth First Search (DFS)

It is a search algorithm where the search tree will be traversed from the root node. It will be traversing, searching for a key at the leaf of a particular branch. If the key is not found, the searcher retraces its steps back (backtracking) to the point from where the other branch was left unexplored, and the same procedure is repeated for that other branch.



**Advantages**

* DFS requires very little memory as it only needs to store a stack of the nodes on the path from the root node to the current node.
* It takes less time to reach the goal node than the BFS algorithm [which is explained later](if it traverses in the right path).

**Disadvantages**

* There is the possibility that many states keep reoccurring, and there is no guarantee of finding a solution.
* The DFS algorithm goes for deep-down searching, and sometimes it may go to the infinite loop

# Breadth-First Search (BFS)

This is another graph search algorithm in AI that traverses breadthwise to search for the goal in a tree. It begins searching from the root node and expands the successor node before expanding further along breadthwise and traversing those nodes rather than searching depth-wise.

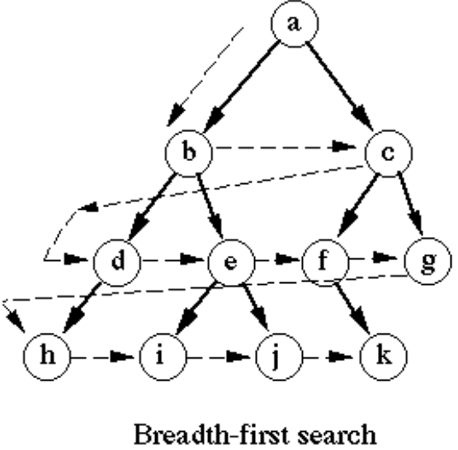
An example of a BFS Algorithm. It starts from the root node A and then traverses node B. Till this step, it is the same as DFS. But here, instead of expanding the children of B as in the case of DFS, we expand the other child of A, i.e., node C because of BFS, and then move to the next level and traverse from D to G and then from H to K in this typical example. To traverse here, we have only taken into consideration the lexicographical order. This is how the BFS Algorithm is implemented.

**Advantages**

* BFS will provide a solution if any solution exists.
* If there is more than one solution for a given problem, then BFS will provide the minimal solution which requires the least number of steps.

**Disadvantages**

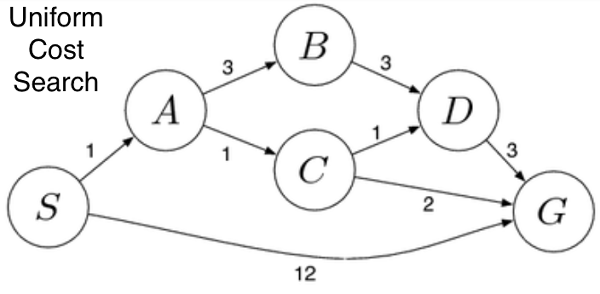
* It requires lots of memory since each level of the tree must be saved in memory to expand to the next level.
* BFS needs lots of time if the solution is far away from the root node.



# Uniform Cost Search Algorithm (UCS)

Uniform Cost Search (UCS) is a graph traversal and search algorithm used in the field of artificial intelligence and computer science. UCS is an informed search algorithm that explores a graph by gradually expanding nodes starting from the initial node and moving towards the goal node while considering the cost associated with each edge or step.

This algorithm is mainly used when the step costs are not the same, but we need the optimal solution to the goal state. In such cases, we use Uniform Cost Search to find the goal and the path, including the cumulative cost to expand each node from the root node to the goal node. It does not go depth or breadth. It searches for the next node with the lowest cost, and in the case of the same path cost, let’s consider lexicographical order in our case.



In the above figure, consider S to be the start node and G to be the goal state. From node S we look for a node to expand, and we have nodes A and G, but since it’s a uniform cost search, it’s expanding the node with the lowest step cost, so node A becomes the successor rather than our required goal node G. From A we look at its children nodes B and C. Since C has the lowest step cost, it traverses through node C. Then we look at the successors of C, i.e., D and G. Since the cost to D is low, we expand along with node D. Since D has only one child G which is our required goal state we finally reach the goal state D by implementing UFS Algorithm. If we have traversed this way, definitely our total path cost from S to G is just 6 even after traversing through many nodes rather than going to G directly where the cost is 12 and 6<<12(in terms of step cost). But this may not work with all cases.

**Advantages**

* Uniform cost search is an optimal search method because at every state, the path with the least cost is chosen.

**Disadvantages**

* It does not care about the number of steps or finding the shortest path involved in the search problem, and it is only concerned about path cost. This algorithm may be stuck in an infinite loop.

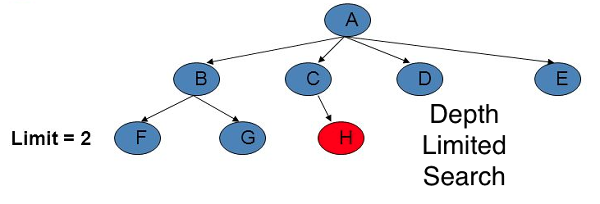
# 

# Depth Limited Search (DLS)

DLS is an uninformed search algorithm. This is similar to DFS but differs only in a few ways. The sad failure of DFS is alleviated by supplying a depth-first search with a predetermined depth limit. That is, nodes at depth are treated as if they have no successors. This approach is called a depth-limited search. The depth limit solves the infinite-path problem. Depth-limited search can be halted in two cases:

Standard Failure Value (SFV): The SFV tells that there is no solution to the problem.

Cutoff Failure Value (CFV): The Cutoff Failure Value tells that there is no solution within the given depth limit.

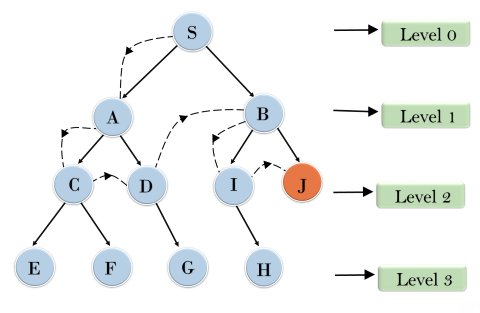


The above figure illustrates the implementation of the DLS algorithm. Node A is at Limit = 0, followed by nodes B, C, D, and E at Limit = 1 and nodes F, G, and H at Limit = 2. Our start state is considered to be node A, and our goal state is node H. To reach node H, we apply DLS. So in the first case, let’s set our limit to 0 and search for the goal.

Since limit 0, the algorithm will assume that there are no children after limit 0 even if nodes exist further. Now, if we implement it, we will traverse only node A as there is only one node in limit 0, which is basically our goal state. If we use SFV, it says there is no solution to the problem at limit 0, whereas LCV says there is no solution for the problem until the set depth limit. Since we could not find the goal, let’s increase our limit to 1 and apply DFS till limit 1, even though there are further nodes after limit 1. But those nodes aren’t expanded as we have set our limit as 1.

Hence nodes A, followed by B, C, D, and E, are expanded in the mentioned order. As in our first case, if we use SFV, it says there is no solution to the problem at limit 1, whereas LCV says there is no solution for the problem until the set depth limit 1. Hence we again increase our limit from 1 to 2 in the notion to find the goal.

Till limit 2, DFS will be implemented from our start node A and its children B, C, D, and E. Then from E, it moves to F, similarly backtracks the path, and explores the unexplored branch where node G is present. It then retraces the path and explores the child of C, i.e., node H, and then we finally reach our goal by applying DLS Algorithm. Suppose we have further successors of node F but only the nodes till limit 2 will be explored as we have limited the depth and have reached the goal state.



his image explains the DLS implementation and could be referred to for better understanding.

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

1. Standard Failure: it indicates that the problem does not have any solutions.
2. Cutoff Failure Value: It defines no solution for the problem within a given depth limit.

**Advantages**

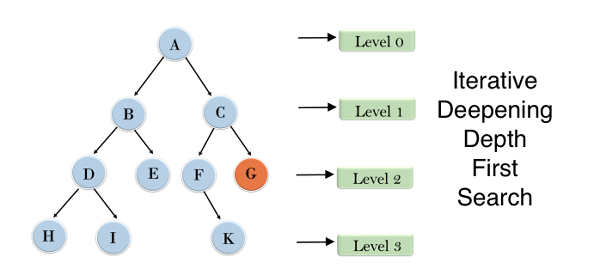
* Depth-limited search is Memory efficient.

**Disadvantages**

* The DLS has disadvantages of completeness and is not optimal if it has more than one goal state.

# Iterative Deepening Depth First Search (IDDFS)

It is a search algorithm that uses the combined power of the BFS and DFS algorithms. It is iterative in nature. It searches for the best depth in each iteration. It performs the Algorithm until it reaches the goal node. The algorithm is set to search until a certain depth and the depth keeps increasing at every iteration until it reaches the goal state.



In the above figure, let’s consider the goal node to be G and the start state to be A. We perform our IDDFS from node A. In the first iteration, it traverses only node A at level 0. Since the goal is not reached, we expand our nodes, go to the next level, i.e., 1 and move to the next iteration. Then in the next iteration, we traverse the node A, B, and C. Even in this iteration, our goal state is not reached, so we expand the node to the next level, i.e., 2, and the nodes are traversed from the start node or the previous iteration and expand the nodes A, B, C, and D, E, F, G. Even though the goal node is traversed, we go through for the next iteration, and the remaining nodes A, B, D, H, I, E, C, F, K, and G(BFS & DFS) too are explored, and we find the goal state in this iteration. This is the implementation of the IDDFS Algorithm.

**Advantages**

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

**Disadvantages**

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

# Bidirectional Search (BS)

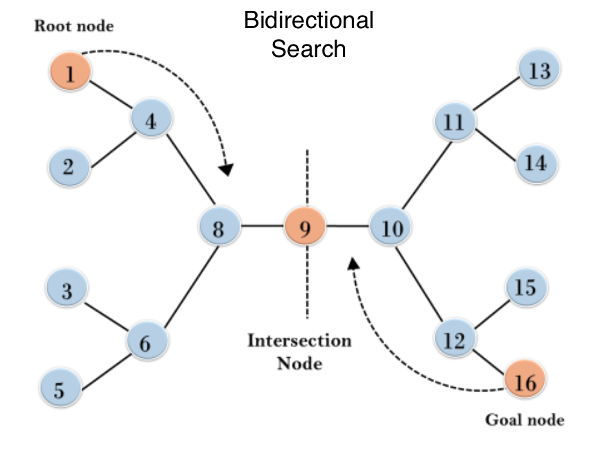
Before moving into bidirectional search, let’s first understand a few terms.

**Forward Search:** Looking in front of the end from the start.

**Backward Search:** Looking from end to the start backward.

So Bidirectional Search, as the name suggests, is a combination of forwarding and backward search. Basically, if the average branching factor going out of node / fan-out, if fan-out is less, prefer forward search. Else if the average branching factor going into a node/fan-in is less (i.e., fan-out is more), prefer backward search. We must traverse the tree from the start node and the goal node, and wherever they meet, the path from the start node to the goal through the intersection is the optimal solution. The BS Algorithm is applicable when generating predecessors is easy in both forward and backward directions, and there exist only 1 or fewer goal states.

This figure provides a clear-cut idea of how BS is executed. We have node 1 as the start/root node and node 16 as the goal node. The algorithm divides the search tree into two sub-trees. So from the start of node 1, we do a forward search, and at the same time, we do a backward search from goal node 16. The forward search traverses nodes 1, 4, 8, and 9, whereas the backward search traverses through nodes 16, 12, 10, and 9. We see that both forward and backward search meets at node 9, called the intersection node. So the total path traced by forwarding search and the path traced by backward search is the optimal solution. This is how the BS Algorithm is implemented.



**Advantages**

* Since BS uses various techniques like DFS, BFS, DLS, etc., it is efficient and requires less memory.

**Disadvantages**

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