ALBUKHARY INTERNATIONAL UNIVERSITY

Artificial Intelligence

LAB FIVE & SIX

Problem Solving Using Search

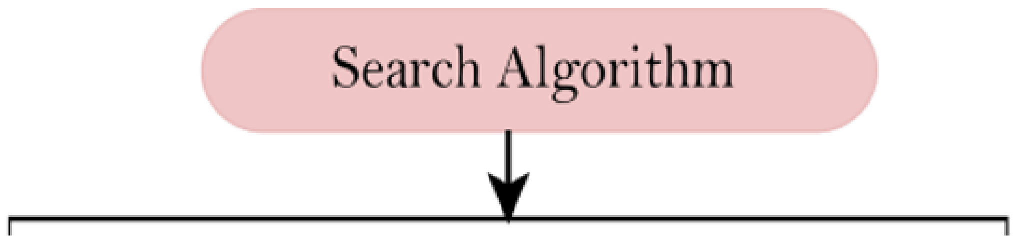
In general, an agent with several immediate options of unknown value can decide what to do by first examining different possible sequences of actions that lead to states of known value, and then choosing the best sequence. This process of looking for such a sequence is called search. A search algorithm takes a problem as input and returns a solution in the form of an action sequence. Once a solution is found, the actions it recommends can be carried out.

Types of search algorithms

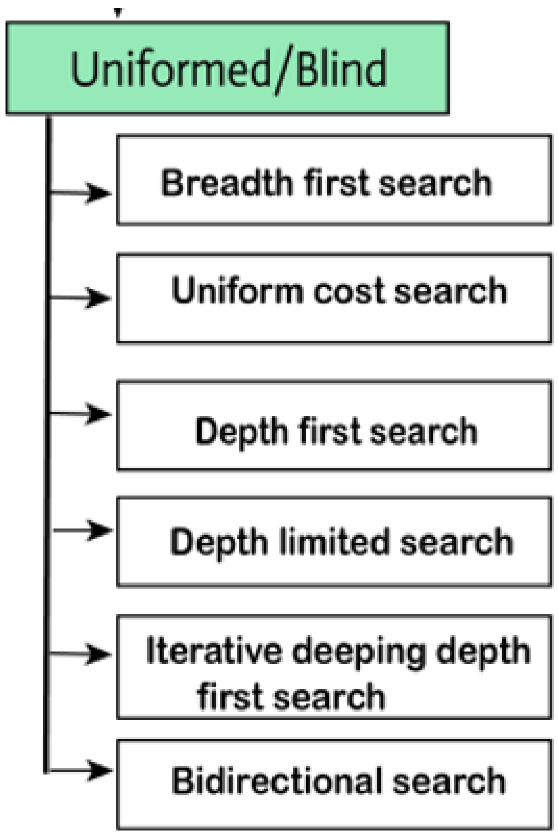
Based on the search problems we can classify the search algorithms into:

1. Uninformed (Blind search) search algorithms
2. Informed search (Heuristic search) algorithms.

As shown in the Figure below:



1. Uninformed Search



1.1 Breadth First Search (BFS)

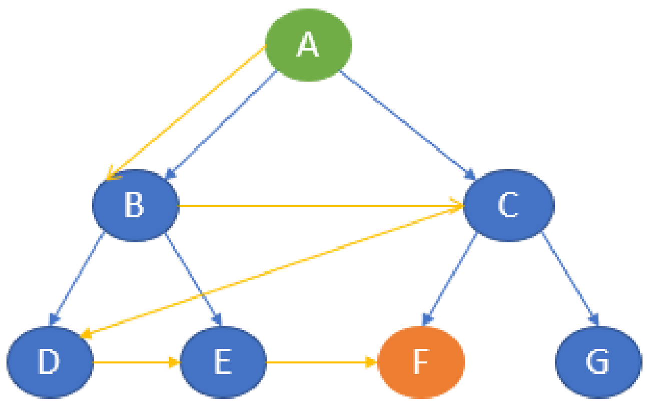
It is of the most common search strategies. It generally starts from the root node and examines the neighbor nodes and then moves to the next level. It uses First-in First-out (FIFO) strategy as it gives the shortest path to achieving the solution.

BFS is used where the given problem is very small and space complexity is not considered.

Here, let's take node A as the start state and node F as the goal state.

The BFS algorithm starts with the start state and then goes to the next level and visits the node until it reaches the goal state.

In this example, it starts from A and then travel to the next level and visits B and C and then travel to the next level and visits D, E, F and G. Here, the goal state is defined as F. So, the traversal will stop at F.



In [ ]:

```
graph = {
 'A' : ['B','C'],
 'B' : ['D', 'E'],
 'C' : ['F', 'G'],
 'D' : [],
 'E' : [],
 'F' : [],
 'G' : []
}
visited = []
queue = []
goal = 'F'
def bfs(visited, graph, node):
    visited.append(node)
    queue.append(node)
    while queue:
        s = queue.pop(0)
        print (s, end = " ")
        for neighbour in graph[s]:
            if neighbour not in visited:
                visited.append(neighbour)
                queue.append(neighbour)
                if goal in visited:
                    break
bfs(visited, graph, 'A')
```

1.2 Depth First Search (DFS)

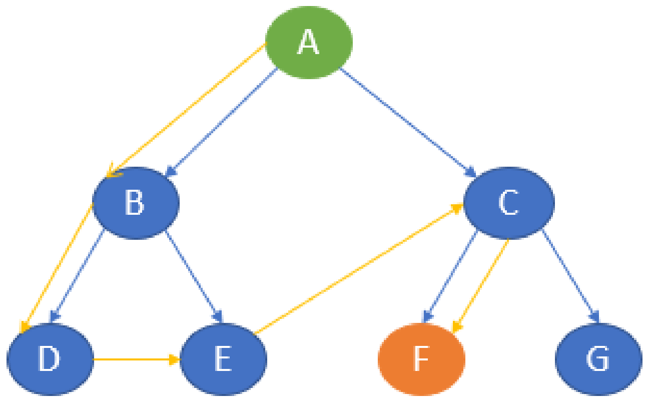
The depth-first search uses Last-in, First-out (LIFO) strategy and hence it can be implemented by using stack. DFS uses backtracking. That is, it starts from the initial state and explores each path to its greatest depth before it moves to the next path.

DFS will follow

Root node —-> Left node —-> Right node

Now, consider the same example tree mentioned above.

Here, it starts from the start state A and then travels to B and then it goes to D. After reaching D, it backtracks to B. B is already visited, hence it goes to the next depth E and then backtracks to B. as it is already visited, it goes back to A. A is already visited. So, it goes to C and then to F. F is our goal state and it stops there.



In [ ]:

```
#Let’s try to code it.
graph = {
 'A' : ['B','C'],
 'B' : ['D', 'E'],
 'C' : ['F', 'G'],
 'D' : [],
 'E' : [],
 'F' : [],
 'G' : []
}
goal = 'F'
visited = set()
def dfs(visited, graph, node):
    if node not in visited:
        print (node)
        visited.add(node)
        for neighbour in graph[node]:
            if goal in visited:
                break
            else:
                dfs(visited, graph, neighbour)
dfs(visited, graph, 'A')
```

1.3 Depth Limited Search (DLS)

Depth-limited works similarly to depth-first search. The difference here is that depth-limited search has a pre-defined limit up to which it can traverse the nodes. Depth-limited search solves one of the drawbacks of DFS as it does not go to an infinite path.

DLS ends its traversal if any of the following conditions exits.

Standard Failure

It denotes that the given problem does not have any solutions.

Cut off Failure Value

It indicates that there is no solution for the problem within the given limit.

Now, consider the same example.

Let's take A as the start node and C as the goal state and limit as 1.

The traversal first starts with node A and then goes to the next level 1 and the goal state C is there. It stops the traversal.

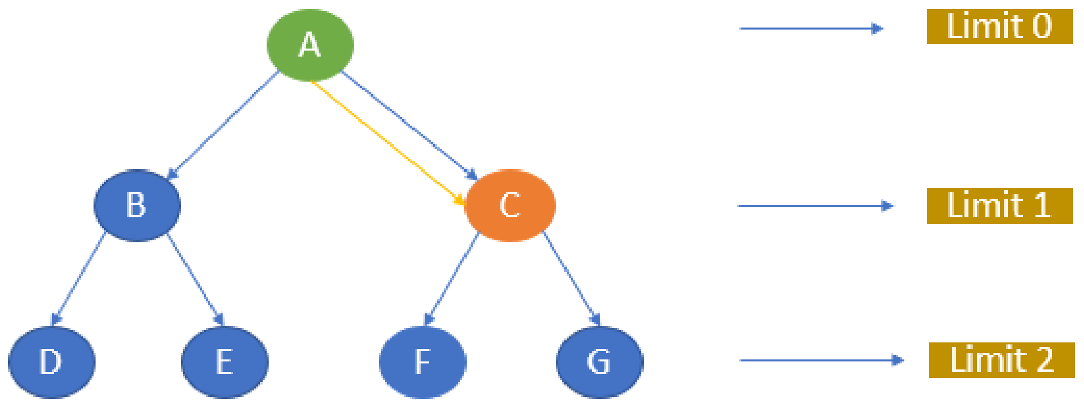
The path of traversal is:

A —-> C

If we give C as the goal node and the limit as 0, the algorithm will not return any path as the goal node is not available within the given limit.

If we give the goal node as F and limit as 2, the path will be A, C, F.

Let's implement DLS.



In [ ]:

```python
graph = {
 'A' : ['B','C'],
 'B' : ['D', 'E'],
 'C' : ['F', 'G'],
 'D' : [],
 'E' : [],
 'F' : [],
 'G' : []
}
def DLS(start,goal,path,level,maxD):
    print('nCurrent level-->',level)
    path.append(start)
    if start == goal:
        print("Goal test successful")
        return path
    print('Goal node testing failed')
    if level==maxD:
        return False
    print('nExpanding the current node',start)
    for child in graph[start]:
        if DLS(child,goal,path,level+1,maxD):
            return path
        path.pop()
    return False
start = 'A'
goal = input('Enter the goal node:-')
maxD = int(input("Enter the maximum depth limit:-"))
print()
path = list()
res = DLS(start,goal,path,0,maxD)
if(res):
    print("Path to goal node available")
    print("Path",path)
else:
    print("No path available for the goal node in given depth limit")
```

In [ ]: