Shivam Tiwari
5117060
BE-COMPS

Experiment No 1.

- **Aim:-** Write a program to implement BFS / DFS algorithm.

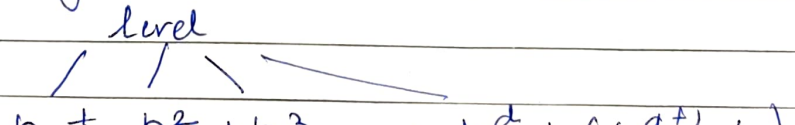
- **Theory:-**

Uninformed search (Blind search): They have no additional information about stats other than provided in the problem defn.

BFS:-

- In BFS root node is expanded first, then the successor of root node are expanded and so on.
- Implemented using first in first out queue data structure where fringe will be stored & processed
- Performance Evaluation.

1. **Completeness:** BFS is complete because if the shallowest goal node is at some finite depth 'd' BFS will eventually find it & generate solution.
2. **Optimality:** The shallowest goal node is not necessarily optimal. BFS will yield optimal solution only when all the actions have same cost.
3. **Time & Space Complexity:-** As the level of search tree grows more time is incurred. In general if search tree is at level d then $O(b^{d+1})$ time is required; where b is the no of nodes generated for each node, starting at root node i.e. root node generates b nodes & each b node generates b more & so on.



Algorithm BFS (G, n):
// Breadth first search of G
{ for i := 1 to n do // mark all vertices.
visited [i] = 0; // unvisited.
for i = 1 to n do.
if [visited [i] = 0] then BFS (i);
}

→ Depth first search:

- It Always expands the deepest node in the current unexpanded node set (fringe) of the search trees. The search goes into depth until there is no more successor node.
- DFS can be implemented with stack (LIFO) data structure. which will explore the latest added node first.

° Performance Evaluation:

1. completeness :- A DFS explores all the nodes hence guarantees soln.
2. Optimality :- As DFS reach to deepest node first, it may ignore some shallow node which can be goal state. Therefore, optimality is expected only when all states have same path cost.
3. time & space complexity:-
DFS requires some moderate amount of memory as it needs to store single path from root to some node to a particular level, along with unexpanded siblings.
time complexity : $O(b^d)$
Space complexity : $O(b^d + 1)$

```
Algorithm DFS [G)
  // Given an undirected / directed graph.
     G (V, E)
// n vertices and an array visited initial set.
// to zero; this algorithm visits all the vertices.
  { visited [v] = 1;
     for each vertex w adjacent from v do.
      {
         if (visited [w] = 0 then DFS w);
      }
  }
```

- **Conclusion:-**

Thus we have studied two blind search algorithms along with its performance measures & implemented it.

Rollno: 5117060

BREADTH FIRST SEARCH

CODE:

```
def bfs(graph, root, goal):
    queue = []
    path = []
    queue.append(graph[root])
    while (len(queue) != 0):
        current = queue.pop(0)
        path.append(current)
        if( current == goal):
            return ("Found", path)
        else:
            if current in graph.keys():
                temp = graph[current]
                for i in temp:
                    queue.append(i)
    return ("Not found", path)
if __name__ == '__main__':
        graph = {0: 9, 9: [7, 3],
                 7: [5, 1], 3: [2, 4]}
        print("Graph is:",graph)
        print("To find 1")
        res, path = bfs(graph, 0, 1)
        print(res)
        print("Path is:",path)
        print("N = 7 , b = 2 , d = 2")
        print("Space Complexity = O(", len(path) - 1, ")")
        print("Time Complexity = O(", len(path) - 1, ")")
```

OUTPUT:

```
Graph is: {0: 9, 9: [7, 3], 7: [5, 1], 3: [2, 4]}
To find 1
Found
Path is: [9, 7, 3, 5, 1]
N = 7 , b = 2 , d = 2
Space Complexity = O( 4 )
Time Complexity = O( 4 )
```

Rollno: 5117060

DEPTH FIRST SEARCH

CODE:

```
def dfs(graph, current, goal, path, time):
    time = time + 1
    path.append(current)
    if (current == goal):
        return ("found", path, time)
    if(current in graph.keys()):
        for i in graph[current]:
            temp = dfs(graph, i, goal, path, time)
            if (temp[0] == "found"):
                return (temp)
    return ("Not found", [-1], time)
if __name__ == '__main__':
        graph = {0: 9, 9: [7, 3],
                 7: [5, 1], 3: [2, 4]}
        root = 0
        print("Graph is:", graph)
        print("To find 1")

        res, path, time = dfs(graph, graph[root], 1, [], 0)
        print(res)
        print("Path is",path)
        print("N = 7 , b = 2 , d = 2")
        print("Time Complexity = O(", len(path), ")")
        print("Space Complexity = O(", time, ")")
```

OUTPUT:

```
OUTPUT: Graph is: {0: 9, 9: [7, 3], 7: [5, 1], 3: [2, 4]}
To find 1
found
Path is [9, 7, 5, 1]
N = 7 , b = 2 , d = 2
Time Complexity = O( 4 )
Space Complexity = O( 3 )
```