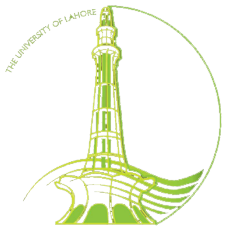

THE UNIVERSITY OF LAHORE
ISLAMABAD CAMPUS

Artificial Intelligence
(CS13217)

Lab Report

Name:	Wardha kanwal
Registration #:	CSU-S15-103
Lab Report #:	03
Dated:	06-04-2018
Submitted To:	Mr. Usman Ahmed

The University of Lahore, Islamabad Campus
Department of Computer Science & Information Technology

Experiment # 3
Depth First Search problem

Objective To understand and implement the Depth First Search problem

Software Tool
1. Python
2. Sublime, version 3.0
3. Operating System, window 8.1

1 Theory

The DFS algorithm is a recursive algorithm that uses the idea of backtracking. It involves exhaustive searches of all the nodes by going ahead, if possible, else by backtracking.

Here, the word backtrack means that when you are moving forward and there are no more nodes along the current path, you move backwards on the same path to find nodes to traverse. All the nodes will be visited on the current path till all the unvisited nodes have been traversed after which the next path will be selected.
This recursive nature of DFS can be implemented using stacks. The basic idea is as follows:
1. Pick a starting node and push all its adjacent nodes into a stack.
2. Pop a node from stack to select the next node to visit and push all its adjacent nodes into a stack.
3. Repeat this process until the stack is empty. However, ensure that the nodes that are visited are marked. This will prevent you from visiting the same node more than once. If you do not mark the nodes that are visited and you visit the same node more than once, you may end up in an infinite loop.

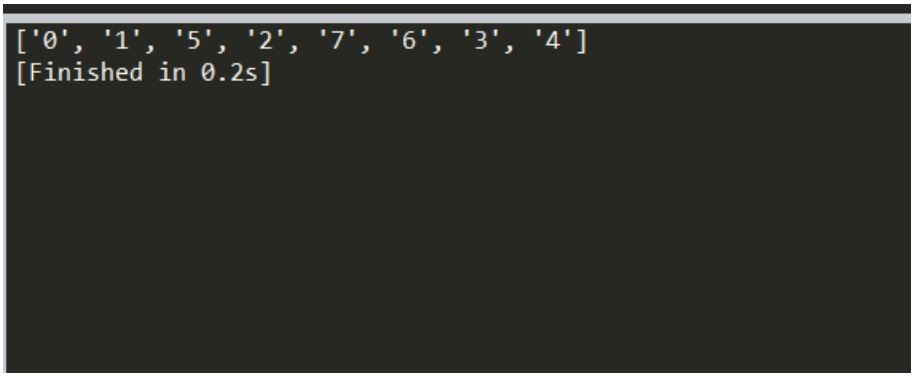

Figure 1: Time Independent Feature Set

2 Task

2.1 Procedure: Task 1

1. Visit the adjacent unvisited vertex. Mark it as visited. Display it. Push it in a stack
2. If no adjacent vertex is found, pop up a vertex from the stack. (It will pop up all the vertices from the stack, which do not have adjacent vertices.)
3. Repeat Rule 1 and Rule 2 until the stack is empty.

2.2 Procedure: Task 2

```
graph1 = {
    '0' : ['1','2','3','4'],
    '1' : ['0','5'],
    '2' : ['0','5'],
    '3' : ['0','6'],
    '4' : ['0','6'],
    '5' : ['1','2','7'],
    '6' : ['3','4','7'],
    '7' : ['5','6']

}
```

```
def dfs(graph, node, visited):
    if node not in visited:
        visited.append(node)
        for n in graph[node]:
            dfs(graph,n, visited)
    return visited

visited = dfs(graph1,'0', [])
print(visited)
```

3 Conclusion

1. Finding Connected components.
2. Topological sorting.
3. Finding Bridges of graph.
4. Memory requirement is Linear WRT Nodes.
5. Less time and space complexity rather than BFS.
6. Solution can be found out by without much more search.