Artificial Intelligence (18CSC305J)

Faculty : Helen Victoria A

Ex- 4 : Team Tesla 2.0

Name	Regno.	Email Id
Abhighyan B	RA1911033010002	ab2134@srmist.edu.in
Sanjana N B	RA1911033010016	sn8740@srmist.edu.in
Roehit Ranganathan	RA1911033010017	rr9344@srmist.edu.in
Venkata Naga Sai Ram Nomula	RA1911033010021	vn4903@srmist.edu.in
K. Dushyant Reddy	RA1911033010029	kr7119@srmist.edu.in

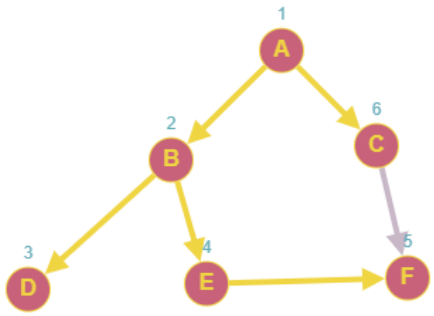
Experiment 4

Implementation and Analysis of DFS and BFS for an application

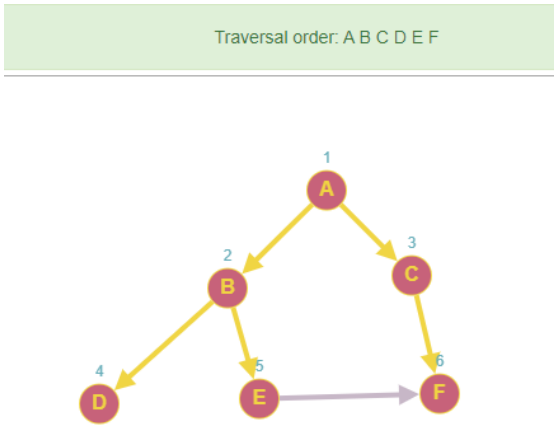
Problem Statement:

DFS: The Depth First Search is an algorithm for traversing or searching tree or graph data structures which uses the idea of backtracking. It explores all the nodes by going forward if possible or uses backtracking.

```
Traversal order: A B D E F C
```



BFS: The Breadth First Search is an algorithm for traversing or searching tree or graph data structures. It explores all the nodes at the present depth before moving on to the nodes at the next depth level.It uses Queue data structure for finding the shortest path.



Algorithm:

1. A Graph is defined as a list, along with two empty arrays as visited_bfs and queue.
2. Define a function bfs with three parameters- an empty array to store visited elements, the graph and the starting node.
3. The node is appended to the queue and visited arrays.
4. A while loop is initiated with the condition queue is TRUE, and variable s is assigned with the zeroth element of the queue popped.
5. S is printed and a for loop is initialized inside the while loop with the condition that the iterator doesn't exceed the elements in the graph of s.
6. If the iterator is not in the visited array, it gets added to the visited and queue arrays.
7. Define a function dfs and a set called visited.
8. The function dfs has three params– visited, graph and node.
9. If node is not visited, it is printed and added to the set of visited.

10. A for loop is initialized with the condition that the graph node is not null, and it calls the dfs function with an iterator as the new node.
11. The functions BFS and DFS are called.

Code:

```
graph = {
  'A' : ['B','C'],
  'B' : ['D', 'E'],
  'C' : ['F'],
  'D' : [],
  'E' : ['F'],
  'F' : []
}

visit_BFS = []
queue = []

def bfs(visit_BFS, graph, node):
  visit_BFS.append(node)
  queue.append(node)

  while queue:
    s = queue.pop(0)
    print (s, end = " ")

    for nearNode in graph[s]:
      if nearNode not in visit_BFS:
        visit_BFS.append(nearNode)
        queue.append(nearNode)

visit_DFS = set()

def dfs(visit_DFS, graph, node):
    if node not in visit_DFS:
        print (node, end=" ")
        visit_DFS.add(node)
        for nearNode in graph[node]:
            dfs(visit_DFS, graph, nearNode)
```

```
print("BFS:" , end =" ")
bfs(visit_BFS, graph, 'A')
print('\n')
print("DFS:" , end =" ")
dfs(visit_DFS, graph, 'A')
```

Output:

```
PS D:\SRM\SEM 6\AI lab\EXP-4> python -u "d:\SRM\SEM 6\AI lab\EXP-4\exp4.py"
BFS: A B C D E F

DFS: A B D E F C
PS D:\SRM\SEM 6\AI lab\EXP-4>
```

Time Complexity:

BFS & DFS

a. Time complexity in Data Structures - O(V+E)
   where, V=no of nodes, E = no of edges

b. Time complexity in Artificial Intelligence - O(b^d)
   where, b=branch factor, d= depth

Real World Solution:

- GPS Navigation systems
- Crawlers in Search Engines
- Broadcasting in Network

Result: BFS and DFS algorithms were successfully implemented.