ST.FRANCIS INSTITUTE OF TECHNOLOGY

Mount Poinsur, S.V.P. Road, Borivli (West), Mumbai - 400103

Computer Engineering Department

Academic Year: 2019-2020 **Class/Branch**: BE COMP

Subject: CSC 703 Artificial Intelligence & Soft Computing **Semester**: VII

Name: Selas Moro **Roll:** 39 **Pid:** 182074 **Class:** BE CMPN B

Experiment No. 3 : BFS and DFS Algorithm

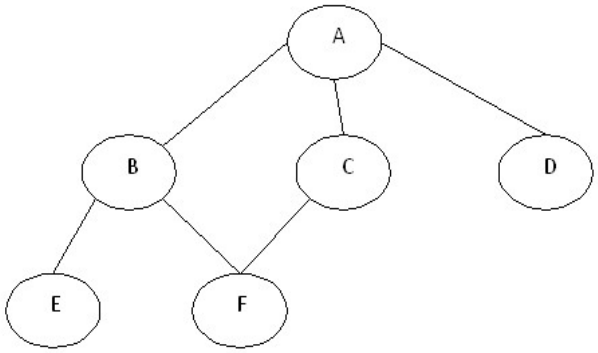
Aim: To implement BFS and DFS algorithm

Theory:

It is the process of searching a solution in state space tree. The breadth first search (BFS) and the depth first search (DFS) are the two algorithms used for traversing and searching a node in a graph. They can also be used to find out whether a node is reachable from a given node or not.

Experiment:

The aim of BFS algorithm is to traverse the graph as close as possible to the root node. Queue is used in the implementation of the breadth first search. Let's see how BFS traversal works with respect to the following graph:

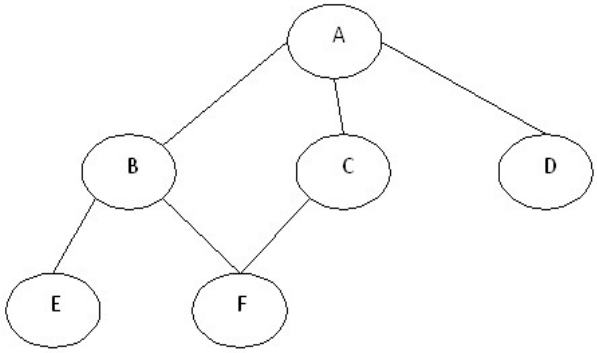


If we do the breadth first traversal of the above graph and print the visited node as the output, it will print the following output. "A B C D E F". The BFS visits the nodes level by level, so it will start with level 0 which is the root node, and then it moves to the next levels which are B, C and D, then the last levels which are E and F.

Algorithmic Steps

1. **Step 1**: Push the root node in the Queue.
2. **Step 2**: Loop until the queue is empty.
3. **Step 3**: Remove the node from the Queue.
4. **Step 4**: If the removed node has unvisited child nodes, mark them as visited and insert the unvisited children in the queue.

The aim of DFS algorithm is to traverse the graph in such a way that it tries to go far from the root node. Stack is used in the implementation of the depth first search. Let's see how depth first search works with respect to the following graph:



As stated before, in DFS, nodes are visited by going through the depth of the tree from the starting node. If we do the depth first traversal of the above graph and print the visited node, it will be "A B E F C D". DFS visits the root node and then its children nodes until it reaches the end node, i.e. E and F nodes, then moves up to the parent nodes.
Algorithmic Steps
Step 1: Push the root node in the Stack.
Step 2: Loop until stack is empty.
Step 3: Peek the node of the stack.
Step 4: If the node has unvisited child nodes, get the unvisited child node, mark it as traversed and push it on stack.
Step 5: If the node does not have any unvisited child nodes, pop the node from the stack.

Post Experiment Exercise:

I have taken an real life example of the puzzle game of "Tower of Hanoi" problem and implemented it using Python.

Code:

```
# Recursive Python function to solve tower of hanoi

def TowerOfHanoi(n , from_rod, to_rod, aux_rod):
    if n == 1:
        print("Move disk 1 from rod",from_rod,"to rod",to_rod)
        return
    TowerOfHanoi(n-1, from_rod, aux_rod, to_rod)
    print("Move disk",n,"from rod",from_rod,"to rod",to_rod)
    TowerOfHanoi(n-1, aux_rod, to_rod, from_rod)

# Driver code
n = 4
```

```
TowerOfHanoi(n, 'A', 'C', 'B')
# A, C, B are the name of rods
```

Output:

```
= RESTART: C:\Users\Administrator.MAHESHC\Documents\SEM 7\Airtificial intelligence\Exp3_post.py
Move disk 1 from rod A to rod B
Move disk 2 from rod A to rod C
Move disk 1 from rod B to rod C
Move disk 3 from rod A to rod B
Move disk 1 from rod C to rod A
Move disk 2 from rod C to rod B
Move disk 1 from rod A to rod B
Move disk 4 from rod A to rod C
Move disk 1 from rod B to rod C
Move disk 2 from rod B to rod A
Move disk 1 from rod C to rod A
Move disk 3 from rod B to rod C
Move disk 1 from rod A to rod B
Move disk 2 from rod A to rod C
Move disk 1 from rod B to rod C
```

Implementation:

Code:

```
#Exp 3- BFS and DFS
print("          ----- The Input Graph is -----")
print("                    A   ")
print("                  /   \\")
print("                 B      C")
print("                /  \\   /  \\")
print("               D    E  F    G")
print("              /           \\")
print("             H             I")

graph = {
   "A": ["B", "C"],
   "B": ["D", "E"],
   "C": ["F", "G"],
   "D": ["H"],
   "E": [],
   "F": [],
   "G": ["I"],
   "H": [],
   "I": []
}
```

```
visited = []
queue = []


def bfs(visited, graph, node, goal):
    visited.append(node)
    queue.append(node)

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

        for neighbour in graph[s]:
            if neighbour not in visited:
                visited.append(neighbour)
                queue.append(neighbour)


visited_dfs = set()


def dfs(visited_dfs, graph, node, goal):
    if node not in visited_dfs:
        print(node, end=" ")
        visited_dfs.add(node)
        for neighbour in graph[node]:
            if(goal in visited_dfs):
                return
            dfs(visited_dfs, graph, neighbour, goal)


g = input("Enter Goal Node for BFS : ")

bfs(visited, graph, "A", str(g).upper())
print()

g = input("Enter Goal Node for DFS : ")

dfs(visited_dfs, graph, "A", str(g).upper())
```

Output:

```
========== RESTART: C:\Users\Administrator.MAHESHC\Documents\SEM 7\SEM_7\AISC Lab\exp03.py ==========
        ----- The Input Graph is -----
                    A
                  /   \
                B      C
               /  \   /  \
              D    E F    G
             /            \
            H              I
Enter Goal Node for BFS : H
A B C D E F G H
Enter Goal Node for DFS : H
A B D H
```

Fig 2 : Finding DFS and BFS for Node : H

```
========== RESTART: C:\Users\Administrator.MAHESHC\Documents\SEM 7\SEM_7\AISC Lab\exp03.py ==========
        ----- The Input Graph is -----
                    A
                  /   \
                B      C
               /  \   /  \
              D    E F    G
             /            \
            H              I
Enter Goal Node for BFS : I
A B C D E F G H I
Enter Goal Node for DFS : I
A B D H E C F G I
```

Fig 3: Finding DFS and BFS for Node : I

Conclusion: In this experiment , I have understood the concepts concerning about BFS and DFS . Also I have done Implementation of sample graph for BFS and DFS in python.