AI LAB MANUAL

Exp 4: Implementation of BFS and DFS

Name:	SHAMUNESH P
Reg.no:	RA1911030010122
Problem:	BFS & DFS
Date:	08-02-22

BFS (Breadth First Search):

Breadth-first search is an algorithm for searching a tree data structure for a node that satisfies a given property. It starts at the tree root and explores all nodes at the present depth prior to moving on to the nodes at the next depth level.

Code: (Python)

```
graph = {
  '5' : ['3','7'],
  '3' : ['2', '4'],
  '7' : ['8'],
  '2' : [],
  '4' : ['8'],
  '8' : []
}

visited = [] # List for visited nodes.
queue = []     #Initialize a queue

def bfs(visited, graph, node): #function for BFS
```

```
  visited.append(node)
  queue.append(node)

  while queue:          # Creating loop to visit each node
    m = queue.pop(0)
    print (m, end = " ")

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

# Driver Code
print("Following is the Breadth-First Search")
bfs(visited, graph, '5')
```

Implementation:

Screenshots

```python
graph = {
  '5' : ['3','7'],
  '3' : ['2', '4'],
  '7' : ['8'],
  '2' : [],
  '4' : ['8'],
  '8' : []
}

visited = [] # List for visited nodes.
queue = []     #Initialize a queue

def bfs(visited, graph, node): #function for BFS
  visited.append(node)
  queue.append(node)

  while queue:          # Creating loop to visit each node
    m = queue.pop(0)
    print (m, end = ' ')

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

# Driver Code
print("Following is the Breadth-First Search")
bfs(visited, graph, '5')    # function calling
```

Output:

```
Following is the Breadth-First Search
5 3 7 2 4 8
```

DFS (Depth First Search):

Depth-first search (**DFS**) is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node (Selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking.

Code: (Python)

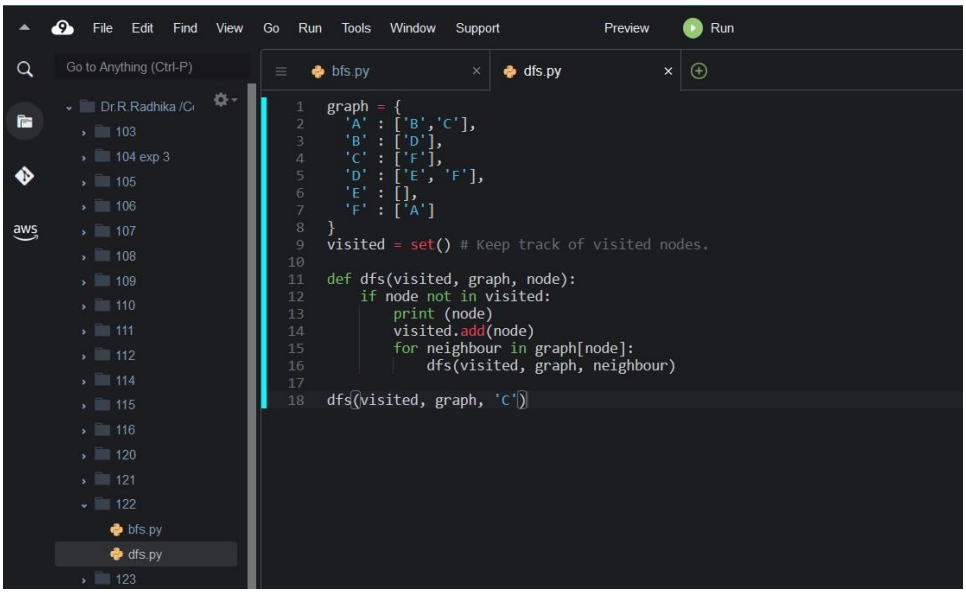
```
graph = {
  'A' : ['B','C'],
  'B' : ['D'],
  'C' : ['F'],
  'D' : ['E', 'F'],
  'E' : [],
  'F' : ['A']
}
visited = set() # Keep track of visited nodes.


def dfs(visited, graph, node):
    if node not in visited:
        print (node)
        visited.add(node)
        for neighbour in graph[node]:
            dfs(visited, graph, neighbour)


dfs(visited, graph, 'C')
```

Implementation:

Screenshots



Output: