ABHINAV RANJAN
RA1911003010003
CSE A1 SECTION
SRMIST , KTR

AI LAB EXP 4 - BFS AND DFS

AIM :

To create a python program exhibiting the implementation of Breadth First Search(BFS) and Depth First Search(DFS) algorithm.

PSEUDO CODE :

1. **BFS**

```
● create a queue Q
● mark v as visited and put v into Q
● while Q is non-empty
●     remove the head u of Q
●     mark and enqueue all (unvisited) neighbours of u
```

2. **DFS**

The pseudocode for Depth-First Search goes as below: In the init() function, notice that we run the DFS function on every node because many times, a graph may contain two different disconnected part and therefore to make sure that we have visited every vertex, we can also run the DFS algorithm at every node.

```
DFS(G, u)
    u.visited = true
    for each v ∈ G.Adj[u]
        if v.visited == false
            DFS(G,v)
init() {
    For each u ∈ G
        u.visited = false
     For each u ∈ G
       DFS(G, u)
}
```

SOURCE CODE :

1. BFS

```
tree = {
    '5':['3','7'],
    '3':['2','4'],
    '7':['8'],
    '2':[],
    '4':['8'],
    '8':[]
}
visited = []
queue = []
def bfs(visited,tree,node):
    visited.append(node)
    queue.append(node)
    s=int(node)
    while queue:
        m=queue.pop(0)
        for i in tree[m]:            # i refers to neighbour
            if i not in visited:
                s = s*int(i)        # cumulatively multiplying the nodes
                queue.append(i)
                visited.append(i)
    return s
print(bfs(visited,tree,'5'));
```

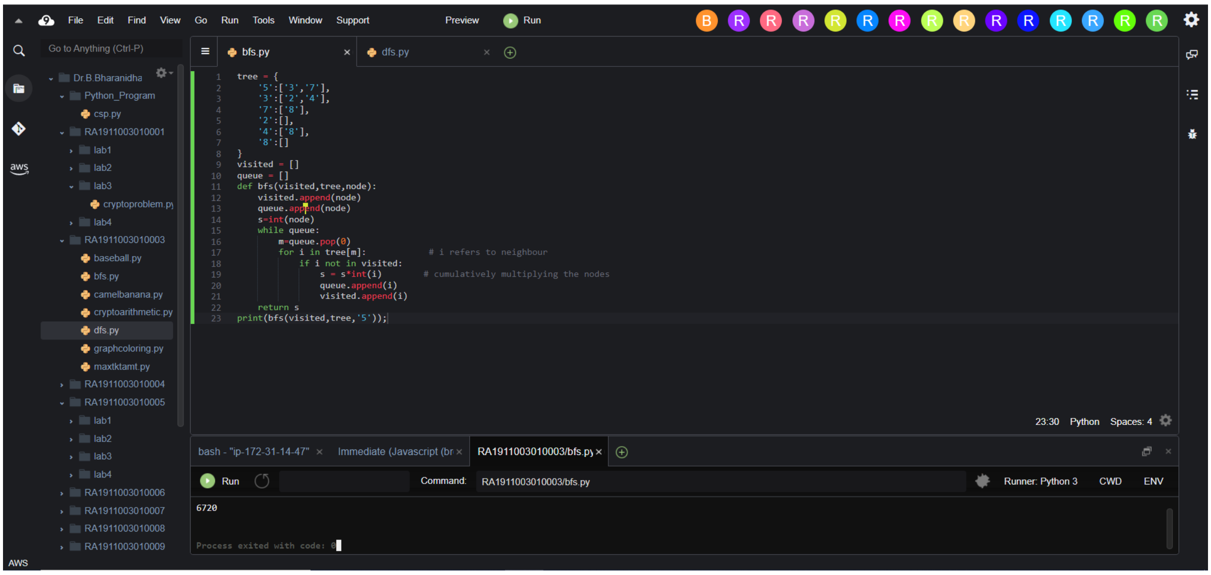
2. DFS

```
from functools import reduce
tree = {
    '5':['3','7'],
    '3':['2','4'],
    '7':['8'],
    '2':[],
    '4':['8'],
    '8':[]
}

visited = set()
data = []
def dfs(visited, graph, node):
    if node not in visited:
        print (node)
        data.append(int(node))
        visited.add(node)
        for neighbour in graph[node]:
            dfs(visited, graph, neighbour)
dfs(visited, tree, '5')
print("the product of nodes is")
print(reduce((lambda x, y: x - y), data))
```

SCREENSHOT OF OUTPUT :

1. BFS



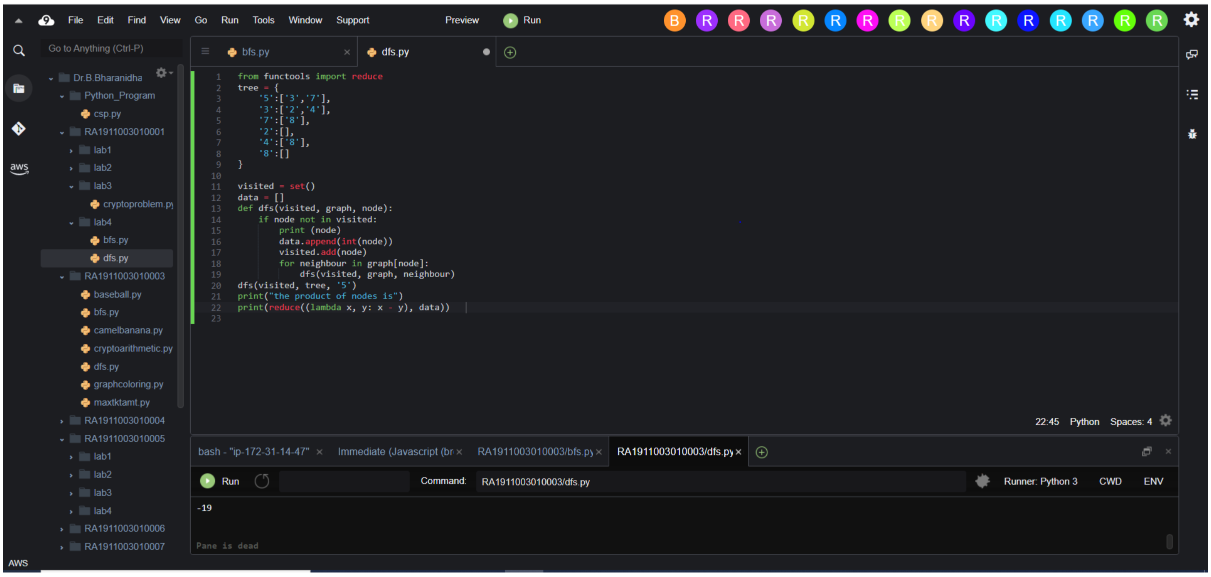
```python
tree = {
    '5':['3','7'],
    '3':['2','4'],
    '7':['8'],
    '2':[],
    '4':['8'],
    '8':[]
}
visited = []
queue = []
def bfs(visited,tree,node):
    visited.append(node)
    queue.append(node)
    s=int(node)
    while queue:
        m=queue.pop(0)
        for i in tree[m]:            # i refers to neighbour
            if i not in visited:
                s = s*int(i)        # cumulatively multiplying the nodes
                queue.append(i)
                visited.append(i)
    return s
print(bfs(visited,tree,'5'));
```

Command: RA1911003010003/bfs.py

```
6720

Process exited with code: 0
```

2. DFS



```python
from functools import reduce
tree = {
    '5':['3','7'],
    '3':['2','4'],
    '7':['8'],
    '2':[],
    '4':['8'],
    '8':[]
}

visited = set()
data = []
def dfs(visited, graph, node):
    if node not in visited:
        print (node)
        data.append(int(node))
        visited.add(node)
        for neighbour in graph[node]:
            dfs(visited, graph, neighbour)
dfs(visited, tree, '5')
print("the product of nodes is")
print(reduce((lambda x, y: x - y), data))
```

Command: RA1911003010003/dfs.py

```
-19

Pane is dead
```

RESULT :

Thus we have successfully implemented python programs and showed the working of BFS and DFS search algorithms