

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  $\in$  G.Adj[u]
        if v.visited == false
            DFS(G,v)

init() {
    For each u  $\in$  G
        u.visited = false
    For each u  $\in$  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

```

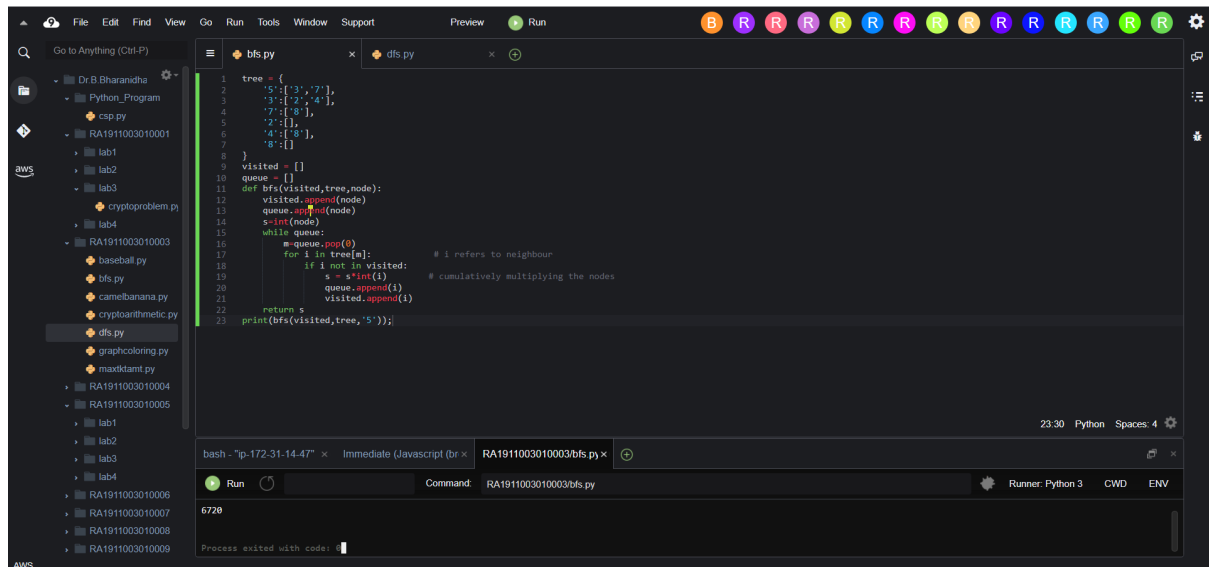
from functools import reduce
tree = {
    '5':['3','7'],
    '3':['2','4'],
    '7':['8'],
    '2':[],
    '4':['8'],
    '8':[]
}

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

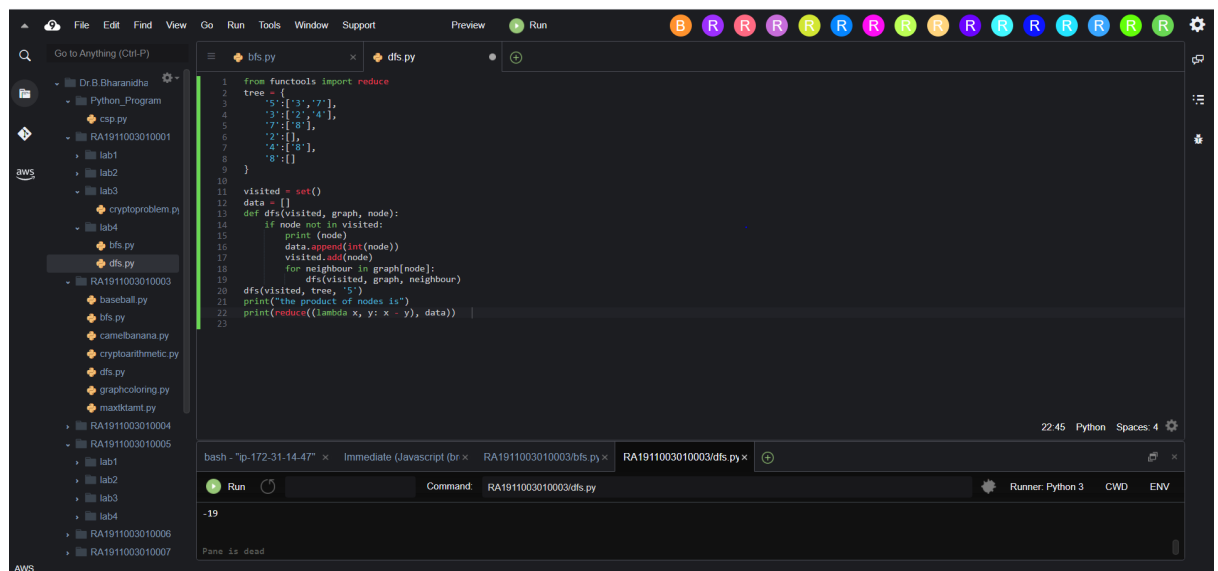


The screenshot shows a code editor with a file explorer on the left and a terminal at the bottom. The file explorer shows a project structure with files like `csp.py`, `lab1`, `lab2`, `lab3`, `lab4`, `cryptoproblem.py`, `baseball.py`, `bfs.py`, `camelbanana.py`, `cryptoarithmetic.py`, `dfs.py`, `graphcoloring.py`, and `maxktamt.py`. The `bfs.py` file is open in the editor, showing a Breadth-First Search implementation. The code defines a tree structure and a function `bfs` that traverses the tree. The terminal shows the command `RA1911003010003/bfs.py` and the output `6720`.

```
1 tree = {
2     '5': ['3', '7'],
3     '3': ['2', '4'],
4     '7': ['8'],
5     '2': [],
6     '4': ['8'],
7     '8': []
8 }
9 visited = []
10 queue = []
11 def bfs(visited, tree, node):
12     visited.append(node)
13     queue.append(node)
14     s = int(node)
15     while queue:
16         m = queue.pop(0)
17         for i in tree[m]:
18             # i refers to neighbour
19             if i not in visited:
20                 # cumulatively multiplying the nodes
21                 queue.append(i)
22                 visited.append(i)
23     return s
24 print(bfs(visited, tree, '5'))
```

6720

2. DFS



The screenshot shows a code editor with a file explorer on the left and a terminal at the bottom. The file explorer shows the same project structure as the first screenshot. The `dfs.py` file is open in the editor, showing a Depth-First Search implementation. The code defines a tree structure and a function `dfs` that traverses the tree. The terminal shows the command `RA1911003010003/dfs.py` and the output `-19`.

```
1 from functools import reduce
2 tree = {
3     '5': ['3', '7'],
4     '3': ['2', '4'],
5     '7': ['8'],
6     '2': [],
7     '4': ['8'],
8     '8': []
9 }
10 visited = set()
11 data = []
12 def dfs(visited, graph, node):
13     if node not in visited:
14         print(node)
15         data.append(int(node))
16         visited.add(node)
17         for neighbour in graph[node]:
18             dfs(visited, graph, neighbour)
19     dfs(visited, tree, '5')
20 print("the product of nodes is")
21 print(reduce(lambda x, y: x * y, data))
```

-19

RESULT :

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