IMPLEMENTATION AND ANALYSIS OF BFS & DFS FOR AN APPLICATION

NAME: **RISHAL RAMESH** ***EXP NO:*** **4**

REG NO: **RA1911030010084**

AIM:

To implement BFS(Breadth first search) and DFS(Depth first search) using python.

BREADTH FIRST SEARCH (BFS):

Breadth-First Search (BFS) is an algorithm used for traversing graphs or trees. Traversing means visiting each node of the graph. Breadth-First Search is a recursive algorithm to search all the vertices of a graph or a tree. BFS in python can be implemented by using data structures like a dictionary and lists. Breadth-First Search in tree and graph is almost the same. The only difference is that the graph may contain cycles, so we may traverse to the same node again.

ALGORITHM:

The steps involved in the BFS algorithm to explore a graph are given as follows –

Step 1: SET STATUS = 1 (ready state) for each node in G

Step 2: Enqueue the starting node A and set its STATUS = 2 (waiting state)

Step 3: Repeat Steps 4 and 5 until QUEUE is empty

Step 4: Dequeue a node N. Process it and set its STATUS = 3 (processed state).

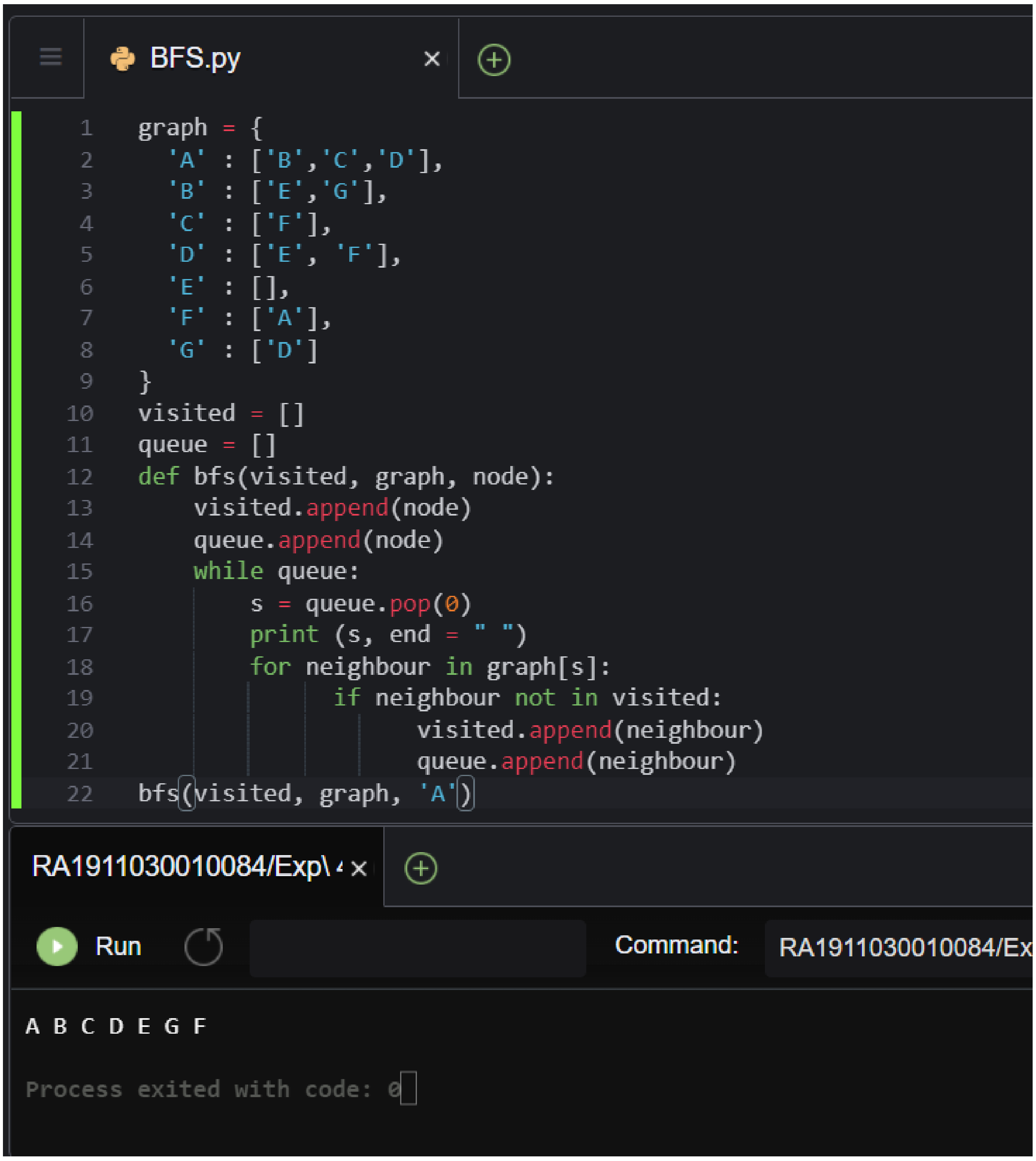
Step 5: Enqueue all the neighbours of N that are in the ready state (whose STATUS = 1) and set their STATUS = 2 (waiting state) [END OF LOOP]

Step 6: EXIT

CODE:

```
graph = {
  'A' : ['B','C','D'],
  'B' : ['E','G'],
  'C' : ['F'],
  'D' : ['E', 'F'],
  'E' : [],
  'F' : ['A'],
  'G' : ['D']
}
visited = []
queue = []
def bfs(visited, graph, node):
    visited.append(node)
    queue.append(node)
    while queue:
        s = queue.pop(0)
        print (s, end = " ")
        for neighbour in graph[s]:
            if neighbour not in visited:
                visited.append(neighbour)
                queue.append(neighbour)
bfs(visited, graph, 'A')
```

OUTPUT:



```python
graph = {
   'A' : ['B','C','D'],
   'B' : ['E','G'],
   'C' : ['F'],
   'D' : ['E', 'F'],
   'E' : [],
   'F' : ['A'],
   'G' : ['D']
}
visited = []
queue = []
def bfs(visited, graph, node):
    visited.append(node)
    queue.append(node)
    while queue:
        s = queue.pop(0)
        print (s, end = " ")
        for neighbour in graph[s]:
            if neighbour not in visited:
                visited.append(neighbour)
                queue.append(neighbour)
bfs(visited, graph, 'A')
```

RA1911030010084/Exp\

RA1911030010084/Ex

```
A B C D E G F

Process exited with code: 0
```

DEPTH FIRST SEARCH (DFS):

Depth-First Search (DFS) is a recursive algorithm that uses the concept of backtracking. It involves thorough searches of all the nodes by going ahead if potential, else by backtracking. Here, the word backtrack means once you are moving forward and there are not any more nodes along the present path, you progress backward on an equivalent path to seek out nodes to traverse. All the nodes are progressing to be visited on the current path until all the unvisited nodes are traversed after which subsequent paths are going to be selected.

ALGORITHM:

The steps involved in the DFS algorithm to explore a graph are given as follows –

Step 1: SET STATUS = 1 (ready state) for each node in G

Step 2: Push the starting node A on the stack and set its STATUS = 2 (waiting state)

Step 3: Repeat Steps 4 and 5 until STACK is empty

Step 4: Pop the top node N. Process it and set its STATUS = 3 (processed state)

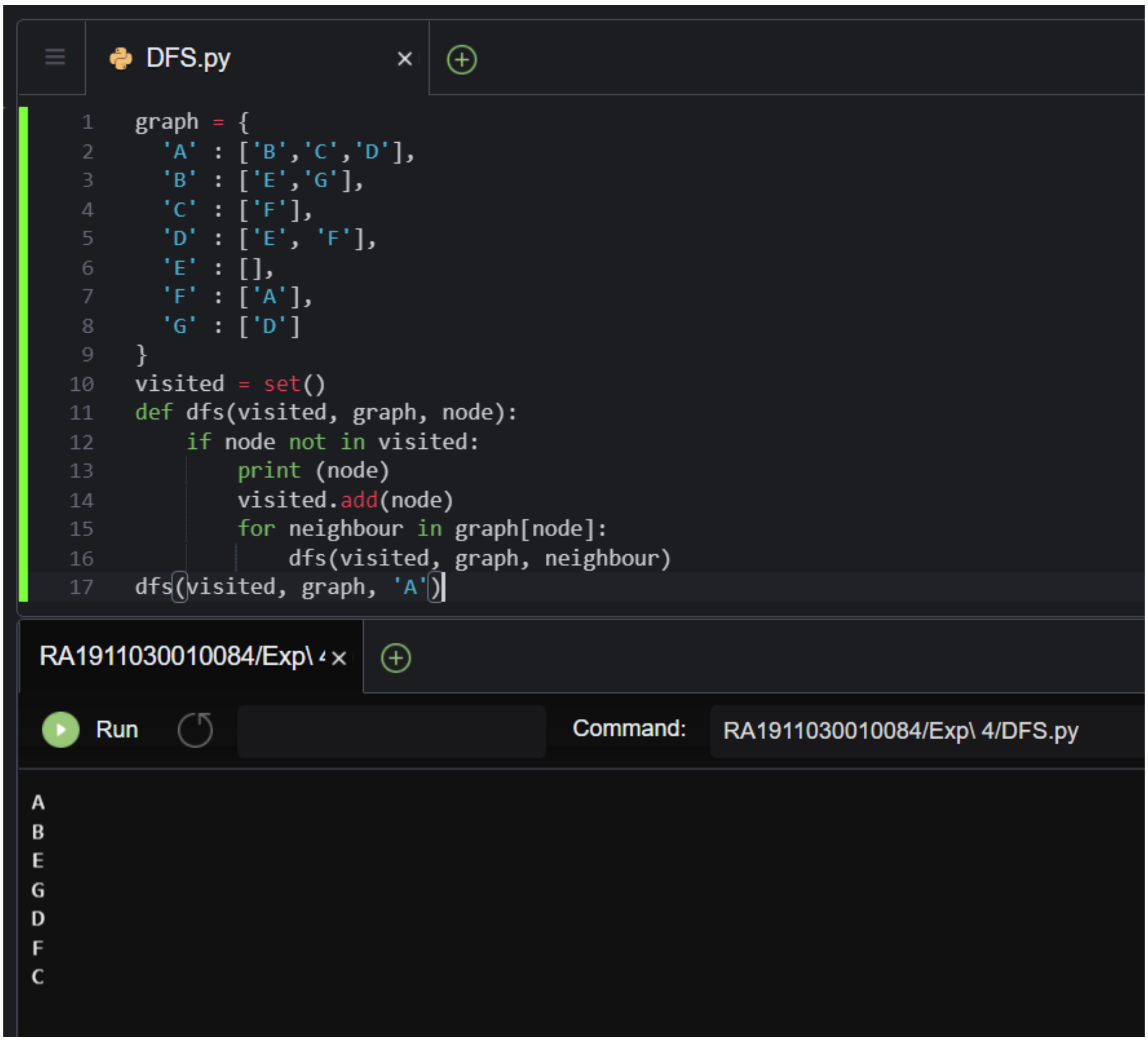
Step 5: Push on the stack all the neighbours of N that are in the ready state (whose STATUS = 1) and set their STATUS = 2 (waiting state)
[END OF LOOP]

Step 6: EXIT

CODE:

```
graph = {
  'A' : ['B','C','D'],
  'B' : ['E','G'],
  'C' : ['F'],
  'D' : ['E', 'F'],
  'E' : [],
  'F' : ['A'],
  'G' : ['D']
}
visited = set()
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, 'A')
```

OUTPUT:



```
graph = {
   'A' : ['B','C','D'],
   'B' : ['E','G'],
   'C' : ['F'],
   'D' : ['E', 'F'],
   'E' : [],
   'F' : ['A'],
   'G' : ['D']
}
visited = set()
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, 'A')
```

```
RA1911030010084/Exp\ 4/DFS.py
```

```
A
B
E
G
D
F
C
```

RESULT:

BFS and DFS were successfully implemented using python in an AWS environment.