

EXERCISE-4

Implementation and analysis of BFS and DFS

NAME: A.HARICHANDANA

REG NO: RA1911026010009

SEC:K1

AIM: Implementation and Analysis of DFS and BFS for an application.

PROBLEM STATEMENT: Given the graph, implement the Breadth-First Search and Depth First Search for the given graph.

ALGORITHM/PROCEDURE FOR BFS AND DFS:

The DSF algorithm follows as:

1. We will start by putting any one of the graph's vertex on top of the stack.
2. After that take the top item of the stack and add it to the visited list of the vertex.
3. Next, create a list of that adjacent node of the vertex. Add the ones which aren't in the visited list of vertexes to the top of the stack.
4. Lastly, keep repeating steps 2 and 3 until the stack is empty

The BFS algorithm is :

1. Start by putting any one of the graph's vertices at the back of the queue.
2. Now take the front item of the queue and add it to the visited list.
3. Create a list of that vertex's adjacent nodes. Add those which are not within the visited list to the rear of the queue.
4. Keep continuing steps two and three till the queue is empty.

BFS:

```
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)
```

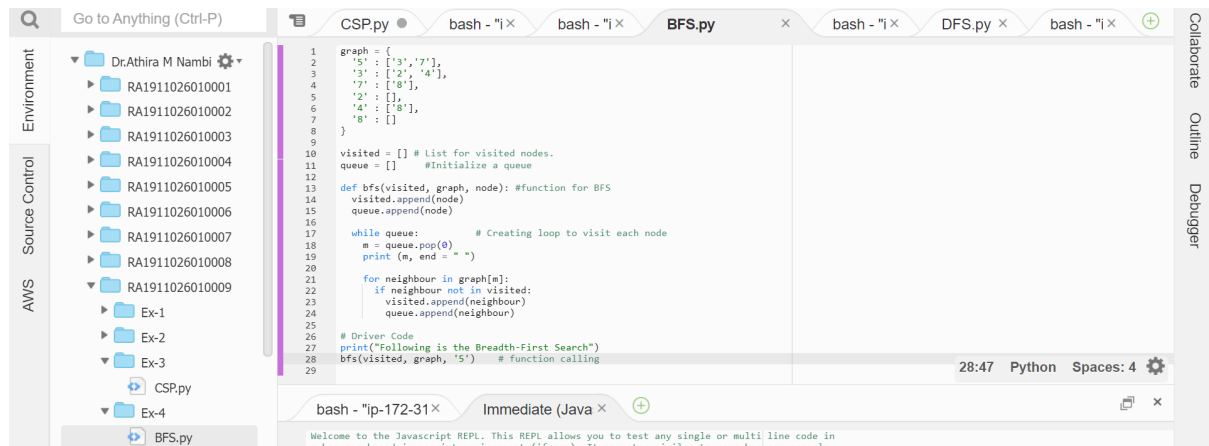
```
            queue.append(neighbour)
```

```
# Driver Code
```

```
print("Following is the Breadth-First Search")
```

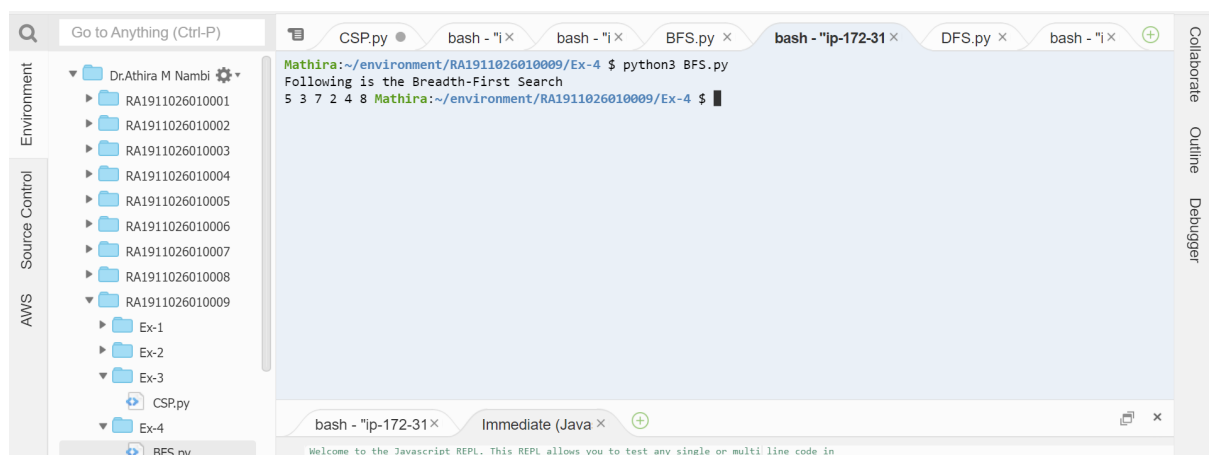
```
bfs(visited, graph, '5') #function calling
```

Screenshots of the Code :



```
1 graph = {
2     '5': ['3','7'],
3     '3': ['2','4'],
4     '7': ['8'],
5     '2': [],
6     '4': ['8'],
7     '8': []
8 }
9
10 visited = [] # List for visited nodes.
11 queue = [] # Initialize a queue
12
13 def bfs(visited, graph, node): #function for BFS
14     visited.append(node)
15     queue.append(node)
16
17     while queue: # Creating loop to visit each node
18         m = queue.pop(0)
19         print (m, end = " ")
20
21         for neighbour in graph[m]:
22             if neighbour not in visited:
23                 visited.append(neighbour)
24                 queue.append(neighbour)
25
26 # Driver Code
27 print("Following is the Breadth-First Search")
28 bfs(visited, graph, '5') # Function calling
29
```

OUTPUT:



```
Mathira:~/environment/RA1911026010009/Ex-4 $ python3 BFS.py
Following is the Breadth-First Search
5 3 7 2 4 8 Mathira:~/environment/RA1911026010009/Ex-4 $
```

DFS:

Using a Python dictionary to act as an adjacency list

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

```
'8' : []
}
```

visited = set() # Set to keep track of visited nodes of graph.

def dfs(visited, graph, node): #function for dfs

if node not in visited:

print (node)

visited.add(node)

for neighbour in graph[node]:

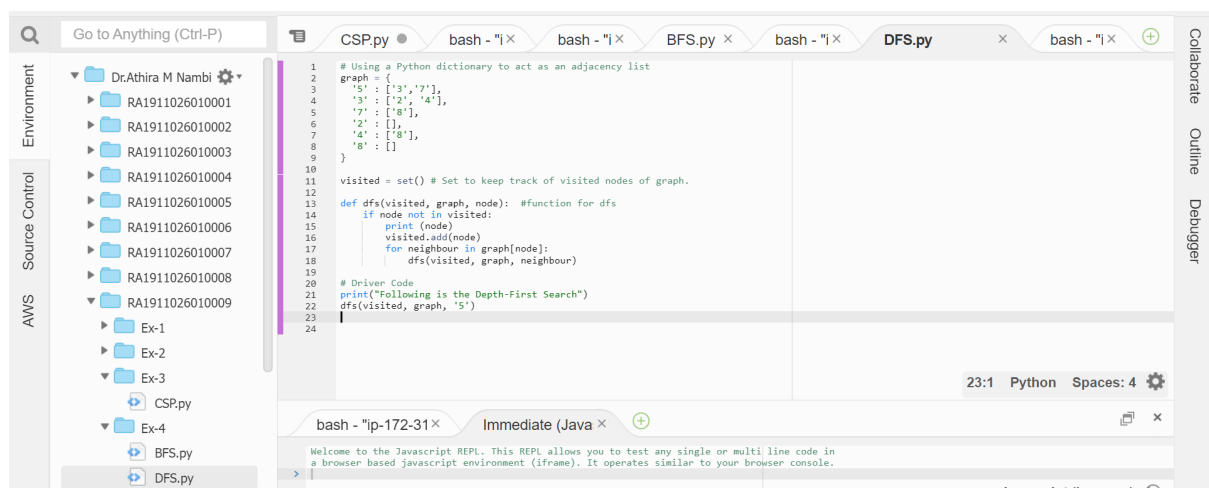
dfs(visited, graph, neighbour)

Driver Code

print("Following is the Depth-First Search")

dfs(visited, graph, '5')

Screenshot of the Code in AWS:



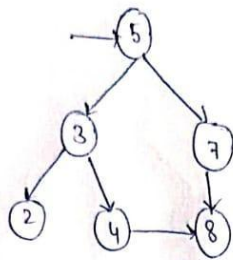
OUTPUT:



MANUAL CALCULATION FOR BFS AND DFS

BFS:

Bfs



Adjacent of 5 : [3, 7]

Adjacent of 3 : [2, 4]

Adjacent of 7 : [8]

Adjacent of 4 : [8]

Adjacent of 2 : []

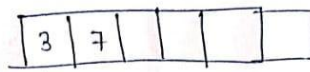
Adjacent of 8 : []

Visited

①

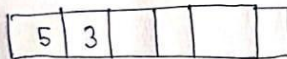


Queue

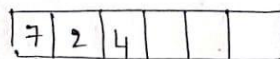


↑
front

②

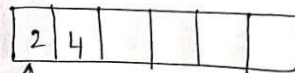
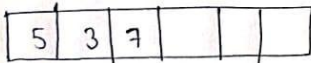


~~Q~~



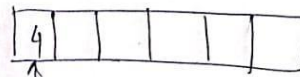
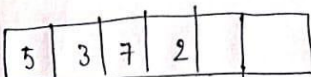
↑
front

③



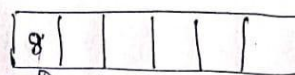
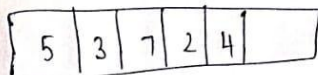
↑
front

④



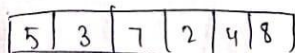
↑
front

⑤



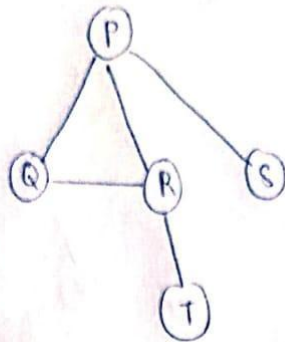
↑
front

⑥



DFS:

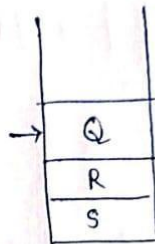
Dfs Sample graph



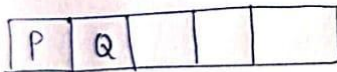
Visited



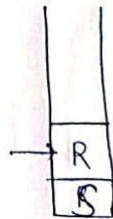
Stack



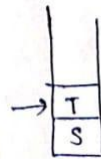
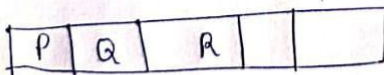
Visited



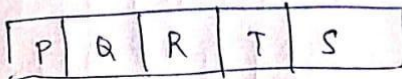
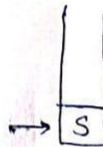
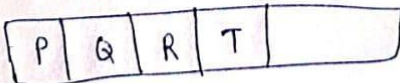
Stack



Visited



Visited



OBSERVATION:

Breadth-First Search and Depth-First Search are the types of Uninformed Search which means that they do not contain any domain knowledge such as closeness, the location of the goal.

In both cases, the search will be applied in such a way that the search tree will be searched without any information about the search space like initial state operators, and test for the goal

Breadth-First Search

- Breadth-First Search (BFS) is an algorithm for traversing an unweighted Graph or a Tree.
- BFS starts with the root node and explores each adjacent node before exploring node(s) at the next level.
- BFS makes use of Queue for storing the visited nodes of the graph/tree.

The time complexity of the breadth-first search algorithm can be stated as $O(|V|+|E|)$ because, in the worst case, it will explore every vertex and edge. The number of vertices in the graph is $|V|$, while the edges are $|E|$.

— The space complexity of the breadth-first search algorithm: The space complexity is $O(|V|)$, where $|V|$ is the number of vertices in the graph, and different data structures are needed to determine which vertices have already been added to the queue.

Depth-First Search

DFS starts with the root node and explores all the nodes along with the depth of the selected path before backtracking to explore the next path. DFS makes use of Stack for storing the visited nodes of the graph/tree.

- The time complexity of DFS if the entire **tree** is traversed is **$O(V)$** where V is the number of nodes.
- Since we are maintaining a stack to keep track of the last visited node, in worst case, the stack could take up to the size of the nodes(or vertices) in the graph. Hence, the space complexity is **$O(V)$** .

RESULT: Successfully implemented Breadth-First Search and Depth-First Search on the given graph.