

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)
```

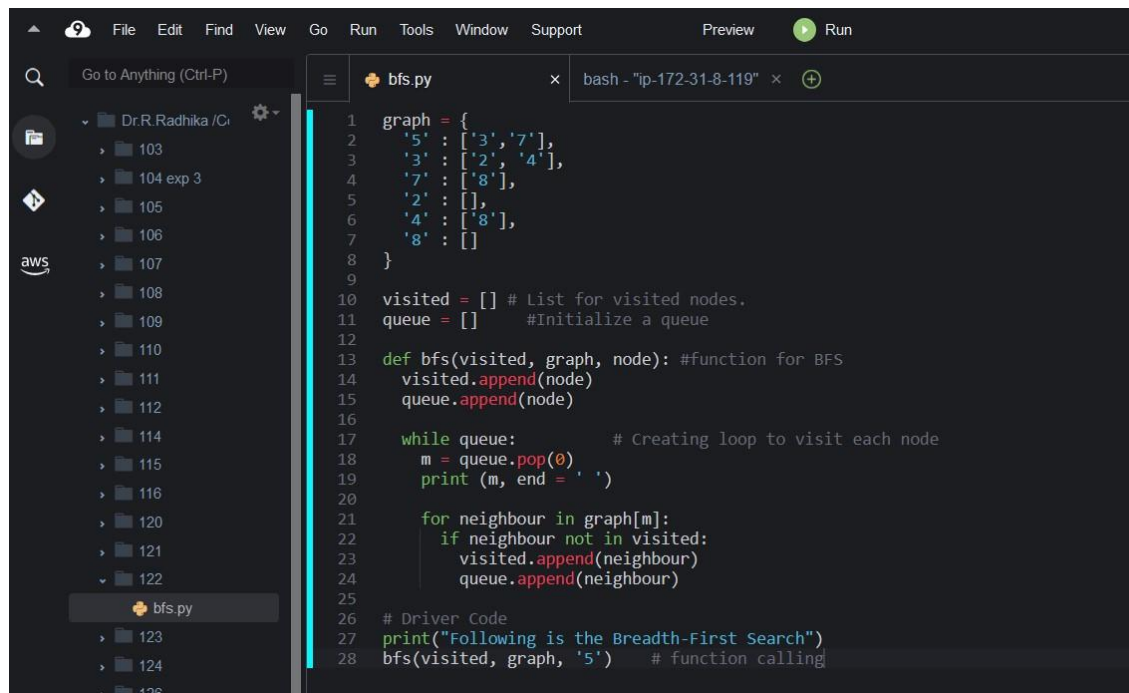
```
# Driver Code
```

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

```
bfs(visited, graph, '5')
```

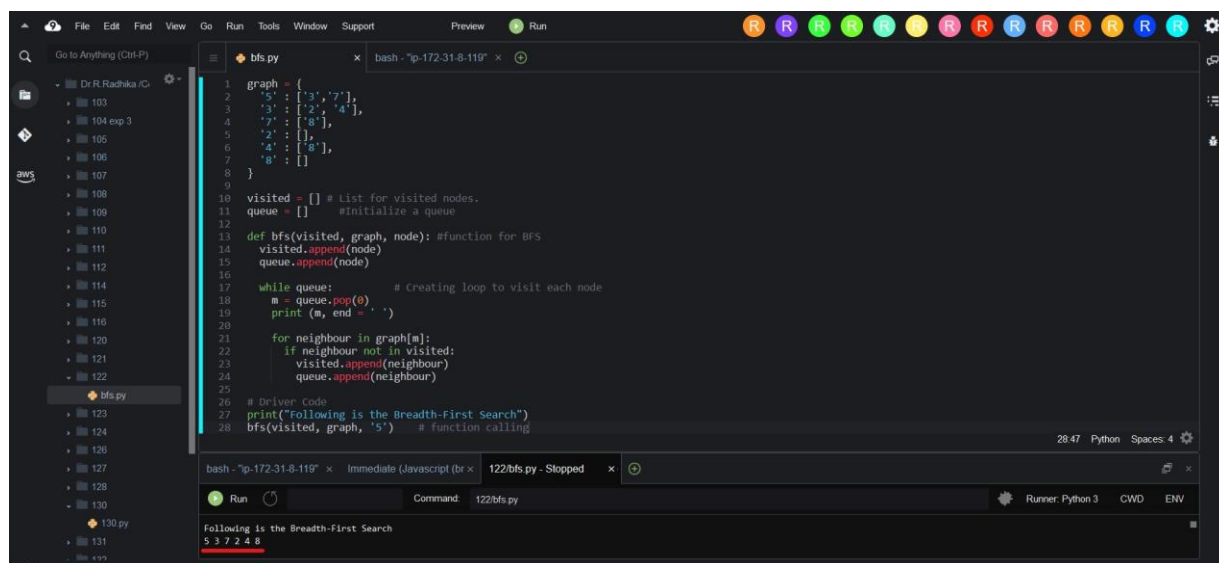
Implementation:

Screenshots



```
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
```

Output:



```
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
```

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

DFS (Depth First Search):

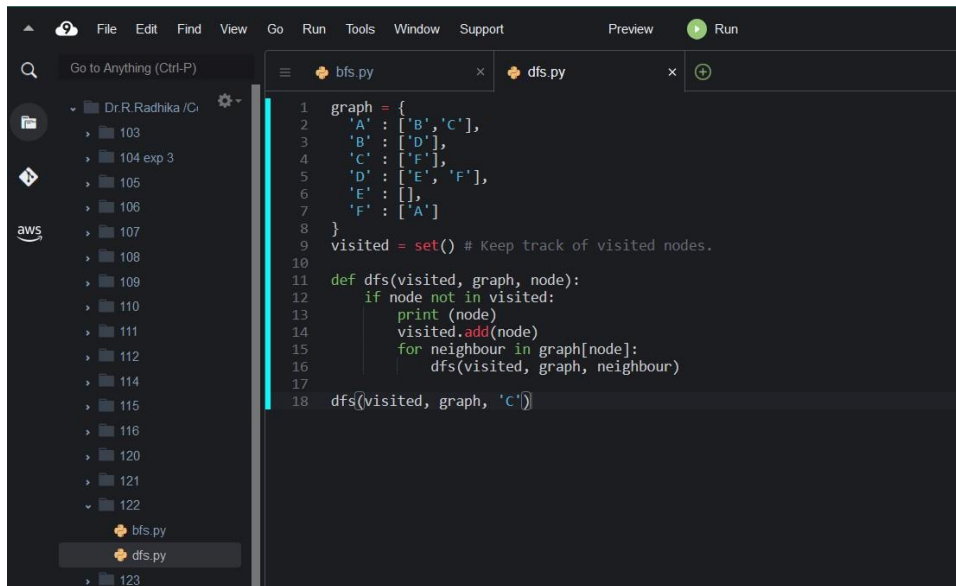
Depth-first search (DFS) is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root_node (Selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking.

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:

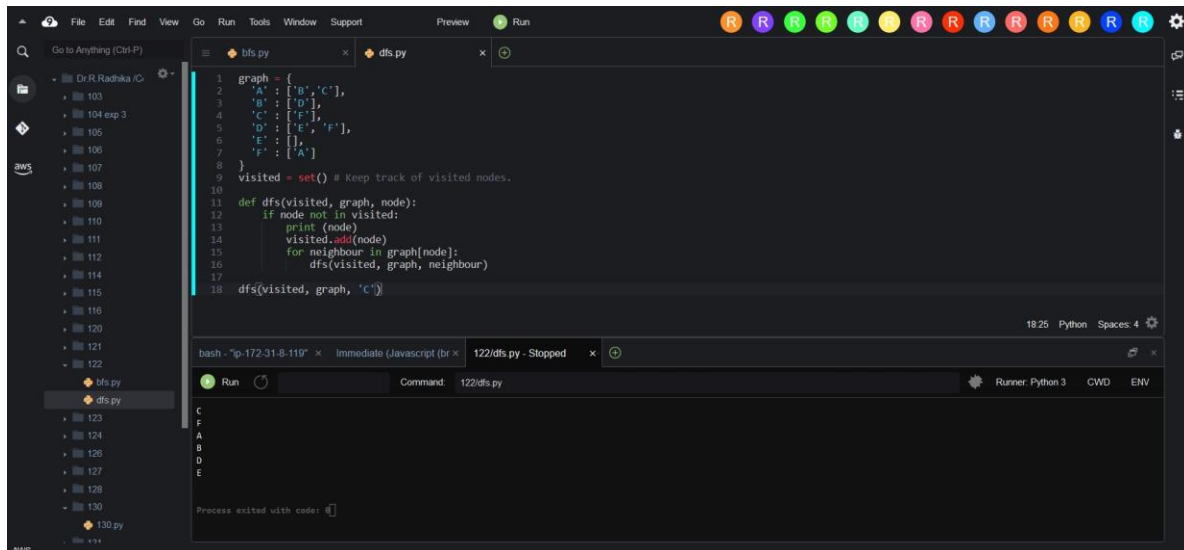
Screenshots



The screenshot shows an IDE with a file explorer on the left and a code editor on the right. The file explorer shows a directory structure with files named 103, 104 exp 3, 105, 106, 107, 108, 109, 110, 111, 112, 114, 115, 116, 120, 121, 122, bfs.py, and dfs.py. The code editor shows the implementation of a DFS algorithm on a graph. The graph is defined as a dictionary where keys are nodes and values are lists of neighbors. The DFS function is defined as follows:

```
1 graph = {
2     'A': ['B', 'C'],
3     'B': ['D'],
4     'C': ['F'],
5     'D': ['E', 'F'],
6     'E': [],
7     'F': ['A']
8 }
9 visited = set() # Keep track of visited nodes.
10
11 def dfs(visited, graph, node):
12     if node not in visited:
13         print (node)
14         visited.add(node)
15         for neighbour in graph[node]:
16             dfs(visited, graph, neighbour)
17
18 dfs(visited, graph, 'C')
```

Output:



The screenshot shows the same IDE as the previous one, but with the output of the DFS algorithm displayed in the console. The output is as follows:

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

The console also shows the command used to run the script: `122/dfs.py - Stopped`. The runner is Python 3, CWD, and ENV.