SRM INSTITUTE OF SCIENCE & TECHNOLOGY

DEPARTMENT OF NETWORKING & COMMUNICATIONS

18CSC305J-ARTIFICIAL INTELLIGENCE

SEMESTER – 6

BATCH-1

REGISTRATION NUMBER	**RA1911003011013**
NAME	**Shivank Arora**

B.Tech- CSE / CC, Third Year (Section: H2)

Faculty In charge: Dr. S. Prabakeran, B.Tech, M.E, PhD
Assistant Professor
School of Computing - Department of Networking and Communications

Year 2021-2022 / Even Semester

INDEX

Ex No	DATE	Title	Page No	Marks
1	08/02/22	**Implementation and Analysis of DFS and BFS for an application**		

Exercise: 1

Date : 8-02-2022

Implementation and Analysis of DFS and BFS .

Breadth-First Search : 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 :

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.

The Depth-First Search : The Depth-First Search 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 :

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.

Tool : Aws Cloud9 and Python 3.9.0

Programming code :

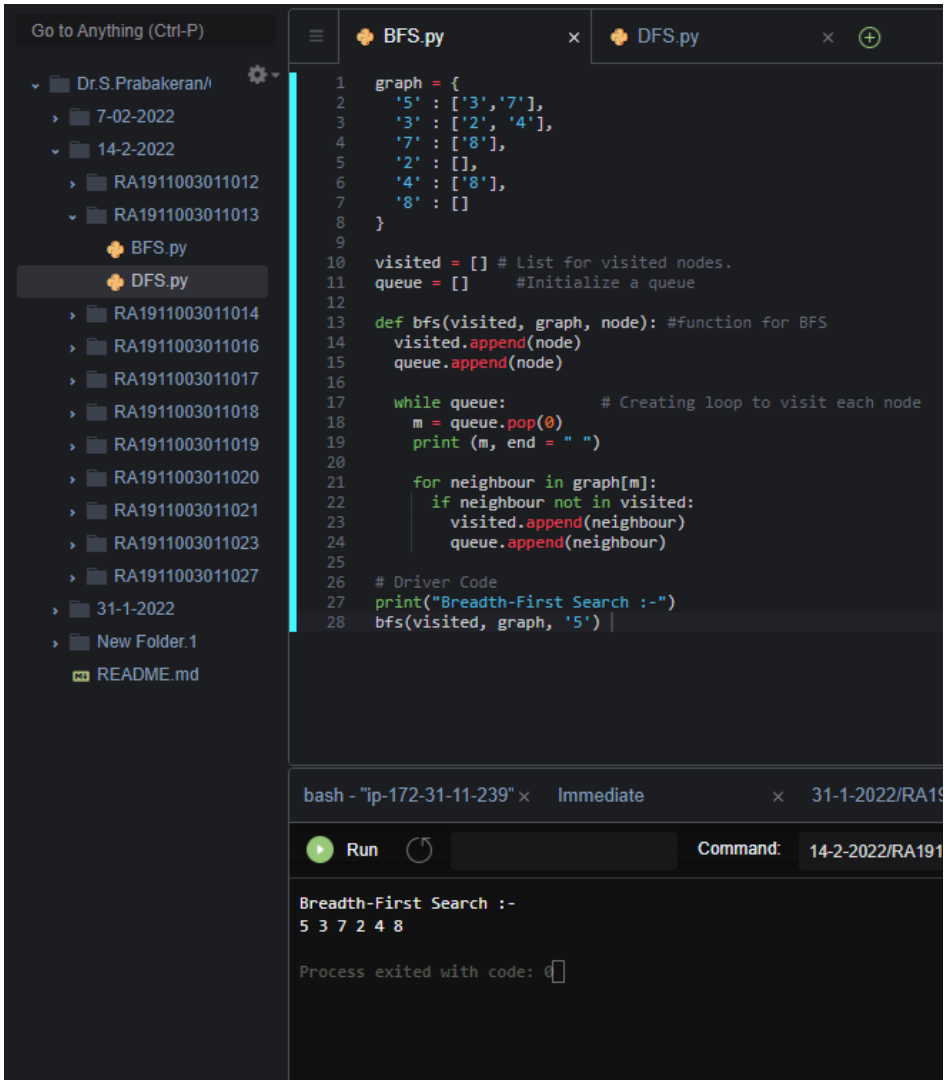
BFS:

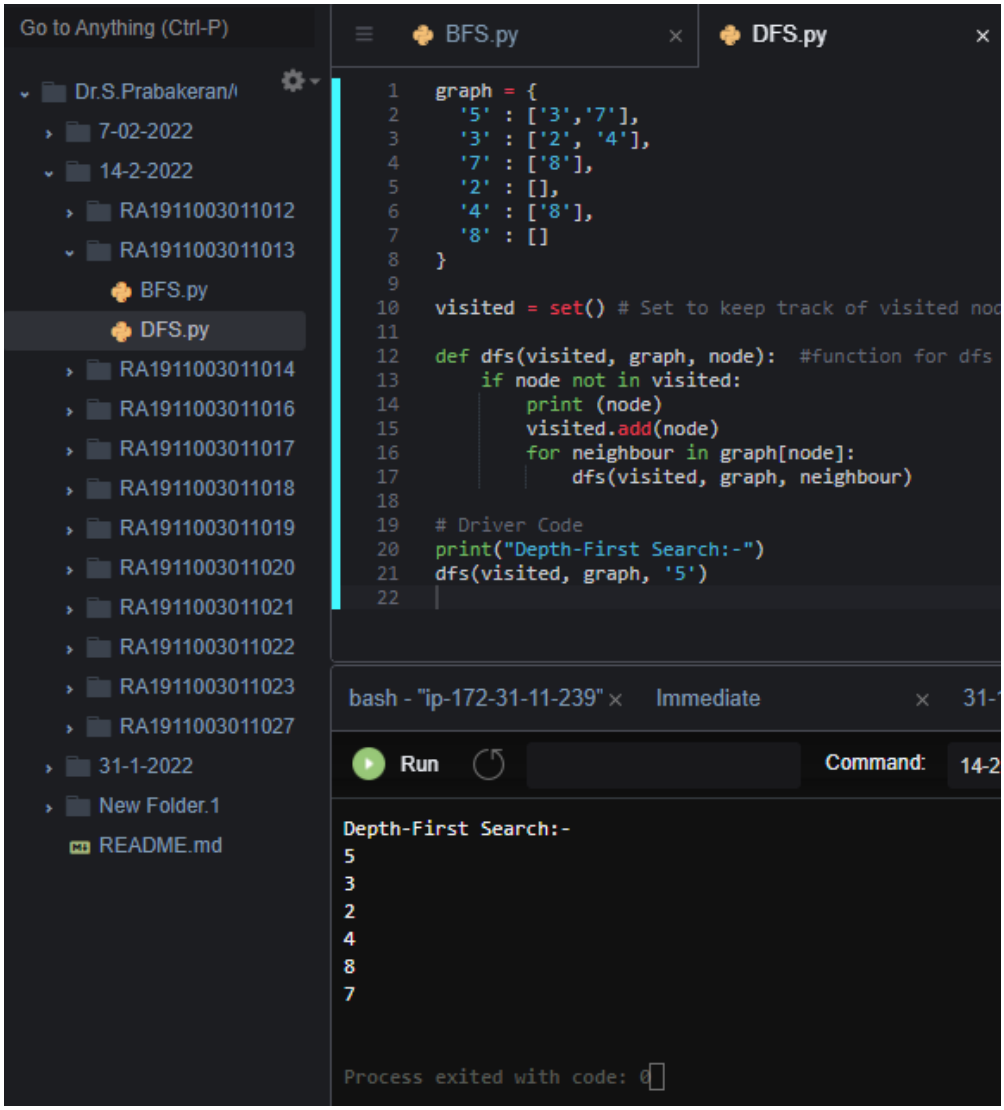
```
graph = {
  '5' : ['3','7'],
  '3' : ['2', '4'],
  '7' : ['8'],
  '2' : [],
  '4' : ['8'],
  '8' : []
}

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("Breadth-First Search:-")
bfs(visited, graph, '5')    # function calling
```

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("Depth-First Search:-")
dfs(visited, graph, '5')
```

Output screen shots :


Go to Anything (Ctrl-P)
Dr.S.Prabakeran/
7-02-2022
14-2-2022
RA1911003011012
RA1911003011013
BFS.py
DFS.py
RA1911003011014
RA1911003011016
RA1911003011017
RA1911003011018
RA1911003011019
RA1911003011020
RA1911003011021
RA1911003011023
RA1911003011027
31-1-2022
New Folder.1
README.md
BFS.py
DFS.py
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("Breadth-First Search :-")
bfs(visited, graph, '5')
bash - "ip-172-31-11-239"
Immediate
31-1-2022/RA1
Run
Command:
14-2-2022/RA191
Breadth-First Search :-
5 3 7 2 4 8
Process exited with code: 0



Result : Successfully Implemented BFS and DFS.