Exp-4

Rishabh Abhani

(RA1911003011023)

AIM:

PROGRAMMING DFS AND BFS IN GRAPH.

Code:

from collections import defaultdict

# This class represents a directed graph using

# adjacency list representation

class Graph:

# Constructor

def \_\_init\_\_(self):

# default dictionary to store graph

self.graph = defaultdict(list)

# function to add an edge to graph

def addEdge(self, u, v):

self.graph[u].append(v)

# A function used by DFS

def DFSUtil(self, v, visited):

# Mark the current node as visited

# and print it

visited.add(v)

print(v, end=' ')

# Recur for all the vertices

# adjacent to this vertex

for neighbour in self.graph[v]:

if neighbour not in visited:

self.DFSUtil(neighbour, visited)

# The function to do DFS traversal. It uses

# recursive DFSUtil()

def DFS(self, v):

# Create a set to store visited vertices

visited = set()

# Call the recursive helper function

# to print DFS traversal

self.DFSUtil(v, visited)

# Driver code

# Create a graph given

# in the above diagram

g = Graph()

g.addEdge(0, 1)

g.addEdge(0, 2)

g.addEdge(1, 2)

g.addEdge(2, 0)

g.addEdge(2, 3)

g.addEdge(3, 3)

print("Following is DFS from (starting from vertex 2)")

g.DFS(2)

BFS:

from collections import defaultdict

# This class represents a directed graph

# using adjacency list representation

class Graph:

# Constructor

def \_\_init\_\_(self):

# default dictionary to store graph

self.graph = defaultdict(list)

# function to add an edge to graph

def addEdge(self,u,v):

self.graph[u].append(v)

# Function to print a BFS of graph

def BFS(self, s):

# Mark all the vertices as not visited

visited = [False] \* (max(self.graph) + 1)

# Create a queue for BFS

queue = []

# Mark the source node as

# visited and enqueue it

queue.append(s)

visited[s] = True

while queue:

# Dequeue a vertex from

# queue and print it

s = queue.pop(0)

print (s, end = " ")

# Get all adjacent vertices of the

# dequeued vertex s. If a adjacent

# has not been visited, then mark it

# visited and enqueue it

for i in self.graph[s]:

if visited[i] == False:

queue.append(i)

visited[i] = True

# Driver code

# Create a graph given in

# the above diagram

g = Graph()

g.addEdge(0, 1)

g.addEdge(0, 2)

g.addEdge(1, 2)

g.addEdge(2, 0)

g.addEdge(2, 3)

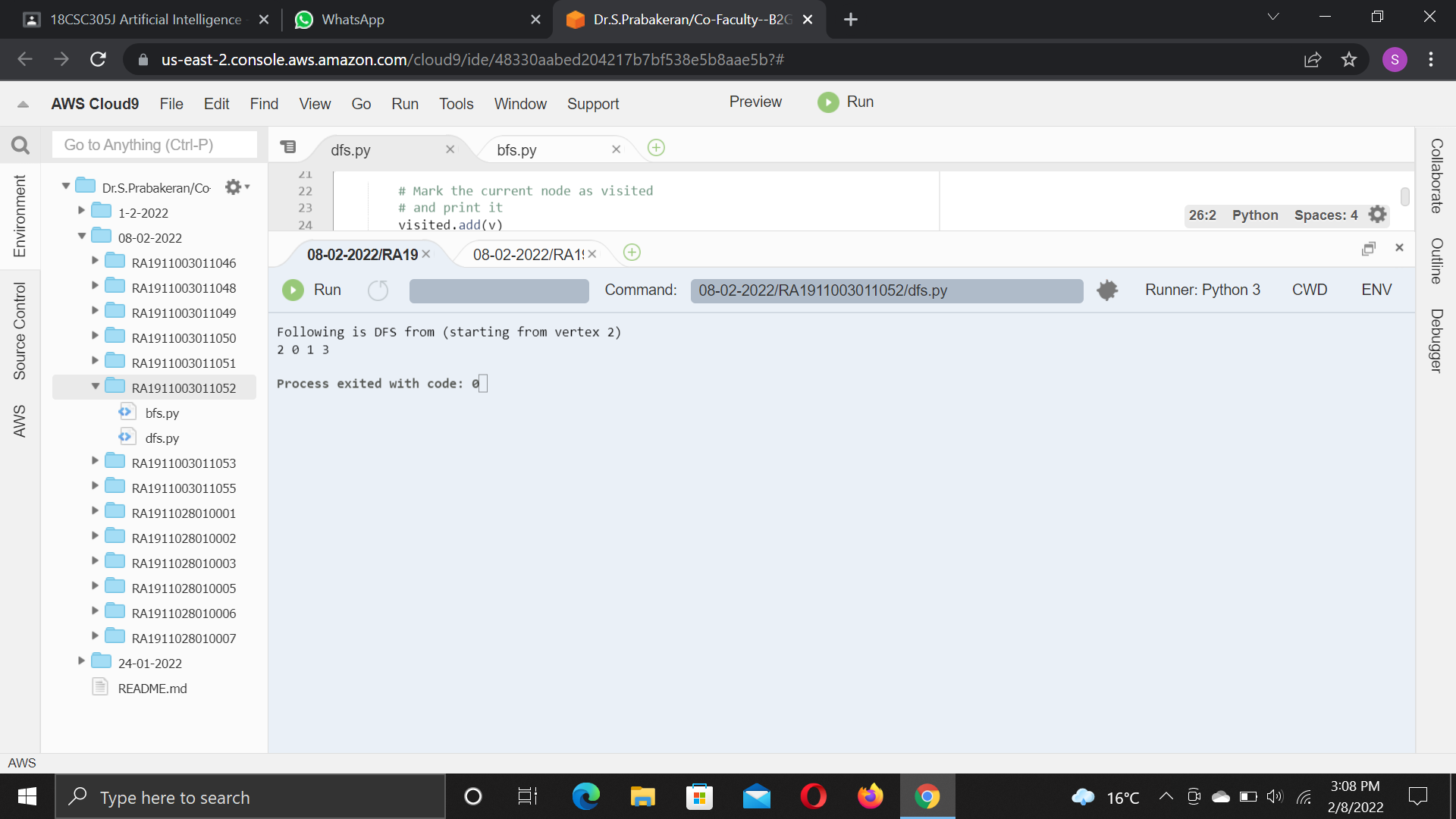
g.addEdge(3, 3)

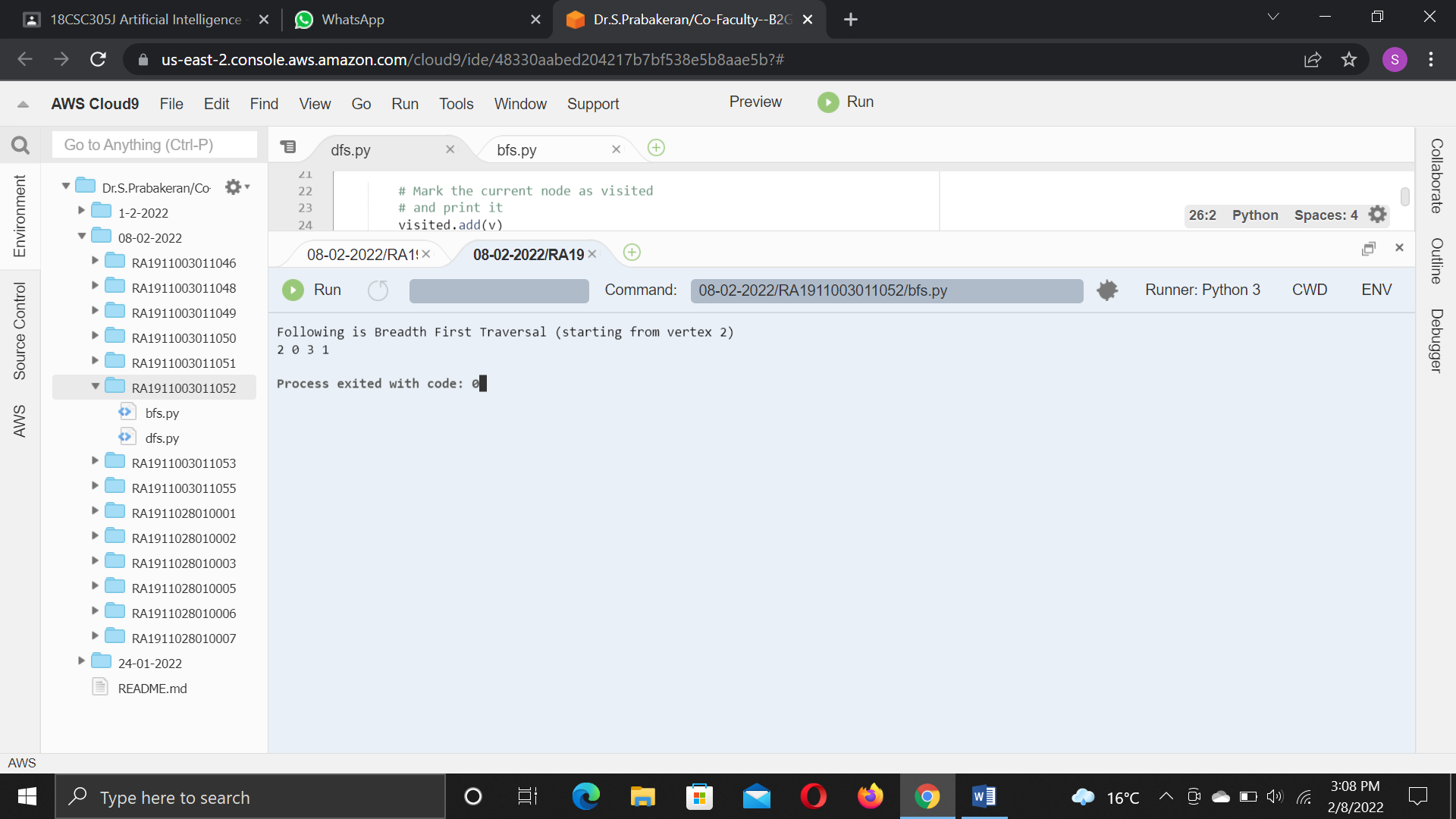
print ("Following is Breadth First Traversal"

" (starting from vertex 2)")

g.BFS(2)

OUTPUT:





Conclusion:

Program for DFS and BFS graph was executed successfully.