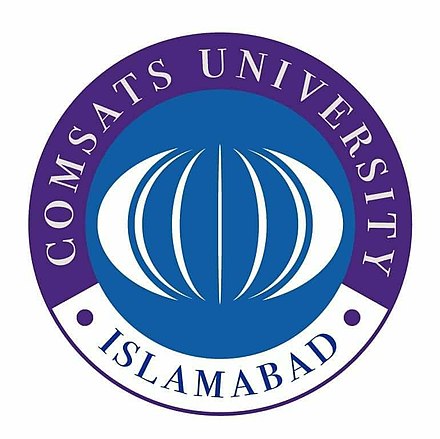
****

**Assignment 1**

**Name: Ayesha Aleem**

**ROLL-No:SP19-BCS-017**

**Section: B**

***CODE***

from queue import PriorityQueue

import matplotlib.pyplot as plt

from datetime import datetime

class Graph:

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

self.graph = {

"Caen": [(241, ("Caen", "Paris")), (120, ("Caen", "Calais")),

(176, ("Caen", "Rennes"))],

"Calais": [(120, ("Calais", "Caen")), (297, ("Calais", "Paris")),

(534, ("Calais", "Nancy"))],

"Nancy":

[(534, ("Nancy", "Calais")), (145, ("Nancy", "Strasbourg")),

(201, ("Nancy", "Dijon")), (372, ("Nancy", "Paris"))],

"Paris": [(241, ("Paris", "Caen")), (297, ("Paris", "Calais")),

(372, ("Paris", "Nancy")), (313, ("Paris", "Dijon")),

(396, ("Paris", "Limoges")), (348, ("Paris", "Rennes"))],

"Dijon": [(335, ("Dijon", "Strasbourg")), (192, ("Dijon", "Lyon")),

(313, ("Dijon", "Paris")), (201, ("Dijon", "Nancy"))],

"Lyon": [(192, ("Lyon", "Dijon")), (104, ("Lyon", "Grenoble")),

(216, ("Lyon", "Avignon")), (389, ("Lyon", "Limoges"))],

"Grenoble": [(104, ("Grenoble", "Lyon")),

(227, ("Grenoble", "Avignon"))],

"Avignon": [(121, ("Avignon", "Montpellier")),

(227, ("Avignon", "Grenoble")),

(99, ("Avignon", "Marseille")),

(216, ("Avignon", "Lyon"))],

"Marseille": [(99, ("Marseille", "Avignon")),

(188, ("Marseille", "Nice"))],

"Nice": [(188, ("Nice", "Marseille"))],

"Montpellier": [(188, ("Montpellier", "Toulouse")),

(121, ("Montpellier", "Avignon"))],

"Toulouse": [(241, ("Toulouse", "Montpellier")),

(313, ("Toulouse", "Limoges")),

(253, ("Toulouse", "Bocdeaux"))],

"Bocdeaux": [(253, ("Bocdeaux", "Toulouse")),

(220, ("Bocdeaux", "Limoges")),

(329, ("Bocdeaux", "Nantes"))],

"Nantes": [(329, ("Nantes", "Limoges")),

(329, ("Nantes", "Bocdeaux")),

(107, ("Nantes", "Rennes"))],

"Rennes": [(348, ("Rennes", "Paris")), (176, ("Rennes", "Caen")),

(107, ("Rennes", "Nantes")),

(244, ("Rennes", "Brest"))],

"Brest": [(244, ("Brest", "Rennes"))],

"Limoges": [(313, ("Limoges", "Toulouse")),

(220, ("Limoges", "Bocdeaux")),

(329, ("Limoges", "Nantes")),

(396, ("Limoges", "Paris")),

(389, ("Limoges", "Lyon"))],

"Strasbourg": [(335, ("Strasbourg", "Dijon")),

(145, ("Strasbourg", "Nancy"))]

}

self.heristics = {

"Caen": 10,

"Calais": 17,

"Nancy": 2,

"Paris": 14,

"Dijon": 13,

"Lyon": 12,

"Grenoble": 11,

"Avignon": 0,

"Marseille": 9,

"Nice": 8,

"Montpellier": 7,

"Toulouse": 5,

"Bocdeaux": 6,

"Nantes": 4,

"Rennes": 2,

"Brest": 3,

"Limoges": 1,

"Strasbourg": 2

}

self.edges = {}

self.weights = {}

self.Cal\_edges()

self.Cal\_weights()

print("edges : ", self.edges)

print("------------------------------------")

print("weights : ", self.weights)

print("------------------------------------")

def Cal\_edges(self):

for key in self.graph:

neighbours = []

for each\_tuple in self.graph[key]:

neighbours.append(each\_tuple[1][1])

self.edges[key] = neighbours

def Cal\_weights(self):

for key in self.graph:

neighbours = self.graph[key]

for each\_tuple in neighbours:

self.weights[each\_tuple[1]] = each\_tuple[0]

def neighbors(self, node):

return self.edges[node]

def get\_cost(self, node1, node2):

return self.weights[(node1, node2)]

def get\_heuristic(self, node):

return self.heristics[node]

def DFS(self, start):

#maintain the explored nodes

explored = []

#quene for pop the elements

quene = [start]

while len(quene) != 0:

node = quene.pop(0)

if node not in explored:

explored.append(node)

neighbours = self.neighbors(node)

neighbours.reverse()

for neighbour in neighbours:

quene.append(neighbour)

return explored

def BFS(self, start):

#maintain the explored nodes

explored = []

#quene for pop the elements

quene = [start]

while len(quene) != 0:

node = quene.pop(0)

if node not in explored:

explored.append(node)

neighbours = self.neighbors(node)

for neighbour in neighbours:

quene.append(neighbour)

return explored

def GS(graph, start, goal):

visited = []

queue = PriorityQueue()

queue.put((0, start))

while not queue.empty():

cost, node = queue.get()

while not queue.empty():

queue.get()

if node not in visited:

visited.append(node)

if node == goal:

break

for i in graph.neighbors(node):

if i not in visited:

queue.put((graph.get\_heuristic(i), i))

return visited

def UCS(graph, start, goal):

visited = []

queue = PriorityQueue()

queue.put((0, start))

while not queue.empty():

cost,node = queue.get()

while not queue.empty():

queue.get()

if node not in visited:

visited.append(node)

if node == goal:

break

for i in graph.neighbors(node):

if i not in visited:

queue.put((graph.get\_cost(node,i), i))

return visited

def Astar(graph, start, goal):

visited = []

queue = PriorityQueue()

queue.put((0, start))

while not queue.empty():

cost, node = queue.get()

while not queue.empty():

queue.get()

if node not in visited:

visited.append(node)

if node == goal:

break

for i in graph.neighbors(node):

if i not in visited:

total\_cost = graph.get\_cost(node, i) + graph.get\_heuristic(node)

queue.put((total\_cost, i))

return visited

print("------------------------------------")

Start = datetime.now()

print("DFS:", DFS(Graph(), "Caen"))

End = datetime.now()

print("-----------------------------------")

Start1 = datetime.now()

print("BFS:", BFS(Graph(), "Brest"))

End1 = datetime.now()

print("\_\_\_\_\_\_\_\_\_\_\_")

Start2 = datetime.now()

print("GS:", GS(Graph(), "Brest","Limoges"))

End2 = datetime.now()

print("\_\_\_\_\_\_\_")

Start3 = datetime.now()

print("UCS",UCS(Graph(),"Avignon","Paris"))

End3 = datetime.now()

print("\_\_\_\_\_\_\_\_\_\_\_")

Start4 = datetime.now()

print("A star:", Astar(Graph(), "Toulouse","Nice"))

End4 = datetime.now()

a=len(DFS(Graph(),"Rennes"))

b=len(BFS(Graph(),"Paris"))

c=len(GS(Graph(),"Brest","Limoges"))

d=len(UCS(Graph(),"Dijon","Rennes"))

e=len(Astar(Graph(), "Toulouse","Nice"))

x=['dfs','bfs','GS','UCS','A-star']

y=[a,b,c,d,e]

plt.bar(x,y)

plt.xlabel('Algorithm name')

plt.ylabel('Nodes Visited')

plt.title("Nodes")

plt.show()

Algotime = []

f = End-Start

g = End1-Start1

h = End2-Start2

i = End3-Start3

j = End4-Start4

k=f.total\_seconds()

l=g.total\_seconds()

m=h.total\_seconds()

n=i.total\_seconds()

o=j.total\_seconds()

x=['dfs','bfs','GS','UCS','A-star']

y=[k,l,m,n,o]

plt.title("Complexity")

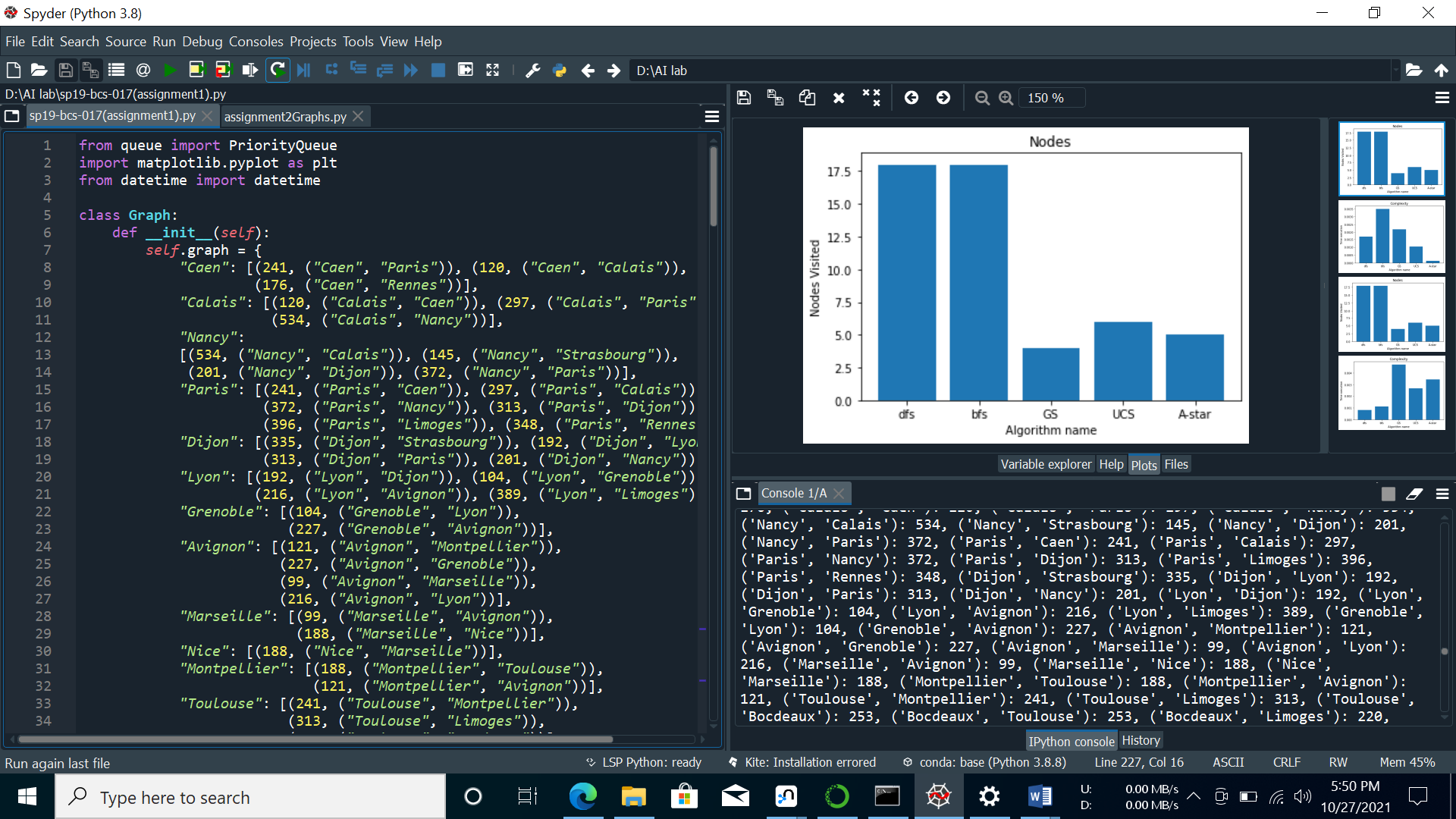
plt.bar(x,y)

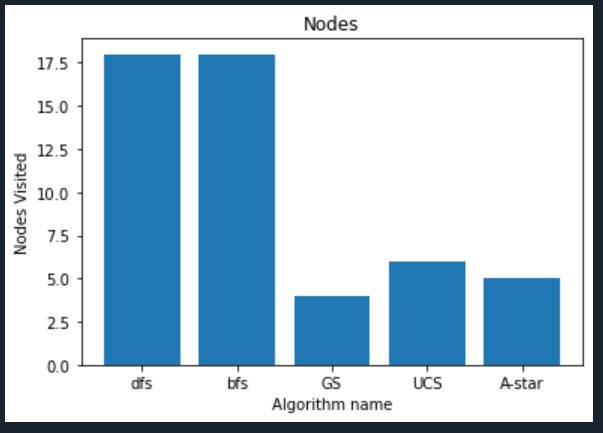
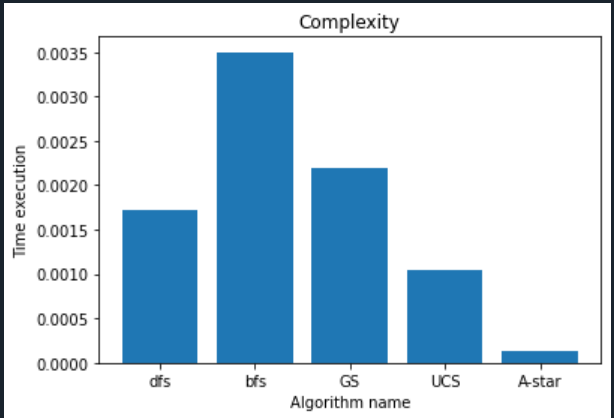
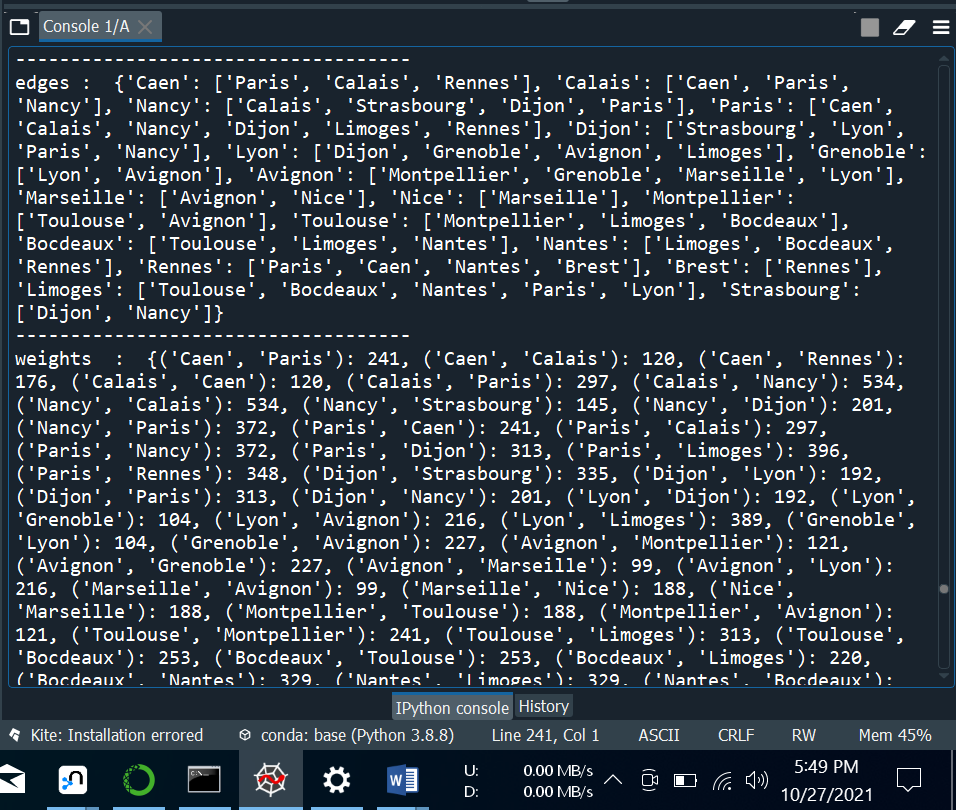
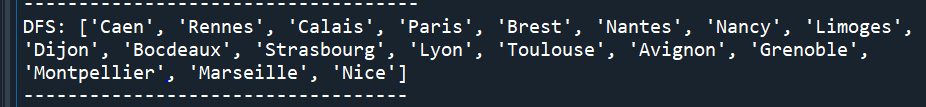
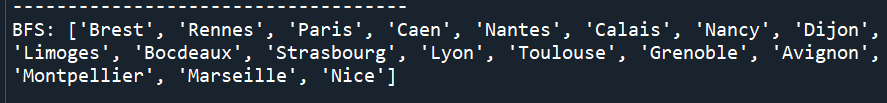
plt.xlabel('Algorithm name')

plt.ylabel('Time execution')

plt.show()

***Output***

****

****