**Bidirectional Code :**

# -\*- coding: utf-8 -\*-

"""

Created on Mon Sep 19 12:21:25 2016

@author: jaysh

"""

# -\*- coding: utf-8 -\*-

"""

Created on Mon Sep 19 11:27:08 2016

@author: jaysh

"""

# This is a Bidirectional Search

# The problem of the program is stored in a list

# The program is always complete if the goal is present

# Variable path represents the path from Arad towards Bucharest

# Variable path1 represents the path from bucharest towards Arad

# Variable s represent the current state of the search from Arad towards Bucharest

# Variable s1 represents the current state of the search from Bucharest to Arad

# This Program is implemented by a stack where the last node from the frontier is taken out and the new node is appended in the last of frontier

#intialise\_frontier = This function initialises the frontier nodes to the start node i.e in this case Arad

#choose\_node = This function chooses the last state from the frontier its like implementing the half of the stack the last node choosen becomes the current state

#update\_frontier = This fucntion updates frontier by inserting those nodes in the frontier which are present in g but they are not explored Basically froniter is the union of the list g and explored

#test\_goal = his function compares the current node with the goal node if the current node is the goal node than it sends True else it sends False

#expand\_node = #This fucntion Expands the current state It shows all the possibility that can be explored from the current state and it also shows the cost of that possibility

#calculate\_path\_cost = #This function back tracks and calculates the total path cost friom arad to bucharest

def graph\_search(start,start1,goal,goal1,problem):

explored = [] #Represents the states which has been already explored

explored1 = []

g = [] #Represents the nodes and one of the nodes will be explored from Arad toward Bucharest

g1 = [] #Represents the nodes and one of the nodes will be explored from Bucharest to Arad

path = [] #Represents the path of the graph from arad to Bucharest

path1 = []#Represents the path of the graph from Bucharest to Arad

current\_state\_A2B='' #Represents the current state from Arad towards Bucharest

current\_state\_B2A=''#Represents the current state from Bucharest towards Arad

frontier = [] #Represents the frontier nodes from Arad to Bucharest

frontier1 = [] #Represents the frontier nodes from Bucharest to Arad

frontier,frontier1=intialise\_frontier(start,start1)

print ('\n Initialisation value of frontier = '+", ".join(frontier))

print ('\n Initialisation value of frontier1 = '+", ".join(frontier1))

while frontier:

path.append(choose\_node(frontier))

path1.append(choose\_node(frontier1))

print ('\n The value of current state = '+path[-1])

print ('\n The value of current state1 = '+path1[-1])

length\_of\_path = len(path)

length\_of\_path1 = len(path1)

if(length\_of\_path > 1):

print ("\nThe parent node of current state "+path[-1]+" is "+path[-2])

print ("\n path value= "+", ".join(path))

if(length\_of\_path1 > 1):

print ("\nThe parent1 node of current state1 "+path1[-1]+" is "+path1[-2])

print ("\n path value1= "+", ".join(path1))

current\_state\_A2B= path[-1]

current\_state\_B2A= path1[-1]

print ('\n frontier bfore updation = '+", ".join(frontier))

print ('\n frontier1 bfore updation = '+", ".join(frontier1))

explored.append(current\_state\_A2B)

explored1.append(current\_state\_B2A)

print ('\n explored node = '+", ".join(explored))

print ('\n explored1 node = '+", ".join(explored1))

if(test\_stop(current\_state\_A2B,current\_state\_B2A,goal,goal1) == False):

print ("\n The curent state is NOT the goal state ")

g = expand\_node(g,problem,current\_state\_A2B,goal,start,start1)

g1 = expand\_node(g1,problem,current\_state\_B2A,goal1,start,start1)

update\_frontier(g,explored,frontier)

update\_frontier(g1,explored1,frontier1)

print ('\n frontier after updation = '+", ".join(frontier))

print ('\n frontier1 after updation = '+", ".join(frontier1))

else:

print ("\n The curent state is the stop state ")

break

g=[]

g1=[]

total\_path\_cost = calculate\_path\_cost(path,problem)

total\_path\_cost1 = calculate\_path\_cost1(path1,problem)

final\_path =[]

final\_explored = ', '.join(explored)

final\_explored1 = ', '.join(explored1)

for elements in path:

if elements not in path1:

final\_path.append(elements)

else:

break

i=1

for elements in path1:

final\_path.append(path1[-i])

i=i+1

final\_path\_cost = total\_path\_cost1 + total\_path\_cost

print("\n FINAL OUTPTUT ----------------------------------------------------")

print("\n The Final Explored states from Arad to Bucharest are :\n "+final\_explored)

print("\n The Final Explored states from Bucharest to Arad are :\n "+final\_explored1)

print ("\n The Path from Arad to Bucharest Bidirectional Search : "+', '.join(path))

print (" Path Cost from Arad To Bucharest = "+str(total\_path\_cost))

print ("\n The Path from Bucharest to Arad using Bidirectional Search : \n "+', '.join(path1))

print (" Path Cost from Bucharest To Arad = "+str(total\_path\_cost1))

print ("\n The joint from Arad to Bucharest using Bidirectional Search : \n "+', '.join(final\_path))

print ("\n Final Path Cost from Arad To Bucharest = "+str(final\_path\_cost))

print("\n FINAL OUTPTUT----------------------------------------------------")

#This function initialises the frontier node to the start node

def intialise\_frontier(start,start1):

frontier = [start]

frontier1=[start1]

return frontier,frontier1

#This function chooses the first state from the frontier its like

#implementing the half of the queue

#The first node choosen becomes the current state

def choose\_node(frontier):

path = frontier.pop()

return path

#This fucntion updates frontier by

#inserting those nodes in the frontier

#which are present in g but they are not explored

#Basically froniter is the union of the list g and explored

def update\_frontier(g,explored,frontier):

for element in g:

if element not in explored:

frontier.append(element)

return frontier

#This function compares the current node with the goal node

# and also compares the condition whether bot the searches have reached a common node

#if the current node is the goal node or both the searches have reached a commmon node than it sends True else it sends False

def test\_stop(s,s1,goal,goal1):

if (s==goal):

return True

elif(s1==goal1):

return True

elif(s==s1):

return True

return False

#This function Expands the current state

# It shows all the possibility that can be explored from the current state

# and it also shows the cost of that possibility

def expand\_node(g,problem,s,goal,start,start1):

if goal==start1:

for place\_distance\_list in problem :

if place\_distance\_list[0] == s :

print ("\n The distance to be explored is :"+place\_distance\_list[0]+" to "+place\_distance\_list[1])

print ("The cost of the above distance to be explored is :"+str(place\_distance\_list[2]))

g.append(place\_distance\_list[1])

elif goal==start:

for place\_distance\_list in problem :

if place\_distance\_list[1] == s :

print ("\n The distance to be explored is :"+place\_distance\_list[1]+" to "+place\_distance\_list[0])

print ("The cost of the above distance to be explored is :"+str(place\_distance\_list[2]))

g.append(place\_distance\_list[0])

return g

#This function back tracks and calculates the

#total path cost friom arad to bucharest

def calculate\_path\_cost(path,problem):

length\_of\_path = len(path)

i=0

path\_cost=0

lister=[]

while i<length\_of\_path-1 :

for place\_distance\_list in problem:

if path[i] == place\_distance\_list[0] and path[i+1] == place\_distance\_list[1] :

lister.append(place\_distance\_list[2])

i=i+1

print (lister)

for ele in lister:

path\_cost=path\_cost+ele

return path\_cost

#This function back tracks and calculates the

#total path cost friom arad to Bucharest to Arad

def calculate\_path\_cost1(path,problem):

length\_of\_path = len(path)

i=0

path\_cost=0

lister=[]

while i<length\_of\_path-1 :

for place\_distance\_list in problem:

if path[i] == place\_distance\_list[1] and path[i+1] == place\_distance\_list[0] :

lister.append(place\_distance\_list[2])

i=i+1

print (lister)

for ele in lister:

path\_cost=path\_cost+ele

return path\_cost

problem = [ ['Arad','Zerind',75], ['Arad','Timisoara',118], ['Arad','Sibiu',140], ['Zerind','Oradea',71],

['Oradea','Sibiu',151], ['Timisoara','Lugoj',111], ['Lugoj','Mehadia',70], ['Mehadia','Drobeta',75],

['Drobeta','Craiova',120], ['Craiova','Rimnicu-Vilcea',146], ['Craiova','Pitesti',138],

['Sibiu','Fagaras',99], ['Sibiu','Rimnicu-Vilcea',80], ['Rimnicu-Vilcea','Pitesti',97],

['Fagaras','Bucharest',211], ['Pitesti','Bucharest',101],['Bucharest','Urziceni',85],

['Bucharest Giurgiu',90], ['Urziceni Vaslui',142], ['Vaslui Iasi',92], ['Iasi Neamt',87],

['Urziceni Hirsova',98], ['Hirsova Eforie',86] ]

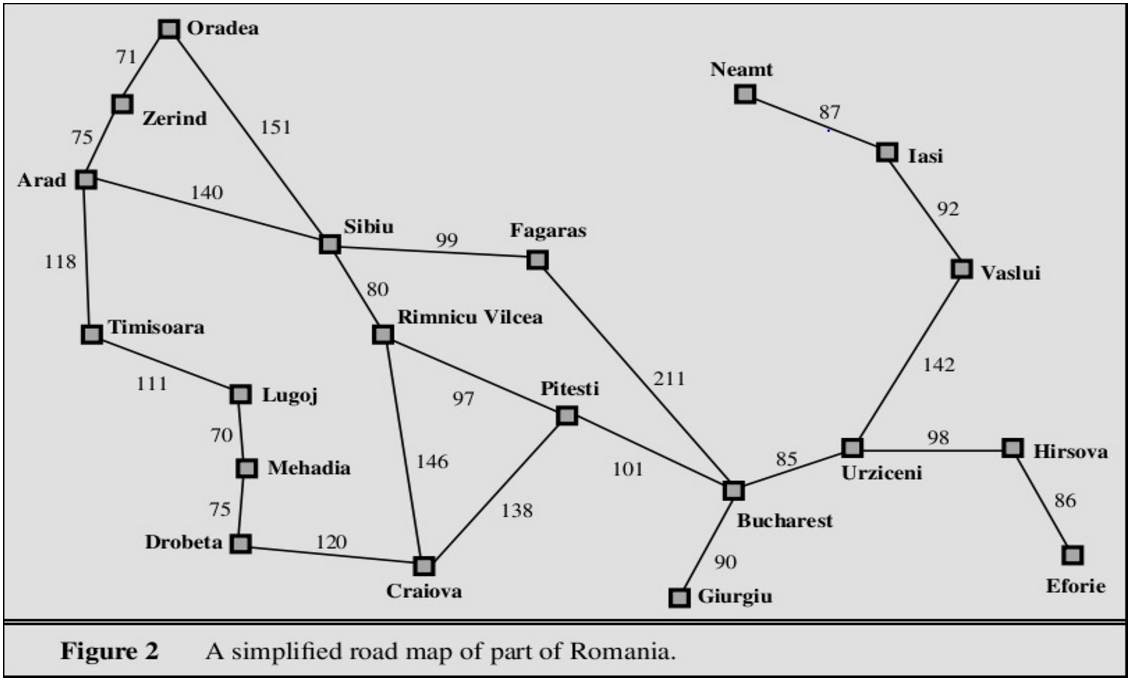
start = "Arad"

goal = "Bucharest"

start1 = goal

goal1 = start

graph\_search(start,start1,goal,goal1,problem)



**OUTPUT :**

Initialisation value of frontier = Arad

Initialisation value of frontier1 = Bucharest

The value of current state = Arad

The value of current state1 = Bucharest

path value= Arad

path value1= Bucharest

frontier bfore updation =

frontier1 bfore updation =

explored node = Arad

explored1 node = Bucharest

The curent state is NOT the goal state

The distance to be explored is :Arad to Zerind

The cost of the above distance to be explored is :75

The distance to be explored is :Arad to Timisoara

The cost of the above distance to be explored is :118

The distance to be explored is :Arad to Sibiu

The cost of the above distance to be explored is :140

The distance to be explored is :Bucharest to Fagaras

The cost of the above distance to be explored is :211

The distance to be explored is :Bucharest to Pitesti

The cost of the above distance to be explored is :101

frontier after updation = Zerind, Timisoara, Sibiu

frontier1 after updation = Fagaras, Pitesti

The value of current state = Sibiu

The value of current state1 = Pitesti

The parent node of current state Sibiu is Arad

path value= Arad, Sibiu

The parent1 node of current state1 Pitesti is Bucharest

path value1= Bucharest, Pitesti

frontier bfore updation = Zerind, Timisoara

frontier1 bfore updation = Fagaras

explored node = Arad, Sibiu

explored1 node = Bucharest, Pitesti

The curent state is NOT the goal state

The distance to be explored is :Sibiu to Fagaras

The cost of the above distance to be explored is :99

The distance to be explored is :Sibiu to Rimnicu-Vilcea

The cost of the above distance to be explored is :80

The distance to be explored is :Pitesti to Craiova

The cost of the above distance to be explored is :138

The distance to be explored is :Pitesti to Rimnicu-Vilcea

The cost of the above distance to be explored is :97

frontier after updation = Zerind, Timisoara, Fagaras, Rimnicu-Vilcea

frontier1 after updation = Fagaras, Craiova, Rimnicu-Vilcea

The value of current state = Rimnicu-Vilcea

The value of current state1 = Rimnicu-Vilcea

The parent node of current state Rimnicu-Vilcea is Sibiu

path value= Arad, Sibiu, Rimnicu-Vilcea

The parent1 node of current state1 Rimnicu-Vilcea is Pitesti

path value1= Bucharest, Pitesti, Rimnicu-Vilcea

frontier bfore updation = Zerind, Timisoara, Fagaras

frontier1 bfore updation = Fagaras, Craiova

explored node = Arad, Sibiu, Rimnicu-Vilcea

explored1 node = Bucharest, Pitesti, Rimnicu-Vilcea

The curent state is the stop state

[140, 80]

[101, 97]

FINAL OUTPTUT ----------------------------------------------------

The Final Explored states from Arad to Bucharest are :

Arad, Sibiu, Rimnicu-Vilcea

The Final Explored states from Bucharest to Arad are :

Bucharest, Pitesti, Rimnicu-Vilcea

The Path from Arad to Bucharest Bidirectional Search : Arad, Sibiu, Rimnicu-Vilcea

Path Cost from Arad To Bucharest = 220

The Path from Bucharest to Arad using Bidirectional Search :

Bucharest, Pitesti, Rimnicu-Vilcea

Path Cost from Bucharest To Arad = 198

The joint from Arad to Bucharest using Bidirectional Search :

Arad, Sibiu, Rimnicu-Vilcea, Pitesti, Bucharest

Final Path Cost from Arad To Bucharest = 418

FINAL OUTPTUT----------------------------------------------------