**UnifromCost Code:**

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

"""

Created on Thu Sep 15 12:49:48 2016

@author: jaysh

"""

#This is Uniform Cost search

# It is implemented by a priority queue where the node with the minimum value is always selected

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

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

g = {} #Represents the nodes and one of the nodes will be explored

path = [] #Represents the path of the graph

current\_state='' #Represents the current state

path\_cost=0 #Represents the cost of the path

flag=True

parent = {}

frontier,parent=intialise\_frontier(start,problem,parent)

path.append(start)

explored.append(start)

while frontier:

minimum,frontier,minimum\_key,minimum\_value,last\_key=first\_update\_frontier(frontier,path)

path.append(last\_key)

print("\n Path VALUE : "+", ".join(path))

current\_state= path[-1]

print ('\n The value of current state = '+current\_state)

blank = []

for keys in frontier:

split = keys.split(" ")

blank.append(split[1])

print ('\n frontier bfore updation = ')

print(blank)

explored.append(current\_state)

print ('\n explored states :'+", ".join(explored))

if(test\_goal(current\_state,goal) == False):

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

g,path\_cost,flag,parent=expand\_node(g,problem,current\_state,path\_cost,goal,flag,parent)

update\_frontier(g,explored,frontier,minimum\_value)

blank = []

for keys in frontier:

split = keys.split(" ")

blank.append(split[1])

print ('\n frontier after updation = ')

print (blank)

else:

print ("\n the current state is the goal state ")

break

g={}

print(frontier)

print(parent)

reverse\_path = [goal]

child=goal

cost=0

while start not in reverse\_path:

for elem in parent :

elements = elem.split(" ")

if elements[1]==child:

reverse\_path.append(elements[0])

cost = cost + parent[elem]

child=elements[0]

print("\n")

print(reverse\_path)

actual\_path = reverse\_path[::-1]

final\_path = ', '.join(actual\_path)

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

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

print("\n The Final Explored states are :"+final\_explored)

print ("\n The Uniform Cost from Arad to Bucharest is :"+ str(minimum\_value))

print ("\n The Path from Arad to Bucharest using Uniform Cost Search :\n "+final\_path)

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

def backtracking(start,goal,parent):

reverse\_path = [goal]

child = goal

cost = 0

while start not in reverse\_path:

for elements in parent:

if elements[1]==child:

reverse\_path.append(elements[0])

cost = cost + elements[2]

child=elements[0]

break

return reverse\_path,cost

#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,minimum\_value):

for element in g:

a = element.split(" ")

if a[1] not in explored:

frontier.update({element:g[element]+minimum\_value})

else :

for key in frontier:

if key == a[1] and g[element]+minimum\_value < frontier[key]+minimum\_value :

frontier[key] = g[element]+minimum\_value

return frontier

#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

def expand\_node(g,problem,s,path\_cost,goal,flag,parent):

for keys in problem :

a = (keys.split(' '))

if a[0] == s:

present = {keys:problem[keys]}

g.update(present)

parent.update(present)

if a[1] == goal:

flag = False

path\_cost=path\_cost+problem[keys]

return g,path\_cost,flag,parent

#This function initialises the frontier node to the start node

def intialise\_frontier(start,problem,parent):

# frontier = {"hey "+start:0}

frontier = {}

for keys in problem :

a = keys.split(" ")

if a[0] == start:

present = {keys:problem[keys]}

frontier.update(present)

parent.update(present)

return frontier ,parent

#It finds the dictionary which is to be appended to the frontier dictionary

def finding\_dicitonary(frontier,problem):

d1 = {}

d2={}

i=0

j=0

for ele in frontier:

b = ele.split(" ")

for key in problem:

a = key.split(" ")

if a[0] == b[1]:

d1={key:problem[key]}

d2.update(d1)

j=j+1

i=i+1

return d2

#This function updates the frontier dictionary

# it calculates the state with minimum cost which will be explored

def first\_update\_frontier(frontier,path):

minimum\_value = 1000000

for keys in frontier:

if(frontier[keys]<minimum\_value):

minimum\_key=keys

minimum\_value=frontier[keys]

print("\n The Action is to expand : "+minimum\_key.split(" ")[1]+" because it has the lowest cost "+str(minimum\_value))

print("\n The Parent node of "+minimum\_key.split(" ")[1]+" is "+path[-1] )

print("\n minimum value ="+str(minimum\_value))

print(frontier)

del frontier[minimum\_key]

print(frontier)

minimum = {minimum\_key:minimum\_value}

a = minimum\_key.split(" ")

return minimum,frontier,minimum\_key,minimum\_value,a[1]

#This 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

def test\_goal(s,goal):

if (s==goal):

return True

return False

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,

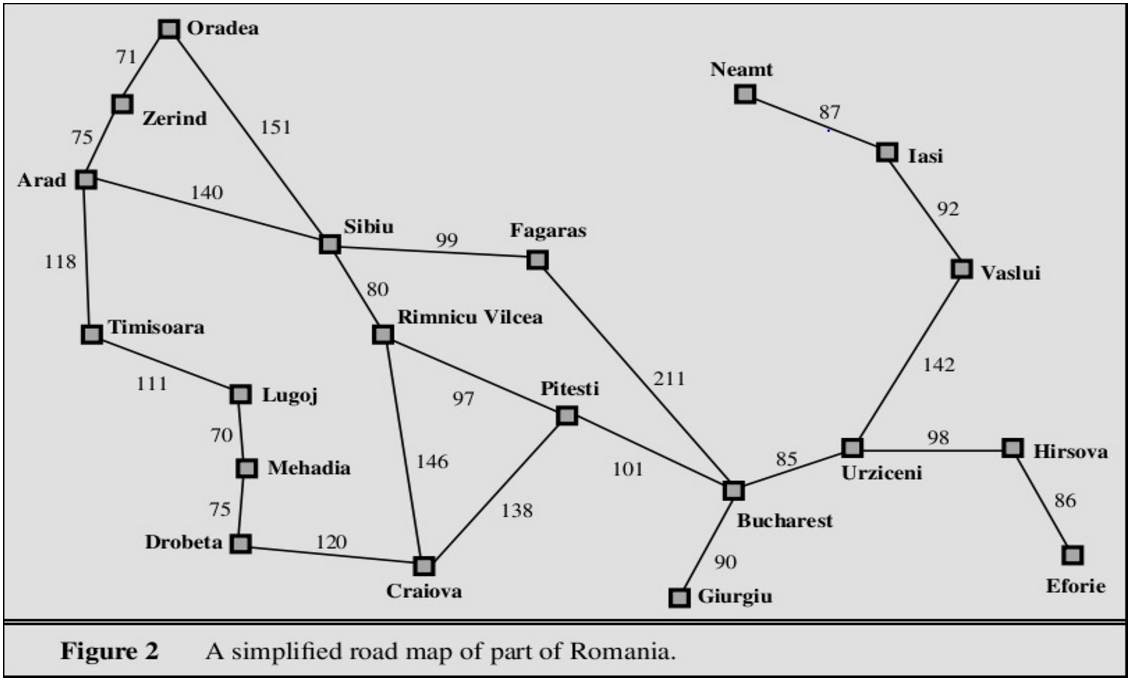
'Bucharest Urziceni':85, 'Bucharest Giurgiu':90, 'Urziceni Vaslui':142, 'Vaslui Iasi':92, 'Iasi Neamt':87, 'Urziceni Hirsova':98,

'Hirsova Eforie':86}

start = "Arad"

goal = "Bucharest"

graph\_search(start,goal,problem)



**OUTPUT :**

The Action is to expand : Zerind because it has the lowest cost 75

The Parent node of Zerind is Arad

minimum value =75

{'Arad Zerind': 75, 'Arad Sibiu': 140, 'Arad Timisoara': 118}

{'Arad Sibiu': 140, 'Arad Timisoara': 118}

Path VALUE : Arad, Zerind

The value of current state = Zerind

frontier bfore updation =

['Sibiu', 'Timisoara']

explored states :Arad, Zerind

The curent state is NOT the goal state

frontier after updation =

['Oradea', 'Sibiu', 'Timisoara']

The Action is to expand : Timisoara because it has the lowest cost 118

The Parent node of Timisoara is Zerind

minimum value =118

{'Zerind Oradea': 146, 'Arad Sibiu': 140, 'Arad Timisoara': 118}

{'Zerind Oradea': 146, 'Arad Sibiu': 140}

Path VALUE : Arad, Zerind, Timisoara

The value of current state = Timisoara

frontier bfore updation =

['Oradea', 'Sibiu']

explored states :Arad, Zerind, Timisoara

The curent state is NOT the goal state

frontier after updation =

['Oradea', 'Sibiu', 'Lugoj']

The Action is to expand : Sibiu because it has the lowest cost 140

The Parent node of Sibiu is Timisoara

minimum value =140

{'Zerind Oradea': 146, 'Arad Sibiu': 140, 'Timisoara Lugoj': 229}

{'Zerind Oradea': 146, 'Timisoara Lugoj': 229}

Path VALUE : Arad, Zerind, Timisoara, Sibiu

The value of current state = Sibiu

frontier bfore updation =

['Oradea', 'Lugoj']

explored states :Arad, Zerind, Timisoara, Sibiu

The curent state is NOT the goal state

frontier after updation =

['Oradea', 'Lugoj', 'Rimnicu-Vilcea', 'Fagaras']

The Action is to expand : Oradea because it has the lowest cost 146

The Parent node of Oradea is Sibiu

minimum value =146

{'Zerind Oradea': 146, 'Timisoara Lugoj': 229, 'Sibiu Rimnicu-Vilcea': 220, 'Sibiu Fagaras': 239}

{'Timisoara Lugoj': 229, 'Sibiu Rimnicu-Vilcea': 220, 'Sibiu Fagaras': 239}

Path VALUE : Arad, Zerind, Timisoara, Sibiu, Oradea

The value of current state = Oradea

frontier bfore updation =

['Lugoj', 'Rimnicu-Vilcea', 'Fagaras']

explored states :Arad, Zerind, Timisoara, Sibiu, Oradea

The curent state is NOT the goal state

frontier after updation =

['Lugoj', 'Rimnicu-Vilcea', 'Fagaras']

The Action is to expand : Rimnicu-Vilcea because it has the lowest cost 220

The Parent node of Rimnicu-Vilcea is Oradea

minimum value =220

{'Timisoara Lugoj': 229, 'Sibiu Rimnicu-Vilcea': 220, 'Sibiu Fagaras': 239}

{'Timisoara Lugoj': 229, 'Sibiu Fagaras': 239}

Path VALUE : Arad, Zerind, Timisoara, Sibiu, Oradea, Rimnicu-Vilcea

The value of current state = Rimnicu-Vilcea

frontier bfore updation =

['Lugoj', 'Fagaras']

explored states :Arad, Zerind, Timisoara, Sibiu, Oradea, Rimnicu-Vilcea

The curent state is NOT the goal state

frontier after updation =

['Pitesti', 'Lugoj', 'Fagaras']

The Action is to expand : Lugoj because it has the lowest cost 229

The Parent node of Lugoj is Rimnicu-Vilcea

minimum value =229

{'Rimnicu-Vilcea Pitesti': 317, 'Timisoara Lugoj': 229, 'Sibiu Fagaras': 239}

{'Rimnicu-Vilcea Pitesti': 317, 'Sibiu Fagaras': 239}

Path VALUE : Arad, Zerind, Timisoara, Sibiu, Oradea, Rimnicu-Vilcea, Lugoj

The value of current state = Lugoj

frontier bfore updation =

['Pitesti', 'Fagaras']

explored states :Arad, Zerind, Timisoara, Sibiu, Oradea, Rimnicu-Vilcea, Lugoj

The curent state is NOT the goal state

frontier after updation =

['Pitesti', 'Mehadia', 'Fagaras']

The Action is to expand : Fagaras because it has the lowest cost 239

The Parent node of Fagaras is Lugoj

minimum value =239

{'Rimnicu-Vilcea Pitesti': 317, 'Lugoj Mehadia': 299, 'Sibiu Fagaras': 239}

{'Rimnicu-Vilcea Pitesti': 317, 'Lugoj Mehadia': 299}

Path VALUE : Arad, Zerind, Timisoara, Sibiu, Oradea, Rimnicu-Vilcea, Lugoj, Fagaras

The value of current state = Fagaras

frontier bfore updation =

['Pitesti', 'Mehadia']

explored states :Arad, Zerind, Timisoara, Sibiu, Oradea, Rimnicu-Vilcea, Lugoj, Fagaras

The curent state is NOT the goal state

frontier after updation =

['Pitesti', 'Bucharest', 'Mehadia']

The Action is to expand : Mehadia because it has the lowest cost 299

The Parent node of Mehadia is Fagaras

minimum value =299

{'Rimnicu-Vilcea Pitesti': 317, 'Fagaras Bucharest': 450, 'Lugoj Mehadia': 299}

{'Rimnicu-Vilcea Pitesti': 317, 'Fagaras Bucharest': 450}

Path VALUE : Arad, Zerind, Timisoara, Sibiu, Oradea, Rimnicu-Vilcea, Lugoj, Fagaras, Mehadia

The value of current state = Mehadia

frontier bfore updation =

['Pitesti', 'Bucharest']

explored states :Arad, Zerind, Timisoara, Sibiu, Oradea, Rimnicu-Vilcea, Lugoj, Fagaras, Mehadia

The curent state is NOT the goal state

frontier after updation =

['Pitesti', 'Bucharest', 'Drobeta']

The Action is to expand : Pitesti because it has the lowest cost 317

The Parent node of Pitesti is Mehadia

minimum value =317

{'Rimnicu-Vilcea Pitesti': 317, 'Fagaras Bucharest': 450, 'Mehadia Drobeta': 374}

{'Fagaras Bucharest': 450, 'Mehadia Drobeta': 374}

Path VALUE : Arad, Zerind, Timisoara, Sibiu, Oradea, Rimnicu-Vilcea, Lugoj, Fagaras, Mehadia, Pitesti

The value of current state = Pitesti

frontier bfore updation =

['Bucharest', 'Drobeta']

explored states :Arad, Zerind, Timisoara, Sibiu, Oradea, Rimnicu-Vilcea, Lugoj, Fagaras, Mehadia, Pitesti

The curent state is NOT the goal state

frontier after updation =

['Bucharest', 'Bucharest', 'Drobeta']

The Action is to expand : Drobeta because it has the lowest cost 374

The Parent node of Drobeta is Pitesti

minimum value =374

{'Pitesti Bucharest': 418, 'Fagaras Bucharest': 450, 'Mehadia Drobeta': 374}

{'Pitesti Bucharest': 418, 'Fagaras Bucharest': 450}

Path VALUE : Arad, Zerind, Timisoara, Sibiu, Oradea, Rimnicu-Vilcea, Lugoj, Fagaras, Mehadia, Pitesti, Drobeta

The value of current state = Drobeta

frontier bfore updation =

['Bucharest', 'Bucharest']

explored states :Arad, Zerind, Timisoara, Sibiu, Oradea, Rimnicu-Vilcea, Lugoj, Fagaras, Mehadia, Pitesti, Drobeta

The curent state is NOT the goal state

frontier after updation =

['Bucharest', 'Bucharest', 'Craiova']

The Action is to expand : Bucharest because it has the lowest cost 418

The Parent node of Bucharest is Drobeta

minimum value =418

{'Pitesti Bucharest': 418, 'Fagaras Bucharest': 450, 'Drobeta Craiova': 494}

{'Fagaras Bucharest': 450, 'Drobeta Craiova': 494}

Path VALUE : Arad, Zerind, Timisoara, Sibiu, Oradea, Rimnicu-Vilcea, Lugoj, Fagaras, Mehadia, Pitesti, Drobeta, Bucharest

The value of current state = Bucharest

frontier bfore updation =

['Bucharest', 'Craiova']

explored states :Arad, Zerind, Timisoara, Sibiu, Oradea, Rimnicu-Vilcea, Lugoj, Fagaras, Mehadia, Pitesti, Drobeta, Bucharest

the current state is the goal state

{'Fagaras Bucharest': 450, 'Drobeta Craiova': 494}

{'Arad Zerind': 75, 'Zerind Oradea': 71, 'Sibiu Rimnicu-Vilcea': 80, 'Oradea Sibiu': 151, 'Lugoj Mehadia': 70, 'Rimnicu-Vilcea Pitesti': 97, 'Pitesti Bucharest': 101, 'Mehadia Drobeta': 75, 'Timisoara Lugoj': 111, 'Arad Sibiu': 140, 'Drobeta Craiova': 120, 'Arad Timisoara': 118, 'Fagaras Bucharest': 211, 'Sibiu Fagaras': 99}

['Bucharest', 'Pitesti', 'Rimnicu-Vilcea', 'Sibiu', 'Oradea', 'Zerind', 'Arad']

FINAL OUTPUT -------------------------------------------------------------

The Final Explored states are :Arad, Zerind, Timisoara, Sibiu, Oradea, Rimnicu-Vilcea, Lugoj, Fagaras, Mehadia, Pitesti, Drobeta, Bucharest

The Uniform Cost from Arad to Bucharest is :418

The Path from Arad to Bucharest using Uniform Cost Search :

Arad, Zerind, Oradea, Sibiu, Rimnicu-Vilcea, Pitesti, Bucharest

FINAL OUTPUT ------------------------------------------------------------