**Depth\_Limited\_Search Code :**

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

"""

Created on Sat Sep 17 23:30:49 2016

You have to give Depth number as input

@author: jaysh

"""

# This is a Depth Limited Search

# The problem of the program is stored in a dictionary

# The program is always complete if the goal is present

# Variable path represents the path of the Depth Limited Search

# Variable s represent the current state of the Depth Limited Search

# 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 node 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,goal,problem,depth\_limit):

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

i = 0

parent = []

frontier=intialise\_frontier(start)

search\_finished = False # Represents whether the search has finished or not with in the given depth limit

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

while frontier and i < int(depth\_limit):

print("This is limit number "+str(i+1))

i = i + 1

path.append(choose\_node(frontier))

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

length\_of\_path = len(path)

if(length\_of\_path > 1):

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

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

current\_state= path[-1]

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

explored.append(current\_state)

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

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

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

g = expand\_node(g,problem,current\_state,parent)

update\_frontier(g,explored,frontier)

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

else:

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

search\_finished = True

break

g=[]

if(search\_finished == True):

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

actual\_path,cost = backtracking(start,goal,parent,path)

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

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

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

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

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

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

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

print ("\n The search was finished at depth value: \n "+str(i))

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

else :

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

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

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

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

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

print ("\n Final Path Cost from Arad To "+path[-1]+" = "+str(total\_path\_cost))

print ("\n The Path didnt complete and we were only able to reach following nodes: \n "+final\_path)

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

def backtracking(start,goal,parent,path):

reverse\_path = [path[-1]]

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 function initialises the frontier node to the start node i.e in this case Arad

def intialise\_frontier(start):

frontier = [start]

return frontier

#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

#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

#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,parent):

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])

parent.append(place\_distance\_list)

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

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"

depth\_limit = input("What is the limit of the depth that u want to set \n ") #Takes the input of the till what depth limit the search should go

graph\_search(start,goal,problem,depth\_limit)

'''

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

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

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

if(search\_finished == True):

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

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

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

print ("\n The search was finished at depth value: \n "+str(i))

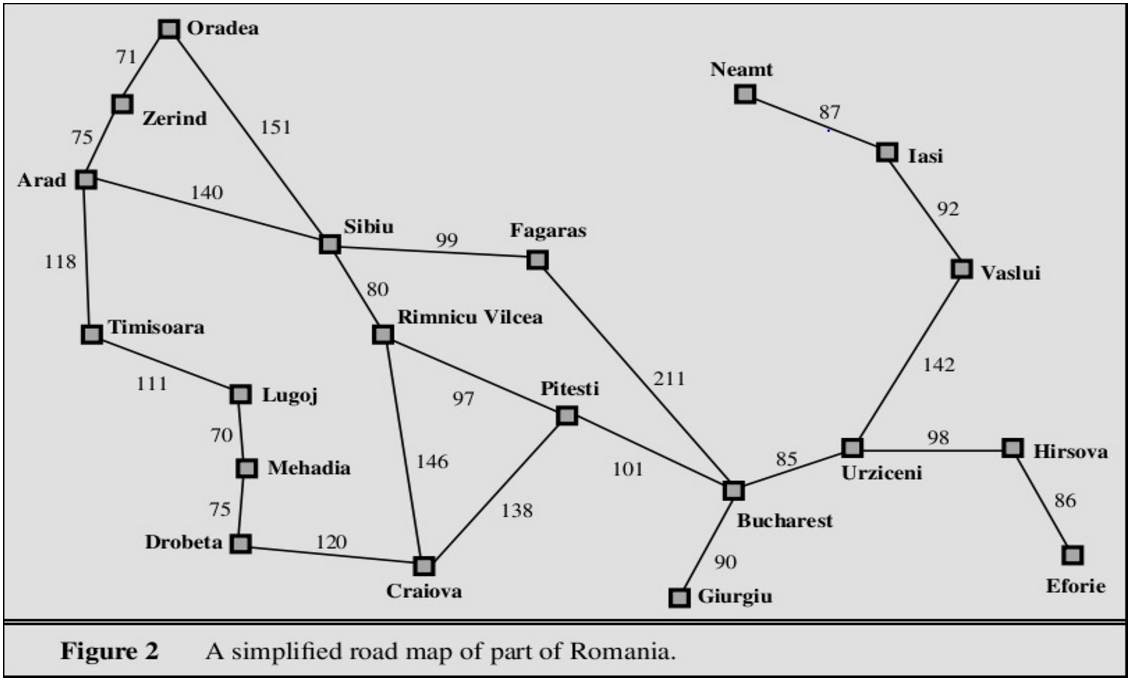
else :

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

print ("\n Final Path Cost from Arad To "+path[-1]+" = "+str(total\_path\_cost))

print ("\n The Path didnt complete and we were only able to reach following nodes: \n "+final\_path)

'''



**OUTPUT :**

What is the limit of the depth that u want to set

5

Initialisation value of frontier = Arad

This is limit number 1

The value of current state = Arad

path value= Arad

frontier bfore updation =

explored node = Arad

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

frontier after updation = Zerind, Timisoara, Sibiu

This is limit number 2

The value of current state = Sibiu

The parent node of current state Sibiu is Arad

path value= Arad, Sibiu

frontier bfore updation = Zerind, Timisoara

explored node = Arad, Sibiu

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

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

This is limit number 3

The value of current state = Rimnicu-Vilcea

The parent node of current state Rimnicu-Vilcea is Sibiu

path value= Arad, Sibiu, Rimnicu-Vilcea

frontier bfore updation = Zerind, Timisoara, Fagaras

explored node = Arad, Sibiu, Rimnicu-Vilcea

The curent state is NOT the goal state

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

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

frontier after updation = Zerind, Timisoara, Fagaras, Pitesti

This is limit number 4

The value of current state = Pitesti

The parent node of current state Pitesti is Rimnicu-Vilcea

path value= Arad, Sibiu, Rimnicu-Vilcea, Pitesti

frontier bfore updation = Zerind, Timisoara, Fagaras

explored node = Arad, Sibiu, Rimnicu-Vilcea, Pitesti

The curent state is NOT the goal state

The distance to be explored is :Pitesti to Bucharest

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

frontier after updation = Zerind, Timisoara, Fagaras, Bucharest

This is limit number 5

The value of current state = Bucharest

The parent node of current state Bucharest is Pitesti

path value= Arad, Sibiu, Rimnicu-Vilcea, Pitesti, Bucharest

frontier bfore updation = Zerind, Timisoara, Fagaras

explored node = Arad, Sibiu, Rimnicu-Vilcea, Pitesti, Bucharest

The curent state is the goal state

[140, 80, 97, 101]

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

The Final Explored states are :

Arad, Sibiu, Rimnicu-Vilcea, Pitesti, Bucharest

Final Path Cost from Arad To Bucharest = 418

The Path from Arad to Bucharest using Uniform Cost Search :

Arad, Sibiu, Rimnicu-Vilcea, Pitesti, Bucharest

The search was finished at depth value:

5

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