Artificial Intelligence-MLA0101

1.Python program to solve Water Jug Problem

Code:

x = 0

y = 0

m = 4

n = 3

print("Initial state = (0,0)")

print("Capacities = (4,3)")

print("Goal state = (2,y)")

while x != 2:

r = int(input("Enter rule"))

if(r == 1):

x = m

elif(r == 2):

y = n

elif(r == 3):

x = 0

elif(r == 4):

y = 0

elif(r == 5):

t = n - y

y = n

x -= t

elif(r == 6):

t = m - x

x = m

y -= t

elif(r == 7):

y += x

x = 0

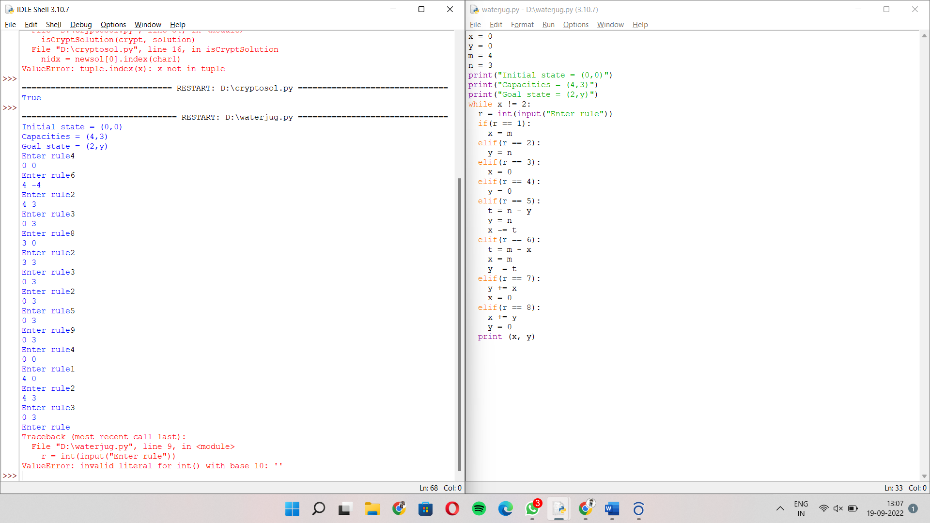
elif(r == 8):

x += y

y = 0

print (x, y)

Screenshorts:



2.Python program for Breadth First Search

Code:

graph = {

'5' : ['3','7'],

'3' : ['2', '4'],

'7' : ['8'],

'2' : [],

'4' : ['8'],

'8' : []

}

visited = [] # List for visited nodes.

queue = [] #Initialize a queue

def bfs(visited, graph, node): #function for BFS

visited.append(node)

queue.append(node)

while queue: # Creating loop to visit each node

m = queue.pop(0)

print (m, end = " ")

for neighbour in graph[m]:

if neighbour not in visited:

visited.append(neighbour)

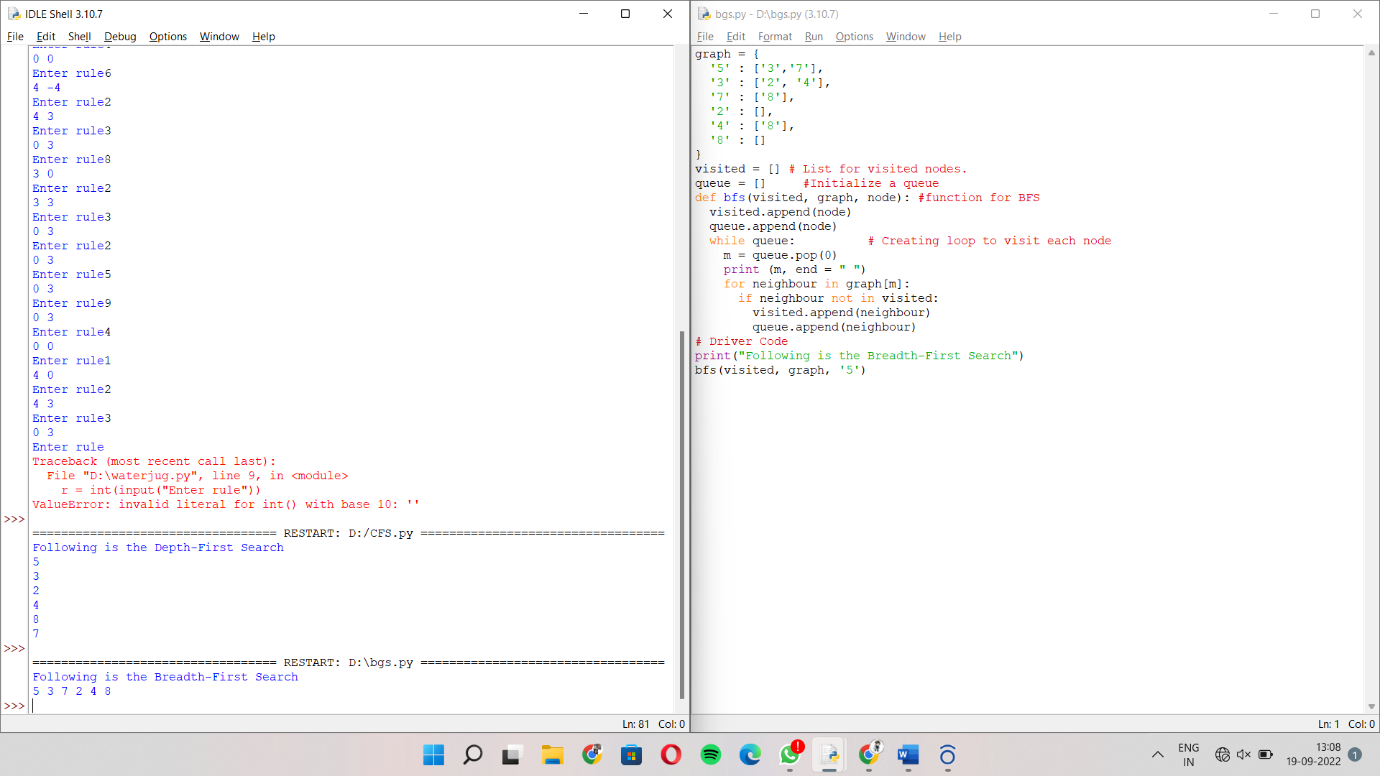
queue.append(neighbour)

# Driver Code

print("Following is the Breadth-First Search")

bfs(visited, graph, '5')

Screenshorts:



3.Python program for Depth First Search

Code:

graph = {

'5' : ['3','7'],

'3' : ['2', '4'],

'7' : ['8'],

'2' : [],

'4' : ['8'],

'8' : []

}

visited = set() # Set to keep track of visited nodes of graph.

def dfs(visited, graph, node): #function for dfs

if node not in visited:

print (node)

visited.add(node)

for neighbour in graph[node]:

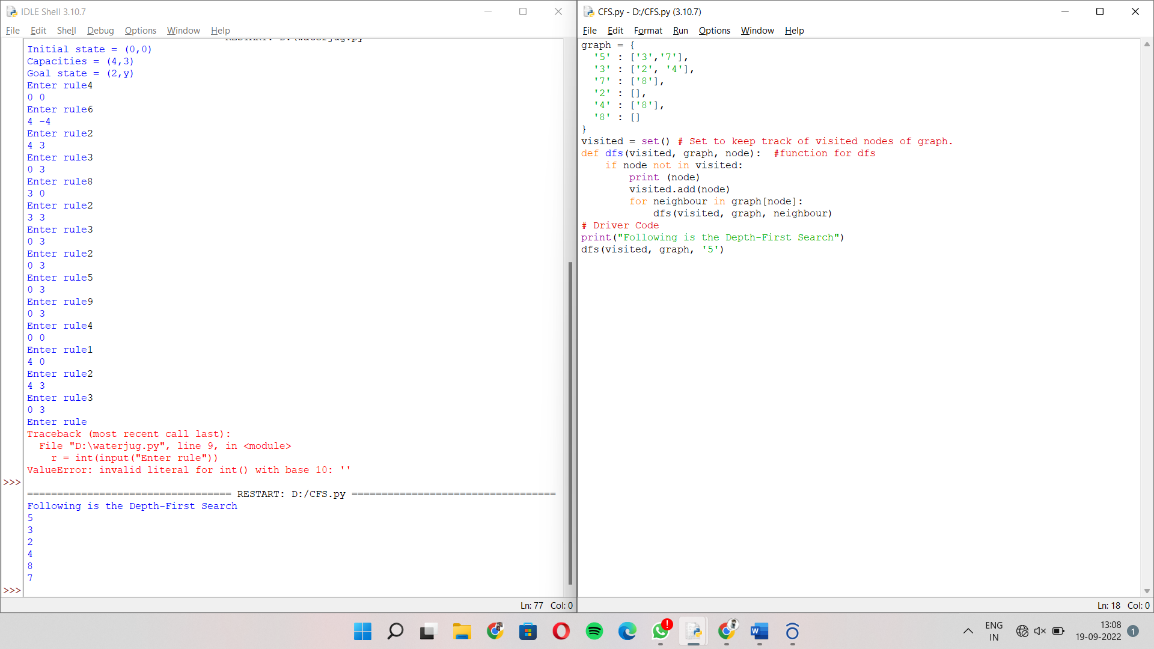
dfs(visited, graph, neighbour)

# Driver Code

print("Following is the Depth-First Search")

dfs(visited, graph, '5')

Screenshorts:



4.Python program to solve Cryptarithmetic Puzzles

Code:

def isCryptSolution(crypt, solution):

newsol = list(zip(\*reversed(solution)))

newstring1 = ''

total = 0

for word in range(len(crypt)-1):

subtotal, sol\_total = 0, 0

newstring = ''

for char in crypt[word]:

idx = newsol[0].index(char)

newstring = newstring + newsol[1][idx]

subtotal = int(newstring)

# if newstring[0] == '0':

# return False

total = total + subtotal

for char1 in crypt[-1]:

nidx = newsol[0].index(char1)

newstring1 = newstring1 + newsol[1][nidx]

sol\_total = int(newstring1)

if total == sol\_total and newstring[0] != '0':

return print('True')

elif total == 0 and newstring[0] == '0' and len(newstring) == 1:

return print('True')

else:

return print('False')

crypt = ["SEND", "MORE", "MONEY"]

solution = [['O', '0'],

['M', '1'],

['Y', '2'],

['E', '5'],

['N', '6'],

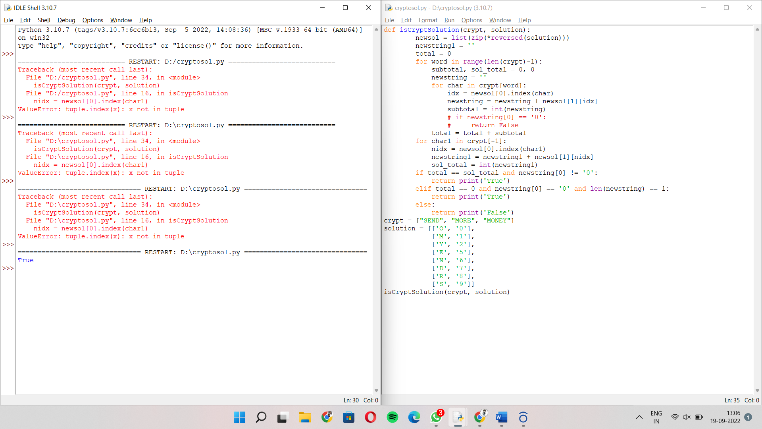
['D', '7'],

['R', '8'],

['S', '9']]

isCryptSolution(crypt, solution)

Screenshorts:



5.Python program to implement Minimax Algorithm

Code:

import math

def minimax (curDepth, nodeIndex,

maxTurn, scores,

targetDepth):

if (curDepth == targetDepth):

return scores[nodeIndex]

if (maxTurn):

return max(minimax(curDepth + 1, nodeIndex \* 2,

False, scores, targetDepth),

minimax(curDepth + 1, nodeIndex \* 2 + 1,

False, scores, targetDepth))

else:

return min(minimax(curDepth + 1, nodeIndex \* 2,

True, scores, targetDepth),

minimax(curDepth + 1, nodeIndex \* 2 + 1,

True, scores, targetDepth))

scores = [9, 3, 5, 6]

treeDepth = math.log(len(scores), 2)

print("The optimal value is : ", end = "")

print(minimax(0, 0, True, scores, treeDepth))

Screenshorts:

