**TRAVEL SALESMAN PROBLEM**

# Python program to find the shortest possible route

# that visits every city exactly once and returns to

# the starting point

from itertools import permutations

def tsp(cost):

# Number of nodes

numNodes = len(cost)

nodes = list(range(1, numNodes))

minCost = float('inf')

# Generate all permutations of the

# remaining nodes

for perm in permutations(nodes):

currCost = 0

currNode = 0

# Calculate the cost of the current permutation

for node in perm:

currCost += cost[currNode][node]

currNode = node

# Add the cost to return to the starting node

currCost += cost[currNode][0]

# Update the minimum cost if the current cost

# is lower

minCost = min(minCost, currCost)

return minCost

if \_\_name\_\_ == "\_\_main\_\_":

cost = [

[0, 10, 15, 20],

[10, 0, 35, 25],

[15, 35, 0, 30],

[20, 25, 30, 0]

]

res = tsp(cost)

print(res)

**8-PUZZLE PROBLEM**

from queue import Queue

import copy

import time

def printNode(node):

print(node[0],node[1],node[2])

print(node[3],node[4],node[5])

print(node[6],node[7],node[8])

global nodeNumber

print('Node:', nodeNumber)

print('Depth:', len(node[9:]))

print('Moves:', node[9:])

print('------')

nodeNumber += 1

def checkFinal(node):

if node[:9]==finalNode:

printNode(node)

return True

if node[:9] not in visitedList:

printNode(node)

queue.put(node)

visitedList.append(node[:9])

return False

if \_\_name\_\_ == '\_\_main\_\_':

startNode = [1,5,4, 3,7,2, 6,8,0]

finalNode = [0,1,2, 3,4,5, 6,7,8]

found = False

nodeNumber = 0

visitedList = []

queue = Queue()

queue.put(startNode)

visitedList.append(startNode)

printNode(startNode)

t0 = time.time()

while (not found and not queue.empty()):

currentNode = queue.get()

blankIndex = currentNode.index(0)

if blankIndex!=0 and blankIndex!=1 and blankIndex!=2:

upNode = copy.deepcopy(currentNode)

upNode[blankIndex] = upNode[blankIndex-3]

upNode[blankIndex-3] = 0

upNode.append('up')

found = checkFinal(upNode)

if blankIndex!=0 and blankIndex!=3 and blankIndex!=6 and found==False:

leftNode = copy.deepcopy(currentNode)

leftNode[blankIndex] = leftNode[blankIndex-1]

leftNode[blankIndex-1] = 0

leftNode.append('left')

found = checkFinal(leftNode)

if blankIndex!=6 and blankIndex!=7 and blankIndex!=8 and found==False:

downNode = copy.deepcopy(currentNode)

downNode[blankIndex] = downNode[blankIndex+3]

downNode[blankIndex+3] = 0

downNode.append('down')

found = checkFinal(downNode)

if blankIndex!=2 and blankIndex!=5 and blankIndex!=8 and found==False:

rightNode = copy.deepcopy(currentNode)

rightNode[blankIndex] = rightNode[blankIndex+1]

rightNode[blankIndex+1] = 0

rightNode.append('right')

found = checkFinal(rightNode)

t1 = time.time()

print('Time:', t1-t0)

print('------')

**MISSIONARIES & CANNIBALS PROBLEM**

#Python program to illustrate Missionaries & cannibals Problem

#This code is contributed by Sunit Mal

print("\n")

print("\tGame Start\nNow the task is to move all of them to right side of the river")

print("rules:\n1. The boat can carry at most two people\n2. If cannibals num greater than missionaries then the cannibals would eat the missionaries\n3. The boat cannot cross the river by itself with no people on board")

lM = 3 #lM = Left side Missionaries number

lC = 3 #lC = Laft side Cannibals number

rM=0 #rM = Right side Missionaries number

rC=0 #rC = Right side cannibals number

userM = 0 #userM = User input for number of missionaries for right to left side travel

userC = 0 #userC = User input for number of cannibals for right to left travel

k = 0

print("\nM M M C C C | --- | \n")

try:

while(True):

while(True):

print("Left side -> right side river travel")

#uM = user input for number of missionaries for left to right travel

#uC = user input for number of cannibals for left to right travel

uM = int(input("Enter number of Missionaries travel => "))

uC = int(input("Enter number of Cannibals travel => "))

if((uM==0)and(uC==0)):

print("Empty travel not possible")

print("Re-enter : ")

elif(((uM+uC) <= 2)and((lM-uM)>=0)and((lC-uC)>=0)):

break

else:

print("Wrong input re-enter : ")

lM = (lM-uM)

lC = (lC-uC)

rM += uM

rC += uC

print("\n")

for i in range(0,lM):

print("M ",end="")

for i in range(0,lC):

print("C ",end="")

print("| --> | ",end="")

for i in range(0,rM):

print("M ",end="")

for i in range(0,rC):

print("C ",end="")

print("\n")

k +=1

if(((lC==3)and (lM == 1))or((lC==3)and(lM==2))or((lC==2)and(lM==1))or((rC==3)and (rM == 1))or((rC==3)and(rM==2))or((rC==2)and(rM==1))):

print("Cannibals eat missionaries:\nYou lost the game")

break

if((rM+rC) == 6):

print("You won the game : \n\tCongrats")

print("Total attempt")

print(k)

break

while(True):

print("Right side -> Left side river travel")

userM = int(input("Enter number of Missionaries travel => "))

userC = int(input("Enter number of Cannibals travel => "))

if((userM==0)and(userC==0)):

print("Empty travel not possible")

print("Re-enter : ")

elif(((userM+userC) <= 2)and((rM-userM)>=0)and((rC-userC)>=0)):

break

else:

print("Wrong input re-enter : ")

lM += userM

lC += userC

rM -= userM

rC -= userC

k +=1

print("\n")

for i in range(0,lM):

print("M ",end="")

for i in range(0,lC):

print("C ",end="")

print("| <-- | ",end="")

for i in range(0,rM):

print("M ",end="")

for i in range(0,rC):

print("C ",end="")

print("\n")

if(((lC==3)and (lM == 1))or((lC==3)and(lM==2))or((lC==2)and(lM==1))or((rC==3)and (rM == 1))or((rC==3)and(rM==2))or((rC==2)and(rM==1))):

print("Cannibals eat missionaries:\nYou lost the game")

break

except EOFError as e:

print("\nInvalid input please retry !!")

**TIC-TAC-TOE PROGRAM**

# random number in Python

# importing all necessary libraries

import numpy as np

import random

from time import sleep

# Creates an empty board

def create\_board():

return(np.array([[0, 0, 0],

[0, 0, 0],

[0, 0, 0]]))

# Check for empty places on board

def possibilities(board):

l = []

for i in range(len(board)):

for j in range(len(board)):

if board[i][j] == 0:

l.append((i, j))

return(l)

# Select a random place for the player

def random\_place(board, player):

selection = possibilities(board)

current\_loc = random.choice(selection)

board[current\_loc] = player

return(board)

# Checks whether the player has three

# of their marks in a horizontal row

def row\_win(board, player):

for x in range(len(board)):

win = True

for y in range(len(board)):

if board[x, y] != player:

win = False

continue

if win == True:

return(win)

return(win)

# Checks whether the player has three

# of their marks in a vertical row

def col\_win(board, player):

for x in range(len(board)):

win = True

for y in range(len(board)):

if board[y][x] != player:

win = False

continue

if win == True:

return(win)

return(win)

# Checks whether the player has three

# of their marks in a diagonal row

def diag\_win(board, player):

win = True

y = 0

for x in range(len(board)):

if board[x, x] != player:

win = False

if win:

return win

win = True

if win:

for x in range(len(board)):

y = len(board) - 1 - x

if board[x, y] != player:

win = False

return win

# Evaluates whether there is

# a winner or a tie

def evaluate(board):

winner = 0

for player in [1, 2]:

if (row\_win(board, player) or

col\_win(board, player) or

diag\_win(board, player)):

winner = player

if np.all(board != 0) and winner == 0:

winner = -1

return winner

# Main function to start the game

def play\_game():

board, winner, counter = create\_board(), 0, 1

print(board)

sleep(2)

while winner == 0:

for player in [1, 2]:

board = random\_place(board, player)

print("Board after " + str(counter) + " move")

print(board)

sleep(2)

counter += 1

winner = evaluate(board)

if winner != 0:

break

return(winner)

# Driver Code

print("Winner is: " + str(play\_game()))