Artificial intelligence

Solve the 8 puzzles using input and output

import copy

from heapq import heappush, heappop

n = 3

rows = [ 1, 0, -1, 0 ]

cols = [ 0, -1, 0, 1 ]

class priorityQueue:

def \_\_init\_\_(self):

self.heap = []

def push(self, key):

heappush(self.heap, key)

def pop(self):

return heappop(self.heap)

def empty(self):

if not self.heap:

return True

else:

return False

class nodes:

def \_\_init\_\_(self, parent, mats, empty\_tile\_posi,

costs, levels):

self.parent = parent

self.mats = mats

self.empty\_tile\_posi = empty\_tile\_posi

self.costs = costs

self.levels = levels

def \_\_lt\_\_(self, nxt):

return self.costs < nxt.costs

def calculateCosts(mats, final) -> int:

count = 0

for i in range(n):

for j in range(n):

if ((mats[i][j]) and

(mats[i][j] != final[i][j])):

count += 1

return count

def newNodes(mats, empty\_tile\_posi, new\_empty\_tile\_posi,

levels, parent, final) -> nodes:

new\_mats = copy.deepcopy(mats)

x1 = empty\_tile\_posi[0]

y1 = empty\_tile\_posi[1]

x2 = new\_empty\_tile\_posi[0]

y2 = new\_empty\_tile\_posi[1]

new\_mats[x1][y1], new\_mats[x2][y2] = new\_mats[x2][y2], new\_mats[x1][y1]

costs = calculateCosts(new\_mats, final)

new\_nodes = nodes(parent, new\_mats, new\_empty\_tile\_posi,

costs, levels)

return new\_nodes

def printMatsrix(mats):

for i in range(n):

for j in range(n):

print("%d " % (mats[i][j]), end = " ")

print()

def isSafe(x, y):

return x >= 0 and x < n and y >= 0 and y < n

def printPath(root):

if root == None:

return

printPath(root.parent)

printMatsrix(root.mats)

print()

def solve(initial, empty\_tile\_posi, final):

pq = priorityQueue()

costs = calculateCosts(initial, final)

root = nodes(None, initial,

empty\_tile\_posi, costs, 0)

pq.push(root)

while not pq.empty():

minimum = pq.pop()

if minimum.costs == 0:

printPath(minimum)

return

for i in range(n):

new\_tile\_posi = [

minimum.empty\_tile\_posi[0] + rows[i],

minimum.empty\_tile\_posi[1] + cols[i], ]

if isSafe(new\_tile\_posi[0], new\_tile\_posi[1]):

child = newNodes(minimum.mats,

minimum.empty\_tile\_posi,

new\_tile\_posi,

minimum.levels + 1,

minimum, final,)

pq.push(child)

initial = [ [ 1, 2, 3 ],

[ 5, 6, 0 ],

[ 7, 8, 4 ] ]

final = [ [ 1, 2, 3 ],

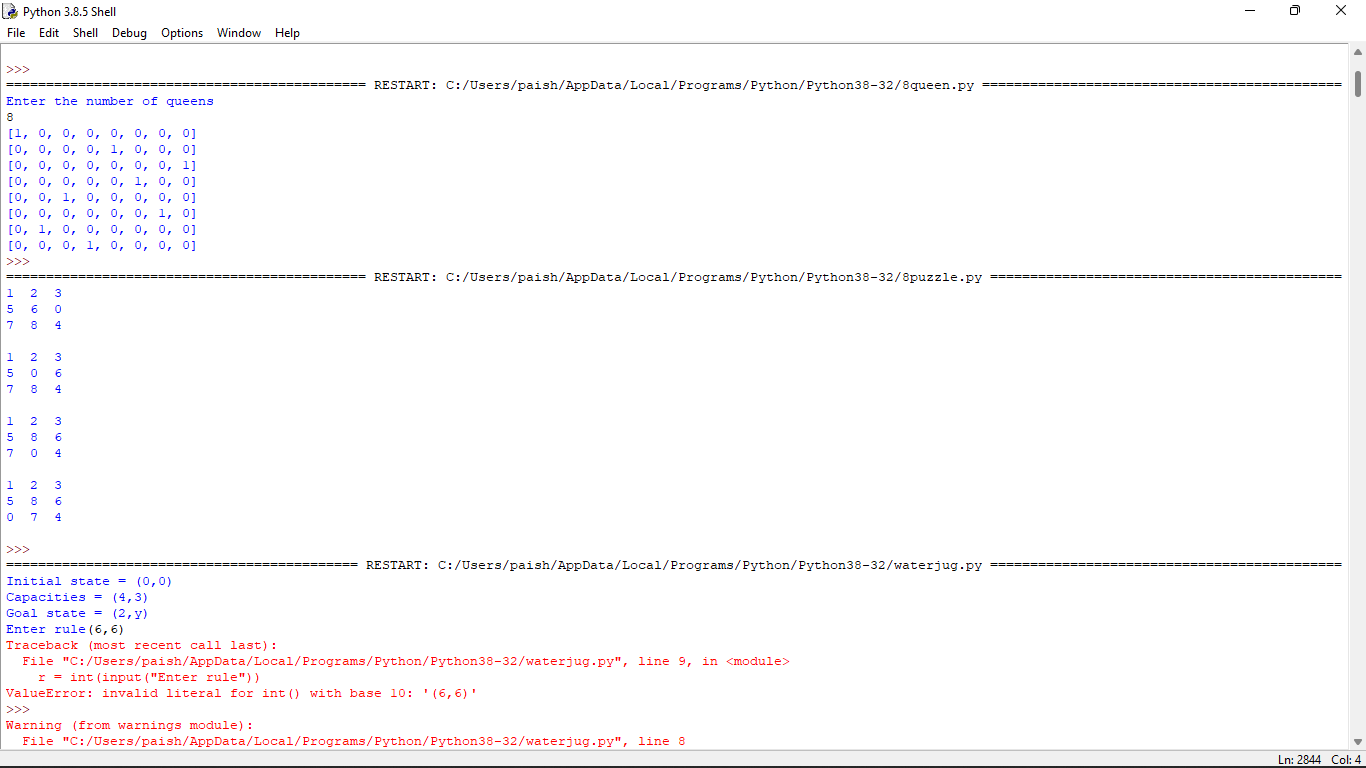
[ 5, 8, 6 ],

[ 0, 7, 4 ] ]

empty\_tile\_posi = [ 1, 2 ]

solve(initial, empty\_tile\_posi, final)

output:



Solve the 8 queens using given input and output

print ("Enter the number of queens")

N = int(input())

board = [[0]\*N for \_ in range(N)]

def attack(i, j):

for k in range(0,N):

if board[i][k]==1 or board[k][j]==1:

return True

for k in range(0,N):

for l in range(0,N):

if (k+l==i+j) or (k-l==i-j):

if board[k][l]==1:

return True

return False

def N\_queens(n):

if n==0:

return True

for i in range(0,N):

for j in range(0,N):

if (not(attack(i,j))) and (board[i][j]!=1):

board[i][j] = 1

if N\_queens(n-1)==True:

return True

board[i][j] = 0

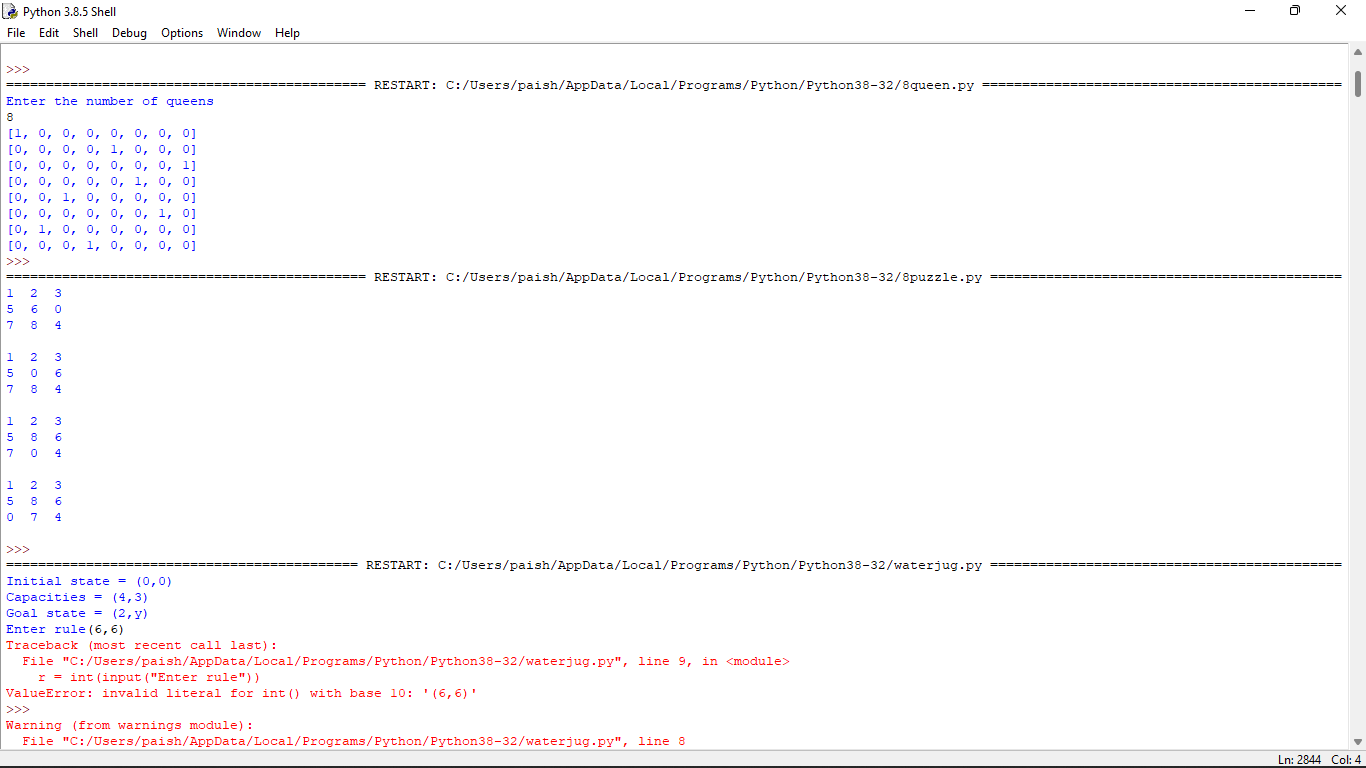
return False

N\_queens(N)

for i in board:

print (i)

output



Solve the water jug problem using input and output

from collections import defaultdict

jug1, jug2, aim = 12, 8, 5

visited = defaultdict(lambda: False)

def waterJugSolver(amt1, amt2):

if (amt1 == aim and amt2 == 0) or (amt2 == aim and amt1 == 0):

print(amt1, amt2)

return True

if visited[(amt1, amt2)] == False:

print(amt1, amt2)

visited[(amt1, amt2)] = True

return (waterJugSolver(0, amt2) or

waterJugSolver(amt1, 0) or

waterJugSolver(jug1, amt2) or

waterJugSolver(amt1, jug2) or

waterJugSolver(amt1 + min(amt2, (jug1-amt1)),

amt2 - min(amt2, (jug1-amt1))) or

waterJugSolver(amt1 - min(amt1, (jug2-amt2)),

amt2 + min(amt1, (jug2-amt2))))

else:

return False

print("Steps: ")

waterJugSolver(0, 0)

output:



Write a python program for missionaries cannibal problem

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 then missionaries then the cannibals would eat the missionaries\n3. The boat cannot cross the river by itself with no people on board")

lM = 3

lC = 3

rM=0

rC=0

userM = 0

userC = 0

k = 0

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

try:

while(True):

while(True):

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

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 then missionaries then the cannibals would eat the missionaries\n3. The boat cannot cross the river by itself with no people on board")

lM = 3

lC = 3

rM=0

rC=0

userM = 0

userC = 0

k = 0

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

try:

while(True):

while(True):

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

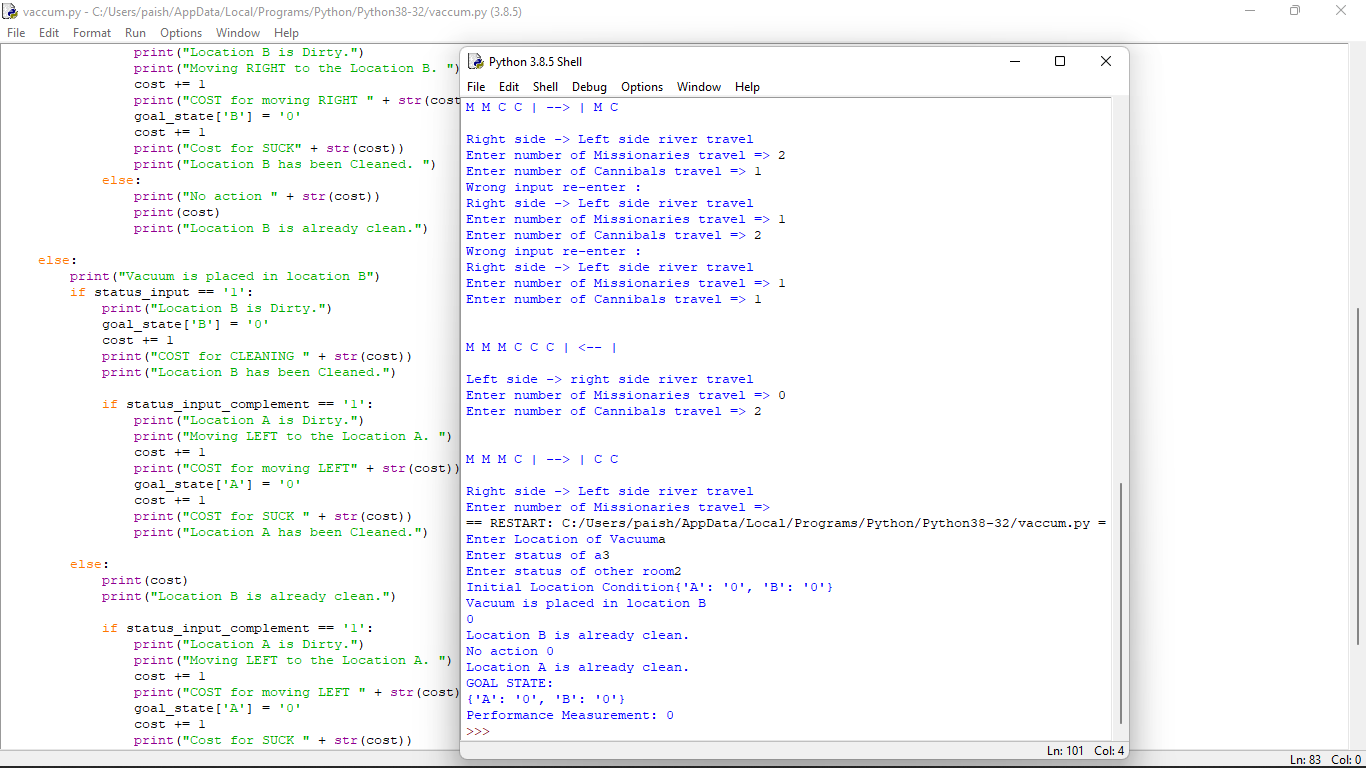
if(((lC==3)and (lM == 1))or((lC==3)and(lM==2))or((lC==2)and(lM==1))or((rC==3)and (rM == 1))or((rC==3)and(rM==2))or((rC==2)and(rM==1))):

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

break

except EOFError as e:

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



Write a python program for vaccum cleaner

def vacuum\_world():

goal\_state = {'A': '0', 'B': '0'}

cost = 0

location\_input = input("Enter Location of Vacuum")

status\_input = input("Enter status of " + location\_input)

status\_input\_complement = input("Enter status of other room")

print("Initial Location Condition" + str(goal\_state))

if location\_input == 'A':

print("Vacuum is placed in Location A")

if status\_input == '1':

print("Location A is Dirty.")

goal\_state['A'] = '0'

cost += 1

print("Cost for CLEANING A " + str(cost))

print("Location A has been Cleaned.")

if status\_input\_complement == '1':

print("Location B is Dirty.")

print("Moving right to the Location B. ")

cost += 1

print("COST for moving RIGHT" + str(cost))

goal\_state['B'] = '0'

cost += 1

print("COST for SUCK " + str(cost))

print("Location B has been Cleaned. ")

else:

print("No action" + str(cost))

print("Location B is already clean.")

if status\_input == '0':

print("Location A is already clean ")

if status\_input\_complement == '1':

print("Location B is Dirty.")

print("Moving RIGHT to the Location B. ")

cost += 1

print("COST for moving RIGHT " + str(cost))

goal\_state['B'] = '0'

cost += 1

print("Cost for SUCK" + str(cost))

print("Location B has been Cleaned. ")

else:

print("No action " + str(cost))

print(cost)

print("Location B is already clean.")

else:

print("Vacuum is placed in location B")

if status\_input == '1':

print("Location B is Dirty.")

goal\_state['B'] = '0'

cost += 1

print("COST for CLEANING " + str(cost))

print("Location B has been Cleaned.")

if status\_input\_complement == '1':

print("Location A is Dirty.")

print("Moving LEFT to the Location A. ")

cost += 1

print("COST for moving LEFT" + str(cost))

goal\_state['A'] = '0'

cost += 1

print("COST for SUCK " + str(cost))

print("Location A has been Cleaned.")

else:

print(cost)

print("Location B is already clean.")

if status\_input\_complement == '1':

print("Location A is Dirty.")

print("Moving LEFT to the Location A. ")

cost += 1

print("COST for moving LEFT " + str(cost))

goal\_state['A'] = '0'

cost += 1

print("Cost for SUCK " + str(cost))

print("Location A has been Cleaned. ")

else:

print("No action " + str(cost))

print("Location A is already clean.")

print("GOAL STATE: ")

print(goal\_state)

print("Performance Measurement: " + str(cost))

vacuum\_world()

