Dfs

def dfs(graph, start, visited=None):

if visited is None:

visited = set()

visited.add(start)

print(start)

for next in graph[start] - visited:

dfs(graph, next, visited)

return visited

graph = {'0': set(['1', '2']),

'1': set(['0', '3', '4']),

'2': set(['0']),

'3': set(['1']),

'4': set(['2', '3'])}

dfs(graph, '0')

**BFS**

**import collections**

**def bfs(graph, root):**

**visited, queue = set(), collections.deque([root])**

**visited.add(root)**

**while queue:**

**vertex = queue.popleft()**

**print(str(vertex) + " ", end="")**

**for neighbour in graph[vertex]:**

**if neighbour not in visited:**

**visited.add(neighbour)**

**queue.append(neighbour)**

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

**graph = {0: [1, 2], 1: [2], 2: [3], 3: [1, 2]}**

**print("Following is Breadth First Traversal: ")**

**bfs(graph, 0)**

**asa**

**#8 puzzle A\***

**class Node:**

**def \_\_init\_\_(self,data,level,fval):**

**self.data = data**

**self.level = level**

**self.fval = fval**

**def generate\_child(self):**

**x,y = self.find(self.data,'\_')**

**val\_list = [[x,y-1],[x,y+1],[x-1,y],[x+1,y]]**

**children = []**

**for i in val\_list:**

**child = self.shuffle(self.data,x,y,i[0],i[1])**

**if child is not None:**

**child\_node = Node(child,self.level+1,0)**

**children.append(child\_node)**

**return children**

**def shuffle(self,puz,x1,y1,x2,y2):**

**if x2 >= 0 and x2 < len(self.data) and y2 >= 0 and y2 < len(self.data):**

**temp\_puz = []**

**temp\_puz = self.copy(puz)**

**temp = temp\_puz[x2][y2]**

**temp\_puz[x2][y2] = temp\_puz[x1][y1]**

**temp\_puz[x1][y1] = temp**

**return temp\_puz**

**else:**

**return None**

**def copy(self,root):**

**temp = []**

**for i in root:**

**t = []**

**for j in i:**

**t.append(j)**

**temp.append(t)**

**return temp**

**def find(self,puz,x):**

**for i in range(0,len(self.data)):**

**for j in range(0,len(self.data)):**

**if puz[i][j] == x:**

**return i,j**

**class Puzzle:**

**def \_\_init\_\_(self,size):**

**self.n = size**

**self.open = []**

**self.closed = []**

**def accept(self):**

**puz = []**

**for i in range(0,self.n):**

**temp = input().split(" ")**

**puz.append(temp)**

**return puz**

**def f(self,start,goal):**

**return self.h(start.data,goal)+start.level**

**def h(self,start,goal):**

**temp = 0**

**for i in range(0,self.n):**

**for j in range(0,self.n):**

**if start[i][j] != goal[i][j] and start[i][j] != '\_':**

**temp += 1**

**return temp**

**def process(self):**

**print("Enter the start state matrix \n")**

**start = self.accept()**

**print("Enter the goal state matrix \n")**

**goal = self.accept()**

**start = Node(start,0,0)**

**start.fval = self.f(start,goal)**

**self.open.append(start)**

**print("\n\n")**

**while True:**

**cur = self.open[0]**

**print("")**

**print(" | ")**

**print(" | ")**

**print(" \\\'/ \n")**

**for i in cur.data:**

**for j in i:**

**print(j,end=" ")**

**print("")**

**if(self.h(cur.data,goal) == 0):**

**break**

**for i in cur.generate\_child():**

**i.fval = self.f(i,goal)**

**self.open.append(i)**

**self.closed.append(cur)**

**del self.open[0]**

**self.open.sort(key = lambda x:x.fval,reverse=False)**

**puz = Puzzle(3)**

**puz.process()**

**dj**

class Graph():

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

self.V = vertices

self.graph = [[0 for column in range(vertices)]

for row in range(vertices)]

def printSolution(self, dist):

print("Vertex \t Distance from Source")

for node in range(self.V):

print(node, "\t\t", dist[node])

def minDistance(self, dist, sptSet):

min = 1e7

for v in range(self.V):

if dist[v] < min and sptSet[v] == False:

min = dist[v]

min\_index = v

return min\_index

def dijkstra(self, src):

dist = [1e7] \* self.V

dist[src] = 0

sptSet = [False] \* self.V

for cout in range(self.V):

u = self.minDistance(dist, sptSet)

sptSet[u] = True

for v in range(self.V):

if (self.graph[u][v] > 0 and

sptSet[v] == False and

dist[v] > dist[u] + self.graph[u][v]):

dist[v] = dist[u] + self.graph[u][v]

self.printSolution(dist)

g = Graph(9)

g.graph = [[0, 4, 0, 0, 0, 0, 0, 8, 0],

[4, 0, 8, 0, 0, 0, 0, 11, 0],

[0, 8, 0, 7, 0, 4, 0, 0, 2],

[0, 0, 7, 0, 9, 14, 0, 0, 0],

[0, 0, 0, 9, 0, 10, 0, 0, 0],

[0, 0, 4, 14, 10, 0, 2, 0, 0],

[0, 0, 0, 0, 0, 2, 0, 1, 6],

[8, 11, 0, 0, 0, 0, 1, 0, 7],

[0, 0, 2, 0, 0, 0, 6, 7, 0]

]

g.dijkstra(0)

**ss**

**import sys**

**A = [64, 25, 12, 22, 11]**

**# Traverse through all array elements**

**for i in range(len(A)):**

**# Find the minimum element in remaining**

**# unsorted array**

**min\_idx = i**

**for j in range(i+1, len(A)):**

**if A[min\_idx] > A[j]:**

**min\_idx = j**

**# Swap the found minimum element with**

**# the first element**

**A[i], A[min\_idx] = A[min\_idx], A[i]**

**# Driver code to test above**

**print ("Sorted array")**

**for i in range(len(A)):**

**print("%d" %A[i],end=" , ")**

**nq**

CODE

#nqueens branch and bound

N = 8

""" A utility function to print solution """

def printSolution(board):

for i in range(N):

for j in range(N):

print(board[i][j], end = " ")

print()

""" A Optimized function to check if

a queen can be placed on board[row][col] """

def isSafe(row, col, slashCode, backslashCode,

rowLookup, slashCodeLookup,

backslashCodeLookup):

if (slashCodeLookup[slashCode[row][col]] or

backslashCodeLookup[backslashCode[row][col]] or

rowLookup[row]):

return False

return True

""" A recursive utility function

to solve N Queen problem """

def solveNQueensUtil(board, col, slashCode, backslashCode,

rowLookup, slashCodeLookup,

backslashCodeLookup):

""" base case: If all queens are

placed then return True """

if(col >= N):

return True

for i in range(N):

if(isSafe(i, col, slashCode, backslashCode,

rowLookup, slashCodeLookup,

backslashCodeLookup)):

""" Place this queen in board[i][col] """

board[i][col] = 1

rowLookup[i] = True

slashCodeLookup[slashCode[i][col]] = True

backslashCodeLookup[backslashCode[i][col]] = True

""" recur to place rest of the queens """

if(solveNQueensUtil(board, col + 1,

slashCode, backslashCode,

rowLookup, slashCodeLookup,

backslashCodeLookup)):

return True

""" If placing queen in board[i][col]

doesn't lead to a solution,then backtrack """

""" Remove queen from board[i][col] """

board[i][col] = 0

rowLookup[i] = False

slashCodeLookup[slashCode[i][col]] = False

backslashCodeLookup[backslashCode[i][col]] = False

""" If queen can not be place in any row in

this column col then return False """

return False

""" This function solves the N Queen problem using

Branch or Bound. It mainly uses solveNQueensUtil()to

solve the problem. It returns False if queens

cannot be placed,otherwise return True or

prints placement of queens in the form of 1s.

Please note that there may be more than one

solutions,this function prints one of the

feasible solutions."""

def solveNQueens():

board = [[0 for i in range(N)]

for j in range(N)]

# helper matrices

slashCode = [[0 for i in range(N)]

for j in range(N)]

backslashCode = [[0 for i in range(N)]

for j in range(N)]

# arrays to tell us which rows are occupied

rowLookup = [False] \* N

# keep two arrays to tell us

# which diagonals are occupied

x = 2 \* N - 1

slashCodeLookup = [False] \* x

backslashCodeLookup = [False] \* x

# initialize helper matrices

for rr in range(N):

for cc in range(N):

slashCode[rr][cc] = rr + cc

backslashCode[rr][cc] = rr - cc + 7

# change 7 accordingly to the number of rows in the chess board.

# example if N = 4 then insteead of 7 write 3

if(solveNQueensUtil(board, 0, slashCode, backslashCode,

rowLookup, slashCodeLookup,

backslashCodeLookup) == False):

print("Solution does not exist")

return False

# solution found

printSolution(board)

return True

# Driver Code

solveNQueens()

CODE

#nqueens backtracking

def is\_attack(i, j, board, N):

# checking for column j

for k in range(1, i):

if(board[k][j] == 1):

return True

# checking upper right diagonal

k = i-1

l = j+1

while (k>=1 and l<=N):

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

return True

k=k+1

l=l+1

# checking upper left diagonal

k = i-1

l = j-1

while (k>=1 and l>=1):

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

return True

k=k-1

l=l-1

return False

def n\_queen(row, n, N, board):

if (n==0):

return True

for j in range(1, N+1):

if(not(is\_attack(row, j, board, N))):

board[row][j] = 1

if (n\_queen(row+1, n-1, N, board)):

return True

board[row][j] = 0 #backtracking

return False

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

board = [[0,0,0,0,0],[0,0,0,0,0],[0,0,0,0,0],[0,0,0,0,0],[0,0,0,0,0]]

n\_queen(1, 4, 4, board)

#printing the matix

for i in range(1, 5):

print(board[i][1:])

**chat**

**import datetime**

**def calculate\_age(birth\_date):**

**today = datetime.date.today()**

**age = today.year - birth\_date.year**

**if today < datetime.date(today.year, birth\_date.month, birth\_date.day):**

**age -= 1**

**return age**

**# Function to process user input and generate a response**

**def process\_input(input, step):**

**switch = {**

**0: lambda: f"Hello {input}! Please enter your date of birth (YYYY-MM-DD):",**

**1: lambda: f"You are {calculate\_age(datetime.datetime.strptime(input, '%Y-%m-%d').date())} years old. How can I assist you today?",**

**2: lambda: "Please provide your requirements:",**

**3: lambda: f"Your requirements are noted as: {input}. Please provide your contact details:",**

**4: lambda: "Thank you for providing your contact details. We will get back to you soon!"**

**}**

**return switch.get(step, lambda: "I'm sorry, I didn't understand. Can you please rephrase or provide more information?")()**

**# Main chatbot interaction loop**

**def chatbot():**

**step = 0**

**user\_input = ""**

**print("Welcome to the Chatbot!")**

**while True:**

**response = process\_input(user\_input, step)**

**print("Chatbot:", response)**

**if step == 4:**

**break**

**user\_input = input("User: ")**

**step += 1**

**# Run the chatbot application**

**chatbot()**

**ex**

print("helloWorld")

QUESTIONS = [

  'Do you have cough?',

  'Do you have a sore throat?',

  'Do you have a fever?',

  'Are you noticing any unexplained excessive sweating?',

  'Do you have an itchy throat?',

  'Do you have a runny nose?',

  'Do you have a stuffy nose?',

  'Do you have a headache?',

  'Do you feel tired without actually exhausting yourself?'

]

THRESHOLD = {

  'Mild': 30,

  'Severe': 50,

  'Extreme': 75

}

def expertSystem(questions, threshold):

  score = 0

  for question in questions:

    print(question+" (Y/N) ")

    ans = input("> ")

    if ans.lower() == 'y':

      print('On a scale of 1-10 how bad is it ?')

      ip = input('> ')

      while((not ip.isnumeric()) or int(ip) < 1 or int(ip) > 10):

        print('Enter a valid input !')

        ip = input('> ')

      score += int(ip)

    print()

    print()

    if score >= threshold['Extreme']:

      print("You are showing symptoms of having EXTREME COVID-19")

      print("Please call +91 8112233445 immediately to immediate assistance")

      print("Based on your symptoms, You will need Immediate Hospitalization")

    elif score >= threshold['Severe']:

      print("Based on your answers You are showing Symptoms of SEVERE COVID-19")

      print("You are advised to contact a COVID-19 Specialist ASAP")

      print("You are prescribed with Favipriavir, Dolo 650 / Crocin 500, Paracetamol, Brufane")

      print("Also coduct a COVID-19 Lab Test ASAP at your own convenience as this might be a false Positive")

      print()

      print()

      print("Lab Testing: https://www.metropolisindia.com/parameter/pune/covid-19-rt-pcr-test")

    elif score >= threshold['Mild']:

      print("Based on your answers You are showing Symptoms of VERY␣MILD COVID-19")

      print("Please Isolate yourself Immediately on a precautionary basis")

      print("As this has a possibility of being a false positive, please consider testing yourself")

      print("At-home testing using Self-Testing kits is recommended, but you can get Lab Tests as well")

      print()

      print()

      print("Self-testing: https://www.flipkart.com/mylab-coviself-covid-19-rapid-antigen-test-kit/p/itm4d34ea09cad97")

expertSystem(QUESTIONS, THRESHOLD)