**GROUP B**

**Assignment No. B1**

* 1. **Problem Statement**

8-Queens Matrix is Stored using JSON/XML having first Queen placed, use back-tracking to place remaining Queens to generate final 8-queen's Matrix using Python.

* 1. **Objective**
* Understand the 8-queen matrix in JSON/XML.
* To generate 8-queen matrix using backtracking.
  1. **Theory**

8-queen matrix using backtracking:

The 8 Queen is the problem of placing 8 chess queens on an 8×8 chessboard so that no two queens attack each other. For example, following is a solution for 4 Queen problem.

![](data:None;base64,)

The expected output is a binary matrix which has 1s for the blocks where queens are placed.

This problem is a typical example of backtracking. Backtracking is depth first traversal of path in graph where nodes are states of the solution and edge are between two states of solution only if one state can be reached from another state.

The idea is to place queens one by one in different columns, starting from the leftmost column. When we place a queen in a column, we check for clashes with already placed queens. In the current column, if we find a row for which there is no clash, we mark this row and column as part of the solution. If we do not find such a row due to clashes then we backtrack and return false.

**Mathematical Model :**

The 8 queens problem can be defined as solution space P :

where P = { S , I , O , F , E }

S = { Start state consisting of one queen placed.}

I = { Input to the system : I1 }

where I1 = { }

O = { Output state : O1 }

where O1 = { Output matrix displayed}

F = { Functions used : f1 , f2 , f3 }

where f1 = { To Display on terminal ,dispboard}

f2 = { To display board on GUI,dispboard2 }

f3 = {To display label on GUI,displabel }

f4={to check if a cell is safe,issafe2}

f5={implement backtracking,solve}

E = { End state of the system }

where E = { All queens placed without attacking each other}

**Algorithm:**

1) Start in the leftmost column.

2) If all queens are placed return true

3) Try all rows in the current column. Do following for every tried row.

a) If the queen can be placed safely in this row then mark this [row, column] as part of the solution and recursively check if placing queen here leads to a solution.

b) If placing queen in [row, column] leads to a solution then return true.

c) If placing queen doesn't lead to a solution then umark this [row, column] (Backtrack) and go to step (a) to try other rows.

4) If all rows have been tried and nothing worked, return false to trigger backtracking.

**Execution command:**

python 8queen.py

* 1. **Input for assignment:**
* Json file containing a matrix such that queen placed at first position.
  1. **Expected Output:**
* 8 queen problem solved with all queens placed.

**6. Test cases:**

**Positive cases:**

1.Input: First queen placed at row 5 and column 4 in json file

Expected Output: All further queens placed accordingly

Actual output:

[0, 0, 0, 0, 1, 0, 0, 0],

[0, 1, 0, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 0, 0, 0, 1],

[1, 0, 0, 0, 0, 0, 0, 0],

[0, 0, 0, 1, 0, 0, 0, 0],

[0, 0, 0, 0, 0, 0, 1, 0],

[0, 0, 1, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 0, 1, 0, 0]

2. Input: First queen placed at row 1 and column 1 in json file

Expected Output: All further queens placed accordingly

Actual Output:

[1, 0, 0, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 0, 0, 1, 0],

[0, 0, 0, 0, 1, 0, 0, 0],

[0, 0, 0, 0, 0, 0, 0, 1],

[0, 1, 0, 0, 0, 0, 0, 0],

[0, 0, 0, 1, 0, 0, 0, 0],

[0, 0, 0, 0, 0, 1, 0, 0],

[0, 0, 1, 0, 0, 0, 0, 0]

3. Input: First queen placed at row 1 and column 7 in json file

Expected Output: All further queens placed accordingly

Actual Output:

[0, 0, 0, 0, 0, 0, 1, 0],

[0, 0, 1, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 0, 0, 0, 1],

[0, 1, 0, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 0, 0, 0, 1],

[1, 0, 0, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 0, 1, 0, 0],

[0, 0, 0, 1, 0, 0, 0, 0]

**Negative cases:**

1.Input:First queen not placed

Expected output: Problem solved with any possible solution

Actual output: Solution displayed with first queen at first cell by default.

**7.Conclusion :**

Hence, we have successfully implemented the program for 8 queen problem using backtracking in Python

**8.Outcomes achieved**

CO-I: To solve problems using mathematical modeling.

**9.FAQs**

1. How is backtracking implemented?

2. Which search strategy implements backtracking? Explain its efficiency.

3. What is the Time Complexity for this code?

4. What is JSON?

5. How is JSON different than XML?

**Code:**

**8q.json:**

{"matrix": [

[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, 0, 0, 0, 0, 0, 0, 0],

[0, 0, 0, 1, 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, 0, 0, 0]]}

**8queen.py:**

import json

import Tkinter

import time

inf=open("8q.json")

board=json.loads(inf.read())

board=board["matrix"]

class myclass:

count = 1

#To Display on terminal

def dispboard(mat,n):

for i in range (0,n):

for j in range(0,n):

print(mat[i][j]),

print

#To display board on GUI

def dispboard2(mat,n):

root = Tkinter.Tk( )

for i in range(0,n):

for j in range(0,n):

Tkinter.Label(root, text='%s'%(mat[i][j]),borderwidth=1 ).grid(row=i,column=j)

root.update( )

time.sleep(5)

root.destroy()

#To display label on GUI

def displabel(strng):

root1 = Tkinter.Tk( )

var = Tkinter.StringVar()

label = Tkinter.Label( root1, textvariable=var, relief=Tkinter.RAISED )

var.set(strng)

label.pack()

root1.update()

time.sleep(3)

root1.destroy()

def issafe2(mat,row,col):

for i in range(8):

for j in range(8):

if(mat[i][j]==1): #if a queen exists here, then check if it attacks our queen

if(row==i):

return False

if(col==j):

return False

if(abs(row-i)==abs(col-j)):

return False

return True

def solve(mat,n,col):

if m.count>=n:

return True

for i in range (0,n):

if issafe2(mat,i,col):

mat[i][col]=1

m.count +=1

if solve(mat,n,(col+1)%8) == True:

return True

mat[i][col]=0

m.count -=1

return False

def main(n):

r1=0

c1=0

for i in range (0,8):

for j in range (0,8):

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

c1=j

r1=i

mat = [[0 for i in range (n)] for j in range(n)]

mat[r1][c1] = 1

#dispboard(mat,n)

displabel("Displaying board with first queen placed..")

dispboard2(mat,n)

if solve(mat,n,c1+1) == False:

print("Doesn't exist")

else:

print

displabel("Solved!")

dispboard2(mat,n)

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

m=myclass()

main(8)

Output:

![](data:None;base64,)

![](data:None;base64,)

![](data:None;base64,)

