ASSIGNMENT NO: B6

1. TITLE:

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. Use suitable Software modelling , Design and testing methods. Justify the selection over other methods.

2. PREREQUISITES:

- 64-bit Fedora or equivalent OS with 64-bit Intel-i5/i7
- Python 2.7

3. OBJECTIVE:

- To learn the concept of Divide and Conquer Strategy.
- To study the design and implementation of 8-Queens Matrix
- To learn 8-queens matrix.

4. INPUT: Unsorted Numbers in array

5. OUTPUT: Sorted array is generated.

6. MATHEMATICAL MODEL:

Let S be the solution perspective of the class Weather Report such that

S={s, e, i, o, f, DD, NDD, success, failure}

s=Start of program

e = the end of program

i=Unsorted numbers.

o=Result of sorted data

f= Found/Not Found

DD=Deterministic data

NDD=NULL

Success-Number is found.

Failure-Number is not found

7. THEORY :

The eight queens puzzle is the problem of placing eight chess queens on an 8 8 chessboard so that no two queens threaten each other. Thus, a solution requires that no two queens

share the same row, column, or diagonal. The eight queens puzzle is an example of the more general n-queens problem of placing n queens on an n n chessboard, where solutions exist for all natural numbers n with the exception of n=2 and n=3.

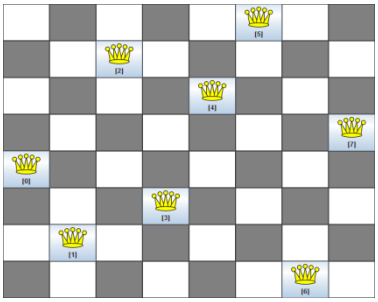

Figure 1: Queen Placing

The problem can be quite computationally expensive as there are 4,426,165,368 possible arrangements of eight queens on an 8X8 board, but only 92 solutions. The eight queens puzzle has 92 distinct solutions. If solutions that differ only by symmetry operations (rotations and reflections) of the board are counted as one, the puzzle has 12 fundamental solution.

- Find an arrangement of 8 queens on a single chess board such that no two queens are attacking one another.

- In chess, queens can move all the way down any row, column or diagonal (so long as no pieces are in the way).

Due to the first two restrictions, it's clear that each row and column of the board will haveexactly one queen.

Conditions For 8 Queen Problem :

1) No two Queens should be in Same rows.

2) No two Queens should be in Same column.

3) No two Queens should be in Same diagonal.

The backtracking strategy is as follows :

1) Place a queen on the first available square in row 1.

2) Move onto the next row, placing a queen on the first available square there (that doesn'tconflict with the previously placed queens).

3) Continue in this fashion until either :

a) you have solved the problem, or

b) You get stuck.

When you get stuck, remove the queens that got you there, until you get to a row where there is another valid square to try.

- When we carry out backtracking, an easy way to visualize what is going on is a tree that shows all the different possibilities that have been tried.

Approach :

- Create a solution matrix of the same structure as chess board.
- Whenever place a queen in the chess board, mark that particular cell in solution matrix.
- At the end print the solution matrix, the marked cells will show the positions of the queens in the chess board.

Conditions For 8 Queen Problem :

1) No two Queens should be in Same rows.
2) No two Queens should be in Same column.
3) No two Queens should be in Same diagonal.

Algorithm :

1) Place the queens column wise, start from the left most column
2) If all queens are placed.
    a) return true and print the solution matrix
3) Else
    a) Try all the rows in the current column.
    b) Check if queen can be placed here safely if yes mark the current cell in solution matrix as 1 and try to solve the rest of the problem recursively.
    c) If placing the queen in above step leads to the solution return true.
    d) If placing the queen in above step does not lead to the solution , BACKTRACK, mark the current cell in solution matrix as 0 and return false.
4. If all the rows are tried and nothing worked, return false and print NO SOLUTION

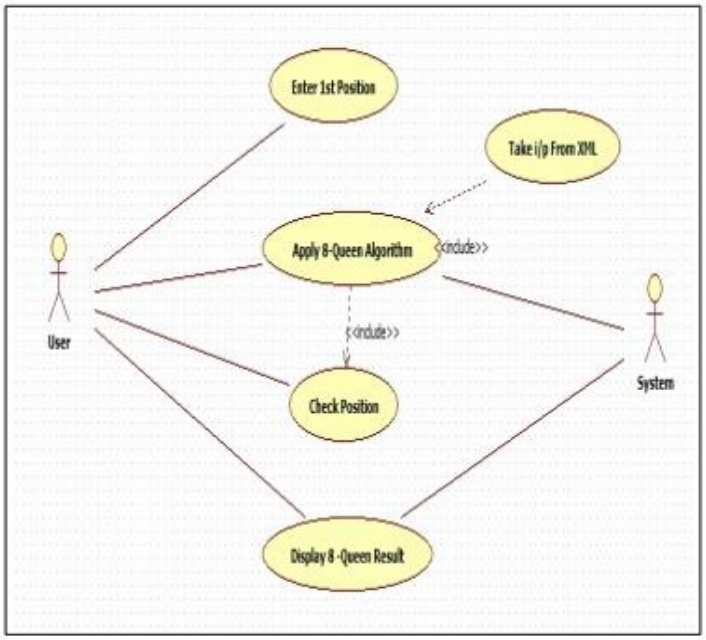

Figure.2: Use Case Diagram

Advantages of backtracking:

The major advantage of the backtracking algorithm is the ability to find and count all the possible solutions rather than just one while offering decent speed. In fact this is the reason it is so widely used. Also one can easily produce a parallel version of the backtracking algorithm increasing speed several times just by starting multiple threads with different starting positions of the first queens.

CONCLUSION:

Thus we have successfully studied Queens Matrix is Stored using JSON/XML having firstQueen placed, use back-tracking to place remaining Queens to generate final 8-queen's Matrix

```
import json

inf=open("8q.json")

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

board=board["matrix"]

for i in board:

    print(i)


def issafe(row,col):

    for i in range(8):

        for j in range(8):

            if(board[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 place(col):

    if(col>=8):     #if all 8 queens are placed, then finish

        print("\t\tCompleted...")

        return True

    for i in range(8):  #checking for all rows in that column

        if(board[i][col]==1):   #if a queen is already placed here,

            return place(col+1) #then simply place for next column

        if(issafe(i,col)):  #is it safe?

            board[i][col]=1 #queen is placed here

            if(place(col+1)==True): #recursive call to place next queen

                return True

            board[i][col]=0     #if not placed, then backtrack, i.e it sets to zero and the
loop iterates to check for next position

    return False
```

```
if(place(0)==True):
    print("solution found")
else:
    print("Solution not possible")
for i in board:
    print(i)
```

```
C:\Users\neera\Documents\be-2\B1>python b1.py
[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, 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, 1, 0, 0, 0]
                Completed...
solution found
[1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 1, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0]
[0, 0, 0, 1, 0, 0, 0, 0]
[0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 0]

C:\Users\neera\Documents\be-2\B1>python b1.py
[1, 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, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0]
                Completed...
solution found
[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]


C:\Users\neera\Documents\be-2\B1>python b1.py
[1, 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, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 1, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0]
Solution not possible
[1, 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, 0, 0, 0, 0, 0, 0]
```

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

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