**MODEL PRACTICAL EXAMINATION**

**NAME:ANIKET SHUKLA**

**REG.NO.:RA1811003010754**

**CSE E2**

**QUESTION:**

Write a Program that attains a solution (which is the objective of the program) with the fixed set of constraints enforced on it. Your Program should clearly demonstrate the constraints and how those constraints are satisfied in the process of solving the problem. You are free to choose the objective of the program, but you must encompass all the constraints of the chosen objective.

**OBJECTIVE** : To implement Constraint Satisfaction Problem on N queens problem.

**ALGORITHM :**

**N Queens problem:** Place N queens on a chessboard of dimension N x N i.e N rows x N columns, such that no two queens can attack each other.  
Consider below chessboards of size 4, the board on the left side is valid in which no two queens can attack each other; whereas the board on the right is invalid.

**N Queens placement idea :**  
The idea behind placing NQueens on a chessboard is as below.  
a) Try placing the queen on (and starting from) the 0’th column of the chessboard.  
b) For every selected column, try placing the queen from the 0’th row of the chessboard such that it does not attack any of the already placed queens.

For every possible valid placement of the queen on the chessboard the next empty column is selected; hence while checking if a queen on a (row, column) is correctly placed we check only the left side of the column. i.e

* We check if there is already a queen on the left in the row.
* We check if there is already a queen on the upper diagonal.
* We check if there is already a queen on the lower diagonal.

If for a selected column, a queen cannot be placed in the current row, try placing it in the next row. If the queen cannot be placed in any of the rows of the selected column, backtrack and try placing the previously placed queen in the next row before trying to place the current one again.

**INPUT CODE**:

**print ("Enter the number of queens")**

**N = int(input())**

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

**def is\_attack(i, j):**

**for k in range(0,N):**

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

**return True**

**#checking diagonals**

**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\_queen(n):**

**#if n is 0, solution found**

**if n==0:**

**return True**

**for i in range(0,N):**

**for j in range(0,N):**

**'''checking if we can place a queen here or not**

**queen will not be placed if the place is being attacked**

**or already occupied'''**

**if (not(is\_attack(i,j))) and (board[i][j]!=1):**

**board[i][j] = 1**

**#recursion**

**#wether we can put the next queen with this arrangment or not**

**if N\_queen(n-1)==True:**

**return True**

**board[i][j] = 0**

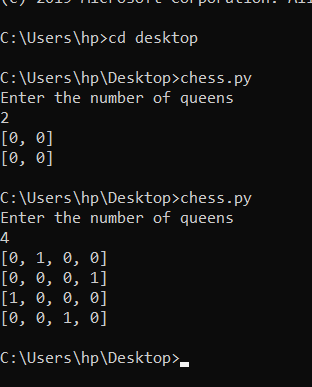
**return False**

**N\_queen(N)**

**for i in board:**

**print (i)**

**OUTPUT SCREENSHOTS :**

****

**RESULT:**

The n queens is implemented successfully.