LACS-Elite-Part011

Solved Challenges 0/1

Back To Challenges List

N Queens - Fill Remaining

ID:12945 **Solved By 0 Users**

In a **N*N** square chessboard, Q queens are placed in Q continuous rows so that they do not attack each other. The program must accept the positions of the Q queens (marked by 1) and print the positions of the remaining N-Q queens in the remaining rows to be placed, so that they do not attack each other.

Note: In Chess, queen can move any direction diagonally. The queen can also move left or right along the row it is present. The queen can also move up or down along the column it is present. The movement can happen till the end of the board is reached or another piece (like Rook, Knight, Bishop, Pawn etc is encountered). But in this program only the N queens are placed and no other pieces will be present on the board.

Boundary Condition(s):
4 <= N <= 10
2 <= Q <= N-1

Input Format:
The first line contains N.
The next N lines contain N values (0 or 1) each separated by a space.

Output Format:
The first line contains the shift in the position followed by the direction.

Example Input/Output 1:
Input:

```
4
0 1 0 0
0 0 0 1
0 0 0 0
0 0 0 0
```

Output:

```
0 1 0 0
0 0 0 1
1 0 0 0
0 0 1 0
```

Explanation:
In this combination the queens do not attack each other.

Example Input/Output 2:

Input:

```
8
0 0 1 0 0 0 0 0
0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 1
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
```

Output:

```
0 0 1 0 0 0 0 0
0 0 0 0 0 1 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 0 0 1 0 0 0
0 1 0 0 0 0 0 0
```

Max Execution Time Limit: 4000 millisecs

Ambiance

Python3 (3.x)

Reset

```
def canPlace(N,row,board,queenrow,queencol,nwdiag,swdiag):
    if(row>=N):
        return True
    if(queenrow[row] == True):
        return canPlace(N, row+1,board,queenrow,queencol,nwdiag,s
    for col in range(N):
        if(nwdiag[row+col] == False and swdiag[col-row+N-1] == Fa
            queencol[col] == False):
            board[row][col] = 1
            nwdiag[row+col] = True
            swdiag[col-row+N-1] = True
            queencol[col] = True
            queenrow[row] = True
            if(canPlace(N,row+1,board,queenrow,queencol,nwdiag,sw
                return True
            else:
                board[row][col] = 0
                nwdiag[row+col] = False
                swdiag[col-row+N-1] = False
                queencol[col] = False
                queenrow[row] = False

    return False

N = int(input())

board = []
for row in range(N):
    row = list(map(int, input().split()))
    board.append(row)

queenrow = [False]*N
queencol = [False]*N
nwdiag = [False] *(2*N -1)
swdiag = [False]*(2*N -1)

for row in range(N):
    for col in range(N):
        if(board[row][col] == 1):
            queenrow[row] = True
            queencol[col] = True
            nwdiag[row + col] = True
            swdiag[col-row+N-1] = True


if(canPlace(N,0,board,queenrow,queencol,nwdiag,swdiag)):
    for row in range(N):
        for col in range(N):
            print(board[row][col],end =" ")
        print()
else:
    print("Not Possible")


```

Code did not pass the execution

TestCase ID: 86153

Input:

```
4
0 1 0 0
0 0 0 1
0 0 0 0
0 0 0 0
```

Expected Output:

```
0 1 0 0
0 0 0 1
1 0 0 0
0 0 1 0
```

Your Program Output:

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

Save Run

□ Run with a custom test case (Input/Output)