Priyanka Srinivas | RA1911026010014

Artificial Intelligence Lab

Exercise 1: The N-Queens Problem

AIM: To solve the N-Queens problem

INTRODUCTION:

N-Queens problem is to place n - queens in such a manner on an n x n chessboard that no queens attack each other by being in the same row, column or diagonal. It can be seen that for n =1, the problem has a trivial solution, and no solution exists for n =2 and n =3. Therefore, the N-Queens problem starts with 4 queens, then scaled up for N-Queens.

ALGORITHM:

1. Parse the cells from the leftmost column.
2. Check if all queens are placed. If yes, then return true.
3. Check all cells in the current column. For each cell:
   A. Check if the queen can be placed safely in the cell. If yes, add the [row1, column1] to the solution set and recursively check for other queens in the following cells.
   B. If recursively checking after adding [row1, column1] leads to a solution where all queens are placed, then return true.
   C. Else, remove [row, column] from the solution set and backtrack to the previous marked [row0, column0] and recursively try for other solutions.
4. Once all rows have been tried without any solution, return false to indicate that no solution is found. This will trigger backtracking.

The following code in python is for 4-Queens.

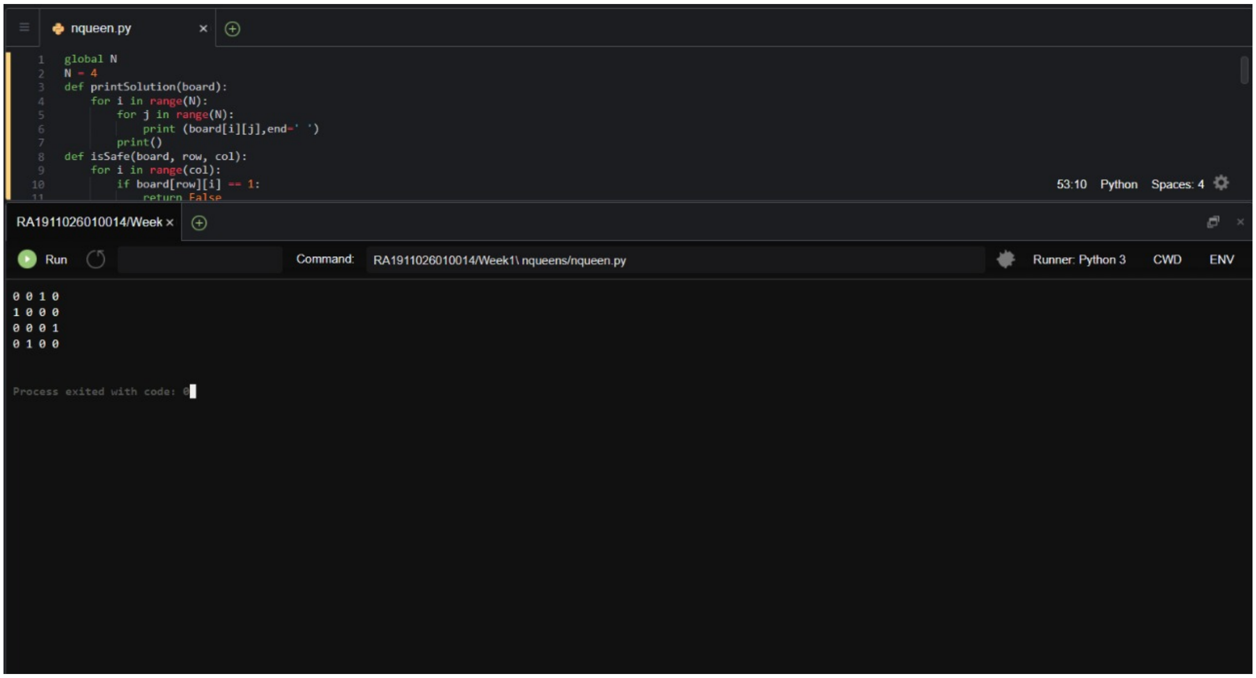
PROGRAM:

```
global N
N = 4

def printSolution(board):
    for i in range(N):
        for j in range(N):
            print (board[i][j], end = " ")
```

```
        print()

# A utility function to check if a queen can
# be placed on board[row][col]. Note that this
# function is called when "col" queens are
# already placed in columns from 0 to col -1.
# So we need to check only the left side for
# attacking queens

def isSafe(board, row, col):
    # Check this row on left side
    for i in range(col):
        if board[row][i] == 1:
            return False

    # Check upper diagonal on left side
    for i, j in zip(range(row, -1, -1),
                    range(col, -1, -1)):
        if board[i][j] == 1:
            return False

    # Check lower diagonal on left side
    for i, j in zip(range(row, N, 1),
                    range(col, -1, -1)):
        if board[i][j] == 1:
            return False

    return True

def solveNQUtil(board, col):
    # base case: If all queens are placed then return true
    if col >= N:
        return True

    # Consider this column and try placing
    # this queen in all rows one by one
    for i in range(N):

        if isSafe(board, i, col):
```

```
            # Place this queen in board[i][col]
            board[i][col] = 1

            # recur to place rest of the queens
            if solveNQUtil(board, col + 1) == True:
                return True

            # If placing queen in board[i][col
            # doesn't lead to a solution, then
            # queen from board[i][col]
            board[i][col] = 0

    # if the queen can not be placed in any row in
    # this column col then return false
    return False

# This function solves the N Queen problem using
# Backtracking. It mainly uses solveNQUtil() to
# solve the problem. It returns false if queens
# cannot be placed, otherwise, return true and
# placement of queens in the form of 1s.
# note that there may be more than one
# solution, this function prints one of the feasible solutions.

def solveNQ():
    board = [ [0, 0, 0, 0],
              [0, 0, 0, 0],
              [0, 0, 0, 0],
              [0, 0, 0, 0] ]

    if solveNQUtil(board, 0) == False:
        print ("Solution does not exist")
        return False

    printSolution(board)
    return True

# Driver Code
solveNQ()
```

OUTPUT:



RESULT:

The N-Queens problem was implemented successfully.