**Implementation of backtracking algorithms to solve n-Queens problem**

**Aim:**

To solve the N-queens problem using backtracking.

**Problem Description:**

The N-queens problem is an NP-hard problem where N queens should be placed on a chessboard such that no queens can attack each other.

**Algorithm:**

* + Initialize an empty chessboard of size N x N.
  + Start with the leftmost column and place a queen in the first row of that column.
  + Move to the next column and place a queen in the first row of that column.
  + Repeat step 3 until either all N queens have been placed or it is impossible to place a queen in the current column without violating the rules of the problem.
  + If all N queens have been placed, print the solution.
  + If it is not possible to place a queen in the current column without violating the rules of the problem, backtrack to the previous column.
  + Remove the queen from the previous column and move it down one row.
  + Repeat steps 4-7 until all possible configurations have been tried.

**Code:**

def printSolution(board,N):

    for i in range(N):

        for j in range(N):

            if board[i][j] == 1:

                print("Q",end=" ")

            else:

                print(".",end=" ")

        print()

def isSafe(board, row, col,N):

    for i in range(col):

        if board[row][i] == 1:

            return False

    for i, j in zip(range(row, -1, -1),

                    range(col, -1, -1)):

        if board[i][j] == 1:

            return False

    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,N):

    if col >= N:

        return True

    for i in range(N):

        if isSafe(board, i, col,N):

            board[i][col] = 1

            if solveNQUtil(board, col + 1,N) == True:

                return True

            board[i][col] = 0

    return False

def solveNQ():

    n = int(input("Enter the value of n:"))

    board = [[0 for i in range(n)] for i in range(n)]

    if solveNQUtil(board, 0,n) == False:

        print("Solution does not exist")

        return False

    printSolution(board,n)

    return True

solveNQ()

**Output:**

**A picture containing text, screenshot, font, black

Description automatically generated**

**TIME COMPLEXITY:**

Generating all possible configurations: The number of possible configurations of placing N queens on an NxN chessboard is N!. Therefore, the time complexity of generating all possible configurations is O(N!)

**ALGORITHM ANALYSIS:**

Backtracking and recursion: The backtracking algorithm used to solve the NQueens problem involves recursive calls. In the worst-case scenario, we may have to explore all possible configurations.

**Result:**

Thus, n queens’ problem has been solved using backtracking.