**Assignment1**

**COSC2007: Assignment 1**

**Student Name: Jeel Tikiwala**

**Student ID: 239659420**

**Exercise No 1:**

The 8-queens problem is a problem of placing n queens on an n× n chessboard, where solutions. The question of the problem is how to place eight queens on a chess board in a way that they are not able to attack each other. There should not be any 2 queens in the same horizontal, vertical or diagonal line. The goal is not just to find one possible solution, but all of them. An advanced way of looking at the problem is by trying to solve the problem for any number of queens. The number of queens can be defined in the source code and the amount of horizontal and vertical lines of the board are defied according to number of queens. The one solution is given in following figure.

The idea is to place queens one by one in different columns, starting from the leftmost column. When we place a queen in a column, there should not be two queens in same rows or columns or diagonals. If we find a row for which there is no repeating queue, we mark this row and column as part of the solution. If we do not find such a row due to attack, then we backtrack.

**Algorithm/ Pseudocode:**

Start

if there is a queen at the left of the current col. //check horizontal

      return false

if there is a queen at the left upper diagonal. //check upper left diagonal

      return false

if there is a queen at the left lower diagonal //check lower left diagonal

      return false;

   return true //otherwise the place is valid

End

**Code:**

**import** java.util.\*;

/\*\*

\* Author: Jeel Tikiwala

\* Student Id: 239659420

\* Assignment 1: N Queens

\* Any and All work in this file is my.

\*/

**public** **class** Nqueen\_N8 {

// helper method to place queen on board

**public** **void** helper(**char**[][] board, List<List<String>> allBoards, **int** col) {

// Base case: if all columns are filled, save the current board state

**if**(col == board.length) {

saveBoard(board, allBoards);

**return**;

}

// iterate through each row in the current column

**for**(**int** row = 0; row < board.length; row++) {

// Check if it's safe to place a queen at the current position

**if**(isSafe(row, col, board)) {

// Place a queen at the current position

board[row][col] = 'Q';

// place queens in the next column through recursion

helper(board, allBoards, col + 1);

// remove the queen and try the next row by backtracking

board[row][col] = '.';

}

}

}

// Method to save the current board state into the list of all boards

**public** **void** saveBoard(**char**[][] board, List<List<String>> allBoards) {

List<String> board1 = **new** ArrayList<>();

// Convert the 2D char array into a list of strings

**for**(**int** i = 0; i < board.length; i++) {

StringBuilder row = **new** StringBuilder();

**for**(**int** j = 0; j < board[0].length; j++) {

row.append(board[i][j]);

}

board1.add(row.toString());

}

// add current board to allboard

allBoards.add(board1);

}

// Method to check if it's safe to place a queen at the given position

**private** **boolean** isSafe(**int** row, **int** col, **char**[][] board) {

// check the horizontal line

**for**(**int** j = 0; j < col; j++) {

**if**(board[row][j] == 'Q') {

**return** **false**;

}

}

// check the upper left

**for**(**int** r = row, c = col; r >= 0 && c >= 0; r--, c--) {

**if**(board[r][c] == 'Q') {

**return** **false**;

}

}

// check lower left

**for**(**int** r = row, c = col; r < board.length && c >= 0; r++, c--) {

**if**(board[r][c] == 'Q') {

**return** **false**;

}

}

// If all the conditions are false, it's safe to place a queen

**return** **true**;

}

// Method to find all solutions to the N-Queens problem

**public** List<List<String>> solveNQueens(**int** n){

List<List<String>> allBoards = **new** ArrayList<>();

// Initialize the chessboard with empty cells

**char**[][] board = **new** **char**[n][n];

**for** (**char**[] row : board) {

Arrays.*fill*(row, '.');

}

// call helper function to start backtracking

helper(board, allBoards, 0);

**return** allBoards;

}

//print all solutions

**public** **void** printSolutions(List<List<String>> solutions) {

**for** (List<String> board : solutions) {

**for** (String row : board) {

System.***out***.println(row);

}

System.***out***.println();

}

}

**public** **static** **void** main(String[] args) {

Nqueen\_N8 solution = **new** Nqueen\_N8();

**int** n = 8;

List<List<String>> solutions = solution.solveNQueens(n);

solution.printSolutions(solutions);

}

}

**Output**:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated