**Experiment 3.2**

**Competitive Coding Lab 9(Backtracking)**

**Student Name: Sahul Kr. Parida UID: 20BCS4919**

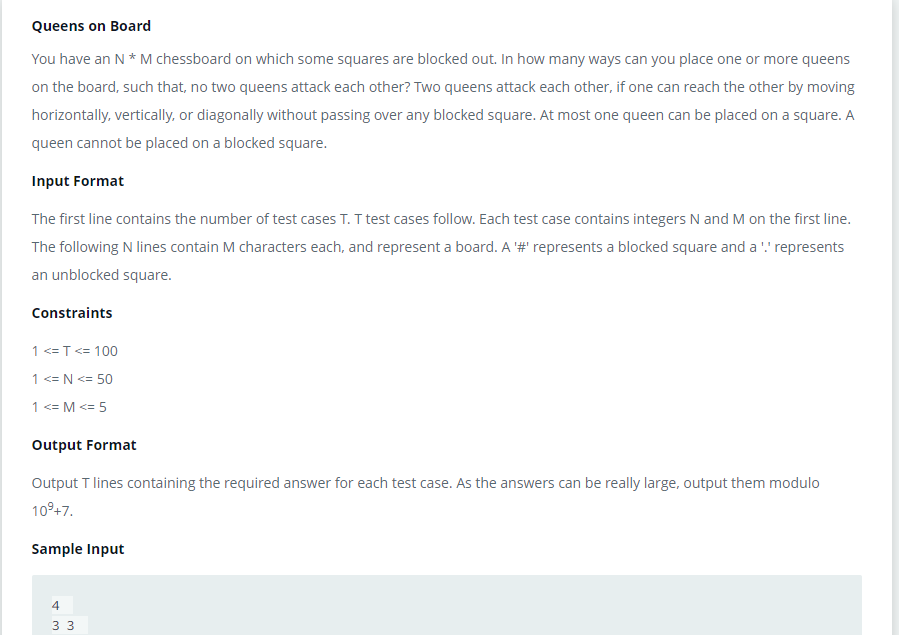
**Branch: CSE Section/Group: WM-904/B**

**Semester: 5th Date of Performance: 08/11/22**

**Subject Name: Competitive Coding(CC) Subject Code: 20CSP-314**

**PROBLEM STATEMENT 9.1: -**

<https://www.hackerrank.com/challenges/queens-on-board/problem>



**SOLUTION:**

static String createRowConfig(int mask)

{

    String rowConfig = Integer.toString(mask, 2);

    while(rowConfig.length() < m) {

      rowConfig = "0" + rowConfig;

    }

    return rowConfig;

  }

  static boolean isValidRowConfig(String rowConf, int row) {

    int count = 0;

    for(int i = 0; i < m; i++) {

      if (board[row].charAt(i) == '#') {

        if (hasQueen(rowConf, i)) {

          return false;

        }

        count = 0;

        continue;

      }

      if (hasQueen(rowConf, i)) {

        if (++count > 1) {

          return false;

        }

      }

    }

    return true;

  }

  static boolean compatible(String rowConf, String attVect) {

    for(int i = 0; i < m; i++) {

      if (!hasQueen(rowConf, i))

      {

        continue;

      }

      if (attackedFromUpperLeft(i, attVect) || attackedFromAbove(i, attVect) ||

          attackedFromUpperRight(i, attVect))

      {

        return false;

      }

    }

    return true;

  }

  static boolean isOpenSpace(int row, int col) {

    if (row < 0 || row >= n) {

      return false;

    }

    if (col < 0 || col >= m) {

      return false;

    }

    return board[row].charAt(col) != '#';

  }

  static boolean hasQueen(String rowConf, int col) {

    if (col < 0 || col >= m) {

      return false;

    }

    return rowConf.charAt(col) == '1';

  }

  static boolean attackedFromUpperLeft(int col, String attVect) {

    if (col <= 0) {

      return false;

    }

    return attVect.charAt(col \* 3) == '1';

  }

  static boolean attackedFromAbove(int col, String attVect) {

    return attVect.charAt((col \* 3) + 1) == '1';

  }

  static boolean attackedFromUpperRight(int col, String attVect) {

    if (col >= m - 1) {

      return false;

    }

    return attVect.charAt((col \* 3) + 2) == '1';

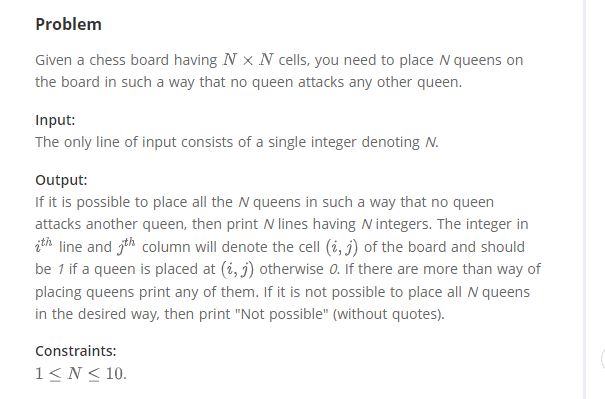
  }

# TEST CASES:

# 

**PROBLEM STATEMENT 9.2: -**

<https://www.hackerearth.com/practice/basic-programming/recursion/recursion-and-backtracking/practice-problems/algorithm/n-queensrecursion-tutorial/>



**SOLUTION:**

#include<iostream>

using namespace std;

bool issafe(int \*\* arr, int x, int y, int n) {

for (int row = 0; row < x; row++) {

if (arr[row][y] == 1) {

return 0;

}

}

int row = x;

int col = y;

while (row >= 0 && col >= 0) {

if (arr[row][col] == 1) {

return 0;

}

row--;

col--;

}

row = x;

col = y;

while (row >= 0 && col < n) {

if (arr[row][col] == 1) {

return 0;

}

row--;

col++;

}

return 1;

}

bool nqueen(int \*\* arr, int x, int n) {

if (x >= n) {

return 1;

}

for (int col = 0; col < n; col++) {

if (issafe(arr, x, col, n)) {

arr[x][col] = 1;

if (nqueen(arr, x + 1, n)) {

return 1;

}

arr[x][col] = 0;

}

}

return 0;

}

int main() {

int n;

cin >> n;

int \*\* arr = new int \* [n];

for (int i = 0; i < n; i++) {

arr[i] = new int[n];

for (int j = 0; j < n; j++) {

arr[i][j] = 0;

}

}

if (nqueen(arr, 0, n)) {

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

cout << arr[i][j] << " ";

}

cout << endl;

}

} else {

cout << "Not possible" << endl;

}

}

# TEST CASES:

# 