EXPERIMENT 3.2 (BACKTRACKING)

Student Name: Jitesh Kumar **UID: 20BCS2334**
Branch: BE-CSE **Section/Group: 903-A**
Semester: 5th **Subject Code: 20CSP-314**

Problem 1

Queens on Board

Code-

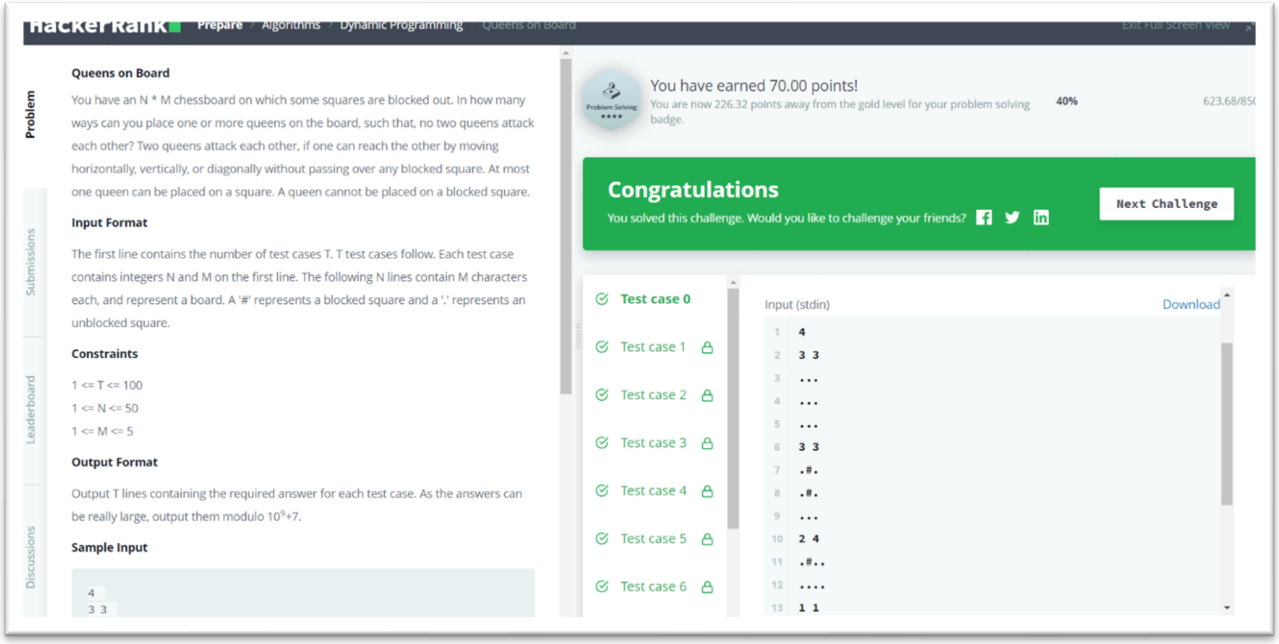
```
#include <vector>
#include <string>
#include <algorithm>
#include <iostream>
#include <unordered_map>
#include <cassert>

using namespace std;

struct Solution2
{
    typedef basic_string<unsigned char>___Board;
    typedef___Board::value_type         __Row;
    long long solve(const vector<string> & B){
        if (B.empty() || B[0].empty())
            return 0;

        for (size_t i = 0; i < B.size(); ++i){
            __Row row = 0;
            for (size_t j = 0; j < B[i].size(); ++j){
                if ('.' == B[i][j])
                    row |= (1 << j);
            }
            row = ~row;
            board.push_back(row);

            __Board p;
            genPlacements(row, p, B[i].size());
            placements.push_back(p);
        }
        bmask = (1 << B[0].size()) - 1;
        return help(0, 0, 0, 0);
    }
//Sargun Kohli 20BCS1515

private:
    static void genPlacements(__Row block,__Board & ret, int M){
        for (int i = 0; i < M; ++i){

            __Row p1 = 1 << i;
            if (0 != (p1 & block))
                continue;
            ret.push_back(p1);

            for (int j = i + 2; j < M; ++j){
                __Row p2 = p1 | (1 << j);
                if (0 != (p2 & block))
                    continue;
                __Row m2 = (1 << j) - (1 << (i + 1));
                if (0 == (m2 & block))
                    continue;
                ret.push_back(p2);

                for (int k = j + 2; k < M; ++k){
                    __Row p3 = p2 | (1 << k);
                    if (0 != (p3 & block))
                        continue;
                    __Row m3 = (1 << k) - (1 << (j + 1));
                    if (0 == (m3 & block))
            }
            continue;   //there is not enough blocks between 3 Qs
        ret.push_back(p3);
```

```
}
```

```
        }
    }
}
__Row calcMask(__Row mask,__Row blocks){
    __Row b = mask & blocks;
    mask &= ~b;
    return (mask & bmask);
}
static int hash(size_t row,__Row lmask,__Row dmask,__Row rmask){
    int r = row;
    r <<= 8;
    r += lmask;
    r <<= 8;
    r += dmask;
    r <<= 8;
    r += rmask;
    return r;
}
long long help(size_t row,__Row lmask,__Row dmask,__Row rmask){
    if (row >= board.size())
        return 0;
    const int h = hash(row, lmask, dmask, rmask);
    unordered_map<int, long long>::const_iterator wh = save.find(h);
    if (wh != save.end())
        return wh->second;
    const __Row blocks = board[row];
    const __Row mask = lmask | dmask | rmask | blocks;
    long long ret = 0;

    lmask = calcMask(lmask, blocks);
    dmask = calcMask(dmask, blocks);
    rmask = calcMask(rmask, blocks);
    if (__Row(-1) != mask){

        const __Board & ps = placements[row];
        for (size_t i = 0; i < ps.size(); ++i){
            const __Row p = ps[i];
```

```
                if (0 != (mask & p))
                    continue;
                ++ret;

                ret += help(row + 1, (lmask | p) << 1, dmask | p, (rmask | p) >> 1);
            }
        }
        ret += help(row + 1, lmask << 1, dmask, rmask >> 1);
        return (save[h] = ret % 1000000007);
    }
    __Board board;
    vector<__Board> placements;
    unordered_map<int, long long> save;
    __Row bmask;
};

typedef Solution2 Solution;

int main()
{

    int t;
    cin >> t;
    while (t--){
        int n, m;
        cin >> n >> m;
        vector<string> b;
        for (int i = 0; i < n; ++i){
            string line;
            cin >> line;
            b.push_back(line);
        }
        cout << Solution().solve(b) << endl;
    }
    return 0;
}
```

Output



Queens on Board

You have an N * M chessboard on which some squares are blocked out. In how many ways can you place one or more queens on the board, such that, no two queens attack each other? Two queens attack each other, if one can reach the other by moving horizontally, vertically, or diagonally without passing over any blocked square. At most one queen can be placed on a square. A queen cannot be placed on a blocked square.

Input Format

The first line contains the number of test cases T. T test cases follow. Each test case contains integers N and M on the first line. The following N lines contain M characters each, and represent a board. A '#' represents a blocked square and a '.' represents an unblocked square.

Constraints

1 <= T <= 100
1 <= N <= 50
1 <= M <= 5

Output Format

Output T lines containing the required answer for each test case. As the answers can be really large, output them modulo 10^9+7.

Sample Input

```
4
3 3
```

You have earned 70.00 points!
You are now 226.32 points away from the gold level for your problem solving badge.
40%
623.68/850

Congratulations
You solved this challenge. Would you like to challenge your friends?
Next Challenge

Test case 0
Test case 1
Test case 2
Test case 3
Test case 4
Test case 5
Test case 6

Input (stdin) Download

```
4
3 3
...
...
...
3 3
.#.
.#.
...
2 4
.#..
....
1 1
```

Problem 2

N-Queens

Code-

import java.util.Scanner;

public class NQueensRecursion {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int n = sc.nextInt();

// sc.close();

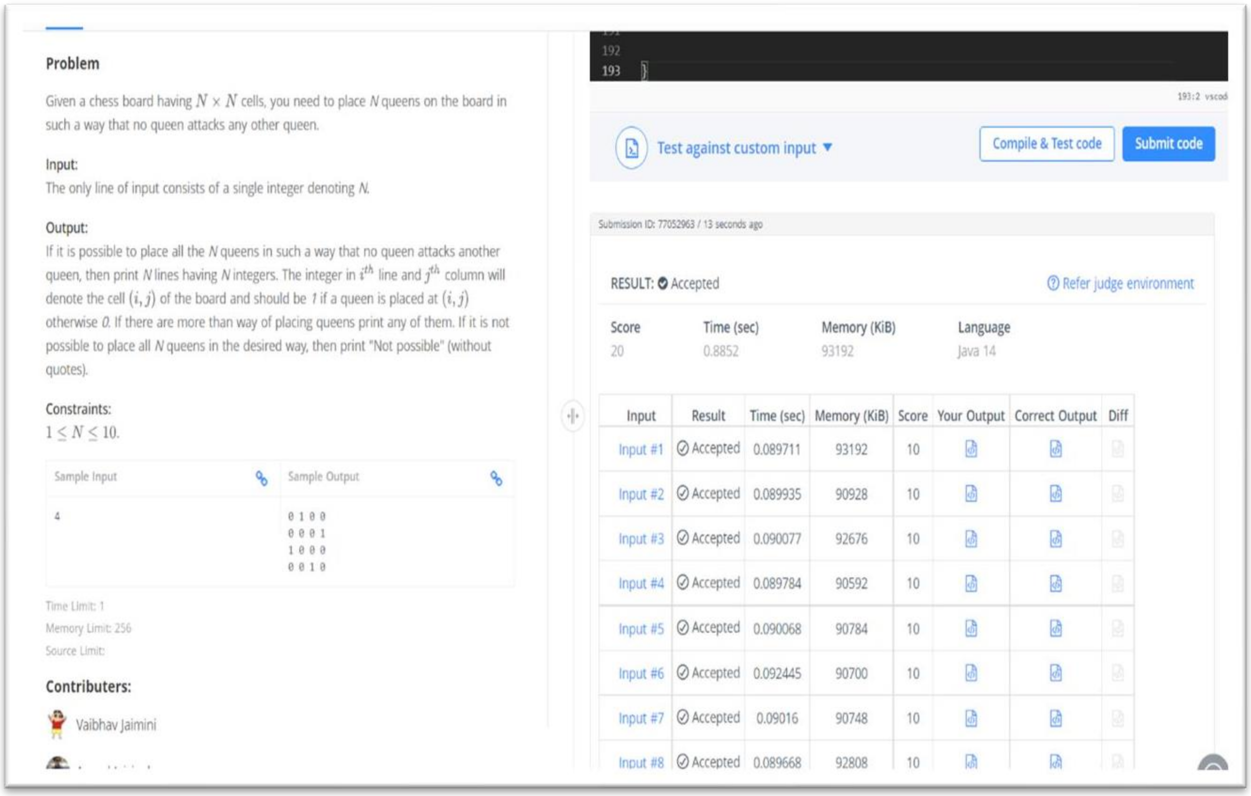
```
int[][] arr = new int[11][11];

int pos = 0;

if (n == 2 || n == 3) {

System.out.println("Not possible");

}

boolean b = nqueen(arr, n, pos);

}

private static boolean nqueen(int[][] arr, int n,int i) {

if (n==i){

//Successfully place the queens in n row position
//Print the rows

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

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

if (arr[j][k]==0){

System.out.print("0 ");

}

else

System.out.print("1 ");
}

System.out.print("\n");

}

return true;

}

//Recursive Case

//Try to place the Queen in the Front row only rest will be handeled by the
recursive leap of faith

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

//check if i,j position is safe to place the Queen or not

if (isSafe(arr,i,j,n)) {

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

//Placing the Queen Assuming it is the Right Position

arr[i][j] = 1;

boolean nextQueenRakhPaRaheHai = nqueen(arr,n, i + 1);

if (nextQueenRakhPaRaheHai) {

return true;

}
```

```
//If we came here it means that the next position is not filled,

//Our assumption is Wrong.

arr[i][j] = 0;

}

}

//You have tried for all position in the current row but couldn't place a queen

return false;

}

private static boolean isSafe(int[][] board, int i, int j, int n){

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

if (board[row][j]==1){

return false;

}

}

//left Diagonal

int x = i;

int y = j;

while (x>=0 &&y>=0){
```

```
if (board[x][y]==1){

return false;

}

x--;

y--;

}

//Right Diagonal

x = i;

y = j;

while (x>=0 && y<n){

if (board[x][y]==1){

return false;

}

x--;

y++;

}

//Since we have checked the row And Columns

return true;

}
```

```
}
```

Output



Problem

Given a chess board having $N \times N$ cells, you need to place N queens on the board in such a way that no queen attacks any other queen.

Input:
The only line of input consists of a single integer denoting N.

Output:
If it is possible to place all the N queens in such a way that no queen attacks another queen, then print N lines having N integers. The integer in i^{th} line and j^{th} column will denote the cell (i, j) of the board and should be 1 if a queen is placed at (i, j) otherwise 0. If there are more than way of placing queens print any of them. If it is not possible to place all N queens in the desired way, then print "Not possible" (without quotes).

Constraints:
$1 \le N \le 10$.

Sample Input	Sample Output

4	0 1 0 0
	0 0 0 1
	1 0 0 0
	0 0 1 0

Time Limit: 1
Memory Limit: 256
Source Limit:

Contributers:

Vaibhav Jaimini

```
}
```

Submission ID: 77052963 / 13 seconds ago

RESULT: Accepted

Score	Time (sec)	Memory (KiB)	Language
20	0.8852	93192	Java 14

Input	Result	Time (sec)	Memory (KiB)	Score	Your Output	Correct Output	Diff
Input #1	Accepted	0.089711	93192	10	📄	📄	📄
Input #2	Accepted	0.089935	90928	10	📄	📄	📄
Input #3	Accepted	0.090077	92676	10	📄	📄	📄
Input #4	Accepted	0.089784	90592	10	📄	📄	📄
Input #5	Accepted	0.090068	90784	10	📄	📄	📄
Input #6	Accepted	0.092445	90700	10	📄	📄	📄
Input #7	Accepted	0.09016	90748	10	📄	📄	📄
Input #8	Accepted	0.089668	92808	10	📄	📄	📄