Experiment - 9

Student Name: Anjali Singh **UID:** 20BCS9239
Branch: CSE **Section/Group:** WM_607 A
Semester: 5th **Subject Code:** 20CSP-314
Subject Name: Competitive Coding

PROBLEM STATEMENT 9.1

Given a chess board having N×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 ith line and jth 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).

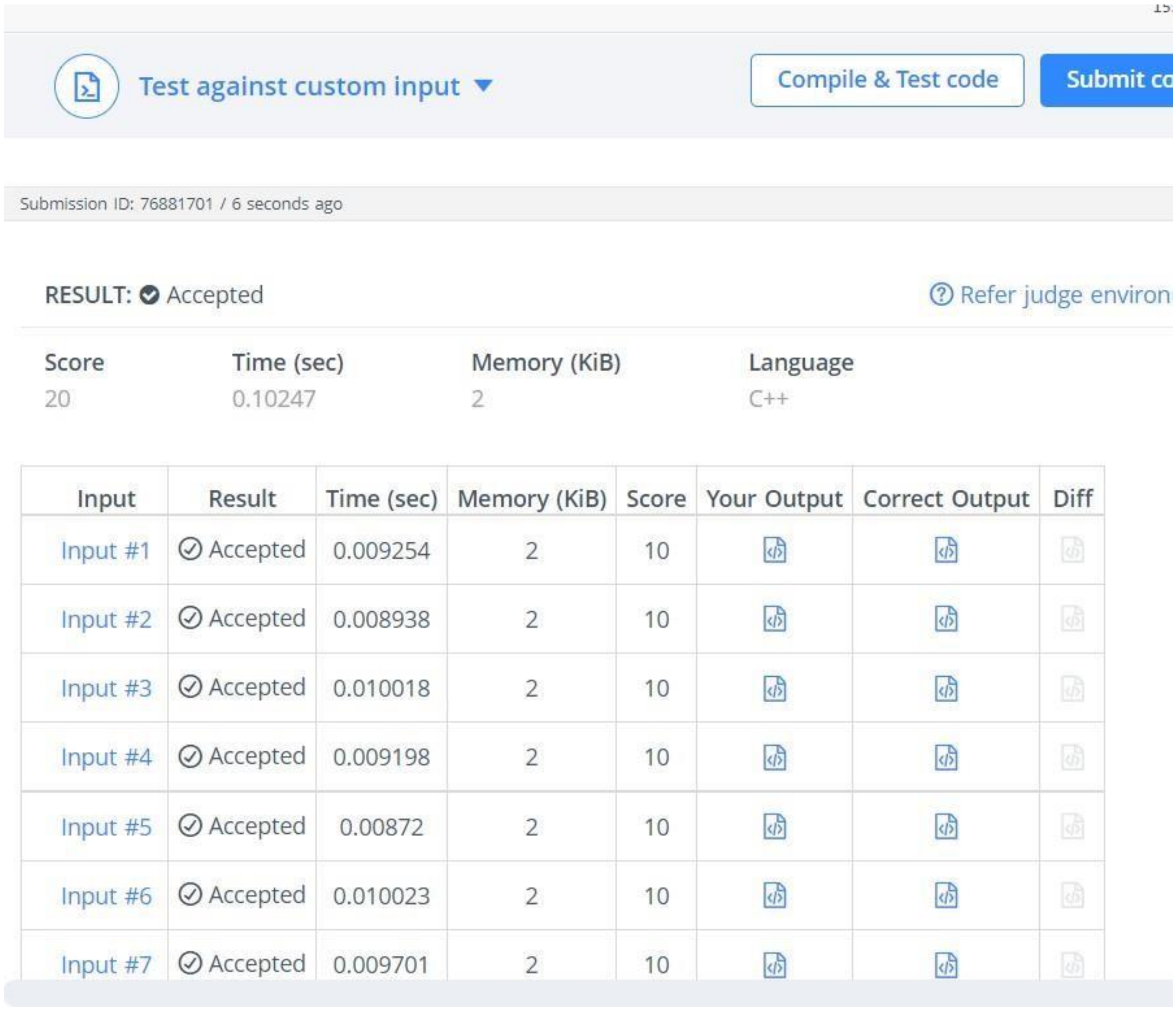
Constraints: 1≤N≤10

Solution:

```
#include<bits/stdc++.h> using namespace std;

int arr[11][11]; bool ap=true; bool is_right(int

n,int row,int col,int arr[][11]) {

    //checking here perticuler col        for(int

i=row-1;i>=0;i--)
```

```
        {
if(arr[i][col]==1)                 return
false;
        }
        //checking left diagonal         for(int i=row-
1,j=col-1;i>=0&&j>=0;i--,j--)
        {
if(arr[i][j]==1)                 return
false;
        }
        //checking for right diagonal         for(int
i=row-1,j=col+1;i>=0&&j<n;i--,j++)
        {                 if(arr[i][j]==1)
return false;         }         return true; } void
queenhelper(int n,int row,int arr[][11])
{         //base case
if(row==n)
        {
if(ap==true){
for(int i=0;i<n;i++)
                {                         for(int j=0;j<n;j++)
                        {
cout<<arr[i][j]<<" ";
                        }
cout<<endl;
                }                 ap=false;
                }
cout<<endl;
        }
        //small calculation         for(int
j=0;j<n;j++)
        {
if(is_right(n,row,j,arr))
                {                          arr[row][j]=1;
queenhelper(n,row+1,arr);
                        //here backtracking use,wait in stack(internal stack)                         arr[row][j]=0;
                }
        }
} void placeNQueens(int
n){ memset(arr,0,11*11*sizeo
f(int)); queenhelper(n,0,arr);
if(ap==true) cout<<"Not
possible"<<endl;


}

int main(){


int n; cin >>
n ;


placeNQueens(n);



   }
```

OUTPUT:



Submission ID: 76881701 / 6 seconds ago

RESULT: Accepted

Score	Time (sec)	Memory (KiB)	Language
20	0.10247	2	C++

Input	Result	Time (sec)	Memory (KiB)	Score	Your Output	Correct Output	Diff
Input #1	Accepted	0.009254	2	10			
Input #2	Accepted	0.008938	2	10			
Input #3	Accepted	0.010018	2	10			
Input #4	Accepted	0.009198	2	10			
Input #5	Accepted	0.00872	2	10			
Input #6	Accepted	0.010023	2	10			
Input #7	Accepted	0.009701	2	10			

PROBLEM STATEMENT 9.2

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

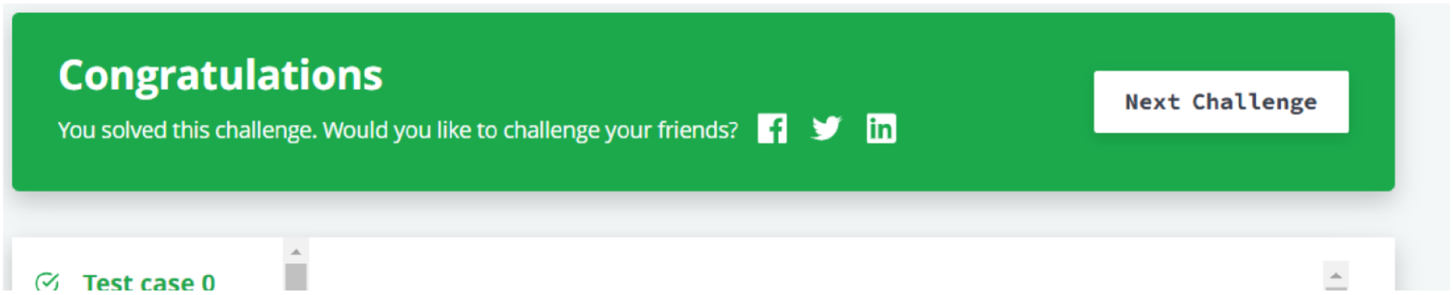
```
4
3 3
...
...
...
3 3   .#.
.#.    ...
2 4
.#..   ....
1 1
#
```

Sample Output

```
17
18    14
0
```

Solution:

```
#include <cstdio>
#include <cstring>
#define MOD 1000000007

// Solution uses memoization of a brute-force solving of all permutations.

// Number of rows and columns.
int n, m;
// Character representation of the grid. char g[50][5];
// Valid configurations for row i (up to 2**5 of them). int good[50][1
<< 5];
// Number of valid configurations in row i. int szg[50];
// Array of N bitmasks containing 111 for each M.
int block[50];
// Number of solutions for a given row and bitmask.
int memo[50][1 << 15];
// The next bitmask, given a particular bitmask.
int memo2[1 << 15];

// Get the next bitmask with queen attack vectors accounted for. int spread(int
mask)
{
  // Solutions been found before, use it.
  if (memo2[mask] != -1) return memo2[mask];

  int nmask = 0;   // For
each square   for (int i = 0; i
< m; i++) {
    // If a left-angling attack vector exists and we're not at the left edge,
// extend it into the left-diagonal square.      if (mask & 1 << 3 * i &&
i > 0) {
      nmask |= 1 << 3 * i - 3;
    }
    // If a vertical attack vector exists, extend it into the square below.
    if (mask & 1 << 3 * i + 1) {
      nmask |= 1 << 3 * i + 1;
    }
    // If a right-angling attack vector exists and we're not at the right edge,     //
extend it into the right-diagonal square.     if (mask & 1 << 3 * i + 2
&& i + 1 < m) {
nmask |= 1 << 3 * i + 5;
    }
  }
  return memo2[mask] = nmask;
}

// Solve for row x, with a mask for squares that are blocked by earlier queens. int solve
(int x, int mask)
{
  // We've reached the end of a solution, so return.
  if (x == n) return 1;

  // Adjust the mask for squares that are already blocked.
mask &= ~block[x];
  // If we've already solved this, return the result.   if
(memo[x][mask] != -1) return memo[x][mask];

  // If not, count solutions.
  int ret = 0;
  // For each configuration of this row   for
(int i = 0; i < szg[x]; i++) {     // If we haven't
tried all configurations yet     if
(!(good[x][i] & mask)) {
      // Try the next row, with rows blocked by queens masked out.
      ret += solve(x + 1, spread(good[x][i] | mask));
// Don't let the number get too big.       if(ret >= MOD)
ret -= MOD;
    }
  }
  return memo[x][mask] = ret;
}

int solve()
{
  // For each row in N   for
(int i = 0; i < n; i++) {
block[i] = 0;     int cmask =
0;     for (int j = 0; j < m; j++)
{       if (g[i][j] == '#') {
        // Create bitmask for #, like 01001         cmask
|= 1 << j;
        // Create bitmask with 111111111111111 (for M==5)
        block[i] |= 7 << 3 * j;
      }
    }

    szg[i] = 0;
    // For each 2**M permutations of queens on this row
    for (int j = 0; j < 1 << m; j++) {
// If the queen is not on a #       if ((j
& cmask) == 0) {         bool valid =
true;         // For each column in M
for (int k = 0; k
< m; k++) {
          // If a queen is in that column, and another queen is on the same row           //
before another #, then that position isn't valid.
          if (j & 1 << k) {              for (int kk = k + 1; kk < m
&& g[i][kk] != '#'; kk++) {                if (j & 1 << kk) {
valid = false;
              }
            }
          }
        }
        if (!valid) continue;

        // If this is a valid configuration of queens for this row, create a         //
bitmask with 111 where a queen is. Add it to the good matrix at row i         //
and increment the size of that row (szg).          int sp = 0;          for (int k = 0;
k < m; k++) {
if (j & 1 << k) {
            sp |= 7 << 3 * k;
          }         }
good[i][szg[i]] = sp;
szg[i]++;
      }
    }
  }

  // Initialize memoization tables to -1 for each entry.
  memset(memo,255,sizeof memo);
memset(memo2,255,sizeof memo2);
//    Solve    recursively.           return
solve(0,0);
}

int main() {
  int runs;
scanf("%d",&runs);   while
(runs--) {
scanf("%d%d",&n,&m);
   // Initialize board.      for (int i = 0;i <
n;i++) scanf("%s",g[i]);    int ret = solve();
ret   =   (ret   -   1   +   MOD)   %   MOD;
printf("%d\n",ret);   }   return 0; }
```


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

OUTPUT: