Symbiosis Institute of Technology

Department of Computer Science and Engineering

Academic Year 2025-26

Design Analysis of Algorithm– Lab

Batch 2023-27 - Sem V

Lab Assignment No:- 8	
Name of Student	Deepti Pal
PRN No.	23070122081
Batch	2023-27 TY
Class	CSE A3
Academic Year & Semester	2025-26 TY , 5th sem
Date of Submission	11 oct 25
Title of Assignment:	Implement a 4-queens problem using backtracking. **Secure Server Deployment** **Problem Statement:** A cybersecurity firm is designing a secure network layout for a data center. The data center consists of a **4x4 server grid**. Each row represents a different **security zone**, and each column represents a **power supply route**. The goal is to **deploy 4 high-security servers** in such a way that:

1. Only **one server is placed in each row**.
2. No two servers share the **same column**.
3. No two servers are on the same **diagonal** (to prevent cascading failures from one zone to another).

This is structurally identical to the classic **4-Queens Problem**, but with a real-world twist. Implement a solution that finds **all valid configurations** for placing the 4 servers, following the rules above.

Input

There is **no input** for this problem. You must generate all valid server placement configurations for a **4x4 grid**.

Output

Return a list of all valid configurations.
Each configuration must be a list of 4 strings, each string of length 4, where:

- `'S'` represents a placed server.
- `'.'` represents an empty space.

```
[
  [
    ".S..",
    "...S",
    "S...",
    "..S."
  ],
  [
    "..S.",
    "S...",
    "...S",
    ".S.."
  ]
]
```

Explanation

There are **2 valid server deployment configurations** for a 4x4 grid that meet the given constraints. In each, no two servers are in the same row, column, or diagonal.

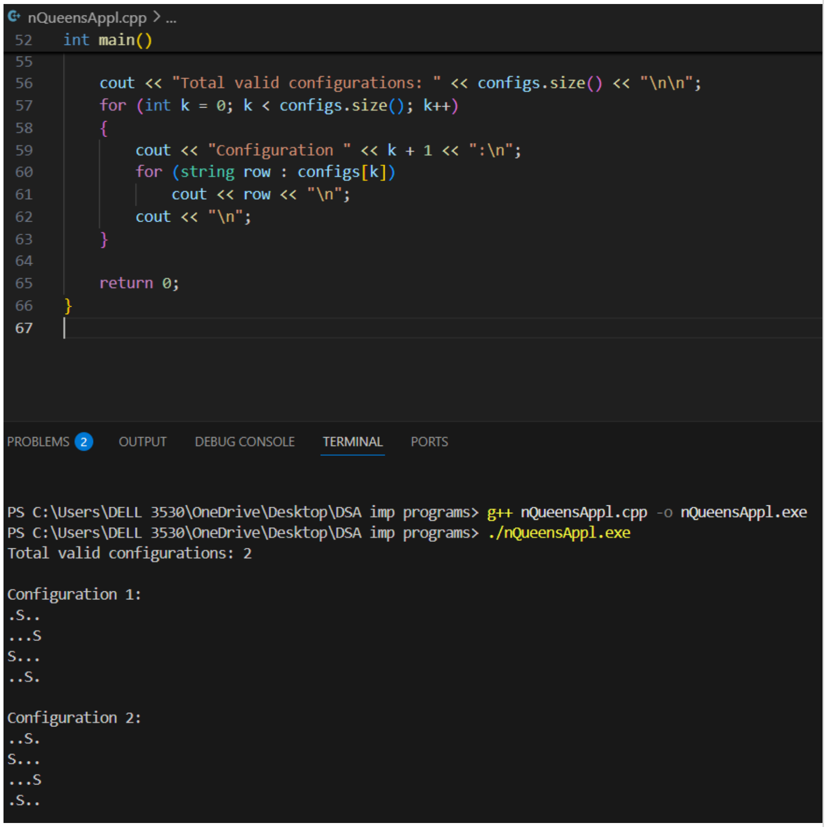
Constraints

- Grid size is fixed to 4x4.
- Exactly one server per row must be placed.
- No two servers can be in the same column or diagonal.

Source code

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;

bool isSafe(vector<string> &board, int row, int col, int n)
{

    for (int i = 0; i < row; i++)
        if (board[i][col] == 'S')
            return false;

    for (int i = row - 1, j = col - 1; i >= 0 && j >= 0; i--, j--)
        if (board[i][j] == 'S')
            return false;

    for (int i = row - 1, j = col + 1; i >= 0 && j < n; i--, j++)
        if (board[i][j] == 'S')
            return false;

    return true;
}

void solve(int row, vector<string> &board, vector<vector<string>>
&result, int n)
{
    if (row == n)
    {

        result.push_back(board);
        return;
    }
    for (int col = 0; col < n; col++)
    {
        if (isSafe(board, row, col, n))
        {
            board[row][col] = 'S';
            solve(row + 1, board, result, n);
            board[row][col] = '.';
        }
    }
}

vector<vector<string>> secureServerDeployment(int n = 4)
{
    vector<vector<string>> result;
    vector<string> board(n, string(n, '.'));
    solve(0, board, result, n);
    return result;
}

int main()
{
    vector<vector<string>> configs = secureServerDeployment();

    cout << "Total valid configurations: " << configs.size() << "\n\n";
    for (int k = 0; k < configs.size(); k++)
    {
        cout << "Configuration " << k + 1 << ":\n";
        for (string row : configs[k])
            cout << row << "\n";
        cout << "\n";
    }

    return 0;
}
```

Output Screenshots (if applicable)

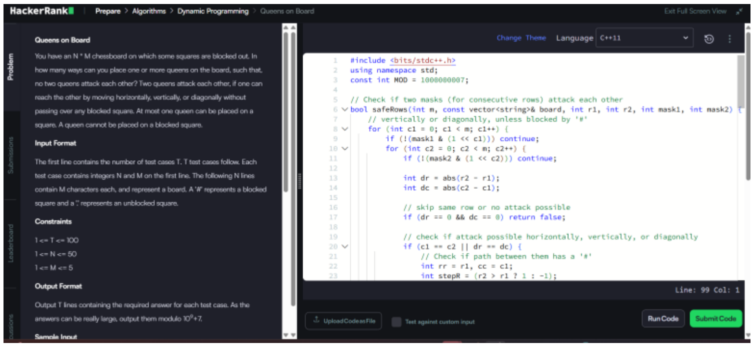


```cpp
int main()

    cout << "Total valid configurations: " << configs.size() << "\n\n";
    for (int k = 0; k < configs.size(); k++)
    {
        cout << "Configuration " << k + 1 << ":\n";
        for (string row : configs[k])
            cout << row << "\n";
        cout << "\n";
    }

    return 0;
}

```

```
PS C:\Users\DELL 3530\OneDrive\Desktop\DSA imp programs> g++ nQueensAppl.cpp -o nQueensAppl.exe
PS C:\Users\DELL 3530\OneDrive\Desktop\DSA imp programs> ./nQueensAppl.exe
Total valid configurations: 2

Configuration 1:
.S..
...S
S...
..S.

Configuration 2:
..S.
S...
...S
.S..
```

Hacker Rank Problem

1. https://www.hackerrank.com/challenges/queens-on-board/problem?utm_source=chatgpt.com



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

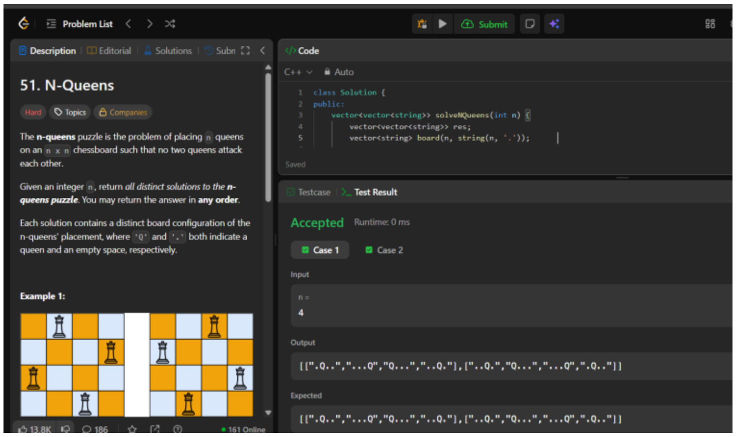
Output T lines containing the required answer for each test case. As the answers can be really large, output them modulo 10^9+7.

Sample Input

```cpp
#include <bits/stdc++.h>
using namespace std;
const int MOD = 1000000007;

// Check if two masks (for consecutive rows) attack each other
bool safeRows(int m, const vector<string>& board, int r1, int r2, int mask1, int mask2) {
    // vertically or diagonally, unless blocked by '#'
    for (int c1 = 0; c1 < m; c1++) {
        if (!(mask1 & (1 << c1))) continue;
        for (int c2 = 0; c2 < m; c2++) {
            if (!(mask2 & (1 << c2))) continue;

            int dr = abs(r2 - r1);
            int dc = abs(c2 - c1);

            // skip same row or no attack possible
            if (dr == 0 && dc == 0) return false;

            // check if attack possible horizontally, vertically, or diagonally
            if (c1 == c2 || dr == dc) {
                // Check if path between them has a '#'
                int rr = r1, cc = c1;
                int stepR = (r2 > r1 ? 1 : -1);
```

Line: 99 Col: 1

2. https://leetcode.com/problems/n-queens/



51. N-Queens

The **n-queens** puzzle is the problem of placing `n` queens on an `n x n` chessboard such that no two queens attack each other.

Given an integer `n`, return *all distinct solutions to the **n-queens puzzle***. You may return the answer in **any order**.

Each solution contains a distinct board configuration of the n-queens' placement, where `'Q'` and `'.'` both indicate a queen and an empty space, respectively.

Example 1:

```
class Solution {
public:
    vector<vector<string>> solveNQueens(int n) {
        vector<vector<string>> res;
        vector<string> board(n, string(n, '.'));
```

Saved

Accepted Runtime: 0 ms

Input

```
n =
4
```

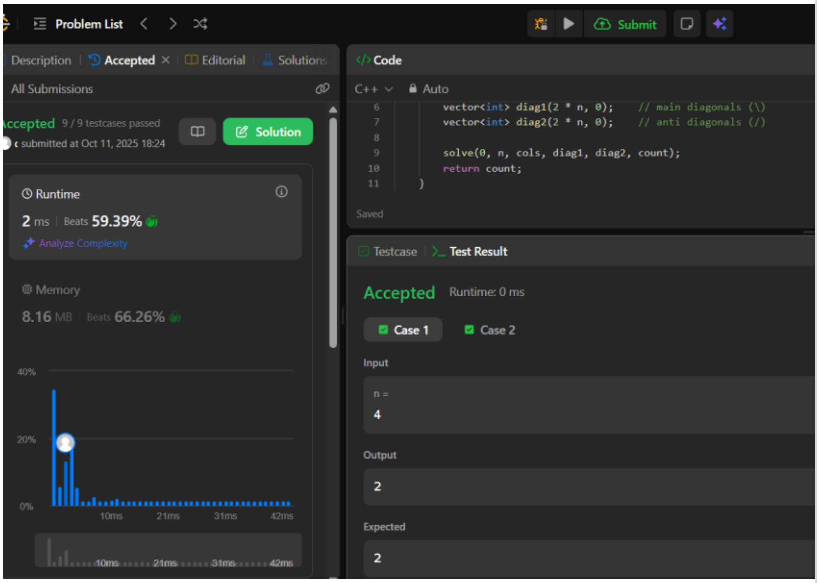
Output

```
[[".Q..","...Q","Q...","..Q."],["..Q.","Q...","...Q",".Q.."]]
```

Expected

```
[[".Q..","...Q","Q...","..Q."],["..Q.","Q...","...Q",".Q.."]]
```

3. https://leetcode.com/problems/n-queens-ii/



ccepted 9 / 9 testcases passed

submitted at Oct 11, 2025 18:24

Runtime

2 ms | Beats 59.39%

Memory

8.16 MB | Beats 66.26%

```
        vector<int> diag1(2 * n, 0);    // main diagonals (\)
        vector<int> diag2(2 * n, 0);    // anti diagonals (/)

        solve(0, n, cols, diag1, diag2, count);
        return count;
    }
```

Saved

Accepted Runtime: 0 ms

Input

```
n =
4
```

Output

```
2
```

Expected

```
2
```

Conclusion	Thus, we have studied the N-queen algorithm using backtracking method.