**Name:** Aditya Jethani

**Roll No:** 21BCP166

**DIV-**3 **G-**5

**DAA Lab assignment 8**

**Aim:** Use Backtracking to solve N-Queens Problem

**Theory:**

The N Queens problem is a well-known problem in computer science and mathematics. It is a classic problem of placing N chess queens on an N x N chessboard so that no two queens attack each other.

The N Queens problem can be solved using different algorithms, including backtracking, recursive, and iterative algorithms. The backtracking algorithm is one of the most commonly used algorithms for solving the N Queens problem. It is an efficient algorithm that uses a depth-first search approach to explore all possible solutions of the problem. The backtracking algorithm works by placing queens in different cells of the chessboard and checking if they are in conflict with each other. If they are not in conflict, the algorithm moves to the next row and places the next queen. If all queens are placed on the board, the algorithm returns true. If a conflict is detected, the algorithm backtracks to the previous cell and tries a different position for the queen.

**CODE:**

#include <iostream>

#include <vector>

using namespace std;

// Function to display the chessboard with queens placed

void display\_board(vector<int>& queens) {

    int n = queens.size();

    //outer loop for rows

    for (int i = 0; i < n; i++) {

        //inner loop for columns

        for (int j = 0; j < n; j++) {

            if (queens[i] == j) {

                cout << " Q ";

            } else {

                cout << " - ";

            }

        }

        cout << endl;

    }

    cout << endl;

}

// Function to check if the current queen placement is valid

bool is\_valid(vector<int>& queens, int row, int col) {

    for (int i = 0; i < row; i++) {

        if (queens[i] == col || abs(queens[i] - col) == abs(i - row)) {

            // A queen can attack another queen if they are in the same column or diagonal

            return false;

        }

    }

    // If no other queen can attack the current queen, then it is a valid placement

    return true;

}

// Function to solve the N queens problem using backtracking

void solve\_n\_queens(vector<int>& queens, int row, int& count) {

    int n = queens.size();

    if (row == n) {

        // If all queens have been placed on the board, we have found a valid solution

        count++;

        cout << "Solution " << count << ":" << endl;

        display\_board(queens);

        return;

    }

    // Try placing the queen in each column of the current row

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

        if (is\_valid(queens, row, col)) {

            queens[row] = col;

            // If the current queen placement is valid, move on to the next row

            solve\_n\_queens(queens, row + 1, count);

            queens[row] = -1;

            // Backtrack and try the next column if the current placement does not lead to a solution

        }

    }

}

int main() {

    int n;

    cout << "Enter the number of queens: ";

    cin >> n;

    // The queens vector stores the column number of the queen in each row

    vector<int> queens(n, -1);

    int count = 0;

    // Start placing queens from the first row

    solve\_n\_queens(queens, 0, count);

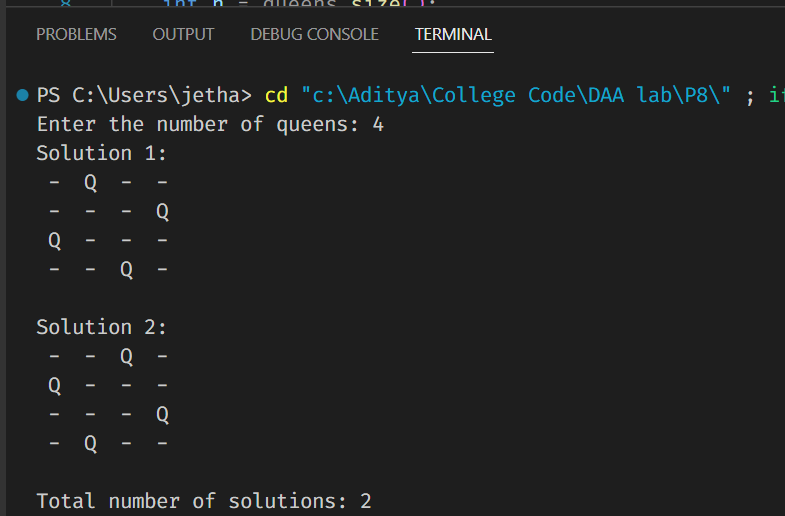
    //displaying the total number of solutions

    cout << "Total number of solutions: " << count << endl;

    return 0;

}

**OUTPUT:**

****