**Experiment-8**

**Aim-** Consider the problem of N queen on an (NxN) chessboard. Two queens are said to attack each other if they are on the same row, column, or diagonal. Implements backtracking algorithm to solve the problem i.e. place N non-attacking queens on the board.

**Theory-**

The N-Queens Problem is a classic problem of placing N queens on an N×N chessboard such that no two queens threaten each other. Using backtracking, the solution places queens one column at a time, ensuring that no two queens share the same row, column, or diagonal. If placing a queen leads to a conflict, the algorithm backtracks and tries the next possible position.

**Software Used –** Visual Studio Code

**Code-**

**#**include <iostream>

#include <vector>

using namespace std;

using namespace std;

class Solution {

public:

// Check if it's safe to place a queen at board[row][col]

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

int r = row, c = col;

// Check upper left diagonal

while (r >= 0 && c >= 0) {

if (board[r][c] == 'Q') return false;

r--;

c--;

}

// Reset to the original position

r = row;

c = col;

// Check left side

while (c >= 0) {

if (board[r][c] == 'Q') return false;

c--;

}

// Reset to the original position

r = row;

c = col;

// Check lower left diagonal

while (r < board.size() && c >= 0) {

if (board[r][c] == 'Q') return false;

r++;

c--;

}

// If no queens are found, it's safe

return true;

}

// Function to place queens on the board

void func(int col, vector<vector<string>>& ans, vector<string>& board) {

// If all columns are filled, add the solution to the answer

if (col == board.size()) {

ans.push\_back(board);

return;

}

// Try placing a queen in each row for the current column

for (int row = 0; row < board.size(); row++) {

// Check if it's safe to place a queen

if (safe(board, row, col)) {

// Place the queen

board[row][col] = 'Q';

// Recursively place queens in the next columns

func(col + 1, ans, board);

// Remove the queen and backtrack

board[row][col] = '.';

}

}

}

// Solve the N-Queens problem

vector<vector<string>> solveNQueens(int n) {

// List to store the solutions

vector<vector<string>> ans;

// Initialize the board with empty cells

vector<string> board(n, string(n, '.'));

// Start placing queens from the first column

func(0, ans, board);

return ans;

}

};

// Main method to test the solution

int main() {

Solution solution;

int n = 4; // Example with 4 queens

vector<vector<string>> solutions = solution.solveNQueens(n);

// Print all solutions

for (const auto& sol : solutions) {

for (const auto& row : sol) {

cout << row << endl;

}

cout << endl;

}

return 0;

}

**Output-**

**A screen shot of a computer

Description automatically generated**