**DAA Assignment 5**

**N-Queens Problem Using Backtracking**

**Neeti Kurulkar**

**#include <iostream>**

**#include <vector>**

**#include <ctime>**

**using namespace std;**

**// Function to check if placing a queen at (row, col) is safe**

**bool isSafe(const vector<int>& rows,**

**const vector<int>& hills,**

**const vector<int>& dales,**

**int row, int col, int n) {**

**// Calculate hill (/) and dale (\) diagonal indices**

**int hillIndex = row - col + n - 1;**

**int daleIndex = row + col;**

**// Safe if row, hill, and dale are free**

**return !rows[row] && !hills[hillIndex] && !dales[daleIndex];**

**}**

**// Column-wise backtracking to place queens**

**bool solveNQueensUtil(int col, int n,**

**vector<int>& queens,**

**vector<int>& rows,**

**vector<int>& hills,**

**vector<int>& dales,**

**int firstCol) {**

**if (col >= n) return true; // All queens placed**

**// Skip the column of the first pre-placed queen**

**if (col == firstCol) {**

**return solveNQueensUtil(col + 1, n, queens, rows, hills, dales, firstCol);**

**}**

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

**if (isSafe(rows, hills, dales, row, col, n)) {**

**// Place queen**

**queens[col] = row;**

**rows[row] = 1;**

**hills[row - col + n - 1] = 1;**

**dales[row + col] = 1;**

**if (solveNQueensUtil(col + 1, n, queens, rows, hills, dales, firstCol))**

**return true; // Solution found**

**// Backtrack**

**queens[col] = -1;**

**rows[row] = 0;**

**hills[row - col + n - 1] = 0;**

**dales[row + col] = 0;**

**}**

**}**

**return false; // No valid placement in this column**

**}**

**// Print the board**

**void printBoard(const vector<int>& queens, int n) {**

**for (int i = 0; i < n; i++) {**

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

**cout << (queens[j] == i ? "Q " : ". ");**

**cout << endl;**

**}**

**}**

**int main() {**

**int n;**

**cout << "Enter the value of N: ";**

**cin >> n;**

**int firstRow, firstCol;**

**cout << "Enter the row and column for the first queen (0-indexed): ";**

**cin >> firstRow >> firstCol;**

**if (firstRow < 0 || firstRow >= n || firstCol < 0 || firstCol >= n) {**

**cout << "Invalid position for first queen." << endl;**

**return 1;**

**}**

**vector<int> queens(n, -1); // queens[col] = row**

**queens[firstCol] = firstRow; // Place first queen**

**vector<int> rows(n, 0); // Rows occupied**

**vector<int> hills(2 \* n - 1, 0); // Major diagonals (/)**

**vector<int> dales(2 \* n - 1, 0); // Minor diagonals (\)**

**// Mark the first queen’s threats**

**rows[firstRow] = 1;**

**hills[firstRow - firstCol + n - 1] = 1;**

**dales[firstRow + firstCol] = 1;**

**clock\_t start = clock();**

**if (solveNQueensUtil(0, n, queens, rows, hills, dales, firstCol)) {**

**cout << "Solution found:\n";**

**printBoard(queens, n);**

**} else {**

**cout << "No solution exists with the first queen at this position.\n";**

**}**

**clock\_t end = clock();**

**double time\_taken = double(end - start) / CLOCKS\_PER\_SEC;**

**cout << "Execution time: " << time\_taken << " seconds" << endl;**

**return 0;**

**}**

**Output:**

**A computer screen shot of a black screen

AI-generated content may be incorrect.**