33. N Queens

#include <stdio.h>

#include <stdbool.h>

#define N 4 // Change this to the desired number of queens

bool isSafe(int row, int col, int chessboard[][N]) {

// Check for queens in the same column above

for (int i = 0; i < row; i++) {

if (chessboard[i][col] == 1) {

return false;

}

}

// Check for diagonals (both left and right)

for (int i = row, j = col; i >= 0 && j >= 0; i--, j--) {

if (chessboard[i][j] == 1) {

return false;

}

}

for (int i = row, j = col; i >= 0 && j < N; i--, j++) {

if (chessboard[i][j] == 1) {

return false;

}

}

return true; // No conflicts found

}

void printSolution(int chessboard[][N]) {

printf("\nSolution:\n");

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

printf(" %d ", chessboard[i][j]);

}

printf("\n");

}

}

bool solveNQueensUtil(int chessboard[][N], int col) {

// Base case: If all queens are placed (all columns are filled)

if (col >= N) {

return true;

}

// Try placing the queen in all possible rows of the current column

for (int i = 0; i < N; i++) {

// Check if placing a queen in this row is safe

if (isSafe(i, col, chessboard)) {

chessboard[i][col] = 1; // Place the queen

// Recursively proceed to place the remaining queens

if (solveNQueensUtil(chessboard, col + 1)) {

return true;

}

// Backtrack if no solution is found in this branch

chessboard[i][col] = 0; // Remove the queen

}

}

// If queen placement is not possible in any row of the current column, then return false

return false;

}

void solveNQueens() {

int chessboard[N][N] = {0};

if (solveNQueensUtil(chessboard, 0)) {

printSolution(chessboard);

} else {

printf("Solution does not exist\n");

}

}

int main() {

solveNQueens();

return 0;

}