Sudoku Solver Report

George Ghiugan

March 10, 2024

1 Project Description

The following C program is a Sudoku Puzzle Solver designed to solve and generate 9x9 Sudoku puzzles using a backtracking algorithm. Given any Sudoku puzzle by the user, the program uses a recursive backtracking approach to populate the grid and solve the puzzle if a solution exists. If there is no solution to a given puzzle, the program will print a "No solution exists" to notify the user. This program allows for a quick way to check the correct answer to a Sudoku Puzzle.

Functions

2 void printGrid(int grid[N][N]):

- The `printGrid` function displays the 9x9 Sudoku grid on the console.
- Prints a vertical bar after every 3 numbers.
- Prints a dashed line across the numbers after every three columns.

3 int isValid(int grid[N][N], int row, int col, int num):

- The `isValid` function checks if it is valid to place a number in the Sudoku Grid. Returns the value 1 if it is valid and 0 if it is not valid. This function is called in the `solveSudoku` function in order to check the validity of numbers across a row or column as part of the backtracking algorithm.

- Check if the number 'num' already exists in the current row or column:

```
for (int x = 0; x < N; x++) {
    if (grid[row][x] == num || grid[x][col] == num) {
        return 0; // Not valid
    }
}
```

- Determine the starting position of the 3x3 subgrid to which the cell belongs.

```
int startRow = row - row % 3;
int startCol = col - col % 3;
```

- Check if 'num' already exists in the 3x3 subgrid:

```
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 3; j++) {
        if (grid[i + startRow][j + startCol] == num) {
            return 0; // Not valid if 'num' is already present in subgrid
        }
    }
}

    return 1;
```

- Function returns 1 if valid (safe) and 0 if it is not valid (unsafe).

4 int solveSudoku(int grid[N][N]):

- The `solveSudoku` function uses Recursive backtracking in order to solve the Sudoku puzzle. Function returns 1 if the Sudoku has been successfully solved and 0 if no solution is found.
- `count++;` initialized at the top in order to count the number of iterations used to solve the Sudoku Grid.
- Iterating through each cell in the Sudoku Grid and attempting to place numbers. Calls `isValid` function to validate the number and continues algorithm until cells are populated:

```
for (int row = 0; row < N; row++) {
    for (int col = 0; col < N; col++) {
        // Check if the cell is empty
        if (grid[row][col] == 0) {
            // Try placing numbers 1 to 9 in the empty cell
            for (int num = 1; num <= 9; num++) {
                // Check if the number placed is valid
                if (isValid(grid, row, col, num)) {
                    // Place the number in the cell
                    grid[row][col] = num;

                    // Recursively call solveSudoku for the next cell
                    if (solveSudoku(grid)) {
                        return 1;
                    }

                    // Backtrack if no solution found
                    grid[row][col] = 0;
                }
            }

            // If no valid number can be placed, backtrack to the
               previous cell
            return 0;
        }
    }
    }
```

5 Code Implementation:

```
#include <stdio.h>

// Declare variables for size of the grid and a counter
#define N 9
int count = 0;

// Function prints the 9x9 Sudoku grid on the console
void printGrid(int grid[N][N]) {
    for (int row = 0; row < N; row++) {
        for (int col = 0; col < N; col++) {
            printf("%d", grid[row][col]);
            if (col % 3 == 2 && col != N - 1) {
                printf(" | ");
            } else if (col == N - 1) {
                printf(" |");  // Print vertical bar at the end of each row
            } else {
                printf(" ");
            }
        }
        printf("\n");
        if (row != N  && (row + 1) % 3 == 0) {
            printf("----------------------\n");
        }
    }
}

// Function checks if it is valid to place a number in the Sudoku Grid
// Returns 1 if valid (safe) and 0 if it is not valid (unsafe)
int isValid(int grid[N][N], int row, int col, int num) {
    // Check if the number 'num' already exists in the current row or column
    for (int x = 0; x < N; x++) {
        if (grid[row][x] == num || grid[x][col] == num) {
            // Not valid if 'num' is already present in the row or column
            return 0;
        }
    }

    // Determine the starting position of 3x3 subgrid where the cell belongs
    int startRow = row - row % 3;
    int startCol = col - col % 3;

    // Check if 'num' already exists in the 3x3 subgrid
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            if (grid[i + startRow][j + startCol] == num) {
                // Not valid if 'num' is already present in the subgrid
                return 0;
            }
        }
    }

    return 1;  // 'num' is valid to be placed in the current cell
}

// Recursive function to solve the Sudoku puzzle using backtracking
// Returns 1 if a solution is found, 0 otherwise.
int solveSudoku(int grid[N][N]) {
    // Increment the iteration count
    count++;

    // Loop through each cell in the grid
    for (int row = 0; row < N; row++) {
        for (int col = 0; col < N; col++) {
            // Check if the cell is empty
            if (grid[row][col] == 0) {
                // Try placing numbers 1 to 9 in the empty cell
                for (int num = 1; num <= 9; num++) {
                    // Check if the number placed is valid
                    if (isValid(grid, row, col, num)) {
                        // Place the number in the cell
                        grid[row][col] = num;
                        // Recursively call solveSudoku for the next cell
                        if (solveSudoku(grid)) {
                            return 1;
                        }

                        // Backtrack if no solution found
                        grid[row][col] = 0;
                    }
                }

                // If no valid number can be placed, return 0
                return 0;
            }
        }
    }
    // Return 1 if the entire puzzle is solved
    return 1;
}

// Initializes the Sudoku Grid and prints messages based on solving outcome
int main() {

    // Hard code the grid to be solved by the program
    int grid[N][N] = {
        {0, 2, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 6, 0, 0, 0, 0, 3},
        {0, 7, 4, 0, 8, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 3, 0, 0, 2},
        {0, 8, 0, 0, 4, 0, 0, 1, 0},
        {6, 0, 0, 5, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 1, 0, 7, 8, 0},
        {5, 0, 0, 0, 0, 9, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 4, 0}
    };

    // Call printGrid function to display the Sudoku Grid on the console
    printf("The input Sudoku puzzle:\n");
    printGrid(grid);

    // Print the Sudoku Solution if found as well as number of iterations
    if (solveSudoku(grid)) {
        printf("\nSolution found after %d iterations:\n", count);
        printGrid(grid);
    }
    // Print Message if given Sudoku Puzzle cannot be solved
    else {
        printf("\nNo solution exists.\n");
    }

    return 0;
}
```