Sudoku Solver Report

Momina Hassan

March 10, 2024

Contents

1 What is Sudoku?

Sudoku is a widely-popular puzzle game where players fill a 9x9 grid with numbers from 1 to 9. There can be no repeats in any row, column, or 3x3 subgrid. This game is easy but requires logical thinking and pattern recognition, which is why its considered a great exercise for the mind.

2 What does `Sudoku_Solver` do?

The `Sudoku_Solver` is a program written in C that takes an unsolved Sudoku puzzle as input and produces a solved puzzle as output. It utilizes recursion and backtracking techniques to orderly place numbers in the grid (from top left to bottom right), ensuring that the solution is correct. Moreover, this program tracks the number of iterations required to solve the puzzle, giving the user insight into the computation complexity of their puzzle.

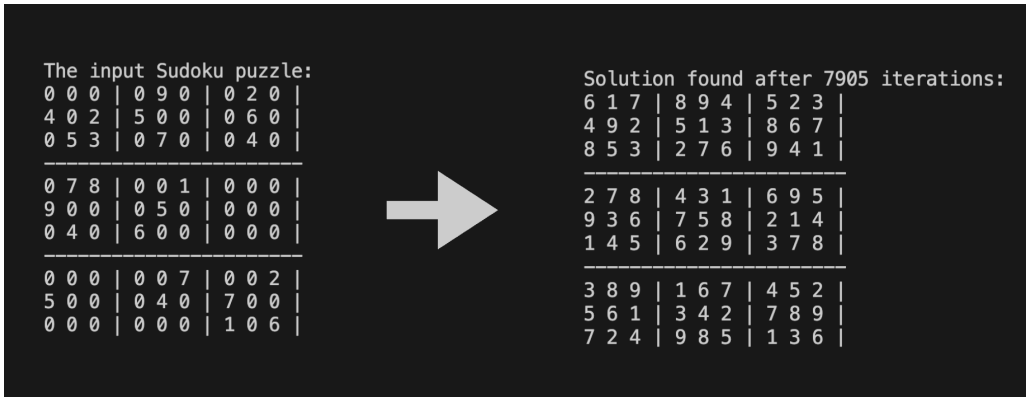

Figure 1: The programs takes the unsolved input and returns a solved puzzle

3 Global variables and Definitions

This code snippet defines key variables and constants used throughout the program. `N` is defined as 9 which represents the 9x9 Sudoku grid. `row` and `col` are initialized to `0` to represent the starting position. `count` is used to track the number of iterations used to produce a solution.

```
#define N 9
int row = 0;
int col = 0;
int count = 0;
```

4 `is_valid` function

This function ensures that the number passed through the parameter is valid and suitable for the designated position in the given row and column. The first loop iterates over the given row and column to ensure the number is not already present. The second loop assesses the 3x3 subgrid by initializing `startRow` and `startCol` indices based on given position. It then iterates through the subgrid ensuring the number is not already present. If neither of these conditions is violated, the function concludes that the number is appropriate for the given position and returns `true`.

```
bool is_valid(int grid[N][N], int row, int col, int num)
   {
    for (int i = 0; i < N; i++) {
        if (grid[row][i] == num || grid[i][col] == num) {
            return false;
        }
    }

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

    for(int i = 0; i < 3; i++) {
        for(int j = 0; j < 3; j++) {
            if(grid[i + startRow][j + startCol] == num) {
                return false;
```

```
            }
        }
    }
    return true;
}
```

5 `Solve_Sudoku` function

This function uses a recursive backtracking algorithm to explore possible solutions for the Sudoku puzzle until a valid solution is found or until all possibilities have been exhausted.

The initial base-case conditions checks if it has reached the last row and column (`row == N-1  col == N`). If so, it returns `true`, indicating a successful solution.

If the current column exceeds the grid's limits (`col == N`), it moves to the next row (`row++`).

When encountering a non-empty cell (`grid [row][col] != 0`), it proceeds to the next column.

If none of the above conditions are met, it means the current cell is empty (0). the function then iterates through numbers 1 to 9 and checks if each number is valid for the current position using the `is_valid` function. If valid, it sets the cell to that number (`grid[row][col] = num;`) and attempts to solve the puzzle recursively for the next column which returns true if successful. If not, it backtracks by resetting the cell value to 0 and continues iterating through the numbers.

```
bool solveSudoku(int grid[N][N], int row, int col) {
    count++;

    if(row == N-1 && col == N) {
        return true;
    }

    if(col == N) {
        col = 0;
        row++;
```

```
    }

    if(grid[row][col] != 0) {
        return solveSudoku(grid, row, col+1);
    }

    for(int num = 1; num < 10; num++) {
        if(is_valid(grid, row, col, num)) {
            grid[row][col] = num;
            if (solveSudoku(grid, row, col + 1)) {
                return true;
            }
            grid[row][col] = 0;
        }
    }
    return false;
}
```

6 `printGrid` function

This function prints the 9x9 Sudoku grid. It iterates over each element in the grid and prints its value. It also counts the rows and columns in order to insert vertical and horizontal lines to separate the 3x3 sub-grids. This enhances the grids readability for the user.

```
void printGrid(int grid[N][N]) {

    int counterRow = 0;

    for (int i = 0; i < N; i++) {
        int counterCol = 0;
        for (int j = 0; j < N; j++) {
            printf("%d", grid[i][j]);
            if (counterCol == 2 || counterCol == 5 ||
               counterCol == 8) {
                printf(" | ");
            } else {
                printf(" ");
            }
            counterCol++;
        }
```

```
            printf("\n");
            if (counterRow == 2 || counterRow == 5) {
                printf("-----------------------\n");
            }
            counterRow++;
        }
}
```

7 `main` function

This is the main function of the program. It initializes a 2D array (`int grid[N][N]`) and is given a hard coded value as the unsolved puzzle input. This function first prints the unsolved input puzzle in the terminal using `printGrid`. It then prints the solved puzzle by calling the `solveSudoku` function. If the function returns true, it prints the solved solution using `printGrid`. If returned false, it tells the user the solution does not exist.

```
int main () {
    int grid[N][N] = {
        {0, 0, 0, 0, 9, 0, 0, 2, 0},
        {4, 0, 2, 5, 0, 0, 0, 6, 0},
        {0, 5, 3, 0, 7, 0, 0, 4, 0},
        {0, 7, 8, 0, 0, 1, 0, 0, 0},
        {9, 0, 0, 0, 5, 0, 0, 0, 0},
        {0, 4, 0, 6, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 7, 0, 0, 2},
        {5, 0, 0, 0, 4, 0, 7, 0, 0},
        {0, 0, 0, 0, 0, 0, 1, 0, 6}};

    printf("The input Sudoku puzzle: \n");
    printGrid(grid);
    printf("\n");

    if(solveSudoku(grid, row, col)) {
        printf("Solution found after %d iterations: \n",
           count);
        printGrid(grid);

    } else {
        printf("No solution exists. \n");
```

```
    }

    return 0;
}
```

8 All combined code for `Sudoku_Solver.c`

```
#include<stdio.h>
#include<stdbool.h>

#define N 9
int row = 0;
int col = 0;
int count = 0;

bool is_valid(int grid[N][N], int row, int col, int num)
{
    for (int i = 0; i < N; i++) {
        if (grid[row][i] == num || grid[i][col] == num) {
            return false;
        }
    }
    int startRow = row - row % 3;
    int startCol = col - col % 3;

    for(int i = 0; i < 3; i++) {
        for(int j = 0; j < 3; j++) {
            if(grid[i + startRow][j + startCol] == num) {
                return false;
            }
        }
    }
    return true;
}

bool solveSudoku(int grid[N][N], int row, int col) {
    count++;
    if(row == N-1 && col == N) {
        return true;
    }

    if(col == N) {
```

```
        col = 0;
        row++;
    }

    if(grid[row][col] != 0) {
        return solveSudoku(grid, row, col+1);
    }

    for(int num = 1; num < 10; num++) {
        if(is_valid(grid, row, col, num)) {
            grid[row][col] = num;

            if (solveSudoku(grid, row, col + 1)) {
                return true;
            }
            grid[row][col] = 0; //backtrack
        }
    }
    return false;
}

void printGrid(int grid[N][N]) {

    int counterRow = 0;

    for (int i = 0; i < N; i++) {
        int counterCol = 0;
        for (int j = 0; j < N; j++) {
            printf("%d", grid[i][j]);
            if (counterCol == 2 || counterCol == 5 ||
               counterCol == 8) {
                printf(" | ");
            } else {
                printf(" ");
            }
            counterCol++;
        }

        printf("\n");
        if (counterRow == 2 || counterRow == 5) {
            printf("-----------------------\n");
        }
        counterRow++;
    }
```

```
}


int main () {
    int grid[N][N] = {
        {0, 0, 0, 0, 9, 0, 0, 2, 0},
        {4, 0, 2, 5, 0, 0, 0, 6, 0},
        {0, 5, 3, 0, 7, 0, 0, 4, 0},
        {0, 7, 8, 0, 0, 1, 0, 0, 0},
        {9, 0, 0, 0, 5, 0, 0, 0, 0},
        {0, 4, 0, 6, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 7, 0, 0, 2},
        {5, 0, 0, 0, 4, 0, 7, 0, 0},
        {0, 0, 0, 0, 0, 0, 1, 0, 6}};

    printf("The input Sudoku puzzle: \n");
    printGrid(grid);
    printf("\n");

    if(solveSudoku(grid, row, col)) {
        printf("Solution found after %d iterations: \n",
           count);
        printGrid(grid);
    } else {
        printf("No solution exists. \n");
    }

    return 0;
}
```