Pranay Rajesh Gheewala
B00826923

CSCI 3901

Date: 25th February 2019

Assignment - 4

Sudoku Problem

Overview

Sudoku is a grid puzzle game. It works on an n^2 x n^2 grid of cells. We place symbols in each cell of the grid so that no symbol appears twice in the same row, in the same column, or in any of the n x n mini-grids. The initial grid begins with some cells having symbols and the remaining cells being blank. The goal of the game is to fill in all cells.

I have created a program that can solve any size of board which will always be a square. As a solution to the above problem that takes initial input rows for the board. I have used backtracking algorithm for solving the sudoku problem which will assign characters one by one to the blank cells. The program checks whether the character exists in the current row, column and current 3*3 board before assigning the character into the cell and recursively check if the character leads to solution or not. If the number does not lead to solution then the program backtracks and checks for next character from the list of symbols entered by the user until a solution is found or if the solution is not found it returns “No solution found”.

Working

In this, program asks user to enter the size of the sudoku board for which user wants to find the solution. Next the user is asked to input the list of characters that will be used to solve the sudoku. The size of the list of characters should be equal to the size of the board. User cannot enter identical characters in the list of characters. Next the user is asked to enter the initial board configuration row by row where every blank value is entered by '.' character.

There are some input validations set in place to check the following:

- It checks whether the size of the list of characters is equal to the size of the board.
- It checks if initial configuration of the board contains values only from the list of characters and the '.' character.
- It checks if the length of input rows is equal to the size of the board and if by mistake user enter less or more characters than the size of the board the user is asked to re-enter that particular row.

Once the initial input rows are entered properly each character is entered into the 2-d array which will represent initial configuration of the board. Next, the function sudokuSolver () is called. The function will loop through the whole board and find the first occurrence of empty cell. As soon as the first empty cell is found it selects one character from the list of characters entered by the user and checks if it already exists in the current row by using existInRow() function, checks if it already current column by using existInCol() function and checks if it already current 3*3 grid by using existInBox() function. If the character is not present in row, column or box, it is assigned to the cell. The program recursively fills the whole board likewise. If this leads to the solution the program returns true. If it does not lead to a solution the program backtracks and assign the '.' character to the character which was assigned from the list of characters to the empty cell. The program will then check for the next element from the list of characters and this process will be continued till the solution is obtained from the sudokuSolver () method. If the method returns true than printBoard() method is called which prints the solution board configuration by traversing rows and columns of the board. The program can solve the sudoku for any board size but the board size will always be a square.

Files

The program does not take any external file to read nor does it store the output in any file. It takes input from the user through console and also displays the output on the console screen.

Algorithm and Design Elements

The program contains the following methods:

1) readInput() – Method to take size of the sudoku board, list of character that will be used in the board and the initial configuration of the game from the user and to check validations like whether the size of the list of character is equal to the size of the board
2) cleanInput() – Method to check validations like whether the size of input row is equal to the size of the board. Whether the characters of the input rows belong in the list of the character
3) sudokuSolver() – Method to solve sudoku recursively using the characters from the list of character and backtrack if the character is not the solution
4) isValid() – Method to check whether the character is present in the row, column, and 3*3 grid
5) existInRow() – Method to check whether the character is present in the row
6) existInCol() – Method to check whether the character is present in the column
7) existInBox() – Method to check whether the character is present in the 3*3 box

Pranay Rajesh Gheewala Date: 25th February 2019
B00826923 CSCI 3901

Assumptions

I have assumed that:

- The size of board given will always be a square.
- The number of symbols in list of characters of the input will match the size of the grid and will not have duplicate values.
- Symbols are case-sensitive.
- For blank values user will always enter '.' Symbol
- User will always provide initial input configuration and it cannot be all blank values

Limitations

There are few limitations of my program

- I have used backtracking algorithm which is less efficient and takes more time to solve the sudoku. The same problem can be solved using linklist which may be more efficient and faster
- As the board size increases time taken to solve the sudoku increases which can be improved using other algorithms

Reference

- https://www.geeksforgeeks.org/sudoku-backtracking-7/