SUDOKU SOLVER

Overview

This program takes in Sudoku puzzle, in the form of grid and attempts to solve by filling in empty locations of grid (represented by dot(.))

Files and external data

Users need to provide size of grid along with a set of allowed values.

Data structures and their relations to each other

Program basically uses Linked List to keep track of the location points of grid. This linked list object acts as key for set of possible values as Sorted Set, bonded together with Hash map. All set of possible values for whole grid is maintained in a Sorted Set data structure of 'Character' type.

The sectors are stored in form of arrays of 4-sized arrays. These arrays are defined according to the size of input matrix. These sub-arrays define the iteration required in terms of row and columns to focus a sector.

Assumptions

- Minimum Grid size must be 4.
- Grid size must be from following sequence : 4,16,25,..
- All elements of grid can be of length 1 only.
- Dot(.) will not be part of possible value set.
- Sequence of possible values inserted is of no significance.
- Grid size will be perfect square all the times.
- Program should allow more characters in rows (than puzzle size) while providing matrix input.

Key algorithms and design elements

Backtracking with constraint satisfaction has been the key implementation strategy for this program.

Program tries to find first empty location in grid and look for first possible element to stay in the location without any constraint/rule violation. Keeping the very element in the location, program tries to further predict pending locations in the grid. These pending locations are filled for every sector in the grid if there's only one possible element for the location. After there is no further room for prediction, the above steps are repeated until the recursive root returns true. If the deepest call doesn't fetch any success the undo method takes charge to undo the changes made across the whole process, in reverse order.

Limitations

- Efficiency and speed of program decreases with increase size of grid.
- A Lot of stack space is required to handle large number of variables during recursive call.
- Program finds only one solution and it stops looking further as soon as it finds first one.

Test Cases

1. Input:
- First input is character.
- Grid size is less than 4.
- Grid size is 4.
- Grid size is not perfect square.
- More values compare to the size of grid are provided for value set.
- Less values compared to the size of grid are provided for value set.
- Values provided for grid value set are repetitive.
- Matrix input is provided line by line.
- Matrix input is provided multiline.
- Matrix input is provided partially line by line and partially multiline.
- Row character length of matrix is smaller then the size of matrix.
- Matrix input with some extra characters than the size of each line are provided.
- Matrix input with some characters out of the possible value set.
2. Sudoku
- A classic 3x3 sudoku puzzle is provided as input with possible values from 1-9.
- A 3x3 puzzle is provided as input with possible values from character set.
- A diabolic sudoku puzzle grid is provided as input.
- A 16x16 puzzle is provided as input.
- A 25x25 puzzle is provided as input.
- A 36x36 puzzle is provided as input.
- A 16x16 non-solvable puzzle is provided as input.
- A 3x3 puzzle with values consisting of characters and integers is provided as input.

Citations and References

1. OpenCourseWare, M. (2019). *Puzzle 8: You Won't Want to Play Sudoku Again | Programming for the Puzzled | Electrical Engineering and Computer Science | MIT OpenCourseWare*. [online] Ocw.mit.edu. Available at: https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-s095-programming-for-the-puzzled-january-iap-2018/puzzle-8-you-wont-want-to-play-sudoku-again/ [Accessed 21 Feb. 2019].