211hw11

Homework 11: Sudoku Solver

Due: Friday 16 April 2010 at 9:59:59 am

The Sudoku Game

Sudoku is a 9*9 grid-based puzzle in which the goal is to place numbers from 1 to 9 in the grid squares taking into account specific constraints. The 9 * 9 puzzle grid can be seen as divided into 9 sub-grids of size 3*3. The 3 main conditions in the classic version of Sudoku state are that each square in the grid contains a number from 1 to 9 and a number cannot be repeated:

- In any line
- In any column
- In any on the 3*3 sub-puzzles

For more information of the game of Sudoku, visit: http://en.wikipedia.org/wiki/Sudoku

A Sudoku puzzle as needed for this assignment may not be well formed and in general may have no solutions or multiple solutions.

The assignment

The purpose of this assignment is to build a Java solver that finds all of the different solutions of the game using a constraint-based approach.

In a constraint-based approach, each square is viewed as a variable that can take on multiple values from a given set. A partial (intermediate) solution is one in which at least one variable has more than one possible value. A final solution is one which each variable has exactly one value (all the sets are singletons).

The required solution (built on the basis of the support code provided) should work as follows. At each step, the program will evaluate a partial solution. Initially, this partial solution corresponds to the input Sudoku puzzle (a 9*9 matrix that is partially filled) in which the values of some squares are known and others are unknown.

To find all solutions for a partial solution, we have to keep track of all the possible values for each square in the grid. (In the support class, the Sudoku grid is modeled by an array of Rows. The Row class contains an ArrayList of HashSets, one HashSet of Integers for each square from that row. The HashSet contains possible values for the square.

To find final solutions from a partial solution, one should look for a square with multiple possible values, generate the different partial solutions by specifying that the square contains one of the values (and impose the restrictions specified by the rules of Sudoku on the content of the other squares), and then recurse for each of these possible solutions until no square has more than one possible value. If square becomes empty, then it will not lead to a final solution. If all squares have exactly one value, then we have a final solution.

The support code provided contains a DrJava project and a few simple Junit tests. Download the code from here.

Your assignment is to implement:

- the method *findSolution* in the *PartialSolution* class.
- at least 5 more tests (3 for findSolution, 1 for *isDeadEnd*, 1 for *isFinal*).

Extra credit

For extra credit, improve the implementation of setElement and/or findSolution as you see fit, in order to improve the performance of the algorithms by examining fewer PartialSolution's. The improvement will be quantified by seeing if there is a decrease in the count of partial solutions in order to solve a puzzle (the count of partial solutions is returned by the *getIntermediateSolutions()* function of the *PartialSolution* class).

Restriction: Keep using the setElement method in findSolution and do not change the contract of the setElement or findSolution methods.

Submission

Submit via Owlspace a .zip file containing all the files from the support code including those that you modified. Don't forget to add as header to the *PartialSolution* class, your names and ids.