Assignment #2: Sudoku (Constraint Satisfaction Problems)

CSCI 364 Fall 2021 Oberlin College

Due: Monday November 15 at 11:59 PM

Background

In class, we used Sudoku puzzles as an illustrative example of Constraint Satisfaction Problems (CSPs) in AI. In a CSP, the goal is to find a complete, consistent assignment of values to a set of variables X (taken from their domains D) satisfying a set of constraints C that limit the valid combinations of variable values. In this assignment, you will have an opportunity to develop a program using CSP solution techniques to solve Sudoku puzzles.

As a reminder, a Sudoku puzzle is a 9x9 grid (81 variables) where each cell in the grid can take on the integer values 1-9 (the domain of each variable). A solution to a Sudoku puzzle is an assignment of values for each cell in the grid such that no two cells in the same row, column, or 3x3 square have the same value.

For example, for an initial configuration of a Sudoku puzzle, you might be given:

```
..3|.2.|6..
9..|3.5|..1
..1|8.6|4..
-----------
..8|1.2|9..
7..|...|..8
..6|7.8|2..
-----------
..2|6.9|5..
8..|2.3|..9
..5|.1.|3..
```

which has the solution:

```
483|921|657
967|345|821
251|876|493
-----------
548|132|976
729|564|138
136|798|245
-----------
372|689|514
814|253|769
695|417|382
```

Gitting Started

You can get started on the assignment by following this link:
https://classroom.github.com/a/H9jJ3Bg6

Assignment

Your assignment is to write a program in **Python or Java** that can take a set of Sudoku puzzles as input from a file, models each puzzle as a CSP, and outputs solutions to each puzzle. You should make use of both Constraint Propagation (i.e., arc consistency AC3) and Backtracking Search algorithms as part of your solution.

Unlike the previous homework assignment, there is no code base to start with. Instead, you are free to develop your program and represent your data structures however you wish. Also, **you are allowed to work in groups of two students** (only one needs to submit the solution on GitHub).

Your GitHub repository will only initially contain two files of Sudoku puzzles:

1. euler.txt, a set of Sudoku puzzles from Project Euler https://projecteuler.net/problem=96
2. magictour.txt, a more difficult set of Sudoku puzzles from http://magictour.free.fr/top95

Each file contains a multiple Sudoku puzzles (one per line), in the following format:

- Each line is a string of 81 characters, where characters in positions 0-8 correspond to the first row of the puzzle, characters in positions 9-17 correspond to the second row of the puzzle, etc.
- Known values are represented by the digits 1-9
- Initially unknown values are represented by a decimal point .

Your program should be able to read in these puzzles, solve them, then output the solutions in the same format (a string of 81 digits, followed by a newline character) in the same order they were read in from file.

Please call your program either SudokuSolver.py (in Python) or SudokuSolver.java (in Java), so that it is called with either:

```
python3 SudokuSolver.py <PuzzleFile>
```

or

```
java SudokuSolver <PuzzleFile>
```

You should not import any modules or libraries not already built into Python and Java, nor should you use any code from online sources.

Within your README.md file, you should include:

1. The solutions to all 145 puzzles in the same format as the input files (please put all of the euler.txt solutions under a header called "Euler" and the magictour.txt solutions under a header called "Magic Tour")
2. A couple paragraphs documenting how you designed and implemented the data structures used in your program – how did you represent the sudoku puzzle a CSP? What design options did you consider, and how did you decide on this implementation?
3. A couple paragraphs describing each of the algorithms you implemented. What do they do, and how do they do it? What choices did you make while implementing them?
4. A paragraph discussing the runtime efficiency of your program. Don't worry about calculating order notation, but you should describe how quickly your program operates on the puzzles. Did you notice certain trends? What did you do to improve the runtime of your implementation so that it could solve the puzzles quickly? Note: my unoptimized Python solution solves magictour.txt in 7 minutes, so yours shouldn't need an hour. Part of your grade will be based on the runtime of your program.
5. A short paragraph describing your experience during the assignment (what did you enjoy, what was difficult, etc.)
6. An estimation of how much time you spent on the assignment, and
7. An affirmation that you adhered to the honor code

Please remember to commit your solution to your repository on GitHub. You do not need to wait until you are done with the assignment; it is good practice to do so not only after each coding session, but maybe after hitting important milestones or solving bugs during a coding session. ***Make sure to document your code***, explaining how you implemented the CSP as a data structure, as well as the algorithms used to solve the CSP.

<u>Honor Code</u>

Each student is to complete this assignment with no more than one partner. However, students are encouraged to collaborate with one another to discuss the abstract design and processes of their implementations. For example, please feel free to discuss the pseudocode for the AC3 and Backtracking Search algorithms to help each other work through issues understanding exactly how the algorithms work. At the same, no code can be shared between groups, nor can groups look at each other's code.

And please be careful – there are many Sudoku solvers with freely available code online. This is a popular assignment and fun exercise for many programmers. It is easy to accidentally stumble upon someone else's solution, so please do not look at or copy anyone else's code (which I don't expect to be a problem!).

Grading Rubric

Your solution and README will be graded based on the following rubric:

Program flow: /10 points
Implementing Sudoku as a CSP: /15 points
Constraint Propagation implementation: /10 points
Backtracking Search implementation: /15 points
Program efficiency: /5 points
Correct solutions to the Euler puzzles: /15 points
Correct solutions to the Magic Tour puzzles: /10 points
Answers to README questions: /15 points
Appropriate code documentation: /5 points

By appropriate code documentation, I mean including a header comment at the top of each file explaining what the file provides, as well as at least one comment per function explaining the purpose of the function.