Artificial Intelligence Spring 2018

Homework 4: CSPs

Due Date: Monday, April 30 at 11:59pm

ACADEMIC HONESTY

As usual, the standard honor code and academic honesty policy applies. We run **automated checks for plagiarism** to ensure only original work is given credit. Submissions isomorphic to (1) those that exist anywhere online, (2) those submitted by classmates, or (3) those submitted by students in prior semesters, will be considered plagiarism.

SUBMISSION

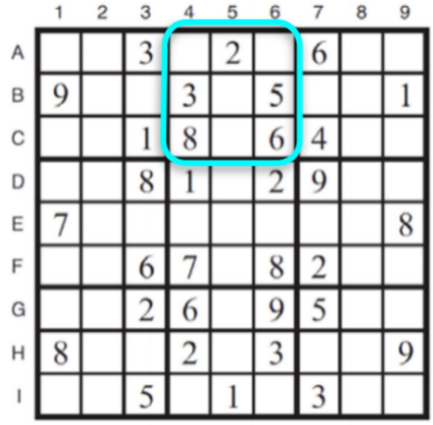
You will submit one zip file, named **hw4_myUNI.zip**, which contains files:

- written.pdf,
- driver.py or driver_3.py [Both are included in the starter code. Update the one you wish to use and **delete the other** before submitting]
- README.txt or .pdf

You can submit as many times as you like before the deadline. Only your most recent submission will be graded. The written submission can be handwritten and scanned (PDF scanning mobile apps typically work well), or typed, but it must be legible in order to receive credit.

WRITTEN

Consider the Sudoku puzzle below. Each variable is named by its row and its column (see Programming “Introduction” for an example). Recall each variable must be assigned a value from 1 to 9 subject to the constraint that no two cells in the same row, column, or box may contain the same value.



	1	2	3	4	5	6	7	8	9
A			3		2		6		
B	9			3		5			1
C			1	8		6	4		
D			8	1		2	9		
E	7								8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1		3		

1. List the names of the variables in the blue circle and their corresponding initial value domains.

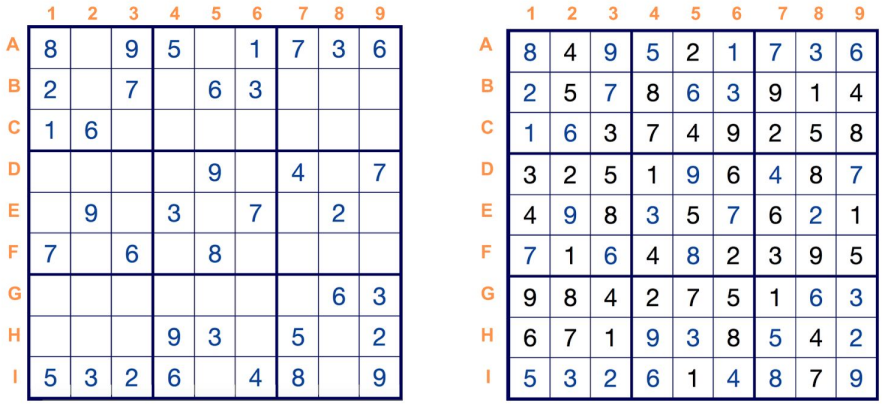
2. Reduce the domain for the four unassigned variables in question 1 by enforcing arc constraints using the entire puzzle board. List the new domains.

3. Assume we have to choose one of the four unassigned variables in question 2 to explore further. Using the minimum remaining value heuristic, which variable or variables should we explore next?

4. Assume we choose A4 to explore next and assume that A6, B5, C5 are the last three remaining variables waiting to be assigned. Using least constraining value rule, which value of A4 should be tried first?

PROGRAMMING

I. Introduction



	1	2	3	4	5	6	7	8	9
A	8		9	5		1	7	3	6
B	2		7		6	3			
C	1	6							
D					9		4		7
E		9		3		7		2	
F	7		6		8				
G								6	3
H				9	3		5		2
I	5	3	2	6		4	8		9

	1	2	3	4	5	6	7	8	9
A	8	4	9	5	2	1	7	3	6
B	2	5	7	8	6	3	9	1	4
C	1	6	3	7	4	9	2	5	8
D	3	2	5	1	9	6	4	8	7
E	4	9	8	3	5	7	6	2	1
F	7	1	6	4	8	2	3	9	5
G	9	8	4	2	7	5	1	6	3
H	6	7	1	9	3	8	5	4	2
I	5	3	2	6	1	4	8	7	9

The objective of Sudoku is to fill a 9x9 grid with the numbers 1-9 so that each column, row, and 3x3 sub-grid (or box) contains one of each digit. You may try out the game here: **sudoku.com**. Sudoku has 81 **variables**, i.e. 81 tiles. The variables are named by **row** and **column**, and are **valued** from 1 to 9 subject to the constraints that no two cells in the same row, column, or box may be the same.

Frame your problem in terms of **variables**, **domains**, and **constraints.** We suggest representing a Sudoku board with a Python dictionary, where each key is a variable name based on location, and value of the tile placed there. Using variable names **A1**… **A9… I1… I9,** the board above has:

- *sudoku_dict*["**B1**"] = **9**, and
- *sudoku_dict*["**E9**"] = **8**.

We give value **zero** to a tile that has not yet been filled.

II. What To Submit

Your program will be executed as follows:

$ python driver.py <input_string>

In the starter zip, sudokus_start.txt, contains hundreds of sample unsolved Sudoku boards, and sudokus_finish.txt the corresponding solutions. Each board is represented as a single line of text, starting from the top-left corner of the board, and listed left-to-right, top-to-bottom.

The first board in sudokus_start.txt is represented as the string:
003020600900305001001806400008102900700000008006708200002609500800203009005010300

Which is equivalent to:

0 0 3 0 2 0 6 0 0
9 0 0 3 0 5 0 0 1
0 0 1 8 0 6 4 0 0
0 0 8 1 0 2 9 0 0
7 0 0 0 0 0 0 0 8
0 0 6 7 0 8 2 0 0
0 0 2 6 0 9 5 0 0
8 0 0 2 0 3 0 0 9
0 0 5 0 1 0 3 0 0

Your program will generate output.txt, containing a single line of text representing the finished Sudoku board. E.g.:
483921657967345821251876493548132976729564138136798245372689514814253769695417382

Test your program using sudokus_finish.txt, which contains the solved versions of all of the same puzzles.

Besides your driver (and any other python code dependency), submit a README.txt with your results and observations, including the:

- **number AND line numbers of boards you could solve from sudokus_start.txt,**
- **running time, and**
- **any other relevant information.**

III. Backtracking Algorithm

Implement **backtracking** search using the **minimum remaining value** heuristic. Pick your own order of values to try for each variable, and apply **forward checking** to reduce variables domains.

- Test your program on sudokus_start.txt.
- Report the puzzles you can solve now, running time, observations.

IV. Important Information

1. Test-Run Your Code

Test, test, test. Make sure you produce an output file with the **exact format** of the example given.

2. Grading Submissions

We test your final program on **20 boards**. Each board is worth **5 points** if solved, and zero otherwise. These boards are similar to those in your starter zip, so if you solve all those, you'll get full credit.

3. Time Limit

No brute-force! Your program should solve puzzles in **well under a minute** per board. Programs with much longer running times will be killed.

4. Just for fun

Try your code on the world's hardest Sudokus! There's nothing to submit here, just for fun. For example:

Sudoku:
800000000003600000070090200050007000000045700000100030001000068008500010090000400

Solution:
812753649943682175675491283154237896369845721287169534521974368438526917796318452