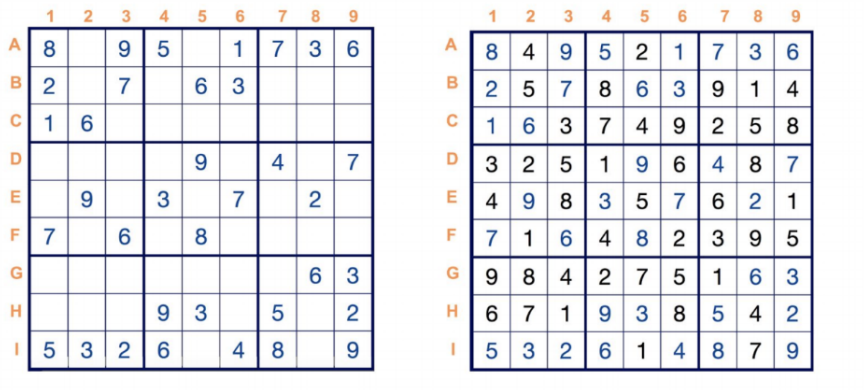
COMS 4701 Artificial Intelligence

Homework 2: Coding - CSPs

Due Date: October 24, 2025

Please read all the sections carefully:

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"] = **2**, and
- sudoku_dict["E2"] = **9**.

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

Executing your program

Your program will be executed as follows:

```
$ python3 sudoku.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. If you place `sudokus_start.txt` and `sudokus_finish.txt` in the same folder as `sudoku.py` the following command will apply your algorithm to all boards and produce a test summary.

```
$ python3 sudoku_tester.py
```

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. All puzzles we provided contained unique solutions.

II. 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` and `sudokus_finish.txt`.
- Report the number of puzzles you can solve and the mean, standard deviation, min, and max of the runtime over all puzzles in `README.txt`.

III. 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

IV. What You Need To Submit

1. Your `sudoku.py` file (and any other python code dependency)
2. A `README.txt` with your results, including the:
   - number of boards you could solve from `sudokus_start.txt`,
   - running time statistics: min, max, mean, and standard deviation.
   - Note: we provided a sample script sudoku_tester.py that you can use as a template when generating your README.txt.

V. Autograder Specifications & Performance

Please keep in mind that your code will be graded on a system with the following specifications:

- **CPU:** 4 Cores
- **RAM:** 6 GB

If your personal development machine is more powerful, be aware that code that runs quickly for you might exceed time or memory limits on the grading environment. We recommend you write efficient code and test it accordingly, keeping these hardware limitations in mind. For example, while your local machine might handle a memory-intensive approach, it could fail on the autograder.

VI. Before You Submit

- Ensure that your files are named `sudoku.py` and `README.txt`
- Ensure that your file compiles and runs.
- After your submission on Gradescope you will receive feedback in 5 minutes on whether your code has the proper filename, output format and execution time. However, if your code is inefficient or takes longer to run, feedback can take up to 25 minutes. Please address any issue and resubmit before the deadline.
- **Make sure** your code passes the **PRE-GRADER check** after you submit on gradescope. This step only verifies that your code:
  - Executes successfully without crashing
  - Produces an output.txt file in the correct format
  - Runs within the short time limit (under around 5 minutes per test case)

  It does not test whether your algorithms are correct.