Assignment 3

Due Date: Oct 22th 2015

Sudoku is a number placement puzzle. In this puzzle you are given an N x N grid of cells. The grid itself is composed of M x K sub-grids. You can place a single digit, drawn from 1 to N, in any cell. Initially the grid will have some of its cells partially filled. The objective of the puzzle is to complete the grid so that:

1. Every cell contains a digit.
2. No digit appears twice in any row, column of the N x N grid or in any row, column of any of the M x K sub-grid.

Figure 2 (a) below is a 12 x 12 Sudoku puzzle made up of 3 x 4 sub-grids and Figure 2 (b) is the solution to this puzzle.

	11								4		
7			2	6			3	5		11	
	6	9		1	12			7		10	
	4	1					10	8		6	
	8		9				12	10			
2				11		1			9		
		8			2		4				7
			3	5				12		4	
	7		4	12					6	8	
	5		8			10	7		11	1	
	1		11	3			5	2			6
		3								12	

Figure 2(a)

8	11	12	1	7	10	5	2	6	4	3	9
7	10	4	2	6	8	9	3	5	12	11	1
3	6	9	5	1	12	4	11	7	2	10	8
11	4	1	12	9	5	2	10	8	7	6	3
6	8	5	9	4	3	7	12	10	1	2	11
2	3	7	10	11	6	1	8	4	9	5	12
5	12	8	6	10	2	11	4	1	3	9	7
1	9	11	3	5	7	8	6	12	10	4	2
10	7	2	4	12	1	3	9	11	6	8	5
12	5	6	8	2	9	10	7	3	11	1	4
9	1	10	11	3	4	12	5	2	8	7	6
4	2	3	7	8	11	6	1	9	5	12	10

Figure 2(b)

Your assignment is to write a program to solve the general Sudoku puzzle using finite domain CSP methods discussed in the class.

The input to your program will be a file formatted as follows:

1. The first line will have three integers separated by ‘,’ and ending in ‘;’ that fix the values for N, M and K in that order.
2. The next N lines describe the board configuration – one per row ended by ‘;’. An empty cell will be denoted by ‘-‘.

See game.txt for the input details

The output of the above puzzle would look like this:

Backtracking:
Execution Time: 5.00000000001e-06
Consistency Checks: 1234
Solution: [[8,11,12,1,7,10,5,2,6,4,3,9], [7,10,4,2,6,8,9,3,5,12,11,1], ….. [4,2,3,7,8,11,6,1,9,5,12,10]]

NOTE: this is just an example, you'll get actual trace to match your output in due time.

You should implement:

1. Backtracking
2. Backtracking + MRV heuristic
3. Backtracking + MRV + Forward Checking
4. Backtracking + MRV + Constraint Propagation
5. Min-conflicts Heuristic

Given an input instance you will generate an output for each of the 5 implementations. Each output will be a solved puzzle. Output should describe the solved puzzle and will have list of size N, with each element is a list of size N. The second term in the output should be the number of consistency checks done to arrive at the solution.

We will provide you with some test data and will test your output on some tricky data at the end just like HW1. Based on these files analyze the pros and cons of these 5 implementations.

Notes:

1. Your program should be written in Python.
2. You can work in pairs of two
3. There are numerous sources on the Web for the Sudoku puzzle. You are encouraged to consult them to gain a good understanding of the puzzle.
4. Division of marks is just like HW1.

Code:

1. config.py: Fill it just like hw1
2. game.txt: will contain the game state.
3. csp.py: All your code should go here, you can do whatever you need in this file as long as you are returning right solution to Sudoku.py
4. sudoku.py: You don't need to change this file.

Submission:
On or before due date you should email to the TA a zip file containing:

- Source code with comments.
- Report saying which method is better or in what condition we should use what among the 5-implemented methods

You have until midnight of the due date to submit the zip file through blackboard (No-email). You should sign up for a demo with the TA. On the demo date you will be given your source files that you should compile and demo to the TA. Your program will be tested on different instances of the puzzle.