CS 4633/6633 Programming Assignment 2: Sudoku

September 27, 2016

This programming assignment is due on Friday, October 21, 2016, at 11:59pm.

Honor code note: You are welcome to discuss the assignment with other students, but you must individually and independently write the code and other deliverables that you submit for the assignment. Any copying of code from outside sources or other students will be considered an honor code violation.

This assignment involves solving Sudoku puzzles using a Constraint Satisfaction approach. You should first download the `PA2.zip` file from Blackboard and unzip in on your computer. The provided code has been tested with Python 2.7, but should work with minor changes on other versions of Python.

The folder you are provided contains:

- `PA2_Handout.pdf` - This file that you are reading
- `puzzles.txt` is the provided puzzle file.
- `pa2.py` - This is the main file that you will be editing. To run the code you should type:

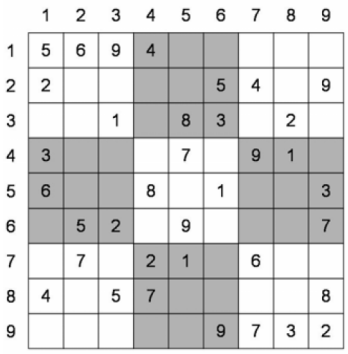
  ```
  python pa2.py puzzles.txt
  ```

  from the command line.

Implementation [80 points]

This assignment requires you to implement portions of a backtracking search that will solve Sudoku puzzles by treating them as a Constraint Satisfaction Problem. A Sudoku puzzle is solved when a digit (1-9) has been assigned to each cell of the puzzle in such a way that no number appears twice in any row, column, or 3 x 3 grid of the puzzle.

You have been provided starter code with some basic functionality, including a definition of a Sudoku puzzle class (`Sudoku`), as well as a definition of a variable corresponding to a single cell of the puzzle (`Cell`) . We will be using the problem formulation that was first presented in the homework problem about CSPs and Sudoku, where we have 81 variables, and each variable has a domain of the



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

Figure 1: A specific instance of a Sudoku game

digits 1-9. The constraints state that variables on the same row, column, and in the same grid cell cannot have the same digit.

Each puzzle is represented as a single line in the `puzzle.txt` file. The starting value of each cell is given in a left-to-right and top-to-bottom traversal of the puzzle's cells. If a starting value is given, then it is represented by the corresponding digit (1-9). If a starting value for a cell is blank, that is represented by a period (.) in the puzzle string. The `to_string()` function in the provided code will return a string of a puzzle in this format.

As an example, the string for the example puzzle shown in Figure 1 would be:

```
5694.....2....54.9..1.83.2.3...7.91.6..8.1..3.52.9...7.7.21.6..4.57....8.....9732
```

In this task, you will be required to implement various functions and determine how well the algorithm works with them implemented. Starter code has been provided on Blackboard. Familiarize yourself with all of the code and comments so that you will be able to complete the assignment. There are helper functions provided for converting from a (row, column) location to a (grid, cell) location in the puzzle. Look for `get_row_column()` and `get_grid_cell()` in the code.

- **Task 1 [1 point]** Modify the `student_name()` function to return your name. This will be used for automated verification that your code works. Look for the place in the code marked "TASK 1 CODE HERE".
- **Task 2 [20 points]** Implement the `forward_check()` function within the `Sudoku` class. This function has two modes, depending on the input `mode` variable.
  - `mode == 'remove'` – In this mode the function performs as normal forward checking. It removes conflicting values from other variables in the same row, column, and grid cell as the input location and value. It returns `False` if any domain is empty as a result of this forward checking, and `True` otherwise.
  - `mode == 'count'` – In this mode no values are actually removed from any variable's domains, but instead the number that would be

removed by forward checking is counted and returned. This is used for the least-constraining value heuristic calculation.

These two modes are provided in one function since the majority of the code is the same for each case. Look for the place in the code marked "TASK 2 CODE HERE".

- **Task 3 [12 points]** Implement the `count_constraints()` function. Given a row and column location, this function should count the number of unassigned variables that are connected to the given location via a constraint. To do this it should look along the row, column, and in the grid for the given location. If should return the number of unassigned variables involved. If a variable hasn't been assigned, then its value should be `None`. This is used to compute the degree heuristic for variable selection. Look for the place in the code marked "TASK 3 CODE HERE".

- **Task 4 [10 points]** Implement the `mrv()` function. (mrv stands for minimum remaining values) Given a puzzle, and a list of unassigned row, column location tuples (format = (row, column)), this function should return a list of those (row, column) tuples from the input list (`unassigned`) that have the minimum remaining values. If there is a single variable with the minimum number of remaining values this should return a list with only that location tuple in it. If more than one location are tied with the same (minimum) number of remaining values then the function should return a list with those location tuples that are tied. This function computes the minimum remaining value heuristic for variable selection. Look for the place in the code marked "TASK 4 CODE HERE".

- **Task 5 [15 points]** Implement the `order_values()` function. Given a row and column location, this function should return a list of the values from that variable's domain, ordered by the least-constraining value heuristic. So, the first value in the list should be the least constraining, and the last value the most constraining. Ties can be broken arbitrarily. This function computes the least-constraining value heuristic for value ordering. Look for the place in the code marked "TASK 5 CODE HERE".

- **Task 6 [22 points]** Implement the main `backtracking_search()` function. This function is given an input puzzle (that is partially filled in) and it will proceed to solve it as follows. If the puzzle is not already solved (the base case of the recursion) then it will select an unassigned variable, order the values for that variable, and proceed to try assigning those values to the variable. For each value, a new copy of the puzzle will be made, the variable value assignment made on this new puzzle, and then forward-checking will be performed to detect any problems with this assignment. If no problems are found, then the function recurses on the new (more complete) puzzle. This function should return either `None` (no solution was found starting from the puzzle that was input) or the solved puzzle object, if it was discovered. Pseudo-code in comments is provided to help you structure this function.

  Look for the place in the code marked "TASK 6 CODE HERE".

With all the functions implemented, your code should be able to quickly solve Sudoku puzzles. If it does not, then you must track down the errors in your code and fix them. To get credit for your code, you must turn in your functional code with the required parts implemented or changed.

Rename your file `pa2_msuid.py` where `msuid` is your Mississippi State id string (i.e. abc123, formed from your initials and a number)

Make sure your works correctly by testing it on the provided puzzles and verifying that solutions are in fact correct. Your code's correctness will be tested automatically on new puzzles. Code will also be visually inspected. This is how the 80 points will be awarded, so make sure your code works and is readable. Make sure your re-named code file is turned in.

Write-up [20 points]

You are required to hand in the following for this assignment:

1. [**15 points**] Your evaluation of the various heuristics used in the search. For minimum-remaining values and the degree heuristic, modify the code to not use the heuristic and report the how quickly the code can solve puzzles. Do the same for the least-constraining value heuristic. See how long it takes to solve a single puzzle without any heuristics. Additionally, run the code without the forward checking, and again note its performance. (For each of these, you can kill the code if it hasn't produced a solution in a reasonable amount of time and report that fact. Don't wait for hours and hours to see if it finally solves something, unless you want to run it overnight for fun, but that is not required.) Specifically report the combination of all these heuristics and functions that solves puzzles the fastest in your implementation.

2. [**5 points**] At least a half-page write-up describing what you learned from this assignment.

Extra credit [20 points] Alternative CSP formulation

You will get up to 20 points extra credit if you create a new version of this program that uses the alternative representation of a Sudoku puzzle that was presented in the homework assignment. Specifically, the representation has the following variables:

- R_i^m is a variable denoting the column location of digit m in the i-th row of the puzzle. There is one such variable for each digit and row, so 81 of them.

- C_j^m is a variable denoting the row location of digit m in the j-th column of the puzzle. There is one such variable for each digit and column, so 81 of them.

- G_k^m is a variable denoting the cell location of digit m in the k-th grid. There is one such variable for each digit and grid, so 81 of them.

The constraints involving these variables include those that were the answer to the homework question.

Create a completely separate file and modify what you need to in order to solve puzzles using this representation.

To pass off the extra credit turn in the code you wrote for it as well and a short paragraph write-up of what you learned, specifically comparing and contrasting the code and performance against the representation used for the normal part of the programming assignment.

Make sure your code has the same interface to the `backtracking_search()` function and that your puzzle's `to_string()` functions correctly with your new variable representation so that your code's correctness can be automatically evaluated. Change the output of the `student_name()` function to return your name, plus the words "Extra Credit" so that it will show up as such in the results. Name your extra credit file the same as your original submission, but with the characters "EC" appended after your msu-id.