**Solving Sudoku with BFS and DFS**

**DUE: Friday, September 23 at 10:00pm (with a 1h55min grace period)**

*Please read the directions carefully. Make sure you understand what is being asked of you before you start on the assignment.*

*START WITH THE PROVIDED SAMPLE CODE in Github (most recent update 9-13-16)*

**THE BIG PICTURE**

The intent of this assignment is to give you a codebase on which to build for the major project, and to confirm your basic understanding of the primary search algorithms Breadth-First Search (BFS) and Depth-First Search (DFS). And if you haven't yet programmed in Python, It will help you acquire the necessary programming skills.

It is an INDIVIDUAL assignment. It must be completed for you to advance in the course. It has been my experience that if a student is unable to complete this assignment, then s/he struggles to keep up with the rest of the course. Please contact Dr. Larson if the deadline is looming and you cannot get it working - the teaching staff can help you!

You will be creating a Sudoku solver using the provided BFS framework. The search framework is modular in that it does not need to be modified\*. You create the problem class, then pass it to BFS. You will also write a DFS framework to solve the Sudoku puzzle. The problem class should also be modular in that it can be passed to the DFS framework without modification.

\*One optional modification to BFS is to use a generator, as suggested in the code comments. This is not required, nor are there any points associated with its implementation.

**DOING WELL**

Quality counts on this assignment. As indicated on the rubric, these are what we will be looking for:

* Good problem representation of the puzzle and actions. The sample puzzles are provided as nested lists, and you will receive full credit using this puzzle representation. However, if a different data structure makes more sense to you, you may use it BUT you have to read in the sample puzzle as provided and output in the same form so that we can use scripts for testing.
* Pruning techniques. At the very least, you should not pursue branches that have duplicates in either a column or row.
* Efficiency in time and space. This means relatively efficient running code and compact representation and code where possible. *Python is a language that lends itself to very compact, dense code. The downside is that it can be hard to read. If you write dense code, please comment it well.*
* Modularity. The problem class you defined should work with the provided BFS (without modification). The solver should work with any 4x4, 6x6, or 9x9 puzzle. We should be able to input any puzzle that is represented by a list of lists (one for each row) into your solver and produce a solution (provided there is a solution and we let it run long enough and don't run out of memory).
* Well organized and documented code. Use good naming conventions for variables and helper functions. If the intent of the code segment is not obvious, add comments to aid the reader..
* Directions were followed. Use requested directory, file, and function names. Provide the requested information. Submit code via GitHub and a summary on Moodle. Submit on time. Use the right version of Python.

**DELIVERABLES**

Starter code is located in the GitHub class repository under *Search*.

Use Python version 3.x (At the command prompt, "python" is v2.x, "python3" is v3.x).

Solver Function

* **Define a function *Solver( puzzle, r, c, search='bfs', printSolution=False )*** where *puzzle* is in the provided list representation and *r* and *c* are the row and column dimensions of the inner box. The default is to use the BFS function to solve the puzzle. The user can also input 'dfs' to instead use DFS, which can be implemented as a recursive function or an iterative one in which you maintain the stack.
* ***Solver()* should return the solution** in the same form as the input puzzle (i.e. a list of lists of rows). If a solution could not be found, return an empty list. If you were not successful in implementing the algorithm, return an empty list.
* **If print=True, *Solver()* should print out the solution** to the problem in a user-friendly form in which each row of the solution is presented on a new line. If no solution exists, then "No Solution" should be printed. If you were not successful in implementing the algorithm, then print out "Not Implemented".

Github Directory Structure (*If you are new to git and github, see Dr. Larson's tutorial pages).*

* Use the github account created specifically for this course, not your personal UMN account.
* Create a directory *Search*
* In *Search*, create a file *Solver.py* that contains the function definition for Solver(). Please make sure that you are importing all necessary modules within Solver.py (or in included files).
* The *Search* directory should also contain the BFS and DFS functions, the problem class, and any other required helper functions/scripts.
* When you are ready to submit, TAG your repository with "search"

Moodle Submission.

* Submit a document that provides the information requested below.
* Both this file and the code are due at the same time.
* Your assignment will not be graded unless the requested information is provided.

Please provide the following information:

* Identify any incomplete parts.
* Identify any known bugs that exist in the code.
* Describe (or provide an example) of a state in your implementation.
* Describe what was pruned (i.e. what constituted an illegal action).
* Describe any modifications that you made to BFS.

**ASSESSMENT**

|  |  |
| --- | --- |
| POINTS | ASSIGNMENT ELEMENT |
| 20 | Implementation Quality: Efficient. Well organized. Well documented. Logical. Good pruning |
| 35 | Implementation Complete for both BFS and DFS. |
| 20 | Implementation Correct: correctly solves sample puzzles, solves any puzzle of sizes specified, and indicates when puzzle unsolvable. This is for both BFS and DFS. |
| 15 | Framework: State properly represented. Problem class complete. Problem class functional with provided BFS. Input and output as requested. Problem class works with both BFS and DFS. |
| 5 | GitHub and Moodle submissions on time and complete. Code is as requested and runs without modification by the TA. Python 3.x is used. |
| 5 | Requested Information is complete |
| 100 | TOTAL POINTS |

TEST PUZZLES:



