Patrick Glowacki
CS_325_400_F2020: Analysis of Algorithms
HW8: Portfolio Project (Written)

Brief Description of Sudoku & My Implementation Using Python

Description of the Game

Sudoku is played on an $n^2 \times n^2$grid that is divided into $n \times n$ equal segments. Initially some cells in the grid contain a number between $1..n^2$. An action inserts a number between $1 \ldots n^2$ into an empty cell. The goal is to fill the grid with numbers such that every row, column, and segment contain the numbers $1 \ldots n^2$ without any repetitions.

My Implementation for Playing Sudoku

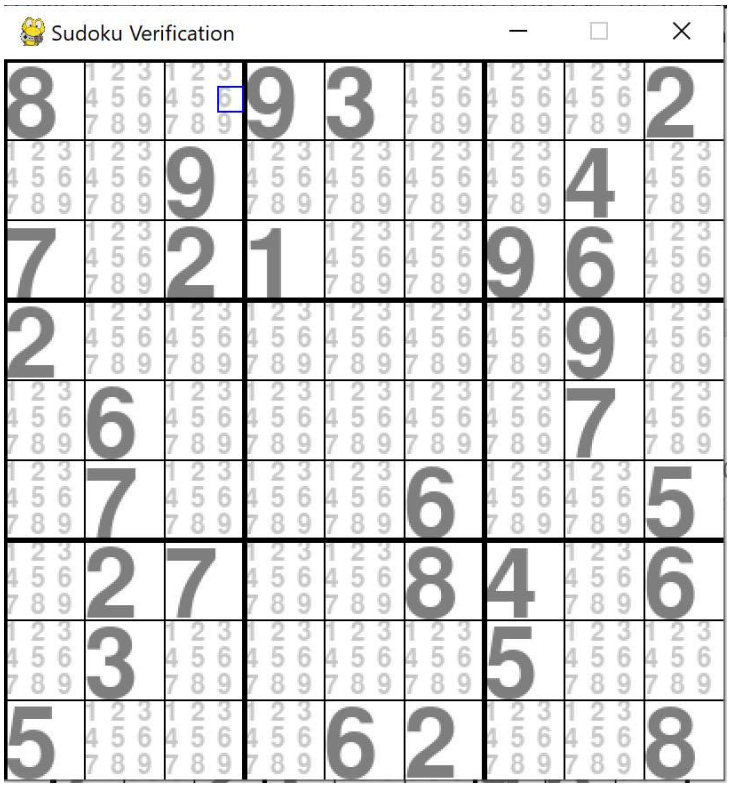
In order to play a game of sudoku, and subsequently verify the solution, my implementation uses Python to create a GUI for the game and then separately runs the verification algorithm. The GUI utilizes the Python module PyGame, which allows you to create fully featured games and multimedia programs in the python language (https://www.pygame.org/docs/).

The initial 9x9 grid, which is hard-coded into the program, is based on the sudoku board shown below in Figure 1.

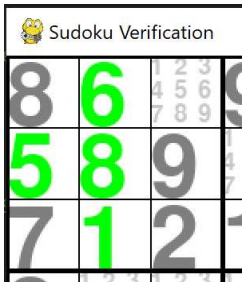
8			9	3				2
		9					4	
7		2	1			9	6	
2							9	
	6						7	
	7				6			5
	2	7			8	4		6
	3					5		
5				6	2			8

Figure 1: The sudoku board for sudoku.py which is hard-coded in for the user.

Once the program is started, which is explained in the README.md file, it will launch a new display window (shown in the image below). Note that for submission of the assignment, the program is hosted virtually via repl.it. The cells that have been hard-coded will show as large gray text. The user will then have to select the value for the remaining open cells. This is done by using the mouse to hover over the value they want to select. The number the mouse is hovering over will have a blue rectangle surrounding it if the mouse is in the range of the cell size.



Once a number is selected, it will then show as in large green text (shown below). Even after a value has been selected, it can still be changed by hovering over the area where the value would otherwise be. Regarding the implementation of the code itself, the sudoku grid uses a python dictionary in which the key is the x,y coordinates and the value is a list of integers. The grid is initialized so that every x,y coordinate has a list of integers [1,2,.., 9]. Once a number is selected, all the integers besides the selected value are replaced with a space (‘ ‘) character.



Once all cells complete (i.e., have a value selected), the PyGame module window will close and the verification algorithm will start. A message, shown below, will display to the terminal to signal that the verification has started and the outcome of the user’s solution.

```
You've entered all values. Verifying solution now...
Each column (x-coordinates) is verified: False
Each row (y-coordinates) is verified: False
Each 3x3 square (9 total) is verified: False
```

Patrick Glowacki
CS_325_400_F2020: Analysis of Algorithms
HW8: Portfolio Project (Written)

Verification Algorithm & Its Time Complexity

Mentioned at the start, a correct solution fills the grid with numbers such that every row, column, and segment contain the numbers $1..n^2$ without any repetitions. The verification algorithm is broken into 3 parts: checking each row, checking each column and checking each 3x3 square of the gird. The algorithm uses brute force to work through the

Each of the three parts has a nested for loop and because the innermost for loop must then go through an n-sized list (from the currentGrid dictionary), the time complexity of the verification algorithm is $O(n^3+n^3+n^3) = O(n^3)$. Thus, the verification algorithm runs in polynomial time.

NP-Complete

Given the following decision question:

> *Given a Sudoku instance, does it have any solutions?*

Using the 2-steps of the proof:

1. **Show that $\in$ $\boldsymbol{NP\ Complete}$**
   The section above just outlined that given a "certificate" – a completed sudoku board with a user's selections – we can verify the solution in polynomial time.
2. **Show that a problem known to be NP-Complete can be reduced to this decision question (i.e. Sudoku instance) and the transformation can be done in polynomial time.**
   For the second step, I will rely on the research of those much smarter. In the article *Reducing the generalised Sudoku problem to the Hamiltonian cycle problem*[1], Michael Haythorpe describes that a "generalized" sudoku instance can be reduced to the Hamiltonian Cycle Problem (HCP). HCP, of course, one of the classical NP-Complete problems. In his article Haythorpe also demonstrates that, "The resultant instance of HCP is a sparse graph of order $O(n^3)$."

Sources

1. Michael Haythorpe,
Reducing the generalised Sudoku problem to the Hamiltonian cycle problem,
AKCE International Journal of Graphs and Combinatorics,
Volume 13, Issue 3,
2016,
Pages 272-282,
ISSN 0972-8600,
https://doi.org/10.1016/j.akcej.2016.10.001.
(http://www.sciencedirect.com/science/article/pii/S097286001630038X)

Patrick Glowacki

README

```
# CS325_Portfolio
CS325 Portfolio Project

This README is for the following program:
    sudoku.py

This code is hosted virtually using repl.it at the following link:
https://repl.it/@pglow11/CS325PortfolioSudoku

    NOTE: Hosting using Repl.it does not provide the best interface or graphics
          for the pygame module. For example, major/minor grid lines, the mouse
          and other small graphics do not display well.

If you do not use the repl.it and would like to run it locally:
0. This program requires Python3 (perferably 3.6+). To ensure you have the
    proper Python environment set up, use the following link
    https://it.engineering.oregonstate.edu/setting-virtual-environments-python
1. Ensure pygame is installed by running the following command:
    $python -m pip install pygame==2.0.0.dev6
2. Run the program sudoku.py by entering the following command:
    $python sudoku.py
3. This will launch a pygame window titled "Sudoku Verification" with a 9x9
    sudoku board.
4. Complete the remaining sudoku cells by selecting one of 9 values for the
    cell with the mouse. As you move the mouse around, it will draw a blue box
    around the value that will be selected if the mouse button is pressed.
5. Once all sudoku cells are filled, the pygame window will close and the
    solution verification will begin. The following message will be displayed
    on the terminal.

    You've entered all values. Verifying solution now...

6. If the solution is correct, the following message will be displayed:

    Each column (x-coordinates) is verified: True
    Each row (y-coordinates) is verified: True
    Each 3x3 square (9 total) is verified: True

7. If any of the three verifications are False, then the solution is not
    correct.
```