Naphong Chadha 6380797

Project: Sudoku Solver

Building the Project

When I started building the project, I read up on the ideas of how to represent a sudoku board efficiently from the website https://norvig.com/sudoku.html. Basically a sudoku board consists of 81 squares each labeled from A1 all the way to I9 and the squares are linked to one another through two ways, 'Units' and 'Peers'. Units are the links for each row, each column, and each 3x3 box while peers are just every single square from the units excluding the current square itself.

After reading upon this I decided to create the project by making a scala file containing 2 different classes where my first class will represent the entire sudoku board which creates all the squares and links each one to its units and peers. Then my second class is to represent each individual square and label them as well as allow each one to know its units and peers and to contain a String data which will be the numbers inside the square at a time.

I decided on this style of implementing my sudoku board instead of just making multiple sets and hashmaps since I felt that this style of code would be more readable, extendable and most importantly that it would be easy to parallelize later on.

Solving the Puzzle

My goal after implementing a board was to be able to properly parse an input of characters to each square of the board and to be able to represent empty squares using a certain set of characters which makes some of my functions work much easier. I ended up following the website and representing empty cells or '.' char with a square containing all the possible numbers so we can search and remove them easily. Then next I implemented multiple eliminations techniques such as 'Elimination' , 'Lone Ranger' and 'Twins' that i got from: https://cse.buffalo.edu/faculty/miller/Courses/CSE633/Sankar-Spring-2014-CSE633.pdf.

After implementing all these strategies and making sure it can work in parallel, I moved on to the brute force method in which I had to try random values from unfilled squares and try to solve it from there one value at a time and one square at a time. And if anything ever breaks then I need to go back to a 'saved point' which has the previous values the board contains before I started the branch of the brute force method. So every square/value that fails I need to go back to a saved point and try a different square/value which can end up creating a bunch of possible branches before I end up with a solved board.

Difficulties I had:

1. Working with the parallel dependency and fixing data races and data types
2. Making the twin strategy work, this took way more time than it should have.
3. The biggest and most time consuming issue was the brute force method in which keeping a saved point to go back to when a value breaks at a certain point was really troublesome and I realized that it might be due to the way I represent my board and square.

Things I could not end up solving or implementing:

1. A case of an unsolvable sudoku board
2. Did not implement a random board generator
3. The brute force method could not be 100% parallelize due to some issues in my code