Solving Sudoku Using Java

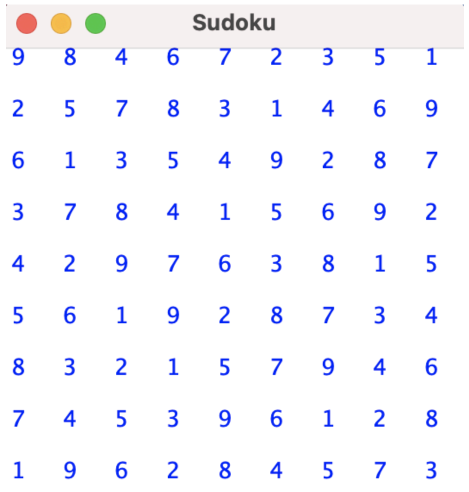
Abstract:

The real world problem that I explored in this project was solving Sudoku puzzles, which are popularly in the New York Times games. To better understand the problem in this project, I experimented with different ways to solve the puzzle and how to efficiently solve it and optimize the memory and time. The core CS concepts I used were depth first search (DFS), randomization, LinkedLists, and Stacks. My key findings involved the inverse relationship between locked values and the timings of the program when solving the puzzle.

Results:

I ran many versions of my code, with different implementations of the solve method, and changing the number of locked numbers in the puzzle. The metrics that were important in this code were the number of initial values used, the timeout rate, or the solve time. This is because I was able to see the relationship between the initial values and the solve time, or sometimes the timeout if the program wasn't able to fully solve the puzzle.

The outcome of my experiment was that the code did not fully work, and would only solve half of the puzzle and then stop no matter how many locked values there were. However, the number of locked values did affect the time it took for that half to solve.



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

Result 1: This is a picture of a fully solved puzzle, however this is from one of the board.txt's that I had the code read and solve, not one of its own creation.

Exploration 1: The relationship between the number of randomly selected initial values and the likelihood of finding a solution for the board are opposite. As the number of initial values increases, the probability of finding a solution decreases. If you run 5 boards for cases with 10, 20, 30, and 40 initial values, the probability of finding a solution decreases. If there are only 10 initial values, the 5 boards will probably have a solution since the puzzle is relatively easier to solve. If the number of initial values increases to 20, 30, and 40, the likelihood of finding a solution decreases. There are also more likely to be timeout issues.

Exploration 2: Yes, as the number of initial values increases, the time taken to solve the board also increases. This is because a higher number of initial values makes the puzzle more difficult and complex, which makes it necessary for a larger search space to find a solution, which also means that there is more effort for the program, leading to longer solving times.

Extensions:

I make a tester for the Sudoku file to test the methods in it, specifically findNextValue, findNextCell, and solve. I followed the format of the LinkedListTest file to create this one, and decided to test those methods because they are the most prevalent ones in this project. The outcome was that all of the tests were passed but I think this is an error because the program doesn't work correctly. The outcome can be replicated in the code-base through using tests in the actual Sudoku file and editing the already existing code.

Reflection:

1. Depth first search was better for this project instead of breadth first search because it has a similar structure to Sudoku since it looks at the path and looks at all the possible numbers in one path and then backtracks and goes to the next path, ensuring that there aren't any repeating numbers in any of the rows or columns. Breadth first search also uses much more memory and requires storing the numbers to remember what has already been used through all the searched spaces.
2. Unfortunately, the solve method is not working properly, so at this point I am unable to implement the random permute method. However, if I could, I would first implement the random permute method to permute the numbers 1-9 that should be in the rows and columns. Then, I would remove certain randomly chosen numbers from the board, and with each removal, I would implement a new method to check the number of possible solutions there are. I would make sure that not too many numbers are removed so that I could still create an optimal and challenging puzzle with one single solution.

Acknowledgements:

I worked with Kamalani on this project and I also went to TA hours. I also used ChatGPT on this project to help me fix my errors but it did not help much.