Bryan Arnold

CSE 2100

Extra Credit Assignment

12/9/16

Description:

The purpose of this program is to find the possible sequences of the unique letters in the puzzle FORTY + TEN + TEN = SIXTY, assign the letters numbers, and see if the number assignments are a solution to the puzzle. My program goes about this first by implementing a LinkedList data structure that keeps track of each unique letter as an element of each node. It then recursively iterates throughout the unique letters in the puzzle and creates new sequences of the unique letters. Once the empty sequence is the same size as the total number of unique letters, another method is called to check if the letter sequence is a solution to the puzzle by assigning each letter different numbers. In the solution method, it first replaces and gets rid of all spaces an +, - symbols from the puzzle. It then assigns each letter a different number from 0-9, then utilizes parseInt to find out if that sequence of letters assigned to those numbers make the solution true. If the solution is true, it displays the letter sequence, number sequence that was assigned, then the direct assignments that solve the puzzle.

Tradeoffs:

The first tradeoff I considered was what data structure or variable type to use when handling the unique sequences/sequence of unique letters. At first I thought using an ArrayList would work, we've used them a lot this semester and they're easy for me to understand conceptually. The problem I ran into was it was difficult to implement it for this specific assignment, mainly when checking for a solution since replacing the letters would be hard, and that the runtime it took when I eventually implemented it took far too long. It would sometimes take up to a minute to find the solution, so I decided to look for a different approach. I then decided to use LinkedList, a singly one to be exact, instead. They are like ArrayLists in understanding yet are more complex so implementation could be more difficult. When I implemented it, the runtime was much quicker than the ArrayList, even though it still takes some time to compute a solution it's not one minute. So, I ended up taking the LinkedList approach for the sake of runtime, since it only required barely anymore conceptualizing from ArrayList, and would only require some LinkedList code that I could take from another project we did (Happy numbers). The other tradeoff I thought about was the solution method and how to check if the puzzle was solved. I initially did severe hard coding for each letter and assigned it a number, but this may have contributed to the severe initial runtime. Instead, I came up with a more generic way of assigning numbers and checking for a solution. This in turn reduced runtime, but made the implementation more difficult since it was hard to make a near generic method for only one puzzle. So, in the end, the generic solution was harder to make, but ran faster than my old, hardcoded solution method.

Extensions:

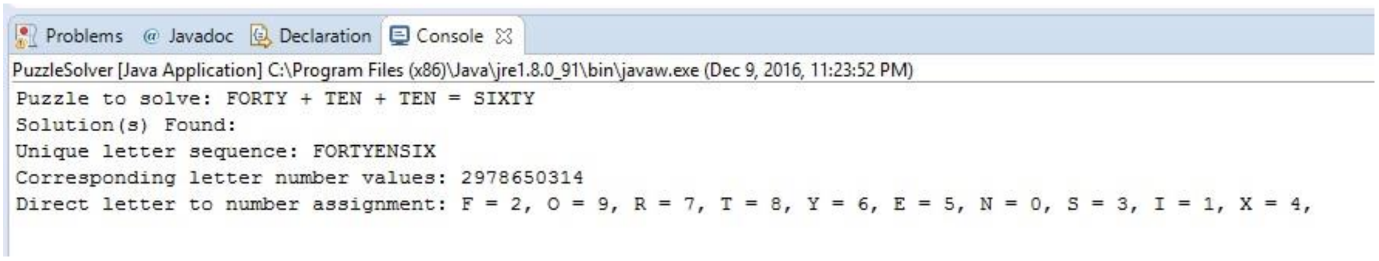
Possible extensions to my project would be using a different data structure. Other ones that maybe could've worked, such as HashTable or Stacks in some manner, could have made the implementation easier. I run low on time and couldn't investigate these, but based on my runtime I suspect that there is a way to use another data structure other than LinkedList. This could've reduced my runtime, code amount, and difficulty if I managed to find another data structure implementation that is more efficient for this project. To improve on these possible extensions, simply finding other implementations that can do the same thing can improve runtime and program overcomplexity.

Test Cases:

There is only one test case for this assignment, to find a solution to the puzzle:

FORTY + TEN + TEN = SIXTY

I could test my LinkedList implementation for correctness, but I know it works since it worked correctly in the happy number assignment and that's where I took most of the implementation from. So, to test out my PuzzleSolver class and program, I would simply need to run my program and see if the solutions it generates are correct. Here are the solutions found for the puzzle:



```
PuzzleSolver [Java Application] C:\Program Files (x86)\Java\jre1.8.0_91\bin\javaw.exe (Dec 9, 2016, 11:23:52 PM)
Puzzle to solve: FORTY + TEN + TEN = SIXTY
Solution(s) Found:
Unique letter sequence: FORTYENSIX
Corresponding letter number values: 2978650314
Direct letter to number assignment: F = 2, O = 9, R = 7, T = 8, Y = 6, E = 5, N = 0, S = 3, I = 1, X = 4,
```

As you can see, there is only one solution to this puzzle, based on my program's results. So, checking if this is correct is very simple. Substitute in the corresponding letter to number assignments into the puzzle and see if it's correct:

FORTY + TEN + TEN = SIXTY

29786 + 850 + 850 = 31486

29786 + 1700 = 31486

31486 = 31486

By this checking of the solutions, I'm confident in the results and functionality of my program (besides the obvious runtime issue, unless no matter the implementation it will take some time to run through all possibilities).