Jasque Saydyk

Professor Patrick Kelley

CS 249 Section 3140

19 March 2017

Project 2 - Linked Lists, Stacks, and Queues

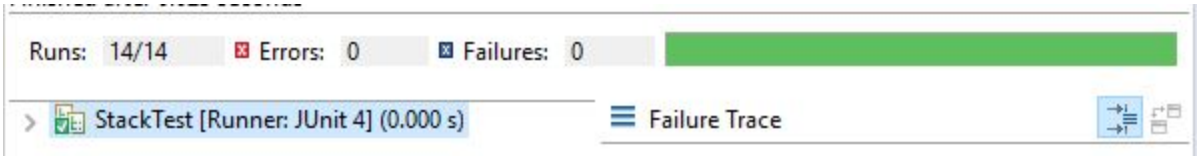
LinkedList.java

I first began writing the LinkedList class, and at the beginning, I wrote the class just by thinking about how each of the methods should work. The method I spent the most time on was the add method, with had so many edge cases that needed to be accounted for. The class took me about two half days of work to finish and pass all the tests for the class. However, when I moved onto doing the Stack class, I had a difficult time as I was constantly getting NullPointerExceptions, with the problem normally pointing to the get method in LinkedList, which didn't make sense, as it was pretty simple to implement. After grinding my wheels against the problem, I realized that my add method was completely broken and was not linking to other nodes correctly. The reason why it took me so long to figure this out was because I wrote a lot of spaghetti code that was confusing to read and look at. Upon realizing this, I re-wrote the class, fullying drawing out what I was doing on paper, then writing pseudocode, then typing out the class. Additionally, I originally had each node store its index before I realized I could easily have for loops do the same thing. After many hours spent carefully thinking and testing each line of code using a personal test class, I was finally able to move on to the rest of the project. This class is O(n) complex, as the most complex think it does is have some for loops scattered about the class, none of which are nested.

Stack.java

When I first began this class, I ran into a lot of NullPointerException problems as my LinkedList was poorly implemented, but when I reimplemented that class, I ran into far fewer problems. The only hiccups I ran into past that was that I implemented the Stack class to push and pop at the end of the list, but when I looked closely at the tests for the class, I realized that it wanted me to push and pop at the head of the list. This was an easy correction to make, and my anger stemmed from the small amount of wasted worked when I realized it. The other problem I ran into was that a single line in the remove method of LinkedList caused one of the stack tests to

failed, and I fail to see why this is the case, as I am confident the line is necessary for the program to be truly flexible. This class is mostly O(1), as most of the class only pops or pushes to the head of the list, with the exception of the toString method, which has a for loop, but also runs the LinkedList get method, which also has a for loop, making this method O(n^2).

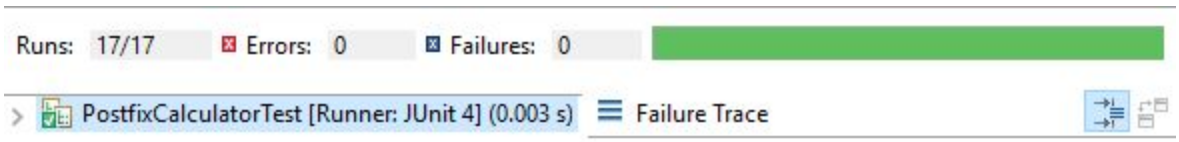


Queue.java

This class was pretty straightforward to implement. The only mistake I made was that, having just implemented the Stack class, I made the push method push at the head of the list, and the pop at the tail, which upon running the tests was discovered to be incorrect. I then had to change those methods to push at the tail and pop at the head. This class is mostly O(1) complexity, as the class only adds and removes from the head and tail of the list, which are done in constant time, with the exception of the toString method, which has a for loop, but also runs the LinkedList get method, which also has a for loop, making this method O(n^2).

PostfixCalculator.java

This class was straightforward to implement, as I choose the simplest way to implement it. I did fiddle with the idea of using regular expressions, which is something that I need to learn anyway, but I lacked the time to do so, as I spent way too much time implementing the LinkedList class. When I implemented the class, I followed the instructions on the lab report, implementing the methods one at a time, then testing to ensure they worked. The complexity of this class is mostly O(1), as most of the class uses the stacks push and pops, with are constant as stated above. The only exception is the inputEquation method which is O(n), as it uses a for loop to format the output of the function.



InfixCalculator.java

This class took a good amount of time for me to understand the problem, requiring me to fill several sheets of paper manually working out how the algorithms should work. As for implementing the class, I followed the lab, slowly implementing each method, and testing to ensure that it worked. I quickly realized that the included tests were very barebone, and didn't full test the methods, so I created my own tests to ensure that the code functioned correctly and gave the correct answers. The complexity of this class is mostly O(1), as most of the class uses the stack and queue to do pushes and pops, which are constant as stated above. The only exception is the convertEquation method which is O(n), as it uses a for loop to create the output of the function.