Curtis Clark
9/21/18

Project 1 - Linked List Line Editor

Commentary

What is the computational complexity of the methods in the implementation?

- getHead()
  - O(1)
- insertAtIndex()
  - O(n)
- deleteNode()
  - O(n)
- edit()
  - O(n)
- insertEnd()
  - O(n)
- find()
  - O(n)
- sanitizeStr()
  - O(1)

Your thoughts on the use of linked lists for implementing a line editor. What are the advantages and disadvantages?

- While considering how to answer this question, I will admit I thought that a line editor based on a string array would be a better implementation. But as I started analyzing the time complexity differences, it becomes immediately apparent that the linked link implementation is superior.

  While it it O(1) to dereference a line to edit when using an array, as opposed to the traversal for lined lists, the structurally altering functions (insert, delete) run

the risk of requiring an array resizing which is O(n). If the insert or delete is not at the end of the array, some or all existing elements will need to shift as well, which guarantees a O(n) complexity stil.

A linked list makes quick work of inserting and deleting before and after nodes, which means no resizing and no shifting of array elements. This would make those methods O(1), given the program has already traversed to the correct node location. This can be done by keeping track of the most recent line updated, the head, and the tail elements. I have O(n) because…

(see ‘What I would do differently‘)

What did you learn from this assignment and what would you do differently if you had to start over?

- What I learned:
    - This assignment taught me the advantages of using a linked list as an alternative to an array. In my work life, it is deceptively easy for me to become complacent and use arrays for data handling. Lessons like this remind me that easier is definitely not always better.
- What I would do differently
    - The biggest change I would make to my code is to create a Document class to house the LinkedList object and to track the most recent active line as a Node* object and the most recent active line number. Had I done this, I would still have a O(n) complexity for insert and delete (due to some unavoidable traversal), but the average time complexity would be WAY lower as my traversals to the “target” node would take less loops.
    - I would keep a reference to the tail element in the LinkedList. This way, the insertAtEnd() function would be O(1).