Damien Bobrek
COP3530: Data Structures and Algorithms
2018-09-21
Programming Assignment 1: Implementing a Line Editor Using Linked Lists

Programming Assignment 1 Commentary

1. For this assignment, I opted to create a doubly-linked list. The list is written to be generic, so it can be used for different data types as needed. Internally, it uses a struct defined in the private scope for its nodes. Inserting, removing, and accessing (insert(), remove(), and access(), respectively) are executed in O(n) time. The various other functions written specifically for dealing with the ends of the list (push_front(), push_back(), pop_front(), pop_back(), front(), and back()) all will always take O(1) time. Getting the size of the linked list with size() also takes O(1) time. The worst case is doing operations in the middle of the list, since reaching the middle takes the longest; in a list of size n, it takes n/2 iterations to reach the middle of the list before any other operations can be done.

The operations available to the user vary in execution time. The command insertEnd takes O(1) time. The commands insert, delete, and edit all takes O(n) time. Sending the quit command also takes O(n) time, since it causes the program to shut down, which means that the destructor for the linked list is run and takes O(n) time to remove all elements. The commands for print and search, however, take O(n^2) time because each individual line is accessed with a call of the access() function, and each call of the access() function is O(n).

2. I think that using a linked list for an editor is useful on documents that are not too large. It is easy to make changes at any line in a document on smaller documents. Taking O(n) time to reach any line in the document is acceptable up until when edits have to be made on a large document, at which point accessing individual lines begins to take too long.

3. This assignment taught me more about using templates in C++ and handling memory without leakage. I learned to work more with templates because my Linked List implementation uses a template. I learned how to handle memory better through using Valgrind to detect memory leaks. Looking for leaks was what actually prompted me to write a proper destructor for my linked list, something I had also not done before.

If I were to do this project again in the future, I would have probably tried to make it simpler for myself. I would probably write a singly-linked list for simplicity. Regardless of whether I write a singly- or doubly-linked list, I would definitely improve it by implementing an iterator. Doing so would allow me to significantly improve the execution time of the print and search commands; they would finish in O(n) time instead by creating an iterator that traverses the list, instead of taking O(n^2) time because of calling the access() function repeatedly.