Sean Minchin

COP3530 Project 1 Commentary

Linked List functions

append: O(n) or O(1) if empty

prepend: O(1)

insert: O(n) or O(1) if inserting at beginning

get: O(n) or O(1) if retrieving head

set: O(n) or O(1) if setting at beginning

remove: O(n) or O(1) if deleting at beginning

print: O(n)

Line Editor functions

insertEnd: O(n)

insert: O(n)

delete: O(n)

edit: O(n)

search: O(n^2)

print: O(n^2)

Advantages and Disadvantages

The main advantage of using a linked list as a line editor is the ease at which the list can grow and shrink in size. If the initial number of lines of text (nodes) in the list, i.e., the initial size of the list is not known (as is the case with this project), then adding and removes lines of text is more computationally efficient than using an array-based list. A major disadvantage, however, is the complexity of editing and retrieving from the list compared to an array-based list. Access by index is constant time with array-based lists, while the process is linear-time with node-based lists.

What I learned, would do differently

I learned about the fundamentals of linked list data structure implementation of the list abstract data type. Although I knew the basic idea behind a linked list before this, implementing it with all its methods gave me a fuller understanding of how the process works under the hood.

If I started over, I would probably make a doubly-linked list with nodes that point to both a previous and next node for better efficiency accessing the list nodes. A doubly-linked list can be traversed without having to keep track of the previous node with a pointer. I would also consider using smart pointers for my nodes to make the process of allocating and freeing memory more streamlined and less error-prone.