Programming Assignment I

Report by: Jason Tufenkdjian

Section 1 - Time Complexity of Methods

For insertions and deletions, the operation itself is O(1) in every case, but this is dependent on the program having the address of the node to change and its previous node beforehand. Due to the nature of how this program implements insertion, deletion, and editing, these operations take O(n) time. That is, it is required to traverse through the list to find the node to be changed and, in some cases, its previous node.

The search function must search through the entire document to find the desired string, which guarantees a time complexity of O(n) in every case.

Command	Inputs	Average Big-O
insert	int at, string s	O(n)
insertEnd	string s	O(n)
edit	int at, string s	O(n)
search	string s	O(n)
delete	int at	O(n)
print	-	O(n)

Fig. 1 Time complexities of commands from this assignment

* insertEnd and print will have time complexity O(1) if the list is empty or has one element

Section 2 - Meditations on using Linked Lists for Line Editors

The use of a singly linked list confers few advantages when it comes time to editing. Each operation is of O(n) time complexity, including insertions at the end of the file. A doubly linked list would at least reduce the time complexity of InsertEnd from O(n) to O(1) immediately. Because it is required to traverse the list for every change in the file, it will perform poorly if the user is required to give the line which s/he wishes to change, one at a time.

The performance of a doubly-linked list text editor could be improved if used through command line. In such a case, the program would keep track of a pointer to the “current line,” where the user’s cursor is currently pointed to. The deletion, insertion, and edit operations would be O(1) in each case since the program has a reference to the current, previous, and next nodes from the onset.

The performance of a full search and print function cannot and would not be improved, but the rest of the operations would perform quite well for a command-line text editor, as shown in Figure 2 below.

Command	Inputs	Worst Case Big-O	Average Big-O	Best Case Big-O
insert	string s	O(1)	O(1)	O(1)
insertEnd	string s	O(1)	O(1)	O(1)
edit	string s	O(1)	O(1)	O(1)
search	string s	O(n)	O(n)	O(n)
print	-	O(n)	O(n)	O(n)
goToLine	int lineNo	O(n)	O(n)	O(1)
delete	int at	O(n)	O(n)	O(n)

Fig. 2 Time complexities of commands for a hypothetical Line Editor utilizing a doubly-linked list

Figure 2 is slightly misleading, however, since each change to the file would require the user to manually go to the line to be changed in order to make his or her changes to the file. In this way, a text editor using a doubly-linked list could potentially be slower by requiring the

intervention of the human user to determine both what and where to make changes through using the program. Figure 2 has an additional command, goToLine(int lineNo), in order to allow the user to jump from one line to another based on whether s/he has found a line that needs to be changed through the search(string s) function. This would make such a program more user friendly by allowing the user to go to a particular paragraph or page easily to make changes.

A doubly-linked list may, indeed, make a good text editor, but perhaps it would not scale well when designing a full document editor. I believe Microsoft Word uses a special type of binary tree designed for strings called a rope, which can achieve O(log(n)) time complexity for insertions and deletions when the tree is balanced.

Section 3 - Lessons and Areas of Improvement

I learned the following things during this assignment:

1. Although linked lists may seem complex to those without much experience, they are deceptively simple to use and implement.
   - The only members necessary for a (singly) linked list is the node's data and a pointer to the next node, the rest is just a matter of sorting out the details.
2. When implementing linked lists, it is very important to keep track of whether you run into a null pointer and are certain of what to do with it if such a case arises
   - Example: for this assignment, it is required to insert a line even if the line number is out of bounds by one (essentially making the function into an insertEnd implementation), while it is required to simply ignore a delete request make out of bounds by one. In each case, the program will traverse through the list until it reaches a null pointer, but the behavior will be different depending on the use case.
3. The biggest weakness of a linked list is traversal. If traversal can be wholly avoided, then a linked list is quite effective, especially as compared to making insertions and deletions on an array.

If given the opportunity to do this assignment again, I would make the following changes:

1. Implement a doubly-linked list instead of a singly-linked list.
   - Would reduce the time complexity of insertEnd to O(1)
2. Add an integer to each node called "lineNo." This would allow the program to check if an insertion or deletion is out of bounds prior to attempting by traversing through the list.
   - Would reduce the time complexity of out-of-bounds requests to O(1) from O(n)
   - Would allow the program to traverse from the back of the list if it is found that the line number for insertions, deletions, or edits is closer to the back of the list than the front, making performance slightly better.