Tong Zhang
Yiru Wang
CSCI 241

Linked List

For the asymptotic analysis of each method they go as follows:

- append_element(…) - The time complexity for the append element function is O(1) – constant time. The way this function works is that it would start at the trailer node and through several steps, simply make its previous node the one chosen to be appended. The amount of time it takes for this to happen is completely independent of the list length as the process will start from the end. Best/Worse case would be O(1). This displays the distinct advantage of a doubly linked list over a singly linked list. With a singly linked list, a current walk must be performed to the end of the linked list.

- insert_element_at(…) – The insertion function is really divided up into two parts. First, would be finding the position the insertion would take place. Second, would be inserting the element itself. The first part would take linear time O(n), as it would have to do a current walk and touch each node until it reaches the desired index. The actual insertion is O(1)- constant time. Added together, the insert element function would have an overall time complexity of O(n). However, the better case scenarios for insert element are more apparent because in many cases it will not have to touch any node after the desired index, as compared to an array which much touch every single element of the index despite where it is inserting as it must perform a shift for all later indices. This is why linked lists hold a distinct advantage as a data structure when compared to arrays. Best case would be O(1), worst case would be O(n).

- remove_element_at(…) – The remove function would in almost all ways be similar to the insert function. The first part of the function is a current walk to find the desired index and the second part would be to remove the element at the desired index. The current walk would have O(n) time complexity while the actual removal would have an O(1) time complexity – giving it an overall time complexity of O(n). However, again, the better cases for this function are more apparent because the function will not have to touch every single node, unlike in an array which will shift every cell it did not touch while going up to the index. Best case would be O(1), worst case would be O(n).

- get_element_at(…) – The get element function would hold to linear time. It can be compared to the first part of the insert and remove function. In that once it reaches the desired index it will return the value stored in the node. This is a slight disadvantage it would have when compared to arrays, as the access time for an array would hold to constant time. Best case would be O(1), worst case would be O(n)

When testing the implementation three general cases were taken into account. The case of an empty list, the case of a list size 1, and the case of a list with multiple elements. For each general type of list, referring to the index functions, a valid index was tried and an invalid index was tried. After each modification of the linked list, the list itself was printed as was its length. For each of these general types the iteration method was also tested. Through each modification of the list, several factors were taken into account. Namely – “Was the modification performed correctly?”, “Did invalid modifications leave the attributes of the list unchanged?” Additionally, for the sake of flexibility, and comprehensiveness, several methods testing the same case were repeated in case previous modifications would affect future ones in that no methods overlapped.