Lab 2 – Linked Lists

Jonatan M. Contreras

University of Texas at El Paso

**Introduction**

For this lab, we were asked to implement a SortedList() class. This class is a linked list implementation that is always sorted in ascending order. This sortedList() class was to implement 10 different functions, which are: Print(self), Insert(self, i), Delete(self, i), Merge(self, M), IndexOf(self, i), Clear(self, i), Min(self), Max(self), HasDuplicates(self), and Select(self, k). We also implemented these functions on a regular List() class and compared running times for each class’s functions.

**Proposed Solution Design and Implementation**

*Part 1: SortedList() Functions:*

Print(self):

The purpose of the Print(self) function is to print the contents of the linked list. I implemented it such that a dummy variable takes the value of the head of the list if it is not empty and traverses the list. While at each node, the function prints the data of that node. If the head is empty, it prints “empty list.” Since Print(self) loops through the entire linked list, it runs at O(N).

Insert(self, i):

Insert(self, i) inserts an element i into the sorted linked list. Since the SortedList() class maintains a linked list in ascending order at all times, the implementation of this method took some thought. If the head is None, the element i becomes the head and tail. Else, a dummy variable iterates through the list, comparing the next Node’s data to i, looking for the appropriate place for i to be placed. The iteration stops right before that element, then places that Node in that spot. Since the function iterates through the list to add an element, worst case of the function is going to the end of the list. It only needs to do this one, so the running time for this function is O(N).

Delete(self, i):

Delete(self, i) finds the element i in the sorted linked list and removes it. For this function, a dummy variable iterates through the linked list, comparing the dummy variable’s next.data to i. If they are equal, the dummy variable’s next becomes the dummy’s variable’s next.next, removing that element. It ends by checking to see if the head has the same data as i, and if it does, changes the head to the head’s next. In the worst case scenario, Delete(self, i) iterates through the whole list once. This causes it to run in O(N) time.

Merge(self, M):

Merge(self, M) merges a sorted linked list M into the current sorted linked list. To achieve this, I created a temporary SortedList() variable that is empty. I then used two dummy variables, one for each sorted list’s head, and iterated through both lists at the same time. While doing this, the function compares the data from each current Node and places the smallest element into the temporary SortedList(). It then moves the iterator of the list that had its element inserted into the temporary list forward. If the lists are not of equal length, one list will be iterated through completely before the other. So after this loop, there is another loop that finishes iterating through the other, inserting its elements where they must go in the temporary sorted list. The function finishes by making the current linked list’s head and tail that of the temporary linked list.

The function iterates through the M sorted list m times, m being the size of M. It also iterates through the current linked list n times, n being the length of the list. At each iteration, of m and n, the function uses the insert function, which runs in n time, to insert the current element being considered into the temporary sorted list. Thus, the running time of this function is o(m\*n + n2).

IndexOf(self, i):

IndexOf(self, i) finds the index of the first instance of the element i in the sorted linked list. To achieve this, a dummy variable is assigned to the head of the list and iterates through the list, comparing the current Node’s data to i. A counter keeps track of the index. If there is a match, it exits the loop and returns the counter. In the worst-case scenario, the function needs to loop through the entire list. Thus, it runs in O(N) time.

Clear(self, i):

Clear(self, i) makes the current list an empty list. To achieve this, the head and tail are set to None. This runs in constant time, or O(1).

Min(self):

Min(self) returns the smallest element in the list. Since the list is sorted, the smallest element is the head of the list. Thus, my implementation returns the head of the list. This is in constant time, or O(1).

Max(self):

Max(self) returns the largest element in the list. Since the list is sorted, the largest element is the tail of the list. Thus, my implementation returns the tail of the list. This is in constant time, or O(1).

HasDuplicates(self):

This function returns True if there are multiple Nodes with the same data or False otherwise. To implement this function, a dummy variable that is assigned with the current list’s head and iterates through the list, comparing the dummy variable’s next element’s data to the dummy variable’s data. If they are equal, the function returns true. Else, the dummy variable is assigned with the next Node. If the iteration goes through the whole list without returning True, it returns false.

In order to complete this process, the list needs to be completely iterated through. This means that the function runs in O(N) time.

Select(self, k):

Select(self, k) returns the kth smallest element in the sorted list. Since the list is already sorted, we only have to iterate through the list up until the kth element. Thus, a dummy variable is assigned to the head of the list and iterates through the list. Using a counter, the function keeps track of the current index and compares it to k. Once counter is equal to k - 1, the function returns the data at that current Node. Since the function, in the worst case scenario, needs to iterate through the whole list, the running time is O(N).

*Part 2: List() Functions:*

**Experimental Results**

**Conclusion**

**Appendix – Source Code**

**Academic Honesty Certification**

I certify that this project is entirely my own work. I wrote, debugged, and tested the code being presented, performed the experiments, and wrote the report. I also certify that I did not share my code or report or provided inappropriate assistance to any student in the class.

Jonatan Contreras