**CS 2302 Data Structures**

**Spring 2019**

**Lab Report #3**

Due: October 4, 2019

Professor: Olac Fuentes

TA: Anindita Nath

**Introduction**

For this lab we were asked to modify the List class which was given to us in class through the completion of several exercises. The modified class would be called SortedList since the insert function would sort the list immediately without having to call a sorting algorithm outside of the insert function. The class contains the functions Print, Insert, Delete, Merge, IndexOf, Clear, Min, Max, HasDuplicates, and Select. These functions are modified since the List is already sorted.

**Proposed Solution Design and Implementation**

**Part #1**

For Part 1, we were asked to implement ten different functions under the SortedList class. These functions are print, insert, delete, merge, index of, clear, min, max, has duplicates, and select. I will go through each function and describe how I solved these functions.

The print function is assigning a temporary variable to the head of the list. It uses a while loop to iterate through the list. It prints the data at the temporary variable and exits the loop when the temporary variable is None or null.

The insert function takes on one parameter which is the integer which will be inserted into the list. If the head is none then it will assign the head with the integer value and make the tail equal to the head. If this is not the case, then the next node after the tail is assigned the integer value. After this the tail is assigned to the next value after the tail. After each case there is a call to the Bubble Sort function which sorts the list while still being in the insert function.

The delete function takes one parameter which is the value that will be deleted from the list. This function, as many others, assigns a temporary variable to the head of the list and iterates through the list using a while loop. If the next node after the temporary variable is none then nothing happens. If this is not the case and the value that is to be deleted equals the next node after the temporary variable, then we make the node after the temporary variable equal to the one next to this one. This essentially skips over by changing what the temporary variable points to.

The merge function takes as a parameter another list. This list is iterated through using the same method used for other functions. Each node of this list is inserted into the primary list. Through this insert function the list is sorted as well.

The index of function takes on one parameter which is a value in the list. This function returns the index where this value is in the list. This function iterates through the list the same way other functions have done by using a while loop which holds the condition that the head of the list is not none and updates to the next node. This function also creates a variable called index which will count the index in the list since a list in python does not have a built-in index. If the data at temporary variable is the same as the parameter passed, then the function returns the index which is updated at the same time that the node is updated in the while loop.

The clear function sets the head of the list to none as well as the tail of the list to none. The minimum functions return the value at the head of list. Since the list is sorted the smallest element will always be the head. This rule applies for the maximum function as well. The maximum function returns the value at the tail of the list. Since the list is sorted the maximum value will always be at the tail of the list.

The has duplicates function searches the list to see if the list contains duplicate elements. It does this by iterating through the list using a while loop, as previously described. The only modification is that the while loop holds the condition that while the head does not equal the tail then the data at the head will be compared to the value next to it. If this is true, then the function returns true. If this is not the case, then the head is updated to be the value next to it and iterates once more. If it goes through this while loop without returning true then the function returns as false.

The select function returns the k-th smallest element in the list with k being the parameter passed in the select function. If the head is none then none is returned. For the select function we will use the same type of iteration as the index of function. This means we will create a temporary variable called index that will count at what index we are at as the while loop iterates. If the index is equal to k then we will return the data of this value in the list.

**Experimental Results**

**Part #1**

﻿Printing list

1 3 5 7

Runtime: 0.0024042129516601562

Inserting 8 into list

Runtime: 4.100799560546875e-05

1 3 5 7 8

Deleting 3 from list

Runtime: 8.106231689453125e-06

1 5 7 8

Merging M to List

2 4 6

Runtime: 2.384185791015625e-05

1 2 4 5 6 7 8

Returning index of 1

0

Runtime: 1.9788742065429688e-05

Returning Min Value

1

Runtime: 0.0032210350036621094

Returining Max Value

8

Runtime: 2.002716064453125e-05

Checking if there are duplicates

False

Runtime: 4.601478576660156e-05

Returing k-th smallest element

1

Runtime: 6.508827209472656e-05

Clearing List:

Runtime: 7.867813110351562e-06

|  |  |  |
| --- | --- | --- |
| Function | SortedList | List |
| Print() | O(n) | O(n) |
| Insert(i) | O(n) | O(n) |
| Delete(i) | O(n) | O(n) |
| Merge(M) | O(n) | O(n) |
| IndexOf(i) | O(n) | O(n) |
| Clear(i) | O(1) | O(1) |
| Min() | O(1) | O(n) |
| Max() | O(1) | O(n) |
| HasDuplicates() | O(n) | O(n) |
| Select(k) | O(n) | O(n) |

Running Time

**Conclusion**

In conclusion this lab taught me of how simple modifications to code can change the time complexity from linear to constant time. This is clearly seen through the difference between the List class and the SortedList class. The main differences are seen when trying to access a certain element that has a specific quality. When trying to find the minimum element in a normal list you have to iterate and compare through each element. When the list is sorted the smallest element will always be at the head. This type of modification has shown me that minor changes to code can drastically make your program run more efficiently.

**Appendix**

**﻿**

**﻿**

**﻿class Node (object):**

**def \_\_init\_\_(self,data,next=None):**

**self.data = data**

**self.next = next**

**class SortedList(object):**

**def \_\_init\_\_(self):**

**self.head = None**

**self.tail = None**

**def isEmpty(self):**

**return self.head == None**

**def AppendList(self,python\_list):**

**for d in python\_list:**

**self.Insert(d)**

**def GetLength(self):**

**t = self.head**

**w = 0**

**while t is not None:**

**w += 1**

**t = t.next**

**return w**

**def BubbleSort(self):**

**listlen = self.GetLength()**

**for i in range (1,listlen):**

**t = self.head**

**for j in range (listlen - 1):**

**if t.data > t.next.data:**

**t.next.data, t.data = t.data, t.next.data**

**t = t.next**

**def Print(self):**

**t = self.head**

**while t is not None:**

**print(t.data,end = ' ')**

**t = t.next**

**print()**

**def Insert(self,x):**

**if self.head is None:**

**self.head = Node(x)**

**self.tail = self.head**

**self.BubbleSort()**

**else:**

**self.tail.next = Node(x)**

**self.tail = self.tail.next**

**self.BubbleSort()**

**def Delete (self,i):**

**t = self.head**

**while t is not None:**

**if t.next is None:**

**return**

**elif i == t.next.data:**

**t.next = t.next.next**

**t = t.next**

**def Merge(self,M):**

**t = M.head**

**while t is not None:**

**self.Insert(t.data)**

**t = t.next**

**def IndexOf(self,i):**

**t = self.head**

**index = 0**

**if t is None:**

**return**

**while t is not None:**

**if t.data == i:**

**return index**

**t = t.next**

**index += 1**

**def Clear (self):**

**self.head = None**

**self.tail = None**

**def Min(self):**

**return self.head.data**

**def Max(self):**

**return self.tail.data**

**def HasDuplicates(self):**

**if self.head == None:**

**return False**

**t = self.head**

**while t != self.tail:**

**if t.data == t.next.data:**

**return True**

**t = t.next**

**return False**

**def Select(self,k):**

**t = self.head**

**index = 0**

**if t is None:**

**return 0**

**while t is not None:**

**if index == k:**

**return t.data**

**t = t.next**

**index += 1**

﻿from Lab\_3 import SortedList

from timeit import time as timer

if \_\_name\_\_ == "\_\_main\_\_":

L1 = SortedList()

L1.AppendList([7,5,3,1])

print ("Printing list ")

start = timer.time()

L1.Print()

end = timer.time()

print('Runtime:',str(end-start))

print ("Inserting 8 into list ")

start = timer.time()

L1.Insert(8)

end = timer.time()

print('Runtime:',str(end-start))

L1.Print()

print ("Deleting 3 from list ")

start = timer.time()

L1.Delete(3)

end = timer.time()

print('Runtime:',str(end-start))

L1.Print()

print ("Merging M to List" )

M = SortedList()

M.AppendList([6,4,2])

M.Print()

start = timer.time()

L1.Merge(M)

end = timer.time()

print('Runtime:',str(end-start))

L1.Print()

print ("Returning index of 1 ")

start = timer.time()

print (L1.IndexOf(1))

end = timer.time()

print('Runtime:',str(end-start))

print ("Returning Min Value ")

start = timer.time()

print (L1.Min())

end = timer.time()

print('Runtime:',str(end-start))

print ("Returining Max Value" )

start = timer.time()

print (L1.Max())

end = timer.time()

print('Runtime:',str(end-start))

print ("Checking if there are duplicates ")

start = timer.time()

print (L1.HasDuplicates())

end = timer.time()

print('Runtime:',str(end-start))

print ("Returing k-th smallest element ")

start = timer.time()

print (L1.Select(0))

end = timer.time()

print('Runtime:',str(end-start))

print ("Clearing List: ")

start = timer.time()

L1.Clear()

end = timer.time()

print('Runtime:',str(end-start))

L1.Print()

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