Lab Assignment 3

Ismael Villalobs

Professor Fuentes

TA: Dita nath

10-8-19

Introduction

The purpose of this lab was to master and get more comfortable with the List class we have seen in the course. We were asked to create our own version of List using SortedList() which ordered the list in ascending order at all times. In addition to the unique creation of different operations that produced max, min, and to check for duplicates, to name a few.

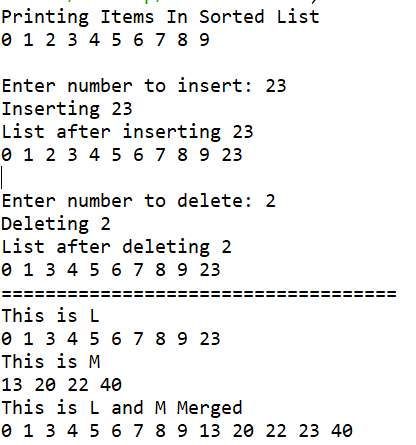
Proposed Solutions

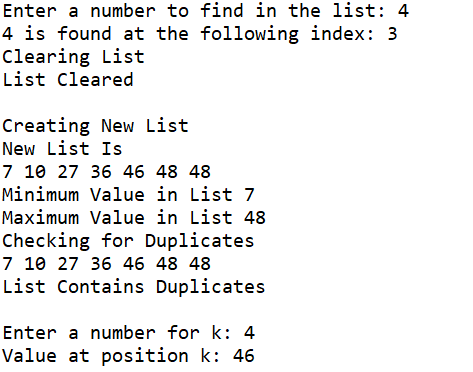
Using the code provided in class List() I used it as a basis to construct my SortedList() class. With the constructors similar to the ones provided I was able to create an *Append* function that used loops and conditional statements to make sure that the list was always in ascending order and this worked in conjunction with the *Insert* function to ensure that the items in the linked list were always inserted in the correct place.

This is how I proposed to solve the functions:

1. **Print**: This prints the elements of the list. This function takes the list *self* loops through the entire list and prints the data of the current head node until it reached the last value and self.head was None.
2. **Insert**: This function inserts element to the existing list. This function takes the list and the item that is being inserted and calls the Append function so that the item is placed the the correct spot within the line so that the list remains in ascending order.
3. **Delete**: This function deleted item selected from the list. For this function when removing integer i from the list it would find the data that is being removed and created a reference to it. The pointer would then point the current to the next item. For instance if the item was t then self.head would point to t.next.
4. **Merge**: This function merges 2 lists into 1 list. This function would call the Insert function and insert the new list M into the original List L. Since it uses the Insert function it will place the elements from M in the correct ascending order to produce a single list.
5. **IndexOf**: This function returns the position of the item chosen. This function uses a counter to find the item selected when the current head reached the item chosen and returned the position found or -1 if the item was not found in the list.
6. **Clear**: This function clears all elements of list. Since our list class uses the head of the list as a reference to iterate through the list if we made the head point to None it would not point to anything thus making the list empty.
7. **Min**: Method returns smallest value in list. This method returns the head node since the list is already sorted in ascending order.
8. **Max**: Method returns the largest value in list. This method should return the tail value since the list is already sorted in ascending order.
9. **HasDuplicates**: This function returns True or False depending on if list has duplicate items. Since the list is already sorted is the item had a duplicate it would be the next value. Thus, if t.data = t.next.data the item is duplicate otherwise there are no duplicates.
10. **Select**: This method returns the data of the node at an index. This method returned the Kth largest element so since the list was already sorted it would use a counter and the k position we want to find and when the counter equals the index we return the data.

Experimental Results





Big O Notation & Runtimes

|  |  |  |
| --- | --- | --- |
| Function | Sorted *List* | *List* |
| Print() | O(n) | O(n) |
| Insert(i) | O(n) | O(1) |
| 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(n) should be O(1) | O(n) |
| Max() | O(n) should be O(1) | O(n) |
| HasDuplicates() | O(n) | O(n) |
| Select(k) | O(n) | O(n) |

Conclusion

In conclusion, this lab was good deliberate practice and it allowed us to implement a custom version of the List class. I am more confident when tackling linked list problems now and better understand their runtimes.

Academic Honesty

“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.”

Name: Ismael Villalobos

Appendix:

'''

Course: CS2302 Data Structures Fall 2019

Author: Ismael Villalobos

Assignment: Lab 3 Linked Lists

Instructor: Dr. Olac Fuentes

TA: Anindita Nath

Last Modified: October 5th 2019

Purpose:

'''

import math

import random

class Node(object):

# constructor

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

self.data = data

self.next = next

# SortedList functions

class SortedList(object):

# constructor

def \_\_init\_\_(self, head=None, tail=None):

self.head = head

self.tail = tail

def Print(self):

# prints items in order using a loop

t = self.head

while t is not None:

print(t.data, end=' ')

t = t.next

print()

def Append(self, x):

node = Node(x)

current = self.head

previous = None

while current is not None and current.data < x:

previous = current

current = current.next

if previous == None:

self.head = node

else:

previous.next = node

node.next = current

def Insert(self, i):

self.Append(i)

def Delete(self, i):

t = self.head

if t is None:

return None

elif t.data == i:

if t.next is None:

self.head = None

else:

self.head = t.next

else:

prev = t

while t is not None:

if t.data == i:

prev.next = t.next

prev = t

t = t.next

def Merge(self, M):

x = M.head

while x is not None:

self.Insert(x.data)

x = x.next

def IndexOf(self, i):

t = self.head

index = 0

while t is not None:

if t.data == i:

return index

index += 1

t = t.next

return -1

def Clear(self):

self.head = None

def Min(self):

if self.head is None:

return math.inf

return self.head.data

def Max(self):

t = self.head

if t is None:

return -math.inf

max = -math.inf

while t is not None:

if t.data > max:

max = t.data

t = t.next

return max

'''

if self.head is None and self.tail is None:

return math.inf

else:

return self.tail.data

'''

def HasDuplicates(self):

t = self.head

if t is None or t.next is None:

return False

while t is not None:

if t.next != None:

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

return True

t = t.next

return False

def Select(self, k):

if self.head is None:

return math.inf

index = 0

t = self.head

while t is not None:

if index == k:

return t.data

index += 1

t = t.next

L = SortedList() # create empty list to use for main output

M = SortedList() # list for merging

# fill list with random values between 0-50

for i in range(0, 10):

L.Append(i)

for i in range(0, 10):

M.Append(random.randint(0, 50))

print('Printing Items In Sorted List')

L.Print()

insertion= int(input('Enter number to insert: '))

print('Inserting',insertion)

print('List after inserting',insertion)

L.Insert(insertion)

L.Print()

deletion= int(input('Enter number to delete: '))

print('Deleting',deletion)

print('List after deleting',deletion)

L.Delete(deletion)

L.Print()

print('====================================')

print('This is L')

L.Print()

print('This is M')

M.Print()

print('This is L and M Merged')

L.Merge(M)

L.Print()

key = int(input("Enter a number to find in the list: "))

print(key, "is found at the following index:", L.IndexOf(key))

print('Clearing List')

L.Clear()

print('List Cleared')

L.Print()

print('Creating New List')

for i in range(0, 10):

L.Append(random.randint(0, 50))

print('New List Is')

L.Print()

m = L.Min()

print('Minimum Value in List',m)

mx = L.Max()

print('Maximum Value in List',mx)

print('Checking for Duplicates')

L.Print()

dup=L.HasDuplicates()

if dup == True:

print('List Contains Duplicates')

else:

print('List Does Not Contain Duplicates')

K = int(input("Enter a number for k: "))

print("Value at position k:", L.Select(K))