Lab 2 Sorting

Fernando De Santiago

University of Texas at El Paso

Lab 2 Sorting

Lab #2 focuses on 4 different types of sorting methods using linked lists. We were given only a base file that needed the additional methods for it to work. The first method was consisting of Bubble sort that was written in one method. The second method was Merge sort which was broken down into 3 parts which consisted of one that splits the list, one that sorts them and the last one that merges them together to have one list. The third was Quick sort which was written in one part on it’s own and has some errors that sometimes doesn’t print out the whole list. The last was Modified quick sort which consisted of just one part as well and was built upon the previous quick sort.

# Bubble Sort

For Bubble sort I began with thinking about the logic behind it along with a picture to help me understand and how to tackle the problem. This was one of the easier methods to write along with the easiest to understand with using lists.

## Merge Sort

Merge sort consisted of 3 methods that work together for it to print out the correct list. The first one creates two lists by calling another method called DL in order to break apart the lists into individual numbers in lists. After that it calls Merge sort in order to make sure that the two lists are split up completely then it calls ML which puts them together in the correct order and returns the list.

### Quick Sort

Quick Sort was the most difficult one for me even with pseudocode as well as a drawn representation since I am not 100% confident using linked lists and the program doesn’t work 100% of the time. But this method consisted of checking a pivot with the other elements in the lists and if the number is less than items in the list then it doesn’t move that one but it’ll change pivot to the next one and change positions if needed till the list is completely sorted

#### Modified Quick Sort

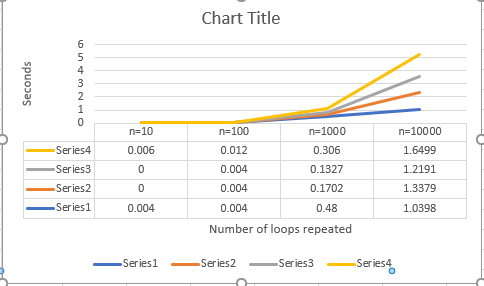
Modified Quick Sort was the second easiest one to do since you were just using a pivot to create 2 lists and whichever list was bigger then you would traverse to that one until you reach the term in the middle if it was sorted. This method was easiest to make since even with linked lists it was just about checking numbers and seeing which lists is bigger or smaller. Some problems were easy to fix since it changed from quick sort and only needed to use some of the code from the other method.

Graph

**Series one is Bubble sort looks like it’s in the worse run time of N^2  
 Series two is merge sort is nLog n**

**Series three is Modified quick sort looks like n log n**

**Series four is quick sort looks like is broken so it’s time is wrong**



In conclusion this lab was built upon understanding of linked lists as well as sorting algorithms. The lab caused me to think outside the box and use references from books as well as talk with classmates to see how I could approach the problems I would face sometimes. I would also spend more time with the TA’s in order to ask for help and see what I was doing wrong. The hardest problem for me was with linked lists in using a pivot since I had never done that before unless in arrays which made me rethink of approaches but overall, I couldn’t figure it out which caused some problems. This lab has taught me in which areas I need to focus on more and how to improve by asking for help when needed.

Source code

# -\*- coding: utf-8 -\*-

"""

Fernando De Santiago

Lab#2

2/24/2019

CS 2302 MW 10:30-11:50

Olac Fuentes, Anindita Nath and Maliheh Zargaran

last edited 2/8/19 9:20:00 PM

"""

import copy

import random

#Node Functions

#Node Functions

class Node(object):

# Constructor

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

self.item = item

self.next = next

def PrintNodes(N):

if N != None:

print(N.item, end=' ')

PrintNodes(N.next)

def PrintNodesReverse(N):

if N != None:

PrintNodesReverse(N.next)

print(N.item, end=' ')

#List Functions

class List(object):

# Constructor

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

self.head = None

self.tail = None

def IsEmpty(L):

return L.head == None

def Append(L,x):

# Inserts x at end of list L

if IsEmpty(L):

L.head = Node(x)

L.tail = L.head

else:

L.tail.next = Node(x)

L.tail = L.tail.next

def Print(L):

# Prints list L's items in order using a loop

temp = L.head

while temp is not None:

print(temp.item, end=' ')

temp = temp.next

print() # New line

def PrintRec(L):

# Prints list L's items in order using recursion

PrintNodes(L.head)

print()

def Remove(L,x):

# Removes x from list L

# It does nothing if x is not in L

if L.head==None:

return

if L.head.item == x:

if L.head == L.tail: # x is the only element in list

L.head = None

L.tail = None

else:

L.head = L.head.next

else:

# Find x

temp = L.head

while temp.next != None and temp.next.item !=x:

temp = temp.next

if temp.next != None: # x was found

if temp.next == L.tail: # x is the last node

L.tail = temp

L.tail.next = None

else:

temp.next = temp.next.next

def PrintReverse(L):

# Prints list L's items in reverse order

PrintNodesReverse(L.head)

print()

def prepend(L,x):

if L.head is None:

L.head = Node(x)

L.tail = Node(x)

else:

newNode = Node(x)

newNode.next = L.head

L.head = newNode

def BS(L):

if IsEmpty(L):

return

Interchanged = False#creating requirement to make a base case

record=L.head

while record.next is not None:#checking that there is a number after head

if record.item>record.next.item:#comparing numbers

catalog=record.item

record.item=record.next.item#swapping values with temporary place holders

record.next.item=catalog

Interchanged= True

record=record.next#moving index to the next one

if Interchanged == True:

BS(L) #recursive to reenter loop and swap values till conditions are met

def ML(L1, L2):

temp = None

if L1 is None:

return L2

if L2 is None:

return L1

if L1.item <= L2.item: #comparing two differnt lists

temp = L1 #storing the smaller list in a temporary list

temp.next = ML(L1.next, L2)#entering a recursively loop to check the following item in the smaller list

else:

temp = L2 #storing the larger list in the temporary list

temp.next = ML(L1, L2.next)#entering a recursively loop to check the following item in the bigger list

return temp

def MS(head):

if head is None or head.next is None: #checking first item in the list and the following item

return head

L1, L2 = DL(head) #splitting a list into 2 list one big and one small

L1 = MS(L1) #sorting the small list

L2 = MS(L2)#sorting the bigger list

head = ML(L1, L2) #combining the lists

return head

def DL(head):

low = head# storing the first item in a list

high = head# storing the first item in another list

if high != None:

high = high.next

while high!= None:

high = high.next

if high!=None:

high = high.next

low = low.next

mid = low.next

low.next = None

return head, mid

def QS(L):

if IsEmpty(L): #checking it's not empty

return

if getLength(L) > 1: #checking that the length is greater than 1

L2 = List()

L3 = List()

pivot = L.head.item #place holder for the item used to compare

Holder = L.head.next#holding the next thing in the list

while Holder is not None:

if pivot > Holder.item: #checking the first item to the next

Append(L2,Holder.item) #appending pivet to the smaller list

else:

Append(L3,Holder.item)#appending to the bigger list

Holder = Holder.next#moving to next item

QS(L2)#recursively enters quick sort

QS(L3)

if L2.head != None:

prepend(L3,pivot)#saving the number at the start of the bigger list

else:

Append(L2,pivot)#saving the number to the end of the smaller list

if L2.head != None:

L.head = L2.head #making the list head the smaller lists head

L.tail = L3.tail#making the list tail to the bigger list tail

L2.tail.next = L3.head#moving the tail of the smaller list equal the head of the bigger one

else:

L.head = L3.head#List head is the bigger list head

L.tail = L3.tail#list tail is the bigger list tail

def MQS(L, mid):

if IsEmpty(L):

return

else:

lowest = List() #create list for small numbers

highest = List()#create list for big numbers

Pivot=L.head.item#Holds number that is used to compare

PlaceHolder=L.head.next#holding the number after the head

while PlaceHolder!=None:

if Pivot<PlaceHolder.item:#comparing first item in the list to the next one

Append(lowest,PlaceHolder.item)#adding the next item in the list to the beginnig of the smallest list

else:

Append(highest, PlaceHolder.item)#adds the next item in the list to the end of the biggest list

PlaceHolder=PlaceHolder.next#moving to next item

if getLength(lowest)>mid:#looking for the mid number

return MQS(lowest,mid)

elif (getLength(lowest))==mid:#finds mid number

return Pivot

else:

return MQS(highest,mid-getLength(lowest)-1)#looking for the mid number but changes by half the list length-1

def getLength(L):

temp = L.head

counter=0;

while temp is not None:

counter+=1

temp = temp.next

return(counter)

def Median1(L):

C = copy.copy(L)

BS(C)

temp=getLength(C)//2

return (ElementAt(C,temp))

def Median2(L):

C = copy.copy(L)

MS(C.head)

temp=getLength(C)//2

return (ElementAt(C,temp))

def Median3(L):

C = copy.copy(L)

QS(C)

temp=getLength(C)//2

return (ElementAt(C,temp))

def ElementAt(L, i):

index =0

temp=L.head

while temp is not None:

if index == i:

print(temp.item)

return temp.item

index=index+1

temp=temp.next

L1 = List()

L2 = List()

L3 = List()

L4 = List()

for j in range(10):

t=random.randrange(100)

Append(L1,t)

for j in range(10):

t=random.randrange(100)

Append(L2,t)

for i in range(10):

t= random.randrange(100)

Append(L3,t)

for i in range(10):

t= random.randrange(100)

Append(L4,t)

PrintRec(L1)

BS(L1)

PrintRec(L1)

Median1(L1)

print('------------------------------')

PrintRec(L2)

L2.head=MS(L2.head)

PrintRec(L2)

Median2(L2)

print('------------------------------')

PrintRec(L3)

print(MQS(L3,getLength(L3)//2))

print('------------------------------')

Print(L4)

QS(L4)

Print(L4)

Median3(L4)

#print(ModifiedQS(L4,getLength(L4)//2))\

def BL(L,n):

AL(L,n)

for i in range(n):

Append(L,i)

L5=List()

BL(L5,6)

Print(L5)

def AL(L,n):

for i in range(n):

i=i+n

return n

AL(L5,5)

Print(L5)

def E(L):

L.head=None

L.tail=None

E(L5)

print(IsEmpty(L5))

Print(L5)