Fernando Zaldivar

February 27, 2019

Lab#2

CS 2302

Data Structures

Spring 2019

**Introduction**

Python has a lot native list however, Python also enable users to create different types of list such as linked list and other data structures. In this lab we use linked list and sort them using different algorithms such as Bubble sort, Merge, Sort and quick sort. The objective is to be able to compare all the running time it take to get all the list sorted.

**Solution**

**Bubble sort**

In order to create a method to sort a linked list using bubble sort, I started off by defining the method, which it would take a list from the class List() the in it would return same list but sorted. The method started with the first integer in the first node and compare it the next nodes. If the original node was larger then it would switch place until it had compared all the nodes. It will then repeat the process until the list is completely sorted.

**Merge Sort**

To complete merge sort I defined several methods the first being merge sort. This algorithm is supposed to take a while list and break in half over and over until the each piece only contains one node. Then recursively it should come back together sorting it as smallest to largest. I used three additional methods, get first half, and get second half, which would do that, take a list and return the first or second half. I also used the method merge which would merge the list in order

**Quick sort**

Quick sort is supposed to also it own method that takes a list from the class List and return a list sorted in order. This method calls for a pivot and index to be defined, Then the as the index is compared to the pivot it switches place whenever the index if the index is lager. Eventually the list is supposed to be sorted in a way where everything to the right is bigger than the pivot and to the left its smaller.

**Circle Pattern**

For this figure we manipulated the draw\_circles in a way like we did to draw the square figure. This method called draw\_circles\_within, would need a number of recursions, a center and a radius. Using the circle method, the first circle was drawn. Then, five more centers were calculated, for the top and bottom circles the x coordinate stayed untouched, but for the top we subtracted the y coordinate plus one third of the radius from the y coordinate, and for the bottom we subtracted one third of the radius minus the y coordinate from the negative radius. For the left and right center, we left the y coordinates untouched, but for the left we added the radius minus one third of the radius to the x coordinate. For the left we subtracted the radius minus one third of the radius from the x coordinate

**Experimental results**

**Bubble sort**

Using the recursive method quicksort, giving a random linked list it would sort it in following the algorithm correctly and it would return the list appropriately like it is supposed to.

**Merge sort**

Unfortunately this method did not work how it supposed to. When you give it a list bigger than two nodes the list is split up appropriately but it is not merged correctly. If the node is larger than the node being compared to the method merge did return them correct, but if it was larger only one node would be returned

**Quick sort**

Unfortunately I could not get this method to work either, it would only define a pivot and an index but would not sort. I believe the recursive calls are wrong. It also need a way for the index to iterate through the list. I ha d a hard time doing this with a Linked list.

**Conclusion**

A big thing in this lab was learning how to manipulate linked list, something I think I still have yet to master. Since I was not able to finish the methods I think I need to review the algorithm in order for me to be able to code it correctly .

**Source code.**

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

'''

Created by: Fernando Zaldivar

Last Modified: Feb 22, 2019

CS 2302

Proffesor Dr. Olac Fuentes

'''

#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()

#This method will take a number, and make a Link list of the length of that number using random integers

import random

def randomLinkedList(n):

L = List()

for i in range(n):

Append (L, random.randint(0,100))

return L

# Takes a List and sorts it using bubble sort algorithim

def bubbleSort(L):

done = False

while done is not True:

temp=L.head

done = True

while temp.next is not None:

if temp.item>temp.next.item:

tempItem = temp.next.item

temp.next.item = temp.item

temp.item =tempItem

done = False

temp=temp.next

return L

#Takes in a list and return the amount of nodes

def countNodes(L):

temp = L.head

count =0

while temp is not None:

count +=1

temp= temp.next

return count

#takes list return the middle node

def getMidNode(L):

length = countNodes(L)

temp=L.head

for i in range (length//2):

temp=temp.next

i+=1

return temp

#take a list iterates through the first half then pionts it None so it only returns the first half

def getfirstHalf(L):

if L.head is not None:

length = countNodes(L)

temp=L.head

for i in range ((length//2)-1):

temp=temp.next

i+=1

temp.next= None

return L

#take a List and return the last haalf by finding the mid node and making it head

def getSecondHalf(L):

L.head=getMidNode(L)

return L

#takes one list and return another identical to it

def copyLinkedList(L):

temp=L.head

NewL = List()

for i in range(countNodes(L)):

Append (NewL, temp.item)

temp=temp.next

return NewL

#take a List and merge sorts it

def mergeSort(L):

temp=copyLinkedList(L)

if countNodes(L)>1:

L1 = getfirstHalf(temp)

L2 = getSecondHalf(L)

mergeSort(L1)

mergeSort(L2)

L=merge(L1,L2)

#takes two lists and merge them together by ordering them

def merge(L1,L2):

temp1=L1.head

temp2=L2.head

NL=List()

while temp1 != None and temp2 != None:

if temp1.item< temp2.item:

Append(NL,temp1.item)

temp1 = temp1.next

else:

Append(NL,temp2.item)

temp2 = temp2.next

if temp1 is None:

while temp2 is not None:

Append(NL,temp2.item)

temp2 = temp2.next

if temp2 is None:

while temp1 is not None:

Append(NL,temp1.item)

temp1 = temp1.next

return NL

#return the median of a sorted list

def getMedian(L):

middle = countNodes(L)//2

temp=L.head

for i in range (middle):

temp=temp.next

i+=1

return temp.item

#return the int at a certain poin in a list

def itemAt(L,n):

temp = L.head

for i in range (n-1):

temp=temp.next

i+=1

return temp.item

#takes a list and sorts through it using quick sort

def quickSort(L):

temp = L.head

print(L.tail.item)

pivot = L.tail.item

i = temp.item

print ('i= ',i)

if i < pivot:

temp=temp.next

i=temp.item

print ('i= ',i)

if i>pivot:

tempP= pivot

#pivot = itemat(L,countNodes(L)-some number)

L=randomLinkedList(20)

Print(L)

L= bubbleSort(L)

Print(L)

median = getMedian(L)

print('median: ' ,median)

mergeList = randomLinkedList(4)

Print (mergeList)

mergeSort(mergeList)

Print (mergeList)

mergeMedian = getMedian(mergeList)

print ('merge median',mergeMedian)

quickList = randomLinkedList(4)

Print (quickList)

quickSort(quickList)