**Practice for Mid-Term Test**

**Using Fundamental Data Structures (Arrays, Linked Lists), Stacks, and Queues**

References: Chapter 3,4,5,6 lecture slides, lab assignments. This material provides the necessary information that you need to complete the exercises.

**Exercise 1**

Write a method for **concatenating two singly linked lists L1 and L2**, into a single list **L3** that contains all the nodes of **L1** followed by all the nodes of **L2**.

**Hint**: Traverse list L1 until you reach the last element. Then, make the last element of L1 point to the first element of L2 as its “next” node.

Let’s use the SinglyLinkedList implementation from Lesson 2 slides. Do not use the tail.

head

TO

LA

BO

VA

Ø

head

TX

CI

AT

ST

Ø

We need to traverse the list L1 to reach the last node. Then, we need to link the last node of L1 to first node of L2. This is easy, knowing that *head* of L2 already is the required link. The code below creates the concatenate method inside SinglyLinkedList class:

//concatenate two singly linked lists

//input: L1, L2 are singly linked lists

//output: the concatenated version

**public** <E> SinglyLinkedList<E> concatenateLists(SinglyLinkedList<E> L1,

SinglyLinkedList<E> L2)

{

//Create a new node v

Node<E> walk = **new** Node<E>(**null**, **null**);

walk = L1.head; //point to head of L1

//traverse L1

**while**(walk.getNext() != **null**)

walk = walk.getNext();

//link to header of list L2

walk.setNext(L2.head);

//More Efficient Approach would be

L1.tail.setNext(L2.head);

L1.tail=L2.tail

//return the concatenated list

**return** L1;

}

//

**public** **static** **void** main(String[] args)

{

SinglyLinkedList<String> list1 = **new** SinglyLinkedList<String>();

list1.addFirst("MSP");

list1.addLast("ATL");

list1.addLast("BOS");

SinglyLinkedList<String> list2 = **new** SinglyLinkedList<String>();

//

list2.addFirst("YYZ");

list2.addLast("MTRL");

list2.addLast("OTW");

System.***out***.println(list1);

System.***out***.println(list2);

//

System.***out***.println(list1.concatenateLists(list1, list2));

System.***out***.println(list1.secondToLast().getElement());

}

Python version

# -\*- coding: utf-8 -\*-  
  
  
class Node:  
 def \_\_init\_\_(self, element, next\_node=None):  
 self.element = element  
 self.next\_node = next\_node  
  
class SinglyLinkedList:  
 def \_\_init\_\_(self):  
 self.head = None  
 self.tail = None  
 self.size = 0  
  
 def \_\_len\_\_(self):  
 return self.size  
  
 def is\_empty(self):  
 return self.size == 0  
  
 def first(self):  
 if self.is\_empty():  
 return None  
 return self.head.element  
  
 def last(self):  
 if self.is\_empty():  
 return None  
 return self.tail.element  
  
 def add\_first(self, e):  
 newest = Node(e, next\_node=self.head)  
 self.head = newest  
 if self.is\_empty():  
 self.tail = self.head  
 self.size += 1  
  
 def add\_last(self, e):  
 newest = Node(e)  
 if self.is\_empty():  
 self.head = newest  
 else:  
 self.tail.next\_node = newest  
 self.tail = newest  
 self.size += 1  
  
 def remove\_first(self):  
 if self.is\_empty():  
 return None  
 answer = self.head.element  
 self.head = self.head.next\_node  
 self.size -= 1  
 if self.is\_empty():  
 self.tail = None  
 return answer  
  
 def \_\_eq\_\_(self, other):  
 if not isinstance(other, SinglyLinkedList) or self.size != len(other):  
 return False  
  
 node1, node2 = self.head, other.head  
 while node1 is not None:  
 if node1.element != node2.element:  
 return False  
 node1, node2 = node1.next\_node, node2.next\_node  
  
 return True  
  
 def \_\_str\_\_(self):  
 result = []  
 node = self.head  
 while node is not None:  
 result.append(str(node.element))  
 node = node.next\_node  
 return "(" + ", ".join(result) + ")"  
#update this concatenate by creating a clone in list1 in list and that way code remains efficient and list 1 is not updated  
 def concatenate(self, list2):# set Next of Tail of list 1 to point to head of list2 and update tail  
 self.tail.next\_node=list2.head  
 self.tail=list2.tail  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 list1 = SinglyLinkedList()  
 list1.add\_first("MSP")  
 list1.add\_last("ATL")  
 list1.add\_last("BOS")  
 print(list1)  
 list2=SinglyLinkedList()  
 list2.add\_first("YYZ")  
 list2.add\_last("YVR")  
 print(list2)  
 list1.concatenate(list2)  
 print(list1)

**Exercise 2**

Write a method for finding **the second-to-last node** in a singly linked list in which the last node is indicated by a null next reference.

tail

head

TX

CI

AT

ST

Ø

Again, we need to traverse the list. However, this time we stop at the node before the last node. Note the use of getNext() method two times in the loop condition, to ensure that we reach the second-to-last node. Add this method to SinglyLinkedList class.

//find the second-to-last node in a list

**public** Node<E> secondToLast()

{

//make sure the list has at least two nodes

**if**(size<2) **throw** **new** IllegalStateException("list must have 2 or more entries");

//navigate

Node<E> walk= head;

**while**(walk.getNext().getNext()!=**null**)

{

walk = walk.getNext();

}

**return** walk;

}

//main method

//main method

**public** **static** **void** main(String[] args)

{

SinglyLinkedList<String> list1 = **new** SinglyLinkedList<String>();

list1.addFirst("MSP");

list1.addLast("ATL");

list1.addLast("BOS");

SinglyLinkedList<String> list2 = **new** SinglyLinkedList<String>();

//

list2.addFirst("YYZ");

list2.addLast("MTRL");

list2.addLast("OTW");

System.***out***.println(list1);

System.***out***.println(list2);

//

System.***out***.println(list1.secondToLast().getElement());

}

Python solution

def secondToLast(self):  
 walk=self.head  
 while(walk.next\_node.next\_node!=None):  
 walk=walk.next\_node  
 return walk  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 list1 = SinglyLinkedList()  
 list1.add\_first("MSP")  
 list1.add\_last("ATL")  
 list1.add\_last("BOS")  
 print(list1)  
 print(list1.secondToLast().element)

**Exercise 3**

Write a short recursive Java method that takes a character string **s** and **outputs its reverse**. For example, the reverse of 'pots&pans' would be 'snap&stop'

Let ***n*** be the index of last character in a string ***s***. For example, if s="ab", then n=1. String method *charAt (int index),* returns the character given its index in a string.

The statement **s.charAt(n)** will return character ’b’. The statement **s.charAt(n-1)** will return “a”. You can see how this can become a recursive call. The **stopping point is n = 0**. See the code below:

**public** **class** ReverseString {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

String s= "COMP";

*reverse*(s);

}

**public** **static** **void** reverse(String s)

{

*reverse*(s, s.length()-1); //reverse it

}

//input: a string and the index of the last character in it

//output: reversed string

**public** **static** **void** reverse(String s, **int** n)

{

//stopping condition

//if( n<0 ) throw new IllegalStateException("String must have one or more chars");

**if**(n>=0)

{

System.***out***.println(s.charAt(n));

//

*reverse*(s,n-1);// recur

}

}

}

Python version

def reversingstring(param):  
 if (param == ""):  
 return ""  
 return reversingstring(param[1:]) + param[0]  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 s="COMP"  
 print(reversingstring(s))

**Exercise 4**

Let A be an array of size **n** containing integers from **1 to n-1 inclusive**, one of which is repeated. Design an algorithm for **finding the integer in A that is repeated**.

**public** **class** RepeatedInt {

**public** **static** **void** main(String[] args) {

// // integers from 1 to n-1 inclusive, one of which is repeated

**int** a[] = {1,2,3,3,4,5};

System.***out***.println(*findRepeatedInt*(a));

}

/\* Let A be an array of size n containing integers

\* from 1 to n-1 inclusive, one of which is repeated.

\* Describe an algorithm for finding the integer in A that is

\* repeated.

\*/

**public** **static** **int** findRepeatedInt(**int**[ ] A) {

**boolean**[ ] found = **new** **boolean**[A.length]; // all elements false, by default

**for** (**int** val : A)

**if** (found[val])

**return** val;

**else**

found[val] = **true**;

**return** -1; // shouldn't happen if input as expected

}

}

def print\_hi(A):  
 count={}  
 for i in A:  
 if not i in count:  
 count[i]=1  
 else:  
 count[i]=count[i]+1  
 print(count)  
# Press the green button in the gutter to run the script.  
if \_\_name\_\_ == '\_\_main\_\_':  
 A = [1,2,3,1,2,3]  
 print\_hi(A)