COMS 228: Introduction to Data Structures, Summer 2013
Homework Assignment 3

June 12, 2013

Assignment Overview

This assignment will give you practice implementing doubly-linked lists, list iterators and using different backing stores for implementing doubly linked lists. For this assignment, we have provided an interface "CS228DoublyLinkedList", you will be responsible for implementing this interface in **two** different ways: using an array-based backing store and a node-based backing store. You MAY NOT implement the List interface by extending "AbstractCollection", "AbstractList", or any other class. They must be written from scratch. You do not need to write a main method, and it is not a requirement to write a client class. You simply need to write the entire implementation for our CS228DoublyLinkedList and ListIterator interfaces.

Specification for CS228DoublyLinkedList

First we will give you a brief overview of the CS228DoublyLinkedList interface. CS228DoublyLinkedList is a generic class that has methods

- public void add(E)
- public boolean add(int, E)
- public void clear()
- public boolean contains(E)
- public boolean empty()
- public E get(int)
- public ListIterator<E> iterator()
- public ListIterator<E> iterator(int)
- public E remove(int)
- public boolean remove(E)
- public in size()

- public E set(int,E)

These methods are well documented in the provided interface's Javadoc and Java Documentation.

In addition, you will implement special methods described below

- public void removeDuplicates()

This method will remove duplicate elements and remove duplicates from the list and retain the first instance of an element. That is, given a list [1,2,3,1], the list will now be [1,2,3].

- public CS228DoublyLinkedList<E> uniqueElements(CS228DoublyLinkedList<E> list)

This method will take in another list and return a list with elements NOT in common. That is, given a list A =[1,2,3,4] and a list B=[4,5,6], the resulting list will be [1,2,3,5,6].

- public CS228DoublyLinkedList<E> intersectionLists(CS228DoublyLinkedList<E> list)

This method will take in another list and return a list with elements in common. That is, given a list A =[1,2,3,4] and a list B=[4,5,6], the resulting list will be [4].

- public void append(CS228DoublyLinkedList<E> list);

This method will append the elements of the given list to the end of this list. That is, given a list A =[1,2,3,4] and a list B=[4,5,6], the resulting list will be [1,2,3,4,4,5,6].

- public CS228DoublyLinkedList<E> filter(Filter<E> f)

This method will return a list given a certain filter condition. Filter is an interface with only one method *boolean accept(E)*. Say we have a filter that returns true when an element is even and list A=[1,2,3,4,5,6], the result of calling **filter** on our list will be [2,4,6].

Specification for ArrayBasedDoublyLinkedList

This class will implement *CS228DoublyLinkedList* and will use an array as a backing store. When an ArrayBasedDoublyLinkedList is instantiated, the initial array size should be 10. When the array is full, double the size of the backing-store array. Need some guidance here.

Specification for NodeBasedDoublyLinkedList

Your objective is to write a generic doubly linked list class called NodeBasedDoublyLinkedList that will implement *CS228DoublyLinkedList* and uses a type variable E.

The NodeBasedDoublyLinkedList class contains a non-static inner class called Node. Obviously, the instances of Node serve as nodes in the linked list. The Node inner class contains six member variables: data of type T, next of type Node, prev of type Node. You should not declare any other member variables for the Node inner class. As we learned, the next field of a node refers to the next node in the list. If the next node does not exist, then next is null. The prev field refers to the previous node in the list. If the previous node does not exist, then prev is null. Note: We will consider two nodes to be equal as long as they contain the same data.

Along with normal nodes, the NodeBasedDoublyLinkedList class needs to include two dummy nodes called head and tail. We consider the list to be empty if it contains only the head and tail nodes. Logically, if the list is empty, then the node next to head is tail, and the node previous to tail is head. Otherwise, the head is previous to the first normal node, and the first normal node is next to the head. Likewise, tail is next to the last normal node, and the last normal node is previous to the tail. The head and tail dummy nodes must have null data fields. Also, the prev field of head and the next field of tail must be null.