Homework # 13 v1.2
due Monday, May 8, 10:00 PM

In this assignment you will implement a spell-checker using a SortedList class that uses merge sort when doing bulk operations (adding or removing multiple elements at once). You will implement operations using nodes in a cyclic linked list.

1 Concerning the SortedCollection ADT

We provide an implementation of the standard Collection ADT using a singly-linked cyclic list with a dummy node. We use a single “tail” pointer to get access to the tail and (by way of the cycle) to the head of the list. We build on the AbstractCollection abstract class which means there are few methods that need to be implemented. The iterator class is provided as well. You need to implement the add, addAll, removeDuplicates and removeAllNew methods (as well as the helper methods listed below). The add operation should be constant time if the new element belongs at the beginning or end. The other two methods should first sort the parameter collection (if any) and then use a linear-time algorithm to perform the add/remove operation.

2 Concerning helper methods for mergesort

Our helper algorithms will use SLLs: null terminated singly-linked lists without dummy nodes. A SLL can be empty (null). Some of the helper methods take a rev flag, then if true, reverses the sense of the comparator.

As described in the textbook, mergesort is a divide-and-conquer algorithm: to sort a list, we divide it into two pieces and recursively sort the pieces and then merge the two sorted sublists. If the list is “very small” (say zero or one elements), sorting is immediate: you don’t need to do anything.

With linked lists, however, it’s awkward to produce a list from the front, because you keep on needing to patch the previous node, which causes extra conditions (“if” statements!). Thus the main merge helper method (revMerge) produces the reverse of the merge of the two lists. Then when working with the reverse lists, we need to sometimes reverse the sense of the comparator.

For this homework, you implement three of the four helper methods:

toSLL(col) Create a singly-linked list in reverse order from a collection. This helper method is already implemented for you, shows how easy it is to create a list in reverse order, and will be useful in implementing other methods of this class.

reverse(l1,l2) This method destructively (using the same nodes) reverses the first SLL and places it in front of the second SLL.

revMerge(l1,l2,rev) This method takes two sorted SLLs and merges them destructively by putting all the nodes of the two SLLs together in one result which is built in reverse order.

If the rev flag is set, the input lists are sorted in reverse order and the comparator should be used in negated fashion. But in all cases, the result uses the opposite ordering of the arguments.

mergeSort(l,count,rev) This method takes a SLL and sorts it destructively and returns a sorted SLL using the same nodes (that’s what it means to be destructive). The count is the number of nodes in the SLL. Since the caller provides the count, we don’t need to first count how many elements there are. If the rev flag is set, then the elements should be sorted in the opposite order to what the comparator normally says; in other words, the comparator’s result should be negated.

If the SLL has fewer than two elements, the method returns it unchanged. Otherwise, it splits the list into two (using count to know where to split), recursively sorts the two lists in the opposite order, and then uses the revMerge helper method to merge the results of sorting the sublists in the reverse order. You should not need to call the reverse method directly.
All the helper methods use the comparator of the enclosing class. Each helper method may use the previous ones, so make use of the unit tests to make sure reverse is working before implementing revMerge and only once the tests for “merge” have been passed, go on to mergeSort.

3 Concerning removeDuplicates

This method should go through a sorted collection, and remove consecutive elements that are equal, according to the comparator, not according to equals (why?). This method is easily implemented with an iterator.

4 Concerning removeAll

The standard collection classes provide removeAll method that takes a collection of objects. By default, this method simply iterates through one collection while looking up every element in the other collection and removing the first one if it is contained in the second. Since cyclic lists, even when sorted, cannot support binary search, the lookup operation will always be slow. We need “remove all” functionality however to support our spell-check application, thus we implement a faster version of removeAll described here.

You will implement removeAllNew which takes a collection of the same type. You should first sort the elements (using the receiver’s comparator) in a new loop. Then you should go through both lists as once, making use of the fact that both lists are sorted. If we find an equal element in the first list we remove it and go forward with the second list. Otherwise, we move forward in either the first or second list. In all three cases, if we find ourselves about to loop around the cycle, the operation is complete. You may find using the (already implemented for you!) iterators very useful. In particular, they handle the removal of the tail node (which might be the very last node!).

Before doing the work described here, it is best to handle the case that either this list or the parameter collection is empty. Getting the empty cases out of the way makes the main body of the algorithm easier.

5 Concerning the Spell Checker

We provide a main program SpellCheck that uses the SortedCollection ADT to spell-check a file. You can check a speech of President Lincoln for spelling errors. (It gives many (88) errors because our spell checker doesn’t know about plurals or common suffixes such as -ly.)

6 Files

The repository for this assignment includes the following files:

src/edu/uwm/cs351/SortedCollection.java The partially complete cyclic list class.

src/TestSortedCollection.java Unit tests for your work.

src/TestEfficiency.java Also efficiency tests. These take a bit longer than usual, but print out messages every second or so.

lib/dictionary.txt List of correctly-spelled words.

lib/lincoln2.txt President Lincoln’s second inaugural address: Sample document to spell-check.

7 What you need to do

You need to implement seven methods in class SortedCollection: reverse, revMerge, mergeSort, add, addAll, removeDuplicates, and removeAllNew.