Lab: Extending TreeSets
CSC 207, “Algorithms and Object-Oriented Design”
Department of Computer Science
Grinnell College
November 19, 2018

Setup

Today’s lab is based on Weiss’s implementation of the TreeSet class in
/home/hajiamini/courses/CSC207/code/TreeSet/TreeSet.java

1. In Eclipse, open a new project and copy Weiss’s definition of the TreeSet class into a
TreeSet.java file within that project. Remove the package declaration in the first line and add
an import directive that brings in all of the classes in the java.util package.

Weiss’s definition of the TreeSet class does not implement all of the methods that are required
by the java.util.SortedSet interface. In particular, it lacks the subSet, headSet, and tailSet
methods. Each of these methods is supposed to return a SortedSet<AnyType> result—a view
of the TreeSet that blocks access to (and modification of) parts of the underlying aa-tree whose
elements lie outside a specified range. The headSet and tailSet methods take single AnyType
values, serving respectively as exclusive upper bounds and inclusive lower bounds on the range to
be included, while subSet takes two AnyType values (the inclusive lower bound and the exclusive
upper bound).

2. For the time being, implement placeholder methods with these names and signatures that
throw an UnsupportedOperationException when invoked. This will enable you to work on the
TreeSet class without Eclipse pestering you about the incompleteness of the implementation.

Adding Floors and Ceilings

Since the elements in a SortedSet are ordered, it is possible to take any value \( v \) of the type to
which the elements belong and search for an element of the set that comes as close as possible to
\( v \) without exceeding it. If \( v \) itself is an element of the set, we want \( v \) itself as the result; otherwise,
we want the element that would immediately precede \( v \) if we were going to add \( v \) to the set. Let’s
call this element \( \lfloor v \rfloor \), the “floor” of \( v \).

3. Add to the TreeSet class a new method, floor, that takes a value of type AnyType as
argument and returns the floor of that value, if the TreeSet contains such an element, or a
null reference if it does not.

4. Add a ceiling method that takes any value of type AnyType as argument and returns that
value, if it is in the TreeSet, and otherwise the element that would immediately follow it.

Adding Views

To implement the headSet, tailSet, and subSet methods, we’ll need to provide views of the
underlying TreeSet. In these views, we’ll override the methods of the underlying TreeSet with
replacements that reflect only the elements that are within the specified range.

5. Define an inner class, TreeSetView, within TreeSet. The TreeSetView class should im-
plement the SortedSet interface, include any additional fields that will be needed in appropriate
views of the enclosing TreeSet, and implement any necessary methods, calling the corresponding
methods of the enclosing TreeSet as needed.

6. The TreeSetView class requires an iterator that views the underlying set. Implement an
iterator class, TreeSetViewIterator, that extends TreeSetIterator. Inside the TreeSetView
class, provide an implementation for the iterator method. This method returns an instance of the TreeSetViewIterator class.

7. The TreeSetViewIterator class should iterate through the underlying set within a fixed range. The upper bound and lower bound of this range should be defined in the TreeSetView class. These bounds need to be set properly based on a particular view of the underlying set. For example, for the tailSet view, the lower bound is the first element of the set and the upper bound is the element computed by the ceiling method.

Define two fields in the TreeSetView class, representing the lower and upper bounds, and initialize them through the TreeSetView class constructor. After passing the correct bounds to the constructor, the headSet, tailSet, and subSet methods return objects of the TreeSetView class to the corresponding methods in the TreeSet class.

8. Define the headSet, tailSet, and subSet methods by constructing and returning objects of the TreeSetView class subject to the appropriate constraints (passing the proper lower and upper bound values to the TreeSetView constructor).