Homework # 9
due Monday, November 11, 10:00 PM

In the previous assignment, we implemented a RasterSet ADT using a binary search tree (BST), but left removal undone, and didn't implement the iterator. In this assignment, we will extend the AbstractSet class, and we will add parent pointers to our nodes to facilitate the iterator operations as well as removal of elements. Furthermore, we implement a BST-SortedSet class, which requires us to iterate in sorted order. A BST makes sorted order fairly easy to derive. It also includes a method called `getNth`, and we add `treeSize` fields to our nodes to make this easier.

1 Concering the Updated BST Data Structure

Our nodes have two new fields: `parent`, holding a reference to the node's parent (or null) and `treeSize`, storing the size of the subtree rooted at that node. The size stored at the root node should therefore be the number of nodes in the tree. We therefore remove the now redundant `manyNodes` field.

Any method that changes the data structure must now update these fields appropriately. We must also check them in our invariant checker. Pay attention to which fields need to change when a node is added or removed especially. Draw pictures!

The `treeSize` field is primarily used to support the `getNth` method. This method returns the nth element in sorted order. Just like other BST operations, we would like to achieve this in logarithmic time. Including this new field allows this. As an example, if we are looking for the 10th elementh, and the left subtree has 32 nodes, we can look for the 10th element in the left subtree. If we are looking for the 35th element, than we can instead look for the 2nd element in the right subtree. (32 in left subtree, 33rd in root, rest in right subtree) This can be done recursively.

Lastly, we no longer use a comparator. Instead, although the class is generic, we only allow data types that implement Comparable to be stored in this ADT. This means they will have a built-in `compareTo` method that can be used for the same purpose. But this does mean only one ordering per data type is possible: the natural ordering given by `compareTo`.

2 Concerning the Iterator Data Structure

We will be using the "Parent Pointers" technique for iterating over the binary search tree, as explained in the Navigating Trees handout.

The iterator data structure will consist of a `nextNode` reference, and a `hasCurrent` field, as well as `myVersion` for fail-fast semantics. At any time, it has a reference to the next node to be visited, but that has not yet been iterated over. This means the iterator's constuctor has some non-trivial work to do. As the ADT requires sorted order, we must use an in-order traversal when iterating over the nodes of the tree. Because the iterator does not explicitly maintain the current element, we have to look backwards through the tree to find it when `remove` is called.

The iterator's `wellFormed` will call the outer classes `wellFormed` method, using *Class*`.this.`*method*`(...)`. After checking the outer class' invariant, the iterator will see if it (the iterator) is stale. if the iterator is stale, then any problems with its representation may be due to inconsistent changes in the main class which are *not* the responsibility of the iterator class. Recall that we check the data structure to protect against errors of *implementation*. A stale iterator shows errors of *use*, not implementation. Make sure that the developer is not blamed for errors by the client!

3 Concerning Removal

If the node being removed has no left child, then its right child (if any) can be used to replace this node entirely. If it has a left child and no right child, the left child can replace this node. If it has two children, we can copy the data from the immediate predecessor (largest smaller descendant), and then remove that predecessor.

In all cases, if an iterator is doing the removing, its reference to the appropriate next node must not be disturbed. Pay attention to where the next element is, and where the `nextNode` reference is pointing. These things should still correspond after removing. Don't break the iteration! Draw pictures to help you handle the replacement correctly.

Collections have two ways to remove elements: one through a method of the collection class and one through the iterator. Removal code is complex and thus it would be poor software construction to implement this capability twice. You should have both methods ultimately call the same code, perhaps using a shared private helper method.

4 The Test Suite

We provide a full complement of tests:

TestBSTSortedSet Our ADT test suite.

TestInternals Tests invariants for the BST and the iterator.

TestEfficiency Updated efficiency tests. Each test should take no more than a few seconds.

5 What you need to do

You need to implement all unimplemented methods in BSTSortedSet You may begin using code from Homework 8, but you will have to account for the new fields, removal, the iterator, and `getNth`.