Assignment 3
COMP 250, Fall 2021

Prepared by Prof. Michael Langer and T.A.s
Posted: Fri. Nov. 5, 2021
Due: Fri. Nov. 19 at 23:59 (midnight)

[document last modified: November 8, 2021]

General instructions

- We are provided you with a scaffold to build your code on, i.e. starter code. You should download the scaffold first and then start coding. As with Assignments 1 and 2, you can find this code either your workspace on Ed or on mycourses.
- Search for the keyword *updated* to find any places where this PDF has been updated.
- T.A. office hours will be posted on mycourses under the "Office Hours" tab. For zoom OH, if there is a problem with the zoom link or it is missing, please email the T.A. and the instructor.
- If you would like a TA to look at your solution code outside of office hours, you may post a *private* question on the discussion board.
- **[Updated Nov. 8]** We have provided you with some examples to test your code. (See TestExposedA3.java on mycourses. The same tests are run on Ed when you submit.) If you pass all these exposed tests, you will get 59/100. *Unlike with Assignments 1 and 2, the remaining tests will be private.* Therefore, we strongly encourage you to come up with more creative test cases, and to share these test cases on the discussion board. There is a crucial distinction between sharing test cases and sharing solution code. We encourage the former, whereas the latter is a serious academic offense.
- **Policy on Academic Integrity:** See the Course Outline Sec. 7 for the general policies. You are also expected to read the posted checklist PDF that specifies what is allowed and what is not allowed on the assignment(s).
- **Late policy:** Late assignments will be accepted up to two days late and will be penalized by 10 points per day. If you submit one minute late, this is equivalent to submitting 23 hours and 59 minutes late, etc. So, make sure you are nowhere near that threshold when you submit.

Please see the submission instructions at the end of this document for more details.

Tertiary Search Tree (TST)

The purpose of this assignment is to give you some experience with recursion. Recursion is a fundamental method in programming and the sooner you learn it, the better! The topic for this assignment is trees.

You will work with a data structure that we will call a *tertiary* search tree or TST.[1] This is like a binary search tree (BST), except that each node in a TST has three children rather than two. We can call the three children left, middle, and right. Each node stores a element whose class type implements Comparable, so you can assume access to a compareTo() method. The subtrees defined by the left, middle, and right children of a node have elements that are less than, equal to, or greater than the element at that node, respectively.

Lecture 25 gave pseudocode for binary search trees (BST) methods. You will implement similar methods for your TST class. Make sure you understand the BST pseudocode before you try to implement a TST version!

You will work with three classes:

TSTNode class (25 points)

A TSTNode has four fields: an element, and three children (left, middle, right). For this class, you will write:

- a constructor TSTNode(T element) **(5 points)** - assigns the given element to this node; the children are automatically initialized to null.
- height() **(10 points)** – returns an int which is the height of the subtree, whose root is this TSTNode
- findMin() **(5 points)**– returns the TSTNode containing the minimum element in the tree; you can use this as a helper method.
- findMax() **(5 points)**– return the TSTNode containing the maximum element in the tree; you can use this as a helper method.

You may add your own helper methods to the TSTNode class. For example, you may wish to overload the constructor.

TST class (55 points)

A TST has just one field: the root TSTNode of the tree. A TST has several methods that are provided to you:

- height() – returns an int which is the height of the tree; it depends on the height() helper method in the TSTNode class (see above)
- toString()– toString() can be used to visualize the tree structure; recall that you can 'call' the toString() method using System.out.print(tree) where tree is of type TST;

[1] This is not to be confused with a "ternary search tree" which is something else. Also, others may have used the term "tertiary search tree" to mean something other than what we mean here. We will ignore these other usages of the term.

- stringify(...) – a helper method used by toString()
- inorderPrintAsList() – prints the elements in order using an enhanced for loop; the enhanced for loop implicitly uses an Iterator, and so you will need to implement the Iterator (see below) before you can use this print method;

Write the following methods for the TST class. Hint: you may find it easier to write some of these methods if you write helper versions which have a TST Node (root) parameter. These helper methods can be either in the TST class or in the TSTNode class.

- insert(T element) **(10 points)** – void method; it creates a new node in the tree and assigns the element of that node; the node should be placed in the correct position; duplicates are allowed.
- contains(T element) **(5 points)** – returns a boolean (true if the tree contains a node with this element, and false otherwise)
- iterator() – returns an TSTIterator for the TST. See the TSTIterator class below. This will be evaluated as part of the tests for TSTIterator.
- rebalance() **(20 points)** – void method; specifications are as follows.

  Just as a BST can become imbalanced as discussed in the lecture, so can a TST. Methods for balancing BST's are covered in COMP 251. For simplicity, we consider the following simple rebalancing method that uses concepts from COMP 250 only.

  We define rebalance() for a TST as follows. Traverse the tree in-order and make an arraylist of the sorted elements. If there are duplicates (multiple instances of the same element), then include them all in the list. Next, recursively partition the list similar to what is done in binary search: consider the element at the middle position $size/2$; this element will be the one at the root node of the rebalanced TST; all elements that are less than, equal to, or greater than the root element will be stored in the left, middle, or right child subtree, respectively. You may use the List.subList() method here.

  Note that this is not necessarily going to give a "balanced" tree in the intuitive sense of balance. For example, if the tree has elements 1, 1, 1, 1, 5, 5, 5, 5, 5, 5 then the "rebalanced" tree would have element 5 at the root node, the root would have no right child, and the root node's left subtree would have height 3. For TST's that have relatively few duplicates, the rebalance() method would do a better job of rebalancing.

- remove(T element) **(20 points)** – returns nothing (void); it finds a node containing this element and removes it; if there are duplicate elements, then exactly one instance must be removed; which one to remove is up to you, and indeed we have no way to distinguish which one you remove without manually examining your code;

  To test correctness of the remove method, we need a policy for the case that only one instance of the element to be removed is present and the node containing that element has both left and right children. In principle, either of two policies could be used to replace the element at that node: take the largest element in the left subtree, or take the smallest element in the right subtree. However, for grading, we want a consistent policy for everyone. We require that you take the largest element of the left subtree. Note that this is the opposite remove policy of what was used in the BST lecture slides: we use the opposite policy to make an important point that sometimes two policies are equally good and yet one needs to use just one in the specification.

TSTIterator class (20 points)

This class implements Iterator. Write the following methods:

- a TSTIterator constructor
- hasNext()
- next()

The constructor should create an ordered list of all the elements in the tree. The hasNext() and next() methods then use this list. For grading, we won't test the constructor directly. Rather we will test that the Iterator works as it should, namely iterating through the list should yield the elements in their proper order.

Note: since TST implements Iterable, you can use the Java enhanced for loop, once you have successfully implemented the TSTIterator class.

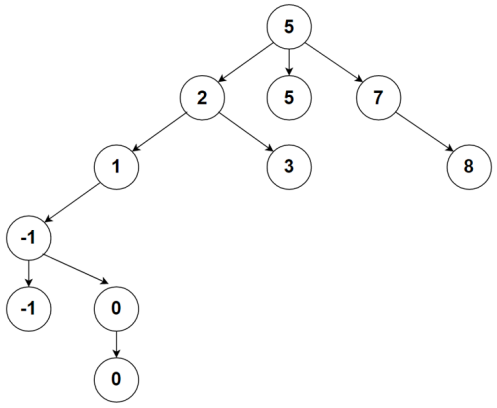
Where to start? How to proceed?

We suggest that you implement and test in the following order:

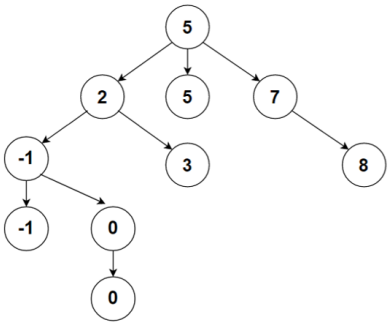
1. TSTNode constructor
2. TST insert() and contains()
3. TSTNode height(), findMin(), findMax()
4. TST iterator()
5. TST rebalance()
6. TST remove()

Example

```
TST<Integer> tree = new TST<Integer>();
tree.insert(5);
tree.insert(5);
tree.insert(2);
tree.insert(7);
tree.insert(8);
tree.insert(1);
tree.insert(-1);
tree.insert(-1);
tree.insert(3);
tree.insert(0);
tree.insert(0);
```

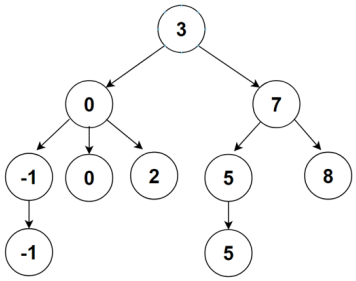



```
tree.remove(1);
```



[Updated Nov. 8] We modified the figure above which shows the effect of remove(1). Note if 1 had a right child before the remove(1) call, then we would have chosen 0 as the root node of the new subtree (since 0 was the largest element in the left subtree of 1).

```
tree.rebalance();
```



Submission

Please follow the instructions on Ed Lessons Assignment 3.

- Ed does not want you to have a package name. You can avoid having package name by developing and testing your code in your project's default package. You should be able to drag your files from this package into Ed.
- We will check that your file imports only what you need, namely List, LinkedList or ArrayList (or both), Iterator, all from java.util. If you use other imports, your code will not pass this check and you will not be able to submit.
- You may submit multiple times. Your assignment will graded using your most recent submission.
- If you submit code that does not compile or run, you will automatically receive a grade of 0. Since we grade only your latest submission, you must ensure that your latest submission compiles and runs.
- The deadline is midnight Friday, Nov. 19. On Ed, this deadline is coded as 12:00 AM on Saturday Nov. 20. (The reason is that Ed will mark anything submitted at 11:59 PM as late.) Similarly, on Ed, the 2 day window for late submission ends at 12:00 AM on Monday Nov. 22.

Good luck and happy coding!