CS 146

Data Structures and Algorithms

Summer Semester 2015

Department of Computer Science
San José State University
Instructor: Ron Mak

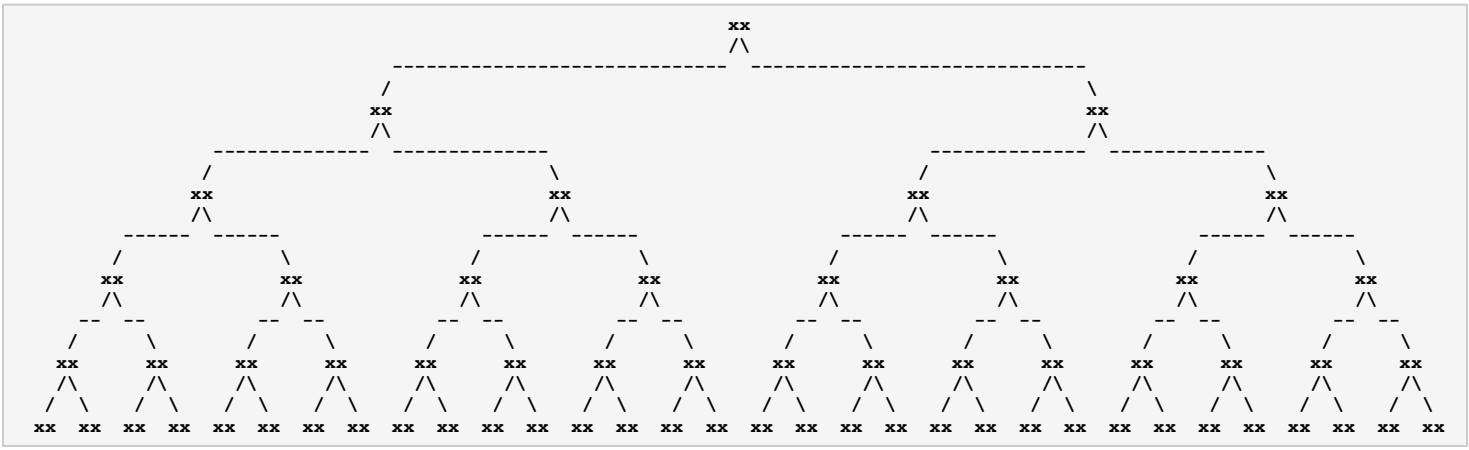
Homework #3
Binary Search Trees and AVL Trees

Assigned:	Tuesday, June 23
Due:	Friday, July 3 at 11:59 pm
	100 points max

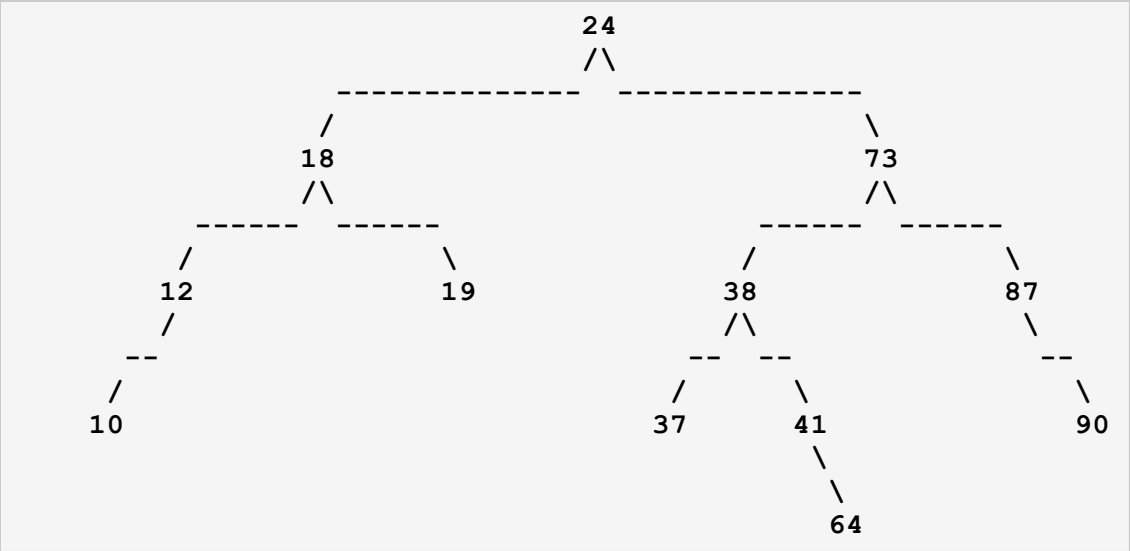
The purpose of this assignment is to give you practice with binary search trees (BST) and AVL trees. It has two parts.

A tree printer

You are provided a **TreePrinter** class that has a **print()** method that will print any arbitrary binary tree. A template for how it prints a tree:



TreePrinter is able to print trees with height up to 5, i.e., 32 node values on the bottom row. An example of an actual printed tree:



Part 1

The first part of the assignment makes sure that you can successfully insert nodes into, and delete nodes from, a BST and an AVL tree.

First, generate a random BST that has height 5 and contains random values from 10 through 99. You may have to generate dozens of trees until you get one that's exactly height 5. Don't worry that the tree is unbalanced. Print the tree.

Now repeatedly delete the root of the tree. Print the tree after each deletion to verify that you did the deletion correctly. Stop when the tree becomes empty.

Second, create an AVL tree node by node. Generate 35 unique random integers 10-99 to insert into the tree. Print the tree after each insertion to verify that you are keeping it balanced. Each time you do a rebalancing, print a message indicating which rotation operation and which node. For example:

```
Double left-right rotation: 76
```

As you did with the BST, repeatedly delete the root of your AVL tree. Print the tree after each deletion to verify that you are keeping it balanced.

A handy AVL tree balance checker:

```
private int checkBalance(BinaryNode node) throws Exception
{
    if (node == null) return -1;
    if (node != null) {
        int leftHeight  = checkBalance(node.getLeft());
        int rightHeight = checkBalance(node.getRight());
        if (   (Math.abs(height(node.getLeft()) - height(node.getRight())) > 1)
            || (height(node.getLeft())  != leftHeight)
            || (height(node.getRight()) != rightHeight)) {
            throw new Exception("Unbalanced trees.");
        }
    }
    return height(node);
}
```

Part 2

The second part of the assignment compares the performance of a BST vs. an AVL tree.

First, generate n random integers. n is some large number, explained below. Time and print how long it takes to insert the random integers one at a time into an initially empty BST. Do not print the tree after each insertion.

Time and print how long it takes to insert the same random integers one at a time into an initially empty AVL tree. Again, do not print the tree after each insertion.

Choose a value of n large enough to give you consistent timings that you can compare. Try values of n = 1,000 10,000 100,000 1,000,000

If $T(n)$ is the time function, how does the growth of $T_{\text{BST}}(n)$ compare with the growth of $T_{\text{AVL}}(n)$?

Second, generate k random integers. k is some large value. Time how long it takes to search your n-node BST for all k random integers. It doesn't matter whether or not the search succeeds.

Time how long it takes to search your n-node AVL tree for the same k random integers.

Compare the grow rates of these two time functions.

Third, perform the same random mixture of m insertions and searches on your n-node BST and then on your n-node AVL tree. m is some large number. Try different ratios of insertions vs. searches. Empirically estimate the ratio where an AVL tree has better performance than a BST for a mixture of insertions and searches.

Code

You can use any code from the lectures or from the textbook. You do not have to use parameterized generic types. You can use raw (nongeneric) types, or **`<Integer>`**.

Teamwork

You may work individually as a team of one, or you can partner with another student as a team of two.

You can be on only one team at a time. If you partner with someone, both of you will receive the same score for this assignment. You'll be able to choose a different partner or work alone for subsequent assignments.

What to turn in

Create a zip file containing:

- Your Java source files.
- Any instructions on how to build and run your code.
- Text files containing your outputs
- A short report (1 or 2 pages) that describes your conclusions from doing this assignment.

Name the zip file after yourself or yourselves.
Examples: **`smith.zip`**, **`smith-jones.zip`**

Each team should email the zip file to ron.mak@sjsu.edu. Your subject line must be:

`CS 146 Assignment #3` ***Your name(s)***

Example:

`CS 146 Assignment #3 Mary Smith & John Jones`

If you work with a partner, you should email only one assignment between the two of you. Whoever emails the assignment should CC the partner so that when I send you your team score, I can just do a "Reply all".