UNIVERSITY OF VIRGINIA, DEPARTMENT OF COMPUTER SCIENCE

Lab 06 - Trees

Dr. Mark R. Floryan

October 1, 2018

1 PRE-LAB

This week, you will be implementing three classes, which are all increasingly complicated Tree objects. You will start by implementing a basic Binary Tree with the three tree traversals convered in class. Then, you will implement and test a working Binary Search Tree. Lastly, you will implement a few methods to complete an AVL Tree.

1. Download the provided starter code
2. Implement some general methods in the BinaryTree class.
3. Implement the BinarySearchTree class
4. Implement the missing methods in the AVLTree class (some of this is done for you to simplify the assignment)
5. Use the provided tester files to verify your implementation works. Note that you should test your code more so than the provided tester does this time. The tester is NOT as thorough as in previous labs.
6. **FILES TO DOWNLOAD:** trees.zip
7. **FILES TO SUBMIT:** lab07.zip

1.1 BinaryTree.java

To begin, implement the BinaryTree class. These are methods that are useful for ANY binary tree (whether balanced or not). You will need to implement the following methods:

```java
public class BinaryTree<T>{
    //Prints the tree using in-order traversal
    private void printInOrder(TreeNode<T> curNode);

    //Prints the tree using pre-order traversal
    private void printPreOrder(TreeNode<T> curNode);

    //Prints the tree using post-order traversal
    private void printPostOrder(TreeNode<T> curNode);
}
```

The Binary Tree contains a few methods that are implemented for you. This includes the main print functions, that simply call the helper functions described above on the root to give off the recursive prints. In addition, a method called printTree() is provided that prints the tree in a somewhat formatted method. This may be useful when debugging your code. Lastly, a simple recursive method that computes the height of the binary tree is provided as an example of recursion in trees.

1.2 BinarySearchTree.java

Next, you will implement a binary search tree. This class will extend the binary tree class from earlier, and thus inherit the print methods defined earlier. Your binary search tree should implement the provided Tree interface, shown below.

```java
public interface Tree<T extends Comparable<T>> {

    public void insert(T data);

    public boolean find(T data);

    public void remove(T data);

    public TreeNode<T> findMax(TreeNode<T> curNode);
}

/* You BST implements the interface above */
public class BinarySearchTree<T extends Comparable<T>>
```

```
                              extends BinaryTree<T> implements Tree<T>{

            //TODO: Implement this class
}
```

You may add other supporting methods to your binary search tree if you find that to be helpful.

1.3 AVLTree.java

Lastly, you will implement an AVL tree that inherits from your binary search tree. An AVL tree can take advantage of the insert and remove methods from the class it inherits from (i.e., BinarySearchTree.java). Thus, to insert into an avl tree, you can call super.insert() and then simply check if the current node needs to be balanced. Some of this implementation is provided for you, but you will have to implement the following methods yourself:

```
public class AVLTree<T extends Comparable<T>>
                              extends BinarySearchTree<T>{

      //Insert and remove are partially coded for you already

      //figures out whether a double or single rotation is
      //needed and in which direction(s)
      private TreeNode<T> balance(TreeNode<T> curNode);

      //rotate right on the curNode provided
      private TreeNode<T> rotateRight(TreeNode<T> curNode);

      //rotate left on the curNode provided
      private TreeNode<T> rotateLeft(TreeNode<T> curNode);

      //compute the balance factor of the given node
      private int balanceFactor(TreeNode<T> node);

}
```

Once you are done, you can look at the two provided tester files to check your implementation. As stated earlier, these files do not rigorously check your implementations, so you should be writing your own test cases in addition to the few provided.

When you are done, submit your entire project as a zip file to Collab.

2 In-Lab

The goal of this in-lab is to continue practicing with trees in a laid back environment. As usual, you will:

1. Get into small groups of two

2. The TAs will present you with some programming challenges regarding trees. These will not be graded, but you are required to attend and to participate.

3. You may think about the challenges before lab below if you'd like, but we highly recommend that you not solve them ahead of time.

4. The TAs will give you time to solve each problem and lead you in sharing solutions with one another.

2.1 Trees

The TAs will lead you in going through the following challenges. It is ok if you do not get through each of these. When coding these, you may want to use your tree implementations from the prelab for these challenges.

1. Write a method that accepts the root node of a binary tree, and flips the tree recursively. More formally, every right child and left child of EVERY node (including internal nodes) should be flipped.

2. Write a method that accepts the root of a binary tree, and returns tree iff the tree is symettrical. More formally, for all values x in the tree. If I can find x by going on some path (e.g, right, right, left), than I can find another copy of x by going the inverse of that path (e.g., left, left, right).

3. Given an expression tree, print out the value of the expression. You can model this as a Binary Tree that stores strings. Use Integer.parseInt() to turn the numeric strings into integer equivalents.

4. On older phones, T9 word was used for texting friends. Users would type numbers on their phone (e.g., 2,2,8) and the system would find words that could be made with those sequence of letters. For example, 2 is either a,b, or c and 8 is either t,u,v. So one valid word the person may have been typing is cat. Write some code that reads in a list of valid english words, and builds a tree such that given a list of numbers typed, traversing

that tree would take me to the valid english words that may have been spoken. Note that you will need a tree that has more than two children here.

5. Continuing the previous question, write a method that given the sequence of numbers typed and the tree above, prints out all the english words that may have been typed.

If you have coded up all of these and they work, then show the TA and you may leave early.

3 Post-Lab

The goal of this post-lab is to write a report analyzing the use of a binary search tree versus an avl tree. For the first time, we will be asking to find a dataset on your own to use for your analysis.

You will perform an experiment by doing the following:

1. Find a large corpus of data. I would recommend a large piece of text. Perhaps a very long article or a an electronic book. You may share datasets with each other if you find good ones.

2. Take your file and insert every element (e.g., word from the book) into both a binary search tree and an avl tree. Use a timer to see how long it takes to insert all of the elements into the avl versus the bst.

3. Call the find method on every word in your dataset. Time how long this operation takes in each data structure total.

4. Print out the height of each tree.

5. Now, repeat the experiment, but this time don't use a real dataset. Use random numbers from 1-10000. Do you notice any differences?

6. Write a report summarizing and analyzing your findings. Which data structure was faster.

7. **FILES TO DOWNLOAD:** None

8. **FILES TO SUBMIT:** PostLabSeven.pdf

3.1 Report

Summarize your experiment and your findings in a report. Make sure to adhere to these general guidelines:

- Your submission MUST BE a pdf document. You will receive a zero if it is not.

- Your document MUST be presented as if submitted to a professional publication outlet. You can use the template posted in the course repository or follow Springer's guidelines for conference proceedings.

- You should write your report as if it is original novel research.
- The grammar / spelling / professionalism of this document should be sound.
- When possible, do not use the first person. Instead of "I ran the code 60 times", use "The code was executed 60 times...".

In addition to the general guidelines above, please follow the following rough outline for your paper:

- **Abstract**: Summarize the entire document in a single paragraph
- **Introduction**: Present the problem, and provide details regarding the two strategies you implemented.
- **Methods**: Describe your methodology for collecting data. How many method calls, how many executions, how you averaged things, etc.
- **Results**: Describe your results from your execution runs.
- **Conclusion**: Interpret your results. Which data structure was faster? Did the results change from a real dataset to a random one? If so, why? How different was the heights of the different trees? Why?

Lastly, your paper MUST contain the following things:

- A table (methods section) summarizing the experiments and how many execution runs were done in each group.
- A table (results section) summarizing the results of experiment 1 (number of inserts / deletes).
- A table (results section) summarizing the results of experiment 2 (number of threads).
- Some kind of graph visualizing the results of the table from the previous bullet.