Lab 9: Turn Around

Pre-Lab Notes

Lab submissions will **not be accepted late** – they must be submitted by **midnight the day after they're assigned** (to account for time zone issues). Some latitude might be allowed (I won't disqualify a submission that's mere minutes late!) but that's purely discretionary.

Labs are treated differently from assignments – you're encouraged to work with fellow students, ask questions of the TA during your lab section, and generally collaborate to complete your work. You should still submit your own work, **including citing with whom you collaborated in your submission**, but using answers developed by others will not be treated as plagiarism.

Nevertheless, you are still encouraged to make a good faith effort to complete your labwork, in order to exercise your understanding of the topics it covers. Lab submissions are marked purely 1 or 0 by whether the TA believes you at least made an honest attempt, even if you didn't succeed, so the value is in what the attempt teaches you rather than simply having the right answer.

1. AVLs and Self-Balancing Binary Search Trees

Our return to Trees has shown us they have a lot of potential as efficient key-based data structures, but also require a lot of work to set up and maintain. The Binary Search Tree is a classic example of a Tree that orders itself for convenient traversals, and allows Tree Search to uncover any key in one journey from the root to a leaf (or at least where that key would belong).

Unfortunately, the height of the BST isn't naturally constrained, and the shape of the Tree ends up depending on the order in which the keys are added – a Tree built by adding keys that are each larger than the previous key will end up with a height of n (where n is the number of keys), spoiling any potential efficiency gains. That's where AVL Trees come in, an extension to BSTs that add a crucial rebalancing step after insertions and removals to keep a BST's height under control.

This rebalancing is more than a little tricky to implement. If the height of a node's left and right child differ by more than one, we say that node has become unbalanced, which will require restructuring the Tree to solve. This process is called **tri-node restructuring**, since it involves nominating three nodes based on their heights and relationships, and then “rotating” them different ways depending on the pattern they fit.

2. Outline

You're going to get quite a bit of code up front this time. Be sure you've downloaded the lab 9 .zip file from the course website's assignment page (the link should be right next to the one for this .pdf).

Inside, you'll find an IntelliJ project for you to complete. It includes a simplified snippet of a Binary Search Tree that's become unbalanced at the root, and it's up to you to complete a restructure method that can fix the tree and restore the balance expected of an AVL Tree.

As written, the code sets up five internal nodes and six external ones, though the external nodes in this case are more like “bystander” nodes that play no active role in a restructuring rotation and could be replaced by whole subtrees in another tree. As such, we've also artificially tweaked one of their height values to be 1 rather than 0, in order to create the imbalance at the root without having to

build out the entire tree.

If you look closely at the hand-coded nodes (particularly the hints in their names and the comments), you might be able to figure out what kind of rotation is called for in this situation, and by matching it to the appropriate graph from the slides you could achieve the needed restructure by just manually rearranging the links between the different nodes. I recommend you try and implement a generalized function that can work by taking in just the unbalanced node and going from there instead, and ideally one that can handle all four possible rotation cases!

There's not much else to tell you, this time. Take the opportunity to play around with rotations and maybe even try building a few more tree configurations to see if you can figure out all four rotations. Hopefully this'll show you what kind of work it'd take to implement a fully-featured AVL-BST.

3. Deliverables

All lab submissions will be done through CourSys at https://courses.cs.sfu.ca/. For this lab, that should include:

1. A .zip archive of your code, organized into a single Java project. To ensure compatibility, it's recommended to use IntelliJ while developing your code, then using the export function under File -> Export -> Project To .zip File. Be sure you downloaded the sample project from the course website! You should zip up the whole thing and send it back, not just your own code.

Be sure you tidy up your code before submitting! Check the Java code style guide posted on the course website, and do your best to provide helpful comments.