COP 4516 Week 3 Individual: Trees (Solution Sketches)

AVL Tree Heist

The problem never asks you to make any AVL Tree rotations, but it does ask you to be able to detect height imbalances. Thus, you should simply maintain the height of a node in each node and update this on every insert operation. If a tree isn't a valid AVL tree at any point in time, then at the end of the insertion, some node on the ancestral path will be unbalanced. Thus, after you recursively insert a node, before you return your answer from the insert, check the heights of the left and right (which should have already been updated). If this difference is more than 1, then the answer for the case is NO. Just mark a global flag to this effect and continue processing all of the insertions. At the end of processing all the insertions, look at the flag to output.

Three common errors on this problem: Trying to answer as soon as an imbalance occurs without reading in all of the data (so that a future case is read in incorrectly), only checking if the end tree is balanced or no, and getting a run-time error on a tree with 0 nodes, which is allowed in the input. It's possible to insert nodes in such a way that in the middle the tree isn't balanced, but by the end, it is. Depending on how one writes the code, if they hard-code accessing the first item indexed, but it doesn't exist, the code could crash.

Which Base is it Anyway?

To interpret a number in octal, multiply the digit in the k^{th} place from the right by 8^k. For hexadecimal, you multiply this digit by 16^k. To avoid precision errors, use integers, don't use a built-in power function. If you read the input in as an integer, it's easier to go through the digits backwards, in which case, you can keep separate variables that update the current power of 8 and 16. If you read the input as a string, you can go through the digits forwards and apply Horner's rule to update the value after processing each digit. (If the current octal value is val, and we just add in the digit c, then the new octal value will be 8*val + c.) Last, if any of the individual digits are 8 or 9, overwrite the octal value calculated to be 0. Also, several languages have this capability built in.

Sorting Exams

The key observation is that our cost function of merging two stacks is the sum of those stacks and that all things being equal, we'd rather have our current merge operation be of minimum cost, which means merging the two minimum stacks. (The exchange argument formally proves this.) Then, we want to find the two new minimum stacks and repeat until there is only one stack. Thus, store all numbers in a priority queue, delete min twice, add these two numbers together and add that to the total cost, then put this sum back in the priority queue and repeat this n-1 times total, where n was the original number of stacks of exams.

Spreading News

Consider solving this problem for the root of some tree of employees. At the root, you get to choose which order to tell each of your direct subordinates. Naturally, you want them in order of which subtree will take the longest to spread the message to the shortest. For example, say that there are four subtrees, and it would take 3, 9, 8 and 8 seconds respectively for the message to completely spread through each of those subtrees. We want to tell the subtree that takes 9 seconds first, followed by one of the subtrees that takes 8 seconds, followed by the next one that takes 8 seconds, followed by the one that takes 3 seconds. This is because waiting to tell the "slowest" subtree could potentially delay the total amount of time the message takes to spread throughout the whole tree. Given the times above and the ordering given, the message actually gets through the second subtree in 1 + 9 = 10 seconds, since it took one second to tell that subordinate, 2 + 8 = 10 seconds for the third subtree since that subordinate waited 2 seconds to get the message, 3 + 8 = 11 seconds for the fourth subtree, since that subordinate waited 3 seconds to get the message, and 4 + 3 = 7, since the last subordinate waited 4 seconds to get the message. Thus, for this case, it would take a minimum of 11 seconds for the message to spread.

If we reordered these calls as say, 9, 3, 8 and 8, then the corresponding finish times would 1+9 = 10 seconds, 2 + 3 = 5 seconds, 3 + 8 = 11 seconds and 4 + 8 = 12 seconds. Thus, this ordering of telling the subordinates would not lead to the minimum answer.

Thus, in the general case do the following:

1. Recursively solve the problem for each child, storing the answer from each recursive call in an array list.

2. Sort that list in reverse order. (So, for our example, it would be 9, 8, 8, and 3.)

3. Add i to the i^{th} of the list, starting with i = 1. (So, for our example it would be 10, 10, 11 and 7.)

4. Return the maximum of the values listed in step 3.

The base case is a single node, which takes time 0.