COP 4516 Spring 2022 Week 4 Individual: Tries (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.

Two 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), and only checking if the end tree is balanced or not. 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.

Dot Game Dominator

Once again a greedy strategy can be used here. You want to eat the largest dot that is strictly smaller than your current size, since at each step, this makes your size grow the most. No competing sequence which eats a smaller dot can beat this greedy strategy. To implement this, a TreeSet can be used, since the class TreeSet has a method lower. However, be careful though - TreeSets don't store duplicates and an error will be caused in situations where there are multiple dots of the same size. To avoid this error, create a TreeSet of objects, where each object stores both the size of the dot and its ID number (you assign this as you read in the dots to be unique). Alternatively, it's very difficult to make data where the number of dots you eat is very large. Or if it is, where it takes you a long time to find the largest dot smaller than you. Thus, a relatively nieve implementation which maintains a sorted list of the dot sizes will pass the data that was generated for this problem.

Ground Game

This question is the banger in the set. Keep a variable that tracks how many levels underground the player is and keep a second variable that stores the maximum levels underground the player has gone. When processing the input, ignore characters moving left(<) and right(>), add one to the count when moving down (v), and subtract one to the count when moving up (^). After each change to the counter, see if the new value is greater than the previously seen maximum. If so, update the maximum.

Strange Lottery Simulator

A regular trie where you store the number of words stored in each subtrie can handle the query asking for the number of participants with a particular prefix. However, if the names can be reversed, that same trie is not helpful. However, what can be done is that two ***different*** tries can be maintained simultaneously. Notice that reversing a name twice just brings it back to its original form. Thus, Whenever a name is added, add it to a trie forward as is and ALSO add its reversed version to a trie reverse. (So, "sharon" would get added to the trie forward AND "norahs" would get added to the trie reverse.) Finally, whenever you receive a query, you just need to know if the current state of the names is regular or reverse. Just keep either a boolean variable or an integer that toggles between 0 and 1, and then answer the query in the appropriate trie based on the value of the variable storing the current state of the names (forward or reverse).