AVL Tree - *Zipcodes*^{3}

CS 151 - Introduction to Data Structures

Assignment 9 - due Friday 4/28

This choice of the final assignment is an individual project with an earlier due date. Compared to the image-processing choice, this is definitely more straight-forward.

1 Requirements

Rewrite your Assignment 5 to use an AVL tree as your main data structure, instead of a sorted `ArrayList`.
Recall that the zipcodes in `uszipcodes.csv` are already listed in lexicographic order and thus using an unbalanced binary search tree (i.e. `LinkedBinaryTree`) will guarantee to result in a linear tree (when you insert in sorted order) and thus will have worst-case $O(n)$ complexity on all operations. This is why an AVL-tree is a good choice. Note that the AVL-tree's advantage over a sorted `ArrayList` is not on lookup - both binary search on sorted array and AVL have $O(logn)$ lookup, however, an AVL supports $O(logn)$ insertion and deletion but an `ArrayList` only has $O(n)$ insertion and deletion because of shifting.

1. Implement a `AVLTree` that extends your `LinkedBinaryTree` from A6, which must also implement the A6 `BinaryTree` interface

2. All requirements for `LinkedBinaryTree` are the same as A6. If your A6 was not completely functional, you should start by fixing that first.

3. The changes you need to make to `LinkedBinaryTree` and `AVLTree` are mostly the same as your lab10, so you should make sure your lab is functional first.

4. `toString` of `AVLTree` should be overriden to return a string listing the in-order traversal of all elements in the tree, in the following format. Each element should represented by its zipcode, followed by height in parethesis. For example, Bryn Mawr should be represented by the string "19010(1)", if

the `PopulatedPlace` representing Bryn Mawr was a leaf on your `AVLTree`. Traversal string should be space separated. See section 2

5. Once your `AVLTree` is functional, modify the rest of the methods to to use your `AVLTree` for storage and lookup.

6. Instead of hard-coding the input files, i.e `uszipcodes.csv` and `ziplocs.csv`, switch to taking input file names from commandline arguments, see section 2.

7. The rest of the restrictions and correctness requirements for Assignment 2/5 remain. Correctness testing (on lookups once the AVL tree is functional) should be the same as well.

8. Time your new implementation using commandline redirection of the following test file: `~dxu/handouts/cs151/tests/a2/in.txt`. That is, issue the following command at the prompt:
   `time java Main -i uszipcodes.csv ziplocs.csv < in.txt > out.txt`
   Compare the result with your A5 implementation. Is there a speedup? Report your findings in your README.

2 Command Line Input

You will receive the two input files on the command line following the `-i` flag as your input, for example: `java Main -i uszipcodes.csv ziplocs.csv`. You can assume that the first file will follow the `uszipcode.csv` file format and the second one will supply additional location information, like `ziplocs.csv`. In addition, support the following flags:

1. `-d`
   indicates that you should print additional debugging information on your AVL tree, in the following format: height of your AVL tree, as an integer on its own line, followed by the `toString` of your tree. For example, if there are only the following three zipcodes in the input (19010, 91729 and 97252), your output should be:

   ```
   2
   19010(1) 91729(2) 97252(1)
   ```

   Print this additional debugging output before taking query input interactively, as usual. If `-d` is not given, program should behave/output exactly the same A2 or A5.

Electronic Submissions

1. **README:** The usual plain text file `README`

    Your name:

    How to compile: Leave empty if it's just `javac Main.java`

    How to run it: Leave empty if it's just `java Main`

    Known Bugs and Limitations: List any known bugs, deficiencies, or limitations with respect to the project specifications. Documented bugs will receive less deduction versus uncaught ones.

    Discussion: As explained above in 1.8.

2. **Source files:** all `.java` files

3. **Data files used:** `uszipcodes.csv` `ziplocs.csv`

DO NOT INCLUDE: Please delete all executable bytecode (`.class`) files prior to submission.

To submit, store everything (README, source files and data files) in a directory called `A9`. Then follow the directions here:
`https://cs.brynmawr.edu/systems/submit_assignments.html`