Binary Trees and Polling Data

CS 115 - Introduction to Data Structures

Assignment 6 - due Friday 3/24

For this assignment and the next one you will be working within the same codebase to look at polling data for the 2020 U.S. presidential election. Before the two main political parties put forward their nominees for president, the Democratic and Republican parties hold primary elections to determine who their nominee will be. While the Republican nominee was essentially predetermined to be President Trump, who the voters of the the Democratic party would elect to receive the party's nomination was still very uncertain (at the time). In order to make repeated predictions about the likely outcome of the Democratic party nomination, pollsters (statisticians) regularly conduct polls (surveys) to sample Democratic primary voters and ask who they plan to vote for. These results are compiled, released, and eagerly tracked by the news media and public to determine which candidate currently has the largest percentage of support.

In this assignment, your job will be to take the poll results given as input via CSV files and update the entries of a binary tree so that it stores the name and current polling percentage for each candidate. In the next assignment you will modify this tree so that you can easily retrieve the name of the candidate currently leading in the polls.

1 Implementing a Binary Tree

Start by implementing the given `BinaryTree` interface (`~dxu/handouts/cs151/code/9-1`) as a `LinkedBinaryTree` so that generic objects that implement the `compareTo` function from the `Comparable` interface can be inserted into your tree.

Requirements:

1. Implement the `BinaryTree` interface as a `LinkedBinaryTree`. Make sure the implementation is entirely recursive and is a linked data structure, i.e., you should *not* use any `for` or `while` loops, arrays, ArrayLists, etc. Hint:

you may find it useful to use private helper methods that are called from the publicly defined method in the interface.

2. Your implementation should be properly encapsulated, i.e. no implementation details should be made visible outside of the `LinkedBinaryTree` class - it should only implement the `BinaryTree` interface. Hint: consider making a `private` inner class to define a `Node` of the tree.

3. Insertion should be done using the `compareTo` method of the given element so that smaller elements are put into the left subtree and larger element are put into the right subtree. Insertion of elements that are already in the tree should *replace* the current element. When you put your polling data into the tree this will be equivalent to updating the poll numbers for a candidate.

4. As part of implementing the BinaryTree interface you will implement all three orders for tree traversal, each returning a string in the form:
   `(element1,element2, ... , elementn)`
   where the order is determined by the correct order of the traversal. Note that these methods should also use a recursive design.

5. Implement a `boolean contains(E element)` operation, which returns `true` if the given element exists in the tree and `false` otherwise.

6. Implement a `boolean remove(E element)` operation, which returns `true` if the given element exisits and is removed, and `false` otherwise. You should be sure that this method works seamlessly with the other methods. For example, in-order traversal should return a string containing the elements in sorted order even after an element has been removed. Describe your chosen design for this method in the `README`.

7. You should override the `toString` method to return a `String` that looks like the following:

   ```
   Tree:
   Pre:    b a c
   In:     a b c
   Post:   a c b
   ```

   Where the first traversal is a pre-order traversal, the second is in-order, and the last is post-order.

2 Storing Polling Data

The polling data you are given will include the candidate's full name, their last name, and the percentage of the people polled who said they would vote for that candidate.

Requirements

1. Create a class to store the polling data.
2. Have your created class implement the `Comparable` interface so that polling data objects are put in alphabetical order based on the candidate's *last* name.
3. Override the `toString` method to return a `String` with the following formatting:
   `Full Name:5.0`
   where Full Name is the candidate's full name and 5.0 was the candidate's polling percentage.

3 Polling Data from CSVs

The website FiveThirtyEight makes polling data for presidential primary candidates available (`https://data.fivethirtyeight.com/`). We have preprocessed this data for you so that only the relevant data is included and you will receive one file per conducted poll.

Each polling data CSV has the following format:

```
answer,candidate_name,pct
Biden,Joseph R. Biden Jr.,25
Sanders,Bernard Sanders,16
```

The first column gives the last name of the candidate, the second column gives the candidate's full name, and the final column is the percent the candidate is polling at in this poll. Note that the given percent can be a floating-point number.

Each file is named something like `dempres_20190310_1.csv` where `dempres` indicates that these are polling results for the Democratic party presidential primary, `20190310` indicates that the polling results were completed on March 10, 2019, and `_1` indicates that these are the results for the first poll completed on that date (there may be multiple from different sources).

Your job is to take the polling data in each file and insert it into the binary tree. Your resulting tree should contain the polling data for each candidate from the most recent date for which there is data from the files given on the command line. Each polling result will only include some of the candidates.

Requirements:

1. Take filename input from the command line into the main method of your `Main.java`. You may be given multiple filenames. An example of given arguments might be:
   `dempres_20190210_1.csv dempres_20190210_2.csv dempres_20190310_1.csv`
   or:
   `dempres_20190210*.csv`
   Recall that unix shell will expand the `*` for you.

2. You should process the given files in increasing date order. You may assume that the files are given in this order.

3. Use your overridden `toString` method to print the tree after polling results from each new date are inserted. Thus, your resulting printed information should include one snapshot of the tree per given polling data CSV.

4 Testing

A test program `CheckFormat_A6.java` has been provided for you. Note that this mostly makes sure that your output format is as expected for our autograder. Although some correctness testing is included, it's minimal. You are expected to do your own testing. Also note that different from A4, `CheckFormat_A6.java` does not replace `Main.java`, but instead will call your `Main.main`.

Some additional test code that you can insert into your `Main` with expected output:

```
BinaryTree<Integer> intTree = new LinkedBinaryTree<Integer>();
intTree.insert(8);
intTree.insert(11);
intTree.insert(5);
intTree.insert(17);
intTree.insert(1);
intTree.insert(9);
intTree.insert(3);
System.out.println(intTree);
// output
//Tree:
//Pre:    8 5 1 3 11 9 17
```

```
//In:     1 3 5 8 9 11 17
//Post:   3 1 5 9 17 11 8

BinaryTree<Character> letterTree = new LinkedBinaryTree<Character>();
letterTree.insert('A');
letterTree.insert('C');
letterTree.insert('G');
letterTree.insert('B');
letterTree.insert('D');
letterTree.insert('G'); // inserting again, should replace
letterTree.insert('F');
letterTree.insert('E');
letterTree.insert('H');
letterTree.insert('I');
System.out.println("size:" + letterTree.size());
System.out.println(letterTree);
// output
//size:9
//Tree:
//Pre:    A C B G D F E H I
//In:     A B C D E F G H I
//Post:   B E F D I H G C A
```

5 Extra Credit

All extra credit should only be done **after** successful completion of all of the base requirements for this assignment. The number of points awarded for extra credit will be smaller than those for completion of the base requirements and the extra credit is designed to be *harder* than those basic requirements as well. You may choose which of the extra credit options below to pursue and can receive credit for some and not others where that makes sense. In the case that you implement ANY extra credit, you must list them in your README so that our grading will know to test for them and grant credit accordingly.

1. In cases where you are given multiple polling results on the same date, average the per-candidate results before inserting the data into the tree, i.e., if Biden received 25% in one poll and 31% in another poll from the same date, the inserted polling result should show Biden receiving 28% of the vote.

2. Make sure that the polls are processed in date order no matter what order they are given to you in on the command line.

6 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: Design of `remove` as explained above, and any extra credit implementations you chose to do.

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

3. **Data files used:** all `.csv` files

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 `A6`. Then follow the directions here:
`https://cs.brynmawr.edu/systems/submit_assignments.html`