Heaps and Polling Data

CS 206 - Introduction to Data Structures

Assignment 7 - due Thursday 4/4

For this assignment, you'll continue your previous investigation of Democratic primary polling data by determining which candidate is currently in the lead.

1 Implementing a Heap

You should start by implementing the given `PriorityQueue` interface with a heap so that generic objects that implement the `compareTo` function from the `Comparable` interface can be inserted into your priority queue.

Requirements:

1. Start with implementing an array-based binary tree `ArrayBinaryTree` that implements the given interface `BinaryTree` (same as A6).

   (a) You are *not* required to use only recursive implementation techniques this time around. You only need to implement a binary tree, *not* a binary search tree.

   (b) Remember that an arbitrary binary tree is not guaranteed to be complete (for example, the user might execute `remove` on any element any time) and you need to handle the case where positions may become `null` through out your tree.

   (c) The three traversal methods have the same specification as A6.

   (d) Override `toString` the same way as in A6, i.e. returning a string of the following format:

   ```
   Tree:
   (b,a,c)
   (a,b,c)
   (a,c,b)
   ```

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

(e) Add a method `toStringBreathFirst` which returns a string containing the elements of the tree traversed in breath-first traversal order. Note that this method is not in the interface because it is not a common traversal order for all binary trees, but is useful for displaying/debugging array-based ones.

2. Now implement the `PriorityQueue` interface as a `ArrayHeap` which extends `ArrayBinaryTree`. Note that it is recommended to add `protected` or `private` helper methods in both `ArrayBinaryTree` and `ArrayHeap` as you design deems necessary. On the other hand, your implementation should be properly encapsulated, i.e. no implementation details should be made visible outside of the `ArrayHeap` or `ArrayBinaryTree` classes. In other words, any method that leaks implementation details should not be public.

   (a) Your implementation of `insert` should use the `compareTo` method of the given element to determine which order to arrange the priority queue. Insertion of elements that are already in the tree should *update* the current element in the priority queue while making any updates necessary to guarantee the heap property. When you put your polling data into the tree this will be equivalent to updating the poll numbers for a candidate. Since this heap will be used to store polling data, you should be implementing this as a *maximum* heap, so that we will be able to easily retrieve the current top candidate. Note that there is very little actual difference between a max-heap and the min-heap you saw in class. The only change should occur with `compareTo`.

   (b) Your implementation of `remove` should ensure the heap property is maintained. If the given element can not be found in the heap, this method should do nothing and return false.

2 Getting the Top Candidates

While a heap is usually required only to return the maximum (or minimum) element, since this will be used to store polling data, it may be interesting to us to retrieve the top few candidates.

Requirements:

1. Add a method `ArrayList<E> peekTopN(int n)` that returns the top elements of the heap in order. The heap should not be any different after the

method was called than it was before the method was called, i.e., this is similar to peek in that it does *not* remove the top element. You should *not* implement this method by removing and then reinserting each element, as this has the potential to modify the heap.

2. Describe your design of the `peekTopN` method in your README file and give a big-O analysis.

3 Command Line Input

As in the previous assignment, you will take filenames that store polling data as arguments to your main method. Your resulting heap 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. The resulting heap should be ordered so that the candidate with the highest percentage of voters in the most recent poll is at the top of the heap.

Additionally, you will add an option for the users to provide a flag that will run your `peekTopN` method to determine and print out the top N candidates. The resulting arguments you should handle will look like this:

```
-n 5 dempres_20190210_1.csv dempres_20190210_2.csv
```

In the above case, the top 5 candidates would be displayed.

Finally, you will add another optional argument to remove some candidates from consideration. The full set of arguments you should handle will look like this:

```
-n 5 -r Biden Bloomberg dempres_20190210_1.csv dempres_20190210_2.csv
```

In the above case you would print out the top 5 candidates who are *not* Biden or Bloomberg based on the polling data in the given files.

Requirements:

1. Take filename input from the command line into the main method of your `Main.java`. You may be given multiple filenames. Process an optional flag to print out the top N candidates. These should be printed out like this (where the example given is for $N = 2$):

   ```
   Top 2 Candidates:
   Joseph R. Biden Jr.:29.0
   Kamala D. Harris:15.0
   ```

   The above printout should happen *once* after all polling data has been inserted.

2. Process the optional $-r$ flag to remove candidates. This should be done so that the top N candidates do *not* include any removed candidates if they are optionally printed out. However you should only perform these remove operations once. One suggested order for handing these flags is to 1) insert all the polling data (printing out the heap after each file is inserted), 2) remove the candidates, and 3) show the top N candidates. In other words, it is expected that when you print out the heap it *will* include the candidates that will later be removed.

3. You many assume that the list of filenames is always last, i.e. the first non-flag argument you encounter is assumed to be the beginning of the list of file names. Make sure you error-check thoroughly, both the arguments to the flags and the flags themselves. Your program should behave rationally no matter how unreasonable the input or the value of flags.

   Remember the order of flags should not matter, that is,
   `-n 5 -r Biden Bloomberg dempres_20190210_1.csv dempres_20190210_2.csv`
   and
   `-r Biden Bloomberg -n 5 dempres_20190210_1.csv dempres_20190210_2.csv`
   will generate the same output.

4. As in the previous assignment, you should print out the heap after each polling file is inserted.

4 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. Handle tied ranks appropriately for the `peekTopN` method. For example, in the case where there are two candidates who are tied for the best, peekTopN for $n = 1$ should print out *both* of those candidates.

2. Add an additional constructor to `ArrayHeap` which constructs the heap using the bottom-up construction method we discussed in class.

5 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 `peekTopN` 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 `A7`. Then follow the directions here:
`https://systems.cs.brynmawr.edu/Submit_assignments`