CS 3410 - Homework 06

Due date: see course Schedule and Blackboard.

Overview

You will use a Heap (PriorityQueue) to place items into boxes.

Requirements – C Version (75 points maximum)

1. Obtain the download, *hw6.zip*. There, you will find *BinaryHeap.java* and two other required files. The *BinaryHeap* file contains a *main* where you will write your code (or write a driver).

2. Do problem 21.25 *a* with the following additional requirements

   a. Read the input (doubles) from a text file: w_1 w_2 w_3 … Hint: you will need: **`BinaryHeap<Double>`**
   b. Print the results in the following format:

      Box 1 – weight = 0.8, contents: 0.1, 0.1, 0.1, 0.2, 0.3
      Box 2 – weight = 0.7, contents: 0.3, 0.4
      etc.

Requirements – B Version (85 points maximum)

1. Same as *C* version with this additional requirement:

   c. Create an *Item* class that implements *Comparable*. An *Item* has a *weight* property. When a weight is read, create with an *Item* and insert it in the heap. Hint: you will need: **`BinaryHeap<Item>.`**

Requirements – A Version (100 points maximum)

1. Same as *B* version with these additional requirements.

   d. You will implement a *closest-to-average heap.* When an item is removed from this heap, it will be the item that is closest (in absolute value) to the (current) average weight. As soon as an item is removed, the average of course changes and so *buildHeap* must be called. Thus, you will no longer rely on *Comparable.* Similarly, each time a weight is added, the average will change and buildHeap will have to be called. You might consider using the constructor that takes a collection. For example, with the input: 0.4, 0.4, 0.6, 0.6, the *C* version will use 3 boxes: 1 (0.4,0.4), 2(0.6), 3(0.6). This version will create 2 boxes: 1(0.4,0.6), 2(0.4,0.6). Hint: think carefully about this. When an item is inserted, it is easy to update the average, but we no longer need to bubble up the hole because there is no guarantee that the closest item will end up at the root. Similar for remove.

   e. Utilize an *Item* class and a *Box* class where a *Box* can contain any number of *Item* instances. Implement a *toString* method on each class and use it to print the results. Note: you can remove the *Comparable* requirement from the *BinaryHeap* class or you can make your *Item* class implement *Comparable*, though comparable will no longer be used.

f. (10 extra points) Explain, in terms of complexity, why this method would be faster than a method where we utilize a sorted list at every stage.

Deliverables

1. (if applicable) A Word document in this format:
   a. Title page:
      CS 3410 – HW 06
      Name
      Date
   b. Answers to question(s).

2. All Code.

3. Email me a zip file, *hw06-lastname.zip* with these items.