CS 251: Project 1
Summer 2019
Due: Sunday, June 16 by 11:59 PM

Overview

The purpose of this project is to introduce you to some introductory algorithms and data structures that will be extended later in the course. It will also introduce you to the programming environment and submission protocol you will be using throughout the course.

General Guidelines

For all programming projects, it is important that you follow the instructions carefully, both regarding the content of your programs and the submission guidelines. You should always compile and run your programs (using the provided test cases) before submission, as well as compiling and running after transferring your files to the submission machine. In general, you are much less likely to be granted points back in a regrade request if it is clear that you have not run the test cases on the submission machine. Give yourself plenty of time for submission so that you can handle any issues that might come up.

Specification

Imagine you are involved in running an online coding competition, where competitors can submit solutions to a problem, which will then be evaluated and given a unique numerical score, the range of which will depend on the number of submissions. When the competition is closed, you must take a list of the **N** submissions and produce a list of the **M** best submissions in order from highest to lowest score.

Input

The input (as command line arguments) to your program will be a filename *filename* and an integer *M*.

- *filename* refers to a file in the current directory containing the list of **N** scores. Input files will consist of **N** lines. Each line will contain a user id, followed by a space, followed by a numerical score. (NOTE: You will not know what **N** is.)
- *M* is an integer indicating how many scores must be included in the output file (the leaderboard).

Output

The output to your program will be a list consisting of the **M** top competitors (the leader-board), each on a separate line containing the user id followed by the numerical score. The Scores should be in order from highest to lowest.

Part 1: Score.java (20 points)
In Part 1, you will implement the Score class, which is a simple class that will model a competitor's score, which consists of the user id (a String), and the numerical score (an int). The methods you must implement are:

- *getUid*: return the user id
- *getScore*: return the numerical score
- *compareTo*: this must implement the *compareTo* from the Comparable interface
- *equals*: return true if the numerical scores are the same, false otherwise

The constructors and a *toString* method are provided for you. DO NOT CHANGE these!

Part 2: LBSorted.java (35 points)
In Part 2, you will implement the leaderboard (LB) as a sorted array. The LB will consist of a Score array, the size of which will be specified by the value of *M*. You should also maintain a variable keeping track of the index in the array where the current minimum value is stored. This is because the simplest way to implement the insert function is to insert items into the array from left to right (i.e. your starting index is the last one in the array.)
The methods you must implement are:

- a constructor that sets the instance variables using M.
- *getMin*: return the minimum Score in the array
- *insert:* insert a new Score into the array as follows…
    - ...if the array is not yet full, insert the new score into the current minIndex and adjust so that the contents are sorted
    - ...if the array is full, compare the new score to the minimum score and…
        - ...if the new score is smaller, do nothing…
        - ...else replace the minimum score with the new score and adjust the array so its contents are sorted…
- a helper method *swap* that takes two int indices and swaps the contents of the array at those indices (e.g. *swap(2, 7)* should swap the values at lb[2] and lb[7]).

The *toString* method and instance variables are provided for you. DO NOT CHANGE THESE!

Part 3: LBUnsorted.java (35 points)
In Part 3, you will implement the leaderboard (LB) as an unsorted array. The unsorted LB will consist of a Score array of size M, specified in the constructor. You should also keep track of an int variable *nextOpen* that keeps track of the next available space in the array (for easy insert). You need to implement the following methods:

- *getMinIndex*: return the index of the minimum Score in the array

- *insert*: insert a new Score into the array as follows…
    - ...if the array is not yet full, insert the new Score into the next open spot…
    - ...else compare the new Score to the minimum Score and replace the minimum only if the new Score is higher.
- *delMin*: delete the minimum Score in the array by setting that element equal to null; return the minimum Score
- a helper method called *sort* that uses a temporary array to sort the LB array by removing the minimum value and putting it in the new array until the original array is empty; set the instance variable to the new array at the end (Hint: this is what *delMin* is for.)

The *toString* method is provided for you. DO NOT CHANGE THIS!

Part 4: Main.java (10 points)

In Part 4, you will implement the Main part of the program. It should take two command line arguments (*filename* and *m*). Then it should open and read the file line by line, inserting the scores into both an instance of LBSorted and an instance of LBUnsorted. In the end, print out the contents of each of the leaderboards to standard out (the test cases will pipe the results into files to be compared to the expected output.) Make sure your output matches the expected output exactly by running the test cases!

<u>What You Are Provided</u>

- **Skeleton Code:** Score.java, Main.java, LBUnsorted.java, LBSorted.java
- **Test files and script:** run_script_distribution.sh will run the test cases and give you a result; testing files for input and expected outputs

<u>Testing</u>

Part 1: Score.java

You are provided with input_1_1.txt which contains test cases for all the methods you implement in Part 1. Feel free to add test cases yourself. To run the test cases, use the script test_part1.sh provided to you. This script compiles your Score.java class and runs it with the input_1_1.txt. Alternatively, you can run your code yourself with input_1_1.txt as an argument and compare the output with the output_1_1.txt provided to you. Your output and the contents of output_1_1.txt should be identical.

Part 4: Main.java

You are provided three test cases for Main.java, which tests both LBSorted and LBUnsorted. Feel free to add any test cases yourself (e.g. you might want to divide up some of the test files to test Parts 2 and 3 separately, but you need to be sure that your submitted Main.java includes both--that is, if you comment out the LBSorted stuff in Main.java to test LBUnsorted by itself, make sure you uncomment it before submitting.)

To run the tests that are provided:

- Use the script **run_script_distribution.sh** file to test your code. This script...
    - ...will first compile your code.
    - ...will run Main.java with using the input files,
    - ...pipe the output to a file,
    - ...and compare the file to the expected output.
- The input and output files are provided for you, and you can see which input file produces which output by seeing the commands in the script (e.g. the line `java Main.java input_2_1.txt 20 > output_4_1.txt` runs the program using the file `input_2_1.txt` as input and `M = 20`, pipes the output to the file `output_4_1.txt`.)
- Your program's output will then be compared to the expected output (e.g. `cmp --silent output_4_1.txt output_4_1_test.txt && echo '###SUCCESS: Files Are Identical! passed ###' || echo '### WARNING: Files Are Different! did not pass (check output_4_1.txt)###'` compares the output file `output_4_1.txt` with the expected output `output_4_1_test.txt` and lets you know if they are the same.)
- Feel free to write your own test cases, but do not submit them.
- To run the test script, run the following command in your terminal: `./run_script_distribution.sh`
- In order to avoid trivial differences in output (e.g. an extra line at the end of a file), the `toString` methods for each applicable class are provided for you, so you can just call `System.out.println`. DO NOT CHANGE the `toString` methods unless you want your test cases to fail.

Grading

Part 1: We will grade Part 1 with tests similar to those provided to you. The output of the following methods will be tested: getUid, getScore, compareTo, equals. The inputs will be similar to those from input_1_1.txt

Part 2: We will test Part 2 by running a version of the Main.java program that only tests LBSorted. The input/output will be similar to what you are provided but will only have the results of LBSorted.

EXAMPLE: `MainSorted test1.txt 4`

test1.txt:

```
abcde 78
bcdef 81
cdefg 102
```

```
defgh 32
efghi 62
fghij 83
```

output:

```
cdefg 102
fghij 83
bcdef 81
abcde 78
```

Part 3: We will test Part 3 by running a version of the Main.java program that only tests LBUnsorted. The input/output will be similar to what you are provided but will only have the results of LBUnsorted. See the example for Part 2.

Part 4: We will test Part 4 by running your Main.java with test cases similar to those provided for you. PLEASE MAKE SURE Main.java includes a call to both LBSorted and LBUnsorted, even if your tests don't all pass. If you pass Part 4 and fail Part 2 or Part 3, we will inspect your code and if the code was altered to make Part 4 appear to pass, you will receive a 0.

EXAMPLE: `Main test1.txt 4`

test1.txt:

```
abcde 78
bcdef 81
cdefg 102
defgh 32
efghi 62
fghij 83
```

output:

```
Sorted:
cdefg 102
fghij 83
bcdef 81
abcde 78
Unsorted:
cdefg 102
fghij 83
bcdef 81
abcde 78
```

Submitting

- You should submit the following AND NOTHING ELSE: `Score.java, LBSorted.java, LBUnsorted.java,` and `Main.java.`
- Do not submit any compiled files (.class) or any test files.
- Assignments will be tested in a Linux environment. You will be able to work on Linux machines in HAAS and LAWSON, and you should be able to ssh into a data.cs machine as well.
- The Java compiler used is javac (v10.0.2). Your project must compile on data.cs.purdue.edu using javac (v10.0.2). If it doesn't compile, it can't be tested, and you'll get a 0!
- To submit files:
    - Login to data.cs.purdue.edu (in the labs or using an ssh connection).
    - Create a folder with your username. Put the required files into the folder. Do not make any sub-directories. All the required files should be in the single folder under your username. Do not change any names of the files.
    - Go to the upper level directory and execute the following command:
      ```
      turnin -v -c cs251 -p project1
      ```
    - If you forget the "-v" flag, your submission would be replaced with an empty one.

Other Notes

As mentioned in the course overview, do not use any libraries not officially sanctioned for this assignment. This project is straightforward in what is expected, but if you have questions about the expectations, ask!