CS2040S: Data Structures and Algorithms

Contest 2: Data Analysis Speed Demon

Competition! *Due: March 31, 11:59pm*

Overview

We live in a world overwhelmed with information. Every two days, we create as much new information as was created in the entire history of human civilization up until 2003 (according to Eric Schmidt, the CEO of Google). We produce more than 3 exabytes of data per day! Much of this data is stored in large databases, and one of the challenges today is to rapidly process and analyze all the data. Since the databases are so large, it requires very fast algorithms and data structures that are highly optimized for maximum efficiency. In this competition, you will try to develop the *fastest* algorithm for analyzing a large dataset.

When analyzing a large dataset, there are many different goals. We will focus on a particular type of data mining in which we want to discover properties and patterns of the underlying data. Frequently, we want to know various statistics: the average, the median and the mode. Often, we also want to know about patterns: how often do users of a certain profile (e.g., males between the ages of 18 and 32) buy a certain item? Often, we want to know about correlations: how often does a user who buys item A click on link B?

For the purpose of this competition, we define an abstract data mining problem that involves finding correlations in our data set. The database consists of a large set of very large data entries. The goal is to find how many pairs of entries are identical, i.e., contain the same information. Your job is to implement a data mining program that reads in the database and performs this analysis *as fast as possible.*

Problem Details

We now describe the details of the competition more precisely.

Input. The input "database" is a file consisting of a set of lines of text, each of which represents one entry. Each line of text contains a large number of characters. (Notice that the lines may consist of thousands, or even tens of thousands, of characters.)

Your program will implement a single public method: `public int processData(String filename)`. This method will take a filename as an input. It should then parse the file and process the data in the file. Finally, it should return an integer as the answer. (We have given you an existing class file template to use.)

The format of the input is as follows. The first line of the database contains a single integer n, which represents the number of entries in the database. It is followed by n lines, each containing an arbitrary number of characters and ended by an end-of-line/line-feed character (ASCII character 10). Note that entries may consist of any of the legal ASCII characters in the range 33 to 126 (inclusive). Characters may be repeated, and entries may be of any length.

Some constraints are as follows:

- $1 \leq n \leq 10^6$
- Number of characters in each line: $\leq$ 5000
- Each character is an ASCII character in the range 33 to 126 (both inclusive)

Output. Your program should calculate the number of pairs of entries that contain an identical multiset (i.e. multiple instances of the same character are allowed) of characters. Notice that the characters may appear in any order: two entries e_1 and e_2 are equivalent if e_1 is equal to some permutation of e_2. You should print your output, which should consist of an integer, followed by a newline character.

Example. The following is an example of an input database:

```
7
BCDEFGH
ABACD
BDCEF
BDCAA
DBACA
DABACA
DABAC
```

The appropriate output in this case is:

```
6
```

In particular, note the following six pairs of equivalent entries:

```
(ABACD, BDCAA)
(ABACD, DBACA)
(ABACD, DABAC)
(BDCAA, DBACA)
(BDCAA, DABAC)
(DBACA, DABAC)
```

Rules

The following are the rules of the competition:

- Your solution must be written in Java.
- You may use any available Java libraries. (You should note the fastest solutions usually would not rely heavily on existing libraries.)
- You are allowed to submit only one final program.

- All code in your submitted program must be written entirely by you. You may still reference any ideas or algorithms that you find on the internet, in books, etc.
- The competition will end on March 31, 11:59pm.
- The top 10 students on the leaderboard after the competition ends will be the winners of the contest.

Submission Details

Here are some instructions for submitting the code:

- You need to submit your solution on Coursemology as usual.
- If you want to qualify for the competition, you will need to separately submit your solution on a different platform. Ensure the `main` method is as given in the template file. More details about this will be given later via an announcement.
- The top 10 winners of the competition will get bonus EXP. There will be additional 900EXP for 1st place, and additional 400EXP for the other 9.

Hints

A few hints toward achieving good performance:

- First, develop and test a working solution that achieves good asymptotic performance. Then improve it.
- Think about the performance of the data structures you are using and the actual costs of the operations involved.
- Remember that for large databases, memory usage is important. Maintaining big data structures that use a lot of memory can be slow.
- Think about data locality in memory: accessing elements in memory that are near to each other is much cheaper than accessing elements that are far away from each other.
- Beware of the small costs that add up. For example, declaring a variable leads to a memory allocation which has a cost.
- Beware of the costs of recursion.
- Profile your solution to determine what to optimize.

Java Help

How do I test my program?

The `main` method in `SpeedDemon.java` can be used to facilitate testing of your solution. You should specify the filename of the file to process as the first command line argument when running the program. There are 2 ways to do this:

Firstly, you can input command line arguments with IntelliJ. From the menu, go to `Run > Edit Configurations`. Fill the `program arguments` field with the filename. You may use the provided example file, `example.in` for testing.

Secondly, you can input command line arguments through the CLI. Compile your java files first.

```
javac SpeedDemon.java
```

Then run the compiled program followed by the filename.

```
java SpeedDemon <filename>
```

The expected output for the example testcase can be found in `example.out`. The other testcases given in `databases.zip` also follow this format of `.in` file containing the input and `.out` files containing expected output.

Please remember to not modify the `main` method for contest submission, as it will be used to run your submissions.

How do I read in a file?

There are many different ways (and some may be faster than others). The easiest, perhaps, is to use the `BufferedReader` and `FileReader` classes:

```
import java.io.BufferedReader;
import java.io.FileReader;
```

The entire access of the file needs to be wrapped in a try/catch loop in order to catch any exceptions that may occur while reading the file:

```
try {
        // Code for accessing the file goes here
} catch (Exception e) {
        System.out.println(e);
}
```

The first thing you need to do is to open the file using the `FileReader`:

```
FileReader dataFile = new FileReader(fileName);
```

You then access the file via a `BufferedReader`:

```
        BufferedReader bufferedDataFile = new BufferedReader(dataFile);
```

You can then read the file:

```
        String line = bufferedDataFile.readline();
```

For more information on the BufferedReader and FileReader (and other file access mechanisms), see the Oracle Java Reference:

`https://docs.oracle.com/javase/8/docs/api/java/io/BufferedReader.html`

How do I measure how fast my program runs?

We have included several sample databases to test your program on. And remember the StopWatch class you used back in the Sorting Detectives Problem Set? That might be useful here too.