CS2040S Data Structures and Algorithms: Problem Set 3

Due: 14 October 2019, 2359 Hours

Harold Soh

This assignment is GRADED. Please take note of the due date and submit your solutions on the Leaderboard `http://cs2040s-speed-demon.dynu.net/`.

NOTE: Any modification or attempted modification to the Leaderboard and / or grading server will result in an immediate 0 for the Problem Set. You have been warned.

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 databaes are so large, it requires very fast algorithms and data structures that are highly optimized for maximum efficiency. In this problem set, you will try to develop the *fastest* algorithm for analyzing a large dataset.

Collaboration Policy. You are encouraged to work with other students on solving these problems. However, you **must** write up your solution **by yourself**. We may, randomly, ask you questions about your solution, and if you cannot answer them, we will assume you have cheated. In addition, when you write up your solution, you **must** list the names of every collaborator, that is, every other person that you talked to about the problem (even if you only discussed it briefly). Any deviation from this policy will be considered cheating, and will be punished severely, including referral to the NUS Board of Discipline.

Task 1: Speed Demon

(Total: 30 points)

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.

For the purpose of this problem set, 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*.

You can download the Problem Set file package `CS2040S_PS3_1920S1.zip` on Piazza.

Problem Details

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 consists of thousands, or even tens of thousands, of characters.) The maximum size of the input file will not exceed 1 Gigabyte, i.e. there will be no more than 1,073,741,824 characters in an input file.

You will code up a function:

```
public static int speedMatch(String dbfilename)
```

Your function will be passed the name of the database file as a parameter. For example, if the database is stored in the file `database.in`, then we will execute your function with the parameter `database.in`

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 character (ASCII character 10, '\n'). Note that entries may consist of any of the 128 legal ASCII characters, except for 10 and 13 (which indicate a new line). Characters may be repeated, and entries may be of any length up to M characters.

Output. Your program should calculate the number of pairs of entries that contain an identical set 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 write your output to `stdout`, which means **your code should not be directly writing any output to a file**. It should consist of an integer, followed by a newline character. The output is guaranteed to fit within a 32-bit signed integer, i.e. Java's `int`.

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 and you may use any available Java libraries.
- You may use any ideas or algorithms that you find on the internet, in books, etc. However, all the submitted code must be written by you.
- You may continue to update your solution up until the deadline.
- As long as your program is correct, you will get 20 points.
- The remaining 10 points will depend on the efficiency of your program. The final program speeds will be determined by our final test *after* the competition ends. We will use the most recent code you submit as your final submission. A program's speed will be computed as an average across a set of datasets. The score will not be relative, i.e., it is possible for *everyone* to get 10 points.
- Correctness is important. Your program may pass all the tests but you will lose points if your grader finds your algorithm is incorrect (e.g., your algorithm may report a pair to match despite having a different set of characters).

Hints

A few hints toward achieving good performance:

- First, develop and test a working solution that achieves good asymptotic performance, then improve it.
- Consider the performance of the data structures you are using and the actual costs of the operations involved.
- 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 the costs of recursion.
- Profile your solution to determine what to optimize.

Java Help and Submission Instructions

Question: How do I read in a file?
There are many different ways. 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.

Question: How do I test my code?
In the file package, you are given two Java files: `MySpeedDemon.java` and `SpeedDemonTester.java`. Your code should mainly be written in `MySpeedDemon.java`, although you are allowed to create other Java files or classes if you need to. To test on any particular testcase, run in your terminal the commands

```
javac MySpeedDemon.java
java MySpeedDemon tests/TestDB_1.in
```

This will compile and run your code, and read in the input file `TestDB_1.in` in the `tests` folder. The output will be printed in the terminal. You may replace `TestDB_1.in` with any other input file in the folder, or your own test inputs. Alternatively, you can run the commands

```
javac SpeedDemonTester.java
java SpeedDemonTester
```

This will run `MySpeedDemon` against the 5 pairs of `.in` and `.ans` files we provide in the `tests` folder, and check the result from your code against the expected result, and produce a `.out` file for the output of your algorithm for each test. The program will also output the time taken to run your algorithm for each input file, and the total sum of all the times.

You are allowed to modify `SpeedDemonTester.java` to test your own test inputs, but please **make sure you have both an input file and answer file**, with the file extensions `.in` and `.ans` respectively.

Question: How do I submit on the Leaderboard?
Instructions are provided on `http://cs2040s-speed-demon.dynu.net/submit`. You will be provided with a secret key to submit your code. **Please do not share this secret key**.

- Upload either a single java file, or a zip file, containing all the relevant .java files (.ZIP, *not* .7z, .rar. .tar.gz or whatever you can think of). If you have a custom class (e.g. HashTable.java), please include it in the zip.
- **Include the names of any collaborators as comments in the top of your main file.**
- Do not submit `SpeedDemonTester.java`, or any of the `.in`, `.out`, and `.ans` files.
- All your classes should not belong to any package (i.e. no `package cs2040S.ps3` at the top)
- Make sure your program follows the submission format exactly i.e. it should take in a filename as the first argument, and print only a single integer to `System.out` before exiting.

— END OF PROBLEM SET —