Com S 435/535 Programming Assignment 1
500 Points
Due: Sep 28, 11:59PM
Late Submission Due: Sep 29, 11:59PM(25% Penalty)

In this programming assignment, your will design a Bloom Filter and use it in a application. Description of a programming assignment is not a linear narrative and may require multiple readings before things start to click. You are encouraged to consult instructor/Teaching Assistant for any questions/clarifications regarding the assignment.

Your programs must be Java, preferably Java 8.1.

1 Bloom Filter

Recall that Bloom Filter is a probabilistic, memory-efficient data structure to store as set S. You will design the following classes/programs.

- BloomFiilterDet
- BloomFilterRan
- FalsePositives
- BloomDifferential
- NaiveDifferential
- EmpericalComparison

1.1 BloomFilterDet

Recall that to implement a Bloom filter, one needs to choose appropriate hash function(s). In lecture, we discussed two types of hash functions—*deterministic hash functions* and *random hash functions*. For this class, you must use the deterministic hash function 64-bit FNV. Your class should have following constructors and methods.

`BloomFilterDet(int setSize, int bitsPerElement).` Creates a Bloom filter that can store a set S of cardinality `setSize`. The size of the filter should approximately be `setSize * bitsPerElement`. The number of hash functions should be the optimal choice which is $\ln 2 \times$`filterSize/setSize`.

`add(String s).` Adds the string s to the filter. Type of this method is void. This method should be case-insensitive. For example, it should not distinguish between "Galaxy" and "galaxy".

`appears(String s).` Returns `true` if s appears in the filter; otherwise returns `false`. This method must also be case-insensitive.

`filterSize()` Returns the size of the filter (the size of the table).

`dataSize()` Returns the number of elements added to the filter.

`numHashes()` Returns the number of hash function used.

In addition to the above methods, you may include additional public/private methods and constructors.

Remark. To implement this class, you need k hash functions. However, 64-bit FNV is a single hash function, that gets a String as input and outputs a single hash value (a 64 bit int). To implement the filter, you need to generate k hash values. How can you do that? Think about it.

1.2 BloomFilterRan

This class is exactly same as before except you can use your own choice of a random hash function. Recall that in the lecture, we discussed two candidates for random hash functions. You should use the hash function that randomly picks a and b and the hash function is defined as $(ax + b)\%p$.

1.3 FalsePositives

Theoretically, the false positive probability is $(0.618)^{\texttt{bitsPerElement}}$. However, this is under the assumption that the hash functions are picked uniformly at random. In practice, it is not feasible to pick (and store) hash functions uniformly at random. So in practice false positive probability could be bit higher. How can you empirically evaluate the false probability of the filters that you designed above? Design an experiment to evaluate the false probability rate, and implement it. Write a program named `FalsePositives` to estimate the false positive probability of both the filters—`BloomFilterDet, BloomFilterRan`.

1.4 Application—Differential Files

Bloom filters can be used in some low-level applications that involve file management. And example is that of *differential files*. Consider a large database of records, where each record is of the form $\langle key, value \rangle$. *key* and *value* could be strings. Suppose that such a data is stored in a single file. Lets name this file as `database.txt`. This file has one record per line. Often we would like to make changes to records. However, it is very inefficient to make changes to the file `database.txt` every time a record is changed (and is prone to errors). A better approach is to create a differential file. Whenever a particular record is changed, then that record is written to a differential file (lets call this file `differential.txt`), and `database.txt` is not changed. Once the differential file becomes large enough (and when the load on the server is low), then `databse.txt` is updated and all contents of `differential.txt` are removed. Typically, the number of records in differential file are less than 10% of the number of records in `database.txt`.

Now consider the query mechanism with such system. User has a `key` and wants to retrieve the `value` associated with that key. For this the system should would first check whether `key` appears in the differential file. If it does not appear in the differential file, then the system will access `database.txt`.

Since the number of records in `differential.txt` are much smaller (10%) compared to number of records in `database.txt`, for most of the queries (90% of the time), the system accesses both the files `differential.txt` and `database.txt`. This makes it somewhat inefficient. A solution

to speed up this process is to store `differential.txt` (or just the keys of `differential.txt`) in main memory. However, this consumes more memory as number of records in `differential.txt` can be quite large. A solution is via using Bloom filters. Create a Bloom filter of all `keys` in `differential.txt`. Store the filter in main memory. When a key arrives as query, first check if that key appears in the Bloom Filter. If it appears in the filter, then consult `differential.txt`, otherwise access `database.txt` directly. Now, for most keys that do not appear in `differential.txt`, the system does not consult `differential.txt` (unless there is a false positive). This is more efficient.

Your job is to write a program that simulates this. Your are given the following files.

- `database.txt`: This is the data base file. Each line of this file is of the following from:

  `word1 word2 word3 word4 year1 n1 m1 year2 n2 m2 .....`

  For example a record would look like

  `Archbishop had given him 1720 8 6 1727 10 4 1758 20 6 .....`
  This means the 4-word phrase `Archbishop had given him` appeared 4 times in 6 books in year 1720, and appeared 10 times in 4 books in year 1727 and so on. Here key is the 4-word phrase.

  This file was created by processing data from Google Books Ngram Viewer (`http://storage.googleapis.com/books/ngrams/books/datasetsv2.html`). This file has nearly 12 Million records.

- `DiffFile.txt`. This is the differential file. This has nearly 1.2 Million records. This is created by picking random records from `database.txt` and making changes to those records.

- You are also given a file named `grams.txt`. This contains all keys that appear in `database.txt`.

Your task is the following. Write a two programs named `BloomDifferential` and `NaiveDifferential` that does the following. Program `BloomDifferential` has

- a method named `createFilter()` that returns a Bloom Filter corresponding to the records in the differential file `DiffFile.txt`.

- a method named `retrieveRecord(String key)` that gets a key as parameter and returns the record corresponding to the record by consulting the Bloom Filter first.

Program `NaiveDifferential` also has a method named `retrieveRecord(String key)` that returns the record corresponding to the key. It does not use Bloom Filter.

How do you compare the performances of the programs `NaiveDifferential` and `BloomDifferential`? Finally, design a program named `EmpericalComparison` that empirically compares the performances of both the programs.

2 Report

Write a brief report that includes the following

- For each class that you created, list specifications of all public and private methods that you have written.
- For the class BloomFilterDet, explain the process via which you are generating k-hash values, and the rationale behind your process.
- The random hash function that you used for the class `BloomFilterRan`, again explain how you generated k hash values.
- The experiment designed to compute false positives and your rationale behind the design of the experiment.
- For both the filters `BloomFilterDet` and `BloomFilterRan` report the false probabilities when `bitsPerElement` are 4, 8 and 10. How do false positives for both classes compare? Which filter has smaller false positives? If there is a considerable difference between the false positives, can you explain the difference? How far away are the false positives from the theoretical predictions?
- Describe the how `EmpericalComparison` is comparing the performances of `BloomDifferential` and `NaiveDifferential`. Explain the rationale behind your design.

3 Guidelines

An obvious choice is to store the bits of the filter in a Boolean array. However, in Java, Boolean array uses a lot more memory. Each cell of the Boolean array may use up to 4 bytes! This defeats the purpose of creating Bloom Filters. Thus your code must use the class `BitSet`. See Java API for more on this class.

Your code should not use any of the Java's inbuilt functionalities to create hash tables. For example, your code should not use classes such as `Hashtable` or `HashMap` in `BloomFilterDet` and `BloomFilterRand`. If you wish you may use the method `hashCode()` from the class `String`. This method returns converts a String to an int.

You are allowed to work in groups of size 2. You are allowed to discuss with your classmates for this assignment. Definition of *classmates*: Students who are taking Com S 435/535 in Fall16. However, You should write your programs and the report alone, without consulting other groups. In your report you should acknowledge the students with whom you discussed and any online resources that you consulted. This will not affect your grade. Failure to acknowledge is considered *academic dishonesty*, and it will affect your grade.

4 What to Submit

Please submit all .java files and the report via blackboard. Your report should be in pdf format. Please do not submit .class files. **Only one submission per group please.**

Have Fun!