Com S 435/535 Programming Assignment 1
500 Points
Due: Oct 2, 11:59PM
Late Submission Due: Oct 3, 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 in Java, preferably Java 8.1. **You are allowed to work in groups of size 2**. Please do not forget to read the guidelines before you start implementation.

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.

- BloomFiilterFNV
- BloomFilterRan
- BloomFilterMurmur
- DynamicFilter
- FalsePositives
- BloomJoin

1.1 BloomFilterFNV

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.

`BloomFilterFNV(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 BloomFilterMurmur

This class is exactly same as before expect that you will use `murmur` hash function instead of FNV. There are several variants of `murmur`, you must use the 64-bit function implemented at `http://d3s.mff.cuni.cz/ holub/sw/javamurmurhash/MurmurHash.java`.

1.3 BloomFilterRan

This class is exactly same as before except you can use your own choice of a random hash function. You should use the following random hash function: Pick smallest prime p that is at least tN, randomly pick a and b from $\{0, \cdots, p-1\}$, and the hash function is defined as $(ax+b)\%p$. Here t denotes `bitsPerElement` and N denotes the `setSize`.

1.4 Dynamic Bloom Filters

Bloom filters assume that the size of the set that needs be stored is known in advance. If our goal is to use 8 bits per element and the set S has N elements, then we will create a filter of size (approximately) 8N. Note that if we add more than N elements to the filter, then the false positive rate will go up. In certain applications, we do not know the size of the set S—it is dynamically generated. Dynamic Bloom Filters address this. They work as follows.

Suppose that our goal is to use ℓ elements for each data item. We first assume that the data set size is 1000 and create a filter F_1 of size approximately 1000ℓ. When the number of elements that are added exceeds 1000, then we create another filter F_2 of size approximately 2000ℓ (this filter can hold 2000 elements). In general, the size of F_i is approximately twice more than the size of F_{i-1}.

Create a class named `DynamicFilter`. You must use `random hash function` for this class. This class is exactly same as the class `BloomFilterRan` except that its constructor takes only one parameter—number of elements per element.

1.5 FalsePositives

Theoretically, for non-dynamic filters 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 filters—`BloomFilterFNV, BloomFilterMurmur, BloomFilterRan, DynamicFilter`.

1.6 Application—Distributed Join

Recall join operation on two relations. For our purposes we only work with 2-ary relations, such a relation has a key and an attribute. Join is performed based on the attribute that is common to both relations. Let us call this common attribute as " join attribute". Consider following two relations $R1$ and $R2$.

Name	Sport
Harry	Baseball
Sally	Cricket
Harriet	Baseball
Harry	Basketball
Jim	Football

Sport	Team
Baseball	Royals
Football	Vikings
Baseball	Mets
Basketball	Bulls
Hockey	Sabres

Here `Sport` is the join attribute between the relations. Natural join of these two relations (based on join attribute `Sport`) is the following table.

Harry	Baseball	Royals
Harry	Baseball	Mets
Harriet	Baseball	Royals
Harriet	Baseball	Mets
Harry	Basketball	Bulls
Jim	Football	Vikings

Suppose that $R1$ is resides in one server and $R2$ is on a different server. How can they compute join? A natural solution is to send one of the tables to the other server. However, this incurs a huge communication cost. A solution to reduce communication is via Bloom Filters. The first server creates a bloom filter consisting all values corresponding to the join attribute'. Upon receiving this the second server prunes its table as follows: For each tuple $\langle a_1, a_2 \rangle$ in its relation check if a_1 (a_1 is a value corresponding to the join attribute) is in the filter or not. If it is in the filter, then place $\langle a_1, a_2 \rangle$ in to a new table/relation $R3$. Now second server sends $R3$ to the first server and the first server performs the joint of $R1$ with $R3$. The size $R3$ could be much smaller than the size of $R2$ and this decreases communication cost.

You goal is to use Bloom filters to simulate this. Your are given two relations $R1$ and $R2$ stored in two files. Assume that they are stored in a tabular format. Each row has exactly two columns. Also, assume that **the first column in both $R1$ and $R2$ is the join attribute**.

Write a class named `BloomJoin` with following method and constructor.

`BloomJoin(String r1, String r2)` where `r1` and `r2` refers to the absolute path of the files that hold relations $r1$ and $r2$.

`join(String r3)` Computes the join of $r1$ and $r2$ and write the resulting relation in file $r3$.

1.7 Data

You may use the data provided to test your program `BloomJoin`. This is synthetic data, each relation file has 2 Million entries. . You can download the data from the PA1 page.

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 classes BloomFilterFNV, and BloomFilterMurmur 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.
- Compare the performances of `BloomFilterRan, BloomFilterFNV, BloomFilterMurmur` when `bitsPerElement` is $4, 8$ and 16. 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? Empericaly compare their performance interms of time taken. How much time it takes to add and search (on average)?
- Create as set with 500000 strings and add the elements to `BloomFilterRan` and `DynamicFilter`. Empericaly, estimate the false positives for both filters. Is there a difference between the false positives? Can you explain the difference (if exists)?

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 implement filters. For example, your code should not use classes such as `Hashtable` or `HashMap` in `BloomFilterFNV`, `BloomFilterMurmur`, `DynamicFilter`, and `BloomFilterRand`. You may use Java's in-built hash tables for `BloomJoin`. 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 Fall 17. 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.

All public methods of the class must be exactly as specified. This includes, names of methods/classes, number of parameters, order of parameters and their types. Any deviation (however minor) from specifications will incur a penalty of at least 40%**. If you wish you write additional methods, they should be private.**

Your grade depends on correctness, false positive rates, and efficiency (memory as well as time) and quality of the report.

4 What to Submit

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

Have Fun!