Computer Science 535
Programming Assignment 1

Minghao Sun,Liangyu Tan

October 6, 2017

- For each class that you created, list specifications of all public and private methods that you have written

1. BloomFilterFNV
a) public BloomFilterFNV(int setSize, int bitsPerElement): constructor
b) public void add(String s): add string s to bloom filter
c) public boolean appears(String s): check if string s appears in bloom filter
d) public int filterSize(): return the size of bloom filter
e) public int dataSize(): return the size of data input to the filter
f) public int numHashes(): return the number of hash functions
g) private long fnvhash(int k, byte[] s): return the hash value, s is the byte array of input string and k is the value to produce different hash functions.
2. BloomFilterMurmur:
a) public BloomFilterMurmur(int setSize, int bitsPerElement): constructor
b) public void add(String s): add string s to bloom filter
c) public boolean appears(String s) : check if string s appears in bloom filter
d) public int filterSize(): return the size of bloom filter
e) public int dataSize(): return the size of data input to the filter
f) public int numHashes(): return the number of hash functions
g) private long mmhash(int k, byte[] s): return the hash value, s is the byte array of input string and k is the value to produce different hash functions.
3. BloomFilterRan:
a) public BloomFilterRan (int setSize, int bitsPerElement): constructor
b) public void add(String s): add string s to bloom filter

c) public boolean appears(String s) : check if string s appears in bloom filter
d) public int filterSize(): return the size of bloom filter
e) public int dataSize(): return the size of data input to the filter
f) public int numHashes(): return the number of hash functions
g) private long ranhash(int k, byte[] s) : return the hash value, s is the byte array of input string and k is the value to produce different hash functions.
h) private int findnextprime(int n) find the smallest prime that greater than n
4. DynamicBloomFilters:
a) public DynamicBloomFilters (int bitsPerElement): constructor
b) public void add(String s): add string s to bloom filter
c) public boolean appears(String s) : check if string s appears in bloom filter
d) private boolean isappears(int i,byte[] $s_char s$) : check if string s appears in ith bloom filter
e) public int filterSize(): return the size of bloom filter
f) public int dataSize(): return the size of data input to the filter
g) public int numHashes(): return the number of hash functions
h) public int getfilterNum(): return how many filter we used
i) private ranhash(int k, byte[] s) : return the hash value, s is the byte array of input string and k is the value to produce different hash functions.
j) private int findnextprime(int n): find the smallest prime that greater than n
5. FalsePositives
a) Public static void main(String[] args): main method to calculate the false positives of filters
b) public static String[] generateString(int number,Random rand, int bitsPerElement):generate random string array
6. Bloomjoin
a) public BloomJoin(String r1, String r2): constructor
b) public void join(String r3): compute the join of r1 and r2 and write the results to r3
c) private static void writefile(String fileName, String content): write the content to file named fileName
d) private static String[][] readWordsFile(String filename): read the file from filename and return the content
e) public static void main(String[] args): run test.

- For the classes BloomFilterFNV, and BloomFilterMurmur explain the process via which you are generating k-hash values, and the rationale behind your process.

The idea is give a number to mask FNV code each time. In those two classes, we choose a prime number named $SEED$, for every byte data is set to XOR ($SEED * i$). Then the FNV-64INIT value changes in each hash function. As the initial values are different, these hash function will generate different hash value independently. In this way, we generate k different hash functions, so the bloom filter should work as we expected.

- The random hash function that you used for the class BloomFilterRan, again explain how you generated k hash values.

From the definition of random hash function: $(ax+b)\%p$. Each time we give a and b two random values from $[0, p-1]$, where p is the smallest prime greater than the product of bitsPerElement and setSize. If we need k random hash function, we just repeat k times to generate a and b randomly. So we have k random hash function.

- The experiment designed to compute false positives and your rationale behind the design of the experiment.

To test the false positive of the filters, the experiment is designed as follows: first, generate a train string array which all the strings are unique and add them into the four kind of bloom filters created; Second, generate another test string array which also contains unique strings which are different with the previous ones and test how many positive responses. So all of these positive responses are false positive, as they are unique strings. The probability of positive results will be use the number of test string array's positive responses divided by test string array's size.

- 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? Empirically, compare their performance in terms of time taken. How much time it takes to add and search (on average)?

At first theoretical predictions of false positive(FP) for BloomFilterRan, BloomFilterFNV, BloomFilterMurmur should be the $(1-p)^{numHashes}$, because $p = 1/2$ and $numHashes = ln2M/N = ln2 * bitsPerElement$ such that the $Pr(FP) = (0.618)^{bitsPerElement}$. However theoretical predictions of false positive(FP) for DynamicBloomFilters is a little different, as we need multiple filters to store our data, when test whether or not a string appears in Bloom Filter, we need test all of those filter one by one. So the probability of a string in one bloom filter and also show true appear(Truth positive) is $P(TP) = 1-(0.618)^{bitsPerElement}$, as bloom filter are independent. Let n to be the number of filters, so the overall $P(TP) = (1-(0.618)^{bitsPerElement})^n$, so the overall $P(FP) = 1-P(TP) = 1-(1-(0.618)^{bitsPerElement})^n$.
The results as following:
bitsPerElement is 4

```
<terminated> FalsePositives [Java Application] /Library/Java/JavaVirtualMachines/jdk1.
-----------------------false positve test-----------------------
Input data size to bloom filter:1000000
Test false positive data size:1000000
bitsPerElement is 4
--------------------------add time-------------------------------
F N V bloom filter add time:0.976 seconds
Murmur bloom filter add time:0.699 seconds
Random bloom filter add time:0.877 seconds
Dynamic bloom filter add time:1.269 seconds
--------------------------search time-----------------------------
F N V  bloom filter search time:0.283 seconds
Murmur bloom filter search time:0.369 seconds
Random bloom filter search time:0.354 seconds
Dynamic bloom filter search time:1.094 seconds
------------------probability of false positive-----------------
The theoretical false positve should be:
                 14.5866 %
The theoretical false positve for dynamic should be:
                 75.8045 %
Bloom Filter     False Positive
 fnv             15.4990 %
 mur             15.5157 %
 Ran             28.0510 %
 dyn             77.9403 %
```

bitsPerElement is 8

```
<terminated> FalsePositives [Java Application] /Library/Java/JavaVirtualMachines/jdk1.
----------------------false positve test-----------------------
Input data size to bloom filter:1000000
Test false positive data size:1000000
bitsPerElement is 8
--------------------------add time-------------------------------
F N V bloom filter add time:1.794 seconds
Murmur bloom filter add time:1.279 seconds
Random bloom filter add time:1.638 seconds
Dynamic bloom filter add time:3.267 seconds
-------------------------search time-----------------------------
F N V  bloom filter search time:0.356 seconds
Murmur bloom filter search time:0.429 seconds
Random bloom filter search time:0.48 seconds
Dynamic bloom filter search time:2.599 seconds
------------------probability of false positive-----------------
The theoretical false positve should be:
                  2.1277 %
The theoretical false positve for dynamic should be:
                 19.3511 %
Bloom Filter     False Positive
 fnv              2.1576 %
 mur              2.1583 %
 Ran              2.9860 %
 dyn             19.3197 %
```

bitsPerElement is 16

```
<terminated> FalsePositives [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_11
----------------------false positve test-----------------------
Input data size to bloom filter:1000000
Test false positive data size:1000000
bitsPerElement is 16
--------------------------add time-------------------------------
F N V bloom filter add time:2.565 seconds
Murmur bloom filter add time:2.315 seconds
Random bloom filter add time:3.112 seconds
Dynamic bloom filter add time:4.449 seconds
--------------------------search time-------------------------------
F N V  bloom filter search time:0.38 seconds
Murmur bloom filter search time:0.374 seconds
Random bloom filter search time:0.496 seconds
Dynamic bloom filter search time:2.791 seconds
-------------------probability of false positive----------------------
The theoretical false positve should be:
                  0.0453 %
The theoretical false positve for dynamic should be:
                  0.4518 %
Bloom Filter     False Positive
 fnv              0.0438 %
 mur              0.0471 %
 Ran              0.0600 %
 dyn              0.4757 %
```

We use the unduplicated 1000000 string as input, and another unduplicated 1000000 which is unique from input string array as test. As both data are unique, such that, False positive can be seen the total number shows in bloom filter. So We use same input data input four kinds of bloom filters. And use same test data to test False positive. In order to avoid bias. I use a large data to test.

From the False positive results We can see the performances of BloomFilterFNV, BloomFilterMurmur are better than BloomFilterRan and DynamicBloomFilters. BloomFilterFNV(No.1), BloomFilterMurmur(No.2) has smaller false positives.

BloomFilterRan, DynamicBloomFilters have high False positive. BloomFilterRan is because may be the random function is well-distributed or randomly as FNV and Murmur, but still show a good performance. DynamicBloomFilters is because we use multiple filters as store and search every filters, the theoretical prediction is much higher than others.

When the bitsPerElement is 4 it has larger difference, with the bitsPerElement becoming larger, the difference between theoretical prediction and experiments is smaller. As if there high variance if the possibility is high, so the result will more randomly around the theoretical value. But when bitsPerElement is 16 the results show good math between theoretical prediction and results of experiments.

The add and search time, from small to large is BloomFilterFNV, BloomFilterMurmur, BloomFilterRan and DynamicBloomFilters. The runtime is depend on data size and bitsPerElement(which is linear related with number of hash function). From the result we can see, when bitsPerElement is 16, the time is almost 2 times of bitsPerElement is 8, and 4 times of

bitsPerElement is 4.

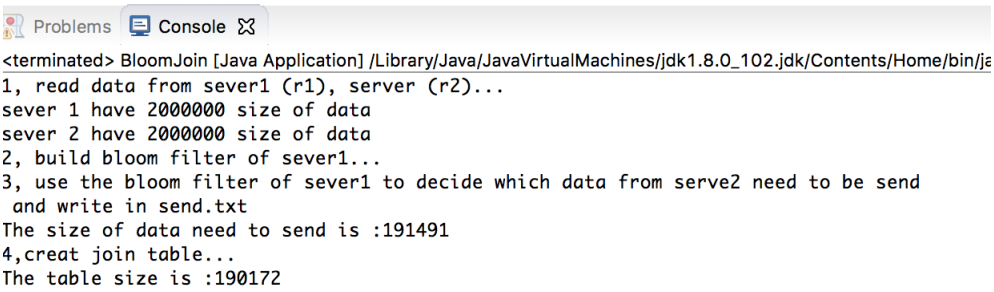
- Create a set with 500000 strings and add the elements to BloomFilterRan and DynamicFilter. Empericaly, estimate the false positives for both filters. Is there a difference between the false positives? Can you explain the difference (if exists)?

```
<terminated> FalsePositives [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.
----------------------false positve test-----------------------
Input data size to bloom filter:500000
Test false positive data size:2000000
bitsPerElement is 16
-------------------------add time-------------------------------
F N V bloom filter add time:1.279 seconds
Murmur bloom filter add time:1.176 seconds
Random bloom filter add time:1.573 seconds
Dynamic bloom filter add time:2.342 seconds
-------------------------search time-----------------------------
F N V  bloom filter search time:1.002 seconds
Murmur bloom filter search time:0.823 seconds
Random bloom filter search time:1.362 seconds
Dynamic bloom filter search time:6.883 seconds
------------------probability of false positive-----------------
The theoretical false positve should be:
                  0.0453 %
The theoretical false positve for dynamic should be:
                  0.4067 %
Bloom Filter     False Positive
 fnv              0.0449 %
 mur              0.0478 %
 Ran              0.0622 %
 dyn              0.4200 %
```

There are very small difference between theoretical prediction and results of experiments. But there are huge difference between BloomFilterRan and DynamicFilter.The theoretical predictions of false positive(FP) for BloomFilterRan should be the $(1-p)^{numHashes}$, because $p = 1/2$ and $numHashes = ln2M/N = ln2 * bitsPerElement$ such that the $Pr(FP) = (0.618)^{bitsPerElement}$. The theoretical predictions of false positive(FP) for DynamicFilter, we need test all of those filter one by one. So the probability of a string in one bloom filter and also show true appear(Truth positive) is $P(TP) = 1-(0.618)^{bitsPerElement}$, as bloom filter are independent. Let n to be the number of filters, so the overall $P(TP) = (1-(0.618)^{bitsPerElement})^n$, so the overall $P(FP) = 1-P(TP) = 1-(1-(0.618)^{bitsPerElement})^n$.

- Experiment results of $Bloomjoin$.

input R1 is path of Relation1.txt, R2 is path of Relation2.txt.



input R1 is path of Relation2.txt, R2 is path of Relation1.txt.