Annie Steenson

Project Assignment 1

Write Up

1. **For each class that you created, list specifications of all public and private methods that you have written.**
2. **For the classes BloomFilterFNV, and BloomFilterMurmur explain the process via which you are generating k-hash values, and the rationale behind your process.**
    a. **BloomFilterFNV**

I did two things to create k-hash values. The first thing was to switch between hashing the string normally and hashing it in reverse. I decided to do this because majority of words are not palindromes. Therefore, the hashing the string in reverse would most likely hash to a different location. The second thing I did was randomly choose two characters to swap within the string. I chose to do this to take advantage of random's close to even distribution.

    b. **BloomFilterMurmur**

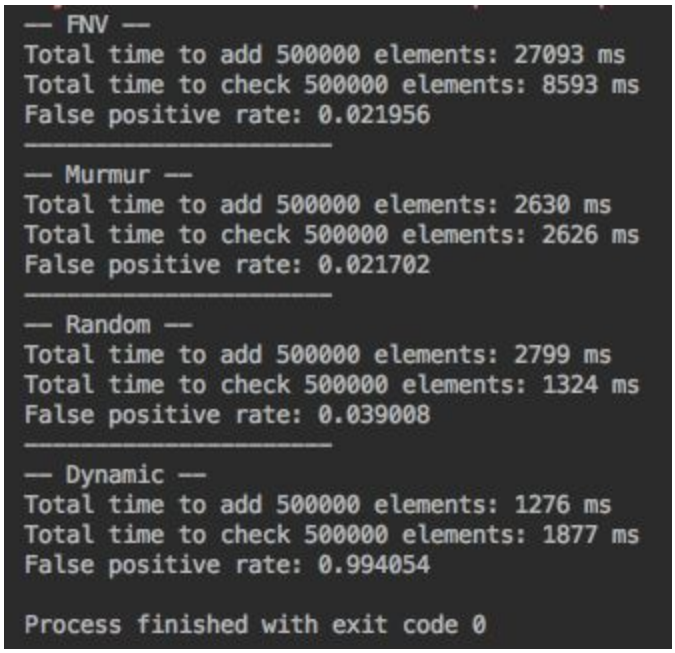
I used a random seed and and'ed it with the input. I did this to introduce more random variation in the hash function.

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

I generated k hash values by using by using the next smallest prime p that is atleast tN. T is bitsPerElement and N is setSize. Using p, I randomly picked two numbers a and b from {0… p-1}. Then used the formula (ax+b)%p. X denotes the hashCode of the string being passed. I used the hashCode because it returns the exact same integer for two equal Objects.

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

Output From FalsePositives.main()



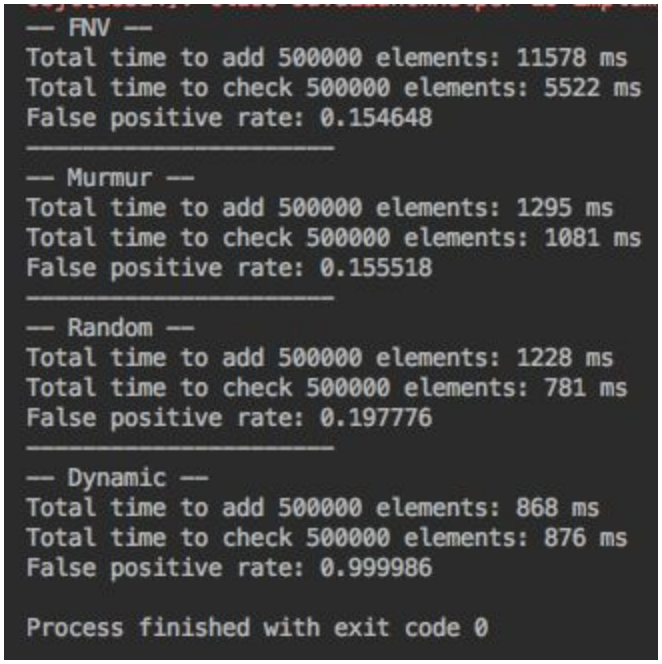
Where setSize = 500,000 and bitsPerElement = 8

I randomly generated 1,000,000 Strings. From those randomly generated Strings, I put 500,000 in the filter under test. For the other 500,000 strings, I checked if the filter responded with a false positive answer. The last 500,000 strings should have been for sure unique and definitely not in the filter.

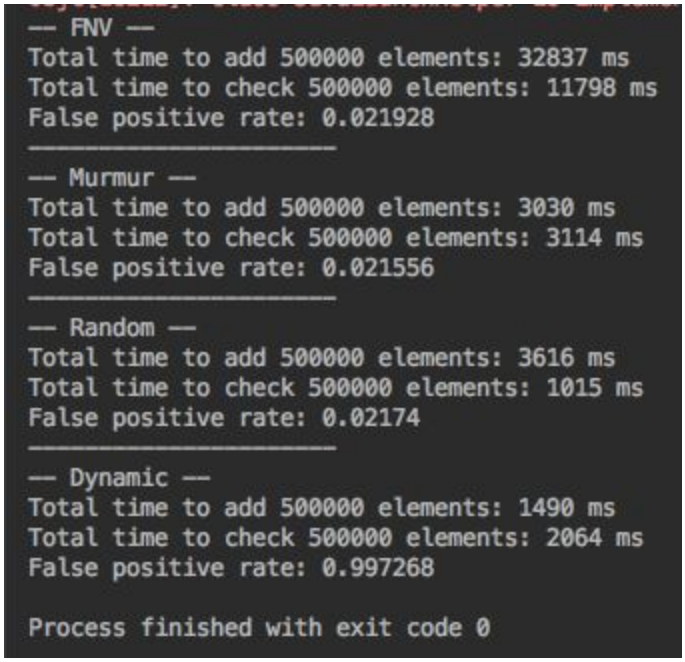
Then I counted up the number of false positive answers and divided by the number of overall elements that should have returned false.

Compare the performances of BloomFilterRan, BloomFilterFNV, BloomFilterMurmur when bitsPerElement is 4, 8 and 16.

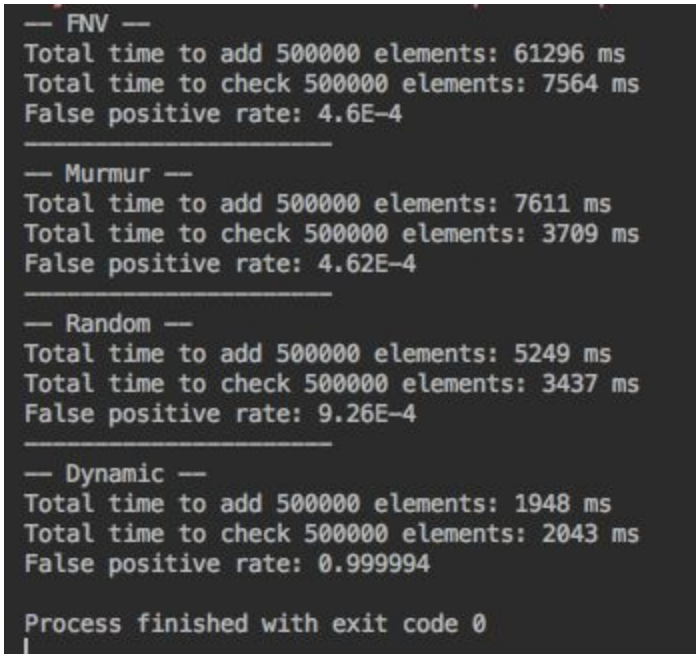
When bitsPerElement is 4



When bitsPerElement is 8



When bitsPerElement is 16



How do false positives for both classes compare?
Over a large filter, the random hash function does much poorer than FNV and Murmur. The FNV and Murmur both improve substantially with larger filters. That means the random hash function has a poor distribution and the FNV and Murmur have much better distributions.

Which filter has smaller false positives?
The FNV and Murmur go head to head for being the smallest false positive. Sometimes when the random seed for the Murmur is not a prime, it does slightly worse than the FNV.

If there is a considerable difference between the false positives, can you explain the difference?
There is a considerable difference with the Random hash function. As described above, it means that the random hash function has a poorer hash distribution. That means many elements hash to the same index. The FNV uses a unique and strongly proven special prime number. The Murmur hash function also uses a special number. My assumption is that these special numbers have unique properties that allow operations on them to close to evenly distribute hash values.

How far away are the false positives from the theoretical predictions?
The theoretical prediction for non-dynamic filters is (0.618)^bitsPerElement
For bitsPerElement = 4, the theoretical prediction = 0.14586594177
For bitsPerElement = 8, the theoretical prediction = 0.02127687297
For bitsPerElement = 16, the theoretical prediction = 0.00045270532

The FNV and Murmur hash functions are extremely close to the theoretical predictions. The hash function is close to the theoretical prediction depending on the primality of the the two random (a, b) numbers picked in the hash function.

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

Time to add 500,000 items (times in ms)

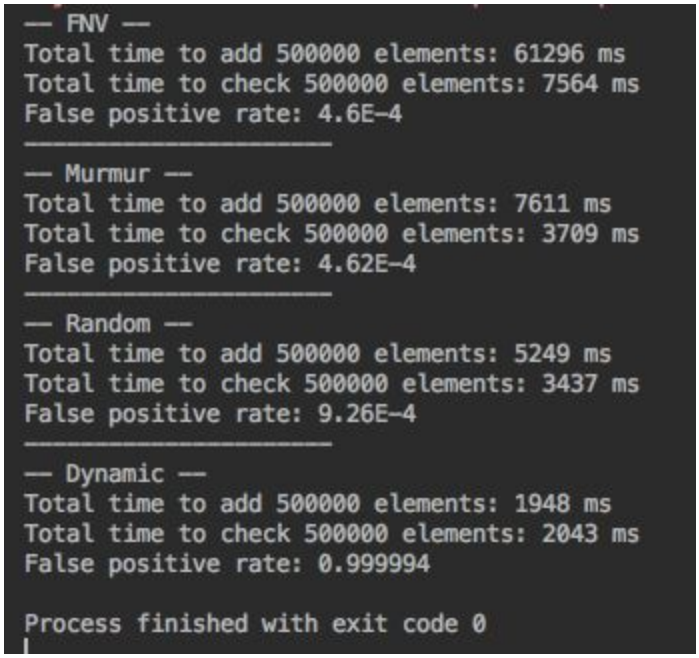
	FNV	Murmur	Random	Dynamic
Total	61296	7611	5249	1948
Average (total/500,000)	0.122	0.015	0.010	0.003

Time to search 500,000 items (times in ms)

	FNV	Murmur	Random	Dynamic
Total	7564	3709	3437	2043
Average (total/500,000)	0.15	0.007	0.006	0.004

For Murmur and Random, the search time is about half the time it takes to add

Create as set with 500,000 strings and add the elements to BloomFilterRan and DynamicFilter. Empirically, estimate the false positives for both filters. Is there a difference between the false positives? Can you explain the difference (if exists)?



The BloomFilterRan does much better than the Dynamic filter. The dynamic filter doesn't do great because it uses multiple filters. With multiple filters, one filter (the last one) goes unfilled. The RandomFilter it gets completely filled because it has a static size.

Resources

Murmur Has
http://d3s.mff.cuni.cz/~holub/sw/javamurmurhash/MurmurHash.java

Fnv-java
https://github.com/jakedouglas/fnv-java/blob/master/src/main/java/com/bitlove/FNV.java

Generating a random prime number in Java
https://stackoverflow.com/questions/24006143/generating-a-random-prime-number-in-java