Bloom Filter Report
Xander Coomes

Introduction

Hashing allows for more efficient storage of data. A bloom filter is a hashing scheme which guarantees constant time for insertions and check-contains operations. However, it lacks deletions and will output false positives at some rate. The following report analyzes the impact of different types of bloom filters on the false positive rate. The primary independent variables considered include: Type of Hash Function and Correlation of Data Hashed. The primary dependent variable considered is the false positive rate. Additional variables include Universe size ($|U| = N$), hashed elements (n), Mersenne Prime used(p), Number of Hash Functions(k), Size of the Hash Table ($m = n \cdot c$)

Hash Function Analysis

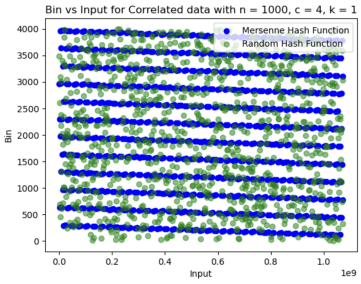
Two types of hash functions were analyzed: the Mersenne Hash Function, and the Random Hash Function. The Mersenne Hash Function is as follows, where a, b are random integers from $\{1, \dots p\}$

$$Bin(x) = ((a \cdot x + b) \mod p) \mod m$$

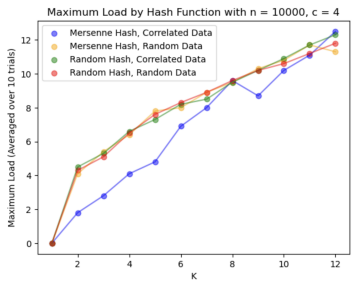
The Random Hash Function begins by seeding the random number generator as a function of s, a constant, predetermined seed, and x, the data being hashed. This way, the hash function will return the same bin given the same input. The hash function is as follows:

$$Bin(x) = \text{random} \in \{0, \dots, m\}$$

Two types of data hashed were analyzed: Random Data, and Correlated Data. Random Data was sampled from the universe U using the built-in python random number generator which uses a Mersenne Twister random number generation scheme. Correlated Data was taken by inputting n integers starting at 0, and ending at N, with a step size of $\frac{N}{n}$.



(a) Bin vs Input with Correlated Data



(b) Maximum Load vs K

Figure 1: Hash Function Statistics

In Figure 1a, the bin distribution of the Mersenne Hash function with Correlated Data appears to be in a nonrandom pattern, whereas the bin distribution of the Random Hash function with Correlated Data appears to be randomly distributed. This discrepancy is less prominent with higher values of c and k. For larger values of n, this pattern is less visible due to graphing software constraints. In Figure 1b, note that the max load of the Mersenne Hash Function with Correlated

Data is slightly lower than that of the other 3 function types, especially for smaller values of k. These findings align with the observation from Figure 1a, in that the Mersenne Hash Function with Correlated Data distributes the data more uniformly, causing a smaller maximum load. One possible reason for this is the Mersenne Hash Function uses algebraic operators, such as multiplication, addition, and modulus, and thus, given a correlated input, it is feasible that the output could be correlated. This correlation could lead to a more uniform distribution.

Bloom Filter Implementation

The parameters for the bloom filter are as follows: Universe Size $(N) = 2^{30}$, Elements Hashed $(n) = 10000$, Mersenne Prime$(p) = 2^{31} - 1$ (when using a Mersenne Prime Hash Function)

A value of $n = 10000$ was selected because the time taken to calculate the results is reasonably small, yet 10000 entries is still large enough to be rigorous. The Universe Size, (N) needs to be much larger than n, but smaller than the Mersenne Prime, and hence 2^{30} seems a reasonable number to achieve this. The Mersenne prime of $2^{31} - 1$ is large enough, yet doesn't cause overflow in python, and larger Mersenne primes, such as $2^{67} - 1$ produce similar results.

False Positive Rate Analysis

Theoretical Derivation

Our objective is to derive the false positive rate and ideal k given a bloom filter. Let S be the set of all elements in the bloom filter. In order to find the false positive rate, we need to find the probability that for $x \notin S$, all k hash functions hash x to a bin with value 1 in the bloom filter. Thus, we need to determine the probability that k bits are 1. To determine the probability that a specific bit is 1, we can determine the probability it is 0, and subtract 1 by that much. The probability that a specific bit is zero after $n \cdot k$ inserts is $(1 - \frac{1}{m})^{n \cdot k}$, which can be approximated as $n \to \infty$ to be $e^{\frac{-k}{c}}$. Thus, the probability that k bits are all equal to 1 will be $(1 - e^{\frac{-k}{c}})^k$, giving the false positive probability. To find the optimal value of k, we can minimize the false positive probability function by taking its derivative with respect to k and setting it equal to 0. This calculation yields the optimal k value of $c \cdot \ln 2$

Correlated Data: False Positive Rate Analysis

False positives are determined by inserting n elements into the bloom filter, either correlated or random. Next, n random numbers are selected which have not been inserted into the bloom filter. The number of false positives is divided by n, giving the false positive rate.

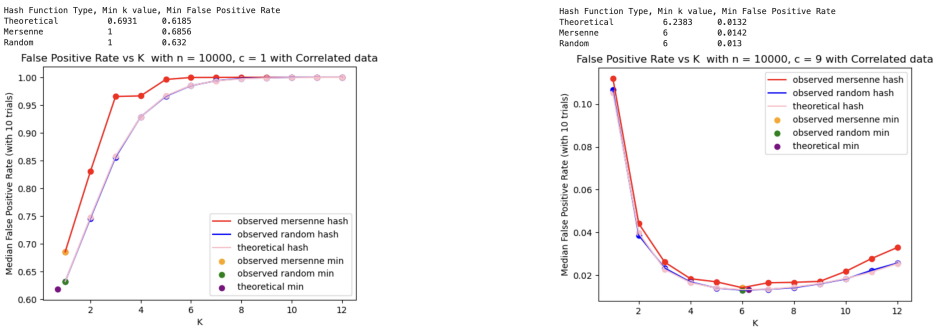


Figure 2: FPR vs K, Correlated Data, for $c = \{1, \ldots, 10\}$

As seen in Figure 2, for lower values of c, such as $1, 2, 3$, the Mersenne Hash Function has a higher false positive rate than the Random Hash Function (which performs identical to the Theoretical Hash Function) with Correlated Data. However, for larger values of c such as $\{5, \ldots, 9\}$ the Mersenne and Random Hash Functions perform nearly identical to the Theoretical Hash Function. For larger values of c, the observed optimal k value in both the Random and Mersenne Hash Functions align with that of the Theoretical Hash Function.

One reason for the discrepancy between the Mersenne Hash Function with Correlated Data and the other 3 combinations of datatype and hash function is because of the number of bins occupied The Mersenne Hash Function with Correlated Data occupies more bins in the bloom filter, likely a product of its distribution being more uniform accross all bins as seen in Figure 1a. In a bloom filter, once a value for k is fixed, a lower bin occupation rate will result in less false positives. A false positive for an entry x occurs when all k hash functions map x to bins that are occupied but x is not contained in the hash table. Thus, because the Mersenne Hash Function with Correlated Data has correlated distribution, it occupies more bins in the bloom filter. When the random data not contained in the bloom filter is used to check false positives, because more bins are occupied, the false positive rate is higher.

Random Data: False Positive Rate Analysis

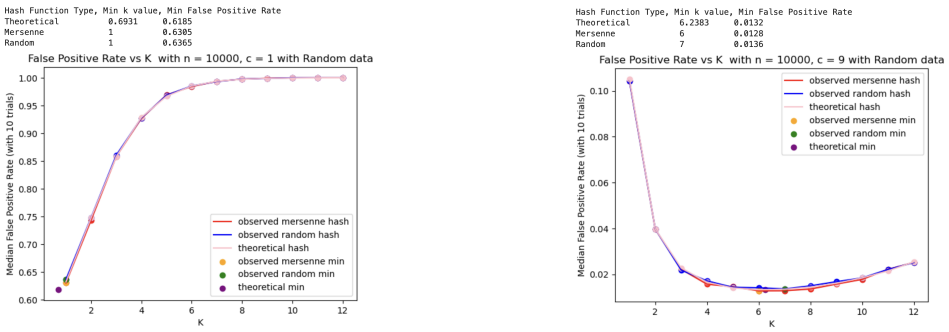


Figure 3: FPR vs K, Random Data, for $c = \{1, \ldots, 10\}$

As seen in Figure 3, with Random data entry, for all values of c, all the Mersenne Hash Function and Random Hash Function perform identical to the Theoretical Hash Function. For all values of c, the Mersenne and Random Hash Function optimal k align with the Theoretical Hash Function optimal k value.

The Random Hash Function requires more time than the Mersenne Hash Function, and Random Data requires more time than the Correlated Data, likely due to the cost of random number generation.

Conclusion

Given correlated data and limited storage space ($c = \{1, 2, 3, 4\}$), the Random Hash Function performs comparable to the Theoretical Hash Function, while the Mersenne Hash Function performs worse, having a higher false positive rate. However, given random data or more storage space, the Mersenne Hash Function and Random Hash Function are comparable in false positive rate, though the Mersenne Hash Function performs faster.