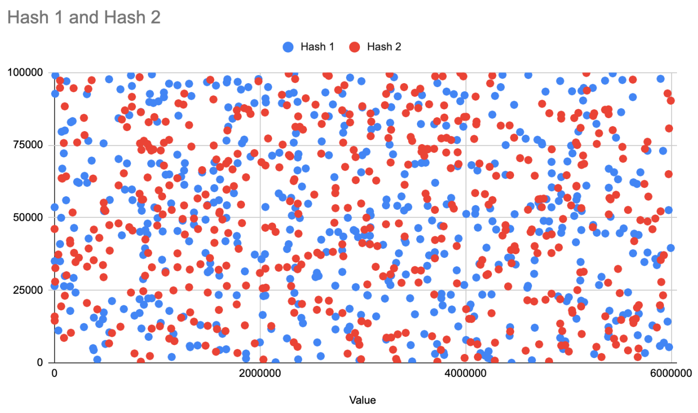
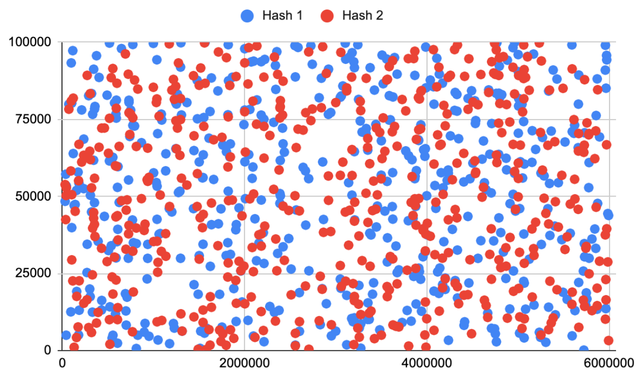
Tamer Tamer

CS 130A
Bloom Filter Project

Part 1

In the first part of this project, I implemented two different hash functions to use later in my bloom filter. The first function (Hash 1) uses a built-in random generator and a seed value to generate the hash value whereas my second function (Hash 2) makes use of the mod operation as well as random number generations to give me a Hash value. After implementing these functions, I wanted to test and analyze them. First, I wanted to check whether one of the hash functions worked better than the other for the same dataset. Therefore, I generated a bunch of random numbers (which in this case are the values on the x axis) and calculated their hash values using Hash 1 and Hash 2. I then proceeded to plot the values on plot 1 (first plot) shown to the right. As we can see from the above plot, the values are pretty spaced out, different and random whether using Hash 1 or Hash 2. Therefore, the choice of hash function did not really matter. Then, I wanted to test whether the dataset hashed mattered so I decided to redo the above plot but by plotting the first 2m even numbers and got the following plot:





If we compare plot 2 to plot 1, we can easily see that the hash values for plot 1 are more spaced out than the ones for the even numbers. This makes sense because hash functions are designed to distribute a wide range of input values as uniformly as possible across the hash space and making the dataset not completely random (i.e. choosing only even numbers) means we are lessening the variety of the input values, therefore affecting the efficiency of the hash function and their "random/spaced out" characteristics.

In summary, both hash functions were effective when it came to generating hash values for random numbers but the choice of dataset definitely matters. The more random the dataset is, the better and more spaced out the hash functions are.

Part 2

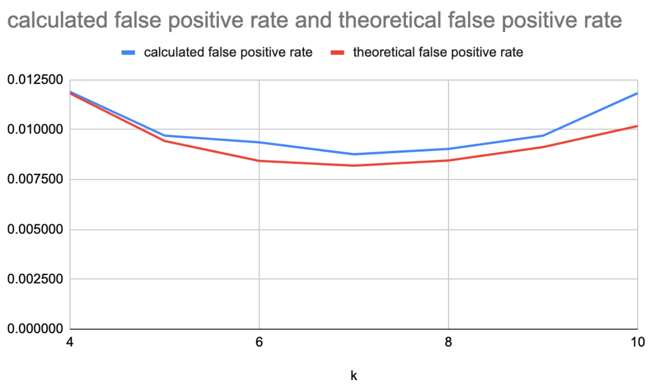
In this part of the project, we implemented the add and contains functionalities of our bloom filter. We first create our filter by creating an array of 0s of size m. For adding a value x to our bloom filter, we hash said value using either our hash function 1 or 2 and repeat this k times using different seeds based on the k value that were generated globally in part 1 of the project. To then check if the filter contains a specific value we check if the bloom filter value is 1 for either hash values from Hash 1 or Hash 2 k times (for all the k hash functions implemented). If they're all 1 in the filter (meaning if contains is incremented the same as the value of k) that means that the value x is probably in the filter and the function returns True. Otherwise, it returns False and the value is definitely not in the filter.

Part 3

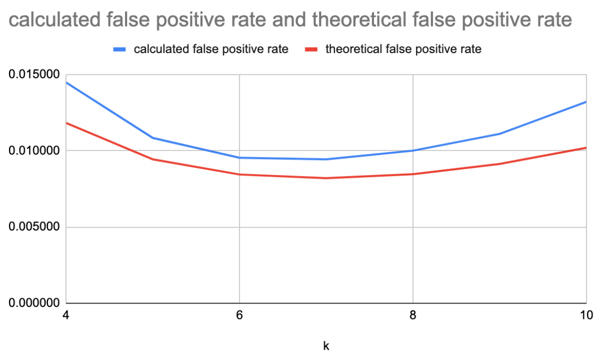
In this part, I'm going to analyze the false positive rates of the bloom filter and try and determine which hash function performed better. For that we're going to plot the theoretical false positive rate and calculated false positive rate vs k.

To get the theoretical values, we use the formula $f = (1 - e^{-\frac{k}{c}})^{k}$ and plug in the different values of k and c.

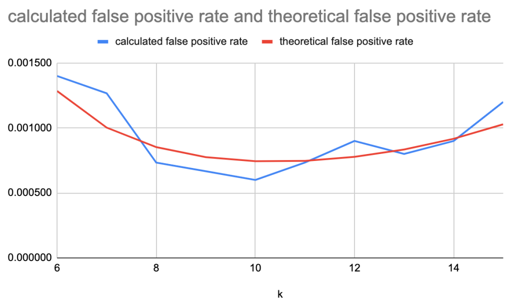
For hash function 1 and c=10, the theoretical optimal k = cln(2) = 10*ln(2) = 6.93 and we get the following plot:



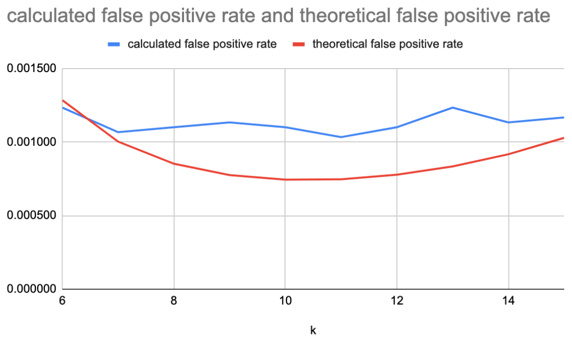
For hash function 2 and c=10, the theoretical optimal k = cln(2) = 10*ln(2) = 6.93 and we get the following plot:



For hash function 1 and c=15, the theoretical optimal k = cln(2) = 15*ln(2) = 10.4 and we get the following plot:



For hash function 2 and c=15, the theoretical optimal k = cln(2) = 15*ln(2) = 10.4 and we get the following plot:



From these plots, we can see that the values gotten are the closest to the theoretical ones using hash 1 for c=10. Even though the results for both hash functions could be considered pretty close and similar, the first hash function is closer to the theoretical curve and therefore, this function is considered more efficient in our case. For c =10, we can see that both hash 1 and hash 2 agreed with the theoretical optimal k being 7. For c = 15, we got an optimal k of 10.4 and we can see that hash 1 and hash 2 both gave optimal k between 10 and 11 which also agrees with our theoretical value even though it was not exact. By analyzing these plots, we can conclude that the value of c (affecting the size of the filter) as well as the hash function chosen for the bloom filter affects the results greatly: choosing c=10 instead of 15 gave us values more numerically consistent with the theoretical ones calculated and an optimal k value that was able to be reproduced exactly and choosing hash function 1 instead of hash function 2 allowed us to get values even closer to the theoretical ones. The parameters chosen for the bloom filter can have a great impact on its performance and how well it can recreate theoretical values calculated and the plots above demonstrate it well.