

# CS 630, Fall 2024, Homework 6

Yifei Bao

November 11, 2024

## Problem 1 *Bloom Filter Transformations*

1. In Bloom filter  $B$ , each element  $x$  has  $r$  hash functions that map the element to specific positions in the bit array of  $B$ , setting those bit positions to 1.

In  $B'$ , the original bit array is split into  $B_1$  and  $B_2$ , and  $B'$  is formed by taking the bit-wise OR of these two halves,  $B' = B_1 \vee B_2$ .

The new hash values in  $B'$  are computed as  $h_i(x) \bmod (m/2)$ . This means that any bit that was set in either  $B_1$  or  $B_2$  will be folded as a 1 in  $B'$  because of boolean or. Therefore, for every element originally in  $B$ , the corresponding bits will still be set in  $B'$ , and the lookup will always return positive.

2.

a. For items that were in  $B$  prior to creating  $B'$ , False negative is when an element that is in the set returns 0 during a lookup. Since  $B'$  is created by taking the bit-wise or of  $B_1$  and  $B_2$ , it keeps all the bit positions set by the original elements. Therefore, the false negative rate is 0.

False positive is when an element that is not in the set returns 1 during a lookup. The false positive rate of  $B$  is  $p = (1 - e^{-rn/m})^r$ . The false positive rate of  $B'$  is  $p' = (1 - e^{-rn/(m/2)})^r$ . Since  $m/2 < m$ , the false positive rate of  $B'$  is larger than that of  $B$ .

b. For items that were not in  $B$ , since they were not in  $B$ , the false negative rate is 0 still.

False positive rate is the same as in part a. The false positive rate of  $B$  is  $p = (1 - e^{-rn/m})^r$ . The false positive rate of  $B'$  is  $p' = (1 - e^{-rn/(m/2)})^r$ . Since  $m/2 < m$ , the false positive rate of  $B'$  is larger than that of  $B$ .

c.  $O(n)$  more items are inserted into  $B'$ , the false negative rate is still 0.

The false positive rate of  $B'$  is  $p' = (1 - e^{-r(n+n)n/(m/2)})^r = (1 - e^{-r(2n)n/(m/2)})^r$ .

## Problem 2 *Hash Evaluations for Bloom Filters*

1. When inserting an item into the Bloom filter, the hash functions are invoked  $k = \lfloor \ln 2(m/n) \rfloor$  times.

2. When checking an item that was inserted into the Bloom filter, the average number of invocations of the hash functions is  $k = \lfloor \ln 2(m/n) \rfloor$ . Because you know the item was inserted, you can check all the hash values to see if they are set to 1.

3. The probability that a bit is set to 0 after  $n$  items were inserted is  $p = (1 - 1/m)^{kn}$ . The average number of invocations of the hash functions is  $1/p$ .

## Problem 3 *hash*

1. We need at least two queries because we have two unknowns in a linear equation.

Time complexity:  $O(1)$ , space complexity:  $O(1)$ .

**Algorithm 1:** Calculate()

```

/* Call the hash function  $h(x) = ax + b \pmod{2^k}$  to get  $h(0)$  and  $h(1)$  */
1  $b = h(0)$ ;
2  $temp = h(1)$ ;
3  $a = (temp - b) \pmod{2^{64}}$ ;
4 return a, b

```

2. Since  $h(x)$  always results in an odd number, it is obvious that  $(ax + b) \pmod{2} = 1$ , which is the same as  $((a \pmod{2})(x \pmod{2}) + (b \pmod{2})) \pmod{2} = 1$ . When  $x$  is even, then the equation becomes  $(b \pmod{2}) \pmod{2} = 1$ , that  $b$  is odd. When  $x$  is odd, then the equation becomes  $(a \pmod{2}) = 0$ , that  $a$  is even.

To avoid the problem, we can make sure that  $a$  is odd, then the hash function will produce both even and odd values.

3. To identify a set of keys that could be chosen to cluster in  $1/1024$  of the keys, we can choose  $x = n * 2^{k-10}$ , thus  $h(x) = a(n * 2^{k-10}) + b \pmod{2^k}$ . In this way, the keys are clustered in  $1/1024$  of the keys.

To avoid the problem, we can select a prime  $m$  close to  $2^k$ , and the hash function is  $(ax + b) \pmod{m}$ . Using a prime modulus breaks the structure that the attacker exploits because powers of two have properties that avoid for such attacks.