Bloom Filters

Dimitris Spyropoulos

Bloom filters are probabilistic space-efficient data structures. They are used exclusively for membership existence in a set. These structures allow users to check if an element *most likely* belongs to a set, or if it *absolutely* doesn't. Bloom filters were conceived by Burton Howard Bloom in 1970.

1. Applications of Bloom Filters

The classic example is using bloom filters to reduce expensive disk lookups for non-existent keys. Google's Bigtable, Apache Cassandra and Postrgresql use them to reduce disk lookups for non-existent rows or columns. Avoiding costly disk lookups increases the performance of a database query operation. Google Chrome browser used to use bloom filters to identify malicious URLS. Any URL was first checked against a local Bloom filter and only if the Bloom filter returned a positive result, a full check of the URL was performed.Bitcoin uses this to speed up wallet synchronization. Even Youtube uses them to make new recommendations to users.

2. Description

Bloom filters are composed of a bit array of m bits. Initially, all bits of the table are set to 0 . The bit array is packed with k different hash functions.

A hash function is any function that is used to map data of arbitrary size to fixed-size values. It takes as input a key and outputs a hash code or digest to index a table. Generally a good hash function should be i) fast to compute and ii)minimize duplication of output values(collisions).

In bloom filters all *k* hash functions are used in element additions and existence checks. Every time we want to add an element to the bloom filter, all k functions are being executed with this element as key. The output of each one of them is a position in the bit array. As a final step, the bits of the outputs' positions are set to 1.

Every time we want to check the existence of an element, we again feed all *k* hash functions with the element, taking in the output *k* array positions. If any of the bits at these positions is 0, the element is *definitely not* in the set. If it was in the set, all the bits would have been 1 during element's insertion. If the bits at all *k* positions are 1, then either the element belongs to a set or the bits have been set to 1 during the insertion of other elements, resulting in a false positive.

3. Example

In this section, we present an example using Bloom filters. We assume having an array of 10 bits that are all set to 0. Also, we assume using two simple hash functions:

1) $h_1(x) = x \ mod \ 10$

2) $h_2(x) = (5x + 4) \ mod \ 10$

Initially, we have the following (empty) bloom filter:

0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0

Insertions

We want to insert to the bloom filter the elements of the set A = [19, 132, 25].

*i)*To insert 19 in bloom filter, we compute the digests of h_1, h_2 :

$h_1(19) = 19 \ mod \ 10 = 9$

$h_2(19) = (5 * 19 + 4) \ mod \ 10 = 99 \ mod \ 10 = 9$

Then, the bit in 9th position of the filter is set to 1. After insertion of 19 the filter is :

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	0	0	0	0	0	0	0	0	**1**

ii) Similarly, to insert 132 we compute $h_1(132) = 2$ and $h_2(132) = 4$. Then the bits in positions 2 and 4 are set to 1. The filter now is :

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	0	**1**	0	**1**	0	0	0	0	1

iii) Finally, regarding insertion of 25, digests of hash functions are $h_1(25) = 5$ and $h_2(25) = 9$. Bit 5 is set 1. Bit 9 is already 1, as it has been set by the insertion of element 19. Bloom filter after the insertion of 25 is :

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	0	1	0	1	**1**	0	0	0	**1**

Existence Checks

Now, we check using the previously formed Bloom filter the existence of the elements 133,25 and 24 in the set A.

i) To check if element 133 exists in A, we first compute the digests of h_1, h_2:

$h_1(133) = 3$

$h_2(133) = 9$

Then we check whether the bits of positions 3 and 9 of the Bloom filter are set to 1. Although bit 9 is set, the bit in position 3 is 0. As a result, the filter returns NO.

ii) To check if element 25 exists in A, we compute $h_1(25) = 5$ and $h_2(25) = 9$. Then we check whether the bits of positions 5 and 9 of the Bloom filter are set to 1. Indeed, both bits are 1. So, Bloom filter returns YES. It is a true positive, as element 25 exists in the set.

iii) To check if element 24 exists in A, we compute $h_1(24) = 24 \ mod \ 10 = 4$ and $h_2(24) = 124 \ mod \ 10 = 4$. Then we check whether the bits of position 4 is set to 1. Indeed, it is set to 1. Although Bloom filter return again YES, element 24 does not exist in the set (resulting in a false positive).

4. Performance of Bloom filter

Generally, a Bloom filter is considered of high quality if it minimizes the false positive probability. This is strongly dependent of the number of bits *m* of the bit array as well as the number *k* of the hash functions(as well as their quality).

Given that we want to insert *n* elements to the filter and the desired false positive probability p , it has been proved that the filter should have size $m = -\frac{n \ln p}{(\ln 2)^2}$. Also, it has been proved that the optimal number of hash functions is $k = -\log_2 p$. As we can observe, it only depends on the desired false positive probability.

According to Kirsch-Mitzenmacher Optimization, instead of using k *different* hash functions we only need two hash functions. The digests of the k hash functions can be computed through the following transformation :

$g_i(x) = h_1(x) + ih_2(x), \ 0 \leq i \leq k - 1$

5. Quality of hash functions

The quality of hash functions plays a crucial role to bloom filter's performance. Hash functions should be independent and uniformly distributed. Also, they should be as fast as possible.

The speed of a hash function can be easily measured through running benchmarks on random input. Uniformity of a hash function can be measured using Chi-Square test. A useful Github repo that analyses quality of hash functions for Bloom Filter can be found here.

6. What we propose

For the needs of Assignment 1, we propose to construct a bloom filter with 3 hash functions. Those hash functions will be of your choice and you'll need to explain them in README. You are not obliged to use the functions of the above repo. Regarding the size of the bloom filter, we propose it to be equal to the first prime $p \geqslant 3 * |R|$, where R is the number of the records in the registry file.