Hands On 5: Bloom filters

Algorithm Design

Ferraro Domenico
559813

1 Problem

Consider the Bloom filters where a *single* random universal hash random function $h: U \to [m]$ is employed for a set $S \subseteq U$ of keys, where U is the universe of keys.
Consider its binary array B of m bits. Suppose that $m \geq c|S|$, for some constant $c > 1$, and that *both* c *and* $|S|$ *are unknown to us*.

1. Estimate the expected number of 1s in B under a uniform choice at random of $h \in H$. Is this related to $|S|$? Can we use it to estimate $|S|$?

2. Consider B and its **rank** function: show how to use *extra* $O(m)$ *bits* to store a space-efficient data structure that returns, for any given i, the following answer in *constant time*:

   $$\boldsymbol{rank}(i) = \#1s \; in \; B[1..i]$$

   [Hint] Easy to solve in extra $O(m \log m)$ bits. To get $O(m)$ bits, use prefix sums on B, and *sample* them. Use a *lookup table* for pieces of B between any two consecutive samples.

2 Solution

2.1 First question

First thing first let's say that $n = |S|$. Let's define the following indicator variable X_i and its probability to be equal to 1

$$X_i = \begin{cases} 1, & if \; B[i] = 1 \\ 0, & otherwise \end{cases}$$

$$\Pr[X_i = 1] = \Pr[B[i] = 1] = 1 - \Pr[B[i] = 0]$$

We have just a single hash function h. After one key x is added, we have the following

$$\Pr[B[i] = 0 \mid x \; added] = 1 - \Pr[h(x) = i] = 1 - \frac{1}{m}$$

After all the n keys have been inserted, we have

$$\Pr[B[i] = 0] = \left(1 - \frac{1}{m}\right)^n$$

Finally, we can compute $\Pr[X_i = 1]$ as

$$\Pr[X_i = 1] = \Pr[B[i] = 1] = 1 - \Pr[B[i] = 0] = 1 - \left(1 - \frac{1}{m}\right)^n$$

Estimate the total number of bits set to one

We can build the variable $Y = \sum_{i=0}^{m-1} X_i$ to estimate the total number of bits set to one

$$E[Y] = E\left[\sum_{i=0}^{m-1} X_i\right] = \sum_{i=0}^{m-1} E[X_i] = \sum_{i=0}^{m-1} 1 - \left(1 - \frac{1}{m}\right)^n = m - \sum_{i=0}^{m-1} \left(1 - \frac{1}{m}\right)^n = m - m\left(1 - \frac{1}{m}\right)^n$$
$$\cong m - me^{-\frac{n}{m}} = m(1 - e^{-\frac{n}{m}})$$

The expected number of 1s in B is $m(1 - e^{-\frac{n}{m}})$, so it is strongly related to $n = |S|$.

Given $\mu = E[Y] = m(1 - e^{-\frac{n}{m}})$ we can use it to also estimate $n = |S|$.

$$\mu = m\left(1 - e^{-\frac{n}{m}}\right) \Leftrightarrow \frac{\mu}{m} = 1 - e^{-\frac{n}{m}}$$
$$\Leftrightarrow 1 - \frac{\mu}{m} = e^{-\frac{n}{m}}$$
$$\overset{use\ logs}{\Longleftrightarrow} \ln\left(1 - \frac{\mu}{m}\right) = -\frac{n}{m}$$
$$\Leftrightarrow -m \ln\left(1 - \frac{\mu}{m}\right) = n$$

2.2 Second question

The baseline solution is to precompute and store the prefix sum of each element of the binary array B, i.e., for each element we will store how many 1s occurs before it. To compute $rank(i)$ for any given i, the procedure just needs to return the i^{th} value of the precomputed prefix sums. Space complexity is $O(m \log m)$ bits because we store m prefix sums and the maximum prefix sum value (i.e., the maximum number of 1s) is m and we need $O(\log m)$ bits to store such value.

Subdivide *B* into blocks

While we can answer to $rank(i)$ in constant time, we are still using too much space than required. To improve space usage, instead of storing all prefix sums, we can store just a portion of them: subdivide B into blocks of length L and, store the prefix sum of the block's ending position. In other words, for each block we precompute and store the prefix sum before the block starts. To compute the rank of position i the procedure sums this value to the number of 1s from the block starting position to i (which is computed by the procedure itself). With this approach less space is needed because less prefix sums are precomputed, but such $rank(i)$ procedure takes time proportional to the block size which is more than constant time. The following is an example binary array B of length $m = 16$, subdivided into blocks of length $L = \log m = 4$:

$$B = [0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1]$$
$$Prefix\ sums = \quad 3 \quad 4 \quad 7 \quad 9$$

To improve time complexity, it would be ideal to also precompute the prefix sum of any position inside the block. For example, given a block [0 1 0 1], we would like to know in advance that, from left to right:

- at the first position, the prefix sum is 0
- at the second, the prefix sum is 1
- at the third, the prefix sum 1 as well
- at the last position, the prefix sum is 2

Sample the blocks

Given a block, to achieve constant time complexity, we also need to retrieve such information in constant time. By sampling the block, we can use the block itself to point to the precomputed prefix sums. In other words, because a block is a combination of $0s$ and $1s$ it is an integer number which can be used to index to its precomputed prefix sums. But, to make it work, it is needed to precompute the prefix sums of any possible block.

The number of all possible blocks of length L is 2^L, so by following this approach we have a lookup table with 2^L rows and L columns. In row x we have the prefix sums of any position of block x. The following is an example with the previous binary array B and the same block length $L = \log m = 4$:

		Prefix sums			
	0000_2	0	0	0	0
	...	...	...	...	...
	0010_2	0	0	1	1
	...	...	...	...	...
Block	0101_2	0	1	1	2
	...	...	...	...	...
	0111_2	0	1	2	3
	...	...	...	...	...
	1101_2	1	2	2	3

Time and space complexity

This approach allows to compute $rank(i)$ in constant time. Space complexity is the space needed to store blocks' prefix sum and the table. The number of blocks is $\frac{m}{L}$ and the bits needed to store a prefix

sum is $O(\log m)$ so space needed to store blocks' prefix sum is $O\left(\frac{m}{L} * \log m\right)$ bits. The table has 2^L rows and L columns and an element of a cell can be stored in $O(\log L)$ bits, so table's space complexity is $O(2^L * L * \log L)$ bits.

Space complexity is dominated by table size. With $L = \frac{logm}{2}$ the table has $\sqrt{m}$ rows and $\frac{\log m}{2}$ columns so space complexity would be $O\left(\sqrt{m} * \log m * \log\log m\right) \leq O(m)$ bits.