University of Pisa

Department of computer science

Algorithm design

Fifth hands-on: Bloom Filters

Domenico Erriquez

1. PROBLEM

1. Consider the Bloom filters where a single random universal hash random function $h: U \rightarrow [m]$ 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.

   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:

   $$rank(i) = \#1 \ in \ B[1..i]$$

   [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 B between any two consecutive samples.

2. SOLUTION FIRST QUESTION

In the first point, we aim to estimate the expected number of 1s in a bloom filter B with m positions using a single universal hash random function. To accomplish this, we define m indicator variables, where m is equal to the number of positions in B. Each indicator variable X_i $i \in [o \dots m-1]$ is defined as follows:

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

We define a variable X as the sum of all indicator variables X_i. By calculating the expected value of X, we are able to estimate the number of 1s in the bloom filter B.

$$E[X] = E\left[\sum_{i=0}^{m-1} X_i\right] = E\sum_{i=0}^{m-1} E[X_i] = \sum_{i=0}^{m-1} Pr[X_i = 1]$$

The probability that $B[i] = 0$ after n keys have been inserted is:

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

So, we have the probability that that $B[i] = 1$ after n keys have been inserted is:

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

Defining $|S| = n$, after n insertion the expected value of X is:

$$E[X] = m * \left(1 - \left(1 - \frac{1}{m}\right)^n\right) \approx m(1 - e^{-\frac{n}{m}})$$

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

Defining $\mu = E[X] = m(1 - e^{-\frac{n}{m}})$ we can use it to estimate the $n = |S|$

$$\mu = m\left(1 - e^{-\frac{n}{m}}\right)$$

$$\frac{\mu}{m} = \left(1 - e^{-\frac{n}{m}}\right)$$

$$\frac{\mu}{m} - 1 = -e^{-\frac{n}{m}}$$

$$1 - \frac{\mu}{m} = e^{-\frac{n}{m}}$$

$$\ln\left(1 - \frac{\mu}{m}\right) = -\frac{n}{m}$$

$$n = -m\ln\left(1 - \frac{\mu}{m}\right)$$

3. SOLUTION 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 (when all the bits are set to 1) and we need $O(\log m)$ bits to store such value.

With the baseline solution is possible to answer $rank(i)$ in constant time, but we have to reduce the space required and reach the goal of $O(m)$ bits.

To improve space usage, instead of storing all prefix sums, we can store just a portion of them: we can subdivide B into blocks of length L and store the prefix sum of the block's ending position. With this approach less space is needed because less prefix sums are precomputed, but $rank(i)$ query takes time proportional to the block size which is more than constant time.

For example, having a binary array B with $m = 16$ positions, and the length of the blocks $L = Log_2\, m = 4$, the array P containing the prefix sums of the blocks is the following:

$$B = [0,0,1,0,0,1,0,1,0,1,0,0,1,0,1,1$$
$$P = [1,3,4,7]$$

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.

So to have constant time complexity we need to find a way to retrieve these information in constant time. A possible solution is to create a lookup table where we save for each row of the table the precomputed prefix sum of any possible block, and since each block is a combination of 0s and 1s it is an integer number and can be used to index to its precomputed prefix sum.
For example, having a binary array B with $m = 16$ positions, and the length of the blocks $L = Log_2\, m = 4$, we have the following lookup table T:

$(0000)_2$	0	0	0	0
…	…	…	…	…
$(0011)_2$	0	0	1	2
…	…	…	…	…
$(0100)_2$	0	1	1	1
…	…	…	…	…
$(1011)_2$	1	1	2	3
…	…	…	…	…
$(1111)_2$	1	2	3	4

The number of all possible blocks of length L is 2^L. So, the table will have 2^L rows and L columns. Finally, the solution to answer a query $rank(i)$ in constant time works like this:

$$rank(i) = \begin{cases} P\left[\frac{i+1}{L} - 1\right], & if\ i + 1\,\%\,L == 0 \\ P\left\lceil\frac{i+1}{L} - 1\right\rceil + T\big[B[L * C, L * C + L - 1]\big][i\,\%\,L], & otherwise \end{cases}$$

Where $C = \left\lceil\frac{i+1}{L}\right\rceil$

3.1. Space complexity

The space used is the space needed to store the block's prefix sums 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 $O(\frac{m}{L} * \log m)$ 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.