Università di Pisa

Fifth hands-on: Bloom Filters

Algorithm Design (2021/2022)

Gabriele Pappalardo
Email: g.pappalardo4@studenti.unipi.it
Department of Computer Science

March 2022

1 Introduction

The problem is composed in two parts:

1. 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. Estimate the expected number of 1s in B under a uniform choice at random of $h \in \mathscr{H}$. 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: $\text{rank}(i) = \#1s \in B[1..i]$

   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 Bloom Filter

A Bloom Filter is a probabilistic data structure, invented in 1970 by Burton Bloom, that allows to check wheter an element x belongs to a set S without storing it. A filter works with $k > 0$ hash functions $h_i : U \to [m]$ belonging to a universal hash family.

2.2 Estimate expected number of bits set

The first point of the hands on asks us to estimate the expected number of bits set in a Bloom Filter using only one hash function. We start defining a new random indicator variable X_i such that:

$$X_i = \begin{cases} 1 & \text{if } B_i = 1 \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Therefore, we can be build a new random variable $Y = \sum_{i=0}^{m-1} X_i$ to estimate the number of bits set to one. We define $n = |S|$, as the unknown value to find.

$$
\begin{aligned}
E[\![Y]\!] = E\left[\!\!\left[\sum_{i=0}^{m-1} X_i\right]\!\!\right] &= \sum_{i=0}^{m-1} E[\![X_i]\!] \\
= \sum_{i=0}^{m-1} 1 - (1 - \frac{1}{m})^n = (m-1)&(1 - (1 - \frac{1}{m})^n) \\
&= m(1 - (1 - \frac{1}{m})^n) \\
= m - m(1 - \frac{1}{m})^n &\simeq m(1 - e^{-\frac{n}{m}})
\end{aligned}
$$

Knowing that $\mu = E[\![Y]\!] \simeq m(1 - e^{-\frac{n}{m}})$, we can solve the equation to find the cardinality of the original set S, which is equal to n.

$$
\begin{aligned}
&\mu \simeq m(1 - e^{-\frac{n}{m}}) \\
\iff & \frac{\mu}{m} = 1 - e^{-\frac{n}{m}} \\
\iff & \frac{\mu}{m} - 1 = -e^{-\frac{n}{m}} \\
\iff & 1 - \frac{\mu}{m} = e^{-\frac{n}{m}} \\
\iff & \ln(1 - \frac{\mu}{m}) = -\frac{n}{m} \\
\iff & m\ln(1 - \frac{\mu}{m}) = -n \\
\iff & n = -m\ln(1 - \frac{\mu}{m})
\end{aligned}
$$

Thus, the number of elements in S is $n = -m\ln(1 - \frac{\mu}{m})$.

2.3 Rank Computation

The second point of the hands-on asks us to compute the rank of the bit array B, using only $O(m)$ bits of space. To reach the requested space complexity, we start off by the baseline solution, the one gave by the hint.

2.3.1 Baseline Solution

Prefix sums allow us to answer the rank function in constant time, in fact, we can build a new array P, that contains in each position i the number of ones up to i.

$$
P_i = \sum_{j=0}^{i} B_j \tag{2}
$$

Since the maximum prefix sum can be $|B| = m$ (when all the bits are set to 1), we need $O(\log m)$ bits to store each sum, bringing us to use $O(m \log m)$ space to store the entire array of prefix sums.

2.3.2 Requested Solution

To achieve the requested space $O(m)$ we sample the prefix sums array P and we create a new lookup table T, described below. First, we have to determine how many samples of the prefix sums we do need. We are going to use $2 * \log m$ samples of prefix sums. Second, how do we build T? We define $L = \frac{\log m}{2}$, and we split the bit array B in L parts. Splitting the bit array leave us with L portions, that we can use to index our lookup table T. How many strings of bits can be represented using L bits? That is, how many rows T will have? Exactly: $\#\text{rows} = 2^L$.

$$
\#\text{rows} = 2^L = 2^{\frac{\log m}{2}} = (2^{\log m})^{\frac{1}{2}} = \sqrt{m} \tag{3}
$$

We need $\sqrt{m}$ rows for T. How many columns will have the table T? #columns $= L = \frac{\log m}{2}$. Therefore, our lookup table T will have a size of $\sqrt{m}\frac{\log m}{2}$. Assuming we have a bit array of size $m = 16$, then the lookup table will look like as shown below.

$$\begin{matrix} (0 & 0)_2 \\ (0 & 1)_2 \\ (1 & 0)_2 \\ (1 & 1)_2 \end{matrix} \; T = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 1 \\ 1 & 2 \end{bmatrix} \tag{4}$$

Our table will contain the partial sums of each of the L possible portions as shown in Equation 4.

At the end, to compute the rank function we are going to need both prefix sums samples and the lookup table T.

2.3.3 Space Complexity

We are keeping $O(\log m)$ samples of prefix sums, which takes up to $O(\log m)$ bits, therefore $O((\log m)^2)$. Plus, for the lookup table T we are using $\sqrt{m}$ rows and $\frac{\log m}{2}$ columns, and for each cell we need $O(\log \log m)$ bits, thus $O(\sqrt{m} \log m(\log \log m))$ bits.

```
uint16_t rank(uint16_t B, uint16_t j) {
    // Assuming we have the sample prefix sums array P
    // and the lookup table T.

    // Size of the bit array B
    int m = 16;
    // How many portions B is splitted up
    int L = log2(m) / 2;

    // Row and Column indecies for the Lookup Table T
    int r = get_row(B, j);
    int c = j % L;

    return P[(int)(j / L) - 1] + T[r][c];
}

uint16_t get_row(uint16_t B, uint16_t i) {
    // Shift and mask to get the row index
    return (B >> (0x0C - (i * 4))) & 0x0F;
}
```

Listing 1: 'Rank function implemented using a bit array of 16 elements.'

At the end we have $O((\log m)^2) + O(\sqrt{m} \log m(\log \log m)) = o(m)$ total bits.