Lecture Notes: Count-min Sketch (Prelude)

Yufei Tao
Chinese University of Hong Kong
taoyf@cse.cuhk.edu.hk

26 Feb, 2012

We will revisit the *set membership* problem in this lecture. Let us recall its definition. The universe $\mathcal{U}$ is the set of integers $\{1, ..., U\}$. S is a subset of $\mathcal{U}$ of size n. We want to store S in a structure such that, given any integer $q \in \mathcal{U}$, we can determine whether q belongs to S. Also remember that we do not need to be correct all the time. Instead, it suffices to achieve the following probabilistic guarantees:

- If $q \in S$, the answer must be *yes*. Namely, in this case, no error is allowed.
- If $q \notin S$, the answer should be *no* with probability at least $1 - \delta$, where $\delta > 0$ is an arbitrarily small constant. Namely, in this case, the algorithm can err with a probability at most δ.

We have learned that the problem can be solved by the bloom filter using $O(n \lg \frac{1}{\delta})$ bits (note: *not* $O(n)$ words which is equivalent to $O(n \lg U)$ bits). Our description was adequate for practical needs: all the details have been provided for you to implement the structure. However, from a theoretical point of view, two issues have been left unresolved:

1. We assumed that there is a "random oracle" that allows us to use ideal hash functions. Review the bloom-filter lecture about what is an ideal hash function. Such a random oracle does not exist yet.
2. Our "analysis" of the bloom filter was sloppy. In fact, we never gave a rigorous proof of its theoretical guarantees, except to mention that such a proof indeed exists, but had to be left out because it was a bit too complicated for the interests of this course.

While the issue of "random oracle" is arguably not a very acute one – after all, many people can comfortably live with it because in practice many hash functions work just well enough to be considered as an oracle – the second issue is much more serious: it has left some vacuum in our knowledge system that prevents us from thinking we have learned everything inside out! We will deal with this next by providing a replacement of the bloom filter to solve the set membership problem. It turns out that we will kill two birds with one stone, because the solution requires only universal hashing!

The structure we will describe in fact resembles a more general technique, called the *count-min sketch*, we will discuss in detail later. The subsequent discussion provides an early look into that technique.

1 A new structure for solving the set membership problem

Our structure consists of d arrays $A_1, ..., A_d$. Each A_i ($1 \leq i \leq d$) is an array of l bits. The values of d and l will be determined later. Initially, every bit $A_i[j]$ for all $j \in [1, l]$ is 0. We choose independently d hash functions $h_1, ..., h_d$ from a universal family. Each hash function maps from domain $\{1, ..., U\}$ to $\{1, ..., l\}$. Remember that each h_i can be described by using only two integers of $O(\lg U)$ bits.

To build the structure, we go through each element $k \in S$. For each $i \in [1, d]$, set $A_i[h_i(k)]$ to 1 (if the bit is already 1, then do nothing). To answer a query q, simply check whether $A_1[h_1(q)], ..., A_d[h_d(q)]$ are *all* 1. If so, then we return *yes*; otherwise, return *no*.

2 Analysis

Arrays $A_1, ..., A_d$ require in total ld bits to store. Plus the storage of the $h_1, ..., h_d$, the total space consumption is $ld + O(d \lg U)$ bits.

It is easy to see that we always answer *yes* if $q \in S$. Next, we analyze the probability that we return *yes* if $q \notin S$. Fix any $i \in [1, d]$. For us to return *yes*, $A_i[h_i(q)]$ must be 1, or in other words, there is some $k \in S$ such that $h_i(q) = h_i(k)$. We say that a *clash* happens. The fact $q \notin S$ implies that $q \neq k$. Since function h_i is universal, we know:

$$\mathbf{Pr}[h_i(q) = h_i(k)] \quad \leq \quad 1/l.$$

By union bound, the probability that there exists *at least one* clash is at most n/l.

If our answer is *yes*, it means that, there must be at least a clash in every array A_i $(1 \leq i \leq d)$. Due to the independence of hash functions, we know that the probability for this to happen is at most $(n/l)^d$. To achieve error probability at most δ, we need:

$$\begin{aligned} (n/l)^d &\leq \delta \\ \Rightarrow d \log_2(n/l) &\leq \log_2 \delta \end{aligned}$$

Now it is time to fix d and l to make the above inequality hold. There are many choices possible. Let us set, for example, $l = 2n$. So the above inequality becomes:

$$\begin{aligned} d \log_2(1/2) &\leq \log_2 \delta \\ \Rightarrow d &\geq \log_2(1/\delta) \end{aligned}$$

Setting $d = \log_2(1/\delta)$, we thus have a structure that uses $O(n \lg \frac{1}{\delta} + \lg \frac{1}{\delta} \lg U) = O(\lg \frac{1}{\delta}(n + \lg U))$ bits.