CSE 632: Analysis of Algorithms II: Randomized Algorithms **Spring 2024**

Lecture 16: Hashing

Lecturer: Zongchen Chen

1 Bloom Filter

Let $U = \{0, 1, \dots, N-1\}$ be a huge universe of possible elements. Our goal is to maintain a subset $S \subseteq U$ of size m where $m \ll N$, so that we can efficiently check if an element $x \in S$ or not. For example, U is the set of all possible password strings, and S is the set of unacceptable passwords. We hope to have fast queries, small space, simple algorithms, and low false positive rate.

A Bloom filter maintains a 0-1 table/array H of size $n = cm$, and uses k hash functions $h_1, \dots, h_k : U \to \{0, 1, \dots, n-1\}$. The table H is initialized to be all-zero, i.e., for each $j = 0$ to $n-1$ we set $H[j] = 0$. We consider two tasks, inserting an element x into S and checking if an element $x \in S$ or not. Deletions are not allowed.

Algorithm 1 Insertion

Input: $x \in U$ to be inserted into S

for $i = 1$ to k **do**
    Compute $h_i(x)$
    $H[h_i(x)] \leftarrow 1$
end for

Algorithm 2 Membership query

Input: $x \in U$ (for which we want to check if $x \in S$)

for $i = 1$ to k **do**
    Compute $h_i(x)$
    if $H[h_i(x)] = 0$ **then**
        Return No & **Halt**
    end if
end for
Return Yes $\triangleright$ Happen iff $H[h_i(x)] = 1$ for all i

If $x \in S$, then Algorithm 2 outputs Yes always. Meanwhile, if $x \notin S$, then it might incorrectly output Yes; this is called a *false positive*. The false positive rate is the probability of getting a false positive, i.e., the failure probability. Our goal is to estimate the false positive rate as a function of k which is the number of hash functions used, and $c = n/m$ which is the ratio between sizes of the table H and the set S. Observe that increasing c (i.e., having a larger table H) always decreases the false positive rate; however, how it depends on k, the number of hash functions, is not so obvious. Our goal is to find the optimal choice of k for a given value of c.

For any location j, we have that

$$\Pr(H[j] = 0) = \left(\left(1 - \frac{1}{n}\right)^k \right)^m = \left(1 - \frac{1}{cm}\right)^{km} \approx e^{-k/c},$$

assuming m is large. Suppose the input x to Algorithm 2 is not in S. Then we have

$$\Pr(\text{output Yes}) = \Pr(\forall i : H[h_i(x)] = 1) \approx \left(1 - e^{-k/c}\right)^k.$$

Therefore, the false positive rate is approximately given by

$$f(c,k) = \left(1 - e^{-k/c}\right)^k.$$

Now fix c, and we aim to find the optimal k which minimizes $f(c,k)$. By calculus, the minimizer k turns out to satisfy

$$\frac{k}{c} = \log 2.$$

So, we set $k = (\log 2)c \approx 0.693c$ (or $c \approx 1.443k$). The false positive rate then becomes

$$\Pr(\text{false positive}) \approx \left(1 - e^{-k/c}\right)^k = \frac{1}{2^k}.$$

Therefore, increasing k (i.e., using more hash functions) while maintaining $k/c = \log 2$ (so c is increasing at the same time) allows us to decrease the false positive rate exponentially fast, at a price of larger time and space complexity.

Remark 1. When $k/c = \log 2$, we have

$$\Pr(H[j] = 0) \approx e^{-k/c} = \frac{1}{2}.$$

So H is a uniformly random 0-1 string after inserting all m elements.

2 Cuckoo Hashing

In Cuckoo hashing, we maintain a hash table H of size $n = cm$ and use two hash functions $h_1, h_2 : U \to \{0, 1, \dots, n-1\}$. Each element $x \in U$ has two possible locations $h_1(x)$ and $h_2(x)$ given by the hash functions. When inserting x, we use whichever is empty. If neither of the locations is empty, then we add x to one of the locations, say $h_2(x)$, and move the element y previously at $h_2(x)$ to the other possible location of y. Note that y is still at one of its possible locations $h_1(y)$ and $h_2(y)$. If the other location of y is empty, then we are done after moving y to there. Otherwise, we put y there and move the element z previously there to its other location. We will repeat this process until success.

A potential problem of Algorithm 3 is the cycle of moves which causes an infinite loop. If this happens, we start over with two new hash functions (rehash), and insert all elements of S from the beginning.

We observe and establish the following properties of Cuckoo hashing, assuming $c = n/m$ is sufficiently large:

- $O(1)$ query time and no errors;
- $O(1)$ expected insertion time;
- Probability of rehashing is at most $1/2$.

Definition 2 (Cuckoo Graph)**.** The Cuckoo graph G is a multigraph (with possibly multiple edges and self-loops) associated with the hash table H and hash functions h_1, h_2. Vertices of G are locations/entries of H, so there are n vertices. Edges of G show possible locations for all elements of S. More specifically, for each $x \in S$, add edge $\{h_1(x), h_2(x)\}$ and label this edge by x. So there are m edges.

We have the following crucial observation.

Observation 3. If G has no cycles (i.e., G is a forest), then all m insertions succeed.

Algorithm 3 Insertion

Input: $x \in U$ to be inserted into S

1: Compute $h_1(x)$
2: **if** $H[h_1(x)]$ is empty **then**
3:     $H[h_1(x)] \leftarrow x$ & **Halt**
4: **end if**
5: Compute $h_2(x)$
6: **if** $H[h_2(x)]$ is empty **then**
7:     $H[h_2(x)] \leftarrow x$ & **Halt**
8: **end if**
9: $y \leftarrow H[i]$ where $i = h_2(x)$
10: $H[i] \leftarrow x$
11: **if** $i = h_1(y)$ **then**
12:     Move y from i to $h_2(y)$
13: **else** $\triangleright$ $i = h_2(y)$
14:     Move y from i to $h_1(y)$
15: **end if**
16: Repeat for y (from step 9) if necessary

Algorithm 4 Membership query

Input: $x \in U$ (for which we want to check if $x \in S$)

Compute $h_1(x)$ and $h_2(x)$
if $H[h_1(x)] = x$ or $H[h_2(x)] = x$ **then**
    Return Yes
else
    Return No
end if

In the rest of this section, we assume that $n \geq 6m$, i.e., $c = n/m \geq 6$. For any locations i, j, and any element $x \in S$, the probability that $\{i, j\}$ is an edge of G labeled by x is equal to $2/n^2$ if $i \neq j$, and $1/n^2$ if $i = j$. First consider the probability of rehashing:

$$\begin{aligned}
\Pr(\text{rehashing}) &\leq \Pr(\exists \text{ cycle in } G) \\
&\leq \sum_{\ell=1}^{\infty} \Pr(\exists \text{ cycle of length } \ell \text{ in } G) \\
&\leq \sum_{\ell=1}^{\infty} n^{\ell} \cdot m^{\ell} \cdot \left(\frac{2}{n^2}\right)^{\ell} \\
&\leq \sum_{\ell=1}^{\infty} \left(\frac{2m}{n}\right)^{\ell} \\
&\leq \sum_{\ell=1}^{\infty} \left(\frac{1}{3}\right)^{\ell} = \frac{1}{2}.
\end{aligned}$$

The expected insertion time of an element x to some location i is controlled by:

$$\begin{aligned}
\mathbb{E}[\text{length of a longest path from } i] &\leq \mathbb{E}[\# \text{ of vertices connected to } i] \\
&= \sum_{j \neq i} \Pr(i \text{ and } j \text{ are connected}).
\end{aligned}$$

For any $j \neq i$, we have

$$
\begin{aligned}
\Pr\left(i \text{ and } j \text{ are connected}\right) &\leq \sum_{\ell=1}^{\infty} \Pr\left(\exists \text{ path from } i \text{ to } j \text{ of length } \ell\right) \\
&\leq \sum_{\ell=1}^{\infty} n^{\ell-1} \cdot m^{\ell} \cdot \left(\frac{2}{n^2}\right)^{\ell} \\
&\leq \frac{1}{n} \sum_{\ell=1}^{\infty} \left(\frac{2m}{n}\right)^{\ell} \\
&\leq \frac{1}{2n}.
\end{aligned}
$$

Therefore,

$$
\mathbb{E}\left[\text{length of a longest path from } i\right] \leq \frac{1}{2}.
$$