**CSE 632 (Fall 2019): Analysis of Algorithms II : Randomized Algorithms**

# Lecture 6-7 (9/13/2019, 9/18/2019): Hashing

*Lecturer: Shi Li* *Scriber: Kairui Wang, Xian Zhou*

## 1 Pseudo randomness

- Computers can not really produce true random numbers. So how can we design a randomized algorithm if we do not have an access to a source of randomness?
- "Randomness is in the eyes of the beholder": Whether the outcome of an experiment is random or not depends on the power of the person who sees it.
- We use "pseudo-randomness" in our program. Almost all the algorithms we shall design are not powerful enough to tell the difference between a true randomness and a pseudo-randomness. Thus they will behave the same as if they are given the true randomness.

## 2 Hashing

The problem of consideration is the following. We have a dictionary of words, and each word is associated with a record. Given a word, how can we locate its record as fast as possible? We assume these words come from a large universe $U$ (e.g., strings of length at most 50), and there is a small set $S \subseteq U$ of size $n = |S|$ and $n \ll |U|$. In an application $S$ can be static or dynamic (in which case elements can be added to and deleted from $S$ and we assume we always have $|S| \leq n$).

Assuming all words have $O(1)$-length. There are three data structures we can use to solve the problem.

- (Self-Balancing) Binary Search Tree
  - The drawback of using BST is that it takes $O(\log n)$ time for accessing a record.
- Prefix Tree (Trie)
  - It takes $O(1)$ time to look up a word. However, a drawback of using a trie is that adding/removing/search for a record is slow when the structure in stored in an external memory.
- Hash map
  - $O(1)$ insertion/deletion/searching time
  - easy to implement and fast even when the data structure is stored in an external memory.

Here is the idea of designing the hash map data structure. We define a "hash function" $h : U \to [m]$. For every word $u \in S \subseteq U$, we store $u$ and its record at the location indexed by $h[u]$. Hopefully $m = O(n)$ so that the memory we use is not so big.

If $u \neq v \in S$, but $h(u) = h(v)$, both their records will be stored at the same location and thus there might be a conflict. There are many ways to address this issue. But for this lecture, we use $m$ linked lists of records (instead of $m$ records), where the $i$-th linked list stores the records for all $u \in S$ with $h(u) = i$. We can perform the following operations: looking up $u$, inserting $u$ and deleting $u$. For all the operations, we first compute $i = h(u)$. For the lookup operation, we scan the $i$-th linked list to check if $u$ is there. For the insertion operation, we add $u$ to the beginning of the $i$-th linked list. For the deletion operation, we scan the $i$-th linked list and delete $u$ once we find it. The worst case running time for lookup and insertion operations is linear in the length of the $i$-th linked list.

To make sure that the linked list has small length, we use a random hash function $h$. Hopefully, the linked lists will be short in expectation.

# 3 Universal Hashing

One can try to choose $h$ randomly from the set of all functions from $U$ to $[m]$. This is equivalent to give every element $u \in U$ a random hash value $h(u)$ in $[m]$, independent of all the other elements. This perfect distribution for $h$ will guarantee that all the linked lists will have small expected length. However, the big issue is storing the function $h$ takes $\Theta(|U|)$ words, which is too big. Indeed, we shall show for the linked lists to be short in expectation, it suffices that $h$ comes from a *universal hash distribution.*

**Definition 1.** *A distribution $\mathcal{H}$ of hash functions $h : u \to \{1,\ 2,\ \ldots\ m\}$ is said to be universal if for every $u \neq v \in U$, we have*

$$\Pr_{h \sim \mathcal{H}}[h(u) = h(u)] = \frac{1}{m}.$$

Notice that the uniform distribution over all functions from $U$ to $[m]$ satisfies the above property and thus is a universal hash distribution. We first show that if $h$ is randomly chosen from a universal hash distribution, then a linked list is short in expectation.

**Lemma 2.** *Let $\mathcal{H}$ be a universal hash distribution with $m = 2n$. Let $h$ be a random hash function from $h$. Then for every $u \in S$, we have*

$$\mathbb{E}[\textit{length of } h(u)\textit{-th linked list}] \leq 1.5.$$

*Proof.*

$$\begin{aligned}
\mathbb{E}[\text{length of } h(u)\text{-th linked list}] &= \mathbb{E}[|\{v \in S : \ h(u) = h(v)\}|] \\
&= \sum_{v \in S} \mathbb{E}\left[1_{h(u)=h(v)}\right] \\
&= \sum_{v \in S} \Pr[h(u) = h(v)] = 1 + \sum_{v \in S \setminus \{u\}} \Pr[h(u) = h(v)] \\
&= 1 + (|S| - 1)\frac{1}{m} \\
&\leq 1.5.
\end{aligned}$$

This finishes the proof of the lemma. □

## 3.1 A Universal Hash Distribution

One method for constructing a hash family is based on the following simple observation: If we have two vectors $x \in \{0,1\}^n$ and $y \in \{0,1\}^n$, and $x \neq y$, and we randomly choose a vector $r \in \{0,1\}^n$, then

$$\Pr\left[\langle r, x\rangle \bmod 2 = \langle r, y\rangle \bmod 2\right] = \Pr\left[\langle r, x \oplus y\rangle \bmod 2 = 0\right] = \frac{1}{2}.$$

It would be convenient to use the field $\mathbb{F}_2$. Recall that the field contains two elements 0 and 1, and the "+" and "×" operations are defined as follows:

| + | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 0 |

| × | 0 | 1 |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |

The "-" operation will be the same as "+" operation for $\mathbb{F}_2$.

Thus, the observation can be simple as follows:

**Lemma 3.** *Let $x, y \in \mathbb{F}_2^n$ and $x \neq y$. Then,*

$$\Pr_{r \sim_R \mathbb{F}_2^n}\left[\langle r, x\rangle = \langle r, y\rangle\right] = \frac{1}{2}.$$

We will use the above lemma to define our hash function distribution. Let us assume $U = \mathbb{F}_2^u, m = 2^b$. We then randomly choose $b$ vectors $z_1, z_2, \cdots, z_b \in \mathbb{F}_2^u$. For simplicity let us define the matrix $Z \in \mathbb{F}_2^{b\times u}$ as

$$Z = \begin{pmatrix} z_1^{\mathrm{T}} \\ z_2^{\mathrm{T}} \\ \vdots \\ z_b^{\mathrm{T}} \end{pmatrix} \in \mathbb{F}_2^{b\times u}.$$

Then, we define the hash function $h$ as follows: for every $x \in U = \mathbb{F}_2^u$, we have

$$h(x) = \begin{pmatrix} \langle z_1, x\rangle \\ \langle z_2, x\rangle \\ \vdots \\ \langle z_b, x\rangle \end{pmatrix} = Zx.$$

**Lemma 4.** *For every $x \neq y \in U = \mathbb{F}_2^u$, we have*

$$\Pr[h(x) = h(y)] = \frac{1}{2^b} = \frac{1}{m}.$$

*Proof.* To have $h(x) = h(y)$, we must have $\langle z_i, x\rangle = \langle z_i, y\rangle$ for every $i \in [b]$. By Lemma 3, this happens with probability exactly $\frac{1}{2^b} = \frac{1}{m}$. □

So the hash distribution $\mathcal{H}$ we constructed is universal. Notice that we only need to store the matrix $Z$ in order to store a randomly sampled function $h$ from the distribution $\mathcal{H}$. So, we only need $ub$ bits to describe the function $h$.

# 4 Perfect Hashing

The universal hashing scheme gives a randomized structure where every linked list is short in expectation. However, it may be the case that with very large probability, some linked list will be long (say, of order $\omega(1)$). That is, some element will require $\omega(1)$ lookup time. The question of this section is the following: suppose the set $S$ of interesting words is static, can we design a hashing scheme where every $u \in S$ has $O(1)$ colliding elements?

Indeed, we can achieve an even stronger property: there are no collision pairs in the hash function scheme. First, we show that if $m$ is much bigger than $n$, with large probability there is no collision pairs.

**Lemma 5.** *Let $\mathcal{H}$ be a universal hashing distribution with $m = n^2$. Then, we have*

$$\Pr_h\left[\forall u \neq v \in S, h(u) \neq h(v)\right] \geq \frac{1}{2}.$$

*Proof.* For every $u, v \in S$, define $\text{same}(u, v) = \begin{cases} 1 & h(u) = h(v) \\ 0 & h(u) \neq h(v) \end{cases}$. Then,

$$\mathbb{E}\left[|\{\{u,v\} : u \neq v \in S, h(u) = h(v)\}|\right] = \mathbb{E}\left[\sum_{\{u,v\}} \text{same}(u,v)\right]$$
$$= \sum_{\{u,v\}} \mathbb{E}[\text{same}(u,v)] = \sum_{\{u,v\}} \frac{1}{m} = \frac{1}{m}\binom{n}{2} \leq \frac{1}{2}.$$

We used the linearity of expectation in the second equality.

Now we use Markov Inequality: Given a non-negative random variable $X$ with $\mu = \mathbb{E}[X]$, we have that $\Pr[X \geq t\mu] \leq \frac{1}{t}$ for every $t \geq 1$. Thus, with probability at most $1/2$, the number of $\{u, v\}$ pairs with $h(u) = h(v)$ is at least 1. This means with probability at least $1/2$, there are no collision pairs. □

We can repeatedly choose the hash function $h$ from $\mathcal{H}$ until we see no collisions. By the expectations of geometric distributions, we need to sample $h$ twice in expectation. Thus, we have constructed a hash scheme without collisions. However, a big issue with this approach is that the memory needed is still $\Theta\left(n^2\right)$, since we need to keep so many heads of linked lists.

## 4.1 A Two-Level Hashing Scheme

To address the above issue, we use two levels of hash functions. For the first level, we use a universal hash distribution $\mathcal{H}$ with $m = 2n$. Then every element $u \in S$ is supposed to be stored in the $h(u)$-th set. However, if there are $n_i \geq 2$ elements $u \in S$ with $h(u) = i$, we shall use a second-level universal hashing distribution $\mathcal{H}_\rangle$ with range size $m_i = n_i^2$ for the $n_i$ elements. As showed by Lemma 5, we can guarantee that there are no collisions between the $n_i$ elements, if we repeatedly select $h_i$ from $\mathcal{H}_\rangle$. Since we apply the procedure for every $i \in [m]$ with $n_i \geq 2$, there are no collisions in the overall two-level scheme.

It remains to bound the memory we need to use for the scheme. For the $i$-th set, we use $m_i = n_i^2$ and thus the memory we need is $O\left(\sum_{i=1}^m n_i^2\right)$. We show that this is small in expectation:

$$\mathbb{E}\left[\sum\nolimits_{i=1}^m n_i^2\right] = \mathbb{E}\left[\left|\,\{(u,v) : u, v \in S, h\,(u) = h\,(v)\}\,\right|\right] = n + \frac{n\,(n-1)}{m} \leq 1.5n.$$

We used the linearity of expectation for the second equality: for every $u = v \in S$, we have $\Pr[h(u) = h(v)] = 1$ and for every $u \neq v \in S$, we have $\Pr[h(u) = h(v)] = \frac{1}{m}$.

Using Markov inequality again, we have

$$\Pr\left[\sum_{i=1}^m n_i^2 \geq 3n\right] \leq \frac{1}{2}.$$

We can repeatedly choose the first level hash function $h$ from $\mathcal{H}$ until the above equality holds; again, we only need to sample $h$ twice in expectation.