Lecture 19 Hashing

The dynamic dictionary problem: We have a universe $\mathcal{U}$ of m elements. Without loss of generality let $\mathcal{U} = \{0, 1, \ldots, m-1\}$. That is, the universe $\mathcal{U}$ of size m is totally ordered.

There is a set S of keys and we assume that the keys are represented in a manner that permits us to perform arithmetic operations over them. Moreover, we will use the RAM model to its full generality.

- This approach avoids the lower bound of $\Omega(\log|S|)$ associated with data structures like balanced search trees and we can achieve $O(1)$ search time.

For the dynamic dictionary problem, we will create a table T of size m. A table is simply an array providing random access.

For each key $k \in \mathcal{U}$, $T[k] = 1$ iff $k \in S$. However this is not a desirable solution as we using space $m >> n$: here m is the size of $\mathcal{U}$ and n is the size of S.

Our goal is to reduce the space used to $O(|S|)$, while maintaining the property that a search or update operation takes $O(1)$ time.

A hash table: a table T consisting of n cells indexed by $N = \{0, 1, \ldots, n-1\}$ and a hash function h which is a mapping from $\mathcal{U}$ to N.

- each cell is a word in memory that holds an element of $\mathcal{U}$, i.e, the word size is $\log m$.

Ideally, we want the hash function to map distinct keys in S to distinct locations in T.
- a collision is said to occur if $h(x) = h(y)$ for $x \neq y$.

A hash function $h: \mathcal{U} \to N$ is said to be perfect for a set $S \subseteq \mathcal{U}$ if $h(x) \neq h(y)$ for $x \neq y$ where $x, y \in S$.
- a perfect hash function can be constructed for any subset S of size $\leq n$.

However perfect hash functions are useless for the dynamic dictionary problem since for any $n < m$, the function h must map some 2 elements of $\mathcal{U}$ to the same location. Thus h cannot be perfect for any set S with these 2 elements.

* So we will relax the definition of perfect hash functions to near-perfect hash functions. Near-perfect hash functions are allowed to cause a small number of collisions at each location in T.

Our goal now is to show a randomized hashing scheme for the dynamic dictionary problem that processes search and update operations in expected time $O(1)$. No assumptions are made about the operation sequence. The expectation is wrt random choices internal to T and h.

Universal hash families

Idea: Choose a family of hash functions $H = \{h: \mathcal{U} \to N\}$ where each $h \in H$ is easily represented and evaluated.

While one fixed $h \in \mathcal{H}$ may not be perfect for very many choices of S, we can ensure that for every set S of size $\leq n$, a large fraction of the hash functions in $\mathcal{H}$ are near-perfect for S in the sense that the number of collisions is small. Thus for any $S \subseteq \mathcal{U}$ of size $\leq n$, a random choice of $h \in \mathcal{H}$ will give the desired performance.

Definition. Let $\mathcal{U} = \{0, 1, \ldots, m-1\}$ and $N = \{0, 1, \ldots, n-1\}$ with $m \geq n$. A family $\mathcal{H}$ of functions from $\mathcal{U}$ to N is said to be 2-universal if $\forall\, x, y \in \mathcal{U}$, $x \neq y$, h is chosen uniformly at random from $\mathcal{H}$, we have $Pr[h(x) = h(y)] \leq 1/n$.

A totally random mapping from $\mathcal{U}$ to N has a collision probability of exactly $1/n$. Thus a random choice from a 2-universal family of hash functions gives a seemingly random function. Observe that the collection of all possible functions from $\mathcal{U}$ to N is a 2-universal family. This is a collection with n^m functions.

Our goal is to obtain smaller such families (those that contain a smaller number of functions). The reason it is possible that a random function $h \in \mathcal{H}$ can simulate a truly random function is because we only want pairwise independence between elements in $\{h(x) : x \in \mathcal{U}\}$. That is why the name is "2-universal".

Constructing such families

Choose a prime $p \geq m$. We will work over the field $\mathbb{Z}_p = \{0, 1, \ldots, p-1\}$.

For all $a, b \in \mathbb{Z}_p$, define the linear function $f_{a,b} : \mathbb{Z}_p \to \mathbb{Z}_p$ as follows:

$$f_{a,b}(x) = ax + b \pmod{p}.$$

Define the hash function $h_{a,b} : \mathbb{Z}_p \to \mathbb{Z}_n$ as:

$$h_{a,b}(x) = f_{a,b}(x) \pmod{n}.$$

We define the family of hash functions $\mathcal{H} = \{h_{a,b} : a, b \in \mathbb{Z}_p, a \neq 0\}$.

Claim. $\mathcal{H}$ is 2-universal.

We will soon prove the above claim. Let us make some simplifying assumptions first. Bertrand's postulate states that there is always a prime number sandwiched between m and $2m$, for every $m \geq 2$. So let us expand our universe and assume $\mathcal{U} = \{0, 1, \dots, p-1\}$

So $f : \mathcal{U} \to \mathcal{U}$.

Claim 1. For all $(x, y) \in \mathbb{Z}_p \times \mathbb{Z}_p$, $x \neq y$, the number of hash functions in $\mathcal{H}$ that cause a collision between x and y is $\leq$ number of pairs (r, s), $r \neq s$, such that $r = s \pmod{n}$.
$r, s \in \mathbb{Z}_p$

Proof. Suppose x and y collide under a specific function $h_{a,b}$. Let $f_{a,b}(x) = r$ and $f_{a,b}(y) = s$. We have $r \neq s$ since $f_{a,b}$ is a bijection. (Please check this.)

So a collision takes place $\Rightarrow r = s \pmod{n}$

Having fixed x and y, for each such choice of $r \neq s$, the values of a and b are uniquely determined as the solution to the following system of linear equations over the field $\mathbb{Z}_p$

$$ax + b = r \pmod p$$
$$ay + b = s \pmod p$$

The number of hash functions that cause x and y to collide is exactly the number of choices $r \neq s$ such that $r = s \pmod n$. □

We will now prove our main claim that the family $\mathcal{H} = \{h_{a,b} : a, b \in \mathbb{Z}_p,\ a \neq 0\}$ is 2-universal.

<u>Proof of 2-universality of $\mathcal{H}$.</u>

For each $z \in \{0, 1, \ldots, n-1\}$, let $A_z = \{x \in \mathbb{Z}_p : x = z \pmod n\}$. It is easy to see that $|A_z| \leq \lceil p/n \rceil$.

In other words, for every $r \in \mathbb{Z}_p$, there are at most $\lceil p/n \rceil$ different choices of $s \in \mathbb{Z}_p$ such that $r = s \pmod n$. Since there are p different choices of $r \in \mathbb{Z}_p$ to begin with and because $r \neq s$, we can conclude that the number of hash functions in $\mathcal{H}$ that cause x and y to collide is $\leq p\left(\lceil p/n \rceil - 1\right) \leq p\left(\frac{p-1}{n} + 1 - 1\right)$

$$= \frac{p \cdot (p-1)}{n}$$

Since $|\mathcal{H}| = p(p-1)$, we have the desired result. □

<u>Claim 2.</u> If $\mathcal{H}$ is 2-universal, then for any $x \in \mathcal{U}$ that we may want to add/delete/look-up and for a random h taken from $\mathcal{H}$, the expected number of collisions between x and other elements in $S \leq |S|/n$

<u>Proof</u> Each $y \in S$ $(y \neq x)$ has a $1/n$ chance of collision with x by definition of "2-universal".

Let $C_{xy} = \begin{cases} 1 & \text{if } x \text{ and } y \text{ collide} \\ 0 & \text{otherwise} \end{cases}$

Let $C_x = \sum_{\substack{y \in S \\ y \neq x}} C_{xy}$ be the total number of collisions of x.

We know that $E[C_{xy}] \leq 1/n$

Thus $E[C_x] = \sum_{\substack{y \in S \\ y \neq x}} E[C_{xy}] \leq |S|/n$

As long as $|S| = O(n)$, the expected search time is $O(1)$. $\square$

Note that this does not mean that the expected worst-case time to search for an element is $O(1)$. The expected worst case search time is $\underbrace{\Theta(\log n / \log\log n)}$ when $|S| = n$.

$\hookrightarrow$ this is equivalent to the following balls-and bins problem: toss n balls independently and uniformly at random into one of n bins. Then whp the fullest bin contains $\Theta(\log n/\log\log n)$ balls.

Perfect Hashing

Suppose there are n bins. Let C_{ij} be the indicator variable that equals 1 if and only if $i \neq j$ and ball i and ball j land in the same bin.

Let $C = \sum_{i<j} C_{ij}$ be the total number of pairwise collisions.

Since the balls are thrown uniformly at random, the prob. of collision is exactly $1/n$. So

$E[C] = \frac{\binom{|S|}{2}}{n}$. If $n = |S|^2$, then the expected number of collisions is $< 1/2$

Thus we have 2 alternatives: use a small hash table, keep the space usage down and the resulting expected worst-case time is $\Theta(\log n / \log\log n)$ whp which is not much better than a binary search tree. On the other hand, we can get constant worst case search time, at least in expectation, by using a table of quadratic size, but that seems wasteful.

– There is a simple way to combine these two ideas to get a data structure of linear expected size, whose expected worst-case search time is $O(1)$ for the static dictionary problem.

At the top level, use a hash table of size n, but instead of linked lists, use secondary hash tables to resolve collisions.

Specifically, the j-th hash table has size n_j^2 where n_j = # of items whose primary hash value is j.

The expected worst case search time in any secondary hash table is $O(1)$, by the earlier analysis.

Claim. If the initial h is picked at random from a 2-universal family H, then $Pr[\sum_i n_i^2 > 4|S|] < \frac{1}{2}$

Proof: We will show that $E[\sum_i n_i^2] < 2|S|$.

Then the above probability follows from Markov's Inequality that shows that for a non-negative random variable x, $Pr[x \geq t \cdot E[x]] \leq 1/t$.

$$\sum n_i^2 = \sum_x \sum_y C_{xy} = |S| + \sum_x \sum_{y \neq x} C_{xy}$$

$$E[\sum n_i^2] = |S| + \sum_x \left(\frac{|S|-1}{n}\right) = |S| + \frac{|S|.(|S|-1)}{n} \leq 2|S|$$

since $|S| = n$.