Notes by Johan Wikström.

Hashing

Why sort?

Searching: for a set $S, |S| = n$. Ask $x \in S$? Binary search answers this in $O(\log(n))$ time but we want to do better.

Alternative: Hashing

Get nice function h, store info on x in $h(x)$.

$$h : U \to [m] = \{0, 1, ..., m-1\}$$

- $w \approx n$ usually.
- h should spread S nicely.
- Best of all worlds $h(x) \neq h(y), x \neq y, x, y \in S$

If h is a fixed function there are always bad sets S. Take $S = \{x | h(x) = i\}$ for some i. No nice theory for fixed functions!

Instead, use set of hash functions $h_\alpha, \alpha \in T$. Pick a random α and use h_α. For each S, a random α is good.

Carter-Wegman pair-wise independent hashing

Pair-wise independence was introduced by Carter and Wegman in the 1970'ies and is defined by the following property. $a, b \in [m], \forall x \neq y$

$$P[h_\alpha(x) = a \wedge h_\alpha(y) = b] = \frac{1}{m^2}$$

Canonical example
$U = \{0, 1\}^\ell, m = 2^t$ Input ℓ **bits**, output t **bits**

$$h_{M,r}(x) = Mx + r$$

- M is $t \times \ell$ matrix
- r is a t bit vector
- $+$ is mod 2, i.e. $+$ is XOR
- $*$ is mod 2, i.e. $*$ is AND

Theorem 1 *$h_{M,r}$ is a family of pairwise independent hash functions, i.e. they have the Carter-Wegman property.*

Proof: For intuition let us study the case $x = 0^\ell = 00...0 \Rightarrow h_{M,r}(x) = r$ $y = 1000... \Rightarrow h_{M,r}(y) = r + m_1$, where m_1 is the first column of M. It is not difficult to see that r and $r + m_1$ are two independent random vectors of t bits.

The strategy of the proof is as follows.

1. Do $t = 1$
2. Observe that output bits behave independently to get general t.

When $t = 1, x \neq y \in \{0,1\}^\ell$ a, b are two bits and M is simply a row vector of ℓ bits. The key equations are

$$Mx + r = \sum_{i=0}^{\ell-1} m_i x_i + r = a$$

$$My + r = \sum_{i=0}^{\ell-1} m_i y_i + r = b$$

Now there exists one i such that $x_i \neq y_i$. We can without loss of generality assume that $i = 0$ and $x_0 = 0$ and $y_0 = 1$. Fix $m_1, ..., m_{\ell-1}$ and we claim that probability over m_0 and r that we get a and b is $1/4$. In other words exactly one value of r and m_0 that gives the desired bits a and b. To see this, note that we need

$$m_0 * x_0 + r = a + \sum_{i=0}^{\ell-1} m_i x_i \qquad \text{+ and - are the same in mod 2}$$

$$m_0 * y_0 + r = b + \sum_{i=0}^{\ell-1} m_i y_i.$$

Since $x_0 = 0$ and $y_0 = 1$, this is equivalent to

$$r = a + \sum_{i=0}^{\ell-1} m_i x_i$$

$$m_0 + r = b + \sum_{i=0}^{\ell-1} m_i y_i$$

and the first equation gives a unique value for r and the second then gives a unique value of m_0. As there are four potential values of r and m_0 this gives probability $1/4$.

Prove for general t using $t = 1$

Look at fig 1. The fact that the equations are independent implies that the probability that you get the vectors a and b is $(\frac{1}{4})^t = (1/2^t)^2$ as the theorem claims. ∎

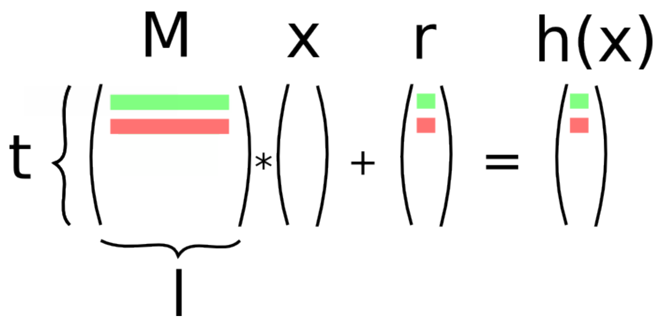

Figure 1: The red and green bits in the output h(x) are independent

More theorems on hashing

Theorem 2 *The expected number of collisions under h_α is* $\frac{n(n-1)}{2m}$

This expectation is over random α and is true for all S.

Proof: A collision is a pair (i, j) such that $i \neq j, h_\alpha(x_i) = h_\alpha(x_j)$. Let

$$E_{ij} = \begin{cases} 1 & x_i \text{ and } x_j \text{ collide} \\ 0 & \text{otherwise} \end{cases}$$

then # collisions is $\sum_{i<j} E_{ij}$. We need to calculate the expectation of this.

$$\begin{aligned} E[\sum_{i<j} E_{ij}] &= \sum_{i<j} E[E_{ij}] \\ &= \sum_{i<j} P[h_\alpha(x_i) = h_\alpha(x_j)] \\ &= \frac{n(n-1)}{2} \frac{1}{m} \end{aligned}$$

since $P[h_\alpha(x_i) = h_\alpha(x_j)] = \frac{1}{m}$. The theorem follows. ∎

We have the following immediate corollary.

Theorem 1 *If* $m > \frac{n(n-1)}{2}$ *there exists a* h_α *with no collisions.*

Proof: We have

$$E[\# \text{ collisions}] = \frac{n(n-1)}{2 * m} < 1$$

and thus there must be some h without collisions as otherwise this expected value would be at least 1. ∎

Theorem 2 *If* $m \geq n(n-1)$ *at least half of all* h_α *has no collisions.*

Proof: Indeed
$$E[\#\text{ collisions}] \leq \frac{1}{2}$$
and if more than half of the h_α would have one collision this expected value would be greater than $\frac{1}{2}$. ∎

Two level hashing(called double hashing in old lecture notes)

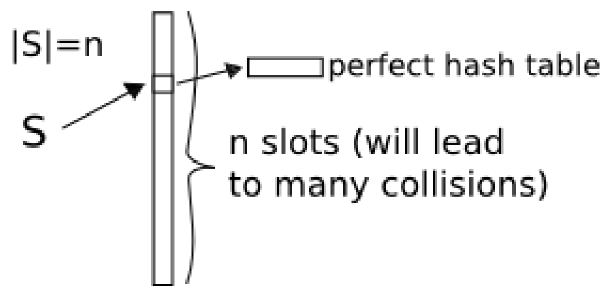

Figure 2: When we get more that two elements that hash to the same value, hash one more time into a smaller hash table.

To be more precise if more than two elements map to i, pick h_{α_i} with minimal range that hashes these elements *perfectly.* In other words look at the set

$$S_i = \{x \in S, h_\alpha(x) = i\}$$

and this set is split perfectly under h_{α_i}. Two level hashing answers "$x \in S$?" in $O(1)$ time by the following high level algorithm.

Compute $h(x)$, if marked "collision" compute $h_{\alpha_{h(x)}}(x)$ to check if it's there.

Theorem 3 *The two-level hashing can be implemented with $O(n)$ space.*

Proof: We need to check how much space is needed for all small perfect hash tables given by h_{α_i}. Let s_i be the number of elements hashing to i. We know that h_{α_i} can be implemented with space $s_i(s_i - 1)$. Total extra space needed is $\sum_{i=0}^{n-1} s_i(s_i - 1)$. Note that his is twice the number of collisions of the outer hash function h. and the expected of such collisions is $= \frac{n(n-1)}{2m}$ when hashing to $[m]$. As in our case we have $m = n$ the expected number of collisions is

$\Rightarrow \frac{n(n-1)}{2n} = \frac{n-1}{2} \approx \frac{n}{2}$
$\Rightarrow E[\text{Extra space needed}] = \frac{2*n}{2} = n$ ∎

Preparing for next lecture

Finding median of $2m+1$ numbers, "middle element"

Attempt 1, sort output - middle element in takes $n \log n$ times.

Faster?

In a modification of Quicksort we can ignore all recursive calls where you know the median can't be. Gives $O(n)$ and we can reduce constant before n by selecting pivot cleverly.