Universal Hashing

Prepared by: Sarah Adel Bargal

Based on: References*

1- Introduction

One of the methods of collision resolution is by chaining. As demonstrated in figure 1 below, h(k_5)=h(k_2)=h(k_7) and therefore they are placed in a chain at the collision slot.

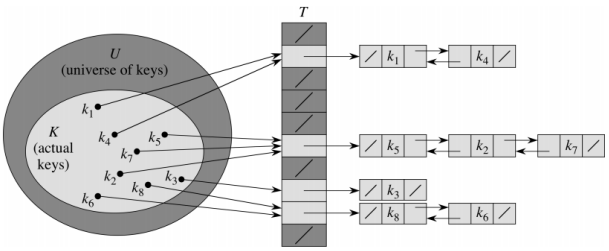

Figure 1: Collision Resolution by chaining [1]

For any hash function h, there exists a 'bad set of keys' that all hash to the same slot, making us end up with one long linked list that we must search through. In fact, it would cost more than searching through the linked list due to hash function computations!

2- Universal Hashing

This section demonstrates a solution to the weakness of hashing presented in section 1; the solution is through randomness. Instead of using a defined hash function, for which an adversary can always find a 'bad set of keys!', the idea is to select a hash function randomly from a family of hash functions! Since this is a real-time decision, an adversary cannot find the 'bad set of keys' since the hash function is not known apriori, and one is guaranteed to do well on average whatever the data is.

Definition 1

Let $\mathcal{U}$ be a universe of keys, and let $\mathcal{H}$ be a finite collection of hash functions mapping $\mathcal{U}$ to {0, 1, … , m-1}.

Definition 2

$\mathcal{H}$ is universal if $\forall x, y \in \mathcal{U}$ where $x \neq y$ $|\{h \in \mathcal{H} : h(x)=h(y)\}| = \frac{|\mathcal{H}|}{m}$

Following from definition 2, if h is chosen uniformly at random from $\mathcal{H}$, the probability of a collision between x and y is:

$$\frac{\#\ functions\ that\ map\ x\ and\ y\ to\ the\ same\ location}{Total\ \#\ of\ functions} = \frac{\frac{|\mathcal{H}|}{m}}{|\mathcal{H}|} = \frac{1}{m}$$

Theorem

Choose h randomly from $\mathcal{H}$. Suppose we are hashing n keys into m slots in table T. Then for a given key x,

$$\text{E[\# collisions with x]} < \frac{n}{m}$$

[Note: $\frac{n}{m}$ = α = load factor of hash table]

Importance of the above theorem

By proving the above theorem, we are saying that if the universal set of hash function exists, then the elements get evenly distributed in the hash table (on average); desired behavior.

Proof

Let C_x be the random variable denoting total collisions of keys in T with x,

and let $C_{xy} = \begin{cases} 1 & if\ h(x) = h(y) \\ 0 & o.w. \end{cases}$

E $[C_x]$ = E $[\sum_{y \in T-x} C_{xy}]$

= $\sum_{y \in T-x} E\,[C_{xy}]$ due to the linearity of expectation

= $\sum_{y \in T-x} \frac{1}{m}$ since the expected value of an indicator random variable is the probability it equals 1

= n-1 $(\frac{1}{m})$ since |T-x| = n-1

< $\frac{n}{m}$

3- Constructing a Universal Set of Hash Functions

This section demonstrates one construction [2, 3, 4] of a universal family of hash functions.

Step1 - *condition*

Select m to be prime.

Step 2 - pre-processing

Decompose key k into r+1 digits: k = $<k_0, k_1, \dots, k_r>$ where $k_i \in \{0, 1, \dots, m-1\}$

(equivalent to writing key k in base m)

Step 3 - *the randomness*

Pick a = $<a_0, a_1, \dots, a_r>$ at random. Each $a_i \in \{0, 1, \dots, m-1\}$

Step 4 - *the hash function*

h_a (k) = $(\sum_{i=0}^{r} a_i\, k_i)$ mod m

Now we need to prove that the construction above indeed creates a universal family of hash functions. In order to do that, we need to know the total number of hash functions in this family, and the fraction of these hash functions that maps and 2 distinct x and y to the same slot.

Theorem

$\mathcal{H}$ is universal ($\mathcal{H}$ being constructed using the 4 steps explained above)

Proof

Part a: Total number of hash functions in the family

$|\mathcal{H}| = m^{r+1}$ since each of the r+1 a_i's has m possible values.

Part b: Number of hash functions that cause distinct x and y to collide

Let $x=<x_0, x_1, \dots, x_r>$

$y=<y_0, y_1, \dots, y_r>$ be different keys (differ in at least one digit).

Without loss of generality, assume x and y differ on the first digit; at position 0.

If x and y collide: $h_a(x) = h_a(y)$

$\Rightarrow (\sum_{i=0}^{r} a_i\, x_i) \bmod m = (\sum_{i=0}^{r} a_i\, y_i) \bmod m$

$\Rightarrow \sum_{i=0}^{r} a_i\, x_i \equiv \sum_{i=0}^{r} a_i\, y_i \pmod m$

$\Rightarrow \sum_{i=0}^{r} a_i\, x_i - \sum_{i=0}^{r} a_i\, y_i \equiv 0 \pmod m$

$\Rightarrow \sum_{i=0}^{r} a_i\, (x_i - y_i) \equiv 0 \pmod m$

$\Rightarrow a_0\, (x_0 - y_0) + \sum_{i=1}^{r} a_i\, (x_i - y_i) \equiv 0 \pmod m$

$\Rightarrow a_0\, (x_0 - y_0) \equiv -\sum_{i=1}^{r} a_i\, (x_i - y_i) \pmod m$

$\Rightarrow a_0\, (x_0 - y_0)\, (x_0 - y_0)^{-1} \equiv (-\sum_{i=1}^{r} a_i\, (x_i - y_i))\, (x_0 - y_0)^{-1} \pmod m$

[Note: the above step applies because m is prime and $x_0 \neq y_0$, therefore $(x_0 - y_0)^{-1}$ exists]

$\Rightarrow a_0 \equiv (-\sum_{i=1}^{r} a_i\, (x_i - y_i))\, (x_0 - y_0)^{-1} \pmod m$

The above congruency implies that once the values of $a_1, \dots\dots, a_r$ are selected, exactly one value of a_0 causes x and y to collide. Counting the possible values of a_0's that cause x and y to collide gives us the number of h_a's we have in $\mathcal{H}$ that cause x and y to collide. This equals the number possible combinations of $a_1, \dots\dots, a_r$ which is m^r since each of the r digits has m possible values.

Part a gives the total number of hash functions to be: m^{r+1}

Part b gives the total number of hash functions that cause collisions between x and y to be: m^r

Therefore, the fraction of the hash functions in the family that cause collisions is: $\frac{m^r}{m^{r+1}} = \frac{1}{m}$,

and therefore $\mathcal{H}$ is universal.

4- Example: Hashing IP Addresses

Universe $\mathcal{U}$ is IP addresses. Each IP address is a 32-bit 4-tuple $<x_1, x_2, x_3, x_4>$ where $x_i \in \{0, \ldots, 255\}$.

Let m be a prime number (m=997 if we need to store 500 IPs for example).

Define h_a for the 4-tuple a = $<a_1, a_2, a_3, a_4>$ where $a_i \in \{0, \ldots, m-1\}$.

h_a : IP address → Slot

Q.) How do I evaluate which slot $<x_1, x_2, x_3, x_4>$ IP address hashes to using the above formulation of $\mathcal{H}$?
A.) Using the following equation which requires constant space and time:
$h_a(x_1, x_2, x_3, x_4) = a_1 x_1 + a_2 x_2 + a_3 x_3 + a_4 x_4 \pmod m$

5- Introduction to Perfect Hashing

So far, we have been dealing with expected value, but what if we need guaranteed worst case performance? This is where Perfect Hashing comes in, but there is a price to pay; the table must be static. Static tables suit certain applications that are not dynamic like: symbol tables, files on a CD.

Idea
A two level scheme with universal hashing at each level such that there are no collisions at level 2. If n_i items hash to slot i, then a table at slot i is used that has size n_i^2, giving sparsity which allows us to easily find hash functions that would not cause collisions at level 2 (collisions may occur at level 1).

A search here takes O(1) time in the worst case for any key.

References*

[1] Thomas Cormen, Charles Leiseson, Ronald Rivest, Clifford Stein, "Introduction to Algorithms", *Chapter 11*.

[2] Prof. Charles E. Leiserson Lecture and Slides.
http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-046j-introduction-to-algorithms-sma-5503-fall-2005/video-lectures/lecture-8-universal-hashing-perfect-hashing/lec8.pdf
http://www.youtube.com/watch?v=bVoTalWk6fo

[3] -Video from Stanford University by Tim Roughgarden.
https://class.coursera.org/algo-004/lecture/68

[4] Jeff Erickson Lecture Notes from University of Illinois at Urbana Champaign.
http://www.cs.uiuc.edu/~jeffe/teaching/algorithms/notes/12-hashing.pdf