Design and Analysis
of Algorithms I

Data Structures

Universal Hash Functions: Performance Guarantees (Chaining)

Overview of Universal Hashing

Next : details on randomized solution (in 3 parts).

Part 1 : proposed definition of a “good random hash function”.
(“universal family of hash functions”)

Part 3 : concrete example of simple + practical such functions

Part 4 : justifications of definition : “good functions” lead to “good performance”

Universal Hash Functions

Definition : Let H be a set of hash function from U to {0,1,2,..,n-1}

H is universal if and only if :
For all $x, y \in U$ (with $x \neq y$)

$$Pr_{h \in H}[x, y\ collide;\ h(x) = h(y)] \leq 1/n$$

(n = # of buckets)

When h is chosen uniformly at random at random from H.
(i.e., collision probability as small as with "gold standard" of perfectly random hashing)

Chaining: Constant-Time Guarantee

Scenario : hash table implemented with chaining. Hash function h chosen uniformly at random from universal family H.

Theorem : [Carter-Wegman 1979]
All operations run in O(1) time. (for every data set S)

Caveats : 1.) in expectation over the random choice of the hash function h. (h = # of buckets) independent of dataset

2.) assumes |S| = O(n) [i.e., load $\alpha = \frac{|S|}{n} = O(1)$]

3.) assumes takes O(1) time to evaluate hash function

Proof (Part I)

Will analyze an unsuccessful Lookup (other operations only faster).

that is- lookup into the hash table given x is not part of dataset. That's the worst case lookup time

So : Let S = data set with |S| = O(n)

# of buckets

Consider Lookup for $x \notin S$

(arbitrary data set S)

Running Time : O(1) + O($\boxed{\text{list length in A[h(x)]}}$)

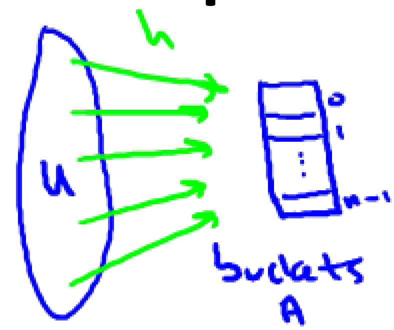
Compute h(x)

Traverse list

L

A random variable, depends on hash function h compute expectation via indicator random variables

A General Decomposition Principle



Collision : distinct x,y in U such that h(x) = h(y).

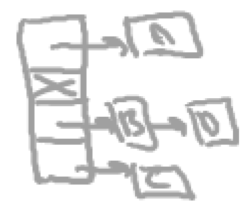
Solution#1: (separate) chaining.

-- keep linked list in each bucket

-- given a key/object x, perform Insert/Delete/Lookup in the list in A[h(x)]

bucket for x

linked list for x



Solution#2 : open addressing. (only one object per bucket)

-- hash function now specifies probe sequence h1(x), h2(x), …

(keep trying till find open slot)

use 2 hash functions

-- examples : linear probing (look consecutively), double hashing

Proof (Part II)

Let L = list length in A[h(x)].

For $y \in S$ (so, $y \neq x$) define $z_y = \begin{cases} 1 \text{ if } h(y) = h(x) \\ 0 \text{ otherwise} \end{cases}$

Note : $$L = \sum_{y \in S} A_y$$

So : $$E[L] = \sum_{y \in S} E[Z_y]$$

Recall

$$z_y = \begin{cases} 1 \text{ if } h(y) = h(x) \\ 0 \text{ otherwise} \end{cases}$$

What does $\mathrm{E}[Z_y]$ evaluate to?

$$E[z_y] = 0 \cdot Pr[z_y = 0] + 1 \cdot Pr[z_y = 1]$$

$$= Pr[h(y) = h(x)]$$

- ○ $\Pr[h(y) = 0]$
- ○ $\Pr[h(y) \neq x]$
- ○ $\Pr[h(y) = h(x)]$
- ○ $\Pr[h(y) \neq h(x)]$

Proof (Part II)

Let L = list length in A[h(x)].

For $y \in S$ (so, $y \neq x$) define $z_y = \begin{cases} 1 \text{ if } h(y) = h(x) \\ 0 \text{ otherwise} \end{cases}$

Note : $$L = \sum_{y \in S} A_y$$

So : $$E[L] = \sum_{y \in S} E[Z_y] = \sum_{y \in S} Pr[h(y) = h(x)]$$

Which of the following is the smallest valid upper bound on $\Pr[h(y) = h(x)]$?

- $1/n^2$
- $1/n$
- $1/2$
- $1 - 1/n$

By definition of a universal family of hash functions

Proof (Part II)

Let L = list length in A[h(x)].

For $y \in S$ (so, $y \neq x$) define $z_y = \begin{cases} 1 \text{ if } h(y) = h(x) \\ 0 \text{ otherwise} \end{cases}$

Note : $$L = \sum_{y \in S} A_y$$

So : $$E[L] = \sum_{y \in S} E[Z_y] = \sum_{y \in S} \underbrace{Pr[h(y) = h(x)]}_{\leq \frac{1}{n}}$$

Since H is universal $\longrightarrow$ $$\leq \sum_{y \in S} \frac{1}{n}$$

$$= \frac{|S|}{n} = load\ \alpha = O(1)$$ Provided |S| = O(n)

Q.E.D.

Tim Roughgarden