Design and Analysis of Algorithms I

Data Structures

Universal Hash Functions: Motivation

Hash Table: Supported Operations

Purpose : maintain a (possibly evolving) set of stuff.
(transactions, people + associated data, IP addresses, etc.)

Insert : add new record

Delete : delete existing record
easier/more common with chaining than open addressing

Lookup : check for a particular record
(a "dictionary")

Using a "key"

AMAZING GUARANTEE
All operations in O(1) time ! *

* 1. properly implemented 2. non-pathological data

Resolving Collisions

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

The Load of a Hash Table

Definition : the load factor of a hash table is

$$\alpha := \frac{\text{\# of objects in hash table}}{\text{\# of buckets of hash table}}$$

Which hash table implementation strategy is feasible for load factors larger than 1?

- Both chaining and open addressing
- Neither chaining nor open addressing
- Only chaining
- Only open addressing

The Load of a Hash Table

Definition : the load factor of a hash table is

$$\alpha := \frac{\text{\# of objects in hash table}}{\text{\# of buckets of hash table}}$$

Note : 1.) α = O(1) is necessary condition for operations to run in constant time.
2.) with open addressing, need $\alpha \ll 1$.

Upshot#1 : good HT performance, need to control load.

Pathological Data Sets

Upshot#2 : for good HT performance, need a good hash function.

Ideal : user super-clever hash function guaranteed
to spread every data set out evenly.

Problem : DOES NOT EXIST! (for every hash function, there is a pathological data set)

Reason : fix a hash function h : U -> {0,1,2,…,n-1}

⇒a la Pigeonhole Principle, there exist bucket i such that at least |u|/n elements of U hash to I under h.

⇒ if data set drawn only from these, everything collides !

Pathological Data in the Real World

Preference : Crosby and Wallach, USENIX 2003.

Main Point : can paralyze several real-world systems (e.g., network intrusion detection) by exploiting badly designed hash functions.

-- open source

-- overly simplistic hash function

(easy to reverse engineer a pathological data set)

Solutions

1. Use a cryptographic hash function (e.g., SHA-2)
   -- infeasible to reverse engineer a pathological data set

2. Use randomization. ⟵In next 2 videos
   -- design a family H of hash functions such that for all data sets S, "almost all" functions $h \in H$ spread S out "pretty evenly".
   (compare to QuickSort guarantee)

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"

Design and Analysis of Algorithms I

Data Structures

Universal Hash Functions: Definition and Example

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 functions from U to {0,1,2,…,n-1}

H is universal if and only if :

for all x,y in U (with $x \neq y$)

$$Pr_{h \in H}[x, y \ collide] \leq \frac{1}{n}$$

ie., h(x) = h(y)

(n = # of buckets)

When h is chosen uniformly at random from H.

(i.e., collision probability as small as with "gold standard" of perfectly random hashing

Consider a hash function family H, where each hash function of H maps elements from a universe U to one of n buckets. Suppose H has the following property: for every bucket I and key k, a 1/n fraction of the hash functions in H map k to i. Is H universal ?

- Yes, always.
- No, never.
- Maybe yes, maybe no (depends on the H).
- Only if the hash table is implemented using chaining.

Yes : Take H = all functions from U to {0,1,2,..,n-1}

No : Take H = the set of n different constant functions

Example: Hashing IP Addresses

Let U = IP addresses (of the form (x_1,x_2,x_3,x_4), with each $x_i \in \{0, 1, 2, ..., 255\}$

Let n = a prime (e.g., small multiple of # of objects in HT)

Construction : Define one hash function ha per 4-tuple a = (a_1,a_2,a_3,a_4) with each $a_i \in \{0, 1, 2, 3, ..., n-1\}$

Define : h_a : IP addrs -> buckets by

n^4 such functions

$$h_a(x_1, x_2, x_3, x_4) = \begin{pmatrix} a_1x_1 + a_2x_2 + \\ a_3x_3 + a_4x_4 \end{pmatrix} mod\ n$$

A Universal Hash Function

Define : $H = \{h_a | a_1, a_2, a_3, a_4 \in \{0, 1, 2, ..., n-1\}\}$

$$h_a(x_1, x_2, x_3, x_4) = \begin{pmatrix} a_1x_1 + a_2x_2 + \\ a_3x_3 + a_4x_4 \end{pmatrix} mod\ n$$

Theorem: This family is universal

Proof (Part I)

Consider distinct IP addresses (x_1,x_2,x_3,x_4), (y_1,y_2,y_3,y_4).

Assume : $x_4 \neq y_4$

Question : collision probability ?

$$(i.e., \underbrace{Prob_{h_a \in H}[h_a(x_1,..,x_4) = h_a(y_1,..,y_4)]})$$

Note : collision <==>

$$a_1x_1 + a_2 + x_2 + a_3 + x_3 + a_4x_4 = a_1y_1 + a_2 + y_2 + a_3 + y_3 + a_4 + y_4 \ (mod\ n)$$

$$<=> a_4(x_4 - y_4) = \sum_{i=1}^{3} a_i(y_i - x_i) \ (mod\ n)$$

Next : condition on random choice of a_1,a_2,a_3. (a_4 still random)

Proof (Part II)

The Story So Far : with a_1, a_2, a_3 fixed arbitrarily, how many choices of a_4 satisfy

$$\underbrace{\boxed{\boxed{a_4}(x_4 - y_4)} = \boxed{\sum_{i=1}^{3} a_i(y_i - x_i) \ (mod\ n)}}_{\text{<==> x,y collide under } h_a}$$

Still random

Some fixed number in {0,1,2,..,n-1}

Key Claim : left-hand side equally likely to be any of {0,1,2,…,n-1}

Reason : $x_4 \neq y_4$ $(x_4 - y_4 \neq 0 \mod n)$

n is prime, a_4 uniform at random

[addendum : make sure n bigger than the maximum value of an ai]

Implies $\text{Prob}[h_a(x) = h_a(y)] = 1/n$

"Proof" by example : n = 7, $x_4 - y_4 = 2$ or 3 mod n

Q.E.D.

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)

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).

# of buckets

So : Let S = data set with |S| = O(n)
Consider Lookup for $x \notin S$ (arbitrary data set S)

Running Time : O(1) + O(list length in A[h(x)])

Compute h(x)

Traverse list

L

A random variable, depends on hash function h

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

<u>Note</u> : $$L = \sum_{y \in S} A_y$$

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

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?

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

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

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

Proof (Part II)

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

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

<u>Note</u> : $$L = \sum_{y \in S} A_y$$

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

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

<u>Note</u> : $L = \sum_{y \in S} A_y$

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

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