CS180 - Hashing (part 2)

Note Title 4/29/2011

Announcements

- Checkpoint Monday
- Program due Friday
- Review in class on last day

Data Storage - Dictionary

insert
find
remove

Ex.

Locker # (key)	Name (data)
26	Dan
355	Kevin
101	Tracy
53	Nitish
201	David
⋮	⋮

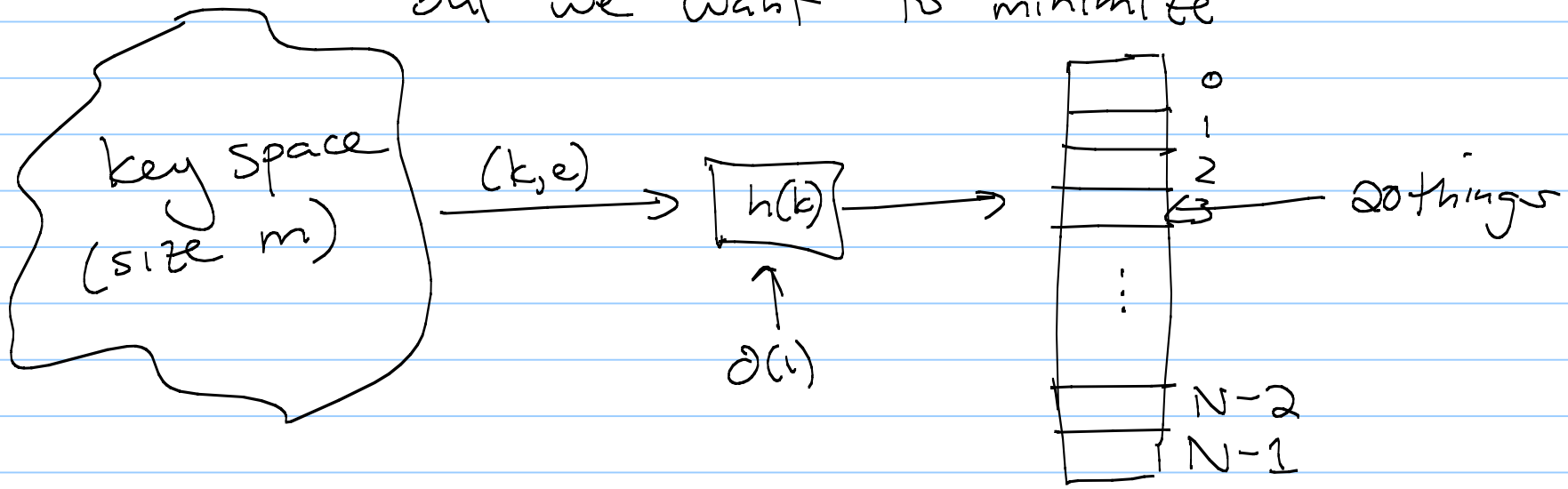
We want to be able to retrieve a name quickly when given a locker number.

(Let n = # of people, & m = # of lockers)

$n \geq m$

Good hash functions:

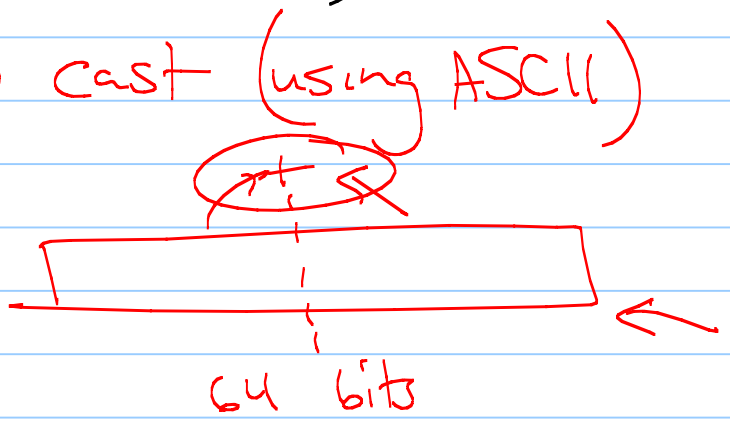
- Are fast — goal: $O(1)$
- Don't have collisions — when $k_1 \neq k_2$ but $h(k_1) = h(k_2)$
  these are unavoidable, but we want to minimize



Step 1: Get a number (& avoid collisions)

char (32-bits) → cast (using ASCII)

float (64-bits)



string: Erin

ASCII: 69 + 114 + 105 + 110 = int

But, in some cases, a strategy like this can backfire!

temp01 and temp10 and pm0te1

all hash to same int

We want to avoid collisions between "similar" strings (or other types).

A Better Idea: Polynomial Hash Codes

Pick $a \neq 1$ and split data into k 32-bit parts: $x = (x_0, x_1, x_2, x_3, \ldots, x_{k-1})$

Let $h(x) = x_0 a^{k-1} + x_1 a^{k-2} + \cdots + x_{k-2} a + x_{k-1}$

Ex: Erin, with $a = 37$

$69 \cdot 37^3 + 114 \cdot 37^2 + 105 \cdot 37 + 110 \cdot 37^0$

riEn: ?

$114 \cdot 37^3 + 105 \cdot 37^2 + 69 \cdot 37 + 110$ #

Side Note: How long does this take?

(In terms of k = # of parts)

$$h(x) = x_0 a^{k-1} + x_1 a^{k-2} + \cdots + x_{k-2} a + x_{k-1}$$

k additions

$$k + (k-1) + (k-2) + \cdots + 1 = O(k^2)$$

$$= \sum_{i=1}^{k} i$$

1 addition + 1 multiplication per term: $O(k)$

Horner's rule: $x_{k-1} + a(x_{k-2} + a(x_{k-3} + \cdots))$

Polynomial Hashing

This strategy makes it less likely that similar keys will collide.

(Works for floats, strings, etc.)

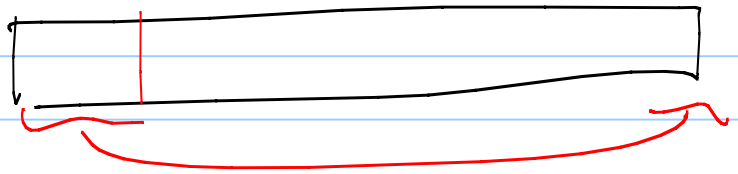
What about overflow? (32 bits is our goal)

- wrap around
- truncate

Cyclic shift hash codes

Alternative to polynomial hashing

Instead of multiplying by a, shift each 32-bit piece by some # of bits.



Also works well in practice.

Step 2: Compression maps

Now we can assume every key k is an integer.

Need to make it between 0 & N-1 (not 0 and 2^{32}).

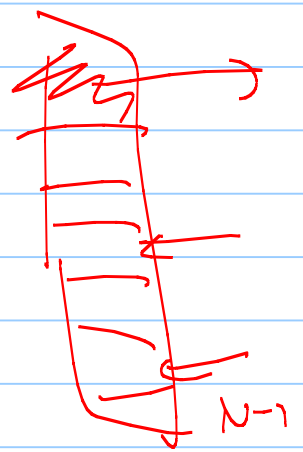
$\neq N$

Ideas:

- map everything to 0

Why is that bad?

all collisions



Modular compression maps

Take $h(k) = k \bmod N$

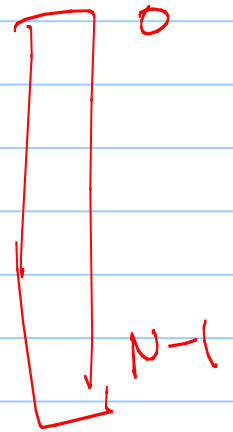
What does mod mean again?

remainder

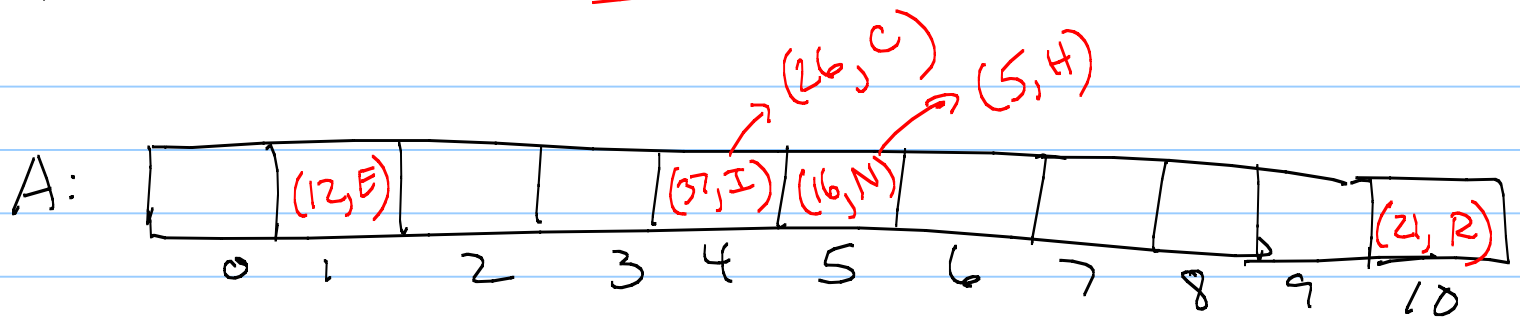
$3 \bmod 10 = 3$

$56 \bmod 10 = 6$

(in C++, %)



Example: $h(k) = k \bmod 11$



key, data

insert: (12, E) (21, R) (37, I) (16, N) (26, C) (5, H)

$h(12) = 12 \bmod 11 = 1$
$h(21) = 21 \bmod 11 = 10$
$h(37) = 37 \bmod 11 = 4$
$h(16) = 5$
$h(26) = 4$
$h(5) = 5$

Node, pair

pointer to a list: list<H>* A[11]

Some Comments:

This works best if the size of the table is a prime number.

Why?

Go take number theory & Cryptography

This minimizes collisions. E.g. even

Base on experimental studies

Strategy 2: MAD : Multiply, add, & divide

First idea: take $h(k) = k \bmod N$

Better: $h(k) = |ak+b| \bmod N$

where a & b are:

- not equal
- less than N
- relatively prime : no common divisors

$\gcd(a, b) = 1$

(Why? Go take number theory!)

End Goal: Simple Uniform Hashing Assumption

For any $k \in$ key space,

$$\Pr[h(k) = i] = \frac{1}{N}$$

(Essentially, elements are "thrown randomly" into buckets.)

Collisions

Can we ever totally avoid collisions?

No

Step 3: Handle collisions (gracefully & quickly)

So how can we handle collisions?

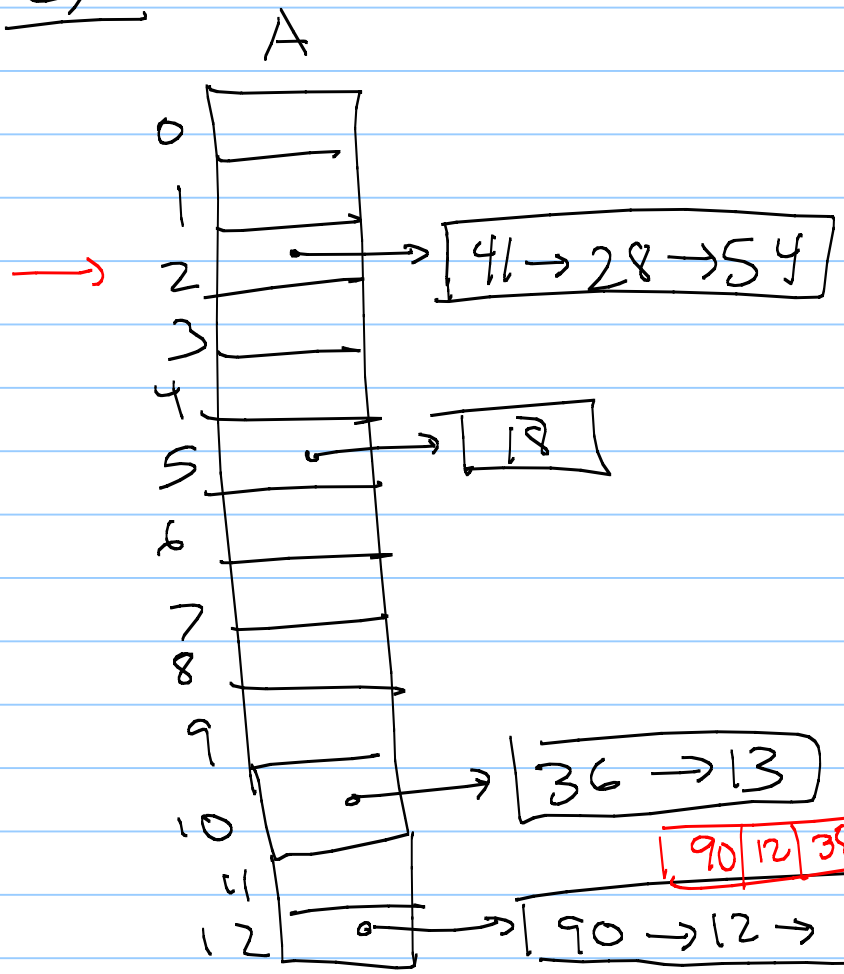
[Hint: Do we have any data structures that can store more than 1 element?]

- trees
- lists
- vectors

hashing $h(k) = O(1)$

Space: $O(n+N)$

Ex:



Running times:

List: $O(1)$ insert
$O(n)$ find
$O(n)$ remove

Vector: $O(n)$ insert
($O(1)$ amortized)
$O(n)$ remove
$O(n)$ find

In practice, much better.
(bad) $O(1)$

Next time:

Other strategies to handle collisions.

m = # keys

n = # of pieces of data

N = size of table