CS180 – Hash Tables (part 2)

Note Title 12/1/2010

Announcements

- Checkpoint is tomorrow
- Review session: Dec. 10
  NOT: 8am, 10am, 2-6

  → 10 am – noon

Hashing

An array is not very space efficient.
We would like to take the key & make is smaller.

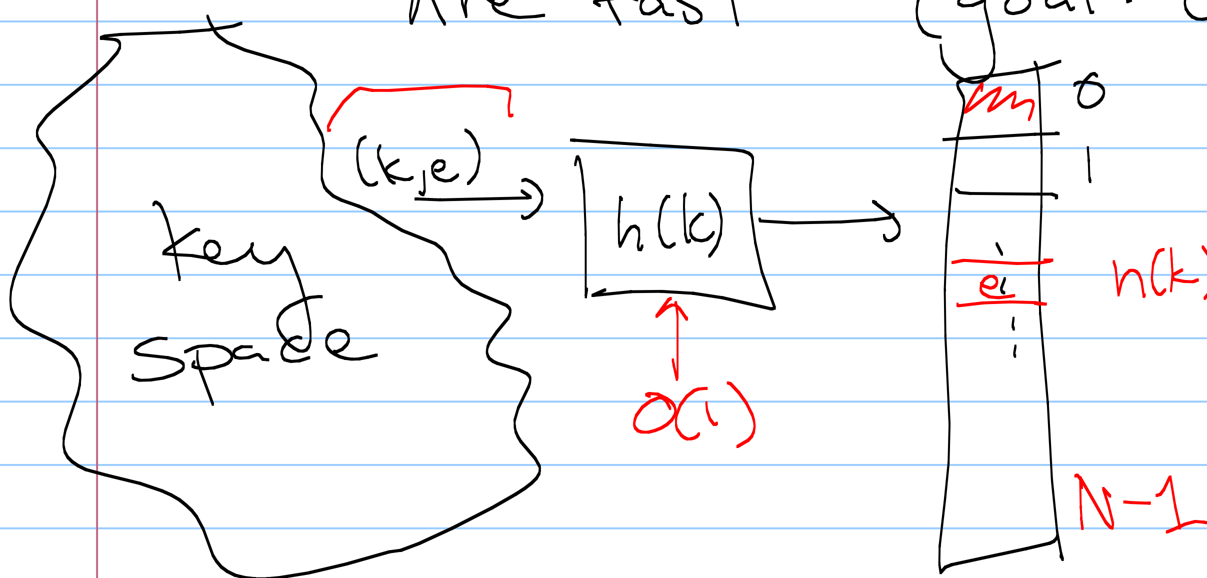
A hash function h maps each key in our dictionary to an integer in the range $[0, N-1]$.

(N should be much smaller than the # of keys.)

Then we store (k,e) in $A[h(k)]$

Good hash functions:

- Are fast (goal: $O(1)$ time)



$N < |\text{key space}|$

- Don't have collisions.

  when $k_1 \neq k_2$ but $h(k_1) = h(k_2)$

Collisions are unavoidable.

1 **First**: map key to a number

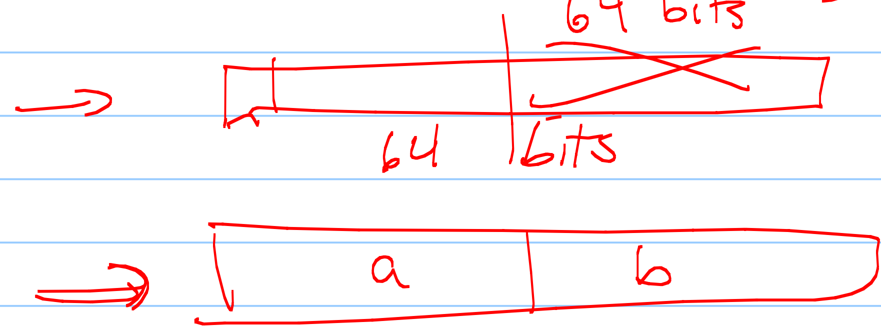
Say we want keys to fit in an int. ← 32 bits

What can we do for int, char, & short types?

int: 32 bits, char: 32 bits, short: 32 bits

Now what about long or float?

long: 64 bits



a >> 32

a+b ← simplist way to hash

This can backfire. Remember ASCII?

128-bits (full ~~newest~~ version)

$\overset{a_0\,a_1\,a_2\,a_3\,a_4\,a_5}{\underbrace{\text{temp01}}_{a_0+a_1+a_2+a_3+a_4+a_5}}$ = ◯ = ◯

temp01 and temp10 & pm0te1

all will go to same #

Goal:

Better way to avoid collisions between "similar" keys.

A better idea: Polynomial Hash codes

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

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

$x_0 \; x_1$

temp01 $\quad a = 37$

"t" $\cdot 37^5 +$ "e" $\cdot 37^4 +$ "m" $\cdot 37^3 +$ "p" $\cdot 37^2 +$

"0" $\cdot 37 +$ "1" $\cdot 37^0$

temp10

"1" $\cdot 37 +$ "0" $\cdot 37^0$

Aside: Efficiency

$$\sum_{i=1}^{n} i = O(n^2)$$

$$x_0 \cdot a^4 + x_1 \cdot a^3 + x_2 \cdot a^2 + x_3 \cdot a + x_4$$

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

$$x_4 + a(x_3 + a(x_2 + a(x_1 + a x_0)))$$

$$O(n)$$

This strategy makes it less likely that "similar" ~~words~~ words/data will collide.

What about overflow? (Remember, we want only 32-bits in key.)

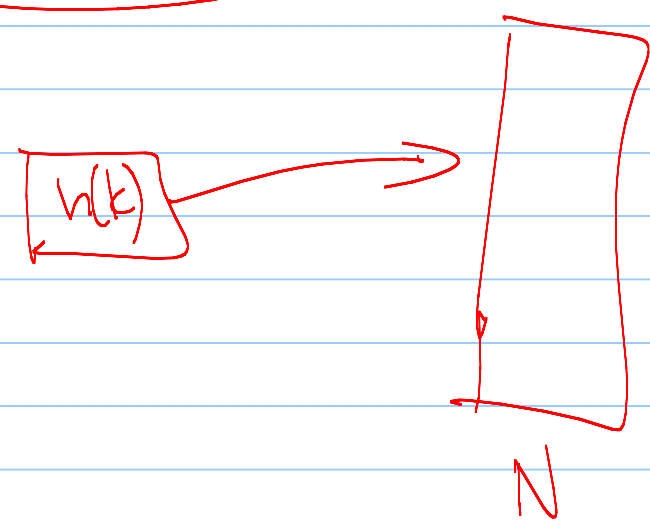
Chop it at 32-bits

(not so great)

Cyclic Shift Hash codes

shift bits in representation somehow

10100010 010001



Compression Map:

2: Once we have an integer key representation:

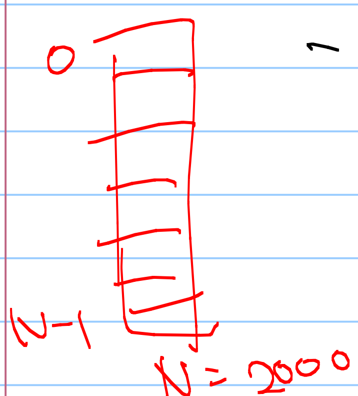
Need to make sure it is between 0 & N-1, so is in our array.

32-bit integer

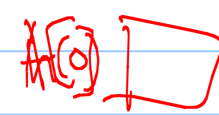
Ideas? map everything to 0 lots of collisions!

Want to spread things out evenly

- modular arithmetic



$h(k) \bmod N$



Compressing number down to something between $\underbrace{0 \text{ and } N-1}$

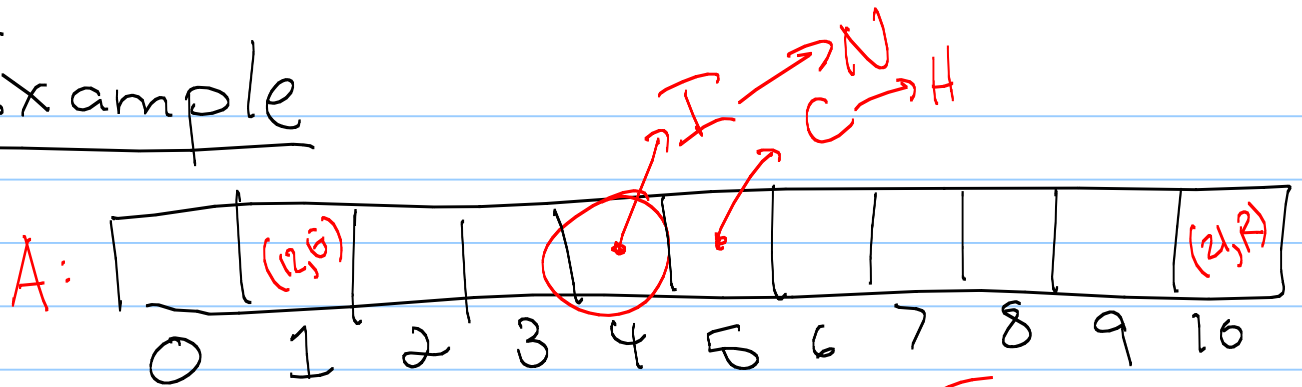
Ideas.

Modular arithmetic

$\underbrace{h(x)}_{\text{int}} \bmod N$

$x \bmod n$

Example



- Map is $h(k) = k \bmod 11$

insert(12, E)   key   data

insert(21, R)

insert(37, I)

insert(26, N)

insert(16, C)

insert(5, H)

$h(12) = 1$ E goes in A[1]

$h(21) = 10$

$h(37) = 4$

$h(26) = 4$

$h(16) = 5$

$h(5) = 5$

Strategy 1: go to next open spot

Strategy 2: Secondary hash

Strategy 3: Chaining

Some comments:

This works better if size of table is a prime number.

Why?

Go take number theory

(check book)

Before: $x \bmod N$

The MAD (multiply add & divide) method is a bit ~~b~~better:

$$h(x) = |ax + b| \bmod N$$

where a & b are

- not equal
- relatively prime ← even better, 2 prime numbers

  $\gcd(a, b) = 1$

  21 → 3, 7

  20 → 2, 4, 5, 10

- less than N

if not, collisions get more frequent

Goal: Simple Uniform Hashing Assumption:

For all $k \in$ keyspace,

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

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

Collisions

Can we ever totally avoid collisions?

No - goal was to minimize them.

How to deal with them?

(had 3 strategies)

Strategy #3:
How can we handle collisions?
(Do we have data structures to store more than one thing??)

- list ← bad search
- vectors ← insert can be bad
- trees ← $O(\log n)$ (in between lists & vectors)