CS 180 — Hash tables (pt. 1)

Note Title 11/19/2009

Announcements

- Program 5 due next Tuesday
- Next assignment will be up by Monday
- Final is Monday the 14th at noon
  (Tell me now if you have a conflict)

Data Storage

Ex:

Locker #	Name
109	Erin
65	Kerim
350	David
54	Mary
210	Austin
⋮	⋮

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

How could we store this?
(& how much space/time would it take?)

Array: locker # is index of array,
name is stored in the array

Lookup: O(1)
Space: O(N) where N is # of lockers
(not students)

List: Lookup: O(m) ← m is # of students
Space: O(m)

(balanced)
BST: Space: O(m)
Lookup: O(log m)

Other examples:

- Course # → schedule info
- Flight # → arrival info
- URL → HTML page
- Color → BMP

Not always easy to figure out how to store and lookup.

Dictionaries:

A structure which supports the following:

```
void insert(keyType &k, dataType &d)
dataType find(keyType &k)
void remove(keyType &k)
```

Notice: an array <u>is</u> a dictionary

Hashing

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

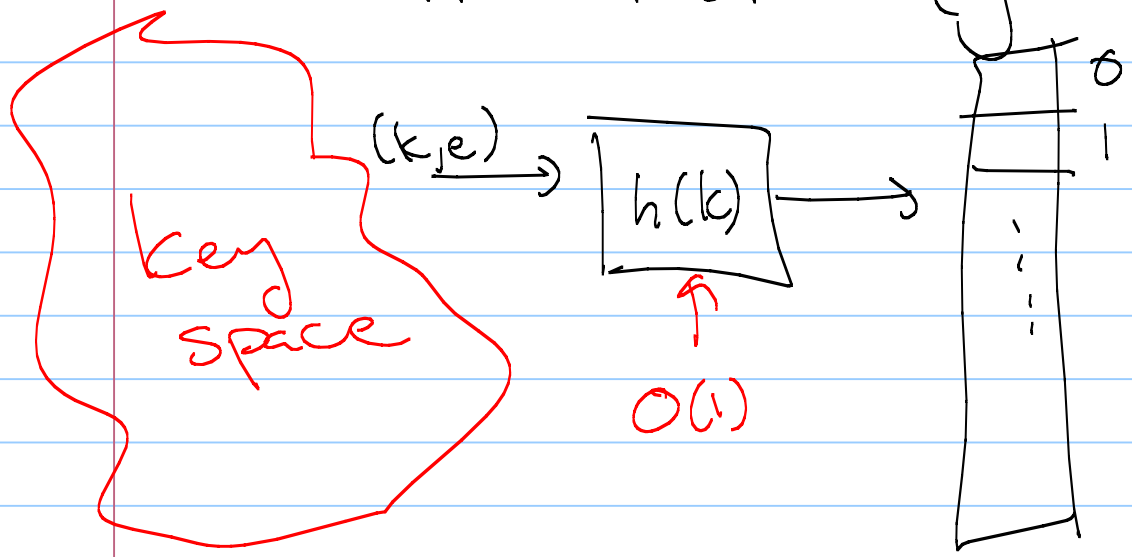
A hash function h maps each key in our dictionary to an integer in the range $[0, N-1]$.

(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)



- Don't have collisions.

First: map key to a number

Say we want keys to fit in an int. (32-bits)

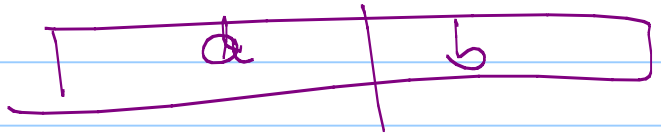
What can we do for int, char, & short types?

int(k) int(k)

Now what about long or float?
(64 bits, not 32)

cast to int - lots of things collide
int(10.3) = int(10.4)

Consider 1st 32-bits + 2nd 32-bits



$\longrightarrow a + b = k$

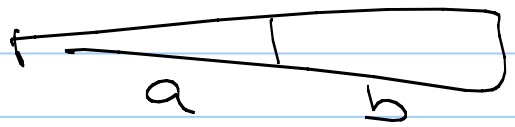
C++ code:

64 bits

32 bits

```
int hashCode(long x) {
    return int((unsigned long(x)>>32) + int(x));
}
```

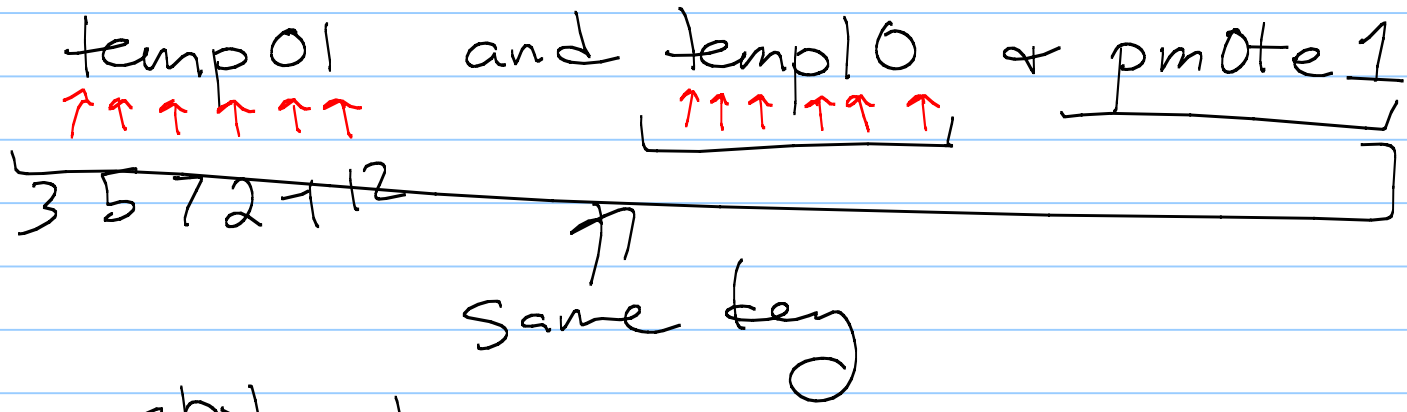
a b



Assuming hashCode is defined on ints.

This can backfire. Remember ASCII?

128-bits (full newest version)



spot tops

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

temp 01

temp 10

$a^5 \quad x \quad a^4 \quad 1 \quad a^3 \quad a^2 \quad a^1 \quad 1$

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

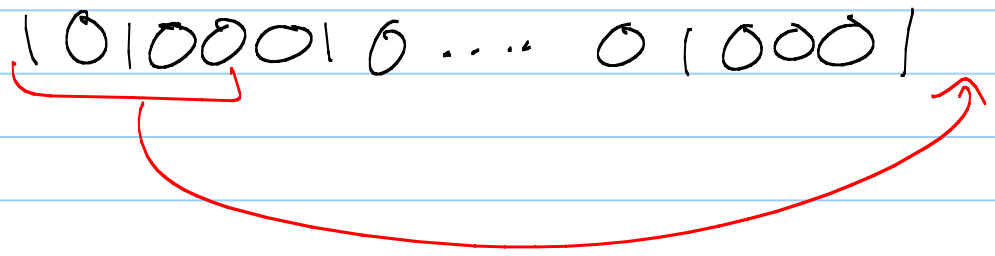
This strategy makes it less likely that "similar" words/data will collide.

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

Chop it at 32-bits

Cyclic Shift Hash codes

shift bits in representation somehow



Compression Map:

Once we have an integer key representation:

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

Ideas? map everything to 0

Want to spread things out evenly

- modular arithmetic