CS 180 — Hash tables (pt. 1)

Note Title 11/19/2009

Announcements

- Program due Saturday - checkpoint Thursday
- Give a "study sheet"
- Review session late next week
- Final is in 2 weeks

Last time : Huffman trees/codes

(Hash Tables)

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 #.

n = # of people
m = # of lockers

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

<u>List</u>: Each node gets 2 data fields
- insert: $O(1)$
- find: $O(n)$
- ☆ space: $O(n)$ ⟵

<u>Array / Vector</u>: m size, if locker is being used, store name in that array spot
- Space: $O(m)$ ⟵
- ☆ find: $O(1)$
- insert: $O(1)$

<u>AVL tree</u>:
- Size: $O(n)$
- insert: $O(\log n)$
- find: $O(\log n)$

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:

C++ (2 templates here)

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

Everything is based on keys!

(not always easy to "compare" keys)

→ (Notice: an array <u>is</u> a dictionary)

Hashing

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

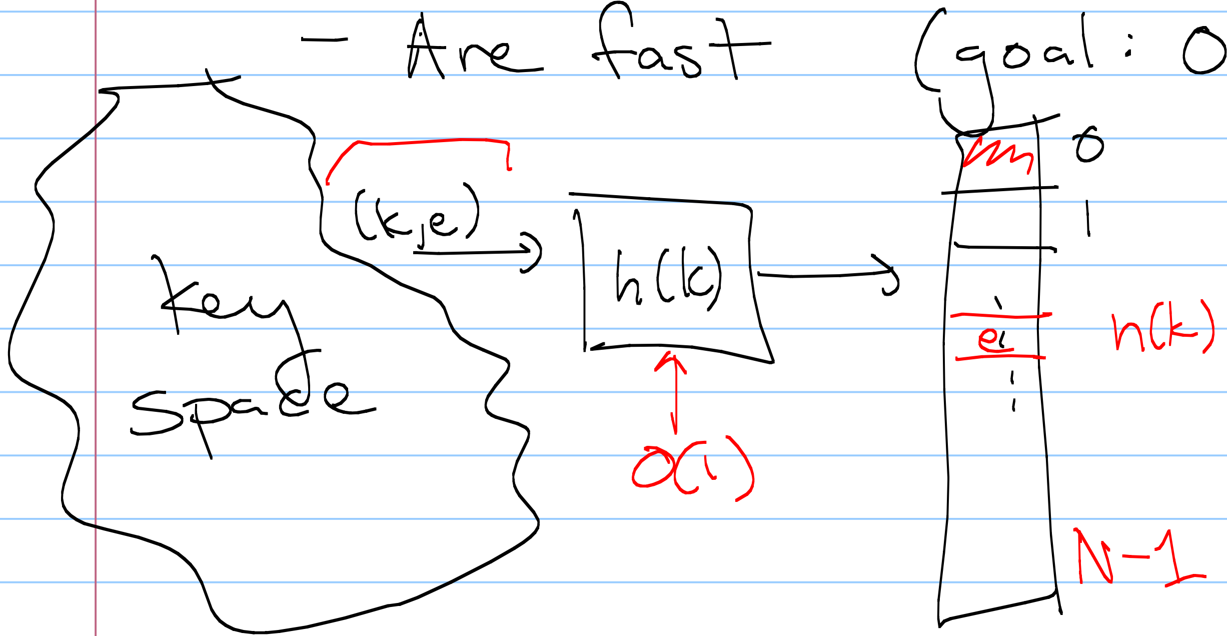
A hash function h maps each key in our dictionary to an integer in the range $[0, N-1]$.

(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.

Collisions are unavoidable.

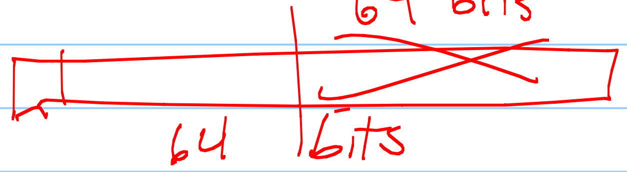
when $k_1 \neq k_2$ but $h(k_1) = h(k_2)$

First: map key to a number

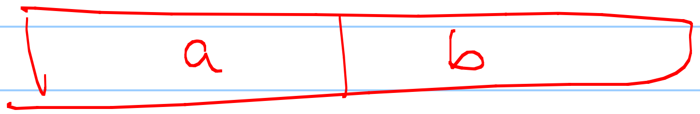
Say we want keys to fit in an $\overset{\text{32 bits}}{\text{int}}$.

What can we do for $\underbrace{\text{int}}_{\text{32 bits}}$, $\underbrace{\text{char}}_{\text{32 bits}}$, & $\underbrace{\text{short}}_{\text{32 bits}}$ types?

Now what about $\underbrace{\text{long}}_{\text{64 bits}}$ or $\underbrace{\text{float}}$?



a >> 32



a+b ← simplist way to hash

32 bits

64 bits

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

chop top 32 bits

chops off bottom 32 bits