CS180 – Hashing

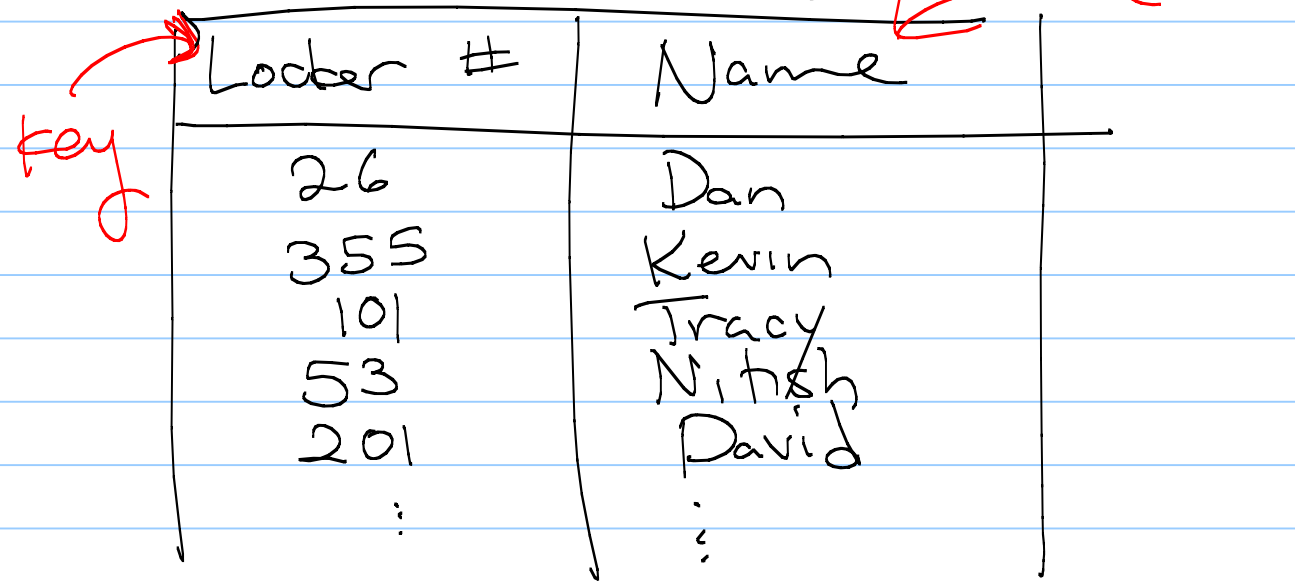
Note Title 4/27/2011

Announcements

- HW7 – graded
  HW8 back soon?
- HW9 – due Tomorrow
- Review Friday, test Monday
- HW10 posted – decoding
  checkpoint Monday before break
- No lab tomorrow

New problem: Data Storage

Ex:



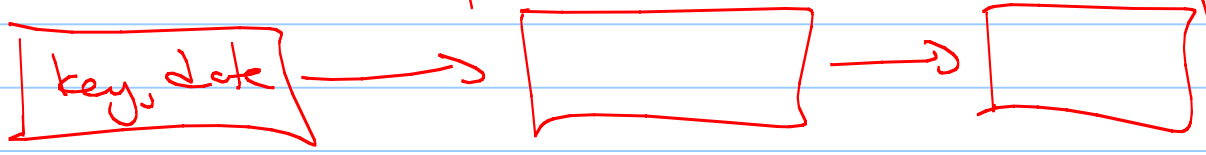
We want to be able to retrieve a name quickly when given a locker number.

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

How could we store this?

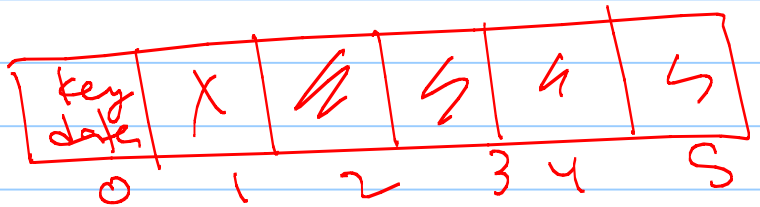
① Lists

Space: O(n) — 1 node per person



find: O(n) insert: O(1)
delete: O(n)

② Vector

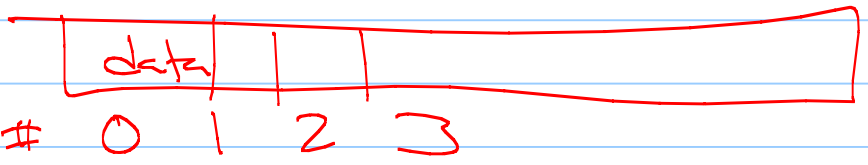


insert: O(1)
(sorted) O(n)
remove: O(n)

Space: O(n)

find: O(n)
O(log n) (if sorted)

③ Array:



space: O(m) → # lockers

find: O(1)

insert: O(1)

delete: O(1)

④ AVL trees - O(log n) for all ops
O(n) size

Other examples

- Course # and schedule info
- Flight # and arrival info
- URL and html page
- Color and BMP

Not always easy to figure out how to store and look up.

Dictionaries ~ associative arrays

A data structure which supports the following:

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

Note: Everything is based on keys!

Data Structures

First thing to note:

An array **is** a dictionary.

key: index

data: value in that spot

Other alternatives:

(see a few slides back)

Hashing

Assuming $m >> n$, an array is not very space efficient.

We would like to use $O(n)$ space, not $O(m)$.

But then the key needs to get smaller.

Additional challenge: nonnumeric keys.

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

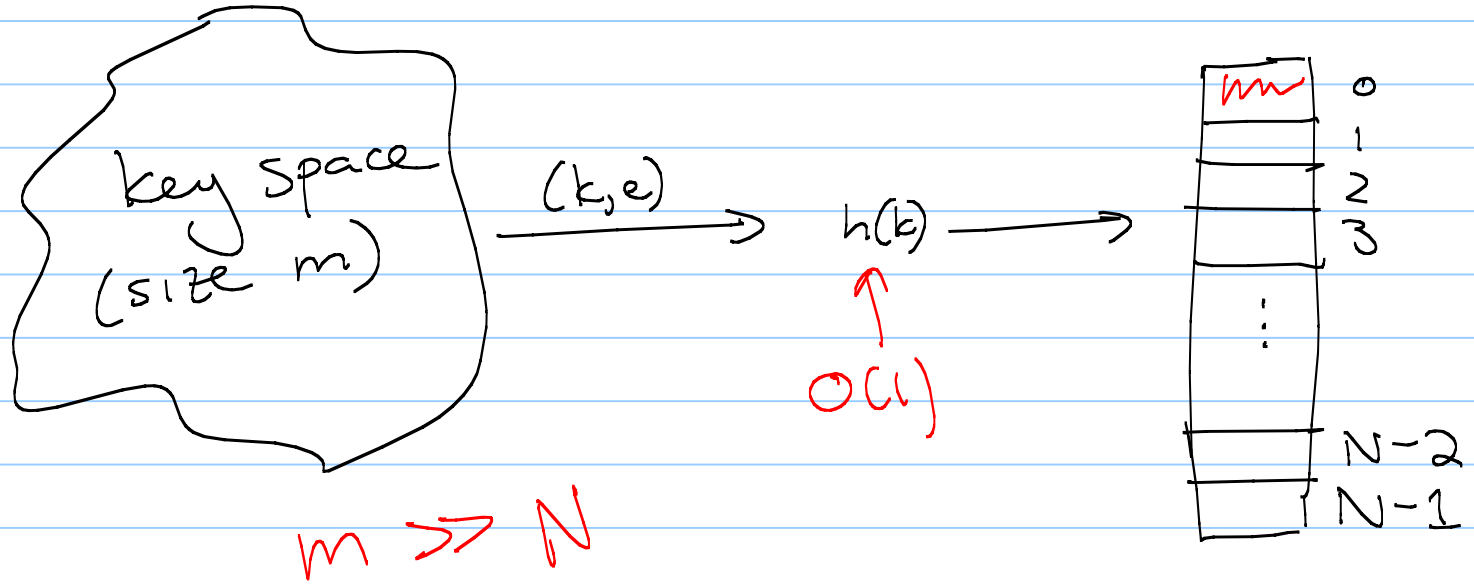
(N should be much smaller than m = # of keys.)

Then given (k, e), we store (k, e) in array spot $A[h(k)]$.

$h(k)=0$

Good hash functions:

- Are fast
- Don't have <u>collisions</u>: $k=k'$ but $h(k)=h(k')$



$m >> N$

So we have a few steps.

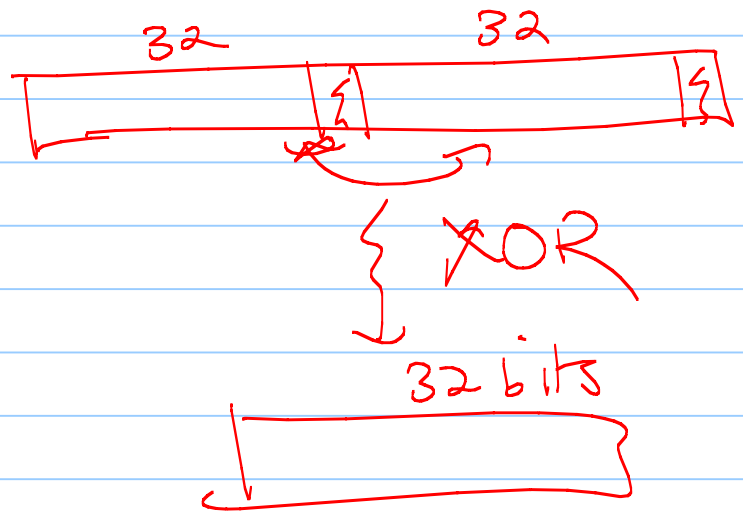
1. Make key a number
2. Compress that number to $[0, N-1]$
3. Since not perfect, handle collisions somehow.

1 Take key and make it a number.
(Remember, keys can be anything!)

Ex: char, int, or short (all 32-bits)

↳ cast immediately to a #

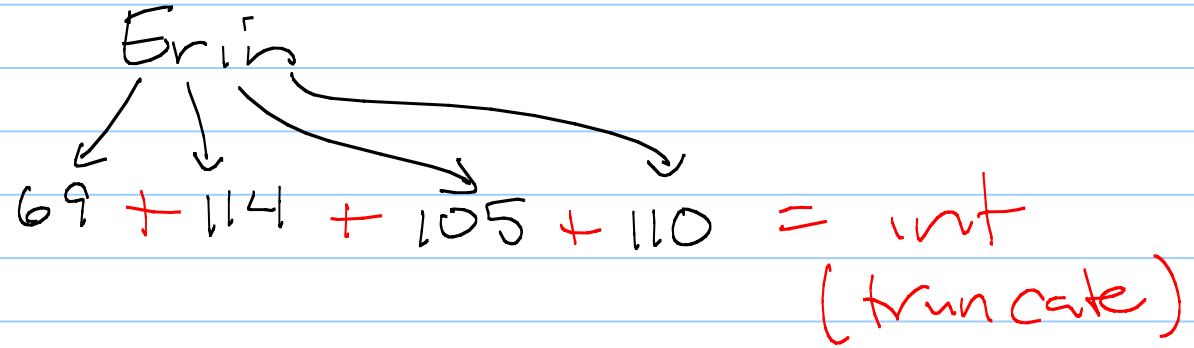
Ex: long or float — 64 bits
(K needs to be 32 bits)



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

What about strings?

(Think ASCII.)



Goal: a single int.

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

temp01 and temp10 and promote1

t+e+m+p+0+1 = t+e+m+p+1+0

→ go to same int

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

Next time: locality