Note Title 4/27/2011

# CS180 – Hashing

## Announcements

- HW10 is posted
- HW9 due Saturday
- Last day in class will be review
- Office hours today: 12–1:30

# Other trees

- Splay trees: After every insert/delete, performs a move-to-root operation called splaying, which gives an <u>amortized</u> O(log n) behavior

- Red-Black trees: more complex than AVL trees, + give only O(1) "rotations" after each insert or delete

## New problem: Data Storage (Dictionary)

Ex:

key

| Locker # | Name |
|---|---|
| 26 | Dan |
| 355 | Kevin |
| 101 | Tracy |
| 53 | Nitish |
| 201 | David |
| ⋮ | ⋮ |

data

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?

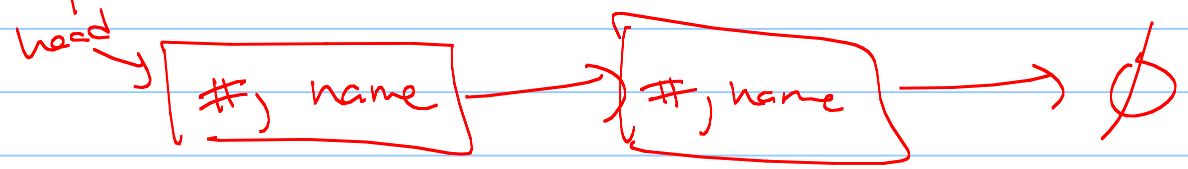
1. Array (or vector) : make array of length m of Strings (or struct holding data)

   access time: $O(1)$

   insert : $O(1)$

   space : $O(m)$

2. List :

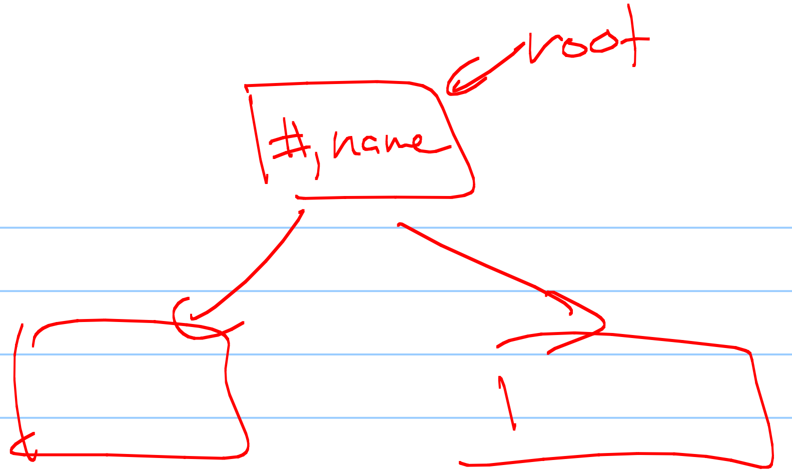


   access: $O(n)$

   insert: $O(1)$

   space: $O(n)$

③ Search tree:
(AVL):
search: $O(\log n)$
insert: $O(\log n)$
size: $O(n)$



Treaps: $O(n)$ find & insert

## 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 (or 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 : integer from 0 to capcaty - 1

data : values are in array

Other alternatives:

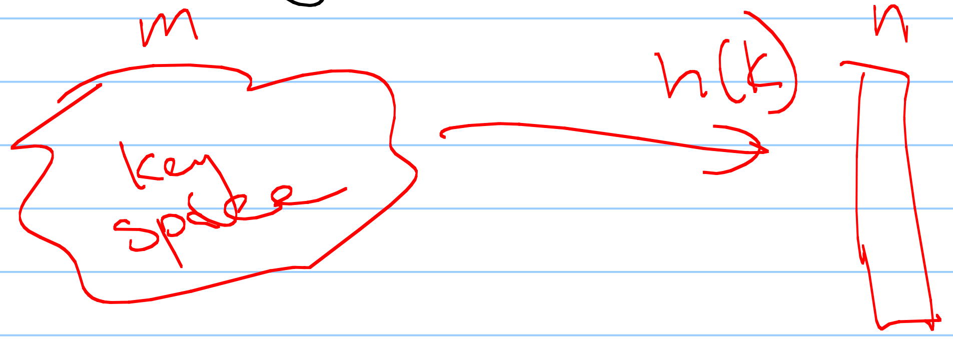
(see 2 slides back)

# 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.



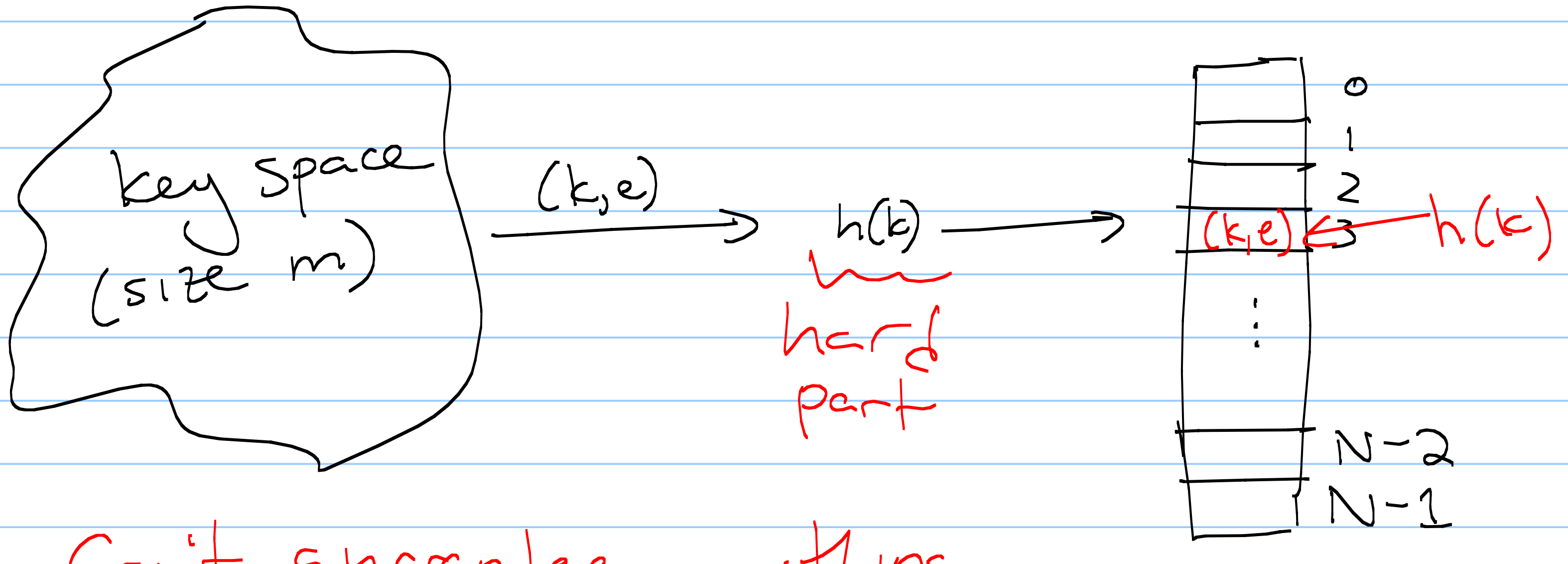
Dfn: A hash function $h$ maps each key in our dictionary to an integer in the range $[0, N-1]$.

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

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

# Good hash functions:

- Are fast ← $O(1)$
- Don't have collisions.
  (minimize collisions)



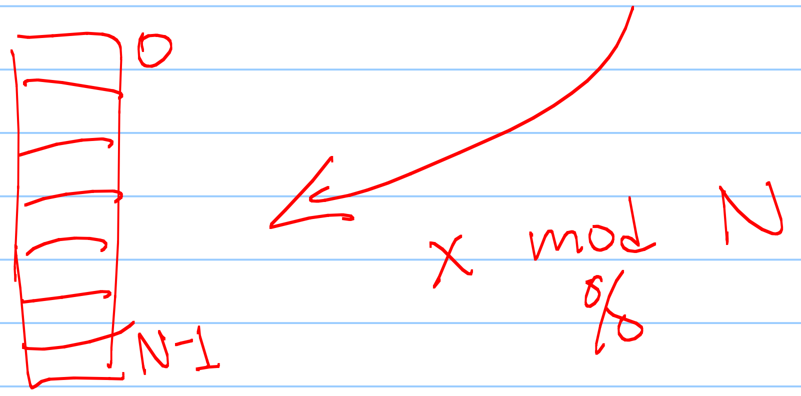
Can't guarantee anything.
(Work pretty well anyway.)

So we have a few steps.

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

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

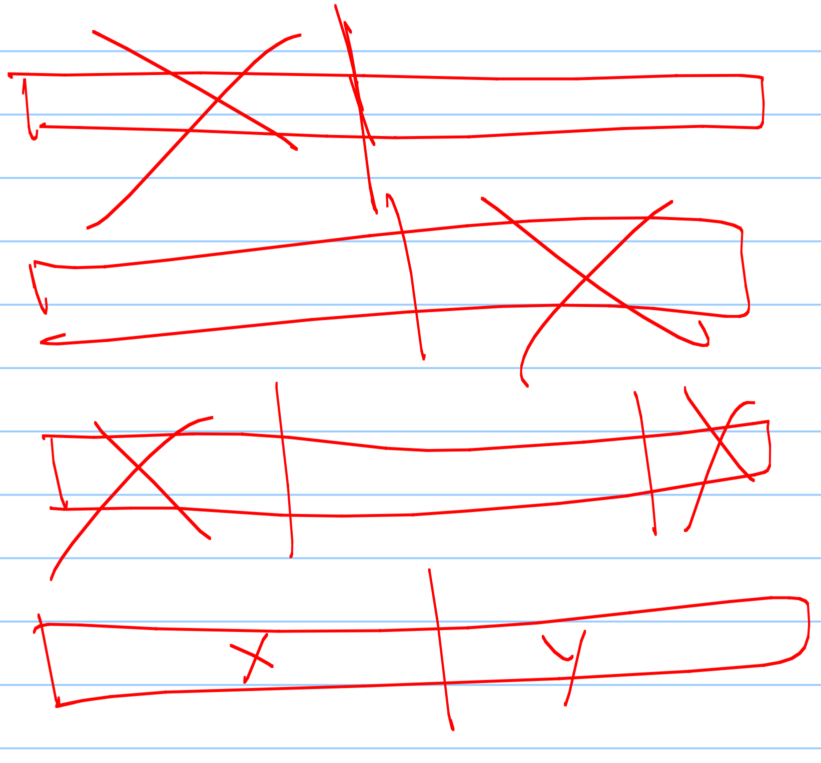
↳ cast to integers 0 to $2^{31}$



Ex: long or float — 64 bits

(k needs to be 32 bits)

①



take "random" 32-bits

$x \oplus y$

$x + y$

$x - y$

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