CS180 - Hash Tables p.3

Note Title 12/2/2010

Announcements

- check point today
- Program is due Sat. by midnight

Dictionaries:

A structure which supports the following:

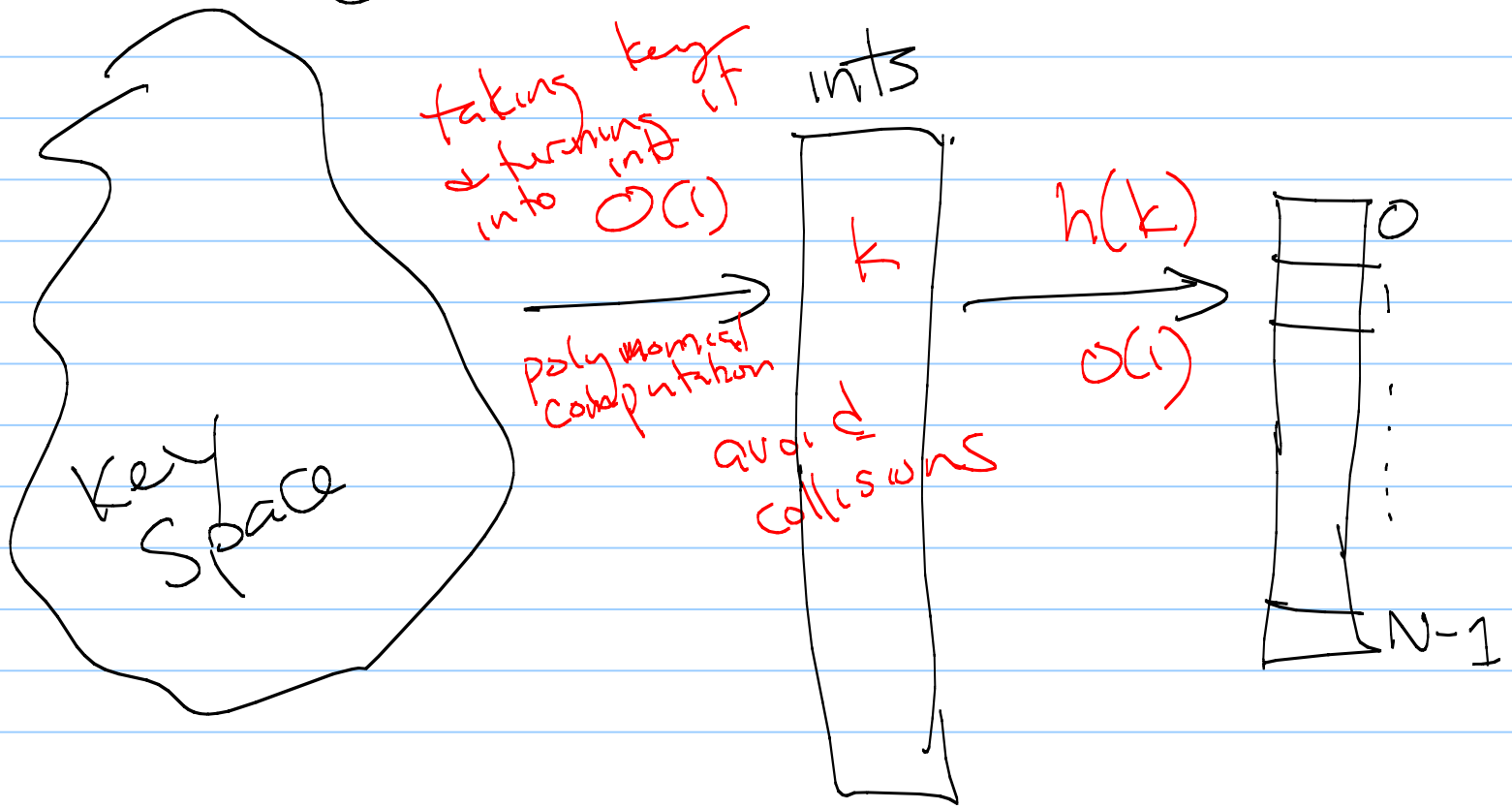
```
void insert(keyType &k, dataType &d)
dataType find(keyType &k)
void remove(keyType &k)
```

Examples: (key, data)

- Locker number & name
- Flight # & arrival info

Hashing for fast lookups

Hashing - big picture



Collisions

Can we ever totally avoid collisions?

NO

Keyspace is larger than our array!

Yesterday - strategies to deal with collisions.

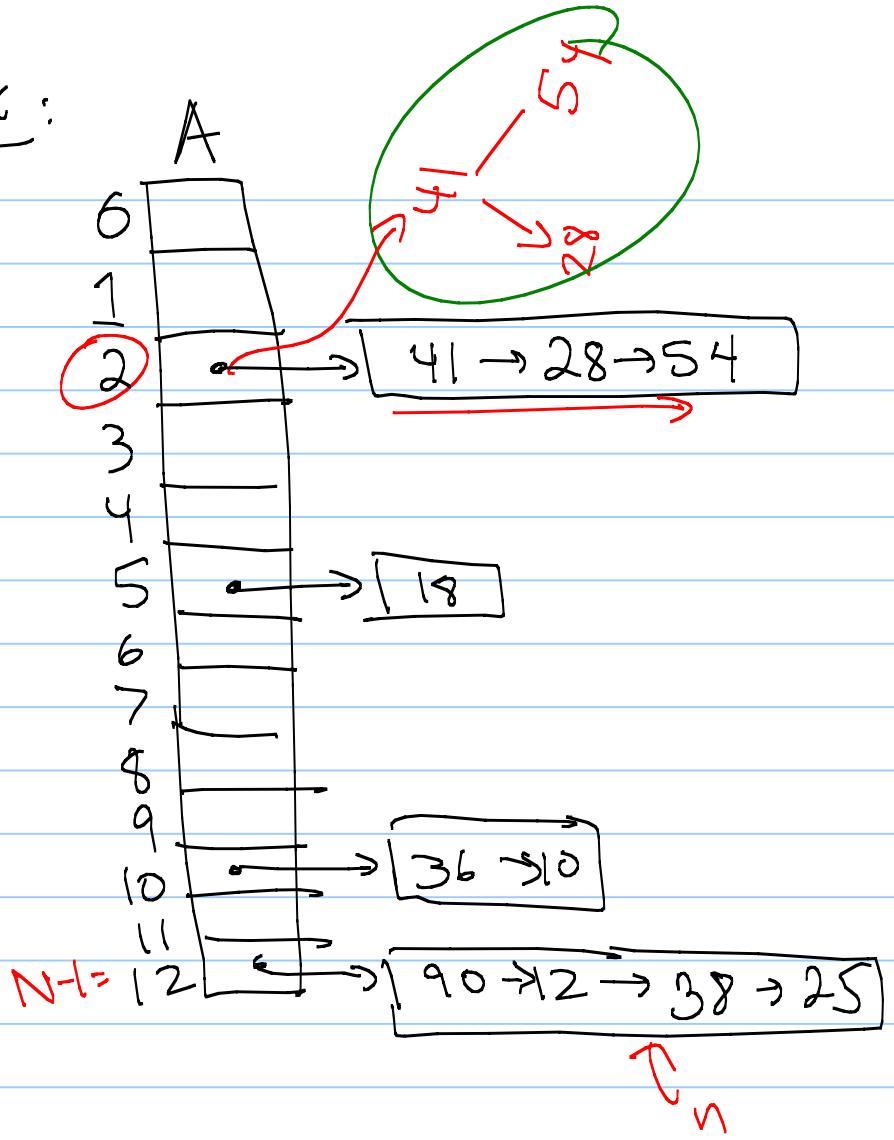
How can we handle collisions?

Strategy #1:

Do we have data structures to store more than one thing??

- vectors
- lists
- tree

Ex:



N = size of table
n = # of elements in the table

Running times:

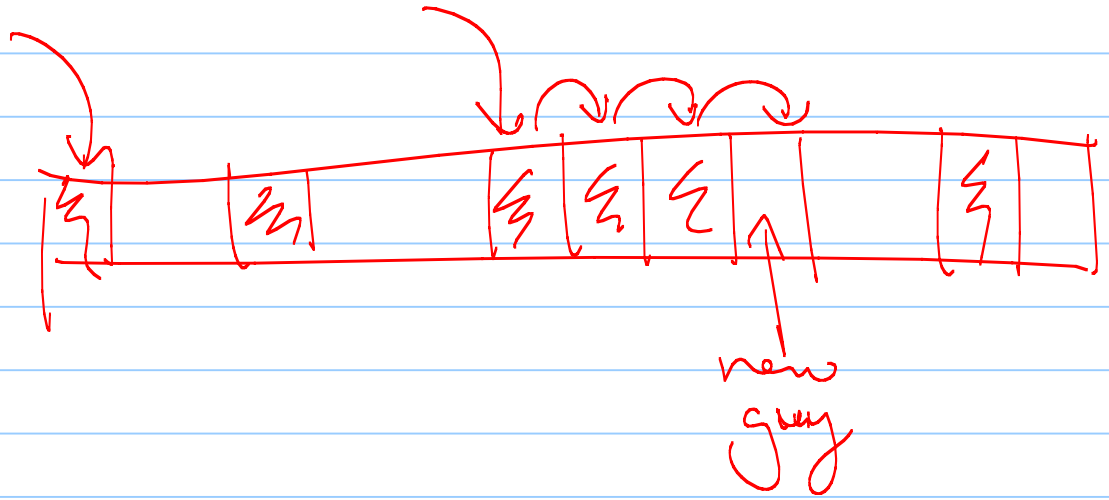
(list)
(tree)

Find: O(n)
(every key hashed to same spot)
O(log n)

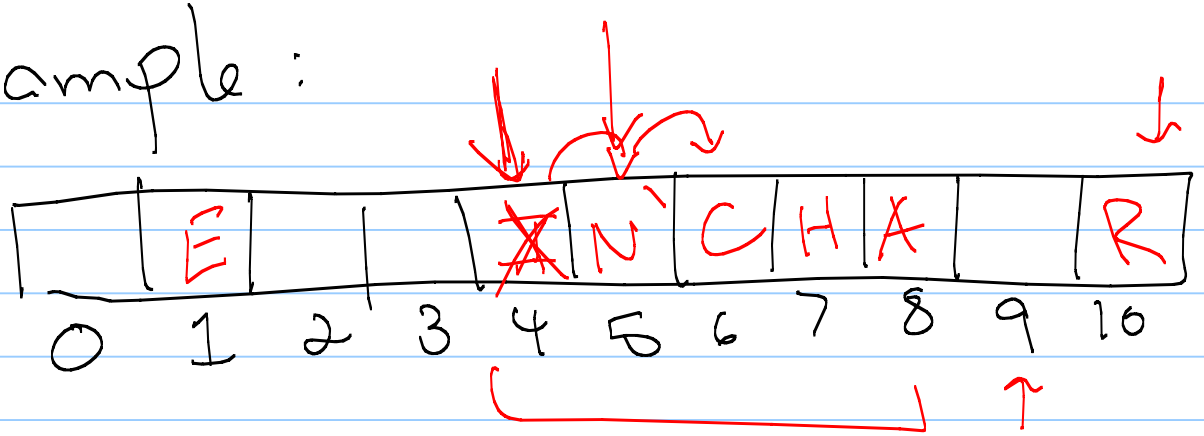
Insert: O(1)
O(log n)

② Linear Probing:

Instead of lists, if we hash to a full spot, just keep checking next spot until it is empty.



Example:



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

key, data

insert(12, E) — $12 \bmod 11 = 1$

insert(21, R) — $21 \bmod 11 = 10$

insert(37, I) — $37 \bmod 11 = 4$

insert(26, N) — $26 \bmod 11 = 4$

insert(16, C) — $16 \bmod 11 = 5$

insert(5, H) — $5 \bmod 11 = 5$

insert(15, A) — $15 \bmod 11 = 4$ $1 = 4$

M = array size
n = # elements

Running times:

Find? Compute h(k) + then walk along until we find (k, letter) or a blank (not a deleted value)

} O(n)

Insert? O(n) (might have a lot of neighbors)

Remove? Instead of removing, mark as deleted.

Quadratic Probing.

Notice: Linear Probing checks spot $A[h(i)+1 \bmod N]$ if $A[h(i)]$ is full.

+2
+3 +4 ...

To avoid clusters, instead try

$A[(h(i) + j^2) \bmod N]$ where $j = 0, 1, 2, 3 \ldots$

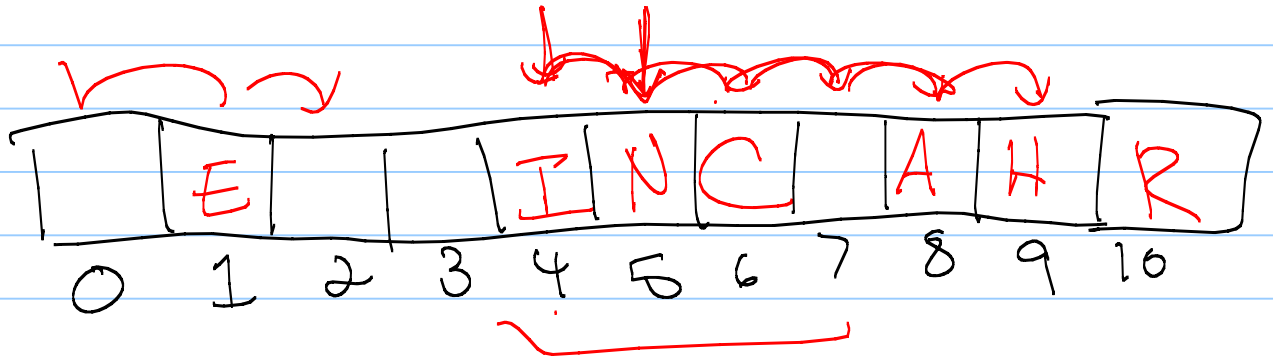
$A[h(i) \bmod N]$ if full

① $A[h(i)+1 \bmod N]$ if full

② $A[h(i) + 2^2 \bmod N]$

$A[h(i) + 3^2 \bmod N)$

$+4^2$



insert(12, E)
insert(21, R)
insert(37, I)
insert(26, N)
insert(16, C)
insert(5, H)
insert(15, A)

12 mod 11 = 1
21 mod 11 = 10 ←
37 mod 11 = 4
26 mod 11 = 4 ←
16 mod 11 = 5 ←
5 mod 11 = 5 ←
15 mod 11 = 4

Quadratic Probing issues:

- still cause "secondary clustering"
- N really must be prime for this to work
  (can't have a lot of divisors)
- Even with N prime, may fail if array is half full

3 Double Hashing

Try $A[h(i)]$

If full:

In linear probing, $+j$

In quadratic, $+j^2$

$$A[h(i) + f(j) \bmod N]$$

$$\rightarrow f(j) = j \cdot h'(k')$$

h' is another hash function

[k' is key of data already stored in $A[h(i)]$

Load Factors

Most of these techniques only work well if $\frac{n}{N} < .5$

Even chaining gets worse if $\frac{n}{N} > .9$

A lot of code periodically checks $\frac{n}{N}$, + rehashes if it is $> .5$

+ checks if many things have been removed