6.006	Lecture 6	Feb. 26, 2008

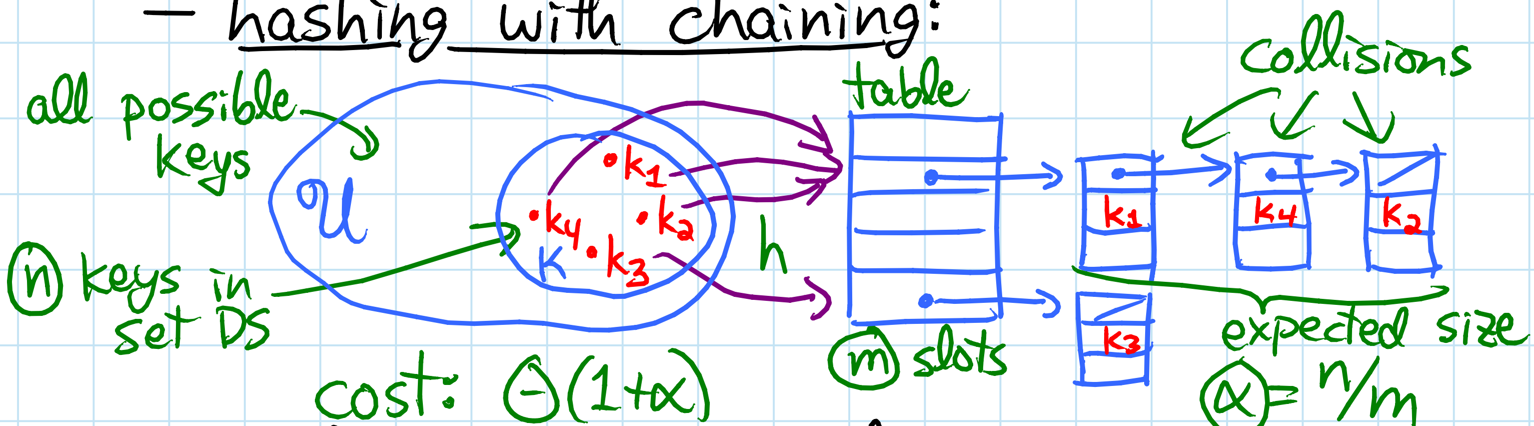
Outline: Hashing II

- table resizing
- amortization
- string matching & Karp-Rabin
- rolling hash

Reading: CLRS 17 & 32.2

Recall:

- hashing with chaining:



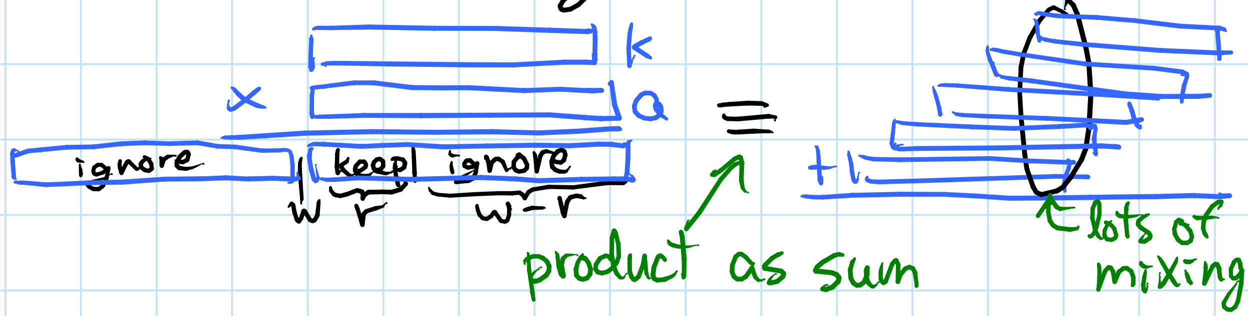
- Multiplication Method:

$$h(k) = [(a \cdot k) \bmod 2^w] \gg (w-r)$$

where m = table size $= 2^r$

w-bit machine words

a = odd integer between 2^{w-1} & 2^w



How large should table be?

- want $m = \Theta(n)$ at all times
- don't know how large n will get @creation
- m too small $\Rightarrow$ slow; m too big $\Rightarrow$ wasteful

Idea: start small (constant)
grow (& shrink) as necessary

Rehashing: to grow or shrink table
hash function must change (m, r)
$\Rightarrow$ must rebuild hash table from scratch

```
for item in old table:
    insert into new table
```

$\Rightarrow \Theta(n+m)$ time $= \Theta(n)$ if $m = \Theta(n)$

How fast to grow? when n reaches m, say

- $m \mathrel{+}= 1$?
  $\Rightarrow$ rebuild every step
  $\Rightarrow$ n inserts cost $\Theta(1+2+\cdots+n) = \Theta(n^2)$
- $m \mathrel{*}= 2$? $m = \Theta(n)$ still $(r \mathrel{+}= 1)$
  $\Rightarrow$ rebuild at insertion 2^i
  $\Rightarrow$ n inserts cost $\Theta(1+2+4+8+\cdots+n)$
  really the next power of 2
  $= \Theta(n)$
- a few inserts cost linear time, but $\Theta(1)$ "on average"

Amortized analysis — common technique in DSs

— like paying rent: \$1500/month ≈ \$50/day

- operation has amortized cost $T(n)$ if k operations cost $\leq k \cdot T(n)$
- "$T(n)$ amortized" roughly means $T(n)$ "on average", but averaged over all ops.
- e.g. inserting into a hash table takes $O(1)$ amortized time

Back to hashing: maintain $m = \Theta(n)$ so also support search in $O(1)$ expected time assuming simple uniform hashing

Delete: also $O(1)$ expected as is

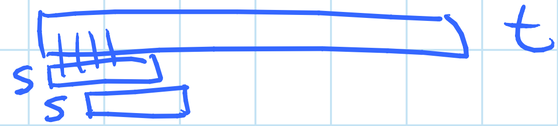
- space can get big with respect to n
  e.g. $n \times$ insert, $n \times$ delete
- solution: when n decreases to $m/4$, shrink to half the size

$\Rightarrow O(1)$ amortized cost for both insert & delete

- analysis harder; see CLRS 17.4

String matching: given two strings s & t, does s occur as a substring of t? (and if so, where & how many times?)

e.g. s = '6.006' & t = your entire INBOX ('grep' on UNIX)



Simple algorithm:

```
any(s == t[i:i+len(s)]
        for i in range(len(t)-len(s)))
```

- $O(|s|)$ time for each substring comparison

$\Rightarrow O(|s| \cdot (|t|-|s|))$ time

$= O(|s| \cdot |t|)$ potentially quadratic

Karp-Rabin algorithm:

- compare h(s) == h(t[i:i+len(s)])
- if hash values match, likely so do strings
  - can check s == t[i:i+len(s)] to be sure ~ cost $O(|s|)$
  - if yes, found match — done
  - if no, happened with probability $< \frac{1}{|s|}$

    $\Rightarrow$ expected cost is $O(1)$ per i
- need suitable hash function
- expected time $= O(|s| + |t| \cdot \text{cost}(h))$
  - naively h(x) costs $|x|$
  - we'll achieve $O(1)$!
  - idea: t[i:i+len(s)] ≈ t[i+1:i+1+len(s)]

Rolling hash ADT: maintain string subject to
- h(): reasonable hash function on string
- h.append(c): add letter c to end of string
- h.skip(c): remove front letter from string, assuming it is c

Karp-Rabin application:

```
for c in s: hs.append(c)
for c in t[:len(s)]: ht.append(c)
if hs()==ht(): ...
for i in range(len(s), len(t)):
    ht.skip(t[i-len(s)])
    ht.append(t[i])
    if hs()==ht(): ...
```

First three lines: $O(|s|)$

Loop: $O(|t|)$

Data structure: treat string as a multidigit number u in base a

a = alphabet size, e.g. 256

- $h() = u \bmod p$ for prime $p \approx |s|$ or $|t|$ (division method)
- h stores $u \bmod p$ & $|u|$, not u
  $\Rightarrow$ smaller & faster to work with ($u \bmod p$ fits in one machine word)
- h.append(c): $(u \cdot a + \mathrm{ord}(c)) \bmod p$
  $= [(u \bmod p) \cdot a + \mathrm{ord}(c)] \bmod p$
- h.skip(c): $[u - \mathrm{ord}(c) \cdot (a^{|u|-1} \bmod p)] \bmod p$
  $= [(u \bmod p) - \mathrm{ord}(c) \cdot (a^{|u|-1} \bmod p)] \bmod p$