6.006 Recitation 6

Agenda	
Ch 17	• amortized analysis
17.1	• normal
17.2	• accounting
	• ex: binary counter
17.4	• Dynamic Tables
	• deletion analysis
	• delete + insert worst case.
ch 32.2	• Rabin-Karp
	• decimal example
	• binary example (?)
	• general case.

Amortized Analysis

- average case : expected cost probabalistic-ally.
- aggregate : total cost over x operations
- amortized : $\frac{\text{aggregate}}{\#\text{ ops.}}$

no probability involved.

ex: rolling dice → average case is 3.5

drawing 52 cards from a deck without replacement → aggregate value $= (13+12+\cdots+2+1)\cdot 4$

amortized value $= \frac{\text{aggr.}}{52} = 7$

→ straight forward analysis

for each operation - determine its cost

$T(n) = \sum_{i=1}^{n} \text{operation } i$

amortized $= \frac{T(n)}{n}$

to generalize determine pattern of costs for operations.

i.e. all ops have a constant cost except when $i = 2^k$ $k = 0, 1, 2 \cdots$

then the operation costs $2i = 2^{k+1}$.

$\boxed{\text{ex}}$ BINARY COUNTER

counter value	binary representation	i^{th} step cost	total cost
0	0 0 0 0	ϕ	1
1	0 0 0 1	1	2
2	0 0 1 0	2	3
3	0 0 1 1	1	4
4	0 1 0 0	3	7
5	0 1 0 1	1	8
6	0 1 1 0	2	10
7	0 1 1 1	1	11
8	1 0 0 0	4	15
9	1 0 0 1	1	16
10	1 0 1 0	2	18

pattern :

n every increment flips the last b

$+$

$n/2$ every other inc flips 2^{nd} bit

$+$

$n/4$ … 4^{th} … 3^{rd}

$+$

$\vdots$

$n/2^i$ … 2^i … $(i+1)^{th}$.

$n + n/2 + n/4 + \cdots 1$

$n(1 + 1/2 + 1/4 + \cdots) \leq 2n$

thus the aggregate cost is $\leq 2n = O(n)$
and the amortized cost is $\leq 2 = \Theta(1)$

→ The Accounting Method

we "charge" some amount for each operation
- the charge is independent of its cost! (if you want)

we pay the actual cost of the operation.

idea: store credit for the times when an operation is expensive.
like insurance!
never let your balance drop below zero.

useful when you have varying operations that contribute to each other's running times.

example in book: stack (push pop multipop)

ex: binary counter
charge \$2 for each insert
- at most one 1 is added – costs \$1
- every 1 has a dollar "stored" on it for when it is flipped back to zero.
- all flips from $1 \to 0$ use previously stored money.

DYNAMIC TABLES

from class:

$\alpha = \frac{n}{m}$

must be below a threshold of 4/5

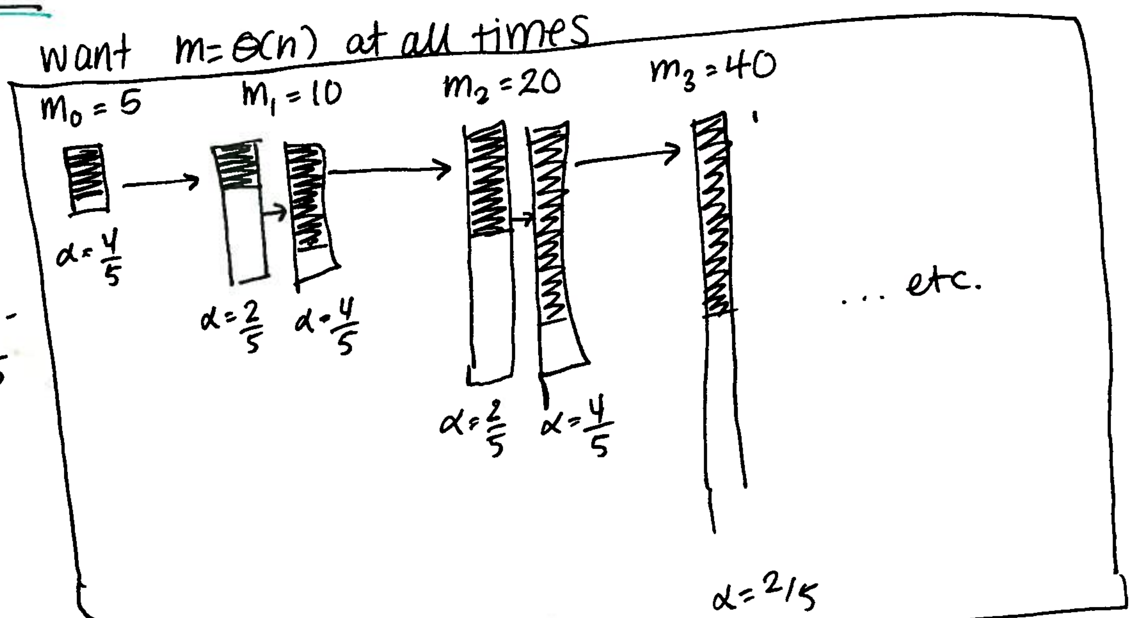

table growth during a series of insert operations.
(shading represents α, not actual locations of values in table)

aggregate: $i = 2^k \to cost(i) = \Theta(i = 2^k)$ else $cost(i) = 1$

aggregate $\quad cost(i) = \begin{cases} 1 & \text{else} \\ 2^k & i = 2^k \end{cases} \quad \Rightarrow T(n) = 2^k + 2^{k-1} + 2^{k-2} + \cdots + n$

$$= n + 2^k(1 + \tfrac{1}{2} + \tfrac{1}{4} + \cdots)$$
$$\leq n + 2^k(2)$$

and $n = 2^k$ so

$$= 3n$$

amortized $\quad \frac{3n}{n} = 3 = \Theta(1)$

Deletions

same idea in reverse $\quad \alpha \geq 1/5$ at all times.

- why not 2/5? b/c then ins, del, ins, del at boundary very expensive.

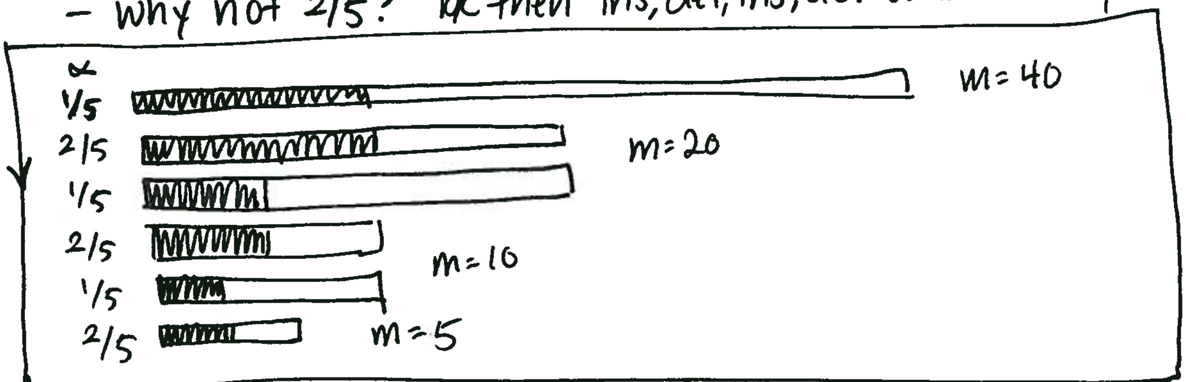

table shrinkage during a series of delete operations.

aggregate: when table reaches $1/5\, m$ (or 2^k in general) we need to shrink otherwise just delete element

$cost(i) = \begin{cases} 2^k & i = 2^k \\ 1 & \text{else} \end{cases} \quad \Rightarrow \quad T(n) =$ same as above $\leq 3n$

(assuming n deletions brings us to an empty list)

Deletions and Insertions : worst case

$$\underset{m=5}{\text{I I I I}} \uparrow^{m=10} \text{D D} \downarrow_{m=5} \text{I I} \uparrow^{m=10} \text{D D} \downarrow_{m=5} \cdots$$

worst case series of operations.

cost: $\underbrace{1\ 1\ 1\ 4\ 1\ 1\ 2}$ $\ 1\ 1\ 4\ 1\ 1\ 2 \ \cdots$

repeating block

cost $= 6 + 4 = 10$

ops $= 6$

amortized $= \frac{10}{6} = \frac{5}{3}$

Rabin Karp

goal: string search = find pattern p in string s.
idea: treat each letter as a digit in base b
then the pattern is a number in base b of length len(p) digits.
see if that number occurs in s by a rolling hash.
use a simple modulo hash

= string of digits: $b=10$

find $\boxed{6123}$ in $\boxed{8\ 7\ 1\ 2\ 4\ 6\ 1\ 2\ 3\ 4\ 9\ 0\ 0\ 1}$

p   s

$h(p) = 6123 \ \% \ 17$ $\boxed{8\ 1\ 5\ 13\ 5\ 3\ 10\ 3\ 5\ 4\ 8}$
$= 3$

actual match   spurious hit

when you find a potential match: check p vs s at that spot to verify.
- need to compute $h(s[i \cdots j])$ from $h(s[i-1 \cdots j-1])$ quickly.
this is where rolling hashes come in:

$$a\ b\ c\ d \Rightarrow 10^3 a + 10^2 b + 10c + d = h(s1)$$
$$a\ b\ c\ d\ 0 \Rightarrow 10^4 a + 10^3 b + 10^2 c + 10d = 10\,h(s1)$$
$$a\ b\ c\ d\ e \Rightarrow 10^4 a + 10^3 b + 10^2 c + 10d + e = 10\,h(s1) + e$$
$$s2 = b\ c\ d\ e \Rightarrow 10^3 b + 10^2 c + 10d + e = 10\,h(s1) + e - 10^4 a$$

so $h(s2) = \underset{b}{10}\,h(s1) + \underset{\text{new digit}}{e} - \underset{b\ \text{len}(p)}{10^4}\ \underset{\text{old digit}}{a}$ which is linear.

work through example above.

= general case

$$(\,h(new) = b\,h(old) + d_{new} - b^{\text{len}(p)} d_{old}\,)\ \%\ m$$

time: $O(\text{len}(p))$ to compute hash
$O(\text{len}(p) + n)$ to compute hashes in S
$O(\text{len}(p) \cdot \#\text{potential matches})$ to verify potential match

= binary example

$\boxed{0\ 0\ 1}$ in $\boxed{1\ 1\ 0\ 0\ 1\ 1\ 0\ 1\ 1\ 0\ 0\ 0\ 1\ 1\ 1\ 0}$

s   mod 3