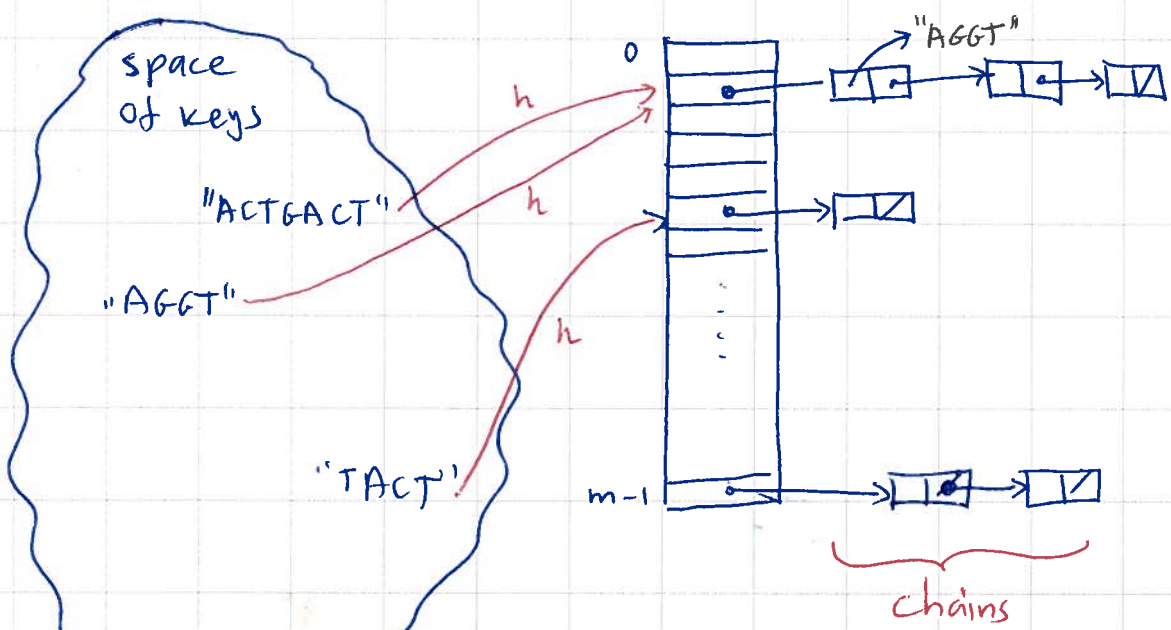
6.006 Lecture 6: Hashing II

- Hash functions 11.3
- Resizing & amortized analysis (CLRS ch 17)
- Rolling hash

Recap of lecture 5:



m slots, n keys, load factor $\alpha = \frac{n}{m}$

Computing $h(x)$ (question ②)

Lots of ways. Here's a good one for integer x's (large integers)

Let p be a prime $p \geq m$

pick a $0 < a < p$

pick b $0 < b < p$

let $h(x) = ((ax+b) \bmod p) \underbrace{\bmod m}_{\text{only if } p > m}$

e.g $m = 1{,}000{,}000$

$p = 1{,}000{,}003$

$a = 314{,}159$

$b = 271{,}828$

} can reuse p, a, b for smaller m's

If keys are not integers, convert them first to integers

$x =$ "ATTGCTAC" treat as a base-4 integer

$x =$ "Boston" treat as a base-52 integer
or base-26
or base 128...

Textbook describes other methods.

Amortized analysis

Some insertions are really expensive (the ones that trigger doubling).

So ~~instead~~ instead of analyzing the worst case cost per operation, we ~~compute co~~ analyze the total cost $T(n)$ for n operations, divided by n: amortized analysis.

A bit like analyzing the average case but without making any probabilistic assumptions.

Suppose we start with $m=5$

$T(1) = 1$ insertion

$T(2) = 1 + 1 = 2$ new insertion

$T(3) = 3$

$T(4) = 3 + 1 + 4 = 8$ new insertion, copying

$T(5) = 9$

$T(6) = 10$

$T(7) = 11$

$T(8) = 11 + 1 + 8 = 20$

if $n = 2^k$: $T(n) = n + \left(n + \frac{n}{2} + \frac{n}{4} + \cdots + 8 + 4\right) \leq n + 2n = 3n$

so $T(n)/n \leq 3$: worst case amortized cost per insertion is 3

Resizing a hash table (and arrays in general)

We want $m = \Theta(n)$ at all times

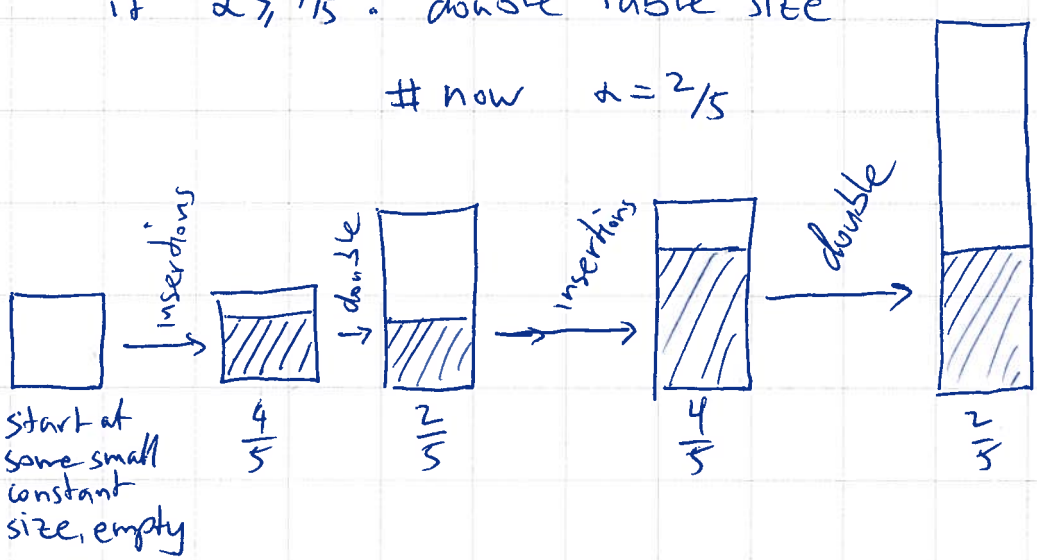
- m too small: large α, slow searching (long chains)
- m too small: expensive to enumerate, initialize
  wastes memory

How to set m if we do not know how large n will be? (or how small)

Adjust m as n changes:

if $\alpha \geq 4/5$: double table size

now $\alpha = 2/5$



If no deletions, $\frac{2}{5} \leq \alpha < \frac{4}{5}$ or $n, m < \text{const}$

Another view of amortized analysis: every insertion deposits 3 "units of work" in a savings account, but ~~used~~ immediatly uses one unit to pay for the insertion.

When the account is full enough to copy entire table, do it (does not work exactly for our example starting from $m=5$, but can be made to work).

Deletions

If $\alpha \leq 1/5$: halve table size

amortized cost per operation is still ≤ 3

Example: deleting so n decreases from $\frac{2}{5}m$ to $\frac{1}{5}m$ leaves $2(\frac{1}{5}m)$ units in the account; this ~~pay~~ pays for copying the remaining $\frac{1}{5}m$ elements (and still leaves ~~te~~ $\frac{1}{5}m$ in the account

Why not halve the table when $\alpha < \frac{2}{5}$?

Rabin-Karp String matching & Rolling hashes

Given pattern $P[1..m]$
text $T[1..n]$ } lists of characters

does P occur in T?

E.g. find "ATG" in "AATCGC..."

Idea:

- Compute $h(P)$
- for each length-m window of T, $T[i, \cdots i+m-1]$

$$\underbrace{A\,A\,T}\;C\;G\;C$$

compute $h(T[i \cdots i+m-1])$ and compare to $h(P)$

if $=$: check to see if really a match

if $\neq$: move on to next i

- Use a hash function h s.t. can compute $h(T[i \cdots i+m-1])$ from $h(T[i-1 \cdots i+m])$ easily

$\Rightarrow$ rolling hash

Example of a rolling hash

	0	1	2	3
Alphabet	A	C	G	T

h = value of string % p for prime p

T = C T A T T A C G T

1 3 0 3 3 0 1 2 3 4 ← base 4

1 8 4 0 9 1_{10}

453 mod p = 1009

delete C

T A T T A C G T G

add G

value of leading c = 1·48 = 1·960 (mod p, precompute!)

so dropping leading c ⇒ h = 453 - 690

= 502 mod p

shifting and adding a G:

$h = \underset{\text{shift}}{4} \cdot 502 + \underset{G}{2} = 1001 \bmod p$

$\left.\begin{array}{ll} \text{subtract} & \times 1 \\ \text{multiply} & \times 1 \\ \text{add} & \times 1 \end{array}\right\}$ cost of moving h to next window