6.006 Lecture 6: Hashing II

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

Recap of lecture 5:

Space of keys

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 $p > m$
pick $a$ $0 < a < p$
pick $b$ $0 < b < p$
let $h(x) = ((ax + b) \mod p) \mod m$
only if $p \geq m$

E.g.
$m = 1,000,000$
$p = 1,000,003$
$a = 314,159$
$b = 231,828$

Can reuse $p,a,b$ for smaller $m$'s

If keys are not integers, convert them first
to integers
$x = "ATTGCTAC"
\text{treat as a base-4 integer}
\text{or base-26}
\text{or base 128...}$

$x = "Boston"
\text{treat as a base-52 integer}$

Textbook describes other methods.
Amortized analysis

Some insertions are really expensive (the ones that trigger doubling).
So instead of analyzing the worst case cost per operation, we 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 $m=5$

\[
\begin{align*}
T(1) &= 4 \
T(2) &= 1 + 4 = 2 \
T(3) &= 3 \
T(4) &= 3 + 4 + 8 = 15 \
T(5) &= 9 \
T(6) &= 10 \
T(7) &= 18 \
T(8) &= 17 + 1 + 8 = 26
\end{align*}
\]

If $n=2^k$, $T(n) = n + (n + \frac{n}{2} + \frac{n}{4} + \cdots + 8 + 4) \leq n + 2n = 3n$

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 \( \alpha \), 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 \( \alpha > \frac{4}{5} \) : double table size

  \[
  \# \text{ now } \alpha = \frac{2}{5}\]

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 immediately 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 $\alpha \leq \frac{1}{5}$: halve table size

amortized cost per operation is still $\leq 3$

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

Why not halve the table when $\alpha < \frac{2}{5}$?
Rabin-Karp string matching & Rolling hashes

Given pattern \( P[1..m] \) \{ lists of characters \}

text \( T[1..n] \)

does \( P \) occur in \( T \)?

E.g. find "BTG" in "AATCGC..."

Idea:
- Compute \( h(P) \)
  - for each length-\( m \) window of \( T \), \( T[i..i+m-1] \)
    - \( \text{AAATCGC} \)
    - \( \text{compute } h(T[i..i+m-1]) \) and compare to \( h(P) \)
      - if =: check to see it really a match
      - if \( \neq \): move on to next

- Use a hash function \( h \) s.t. can compute
  \( h(T[i..i+m-1]) \) from \( h(T[i-1..i+m]) \) easily
  \( \Rightarrow \) rolling hash
Example of a rolling hash

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 4 base 4

1 8 4 0 9 1 10

4 5 3 mod p = 1009

\[ \text{delete C} \]

T A T T A C G T

\[ \text{\downarrow add G} \]

value of leading C = 1 - 48 = 1.960

so dropping leading C \( \Rightarrow \) h = 453 - 690

= 502 mod p

\[ \text{\mod p, precompute!} \]

\[ \text{\downarrow shifting and adding a G:} \]

h = 4 \cdot 502 + 2 = 1004 \mod p

\[ \text{\\uparrow shift \uparrow G} \]

\[ \text{subtract \times 1} \]

multiply \times 1 \}

\[ \text{cost of moving h to next window} \]

add \times 1