Lecture Overview

- Table Resizing
- Amortization
- String Matching and Karp-Rabin
- Rolling Hash

Readings

CLRS Chapter 17 and 32.2.

Recall:
Hashing with Chaining:

![Diagram of Chaining in a Hash Table]

Figure 1: Chaining in a Hash Table

Multiplication Method:

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

where \( m = \) table size = \( 2^r \)

\( w = \) number of bits in machine words

\( a = \) odd integer between \( 2^{w-1} \) and \( 2^w \)
Figure 2: Multiplication Method

How Large should Table be?

- want \( m = \theta(n) \) at all times
- don’t know how large \( n \) will get at creation
- \( m \) too small \( \implies \) slow; \( m \) too big \( \implies \) wasteful

Idea:
Start small (constant) and grow (or shrink) as necessary.

Rehashing:
To grow or shrink table hash function must change \((m, r)\)

\[ \implies \text{must rebuild hash table from scratch} \]
for item in old table:
insert into new table
\[ \implies \Theta(n + m) \text{ time} = \Theta(n) \text{ if } m = \Theta(n) \]
How fast to grow?

When $n$ reaches $m$, say

- $m + = 1$?
  - $\Rightarrow$ rebuild every step
  - $\Rightarrow$ $n$ inserts cost $\Theta(1 + 2 + \cdots + n) = \Theta(n^2)$

- $m \ast = 2$? $m = \Theta(n)$ still ($r + = 1$)
  - $\Rightarrow$ rebuild at insertion $2^i$
  - $\Rightarrow$ $n$ inserts cost $\Theta(1 + 2 + 4 + 8 + \cdots + n)$ where $n$ is really the next power of 2
  - $\Rightarrow \Theta(n)$

- a few inserts cost linear time, but $\Theta(1)$ “on average”.

Amortized Analysis

This is a common technique in data structures - like paying rent: $\$1500$/month $\approx \$50$/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 time

- 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 and delete - analysis is harder; (see CLRS 17.4).

String Matching

Given two strings $s$ and $t$, does $s$ occur as a substring of $t$? (and if so, where and how many times?)

E.g. $s = \text{‘6.006’}$ and $t = \text{your entire INBOX (‘grep’ on UNIX)}$
Simple Algorithm:

Any \((s == t[i : i + \text{len}(s)] \text{ for } i \in \text{range}(\text{len}(t) - \text{len}(s)))\)
- \(O(|s|)\) time for each substring comparison
  \(\implies O(|s| \cdot (|t| - |s|))\) time
  \(= O(|s| \cdot |t|)\) potentially quadratic

Karp-Rabin Algorithm:

- Compare \(h(s) == h(t[i : i + \text{len}(s)])\)
- If hash values match, likely so do strings
  - can check \(s == t[i : i + \text{len}(s)]\) to be sure \(\sim O(|s|)\)
  - if yes, found match — done
  - if no, happened with probability \(< \frac{1}{|s|}\)
    \(\implies\) 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: \(t[i : i + \text{len}(s)] \approx t[i + 1 : i + 1 + \text{len}(s)]\).

Rolling Hash ADT

Maintain string subject to

- \(h()\): reasonable hash function on string
- \(h\.\text{append}(c)\): add letter \(c\) to end of string
- \(h\.\text{skip}(c)\): 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(): ...
```

This first block of code is $O(|s|)$

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

The second block of code is $O(|t|)$

Data Structure:

Treat string as a multidigit number $u$ in base $a$ where $a$ denotes the alphabet size. E.g. 256

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