Lecture 20

Universal hashing

Hash functions must be deterministic, but we do not want to let users know what they are. However, they can often guess. As with quicksort, a malicious user can submit input data that makes the algorithm run an unreasonably long time, which can cause security and performance problems.

Example 20.1. Java String `hashcode` computes the address of a string using the Unicode values for each character ($A - Z$ corresponds to 65-90, $a - z$ to 97-122). For $s = s[0] \cdots s[n-1]$ use integer arithmetic to calculate the hash via $s[0] \cdot 31^{n-1} + s[1] \cdot 31^{n-2} + \ldots + s[n-1]$. Can you find two 2-character strings with the same hashcode? Why might this cause a problem in practice?

eg.
b c
98 67

$31 \times 98 + 67 = 31 \times 97 + 31 + 67$

$= 31 \times \underbrace{97}_{a} + \underbrace{98}_{b}$

$\therefore h(bC) = h(ab)$

$s_0 31 + s_1 = s_0' 31 + s_1'$

$31(\underbrace{s_0 - s_0'}_{\text{want to be 1}}) = \underbrace{s_1' - s_1}_{\text{want to be 31}}$

lower/upper case difference: $0 \cdots 25$

mix lC/uC diff: $7 \cdots 57$

A nice way to use randomization is to choose the hash function at runtime randomly from a family of hash functions. Ideally, the probability of a collision between two elements is only $1/m$, which is what a random guess would yield, so malicious users cannot force collisions. A family of hash functions that satisfies this is called ***universal***.

Formally, if F is a universal family of hash functions then for any keys $k \neq k'$,

$$\frac{1}{|F|} |\{h \in F : h(k) = h(k')\}| \leq \frac{1}{m}.$$

Example 20.2. Construct the family of hash function $F = \{h_{ab}\}$ by taking a prime $p \geq m$ and for $0 < a < p, 0 \leq b < p$ define

$$h_{ab}(x) = ax + b \mod p \mod m.$$

Prove that this is a universal family.

$F = \{h_{ab} : 0 < a < p,\ 0 \leq b < p\}$

$|F| = (p-1)p$

Get collision if and only if (iff): between keys x & y

$(ax+b) \bmod p \bmod m = (ay+b) \bmod p \bmod m$

iff $(ax+b) \bmod p - (ay+b) \bmod p$ is divisible by m

iff $a(x-y) \bmod p$ is divisible by m

iff $a \bmod p \longrightarrow a[(x-y) \bmod p] = im$ *

a between 0 and p

$\therefore = a$

cases:

1) $x - y \equiv 0 \bmod p \longrightarrow \frac{1}{p}$ chance of choosing 0

2) $x - y \not\equiv 0 \bmod p$

multiplicative inverse of $(x-y)$

Now since p is prime $(x-y) \not\equiv 0 \rightarrow (x-y)t \equiv 1 \bmod p$

from *: $a(x-y)t \bmod p = rmt$

$a = rmt$ find r.

we know $a < p$

$\therefore rmt \leq p-1$

$r \leq \frac{p-1}{m}$

so number of distinct values of (i) given different values of (a) is at most $\frac{p-1}{m}$

therefore number of hab causing a collision is at most $\frac{p(p-1)}{m}$ where $\frac{p-1}{m}$ is the amount of (a)s, and p is the potential amount of (b)s

so probability of collision is $\leq \frac{1}{p(p-1)} \frac{p(p-1)}{m} = \frac{1}{m}$

Combining 1 & 2

$$p(\text{collision}) = \frac{1}{p} + \frac{1}{m} \leq \frac{2}{m}$$

20.1 Summary of Table implementations

We summarise the table implementations that we have looked at so far in terms of worst and average case.

Table 20.1: Worst case running time (asymptotic order)

Data structure	Insert	Find	Delete	Traverse
Unsorted list (array)	1	n	n	n
Unsorted list (pointer)	1	n	n	n
Sorted list (array)	n	$\log n$	n	n
Sorted list (pointer)	n	n	n	n
Binary search tree	n	n	n	n
Balanced BST	$\log n$	$\log n$	$\log n$	n
Hash table (chaining)	n	n	n	$m+n$

Table 20.2: Average case running time (asymptotic order)

Data structure	Insert	Find	Delete	Traverse
Unsorted list (array)	1	n	n	n
Unsorted list (pointer)	1	n	n	n
Sorted list (array)	n	$\log n$	n	n
Sorted list (pointer)	n	n	n	n
Binary search tree	$\log n$	$\log n$	$\log n$	n
Balanced BST	$\log n$	$\log n$	$\log n$	n
Hash table (chaining)	λ	λ	λ	$m+n$

Example 20.3. If we want to print out all items of a table in sorted order, what is the best data structure? If we do not mind an occasional long wait and mostly do lookups, which is best? If we do a lot of insertions and deletions and few lookups, and are risk-averse, which is best?