Today: hashing but first a toy problem

Balls into bins:

n balls & n bins (or buckets)

Each ball is thrown into a random bin (independent of what happened for the other balls)

Let load of bin i = # of balls assigned to bin i.

How large is the max load?
= Max # of balls in a bin.

Max load is $O(\log n)$ with high probability $\left(\geq 1 - \frac{1}{\text{poly}(n)}\right)$

Can refine to $\Theta\left(\frac{\log n}{\log\log n}\right)$

Better idea:

Assigns balls sequentially into bins

For $i = 1 \to n$

- Choose 2 random bins say j & k
- Let $L(j)$ & $L(k)$ denote their current load
- If $L(j) < L(k)$, assign ball i to bin j

  else assign ball i to bin k

  (So ball i goes to the smaller of the 2 bins)

Then max load is $O(\log\log n)$ with high prob.

If choose d bins (instead of 2) then get $O\left(\frac{\log\log n}{\log d}\right)$

Hashing:

Running example:

Database of unacceptable Passwords

Queries check if a password is allowed or not. Need to check quickly

HUGE set U = universe of possible passwords

hash table H size n

$H = [0, 1, \ldots, n-1]$

map elements of U into bins $\{0, 1, \ldots, n-1\}$

Use hash function $h: U \to \{0, 1, \ldots, n-1\}$

Chain hashing, $H[i]$ is a linked list of elements stored there.

Start with empty list at each $H[i]$.

Let S be the set of unacceptable passwords

We store S in H.

To add $x \in U$ into S,

- compute $h(x)$
- add x onto linked list at $H[h(x)]$

To check: is $y \in S$?

- compute $h(y)$
- check linked list at $H[h(y)]$ to see if it contains y.

query time = load at bin $h(y)$.

$|S| = m$, $|H| = n$

We're putting m balls into n bins

If h is a random function so $h(x)$ is random over $\{0, 1, \ldots, n-1\}$

Max query time is max load.

When $m = n$ max load is $O(\log n)$.

To get max load $O(1)$ need $n = \Omega(m^2)$.

Better approach:

Use 2 hash functions

$$h_1 : U \to \{0, 1, \ldots, n-1\}$$
$$\& \; h_2 : U \to \{0, 1, \ldots, n-1\}$$

To add $x \in U$ into S,

- compute $h_1(x)$ & $h_2(x)$
- add x to least loaded of $h_1(x)$ & $h_2(x)$

To check: is $y \in S$?

- compute $h_1(y)$ & $h_2(y)$
- check list at $H[h_1(y)]$ & $H[h_2(y)]$

(Requires checking 2 lists but both are much smaller)

Now for random h_1 & h_2,
if $m = n$, max load is only $O(\log\log n)$.

Bloom filters:

Faster queries = $O(1)$ time
Less space
Simpler
But: false positives with small probability

Maintain set $S \subset U$.

Operations:

Insert(x): add x into S

Query(x): is $x \in S$?

if $x \in S$, we **always** output YES

if $x \notin S$, we usually output NO

but we have a false positive (output YES) with small probability

false positive rate.

H is a 0-1 array of size n. (no linked list!)
Start with H as all 0's.
hash function $h: U \to \{0, 1, \ldots, n-1\}$

To add x into S, set $H[h(x)] = 1$.

To check if y is in S,
if $H[h(y)] = 1$ then output YES
if $H[h(y)] = 0$ then output NO.

Problem:
if $y \notin S$ but we added z into S
where $h(y) = h(z)$
then we get a false positive for
the query: is $y \in S$?

Better approach:

k hash functions: $h_1, h_2, \ldots, h_k$

for each i, $h_i : U \rightarrow \{0, 1, \ldots, n-1\}$

Start: Set $H[i] = 0$ for all $i \in \{0, 1, \ldots, n-1\}$

So H is the all 0-vector.

To add x into S:

for $i = 1 \rightarrow k$, set $H[h_i(x)] = 1$

(so set k bits to 1)

To check if $y \in S$?

if for all i, $H[h_i(y)] = 1$

then output YES

else (if 1 or more $= 0$)

output NO.

---

if $y \in S$, then for query: is $y \in S$?

we always output YES

if $y \notin S$, then we might output YES if

for all i, there is $z \in S$ & j

where $h_i(y) = h_j(z)$.

What is the probability of a false positive?

Recall, hash table H of size $|H| = n$

& storing database S of size $m = |S|$

where $|H| = n > |S| = m$

Want to make n as small as possible

let $c = \frac{n}{m}$ so $c > 1$.

want c small.

For $y \notin S$ to get a false positive need that the k bits at $h_1(y), \dots, h_k(y)$ are set to 1.

For a specific bit b where $b \in \{0, \dots, n-1\}$ what's the probability $H[b] = 1$?

$$Pr(H[b]=1) = 1 - Pr(H[b]=0)$$

We are throwing mk balls randomly into n bins,

$$Pr(H[b]=0) = Pr(\text{all } mk \text{ balls miss bin } b)$$

$$= \left(1 - \frac{1}{n}\right)^{mk} \approx e^{-mk/n} = e^{-k/c}$$ since $c = n/m$

Recall $e^{-x} = 1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} \pm \dots \geq 1 - x$

& for small x, $e^{-x} \approx 1 - x$.

$$Pr(H[b]=1) = 1 - e^{-k/c}$$

For $y \notin S$,

$\Pr(\text{false positive for } y)$

$$= \Pr\left(\text{for all } i,\ H[h_i(y)] = 1\right)$$

$$\approx \left(1 - e^{-k/c}\right)^k = \underline{\text{false positive rate}}$$

What's the best choice for k?

k big $\Rightarrow$ putting too many 1's in H

k small $\Rightarrow$ checking too few bits of H.

Let $f = \left(1 - e^{-k/c}\right)^k$ = false positive rate.

Let's minimize f as a function of k.

Let $g = \ln f = k \ln\left(1 - e^{-k/c}\right)$

So $f = e^g$ & we'll minimize g to make it easier.

$$\frac{dg}{dk} = \ln\left(1 - e^{-k/c}\right) + \frac{k}{1 - e^{-k/c}} \times \frac{1}{c} \times e^{-k/c}$$

Set $k = c \ln 2$.

Then $\frac{dg}{dk} = -\ln 2 + \ln 2 = 0$ & can check this is a minimum.

Plugging in $k = c\ln 2$, the false positive rate is

$$f = \left(1 - e^{-k/c}\right)^k = \left(\frac{1}{2}\right)^{c\ln 2} = \left(\left(\frac{1}{2}\right)^{\ln 2}\right)^c \approx (.6185)^c$$

and

$$\Pr(H[b] = 0) \approx e^{-k/c} = \frac{1}{2}$$

So H is a random 0-1 string.

Examples: false positive rate:

$k=1$:	$c=10$:	.09516
	$c=100$:	.00995
$k=c\ln 2$:	$c=10$:	.0082
	$c=100$:	1.3×10^{-21}