Analyzing typical worst-case insert/lookup time (under SUHA) when \( n \) items hashed to \( n \) slots.

Typical max load when \( n \) balls thrown into \( n \) bins

Recall: we ended with \( \Pr[T_{[0]} \text{ gets } k \text{ balls}] \leq \frac{1}{k!} \)

\[ \Pr[T_{[0]} \text{ gets } \geq 10 \text{ balls}] \leq \Pr[10] + \Pr[11] + \ldots \]

"union bound"

\[ \leq \frac{1}{10!} + \frac{1}{11!} + \frac{1}{12!} + \ldots \]

\[ = \frac{1}{10!} \left(1 + \frac{1}{11!} + \frac{1}{12!} + \frac{1}{13!} + \ldots\right) \]

\[ \leq \frac{1}{10!} \left(1 + \frac{1}{11} + \frac{1}{11^2} + \frac{1}{11^3} + \ldots\right) \]

\[ = \frac{1}{10!} \left(\frac{1}{1 - \frac{1}{11}}\right) = \frac{1}{10!} \cdot \frac{11}{10} \]

\[ = \frac{11}{10} \cdot \frac{1}{10!} \]

"just slightly bigger than \( \frac{1}{10!} \)"

\[ \Pr[T_{[0]} \text{ gets } \geq k \text{ balls}] \leq \frac{k+1}{k} \cdot \frac{1}{k!} \leq 2 \cdot \frac{1}{k!} \]

"more like \( \approx 1 \cdot \frac{1}{k!} \), but oh well"

\[ \Pr[T_{[1]} \text{ gets } \geq k \text{ balls}] \leq \frac{2^{\frac{1}{k!}}}{k!} \]

\[ \Pr[T_{[2]}] \leq \ldots \]

\[ \Pr[T_{[0]} \geq k \text{ or } T_{[1]} \geq k \text{ or } \ldots \text{ or } T_{[n-1]} \geq k] \leq \]

\[ \Pr[T_{[0]} \geq k] + \Pr[T_{[1]} \geq k] + \ldots + \Pr[T_{[n-1]} \geq k] \]

\[ \leq \frac{2}{k!} + \frac{2}{k!} + \ldots + \frac{2}{k!} = \frac{2}{k!} \cdot n \]

\[ \Pr[\text{max load } \geq k] \leq \frac{k}{k!} \]
Say \( n = 1000 \). \[ \Pr \left[ \max \text{ load } > k \right] \leq \frac{2000}{k!} \]

\[ \Rightarrow 6 \leq \frac{2000}{720} \]

\[ \Rightarrow 7 \leq \frac{2000}{5040} \times 4 \]

\[ \Rightarrow 8 \leq \frac{2000}{40320} \times 0.5 \]

Say \( n = 10^6 \), want to say \( \Pr \left[ \max \text{ load } > k \right] \leq \frac{1}{500} \) (say)

What \( k \)? Need \( \frac{2 \times 10^6}{k!} \leq \frac{1}{500} \Rightarrow k! > 10^9 \Rightarrow k > 12 \)

"For 1M balls in 1M bins, \( \max \text{ load } \) \( \leq 1 \) except w.prob \( \leq \frac{1}{500} \)"

For general \( n \), need \( k! > 1000n \).

For \( n = 10^6 \), need \( k! > 1000 \times 10^6 \).

How big is \( k! \)?

\[ \log_2(k!) = \log \left( k \cdot (k-1) \cdot (k-2) \ldots \cdot 3 \cdot 2 \cdot 1 \right) \]

\[ = \log k + \log (k-1) + \log (k-2) + \ldots + \log \left( \frac{k}{2} \right) + \ldots + \log \left( \frac{k}{2} \right) \]

\[ = \frac{k}{2} \log \left( \frac{k}{2} \right) = \frac{k}{2} \left( \log k - 1 \right) \geq \frac{k}{2} \cdot \frac{\log k}{2} \]

\[ = \frac{1}{2} k \log k \geq 16 \log 10 \]

\[ k! \text{ is a } \Theta(k \log k) \text{-digit number}. \]

Fact: \( 22! \) has 22 digits.

\( 23! \approx 10^{23} \cdot 22! \) has \( \geq 23 \) digits

\( 24! \approx 10^{24} \cdot 23! \) has \( \geq 24 \) digits

\( k! \) has \( \geq k \) digits (ex: more like \( \Theta(k \log k) \) digits)

\( n \) has \( \sim \log_{10} n \) digits. So if \( k \geq \log_{10} n + 3 \), \( k! > 1000n \).

So "For \( n \) balls in \( n \) bins, \( \max \text{ load } \leq O(1/n) \) except w.prob \( \leq \frac{1}{500} \)." (ex: \( \leq O(\log n) \))
"The power of 2 choices"

Idea: Get hold of two ("independent") hash functions, $h_1, h_2$: $\{\text{strings}\} \to \{0, 1, \ldots\}$

- Insert(s): Look at $T[h_1(s)], T[h_2(s)]$, linked lists.
- Append s to whichever is shorter (break ties count.)
- Lookup(s): Look for s in both $T[h_1(s)], T[h_2(s)]$.

Can't really make things more than $2x$ worse.

Surprise: worst-case time goes way down, typically.

Balls & bins ver: To "throw" a ball, pick 2 bins at random, put ball in less-loaded bin.

Theorem: For $n = \alpha \cdot 2^m$ balls, with high probability, max load is $O(\log \log n)$. Way better than $O(\log n)$.

Proof is a little elaborate, so I'll just give...

Idea of why: After throwing $n$ balls, let

$\alpha_2$ = fraction of bins with $\geq 2$ balls

Claim: $\alpha_2 \leq \frac{1}{4}$. Because if $\frac{3}{4}$ bins have $\geq 2$ balls, that's $> n$ balls, $n \geq$.

Say we've thrown some of the balls, $\alpha_2 \leq \frac{1}{2}$ now. It's even $\leq \frac{1}{2}$ at end, so things only better now.

New ball thrown: What is prob. new ball ends up at "height" $\geq 3$?

- Both bins it looks at must have $\geq 2$ balls
- prob $\leq \alpha_2^2 \leq \frac{1}{4}$.
- "intuitively", $\alpha_3$: fraction of bins with $\geq 3$ balls $\leq \frac{1}{4}$

Each ball thrown has $\frac{1}{4}$ chance of being in a bin of height $\geq 3$.
New ball thrown: what is prob. it ends up at height $\geq 4$?

Both bins it looks at must have $\geq 3$ balls.

\[ \Rightarrow \text{prob. } \leq \alpha_3 \leq \left(\frac{4}{16}\right)^2 \leq \frac{1}{16}. \]

\[ \alpha_4 = \text{frac. 6 bins with } > 4 \text{ balls } \leq \frac{1}{16} \]

\[ \alpha_5 \leq \left(\frac{1}{16}\right)^2 = \frac{1}{256} = 2^{-8} \]

\[ \alpha_6 \leq \alpha_5 \leq 2^{-6} \]

\[ \alpha_7 \leq \alpha_6 \leq 2^{-3} \]

\[ \alpha_{k+2} \leq 2^{-2^k} \]

Intuitively if $\alpha_{k+2} < \frac{1}{n}$, then probably no bin have $> k+2$ balls

\[ \Rightarrow 2^{-2^k} < \frac{1}{n} \Rightarrow 2^{2^k} > n \Rightarrow k > \log \log n. \]

So "probably" max load $\leq \log \log n + 2$.

Ex: 3 choices still "only" gives $\Theta(\log \log n)$. [So just stick with $2$!]

Bloom Filters: Say: *m truly enormous (billion, trillion...)*

* strings you're storing also large $\sim L$ 64 bits
  
  (e.g. tweets: $\approx 250 \times 8 = 2000$ bits).

Can't possibly use less than $Lm$ bits of space, right? [Right?? And hashing within $m$ uses $\sim L \log n$]

Use $\approx 8.66m$ bits total! $230x$ savings $\sqrt{230x}$ fewer servers!!!

What's the catch??

Lookup errors: Lookup(s) wrongly says "yes" (even though was never stored)

\[ \approx 6.66 \approx 1.44 \times 6 \times 2^{-1.6} \]

1.6% = $2^{-k}$, for $k = 6$. Can choose other $k$.

Trade off: 1.44m extra bits, to halve lookup error prob.
How Bloom Filters work

Pick small \( k \), e.g., \( k = 6 \).

Set \( N = k \cdot m \) \( \left[ \text{rounded off to integer} \right] \), # of bits used.

Allocate array \( T[0...N-1] \) of bits, init. all 0's.

Choose \( k \) "independent" hash functions \( h_1,...,h_k: \{ \text{strings} \} \to \{0,1,...,N-1\} \).

Insert \((s)\): Set \( T[h_1(s)] = 1, \ldots , T[h_k(s)] = 1 \). \( \square \) maybe some were already 1 \( \square \)

Lookup \((s)\): Return \( \text{AND of } T[h_1(s)], \ldots , T[h_k(s)] \).

Space: \( \approx 1.44k \) bits per item \( \smile \)

Time: \( O(1) \) \( \left[ O(k) \right] \) operations \( \smile \)

Delete \((s)\): not possible, even slowly \( \ominus \)

"False positive problem": Lookup\((s)\) may return True even if \( s \) was never inserted.

[No "false negs" \( \smile \).]

Analysis: Prob. [false positive lookup] \( \leq ? \)

Ideas

"Again, too fiddly to do rigorously here, so we'll cheat a little!"

Q1: After inserting \( m \) items, what frac. of 6 bits in \( T[] \) do we expect are 0? Still

A: Under SUHA, it's like throwing \( km \) balls into \( N = \frac{km}{ln^2} \) bins \( \left( \lambda = \frac{km}{ln^2} \right) \), asking about frac. of empty bins

\[ \Pr[\text{bin 1 empty}] = \left(1 - \frac{1}{N}\right)^{km} \]
Most useful approx. ever:

\[ 1 + x \approx e^x \text{ if } x \text{ is tiny} \]

\[ \approx 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \ldots \]

If \( |x| = 10^{-6} \)

\[ \frac{10^{-12}}{2} = \text{negligible} \]

**Case 1:** \( x = -\frac{1}{N} \) is tiny,

\[ 1 + \left(-\frac{1}{N}\right) \approx e^{-\frac{1}{N}} \]

\[ \left(1 - \frac{1}{N}\right)^k \approx \left(e^{-\frac{1}{N}}\right)^k = e^{-\frac{k}{N}} \]

\( N = \frac{k}{\ln^2 m} \)

\[ e^{-\ln^2 \frac{1}{\alpha}} = e^{\alpha^2} = \frac{1}{\alpha}. \]

**Case 2:** \( \Pr[ \text{bin 1 empty} ] \approx \frac{1}{N} \)

[Same for every part of bin,]

We expect after \( m \) inserts, \( T[\cdot] \) is about 50-50 0's and 1's

Now say we do lookup(s), where \( s \) has never been inserted.

By SUHA, \( h_i(s), \ldots, h_k(s) \) act like \( k \) independent

\[ \Pr[\text{AND of } T[h_1(s)], \ldots, T[h_k(s)]] = \text{True} ] \]

\[ = \Pr[\text{false pos.}] = \left(\frac{1}{2}\right)^k. \]

Summary:

- can store \( m \) items of any size
- using \( \approx 1.44k \) bits per item,
- with \( O(1) \)-time lookups/inserts,
- and Insert false-positive prob \( \approx 2^{-k} \).