15-853: Algorithms in the Real World

Announcements:

• HW2 due this Friday noon.
  • Small correction made in the BWT question.
• Naama’s office hour cancelled. Francisco holding additional office hours instead.

• Mid-semester grades released.

• Graph compression guest lecture on Oct 29
• There will be Cryptography lectures on Oct 31 and a following lecture
15-853: Algorithms in the Real World

Announcements:

• Start thinking about the project and the team
  • Tentatively by Friday Nov 8 team and project should be finalized
  • Use the class email list with subject beginning “project-team-finding” to ping your classmates to form teams
15-853: Algorithms in the Real World

Hashing:
- Concentration bounds
- Load balancing: balls and bins
- Hash functions
Recap: Load balancing

N balls and N bins
Randomly put balls into bins

**Theorem:** The max-loaded bin has $O\left(\frac{\log N}{\log \log N}\right)$ balls with probability at least $1 - \frac{1}{N}$.

**Proof.** High level steps:

1. We will first look at probability of any particular bin receiving more than $O\left(\frac{\log N}{\log \log N}\right)$ balls.

2. Then we will look at the probability of there being a (i.e., at least one) bin with more than these many balls.
Load balancing

**Theorem:** The max-loaded bin has $O\left(\frac{\log N}{\log \log N}\right)$ balls with probability at least $1 - 1/N$.

**Proof 1.**

$P(\text{bin } i \text{ has at least } k \text{ balls}) \leq \frac{1}{k!}$

Using Sterling’s approximation and choosing

$k = O\left(\frac{\log N}{\log \log N}\right)$ gives the desired result.

**Proof 2.**

Can also prove this result using the Chernoff bound on Binomial R.V.

Q: What is the Binomial R.V. here?
Load balancing

Another useful and interesting result.
It turns out that the **bound is tight**!

**Theorem.** With high probability the max load is \( \Omega \left( \frac{\log n}{\log \log n} \right) \)

Uniformly randomly placing balls into bins does not balance the load after all!
Load balancing: power-of-2-choice

When a ball comes in, pick two bins and place the ball in the bin with smaller number of balls.

Turns out with just checking two bins maximum number of balls drops to \( O(\log \log n) \)!

=> called “power-of-2-choices”

Intuition: Ideas?

Even though max loaded bins has \( O\left(\frac{\log N}{\log \log N}\right) \) balls, most bins have far fewer balls.
Load balancing: power-of-2-choice

Proof (Intuition):

For a ball $b$, let

$\text{height}(b) = \text{number of balls in its bin after placing } b$

Probability of an incoming ball getting height 3 is at most ?

- Q: What needs to happen for this?
- Q: Fraction of bins that can have $\geq 2$ balls?
  - at most $\frac{1}{2}$ (since there are only $N$ balls)

$\frac{1}{2} \times \frac{1}{2} = \frac{1}{4}$

So expected number of bins with 3 balls is at most $= \frac{N}{4}$
Proof (Intuition) cont.: 
(For a ball b, let height(b) = number of balls in its bin after placing b)

Probability of an incoming ball getting height 4 is at most \( \frac{1}{4} \times \frac{1}{4} = \frac{1}{16} = \frac{1}{2^{4-2}} \)

Probability of an incoming ball getting height h is at most \( \frac{1}{2^{h-2}} \)

Choosing \( h = O(\log \log N) + 2 \) gives probability \( \frac{1}{N} \).
Load balancing: power-of-d-choice

When a ball comes in, **pick d bins** and place the ball in the bin with smallest number of balls.

**Theorem:**
For any \( d \geq 2 \) the \( d \)-choice process gives a maximum load of

\[
\frac{\log \frac{\log N}{\log d}}{\log d} \pm O(1)
\]

with probability at least \( 1 - O(1/N) \)

**Observations:**
Just looking at two bins gives huge improvement. Diminishing returns for looking at more than 2 bins.
Hashing

Central concept in CS

Numerous applications:
- Dictionary data structures, load balancing, placement, ...

Setting:
A large set of (possible) values: called universe $U$
Interested in only a subset of this: $S$
Let $|S| = N$ (typically $N < |U|$)

Roughly, hashing is a way to map elements of $U$ onto smaller number of values such that with high probability there are not too many collisions among elements of $S$. 
Hashing

Concrete running application for this module: dictionary.

Setting:
- A large universe of keys (e.g., set of all strings of certain length): denoted by $\mathbf{U}$
- The actual dictionary $\mathbf{S}$ (subset of $\mathbf{U}$)

Operations:
- $\text{add}(x)$: add a key $x$
- $\text{query}(q)$: is key $q$ there?
- $\text{delete}(x)$: remove the key $x$
"....with high probability there are not too many collisions among elements of S"

On what is this probability calculated over?

Two approaches:
1. Input is random
2. Input is arbitrary, but the hash function is random

Input being random is typically not valid for many applications. So we will use 2.

- We will assume a family of hash functions $H$.
- When it is time to hash $S$, we choose a random function $h \in H$
Let \([M] = \{0, 1, \ldots, M-1\}\)

We design a hash function \(h: U \rightarrow [M]\)

1. Small probability of distinct keys colliding:
   1. If \(x \neq y \in S\), \(P[h(x) = h(y)]\) is “small”
2. Small range, i.e., small \(M\) so that the hash table is small
3. Small number of bits to store \(h\)
4. \(h\) is easy to compute
Ideal Hash Function

Perfectly random hash function:
For each $x \in S$, $h(x) = a$ uniformly random location in $[M]$

Properties:
- Low collision probability: $P[h(x) = h(y)] = 1/M$ for any $x \neq y$
- Even conditioned on hashed values for any other subset $A$ of $S$, for any element $x \in S$, $h(x)$ is still uniformly random over $[M]$

Q: Problem with this ideal approach?
1. Too large to store this hash function: $\log M$ bits needed for each element in $S$ (since it can hash to any of the $M$ locations)
2. Also computing $h$ is going to be a table lookup
Universal Hash functions

Captures the basic property of non-collision. Due to Carter and Wegman (1979)

**Definition:** A family $H$ of hash functions mapping $U$ to $[M]$ is universal if for any $x \neq y \in U$,

$$P[h(x) = h(y)] \leq 1/M$$

**Note:** Must hold for every pair of distinct $x$ and $y \in U$. 
Universal Hash functions

A simple construction of universal hashing:

Assume $|U| = 2^u$ and $|M| = 2^m$
Let $A$ be a $m \times u$ matrix with random binary entries.
For any $x \in U$, view it as a $u$-bit binary vector, and define

$$h(x) := Ax$$

where the arithmetic is modulo 2.

Q: How many hash functions in this family?

$2^{um}$
Universal Hash functions

A simple construction of universal hashing:
Let A be a m x u matrix with uniformly random binary entries.

\[ h(x) = Ax \]

where the arithmetic is modulo 2.

**Theorem.** The family of hash functions defined above is universal.

**Proof.** Ideas?

\[ h(x) = h(y) \iff Ax = Ay \]

for \( x \neq y \)

\[ A(x - y) = 0 \]

\[ \Rightarrow A2 = 0 \text{ for } z \neq 0 \]
Universal Hash functions

Want to show $P(Az=0) \leq \frac{1}{M}$ for any $z \neq 0$

Let $z_i* \neq 0$ (i.e. since $z \neq 0$)

$Az=0$ $\Rightarrow$ $\sum A_jz_j = 0$

$A_{i*} = - \sum_{j \neq i*} A_jz_j$

$m$ length random binary vector

Fixed vector

Prob of above $= \left(\frac{1}{2}\right)^m = \frac{1}{M}$
Application: Hash table

One of the main applications of hash functions is in hash tables (for dictionary data structures)

Handling collisions:

Closed addressing

Each location maintains some other data structure

One approach: “separate chaining”

Each location in the table stores a linked list with all the elements mapped to that location.

Look up time = length of the linked list

To understand lookup time, we need to study the number of many collisions.
Application: Hash table

Let us study the number of many collisions:
Let $C(x)$ be the number of other elements mapped to the value where $x$ is mapped to.

Q: What is $E[C(x)]$?

$E[C(x)] = (N-1)/M$

Hence if we use $M = N = |S|$, lookups take constant time in expectation.

Item deletion is also easy.

Let $C =$ total number of collisions

Q: What is $E[C]$?

$\binom{N}{2}^{1/M}$
Application: Hash table

Can we design a collision free hash table?

Suppose we choose $M \geq N^2$

Q: $P[\text{there exists a collision}] = ?$

$\frac{1}{2}$

$\Rightarrow$ Can easily find a collision free hash table!

$\Rightarrow$ Constant lookup time for all elements! (worst-case guarantee)

But this is large a space requirement.
(Space measured in terms of number of keys)

Can we do better? $O(N)$? (while providing worst-case guarantee?)