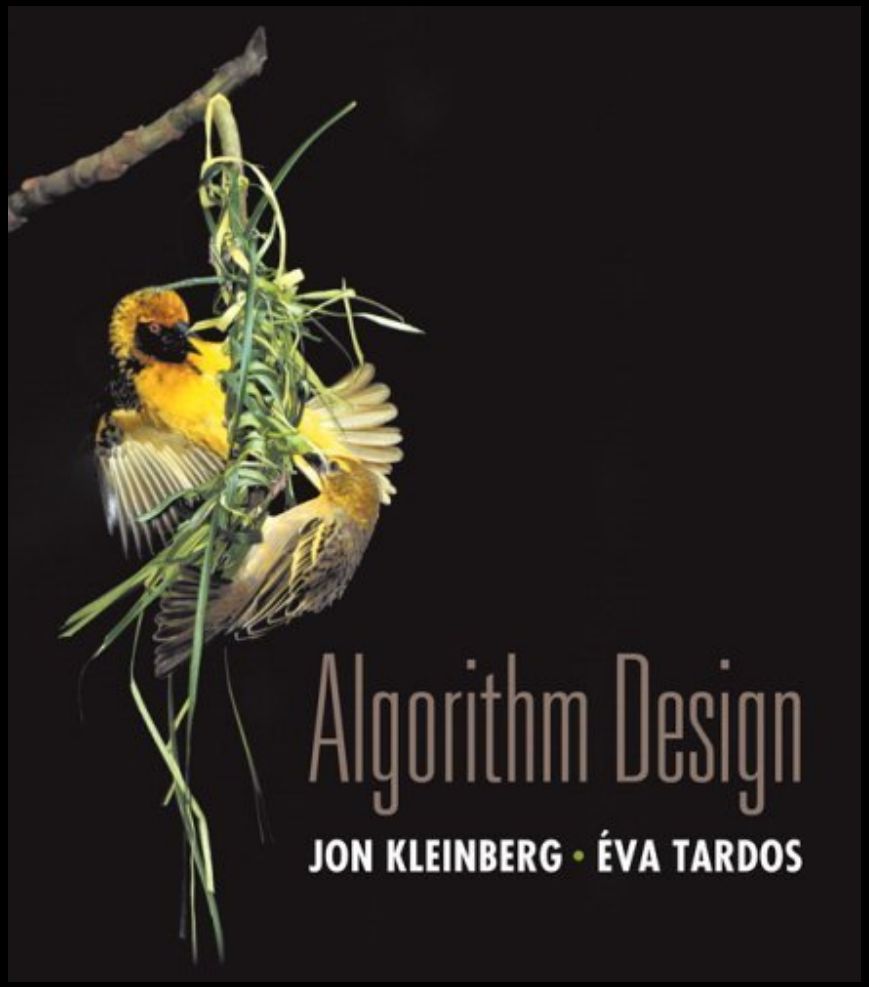



# Chapter 13

## Randomized Algorithms

## Part 2

Slides by Kevin Wayne, modified by Rasmus Pagh

# 13.10 Load Balancing

# Chernoff Bounds (13.6)

Q: If we hash m items into n buckets, how many items will be in the largest bucket?

Theorem. Suppose $X_1, \ldots, X_n$ are independent 0-1 random variables. Let $X = X_1 + \ldots + X_n$. Then for any $\mu \geq E[X]$ and for any $\delta > 0$, we have

Upper tail bound: $$\Pr[X > (1+\delta)\mu] < \left[\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right]^{\mu}$$

"sum of independent 0-1 random variables is tightly centered on the mean"

Lower tail bound: $$\Pr[X < (1-\delta)\mu] < e^{-\delta^2 \mu/2}$$

# Load Balancing: Average load 1

Theorem. For m=n items, with probability ≥ 1 - 1/n no bucket receives more than $C \log n / \log \log n$ items, for some constant $C$.

Fact: this bound is asymptotically tight: with high probability, some bucket receives $\Theta(\log n / \log \log n)$

Analysis.

- Let $X_i$ = number of items hashed to bucket i.
- Let $Y_{ij}$ = 1 if item j hashed to bucket i, and 0 otherwise.
- We have $E[Y_{ij}] = 1/n$
- Thus, $X_i = \sum_j Y_{ij}$, and $\mu = E[X_i] = 1$.
- Apply Chernoff to bound probability that $X > C \log n / \log \log n$.

# Load Balancing: Larger buckets

Theorem. If there are m = 16n ln n items (avg. $\mu$ = 16 ln n items/bucket). Then with high probability every bucket will have between half and twice the average load.

Pf.

- Let $X_i$ , $Y_{ij}$ be as before.
- Applying Chernoff upper (lower) tail bound with $\delta$ = 1 ($\delta$ = 0.5) yields:

$$\Pr[X_i > 2\mu] < \left(\frac{e}{4}\right)^{16n\ln n} < \left(\frac{1}{e}\right)^{\ln n} = \frac{1}{n^2} \qquad \Pr[X_i < \tfrac{1}{2}\mu] < e^{-\frac{1}{2}\left(\frac{1}{2}\right)^2(16n\ln n)} = \frac{1}{n^2}$$

- Union bound $\Rightarrow$ every bucket has load between half and twice the average with probability $\geq 1 - 2/n$. ▪

# 13.6 Universal Hashing

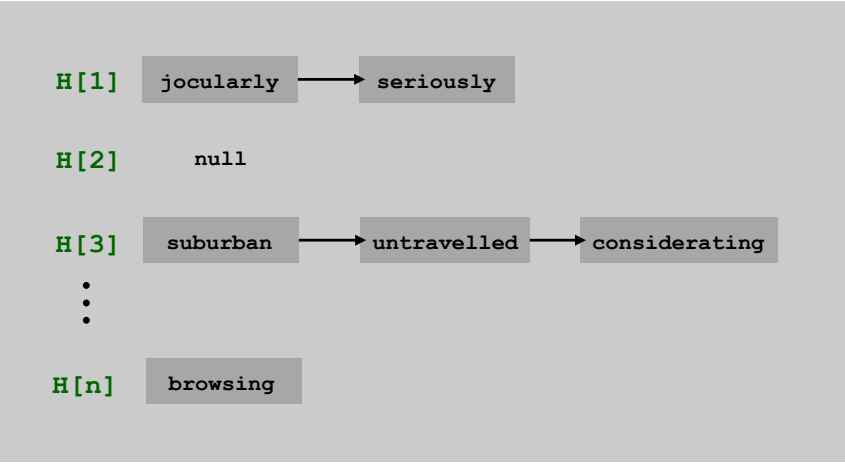
# Hashing

Hash function. $h : U \rightarrow \{ 0, 1, \ldots, n-1 \}$.

Basic hashing. Create an array H of size n. When processing element u, access array element H[h(u)].

Collision. When $h(u) = h(v)$ but $u \neq v$.

- A collision is expected after $\Theta(\sqrt{n})$ insertions ("birthday paradox").
- Separate chaining: H[i] stores linked list of elements u with h(u) = i.



# Ad Hoc Hash Function

Ad hoc hash function.

```
int h(String s, int n) {
   int hash = 0;
   for (int i = 0; i < s.length(); i++)
      hash = (31 * hash) + s[i];
   return hash % n;
}
```

hash function ala Java string library

Deterministic hashing. If $|U| \geq n^2$, then for any fixed hash function h, there is a subset $S \subseteq U$ of n elements that all hash to same slot. Thus, $\Theta(n)$ time per search in worst-case.

Q. But isn't ad hoc hash function good enough in practice?

# Algorithmic Complexity Attacks

When can't we live with ad hoc hash function?

- Obvious situations: aircraft control, nuclear reactors.
- Surprising situations: denial-of-service attacks.

malicious adversary learns your ad hoc hash function (e.g., by reading Java API) and causes a big pile-up in a single slot that grinds performance to a halt

Real world exploits. [Crosby-Wallach 2003]

- Bro server: send carefully chosen packets to DOS the server, using less bandwidth than a dial-up modem
- Perl 5.8.0: insert carefully chosen strings into associative array.
- Linux 2.4.20 kernel: save files with carefully chosen names.

# Hashing Performance

Idealistic hash function. Maps m elements uniformly at random to n hash slots.

- Running time depends on length of chains.
- Average length of chain = $\alpha$ = m / n.
- Choose $n \approx m \Rightarrow$ on average O(1) per insert, lookup, or delete.

Challenge. Achieve idealized randomized guarantees, but with a hash function that can be described and evaluated efficiently.

Approach. Use randomization in the choice of h.

adversary knows the randomized algorithm you're using,
but doesn't know random choices that the algorithm makes

# Universal Hashing

Universal class of hash functions. [Carter-Wegman, late 1970s]

- For any pair of elements u, v ∈ U, $\Pr_{h \in H}[\, h(u) = h(v) \,] \le 1/n$
- Can select random h efficiently.
- Can compute h(u) efficiently.

chosen uniformly at random

Ex. U = { a, b, c, d, e, f }, n = 2.

| | a | b | c | d | e | f |
|---|---|---|---|---|---|---|
| $h_1(x)$ | 0 | 1 | 0 | 1 | 0 | 1 |
| $h_2(x)$ | 0 | 0 | 0 | 1 | 1 | 1 |

$H = \{h_1, h_2\}$
$\Pr_{h \in H}[h(a) = h(b)] = 1/2$
$\Pr_{h \in H}[h(a) = h(c)] = 1$
$\Pr_{h \in H}[h(a) = h(d)] = 0$
. . .

not universal

| | a | b | c | d | e | f |
|---|---|---|---|---|---|---|
| $h_1(x)$ | 0 | 1 | 0 | 1 | 0 | 1 |
| $h_2(x)$ | 0 | 0 | 0 | 1 | 1 | 1 |
| $h_3(x)$ | 0 | 0 | 1 | 0 | 1 | 1 |
| $h_4(x)$ | 1 | 0 | 0 | 1 | 1 | 0 |

$H = \{h_1, h_2, h_3, h_4\}$
$\Pr_{h \in H}[h(a) = h(b)] = 1/2$
$\Pr_{h \in H}[h(a) = h(c)] = 1/2$
$\Pr_{h \in H}[h(a) = h(d)] = 1/2$
$\Pr_{h \in H}[h(a) = h(e)] = 1/2$
$\Pr_{h \in H}[h(a) = h(f)] = 0$
. . .

universal

# Universal Hashing

Universal hashing property. Let $h \in H$ be chosen at random. For any subset $S \subseteq U$ of size at most n, the expected number of items in S that collide with u is at most 1.

Pf. For any element $s \in S$, define indicator random variable $X_s = 1$ if $h(s) = h(u)$ and 0 otherwise. Let X be a random variable counting the total number of collisions with u.

$$E_{h \in H}[X] = E[\sum_{s \in S} X_s] = \sum_{s \in S} E[X_s] = \sum_{s \in S} \Pr[X_s = 1] \leq \sum_{s \in S} \frac{1}{n} = |S| \frac{1}{n} \leq 1$$

linearity of expectation

$X_s$ is a 0-1 random variable

universal (assumes $u \notin S$)

# Designing a Universal Family of Hash Functions

Modulus. Choose a prime number $p \approx n$.

[Chebyshev 1850]: There exists a prime between $n$ and $2n$. ← we need prime to be sure of randomness here

Integer encoding. Identify each element $u \in U$ with a base-p integer of r digits: $x = (x_1, x_2, \ldots, x_r)$.

Hash function. Let $A$ = set of all r-digit, base-p integers. For each $a = (a_1, a_2, \ldots, a_r)$ where $0 \le a_i < p$, define

$$h_a(x) = \left( \sum_{i=1}^{r} a_i x_i \right) \bmod p$$

Theorem. $H = \{ h_a : a \in A \}$ is a universal class of hash functions.

Useful fact. Let p be prime, and let $z \neq 0 \bmod p$. Then $\alpha z = m \bmod p$ has at most one solution $0 \le \alpha < p$.

# Another universal family

Tabulation hashing:

- Input is viewed as a string $x_1x_2...x_n$ (e.g. 1 byte/character).
- Have n arrays $T_1,...,T_n$ each with 256 random numbers from $\{1,...,r\}$.
- $h(x_1x_2...x_n)=(T_1[x_1]+...+T_n[x_n]) \bmod r$.

Theorem: Tabulation hashing is universal.

Patrascu and Thorup (STOC '11, JACM '12) showed many good properties of tabulation hashing: Works with cuckoo hashing, linear probing, Chernoff bounds,…