13. RANDOMIZED ALGORITHMS

- content resolution
- global min cut
- linearity of expectation
- max 3-satisfiability
- universal hashing
- Chernoff bounds
- load balancing
Randomization

Algorithmic design patterns.

- Greedy.
- Divide-and-conquer.
- Dynamic programming.
- Network flow.
- Randomization.

Randomization. Allow fair coin flip in unit time.

Why randomize? Can lead to simplest, fastest, or only known algorithm for a particular problem.

Ex. Symmetry breaking protocols, graph algorithms, quicksort, hashing, load balancing, Monte Carlo integration, cryptography.
13. Randomized Algorithms

- content resolution
- global min cut
- linearity of expectation
- max 3-satisfiability
- universal hashing
- Chernoff bounds
- load balancing
Contention resolution in a distributed system

Contention resolution. Given $n$ processes $P_1, \ldots, P_n$, each competing for access to a shared database. If two or more processes access the database simultaneously, all processes are locked out. Devise protocol to ensure all processes get through on a regular basis.

Restriction. Processes can't communicate.

Challenge. Need symmetry-breaking paradigm.
Contention resolution: randomized protocol

Protocol. Each process requests access to the database at time $t$ with probability $p = 1/n$.

Claim. Let $S[i, t] = \text{event that process } i \text{ succeeds in accessing the database at time } t$. Then $1 / (e \cdot n) \leq \Pr [S(i, t)] \leq 1/(2n)$.

Pf. By independence, $\Pr [S(i, t)] = p (1 - p)^{n-1}$.

• Setting $p = 1/n$, we have $\Pr [S(i, t)] = 1/n (1 - 1/n)^{n-1}$.

Useful facts from calculus. As $n$ increases from 2, the function:

• $(1 - 1/n)^{n-1}$ converges monotonically from $1/4$ up to $1/e$.
• $(1 - 1/n)^{n-1}$ converges monotonically from $1/2$ down to $1/e$. 

Contention Resolution: randomized protocol

Claim. The probability that process $i$ fails to access the database in $n$ rounds is at most $1/e$. After $e \cdot n (c \ln n)$ rounds, the probability $\leq n^{-c}$.

Pf. Let $F[i, t] =$ event that process $i$ fails to access database in rounds 1 through $t$. By independence and previous claim, we have

$$\Pr[F[i, t]] \leq (1 - \frac{1}{en})^t.$$ 

- Choose $t = [e \cdot n]$:
  $$\Pr[F(i, t)] \leq \left(1 - \frac{1}{en}\right)^{en} \leq \left(1 - \frac{1}{en}\right)^{en} \leq \frac{1}{e}$$

- Choose $t = [e \cdot n \cdot c \ln n]$:
  $$\Pr[F(i, t)] \leq \left(\frac{1}{e}\right)^{c \ln n} = n^{-c}$$
Contention Resolution: randomized protocol

Claim. The probability that all processes succeed within $2e \cdot n \ln n$ rounds is $\geq 1 - 1/n$.

Pf. Let $F[t] = \text{event that at least one of the } n \text{ processes fails to access database in any of the rounds 1 through } t$.

\[
\Pr[ F[t] ] = \Pr\left[ \bigcup_{i=1}^{n} F[i, t] \right] \leq \sum_{i=1}^{n} \Pr[ F[i, t] ] \leq n \left(1 - \frac{1}{\ln n}\right)^t
\]

\[\text{union bound} \quad \text{previous slide}\]

• Choosing $t = 2 \lceil en \rceil \lceil c \ln n \rceil$ yields $\Pr[F[t]] \leq n \cdot n^2 = 1/n$. ■

Union bound. Given events $E_1, \ldots, E_n$, $\Pr\left[ \bigcup_{i=1}^{n} E_i \right] \leq \sum_{i=1}^{n} \Pr[E_i]$
13. RANDOMIZED ALGORITHMS

- content resolution
- global min cut
- linearity of expectation
- max 3-satisfiability
- universal hashing
- Chernoff bounds
- load balancing

Section 13.2
Global minimum cut

Global min cut. Given a connected, undirected graph $G = (V, E)$, find a cut $(A, B)$ of minimum cardinality.

Applications. Partitioning items in a database, identify clusters of related documents, network reliability, network design, circuit design, TSP solvers.

Network flow solution.
- Replace every edge $(u, v)$ with two antiparallel edges $(u, v)$ and $(v, u)$.
- Pick some vertex $s$ and compute min $s$-$v$ cut separating $s$ from each other vertex $v \in V$.

False intuition. Global min-cut is harder than min $s$-$t$ cut.
Contraction algorithm

**Contraction algorithm.** [Karger 1995]

- Pick an edge \( e = (u, v) \) uniformly at random.
- **Contract** edge \( e \).
  - replace \( u \) and \( v \) by single new super-node \( w \)
  - preserve edges, updating endpoints of \( u \) and \( v \) to \( w \)
  - keep parallel edges, but delete self-loops
- Repeat until graph has just two nodes \( v_1 \) and \( v_1 \).
- Return the cut (all nodes that were contracted to form \( v_1 \)).
**Contraction algorithm.** [Karger 1995]

- Pick an edge \( e = (u, v) \) uniformly at random.
- **Contract** edge \( e \).
  - replace \( u \) and \( v \) by single new super-node \( w \)
  - preserve edges, updating endpoints of \( u \) and \( v \) to \( w \)
  - keep parallel edges, but delete self-loops
- Repeat until graph has just two nodes \( v_1 \) and \( v_1 \).
- Return the cut (all nodes that were contracted to form \( v_1 \)).

Reference: Thore Husfeldt
Contraction algorithm

Claim. The contraction algorithm returns a min cut with prob $\geq 2 / n^2$.

Pf. Consider a global min-cut $(A^*, B^*)$ of $G$.

- Let $F^*$ be edges with one endpoint in $A^*$ and the other in $B^*$.
- Let $k = |F^*| = \text{size of min cut}$.
- In first step, algorithm contracts an edge in $F^*$ probability $k / |E|$.
- Every node has degree $\geq k$ since otherwise $(A^*, B^*)$ would not be a min-cut $\Rightarrow |E| \geq \frac{1}{2} k n$.
- Thus, algorithm contracts an edge in $F^*$ with probability $\leq 2 / n$. 
**Contraction algorithm**

**Claim.** The contraction algorithm returns a min cut with prob $\geq \frac{2}{n^2}$.

**Pf.** Consider a global min-cut $(A^*, B^*)$ of $G$.
- Let $F^*$ be edges with one endpoint in $A^*$ and the other in $B^*$.
- Let $k = |F^*| = $ size of min cut.
- Let $G'$ be graph after $j$ iterations. There are $n' = n - j$ supernodes.
- Suppose no edge in $F^*$ has been contracted. The min-cut in $G'$ is still $k$.
- Since value of min-cut is $k$, $|E'| \geq \frac{1}{2} kn'$.
- Thus, algorithm contracts an edge in $F^*$ with probability $\leq \frac{2}{n'}$.
- Let $E_j = $ event that an edge in $F^*$ is not contracted in iteration $j$.

\[
\Pr[E_1 \cap E_2 \cdots \cap E_{n-2}] = \Pr[E_1] \times \Pr[E_2 \mid E_1] \times \cdots \times \Pr[E_{n-2} \mid E_1 \cap E_2 \cdots \cap E_{n-3}] \\
\geq \left(1 - \frac{2}{n}\right) \left(1 - \frac{2}{n-1}\right) \cdots \left(1 - \frac{2}{4}\right) \left(1 - \frac{2}{3}\right) \\
= \left(\frac{n-2}{n}\right) \left(\frac{n-3}{n-1}\right) \cdots \left(\frac{2}{4}\right) \left(\frac{1}{3}\right) \\
= \frac{2}{n(n-1)} \\
\geq \frac{2}{n^2}
\]
**Contraction algorithm**

**Amplification.** To amplify the probability of success, run the contraction algorithm many times.

**Claim.** If we repeat the contraction algorithm \(n^2 \ln n\) times, then the probability of failing to find the global min-cut is \(\leq 1 / n^2\).

**Pf.** By independence, the probability of failure is at most

\[
\left(1 - \frac{2}{n^2}\right)^{n^2 \ln n} = \left\lfloor \left(1 - \frac{2}{n^2}\right)^{\frac{1}{2} n^2}\right\rfloor^{2 \ln n} \leq \left(e^{-1}\right)^{2 \ln n} = \frac{1}{n^2}
\]

\((1 - 1/x)^x \leq 1/e\)
Contraction algorithm: example execution

Reference: Thore Husfeldt
Global min cut: context

**Remark.** Overall running time is slow since we perform $\Theta(n^2 \log n)$ iterations and each takes $\Omega(m)$ time.

**Improvement.** [Karger-Stein 1996] $O(n^2 \log^3 n)$.

- Early iterations are less risky than later ones: probability of contracting an edge in min cut hits 50% when $n / \sqrt{2}$ nodes remain.
- Run contraction algorithm until $n / \sqrt{2}$ nodes remain.
- Run contraction algorithm *twice* on resulting graph and return *best* of two cuts.

**Extensions.** Naturally generalizes to handle positive weights.

**Best known.** [Karger 2000] $O(m \log^3 n)$.

...faster than best known max flow algorithm or deterministic global min cut algorithm
13. RANDOMIZED ALGORITHMS

- content resolution
- global min cut
- linearity of expectation
- max 3-satisfiability
- universal hashing
- Chernoff bounds
- load balancing

Section 13.3
**Expectation**

**Expectation.** Given a discrete random variables $X$, its expectation $E[X]$ is defined by:

$$E[X] = \sum_{j=0}^{\infty} j \Pr[X = j]$$

**Waiting for a first success.** Coin is heads with probability $p$ and tails with probability $1-p$. How many independent flips $X$ until first heads?

$$E[X] = \sum_{j=0}^{\infty} j \cdot \Pr[X = j] = \sum_{j=0}^{\infty} j (1-p)^{j-1} p = \frac{p}{1-p} \sum_{j=0}^{\infty} j (1-p)^j = \frac{p}{1-p} \cdot \frac{1-p}{p^2} = \frac{1}{p}$$

\[ j-1 \text{ tails} \quad 1 \text{ head} \]
Expectation: two properties

**Useful property.** If $X$ is a 0/1 random variable, $E[X] = \Pr[X = 1]$.

\[ E[X] = \sum_{j=0}^{\infty} j \cdot \Pr[X = j] = \sum_{j=0}^{1} j \cdot \Pr[X = j] = \Pr[X = 1] \]

**Linearity of expectation.** Given two random variables $X$ and $Y$ defined over the same probability space, $E[X + Y] = E[X] + E[Y]$.

**Benefit.** Decouples a complex calculation into simpler pieces.
Guessing cards

**Game.** Shuffle a deck of $n$ cards; turn them over one at a time; try to guess each card.

**Memoryless guessing.** No psychic abilities; can't even remember what's been turned over already. Guess a card from full deck uniformly at random.

**Claim.** The expected number of correct guesses is 1.

**Pf.** [surprisingly effortless using linearity of expectation]
- Let $X_i = 1$ if $i^{th}$ prediction is correct and 0 otherwise.
- Let $X = \text{number of correct guesses} = X_1 + \ldots + X_n$.
- $E[X_i] = \Pr[X_i = 1] = 1/n$.
- $E[X] = E[X_1] + \ldots + E[X_n] = 1/n + \ldots + 1/n = 1$. □

↑ linearity of expectation
Guessing cards

**Game.** Shuffle a deck of $n$ cards; turn them over one at a time; try to guess each card.

**Guessing with memory.** Guess a card uniformly at random from cards not yet seen.

**Claim.** The expected number of correct guesses is $\Theta(\log n)$.

**Pf.**
- Let $X_i = 1$ if $i^{th}$ prediction is correct and 0 otherwise.
- Let $X = \text{number of correct guesses} = X_1 + \ldots + X_n$.
- $E[X_i] = \Pr[X_i = 1] = 1 / (n - i - 1)$.
- $E[X] = E[X_1] + \ldots + E[X_n] = 1/n + \ldots + 1/2 + 1/1 = H(n)$.

\[ \ln(\ln(n+1)) < H(n) < 1 + \ln n \]

The linearity of expectation is used here.
Coupon collector

Each box of cereal contains a coupon. There are $n$ different types of coupons. Assuming all boxes are equally likely to contain each coupon, how many boxes before you have $\geq 1$ coupon of each type?

Claim. The expected number of steps is $\Theta(n \log n)$.

Pf.

• Phase $j = \text{time between } j \text{ and } j + 1 \text{ distinct coupons.}$
• Let $X_j = \text{number of steps you spend in phase } j$.
• Let $X = \text{number of steps in total} = X_0 + X_1 + \ldots + X_{n-1}$.

$$E[X] = \sum_{j=0}^{n-1} E[X_j] = \sum_{j=0}^{n-1} \frac{n}{n-j} = n \sum_{i=1}^{n} \frac{1}{i} = nH(n)$$

prob of success = $(n - j) / n$

$\Rightarrow$ expected waiting time $= n / (n - j)$
13. RANDOMIZED ALGORITHMS

- content resolution
- global min cut
- linearity of expectation
- max 3-satisfiability
- universal hashing
- Chernoff bounds
- load balancing

Section 13.4
Maximum 3-satisfiability

**Maximum 3-satisfiability.** Given a 3-SAT formula, find a truth assignment that satisfies as many clauses as possible.

\[
\begin{align*}
C_1 &= x_2 \lor \overline{x}_3 \lor \overline{x}_4 \\
C_2 &= x_2 \lor x_3 \lor x_4 \\
C_3 &= \overline{x}_1 \lor x_2 \lor x_4 \\
C_4 &= \overline{x}_1 \lor \overline{x}_2 \lor x_3 \\
C_5 &= x_1 \lor x_2 \lor x_4
\end{align*}
\]

**Remark.** NP-hard search problem.

**Simple idea.** Flip a coin, and set each variable true with probability \( \frac{1}{2} \), independently for each variable.
**Maximum 3-satisfiability: analysis**

**Claim.** Given a 3-SAT formula with $k$ clauses, the expected number of clauses satisfied by a random assignment is $7k/8$.

**Pf.** Consider random variable $Z_j = \begin{cases} 1 & \text{if clause } C_j \text{ is satisfied} \\ 0 & \text{otherwise} \end{cases}$.

- Let $Z = \text{weight of clauses satisfied by assignment } Z_j$.

\[
E[Z] = \sum_{j=1}^{k} E[Z_j] = \sum_{j=1}^{k} \Pr[\text{clause } C_j \text{ is satisfied}] = \frac{7}{8} k
\]
**The Probabilistic Method**

**Corollary.** For any instance of 3-SAT, there exists a truth assignment that satisfies at least a \(\frac{7}{8}\) fraction of all clauses.

**Pf.** Random variable is at least its expectation some of the time. □

**Probabilistic method.** [Paul Erdös] Prove the existence of a non-obvious property by showing that a random construction produces it with positive probability!
Maximum 3-satisfiability: analysis

Q. Can we turn this idea into a $7/8$-approximation algorithm?
A. Yes (but a random variable can almost always be below its mean).

Lemma. The probability that a random assignment satisfies $\geq 7k / 8$ clauses is at least $1 / (8k)$.

Pf. Let $p_j$ be probability that exactly $j$ clauses are satisfied; let $p$ be probability that $\geq 7k / 8$ clauses are satisfied.

$$\frac{7}{8}k = E[Z] = \sum_{j \geq 0} j p_j$$

$$= \sum_{j < 7k/8} j p_j + \sum_{j \geq 7k/8} j p_j$$

$$\leq \left(\frac{7}{8}k - \frac{1}{8}\right) \sum_{j < 7k/8} p_j + k \sum_{j \geq 7k/8} p_j$$

$$\leq \left(\frac{7}{8}k - \frac{1}{8}\right) \cdot 1 + k p$$

Rearranging terms yields $p \geq 1 / (8k)$. 


Maximum 3-satisfiability: analysis

Johnson's algorithm. Repeatedly generate random truth assignments until one of them satisfies $\geq 7k/8$ clauses.

Theorem. Johnson's algorithm is a 7/8-approximation algorithm.

Pf. By previous lemma, each iteration succeeds with probability $\geq 1/(8k)$. By the waiting-time bound, the expected number of trials to find the satisfying assignment is at most $8k$. ▪
Maximum satisfiability

Extensions.
- Allow one, two, or more literals per clause.
- Find max weighted set of satisfied clauses.

Theorem. [Asano-Williamson 2000] There exists a 0.784-approximation algorithm for 3-SAT.

Theorem. [Karloff-Zwick 1997, Zwick+computer 2002] There exists a 7/8-approximation algorithm for version of MAX-3-SAT where each clause has at most 3 literals.

Theorem. [Håstad 1997] Unless $P = NP$, no $\rho$-approximation algorithm for MAX-3-SAT (and hence MAX-SAT) for any $\rho > 7/8$.

very unlikely to improve over simple randomized algorithm for MAX-3SAT
Monte Carlo vs. Las Vegas algorithms

**Monte Carlo.** Guaranteed to run in poly-time, likely to find correct answer. 
*Ex:* Contraction algorithm for global min cut.

**Las Vegas.** Guaranteed to find correct answer, likely to run in poly-time. 
*Ex:* Randomized quicksort, Johnson's MAX-3-SAT algorithm.

**Remark.** Can always convert a Las Vegas algorithm into Monte Carlo, but no known method (in general) to convert the other way.
**RP and ZPP**

**RP.** [Monte Carlo] Decision problems solvable with one-sided error in poly-time.

One-sided error.
- If the correct answer is *no*, always return *no*.
- If the correct answer is *yes*, return *yes* with probability $\geq \frac{1}{2}$.

**ZPP.** [Las Vegas] Decision problems solvable in expected poly-time.

**Theorem.** $P \subseteq ZPP \subseteq RP \subseteq NP$.

**Fundamental open questions.** To what extent does randomization help?
Does $P = ZPP$? Does $ZPP = RP$? Does $RP = NP$?
Section 13.6

13. Randomized Algorithms

- content resolution
- global min cut
- linearity of expectation
- max 3-satisfiability
- universal hashing
- Chernoff bounds
- load balancing
Dictionary data type

**Dictionary.** Given a universe $U$ of possible elements, maintain a subset $S \subseteq U$ so that inserting, deleting, and searching in $S$ is efficient.

**Dictionary interface.**
- `create()`: initialize a dictionary with $S = \emptyset$.
- `insert(u)`: add element $u \in U$ to $S$.
- `delete(u)`: delete $u$ from $S$ (if $u$ is currently in $S$).
- `lookup(u)`: is $u$ in $S$?

**Challenge.** Universe $U$ can be extremely large so defining an array of size $|U|$ is infeasible.

**Applications.** File systems, databases, Google, compilers, checksums P2P networks, associative arrays, cryptography, web caching, etc.
Hashing

Hash function. \( h : U \to \{ 0, 1, \ldots, n - 1 \} \).

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 \( \Theta(\sqrt{n}) \) random insertions.
- Separate chaining: \( H[i] \) stores linked list of elements \( u \) with \( h(u) = i \).
Ad-hoc hash function

Ad hoc hash function.

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

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.

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

Ideal hash function. Maps $m$ elements uniformly at random to $m$ 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 where you can easily find items where you put them.

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 family of hash functions. [Carter-Wegman 1980s]

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

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

\[
\begin{array}{cccccc}
\text{a} & \text{b} & \text{c} & \text{d} & \text{e} & \text{f} \\
\hline
\text{h}_1(x) & 0 & 1 & 0 & 1 & 0 & 1 \\
\text{h}_2(x) & 0 & 0 & 0 & 1 & 1 & 1 \\
\end{array}
\]

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

... not universal

\[
\begin{array}{cccccc}
\text{a} & \text{b} & \text{c} & \text{d} & \text{e} & \text{f} \\
\hline
\text{h}_1(x) & 0 & 1 & 0 & 1 & 0 & 1 \\
\text{h}_2(x) & 0 & 0 & 0 & 1 & 1 & 1 \\
\text{h}_3(x) & 0 & 0 & 1 & 0 & 1 & 1 \\
\text{h}_4(x) & 1 & 0 & 0 & 1 & 1 & 0 \\
\end{array}
\]

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

... universal
Universal hashing: analysis

**Proposition.** Let $H$ be a universal family of hash functions; let $h \in H$ be chosen uniformly at random from $H$; and let $u \in U$. 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 $u$.

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

- **linearity of expectation**
- $X_s$ is a 0-1 random variable
- universal (assumes $u \not\in S$)

**Q.** OK, but how do we design a universal class of hash functions?
Designing a universal family of hash functions

**Theorem.** [Chebyshev 1850] There exists a prime between \( n \) and \( 2n \).

**Modulus.** Choose a prime number \( p \approx n \). \( \rightarrow \) no need for 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 \( 0 \leq a_i < p \), define

\[
    h_a(x) = \left( \sum_{i=1}^{r} a_i x_i \right) \mod p
\]

**Hash function family.** \( H = \{ h_a : a \in A \} \).
Designing a universal family of hash functions

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

**Pf.** Let $x = (x_1, x_2, \ldots, x_r)$ and $y = (y_1, y_2, \ldots, y_r)$ be two distinct elements of $U$. We need to show that $\Pr[h_a(x) = h_a(y)] \leq 1/n$.

- Since $x \neq y$, there exists an integer $j$ such that $x_j \neq y_j$.
- We have $h_a(x) = h_a(y)$ iff

$$a_j (y_j - x_j) \mod p = \sum_{i \neq j} a_i (x_i - y_i) \mod p$$

- Can assume $a$ was chosen uniformly at random by first selecting all coordinates $a_i$ where $i \neq j$, then selecting $a_j$ at random. Thus, we can assume $a_i$ is fixed for all coordinates $i \neq j$.
- Since $p$ is prime, $a_j z = m \mod p$ has at most one solution among $p$ possibilities. \(\text{see lemma on next slide}\)
- Thus $\Pr[h_a(x) = h_a(y)] = 1/p \leq 1/n$. \(\blacksquare\)
Number theory fact

**Fact.** Let $p$ be prime, and let $z \neq 0 \mod p$. Then $\alpha z = m \mod p$ has at most one solution $0 \leq \alpha < p$.

**Pf.**

- Suppose $\alpha$ and $\beta$ are two different solutions.
- Then $(\alpha - \beta) z = 0 \mod p$; hence $(\alpha - \beta) z$ is divisible by $p$.
- Since $z \neq 0 \mod p$, we know that $z$ is not divisible by $p$; it follows that $(\alpha - \beta)$ is divisible by $p$.
- This implies $\alpha = \beta$. □

**Bonus fact.** Can replace "at most one" with "exactly one" in above fact.

**Pf idea.** Euclid's algorithm.
13. Randomized Algorithms

- content resolution
- global min cut
- linearity of expectation
- max 3-satisfiability
- universal hashing
- Chernoff bounds
- load balancing

Section 13.9
Chernoff Bounds (above mean)

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

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

**Pf.** We apply a number of simple transformations.

- For any $t > 0$,

$$\Pr[X > (1 + \delta)\mu] = \Pr\left[e^{tX} > e^{t(1+\delta)\mu}\right] \leq e^{-t(1+\delta)\mu} \cdot \mathbb{E}[e^{tX}]$$

  - $f(x) = e^{tx}$ is monotone in $x$
  - Markov’s inequality: $\Pr[X > a] \leq \mathbb{E}[X] / a$

- Now

$$\mathbb{E}[e^{tX}] = \mathbb{E}[e^{t\sum X_i}] = \prod_i \mathbb{E}[e^{tX_i}]$$

  - definition of $X$
  - independence
Chernoff Bounds (above mean)

**Pf.** [ continued ]

- Let $p_i = \Pr [X_i = 1]$. Then,

\[
E[e^{t X_i}] = p_i e^t + (1 - p_i) e^0 = 1 + p_i (e^t - 1) \leq e^{p_i (e^t - 1)}
\]

for any $\alpha \geq 0$, $1 + \alpha \leq e^{\alpha}$

- Combining everything:

\[
\Pr[X > (1 + \delta) \mu] \leq e^{-t(1+\delta)\mu} \prod_i E[e^{t X_i}] \leq e^{-t(1+\delta)\mu} \prod_i e^{p_i (e^t - 1)} \leq e^{-t(1+\delta)\mu} e^{\mu(e^t - 1)}
\]

previous slide inequality above

- Finally, choose $t = \ln(1 + \delta)$. ■
Chernoff Bounds (below mean)

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

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

**Pf idea.** Similar.

**Remark.** Not quite symmetric since only makes sense to consider $\delta < 1$. 
13. Randomized Algorithms

- content resolution
- global min cut
- linearity of expectation
- max 3-satisfiability
- universal hashing
- Chernoff bounds
- load balancing
Load Balancing

**Load balancing.** System in which $m$ jobs arrive in a stream and need to be processed immediately on $m$ identical processors. Find an assignment that balances the workload across processors.

**Centralized controller.** Assign jobs in round-robin manner. Each processor receives at most $\lfloor m/n \rfloor$ jobs.

**Decentralized controller.** Assign jobs to processors uniformly at random. How likely is it that some processor is assigned "too many" jobs?
Load balancing

Analysis.

- Let $X_i = \text{number of jobs assigned to processor } i$.
- Let $Y_{ij} = 1$ if job $j$ assigned to processor $i$, and 0 otherwise.
- We have $E[Y_{ij}] = 1/n$.
- Thus, $X_i = \sum_j Y_{ij}$, and $\mu = E[X_i] = 1$.
- Applying Chernoff bounds with $\delta = c - 1$ yields $\Pr[X_i > c] < \frac{e^{c-1}}{c^c}$.

- Let $\gamma(n)$ be number $x$ such that $x^x = n$, and choose $c = e \gamma(n)$.

\[
\Pr[X_i > c] < \frac{e^{c-1}}{c^c} < \left(\frac{e}{c}\right)^c = \left(\frac{1}{\gamma(n)}\right)^{e\gamma(n)} < \left(\frac{1}{\gamma(n)}\right)^{2\gamma(n)} = \frac{1}{n^2}
\]

- Union bound $\Rightarrow$ with probability $\geq 1 - 1/n$ no processor receives more than $e \gamma(n) = \Theta(\log n / \log \log n)$ jobs.

Bonus fact: with high probability, some processor receives $\Theta(\log n / \log \log n)$ jobs.
Load balancing: many jobs

**Theorem.** Suppose the number of jobs $m = 16\, n \ln n$. Then on average, each of the $n$ processors handles $\mu = 16 \ln n$ jobs. With high probability, every processor will have between half and twice the average load.

**Pf.**

- Let $X_i, Y_{ij}$ be as before.
- Applying Chernoff bounds with $\delta = 1$ yields

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

\[
\Pr[X_i < \frac{1}{2}\mu] < e^{-\frac{1}{2}(\frac{1}{2})^2(16n\ln n)} = \frac{1}{n^2}
\]

- Union bound $\implies$ every processor has load between half and twice the average with probability $\geq 1 - 2/n$.  □