Lecture 4 (part 2) — Fingerprints, Hashing & the Isolation Lemma

> Course **2-INF-135/15 Pravdepodobnostné algoritmy**, LS 2025/26. Source slides: `04_metody23.pdf`, slides 5–34. Continues the **methods** block; part 1 (§1–§4) was *eliminating the adversary*. This part is the **second method — fingerprints (odtlačky)** — and its biggest customers, **hashing** and **matching**.

Where we are

Part 1 was the first design method: randomize *your strategy* so the adversary has no fixed target. This part is the second method: **fingerprints**. The slogan is one line:

> **Don't compare the giant objects — compare tiny random *summaries* of them.**

We will see the same trick wear five different costumes (matrix products, polynomials, branching programs, text search, graph isomorphism), then watch it grow into the theory of **universal hashing**, and finally into the **Isolation Lemma**, which turns "does a matching exist?" into a determinant and makes the matching *unique* so parallel machines can agree on it.

---

5. The fingerprint method (metóda odtlačkov)

We want to decide **equality / equivalence of two complex objects** O_1, O_2 — objects so big that comparing them directly is expensive (matrices, polynomials in normal form, long strings, function tables). The idea:

> Take a **family** M of maps from the *full* representation down to a *small* partial one. Pick one **at random**, $h \in_R M$, and compare the **fingerprints** $h(O_1) \stackrel{?}{=} h(O_2)$.
>
> - If $h(O_1) = h(O_2)$: report **"(probably) equivalent."**
> - If $h(O_1) \neq h(O_2)$: report **"definitely not equivalent."**

Three requirements make this an algorithm rather than a wish:

1. h is **efficiently computable**;
2. the comparison $h(O_1) \stackrel{?}{=} h(O_2)$ is **efficient** (fingerprints are small);
3. **soundness in one direction**: $O_1 = O_2 \;\Rightarrow\; h(O_1) = h(O_2) \quad (\text{always}),$
   $O_1 \neq O_2 \;\Rightarrow\; h(O_1) \neq h(O_2) \quad \text{for *enough* } h \in M.$

> **The error is one-sided.** Different objects *always* differ; equal objects *never* differ. The only mistake is a **false positive** — two different objects that happen to collide on this particular random h. Requirement 3 guarantees only a *small fraction* of h's collide, so the failure probability is small. This is exactly a **co-RP / Monte-Carlo** flavour: an "unequal" verdict is gospel, an "equal" verdict is *probably* true, and we amplify by repeating with fresh h. *(Ties back to Lecture 3's one-sided error classes.)*

The whole rest of this section is: *pick the right family M for the objects at hand.*

---

6. Freivalds — verifying a matrix product in $O(n^2)$

The problem. Someone claims $A \cdot B = C$ for $n \times n$ matrices. *Checking* by recomputing AB costs $O(n^3)$ naively (or $O(n^{2.37})$ with fast multiplication — complicated and constant-heavy). Can we **verify** a claimed product faster than computing one? Yes — in $O(n^2)$, the cost of barely touching the matrices.

The fingerprint. Project everything through a random $0/1$ vector $\alpha \in_R \{0,1\}^n$:

$$\text{check} \quad A(B\alpha) \stackrel{?}{=} C\alpha.$$

Computed **right-to-left**: $B\alpha$ is a matrix–vector product ($O(n^2)$), then $A(B\alpha)$ another ($O(n^2)$), and $C\alpha$ another. Total $O(n^2)$ — never form AB.

Correctness. The fingerprint of a matrix X is the vector $X\alpha$.

- If $AB = C$ then $A(B\alpha) = (AB)\alpha = C\alpha$ **always** (requirement 3, the safe side).
- If $AB \neq C$, let $D = AB - C \neq 0$. We need $\Pr[D\alpha = 0] \leq \frac{1}{2}$.

> **The one-line heart.** $D \neq 0$ has some nonzero entry d_{ij}. Look at row i: $(D\alpha)_i = \sum_k d_{ik}\alpha_k = d_{ij}\,\alpha_j + (\text{terms not involving } \alpha_j).$ Reveal every coordinate of α **except** α_j first (principle of deferred decisions). Now $(D\alpha)_i = 0$ forces α_j to one specific value. But α_j is a fair coin over $\{0,1\}$, so it hits that value with probability $\leq \frac{1}{2}$. Hence $\Pr[(D\alpha)_i = 0] \leq \frac{1}{2}$, so $\Pr[D\alpha = 0] \leq \frac{1}{2}$.

$$\boxed{\Pr[A(B\alpha) = C\alpha \mid AB \neq C] \leq \tfrac{1}{2}.}$$

Repeat with t independent α's to drive the error to 2^{-t}.

> **Punchline.** Verifying a product is *strictly easier* than computing it — and the proof never needed anything about A, B, C beyond "$D \neq 0$ has a nonzero entry." A random linear projection cannot kill a nonzero matrix more than half the time. *(Exam-critical: the "matrix match" fingerprint, exam-1 Q1 / exam-3 Q1.)*

---

7. Polynomial identity testing & the Schwartz–Zippel lemma

Freivalds is the matrix face of a much bigger idea: **two objects are equal iff a certain polynomial is identically zero, and a nonzero polynomial is rarely zero at a random point.**

Univariate warm-up

To test $p_1(x) \stackrel{?}{=} p_2(x)$: form $q = p_1 - p_2$. If $p_1 \neq p_2$ then $q \not\equiv 0$ has degree $\leq n$, so **at most n roots**. Pick $\alpha \in_R S$ for a large set $S \subseteq F$:

$$\Pr[q(\alpha) = 0 \mid p_1 \neq p_2] \;\leq\; \frac{n}{|S|}.$$

Deterministically you'd expand to normal form — possibly **exponential** time. Randomly you just **evaluate at one point**. This single idea powers *perfect matching* (§14), *read-once branching programs* (§8), and *pattern matching* (§9).

Schwartz–Zippel (the multivariate engine)

> **Lemma (Schwartz–Zippel).** Let $Q(x_1, \ldots, x_n) \in F[x_1, \ldots, x_n]$ have **total degree** d, let $S \subseteq F$, and pick $\alpha_1, \ldots, \alpha_n \in_R S$ independently. Then $\Pr[\,Q(\alpha_1, \ldots, \alpha_n) = 0 \mid Q \not\equiv 0\,] \;\leq\; \frac{d}{|S|}$.

The univariate fact ("degree $d \Rightarrow \leq d$ roots") is the $n = 1$ case; this lifts it to many variables at the *same* rate $d/|S|$.

Proof — induction on the number of variables n.

- **Base $n = 1$:** a nonzero univariate polynomial of degree $\leq d$ has $\leq d$ roots, so $\Pr \leq d/|S|$.

- **Step $n > 1$:** let $k \le d$ be the highest power of x_1 appearing in Q, and peel x_1 out: $Q(x_1, \dots, x_n) = \sum_{i=0}^{k} x_1^i \, Q_i(x_2, \dots, x_n)$, where the leading coefficient $Q_k \not\equiv 0$ is a polynomial in $x_2, \dots, x_n$ of total degree $\le d - k$. Now $Q(\alpha) = 0$ splits into two cases over the *tail* variables $\alpha' = (\alpha_2, \dots, \alpha_n)$:
  - **(a) $Q_k(\alpha') = 0$.** By the induction hypothesis on the $(n-1)$-variable, degree-$(d-k)$ polynomial Q_k: $\Pr[Q_k(\alpha') = 0] \le \frac{d-k}{|S|}$.
  - **(b) $Q_k(\alpha') \neq 0$.** Then $q(x_1) := Q(x_1, \alpha')$ is a *genuine* univariate polynomial of degree exactly k (its top coefficient $Q_k(\alpha') \neq 0$), so $\Pr[q(\alpha_1) = 0 \mid Q_k(\alpha') \neq 0] \le \frac{k}{|S|}$.

  Glue with $\Pr[A] \le \Pr[B] + \Pr[A \mid \overline{B}]$ (here B = "case (a)"):
  $\Pr[Q(\alpha) = 0] \le \frac{d-k}{|S|} + \frac{k}{|S|} = \frac{d}{|S|}$. ∎

> **The deep point.** Peeling off the highest-degree variable lets the induction *split the degree budget* $d = (d-k) + k$ exactly between "the leading coefficient vanishes" and "the leading coefficient survives but the univariate slice vanishes." Neither case can be bad more than its share, and they add up to the full degree. That clean additivity is why the bound stays $d/|S|$ no matter how many variables.

Variant: counting roots over $\mathbb{Z}_p$

A sibling lemma counts roots directly. If $Q(x_1, \dots, x_n)$ over $\mathbb{Z}_p$ has **per-variable** degree $\le d$, then it has at most $n\,d\,p^{n-1}$ roots — so a uniform random point in $\mathbb{Z}_p^n$ is a root with probability $\le nd/p$. (Same induction: either all the Q_i vanish — $(n-1)d\,p^{n-1}$ roots by induction — or some $Q_j \neq 0$ and the univariate slice contributes $\le k\,p^{n-1}$; together $\le n d\,p^{n-1}$.)

---

8. Read-once branching programs (1BP equivalence)

A beautiful application: deciding whether two **branching programs** compute the same Boolean function, with *no* known efficient deterministic algorithm — but an easy fingerprint one.

1BP (read-once branching program): an acyclic graph; each internal vertex is labelled by a variable; each has two out-edges labelled 0 and 1. To evaluate on input $\alpha \in \{0,1\}^n$, start at the root and at each vertex follow the edge labelled by the *tested variable's value*. **Read-once** = every variable is tested at most once on any path. A **1-path** runs from the root to the **"1"-leaf**; $P(\mathrm{BP})$ is the set of 1-paths. The program represents f if $P(\mathrm{BP})$ is exactly the set of α with $f(\alpha) = 1$.

Equivalence is a polynomial identity. Turn each 1-path $y = x_1^{v_1} x_2^{v_2} \cdots$ into a monomial that is 1 exactly on that path's inputs:

$$p(y) = \prod_{j=1}^{n} p_j, \qquad p_j = \begin{cases} x_j, & v_j = 1, \\ 1 - x_j, & v_j = 0, \end{cases} \qquad Q_{\mathrm{BP}}(x_1, \ldots, x_n) = \sum_{y \in P(\mathrm{BP})} p(y).$$

On any Boolean input α, **exactly one** path's monomial is 1 (the path the input follows) and the rest are 0, so $Q_{\mathrm{BP}}(\alpha) = f(\alpha)$ on the whole cube. Because each Q_{BP} is **multilinear**, it is determined by its values on $\{0,1\}^n$. Hence

$$\mathrm{BP}_1 \equiv \mathrm{BP}_2 \iff f_1 = f_2 \text{ on } \{0,1\}^n \iff Q_{\mathrm{BP}_1} = Q_{\mathrm{BP}_2} \text{ as polynomials.}$$

Now test $Q_{\mathrm{BP}_1} - Q_{\mathrm{BP}_2} \overset{?}{\equiv} 0$ with **Schwartz–Zippel**: evaluate both at a random point over a large field S (going *beyond* $\{0,1\}$ to get a real probability gap). Example monomial sum:
$Q(x_1, x_2, x_3) = (1 - x_1)x_2x_3 + (1 - x_1)(1 - x_2)(1 - x_3) + x_1x_2.$

> **Why this is striking.** We never expand the function table (2^n entries). We compare two programs by *one random evaluation* of their arithmetizations. The same "Boolean function → multilinear polynomial" move ("arithmetization") is the seed of the whole **IP = PSPACE** story.

---

9. Pattern matching — Karp–Rabin fingerprints

The problem. Text $X = x_1 \ldots x_n$, pattern $Y = y_1 \ldots y_m$ (bits). Find an occurrence: a position j with $X(j) := x_j x_{j+1} \ldots x_{j+m-1} = Y$.

The fingerprint = remainder modulo a random prime. Read each m-bit window as a number. Pick a random prime $p \in_R \{\text{primes} \le \tau\}$ (with τ a function of m, n to be tuned), and compare remainders:

$$O_p(X(j)) := X(j) \bmod p, \qquad O_p(Y) := Y \bmod p, \qquad \text{report } j \text{ if } O_p(X(j)) = O_p(Y).$$

Why it's fast — the rolling hash. $O_p(Y)$ is computed once. Each window fingerprint is obtained from the previous one in $O(1)$ arithmetic operations ($O(\log p)$ bit-ops): drop the top bit, shift, add the new bit, all mod p:

$$O_p(X(k)) \;=\; \big(2 \cdot O_p(X(k-1)) - x_{k-1}\, 2^m + x_{k+m-1}\big) \bmod p.$$

Total time $O(n + m)$ instead of $O(nm)$ for naive matching.

Error analysis. A false match at j means $p \mid |X(j) - Y|$, where $|X(j) - Y|$ is an m-bit number, hence $< 2^m$, hence has **fewer than m prime divisors**. The number of primes $\le \tau$ is $\approx \tau / \ln \tau$ (prime number theorem). So per position $\Pr[\text{false match}] \le \dfrac{m}{\tau / \ln \tau}$, and summing over the $\le n$ positions:

$$\sum_j \Pr[\,O_p(Y) = O_p(X(j)) \mid Y \neq X(j)\,] \;\le\; \frac{nm\ln\tau}{\tau} \;=\; O\!\left(\frac{nm\log\tau}{\tau}\right).$$

Choosing $\tau = n^2 m \log(n^2 m)$ makes this $\le 2/n$.

Monte-Carlo vs. Las-Vegas.

- **MC:** on each fingerprint match, *verify* by direct character comparison — total extra $O(n+m)$, error $\le 2/n$.
- **LV:** verify every reported match; on a **false** match, **restart with a fresh prime**. Expected work stays $O(n+m)$ and $\Pr[\text{more than } k \text{ restarts}] \le 1/n^k$.

> **Punchline.** A whole m-bit window is crushed to a number mod a small prime, and the rolling update means the fingerprint of the *next* window is one cheap step from the current — turning $O(nm)$ into $O(n+m)$. The error is one-sided and verifiable, so the Monte-Carlo algorithm upgrades to Las-Vegas for free.

---

10. Fingerprints across a conversation — interactive proofs

The fingerprint idea generalizes from "one random check" to a **dialogue**. An **interactive protocol** is a pair (P, V):

	Prover P (dôkaz / student)	**Verifier V (verifikácia / teacher)**
power	**unbounded** computation	**polynomial-time, randomized**
sees	the public messages	its own random bits (*P does not*)

They exchange messages; the verifier has the **last word**. We want:

$$x \in L \;\Rightarrow\; \exists P:\; V(P,x) = 1 \quad (\text{a true claim has a convincing prover — *completeness*}),$$

$$x \notin L \;\Rightarrow\; \forall P:\; \Pr[V(P,x) = 1] \le \tfrac{1}{2} \quad (\text{a false claim survives only by luck — *soundness*}).$$

IP is the class of languages with such a protocol.

Graph non-isomorphism — the showcase

$$L_{GI} = \{(G_1, G_2) : \exists \tau,\; G_1 = \tau(G_2)\} \in \mathsf{NP} \quad (\text{witness} = \tau),$$

$$L_{GNI} = \{(G_1, G_2) : \forall \tau,\; G_1 \neq \tau(G_2)\} \in \mathsf{coNP}.$$

For **non-isomorphism** we have **no short certificate** — how do you exhibit a proof that *no* permutation works? Interaction + randomness gives one:

> **Protocol (verifier V moves first):**
>
> 1. V secretly flips $i \in_R \{1,2\}$ and picks a random permutation τ.
> 2. V computes $H = \tau(G_i)$ and sends H to P.
> 3. P answers with $j \in \{1,2\}$ — its guess for which graph H came from.
> 4. V accepts "G_1, G_2 **non-isomorphic**" iff $i = j$.

> **Theorem.** If G_1, G_2 are non-isomorphic, an honest P convinces V with certainty. If they **are** isomorphic, *any* (even cheating) P convinces V with probability $\leq \frac{1}{2}$.

Why it works — the intuition.

- **Non-isomorphic:** H is isomorphic to *exactly one* of G_1, G_2. The all-powerful P can simply test which, so $j = i$ **always** $\Rightarrow V$ accepts.
- **Isomorphic:** now H is isomorphic to *both*, and a random relabelling τ of G_1 has **exactly the same distribution** as a random relabelling of G_2. So H leaks **zero information** about the secret i — P can only guess, $\Pr[i = j] \leq \frac{1}{2}$.

> **The deep point.** Randomness lets the verifier pose a question whose answer P knows *iff the claim is true*: "tell apart two graphs" is possible only when they really are different. The verifier's hidden coin is the fingerprint P cannot fake. This is the entry point to $\mathsf{IP} = \mathsf{PSPACE}$ — interaction + randomness buys "proofs" that ordinary coNP certificates seem unable to give.

---

11. Hashing I — linear probing and balls-in-boxes

Fingerprints meet **data structures**: a hash function is a fingerprint we *store things by*. Before the theory, two concrete analyses.

Linear probing in space $n = 3m$ has $O(1)$ expected FIND

Store m keys in a table T of size $n = 3m$ (load factor $\frac{1}{3}$). On a collision at slot $h(x)$, probe $h(x)+1, h(x)+2, \ldots$ until a free slot. Claim: **expected probe length is $O(1)$.**

Proof gadget — a binary tree over the table. Build a complete binary tree whose **leaves are the table slots**. A vertex v at height k covers 2^k consecutive slots. Say a key x *hashes into* v if $h(x)$ lands in v's subtree. The expected number of keys hashing into v is $\mu = \frac{m}{n} \cdot 2^k = \frac{2^k}{3}$.

Call v **dangerous** if at least 2μ keys hash into it. By **Chernoff** ($\Pr[X \geq (1+\delta)\mu] \leq \left(e^\delta/(1+\delta)^{1+\delta}\right)^\mu$ with $\delta = 1$):

$$\Pr[\#\text{keys} \geq 2\mu] \;\leq\; \left(\frac{e}{4}\right)^\mu.$$

Since $e/4 < 1$, dangerous vertices are *exponentially* unlikely in their height. A long **run** (cluster) B of length $b \in \{2^\ell, \ldots, 2^{\ell+1}\}$ forces **at least one of ~ 3 subtrees of height $\ell - 2$** covering it to be dangerous, so

$$\Pr[\text{run length} \in \{2^\ell, \ldots, 2^{\ell+1}\}] \;\leq\; 3\left(\frac{e}{4}\right)^{2^{\ell-2}/3},$$

$$E[\text{FIND}] = \sum_b b\, p_b \;\leq\; 3\sum_\ell 2^\ell \left(\frac{e}{4}\right)^{2^{\ell-2}/3} = O(1).$$

The combinatorial core ("a long run needs a dangerous subtree") is a **capacity / pigeonhole** argument: if all three covering subtrees of height $\ell - 2$ were *safe* (each holding $< 2\mu$ keys), they could not supply enough keys to fill a contiguous run that long — there would be a **hole** (empty slot) inside it, contradicting that it is one unbroken run.

> **Deep point.** The slack load factor $\frac{1}{3}$ is doing the work: at every height the *expected* fill is a third of capacity, so being *double* the mean (dangerous) is a large deviation, and Chernoff makes it vanish fast enough that the expected cluster length — hence expected FIND — is a constant independent of m.

Balls in boxes (guličky a krabice)

Throw m balls uniformly into n boxes. Three staples:

1. **Load:** X = balls in a fixed box, $E[X] = m/n$.
2. **Empty boxes:** $Z_i = 1$ iff box i is empty; $E[Z_i] = (1 - \frac{1}{n})^m \approx e^{-m/n}$, so $E[Z] = n\, e^{-m/n}$. For $m = n$: $E[Z] \approx n/e$ (about a third of the boxes stay empty even with as many balls as boxes).
3. **First collision (birthday):** with k balls, $\Pr[\text{no collision}] = \prod_{i=1}^{k-1}\left(1 - \frac{i}{n}\right) \leq \prod_{i=1}^{k-1} e^{-i/n} = e^{-k(k-1)/(2n)}$. This drops below $\frac{1}{2}$ once $k(k-1)/(2n) \geq \ln 2$, i.e. $k \sim \sqrt{n}$.

> **The $\sqrt{n}$ threshold (birthday paradox)** is the single most-reused fact in hashing: collisions become likely at $k \approx \sqrt{n}$ items, which is exactly why a table that wants *no* collisions among m keys needs $\sim m^2$ slots (§13).

---

12. Universal hash families

Setup. Universe $U = \{0, \dots, m-1\}$, table $T = \{0, \dots, n-1\}$, hash $h : U \to T$. For *any single fixed* h there is a bad input (some set of keys that all collide). The adversary wins against any fixed function. **Solution — randomize the function**: keep a *family* $H = \{h\}$ and pick $h \in_R H$. This is **"eliminate the adversary" (part 1) applied to the hash function itself** — and it yields a good *deterministic* structure for a static dictionary (fix the lucky h once).

Definitions

For $m \geq n$ and a family H of functions $U \to T$:

Property	Condition (for all *distinct* $x_1, \dots, x_k$; $h \in_R H$)
universal	$\Pr[h(x) = h(y)] \leq \dfrac{1}{n}$ for $x \neq y$
k-universal	$\Pr[h(x_1) = \cdots = h(x_k)] \leq \dfrac{1}{n^{\,k-1}}$
strongly k-universal (k-independent)	$\forall y_1, \dots, y_k: \;\; \Pr[h(x_1) = y_1, \dots, h(x_k) = y_k] = \dfrac{1}{n^{\,k}}$

Universal = "collisions no more likely than for a *truly random* function." Strongly k-universal = "on any k inputs the outputs *look fully independent and uniform*."

Counting collisions with a 2-universal family

Let $S = \{x_1, \dots, x_m\}$, $X_{ij} = 1$ iff $h(x_i) = h(x_j)$, and $X = \sum_{i<j} X_{ij}$ the number of colliding pairs:

$$E[X] = \sum_{i<j} \Pr[h(x_i) = h(x_j)] \leq \binom{m}{2} \frac{1}{n} \leq \frac{m^2}{2n}.$$

A box holding Y keys creates $\binom{Y}{2} \approx Y^2/2$ collisions, so by **Markov** $\Pr[X \geq m^2/n] \leq \frac{1}{2}$, giving $\Pr[Y \geq m\sqrt{2/n}] \leq \frac{1}{2}$; for $m = n$, $\Pr[\text{max load} \geq \sqrt{2n}] \leq \frac{1}{2}$. Good enough for *one level* — but the max bin is still $\sim \sqrt{n}$, which §13 fixes.

A concrete 2-universal family

$$h_{a,b}(x) = \big((ax+b) \bmod p\big) \bmod n, \qquad p \geq m \geq n \text{ prime},$$

$$H = \{h_{a,b} : 1 \leq a \leq p-1,\ 0 \leq b \leq p-1\}, \quad |H| = p(p-1).$$

> **Lemma.** H is 2-universal: $\Pr[h_{a,b}(x_1) = h_{a,b}(x_2)] \leq \frac{1}{n}$ for $x_1 \neq x_2$.

Proof. First, $x_1 \neq x_2 \Rightarrow ax_1 + b \not\equiv ax_2 + b \pmod p$ (since $a \neq 0$). For fixed $x_1 \neq x_2$, the map $(a,b) \mapsto (u,v) = (ax_1+b,\ ax_2+b) \bmod p$ is a **bijection** onto pairs (u,v) with $u \neq v$ (solve the 2×2 system — $x_1 \neq x_2$ makes it invertible). A collision means $u \equiv v \pmod n$ with $u \neq v$. For each u there are at most $\lceil p/n \rceil - 1 \leq (p-1)/n$ values $v \neq u$ with $v \equiv u \pmod n$. So the number of colliding pairs is $\leq p \cdot \frac{p-1}{n}$, and

$$\Pr[\text{collision}] \ \leq\ \frac{p(p-1)/n}{p(p-1)} \ =\ \frac{1}{n}. \qquad \blacksquare$$

Strongly 2-universal families

Scalar version. $U = T = \{0, \ldots, p-1\}$, p prime, $h_{a,b}(x) = (ax+b) \bmod p$,
$H = \{h_{a,b} : 0 \leq a, b \leq p-1\}$, $|H| = p^2$.

> **Lemma.** $\Pr[h(x_1) = y_1,\ h(x_2) = y_2] = 1/p^2$ for distinct x_1, x_2.

Because the linear system $ax_1 + b = y_1,\ ax_2 + b = y_2 \pmod p$ has a **unique** solution (a,b) (Vandermonde, $x_1 \neq x_2$). So *exactly one* of the p^2 functions sends $x_1 \mapsto y_1,\ x_2 \mapsto y_2$ — probability $1/p^2$.

Vector version. $U = \{0, \ldots, p^k - 1\}$, $T = \{0, \ldots, p-1\}$, identify
$u \leftrightarrow (u_0, \ldots, u_{k-1}) \in \{0, \ldots, p-1\}^k$, and $h_{a,b}(u) = \left(\sum_{i=0}^{k-1} a_i u_i + b\right) \bmod p$. If $u_1 \neq u_2$ they differ in some coordinate i; fixing the other a_j, the two equations
$a_i u_{1,i} + b = \ldots,\ a_i u_{2,i} + b = \ldots$ pin down (a_i, b) uniquely among p^2 choices, giving
$\Pr[h(u_1) = y_1 \wedge h(u_2) = y_2] = 1/p^2$. Still strongly 2-universal.

> **Why "strongly" matters.** Universal controls *collisions*; strongly 2-universal controls *the actual output distribution on any two points* — and that is what limited-independence derandomization needs (Lecture 2's pairwise independence is exactly strong 2-universality).

---

13. Perfect hashing — $O(1)$ worst-case lookups

A **static** dictionary S ($|S| = m$) wants $O(1)$ **worst-case** FIND, not just expected. With 2-universal H and chaining, the *expected* bin size is great but the *max* bin is $\sim \sqrt{n}$ — too slow in the worst case. We want a **perfect** hash: no collisions on S.

First: expected bin size

> **Lemma.** For h from a 2-universal family and $X = |\mathrm{bin}(h(x))|$,
> $E[X] = \begin{cases} m/n, & x \notin S, \\ 1 + (m-1)/n, & x \in S. \end{cases}$

(Indicators $X_i = 1$ iff $h(x) = h(s_i)$; $E[X_i] = 1/n$ for $s_i \neq x$, plus the certain self-term when $x \in S$.) For $n = m$ this is $E[X] \leq 2$ — but *somewhere* a bin still has $\sim \sqrt{n}$ keys. So average is fine, worst case is not.

Perfect hashing in $O(m^2)$ space

> If $n \geq m^2$ then $\Pr[h \text{ is perfect on } S] \geq \frac{1}{2}$.

With $X = \sum_{i<j} X_{ij}$ counting collisions, $E[X] \leq \binom{m}{2}/n < m^2/(2n) \leq \frac{1}{2}$, so by Markov $\Pr[X \geq 1] \leq \frac{1}{2} \Rightarrow \Pr[\text{no collision}] \geq \frac{1}{2}$. **Find one** by Las Vegas: try random h's; $E[\#\text{tries}] = 2$. Lookup is then $O(1)$ worst case — but space is a wasteful $O(m^2)$.

Two-level perfect hashing in $O(m)$ — the FKS scheme

> **Lemma.** Two-level hashing gives perfect hashing in $O(m)$ space.

- **Level 1:** a 2-universal h into $n = m$ bins; bin i gets b_i keys.
- **Level 2:** each bin i gets its *own* perfect hash table of **quadratic size** b_i^2 (perfect within the bin w.p. $\geq \frac{1}{2}$ by the $O(m^2)$ result above).

The only worry is total second-level space $\sum_i b_i^2$. Pick a level-1 h whose collision count satisfies $X = \sum_i \binom{b_i}{2} \leq m$ (possible since $E[X] \leq m^2/(2n) = m/2$, so $\Pr[X \geq m] \leq \frac{1}{2}$). Then, using $b^2 = 2\binom{b}{2} + b$,

$$\sum_i b_i^2 \;=\; 2\sum_i \binom{b_i}{2} + \sum_i b_i \;\leq\; 2m + m \;=\; 3m \;=\; O(m).$$

> **Punchline (FKS, Fredman–Komlós–Szemerédi).** $O(m)$ space, $O(1)$ **worst-case** lookup, for a static set — optimal. The trick is fractal: the *same* "quadratic table ⇒ no collisions" idea is applied once globally (to bound $\sum b_i^2$) and once *inside each bin* (to make each bin perfect). The birthday $\sqrt{n}$ threshold from §11 is why quadratic is exactly the right size at each level.

---

14. Matchings via matrices — Tutte and the Isolation Lemma

The grand finale: fingerprints decide whether a graph has a **perfect matching**, and the **Isolation Lemma** makes that matching *unique* so we can even compute it in parallel.

Tutte matrix — matching existence is a polynomial identity

For a graph G on n vertices, the **Tutte matrix** A has an indeterminate per edge:

$$A(i,j) = \begin{cases} x_{ij}, & (i,j) \in E,\ i < j, \\ -x_{ij}, & (i,j) \in E,\ i > j, \\ 0, & (i,j) \notin E. \end{cases}$$

(It is skew-symmetric.)

> **Theorem (Tutte).** G has a perfect matching $\iff \det(A) \not\equiv 0$ (as a polynomial in the x_{ij}).

Why. Expand $\det(A) = \sum_{\pi \in S_n} (-1)^{\operatorname{sgn}(\pi)} \prod_{i=1}^{n} A(i,\pi(i))$, and read each permutation π as a **cycle cover** of G.

- **Odd-length cycles cancel:** reversing the orientation of an odd cycle flips that term's sign, so π and its reversed partner add to 0. Permutations with any odd cycle contribute nothing.
- **Even cycles survive** (reversal keeps the sign), and an all-even cycle cover yields a **perfect matching** (take alternate edges of each even cycle).

So $\det(A) \not\equiv 0 \iff$ some surviving (all-even, in particular the matching) term exists $\iff G$ has a perfect matching. **Testing $\det(A) \not\equiv 0$ is polynomial identity testing** — by **Lovász**, substitute random values $x_{ij} \in_R \mathbb{Z}_p$ with $p = \Omega(n^2)$ and apply Schwartz–Zippel: if a matching exists, $\det \neq 0$ with high probability.

> **Exam-critical.** "Matching exists" ≡ "this determinant is a nonzero polynomial" ≡ "a random evaluation is nonzero." The fingerprint method decides a *combinatorial* property through a *numeric* check.

Making the minimum matching unique (toward the algorithm)

Existence is not enough if we want to **output** a matching — and in parallel we cannot just "pick one." So put random **weights** on edges and aim for a *unique* minimum matching. Replace $x_{ij} \mapsto \pm 2^{w_{ij}}$:

$$B(i,j) = \begin{cases} 2^{w_{ij}}, & A(i,j) = x_{ij}, \\ -2^{w_{ij}}, & A(i,j) = -x_{ij}, \\ 0, & A(i,j) = 0. \end{cases}$$

> **Theorem.** If G has a **unique** minimum-weight perfect matching M of weight W, then $\det(B) \neq 0$ and 2^{2W} is the **largest power of 2 dividing** $\det(B)$.

Each π contributes $\mathbf{val}(\pi) = \prod_i B(i, \pi(i))$ with $|\mathbf{val}(\pi)| = 2^{(\text{sum of edge weights used})}$. Odd cycles cancel. An even-cycle cover decomposes into two matchings M_1, M_2, contributing $2^{W(M_1)+W(M_2)}$. The special "doubled M" permutation (each matched edge as a 2-cycle) contributes 2^{2W}. Because M is the **unique** minimum, *every other* even permutation has $W(M_1) + W(M_2) > 2W$, so the term 2^{2W} is the **lowest** power of 2 and **cannot cancel** (it is alone at that level). Hence $v_2(\det B) = 2W$ exactly.

> **Theorem (read off the edges).** With M the unique minimum matching of weight W, $(i,j) \in M \iff \frac{\det(B_{ij})\, 2^{w_{ij}}}{2^{2W}}$ **is odd,** where B_{ij} is the minor deleting row i, column j.

The minor isolates permutations sending $i \mapsto j$; if $(i,j) \in M$, exactly one even cycle contributes at the minimal level 2^{2W} (odd ratio); if $(i,j) \notin M$, every contribution is $\geq 2^{2W+1}$ (even ratio). One determinant per edge tells you membership.

The Isolation Lemma — where the uniqueness comes from

But how do we *guarantee* a unique minimum matching? Random weights — and the reason is completely general, nothing to do with graphs:

> **Isolation Lemma.** Let $(X, \mathcal{F})$ be a set system, $X = \{x_1, \ldots, x_m\}$, $\mathcal{F} = \{S_1, \ldots, S_k\}$ with $S_i \subseteq X$, and weights $w(S) = \sum_{x \in S} w(x)$. If each $w(x_i) \in_R \{1, \ldots, 2m\}$ independently, then $\Pr[\exists \text{ a unique minimum-weight set in } \mathcal{F}] \geq \frac{1}{2}$.

(For matching: X = edges, $\mathcal{F}$ = perfect matchings. So random edge weights isolate a unique minimum perfect matching w.p. $\geq \frac{1}{2}$.)

Proof — the threshold trick. Reveal the weights one element at a time. Fix all weights except $w(x_i)$ and define the **threshold**
$\alpha_i = \big(\min \text{ weight of a set not containing } x_i\big) - \big(\min \text{ weight of a set containing } x_i, \text{ with } w(x_i) \text{ set to } 0\big)$.
Both terms are independent of $w(x_i)$. Now compare:

- if $w(x_i) < \alpha_i$: including x_i is *strictly* cheaper $\Rightarrow$ x_i is in **every** minimum-weight set;
- if $w(x_i) > \alpha_i$: excluding x_i is strictly cheaper $\Rightarrow$ x_i is in **no** minimum-weight set;
- if $w(x_i) = \alpha_i$: a tie — x_i is **ambiguous** ("uncertain").

The first two cases force x_i's membership; only the tie leaves it undecided. Since α_i does **not** depend on $w(x_i)$, $\Pr[x_i \text{ ambiguous}] = \Pr[w(x_i) = \alpha_i] \le \frac{1}{2m}$. Union bound over the m elements: $\Pr[\exists \text{ ambiguous element}] \le m \cdot \frac{1}{2m} = \frac{1}{2}$. If **no** element is ambiguous, every element's membership in the minimum set is forced — so the minimum-weight set is **unique**. Hence $\Pr[\text{unique minimum}] \ge \frac{1}{2}$. ■

> **The deep point.** Uniqueness can fail *only* if some element lands exactly on its own threshold — and that threshold was fixed *before* its weight was drawn, so it is a bullseye hit with probability $\le 1/2m$. The weights need only be drawn from a range of size $2m$ (linear in the ground set), independent of how many — possibly *exponentially* many — sets $\mathcal{F}$ contains. That is the magic: we tame an exponential family with $O(m)$-range random weights.

---

15. RNC perfect matching — the payoff

Put it together. **Perfect matching** ∈ **RNC** (randomized poly-log parallel time):

> **Input:** $G(V, E)$ containing a perfect matching.
>
> 1. For every edge e, choose a random weight $w(i,j) \in_R \{1, \dots, 2m\}$.
> 2. Build the weighted matrix B ($B(i,j) = \pm 2^{w(i,j)}$).
> 3. Compute $\det(B)$.
> 4. Find the largest W with $2^{2W} \mid \det(B)$ — the minimum matching weight.
> 5. Compute the adjugate $\operatorname{adj}(B) = \det(B) \cdot B^{-1}$ (all minors at once).
> 6. For every edge, $r(i,j) = \det(B_{ij})\, 2^{w(i,j)} / 2^{2W}$.
> 7. Put $(i,j) \in M$ iff $r(i,j)$ is **odd**.

By the Isolation Lemma, step 1 yields a **unique** minimum perfect matching w.p. $\ge \frac{1}{2}$; steps 3, 5 (determinant and adjugate) are in **NC** (parallelizable). The **only** randomness is the weights.

> **Punchline (Mulmuley–Vazirani–Vazirani).** The hard part of parallel matching is not *finding* a matching but **agreeing** on one — independent processors computing in parallel have no way to coordinate a choice among many matchings. The Isolation Lemma dissolves the problem: random weights make the minimum matching **unique**, so it becomes a *canonical* object every processor computes the same way, read off edge-by-edge from one determinant and its minors. Randomness here buys not speed but **coordination**.

Recurring themes from this part

Theme	Where it appeared
Fingerprint = small random projection of a big object	Freivalds (§6), Karp–Rabin (§9), hash functions (§12)
Equality → "is this polynomial $\equiv 0$?" → random evaluation	Schwartz–Zippel (§7), 1BP (§8), Tutte matching (§14)
One-sided error, then verify/amplify (Monte-Carlo → Las-Vegas)	fingerprints (§5), Karp–Rabin LV (§9), perfect hashing (§13)
Eliminate the adversary, hashing edition (random h from a family)	universal families (§12)
Birthday $\sqrt{n}$ threshold sets the table size	balls-in-boxes (§11) → FKS quadratic tables (§13)
Chernoff "double the mean is rare"	linear probing dangerous vertices (§11)
Combinatorial property decided by a numeric determinant	Tutte matrix (§14)
Randomness for *coordination*, not speed (make the answer unique)	Isolation Lemma → RNC matching (§14–§15)

The one sentence tying it together:

> **Stop comparing the elephants — compare their shadows.** A random projection (a vector, a prime, a field point, a hash) shrinks an intractable equality test to a cheap one with only one-sided error; pushed further, the same idea hashes data in $O(1)$, decides matching through a determinant, and — via the Isolation Lemma — *isolates* a single canonical solution that even

> parallel machines can agree on.