CS 170 — Lecture: Universal Hashing

1 Motivation: why randomize a hash function?

A hash table stores n keys drawn from a large universe $\mathcal{U} = \{0, 1, \ldots, U-1\}$ in an array of m buckets, using some function $h : \mathcal{U} \to [m] \coloneqq \{0, 1, \ldots, m-1\}$ to decide where each key lives. If two keys land in the same bucket, we resolve the collision by chaining. The cost of an INSERT, LOOKUP, or DELETE for a key x is proportional to the length of the chain at bucket $h(x)$.

For any *fixed* hash function h, as long as $U > m \cdot n$ there is always a set of n keys that all collide at a single bucket. (By pigeonhole, some bucket has at least $U/m > n$ preimages; pick n of them.) So for any deterministic h, an adversary who knows the code can submit an input on which lookups take $\Theta(n)$ time. This is bad: sorted arrays already give $O(\log n)$ lookups without randomness.

The way out is to pick h *at random* from a carefully designed family $\mathcal{H}$. The adversary is allowed to know $\mathcal{H}$, but not our particular choice of $h \in \mathcal{H}$. What property of $\mathcal{H}$ do we need so that *no matter what input the adversary picks*, chains are short in expectation?

2 Universal and pairwise-independent hash families

We always think of h as a random variable drawn uniformly from a finite family $\mathcal{H}$ of functions $\mathcal{U} \to [m]$, independently of the input.

Definition 1 (Universal family)**.** A family $\mathcal{H}$ of functions $\mathcal{U} \to [m]$ is *universal* (or 2-universal) if for every pair of distinct keys $x \neq y \in \mathcal{U}$,

$$\mathbb{P}_{h \sim \mathcal{H}}\big[h(x) = h(y)\big] \;\leq\; \frac{1}{m}.$$

So a random function from a truly random family would have collision probability exactly $1/m$; being "universal" means no worse than that. For our streaming application next lecture we will need a slightly stronger property.

Definition 2 (Pairwise-independent / 2-wise independent family)**.** A family $\mathcal{H}$ of functions $\mathcal{U} \to [m]$ is *pairwise independent* if for every $x \neq y \in \mathcal{U}$ and every $a, b \in [m]$,

$$\mathbb{P}_{h \sim \mathcal{H}}\big[h(x) = a \text{ and } h(y) = b\big] \;=\; \frac{1}{m^2}.$$

Equivalently, the random variables $h(x)$ and $h(y)$ are each uniform on $[m]$ and are independent of each other (but not necessarily independent across three or more keys).

Remark 3. Pairwise independence implies universality: summing $\mathbb{P}[h(x) = a \text{ and } h(y) = a]$ over all $a \in [m]$ gives $m \cdot 1/m^2 = 1/m$. The converse does not hold in general.

We will often just say "pick a random h from a pairwise-independent family" and use both properties interchangeably.

3 Application: hash tables with short chains

Fix any sequence $x_1, \ldots, x_n$ of *distinct* keys. (Duplicates are trivial: they all end up in the same bucket, which is actually what you want.) Pick h from a universal family.

Theorem 4. *For any fixed query key x (whether or not x is among $x_1, \ldots, x_n$), the expected number of stored keys hashing to the same bucket as x is at most n/m.*

Proof. Let $C_i = \mathbf{1}[h(x_i) = h(x)]$, the indicator that x_i collides with the query x. The quantity we care about is $C = \sum_{i : x_i \neq x} C_i$.

Case 1: $x \notin \{x_1, \ldots, x_n\}$. Then every x_i is distinct from x, so by universality $\mathbb{E}[C_i] = \mathbb{P}[h(x_i) = h(x)] \leq 1/m$. Linearity of expectation gives $\mathbb{E}[C] \leq n/m$.

Case 2: $x = x_j$ for some j. Then $\mathbb{E}[C] = \sum_{i \neq j} \mathbb{P}[h(x_i) = h(x)] \leq (n-1)/m < n/m$.

Either way, $\mathbb{E}[C] \leq n/m$. □

Corollary 5. *If we size the table so that $m \geq n$ (load factor ≤ 1), then every* Insert/Lookup/Delete *takes $O(1)$ expected time.*

The key point: **the expectation is over the random choice of h, not over any assumption about the input.** The adversary can pick the keys; we pick the hash function.

Remark 6 (What "expectation" means here)**.** The guarantee $\mathbb{E}[C] \leq n/m$ is a statement about our expected running time on any worst-case input. It is not a concentration bound. A tail bound saying "with high probability *every* bucket is short" needs more work (Markov on C gives one direction but is weak; one can upgrade with the second moment, which is exactly what pairwise independence buys). For this lecture, linearity of expectation is all we need.

4 A concrete construction: multiply–add mod prime

We want a small family $\mathcal{H}$ so that we can store "which h we chose" cheaply, and so that evaluating $h(x)$ is fast.

Pick any prime $p \geq U$. Identify $\mathcal{U}$ with $\{0, 1, \ldots, p-1\} = \mathbb{Z}/p\mathbb{Z} = \mathbb{F}_p$. For each pair $(a, b) \in \mathbb{F}_p \times \mathbb{F}_p$ with $a \neq 0$, define

$$h_{a,b}(x) \;=\; \big((ax + b) \bmod p\big) \bmod m.$$

The family is $\mathcal{H} = \{h_{a,b} : a \in \{1, \ldots, p-1\}, b \in \{0, \ldots, p-1\}\}$, so $|\mathcal{H}| = p(p-1)$. Picking $h \in \mathcal{H}$ uniformly at random means picking a, b uniformly at random (with $a \neq 0$) and using $h_{a,b}$.

Theorem 7. *The family $\mathcal{H}$ above is pairwise independent when restricted to outputs in $\mathbb{F}_p$. Consequently, it is universal as a family $\mathcal{U} \to [m]$.*

We will prove the cleaner claim (pairwise independence in $\mathbb{F}_p$) first, and then extract universality on $[m]$.

Lemma 8. *For every pair of distinct keys $x \neq y \in \mathbb{F}_p$ and every pair of target values $(s,t) \in \mathbb{F}_p \times \mathbb{F}_p$, there is* exactly one *pair $(a,b) \in \mathbb{F}_p \times \mathbb{F}_p$ such that*

$$ax + b \equiv s \pmod{p} \quad \textit{and} \quad ay + b \equiv t \pmod{p}.$$

Proof. Subtract the two equations: $a(x-y) \equiv s - t \pmod{p}$. Since p is prime and $x \neq y$, the element $x - y$ is nonzero in $\mathbb{F}_p$ and therefore invertible, so

$$a \;\equiv\; (s-t)(x-y)^{-1} \pmod{p}.$$

Then $b \equiv s - ax \pmod{p}$ is also uniquely determined. So the system has a unique solution in $\mathbb{F}_p \times \mathbb{F}_p$. □

Proof of Theorem **??**. Fix $x \neq y \in \mathbb{F}_p$ and targets $s, t \in \mathbb{F}_p$. By Lemma **??**, the event $\{ax + b \equiv s,\ ay + b \equiv t\}$ happens for exactly one $(a,b) \in \mathbb{F}_p \times \mathbb{F}_p$. We draw (a,b) uniformly from a slightly smaller set — we exclude $a = 0$, leaving $p(p-1)$ possibilities — so

$$\mathbb{P}\big[ax + b = s \wedge ay + b = t\big] \;=\; \frac{\#\{(a,b) \in \{1,\dots,p-1\} \times \mathbb{F}_p : \dots\}}{p(p-1)}.$$

The unique solution from the lemma lies in the allowed set unless it has $a = 0$. But $a = 0$ forces $s = t$ (both equal b), so as long as $s \neq t$ the solution is valid, and

$$\mathbb{P}[ax + b = s \wedge ay + b = t] \;=\; \frac{1}{p(p-1)} \;\approx\; \frac{1}{p^2} \qquad (s \neq t).$$

This is the statement that the pair $(ax + b, ay + b)$ is uniformly distributed on $\{(s,t) \in \mathbb{F}_p^2 : s \neq t\}$. (The tiny deviation from $1/p^2$ comes from excluding $a = 0$; it is a lower-order issue and is often avoided by allowing $a = 0$ and accepting a slightly weaker universality bound.)

Now map into $[m]$: $h_{a,b}(x) = (ax + b \bmod p) \bmod m$. For any $x \neq y$,

$$\begin{aligned}\mathbb{P}[h_{a,b}(x) = h_{a,b}(y)] &= \sum_{s \neq t \in \mathbb{F}_p,\ s \equiv t \pmod{m}} \mathbb{P}[ax + b = s \wedge ay + b = t] \\ &\leq \sum_{s \in \mathbb{F}_p} \cdot(\#\ t \neq s \text{ with } t \equiv s \bmod m) \cdot \frac{1}{p(p-1)}.\end{aligned}$$

The number of $t \in \mathbb{F}_p$ congruent to a given s modulo m is at most $\lceil p/m \rceil \leq (p + m - 1)/m$, and subtracting 1 for $t = s$ gives at most $(p-1)/m$. So

$$\mathbb{P}[h(x) = h(y)] \;\leq\; p \cdot \frac{p-1}{m} \cdot \frac{1}{p(p-1)} \;=\; \frac{1}{m}.$$

This is universality. □

Remark 9 (What we actually used)**.** The algebraic heart of the proof is Lemma **??**: the map $(a,b) \mapsto (ax + b, ay + b)$ is a *bijection* on $\mathbb{F}_p^2$ whenever $x \neq y$. Pairwise independence in $\mathbb{F}_p$ is an immediate rewrite of that fact in probabilistic language.

5 A second application: Rabin–Karp string matching

So far we have used hashing to index into a table. Let me show you a completely different use: *comparing* objects quickly. The algorithm is Rabin–Karp string matching (1987), and it runs in expected linear time.

The problem. Given a text $T \in \Sigma^n$ and a pattern $P \in \Sigma^m$, decide whether P appears as a contiguous substring of T, and if so at what position. (Write T_k for the length-m window $T[k\,..\,k+m-1]$; we want to find k with $T_k = P$.) The naive algorithm compares P against T_k character by character for each of the $n-m+1$ starting positions, taking $O(nm)$ time in the worst case. We will do it in $O(n+m)$ expected time.

Idea: replace strings by fingerprints. If we had a "fingerprint" function $h : \Sigma^m \to \mathbb{F}_p$ that we could compute cheaply, we could compare fingerprints instead of strings: the comparison $h(T_k) \stackrel{?}{=} h(P)$ is a single arithmetic operation. Two dangers:

1. We need to compute $h(T_k)$ for all $n-m+1$ windows quickly, not in $\Theta(m)$ time each (otherwise we are back to $\Theta(nm)$).
2. A false positive $h(T_k) = h(P)$ with $T_k \neq P$ forces us to do a full character check and pay $\Theta(m)$; we need to be sure false positives are rare.

Both will be handled by picking the right hash family.

A polynomial rolling hash. Identify Σ with $\{0, 1, \ldots, p-1\} = \mathbb{F}_p$ for a prime p we will choose below. Read a length-m string $s_0 s_1 \cdots s_{m-1}$ as the polynomial $s_0 x^{m-1} + s_1 x^{m-2} + \cdots + s_{m-1}$, and define, for a random $a \in \mathbb{F}_p$,

$$h_a(s_0, \ldots, s_{m-1}) \;=\; s_0 a^{m-1} + s_1 a^{m-2} + \cdots + s_{m-1} \pmod p.$$

Two useful facts:

Fact 1 (rolling update). From $h_a(T_k)$ we can compute $h_a(T_{k+1})$ in $O(1)$ arithmetic operations:

$$h_a(T_{k+1}) \;=\; a \cdot \big(h_a(T_k) - T[k] \cdot a^{m-1}\big) + T[k+m] \pmod p.$$

(Pre-compute $a^{m-1} \bmod p$ once.) So sliding the window across T costs $O(n)$ total.

Fact 2 (collision bound). For any two *distinct* length-m strings $S \neq P$,

$$\mathbb{P}_{a \in \mathbb{F}_p}\big[h_a(S) = h_a(P)\big] \;\leq\; \frac{m-1}{p}.$$

Why: $h_a(S) - h_a(P) = Q(a) \bmod p$, where $Q(x)$ is the nonzero polynomial $S(x) - P(x)$ (nonzero because $S \neq P$, viewing strings as coefficient sequences) of degree at most $m-1$. A nonzero polynomial of degree d has at most d roots in $\mathbb{F}_p$. So the set of "bad" a's has size $\leq m-1$, out of p.

This is the same idea as universal hashing: an adversary might choose S, P with all sorts of structure, but the random a is chosen *after* the adversary commits, and the probability we hit a root is $\leq (m-1)/p$.

The algorithm.

1. Pick a prime p with $p \geq nm^2$ (or any $p \geq Cnm$ for a constant C), and a random $a \in \mathbb{F}_p$.
2. Compute $h_a(P)$ and $h_a(T_0)$ in $O(m)$ time.
3. For $k = 0, 1, \ldots, n-m$: if $h_a(T_k) = h_a(P)$, do a full character-by-character comparison of T_k with P; report any match. Then update $h_a(T_{k+1})$ from $h_a(T_k)$ using Fact 1.

Running-time analysis. Let F be the number of *false positives*: indices k where $h_a(T_k) = h_a(P)$ but $T_k \neq P$. The total running time is

$$O(m) \;+\; O(n) \;+\; O(m) \cdot (\#\text{ real matches} + F),$$

where the three terms are the initial hash, the rolling updates, and the full-string verifications. The number of real matches is at most $n - m + 1$, but on any input where the pattern does not occur excessively often this is small; in any case what we want is a bound on F.

By Fact 2 and linearity of expectation,

$$\mathbb{E}[F] \;\leq \sum_{k:T_k \neq P} \mathbb{P}[h_a(T_k) = h_a(P)] \;\leq\; (n-m+1) \cdot \frac{m-1}{p} \;\leq\; \frac{nm}{p}.$$

With $p \geq nm^2$ this gives $\mathbb{E}[F] \leq 1/m$, so the expected work spent chasing false positives is $O(m) \cdot \mathbb{E}[F] = O(1)$. Total expected running time: $O(n+m)$.

Remark 10 (Same moral, different hash family)**.** The hash family here is not exactly the $(ax + b) \bmod p$ family we constructed above — it is a polynomial evaluation at a random point — but the flavor is identical. Both are "random polynomial at a random point," and both exploit the fact that a nonzero polynomial of low degree has few roots in $\mathbb{F}_p$. The hash-table analysis uses degree-1 polynomials; Rabin–Karp uses degree-$(m-1)$ polynomials. The take-away is more about the *strategy* than the specific family: design your randomness so that the worst case for the adversary corresponds to a rare algebraic coincidence.

6 Summary and what's next

A *universal* family gives $\mathbb{P}[h(x) = h(y)] \leq 1/m$ for distinct x, y. A *pairwise-independent* family makes $h(x)$ and $h(y)$ jointly uniform on $[m]^2$ for distinct x, y, which is strictly stronger and will be what we need next lecture.

A universal family is all we need to make hash tables work in $O(1)$ expected time against a worst-case input, via a two-line linearity-of- expectation argument. And a concrete universal and pairwise-independent family is cheap: one prime, two random elements of $\mathbb{F}_p$, one multiplication and one addition to evaluate.

The same philosophy powered Rabin–Karp: we replaced a $\Theta(nm)$ worst-case string-matching algorithm with a $O(n+m)$ expected-time algorithm by hashing length-m strings down to a single element of $\mathbb{F}_p$, with a random "polynomial at a random point" hash whose collisions are controlled by the fact that a nonzero polynomial of degree $m-1$ has at most $m-1$ roots.

Next lecture we take the same hash family and use it somewhere completely different — a *streaming* algorithm that reads a torrent of data in one pass and estimates the number of distinct elements using only a handful of words of memory. The expectation will be exactly over the same randomness we introduced today.

Acknowledgements: these notes follow the standard CS 170 presentation (cf. Dasgupta–Papadimitriou–Vazirani and past CS 170 lecture notes).