Hashing (3)
Perfect Hashing

We have \( n \) fixed keys \( a_1, \ldots, a_n \)

We want to be able to build a table with these keys, so that lookups take constant time — in the worst case

Basic strategy: universal hashing

\( m = \# \) slots

We don’t want any collisions
Union bound:

\[
\Pr[\text{collision}] \leq \sum_{i=1}^{n} \sum_{j=1}^{i-1} \Pr[h_R(a_i) = h_R(a_j)] \\
\leq \frac{n(n-1)}{2m}
\]

Assume \( m \geq n(n-1) \), so that we get a collision with probability \( \leq \frac{1}{2} \)

Strategy:

repeat
  choose a random hash function
  hash \( a_1, \ldots, a_n \) using this hash function
until no collisions
Good news: each iteration succeeds with probability \( \geq \frac{1}{2} \)
\[\because \text{expected \# of iterations} \leq 2\]

Bad news: *HUGE* table (mostly empty)

A better approach: two levels of universal hashing

- Level 1 segregates keys so that not too many go into any one slot
- Level 2 applies the basic strategy to each Level-1 slot
Suppose there are $m \geq 2n$ Level-1 slots

Step 1:

repeat
  choose a random hash function hash $a_1, \ldots, a_n$ using this function
  let $L_s := \# \text{ keys in slot } s$
  let $V' := \sum_s L_s(L_s - 1) = \sum_s L_s^2 - n$
  until $V' \leq n$

Step 2:

For each Level-1 slot $s$, use Basic Strategy to hash all keys in slot $s$ into a hash table with (at least) $L_s(L_s - 1)$ slots
Analysis

Tool: Markov’s inequality

let $X$ be a random variable taking non-negative values
let $\mu := \mathbb{E}[X]$
For all $t > 0$: $\Pr[X \geq t] \leq \mu/t$
Set $t = 2\mu$: $\Pr[X \geq 2\mu] \leq 1/2$

Step 1:

Previous lecture (Hashing (1)): $\mathbb{E}[V'] \leq n^2/m \leq n/2$
Markov says: $\Pr[V' \geq n] \leq 1/2$
$\therefore$ expected # of iterations $\leq 2$
Analysis (cont’d)

Step 2:

For each slot $s$, we build a sub-table with (at least) $L_s(L_s - 1)$ slots

$\therefore$ we can quickly find a good hash function for this sub-table

Summary:

- Total expected running time $= O(n)$
- Total size of data structure $= O(n)$
Another hash application: fast pattern matching

**Problem:** Given strings $a = a_1 \cdots a_n$, and $b = b_1 \cdots b_t$, test if $b$ is a substring of $a$

**Naive algorithm:** time $O(nt)$

**Faster algorithms:** time $O(n)$ (assume $t \leq n$)

- A simple, randomized algorithm (Karp, Rabin)
- A trickier deterministic algorithm (Knuth, Morris, Pratt)
The Karp/Rabin Algorithm

Let \( \{h_{\lambda}\}_{\lambda \in \Lambda} \) be an \( \epsilon \)-universal family of hash functions on strings of length \( t \)

Algorithm:

- choose a random hash function index \( \lambda \)
- \( s \leftarrow h_{\lambda}(b) \)
- for \( i \leftarrow 1 \) to \( n - t + 1 \) do
  - \( s_i \leftarrow h_{\lambda}(a_i \cdots a_{i+t-1}) \)
  - if \( s = s_i \) then
    - if \( b = a_i \cdots a_{i+t-1} \) then
      return “match”
  - return “no match”
Running time analysis: two factors

- time to compute hash function
- expected time spent processing “false positives”: \(O(\varepsilon \cdot n \cdot t)\)

Use “polynomial evaluation” hash:

- view \(a_i’s, b_j’s, \lambda\) as elements of \(\mathbb{Z}_m\), where \(m\) is prime
- \(h_\lambda(a_1 \cdots a_t) = a_1 \lambda^{t-1} + \cdots + a_{t-1}\lambda + a_t\)
- \(\varepsilon < t/m\)
- time to evaluate each \(h_\lambda\): \(O(t)\) naively, but we can do better
Horner’s rule for polynomial evaluation

Given a polynomial \( f = a_1x^{t-1} + \cdots + xa_{t-1} + a_t \) and a value \( \lambda \), compute the value \( f(\lambda) \):

\[
\text{acc} \leftarrow a_1 \\
\text{for } i \text{ in } [2..t] \text{ do} \\
\quad \text{acc} \leftarrow \text{acc} \cdot \lambda + a_i
\]

How it works:

\[
\text{acc} = a_1 \\
\text{acc} = a_1 \lambda + a_2 \\
\text{acc} = a_1 \lambda^2 + a_2 \lambda + a_3 \\
\text{acc} = a_1 \lambda^3 + a_2 \lambda^2 + a_3 \lambda + a_4 \\
\ldots
\]
Computing a “Rolling Hash”

\[ a_1 \lambda^{t-1} + a_2 \lambda^{t-2} + \cdots + a_t \]

\[-a_1 \lambda^{t-1} \]

\[ \frac{a_2 \lambda^{t-2} + \cdots + a_t}{\lambda} \]

\[ a_2 \lambda^{t-1} + \cdots + a_t \lambda + a_{t+1} \]
Karp/Rabin: conclusions

Assume $m$ is near machine word size (e.g., $2^{64}$)

Assume arithmetic in $\mathbb{Z}_m$ takes time $O(1)$

Time to compute hashes: $O(n)$

Expected time to process false positives: $O(nt^2/m)$, which is $O(n)$ for “reasonable” $t$ (e.g., $t < 2^{32}$)

Karp/Rabin: not the fastest, but for multi-pattern matching, it is very good

Can be adapted to 2D matching (exercise)
Beyond Pairwise independence: Uniform Hashing Assumption

Let $\mathcal{H} = \{h_\lambda\}_{\lambda \in \Lambda}$ be a family of hash functions from $\mathcal{U}$ to $\mathcal{V}$

Let $R$ be uniformly distributed over $\Lambda$

**Uniform Hashing Assumption:**

The random variables $h_R(\alpha)$ for $\alpha \in \mathcal{U}$ are mutually independent, with each $h_R(\alpha)$ uniformly distributed over $\mathcal{V}$
A very strong assumption
Hard to achieve in practice
Often the assumption is just heuristically applied
  “off the shelf” cryptographic functions
The Max Load — Revisited

Suppose we hash \( n \) keys into \( n \) slots

Let \( M = \max \# \) of keys that hash to any one slot

**Theorem.** Under the Uniform Hashing Assumption,

\[
E[M] = O\left(\frac{\log n}{\log \log n}\right).
\]

*Note: compare to \( O(\sqrt{n}) \) for pairwise independent hashing*

“Balls and bins” interpretation: if you throw \( n \) balls into \( n \) bins, the expected value of the max number of balls in any bin is \( O(\log n / \log \log n) \)
Recall tail sum formula:

If $X$ be a random variable that takes only non-negative integer values, then

$$E[X] = \sum_{j \geq 1} \Pr[X \geq j]$$

Proof of Theorem.

**Claim 1:** for $j = 1, \ldots, n$: $\Pr[M \geq j] \leq n/j!$

Proof: We are hashing $n$ keys

$M \geq j \iff$ some subset of $j$ keys hash to same slot

For any fixed subset of $j$ keys, this happens with probability $1/n^{j-1}$:

- the first key can hash into any slot
- the other $j-1$ must hash into the same slot
Apply Union Bound, sum over all subsets of size $j$:

$$\Pr[M \geq j] \leq \binom{n}{j} \cdot \frac{1}{n^{j-1}}$$

$$= \frac{n(n-1)\cdots(n-j+1)}{j!} \cdot \frac{1}{n^{j-1}}$$

$$\leq \frac{n}{j!}$$

That proves the claim
Define \( f(n) := \) least \( j \) such that \( j! \geq n \)

[“inverse factorial” function]

**Claim 2:** \( f(n) = O(\log n / \log \log n) \)

Proof: we want \( j! \geq n \), or equivalently, \( \ln(j!) \geq \ln(n) \)

We know

\[
\ln(j!) = \sum_{i=1}^{j} \ln(i) = j \ln(j) + O(j) \geq \frac{1}{2} j \ln(j)
\]

for sufficiently large \( j \)

Suppose \( j \geq 4 \ln(n) / \ln(\ln(n)) \)

For \( n \) sufficiently large:

\[
\ln(j) \geq \ln(\ln(n))/2
\]

\[
j \ln(j) \geq 2 \ln(n)
\]

\[
\ln(j!) \geq \frac{1}{2} j \ln(j) \geq \ln(n)
\]
Finishing the proof . . .

If \( j_0 := f(n) \), we have

\[
E[M] = \sum_{j \geq 1} \Pr[M \geq j] = \sum_{j \leq j_0} \Pr[M \geq j] + \sum_{j > j_0} \Pr[M \geq j] \\
\leq \sum_{j \leq j_0} 1 + \sum_{j > j_0} \frac{n}{j!} = j_0 + \frac{n}{j_0!} \sum_{j > j_0} \frac{j_0!}{j!} \\
\leq j_0 + \sum_{i \geq 1} 1/2^i \\
= j_0 + 1 = O(\log n / \log \log \log n) \quad \text{QED}
\]