CS 7800: Advanced Algorithms Spring 2026

Lecture 10 — February 11, 2026

Prof. Prashant Pandey *Scribe: Alexandria Stevenson*

1 Hashing

> The three most important data structures: *hashing, hashing,* and *hashing.*
> — Udi Mauber

Main types of hashing:

1. Universal hashing
2. Independent K-wise hashing
3. Chaining
4. Perfect hashing (*not covered in today's lecture*)

1.1 The Hash Function

Hash function h takes keys from universe U:

$$\underbrace{\{0, 1, 2, 3, \ldots, U-1\}}_{\text{Universe } U \text{ of keys}} \longrightarrow \underbrace{\{0, 1, 2, 3, \ldots, m-1\}}_{\text{Table indices with table size } m}$$

1.2 Totally Random Hash Function

$$\mathbb{P}(h(x) = t) = \frac{1}{m}$$

- Independent of $h(y)$ for all $x \neq y \in U$
- Any key from U lands uniformly at random (independent from past and future insertion events)
- n keys, m buckets $\rightarrow$ expected number of keys in each bucket $= \frac{n}{m}$
- Maximum load in any bucket: $\Theta(\frac{\lg n}{\lg \lg n})$

Space needed for totally random hash function:

$$\Theta(U \cdot \lg m)$$

However:

- This is **ideal** but not practical; in practice, can deviate
  - Example: if $U = 2^{64}$, this would require too much storage
- To guarantee bound: need to record in table
  - For every item, need to identify bucket ($\lg m$ bits)

1.3 Universe Hashing [Carter and Wegman JCSS 1979]

$$\mathop{\mathbb{P}}_{h \in H}[\, h(x) = h(y)\,] = O\left(\frac{1}{m}\right) \quad \text{for all } x \neq y \in U$$

Guarantee coming from technique where we have family of hash functions and pick uniformly from family.

1.3.1 Example 1

$$h(x) = [ax \bmod p] \bmod m, \quad 0 < a < p,\ p > |U|$$
$$\text{where } a \text{ and } p \text{ are constants, and } p \text{ is prime.}$$

For family of hash functions: pick randomly and uniformly, guarantees property of universe hashing. Note: a is picked randomly and uniformly.

1.3.2 Example 2: Multiple Shift

$$h(x) = (ax) \gg (\lg U - \lg M)$$

Equivalent but more practical way to implement **Example 1** (bit shifts are cheaper than modulo operations). M is usually a power of 2.

1.4 K-Wise Independent [Wegman and Carter JCSS 1981]

$$\mathop{\mathbb{P}}_{h \in H}(h(x_1) = t_1 \text{ and } h(x_2) = t_2 \ldots h(x_k) = t_k) = O(\frac{1}{m^k})$$
$$\text{true for all distinct } x_1, x_2, \ldots, x_k \in U \text{ and all } t_1, t_2, \ldots, t_k \in [0, m]$$
$$(t_1, t_2, \text{ ... does not necessarily have to be distinct})$$

If $k = 2$: **Pairwise Independent**

$$\mathop{\mathbb{P}}_{h \in H}(h(x_1) = t_1 \text{ and } h(x_2) = t_2) = O(\frac{1}{m^2})$$
$$\text{for all } x_1 \neq x_2 \in U$$

Used most in practice.

How to build pairwise independent hash function: i.e. distributing item into cluster space

$$h(x) = [(ax + b) \mod p] \mod m$$
$$\text{for } 0 < a < p \text{ and } 0 \leq b < p$$

How to build K-Wise independent hash function:

$$h(x) = \left[(\sum_{i=0}^{k-1} a_i x^i) \mod p \right] \mod m$$
$$\text{for } 0 < a_{k-1} < p \text{ and } 0 \leq a_i < p \text{ for } i < k-1$$

Why is pairwise independent hashing stronger than universal hashing?

Intuition: Introduces more randomness.

- In Universe Hashing: collision from wraparound during $\mod m$ operation
- The $+b$ in pairwise hashing addresses scaling in randomness more
- *Side note:* When actually implementing, use SHIFT not MOD (due to slower speed)

1.5 Tabulation Hashing

- View vector x as vector of characters: $x_1, x_2, x_3, \ldots, x_c$
- Create totally random hashtable T_i of precomputed values on each character
- Require $O(c * U^{1/c})$ words of space

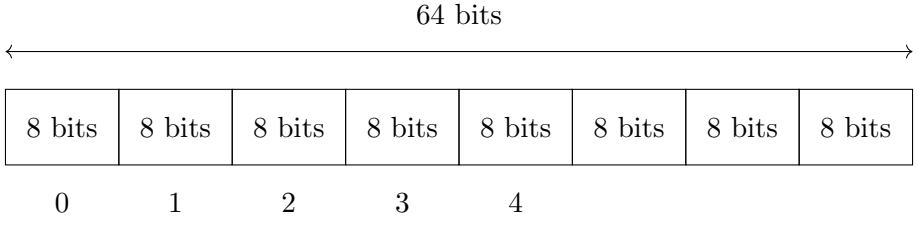
Hash Function:

$$h(x) = T_1(x_1) \oplus T_2(x_2) \oplus T_3(x_3) \oplus \cdots \oplus T_c(x_c)$$

Properties:

- Complexity: $O(c)$ time to compute
- 3-wise Independent

More efficient than pairwise but requires more space.

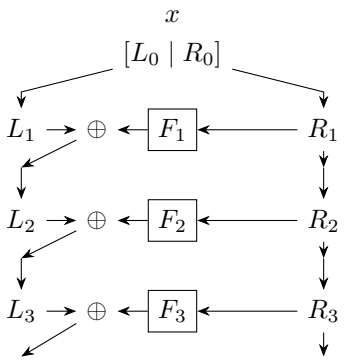


1.6 Converting Non-Invertible to Invertible Hash Functions

All of the hash functions mentioned so far have been one-way (not invertible).

To be invertible: size $m \geq U$ (i.e. no collisions must occur)

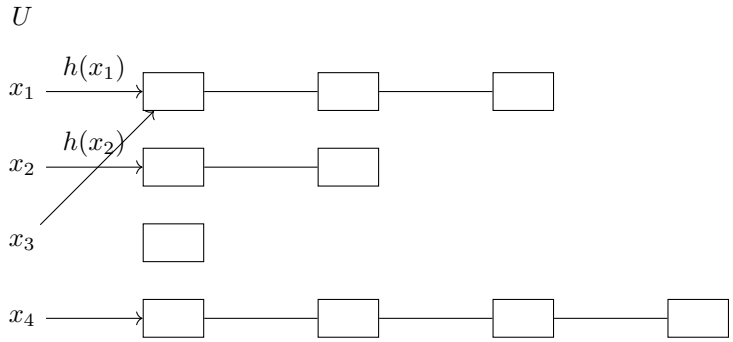
1.6.1 Feistel Network [Luby Rockoff Block Cipher]



Take any strong hash function and use to make a strong cipher.

1.7 Chaining Hash Table

Another other hash table we've discussed is called **open addressing**.



n items, m buckets Expected items per bucket: $O(\frac{n}{m})$

Expected Constant Time Complexity of Hash Tables

Are these bounds accurate for chaining?
Policy: Resize when $n = m$, but when $n < m$, can you guarantee $O(1)$?

- $O(1)$ on average but in worst case analysis, many not be
- If $m = \Omega(n)$, $O(1)$

1.7.1 Variance in Size of Buckets

Standard does not guarantee $O(1)$ in variance

Expected Chain Length

$$E[c_t = \text{length of chain } t = \sum_i \mathbb{P}(h(x_i) = t)$$

Suppose we use universal hash function:

$$E[c_t] = \sum_n (\frac{1}{m}) = O(\frac{n}{m})$$

$$m = \Omega(n)$$

$$= O(1)$$

Variance of Chain Lengths

$$\text{var}[c_t] = E[c_t^2] - E[c_t]^2$$

$$E[c_t^2] = \frac{1}{m} \sum E[c_t^2] = \frac{1}{m} \sum_{i \neq j} \underbrace{\mathbb{P}(h(x_i) = h(x_j))}_{\text{Probability that 2 distinct items will hit same bucket}}$$

Universe hash:

$$E[c_t^2] \leq \frac{1}{m} n^2 O(\frac{1}{m}) = O(\frac{n^2}{m^2}) = O(1)$$

$$m = \Omega(n)$$

Summary

- Low variance reduces probability that chain lengths deviate from average
- All chain lengths will be of similar size

References

[1] J. Lawrence Carter, Mark N. Wegman. Universal Classes of Hash Functions. *J. Comput. Syst. Sci.*, 18(2):143–154, 1979.

[2] J. Lawrence Carter, Mark N. Wegman. New Hash Functions and Their Use in Authentication and Set Equality. *J. Comput. Syst. Sci.*, 22(3):265–279, 1981.