CMPUT 498/501: Advanced Algorithms **Fall 2025**

Lecture 9 (Oct 1, 2025): Hashing

Lecturer: Mohammad R. Salavatipour *Scribe: Abel Romer, Baxter Madore*

9.1 Introduction

Hashing is a very important technique used in the efficient implementation of dictionaries, cryptography, data streaming, complexity theory and other contexts that require data storage and fast look-ups. A hash function allows data to be mapped into and retrieved from a table in constant time, and often relies on randomness to minimize the probability of collisions (mapping keys to the same value).

We define U to be the universe of possible keys, and assume $|U| = m$ to be very large. Let $S \subset U$ be a set of keys mapped into our hash table, and assume $|S|$ is much smaller than m. We want to support the following operations on S:

- MakeSet(S): create a hash table for S;
- Insert(i, S): add item i to S;
- Delete(i, S): remove item i from S;
- Find(i, S): find item i in S.

By storing our keys in a binary search tree, we can perform these operations in $O(\log n)$ time. Assuming we can perform arithmetic operations in $O(1)$, we will aim to improve on this runtime with a hashing function and table.

Definition 1 *Let $T[1...n]$ be a table and $h : U \to [n]$ be a hash function mapping $x \in U$ to index $h(x)$ in T.*

We define a *collision* to occur if for two different keys $x, y \in S$, $h(x) = h(y)$. Clearly, when designing a hash function h, we aim to minimize the number of collisions of elements in S. In particular, if h guarantees no collisions in S, we say that h is a *perfect hash function* for S.

We can always find a perfect hash function for any given set S, but it is impossible to find a perfect hash function for all S in U simultaneously, if $|S| < m$.

9.2 Hash functions

How do we define hash functions that minimize collisions? One idea is to randomly map elements of S to $[n]$. This is equivalent to throwing n balls into n bins, uniformly at random. From the previous lecture, we know the probability of a collision occurring in a particular bin to be $\frac{1}{n}$, and the maximum number of collisions in a given bucket to be

$$\Theta\left(\frac{\ln n}{\ln \ln n}\right).$$

We define a set of hash functions with at most a $\frac{1}{n}$ probability of collision as a *universal hash family*. Formally:

Definition 2 *A family $\mathcal{H}$ of hash functions $h : U \to T$ is universal if for all $x, y \in U$, where $x \neq y$, and for h chosen uniformly at random from $\mathcal{H}$:*

$$\Pr[h(x) = h(y)] \leq \frac{1}{n}.$$

This is a useful definition, because in practice it is infeasible to select a hash function $h : U \to [n]$ uniformly at random from the complete set of $n^{|U|}$ possible hash functions. A uniform hash family allows us to select h from a far more manageable subset $\mathcal{H}$. Furthermore, most constructions of $\mathcal{H}$ imply a stronger property, called *2-universal hashing.*

Definition 3 *Let $\mathcal{H}$ be a family of hash functions $h : U \to T$. We say that $\mathcal{H}$ is 2-universal, if for all $x, y \in U, x \neq y$ and all $n_1, n_2 \in [n]$:*

$$\Pr[h(x) = n_1 \wedge h(y) = n_2] = \frac{1}{n^2}.$$

Essentially, this strengthens our definition of universality by requiring that for any $h \in \mathcal{H}$, the values $h(x)$ and $h(y)$ are independent.

A natural follow-up question is whether we can define a family of hash functions with a lower probability of collisions. It turns out that we cannot improve the bound of $\frac{1}{n^2}$ by much.

Lemma 1 *For $U = [m]$ and any family $\mathcal{H}$ of hash functions $h : U \to T$, there exist $x, y \in U$ such that*

$$\Pr_{h \in \mathcal{H}}[h(x) = h(y)] \geq \frac{1}{n} - \frac{1}{m}.$$

We can generalize the pairwise independence of 2-universal families to k-wise independence:

Definition 4 *Let $\mathcal{H}$ be a family of hash functions $h : U \to [n]$. We say that $\mathcal{H}$ is k-wise independent, if for any k distinct keys $x_1, ..., x_k \in U$ and any k distinct values $\alpha_1, ..., \alpha_k \in [n]$:*

$$\Pr_{h \in \mathcal{H}}[h(x_1) = \alpha_1 \wedge ... \wedge h(x_k) = \alpha_k] \leq \frac{1}{n^k}.$$

Theorem 1 *Consider any sequence of operations with at most s inserts using a hash function $h : U \to [n]$ drawn uniformly at random from a universal family $\mathcal{H}$. Then the expected cost of each operation is at most $1 + \frac{s}{n}$.*

Proof. Consider the operation of inserting some $x \in U$. If there are z elements in $h(x)$, then the cost of this operation is $z + 1$. We bound $\mathbb{E}[z]$. For each item $y \in s$, define $z_y = 1$ if and only if $y \in h(x)$. Then:

$$\mathbb{E}[z] = \sum_{y \in s} \mathbb{E}[z_y] \leq s \cdot \Pr[h(x) = h(y)] \leq \frac{s}{n}.$$

■

9.3 Constructing a Universal Family

Clearly, it is useful to have a universal family of hash questions, so the natural question is how we might construct one. We present a simple construction below.

Suppose that we have a universe U with size $|U| = 2^u$ and a table of size $n = 2^m$. Consider a $u \times m$ matrix A with entries chosen from $\{0,1\}$ uniformly at random. For each $x \in U$, consider x as a u-bit vector and define the hash function $h_A(x) = Ax \mod 2$. Observe that there are $2^{m \cdot u}$ possible $u \times m$ binary matrices, and so this creates a hash family $\mathcal{H}$ of size $2^{m \cdot u}$. We prove that this hash family is universal.

Theorem 2 *$\mathcal{H}$ is a universal hash family.*

Proof. To show that $\mathcal{H}$ is universal, we must prove that for all $x, y \in U$, $x \neq y$,

$$\Pr[h_A(x) = h_A(y)] \leq \frac{1}{n}.$$

Observe that $h_A(x) = h_A(y) \iff h_A(x - y) = \vec{0}$. Let $z = x - y$. Then:

$$\Pr[h_A(x) = h_A(y)] \iff \Pr[h_A(z) = \vec{0}] \iff \Pr[Az = \vec{0}].$$

Since $n = 2^m$ and $U = \{0,1\}^u$, we want to show that for all $z \in \{0,1\}^u$:

$$\Pr[Az = \vec{0}] \leq \frac{1}{2^m}.$$

Since $x \neq y$, it must be the case that $z \neq \vec{0}$, and so z must contain at least one non-zero entry. Call this non-zero entry $z_{i*} = 1$.

Let $A_1, ..., A_u$ be the column vectors of A, and note that $Az = \sum_{i \in [u]} z_i A_i$. Then, if $Az = \vec{0}$, it must be the case that the column vector A_{i*} is equal to $\sum_{i \neq i*} z_i A_i$.

What is the probability of this happening? Since each entry of A_{i*} is chosen u.r. with probability $\frac{1}{2}$, the probability that they all match with $\sum_{i \neq i*} z_i A_i$ is $\frac{1}{2^m} = \frac{1}{n}$, and so

$$\Pr[Az = \vec{0}] \leq \frac{1}{2^m}.$$

■

9.4 Another Universal Hash Construction

The first universal hash construction takes up a lot of space. A better construction uses a prime number p, where $m < p < 2m$, and m is the size of the universe.

Denote the integers $\{0, 1, 2, 3, ..., p - 1\}$ as $\mathbb{Z}_p$.

$\forall a, b \in \mathbb{Z}_p, a \neq 0$, define a function f_{ab} from $U \to \mathbb{Z}_p$ as $f_{ab}(x) = ax + b \mod p$. The hash of an element x with parameters a and b is $h_{ab}(x) = f_{ab}(x) \mod n$.

Define the hash family H to be $\{h_{ab} | a, b \in \mathbb{Z}_p, a \neq 0\}$.

H has only p choices of b and $(p-1)$ choices of a, so there are $p(p-1)$ possible functions, with each function taking $O(\log M)$ bits to store a and b. We prove that H is universal (i.e. for any $x \neq y \in U$, $\Pr[h(x) = h(y)] \leq \frac{1}{n}$).

Theorem 3 *H is universal.*

Proof. Note that $f_{ab}(x) = (ax + b) \mod p$ and $h_{ab}(x) = f_{ab}(x) \mod n$. Furthermore:

$$h_{ab}(x) = h_{ab}(y) \implies f_{ab}(x) \equiv f_{ab}(y) \mod n.$$

Claim 1 $\forall r, s \in \mathbb{Z}_p$:

$$\Pr[f_{ab}(x) = r \wedge f_{ab} = s] = \begin{cases} 0 & r = 0 \\ \frac{1}{p(p-1)} & r \neq 0 \end{cases}$$

Proof. If $f_{ab}(x) = r$ and $f_{ab}(y) = s$, then

$$\begin{aligned} ax + b \mod p &= r \\ ay + b \mod p &= s. \end{aligned}$$

For unknown a and b, this system has a unique solution in $\mathbb{Z}_p$: $a = \frac{r-s}{x-y}$. Since we require $x \neq y$, a is non-zero iff $r \neq s$. Thus, there is exactly 1 out of all possible $p(p-1)$ functions that give both $f_{ab}(x) = r$ and $f_{ab}(y) = s$. ∎

We have that $h_{ab}(x) = h_{ab}(y)$ iff $r = s \mod n$. Then:

$$\Pr[h_{ab}(x) = h_{ab}(y)] = \frac{1}{p(p-1)} \times |\{(r, s) \mid r \neq s \text{ and } r \equiv s \mod n\}|. \tag{9.1}$$

Since there are $\lceil \frac{P}{n} \rceil - 1$ possible values for s such that $r = s \mod n$, the size of the set in the last term of Equation 9.1 is at most $P(\lceil \frac{P}{n} \rceil - 1) \leq \frac{p(p-1)}{n}$. This gives us the following result:

$$\Pr[h_{ab}(x) = h_{ab}(y)] \leq \frac{1}{p(p-1)} \cdot \frac{p(p-1)}{n} = \frac{1}{n} \tag{9.2}$$

Therefore, H is universal. ∎

9.5 Perfect Hashing

In this section, we consider only *static* dictionaries, where the set S of elements in the dictionary is fixed and the only concern is the "find" operation.

Definition 5 *A family $\mathcal{H}$ of hash functions is called perfect if for every subset $S \subset U$, there is a hash function $h \in \mathcal{H}$ such that h is perfect for S.*

Suppose $|S| = N$. We show a two-level hashing that gives perfect hashing scheme with $O(N)$ space and constant look-up time, and uses universal hashing.

Claim 2 *If we use universal hashing for a set S with size N into a table of size $n = O(N)$, then with probability $\geq \frac{1}{2}$, chain sizes are $O(\sqrt{n})$.*

Proof. For all $x, y \in S$, let

$$c_{xy} = \begin{cases} 1 & h(x) = h(y) \\ 0 & h(x) \neq h(y) \end{cases}$$

and define $c = \sum_{x \neq y \in S} c_{xy}$. Then we have the following:

$$\mathbb{E}[C] = \mathbb{E}[\sum_{x \neq y} c_{xy}] = \sum_{x \neq y} \mathbb{E}[c_{xy}] \leq \binom{N}{2} \frac{1}{n}. \tag{9.3}$$

If $n = N$, this value is approximately $\frac{n}{2}$. By Markov's Inequality, $\Pr[c \geq N] \leq \frac{1}{2}$. In a chain of size $\geq \sqrt{2N}$, there are $\binom{\sqrt{2N}}{2} \geq N$ collisions, so the max chain size is $O(\sqrt{N})$ with probability at least $\frac{1}{2}$. ■

If L_i is the length of a chain at index i, then the total number of collisions is $\sum_i \binom{L_i}{2}$. Therefore, with probability at least $\frac{1}{2}$, $\sum_i L_i^2 \leq 3N$.

Denote the first-level hash function mapping $U \to [N]$ as h^*. For all L_i keys mapped into location i, build a secondary table of size $M_i = L_i^2$, and use a second-level universal hash function h_i^* to map each of the L_i keys into $[M_i]$. Setting n to L_i^2 in Equation 9.3 gives a $\leq \frac{1}{2}$ probability of there being any collisions in the second-order hash table, and in the case of a collision, a different second-order hash function is tried until there are no collisions.

To look up an item q in the hash table, check $h^*(q)$ for the location of the second-order hash table, then check $h_i^*(q)$ to get the location of q. The total amount of space for the first-order hash function is N, and the amount of space used for all of the second-order hash tables is $\sum_i O(L_i^2) = O(N)$.

9.5.1 Perfect Hashing Summary

For a static set $S \subseteq U$ with $|S| = N$:

1. Pick a random h from a universal hash family H mapping $U \to N$. Continue to resample h until the total number of collisions is $\leq 3N$.
2. Let L_i be the number of elements $x \in S$ where $h(x) = i$. Then, $\sum \binom{L_i}{2} \leq 3N$ with probability at least $\frac{1}{2}$.
3. For each $i \in [N]$, create a table of size $4L_i^2$ and pick a second-level hash h_i from a universal hash family mapping $U \to [4L_i^2]$. Map all L_i keys in location i using h_i^*. The probability of a collision on the second-level hash is $\leq \frac{1}{2}$. Continue to resample h until there are no collisions in the second-level table.
4. For a query on item q, first look at $h(q)$ for the first level, then $h_{h(q)}(q)$ for the second level to find the item.

9.6 Bloom Filters

In some scenarios, most of the queries to a hash table come back negative. In these cases, it may be more efficient to maintain a fast primary hash that may return a false positive.

If the filter returns a negative result (i.e. the item is not in the table), then we guarantee that the item cannot exist in the table. However, if the filter returns a positive result, then we may use a secondary data structure to check for a false positive.

Recall that if we have N items from a universe of size m, we need $O(N \log m)$ bits to store a proper hash table, as it takes $\log m$ bits to represent each item. The idea behind a Bloom filter is to have a hash that maps each element to a single bit. However, in this case any collision would lead to a false positive. Here is how a Bloom filter that maps items from $S \subset U$ would work:

- Keep an n-bit vector as the hash table.
- Let $h_1, h_2, ..., h_k$ be k random hash functions from U to $[n]$. For the purposes of analysis, assume that the hash function's output is a random number between 1 and n.

- Given an element $x \in S$, set all $h_i(x)$ bits to 1.

To perform a lookup for an item y, check if all k bits of $h_i(y)$ are 1. Answer "yes" if they are all 1, and "no" otherwise.

The hashing procedure ensures that an item y is not in the hash table if any bit $h_i(y)$ is zero so there are never false negatives, but it is possible that all bits of $h_i(y)$ have been set to 1 by items which are not y and the lookup returns a false positive.

The probability that a particular item sets a particular bit to 1 is $(1-\frac{1}{n})^k$, so the probability that any individual bit is zero after adding N items is

$$(1-\frac{1}{n})^{kN} \approx e^{\frac{-kN}{n}}.$$

Denote this probability as p.

The probability of a false positive on a given item is equal to the probability that all k locations in the item's hash are set to 1, which has probability $(1-p)^k = f$. Let

$$g = k\ln(1-e^{\frac{-kN}{n}}).$$

Then we have that $f = e^g$.

To find the optimal value of k that minimizes the number of false positives for a fixed n and N, calculate the derivative of g with respect to k.

$$\frac{dg}{dk} = \ln(1-e^{\frac{-kN}{n}}) + \frac{kN}{n} \cdot \frac{e^{\frac{-kN}{n}}}{1-e^{\frac{-kN}{n}}}$$

This derivative is zero when $k = \ln 2(\frac{n}{N})$, and gives a false positive probability of $0.6185^{\frac{n}{N}}$. For example, if $n = 2N$ then the false positive probability is around 38%, and if $n = 8N$ then the false positive probability is around 2%.

In general, for an arbitrary false positive tolerance ϵ, the number of bits that need to be used is about $1.44\log(\frac{1}{\epsilon})$ per entry, or $1.44N\log(\frac{1}{\epsilon})$ for the whole table.