Lecture 7: K-wise Independence and Hashing

1 K-wise Independence and Hashing

Recall we defined pairwise independence and showed how to construct n pairwise independent bits $\{0,1\}$ from $O(\log n)$ truly random bits. We will initially focus on pairwise independent setting and now think about generating a pairwise independent random variables but from the large $\{0,1,2,\ldots,m-1\}$ instead of just bits.

Definition 1. *$X_1, X_2, \ldots, X_n \in \{0,1,\ldots,m-1\}$ are pairwise independent if:*

(i) X_i is uniformly distributed over $\{0,1,\ldots,m-1\}$ for all $i \in [n]$

(ii) For $i \neq j$, $\Pr[X_i = r \text{ and } X_j = s] = \frac{1}{m^2}$ for all $r, s \in \{0,1,\ldots,m-1\}$

Suppose $m = 2^h$. Then we can use previous bit scheme for each of the h bits. This would require $O(h \log n) = O(\log m \log n)$ bits and is not that efficient. We will see a way to get $O(\log n + \log m)$ bits.

We consider the setting $n \leq m$ and achieve a good scheme when m is a prime power p^k where p is a prime number. We recall some facts about prime numbers.

Lemma 1. *$\mathbb{Z}_p = \{0,1,2,\ldots,p-1\}$ under $+$ and $\times \bmod p$ is a field.*

Corollary 1. *Suppose $i, j \in \mathbb{Z}_p$ with $i \neq j$. Then for any $r, s \in \mathbb{Z}_p$, there exists a unique pair (a,b) that satisfy the equations:*

$$ai + b \equiv r \pmod{p} \tag{1}$$
$$aj + b \equiv s \pmod{p} \tag{2}$$

Proof. We solve these equations for a, b as we do over reals since $\mathbb{Z}_p$ is a field:

$$a(i-j) \equiv r - s \pmod{p} \tag{3}$$
$$a \equiv (r-s)(i-j)^{-1} \pmod{p} \tag{4}$$

Since $i \neq j$ and $\mathbb{Z}_p$ is a field, $(i-j)^{-1}$ exists and is unique in $\mathbb{Z}_p$. Then $b \equiv r - ai \pmod{p}$ or $b \equiv s - aj \pmod{p}$. □

1.1 Constructing pairwise independent random variables $X_0, X_1, \ldots, X_{p-1}$ with range $\{0,1,2,\ldots,p-1\}$

1. Pick $a, b \in \mathbb{Z}_p$ independently
2. For each i, set $X_i = ai + b \bmod p$

Claim 1. *X_i is uniformly distributed over $\mathbb{Z}_p$.*

Proof. Fix i. Then ai is fixed. Since b is uniformly random in $\mathbb{Z}_p$, $ai + b$ will be uniformly random in $\mathbb{Z}_p$. □

Claim 2. *For $i \neq j$, X_i and X_j are pairwise independent, i.e., $\Pr[X_i = r \text{ and } X_j = s] = \frac{1}{p^2}$.*

Proof. There exists a unique $(a, b) \in \mathbb{Z}_p^2$ such that:

$$ai + b \equiv r \pmod{p} \tag{5}$$
$$aj + b \equiv s \pmod{p} \tag{6}$$

Therefore $\Pr[X_i = r \text{ and } X_j = s] = \frac{1}{p^2}$. □

Amount of randomness required is $2 \log p = O(\log n)$.

If one notices in the preceding construction, we only used the fact that $\mathbb{Z}_p$ is a field. Thus we could have done the same construction over any finite field. An important result in algebra is:

Theorem 1. *If $\mathbb{F}$ is a finite field, then $|\mathbb{F}| = p^k$ for some prime p and integer $k \geq 1$. Moreover, for every prime p and integer k, there is a finite field of order p^k, and all finite fields of the same order are isomorphic.*

Note that $+$ and $\times$ in a finite field of order p^k need to be defined, and one needs to handle the computational aspects. For our purposes, we will treat them as $O(1)$ time operations.

The advantage of using fields of size p^k is that we can use $p = 2$ and any integer k. Powers of 2 are natural for CS.

Thus we can obtain n pairwise independent random variables over $\{0, 1, \ldots, 2^k - 1\}$ when $n \leq 2^k$.

Suppose we want $n \leq m$. Say m is also a power of 2, $m = 2^h$. Then we can generate n r.v.s over $\{0, 1, \ldots, 2^h - 1\}$ and drop the first $\ell - h$ bits. It is an easy exercise to argue that this works.

$n > m$ is easy. Since we can generate n r.v.s and just use m of them.

2 K-wise Independence

Definition 2. *$X_1, X_2, \ldots, X_n \in \{0, 1, \ldots, m-1\}$ are k-wise independent if:*

(i) X_i for all $i \in [n]$ are uniformly distributed over $\{0, 1, \ldots, m-1\}$

(ii) Any k of the given random variables are independent

One can generalize the construction for $k = 2$ to larger k via polynomials.

Let $\mathbb{F}$ be a finite field of order m. Pick $a_0, a_1, a_2, \ldots, a_{k-1} \in \mathbb{F}$ uniformly and independently from $\mathbb{F}$.

Consider polynomial:

$$p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{k-1} x^{k-1}$$

Now set each $X_i = p(i)$.

So we need to pick k coefficients, so total of $O(k \log m)$ bits.

2.1 Why does the construction work?

Suppose $i_1, i_2, \ldots, i_k$ are distinct elements of the field. Let $j_1, j_2, \ldots, j_k$ be arbitrary elements of the field. We claim $p(i_1) = j_1, p(i_2) = j_2, \ldots, p(i_k) = j_k$ is satisfied by a unique degree $\leq k-1$ polynomial.

First, there exists a degree $\leq k-1$ polynomial by Lagrange interpolation. Consider the k constraints:

$$p(i_1) = j_1, p(i_2) = j_2, \ldots, p(i_k) = j_k$$

Suppose we have two such polynomials p and q. Then consider $p - q$. This has k non-trivial roots, which is not possible for a non-zero polynomial of degree $< k$.

3 Hashing and Hash Tables

Dictionary data structure is perhaps the most basic and important data structure in programming. We want to store and retrieve a bunch of key-value pairs where keys are assumed to be distinct and come from some large universe U of objects. Say $|U| = N$.

However, at any time we only have a small subset $n \ll N$ of keys in our data structure.

3.1 Operations

- `insert(x)`: add x to stored set
- `find(x)`: is x in set?
- `delete(x)`: remove x from set if it is there

Since everything in a computer can be represented as a string, we can assume WLOG that U is ordered. Typically we deal with U being numbers, but in practice U can be complex objects such as images, tuples of strings, etc.

Dictionaries over an ordered universe U can be implemented via pointer-based tree data structures: $O(\log n)$ comparisons of unwieldy objects. Pointer-based data structures can be bad for cache access for many applications.

3.2 Hashing

Use array/tables so memory access is better. Hash functions allow mapping from unwieldy objects to small integers. Can get $O(1)$ on average.

If U is a finite universe of size N, $h : U \to [k]$ is a hash function. Let $\mathcal{H}_{all}$ be the set of all hash functions from U to $[k]$.

$\mathcal{H} \subseteq \mathcal{H}_{all}$ is a family of hash functions.

3.3 Hashing

1. Fix some family of "nice" hash functions $\mathcal{H} \subseteq \mathcal{H}_{all}$
2. Pick a random $h \in \mathcal{H}$
3. Let $S \subseteq U$ where $|S| = n$
4. Map each $x \in S$ to $h(x)$

Assumption: we have an array of size k where location i maps to location $h(x)$. Goal is to store x in location $h(x)$. But what if $h(x) = h(y)$ for some $x \neq y$? **Collision!**

Hashing data structures differ in how they handle collisions. We will see two ideas.

4 Hash tables with chaining

U is universe with N elements. We will assume that we want to store a set $S \subseteq U$ where $|S| = n$ and we know n. Hence we can allocate space roughly $O(n)$. If we don't know n, we can guess and double and rebuild.

Given knowledge of n, it is common to set up table size k to be cn for some small constant c, and then pick a hash family $\mathcal{H}$ of functions that map U to $[k]$, where $[k] = \{0, 1, \ldots, k-1\}$.

In chaining, all elements hashed to a bucket index $i \in [k]$ are stored in a linked list associated with i.

We hope that the hash function spreads the items nicely and that the lists at each bucket are small on average.

Operation: each of the operations `insert(x)`, `find(x)`, `delete(x)` take time proportional to $1 + \ell(x)$ where $\ell(x)$ is the length of the list of elements stored at $h(x)$.

4.1 What is a good hash function?

Suppose we have S of n elements and a hash table of size $k = n$. Then if we choose $\mathcal{H}_{all}$ as our family and pick $h \in \mathcal{H}_{all}$ at random, then it is like balls and bins: $h(x)$ for any x will be uniformly distributed, and for $x, y \in S$ with $x \neq y$, $h(x)$ and $h(y)$ will be independent.

We will call $\mathcal{H}_{all}$ an ideal hash family and a random h from it as an ideal hash function.

The problem is that $\mathcal{H}_{all}$ is a very large and complex family, and a random h has no structure, so computing $h(x)$ is not efficient.

So what we want are hash function families that have the following features:

1. For every $h \in \mathcal{H}$, h must be efficiently computable
2. Sampling a random h from $\mathcal{H}$ should be efficient
3. A random h from $\mathcal{H}$ should behave as closely as possible to an ideal hash function

Definition 3. *A hash family $\mathcal{H}$ from U to $[k]$ is **strongly 2-universal** if the following properties hold for a random h chosen from $\mathcal{H}$:*

(i) For all $x \in U$, $h(x)$ is uniformly distributed over $[k]$

(ii) For all $x, y \in U$ with $x \neq y$, $h(x)$ and $h(y)$ are independent, i.e., $\Pr[h(x) = i \text{ and } h(y) = j] = \frac{1}{k^2}$

We now define a weaker version:

Definition 4. *A hash family $\mathcal{H}$ is **2-universal** if for all $x, y \in U$ with $x \neq y$:*

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

Observation: A strongly 2-universal hash family is also 2-universal.

Suppose we had a 2-universal hash family:

Lemma 2. *Let $S \subseteq U$ with $|S| = n$. Fix $x \in S$. Suppose h is chosen from a 2-universal hash family $\mathcal{H} : U \to [k]$. Let S_x be the random set $\{y \in S : h(x) = h(y)\}$. Then $\mathbb{E}[|S_x|] \leq 1 + \frac{n-1}{k}$.*

Proof. Fix $y \in S$ with $y \neq x$. Let I_y be the indicator that $h(y) = h(x)$. Then:

$$|S_x| = 1 + \sum_{y \in S, y \neq x} I_y$$

$$\mathbb{E}[|S_x|] = 1 + \sum_{y \in S, y \neq x} \mathbb{E}[I_y] = 1 + \sum_{y \in S, y \neq x} \Pr[h(y) = h(x)] \leq 1 + \frac{n-1}{k}$$

□

Corollary 2. *For any sequence of n operations starting from an empty set, the expected cost of the operations is $O(n(1 + \frac{n}{k}))$. If $k = cn$ for constant $c > 0$, then expected cost is $O(n)$.*

4.2 How do we construct strongly 2-universal and 2-universal hash families?

We already did this when constructing pairwise independent random variables!

Say we create $X_0, X_1, \ldots, X_{n-1}$ pairwise independent r.v.s with range $[k]$. Then we think of X_i as $h(i)$.

To repeat: Let $N = 2^\ell$ for some ℓ and $k = 2^h$ for some $h \leq \ell$. Consider field $\mathbb{F}$ of size 2^ℓ. Associate U with $N = 2^\ell$.

$$\mathcal{H} = \{h_{a,b} : a, b \in \mathbb{F}\}$$

$$h_{a,b}(i) = ai + b$$

For $i \neq j$, if a, b chosen independently and uniformly at random, $h_{a,b}(i)$ and $h_{a,b}(j)$ are independent and identically distributed over $\mathbb{F}$. We can then truncate the bits to get distribution over $[k]$.

5 2-Universal Family

Although the preceding construction works, there is a simpler family that yields a 2-universal family.

Let p be a prime $\geq N$. Define:

$$\mathcal{H} = \{h_{a,b} : a \in \{1, 2, \ldots, p-1\}, b \in \mathbb{Z}_p\}$$

$$h_{a,b}(i) = (ai + b \bmod p) \bmod k$$

Claim 3. *$\mathcal{H}$ is a 2-universal hash family.*

Note the difference with strongly 2-universal family: We did not allow a to be 0. This avoids the not-so-good/interesting hash function h_0 which maps all elements to 0. Second, by taking a simple mod$k$, we lose the uniformity property. Nevertheless, the family is 2-universal and is quite simple to describe and implement. Can use standard arithmetic on $\mathbb{Z}_p$.

5.1 Sketch of universality

Lemma 3. *Fix $i \neq j$ and $r, s \in \mathbb{Z}_p$. Exactly one pair (a, b) with $a \neq 0$ and $a, b \in \mathbb{Z}_p$ satisfies:*

$$ai + b \equiv r \pmod{p} \tag{7}$$

$$aj + b \equiv s \pmod{p} \tag{8}$$

Proof. $a(i-j) \equiv r-s \pmod{p}$, so $a \equiv (r-s)(i-j)^{-1} \pmod{p}$. If $i \neq j$, then $ai+b \equiv aj+b \bmod p$ iff $a = 0$, so no collisions before folding. □

Think of $h_{a,b}(i)$ as a two-step process:

1. $r = ai + b \bmod p$
2. $h = r \bmod k$

If $i \neq j$, $r \neq s$, but r can be equal to s after mod$k$.

Lemma 4. *Among pairs $(r, s) \in \mathbb{Z}_p \times \mathbb{Z}_p$ such that $r \bmod k = s \bmod k$, there are $\frac{p^2-p}{k}$ such pairs (assuming $p \geq k$).*

Proof. For any r, there are $\frac{p}{k}$ values of s such that $r \bmod k = s \bmod k$, but this includes $s = r$. So total is $p \cdot \frac{p}{k} - p = \frac{p^2-p}{k}$, since we don't want to count the pair (r, r) where $r = s$. □

Fixing $i \neq j$, random $a \neq 0$ and $b \in \mathbb{Z}_p$ creates a random pair $(r, s) \in \mathbb{Z}_p \times \mathbb{Z}_p$ with $r \neq s$. Total of $p(p-1)$ pairs. Pairs (r, s) where $r \neq s$ that collide after folding: $\frac{p^2-p}{k}$.

Hence $\Pr[h_{a,b}(i) = h_{a,b}(j)] = \frac{p^2-p}{k \cdot p(p-1)} = \frac{1}{k}$.

6 Hash Tables: Linear Probing

Chaining is simple and easy to analyze. In terms of practice, using linked lists and dynamic memory allocation is not so great. There are other hashing techniques such as linear probing and cuckoo hashing that try to take advantage of arrays.

Linear probing is a technique that handles collisions by scanning along the array. If $h(x)$ is occupied, see Kent's notes for description.