Lecture 7

k-wise independence and Hashing

Recall we defined pairwise independence and showed how to construct n pairwise independ bits ($\{0,1\}$ rvs) from $O(\lg n)$ true random bits.

We will initially focus on pairwise independent setting and now think about generating n pairwise indep random variables but from the range $\{0,1,2,\ldots,m-1\}$ ($[m]$) instead of just bits.

Defn: $X_1, X_2, \ldots, X_n \in [m]$ are pairwise independent if

(i) X_i is uniformly distributed over $[m]$ $\forall i \in [n]$

(ii) $\forall i \neq j \in [n]$ $\Pr[X_i = s \text{ and } X_j = t]$
$= \frac{1}{m^2}$ $\forall s, t \in [m]$.

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 $n = m$ and achieve a good scheme when $n = m = p$ where p is a prime number.

We recall some facts about prime numbers.

Lemma: $\mathbb{Z}_p = \{0, 1, 2, \ldots, p-1\}$ forms a field under $+$ and $\times$ mod p.

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

$$ai + b = r \mod p$$
$$aj + b = s \mod p.$$

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

$$a(i-j) = r-s$$

Since $i \neq j$ $a = \frac{r-s}{i-j}$ exists and unique in $\mathbb{Z}_p$.

Then $b = ai - r$
or $aj - s$.

□.

Constructing pairwise independent random variables $X_0, X_2, .., X_{p-1}$ with range $\{0,1,2,..,p-1\}$.

1. Pick $a, b \in Z_p$ independently

2. For each i set

$$X_i = ai + b \mod p.$$

Claim: X_i uniformly distributed over Z_p.

Proof: Fix a. Then ai is fixed. Since b is uniformly random in Z_p $ai + b$ will be uniformly random in Z_p.

□

Claim: For $i \neq j$ X_i and X_j are pairwise independent.

$$\Pr[X_i = r \text{ and } X_j = s] = ?$$

$$= \frac{1}{p^2}.$$

Unique $a, b \in \mathbb{Z}_p$ s.t.

$$a i + b = r \qquad i \neq j$$
$$a j + b = s$$

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

□

Amount of randomness required is $2\lceil \log p \rceil$ so $O(\log n)$.

If one notices 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: If $\mathbb{F}$ is a finite field then $|\mathbb{F}| = p^k$ for some prime p and integer ≥ 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 × 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 $[n]$ when n is 2^ℓ.

Suppose we want $m < n$.
Say m is also a power of 2, 2^h. Then we can generate n RVs over $2^\ell > 2^h$ and drop the first $\ell - h$ bits.
It is easy exercise to argue that this works.

$n < m$ is easy since we can generate m RVs and not use $m - n$ of them.

k-wise independence

Defn: $X_1, X_2, \ldots, X_n \in [m]$ are k-wise independent if

(i) X_i, $i \in [n]$ is uniformly distributed over $[m]$

(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 n.

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 each

$$X_i = p(i).$$

So we need to pick k #s so total of $O(k \log n)$ bits.

Why does the construction work?

Suppose $i_1, i_2, \ldots, i_k$ are <u>distinct</u> elements of the field. Let $j_1, j_2, \ldots, j_k$ be arbitrary elements of the field.

We claim

$$\begin{aligned} p(i_1) &= j_1 \\ p(i_2) &= j_2 \\ &\vdots \\ p(i_k) &= j_k \end{aligned}$$

is satisfied by a unique degree k polynomial.

Find f a degree $k-1$ polynomial.

By Lagrange interpolation.

Consider

$$\sum_{l=1}^{k} j_l \cdot \prod_{h \neq l} \frac{(X - i_h)}{(i_l - i_h)}$$

satisfies the conditions.

Suppose we have two such polynomials p and q. Then consider $(p-q)$. This has k non-trivial roots which is not possible.

□.

Hashing and Hash Tables

Dictionary data structure is perhaps the most basic and important data structure in programming.

Want to store and retrieve a bunch of (key, value) pairs where keys are assumed to be distinct and come from some large universe $\mathcal{U}$ of objects. Say

$$|\mathcal{U}| = N$$

However at any time we only have a small subset n of keys in our data structure.

Operations

insert(x) — key. add x to stored set

find(x) is x in set?

delete(x) remove x from set. if it is

Since everything in a computer can be represented as a string we can assume w.l.o.g 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 binary data structures.

Disadvantage:
- $O(\log n)$ comparisons of unwieldy objects
- Pointer based data structures can be bad for memory access.

Hashing

- Use arrays / tables so memory access is better.
- hash functions allow mapping large unwieldy objects to small integers.
- can get $O(1)$ on <u>average</u>.

<u>Set up:</u>

U a finite universe of size N

$h: U \to [K]$ is a hash function

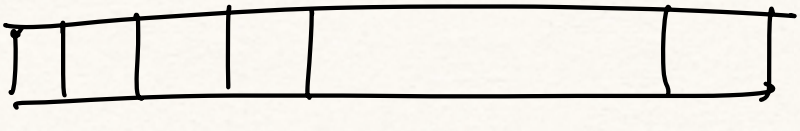
Let H_{all} be the set of all hash functions from $U \to [K]$.

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

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 $S \subseteq \mathcal{U}$ where $|S| = n$.
4. map each $x \in S$ to $h(x)$.

Assumption: we have an array of size k (hash table).



x 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.

Hash tables with chaining

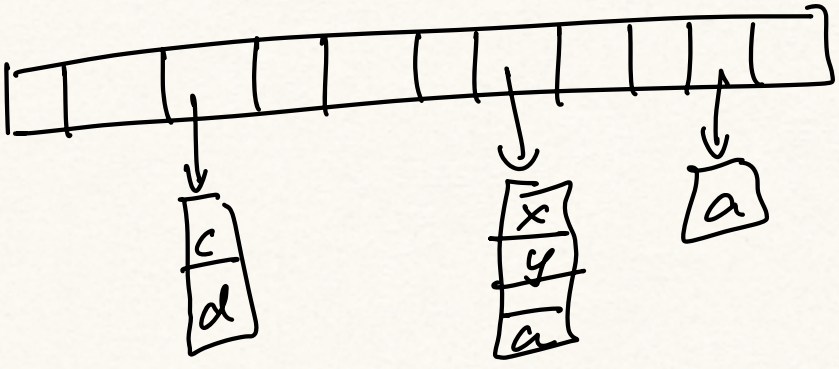
$\mathcal{U}$ universe with N elements

We will assume that we want to store a set $S \subseteq \mathcal{U}$ where $|S| = n$ and we know n. Hence we can allocate space roughly $\sim O(n)$

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. and then pick a hash family $\mathcal{H}$ of functions that map $\mathcal{U}$ to $[k]$ or $[N] \to [k]$.

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.

Observation: each of the operations insert(x), find(x), delete(x) take time proportional to the $\# |\ell(x)|$

where $l(x)$ is the list of elements stored at $h(x)$.

What is a good hash function?

Suppose we have S of n elements.
And a hash table of size $\sim n$.
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 $\forall x, y \in S$
$x \neq y$ $h(x)$ and $h(y)$ will be independent.

We will call $\mathcal{H}_{all}$ "ideal" hash family and a random h as 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 functions/families that have the following features

(i) a every $h \in \mathcal{H}$ must be efficiently computable

(ii) sampling a random h from $\mathcal{H}$ should be efficient

(iii) a random h from H should behave as closely as possible to an ideal hash function.

Defn: A hash family H from $[N] \to [k]$ is strongly 2-universal if the following properties hold for a random h chosen from H.

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

(ii) $\forall x, y \in U$ $x \neq y$

$h(x)$ and $h(y)$ are independent

ie $\Pr_{h \sim H}[h(x) = i, h(y) = j] = \frac{1}{k^2}$.

We now define a "weaker" version.

Defn: A hash family $\mathcal{H}$ is 2-universal if $\forall\ x, y \in \mathcal{U}$ $x \neq y$

$$\Pr_{h \sim \mathcal{H}}[h(x) = h(y)] \leq \frac{1}{k}.$$

Observation: A strongly universal hash family is also universal.

Suppose we had a universal hash family.

Lemma: Let $S \subseteq \mathcal{U}$ with $|S| = n$.

Fix $x \in \mathcal{U}$. Suppose h is chosen

from a universal hash family $\mathcal{H}$.
$[U] \to [k]$. Let S_x be the
random set $\{y \mid h(x) = y\}$.

Then $E[|S_x|] = 1 + \frac{n}{k}$.

Proof: Fix $y \in S$ $y \neq x$.

Let D_y be indicator that $h(y) = h(x)$

$$|S_x| = 1 + \sum_{\substack{y \neq x \\ y \in S}} D_y$$

$$E[|S_x|] = 1 + \sum_{y \neq x} E[D_y]$$

$$\leq 1 + \frac{n}{k} \quad \square.$$

Corollary: For any sequence of n
operations starting from an empty
set the expected cost of the

operation is $O(n \cdot (1 + \frac{n}{k}))$.

If $k = \Omega(n)$, then expected cost is $O(n)$.

How do we construct strongly universal and universal hash families?

We already did when constructing pairwise independent random variables!

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

To repeat

Let $N = 2^l$ for some l

and $K = 2^h$ for some $h < l$.

Consider field $\mathbb{F}$ of size 2^l.

associate $[N]$ with $[N]$.

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

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

For $i \neq j$ if a, b chosen independently

and at random

$h_{a,b}(i)$ and $h_{a,b}(j)$ are

independent and identically

distributed over $[N]$. We can then truncate the bits to get distribution over $[k]$.

Universal Family.

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

Let p be a prime $> N$.

Define $H = \{ h_{a,b} \mid a \in \{1, \ldots, p-1\}, b \in \mathbb{Z}_p \}$

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

Claim: $\mathcal{H}$ is a universal hash family.

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

Sketch of universality

Lemma: Fix i, j $i \neq j$, $r \neq s$ $\in \mathbb{Z}_p$.

Exactly one pair (a, b) $a \neq 0$ $a, b \in \mathbb{Z}_p$ s.t.

$$ai + b = r$$
$$aj + b = s$$

Proof:

$$a = \frac{r - s}{i - j} \mod p$$

$$b = r - ai \mod p$$

□.

If $i \neq j$ then $ai + b \neq aj + b \mod p$ if $a \neq 0$.

$\Rightarrow$ no collisions before "folding".

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

$$x = ai + b \bmod p$$

$$x' = x \bmod k.$$

If $i \neq j$ $\quad x \neq s$

but x' can be equal to s'

Lemma: # of pairs $(x, s) \in \mathbb{Z}_p \times \mathbb{Z}_p$

$x \neq s$ such that $x \bmod k = s \bmod k$

is $\dfrac{p(p-1)}{k}$.

Proof: For any fixed x # of s such

that $x \bmod k = s \bmod k$

$\leq \left\lceil \frac{p}{m} \right\rceil$ but this includes x.

So total is $p\left(\left\lceil \frac{p}{m} \right\rceil - 1\right)$ since we don't

want to count (x,s) pairs.

$$\leq \frac{p(p-1)}{m} \ldots$$

D.

Fixing i, j random (a,b) $a \neq 0$ $a,b \in \mathbb{Z}_p$

creates a random pair $(x,s) \in \mathbb{Z}_p \times \mathbb{Z}_p$

$x \neq s$. Total of $p(p-1)$ pairs.

pairs (x,s) s.t. $x \neq s$ that collide

after folding $\leq \frac{p(p-1)}{m}$.

Hence $h_{a,b}(i) = h_{a,b}(j) \leq \frac{1}{m}$.

□.

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 A if h(x) is occupied.