Randomized Data Structures

We look at the role of randomization in building data structures that are elegant and efficient.

Example 1: Universal Hash Functions

Many applications call for a dynamic dictionary, i.e., a data structure for storing sets of keys $S$ that supports the operations INSERT, DELETE and FIND. We assume that the keys are drawn from a large universe $U = \{0, 1, \ldots, m - 1\}$.

We will hash the keys in $S$ into a hash table $T = \{0, 1, \ldots, n - 1\}$ using a hash function $h : U \rightarrow T$. I.e., we store element $x \in S$ at location $h(x)$ of $T$. Typically, we will want $n$ to be much smaller than $m$, and comparable to $|S|$, the size of the set to be stored.

We assume that each location of $T$ is able to hold a single key. If $h$ maps several elements of $S$ to a single location, we store them in an auxiliary data structure (say, a linked list) at that location. The time to perform any of the above operations is proportional to the time to evaluate a single location, we store them in an auxiliary data structure (say, a linked list) at that location.

We assume that each location of $T$ is able to hold a single key. If $h$ maps several elements of $S$ to a single location, we store them in an auxiliary data structure (say, a linked list) at that location. The time to perform any of the above operations is proportional to the time to evaluate $h$ (to find the location $h(x)$) plus the length of the list at $h(x)$ (since the operation may have to search the entire linked list). So good performance depends on having few collisions in the table.

Traditionally, people have developed hash functions that give a small expected number of collisions assuming that the sequence of operations is random. But such schemes based on a deterministic hash function $h$ are bound to be very bad for some sequences (see the next two exercises).

Ex: Show that any fixed hash function $h : U \rightarrow T$ must map at least $\frac{m}{n}$ elements of $U$ to some location in $T$. Deduce that, if $m$ is much larger than $n$, then there will be sets $S \subseteq U$ that are all mapped by $h$ to a single location in $T$. □

Ex: A hash function $h$ is said to be perfect for a set $S \subseteq U$ if it causes no collisions on $S$. Show that, for any particular set $S$ of size $\leq n$, it is possible to construct a hash function that is perfect for $S$, but that it is not possible to construct a hash function that is perfect for all $S$ of this size. Show also that, for any fixed hash function $h$, the maximum possible number of sets $S$ of size $n$ for which $h$ is perfect is $\binom{m}{n}$. Compare this with the total number of such sets $S$. □

Instead, we will use a random hash function chosen from a suitable family. Building randomization into the hash function will mean that there will be no bad sequences.

Definition: A family $\mathcal{H}$ of hash functions $h : U \rightarrow T$ is 2-universal if, for all $x, y \in U$ with $x \neq y$, and for $h$ chosen u.a.r. from $\mathcal{H}$, we have $\Pr[h(x) = h(y)] \leq \frac{1}{n}$. □

Note that the functions in a 2-universal family “behave at least as well as” random functions wrt collisions on pairs of keys. The following fact illustrates why this is an appropriate definition:

Theorem: Consider any sequence of operations with at most $s$ inserts performed using a hash function $h$ chosen u.a.r. from a 2-universal family. The expected cost of each operation is proportional to (at most) $1 + \frac{s}{n}$.

Proof: Consider one of the operations, involving an element $x$. The cost of this operation is proportional to $1 + Z$, where $Z$ is the number of elements currently stored at $h(x)$. What is the expectation $E(Z)$? Well, let $S$ be the set of all (at most) $s$ elements that are ever inserted, and for each $y \in S$ let $Z_y$ be the indicator r.v. of the event that $y$ is currently stored at $h(x)$. Thus $Z = \sum_{y \in S} Z_y$ and $E(Z) = \sum_{y \in S} E(Z_y)$. Since $h$ is chosen from a 2-universal family, we have $E(Z_y) \leq \Pr[h(x) = h(y)] \leq \frac{1}{n}$. Hence $E(Z) \leq \frac{s}{n}$. This completes the proof. □
So what? Well, choose a table size \( n \) that is at least as large as the largest set \( S \) we will ever want to store, so that \( n \geq s \). Then the above Theorem ensures that the expected cost per operation is (proportional to) at most 2. I.e., we have constant expected time per operation, for any sequence of requests: there are no bad sequences.

Q: How do we construct a 2-universal family?
A: Simply make \( \mathcal{H} = \) set of all functions \( h: U \to T \)

Ex: Verify that this family is indeed 2-universal. \( \square \)

But is this a good choice? Actually no, because there are \( n^m \) functions in the family, and so it takes \( O(m \log n) \) bits to represent any of them. (Check you understand this.) Since the universe size \( m \) is assumed to be huge, this is impractical. What we need is a 2-universal family that is small and that is efficient to work with.

A 2-universal family

Let \( p \) be a prime with \( p \geq m \). Since for any \( m \) there exists a prime between \( m \) and \( 2m \), we can assume that \( p \leq 2m \).

Our hash functions will operate over the field \( \mathbb{Z}_p = \{0, 1, \ldots, p-1\} \), which includes our universe \( U \). (So if we get a family that is 2-universal over \( \mathbb{Z}_p \), it will certainly be 2-universal over \( U \) also.)

For \( a, b \in \mathbb{Z}_p \), define the function \( h_{ab}: \mathbb{Z}_p \to T \) by

\[
h_{ab}(x) = ((ax + b) \mod p) \mod n.
\]

Our hash family will be \( \mathcal{H} = \{ h_{ab}: a, b \in \mathbb{Z}_p, a \neq 0 \} \).

The key point here is that \( \mathcal{H} \) contains only \( p(p-1) \) functions (why?), and specifying a function \( h_{ab} \) requires only \( O(\log p) = O(\log m) \) bits. (Compare the \( O(m \log n) \) bits required for a purely random function.) To choose \( h_{ab} \in \mathcal{H} \), we simply select \( a, b \) independently and u.a.r. from \( \mathbb{Z}_p - \{0\} \) and \( \mathbb{Z}_p \) respectively. Moreover, evaluating \( h_{ab}(x) \) takes only a few arithmetic operations on \( O(\log m) \)-bit integers.

So this hash family is very efficient. But is it “random enough”? Surprisingly it is, as we now see:

Claim: The above family \( \mathcal{H} \) is 2-universal.

Proof: Consider any \( x, y \in \mathbb{Z}_p \) with \( x \neq y \). We need to figure out \( \Pr[h_{ab}(x) = h_{ab}(y)] \), where \( h_{ab} \) is chosen u.a.r. from \( \mathcal{H} \).

For convenience, define \( g_{ab}(x) = (ax + b) \mod p \), so that \( h_{ab}(x) = g_{ab}(x) \mod n \).

How can \( h_{ab}(x) = h_{ab}(y) \)? For this to happen, we must have

\[
g_{ab}(x) = g_{ab}(y) \mod n. \tag{*}
\]

So let's focus first on \( g_{ab} \). Let \( \alpha, \beta \) be any numbers in \( \mathbb{Z}_p \). I claim that

\[
\Pr[g_{ab}(x) = \alpha \land g_{ab}(y) = \beta] = \begin{cases} 0 & \text{if } \alpha = \beta; \\
\frac{1}{p(p-1)} & \text{otherwise}. \end{cases} \tag{**}
\]

To see this, note that if \( g_{ab}(x) = \alpha \) and \( g_{ab}(y) = \beta \) then we must have, in the field \( \mathbb{Z}_p \),

\[
ax + b = \alpha \quad \text{and} \quad ay + b = \beta.
\]

But these two linear equations in the two unknowns \( a, b \) have a unique solution in \( \mathbb{Z}_p \), namely \( a = (\alpha - \beta)(x - y)^{-1} \) and a similar expression for \( b \). (Check this.) And since \( x \neq y \), \( a \) is non-zero.
if and only if $\alpha \neq \beta$. This means that there is exactly one function $g_{ab}$ that gives us the values $g_{ab}(x) = \alpha$ and $g_{ab}(y) = \beta$ (and no function when $\alpha = \beta$). Since there are $p(p-1)$ functions in all, and we are picking one u.a.r., we’ve verified (**)).

Now let’s return to condition (∗). This tells us that we’ll get $h_{ab}(x) = h_{ab}(y)$ if and only if $\alpha = \beta \pmod{n}$, i.e., $\alpha$ and $\beta$ must be in the same residue class $\pmod{n}$. And from (**) we see that all such pairs with $\alpha \neq \beta$ have probability $\frac{1}{p(p-1)}$. So we have

$$\Pr[h_{ab}(x) = h_{ab}(y)] = \frac{1}{p(p-1)} \times |\{ (\alpha, \beta) : \alpha \neq \beta \text{ and } \alpha = \beta \pmod{n} \}|.$$

(***)

How many pairs $(\alpha, \beta)$ are there which satisfy $\alpha \neq \beta$ and $\alpha = \beta \pmod{n}$? Well, there are $p$ choices for $\alpha$, and for each one the number of values of $\beta$ is one less than the size of the residue class of $\alpha$. Each residue class $\pmod{n}$ clearly has size at most $\lceil \frac{p}{n} \rceil$. So the number of such $(\alpha, \beta)$ pairs is

$$\leq p \left( \lceil \frac{p}{n} \rceil - 1 \right) \leq \frac{p(p-1)}{n}.$$

Plugging this into (***), gives

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

which is exactly the condition for 2-universality. □

**Ex:** Why did we work with $\mathbb{Z}_p$ for a prime $p \geq m$, rather than directly with $\mathbb{Z}_m = U$? □

**Ex:** Consider the family $\mathcal{H'} = \{ h_{ab} : a, b \in \mathbb{Z}_p \}$ (i.e., we have removed the restriction that $a \neq 0$). Is this family also 2-universal? □