Based on earlier notes by Alistair Sinclair/Manuel Blum/Douglas Young.

Randomized Hashing

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 $\mathcal{U} = \{0, 1, \ldots, m-1\}$.

We will hash the keys in S into a hash table $\mathcal{T} = \{0, 1, \ldots, n-1\}$ using a hash function $h : \mathcal{U} \to \mathcal{T}$. I.e., we store element $x \in S$ at location $h(x)$ of $\mathcal{T}$. 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 $\mathcal{T}$ 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 $h : \mathcal{U} \to \mathcal{T}$ must map at least $\frac{m}{n}$ elements of $\mathcal{U}$ to some location in $\mathcal{T}$. Deduce that, if m is much larger than n, then there will be sets $S \subseteq \mathcal{U}$ that are all mapped by h to a *single* location in $\mathcal{T}$. $\square$

Ex: A hash function h is said to be perfect for a set $S \subseteq \mathcal{U}$ 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 $(\frac{m}{n})^n$. Compare this with the total number of such sets S. $\square$

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

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 $\mathbf{E}[Z]$? 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 $\mathbf{E}[Z] = \sum_{y\in S} \mathbf{E}[Z_y]$. Since h is chosen from a 2-universal family, we have $\mathbf{E}[Z_y] \le \Pr[h(x) = h(y)] \le \frac{1}{n}$. Hence $\mathbf{E}[Z] \le \frac{s}{n}$. This completes the proof. $\square$

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 $n \ge s$. 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 <u>all</u> functions $h : \mathcal{U} \to \mathcal{T}$

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 <u>small</u> and that is efficient to work with.

<u>A 2-universal family</u>

Let p be a prime with $p \ge m$. Since for any m there exists a prime between m and $2m$, we can assume that $p \le 2m$.

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

For $a, b \in \mathbf{Z}_p$, define the function $h_{a,b} : \mathbf{Z}_p \to \mathcal{T}$ by

$$h_{a,b}(x) = ((ax + b) \bmod p) \bmod n.$$

Our hash family will be $\mathcal{H} = \{h_{a,b} : a, b \in \mathbf{Z}_p, a \ne 0\}$.

The key point here is that $\mathcal{H}$ contains only $p(p-1)$ functions (why?), and specifying a function $h_{a,b}$ requires only $O(\log p) = O(\log m)$ bits. (Compare the $O(m \log n)$ bits required for a purely random function.) To choose $h_{a,b} \in \mathcal{H}$, we simply select a, b independently and u.a.r. from $\mathbf{Z}_p - \{0\}$ and $\mathbf{Z}_p$ respectively. Moreover, evaluting $h_{a,b}(x)$ 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 $x, y \in \mathbf{Z}_p$ with $x \ne y$. We need to figure out $\Pr[h_{a,b}(x) = h_{a,b}(y)]$, where $h_{a,b}$ is chosen u.a.r. from $\mathcal{H}$.
For convenience, define $g_{a,b}(x) = (ax + b) \bmod p$, so that $h_{a,b}(x) = g_{a,b}(x) \bmod n$.

How can $h_{a,b}(x) = h_{a,b}(y)$? For this to happen, we must have

$$g_{a,b}(x) = g_{a,b}(y) \bmod n. \qquad (*)$$

So let's focus first on $g_{a,b}$. Let α, β be any numbers in $\mathbf{Z}_p$. I claim that

$$\Pr[g_{a,b}(x) = \alpha \wedge g_{a,b}(y) = \beta] = \begin{cases} 0 & \text{if } \alpha = \beta; \\ \frac{1}{p(p-1)} & \text{otherwise.} \end{cases} \qquad (**)$$

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

$$ax + b = \alpha \qquad \text{and} \qquad ay + b = \beta.$$

But these two linear equations in the two unknowns a, b have a *unique* solution in $\mathbf{Z}_p$, 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 $g_{a,b}$ that gives us the values $g_{a,b}(x) = \alpha$ and $g_{a,b}(y) = \beta$ (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 $h_{a,b}(x) = h_{a,b}(y)$ if and only if $\alpha = \beta \bmod n$, i.e., α and β must be in the same residue class mod n. And from $(**)$ we see that all such pairs with $\alpha \neq \beta$ have probability $\frac{1}{p(p-1)}$. So we have

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

How many pairs (α, β) are there which satisfy $\alpha \neq \beta$ and $\alpha = \beta \bmod n$? Well, there are p choices for α, and for each one the number of values of β is one less than the size of the residue class of α. Each residue class mod n clearly has size at most $\lceil \frac{p}{n} \rceil$. So the number of such (α, β) pairs is $\leq p(\lceil \frac{p}{n} \rceil - 1) \leq \frac{p(p-1)}{n}$.

Plugging this into $(***)$ gives

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

which is exactly the condition for 2-universality. $\square$

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

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

Hashing in Worst-Case Constant Time

We have seen that, if the hash function h is chosen u.a.r. from a 2-universal family $\mathcal{H}$, then the expected cost for any dictionary operation on a set S is $1 + \frac{|S|}{n} \leq 2$, provided we ensure that the the hash table size n is at least as large as $|S|$. Therefore, the expected cost of any sequence of k dictionary operations is at most $2k$. Unfortunately this says nothing about the cost of the worst operation in the sequence: there might be a small number of operations that are very slow.

Ex: To see this point, give a simple example of positive integer-valued random variables $X_1, X_2, \ldots, X_k$ each of which has constant expectation but such that, with probability 1, $\mathbf{E}[\max_i X_i]$ grows with k. Obviously your random variables will not be independent. $\square$

There is a clever way of getting around this problem, which was discovered by Fredman, Komlós and Szemerédi in 1984. Their scheme achieves constant time for *every* operation in a sequence, though it works only in the more restricted model where we assume that all INSERT operations precede all FINDs and there are no DELETEs. (This is usually referred to as a static dictionary.)

The high-level idea is known as "double hashing": we apply two hash functions to each key. More precisely, we first apply a *primary* hash function to all keys. Then, for each group of keys that collide under the primary hash function, we apply a *secondary* hash function to separate that group. All hash functions used will be chosen from a 2-universal family $\mathcal{H}$ such as the one defined on page 2 of Note 9.

First, let's consider the secondary hash functions. Suppose we want to completely separate a set of b keys, i.e., we want to hash them with *no collisions*. To be precise, let's define a collision to be a pair of keys $\{x, y\}$ such that $h(x) = h(y)$.

Claim 1: If we use a 2-universal hash family to hash a set of b keys into a table of size b^2, then $\Pr[\text{there is a collision}] \le \frac{1}{2}$.

Proof: By definition of a 2-universal family, we know that

$$\Pr[h(x) = h(y)] \le \tfrac{1}{b^2} \qquad \text{for every pair } \{x, y\},$$

since the table size is $n = b^2$. But there are exactly $\binom{b}{2}$ pairs $\{x, y\}$, so by linearity of expectation we get

$$\mathbf{E}[\#\text{ collisions}] \le \binom{b}{2} \times \tfrac{1}{b^2} = \tfrac{b(b-1)}{2b^2} < \tfrac{1}{2}.$$

Now by Markov's inequality we conclude the claimed result. $\square$

Ex: How does this result relate to the fact about collisions we established about balls and bins on page 3 of Note 1? $\square$

The above claim tells us how to do the secondary hashing: suppose the primary hash function creates "buckets" (i.e., groups of colliding keys) of sizes $b_1, b_2, \ldots, b_r$ (for some r). Then for each group i, we pick a secondary hash function h_i from a 2-universal family with table size b_i^2, where b_i is the size of the group. This function hashes the keys in group i to a secondary table of size b_i^2, and by Claim 1 it will create a collision with probability at most $\frac{1}{2}$. So we can keep trying random secondary functions until we find one that works (i.e., has no collisions) for the group; the expected number of trials we'll need is at most 2.

Once we have all the required secondary functions, we store them in the cells of the primary hash table. To hash a key, we first use the primary hash function to identify its group (cell in the primary table), and then use the secondary function stored there to hash the key to its secondary table, where the key itself is stored.

Let's turn now to the primary hash function. What properties do we need this function to have? Well, notice that the size of the above data structure will be proportional to the size of the primary table plus $\sum_i b_i^2$ (which is the total size of all the secondary tables). So, to be consistent with our requirement that the data structure size be proportional to the size of the stored set S, we need that the quantity $\sum_i b_i^2$ (the sum of squares of the bucket sizes) is at most const $\times |S|$, while the size of the primary table is also of this order of magnitude.

Once again, this is easy to achieve using 2-universal hash functions:

Claim 2: If we use a 2-universal hash family to hash a set S into a table of size $n \geq |S|$, then the bucket sizes b_i satisfy

$$\Pr[\sum_i b_i^2 \geq 4|S|\,] \leq \tfrac{1}{2}.$$

Proof: Note that $\sum_i b_i^2$ is exactly equal to $2 \times (\#\text{collisions}) + \sum_i b_i$. (Why?) Using linearity of expectation, and the fact that $\Pr[h(x) = h(y)] \leq \frac{1}{n}$, we therefore get

$$\mathbf{E}[\sum_i b_i^2] = 2\mathbf{E}[\#\text{collisions}] + \mathbf{E}[\sum_i b_i] \leq 2\binom{|S|}{2} \times \tfrac{1}{n} + |S| \leq 2|S|,$$

using the fact that $|S| \leq n$. Then a simple application of Markov's inequality gives us the claim. $\square$

Just as for the secondary hash functions, we can pick primary hash functions from a 2-universal family with table size $n \approx |S|$ until we find one for which $\sum_i b_i^2 \leq 4|S|$. By Claim 2, we expect to have to try at most twice.

Putting it all together, once we have picked primary and secondary hash functions satisfying Claims 2 and 1 respectively, we have constructed a data structure that supports *any* FIND operation on our set S in constant time. The size of the data structure is proportional to $|S|$, and the expected time to construct it is also proportional to $|S|$.