★ 11.3.3 Universal hashing

If a malicious adversary chooses the keys to be hashed by some fixed hash function, then the adversary can choose n keys that all hash to the same slot, yielding an average retrieval time of $\Theta(n)$. Any fixed hash function is vulnerable to such terrible worst-case behavior; the only effective way to improve the situation is to choose the hash function *randomly* in a way that is *independent* of the keys that are actually going to be stored. This approach, called ***universal hashing***, can yield provably good performance on average, no matter which keys the adversary chooses.

In universal hashing, at the beginning of execution we select the hash function at random from a carefully designed class of functions. As in the case of quicksort, randomization guarantees that no single input will always evoke worst-case behavior. Because we randomly select the hash function, the algorithm can behave differently on each execution, even for the same input, guaranteeing good average-case performance for any input. Returning to the example of a compiler's symbol table, we find that the programmer's choice of identifiers cannot now cause consistently poor hashing performance. Poor performance occurs only when the compiler chooses a random hash function that causes the set of identifiers to hash poorly, but the probability of this situation occurring is small and is the same for any set of identifiers of the same size.

Let $\mathcal{H}$ be a finite collection of hash functions that map a given universe U of keys into the range $\{0, 1, \ldots, m-1\}$. Such a collection is said to be ***universal*** if for each pair of distinct keys $k, l \in U$, the number of hash functions $h \in \mathcal{H}$ for which $h(k) = h(l)$ is at most $|\mathcal{H}|/m$. In other words, with a hash function randomly chosen from $\mathcal{H}$, the chance of a collision between distinct keys k and l is no more than the chance $1/m$ of a collision if $h(k)$ and $h(l)$ were randomly and independently chosen from the set $\{0, 1, \ldots, m-1\}$.

The following theorem shows that a universal class of hash functions gives good average-case behavior. Recall that n_i denotes the length of list $T[i]$.

Theorem 11.3

Suppose that a hash function h is chosen randomly from a universal collection of hash functions and has been used to hash n keys into a table T of size m, using chaining to resolve collisions. If key k is not in the table, then the expected length $\mathrm{E}[n_{h(k)}]$ of the list that key k hashes to is at most the load factor $\alpha = n/m$. If key k is in the table, then the expected length $\mathrm{E}[n_{h(k)}]$ of the list containing key k is at most $1 + \alpha$.

Proof We note that the expectations here are over the choice of the hash function and do not depend on any assumptions about the distribution of the keys. For each pair k and l of distinct keys, define the indicator random variable

$X_{kl} = \mathrm{I}\{h(k) = h(l)\}$. Since by the definition of a universal collection of hash functions, a single pair of keys collides with probability at most $1/m$, we have $\Pr\{h(k) = h(l)\} \le 1/m$. By Lemma 5.1, therefore, we have $\mathrm{E}[X_{kl}] \le 1/m$.

Next we define, for each key k, the random variable Y_k that equals the number of keys other than k that hash to the same slot as k, so that

$$Y_k = \sum_{\substack{l \in T \\ l \ne k}} X_{kl} .$$

Thus we have

$$\begin{aligned}
\mathrm{E}[Y_k] &= \mathrm{E}\left[\sum_{\substack{l \in T \\ l \ne k}} X_{kl}\right] \\
&= \sum_{\substack{l \in T \\ l \ne k}} \mathrm{E}[X_{kl}] \qquad \text{(by linearity of expectation)} \\
&\le \sum_{\substack{l \in T \\ l \ne k}} \frac{1}{m} .
\end{aligned}$$

The remainder of the proof depends on whether key k is in table T.

- If $k \notin T$, then $n_{h(k)} = Y_k$ and $|\{l : l \in T \text{ and } l \ne k\}| = n$. Thus $\mathrm{E}[n_{h(k)}] = \mathrm{E}[Y_k] \le n/m = \alpha$.
- If $k \in T$, then because key k appears in list $T[h(k)]$ and the count Y_k does not include key k, we have $n_{h(k)} = Y_k + 1$ and $|\{l : l \in T \text{ and } l \ne k\}| = n - 1$. Thus $\mathrm{E}[n_{h(k)}] = \mathrm{E}[Y_k] + 1 \le (n-1)/m + 1 = 1 + \alpha - 1/m < 1 + \alpha$. ■

The following corollary says universal hashing provides the desired payoff: it has now become impossible for an adversary to pick a sequence of operations that forces the worst-case running time. By cleverly randomizing the choice of hash function at run time, we guarantee that we can process every sequence of operations with a good average-case running time.

Corollary 11.4

Using universal hashing and collision resolution by chaining in an initially empty table with m slots, it takes expected time $\Theta(n)$ to handle any sequence of n INSERT, SEARCH, and DELETE operations containing $O(m)$ INSERT operations.

Proof Since the number of insertions is $O(m)$, we have $n = O(m)$ and so $\alpha = O(1)$. The INSERT and DELETE operations take constant time and, by Theorem 11.3, the expected time for each SEARCH operation is $O(1)$. By linearity of expectation, therefore, the expected time for the entire sequence of n operations is $O(n)$. Since each operation takes $\Omega(1)$ time, the $\Theta(n)$ bound follows. ■

Designing a universal class of hash functions

It is quite easy to design a universal class of hash functions, as a little number theory will help us prove. You may wish to consult Chapter 31 first if you are unfamiliar with number theory.

We begin by choosing a prime number p large enough so that every possible key k is in the range 0 to $p-1$, inclusive. Let $\mathbb{Z}_p$ denote the set $\{0, 1, \ldots, p-1\}$, and let $\mathbb{Z}_p^*$ denote the set $\{1, 2, \ldots, p-1\}$. Since p is prime, we can solve equations modulo p with the methods given in Chapter 31. Because we assume that the size of the universe of keys is greater than the number of slots in the hash table, we have $p > m$.

We now define the hash function h_{ab} for any $a \in \mathbb{Z}_p^*$ and any $b \in \mathbb{Z}_p$ using a linear transformation followed by reductions modulo p and then modulo m:

$$h_{ab}(k) = ((ak + b) \bmod p) \bmod m \ . \tag{11.3}$$

For example, with $p = 17$ and $m = 6$, we have $h_{3,4}(8) = 5$. The family of all such hash functions is

$$\mathcal{H}_{pm} = \left\{h_{ab} : a \in \mathbb{Z}_p^* \text{ and } b \in \mathbb{Z}_p\right\} \ . \tag{11.4}$$

Each hash function h_{ab} maps $\mathbb{Z}_p$ to $\mathbb{Z}_m$. This class of hash functions has the nice property that the size m of the output range is arbitrary—not necessarily prime—a feature which we shall use in Section 11.5. Since we have $p-1$ choices for a and p choices for b, the collection $\mathcal{H}_{pm}$ contains $p(p-1)$ hash functions.

Theorem 11.5
The class $\mathcal{H}_{pm}$ of hash functions defined by equations (11.3) and (11.4) is universal.

Proof Consider two distinct keys k and l from $\mathbb{Z}_p$, so that $k \neq l$. For a given hash function h_{ab} we let

$$\begin{aligned} r &= (ak + b) \bmod p \ , \\ s &= (al + b) \bmod p \ . \end{aligned}$$

We first note that $r \neq s$. Why? Observe that

$$r - s \equiv a(k - l) \pmod{p} \ .$$

It follows that $r \neq s$ because p is prime and both a and $(k - l)$ are nonzero modulo p, and so their product must also be nonzero modulo p by Theorem 31.6. Therefore, when computing any $h_{ab} \in \mathcal{H}_{pm}$, distinct inputs k and l map to distinct values r and s modulo p; there are no collisions yet at the "mod p level." Moreover, each of the possible $p(p-1)$ choices for the pair (a,b) with $a \neq 0$ yields a *different* resulting pair (r,s) with $r \neq s$, since we can solve for a and b given r and s:

$$\begin{aligned} a &= \big((r-s)((k-l)^{-1} \bmod p)\big) \bmod p \ , \\ b &= (r - ak) \bmod p \ , \end{aligned}$$

where $((k-l)^{-1} \bmod p)$ denotes the unique multiplicative inverse, modulo p, of $k-l$. Since there are only $p(p-1)$ possible pairs (r,s) with $r \neq s$, there is a one-to-one correspondence between pairs (a,b) with $a \neq 0$ and pairs (r,s) with $r \neq s$. Thus, for any given pair of inputs k and l, if we pick (a,b) uniformly at random from $\mathbb{Z}_p^* \times \mathbb{Z}_p$, the resulting pair (r,s) is equally likely to be any pair of distinct values modulo p.

Therefore, the probability that distinct keys k and l collide is equal to the probability that $r \equiv s \pmod{m}$ when r and s are randomly chosen as distinct values modulo p. For a given value of r, of the $p-1$ possible remaining values for s, the number of values s such that $s \neq r$ and $s \equiv r \pmod{m}$ is at most

$$\begin{aligned} \lceil p/m \rceil - 1 &\leq ((p+m-1)/m) - 1 \quad \text{(by inequality (3.6))} \\ &= (p-1)/m \ . \end{aligned}$$

The probability that s collides with r when reduced modulo m is at most $((p-1)/m)/(p-1) = 1/m$.

Therefore, for any pair of distinct values $k, l \in \mathbb{Z}_p$,

$$\Pr\{h_{ab}(k) = h_{ab}(l)\} \leq 1/m \ ,$$

so that $\mathcal{H}_{pm}$ is indeed universal. ∎

Exercises

11.3-1

Suppose we wish to search a linked list of length n, where each element contains a key k along with a hash value $h(k)$. Each key is a long character string. How might we take advantage of the hash values when searching the list for an element with a given key?

11.3-2

Suppose that we hash a string of r characters into m slots by treating it as a radix-128 number and then using the division method. We can easily represent the number m as a 32-bit computer word, but the string of r characters, treated as a radix-128 number, takes many words. How can we apply the division method to compute the hash value of the character string without using more than a constant number of words of storage outside the string itself?