1.3 Hash functions

In this section, we discuss some issues regarding the design of good hash functions and then present three schemes for their creation: hashing by division, hashing by multiplication, and universal hashing.

What makes a good hash function?

A good hash function satisfies (approximately) the assumption of simple uniform hashing: each key is equally likely to hash to any of the m slots. More formally, let us assume that each key is drawn independently from U according to a probability distribution P; that is, P(k) is the probability that k is drawn. Then the assumption of simple uniform hashing is that

$$\sum_{k:h(k)=j} P(k) = \frac{1}{m} \qquad \text{for } j = 0, 1, \ldots, m-1 \,. \tag{1.1}$$

Unfortunately, it is generally not possible to check this condition, since P is usually unknown.

Sometimes (rarely) we do know the distribution P. For example, suppose the keys are known to be random real numbers k independently and uniformly distributed in the range $0 \le k < 1$. In this case, the hash function

$h(k) = \lfloor km \rfloor$

can be shown to satisfy equation (1.1).

Interpreting keys as natural numbers

Most hash functions assume that the universe of keys is the set N = {0,1,2, . . .} of natural numbers. Thus, if the keys are not natural numbers, a way must be found to interpret them as natural numbers. For example, a key that is a character string can be interpreted as an integer expressed in suitable radix notation. Thus, the identifier pt might be interpreted as the pair of decimal integers (112,116), since p = 112 and t = 116 in the ASCII character set; then, expressed as a radix-128 integer, pt becomes (112 * 128) + 116 = 14452. It is usually straightforward in any given application to devise some such simple method for interpreting each key as a (possibly large) natural number. In what follows, we shall assume that the keys are natural numbers.

1.3.1 The division method

In the division method for creating hash functions, we map a key k into one of m slots by taking the remainder of k divided by m. That is, the hash function is

h(k) = k mod m .

For example, if the hash table has size m = 12 and the key is k = 100, then h(k) = 4. Since it requires only a single division operation, hashing by division is quite fast.

When using the division method, we usually avoid certain values of m. For example, m should not be a power of 2, since if m = 2p, then h(k) is just the p lowest-order bits of k. Unless it is known a priori that the probability distribution on keys makes all low-order p-bit patterns equally likely, it is better to make the hash function depend on all the bits of the key. Powers of 10 should be avoided if the application deals with decimal numbers as keys, since then the hash function does not depend on all the decimal digits of *k*. Finally, it can be shown that when $m = 2^p - 1$ and *k* is a character string interpreted in radix 2^p, two strings that are identical except for a transposition of two adjacent characters will hash to the same value.

Good values for m are primes not too close to exact powers of 2. For example, suppose we wish to allocate a hash table, with collisions resolved by chaining, to hold roughly n = 2000 character strings, where a character has 8 bits. We don't mind examining an average of 3 elements in an unsuccessful search, so we allocate a hash table of size m = 701. The number 701 is chosen because it is a prime near α = 2000/3 but not near any power of 2. Treating each key k as an integer, our hash function would be

h(k) = k mod 701 .

As a precautionary measure, we could check how evenly this hash function distributes sets of keys among the slots, where the keys are chosen from "real" data.

1.3.2 The multiplication method

The multiplication method for creating hash functions operates in two steps. First, we multiply the key k by a constant A in the range 0 < A < 1 and extract the fractional part of kA. Then, we multiply this value by m and take the floor of the result. In short, the hash function is

$h(k) = \lfloor m\,(k\,A \bmod 1)\rfloor$,

where "k A mod 1" means the fractional part of kA, that is, $kA - \lfloor kA \rfloor$.

An advantage of the multiplication method is that the value of m is not critical. We typically choose it to be a power of 2(m = 2p for some integer p-since we can then easily implement the function on most computers as follows. Suppose that the word size of the machine is w bits and that k fits into a single word. Referring to Figure 1.4, we first multiply k by the w-bit integer ⌊ A * 2w ⌋. The result is a 2w-bit value r1 2w + r0, where r1 is the high-order word of the product and r0 is the low-order word of the product. The desired p-bit hash value consists of the p most significant bits of r0.

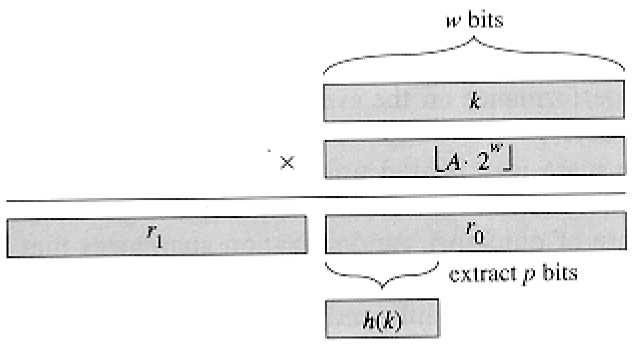

Figure 1.4 The multiplication method of hashing. The w-bit representation of the key k is multiplied by the w-bit value ⌊A.2^w⌋, where 0 < A < 1 is a suitable constant. The p highest-order bits of the lower w-bit half of the product form the desired hash value h(k).

Although this method works with any value of the constant A, it works better with some values than with others. The optimal choice depends on the characteristics of the data being hashed. The choice of A in some detail and suggests that

$$A \approx (\sqrt{5}-1)/2 = 0.6180339887\ldots \qquad (1.2)$$

is likely to work reasonably well.

As an example, if we have k = 123456, m = 10000, and A as in equation (1.2), then

h(k) = ⌊10000 ⋆ (123456 ⋆ 0.61803 mod 1)⌋

= ⌊10000 ⋆ (76300.0041151 mod 1)⌋

= ⌊10000 ⋆ 0.0041151⌋

= ⌊41.151⌋

= 41

Reference :

Text Book-T. H. Cormen, C. E. Leiserson , R. L. Rivest and C. Stein : Introduction to Algorithms, Third Edition ,The MIT Press Cambridge, Massachusetts London, England .