Primes in Division Method of Hashing

Nitin Verma
mathsanew.com

February 6, 2021

Every hashtable uses a hash-function to map an input key to an integer in $\{0, 1, 2, \ldots, m-1\}$, where m is the hashtable's size. For the *Division Method* of hashing, which simply takes mod m of the input integer key, we find that it is generally recommended to use m which is a prime number. In this article, we discuss one reason behind this recommendation.

The original form of input keys can be any data-type including integer, char, string, struct/object containing multiple members, array of another type etc. But they are generally transformed into a non-negative integer for inputting into the hash-function. We will refer this integer form as key $k \geq 0$. Note that during implementation, this key transformation process may be combined with the hash-function computation. But for this discussion, we will say that an integer key k is the input to the hash-function.

For any integer $n \geq 0$, we will use $\mathbb{Z}_n$ to refer to the set of integers $\{0, 1, 2, \ldots, n-1\}$.

One of the most desired properties of a hash-function is that it should distribute the input keys among m slots as uniformly as possible. This helps in reducing the number of collisions among keys. But designing such function can be difficult, just because the input keys are not known in advance. In fact, for any choice of the hash-function, the actual input keys may only consist of all possible keys which map to a few slots. But still, we can try our best in selecting the function based on a very rough idea about the set of all possible keys.

There is no single recommendation about hash-functions, which can work best for all sets of keys. And as we will see below, the recommendation to use a prime m for Division Method too need not work the same in all situations.

Division Method and Key Patterns

In the Division Method of hashing, the hash-function $h(k)$ is simply:

$$h(k) = k \bmod m$$

It can be proved that, if the input keys k are randomly uniformly distributed over any set of mn consecutive integers (for some positive integer n), then their mod m values too will be uniformly distributed over $\mathbb{Z}_m = \{0, 1, 2, \ldots, m-1\}$. That is, their hash-values $h(k)$ will be uniformly distributed over $\mathbb{Z}_m$. In this situation with input keys, m being a prime or not has no importance.

But often it is not possible to be certain that the input keys are such random. They all, or a subset of them, may exhibit some patterns. For example, a subset of input keys may follow a "linear-pattern", where each key has the form $b + ai$, where a and b are fixed integers and i acts as the variable integer. Such pattern can emerge due to a variety of reasons. For example:

1. if keys are pointers, which are all multiples of 8. Here a = 8.
2. if keys are all even or all odd. Here $a = 2$.
3. if keys are prices of goods most of which end at '9', then a large subset of keys will be in form $9 + 10i$.
4. consider the case when the key is formed by combining multiple components, like multiple members of an struct/object or an array. Say there are l components, each an integer: $c_0, c_1, c_2, \ldots, c_{l-1}$. Suppose, they are multiplied by fixed integers r_i to derive the integer key k as:

   $$k = \sum_{i=0}^{l-1} c_i r_i$$

   Say, a group of such keys have all components in common, except one component say c_n. Then, these keys will have k in form $b + (r_n)c_n$, where b is fixed and component c_n varies.

   A common example of this is strings, where each character is a component c_i, last character being c_0. Here, $0 \leq c_i \leq 127$ and each r_i can be chosen r^i, where $r = 128$.

   Suppose a group of string keys have their last few, say n, chars common. Then every such string will have k in form $b + (r^n)i$, where b is the numeric value of the n common chars, $b = \sum_{i=0}^{n-1} c_i r^i$. For example, addresses often have their last part as country name, which would be common among many addresses. Similarly, email-addresses have their last part, the domain, quite common.

Prime m

Now we will understand how choosing m to be a prime number becomes important in such linear-pattern situations.

Consider m input keys in form $b + ai$, with i taking values of any m consecutive integers. The hash-values for these keys are:

$$h(k) = (b + ai) \bmod m = (b + a(i \bmod m)) \bmod m \tag{1}$$

We know that taking mod m of any m consecutive integers (here, i) will simply give: $\{0, 1, 2, \ldots, m-1\}$. So, the hash-values taken by these m keys are simply:

$$(b + ai) \bmod m, \quad i \in \{0, 1, 2, \ldots, m-1\}$$

To know further about these values, we refer to an earlier article titled *Multiples of an Integer Modulo Another Integer*, specifically its Corollaries 3 and 4, which are:

Corollary 1. *For any integers a, b, n with $n > 0$, $a \bmod n \neq 0$, and $g = gcd(a, n)$, the set of integers $A = \{(ai+b) \bmod n : i \in \mathbb{Z}_n\}$ consists of values $b, b+g, b+2g, \ldots, b+((n/g)-1)g$ after taking their mod n.*

Corollary 2. *For any integers a, b, n with $n > 0$, a and n coprime (i.e. $g = gcd(a, n) = 1$), the set of integers $A = \{(ai+b) \bmod n : i \in \mathbb{Z}_n\}$ is same as $\mathbb{Z}_n = \{0, 1, 2, \ldots, n-1\}$.*

Say $gcd(a, m)$ is g. So the hash-values of these m keys are the below m/g distinct values, all taken mod m:

$$b, b+g, b+2g, \ldots, b+(m/g-1)g.$$

Now consider the different values of g:

1. $g = 1$: in this case, the hash-values of the m keys will be all distinct and due to corollary 2, consist of the set: $\{0, 1, 2, \ldots, m-1\}$. So this will be the most uniform distribution for these m keys. $g = 1$ means a and m are coprime.

Since the actual input keys of the hashtable may exhibit multiple such patterns with different a and it is not always possible to predict these patterns, knowing a may not be possible. So, to make a and m coprime, the simpler way is to pick m to be a prime. That would ensure that $gcd(a, m) = 1$ for all a, except when a is a multiple of m.

2. $g = m$: it means, a is a multiple of m. In this case, the hash-values of all m keys will be the same, $b \bmod m$.

Even if m is prime, there is one situation where it will not be effective. That is when a itself is a multiple of m, which makes $g = m$. So, while we choose a prime m, we should try to avoid a value which divides any possible a of the key patterns.

3. $1 < g < m$: note that, for any g, the m input keys will only take m/g distinct hash-values. That is, if $g > 1$, they will be distributed over only a portion $(1/g)$ of the total m slots. Such situation can arise when m is a composite (not prime) and a has a common factor with m, giving $gcd(a, m) > 1$.

In above analysis, we considered m input keys having form $b + ai$, with i among any m consecutive integers. These m keys were found to map to m/g distinct slots, $1 \leq g \leq m$. What can we say about any set S of keys in the same form $(b + ai)$, but i taking arbitrary integer values? Due to equation (1), we know that $h(k)$ values of these keys will depend upon $(i \bmod m)$, not i. But, any $(i \bmod m)$ value comes only from $\{0, 1, 2, \ldots, m - 1\}$. So, the hash-values of these S keys will be nothing but some subset of the hash-values taken by keys with i in $\{0, 1, 2, \ldots, m-1\}$, which is what we discussed above. That means, the keys of S can only map within the m/g slots found above, and so we should try to keep $g = 1$ as discussed.

■