Hash Tables - Part 11

Robin Dawes

February 19, 2021

Doubles down on collision resolution with a technique called double hashing.

.

Double Hashing

Double hashing attempts to combine the best thing about of linear probing (each probe sequence contains all addresses) with the strong point of quadratic probing (reduced primary clustering). The technique is simple: we include a second hash function $h''(k)$, and define

$$h(k,i) = (h'(k) + h''(k) * i) \bmod m$$

Double hashing is effectively a generalization of linear probing, except that instead of having a fixed "step size" that determines how far we jump forward in the hash table on each iteration (in linear probing, the step size is 1), we use the key itself to determine the step size. Since the key is used in two different hash functions to determine the initial address in the probing sequence *and* the step size, the probability that two keys will have exactly the same probing sequence is greatly reduced. This reduces both primary and secondary clustering.

EXAMPLE:

Let $m = 10$, let $h'(k)$ = "sum of the digits of k", and let $h''(k) = k^2$

All three of these choices are quite bad from an objective point of view ... but they provide a simple illustration of the double hashing process.

Consider the probe sequence for $k_1 = 13$

i	$h(k_1, i)$
0	4
1	3
2	2
3	1
...	...

Now consider the probing sequence for $k_2 = 22$

i	$h(k_2, i)$
0	4
1	8
2	2
3	6
...	...

Here we see that even though the probe sequences for k_1 and k_2 start in the same address (4) they are completely dissimilar after that.

We now have potentially m^2 different probe sequences, as opposed to the m probe sequences we get with quadratic probing. It is easy to see that the use of $h''(k)$ reduces primary clustering . But what about the problem of missing empty addresses? We can see that in the example above, the probe sequence for k_2 will only try the even-numbered addresses in T. There is an even worse possibility: if $h''(k) = 0$ for some k then the probe sequence for that key will never move from the initial location given by $h'(k)$.

Unless $h''(k) = 1 \quad \forall\, k$, which would be quite pointless!

However if we can ensure that m and $h''(k)$ are relatively prime for all k, each probe sequence will include all addresses ... and this is quite easy to achieve: for example if m is a power of 2, and $h''(k)$ is odd for all k , then we satisfy the requirement.

This is easy to prove based on material covered in CISC-203.

Making sure $h''(k)$ is always odd is also easy:

```
h"(k):
    x = "some computation based on k, such as k^2, or (k + p1)^p2  where
         p1 and p2 are primes, etc."
```

```
if x % 2 == 1:
    return x
elif x == m-1:   # this test ensures that we never return m
    return x-1
else:
    return x+1
```

There are of course infinitely many other ways to ensure that m and $h''(k)$ are always relatively prime.

Double hashing is considered to be one of the most effective collision resolution methods in use.

Many discussions of hashing suggest that the table size m should always be prime because that reduces some of the potential for clustering when we do the "$\bmod m$" step of the hashing. Using a prime number for m also guarantees that m and $h''(k)$ are relatively prime. But as we have just seen, there are sometimes very good reasons for letting $m = 2^t$ for some integer t. Here is another good reason: if we can perform bit-level operations on integers (which is easy in the "C" family of languages, as well as others), then computing $x \bmod 2^t$ (where x is any integer) is trivial: we simply throw away all but the last t bits of x . This can accelerate the real time efficiency of our hash function.

Including a previous unit of these notes!