Hashing Part 1 – continued

Several students had questions after class and during office hours, and I thought it would be useful to share some of the answers.

Q1: Can you explain why quadratic probing goes bad when m = 12?

Yes, but this will be a long answer. The TLDR summary is "12 has lots of factors and this creates repetition in the quadratic probing sequence" . If you're ok with just knowing that and not worrying about why, skip to the next question.

Here is a more thorough explanation of why 12 (or any number with lots of factors) is a bad choice for a table size when we are using quadratic probing. To keep the arithmetic simple for now, we will continue to use $c_1 = c_2 = 1$. And the end of this answer I will discuss using other values for c_1 and c_2 So the critical step in the insert algorithm is where we calculate the next address to check with $a = (v + c_1 * i + c_2 * i^2)\%m$ and with $m = 12$ and $c_1 = c_2 = 1$ this simplifies to $a = (v + i + i^2)\%12$

Consider the probing sequence when v = 3 (remember that `v = h'(k)`) . I'm just using v = 3 as an example … the argument would be the same no matter what v is.

i	$i + i^2$	$(v + i + i^2)\%12$
0	0	3
1	2	5
2	6	9
3	12	3
4	20	11
5	30	9
6	42	9
7	56	11
8	72	3
9	90	9
10	110	5
11	132	3

We can see that the addresses we check are repeating, which means we are wasting time checking locations we have already examined … and it looks like there are some addresses we never check. Why??? That's going to take us on a bit of a journey back to CISC-203.

Consider the calculation of the next address to be examined.

$$\begin{aligned} a &= (v + i + i^2) \ \% \ 12 \\ &= (v + i * (i + 1)) \ \% \ 12 \\ &= (v + (i * (i + 1) \ \% \ 12)) \ \% \ 12 \end{aligned}$$

If we define $\delta(i) = i * (i + 1) \ \% \ 12$

then we can write $a = (v + \delta(i)) \ \% \ 12$

and that's useful because it makes it very clear that if for some i_1 and some i_2 we find that $\delta(i_1) = \delta(i_2)$ then the address values for iteration i_1 and iteration i_2 will be the same.

Since we are now focusing on $\delta(i)$, let's tabulate that:

i	$i * (i + 1)$	$\delta(i)$
0	0*1 = 0	0
1	1*2 = 2	2
2	2*3 = 6	6
3	3*4 = 12	0
4	4*5 = 20	8
5	5*6 = 30	6
6	6*7 = 42	6
7	7*8 = 56	8
8	8*9 = 72	0
9	9*10 = 90	6
10	10*11 = 110	2
11	11*12 = 132	0

Ok so we can see the δ values are repeating (and we can see that the value of v really only determines the starting point of the probing sequence, not the pattern).

Consider the calculation for i = 5 : $\delta(5) = (5 * 6)\ \%12$

The value of $\delta(5)$ is the remainder when $5 * 6$ is divided by 12.

In other words, $\delta(5) = r$, where $5 * 6 = k * 12 + r$ with $0 \leq r \leq 11$

This actually gives us a lot of information about r :

$$5 * 6 = k * 12 + r$$
$$= k * 2 * 6 + r$$

So

$$r = 5 * 6 - k * 2 * 6$$
$$= (5 - k * 2) * 6$$

This tells us that r must be a multiple of 6! But we also know $0 \leq r \leq 11$.... so the only two possible values for r are $r = 0$ or $r = 6$.... so $\delta(5) = 0$ or $\delta(5) = 6$

And look! Both 0 and 6 have already appeared in the sequence of δ values ($\delta(0) = 0$ and $\delta(2) = 6$) ... so $\delta(5)$ **must** take us back to an address we have already visited!

We got this restriction on $\delta(5)$ by noticing that $5 * 6$ is a multiple of 6 and also $k * 12$ is a multiple of 6. Now we can do the same analysis for $i = 0$ because $0 * 1$ is a multiple of 6, for $i = 2$ because $2 * 3$ is a multiple of 6, for $i = 3$ because $3 * 4$ is a multiple of 6, for $i = 6$ because $6 * 7$ is a multiple of 6, for $i = 8$ because $8 * 9$ is a multiple of 6, for $i = 9$ because $9 * 10$ is a multiple of 6, and for $i = 11$ because $11 * 12$ is a multiple of 6. So $\forall\ i \in \{0, 2, 3, 5, 6, 8, 9, 11\},\quad \delta(i) \in \{0, 6\}$ ... that's a lot of repetition.

Let's consider an even worse value for m: what if $m = 60$? 60 divides by 2, 3, 4, 5, 6, 10, 12, 15, 20 and 30.... so any time $i * (i + 1)$ divides by any of these factors, that will put restrictions on $\delta(i)$ and that will cause repetition in the probe sequence.

The more factors m has, the more repetition there will be in the probe sequence … which strongly supports the idea that we should choose m to have as few factors as possible. In other words we should choose a prime number for m.

Now what if we keep $m = 12$ but choose other values for c_1 and c_2? Let's consider $c_1 = 2$ and $c_2 = 3$

We get

$$a = (v + 2 * i + 3 * i^2) \ \% \ 12$$
$$= (v + i * (2 + 3 * i) \ \%12) \ \% \ 12$$

Now we can define $\delta(i) = i * (2 + 3 * i) \ \% \ 12$ and we get

$a = (v + \delta(i)) \ \% \ 12$, just as before – but now using the new definition of δ.

We can see that if the values of $\delta(i)$ turn out to be multiples of 6, we will have the same repetitive behaviour as before:

i	$i * (2 + 3 * i)$	Multiple of 6?
0	0*0	Yes
1	1*5	No
2	2*8	No
3	3*11	No
4	4*14	No
5	5*17	No
6	6*20	Yes
7	7*23	No
8	8*26	No
9	9*29	No
10	10*32	No
11	11*35	No

Well that looks a lot better! But it's not as good as it looks: for every even value of i, $i * (2 + 3 * i)$ is a multiple of 4 ... and that means that the δ values for these even i values must all be in $\{0, 4, 8\}$. We've removed one pattern but created another.

You can experiment with other values of c_1 and c_2. It's hard to find a good combination.

Bottom line: choosing $m = 12$ makes it hard to get quadratic probing to work nicely ... and the same is true whenever m has lots of factors. As stated above, quadratic probing is at its best when m is a prime number.

Q2: Why do you say that using $h(k) = k \ \% \ m$ is a bad hashing function?

One of the goals when choosing a hashing function is to make use of all the information in the key. Using $k \ \% \ m$ can sometimes fail to do this. A simple example is when m is 10 (I know I have argued elsewhere that we should choose a prime number for m … but for this moment I am going to ignore my own advice.)

If m is 10, then we are effectively throwing away all digits of k except the last one.

Similarly, if m = 20, 25, 50 or 100 then the only digits of k that contribute to $k \ \% \ m$ are the last two ($abcdef \ \% \ 25 = ef \ \% \ 25$, etc)

Those observations are trivial but it is possible to go further. If you slogged through the answer to the previous question, you saw an argument that can be adapted to show that there can be an issue when k and m have common factors – this can lead to some addresses in T being more "popular" than others … ie they can be the target of more than their share of keys … which leads to more collisions.

In the real world keys often have patterns and similarities. For example if the keys are student numbers from a particular class, most of the students in the class will be in the same year so the first few digits of their numbers will be very similar. Sometimes using $k \ \% \ m$ as our hashing function can translate those similarities into collisions in the table.

Q3: What about defining h(k) = "sum of digits of k" % m? (So with $m = 13$, $h(7428) = (7 + 4 + 2 + 8) \ \% \ 13 = 21 \ \% \ 13 = 8$)

That uses all the information in the key, so is it a good hash function?

It looks like it might be, but it hides a subtle flaw! Here's an example:

Suppose we need to store 2000 key values, and each key value is a 5-digit integer. We will clearly need a table T with at least 2000 addresses … the exact size of the table does not concern us right now.

Think about the keys: the smallest is 00000 and the largest is 99999. That means that the "sum of digits of k" operation can only give values from 0+0+0+0+0 = 0 up to 9+9+9+9+9 = 45.

That means that **every one of the 2000 keys** will get hashed onto the addresses from 0 up to 45 … **that's an average of more than 40 keys per address**! The number of collisions will be huge, and the probe sequences to resolve the collisions will be extremely long. Disaster!

Q4: What $would$ be a good hash function for the scenario in the example just above?

This is a topic we will discuss in class. There are some very effective hash functions available. Here's a really simple one:

compute k^2

throw away the last couple of digits of k^2 and the first couple of digits of k^2

this leaves a number with about 6 digits … call it x

compute $x \ \% \ m$

As you might guess, this is called the "mid-square method". I'm not expecting you to know this for Friday's test.