Quadratic Probing

Quadratic probing is similar to linear probing except that instead of \( f(i) = i \), we use
\[
    f(i) = c_1 \cdot i + c_2 \cdot i^2,
\]
where \( c_1 \) and \( c_2 \) are constants (usually but not always positive integers).

The algorithms we developed for linear probing (using "empty" and "deleted" flag values) need only the new \( f(i) \) function to be added.

Quadratic_Probing_Insert(k):

\[
\begin{align*}
&i = 0 \\
&v = h'(k) \\
&a = v \\
&\text{while (}i < m\text{) and (}T[a]\text{ not "empty") and (}T[a]\text{ not "deleted")}:
&\hspace{1cm} i \leftarrow i + 1 \\
&\hspace{1cm} a = (v + c1*i + c2*i^2) \mod m \\
&\text{if (}T[a]\text{ is "empty") or (}T[a]\text{ is "deleted"):} \\
&\hspace{1cm} T[a] = k \\
&\text{else:} \\
&\hspace{1cm} \text{report "insert failed"}
\end{align*}
\]

Quadratic_Probing_Search(k):

\[
\begin{align*}
&i = 0 \\
&v = h'(k) \\
&a = v \\
&\text{while (}i < m\text{) and (}T[a]\text{ not "empty") and (}T[a]\text{ != k):} \\
&\hspace{1cm} i \leftarrow i + 1 \\
&\hspace{1cm} a = (v + c1*i + c2*i^2) \mod m \\
&\text{if } T[a] = k: \\
&\hspace{1cm} \text{report "found it"} \\
&\text{else:} \\
&\hspace{1cm} \text{report "search failed"}
\end{align*}
\]

Quadratic_Probing_Delete(k):

\[
\begin{align*}
&i = 0 \\
&v = h'(k) \\
&a = v \\
&\text{while (}i < m\text{) and (}T[a]\text{ not "empty") and (}T[a]\text{ != k):} \\
&\hspace{1cm} i \leftarrow i + 1 \\
&\hspace{1cm} a = (v + c1*i + c2*i^2) \mod m \\
&\text{if } T[a] = k: \\
&\hspace{1cm} T[a] = \text{"deleted"} \\
&\hspace{1cm} \text{report "successful deletion"} \\
&\text{else:} \\
&\hspace{1cm} \text{report "search failed - could not delete"}
\end{align*}
\]
Quadratic probing greatly reduces the effect of primary clustering. To illustrate this, consider a simple example: let $c_1 = c_2 = 1$, and let $m = 11$. Let $k_1$ and $k_2$ be two keys. Suppose $h'(k_1) = 0$. Then $k_1$'s probe sequence is

<table>
<thead>
<tr>
<th>$i$</th>
<th>$h(k_1, i)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

(Check to make sure you understand how the values in this probe sequence are computed!)

Now suppose $h'(k_2) = 2$. Then $k_2$'s probe sequence is

<table>
<thead>
<tr>
<th>$i$</th>
<th>$h(k_2, i)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Even though the probe sequences both contain 2, they go off in different directions after that. Note that they also both contain the value 0 – in $k_1$'s probe sequence it is followed by 2. What is it followed by in $k_2$'s probe sequence?

In general, two probe sequences may hit the same address at any point, but then hit different addresses after that. This greatly reduces the problem of primary clustering – compare this to linear probing, in which two probe sequences are locked together as soon as they share a common value.

Note that with quadratic probing there is still a problem with what is called **secondary clustering**: if $h'(k_1) = h'(k_2)$, the probe sequences for $k_1$ and $k_2$ will be identical. Thus there are only $m$ different probe sequences, out of a possible $m!$ sequences in which we could conceivably search the table. Secondary clustering is much less of a problem than primary clustering.
However, quadratic probing has a potentially much bigger problem: unless \( m, \ c_1 \) and \( c_2 \) are carefully chosen, a probe sequence may only include a subset of the possible addresses. For example, let \( m = 12, c_1 = 1 \) and \( c_2 = 1 \). Suppose \( h'(k_1) = 0 \). The probe sequence for \( k_1 \) is 0, 2, 6, 0, 8, 6, 8, 0, 6 etc. ... we seem to be trapped in repeated visits to a very small set of addresses. In fact it is easy to see that this probe sequence will never contain any odd addresses: we have

\[
h(k_1, i) = (h'(k_1) + i + i^2) \mod 12
= (0 + i \times (i + 1)) \mod 12
= (i \times (i + 1)) \mod 12
\]

and since \( i \times (i + 1) \) is always even, \( (i \times (i + 1)) \mod 12 \) will also always be even – so this probe sequence will never contain any odd addresses. (Aren’t you glad we did all that modular arithmetic in CISC-203?) It is a bit more challenging to determine whether or not 4 and/or 10 ever occurs in the probe sequence we have started to write out in this example – I leave that to you as an exercise for a rainy day with nothing good on TV.

Why is this important? Suppose we are attempting to insert \( k_1 \) into the hash table, and all the even addresses are full but all the odd addresses are empty. Our insert attempt will fail because \( k_1 \)’s probe sequence never looks at the odd addresses – so we can’t insert the new data even though the table is half empty. This is not good!

You may have noticed a difference between the two examples we have done. In the first one we let \( m = 11 \), and things worked out ok. In the second example we let \( m = 12 \) and things went sideways on us. The difference of course is that 11 is a prime number and 12 is not. As a simple illustration of why this is relevant, when we are computing the expression

\( c_1 \times i + c_2 \times i^2 \) ... which we can write as \( (c_1 + c_2 \times i) \times i \) ... there are lots of ways this can turn out to be a multiple of 12 (for example, the first term can be a multiple of 3 and the second term can be a multiple of 4 (or vice versa), or the first term can be a multiple of 2 and the second term can be a multiple of 6 (or vice versa), or either term can be a multiple of 12). And if this expression is a multiple of 12, then \( h(k, i) \) becomes just \( h'(k) \mod 12 \) for this value of \( i \). This means that \( h'(k) \mod 12 \) will show up quite frequently in the probe sequence for \( k \). At the very least we are frequently revisiting an address that we have already looked at (which is a waste of time), and at worst there is a big risk that the probe sequence will contain even more restrictive patterns such as the one we saw above.

By contrast, there are relatively few ways that \( c_1 \times i + c_2 \times i^2 \) (that is, \( (c_1 + c_2 \times i) \times i \)) can turn out to be a multiple of 11: it only happens when one or both of the terms are themselves
multiples of 11. Thus with a table size of 11 we are less likely to see probe sequences that return to their starting points over and over, such as we saw for a table of size 12.

This is just a tiny step towards a proper discussion of the best way to choose the size of your hash table, but it suggests a solid fundamental idea: **probe sequences will be less likely to fall into patterns if we let m be a prime number.**

A full discussion of the best way to choose $m, c_1$ and $c_2$ for quadratic probing is beyond the scope of CISC-235 ... but I encourage you to do some independent reading on this topic. The number theory we studied in CISC-203 will help you.