Hashing with linear probing (part 1)

The main advantage of hashing with linear probing instead of linked lists is a large reduction in space requirements. There are no linked lists; instead the elements of the set are kept directly in an array $b$. We show the array for an empty set — empty array elements are assumed to contain null.

As usual, an element $e_1$ is hashed to a bucket, and if the bucket contains null, $e_1$ is placed there.

$$
\begin{array}{cccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
\end{array}
$$

(assume $e_1$ hashes to 6)

Handling collisions

If another element $e_2$ hashes to an occupied bucket, a “collision” is said to have occurred, and $e_2$ must be placed elsewhere. In linear probing, it is placed in the next bucket (with wraparound) that is null. We show the placement of two more values that hash to bucket 6 or 7.

$$
\begin{array}{cccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
\end{array}
$$

(assume $e_2, e_3, e_4$ hash to 6 or 7)

In summary: If an element hashes to bucket $h$, the element is placed in the first of the following buckets that contains null.

$$h, (h+1) \mod b, (h+2) \mod b, (h+3) \mod b, \ldots$$

Searching for a value

Searching for an element involves hashing it to a bucket and looking at that bucket and the following buckets in wraparound fashion until either the element is found or null is found, in which case the element is not in the set. Suppose $e_1, e_2,$ and $e_3$ all hashed to 6 and were added as shown. A test of whether $e_3$ is in the set is successful: Linear probing looks in buckets 6, 7, and 8, and $e_3$ is there. A test whether $e_4$ is in the set is not successful.

$$
\begin{array}{cccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
\end{array}
$$

Operation remove

Suppose $e_2$ is removed. Now, a test of whether $e_3$ is in the set is unsuccessful, because linear probing stops as soon as a null bucket is found.

$$
\begin{array}{cccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
\end{array}
$$

Clearly, removing $e_2$ cannot be done simply by setting the bucket to null. Instead, we use a boolean value for each bucket to indicate whether it contains a value. Removing $e_2$ then consists only in setting its boolean to false.

$$
\begin{array}{cccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
\end{array}
$$

Now, testing for the presence of a value in the set consists of testing not only the bucket but the boolean that indicates whether the value is in the set. We give an example. Again, suppose $e_1, e_2,$ and $e_3$ all hash to 6 and were added as shown. Let’s remove $e_2$, leaving the bucket alone but setting the boolean to false (as shown above). Now, a test whether $e_3$ is in the set still says yes because the search stops with a negative answer only when a null bucket is encountered.

©David Gries, 2018