Appendix F
Hash Tables

William Stallings
Copyright 2014

Supplement to
Operating Systems, Eighth Edition
Pearson 2014
http://williamstallings.com/OperatingSystems/

Consider the following problem. A set of N items is to be stored in a table. Each item consists of a label plus some additional information, which we can refer to as the value of the item. We would like to be able to perform a number of ordinary operations on the table, such as insertion, deletion, and searching for a given item by label.

If the labels of the items are numeric, in the range 0 to $M - 1$, then a simple solution would be to use a table of length M. An item with label i would be inserted into the table at location i. As long as items are of fixed length, table lookup is trivial and involves indexing into the table based on the numeric label of the item. Furthermore, it is not necessary to store the label of an item in the table, because this is implied by the position of the item. Such a table is known as a **direct access table**.

If the labels are nonnumeric, then it is still possible to use a direct access approach. Let us refer to the items as A[1],…A[N]. Each item A[i] consists of a label, or key, k_i, and a value v_i. Let us define a mapping function I(k) such that I(k) takes a value between 1 and M for all keys and $I(k_i) \neq I(k_j)$ for any i and j. In this case, a direct access table can also be used, with the length of the table equal to M.

The one difficulty with these schemes occurs if M is much greater than N. In this case, the proportion of unused entries in the table is large, and this is an inefficient use of memory. An alternative would be to use a table of length N and store the N items (label plus value) in the N table entries. In this scheme, the amount of memory is minimized but there is now a processing burden to do table lookup. There are several possibilities:

- **Sequential search:** This brute-force approach is time consuming for large tables.

Table F.1 Average Search Length for One of *N* items in a Table of Length *M*

Technique	Search Length
Direct	1
Sequential	$\frac{M+1}{2}$
Binary	$\log_2 M$
Linear hashing	$\frac{2-N/M}{2-2N/M}$
Hash (overflow with chaining)	$1+\frac{N-1}{2M}$

- **Associative search:** With the proper hardware, all of the elements in a table can be searched simultaneously. This approach is not general purpose and cannot be applied to any and all tables of interest.
- **Binary search:** If the labels or the numeric mapping of the labels are arranged in ascending order in the table, then a binary search is much quicker than sequential (Table F.1) and requires no special hardware.

The binary search looks promising for table lookup. The major drawback with this method is that adding new items is not usually a simple process and will require reordering of the entries. Therefore, binary search is usually used only for reasonably static tables that are seldom changed.

We would like to avoid the memory penalties of a simple direct access approach and the processing penalties of the alternatives listed previously. The most frequently used method to achieve this compromise is **hashing**. Hashing, which was developed in the 1950s, is simple to implement and has

two advantages. First, it can find most items with a single seek, as in direct accessing, and second, insertions and deletions can be handled without added complexity.

The hashing function can be defined as follows. Assume that up to N items are to be stored in a **hash table** of length M, with $M \geq N$, but not much larger than N. To insert an item in the table,

I1. Convert the label of the item to a near-random number n between 0 and $M - 1$. For example, if the label is numeric, a popular mapping function is to divide the label by M and take the remainder as the value of n.

I2. Use n as the index into the hash table.

a. If the corresponding entry in the table is empty, store the item (label and value) in that entry.

b. If the entry is already occupied, then store the item in an overflow area, as discussed subsequently.

To perform table lookup of an item whose label is known,

L1. Convert the label of the item to a near-random number n between 0 and $M - 1$, using the same mapping function as for insertion.

L2. Use n as the index into the hash table.

a. If the corresponding entry in the table is empty, then the item has not previously been stored in the table.

b. If the entry is already occupied and the labels match, then the value can be retrieved.

c. If the entry is already occupied and the labels do not match, then continue the search in the overflow area.

Hashing schemes differ in the way in which the overflow is handled. One common technique is referred to as the **linear hashing** technique and is commonly used in compilers. In this approach, rule I2.b becomes

I2.b. If the entry is already occupied, set $n = n + 1$ (mod M) and go back to step I2.a.

Rule L2.c is modified accordingly.

Figure F.1a is an example. In this case, the labels of the items to be stored are numeric, and the hash table has eight positions ($M = 8$). The mapping function is to take the remainder upon division by 8. The figure assumes that the items were inserted in ascending numerical order, although this is not necessary. Thus, items 50 and 51 map into positions 2 and 3, respectively, and as these are empty, they are inserted there. Item 74 also maps into position 2, but as it is not empty, position 3 is tried. This is also occupied, so the position 4 is ultimately used.

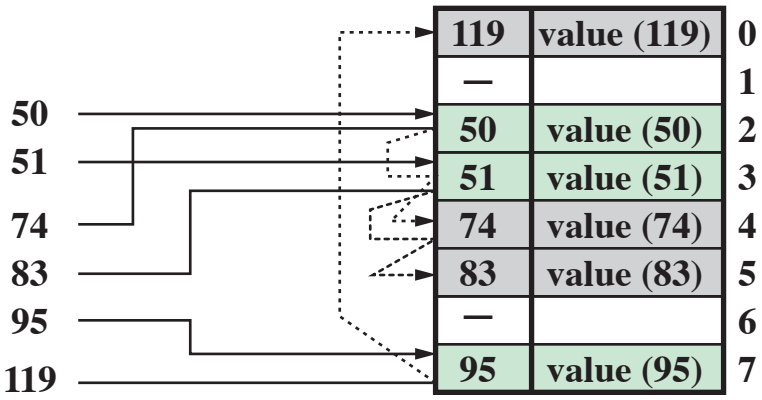
It is not easy to determine the average length of the search for an item in an open hash table because of the clustering effect. An approximate formula was obtained by Schay and Spruth:[1]

$$\text{Average search length} = \frac{2 - r}{2 - 2r}$$

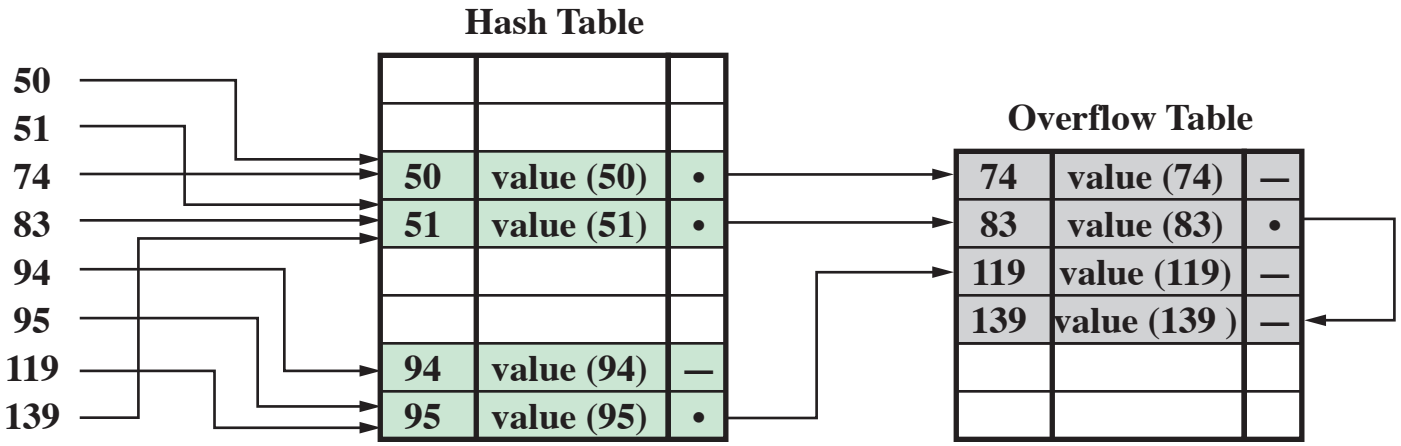
where $r = N/M$. Note that the result is independent of table size and depends only on how full the table is. The surprising result is that with the table 80% full, the average length of the search is still around 3.

---

[1] Schay, G., and Spruth, W. "Analysis of a File Addressing Method." *Communications of the ACM,* August 1962.



(a) Linear rehashing



(b) Overflow with chaining

Figure F.1 Hashing

Even so, a search length of 3 may be considered long, and the linear hashing table has the additional problem that it is not easy to delete items. A more attractive approach, which provides shorter search lengths (Table 8.7) and allows deletions as well as additions, is **overflow with chaining**. This technique is illustrated in Figure F.1b. In this case, there is a separate table into which overflow entries are inserted. This table includes pointers passing down the chain of entries associated with any position in the hash table. In this case, the average search length, assuming randomly distributed data, is

$$\text{Average search length} = 1 + \frac{N-1}{2M}$$

For large values of *N* and *M*, this value approaches 1.5 for *N* = *M*. Thus, this technique provides for compact storage with rapid lookup.