For separate chaining, assuming a load factor of 1, this is one version of the classic **balls and bins problem**: Given N balls placed randomly (uniformly) in N bins, what is the expected number of balls in the most occupied bin? The answer is well known to be $\Theta(\log N / \log \log N)$, meaning that on average, we expect some queries to take nearly logarithmic time. Similar types of bounds are observed (or provable) for the length of the longest expected probe sequence in a probing hash table.

We would like to obtain $O(1)$ worst-case cost. In some applications, such as hardware implementations of lookup tables for routers and memory caches, it is especially important that the search have a definite (i.e., constant) amount of completion time. Let us assume that N is known in advance, so no rehashing is needed. If we are allowed to rearrange items as they are inserted, then $O(1)$ worst-case cost is achievable for searches.

In the remainder of this section we describe the earliest solution to this problem, namely perfect hashing, and then two more recent approaches that appear to offer promising alternatives to the classic hashing schemes that have been prevalent for many years.

5.7.1 Perfect Hashing

Suppose, for purposes of simplification, that all N items are known in advance. If a separate chaining implementation could guarantee that each list had at most a constant number of items, we would be done. We know that as we make more lists, the lists will on average be shorter, so theoretically if we have enough lists, then with a reasonably high probability we might expect to have no collisions at all!

But there are two fundamental problems with this approach: First, the number of lists might be unreasonably large; second, even with lots of lists, we might still get unlucky.

The second problem is relatively easy to address in principle. Suppose we choose the number of lists to be M (i.e., *TableSize* is M), which is sufficiently large to guarantee that with probability at least $\frac{1}{2}$, there will be no collisions. Then if a collision is detected, we simply clear out the table and try again using a different hash function that is independent of the first. If we still get a collision, we try a third hash function, and so on. The expected number of trials will be at most 2 (since the success probability is $\frac{1}{2}$), and this is all folded into the insertion cost. Section 5.8 discusses the crucial issue of how to produce additional hash functions.

So we are left with determining how large M, the number of lists, needs to be. Unfortunately, M needs to be quite large; specifically $M = \Omega(N^2)$. However, if $M = N^2$, we can show that the table is collision free with probability at least $\frac{1}{2}$, and this result can be used to make a workable modification to our basic approach.

Theorem 5.2

If N balls are placed into $M = N^2$ bins, the probability that no bin has more than one ball is less than $\frac{1}{2}$.

Proof

If a pair (i, j) of balls are placed in the same bin, we call that a collision. Let $C_{i,j}$ be the expected number of collisions produced by any two balls (i, j). Clearly the probability that any two specified balls collide is $1/M$, and thus $C_{i,j}$ is $1/M$, since the number of collisions that involve the pair (i, j) is either 0 or 1. Thus the expected number of

collisions in the entire table is $\sum_{(i,j),\ i<j} C_{i,j}$. Since there are $N(N-1)/2$ pairs, this sum is $N(N-1)/(2M) = N(N-1)/(2N^2) < \frac{1}{2}$. Since the expected number of collisions is below $\frac{1}{2}$, the probability that there is even one collision must also be below $\frac{1}{2}$.

Of course, using N^2 lists is impractical. However, the preceding analysis suggests the following alternative: Use only N bins, but resolve the collisions in each bin by using hash tables instead of linked lists. The idea is that because the bins are expected to have only a few items each, the hash table that is used for each bin can be quadratic in the bin size. Figure 5.24 shows the basic structure. Here, the primary hash table has ten bins. Bins 1, 3, 5, and 7 are all empty. Bins 0, 4, and 8 have one item, so they are resolved by a secondary hash table with one position. Bins 2 and 6 have two items, so they will be resolved into a secondary hash table with four (2^2) positions. And bin 9 has three items, so it is resolved into a secondary hash table with nine (3^2) positions.

As with the original idea, each secondary hash table will be constructed using a different hash function until it is collision free. The primary hash table can also be constructed several times if the number of collisions that are produced is higher than required. This scheme is known as **perfect hashing**. All that remains to be shown is that the total size of the secondary hash tables is indeed expected to be linear.

Theorem 5.3

If N items are placed into a primary hash table containing N bins, then the total size of the secondary hash tables has expected value at most $2N$.

Proof

Using the same logic as in the proof of Theorem 5.2, the expected number of pairwise collisions is at most $N(N-1)/2N$, or $(N-1)/2$. Let b_i be the number of items that hash to position i in the primary hash table; observe that b_i^2 space is used for this cell

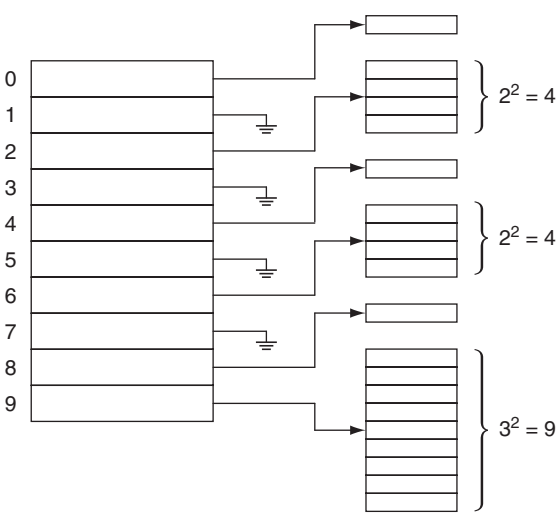

Figure 5.24 Perfect hashing table using secondary hash tables

in the secondary hash table, and that this accounts for b_i $(b_i - 1)/2$ pairwise collisions, which we will call c_i. Thus the amount of space, b_i^2, used for the ith secondary hash table is $2c_i + b_i$. The total space is then $2\sum c_i + \sum b_i$. The total number of collisions is $(N - 1)/2$ (from the first sentence of this proof); the total number of items is of course N, so we obtain a total secondary space requirement of $2(N - 1)/2 + N < 2N$.

Thus, the probability that the total secondary space requirement is more than $4N$ is at most $\frac{1}{2}$ (since, otherwise, the expected value would be higher than $2N$), so we can keep choosing hash functions for the primary table until we generate the appropriate secondary space requirement. Once that is done, each secondary hash table will itself require only an average of two trials to be collision free. After the tables are built, any lookup can be done in two probes.

Perfect hashing works if the items are all known in advance. There are dynamic schemes that allow insertions and deletions (**dynamic perfect hashing**), but instead we will investigate two newer alternatives that appear to be competitive in practice with the classic hashing algorithms.

5.7.2 Cuckoo Hashing

From our previous discussion, we know that in the balls and bins problem, if N items are randomly tossed into N bins, the size of the largest bin is expected to be $\Theta(\log N / \log \log N)$. Since this bound has been known for a long time, and the problem has been well studied by mathematicians, it was surprising when, in the mid 1990s, it was shown that if, at each toss, two bins were randomly chosen and the item was tossed into the more empty bin (at the time), then the size of the largest bin would only be $\Theta(\log \log N)$, a significantly lower number. Quickly, a host of potential algorithms and data structures arose out of this new concept of the "power of two choices."

One of the ideas is **cuckoo hashing**. In cuckoo hashing, suppose we have N items. We maintain two tables, each more than half empty, and we have two independent hash functions that can assign each item to a position in each table. Cuckoo hashing maintains the invariant that an item is always stored in one of these two locations.

As an example, Figure 5.25 shows a potential cuckoo hash table for six items, with two tables of size 5 (these tables are too small, but serve well as an example). Based on the

Table 1	
0	B
1	C
2	
3	E
4	

Table 2	
0	D
1	
2	A
3	
4	F

A: 0, 2
B: 0, 0
C: 1, 4
D: 1, 0
E: 3, 2
F: 3, 4

Figure 5.25 Potential cuckoo hash table. Hash functions are shown on the right. For these six items, there are only three valid positions in Table 1 and three valid positions in Table 2, so it is not clear that this arrangement can easily be found.