b) The storage allocation for hash tables is often somewhat difficult; we have to dedicate a certain area of the memory for use as the hash table, and it may not be obvious how much space should be allotted. If we provide too much memory, we may be wasting storage at the expense of other lists or other computer users; but if we don't provide enough room, the table will overflow. By contrast, the tree search and insertion algorithms deal with trees that grow no larger than necessary. In a virtual memory environment we can keep memory accesses localized if we use tree search or digital tree search, instead of creating a large hash table that requires the operating system to access a new page nearly every time we hash a key.

c) Finally, we need a great deal of faith in probability theory when we use hashing methods, since they are efficient only on the average, while their worst case is terrible! As in the case of random number generators, we can never be completely sure that a hash function will perform properly when it is applied to a new set of data. Therefore hash tables are inappropriate for certain real-time applications such as air traffic control, where people's lives are at stake; the balanced tree algorithms of Sections 6.2.3 and 6.2.4 are much safer, since they provide guaranteed upper bounds on the search time.

History. The idea of hashing appears to have been originated by H. P. Luhn, who wrote an internal IBM memorandum in January 1953 that suggested the use of chaining; in fact, his suggestion was one of the first applications of linked linear lists. He pointed out the desirability of using buckets that contain more than one element, for external searching. Shortly afterwards, A. D. Lin carried Luhn's analysis further, and suggested a technique for handling overflows that used "degenerative addresses"; for example, the overflows from primary bucket 2748 were put in secondary bucket 274; overflows from that bucket went to tertiary bucket 27, and so on, assuming the presence of 10000 primary buckets, 1000 secondary buckets, 100 tertiary buckets, etc. The hash functions originally suggested by Luhn were digital in nature; for example, he combined adjacent pairs of key digits by adding them mod 10, so that 31415926 would be compressed to 4548.

At about the same time the idea of hashing occurred independently to another group of IBMers: Gene M. Amdahl, Elaine M. Boehme, N. Rochester, and Arthur L. Samuel, who were building an assembly program for the IBM 701. In order to handle the collision problem, Amdahl originated the idea of open addressing with linear probing.

Hash coding was first described in the open literature by Arnold I. Dumey, *Computers and Automation* **5**, 12 (December 1956), 6–9. He was the first to mention the idea of dividing by a prime number and using the remainder as the hash address. Dumey's interesting article mentions chaining but not open addressing. A. P. Ershov of Russia independently discovered linear open addressing in 1957 [*Doklady Akad. Nauk SSSR* **118** (1958), 427–430]; he published empirical results about the number of probes, conjecturing correctly that the average number of probes per successful search is < 2 when $N/M < 2/3$.

A classic article by W. W. Peterson, *IBM J. Research & Development* **1** (1957), 130–146, was the first major paper dealing with the problem of searching in large files. Peterson defined open addressing in general, analyzed the performance of uniform probing, and gave numerous empirical statistics about the behavior of linear open addressing with various bucket sizes, noting the degradation in performance that occurred when items were deleted. Another comprehensive survey of the subject was published six years later by Werner Buchholz [*IBM Systems J.* **2** (1963), 86–111], who gave an especially good discussion of hash functions. Correct analyses of Algorithm L were first published by A. G. Konheim and B. Weiss, *SIAM J. Appl. Math.* **14** (1966), 1266–1274; V. Podderjugin, *Wissenschaftliche Zeitschrift der Technischen Universität Dresden* **17** (1968), 1087–1089.

Up to this time linear probing was the only type of open addressing scheme that had appeared in the literature, but another scheme based on repeated random probing by independent hash functions had independently been developed by several people (see exercise 48). During the next few years hashing became very widely used, but hardly anything more was published about it. Then Robert Morris wrote a very influential survey of the subject [*CACM* **11** (1968), 38–44], in which he introduced the idea of random probing with secondary clustering. Morris's paper touched off a flurry of activity that culminated in Algorithm D and its refinements.

It is interesting to note that the word "hashing" apparently never appeared in print, with its present meaning, until the late 1960s, although it had already become common jargon in several parts of the world by that time. The first published appearance of the word seems to have been in H. Hellerman's book *Digital Computer System Principles* (New York: McGraw–Hill, 1967), 152; the only previous occurrence among approximately 60 relevant documents studied by the author as this section was being written was in an unpublished memorandum written by W. W. Peterson in 1961. Somehow the verb "to hash" magically became standard terminology for key transformation during the mid-1960s, yet nobody was rash enough to use such an undignified word in print until 1967!

Later developments. Many advances in the theory and practice of hashing have been made since the author first prepared this chapter in 1972, although the basic ideas discussed above still remain useful for ordinary applications. For example, the book *Design and Analysis of Coalesced Hashing* by J. S. Vitter and W.-C. Chen (New York: Oxford Univ. Press, 1987) discusses and analyzes several instructive variants of Algorithm C.

From a practical standpoint, the most important hash technique invented in the late 1970s is probably the method that Witold Lipski called *linear hashing* [*Proc. 6th International Conf. on Very Large Databases* (1980), 212–223]. Linear hashing — which incidentally has nothing to do with the classical technique of linear probing — allows the number of hash addresses to grow and/or contract gracefully as items are inserted and/or deleted. An excellent discussion of linear

hashing, including comparisons with other methods for internal searching, has been given by Per-Åke Larson in *CACM* **31** (1988), 446–457; see also W. G. Griswold and G. M. Townsend, *Software Practice & Exp.* **23** (1993), 351–367, for improvements when many large and/or small tables are present simultaneously. Linear hashing can also be used for huge databases that are distributed between many different sites on a network [see Litwin, Neimat, and Schneider, *ACM Trans. Database Syst.* **21** (1996), 480–525]. An alternative scheme called *extendible hashing*, which has the property that at most two references to external pages are needed to retrieve any record, was proposed at about the same time by R. Fagin, J. Nievergelt, N. Pippenger, and H. R. Strong [*ACM Trans. Database Syst.* **4** (1979), 315–344]. Both linear hashing and extendible hashing are preferable to the B-trees of Section 6.2.4, when the order of keys is unimportant.

In the theoretical realm, more complicated methods have been devised by which it is possible to guarantee $O(1)$ maximum time per access, with $O(1)$ average amortized time per insertion and deletion, regardless of the keys being examined; moreover, the total storage used at any time is bounded by a constant times the number of items currently present, plus another additive constant. This result, which builds on ideas of Fredman, Komlós, and Szemerédi [*JACM* **31** (1984), 538–544], is due to Dietzfelbinger, Karlin, Mehlhorn, Meyer auf der Heide, Rohnert, and Tarjan [*SICOMP* **23** (1994), 738–761].

EXERCISES

1. [*20*] When the instruction `9H` in Table 1 is reached, how small and how large can the contents of rI1 possibly be, assuming that bytes 1, 2, 3 of K each contain alphabetic character codes less than 30?

2. [*20*] Find a reasonably common English word not in Table 1 that could be added to that table without changing the program.

3. [*23*] Explain why no program beginning with the five instructions

```
LD1  K(1:1)   or   LD1N  K(1:1)
LD2  K(2:2)   or   LD2N  K(2:2)
INC1 a,2
LD2  K(3:3)
J2Z  9F
```

could be used in place of the more complicated program in Table 1, for any constant a, since unique addresses would not be produced for the given keys.

4. [*M30*] How many people should be invited to a party in order to make it likely that there are *three* with the same birthday?

5. [*15*] Mr. B. C. Dull was writing a FORTRAN compiler using a decimal `MIX` computer, and he needed a symbol table to keep track of the names of variables in the FORTRAN program being compiled. These names were restricted to be at most ten characters in length. He decided to use a hash table with $M = 100$, and to use the fast hash function $h(K) =$ leftmost byte of K. Was this a good idea?

6. [*15*] Would it be wise to change the first two instructions of (3) to `LDA K; ENTX 0`?

7. [*HM30*] (*Polynomial hashing.*) The purpose of this exercise is to consider the construction of polynomials $P(x)$ such as (10), which convert n-bit keys into m-bit