EXHIBIT 17

Volume 3 / **Sorting and Searching**

THE ART OF COMPUTER PROGRAMMING

Reading, Massachusetts
Menlo Park, California · London · Amsterdam · Don Mills, Ontario · Sydney

This book is in the
ADDISON-WESLEY SERIES IN
COMPUTER SCIENCE AND INFORMATION PROCESSING

Consulting Editors
RICHARD S. VARGA and MICHAEL A. HARRISON

ISBN 0-201-03803-X

will have the same month and day of birth! In other words, if we select a random function which maps 23 keys into a table of size 365, the probability that no two keys map into the same location is only 0.4927 (less than one-half). Skeptics who doubt this result should try to find the birthday mates at the next large parties they attend. [The birthday paradox apparently originated in unpublished work of H. Davenport; cf. W. W. R. Ball, *Math. Recreations and Essays* (1939), 45. See also R. von Mises, *İstanbul Üniversitesi Fen Fakültesi Mecmuasi* **4** (1939), 145–163, and W. Feller, *An Introduction to Probability Theory* (New York: Wiley, 1950), Section 2.3.]

On the other hand, the approach used in Table 1 is fairly flexible [cf. M. Greniewski and W. Turski, *CACM* **6** (1963), 322–323], and for a medium-sized table a suitable function can be found after about a day's work. In fact it is rather amusing to solve a puzzle like this.

Of course this method has a serious flaw, since the contents of the table must be known in advance; adding one more key will probably ruin everything, making it necessary to start over almost from scratch. We can obtain a much more versatile method if we give up the idea of uniqueness, permitting different keys to yield the same value $f(K)$, and using a special method to resolve any ambiguity after $f(K)$ has been computed.

These considerations lead to a popular class of search methods commonly known as *hashing* or *scatter storage* techniques. The verb "to hash" means to chop something up or to make a mess out of it; the idea in hashing is to chop off some aspects of the key and to use this partial information as the basis for searching. We compute a *hash function* $h(K)$ and use this value as the address where the search begins.

The birthday paradox tells us that there will probably be distinct keys $K_i \neq K_j$ which hash to the same value $h(K_i) = h(K_j)$. Such an occurrence is

HIS	I	IN	IS	IT	NOT	OF	ON	OR	THAT	THE	THIS	TO	WAS	WHICH	WITH	YOU
Contents of rI1 after executing the instruction, given a particular key K																
−8	−9	−9	−9	−9	−15	−16	−16	−16	−23	−23	−23	−23	−26	−26	−26	−28
−8	−9	−9	−9	−9	−15	−16	−16	−16	−23	−23	−23	−23	−26	−26	−26	−28
−7	−17	−2	5	6	−7	−18	−9	−5	−23	−23	−23	−15	−33	−26	−25	−20
−7	−17	−2	5	6	−7	−18	−9	−5	−23	−23	−23	−15	−33	−26	−25	−20
18	−1	29	.	.	25	4	22	30	1	1	1	17	−16	−2	0	12
18	−1	29	5	6	25	4	22	30	1	1	1	17	−16	−2	0	12
18	−1	29	5	6	25	4	22	30	1	1	1	17	−16	−2	0	12
12	.	.	.	.	20	.	.	.	−26	−22	−18	.	−22	−21	−5	8
12	.	.	.	.	20	.	.	.	−26	−22	−18	.	−22	−21	−5	8
.	.	.	.	.	.	.	.	.	−14	−6	2	.	11	−1	29	.
.	.	.	.	.	.	.	.	.	−14	−6	2	.	11	−1	29	.
.	.	.	.	.	.	.	.	.	−14	−6	2	.	11	−1	29	.
.	.	.	.	.	.	.	.	.	−10	.	−2	.	.	−5	11	.
.	.	.	.	.	.	.	.	.	−10	.	−2	.	.	−5	11	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	21	.
12	−1	29	5	6	20	4	22	30	−10	−6	−2	17	11	−5	21	8
12	−1	29	5	6	20	4	22	30	−10	−6	−2	17	11	−5	21	8

called a *collision*, and several interesting approaches have been devised to handle the collision problem. In order to use a scatter table, a programmer must make two almost independent decisions: He must choose a hash function $h(K)$, and he must select a method for collision resolution. We shall now consider these two aspects of the problem in turn.

Hash functions. To make things more explicit, let us assume throughout this section that our hash function h takes on at most M different values, with

$$0 \le h(K) < M, \tag{1}$$

for all keys K. The keys in actual files that arise in practice usually have a great deal of redundancy; we must be careful to find a hash function that breaks up clusters of almost identical keys, in order to reduce the number of collisions.

It is theoretically impossible to define a hash function that creates random data from the nonrandom data in actual files. But in practice it is not difficult to produce a pretty good imitation of random data, by using simple arithmetic as we have discussed in Chapter 3. And in fact we can often do even better, by exploiting the nonrandom properties of actual data to construct a hash function that leads to fewer collisions than truly random keys would produce.

Consider, for example, the case of 10-digit keys on a decimal computer. One hash function that suggests itself is to let $M = 1000$, say, and to let $h(K)$ be three digits chosen from somewhere near the middle of the 20-digit product $K \times K$. This would seem to yield a fairly good spread of values between 000 and 999, with low probability of collisions. Experiments with actual data show, in fact, that this "middle square" method isn't bad, provided that the keys do not have a lot of leading or trailing zeros; but it turns out that there are safer and saner ways to proceed, just as we found in Chapter 3 that the middle square method is not an especially good random number generator.

Extensive tests on typical files have shown that two major types of hash functions work quite well. One of these is based on division, and the other is based on multiplication.

The division method is particularly easy; we simply use the remainder modulo M:

$$h(K) = K \bmod M. \tag{2}$$

In this case, some values of M are obviously much better than others. For example, if M is an even number, $h(K)$ will be even when K is even and odd when K is odd, and this will lead to a substantial bias in many files. It would be even worse to let M be a power of the radix of the computer, since $K \bmod M$ would then be simply the least significant digits of K (independent of the other digits). Similarly we can argue that M probably shouldn't be a multiple of 3 either; for if the keys are alphabetic, two keys which differ from each other only by permutation of letters would then differ in numeric value by a multiple of 3. (This occurs because $10^n \bmod 3 = 4^n \bmod 3 = 1$.) In general, we want to avoid values of M which divide $r^k \pm a$, where k and a are small numbers and r is the radix of the alphabetic character set (usually $r = 64$, 256, or 100),