

Volume 3 / **Sorting and Searching**

H *Stanford University*

**THE ART OF
COMPUTER PROGRAMMING**

SHING COMPANY

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

Second printing, March 1975

Copyright © 1973 by Addison-Wesley Publishing Company, Inc. Philippines copyright 1973 by Addison-Wesley Publishing Company, Inc.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher. Printed in the United States of America. Published simultaneously in Canada. Library of Congress Catalog Card No. 67-26020.

ISBN 0-201-03803-X
BCDEFGHIJ-MA-79876

This book forms a natural part of the text in Chapter 2, because it contains basic structural ideas. The book is only for those students who have a notion of general-purpose programming. But in fact the area of programming is discussing a wide variety of

How are good algorithms designed?
How can given algorithms be made more efficient?
How can a person design an algorithm for the same application?
In what senses can an algorithm be said to be efficient?
How does the efficiency of an algorithm depend on the data?
How can external files be used to store data with large data bases?

Indeed, I believe that the answer to these questions lies somewhere in the content of this book.

This volume contains material that is concerned with sorting and searching. It has been divided chiefly into two parts. The first part also contains supplementary material (Section 5.1). Chapter 6 deals with files; this is subdivided into two parts: one dealing with files of keys, or by digital means, and the other dealing with the problem of secondary storage.

6.4 HASHING

So far we have considered search methods based on comparing the given argument K to the keys in the table, or using its digits to govern a branching process. A third possibility is to avoid all this rummaging around by doing some arithmetical calculation on K , computing a function $f(K)$ which is the location of K and the associated data in the table.

For example, let's consider again the set of 31 English words which we have subjected to various search strategies in Section 6.2.2 and 6.3. Table 1 shows a short MIX program which transforms each of the 31 keys into a unique number $f(K)$ between -10 and 30 . If we compare this method to the MIX programs for the other methods we have considered (e.g., binary search, optimal tree search, trie memory, digital tree search), we find that it is superior from the standpoint of both space and speed, except that binary search uses slightly less space. In fact, the average time for a successful search, using the program of Table 1 with the frequency data of Fig. 12, is only about $17.8u$, and only 41 table locations are needed to store the 31 keys.

Unfortunately it isn't very easy to discover such functions $f(K)$. There are $41^{31} \approx 10^{50}$ possible functions from a 31-element set into a 41-element set, and only $41 \cdot 40 \cdot \dots \cdot 11 = 41!/10! \approx 10^{43}$ of them will give distinct values for each argument; thus only about one of every 10 million functions will be suitable.

Functions which avoid duplicate values are surprisingly rare, even with a fairly large table. For example, the famous "birthday paradox" asserts that if 23 or more people are present in a room, chances are good that two of them

Table 1

TRANSFORMING A SET OF KEYS INTO UNIQUE ADDRESSES

		A	AND	ARE	AS	AT	BE	BUT	BY	FOR	FROM	HAD	HAVE	HE	HER
Instruction															
LD1N	K(1:1)	-1	-1	-1	-1	-1	-2	-2	-2	-6	-6	-8	-8	-8	-8
LD2	K(2:2)	-1	-1	-1	-1	-1	-2	-2	-2	-6	-6	-8	-8	-8	-8
INCL	-3,2	-9	6	10	13	14	-5	14	18	2	5	-15	-15	-11	-11
J1P	*+2	-9	6	10	13	14	-5	14	18	2	5	-15	-15	-11	-11
INCL	16,2	7	16	2	2	10	10
LD2	K(3:3)	7	6	10	13	14	16	14	18	2	5	2	2	10	10
J2Z	9F	7	6	10	13	14	16	14	18	2	5	2	2	10	10
INCL	-23,2	.	-18	-13	.	.	.	9	.	-7	-7	-22	-1	.	1
J1P	9F	.	-18	-13	.	.	.	9	.	-7	-7	-22	-1	.	1
INCL	11,2	.	-3	3	23	20	-7	35	.	.	.
LDA	K(4:4)	.	-3	3	23	20	-7	35	.	.	.
JAZ	9F	.	-3	3	23	20	-7	35	.	.	.
DECL	-5,2	9	.	15	.	.	.
J1N	9F	9	.	15	.	.	.
INCL	10	19	.	25	.	.	.
9H LDA	K	7	-3	3	13	14	16	9	18	23	19	-7	25	10	1
COMPA	TABLE,1	7	-3	3	13	14	16	9	18	23	19	-7	25	10	1
JNE	FAILURE	7	-3	3	13	14	16	9	18	23	19	-7	25	10	1

will have the same month and day. This is the birthday paradox. The domain function which maps no two keys map into the same value. Skeptics who doubt this result can consult the large parties they attend. This is unpublished work of H. M. Meyer (1939), 45. See also *Mecmuasi* 4 (1939), 145-146. *Theory* (New York: Wiley, 1939).

On the other hand, the birthday paradox is a suitable function can be used to solve a puzzle.

Of course this method must be known in advance. It is not a more versatile method if it requires more keys to yield the same value. The ambiguity after $f(K)$ has been removed.

These considerations known as *hashing* or *scrambling* chop something up or to break up some aspects of the key. We compute the address where the search begins.

The birthday paradox asserts that $K_i \neq K_j$ which hash to

HIS	I	IN	IS	IT	NOT
-8	-9	-9	-9	-9	-15
-8	-9	-9	-9	-9	-15
-7	-17	-2	5	6	-7
-7	-17	-2	5	6	-7
18	-1	29	.	.	25
18	-1	29	5	6	25
18	-1	29	5	6	25
12	20
12	20
.
.
.
.
12	-1	29	5	6	20
12	-1	29	5	6	20
12	-1	29	5	6	20

comparing the given argument over a branching process. Found by doing some arithmetic which is the location of K

English words which we have 2 and 6.3. Table 1 shows keys into a unique number and to the MIX programs for search, optimal tree search, superior from the standpoint of space slightly less space. In ; the program of Table 1 u , and only 41 table loca-

h functions $f(K)$. There set into a 41-element set, r will give distinct values million functions will be

risingly rare, even with a y paradox" asserts that if e good that two of them

QUEUE ADDRESSES

r11	FROM	HAD	HAVE	HE	HER
-6	-6	-8	-8	-8	-8
-6	-6	-8	-8	-8	-8
2	5	-15	-15	-11	-11
2	5	-15	-15	-11	-11
.	.	2	2	10	10
2	5	2	2	10	10
2	5	2	2	10	10
-7	-7	-22	-1	.	1
-7	-7	-22	-1	.	1
23	20	-7	35	.	.
23	20	-7	35	.	.
23	20	-7	35	.	.
.	9	.	15	.	.
.	9	.	15	.	.
.	19	.	25	.	.
23	19	-7	25	10	1
23	19	-7	25	10	1
23	19	-7	25	10	1

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 Mecmuası* 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 r11 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
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

$$0 \leq h(K) < M, \quad (1)$$

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 :

$$h(K) = K \bmod M. \quad (2)$$

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 $r = 64, 256, \text{ or } 100$),

since a remainder modulo position of the key digits a prime number such that has been found to be quit

For example, on the 1 $h(K)$ by the sequence

```
LDX 1
ENTA 1
DIV 1
```

The multiplicative hash is harder to describe because instead of with integers. usually 10^{10} or 2^{30} for M if we imagine the radix p choose some integer const

h

In this case we usually let $h(K)$ consists of the leading

In MIX code, if we let h hash function is

```
LDA K
MUL A
ENTA 0
SLB m
```

Now $h(K)$ appears in register shift instructions, this section on many machines multipl

In a sense this method could for example take $1/2$ the reciprocal of a constant that (5) is almost a "middle" ence: We shall see that it has good properties.

One of the nice features was lost in (5); we could after (5) has finished. The algorithm can be used to that $K = (A'(AK \bmod 2^n))$ tents of register X just b

$K_1 \neq$