Data Types and Structures

C. C. Gotlieb and L. R. Gotlieb

Data Types and Structures

C. C. Gotlieb and L. R. Gotlieb

University of Toronto

Prentice-Hall, Inc., Englewood Cliffs, New Jersey 07632

Library of Congress Cataloging in Publication Data

GOTLIEB, C. C.
Data types and structure.

Includes bibliographies and index.
1. Data structures (Computer science)
2. Programming languages (Electronic computers)
I. Gotlieb, L. R., (date) joint author. II. Title.
QA76.9.D35G67 001.6'42 77-25017
ISBN 0-13-197095-X

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

PRENTICE-HALL INTERNATIONAL, INC., *London*
PRENTICE-HALL OF AUSTRALIA PTY. LIMITED, *Sydney*
PRENTICE-HALL OF CANADA, LTD., *Toronto*
PRENTICE-HALL OF INDIA PRIVATE LIMITED, *New Delhi*
PRENTICE-HALL OF JAPAN, INC., *Tokyo*
PRENTICE-HALL OF SOUTHEAST ASIA PTE. LTD., *Singapore*
WHITEHALL BOOKS LIMITED, *Wellington, New Zealand*

Preface

Data structures was identified as a distinct subject in ACM's Curriculum 68, which is generally recognized as the first comprehensive formulation of topics to be included in a university program in computing and information sciences. But the scope of the subject was not clearly defined there. As evidenced by the flow of papers, survey articles and books since Curriculum 68, all having some variant of the term "data structures" in their title, there has been a steady evolution in the concept. At first data structures was almost synonymous with graph theory—particularly the theory of lists and trees which lends itself naturally to the description of hierarchical data. Then the concept was extended to include networks, the algebraic theory of sets, relations, lattices, and so on, becoming what is now regarded as *discrete structures*. Following the publications of the Codasyl Task Force it was realized that the mathematical concepts had to be further enlarged by a treatment of *storage structures*, for the fact that data must be represented in computer storage introduces major considerations not present at the mathematical level. The ACM Conference on "Data: Abstraction, Definition and Structure" served to further delineate the subject. In that conference there were several contributions on data types, a concept which appeared in the first versions of Algol, but which until then had not received major attention.

To some the subject of data structures is still the theory of discrete algebraic structures—studied by considering sets of data and the operations on them, by setting up classifications of data types based on the operations, and by devising methods of defining types in computer languages and using the types in applications. We have chosen to regard the storage structure as being an integral part of the data structure, and in Chap. 2, which provides a framework for the rest of the book, the storage structure is explicitly included in the formal definition. Also, throughout the later chapters

repeated emphasis is given to methods and problems arising out of the storage representation; for example, the representation of arrays in Chap. 5, and the problems of allocating and deallocating storage in Chap. 8. Because computer storage is so important to data structures, a course on the hardware aspects of computers would be a useful prerequisite to the data structures course for which this book is intended as the text, but such a course is not essential. Because discrete structures *are* essential, Chap. 1 consists of a brief survey of the ideas necessary for an understanding of the book. In many computer science programs a course on discrete structures will be taken as a prerequisite, or simultaneously, with a course on data structures.

There is a natural place for the data structure course in a computing and information sciences program. This place is just *after* the introductory courses in computing, generally devoted to explaining the concept of an algorithm, developing programming style, and introducing the student to one or more programming languages, but just *before* the detailed courses on compiler theory, operating systems, and file systems, which mark a specialist program. Thus a data structures course, and hence this book, is directed at all who propose to specialize in computer science, information science, or system science, and to those in related fields such as applied mathematics or industrial engineering who wish to go at least a little way beyond the elementary courses on computer programming. It is true that only the specialist is likely to have to produce working programs on some of the now highly developed techniques such as sorting and garbage collection discussed in the book. But these processes are so common and so important that an understanding of their principles is part of the basic knowledge which anyone seriously engaged in programming should carry with him. The detail has been suppressed to a basic level.

The concepts of data type and data structures are introduced in Chap. 2, where the framework for later chapters is set up; the common types, strings, lists, arrays, trees, sets and graphs are studied in detail in Chaps. 3 through 7. Beyond these fundamentals the book is intended to bring out certain ideas which we regard as particularly important. These can be summarized as follows:

- There is an intimate association between data type definition and programming language design. Many programming languages have their main justification in their efficient, even brilliant exploitation of some particular data types, and it is pedagogically sound and interesting to use such languages in the presentation of data structures, and to build examples and problems around them. For this reason no single language has been adopted for algorithms. A simple Algol-like language, with sufficient notation to select fields and work with pointer variables, has been adopted. There has been no exposition of programming style, although an effort has been made to express the algorithms in a form acceptable today. It is expected that in many cases the student will rework the algorithms in actual languages such as APL, SNOBOL, PASCAL or PL/I to show how the data structures in the languages lend themselves to implementations of the procedures. To help in this, occasionally an algorithm is expressed in one of these languages.
- Most of the recently developed programming languages allow the user to intro-

duce new data types appropriate to the application being considered. This naturally leads to problems in choosing and designing data structures. Although this process is part of problem solving, and no handbook approach is available, systematic methods of building up and choosing data types and structures are emerging. An illustration of these methods is given in Chap. 7, in order to bring together, in one situation, many of the criteria which have to be applied when comparing and choosing data structures—criteria which appear throughout Chaps. 2 to 7. In addition, a detailed example of data structure design is given in the Appendix.

- The most challenging aspects of describing and representing data are met in attempting to deal with large files and with the file collections of data base systems. File structures and data base management systems are subjects in their own right, and there are whole books on them. But the principles and concepts required to understand them are precisely those required to understand simpler structures. They are more complex because the data volumes are larger, the storage mappings involve multilevel stores, and the basic types are combined into composite ones. But the underlying *structural* aspects are the logical extensions of those encountered in studying the more elementary types, and it is important to bring this out in any serious treatment. Thus Chaps. 9 and 10 are intended to provide the connection between data structures and the more system-oriented subjects of file structures and data base management (generally studied later in the computer science curriculum) and to lay the foundation for the later courses on those subjects.
- Although many aspects of data structures are firmly based in mathematics (especially algebra, graph theory and combinatorics), there are other aspects (centered on programming languages), which are less formal. Thus the subject is still in a state of intensive development. Some of these current developments are introduced in Chap. 2; others are treated in the last chapters. Also, to suggest how future developments might influence data structures, we have included accounts of such topics as associative memories, and data manipulation languages. Although the content of these latter chapters is, inevitably, more subject to change than that of the earlier ones, in our opinion there has been sufficient acceptance of the approaches to justify their inclusion in an undergraduate text.

At the University of Toronto, the data structures course of which this book is the outgrowth, is taken by computer science majors in their third year. At this point they will have had an introductory course on algorithms and structured programming, a course on programming languages (where three or four languages including PL/C, Algol W and SNOBOL are taught), and a course on computer architecture. They will have had experience in working with several types of structures, but no systematic overview of them. The data structures course is necessary prerequisite for most of the advanced courses offered in the senior year. Because the course is taken mostly by students who have not yet had a discrete structure course, and because it is a one-semester subject, it has been customary to teach mainly the material to the middle of

Chap. 7, deferring the later chapters to courses on File Structures and Data Base Systems. The whole book can be covered in a two-semester course. The bibliographic material and reference citations in the footnotes provide the basis to explore any topic for which greater depth is considered desirable.

It is hardly necessary to state that solving problems and doing exercises are essential parts of any course; examples and exercises are given at the end of each chapter. Mathematical formulation and solution of problems are important. But even more important is a thorough understanding of the concepts, so that the student will appreciate for each data type, those features which make it especially suitable for particular applications and problems. It is especially desirable that students should know how to use quantitative methods in system design and evaluation. Many of the end-of-chapter exercises are intended to test understanding, and to encourage familiarity with quantitative comparisons with regard to data structure choices.

Acknowledgments

We are grateful to colleagues, students and others, at Toronto and elsewhere, who have read various sections of the manuscript, making corrections and suggesting changes. Among them we would particularly like to mention M. Bloore, M. Brodie, I. Duff, G. Gonnet, T. E. Hull, J. D. Lipson, F. Lochovsky, E. Schonberg, J. Schwartz, R. Sharma, F. Tompa, and D. Tsichritzis. R. Moenck and R. Perrault (at the Scarborough Campus of the University of Toronto), J. Mylopoulos (at the University of Toronto) and R. Peebles (at the University of Waterloo) have all used preliminary versions of the book in their data structure courses, thereby providing additional experience to our own classroom use of the text.

The careful, detailed comments by the referees on the original manuscript not only helped us avoid mistakes, but have undoubtedly led to a book which is clearer and better organized than it would otherwise have been. We would especially like to acknowledge the contribution of Jeffrey Ullman in this respect.

Finally we would like to express our thanks to Mrs. M. I. Chepely and Mrs. V. Shum, who piloted the manuscript through its many, many versions, doggedly overcoming all the difficulties imposed by text-editors, terminals, computers and authors.

C. C. GOTLIEB
L. R. GOTLIEB

Contents

chapter 1

Mathematical Preliminaries

While a more precise definition of data structures will be given in Chapter 2, it is sufficient for now to regard them as sets of data that can be stored in electronic computers, and on which operations can be carried out. The distinction between data structures and mathematical structures is not sharp, but basically mathematics is concerned with the abstract properties of structures and relations on them rather than how such structures are represented in some device. In this chapter mathematical concepts and notation relevant to the study of data structures are presented.

1.1 Sets and Mathematical Logic

The notion of *set* as a collection of members or elements will be taken as primitive. The following notation is used to denote sets and express simple facts about them. We write

$x \in X$ for x is an element, or member, of the set X

$x \notin X$ for x is *not* an element of the set X.

When the order in which elements are listed is important, this is indicated explicitly. $\{x_1, x_2, \ldots, x_n\}$ denotes the *unordered set* of elements $x_1, \ldots, x_n$ and $\langle x_1, \ldots, x_n \rangle$ denotes the *ordered set*.

The *index* of an element in an ordered set is the integer that specifies the element's position. The first element may be given either the index 0 or 1. Both the notations x_p and x[p] will be used to specify the pth element of an ordered set.

The *cardinality* of a set, $|S|$, is the number of elements in it. Two important sets are the null set containing no elements, denoted by $\varnothing$, and the power set of a given set A,

denoted by $P(A)$. $P(A)$ is the set of all subsets of A. If $|A| = n$, $|P(A)| = 2^n$, since $P(A)$ is constructed by either choosing or deleting each of the elements in A.

Given two sets A and B, we write $A \supseteq B$ or $B \subseteq A$ if every member of B is in A, and say that A includes B or that B is a *subset* of A. B is a *proper subset* of A ($A \supset B$) if there are members in A that are not members of the subset B.

$A = B$ if A and B have the same elements

$A \cup B$ is the set consisting of elements either in A or B

$A \cap B$ is the set whose elements are both in A and B

$A - B$ is the set of elements of A that are not in B

$\bar{A}$ is the set of those elements not in A but which are in some universal set, $X \supset A$, implied to exist.

These definitions may also be considered as definitions of the operators $\cup$ (union), $\cap$ (intersection), $-$ (difference), and $\bar{}$ (complement) for which the operands are sets.

For arbitrary sets A, B, and C, the operators $\cup$ and $\cap$ obey certain laws that can be proved directly from the definitions. They are:

Commutative: $A \cup B = B \cup A$ and $A \cap B = B \cap A$

Associative: $A \cup (B \cup C) = (A \cup B) \cup C$ and $A \cap (B \cap C) = (A \cap B) \cap C$

Distributive:

$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$ and $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$.

Each of the two operators $\cup$ and $\cap$ distributes over the other.

Example 1.1

Operations on sets can be illustrated by plane figures, called *Venn diagrams*, in which points are regarded as set elements. Figure 1.1(a) illustrates the distributive law

$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$$

for the case where $A \cap B \cap C \neq \varnothing$. Similarly, Figure 1.1(b) illustrates the theorem

$$\overline{\bar{A} \cup \bar{B}} = A \cap B,$$

which can be proved from the definitions for the operators. In this book, all the sets will be *countably* (or denumerably) *infinite*; that is, their elements can be placed in correspondence with the integers. Since the number of points in the plane is nondenumerable, the Venn diagrams illustrate that set and set operations are meaningful for sets with an infinite number of elements, even when the number is nondenumerable.

The *Cartesian product* of two sets $A \times B$ is the set of ordered pairs of elements in which the first is a member of A and the second a member of B:

$$\{(a, b) \mid a \in A, b \in B\}.$$

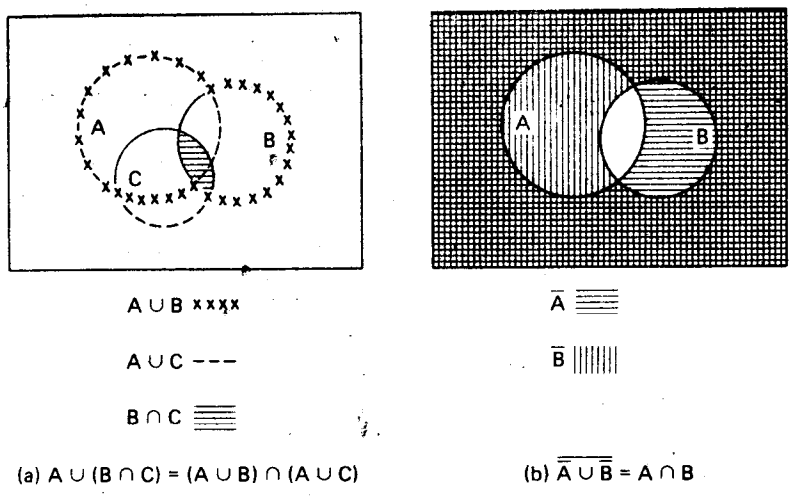

Fig. 1.1 Venn-diagram illustrations of set theorems

In addition to sets, other primitive concepts are taken from mathematical logic. $X = \{x \mid p(x)\}$ is the set of elements x such that predicate (or statement) p is true. *Boolean variables* take on the values *true* or *false* (also denoted by 1 and 0), and Boolean operations over variables, $\wedge$ (and), $\vee$ (or), and $\neg$ (negation) are defined by truth tables as shown in Table 1.1.

TABLE 1.1 Truth tables for Boolean operators

A \ B	*true*	*false*
true	*true*	*false*
false	*false*	*false*

$A \wedge B$

A \ B	*true*	*false*
true	*true*	*true*
false	*true*	*false*

$A \vee B$

A	$\bar{A}$
true	*false*
false	*true*

$\bar{A}$

There is a complete correspondence between sets and Boolean variables, and between set operations and Boolean operations. For example, corresponding to the distributive law for sets, there is the distributive law for Boolean variables:

$$A \wedge (B \vee C) = (A \wedge B) \vee (A \wedge C)$$

and

$$A \vee (B \wedge C) = (A \vee B) \wedge (A \vee C).$$

1.2 Relations

A *binary relation* ρ on the pair of sets X and Y is a subset of the Cartesian product $X \times Y$. If $(x \in X, y \in Y)$ is a member of the subset, we write $x \rho y$. In particular, a binary relation on the set X is a subset of $X \times X$ (also written as X^2), and is designated (X, ρ).

If M is the set of months, {January, February, . . . , December}, and S the set of seasons, {Spring, Summer, Fall, Winter}, Table 1.2 illustrates the relation "M is a month in S." Members of $M \times S$ belonging to the relations are shown as *true*; others are *false*.

TABLE 1.2 Relation of months and seasons

M	Spring	Summer	Fall	Winter
January	*false*	*false*	*false*	*true*
February	*false*	*false*	*false*	*true*
March	*true*	*false*	*false*	*true*
April	*true*	*false*	*false*	*false*
May	*true*	*false*	*false*	*false*
June	*true*	*true*	*false*	*false*
July	*false*	*true*	*false*	*false*
August	*false*	*true*	*false*	*false*
September	*false*	*true*	*true*	*false*
October	*false*	*false*	*true*	*false*
November	*false*	*false*	*true*	*false*
December	*false*	*false*	*true*	*true*

For finite sets X and Y with n and m elements, respectively, the relation (X, Y, ρ) can be represented by an $n \times m$ matrix of rows and columns corresponding to the elements of X and Y, and in which an entry is 1 iff two elements are related and 0 otherwise. This is the *matrix associated with the relation*. It is a *Boolean matrix* whose elements correspond to the values *true* or *false*. For (X, ρ), the matrix is square. Table 1.3 illustrates the matrix associated with the relation $(\{1, 2, 3, 4,\} <)$.

Note that a binary relation (X, ρ) which determines a subset $R \subseteq X^2$ also defines a complement $\bar{R} = X^2 - R$ and a corresponding complementary relation $(X, \bar{\rho})$. The complementary relation for $(\{1, 2, 3, 4, 5\}, <)$ is $(\{1, 2, 3, 4\} \not<)$, and its members are given by the zero entries in Table 1.3.

TABLE 1.3 Matrix associated with a relation

	1	2	3	4
1	0	1	1	1
2	0	0	1	1
3	0	0	0	1
4	0	0	0	0

The binary relation is not the most general type of relation that might be considered between set elements. We might, for example, consider a ternary relation on three sets as a subset of the Cartesian product $X \times Y \times Z$. However, binary relations have enough complexity to be applicable to a wide variety of problems, and further, it is often useful to regard a ternary relation as a set of binary relations. We can look on the three-dimensional matrix associated with the relation on $X \times Y \times Z$ as a set of X, Y

planes for each $z \in Z$; in effect, there is a binary relation on $X \times Y$ for every z. (Similarly, of course, we can view the ternary relation as a set of binary relations on $Y \times Z$ for each $x \in X$, or on $X \times Z$ for $y \in Y$.)

The relation (X, ρ) is said to be:

Reflexive:	if $x \rho x$ for all $x \in X$
Irreflexive:	if $x \rho x$ for no $x \in X$
Symmetric:	if $x \rho y \Rightarrow y \rho x$ for all $x, y \in X$
Antisymmetric:	if $x \rho y \wedge y \rho x \Rightarrow x = y$ for all $x, y \in X$
Transitive:	if $x \rho y \wedge y \rho z \Rightarrow x \rho z$ for all $x, y, z \in X$.

Relations with different combinations of reflexive/irreflexive, symmetric/antisymmetric, and transitive/nontransitive are interesting both in life and mathemtics. For example, the sibling relationship is irreflexive and symmetric; the relation "x is a substructure of y" is irreflexive, antisymmetric, and transitive. The equivalence relation, and the partial order, considered next, are especially important because they are often encountered.

A relation that is reflexive, symmetric, and transitive is said to be an *equivalence relation*, denoted by E. A *partition*, Π, of a set X, is a family of subsets $X_1, \ldots, X_k \subseteq X$ such that

1. $X_i \neq \varnothing$, all i (the subsets are not empty).
2. $i \neq j \Rightarrow X_i \cap X_j = \varnothing$ (the subsets are disjoint).
3. $\bigcup_i X_i = X$ (the subsets, collectively, make up X).

Given a partition, Π, of X, a binary relation, $E(\Pi)$ is defined on X by making $x_i E(\Pi) x_j$ mean that x_i and x_j are related iff they are in the same subset. It is evident that $E(\Pi)$ is symmetric, transitive, and reflextive, and hence an equivalence relation. It is not difficult to show the converse also: that is, that the existence of an equivalence relation on a set implies a partition. A subset, $C(a)$, of the partition consists of all members such that $xE(\Pi)a$:

$$C(a) = \{x \in X \mid xE(\Pi)a\} \subset X.$$

A commonly cited example of an equivalence relation defined on the integers N is the set of residue classes for a given (integer) modulus k. This is the set of integers remaining after dividing N by k, and it is written N mod k. Each integer n can be written in the form $i \times k + m$, where i is an integer and $0 \leq m < k$. The effect is a partition of the integers into the residue classes $\{C(0), C(1), \ldots, C(k-1)\}$. For example, if 7 is the modulus, the possible remainders after dividing any integer by 7 are 0, 1, 2, 3, 4, 5, 6, and these represent the residue classes $C(0), \ldots, C(6)$. Integers belonging to $C(1)$ are of the form $7 \times i + 1$.

A *partial order relation* $(X, \leq)$ has the following properties:

1. $x \leq x$ (reflexive).
2. $x \leq y \wedge y \leq x \Rightarrow x = y$ (antisymmetric).
3. $x \leq y \wedge y \leq z \Rightarrow x \leq z$ (transitive).

Sets for which there is a partial order relation between members are called *posets*. From the relation $\leq$ we can define other relations,[1] which may be represented by $\geq$, $<$, and $>$. For example,

$$x \leq y \Rightarrow y \geq x$$

$$x < y \Rightarrow x \leq y \wedge x \neq y.$$

The relation $\geq$ is also a partial order, but $>$ (and also $<$) is irreflexive and hence not a partial order.

Note that $x \not\geq y$ does *not* imply that $x < y$ since the $<$ relation is not defined as the complement of $\geq$. Suppose, for example, that $\geq$ is the inclusion relation on a set; this is a partial ordering, and $X > Y$ is interpreted to mean that Y is a proper subset of X. Given two arbitrary sets, we cannot say that one is necessarily a subset of the other.

If for any two elements $x \geq y \vee x < y$, we have a *linear* or *simple order*. In such a case the members of the poset can be displayed as points on a line, and the set is called a *chain*.

Any finite poset can be represented by a *Hasse diagram* in which:

1. Each distinct element is represented by a point.
2. The relation $x < y$ is represented by a line that rises steadily from x to y.

If x is below y in a Hasse diagram, it does not follow that $x < y$ unless there is a line between them.

Example 1.2

Figure 1.2 shows the Hasse diagram for the poset whose elements are the areas representing subsets in the accompanying Venn diagram. The relation is set inclusion.

Let $(X, \leq)$ be a relation and $X \supseteq Y \supset \bigcirc$. Then an element $x \in X$ is an upper bound of Y iff for all $y \in Y$, $y \leq x$. The least element of all the upper bounds of Y

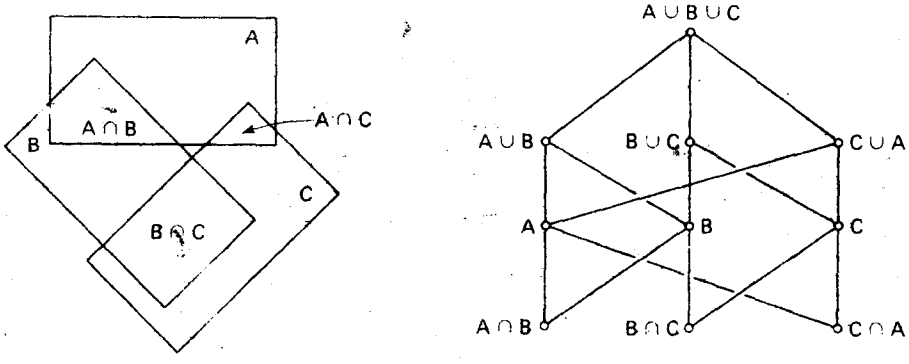

Fig. 1.2 Hasse diagram

[1]There are several common ways of expressing these relations verbally: for example, x has rank less than or equal to y for $x \leq y$, x precedes y for $x \prec y$, and x is a successor of y for $x \succ y$.