Algorithms & Data Structures

Niklaus Wirth

Algorithms and Data Structures

Translator's note. *This book was translated into Russian in 2009 for specific teaching purposes. Along the way, Pascal-to-Modula-2-to-Oberon conversion typos were corrected and some changes to programs were made. The changes (localized in sects. 1 and 3) were agreed upon with the author in April, 2009. Their purpose was to facilitate verification of the program examples that are now in perfect running order. Most notably, section 1.9 now uses the Dijkstra loop introduced in Oberon-07 (see Appendix C).*

This book can be downloaded from the author's site:

http://www.inf.ethz.ch/personal/wirth/books/AlgorithmE1/AD2012.pdf

The program examples can be downloaded from the site that promulgates Oberons as a unique foundation to teach kids from age 11 all the way up through the university-level compiler construction and software architecture courses:

http://www.inr.ac.ru/~info21/ADen/

where the most recently corrected version may be available.

Please send typos and bugs to: info21@inr.ac.ru

Thanks are due to Wojtek Skulski, Nicholas J. Schwartz and — the longest list of typos by far — Helmut Zinn.

—Fyodor Tkachov, Moscow, 2012-02-18
Last update 2012-03-06

Table of Contents

Preface
Preface To The 1985 Edition
Notation
1 Fundamental Data Structures 9
1.1 Introduction
1.2 The Concept of Data Type
1.3 Standard Primitive Types
1.3.1 The type INTEGER
1.3.2 The type REAL
1.3.3 The type BOOLEAN
1.3.4 The type CHAR
1.3.5 The type SET
1.4 The Array Structure
1.5 The Record Structure
1.6 Representation of Arrays, Records, and Sets
1.6.1 Representation of Arrays
1.6.2 Representation of Recors
1.6.3 Representation of Sets
1.7 The File (Sequence)
1.7.1 Elementary File Operators
1.7.2 Buffering Sequences
1.7.3 Buffering between Concurrent Processes
1.7.4 Textual Input and Output
1.8 Searching
1.8.1 Linear Search
1.8.2 Binary Search
1.8.3 Table Search
1.9 String Search
1.9.1 Straight String Search
1.9.2 The Knuth-Morris-Pratt String Search
1.9.3 The Boyer-Moore String Search
Exercises
References
2 Sorting 49
2.1 Introduction
2.2 Sorting Arrays
2.2.1 Sorting by Straight Insertion
2.2.2 Sorting by Straight Selection
2.2.3 Sorting by Straight Exchange
2.3 Advanced Sorting Methods
2.3.1 Insertion Sort by Diminishing Increment
2.3.2 Tree Sort
2.3.3 Partition Sort
2.3.4 Finding the Median
2.3.5 A Comparison of Array Sorting Methods
2.4 Sorting Sequences
2.4.1 Straight Merging

2.4.2 Natural Merging
2.4.3 Balanced Multiway Merging
2.4.4 Polyphase Sort
2.4.5 Distribution of Initial Runs
Exercises
References
3 Recursive Algorithms 99
3.1 Introduction
3.2 When Not to Use Recursion
3.3 Two Examples of Recursive Programs
3.4 Backtracking Algorithms
3.5 The Eight Queens Problem
3.6 The Stable Marriage Problem
3.7 The Optimal Selection Problem
Exercises
References
4 Dynamic Information Structures 129
4.1 Recursive Data Types
4.2 Pointers
4.3 Linear Lists
4.3.1 Basic Operations
4.3.2 Ordered Lists and Reorganizing Lists
4.3.3 An Application: Topological Sorting
4.4 Tree Structures
4.4.1 Basic Concepts and Definitions
4.4.2 Basic Operations on Binary Trees
4.4.3 Tree Search and Insertion
4.4.4 Tree Deletion
4.4.5 Tree Deletion
4.5 Balanced Trees
4.5.1 Balanced Tree Insertion
4.5.2 Balanced Tree Deletion
4.6 Optimal Search Trees
4.7 B-Trees
4.7.1 Multiway B-Trees
4.7.2 Binary B-Trees
4.8 Priority Search Trees
Exercises
References
5 Key Transformations (Hashing) 200
5.1 Introduction
5.2 Choice of a Hash Function
5.3 Collision handling
5.4 Analysis of Key Transformation
Exercises
References
Appendices 207
A. The ASCII Character Set

B. The Syntax of Oberon
C. The Dijkstra loop
Index

Preface

In recent years the subject of computer programming has been recognized as a discipline whose mastery is fundamental and crucial to the success of many engineering projects and which is amenable to scientific treatement and presentation. It has advanced from a craft to an academic discipline. The initial outstanding contributions toward this development were made by E.W. Dijkstra and C.A.R. Hoare. Dijkstra's *Notes on Structured Programming* [1] opened a new view of programming as a scientific subject and intellectual challenge, and it coined the title for a "revolution" in programming. Hoare's *Axiomatic Basis of Computer Programming* [2] showed in a lucid manner that programs are amenable to an exacting analysis based on mathematical reasoning. Both these papers argue convincingly that many programming errors can be prevented by making programmers aware of the methods and techniques which they hitherto applied intuitively and often unconsciously. These papers focused their attention on the aspects of composition and analysis of programs, or more explicitly, on the structure of algorithms represented by program texts. Yet, it is abundantly clear that a systematic and scientific approach to program construction primarily has a bearing in the case of large, complex programs which involve complicated sets of data. Hence, a methodology of programming is also bound to include all aspects of data structuring. Programs, after all, are concrete formulations of abstract algorithms based on particular representations and structures of data. An outstanding contribution to bring order into the bewildering variety of terminology and concepts on data structures was made by Hoare through his *Notes on Data Structuring* [3]. It made clear that decisions about structuring data cannot be made without knowledge of the algorithms applied to the data and that, vice versa, the structure and choice of algorithms often depend strongly on the structure of the underlying data. In short, the subjects of program composition and data structures are inseparably interwined.

Yet, this book starts with a chapter on data structure for two reasons. First, one has an intuitive feeling that data precede algorithms: you must have some objects before you can perform operations on them. Second, and this is the more immediate reason, this book assumes that the reader is familiar with the basic notions of computer programming. Traditionally and sensibly, however, introductory programming courses concentrate on algorithms operating on relatively simple structures of data. Hence, an introductory chapter on data structures seems appropriate.

Throughout the book, and particularly in Chap. 1, we follow the theory and terminology expounded by Hoare and realized in the programming language *Pascal* [4]. The essence of this theory is that data in the first instance represent abstractions of real phenomena and are preferably formulated as abstract structures not necessarily realized in common programming languages. In the process of program construction the data representation is gradually refined in step with the refinement of the algorithm to comply more and more with the constraints imposed by an available programming system [5]. We therefore postulate a number of basic building principles of data structures, called the fundamental structures. It is most important that they are constructs that are known to be quite easily implementable on actual computers, for only in this case can they be considered the true elements of an actual data representation, as the molecules emerging from the final step of refinements of the data description. They are the record, the array (with fixed size), and the set. Not surprisingly, these basic building principles correspond to mathematical notions that are fundamental as well.

A cornerstone of this theory of data structures is the distinction between fundamental and "advanced" structures. The former are the molecules themselves built out of atoms that are the components of the latter. Variables of a fundamental structure change only their value, but never their structure and never the set of values they can assume. As a consequence, the size of the store they occupy remains constant.

"Advanced" structures, however, are characterized by their change of value and structure during the execution of a program. More sophisticated techniques are therefore needed for their implementation. The sequence appears as a hybrid in this classification. It certainly varies its length; but that change in structure is of a trivial nature. Since the sequence plays a truly fundamental role in practically all computer systems, its treatment is included in Chap. 1.

The second chapter treats sorting algorithms. It displays a variety of different methods, all serving the same purpose. Mathematical analysis of some of these algorithms shows the advantages and disadvantages of the methods, and it makes the programmer aware of the importance of analysis in the choice of good solutions for a given problem. The partitioning into methods for sorting arrays and methods for sorting files (often called internal and external sorting) exhibits the crucial influence of data representation on the choice of applicable algorithms and on their complexity. The space allocated to sorting would not be so large were it not for the fact that sorting constitutes an ideal vehicle for illustrating so many principles of programming and situations occurring in most other applications. It often seems that one could compose an entire programming course by choosing examples from sorting only.

Another topic that is usually omitted in introductory programming courses but one that plays an important role in the conception of many algorithmic solutions is recursion. Therefore, the third chapter is devoted to recursive algorithms. Recursion is shown to be a generalization of repetition (iteration), and as such it is an important and powerful concept in programming. In many programming tutorials, it is unfortunately exemplified by cases in which simple iteration would suffice. Instead, Chap. 3 concentrates on several examples of problems in which recursion allows for a most natural formulation of a solution, whereas use of iteration would lead to obscure and cumbersome programs. The class of backtracking algorithms emerges as an ideal application of recursion, but the most obvious candidates for the use of recursion are algorithms operating on data whose structure is defined recursively. These cases are treated in the last two chapters, for which the third chapter provides a welcome background.

Chapter 4 deals with dynamic data structures, i.e., with data that change their structure during the execution of the program. It is shown that the recursive data structures are an important subclass of the dynamic structures commonly used. Although a recursive definition is both natural and possible in these cases, it is usually not used in practice. Instead, the mechanism used in its implementation is made evident to the programmer by forcing him to use explicit reference or pointer variables. This book follows this technique and reflects the present state of the art: Chapter 4 is devoted to programming with pointers, to lists, trees and to examples involving even more complicated meshes of data. It presents what is often (and somewhat inappropriately) called list processing. A fair amount of space is devoted to tree organizations, and in particular to search trees. The chapter ends with a presentation of scatter tables, also called "hash" codes, which are often preferred to search trees. This provides the possibility of comparing two fundamentally different techniques for a frequently encountered application.

Programming is a constructive activity. How can a constructive, inventive activity be taught? One method is to crystallize elementary composition priciples out many cases and exhibit them in a systematic manner. But programming is a field of vast variety often involving complex intellectual activities. The belief that it could ever be condensed into a sort of pure recipe teaching is mistaken. What remains in our arsenal of teaching methods is the careful selection and presentation of master examples. Naturally, we should not believe that every person is capable of gaining equally much from the study of examples. It is the characteristic of this approach that much is left to the student, to his diligence and intuition. This is particularly true of the relatively involved and long example programs. Their inclusion in this book is not accidental. Longer programs are the prevalent case in practice, and they are much more suitable for exhibiting that elusive but essential ingredient called style and orderly structure. They are also meant to serve as exercises in the art of program reading, which too often is neglected in favor of program writing. This is a primary motivation behind the inclusion of larger programs as examples in their entirety. The

reader is led through a gradual development of the program; he is given various snapshots in the evolution of a program, whereby this development becomes manifest as a stepwise refinement of the details. I consider it essential that programs are shown in final form with sufficient attention to details, for in programming, the devil hides in the details. Although the mere presentation of an algorithm's principle and its mathematical analysis may be stimulating and challenging to the academic mind, it seems dishonest to the engineering practitioner. I have therefore strictly adhered to the rule of presenting the final programs in a language in which they can actually be run on a computer.

Of course, this raises the problem of finding a form which at the same time is both machine executable and sufficiently machine independent to be included in such a text. In this respect, neither widely used languages nor abstract notations proved to be adequate. The language Pascal provides an appropriate compromise; it had been developed with exactly this aim in mind, and it is therefore used throughout this book. The programs can easily be understood by programmers who are familiar with some other high-level language, such as ALGOL 60 or PL/1, because it is easy to understand the Pascal notation while proceeding through the text. However, this not to say that some proparation would not be beneficial. The book *Systematic Programming* [6] provides an ideal background because it is also based on the Pascal notation. The present book was, however, not intended as a manual on the language Pascal; there exist more appropriate texts for this purpose [7].

This book is a condensation and at the same time an elaboration of several courses on programming taught at the Federal Institute of Technology (ETH) at Zürich. I owe many ideas and views expressed in this book to discussions with my collaborators at ETH. In particular, I wish to thank Mr. H. Sandmayr for his careful reading of the manuscript, and Miss Heidi Theiler and my wife for their care and patience in typing the text. I should also like to mention the stimulating influence provided by meetings of the Working Groups 2.1 and 2.3 of IFIP, and particularly the many memorable arguments I had on these occasions with E. W. Dijkstra and C.A.R. Hoare. Last but not least, ETH generously provided the environment and the computing facilities without which the preparation of this text would have been impossible.

Zürich, Aug. 1975 N. Wirth

[1] E.W. Dijkstra, in: O.-J. Dahl, E.W. Dijkstra, C.A.R. Hoare. Structured Programming. F. Genuys, Ed., New York, Academic Press, 1972, pp. 1-82.

[2] C.A.R. Hoare. *Comm. ACM*, 12, No. 10 (1969), 576-83.

[3] C.A.R. Hoare, in Structured Programming [1], cc. 83-174.

[4] N. Wirth. The Programming Language Pascal. *Acta Informatica*, 1, No. 1 (1971), 35-63.

[5] N. Wirth. Program Development by Stepwise Refinement. *Comm. ACM*, 14, No. 4 (1971), 221-27.

[6] N. Wirth. Systematic Programming. Englewood Cliffs, N.J. Prentice-Hall, Inc., 1973.

[7] K. Jensen and N. Wirth. PASCAL-User Manual and Report. Berlin, Heidelberg, New York; Springer-Verlag, 1974.

Preface To The 1985 Edition

This new Edition incorporates many revisions of details and several changes of more significant nature. They were all motivated by experiences made in the ten years since the first Edition appeared. Most of the contents and the style of the text, however, have been retained. We briefly summarize the major alterations. The major change which pervades the entire text concerns the programming language used to express the algorithms. Pascal has been replaced by Modula-2. Although this change is of no fundamental influence to the presentation of the algorithms, the choice is justified by the simpler and more elegant syntactic structures of Modula-2, which often lead to a more lucid representation of an algorithm's structure. Apart from this, it appeared advisable to use a notation that is rapidly gaining acceptance by a wide community, because it is well-suited for the development of large programming systems. Nevertheless, the fact that Pascal is Modula's ancestor is very evident and eases the task of a transition. The syntax of Modula is summarized in the Appendix for easy reference.

As a direct consequence of this change of programming language, Sect. 1.11 on the sequential file structure has been rewritten. Modula-2 does not offer a built-in file type. The revised Sect. 1.11 presents the concept of a sequence as a data structure in a more general manner, and it introduces a set of program modules that incorporate the sequence concept in Modula-2 specifically.

The last part of Chapter 1 is new. It is dedicated to the subject of searching and, starting out with linear and binary search, leads to some recently invented fast string searching algorithms. In this section in particular we use assertions and loop invariants to demonstrate the correctness of the presented algorithms.

A new section on priority search trees rounds off the chapter on dynamic data structures. Also this species of trees was unknown when the first Edition appeared. They allow an economical representation and a fast search of point sets in a plane.

The entire fifth chapter of the first Edition has been omitted. It was felt that the subject of compiler construction was somewhat isolated from the preceding chapters and would rather merit a more extensive treatment in its own volume.

Finally, the appearance of the new Edition reflects a development that has profoundly influenced publications in the last ten years: the use of computers and sophisticated algorithms to prepare and automatically typeset documents. This book was edited and laid out by the author with the aid of a Lilith computer and its document editor Lara. Without these tools, not only would the book become more costly, but it would certainly not be finished yet.

Palo Alto, March 1985 N. Wirth

Notation

The following notations, adopted from publications of E.W. Dijkstra, are used in this book.

In logical expressions, the character & denotes conjunction and is pronounced as and. The character ~ denotes negation and is pronounced as not. Boldface **A** and **E** are used to denote the universal and existential quantifiers. In the following formulas, the left part is the notation used and defined here in terms of the right part. Note that the left parts avoid the use of the symbol "...", which appeals to the readers intuition.

Ai: $m \le i < n : P_i$ P_m & P_{m+1} & ... & P_{n-1}

The P_i are predicates, and the formula asserts that for all indices i ranging from a given value m to, but excluding a value n P_i holds.

Ei: $m \le i < n : P_i$ P_m or P_{m+1} or ... or P_{n-1}

The P_i are predicates, and the formula asserts that for some indices i ranging from a given value m to, but excluding a value n P_i holds.

Si: $m \le i < n : x_i = x_m + x_{m+1} + ... + x_{n-1}$

MIN i: $m \le i < n : x_i =$ minimum(x_m , ... , x_{n-1})

MAX i: $m \le i < n : x_i =$ maximum(x_m , ... , x_{n-1})