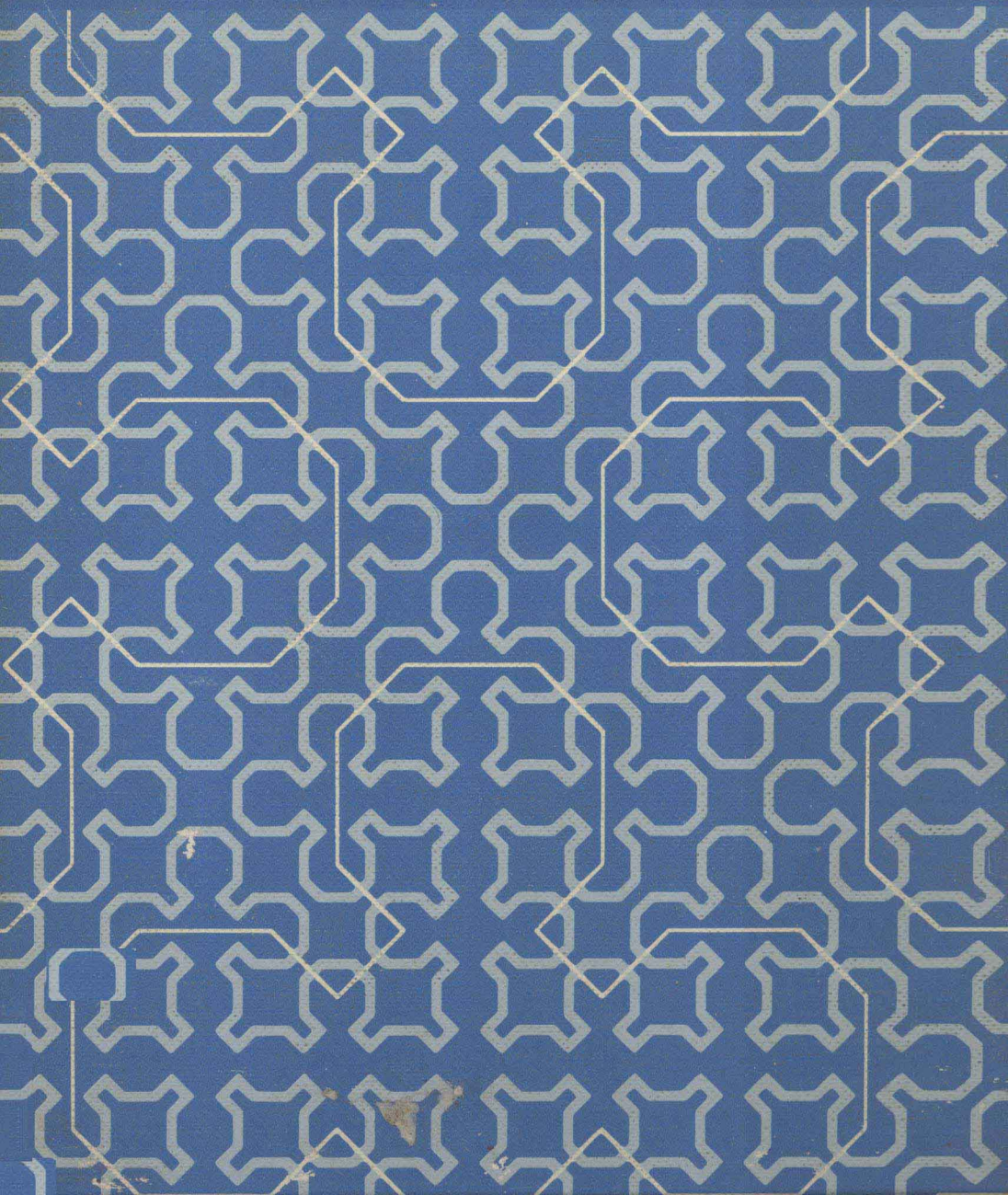
Cambridge Computer Science Texts · 9

An introduction to Computational Combinatorics

E. S. PAGE and L. B. WILSON



Cambridge Computer Science Texts ·

An Introduction to Computational Combinatorics

E. S. PAGE
Vice-Chancellor
University of Reading

L. B. WILSON
Computing Laboratory
University of Newcastle upon Tyne

Cambridge University Press
Cambridge
London · New York · Melbourne

Published by the Syndics of the Cambridge University Press
The Pitt Building, Trumpington Street, Cambridge CB2 1RP
Bentley House, 200 Euston Road, London NW1 2DB
32 East 57th Street, New York, NY 10022, USA
296 Beaconsfield Parade, Middle Park, Melbourne 3206, Australia

First published 1979

Reproduced, printed and bound in Great Britain by
Cox & Wyman Ltd, London, Fakenham and Reading

ISBN 0 521 22427 6 hard covers
ISBN 0 521 29492 4 paperback

Preface

By the time students have done some programming in one or two languages and have learnt the common ways of representing information in a computer they may wish to embark upon further study of theoretical or applied topics in computing science. Most of them will encounter problems which need one or more of the techniques described in this book; for example, the analyses of certain algorithms and of some models of scheduling strategies in an operating system depend upon the formation and solution of difference equations; the tasks of making lists of possible alternatives and of answering questions about them crop up in as diverse areas as stock records and the theory of grammars; searches for discrete optima - or for the best that can be found with just so much computing - occur in manufacturing, design and many operational research investigations. These examples would be justification enough for the teaching of this material but we believe that the field of computational combinatorics itself contains fascinating problems and we hope that this introduction gives a glimpse of some of them.

It will be obvious from a count of the numbered equations that some chapters are more mathematical than others, and that chapters 2 to 4 have a higher 'equation count' than the rest. We could almost as easily have changed the order of presentation and put this material near the end, and some readers and lecturers will prefer to do so. However, mathematical techniques have to be acquired if one is to calculate how large or lengthy a combinatorial calculation will be. Problems can grow so big so quickly that much human and computer time, and money, can be wasted if an infeasible computation is started. We therefore put this material first in our courses but it could be postponed and the more practical programming topics treated first instead, referring to chapters 2 to 4 for definitions where necessary.

It is convenient for us to teach most of the material in this book during the second year of certain groups of students at Newcastle; but the mathematical and computing science preparation that students have received will vary from place to place and elsewhere other times might be more appropriate. The mathematics needed is elementary algebra and calculus - for example, an isolated use of Taylor's theorem occurs in chapter 2 and simple first order differential equations in chapter 3 - and the students should have sufficient computing experience to be able to understand the algorithms which are described here in an Algol-like language, and to program them for running on the machine available to them. The students can check their grasp of what they have read by tackling the exercises given at the end of the chapters; some of them are straightforward drill but others are longer, taken from university examination papers, which will give an idea of the standard required in a written paper at this level. There are also suggestions for some programming tasks which can be expanded if the time and course structure so demand. Some hints on the solutions of the exercises are given at the end of the book; we hope that they are sufficiently detailed to indicate the way to proceed to someone who is stuck but not so full as to be specimen answers which a student might find attractive to copy.

We are pleased to acknowledge our gratitude to the Universities of Newcastle upon Tyne, Warwick and Linkoping for their permission to include questions from their examination papers.

However hard authors try to eliminate all errors and misprints they will be fortunate if they succeed in doing so; if any remain in spite of our own efforts and the helpful comments of some of our colleagues we apologise.

E. S. Page
L. B. Wilson

Contents

1 · The Problems of Computational Combinatorics

1.1 INTRODUCTION

Digital computers represent the items which they store and manipulate in a discrete form. The operations that are performed on the items are exact and they are necessarily finite in number even though each operation is completed very quickly. Each item may need only a few bits or bytes for its storage but there may be many such items. When one plans a computer application one normally needs to know, at least approximately, how much storage will be required and about how big a computation it is. A major component of the storage needed may be the number of items of a particular type that have to be stored. Similarly, a knowledge of how many operations of the various types the computation involves will help in assessing how much computing will be needed. These quantities, both dependent upon the answer to a question 'How many?', are important if one attempts to compare different methods of achieving the desired end or even to decide whether the computation is feasible. In computing some use of one resource of the computing system can nearly always be reduced at the expense of making use of another of the resources; for example, there can nearly always be a trade of storage for processing time, and vice versa. For a proper comparison of the alternative methods the amounts need to be quantified. One needs to be able to answer the questions which start 'How many?' - one of the interests included in the subject of computational combinatorics.

1.2 WHERE?

For some applications in computing, the answers to questions of the 'How many?' type may be sufficient; for example, if two algorithms which each perform a certain computation are being compared it may be enough to know how many operations of the various types each algorithm requires

and how many storage locations each needs. In some other applications an answer to the question 'How many?', whether approximate or exact, may only be enough to determine whether or not the problem can be tackled at all. For example, the task of sorting some items into order within the main storage of the machine will only be possible by the given algorithm if the number of items is small enough so that the storage needed for the final list and for intermediate working storage is within the bounds of what can be spared for this program. However, once an ordered list has been obtained, other questions may need answering. Does a certain item appear in that ordered list, and, if so, where is it? Such a question might call for a suitable searching algorithm but it might be possible to answer it by calculation without searching the list itself. The converse question 'Which item is at a given position in that order?' could present a task of retrieval but it might be possible to identify and construct the item without referring to the order by performing an appropriate computation. The latter approach would be the only feasible one if the number of items were so large that not all could be conveniently stored and also if the items had not been explicitly generated and it was wasteful to do so. For example, the number of permutations of n items increases rapidly with n and soon becomes too large to generate and store (there are over 400 million even for $n = 12$) but there are several ways of defining an order so that the questions can be answered directly. In other problems answers to 'Where?' and 'Which?' questions may be found most suitably by applying one of the number of techniques available for searching and retrieval, but there can be cases where such techniques are not applicable or can be replaced by more efficient ones special to the application. In a sense, all these problems are included in computational combinatorics but as algorithms for searching, sorting and retrieval are normally introduced at an early stage of study in computing science the latter are only mentioned incidentally in this book.

The problem of sorting - given a set of items, put them in order - has a counterpart when the items are not given explicitly, but instead are defined in some implicit manner. For example, the set of items may be all possible arrangements of the four letters A, B, C, D. In such a case, the problem is to generate the set as well as to place the items in order, but it is often possible to choose the method of generation which

will produce the desired order directly, and also enable other questions about the order to be answered easily.

1.3 WHICH IS BEST?

Problems of optimisation have claimed the attention of mathematicians from earliest times; some appear elementary - like finding the shortest distance from a point to a given circle - others, while calling for more advanced methods, are routine - like the 'soup tin' problem of obtaining the cylinder of greatest volume for a given surface area, or discovering the closed curve of given length which encloses the greatest area. Yet others are comparatively simple in specific cases but difficult in general - the task of showing that at most four paints are needed to colour a given planar map is of a different type and order of difficulty from proving that all such maps can be coloured by at most four paints. The problems of optimisation encountered in computational combinatorics are, not surprisingly, concerned with discrete rather than continuous variables, and are predominantly of the apparently trivial type 'Which is the best item out of this finite set of items?'. Of course, the method by which one item is compared with another may be complicated but the principal difficulty with such problems is that the set concerned, though finite, is large. Indeed, it may be so large that an exhaustive search through it will be beyond the resources of the computing equipment available. Thus we look for algorithms better than mere exhaustive searching. For example, we know that there are only (!) $n!$ different ways of visiting n towns once and once only starting from another given town; one of these routes must have the shortest length and so it must be easy to discover it and so solve this 'travelling salesman' problem: so it must, were it not for the vastness of $n!$ for n more than a few handfuls. A naive approach soon founders. We need algorithms which use the properties of the set and of the criteria determining the optimum to reduce the size of the computation. The construction of such algorithms and the assessment of the work they entail is another of the topics studied in computational combinatorics.

A more theoretical aspect, but one often with great practical significance, is that which describes attempts to quantify how 'difficult' classes of computations are. This field of study, called computational complexity, assesses how the numbers of operations of various types in the best possible

algorithms increase with the size of the problem. Sometimes the study yields a constructive proof leading to a practical algorithm; in other cases, the most that can be done is to show the equivalence of different classes of (rather difficult) problems and to conjecture how fast the number of operations required increases with the size of the problem. We restrict ourselves to the mention of one or two problems of this kind.

1.4 METHODS

Readers will recall from their studies in elementary algebra the method of proof by mathematical induction. In a simple form this requires one to demonstrate that if a proposition is true for a general value of some integer parameter n, it is also true for the next larger integer, $n + 1$, and then to exhibit a case for a particular value of n, usually a small one ($n = 1$ or 2 perhaps), in which the proposition is true. It then follows that the proposition is true for any greater integer value of n quite generally. The idea of relating the situation for one value of n with other, usually adjacent, values (say $n - 1$ and $n - 2$) is one which is used widely in combinatorial problems. These recurrence relations when they can be found are most useful tools in many problems in computational combinatorics. Sometimes they can be solved as difference equations to answer questions of the 'How many?' kind; in other examples they can be used as the basis of algorithms to compute 'How many?' for specific cases or to generate lists and solutions for bigger problems from smaller ones that have already been solved, and to exhibit the structure of problems and to reveal relations between different problems. The methods which are introduced in this book first show how to solve some of the important types of difference equations and how to apply them to problems in computational combinatorics. Recurrence relations are used to develop various algorithms, and appear frequently in different guises - sometimes a generating function is used to solve the corresponding difference equation, while at other times a generating function gives a means of deriving a recurrence relation. The recurrence relation itself provides some unity over a field which includes a great variety of different kinds of problems.

2 · Constant Coefficient Difference Equations

2.1 INTRODUCTION

Many problems which seek to determine how many objects of a particular type there are, can be made to depend upon a single variable which takes integer values; these values are often the natural numbers $0, 1, 2, 3, 4, \ldots$ but sometimes they may be just a subset like the even positive integers, $2, 4, 6, \ldots$ For example, the elementary problem of determining how many different permutations of n different objects there are, can be reduced to discovering the appropriate function of n, which we could write as $p(n)$ or p_n for $n = 0, 1, 2, \ldots$ Other problems of the same general type involve two or more independent integer variables; to find the number of combinations of n things taken m at a time we seek a function which we can write as

$$c(m, n) \quad \text{or} \quad c_{m,n}, \quad {}^{n}C_{m} \quad \text{or} \quad \binom{n}{m}$$

depending on the notation chosen.

One way of approaching the desired solutions is by looking for a reasoned argument which will relate the unknown function for a general value n (in the one variable case) with values of the unknown function at one or more smaller values of n. If such a recurrence relation can be produced, it can usually be made the basis of an algorithm for computing values of the desired function, while in certain very useful classes of cases the relation can be solved to give an explicit expression for the unknown function. This chapter and the next are concerned with some of the techniques that are available for finding the explicit solutions of certain recurrence relations and with illustrating the sorts of combinatorial problems which are amenable to this approach.

2.1.1 **Differences and Definitions**

If u is a function of one variable n, which takes integer values $0, 1, 2, \ldots$, the difference of the function values at two consecutive arguments

$$u_{n+1} - u_n$$

is called the first (forward) difference of u_n; it is often written

$$\Delta u_n = u_{n+1} - u_n$$

where Δ is the forward difference operator. An alternative way of writing the difference is in terms of the shift operator, $\mathbf{E}$, where

$$\mathbf{E}u_n = u_{n+1}$$

so that

$$\Delta u_n = \mathbf{E}u_n - u_n$$

The operators Δ and $\mathbf{E}$ both require a certain operation to be performed on the functions to which they are applied, and, of course, are not ordinary algebraic quantities. In spite of this it will be found that much algebraic manipulation of Δ and $\mathbf{E}$ turns out to lead to legitimate results, and with this cautionary word we shall proceed to 'multiply' and to 'expand' them in what follows.

For example, since

$$\Delta u_n = (\mathbf{E}-1)u_n$$

we have a formal relation

$$\Delta \equiv \mathbf{E}-1 \qquad (2.1)$$

The second difference (i. e. the difference of the first difference)

$$\begin{aligned}\Delta^2 u_n &= \Delta u_{n+1} - \Delta u_n \\ &= (u_{n+2}-u_{n+1}) - (u_{n+1}-u_n) \\ &= u_{n+2} - 2u_{n+1} + u_n\end{aligned}$$

can be derived in terms of the function values more swiftly by using (2.1);

$$\begin{aligned}\Delta^2 u_n &= (E-1)^2 u_n \\ &= (E^2-2E+1)u_n\end{aligned}$$

More generally for any positive integer k,

$$\Delta^k u_n = u_{n+k} - \binom{k}{1}u_{n+k-1} + \binom{k}{2}u_{n+k-2} - \ldots + (-1)^k u_n \qquad (2.2)$$

The relation can be used in the other sense, too. From (2.1)

$$E = 1 + \Delta \qquad (2.3)$$

and the function values are given in terms of differences by

$$u_{n+k} = E^k u_n = \Delta^k u_n + \binom{k}{1}\Delta^{k-1}u_n + \binom{k}{2}\Delta^{k-2}u_n + \ldots + u_n \qquad (2.4)$$

If there exists some relation $F(u_n, u_{n+1}, \ldots, u_{n+r}, n) = 0$ between the values of the unknown function u at a set of values of the independent variable n which has extent exactly $r + 1$, then the function u is said to satisfy a difference equation of order r. In the form quoted, the function values themselves appear and the equation is often called a recurrence relation. When it is equivalently expressed in terms of differences, say, in the form $g(u_n, \Delta u_n, \Delta^2 u_n, \ldots, \Delta^r u_n, n) = 0$, it is usually called a difference equation of order r.

If the relation between the function values and the differences is such that it is linear in the unknown function, so that no powers of u or products of u values at different arguments appear, we have a linear difference equation. Unless a non-linear equation happens to have a rather convenient form, its solution in explicit terms is likely to be impossible. A few examples where some progress can be made do occur, but most of our attention is directed towards the solution in explicit

terms of linear difference equations, commencing with those in which all the coefficients of the unknown function or its differences are constants, i. e. the 'linear difference equation with constant coefficients'. The remainder of this chapter deals only with equations of this type.

2. 2 THE HOMOGENEOUS EQUATION

The constant coefficient linear difference equation of order r can be written

$$a_o u_{n+r} + a_1 u_{n+r-1} + \ldots + a_r u_n = \phi(n) \tag{2. 5}$$

where the coefficients a_i $(i = 0, \ldots, r)$ are constants and the function $\phi(n)$ is given. The equation is in homogeneous form if $\phi(n) \equiv 0$. Clearly a solution of the complete (i. e. non-homogeneous) equation can have added to it any multiple of a solution of the homogeneous equation and the sum will still be a solution. Accordingly we now seek the general solution of (2. 5) for the case $\phi(n) \equiv 0$ and in the next section we deal with more general cases.

By trial, it is evident that $u_n = m^n$ is a solution for those values of m which satisfy the equation

$$m^n \{a_o m^r + a_1 m^{r-1} + \ldots + a_r\} = 0 \tag{2. 6}$$

The solution $m = 0$ is of no interest. The remaining condition on the value of m is called the indicial equation. This r^{th} degree polynomial equation has, of course, r roots and the form of the solution of the homogeneous difference equation depends upon whether the roots of the indicial equation are distinct or whether certain of them are repeated.

Case 1. Distinct Roots

Suppose the indicial equation of (2. 6) has r different roots; let them be $m_1, m_2, \ldots, m_r$; then the general solution of the homogeneous equation is

$$u_n = A_1 m_1^n + A_2 m_2^n + \ldots + A_r m_r^n \tag{2. 7}$$

where $A_1, \ldots, A_r$ are constants which may be chosen to satisfy any boundary conditions upon the solution. The most commonly encountered such conditions are when the first r values $u_0, u_1, \ldots, u_{r-1}$ are known.

Example 1. To solve $u_{n+2} - 7u_{n+1} + 12u_n = 0$ subject to the boundary conditions $u_0=2$, $u_1=7$.

The indicial equation is

$$m^2 - 7m + 12 = 0$$

which has roots $m = 3, 4$, and so the general solution of the difference equation is

$$u_n = A3^n + B4^n$$

where A, B are any constants. In order to satisfy the boundary conditions, however, we must obtain the given values for u_0, u_1. Hence

$$\begin{aligned} A + B &= 2 \\ 3A + 4B &= 7 \end{aligned}$$

which give $A = B = 1$, and the required solution is

$$u_n = 3^n + 4^n$$

Case 2. Multiple Roots

Where two or more roots of the indicial equation are coincident, the previous form of the solution (2. 7) loses one or more of its independent constants A_i and it is clear that they could not be uniquely determined by r initial values of the u's. In this case we can try a solution of the form $u_n = m^n v_n$ and hope to obtain an equation for v_n that we can solve. In operator form the homogeneous equation (2. 5) is

$$(a_0 m^{n+r} E^{n+r} + \ldots + a_r m^n E^n) v_0 = 0 \qquad (2.8)$$

which can be written as

$$m^n E^n [f(mE)] v_0 = 0 \qquad (2.9)$$

where $f(x)$ is the polynomial in x of the indicial equation (2.6).

If now we write $E = 1 + \Delta$ and expand the polynomial f by Taylor's theorem, we can write (2.9) as

$$[f(m) + \frac{mf'(m)\Delta}{1!} + \frac{m^2 f''(m)\Delta^2}{2!} + \ldots + \frac{m^r f^{(r)}(m)\Delta^r}{r!}] v_0 = 0 \qquad (2.10)$$

It was mentioned in the last section that we intended to 'expand' the operator expressions in cavalier fashion but that the results would justify the means. They do so here.

We note that the expansion terminates and that it has just $r + 1$ terms since f is a polynomial of degree r ; moreover since we have supposed that m is a multiple root of multiplicity k say, the first $k-1$ derivatives at $x = m$ must vanish.

$$f(m) = f'(m) = \ldots = f^{(k-1)}(m) = 0$$

Hence one solution of (2.8) is given by $\Delta^k v_0 = 0$. In just the same way that differentiating a polynomial of degree r produces a polynomial of degree $r - 1$, so differencing a polynomial reduces the degree by one, for

$$\Delta n^k = (n+1)^k - n^k$$

$$= kn^{k-1} + \text{terms with lower powers of } n$$

Accordingly the solution we need is

$$u_n = (A_1 + A_2 n + \ldots + A_k n^{k-1}) m_1^n + A_{k+1} m_{k+1}^n + \ldots + A_r m_r^n \qquad (2.11)$$

Thus the part of the solution of a homogeneous equation corresponding to a root of the indicial equation which is repeated k times is the product of an arbitrary polynomial of degree $(k-1)$ in n and the root raised to the n^{th} power.