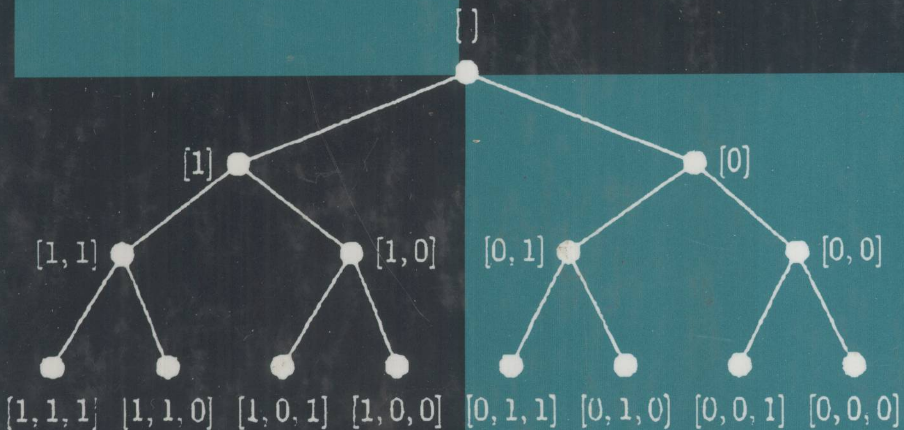
COMBINATORIAL ALGORITHMS

Generation, Enumeration, and Search



Donald L. Kreher
Douglas R. Stinson

COMBINATORIAL ALGORITHMS

Generation, Enumeration, and Search

Donald L. Kreher
Department of Mathematical Sciences
Michigan Technological University

Douglas R. Stinson
Department of Combinatorics and Optimization
University of Waterloo

CRC Press
Boca Raton London New York Washington, D.C.

Library of Congress Cataloging-in-Publication Data

Kreher, Donald L.
Combinatorial algorithms : generation, enumeration, and search / Donald L. Kreher, Douglas R. Stinson.
p. cm. -- (CRC Press series on discrete mathematics and its applications)
Includes bibliographical references and indexes.
ISBN 0-8493-3988-X (alk. paper)
1. Combinatorial analysis. 2. Algorithms. I. Stinson, Douglas R. (Douglas Robert), 1956– . II. Title. III. Series.
QA164.K73 1998
511′.6—dc21

98-41243
CIP

International Standard Book Number 0-8493-3988-X
Library of Congress Card Number 98-41243
Printed in the United States of America 1 2 3 4 5 6 7 8 9 0
Printed on acid-free paper

The CRC Press Series on

DISCRETE MATHEMATICS AND ITS APPLICATIONS

Series Editor

Kenneth H. Rosen, Ph.D.

AT&T Bell Laboratories

Charles J. Colbourn and Jeffrey H. Dinitz,
The CRC Handbook of Combinatorial Designs

Steven Furino, Ying Miao, and Jianxing Yin,
Frames and Resolvable Designs: Uses, Constructions, and Existence

Jacob E. Goodman and Joseph O'Rourke,
Handbook of Discrete and Computational Geometry

Charles C. Lindner and Christopher A. Rodgers
Design Theory

Daryl D. Harms, Miroslav Kraetzl, Charles J. Colbourn, and John S. Devitt,
Network Reliability: Experiments with A Symbolic Algebra Environment

Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone,
Handbook of Applied Cryptography

Richard A. Mollin, Quadratics

Richard A. Mollin, Fundamental Number Theory with Applications

Douglas R. Stinson, Cryptography: Theory and Practice

Preface

Our objective in writing this book was to produce a general, introductory textbook on the subject of combinatorial algorithms. Several textbooks on combinatorial algorithms were written in the 1970s, and are now out-of-date. More recent books on algorithms have either been general textbooks, or books on specialized topics, such as graph algorithms to name one example. We felt that a new textbook on combinatorial algorithms, that emphasizes the basic techniques of generation, enumeration and search, would be very timely.

We have both taught courses on this subject, to undergraduate and graduate students in mathematics and computer science, at Michigan Technological University and the University of Nebraska-Lincoln. We tried to design the book to be flexible enough to be useful in a wide variety of approaches to the subject.

We have provided a reasonable amount of mathematical background where it is needed, since an understanding of the algorithms is not possible without an understanding of the underlying mathematics. We give informal descriptions of the many algorithms in this book, along with more precise pseudo-code that can easily be converted to working programs. C implementations of all the algorithms are available for free downloading from the website

`http://www.math.mtu.edu/~kreher/cages.html`

There are also many examples in the book to illustrate the workings of the algorithms.

The book is organized into eight chapters. Chapter 1 provides some background and notation for fundamental concepts that are used throughout the book. Chapters 2 and 3 are concerned with the generation of elementary combinatorial objects such as subsets and permutations, to name two examples. Chapter 4 presents the important combinatorial search technique called backtracking. It includes a discussion of pruning methods, and the maximum clique problem is studied in detail. Chapter 5 gives an overview of the relatively new area of heuristic search algorithms, including hill-climbing, simulated annealing, tabu search and genetic algorithms. In Chapter 6, we study several basic algorithms for permutation groups, and how they are applied in solving certain combinatorial enumeration and search problems. Chapter 7 uses techniques from the previous chapter to develop algorithms for testing isomorphism of combinatorial objects. Finally, Chapter 8 discusses the technique of basis reduction, which is an important technique in solving certain combinatorial search problems.

There is probably more material in this book than can be covered in one semester. We hope that it is possible to base several different types of courses on this book. An introductory course suitable for undergraduate students could cover most of the material in Chapters 1–5. A second or graduate course could concentrate on the more advanced material in Chapters 6–8. We hope that, aside from its primary purpose as a textbook, researchers and practitioners in all areas of combinatorial computing will find this book useful as a source of algorithms for practical use.

We would like to thank the many people who provided encouragement while we wrote this book, pointed out typos and errors, and gave us useful suggestions on material to include and how various topics should be treated. In particular, we would like to convey our thanks to Mark Chateauneuf, Charlie Colbourn, Bill Kocay, François Margot, Wendy Myrvold, David Olson, Partic Östergård, Jamie Radcliffe, Stanisław Radziszowski and Mimi Williams.

Donald L. Kreher
Douglas R. Stinson

To our wives, Carol and Janet.

Contents

1

Structures and Algorithms

1.1 What are combinatorial algorithms?

In this book, we are primarily interested in the study of algorithms to investigate combinatorial structures. We will call such algorithms *combinatorial algorithms*, and informally classify them according to their desired purpose, as follows.

generation *Construct all the combinatorial structures of a particular type.* Examples of the combinatorial structures we might wish to generate include subsets, permutations, partitions, trees and Catalan families. A generation algorithm will list all the objects under consideration in a certain order, such as a lexicographic order. It may be desirable to predetermine the position of a given object in the generated list without generating the whole list. This leads to the discussion of *ranking*, which is studied in Chapters 2 and 3. The inverse operation of ranking is *unranking* and is also studied in these two chapters.

enumeration *Compute the number of different structures of a particular type.* Every generation algorithm is also an enumeration algorithm, since each object can be counted as it is generated. The converse is not true, however. It is often easier to enumerate the number of combinatorial structures of a particular type than it is to actually list them. For example, the number of k-subsets of an n-element set is

$$\binom{n}{k} = \frac{n!}{(n-k)!k!},$$

which is easily computed. On the other hand, listing all of the k-subsets is more difficult.

There are many situations when two objects are different representations of the "same" structure. This is formalized in the idea of isomorphism of structures. For example, if we permute the names of the vertices of a graph, the resulting two graphs are isomorphic. Enumeration of the number of non-isomorphic structures of a given type often involves algorithms for isomorphism testing, which is the main topic studied in Chapter 7. Algorithms for testing isomorphism depend on various group-theoretic algorithms, which are studied in Chapter 6.

search *Find at least one example of a structure of a particular type (if it exists).* A typical example of a search problem is to find a clique of a specified size in a given graph. Generating algorithms can sometimes be used to search for a particular structure, but for many problems, this may not be an efficient approach. Often, it is easier to find one example of a structure than it is to enumerate or generate all the structures of a specified type.

A variation of a search problem is an optimization problem, where we want to find the optimal structure of a given type. Optimality will be defined for a particular structure according to some specified measure of "profit" or "cost". For example, the maximum clique problem requires finding a clique of largest size in a given graph. (The size of a clique is the number of vertices it contains.)

Many interesting and important search and optimization problems belong to the class of NP-hard problems, for which no efficient (i.e., polynomial-time) algorithms are known to exist. The maximum clique problem mentioned above falls into this class of problems. For NP-hard problems, we will often use algorithms based on the idea of backtracking, which is the topic of Chapter 4. An alternative approach is to try various types of heuristic algorithms. This topic is discussed in Chapters 5 and 8.

1.2 What are combinatorial structures?

The structures we study in this book are those that can be described as collections of k-element subsets, k-tuples, or permutations from a parent set. Given such a structure, we may be interested in all of the substructures contained within it of a particular or optimal type. On the other hand, we may wish to study all structures of a given form. We introduce some of the types of combinatorial structures in this section that will be used in later parts of the book.

1.2.1 Sets and lists

The basic building blocks of combinatorial structures are finite sets and lists. We review some basic terminology and notation now.

A (finite) *set* is a finite collection of objects called the *elements* of the set. We write the elements of a set in brace brackets. For example, if we write $X = \{1, 3, 7, 9\}$, then we mean that X is a set that contains the elements $1, 3, 7$ and 9.

A set is an unordered structure, so $\{1, 3, 7, 9\} = \{7, 1, 9, 3\}$, for example. Also, the elements of a set are distinct. We write $x \in X$ to indicate that x is an element of the set X.

The *cardinality* (or size) of a set X, denoted $|X|$, is the number of elements in X. For example, $|\{1, 3, 7, 9\}| = 4$. For a nonnegative integer k, a *k-set* is a set of cardinality k. The *empty set* is the (unique) set that contains no elements. It is a 0-set and is denoted by $\emptyset$.

If X and Y are sets, then we say that X is a *subset* of Y if every element of X is an element of Y. This is equivalent to the following condition:

$$x \in X \Rightarrow x \in Y.$$

If X is a subset of Y, then we write $X \subseteq Y$. A *k-subset* of Y is a subset of Y that has cardinality k.

A (finite) *list* is an ordered collection of objects which are called the *items* of the list. We write the items of a list in order between square brackets. For example, if we write $X = [1, 3, 1, 9]$, then we mean that X is the list that contains the items $1, 3, 1$ and 9 in that order. Since a set is an ordered structure, it follows that $[1, 3, 1, 9] \neq [1, 1, 9, 3]$, for example. Note that the items of a list need not be distinct.

The *length* of a list X is the number of items (not necessarily distinct) in X. For example, $[1, 3, 1, 9]$ is a list of length 4. For a nonnegative integer n, an *n-tuple* is a list of length n. The *empty list* is the (unique) list that contains no elements; it is written as $[\ ]$. If X is a list of length n, then the items in X are denoted $X[0], X[1], \ldots, X[n-1]$, in order. We usually denote the first item in the list X as $X[0]$, as is done in the C programming language. Thus, if $X = [1, 3, 1, 9]$, then $X[0] = 1$, $X[1] = 3$, $X[2] = 1$ and $X[3] = 9$. However, in some situations, we may list the elements of X as $X[1], X[2], \ldots, X[n]$. An alternative notation for a list, that we will sometimes use, is to write the items in the list X in subscripted form, as $X_0, X_1, \ldots, X_{n-1}$.

The *Cartesian product* (or cross product) of the sets X and Y, denoted by $X \times Y$, is the set of all ordered pairs whose first item is in X and whose second item is in Y. Thus

$$X \times Y = \{[x, y] : x \in X \text{ and } y \in Y\}.$$

For example, if $X = \{1, 3, 7, 9\}$ and $Y = \{0, 2, 4\}$, then

$$\{1, 3, 7, 9\} \times \{0, 2\} = \{[1, 0], [1, 2], [3, 0], [3, 2], [7, 0], [7, 2], [9, 0], [9, 2]\}.$$

If X is a finite set of cardinality n, then a *permutation* of X is a list π of length n such that every element of X occurs exactly once in the list π. There are exactly $n! = n(n-1) \cdots 1$ permutations of an n-set. For a positive integer $k < n$, a *k-permutation* of X is a list π of length k such that every element of X occurs at most once in the list π. There are exactly

$$\frac{n!}{(n-k)!} = n(n-1) \cdots (n-k+1)$$

k-permutations of an n-set.

1.2.2 Graphs

We begin by defining the concept of a graph.

Definition 1.1: A *graph* consists of a finite set $\mathcal{V}$ of *vertices* and a finite set $\mathcal{E}$ of *edges*, such that each edge is a two element subset of vertices. It is customary to write a graph as an ordered pair, $(\mathcal{V}, \mathcal{E})$.

A *complete graph* is a graph in which $\mathcal{E}$ consists of all two element subsets of $\mathcal{V}$. If $|\mathcal{V}| = n$, then the complete graph is denoted by K_n.

We will usually represent the vertices of a graph $(\mathcal{V}, \mathcal{E})$ by dots, and join two vertices x and y by a line whenever $\{x, y\} \in \mathcal{E}$. A vertex x is *incident* with an edge e if $x \in e$. The *degree* of a vertex $x \in \mathcal{V}$, denoted by $\deg(x)$, is the number of edges that are incident with the vertex x. A graph is *regular* of degree d if every vertex has degree d. In Example 1.1 we present a graph that is regular of degree three.

Example 1.1 *A graph*
Let $\mathcal{V} = \{0, 1, 2, 3, 4, 5, 6, 7\}$, and let $\mathcal{E} = \{\{0, 1\}, \{0, 2\}, \{2, 3\}, \{1, 3\}, \{0, 4\}, \{1, 5\}, \{2, 6\}, \{3, 7\}, \{4, 5\}, \{4, 6\}, \{6, 7\}, \{5, 7\}\}$. This graph is called the *cube* and can be represented by the diagram in Figure 1.1. □

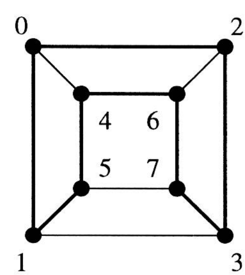


FIGURE 1.1
The cube and a Hamiltonian circuit.

One of the many interesting substructures that can occur in a graph is a *Hamiltonian circuit*. This is a closed path that passes through every vertex exactly once. The list $[0, 2, 3, 7, 6, 4, 5, 1]$ describes a Hamiltonian circuit in the graph in Figure 1.1. It is indicated by thick lines in the diagram. Note that different lists can represent the same Hamiltonian circuit. In fact, there are $2n$ such lists, where n is the number of vertices in the graph, since we can pick any of the n vertices as the "starting point" and then traverse the circuit in two possible directions.

A graph $(\mathcal{V}, \mathcal{E})$ is a *weighted graph* if there is a weight function $w : \mathcal{E} \rightarrow \mathbb{R}$ associated with it. The weight of a substructure, such as a Hamiltonian circuit, is defined to be the sum of the weights of its edges. Finding a smallest weight Hamiltonian circuit in a weighted graph is called the Traveling Salesman problem and is discussed in Chapter 4.

Another example of a substructure in a graph is a clique. A *clique* in a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a subset $S \subseteq \mathcal{V}$ such that $\{x, y\} \in \mathcal{E}$ for all $x, y \in S$, $x \neq y$. A clique that has the maximum size among all cliques of $\mathcal{G}$ is called a *maximum clique*. In Figure 1.1, every edge $\{x, y\}$ determines a maximum clique of the graph, since there are no cliques of size 3. Finding a maximum clique in a graph is called the Maximum Clique problem and is discussed in Chapter 4.

1.2.3 Set systems

We next define a generalization of a graph called a set system.

Definition 1.2: A *set system* consists of a finite set $\mathcal{X}$ of *points* and a finite set $\mathcal{B}$ of *blocks*, such that each block is a subset of $\mathcal{X}$. We use the notation $(\mathcal{X}, \mathcal{B})$ to denote a set system.

Observe that a graph is just a set system in which every block has cardinality two. Another simple example of a set system is a partition of a set $\mathcal{X}$. A *partition* is a set system $(\mathcal{X}, \mathcal{B})$ in which $A \cap B = \emptyset$ for all $A, B \in \mathcal{B}$ with $A \neq B$, and $\cup_{A \in \mathcal{B}} A = \mathcal{X}$.

We now define another, more complicated, combinatorial structure, and then formulate it as a special type of set system.

Definition 1.3: A *Latin square* of order n is an n by n array A, whose entries are chosen from an n-element set $\mathcal{Y}$, such that each symbol in $\mathcal{Y}$ occurs in exactly one cell in each row and in each column of A.

Example 1.2 *A Latin square of order four*
Let $\mathcal{Y} = \{1, 2, 3, 4\}$ and let

$A =$

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

.

□

Suppose that A is a Latin square of order n on a symbol set $\mathcal{Y}$. Label the n rows of A with the n elements of $\mathcal{Y}$, and likewise label the n columns of A with the n elements of $\mathcal{Y}$. Now define a set system $(\mathcal{X}, \mathcal{B})$ as follows:

$$\mathcal{X} = \mathcal{Y} \times \{1, 2, 3\},$$

and

$$\mathcal{B} = \{\{(y_1, 1), (y_2, 2), (A[y_1, y_2], 3)\} : y_1, y_2 \in \mathcal{Y}\}.$$

If we start with the Latin square of order four that was presented in Example 1.2, then we obtain the following set system $(\mathcal{X}, \mathcal{B})$:

$$\mathcal{X} = \{1, 2, 3, 4\} \times \{1, 2, 3\}$$

$$\mathcal{B} = \left\{ \begin{array}{l} \{[1,1],[1,2],[1,3]\},\ \{[1,1],[2,2],[2,3]\}, \\ \{[1,1],[3,2],[3,3]\},\ \{[1,1],[4,2],[4,3]\}, \\ \{[2,1],[1,2],[2,3]\},\ \{[2,1],[2,2],[1,3]\}, \\ \{[2,1],[3,2],[4,3]\},\ \{[2,1],[4,2],[3,3]\}, \\ \{[3,1],[1,2],[3,3]\},\ \{[3,1],[2,2],[4,3]\}, \\ \{[3,1],[3,2],[1,3]\},\ \{[3,1],[4,2],[2,3]\}, \\ \{[4,1],[1,2],[4,3]\},\ \{[4,1],[2,2],[3,3]\}, \\ \{[4,1],[3,2],[2,3]\},\ \{[4,1],[4,2],[1,3]\} \end{array} \right\}$$

If we look at the blocks in this set system we see that every pair of the form $\{(y_1, i), (y_2, j)\}$, where $i \neq j$, occurs in exactly one block in $\mathcal{B}$. This motivates the following definition.

Definition 1.4: A *transversal design* $TD(n)$ is a set system $(\mathcal{X}, \mathcal{B})$ for which there exists a partition $\{\mathcal{X}_1, \mathcal{X}_2, \mathcal{X}_3\}$ of $\mathcal{X}$ such that the following properties are satisfied:

1. $|\mathcal{X}| = 3n$, and $|\mathcal{X}_i| = n$ for $1 \leq i \leq 3$,
2. For every $B \in \mathcal{B}$ and for $1 \leq i \leq 3$, $|B \cap \mathcal{X}_i| = 1$.
3. For every $x \in \mathcal{X}_i$ and every $y \in \mathcal{X}_j$ with $i \neq j$, there exists a unique block $B \in \mathcal{B}$ such that $\{x, y\} \subseteq B$.

If we chose the partition $\{\mathcal{X}_1, \mathcal{X}_2, \mathcal{X}_3\}$ of $\mathcal{X} = \{1, 2, 3, 4\} \times \{1, 2, 3\}$ so that

$$\mathcal{X}_1 = \{[1,1],[2,1],[3,1],[4,1]\}$$

$$\mathcal{X}_2 = \{[1,2],[2,2],[3,2],[4,2]\},$$

and

$$\mathcal{X}_3 = \{[1,3],[2,3],[3,3],[4,3]\},$$

then it is easy to check that the set system that we constructed above from the Latin square of order four is a $TD(4)$. In general, we can construct a $TD(n)$ from a Latin square of order n, and vice versa.