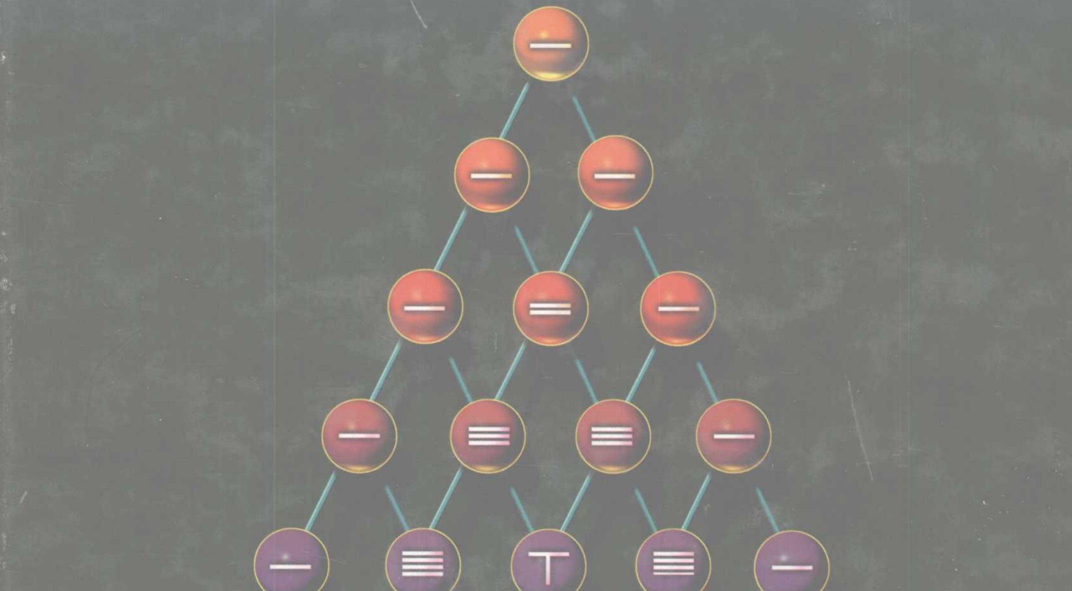
APPLIED COMBINATORICS

FOURTH EDITION



ALAN TUCKER

APPLIED COMBINATORICS

ALAN TUCKER

SUNY Stony Brook

JOHN WILEY & SONS, INC.

Acquisitions Editor *Kimberly Murphy*
Marketing Manager *Julie Lindstrom*
Senior Production Editor *Petrina Kulek*
Director Design *Madelyn Lesure*
Illustration Coordinator *Gene Aiello*

This book was set in Times Roman by TechBooks and printed and bound by Von-Courier/Westford.
The cover was printed by Phoenix Color.

This book is printed on acid-free paper. ♾

ISBN 0-471-43809-X

Printed in the United States of America

10 9 8 7 6 5 4

PREFACE

Combinatorial reasoning underlies all analysis of computer systems. It plays a similar role in discrete operations research problems and in finite probability. Two of the most basic mathematical aspects of computer science concern the speed and logical structure of a computer program. Speed involves enumeration of the number of times each step in a program can be performed. Logical structure involves flow charts, a form of graphs. Analysis of the speed and logical structure of operations research algorithms to optimize efficient manufacturing or garbage collection entails similar combinatorial mathematics. Determining the probability that one of a certain subset of equally likely outcomes occurs requires counting the size of the subset. Such combinatorial probability is the basis of many nonparametric statistical tests. Thus, enumeration and graph theory are used pervasively throughout the mathematical sciences.

This book teaches students in the mathematical sciences how to reason and model combinatorially. It seeks to develop proficiency in basic discrete math problem solving in the way that a calculus textbook develops proficiency in basic analysis problem solving.

The three principal aspects of combinatorial reasoning emphasized in this book are: the systematic analysis of different possibilities, the exploration of the logical structure of a problem (e.g., finding manageable subpieces or first solving the problem with three objects instead of n), and ingenuity. Although important uses of combinatorics in computer science, operations research, and finite probability are mentioned, these applications are often used solely for motivation. Numerical examples involving the same concepts use more interesting settings such as poker probabilities or logical games.

Theory is always first motivated by examples, and proofs are given only when their reasoning is needed to solve applied problems. Elsewhere, results are stated without proof, such as the form of solutions to various recurrence relations, and then applied in problem solving. Occasionally, a few theorems are stated simply to give students a flavor of what the theory in certain areas is like.

Since 1980, collegiate curriculum recommendations from the Mathematical Association of America have included combinatorial problem solving as an important component of training in the mathematical sciences. Combinatorial problem solving underlies a wide spectrum of important subjects in the computer science curriculum. Indeed, it is expected that most students in a course using this book will be computer science majors. For both mathematics majors and computer science majors, this author believes that general reasoning skills stressed here are more important than mastering a variety of definitions and techniques.

This book is designed for use by students with a wide range of ability and maturity (sophomores through beginning graduate students). The stronger the students, the harder the exercises that can be assigned. The book can be used for a one-quarter, two-quarter, or one-semester course depending on how much material is used. It may also be used for a one-quarter course in applied graph theory or a one-semester or one-quarter course in enumerative combinatorics (starting from Chapter 5). A typical one-semester undergraduate discrete methods course should cover most of Chapters 1 to 3 and 5 to 8, with selected topics from other chapters if time permits.

Instructors are strongly encouraged to obtain a copy of the instructor's guide accompanying this book. The guide has an extensive discussion of common student misconceptions about particular topics, hints about successful teaching styles for this course, and sample course outlines (weekly assignments, tests, etc.).

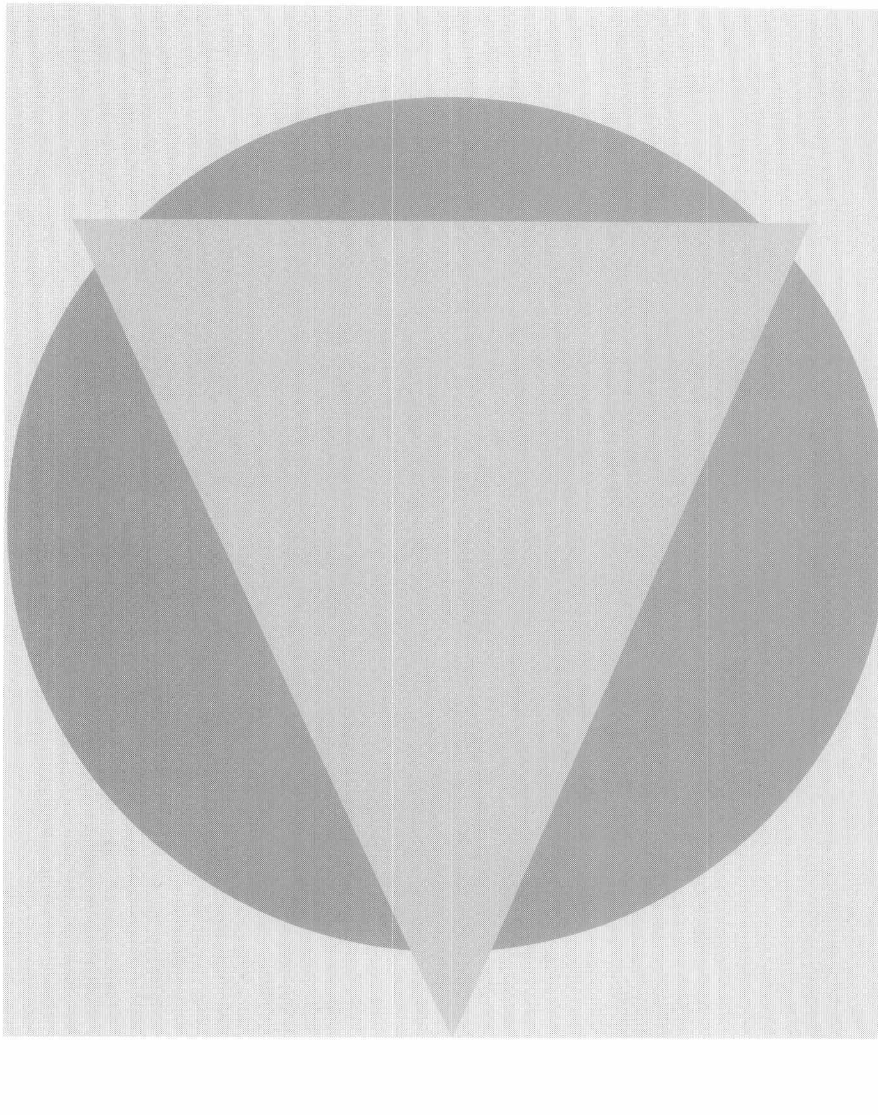
The fourth edition of this book, like the second and third editions, puts the graph theory chapters first. The pictures associated with graphs have proven to be a valuable aid in introducing students to combinatorial reasoning. The reasoning required in enumeration problems quickly becomes very challenging—for too many students, overwhelming—without the preparation in combinatorial reasoning provided by graph theory. This edition has new examples, expanded discussions, and additional exercises throughout the text.

Many people gave useful comments about early drafts and the first edition of this text; Jim Frauenthal and Doug West were especially helpful. The idea for this book is traceable to a combinatorics course taught by George Dantzig and George Polya at Stanford in 1969, a course for which I was the grader. Many instructors who have used earlier editions of this book have supplied me with valuable feedback and suggestions that have, I hope, made this edition better. I gratefully acknowledge my debt to them. Ultimately, my interest in combinatorial mathematics and in its effective teaching rests squarely on the shoulders of my father, A. W. Tucker, who had long sought to give finite mathematics a greater role in mathematics as well as in the undergraduate mathematics curriculum. Finally, special thanks go to former students of my combinatorial mathematics courses at Stony Brook. It was *they* who taught *me* how to teach this subject.

Alan Tucker
Stony Brook, New York

CONTENTS

PART ONE
GRAPH THEORY



CHAPTER 1
ELEMENTS OF GRAPH THEORY

1.1 GRAPH MODELS

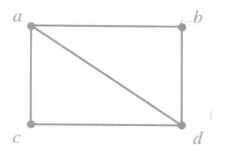
The first four chapters deal with graphs and their applications. A **graph** $G = (V, E)$ consists of a finite set V of **vertices** and a set E of **edges** joining different pairs of distinct vertices.* Figure 1.1a shows a depiction of a graph with $V = \{a, b, c, d\}$ and $E = \{(a, b), (a, c), (a, d), (b, d), (c, d)\}$. We represent vertices with points and edges with lines joining the prescribed pairs of vertices. This definition of a graph does not allow two edges to join the same two vertices. Also an edge cannot "loop" so that both ends terminate at the same vertex—an edge's end vertices must be distinct. The two ends of an undirected edge can be written in either order, (b, c) or (c, b). We say that vertex a is **adjacent** to vertex b when there is an edge from a to b.

Sometimes the edges are ordered pairs of vertices, called **directed edges**. In a **directed graph**, all edges are directed. See the directed graph in Figure 1.1b. We write $(\vec{b, c})$ to denote a directed edge from b to c. In a directed graph, we allow one edge in each direction between a pair of vertices. See edges $(\vec{a, c})$ and $(\vec{c, a})$ in Figure 1.1b.

The combinatorial reasoning required in graph theory, and later in the enumeration part of this book, involves different types of analysis than used in calculus and high school mathematics. There are few general rules or formulas for solving these problems. Instead, each question usually requires its own particular analysis. This analysis sometimes calls for clever model building or creative thinking but more often consists of breaking the problem into many cases (and subcases) that are easy enough to solve with simple logic or basic counting rules. A related line of reasoning is to solve a special case of the given problem and then to find ways to extend that reasoning to all the other cases that may arise. In graph theory, combinatorial arguments are made a little easier by the use of pictures of the graphs. For example, a case-by-case argument is much easier to construct when one can draw a graphical depiction of each case.

Graphs have proven to be an extremely useful tool for analyzing situations involving a set of elements in which various pairs of elements are related by some property. The most obvious examples of graphs are sets with physical links, such as electrical networks, where electrical components (transistors) are the vertices and

*What this book calls a graph is referred to in many graph theory books as a *simple graph*. In general, graph theory terminology varies a little from book to book.



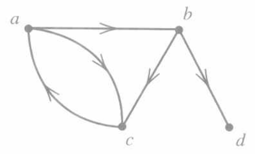


Figure 1.1

connecting wires are the edges; or telephone communication systems, where telephones and switching centers are the vertices and telephone lines are the edges. Road maps, oil pipelines, and subway systems are other examples.

Another natural form of graphs is sets with logical or hierarchical sequencing, such as computer flow charts, where the instructions are the vertices and the logical flow from one instruction to possible successor instruction(s) defines the edges; or an organizational chart, where the people are the vertices and if person A is the immediate superior of person B then there is an edge $(\vec{A}, B)$. Computer data structures, evolutionary trees in biology, and the scheduling of tasks in a complex project are other examples.

The emphasis in this book will be on problem solving, with problems about general graphs and applied graph models. Observe that we will not have any numbers to work with, only some vertices and edges. At first, this may seem to be highly nonmathematical. It is certainly very different from the mathematics that one learns in high school or in calculus courses. However, disciplines such as computer science and operations research contain as much graph theory as they do standard numerical mathematics.

This section consists of a collection of illustrative examples about graphs. We will solve each problem from scratch with a little logic and systematic analysis. Many, of these examples will be revisited in greater depth in subsequent chapters.

The following three graph theory terms are used in the coming examples. A **path** P is a sequence of distinct vertices, written $P = x_1\text{–}x_2\text{–}\cdots\text{–}x_n$, with each pair of consecutive vertices in P joined by an edge. If in addition, there is an edge (x_n, x_1), the sequence is called a **circuit**, written $x_1\text{–}x_2\text{–}\cdots\text{–}x_n\text{–}x_1$. For example, in Figure 1.1*a*, *b-d-a-c* forms a path, while *a-b-d-c-a* forms a circuit. A graph is **connected** if there is a path between every pair of vertices. The removal of certain edges or vertices from a connected graph G is said to *disconnect* the graph if the resulting graph is no longer connected; that is, if at least one pair of vertices is no longer joined by a path. The graph in Figure 1.1*a* is connected but the removal of edges (a, b) and (b, d) will disconnect it.

Example 1: Matching

Suppose that we have five people A, B, C, D, E and five jobs a, b, c, d, e, and various people are qualified for various jobs. The problem is to find a feasible one-to-one matching of people to jobs, or to show that no such matching can exist. We can represent this situation by a graph with vertices for each person and for each job, with edges joining people with jobs for which they are qualified. Does there exist a feasible matching of people to jobs for the graph in Figure 1.2?

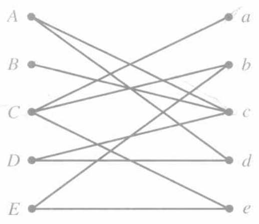


Figure 1.2

C,g,d,ce

C, c, E, d

The answer is no. The reason can be found by considering people *A*, *B*, and *D*. These three people as a set are collectively qualified for only two jobs, *c* and *d*. Hence there is no feasible matching possible for these three people, much less all five people. An algorithm for finding a feasible matching, if any exists, will be presented in Chapter 4. Such matching graphs in which all the edges go horizontally between two sets of vertices are called **bipartite**. Bipartite graphs are discussed further in Section 1.3. ∎

?

Example 2: Spelling Checker

A spelling checker looks at each word X (represented in a computer as a binary number) in a document and tries to match X with some word in its dictionary. The dictionary typically contains close to 100,000 words. To understand how this checking works, we consider the simplified problem of matching an unknown letter X with one of the 26 letters in the English alphabet. In the spirit of the strategy humans use to home in on the page in a dictionary where a given word appears, the computer search procedure would first compare the unknown letter X with M, to determine whether X $\le$ M or X $>$ M. The answer to this comparison locates X in the first 13 letters of the alphabet or the second 13 letters, thus cutting the number of possible letters for X in half. This strategy of cutting the possible matches in half can be continued with as many comparisons as needed to home in on X's letter. For example, if X $\le$ M, then we could test whether or not X $\le$ G; if X $>$ M, we could test whether X $\le$ S.

This testing procedure is naturally represented by a directed graph called a **tree**. Figure 1.3 shows the first three rounds of comparisons for the letter-matching procedure. The vertices represent the different letters used in the comparisons. The left descending edge from a vertex Q points to the letter for the next comparison if X $\le$ Q and the right descending edge from Q points to the next letter if X $>$ Q.

For our original spelling-checker problem, a word processor would use a similar, but larger, tree of comparisons. With just 12 rounds of comparisons, it could reduce

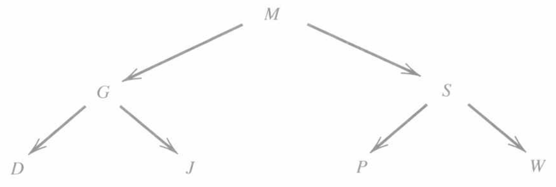


Figure 1.3

3 a d g k

4 b c e f h j

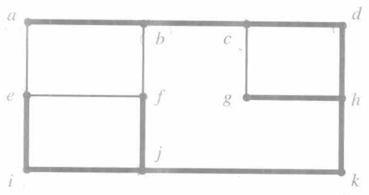


Figure 1.4

the number of possible matches for an unknown word X from 100,000 down to 25, about the number of words in a column of a page in a dictionary. (Once reduced to a list of about 25 possibilities, the search for X would usually run linearly down that list, just as a human would.) ■

Chapter 3 examines trees and their use in various search problems. Trees can be characterized as graphs that are connected and have a unique path between any pair of vertices (ignoring the directions of directed edges). The next example uses trees in a very different way.

Example 3: Network Reliability

Suppose the graph in Figure 1.4 represents a network of telephone lines (or electrical transmission lines). We are interested in the network's vulnerability to accidental disruption. We want to identify those lines and switching centers that must stay in service to avoid disconnecting the network.

There is no telephone line (edge) whose removal will disconnect the telephone network (graph). Similarly, there is no vertex whose removal disconnects the graph.

Is there any pair of edges whose removal disconnects the graph? There are several such pairs. For example, we see that if the two edges incident to *a* are removed, vertex *a* is isolated from the rest of the network. A more interesting disconnecting pair of edges is (*b*, *c*), (*j*, *k*). It is left as an exercise to find all disconnecting sets consisting of two edges for the graph in Figure 1.4.

Let us take a different tack. Suppose we want to find a minimal set of edges needed to link together the eleven vertices in Figure 1.4. There are several possible minimal connecting sets of edges. By inspection, we find the following one: (*a*, *b*), (*b*, *c*), (*c*, *d*), (*d*, *h*), (*h*, *g*), (*h*, *k*), (*k*, *j*), (*j*, *f*), (*j*, *i*), (*i*, *e*); the edges in this minimal connecting set are darkened in Figure 1.4. A minimal disconnecting set will always be a tree. One interesting general result about these sets is that if the graph *G* has *n* vertices, then a minimal connecting set for *G* (if any exists) always has $n - 1$ edges. ■

The number of edges incident to a vertex is called the **degree** of the vertex.

Example 4: Street Surveillance

Now suppose the graph in Figure 1.4 represents a section of a city's street map. We want to position police at corners (vertices) so that they can keep every block (edge) under surveillance, that is, every edge should have police at (at least) one of its end vertices. What is the smallest number of police that can do this job?

Let us try to get a lower bound on the number of police needed. The map has 14 blocks (edges). Corners b, c, e, f, h, and j each have degree 3, and corners a, d, g, i, and k each have degree 2. Since 4 vertices can be incident to at most $4 \times 3 = 12$ edges but there are 14 edges in all, we will need at least five police. If all five police were positioned at degree-3 vertices, then $5 \times 3 = 15$ edges are watched by the 5 police. Since there are only 14 edges, some edge would be covered by police at both end vertices. If four police are at degree-3 vertices and one at a degree-2 vertex, then exactly 14 edges are watched—and no edge need be covered at both ends. (If fewer than 4 of the 5 police are at degree-3 vertices, we could not watch all 14 edges). With these general observations, we are ready for a systematic analysis to try to find 5 vertices that are collectively adjacent to all 14 edges.

Consider edge (c, d). Suppose it is watched by an officer at vertex d. Then vertex c (the other end vertex of edge (c, d)) cannot also have an officer, since we noted above that if we use a degree-2 vertex, such as d, then no edge can be watched from both end vertices. However, if vertex c cannot be used, then edge (c, g) must be watched from its other end vertex g. But now we are using two degree-2 vertices, d and g. We noted above that at most one of the five police can be placed at a degree-2 vertex. We got into this trouble by assuming that edge (c, d) is watched from vertex d.

Now assume no officer is at vertex d. Then we must watch edge (c, d) with an officer at vertex c. Since vertex d cannot be used, edge (d, h) can be watched only by placing an officer at vertex h. Next look at edge (h, k). It is already watched by vertex h. Then we assert that (h, k) cannot also be watched by an officer at vertex k, since k has degree-2 and we noted above that if we use a degree-2 vertex, no edge can be watched from both ends. We conclude that there cannot be an officer at vertex k. Then edge (k, j) can be watched only by placing an officer at vertex j. We now have officers required to be at vertices c, h, and j.

Similar reasoning shows that with an officer at vertex j, there cannot be an officer at vertex i; then there must be an officer at vertex e; there cannot be an officer at vertex a; and there must be an officer at vertex b. In sum, we have shown that we should place police at vertices c, h, j, e, and b. A check shows that these five vertices do indeed watch all 14 edges. Since our reasoning forced us to use these five vertices, no other set of five vertices can work.

At the beginning of this example, we showed that at least five corners were needed to keep all the blocks (edges) under surveillance. Now we have produced a set of five corners that achieve such surveillance. It then follows that five is the minimum number of corners.

We conclude this example by noting that in this surveillance situation, one can also consider watching the vertices rather than the edges: How few officers are needed to watch all the vertices? We use the same type of argument as in the block surveillance problem to get a lower bound on the number of corners needed for corner surveillance. An officer at vertex x is considered to be watching vertex x and all vertices adjacent to x. There are 11 vertices, and 6 of these vertices watch 4 vertices (themselves and 3 adjacent vertices). Thus three is the theoretical minimum. This minimum can be achieved. Details are left as an exercise. ■

A set C of vertices in a graph G with the property that every edge of G is incident to at least one vertex in C is called an **edge cover**. The previous example was asking for an edge cover of minimal size in Figure 1.4. The reasoning in Example 4 illustrates the kind of systematic case-by-case analysis that is common in graph theory.

One goal of graph theory is to find useful relationships between seemingly unrelated graph concepts that arise from different settings. The following example introduces the concept of independent sets, which have an unexpected relation to edge covers.

Example 5: Scheduling Meetings

Consider the following scheduling problem. A state legislature has many committees that meet for one hour each week. One wants a schedule of committee meeting times that minimizes the total number of hours but such that two committees with overlapping membership do not meet at the same time.

This situation can be modeled with a graph in which we create a vertex for each committee and join two vertices by an edge if they represent committees with overlapping membership. Suppose that the graph in Figure 1.4 now represents the membership overlap of 11 legislative committees. For example, vertex c's edges to vertices b, d, and g in Figure 1.4 indicate that committee c has overlapping members with committees b, d, and g.

A set of committees can all meet at the same time if there are no edges between the corresponding set of vertices. A set of vertices without an edge between any two is called an **independent set** of vertices. Our scheduling problem can now be restated as seeking a minimum number of independent sets that collectively include all vertices. This problem is discussed in depth in Section 2.3.

If we want to know how many committees can meet at one time, we are asking the graph question: What is the largest independent set of the graph? Although it is very hard in general to find the largest independent set in a graph, for the graph in Figure 1.4, a little examination shows that there is one independent set of size 6, a, d, f, g, i, k. All other independent sets have five or fewer vertices.

Now for the unexpected link with edge covers. If V is the set of vertices in a graph G, then I will be an independent set of vertices if and only if $V - I$ is an edge cover! Why? Because if there are no edges between two vertices in I, then every edge involves (at least) one vertex not in I, that is, a vertex in $V - I$. Conversely, if C is an edge cover so that all edges have at least one end vertex in C, then there is no edge joining two vertices in $V - C$. So $V - C$ is an independent set. Check that in Figure 1.4, the vertices not in the independent set a, d, f, g, i, k form edge cover b, c, e, h, j.

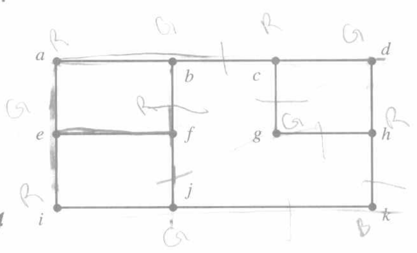


Figure 1.4

A consequence of this relationship is that if I is an independent set of largest possible size in a graph, then $V - I$ will be an edge cover of smallest possible size. So finding a maximal independent set is equivalent to finding a minimal edge cover. ■

Instead of modeling a geometric or other problem with a graph, it is possible to model a graph with a geometric configuration. A graph G is called an **interval graph** if there is a one-to-one correspondence between the vertices of G and a collection of intervals on the line so that two vertices of G are adjacent when the corresponding intervals overlap. We now consider two problems in which we model a real-world situation with a graph and then seek to model the resulting graph with an overlapping family of intervals. In Section 2.3 an interval graph model arises in VLSI (Very Large Scale Integrated) circuitry design.

Example 6: Interval Graph Modeling

A competition graph, used in ecology, has a vertex for each of a given set of species and an edge joining each pair of species that feed on a common prey (i.e., the two species compete for food). See the competition graph in Figure 1.5*a*. Ecologists have given considerable attention to viewing competition graphs as interval graphs for the following reason (see reference [3] for further details). Each species is thought of having a "niche" in its ecological system, its position in terms of food it eats and

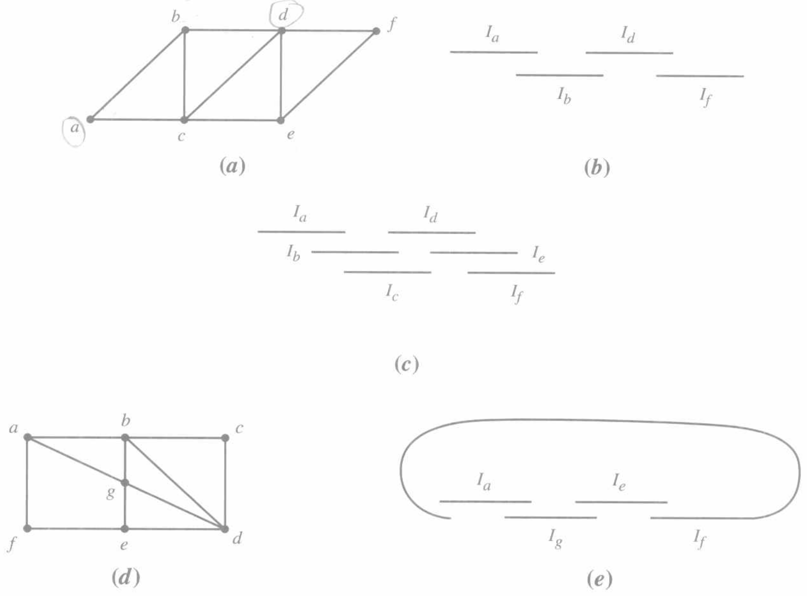


Figure 1.5