Chapter 3

Introduction to Graph Theory

3.1 FUNDAMENTAL CONCEPTS[1]

3.1.1 Some Examples

In Example 1.4 we introduced informally the notion of a graph. In this chapter we study graphs and directed analogues of graphs, called digraphs, and their numerous applications. Graphs are a fundamental tool in solving problems of combinatorics. In turn, many of the counting techniques of combinatorics are especially useful in solving problems of graph theory. The theory of graphs is an old subject that has been undergoing a tremendous growth in interest in recent years. From the beginning, the subject has been closely tied to applications. It was invented by Euler [1736] in the process of settling the famous Königsberg bridge problem, which we discuss in Section 11.3.1. Graph theory was later applied by Kirchhoff [1847] to the study of electrical networks, by Cayley [1857, 1874] to the study of organic chemistry, by Hamilton to the study of puzzles, and by many mathematicians and nonmathematicians to the study of maps and map-coloring. In the twentieth cen-

[1] The topics in graph theory introduced in this chapter were chosen for three reasons. First, they illustrate quickly the nature and variety of applications of the subject. Second, they can be used to illustrate the counting techniques introduced in Chapter 2. Third, they will be used to illustrate the counting and existence results in Chapters 5–8. We return to graph theory more completely in Chapter 11, which begins a sequence of three chapters on graphs and networks and begins an introduction to the algorithmic aspects of graph theory. In the undergraduate combinatorics course at Rutgers taught by Fred Roberts, he does not cover much graph theory, as there is a separate undergraduate graph theory course. Accordingly, he goes through this chapter very rapidly. He covers Sections 3.1.1 and 3.1.2; all of Section 3.2 (but in about 30 minutes, with little emphasis on the exercises); 3.3.1, 3.3.3; 3.4.1, 3.4.2; 3.5.1, 3.5.2, 3.5.4, 3.5.5 (only the proof of Theorem 3.16); and 3.5.6 (without sketching the proof of Cayley's Theorem). Other sections can be added to expand on the material covered. In a graph theory course or in a combinatorics course with more emphasis on graphs or on computing or on theory, more material from this chapter should be included.

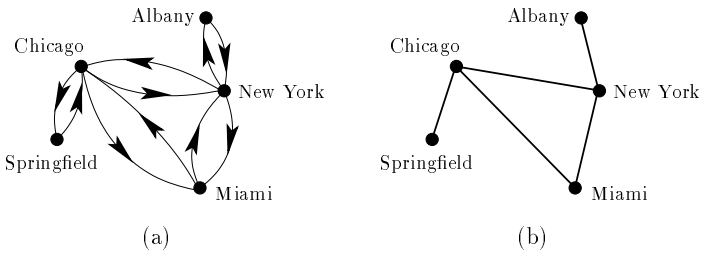


Figure 3.1: Direct air links.

tury, graph theory has been used increasingly in electrical engineering, computer science, chemistry, political science, ecology, molecular biology, transportation, information processing, and a variety of other fields.

To illustrate applications of graph theory, and to motivate the formal definitions of graph and digraph that we introduce, let us consider several examples.

Example 3.1 Transportation Networks Graphs and digraphs arise in many transportation problems. For example, consider any set of locations in a given area, between which it is desired to transport goods, people, cars, and so on. The locations could be cities, warehouses, street corners, airfields, and the like. Represent the locations as points, as shown in the example of Figure 3.1(a), and draw an arrow or directed line (or curve) from location x to location y if it is possible to move the goods, people, and so on, directly from x to y. The situation where all the links are two-way can be more simply represented by drawing a single undirected line between two locations that are directly linked rather than by drawing two arrows for each pair of locations [see Figure 3.1(b)]. Interesting questions about transportation networks are how to design them to move traffic efficiently, how to make sure that they are not vulnerable to disruption, and so on. ■

Example 3.2 Communication Networks Graphs are also used in the study of communications. Consider a committee, a corporate body, or any similar organization in which communication takes place. Let each member of the organization be represented by a point, as in Figure 3.2, and draw a line with an arrow from member x to member y if x can communicate directly with y. For example, in the police force of Figure 3.2, the captain can communicate directly with the dispatcher, who in turn can reach the captain via either of the lieutenants.[2] Typical questions asked about such a "communication network" are similar to questions about transportation networks: How can the network be designed efficiently, how easy is it to disrupt communications, and so on? The modern theory of communication networks is often concerned with networks of interacting communications and computing devices, and the graphs that arise are huge. ■

[2]See Kemeny and Snell [1962, Ch. 8] for a more detailed discussion of a similar communication network of a police force.

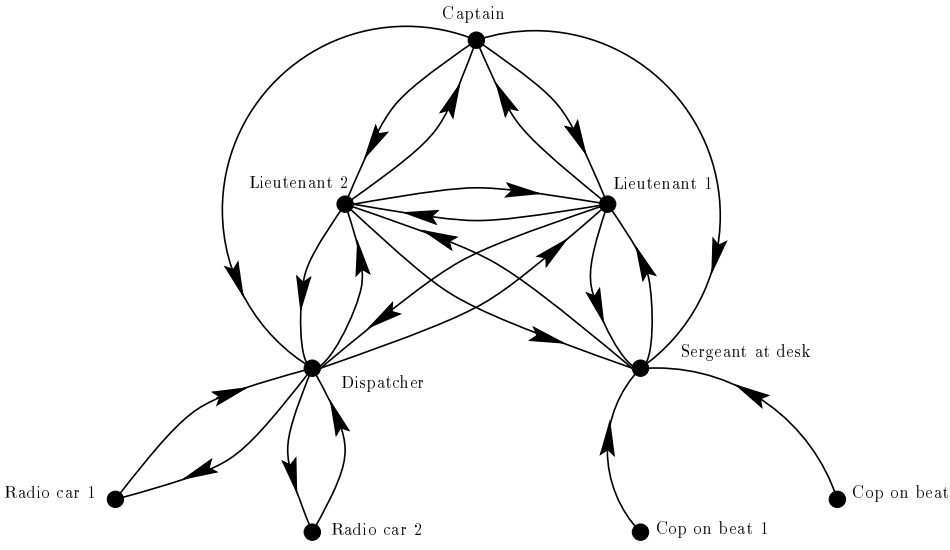


Figure 3.2: Communication network of part of a police force.

Example 3.3 Physical Networks Graphs often correspond to physical systems. For example, an electrical network can be thought of as a graph, with points as the electrical components, and two points joined by an undirected line if and only if there is a wire connecting them. Similarly, in telephone networks, we can think of the switching centers and the individual telephones as points, with two points joined by a line if there is a direct telephone line between them. Oil or gas pipelines of the kind studied in Example 1.3 can also be translated into graphs in this way. We usually seek the most efficient or least expensive design of a network meeting certain interconnection requirements, or seek a network that is least vulnerable to disruption. ■

Example 3.4 Reliability of Networks If a transportation, communication, electrical, or computer network is modeled by a graph, failures of components of the network can correspond to failures of points or lines. Good networks are designed with redundancy so that failures of some components do not lead to failure of the whole network (see Example 2.21). For instance, suppose that we have a network of six computers as shown in Figure 3.3 with direct links between them indicated by lines in the figure. Suppose there is a certain probability that each link will fail. What is the probability that every pair of computers in the system will be able to communicate (at least indirectly, possibly through other computers) if some of the links can fail? Could the network have been designed differently with the same number of links so as to increase this probability? ■

Example 3.5 Searching for Information on the Internet Because of the dramatic growth in the number of web sites and pages on the Internet, search engines

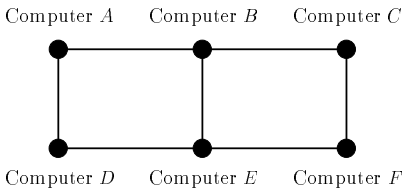
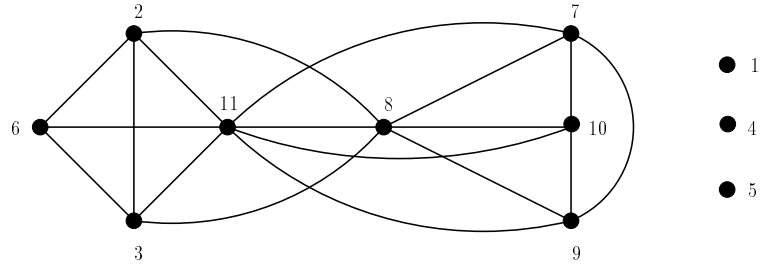


Figure 3.3: A network.

such as Google were introduced. The systems construct a database of web pages that has to be updated continually. The updating is done by robots that periodically traverse the web's hypertext structure. Let each web site be represented by a point and draw a line with an arrow in it from one web site to another if a robot can visit the second after the first. In what order should the web sites be visited so as to minimize the fraction of time that pages in the database are out of date? The answer depends in part on estimates of probabilities that given web pages are updated. Other applications of graph theory in searching on the Internet arise when we see the "most relevant" web site for a user query—this is the matching problem discussed in Chapter 12—or a set of web sites with at least a sufficiently high "relevance score." ■

Example 3.6 Analysis of Computer Programs Graphs have extensive applications in the analysis of computer programs. One approach is to subdivide a large program into subprograms, as an aid to understanding it, documenting it, or detecting structural errors. The subprograms considered are program blocks, or sequences of computer instructions with the property that whenever any instruction in the sequence is executed, all instructions in the sequence are executed. Let each program block be represented by a point, much as we represented cities in Figure 3.1. If it is possible to transfer control from the last instruction in program block x to the first instruction in program block y, draw a line with an arrow from x to y. The resulting diagram is called a *program digraph*. (A flowchart is a special case where each program block has one instruction.) Certain program blocks are designated as starting and stopping blocks. To detect errors in a program, we might use a compiler to ask if there are points (program blocks) from which it is never possible to reach a stopping point by following arrows. Or we might use the compiler to ask if there are points that can never be reached from a starting point. (If so, these correspond to subroutines which are never called.) The program digraph can also be used to estimate running time for the program. Graphs have many other uses in computer science, for instance in the design and analysis of computers and digital systems, in data structures, in the design of fault-tolerant systems, and in the fault diagnosis of digital systems. ■

Example 3.7 Competition Among Species Graphs also arise in the study of ecosystems. Consider a number of species that make up an ecosystem. Represent the species (or groups of species) as points, as in Figure 3.4, and draw an undirected line between species x and species y if and only if x and y compete. The resulting diagram is called a *competition graph* (or *niche overlap graph*). Questions one can



Key:

1. Canopy: leaves, fruit, flowers
2. Large ground animals: large mammals, birds
3. Insects
4. Trunk, fruit, flowers
5. Ground: roots, fallen fruit, leaves, trunks
6. Fungi
7. Middle-zone scansorial animals: mammals in both canopy and ground zones
8. Canopy animals: birds, fruit bats, other mammals
9. Upper air mammals: birds and bats, insectivorous
10. Middle-zone flying animals: birds, insectivorous bats
11. Small ground animals: birds, small mammals

Figure 3.4: A competition graph for species in a Malaysian rain forest. (From data of Harrison [1962], as adapted by Cohen [1978]. Graph from Roberts [1978].)

ask about competition graphs include questions about their structural properties (e.g., how "connected" are they, and what are their connected "pieces"); and about the "density" of lines (ratio of the number of lines present to the number of lines possible). ■

Example 3.8 Tournaments To give yet another application of graphs, consider a round-robin tournament[3] in tennis, where each player must play every other player exactly once, and no ties are allowed. One can represent the players as points and draw an arrow from player x to player y if x "beats" y, as in Figure 3.5. Similar tournaments arise in a surprisingly large number of places in the social, biological, and environmental sciences. Psychologists perform a pair comparison preference experiment on a set of alternatives by asking a subject, for each pair of alternatives, to state which he or she prefers. This defines a tournament, if we think of the alternatives as corresponding to the players and "prefers" as corresponding to "beats." Tournaments also arise in biology. In the farmyard, for every pair of chickens, it has been found that exactly one "dominates" the other. This "pecking order" among chickens again defines a tournament. (The same is true of other species of animals.) In studying tournaments, a basic problem is to decide on the "winner" and to rank the "players." Graph theory will help with this problem, too. ■

Example 3.9 Information Retrieval[4] In an information retrieval system on a computer, each document being indexed is labeled with a number of *index terms*

[3] This is not the more common elimination tournament.

[4] This example is due to Deo [1974].

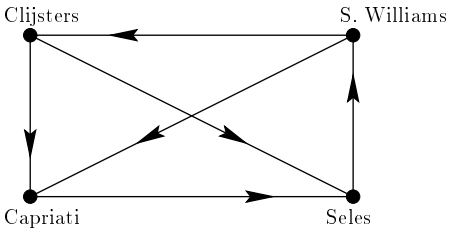


Figure 3.5: A tournament.

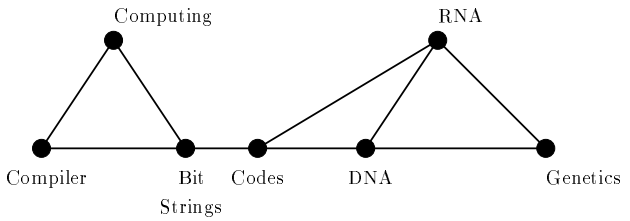


Figure 3.6: Part of a similarity graph.

or *descriptors*. Let the index terms be drawn as points and join two points with a line if the corresponding index terms are closely related, as in Figure 3.6. The diagram that results is called a *similarity graph*. It can be used to produce a classification of documents and to help in information retrieval: One provides some index terms and the information retrieval system produces a list of related terms and the corresponding documents. In sophisticated information retrieval applications in large databases such as the World Wide Web (as in Example 3.5), we might measure the relevance of a document or web page for a particular query term and seek to find a web page that has maximum relevance. ■

3.1.2 Definition of Digraph and Graph

These examples all give rise to directed or undirected graphs. To be precise, let us define a *directed graph* or *digraph* D as a pair (V, A), where V is a nonempty set and A is a set of ordered pairs of elements of V. V will be called the set of *vertices* and A the set of *arcs*. (Some authors use the terms *node, point*, and so on, in place of *vertex*, and the terms *arrow, directed line, directed edge*, or *directed link* in place of *arc*.) If more than one digraph is being considered, we will use the notation $V(D)$ and $A(D)$ for the vertex set and the arc set of D, respectively. Usually, digraphs are represented by simple diagrams such as those of Figure 3.7. Here, the vertices are represented by points and there is a directed line (or arrow, not necessarily straight) heading from u to v if and only if (u, v) is in A. For example, in the digraph D_1 of Figure 3.7, V is the set $\{u, v, w, x\}$ and A is the set

$$\{(u, v), (u, w), (v, w), (w, x), (x, u)\}.$$

If there is an arc from vertex u to vertex v, we shall say that u is *adjacent* to v.

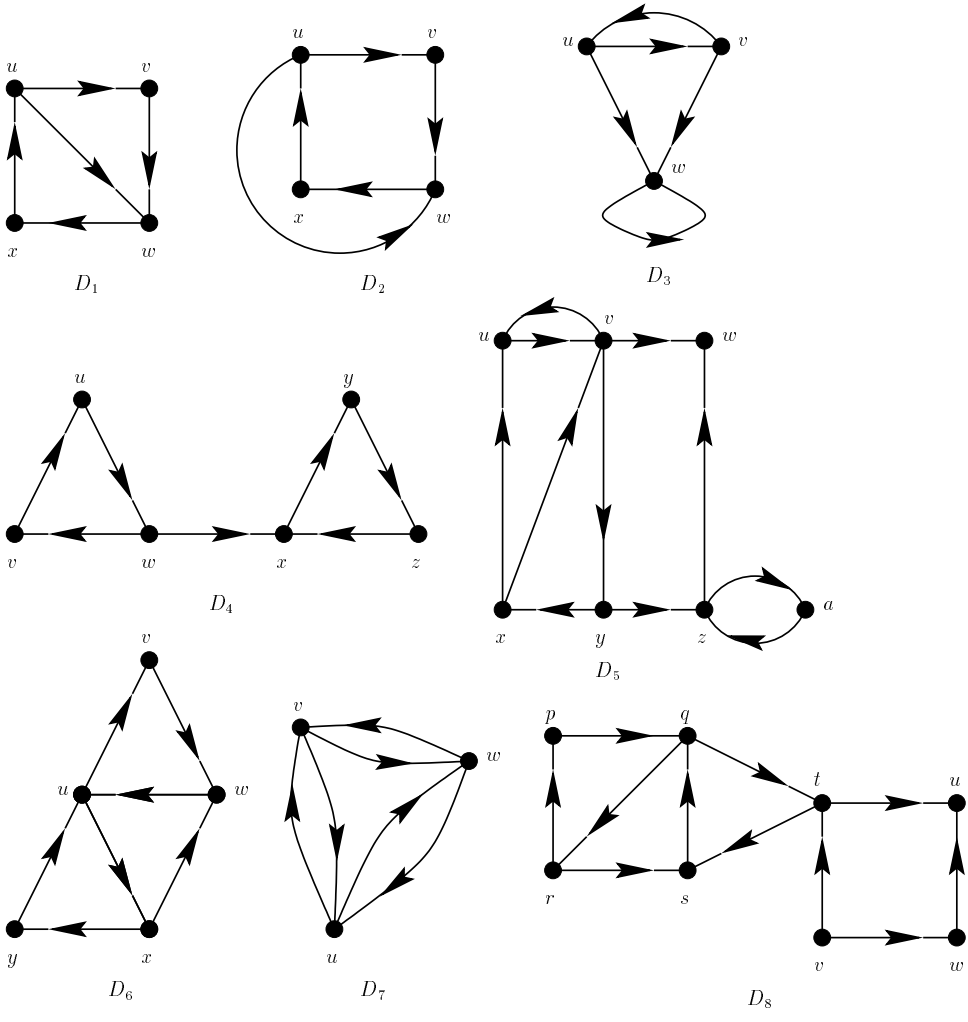


Figure 3.7: Digraphs.

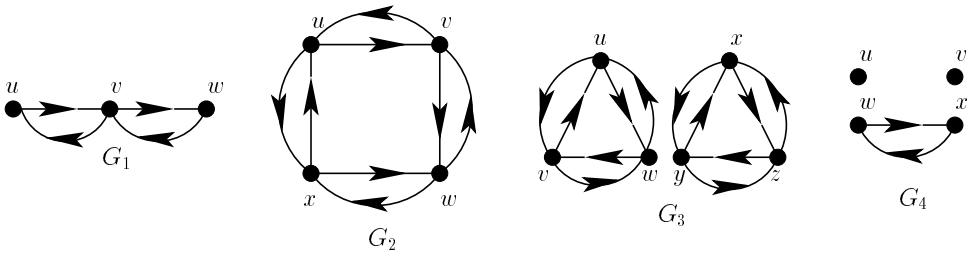


Figure 3.8: Graphs.

Thus, in Figure 3.7, in digraph D_1, u is adjacent to v and w, w is adjacent to x, and so on.

Note: The reader should notice that the particular placement of the vertices in a diagram of a digraph is unimportant. The distances between the vertices have no significance, the nature of the lines joining them is unimportant, and so on. Moreover, whether or not two arcs cross is also unimportant; the crossing point is not a vertex of the digraph. All the information in a diagram of a digraph is included in the observation of whether or not a given pair of vertices is joined by a directed line or arc and in which direction the arc goes.[5] Thus, digraphs D_1 and D_2 of Figure 3.7 are the same digraph, only drawn differently. In the next subsection, we shall say that D_1 and D_2 are isomorphic.

In a digraph, it is perfectly possible to have arcs in both directions, from u to v and from v to u, as shown in digraph D_3 of Figure 3.7, for example. It is also possible to have an arc from a vertex to itself, as is shown with vertex w in digraph D_3. Such an arc is called a *loop*. It is not possible, however, to have more than one arc from u to v. Often in the theory and applications of digraphs, such multiple arcs are useful—this is true in the study of chemical bonding, for example—and then one studies *multigraphs* or better, *multidigraphs*, rather than digraphs.

Very often, there is an arc from u to v whenever there is an arc from v to u. In this case we say that the digraph (V, A) is a *graph*. Figure 3.8 shows several graphs. In the drawing of a graph, it is convenient to disregard the arrows and to replace a pair of arcs between vertices u and v by a single nondirected line joining u and v. (In the case of a directed loop, it is replaced by an undirected one.) We shall call such a line an *edge* of the graph and think of it as an unordered pair of vertices $\{u, v\}$. (The vertices u and v do not have to be distinct.) If there is an edge $\{u, v\}$ in the graph, we call u and v *neighbors*. The graph drawings obtained from those of Figure 3.8 in this way are shown in Figure 3.9. Thus, a graph G may be defined as a pair (V, E), where V is a set of vertices and E is a set of unordered pairs of elements from V, the edges. If more than one graph is being considered, we will use the notation $V(G)$ and $E(G)$ for the vertex set and the edge set of G, respectively.

At this point, let us make explicit several assumptions about our digraphs and graphs. Many graph theorists make explicit the following assumption: There are no

[5] This isn't quite true. Some properties of digraphs may be associated with the actual placement of vertices and arcs in a diagram, as we shall see later in the text.

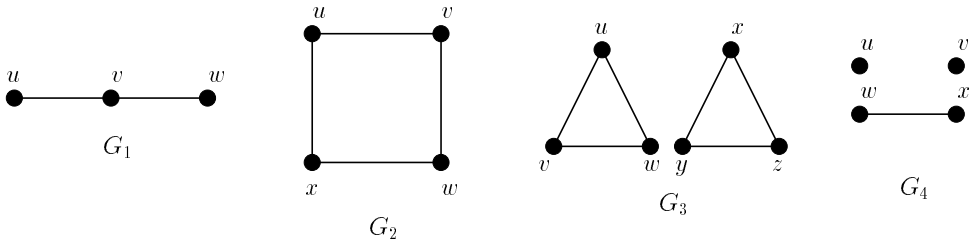


Figure 3.9: The graphs of Figure 3.8 with arcs replaced by edges.

multiple arcs or edges, that is, no more than one arc or edge from vertex u to vertex v. For us, this assumption is contained in our definition of a digraph or graph. We shall assume, at least at first, that digraphs and graphs have no loops. (Almost everything we say for loopless digraphs and graphs will be true for digraphs and graphs with loops.) We shall also limit ourselves to digraphs or graphs with finite vertex sets. Let us summarize these assumptions as follows:

Assumptions: Unless otherwise specified, all digraphs and graphs referred to in this book have finite vertex sets, have no loops, and are not allowed to have multiple arcs or edges.

Let the *degree* of vertex u of graph G, $\deg(u)$ or $\deg_G(u)$, count the number of neighbors of u. Note that if we sum up the degrees of all vertices of G, we count each edge twice, once for each vertex on it. Thus, we have the following theorem.

Theorem 3.1 If G is any graph of e edges,

$$\sum_{u \in V(G)} \deg(u) = 2e.$$

3.1.3 Labeled Digraphs and the Isomorphism Problem[6]

A *labeled digraph* or *graph* of n vertices is a digraph or graph which has the integers $1, 2, \ldots, n$ assigned, one to each vertex. Two labeled digraphs or graphs can be, for all practical purposes, the same. For instance, Figure 3.10 shows an unlabeled graph G and three labelings of the vertices of G. The first two labelings are considered the same in the following sense: Their edge sets are the same. However, the first and third labelings are different, because, for instance, the first has an edge $\{3, 4\}$ whereas the third does not.

As a simple exercise in counting, let us ask how many distinct labeled graphs there are which have n vertices, for $n \geq 2$. The answer is most easily obtained if we observe that a labeled graph with n vertices can have at most $C(n, 2)$ edges, for $C(n, 2)$ is the number of unordered pairs of vertices from the n vertices. Let us suppose that the graph has e edges. Then we must choose e edges out of these

[6] This subsection is optional. It is placed here as a good application of the counting techniques of Chapter 2, and the concepts are used occasionally. The material can be returned to when it is needed.

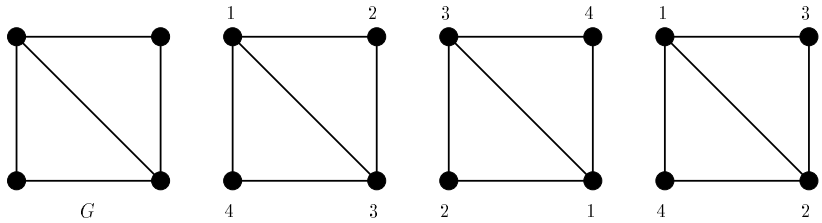


Figure 3.10: A graph G and three labelings of its vertices.

$C(n,2)$ possible edges. Hence, we see that the number $L(n,e)$ of labeled graphs with n vertices and e edges is given by

$$L(n,e) = C\left(C(n,2),e\right) = \binom{\binom{n}{2}}{e}. \tag{3.1}$$

Thus, by the sum rule and Equation (3.1), the number $L(n)$ of labeled graphs of n vertices is given by

$$L(n) = \sum_{e=0}^{C(n,2)} L(n,e) = \sum_{e=0}^{C(n,2)} C\left(C(n,2),e\right). \tag{3.2}$$

For instance, if $n = 3$, then $L(3,0) = 1, L(3,1) = 3, L(3,2) = 3, L(3,3) = 1$, and $L(3) = 8$. Figure 3.11 shows the eight labeled graphs of 3 vertices.

Note that Equation (3.2) implies that the number of distinct labeled graphs grows very fast as n grows. To see that, note that if $r = C(n,2)$, then using Theorem 2.8,

$$L(n) = \sum_{e=0}^{r} C(r,e) = 2^r,$$

so

$$L(n) = 2^{n(n-1)/2}. \tag{3.3}$$

The number given by (3.3) grows exponentially fast as n grows. There are just too many graphs to answer most graph-theoretical questions by enumerating graphs.

Two labeled digraphs are considered *the same* if their arc sets are the same. How many different labeled digraphs are there with n vertices? Since any arc is an ordered pair of vertices and loops are not allowed, there are, by the product rule, $n(n-1)$ possible arcs. The number $M(n,a)$ of labeled digraphs with n vertices and a arcs is given by

$$M(n,a) = C\left(n(n-1),a\right) = \binom{n(n-1)}{a}. \tag{3.4}$$

Again by the sum rule, the number $M(n)$ of labeled digraphs with n vertices is thus given by

$$M(n) = \sum_{a=0}^{n(n-1)} M(n,a) = \sum_{a=0}^{n(n-1)} C\left(n(n-1),a\right). \tag{3.5}$$

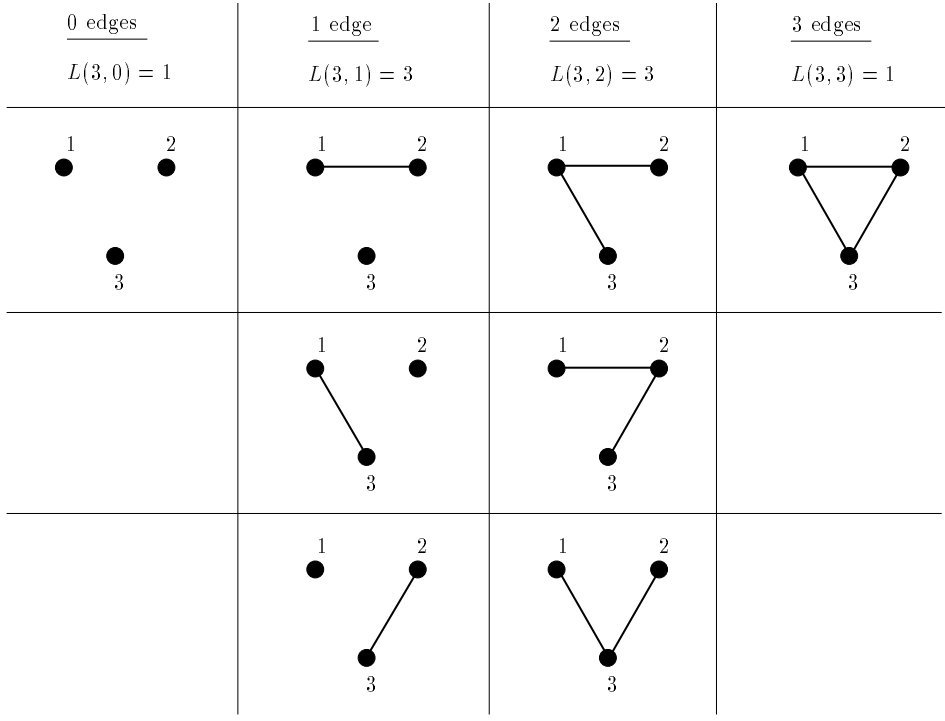


Figure 3.11: The eight different labeled graphs of 3 vertices.

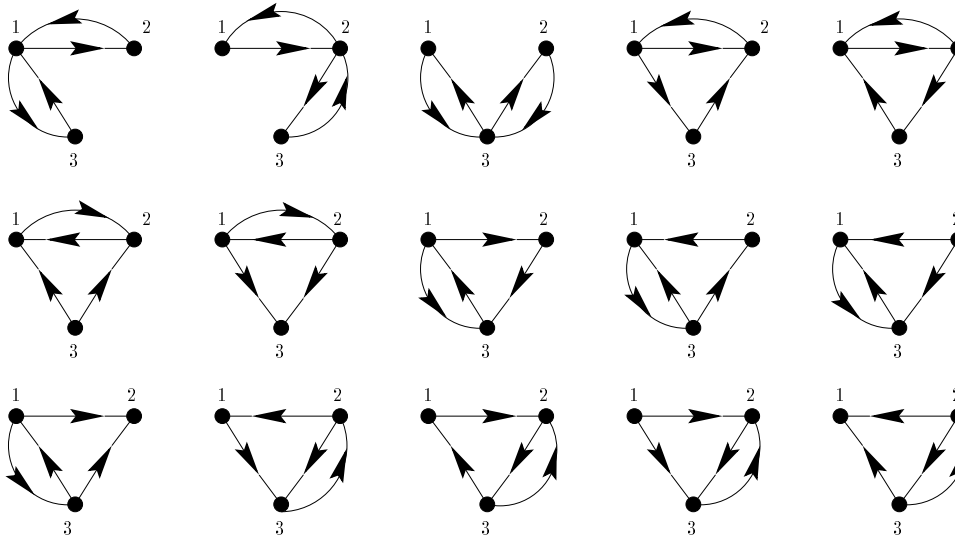


Figure 3.12: The 15 labeled digraphs of 3 vertices and 4 arcs.

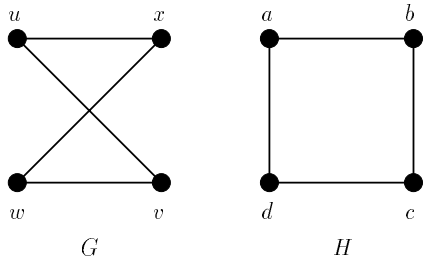


Figure 3.13: Graphs G and H are isomorphic.

For instance, if $n = 3$, then $M(3,0) = 1, M(3,1) = 6, M(3,2) = 15, M(3,3) = 20, M(3,4) = 15, M(3,5) = 6, M(3,6) = 1$, and $M(3) = 64$. Figure 3.12 shows the 15 labeled digraphs with 3 vertices and 4 arcs.

Using an argument similar to the one given above for $L(n)$, it can be shown that

$$M(n) = 2^{n(n-1)}. \tag{3.6}$$

The proof of Equation (3.6) is left to the exercises.

Two unlabeled graphs (digraphs) G and H, each having n vertices, are considered *the same* if the vertices of both can be labeled with the integers $1, 2, \ldots, n$ so that the edge sets (arc sets) consist of the same unordered (ordered) pairs, that is, if the two graphs (digraphs) can each be given a labeling that shows them to be the same as labeled graphs (digraphs). If this can be done, we say that G and H are *isomorphic*.

For instance, the graphs G and H of Figure 3.13 are isomorphic, as is shown by labeling vertex u as 1, v as 2, w as 3, x as 4, a as 1, b as 2, c as 3, and d as 4. The digraphs D_1 and D_2 of Figure 3.7 are also isomorphic; labeling u as 1, v as 2, w as 3, and x as 4 in both digraphs demonstrates this.

Although it is easy to tell whether or not two labeled graphs or digraphs are the same, the problem of determining whether or not two unlabeled graphs or digraphs are the same, that is, isomorphic, is a very difficult one indeed. This is called the *isomorphism problem*, and it is one of the most important problems in graph theory. The most naive algorithm for determining if two graphs G and H of n vertices are isomorphic would fix a labeling of G using the integers $1, 2, \ldots, n$, and then try out all possible labelings of H using these integers. Thus, this algorithm has computational complexity $f(n) = n!$, and we have already seen in Sections 2.3 and 2.4 that even for moderate n, such as $n = 25$, considering this many cases is infeasible. Although better algorithms are known, the best algorithms known for solving the isomorphism problem have computational complexity which is exponential in the size of the problem; that is, they require an unfeasibly large number of steps to compute if the number of vertices gets moderately large. See Reingold, Nievergelt, and Deo [1977, Sec. 8.5], Deo [1974, Sec. 11-7], and Kreher and Stinson [1998] for some discussion. Polynomial algorithms have been found when the graphs in question have certain properties. See, for example, Luks [1982], Bodlaender [1990], and Ponomarenko [1984, 1992].

EXERCISES FOR SECTION 3.1

1. In the digraph of Figure 3.1, identify:

   (a) The set of vertices (b) The set of arcs

2. Repeat Exercise 1 for the digraph of Figure 3.5.
3. Repeat Exercise 1 for the digraph D_4 of Figure 3.7.
4. In each of the graphs of Figure 3.9, identify:

   (a) The set of vertices (b) The set of edges

5. In digraph D_5 of Figure 3.7, find a vertex adjacent to vertex y.
6. In the graph of Figure 3.1, find all neighbors of the vertex New York.
7. Draw a transportation network with the cities New York, Paris, Vienna, Washington, DC, and Algiers as vertices, and an edge joining two cities if it is possible to travel between them by road.
8. Draw a communication network for a team fighting a forest fire.
9. Draw a digraph representing the following football tournament. The teams are Michigan, Ohio State, and Northwestern. Michigan beats Ohio State, Ohio State beats Northwestern, and Northwestern beats Michigan.
10. Draw a program digraph for a computer program of your choice.
11. Draw a similarity graph involving some terms related to ecology.
12. A *food web* is a digraph whose vertices are some species in an ecosystem and which has an arc from x to y if x preys on y. Draw a food web for the set of species {deer, mountain lion, eagle, mouse, fox, grass}.
13. Sometimes, we say that two species in an ecosystem *compete* if they have a common prey. We can build a competition graph (Example 3.7) from a food web in this way. Find the competition graph for the food web of Exercise 12.
14. Generalizing Exercise 13, we can define the competition graph G corresponding to any digraph D by letting $V(G) = V(D)$ and letting $\{x, y\} \in E(G)$ if and only if there is $a \in V(D)$ so that (x, a) and (y, a) are in $A(D)$. Find the competition graph corresponding to each of the digraphs of Figure 3.7.
15. Show that in a graph G with n vertices and e edges, there is a vertex of degree at least $2e/n$.
16. Can the number of vertices of odd degree in a graph be odd? Why?
17. Figure 3.14 shows a graph and three labelings of its vertices.

    (a) Are the first two labelings the same? Why?

    (b) Are the first and the third? Why?

18. Figure 3.15 shows a digraph and three labelings of its vertices.

    (a) Are the first two labelings the same? Why?

    (b) Are the first and the third? Why?

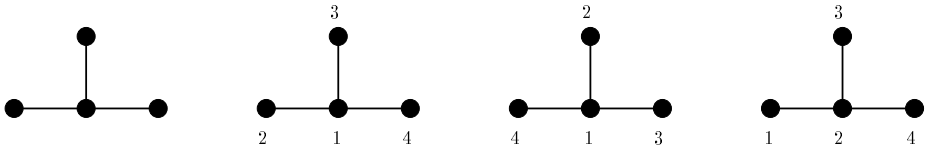


Figure 3.14: A graph and three labelings of its vertices.

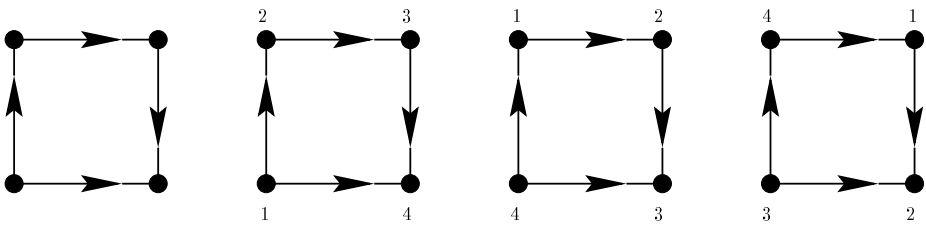


Figure 3.15: A digraph and three labelings of its vertices.

19. Find the number of labeled graphs with 4 vertices and 2 edges by using Equation (3.1). Check by drawing all such graphs.
20. How many different labeled graphs are there with 4 vertices and an even number of edges?
21. Find the number of labeled digraphs with 4 vertices and 2 arcs by using Equation (3.4). Check by drawing all such digraphs.
22. Prove Equation (3.6).
23. Are the graphs of Figure 3.16(a) isomorphic? Why?
24. Are the graphs of Figure 3.16(b) isomorphic? Why?
25. Are the digraphs of Figure 3.17(a) isomorphic? Why?
26. Are the digraphs of Figure 3.17(b) isomorphic? Why?
27. Are the digraphs of Figure 3.17(c) isomorphic? Why?
28. An *orientation* of a graph arises by replacing each edge $\{x, y\}$ by one of the arcs (x, y) or (y, x). For instance, the digraph of Figure 3.15 is an orientation of graph H of Figure 3.13. For each of the graphs of Figure 3.16(a), find all nonisomorphic orientations.

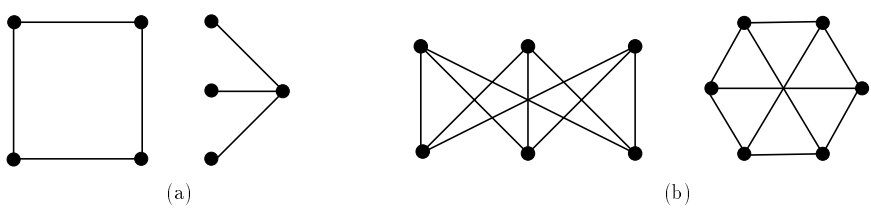


Figure 3.16: Graphs for Exercises 23, 24, 28, Section 3.1.

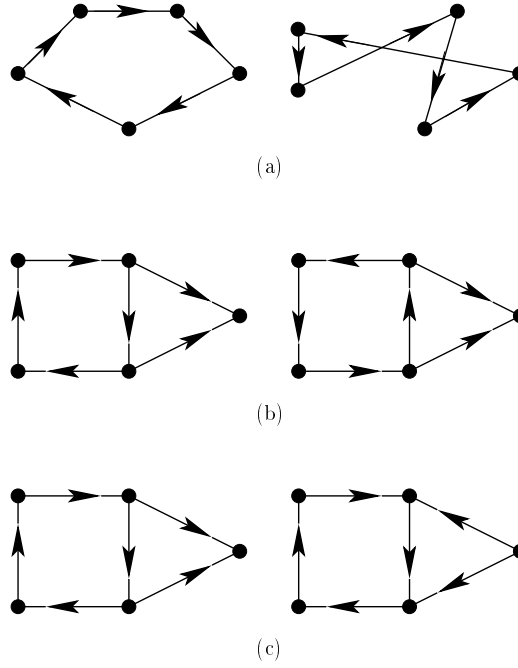


Figure 3.17: Digraphs for Exercises 25, 26, 27, Section 3.1.

29. Suppose that G and H are two graphs with the same number of vertices and the same number of edges. Suppose that α_k is the number of vertices in G with exactly k neighbors, and β_k is the number of vertices in H with exactly k neighbors. Suppose that $\alpha_k = \beta_k$ for all k. Are G and H necessarily isomorphic? Why?

30. Repeat Exercise 29 if $\alpha_2 = \beta_2 = |V(G)| = |V(H)|$ and $\alpha_k = \beta_k = 0$ for $k \neq 2$.

3.2 CONNECTEDNESS

3.2.1 Reaching in Digraphs

In a communication network, a natural question to ask is: Can one person initiate a message to another person? In a transportation network, an analogous question is: Can a car move from location u to location v? In a program digraph, we are interested in determining if from every vertex it is possible to follow arcs and ultimately hit a stopping vertex. All of these questions have in common the following idea of reachability in a digraph $D = (V, A)$: Can we reach vertex v by starting at vertex u and following the arcs of D?

To make this concept precise, let us introduce some definitions. A *path* in D is a sequence

$$u_1, a_1, u_2, a_2, \ldots, u_t, a_t, u_{t+1}, \tag{3.7}$$

Table 3.1: Reaching and Joining

Digraph D	Graph G
$u_1, a_1, u_2, a_2, \ldots, u_t, a_t, u_{t+1}$	$u_1, e_1, u_2, e_2, \ldots, u_t, e_t, u_{t+1}$
Reaching	Joining
Path:	*Chain*:
a_i is (u_i, u_{i+1})	e_i is $\{u_i, u_{i+1}\}$
Simple path:	*Simple chain*:
Path and	Chain and
u_i distinct	u_i distinct
Closed path:	*Closed chain*:
Path and	Chain and
$u_{t+1} = u_1$	$u_{t+1} = u_1$
Cycle (simple closed path):	*Circuit (simple closed chain)*:
Path and	Chain and
$u_{t+1} = u_1$ and	$u_{t+1} = u_1$ and
u_i distinct, $i \leq t$ and	u_i distinct, $i \leq t$ and
(a_i distinct)[a]	e_i distinct

[a] This follows from u_i distinct, $i \leq t$.

where $t \geq 0$, each u_i is in V, that is, is a vertex, and each a_i is in A, that is, is an arc, and a_i is the arc (u_i, u_{i+1}). That is, arc a_i goes from u_i to u_{i+1}. Since t might be 0, u_1 alone is a path, a path from u_1 to u_1. The path (3.7) is called a *simple path* if we never use the same vertex more than once.[7] For example, in digraph D_5 of Figure 3.7, $u, (u, v), v, (v, w), w$ is a simple path and $u, (u, v), v, (v, y), y, (y, x), x, (x, v), v, (v, w), w$ is a path that is not a simple path since it uses vertex v twice. Naming the arcs is superfluous when referring to a path, so we simply speak of (3.7) as the path $u_1, u_2, \ldots, u_t, u_{t+1}$.

A path (3.7) is called *closed* if $u_{t+1} = u_1$. In a closed path, we end at the starting point. If the path (3.7) is closed and the vertices $u_1, u_2, \ldots, u_t$ are distinct, then (3.7) is called a *cycle* (a simple closed path[8]). (The reader should note that if the vertices $u_i, i \leq t$, are distinct, the arcs a_i must also be distinct.)

To give some examples, the path u, v, w, x, u in digraph D_1 of Figure 3.7 is a cycle, as is the path u, v, w, x, y, u in digraph D_6. But the closed path u, v, y, x, v, y, x, u of D_5 is not a cycle, since there are repeated vertices. In general, in counting or listing cycles of a digraph, we shall not distinguish two cycles that use the same vertices and arcs in the same order but start at a different vertex. Thus, in digraph D_6 of Figure 3.7, the cycle w, x, y, u, v, w is considered the same as the cycle

[7] One of the difficulties in learning graph theory is the large number of terms that have to be mastered early. To help the reader overcome this difficulty, we have included the terms *path, simple path*, and so on, in succinct form in Table 3.1

[8] A simple closed path is, strictly speaking, not a simple path.

u, v, w, x, y, u. The *length* of a path, simple path, cycle, and so on is the number of arcs in it. Thus, the path (3.7) has length t. In digraph D_5 of Figure 3.7, u, v, y, x, v is a path of length 4, u, v, y, z is a simple path of length 3, and u, v, y, x, u is a cycle of length 4. We say that v is *reachable* from u if there is a path from u to v. Thus, in D_5, z is reachable from u. However, u is not reachable from z.

A digraph $D = (V, A)$ is called *complete symmetric* if for every $u \neq v$ in V, the ordered pair (u, v) is an arc of D. For instance, the digraph D_7 of Figure 3.7 is complete symmetric. A complete symmetric digraph on n vertices has $n(n-1)$ arcs. Let us ask how many simple paths it has of a given length k, if $k \leq n$, the number of vertices. The answer is that to find such a path, we choose any $k+1$ vertices, and then order them. Thus, we have $C(n, k+1) \cdot (k+1)! = P(n, k+1)$ such paths.

3.2.2 Joining in Graphs

Suppose that $G = (V, E)$ is a graph. Terminology analogous to that for digraphs can be introduced. A *chain* in G is a sequence

$$u_1, e_1, u_2, e_2, \ldots, u_t, e_t, u_{t+1}, \tag{3.8}$$

where $t \geq 0$, each u_i is a vertex, and each e_i is the edge $\{u_i, u_{i+1}\}$. A chain is called *simple* if all the u_i are distinct and *closed* if $u_{t+1} = u_1$. A closed chain (3.8) in which $u_1, u_2, \ldots, u_t$ are distinct and $e_1, e_2, \ldots, e_t$ are distinct is called a *circuit* (a simple closed chain[9]). The *length* of a chain, circuit, and so on of form (3.8) is the number of edges in it. We say that u and v are *joined* if there is a chain from u to v.

To give some examples, in the graph of Figure 3.18,

$$r, \{r,t\}, t, \{t,w\}, w, \{w,u\}, u, \{u,t\}, t, \{t,r\}, r, \{r,s\}, s, \{s,u\}, u, \{u,t\}, t, \{t,w\}, w$$

is a chain. This chain can be written without reference to the edges as r, t, w, u, t, r, s, u, t, w. A simple chain is given by r, t, u, w, x. A circuit is given by r, t, u, s, r. Finally, $p, \{p,r\}, r, \{r,p\}, p$ is not considered a circuit, since $e_1 = e_t$. (The edges are unordered, so $\{p,r\} = \{r,p\}$.) Note that without the restriction that the edges be distinct, this would be a circuit. In the analogous case of digraphs, we did not have to assume arcs distinct for a cycle, since that follows from vertices distinct.

The *complete graph* on n vertices, denoted K_n, is defined to be the graph in which every pair of vertices is joined by an edge. In K_n, there are $C(n, 2)$ edges. The number of simple chains of length k in K_n is given by $P(n, k+1)$. (Why?)

3.2.3 Strongly Connected Digraphs and Connected Graphs

One reason graph theory is so useful is that its geometric point of view allows us to define various structural concepts. One of these concepts is connectedness. A

[9] We shall see below why the restriction that the edges be distinct is added. Also, it should be noted that a simple closed chain is, strictly speaking, not simple.

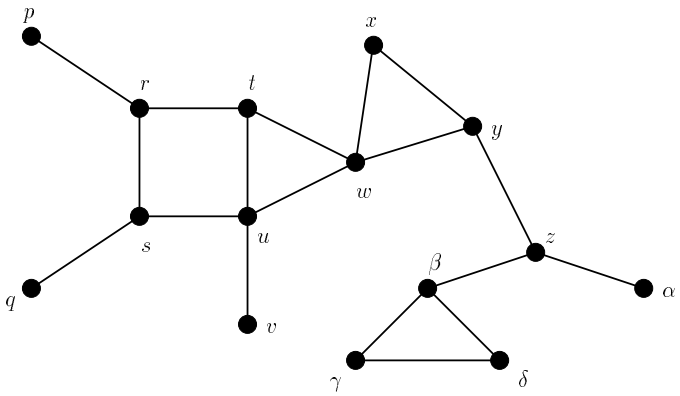


Figure 3.18: A graph.

digraph is said to be *strongly connected* if for every pair of vertices u and v, v is reachable from u and u is reachable from v. Thus, digraph D_6 of Figure 3.7 is strongly connected, but digraph D_5 is not. If a communication network is strongly connected, every person can *initiate* a communication to every other person. If a transportation network is not strongly connected, there are two locations u and v so that one cannot go from the first to the second, or vice versa. In Section 11.2 we study how to obtain strongly connected transportation networks. A program digraph is never strongly connected, for there are no arcs leading out of a stopping vertex. A tournament can be strongly connected (see Figure 3.5); however, it does not have to be. In a strongly connected tournament, it is hard to rank the players, since we get situations where u_1 beats u_2, who beats $u_3, \ldots$, who beats u_t, who beats u_1.

We say that a graph is *connected* if between every pair of vertices u and v there is a chain. This notion of connectedness coincides with the one used in topology: The graph has one "piece." In Figure 3.9, graphs G_1 and G_2 are connected while G_3 and G_4 are not. Physical networks (electrical, telephone, pipeline) are usually connected. Indeed, we try to build them so that an outage at one edge does not result in a disconnected graph.

Algorithms to test whether or not a graph is connected have been designed in a variety of ways. The fastest are very good. They have computational complexity that is linear in the number of vertices n plus the number of edges e. In the language of Section 2.18, they take on the order of $n + e$ steps. [They are $O(n + e)$.] Since a graph has at most $\binom{n}{2}$ edges, we have

$$e \leq \binom{n}{2} = \frac{n(n-1)}{2} \leq n^2.$$

Thus, $n + e \leq n + n^2$, so these algorithms take a number of steps on the order of $n + n^2$, which is a polynomial in n. [In the notation of Section 2.18, they are $O(n^2)$.

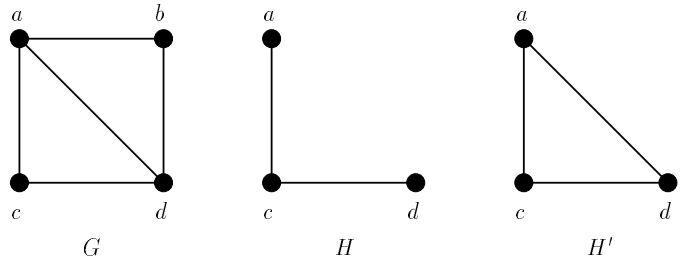


Figure 3.19: H is a subgraph of G and H' is a generated subgraph.

We say they are quadratic in n.] Similar results and algorithms apply to digraphs and the concept of strong connectedness. We discuss some of these algorithms in Section 11.1. See also Aho, Hopcroft, and Ullman [1974], Even [1979], Frank [1995], Gibbons [1985], Golumbic [1980], Reingold, Nievergelt, and Deo [1977], or West [2001].

3.2.4 Subgraphs

In what follows it will sometimes be useful to look at parts of a graph. Formally, suppose that $G = (V, E)$ is a graph. A *subgraph* $H = (W, F)$ is a graph such that W is a subset of V and F is a set of unordered pairs of vertices of W which is a subset of E. Thus, to define a subgraph of G, we choose from G some vertices and some edges joining the chosen vertices. For instance, in Figure 3.19, graphs H and H' are both subgraphs of graph G. In H', the edge set consists of all edges of G joining vertices of $W = \{a, c, d\}$. In such a case, we say that H' is the *subgraph generated* or *induced* by the vertices of W.

Similar concepts apply to digraphs. If $D = (V, A)$ is a digraph, then a *subgraph* (or *subdigraph*) $J = (W, B)$ of D is a digraph with W a subset of V and B a set of ordered pairs of vertices of W which is a subset of A. J is a *generated subgraph* if B is all arcs of D that join vertices in W. For instance, in Figure 3.20, digraph J is a subgraph of digraph D and digraph J' is the subgraph generated by the vertices a, c, and d.

As a simple application of these ideas, let us ask how many subgraphs of k vertices there are if we start with the complete symmetric digraph on n vertices. To find such a subgraph, we first choose the k vertices; this can be done in $C(n, k)$ ways. These vertices are joined by $k(k-1)$ arcs in D. We may choose any subset of this set of arcs for the subgraph; that is, we may choose arcs for the subgraph in $2^{k(k-1)}$ ways. Thus, by the product rule, there are

$$C(n, k) \cdot 2^{k(k-1)}$$

subgraphs of k vertices.

Example 3.10 Reliability of Systems (Example 2.21 Revisited) In Example 2.21 we studied systems consisting of components that might or might not work,

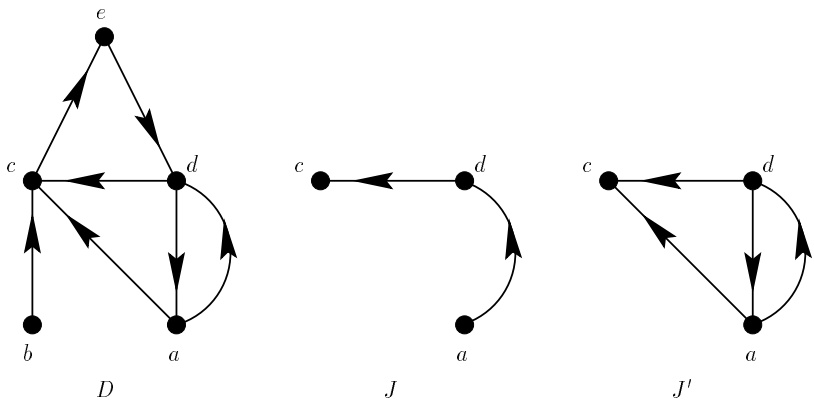


Figure 3.20: J is a subgraph of D and J' is the subgraph generated by vertices a, c, and d.

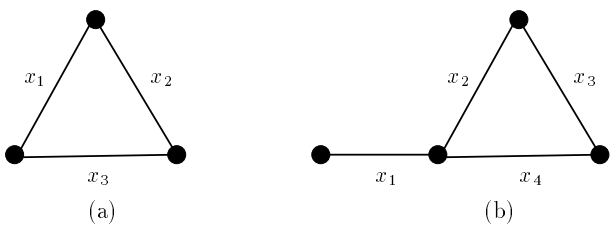


Figure 3.21: Two systems.

and introduced rules for determining, given which components are working, whether or not the system works. In studying reliability of such systems, we commonly represent a system by a graph G and let each edge correspond to a component. Then in many situations (see Example 3.4) it makes sense to say that the system works if and only if every pair of vertices is joined by a chain of working components, i.e., if and only if the subgraph H consisting of all vertices of G and the working edges of G is connected. Consider, for example, the graph G of Figure 3.21(a). There are three components, labeled x_1, x_2, and x_3. Clearly, the system works if and only if at least two of these components work. Similarly, if G is as in Figure 3.21(b), the system works if and only if component x_1 works and at least two of the remaining three components work. ■

3.2.5 Connected Components

Suppose that $G = (V, E)$ is a graph. A *connected component* or a *component* of G is a connected, generated subgraph H of G which is maximal in the sense that no larger connected generated subgraph K of G contains all the vertices of H. For example, in the graph G of Figure 3.22, the subgraph generated by the vertices a, b,

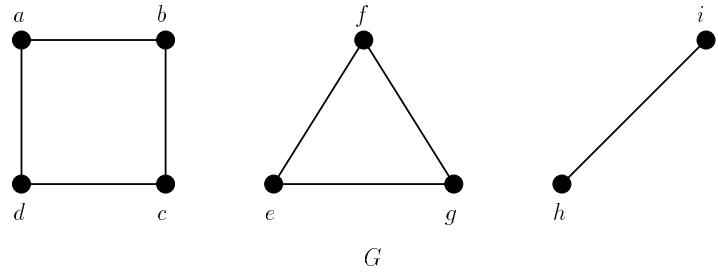


Figure 3.22: There are three components, the subgraphs generated by vertices a, b, c, and d, by vertices e, f, and g, and by vertices h and i.

and c is connected, but it is not a component since the subgraph generated by the vertices a, b, c, and d is a connected generated subgraph containing all the vertices of the first subgraph. This second subgraph is a component. There are three components in all, the other two being the subgraphs generated by vertices e, f, and g and by h and i. These components correspond to the "pieces" of the graph. Clearly, connected graphs have exactly one component. In the information retrieval situation of Example 3.9, to give one simple application, components produce a natural classification of documents. In the competition graph of Figure 3.4, there are four components (three consisting of one vertex each). Real-world competition graphs tend to have at least two components. Concepts for digraphs analogous to connected components are studied in the exercises.

EXERCISES FOR SECTION 3.2

1. For the digraph D_8 of Figure 3.7:
   (a) Find a path that is not a simple path.
   (b) Find a closed path.
   (c) Find a simple path of length 4.
   (d) Determine if $q, (q,t), t, (t,s), s, (s,q), q$ is a cycle.
   (e) Find a cycle of length 3 containing vertex p.
2. For the graph of Figure 3.18:
   (a) Find a closed chain that is not a circuit.
   (b) Find the longest circuit.
   (c) Find a chain different from the one in the text which is not simple.
   (d) Find a closed chain of length 6.
3. Give an example of a digraph and a path in that digraph which is not a simple path but has no repeated arcs.
4. Give an example of a graph in which the shortest circuit has length 5 and the longest circuit has length 8.

5. For each digraph of Figure 3.7, determine if it is strongly connected.
6. Which of the graphs of Figure 3.23 are connected?
7. For each digraph of Figure 3.24:
   (a) Find a subgraph that is not a generated subgraph.
   (b) Find the subgraph generated by vertices 5, 8, and 9.
   (c) Find a strongly connected generated subgraph.
8. For the graph of Figure 3.25:
   (a) Find a subgraph that is not a generated subgraph.
   (b) Find a generated subgraph that is connected but not a connected component.
   (c) Find all connected components.
9. A digraph is *unilaterally connected* if for every pair of vertices u and v, either v is reachable from u or u is reachable from v, but not necessarily both.
   (a) Give an example of a digraph that is unilaterally connected but not strongly connected.
   (b) For each digraph of Figure 3.7, determine if it is unilaterally connected.
10. A digraph is *weakly connected* if when all directions on arcs are disregarded, the resulting graph (or possibly multigraph) is connected.
    (a) Give an example of a digraph that is weakly connected but not unilaterally connected.
    (b) Give an example of a digraph that is not weakly connected.
    (c) For each digraph of Figure 3.7, determine if it is weakly connected.
11. Prove that if v is reachable from u in digraph D, there is a simple path from u to v in D.
12. Suppose that a system defined by a graph G works if and only if the vertices of G and the working edges form a connected subgraph of G. Under what circumstances does each of the systems given in Figure 3.26 work?
13. In a digraph D, a *strong component* is a strongly connected, generated subgraph which is maximal in the sense that it is not contained in any larger, strongly connected, generated subgraph. For example, in digraph D_5 of Figure 3.7, the subgraph generated by vertices x, y, v is strongly connected, but not a strong component since the subgraph generated by x, y, v, u is also strongly connected. The latter is a strong component. So is the subgraph generated by the single vertex w and the subgraph generated by the vertices z and a. There are no other strong components. (For applications of strong components to communication networks, to energy demand, and to Markov models of probabilistic phenomena, see Roberts [1976].) Find all strong components of each digraph of Figure 3.7.
14. Find all strong components for the police force of Figure 3.2.

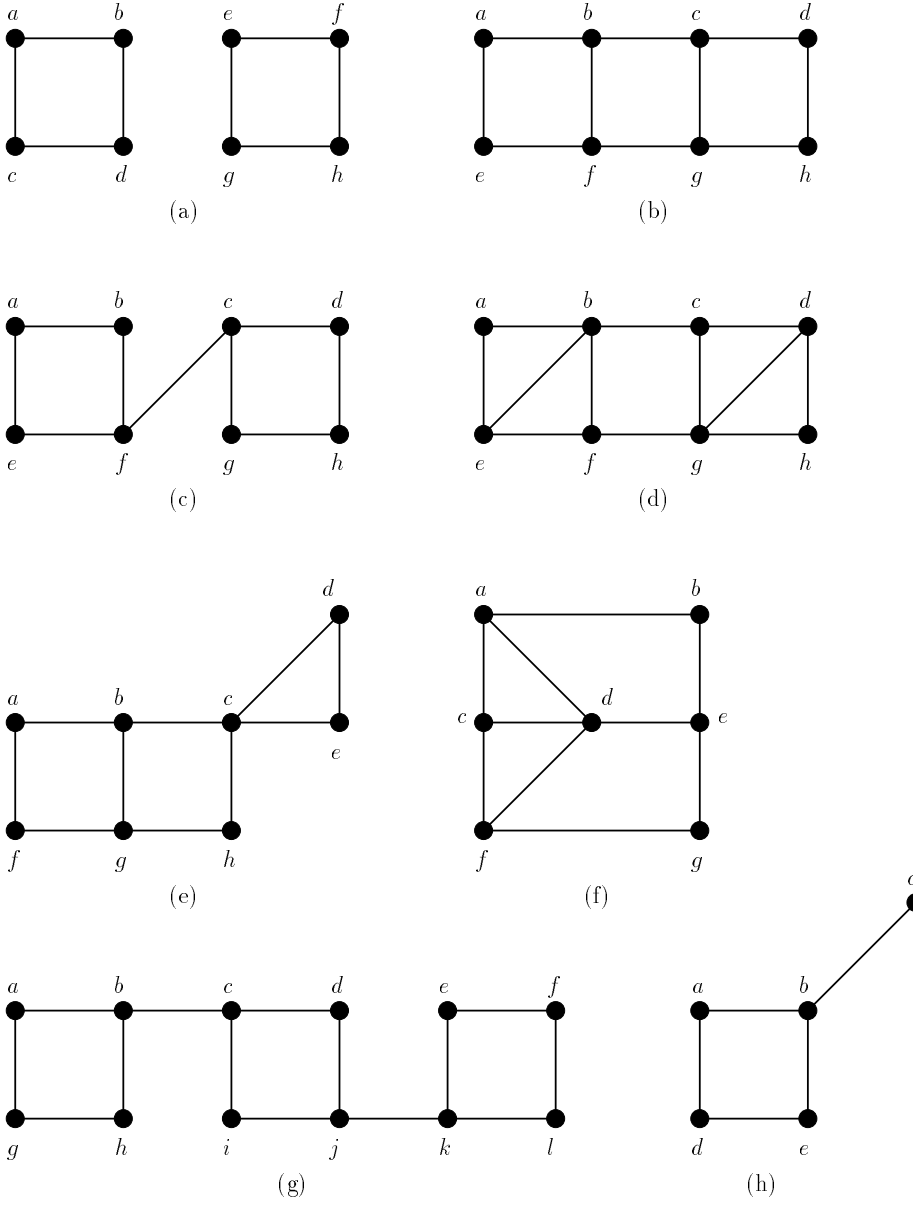


Figure 3.23: Graphs for exercises of Section 3.2.

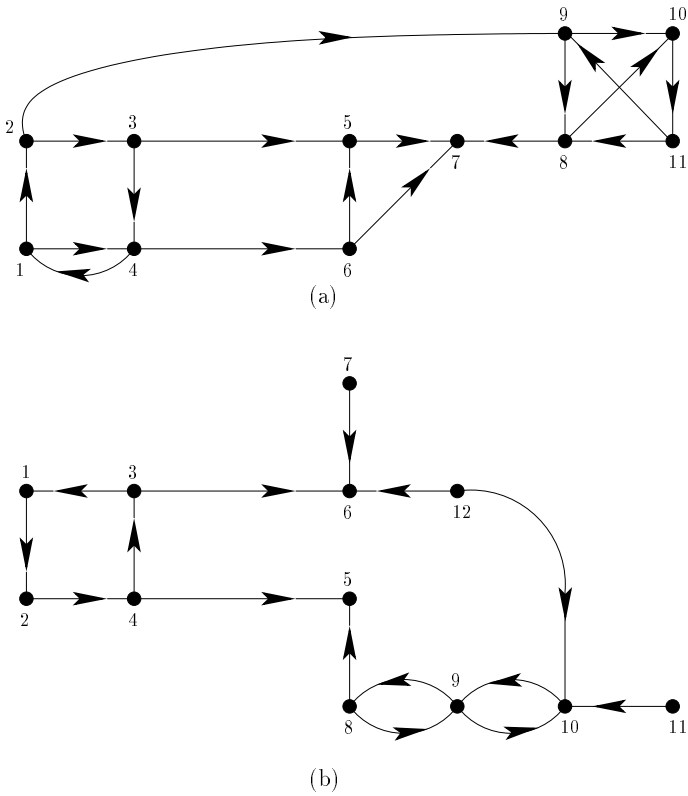


Figure 3.24: Digraphs for exercises of Section 3.2.

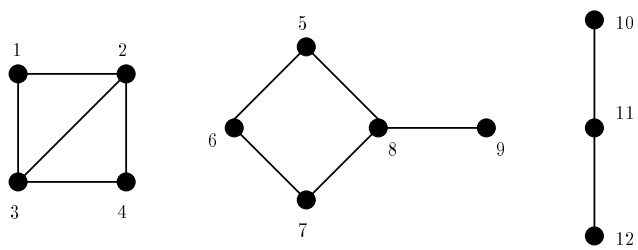


Figure 3.25: Graph for exercises of Section 3.2.

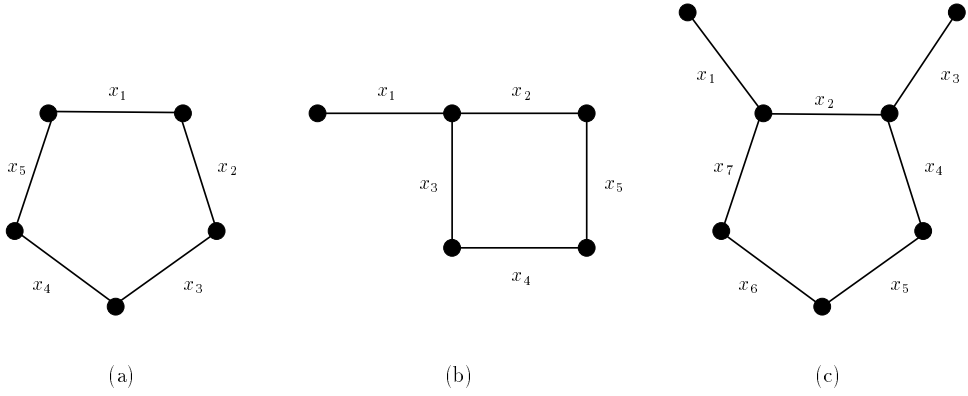


Figure 3.26: Systems for Exercise 12, Section 3.2.

15. In a digraph D, show that:

    (a) Every vertex is in some strong component.

    (b) Every vertex is in at most one strong component.

16. Show that a graph is connected if and only if it has a chain going through all the vertices.

17. Prove that a digraph is strongly connected if and only if it has a closed path going through all the vertices.

18. Prove that in a unilaterally connected digraph D, in any set of vertices, there is a vertex that can reach (using arcs of D) all others in the set.

19. In a communication network, suppose that an arc from x to y means that a message can be sent directly from x to y. If we want to place a message with as small a set of vertices as possible, so that it is possible for the message to reach all other vertices (perhaps in more than one step), what is the smallest number of vertices needed if the communication network is:

    (a) Strongly connected?

    (b) Unilaterally connected?

20. In Exercise 19, if the communication network is weakly connected, can we always place a message with at most half of the vertices to guarantee that it can reach all other vertices?

21. Show from the result of Exercise 18 that a digraph is unilaterally connected if and only if it has a path going through all the vertices.

22. (a) Give an example of a strongly connected digraph that has no cycle through all the vertices.

    (b) Does every unilaterally connected digraph have a simple path through all the vertices?

23. A *weak component* of a digraph is a maximal, weakly connected, generated subgraph. For each digraph of Figure 3.24, find all weak components.

24. A *unilateral component* of a digraph is a maximal, unilaterally connected, generated subgraph.

    (a) Find a unilateral component with five vertices in digraph (b) of Figure 3.24.

    (b) Is every vertex of a digraph in at least one unilateral component?

    (c) Can it be in more than one?

25. A digraph is *unipathic* if whenever v is reachable from u, there is exactly one simple path from u to v.

    (a) Is the digraph D_4 of Figure 3.7 unipathic?

    (b) What about the digraph of Figure 3.15?

26. For a digraph that is strongly connected and has n vertices, what is the least number of arcs? What is the most? (Observe that a digraph which is strongly connected with the least number of arcs is very vulnerable to disruption. How many links is it necessary to sever in order to disrupt communications?)

27. (Harary, Norman, and Cartwright [1965]) Refer to the definition of unipathic in Exercise 25. Can two cycles of a unipathic digraph have a common arc? (Give a proof or counterexample.)

28. (Harary, Norman, and Cartwright [1965]) If D is strongly connected and has at least two vertices, does every vertex have to be on a cycle? (Give a proof or counterexample.)

29. Suppose that a digraph D is not weakly connected.

    (a) If D has four vertices, what is the maximum number of arcs?

    (b) What if D has n vertices?

30. Do Exercise 29 for digraphs that are unilaterally connected but not strongly connected.

31. Do Exercise 29 for digraphs that are weakly connected but not unilaterally connected.

32. The reliability of a network modeled as a digraph D can be measured by how much its connectedness changes when a single arc or vertex fails. Let $D-u$ be the subgraph generated by vertices different from u. Give examples of digraphs D and vertices u with the following properties, or show that there are no such digraphs:

    (a) D is strongly connected and $D-u$ is unilaterally but not strongly connected.

    (b) D is strongly connected and $D-u$ is not unilaterally connected.

    (c) D is unilaterally but not strongly connected and $D-u$ is not unilaterally connected.

33. Repeat Exercise 32 for $D-a$, where a is an arc of D and $D-a$ is the subgraph of D obtained by removing arc a.

34. (Harary, Norman, and Cartwright [1965]) If D is a digraph, define the *complementary digraph* D^c as follows: $V(D^c) = V(D) = V$ and an ordered pair (u, v) from $V \times V$ (with $u \neq v$) is in $A(D^c)$ if and only if it is not in $A(D)$. For example, if D is the digraph of Figure 3.27, then D^c is the digraph shown. Give examples of digraphs D that are weakly connected, not unilaterally connected, and such that:

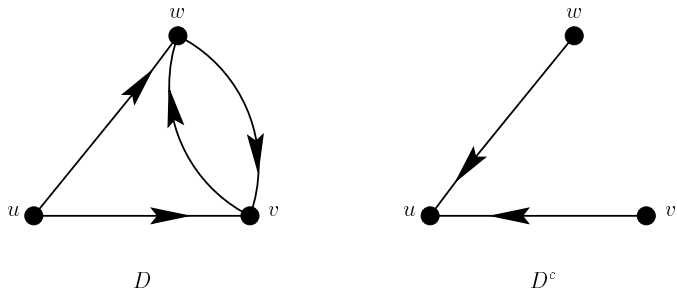


Figure 3.27: A digraph D and its complementary digraph D^c.

(a) D^c is strongly connected.

(b) D^c is unilaterally connected but not strongly connected.

(c) D^c is weakly connected but not unilaterally connected.

35. Find the number of distinct cycles of length k in the complete symmetric digraph of n vertices if two cycles are considered the same if one can be obtained from the other by changing the starting vertex.

3.3 GRAPH COLORING AND ITS APPLICATIONS

3.3.1 Some Applications

In Example 1.4 we considered the problem of scheduling meetings of committees in a state legislature and translated that into a problem concerning graphs. In this section we formulate the graph problem as a problem of coloring a graph. We remark on a number of applications of graph coloring. In the next section we apply the counting tools of Chapter 2 to count the number of graph colorings.

Example 3.11 Scheduling Meetings of Legislative Committees (Examples 1.4, 2.33, 2.35 Revisited) In the committee scheduling problem, we draw a graph G where the vertices of G are all the committees that need to be assigned regular meeting times and two committees are joined by an edge if and only if they have a member in common. Now we would like to assign a meeting time to each committee in such a way that if two committees have a common member, that is, if the corresponding vertices are joined by an edge in G, then the committees get different meeting times. Instead of assigning meeting times, let us think of assigning a color (corresponding to a meeting time) to each vertex of G, in such a way that if two vertices are joined by an edge, they get a different color. Committee scheduling is a prime example of *graph coloring* (*vertex coloring* or *coloring* are also used): Coloring the vertices of a graph so that adjacent vertices get different colors. (Edge coloring a graph will be addressed in the exercises.) If such an assignment can be carried out for G using at most k colors, we call it a *k-coloring* of G and

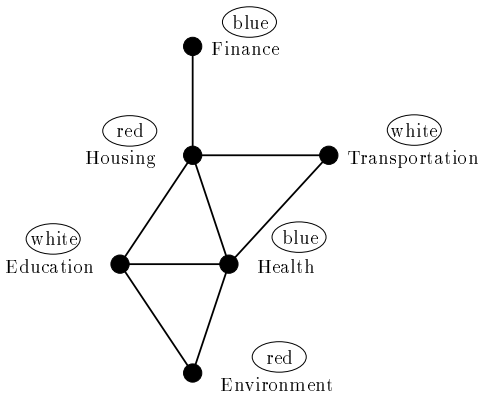


Figure 3.28: A 3-coloring for the graph G of Figure 1.1. The color assigned to a vertex is circled.

say G is *k-colorable*. The smallest number k such that G is k-colorable is called the *chromatic number* of G and is denoted $\chi(G)$. To illustrate these ideas, let us return to the graph of Figure 1.1 and call that graph G. A 3-coloring of G is shown in Figure 3.28. The three colors used are red, white, and blue. Note that this graph also has a 4-coloring. Indeed, by our definition, this figure shows a 4-coloring—we do not require that each color be used. A 4-coloring that uses four colors would be obtained from this coloring by changing the color of the Finance vertex (or for that matter any vertex) to green. We can always increase the number of colors used (up to the number of vertices). Thus, the emphasis is on finding the smallest number of colors we can use, that is, $\chi(G)$. Here, $\chi(G)$ obviously equals 3, for the three vertices Education, Housing, and Health must all get different colors. In this section we describe applications of graph coloring. As we remarked in Example 1.4, other applications with the same flavor as scheduling committee meetings involve scheduling final exam and class meeting times in a university, scheduling job assignments in a factory, and many other such problems. ■

Example 3.12 Index Registers and Optimizing Compilers (Tucker [1984]) In an optimizing compiler, it is more efficient to temporarily store the values of frequently used variables in index registers in the central processor, rather than in the regular memory, when computing in loops in a program. We wish to know how many index registers are required for storage in connection with a given loop. We let the variables that arise in the loop be vertices of a graph G and draw an edge in G between two variables if at some step in the loop, they will both have to be stored. Then we wish to assign an index register to each variable in such a way that if two variables are joined by an edge in G, they must be assigned to different registers. The minimum number of registers required is then given by the chromatic number of G. ■

Example 3.13 Channel Assignments Television transmitters in a region are to be assigned a channel over which to operate. If two transmitters are within 100 miles of each other, they must get different channels. The problem of assigning channels can be looked at as a graph coloring problem. Let the vertices of a graph G be the transmitters and join two transmitters by an edge if and only if they are within 100 miles of each other. Assign a color (channel) to each vertex so that if two vertices are joined by an edge, they get different colors. How few channels are needed for a given region? This is the chromatic number of G. (For more information on applications of graph coloring to television or radio-frequency assignments, see, for example, Cozzens and Roberts [1982], Hale [1980], Opsut and Roberts [1981], Roberts [1991], van den Heuvel, Leese, and Shepherd [1998], or Welsh and Whittle [1999].) ■

Example 3.14 Routing Garbage Trucks Let us next consider a routing problem posed by the Department of Sanitation of the City of New York (see Beltrami and Bodin [1973] and Tucker [1973]).[10] It should be clear that techniques like those to be discussed can be applied to other routing problems, for example milk routes and air routes. A garbage truck can visit a number of sites on a given day. A *tour* of such a truck is a schedule (an ordering) of sites it visits on a given day, subject to the restriction that the tour can be completed in one working day. We would like to find a set of tours with the following properties:

1. Each site i is visited a specified number k_i times in a week.

2. The tours can be partitioned among the six days of the week (Sunday is a holiday) in such a way that (a) no site is visited twice on one day[11] and (b) no day is assigned more tours than there are trucks.

3. The total time involved for all trucks is minimal.

In one method proposed for solving this problem, one starts with any given set of tours and improves the set successively as far as total time is concerned. (In the present state of the art, the method comes close to a minimal set, but does not always reach one.) At each step, the given improved collection of tours must be tested to see if it can be partitioned in such a way as to satisfy condition (2a), that is, partitioned among the six days of the week in such a way that no site is visited twice on one day. Thus, we need an efficient test for "partitionability" which can be applied over and over. Formulation of such a test reduces to a problem in graph coloring, and that problem will be the one on which we concentrate. (The reader is referred to Beltrami and Bodin [1973] and to Tucker [1973] for a description of the treatment of the total problem.)

To test if a given collection of tours can be partitioned so as to satisfy condition (2a), let us define a graph G, the *tour graph*, as follows. The vertices of G are

[10] Other applications of graph theory to sanitation are discussed in Section 11.4.3.

[11] Requirement (a) is included to guarantee that garbage pickup is spread out enough to make sure that there is no accumulation.

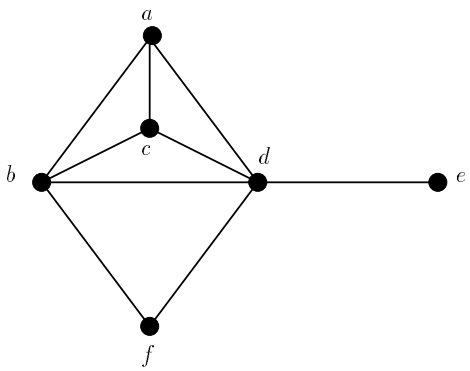


Figure 3.29: A graph of clique number 4.

the tours in the collection, and two distinct tours are joined by an edge if and only if they service some common site. Then the given collection of tours can be partitioned into six days of the week in such a way that condition (2a) is satisfied if and only if the collection of vertices $V(G)$ can be partitioned into six classes with the property that no edge of G joins vertices in the same class. It is convenient to speak of this question in terms of colors. Each class in the partition is assigned one of six colors and we ask for an assignment of colors to vertices such that no two vertices of the same color are joined by an edge.[12] The question about tours can now be rephrased as follows: Is the tour graph 6-colorable? ■

Example 3.15 Fleet Maintenance Vehicles (cars, trucks, ships, planes) come into a maintenance facility at scheduled times for regular maintenance. Two vehicles in the facility in overlapping time periods must be assigned different spaces. How many spaces does the facility require? This problem can also be formulated as a graph coloring problem. Let the vertices of a graph G be the vehicles scheduled for maintenance and join two vehicles by an edge if they are scheduled at overlapping times. Assign a color (space) to each vertex so that if two vertices are joined by an edge, they get different colors (spaces). The answer to our question is given by $\chi(G)$. ■

Example 3.16 Clique Number and Chromatic Number Suppose that G is a graph. A *clique* in G is a collection of vertices, each joined to the other by an edge. For instance, in the graph G of Figure 3.29, $\{a,b,c\}, \{d,e\}$, and $\{a,b,c,d\}$ are cliques. The *clique number* of G, $\omega(G)$, is the size of the largest clique of G. In our example, $\omega(G) = 4$. Since all vertices in a clique must receive different colors, this implies that $\chi(G) \geq \omega(G)$. Can you give an example of a graph in which $\chi(G) > \omega(G)$? ■

[12] This idea is due to Tucker [1973].

Example 3.17 Chromatic Number and Independence Number Suppose that G is a graph and W is a subset of the vertex set of G. W is called an *independent set* of G if no two vertices of W are joined by an edge. The *independence number* $\alpha(G)$ is the size of a largest independent set of G. For instance, in the graph of Figure 3.29, $\{a, f\}$ and $\{e, c, f\}$ are independent sets. There is no independent set of four vertices, so $\alpha(G) = 3$. Let us suppose that the vertices of G have been colored. There can be no edges between vertices of the same color, so all vertices of a given color define an independent set. Thus, a coloring of the $n = |V(G)|$ vertices of G in $\chi(G)$ colors partitions the vertices into $k = \chi(G)$ "color classes," each defining an independent set. The average size of such an independent set is $n/k = |V(G)|/\chi(G)$. Thus, by an application of the pigeonhole principle (Corollary 2.15.2), there is at least one independent set of size at least n/k, that is,

$$\alpha(G) \geq \frac{|V(G)|}{\chi(G)}$$

or

$$\chi(G)\alpha(G) \geq |V(G)|. \tag{3.9}$$ ■

Example 3.18 Course Scheduling Suppose that a university lets its professors schedule their classes at any time(s) during the week. Let us assume that any one course must be scheduled to consume three hours of classroom time. For example, one semester of classes might look as follows:

FALL 1999
Math 027: {Monday 3–4, Thursday 1–2:15, Friday 3–3:45}
Econ 321: {Tuesday 3–4:30, Thursday 3–4:30}
Span 114: {Monday 8–9, Wednesday 8–9, Friday 8–9}
⋮

How does a student pick classes that don't overlap? The university can build a graph where each vertex represents a different course at their school. If two courses have any time in which they overlap, draw an edge between those two vertices. A student wanting n courses must then pick n vertices which form an independent set in the graph. ■

Example 3.19 Map-coloring The problem of coloring maps is an old and important problem which has been one of the prime stimulants for the development of graph theory. To explain the map-coloring problem, let us consider the map of Figure 3.30. It is desired to color the countries on the map in such a way that if two countries share a common boundary, they get a different color. Of course, each country can be colored in a different color. However, for many years, cartographers have been interested in coloring maps with a small number of colors, if possible. We can start coloring the countries in the map of Figure 3.30 by coloring country 1 red (see Figure 3.31). Then country 2, which shares a boundary with country 1, must get a different color, say blue. Country 3 shares a boundary with each of the other

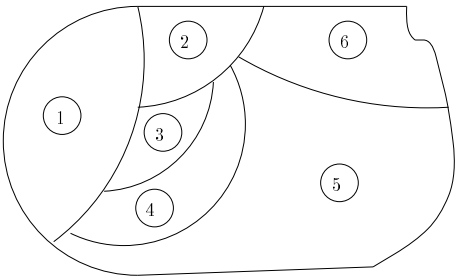


Figure 3.30: A map.

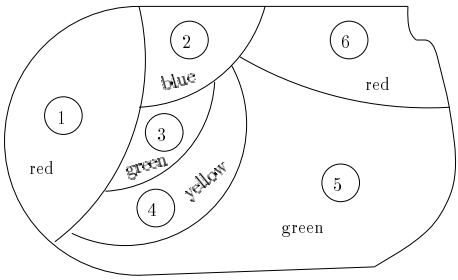


Figure 3.31: A coloring of the map in Figure 3.30.

countries colored so far, so it must get still a different color, say green. Country 4 shares a boundary with all of the first three countries, so it must get still a fourth color, say yellow. Country 5 shares a boundary with countries 1, 2, and 4, but not with 3. Thus, it is possible to color country 5 green. Finally, country 6 cannot be blue or green. In Figure 3.31, we have colored it red. Notice that the map has been colored with four colors. No one has ever found a map for which more than four colors are needed, provided that "map" and "boundary" are defined precisely so as to eliminate such things as countries having two pieces, countries whose common boundary is a single point, and so on. For more than 100 years, it was conjectured that every map could be colored in four or fewer colors. However, despite the work of some of the best mathematical minds in the world, this *four-color conjecture* was neither proved nor disproved, and the four-color problem remained unsolved. Finally, in 1977, the four-color conjecture was proved (see Appel and Haken [1977], Appel, Haken, and Koch [1977]). The original proof of the four-color theorem involved the use of high-speed computers to check certain difficult cases and involved some 1200 hours of computer time. (Recent work has led to "simpler" proofs of the four-color theorem; see, for example, Robertson, *et al.* [1997].)

One of the major steps in handling the map-coloring problem and k-colorings of maps was to translate the map-coloring problem into an equivalent but somewhat more tractable problem. Let the nation's capital of each country be represented by a point. Join two of these capitals by a (dashed) line if the corresponding countries share a common boundary. This gives rise to the lines of Figure 3.32. In Figure 3.33 the diagram is redrawn with only the capitals and the lines joining them remaining. This diagram defines a graph. Instead of coloring a whole country, we can think of just coloring its capital. In terms of a graph such as that in Figure 3.33, the requirement is that if two capitals or vertices are joined by an edge, they must get different colors. Thus, a map is colorable in k colors if and only if the corresponding graph is k-colorable. ■

Graph coloring and its generalizations have numerous applications in addition to those described here, for instance to time sharing in computer science, phasing traffic lights in transportation science, and various scheduling and maintenance

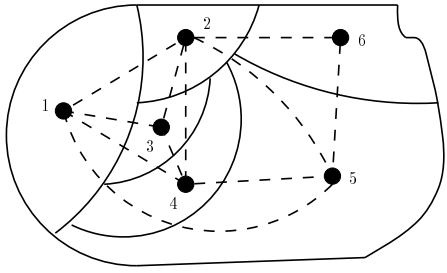


Figure 3.32: A dashed line joins two capitals if and only if their corresponding countries share a common boundary.

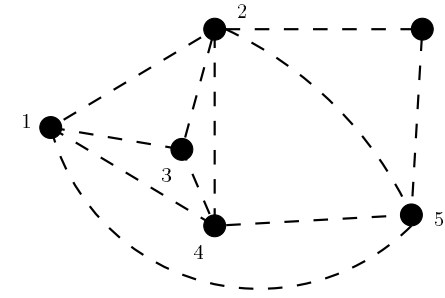


Figure 3.33: The graph of linked capitals from the map of Figure 3.32.

problems in operations research. See Opsut and Roberts [1981] and Roberts [1991] for descriptions of some of these problems.

3.3.2 Planar Graphs

The graph of Figure 3.33 has the property that no two edges cross except at vertices of the graph. A graph that has this property, or which has an equivalent (isomorphic) redrawing with this property, is called *planar*. Every map gives rise to a planar graph, and, conversely, every planar graph comes from a map. Thus, the four-color theorem can be stated as the following theorem in graph theory: Every planar graph is 4-colorable. The first graph of Figure 3.34 is planar, even though its drawing has edges crossing. The second graph of Figure 3.34, which is equivalent (isomorphic) to the first graph, is drawn without edges crossing. The first graph in Figure 3.34 is the complete graph K_4. Thus, K_4 is planar. The graph of Figure 3.35(a), which is K_5, is not planar. No matter how you locate five points in the plane, it is impossible to connect them all with lines without two of these lines crossing. The reader is encouraged to try this. The graph of Figure 3.35(b) is another example of a graph that is not planar. This graph is called the *water-light-gas graph* and is denoted by $K_{3,3}$. We think of three houses and three utilities and try to join each house to each utility. It is impossible to do this without some lines crossing. Again, the reader is encouraged to try this. The problem of determining if a graph is planar has a variety of applications. For instance, in electrical engineering, the planar graphs correspond exactly to the possible printed circuits. In Section 11.6.4 we show the use of planar graphs in a problem of facilities design. Kuratowski [1930] showed that in some sense, K_5 and $K_{3,3}$ are the only nonplanar graphs.[13]

To make precise the sense in which K_5 and $K_{3,3}$ are the only nonplanar graphs, let us say that graph G' is obtained from graph G by *subdivision* if we obtain G' by adding vertices on one edge of G. In Figure 3.36, graph G'_i is always obtained

[13] The rest of this subsection may be omitted.

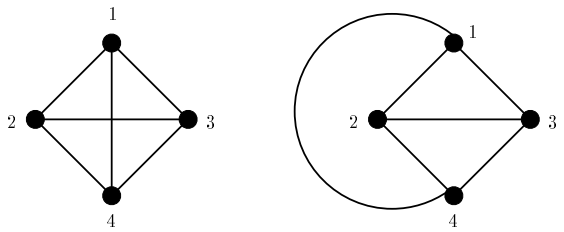


Figure 3.34: The first graph is planar, as is demonstrated by the second graph and the isomorphism shown by the vertex labelings.

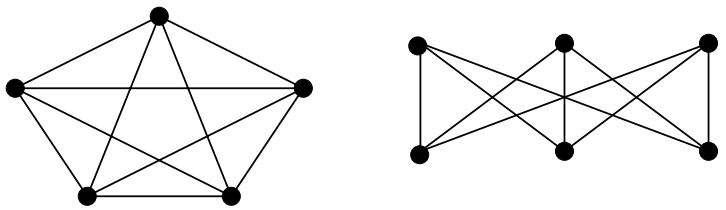


Figure 3.35: Two nonplanar graphs, K_5 and $K_{3,3}$.

from graph G_i by subdivision. Two graphs G and G' are called *homeomorphic* if both can be obtained from the same graph H by a sequence of subdivisions. For example, any two simple chains are homeomorphic. Figure 3.37 shows two graphs G and G' obtained from a graph H by a sequence of subdivisions. Thus, G and G' are homeomorphic (and, incidentally, homeomorphic to H).

Theorem 3.2 (Kuratowski [1930]) A graph is planar if and only if it has no subgraph[14] homeomorphic to K_5 or $K_{3,3}$.

For a proof of Theorem 3.2, we refer the reader to Harary [1969], Bondy and Murty [1976], or Makarychev [1997]. According to Kuratowski's Theorem, the graph G of Figure 3.38 is not planar because it is homeomorphic to K_5 and the graph G' is not planar because it has a subgraph H homeomorphic to $K_{3,3}$.

Suppose that $e = \{x, y\}$ is an edge of G. *Contracting* e means identifying the two vertices x and y. The new combined vertex is joined to all those vertices to which either x or y were joined. If both x and y were joined to a vertex z, only one of the edges from the combined vertex to z is included. In Figure 3.36, graph G_i'' is always obtained from graph G_i by contraction of edge e. *Contracting* G to G' means contracting a sequence of edges of G to obtain G'. Another characterization of planar graphs using contraction is given by the following theorem, due to Halin [1964], Harary and Tutte [1965], and Wagner [1937].

[14] Not necessarily a generated subgraph.

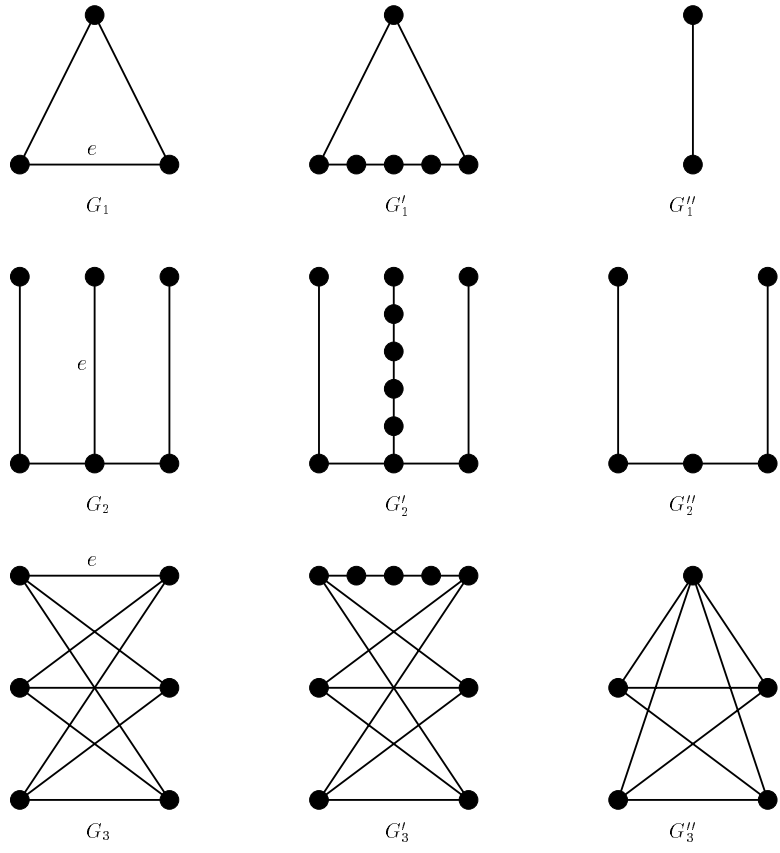


Figure 3.36: Graph G_i' is obtained from graph G_i by subdivision of edge e. Graph G_i'' is obtained from graph G_i by contraction of edge e.

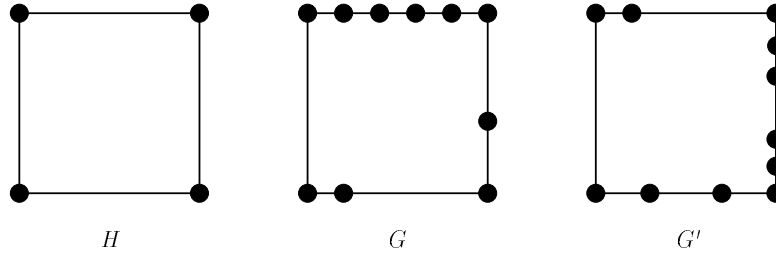


Figure 3.37: G and G' are homeomorphic because they are each obtained from H by subdivision.

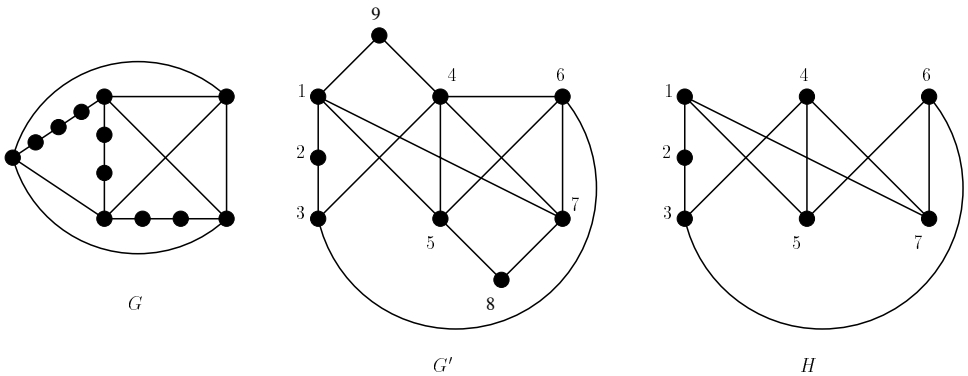


Figure 3.38: G is homeomorphic to K_5 and G' has the subgraph H which is homeomorphic and contractible to $K_{3,3}$.

Theorem 3.3 A graph G is planar if and only if no subgraph of G can be contracted to K_5 or $K_{3,3}$.

For a proof of Theorem 3.3, see Harary [1969], Bondy and Murty [1976], or Makarychev [1997]. By Theorem 3.3, G' of Figure 3.38 is not planar because it has a subgraph H that can be contracted to $K_{3,3}$ by contracting edge $\{1,2\}$ or $\{2,3\}$.

Before closing this subsection, we note that Kuratowski's Theorem does not give a good algorithm for testing a graph for planarity. However, there are such algorithms. For a good discussion of them, see Even [1979]. In particular, see Demourcron, Malgrance, and Pertuiset [1964] and Klotz [1989] for quadratic $[O(n^2)]$ algorithms or Hopcroft and Tarjan [1974] and Booth and Lueker [1976] for linear $[O(n)]$ algorithms.

3.3.3 Calculating the Chromatic Number

Let us study the colorability of various graphs. The graph K_4 is obviously colorable in four colors but not in three or fewer. Thus, $\chi(K_4) = 4$. The graph K_5 is colorable in five colors, but not in four or fewer (why?). Thus, $\chi(K_5) = 5$. (This is not a counterexample to the four-color theorem, since we have pointed out that K_5 is not a planar graph and so could not arise from a map.) $K_{3,3}$ in Figure 3.35 is colorable in two colors: Color the top three vertices red and the bottom three blue. Since clearly two colors are needed, the chromatic number of $K_{3,3}$ is 2.

Let us return briefly to the tour graph problem of Example 3.14. In general, to apply the procedure for finding a minimal set of tours, one has to have an algorithm, which can be applied quickly over and over again, for deciding whether a given graph is k-colorable. Unfortunately, there is not always a "good," that is, polynomial, algorithm for solving this problem. Indeed, in general, it is not known whether there is a "good" algorithm (in the sense of Section 2.4) for deciding if a given graph is k-colorable. This problem is NP-complete in the sense of Section 2.18,

so is difficult in a precise sense. The garbage truck routing problem is thus reduced to a difficult mathematical question. However, formulation in precise mathematical terms has made it clear why this is a hard problem, and it has also given us many tools to use in solving it, at least in special cases. Let us remark that in a real-world situation, it is not sufficient to say that a problem is unsolvable or hard. Imagine a \$500,000 consultant walking into the mayor's office and reporting that after careful study, he or she has concluded that the problem of routing garbage trucks is hard! Garbage trucks must be routed. So what can you do in such a situation? The answer is, you develop partial solutions, you develop solutions that are applicable only to certain special situations, you modify the problem, or in some cases, you even "lie." You lie by using results that are not necessarily true but seem to work. One such result is the Strong Perfect Graph Conjecture or the Strong Berge Conjecture, which goes back to Claude Berge [1961, 1962]. (To understand the following reasoning, it is not important to know what this conjecture says. See Golumbic [1980] for a detailed treatment of the conjecture or Roberts [1976, 1978] or Tucker [1973] for a treatment of the conjecture and its applications to garbage trucks and routing.) As Tucker [1973] pointed out, if the conjecture is true, there is an efficient algorithm for determining if a given graph is colorable in a given number of colors, at least in the context of the tour graph problem, where the tour graph is changed each time only locally and not globally. Thus, Tucker argued, it pays to "lie" and to use the Strong Berge Conjecture in routing garbage trucks. What could go wrong? The worst thing that could happen, said Tucker, is the following. One applies the conjecture to garbage truck routing and finds a routing that is supposedly assignable to the 6 days of the week, but which in fact cannot be so assigned. In this worst case, think of the boon to mathematics: We would have found a counterexample to the Strong Berge Conjecture! This remarkable argument, however, is no longer necessary. After over 40 years, the Strong Berge Conjecture was proved by Chudnovsky, *et al.* [2002] (see Mackenzie [2002]).

3.3.4 2-Colorable Graphs

Let us note next that there is one value of k for which it is easy to determine if G is k-colorable. This is the case $k = 2$.[15] A graph is 2-colorable if and only if the vertices can be partitioned into two classes so that all edges in the graph join vertices in the two different classes. (Why?) A graph with this kind of a partition is called *bipartite*. The depth-first search procedure to be described in Section 11.1 gives a polynomial algorithm for testing if a graph is bipartite (see Reingold, Nievergelt, and Deo [1977, pp. 399–400]).

There is also a useful characterization of 2-colorable graphs, which we state next. Let Z_p be the graph that consists of just a single circuit of p vertices. Figure 3.39 shows Z_3, Z_4, Z_5, and Z_6. It is easy enough to show that Z_4 and Z_6 are 2-colorable. A 2-coloring for each is shown. Clearly, Z_3 is not 2-colorable. Z_5 is also not 2-colorable. This takes a little proving, and we leave the proof to the reader. In

[15] For a discussion of other cases where there are good algorithms for determining if G is k-colorable, see Garey and Johnson [1979], Golumbic [1980], and Jensen and Toft [1995].

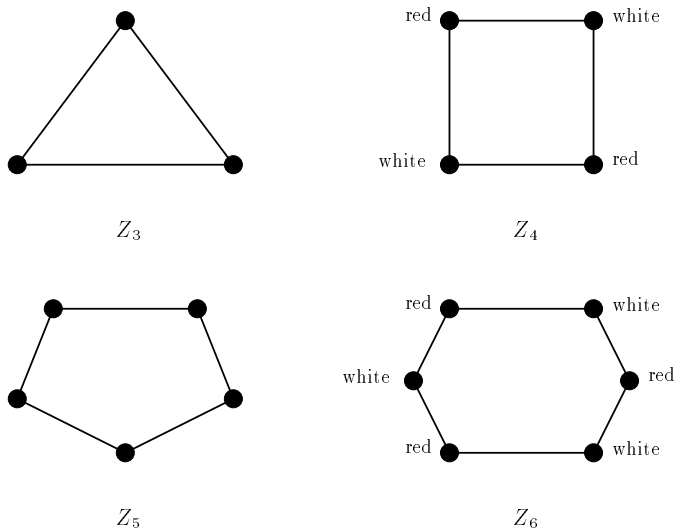


Figure 3.39: The graphs Z_p for $p = 3, 4, 5$, and 6.

general, it is easy to see that Z_p is 2-colorable if and only if p is even.

Now suppose that we start with Z_5, possibly add some edges, and then add new vertices and edges joining these vertices or joining them to vertices of Z_5. We might get graphs such as those in Figure 3.40. Now none of these graphs is 2-colorable. For a 2-coloring of the whole graph would automatically give a 2-coloring of Z_5. This is a general principle: A k-coloring of any graph G is a k-coloring of all subgraphs of G. Thus, any graph containing Z_5 as a subgraph is not 2-colorable. The same is true for Z_3, Z_7, Z_9, and so on. If G has any circuit of odd length, the circuit defines a subgraph of the form Z_p, for p odd; thus G could not be 2-colorable. The converse of this statement is also true, and we formulate the result as a theorem.

Theorem 3.4 (König [1936]) A graph is 2-colorable if and only if it has no circuits of odd length.

To prove the converse part of Theorem 3.4, we start with a graph G with no circuits of odd length and we present an algorithm for finding a 2-coloring of G. We may suppose that G is connected. (Otherwise, we can color each connected component separately.) Pick an arbitrary vertex x. Color x blue. Color all neighbors of x red. For each of these neighbors, color its uncolored neighbors blue. Continue in this way until all vertices are colored. The algorithm is illustrated by the graph of Figure 3.41, which is connected and has no odd-length circuits. Here, x is chosen to be a and the 2-coloring is shown.

To implement this algorithm formally, vertices that have been colored are saved in an ordered list called a *queue*. At each stage of the algorithm, we find the first vertex y in the queue and remove (pop) it from the queue. We find its uncolored neighbors and color them the opposite color of y. We then add these neighbors

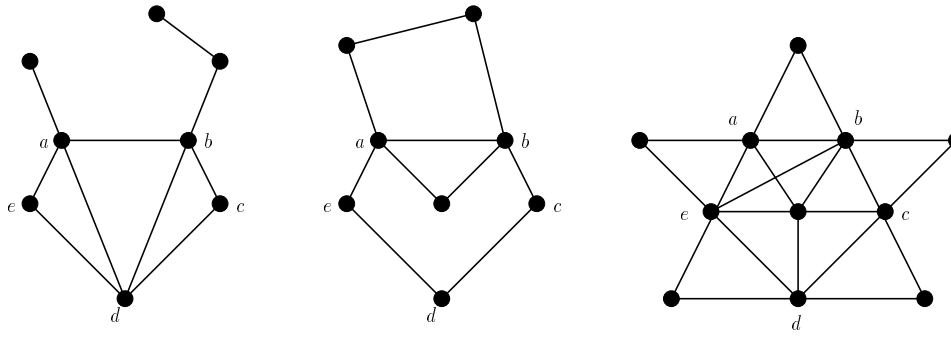


Figure 3.40: Graphs containing Z_5 as a subgraph. The vertices of Z_5 are labeled a, b, c, d, and e.

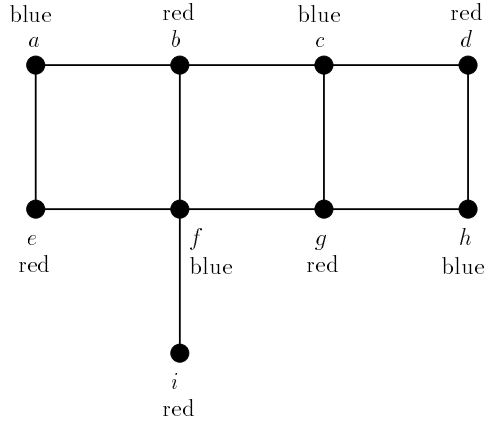


Figure 3.41: A connected graph without odd-length circuits. A 2-coloring obtained by first coloring vertex a is shown.

Table 3.2: Applying Algorithm 3.1 to the Graph of Figure 3.41

Vertices currently considered	Vertices colored	New queue Q
	a (blue)	a
a	b, e (red)	b, e
b	c, f (blue)	e, c, f
e	none	c, f
c	d, g (red)	f, d, g
f	i (red)	d, g, i
d	h (blue)	g, i, h

to the end of the queue. We continue until all vertices have been colored. The algorithm is formally stated as Algorithm 3.1. Here, Q is the queue.

Algorithm 3.1: Two-Coloring

Input: A graph $G = (V, E)$ that is connected and has no odd-length circuits.
Output: A coloring of the vertices of G using the two colors blue and red.

Step 1. Initially, all vertices of V are uncolored and Q is empty.

Step 2. Pick x in V, color x blue, and put x in Q.

Step 3. Let y be the first vertex in Q. Remove y from Q.

Step 4. Find all uncolored neighbors of y. Color each in the color opposite that used on y. Add them to the end of Q in arbitrary order.

Step 5. If all vertices are colored, stop. Otherwise, return to Step 3.

To illustrate Algorithm 3.1, consider the graph of Figure 3.41. The steps are summarized in Table 3.2. Pick x to be vertex a, color a blue, and put a into the queue. Find the uncolored neighbors of a, namely b and e. Color them red (that is, the opposite of a's color). Remove a from the queue Q and add b and e, say in the order b, e. Pick the first vertex in Q; here it is b. Remove it from Q. Find its uncolored neighbors; they are c and f. Color these the opposite color of that used for b, namely, blue. Add them to the end of Q in arbitrary order. If c is added first, then Q now is e, c, f. Remove the first vertex in Q, namely e. It has no uncolored neighbors. Q is now c, f. Go to the first vertex in Q, namely c, and remove it. Find its uncolored neighbors, d and g, and color them the opposite of c, namely, red. Add d and g to the end of Q, say d first. Q is now f, d, g. Next remove f from the head of Q, color its uncolored neighbor i red, and add i to the end of Q. Q is now d, g, i. Finally, remove d from the head of Q, color its uncolored neighbor h blue, and add h to the end of Q. Stop since all vertices have now been colored.

The procedure we have used to visit all the vertices is called *breadth-first search*. It is a very efficient computer procedure that has many applications in graph theory.

We shall return to breadth-first search, and the related procedure called depth-first search, in Section 11.1 when we discuss algorithms for testing a graph for connectedness. Algorithm 3.1 is a "good" algorithm in the sense of Sections 2.4 and 2.18. It is not hard to show that its complexity is of the order $n + e$, where n is the number of vertices of the graph and e is the number of edges. Since a graph has at most $\binom{n}{2}$ edges, we reason as in Section 3.2.3 to conclude that

$$e \leq \binom{n}{2} = \frac{n(n-1)}{2} \leq n^2.$$

Thus, Algorithm 3.1 takes at most a number of steps on the order of $n + n^2$, which is a polynomial in n. [In the notation of Section 2.18, the algorithm is $O(n^2)$.]

To show that Algorithm 3.1 works, we have to show that every vertex eventually gets colored and that we attain a graph coloring this way. Connectedness of the graph G guarantees that every vertex eventually gets colored. (We omit a formal proof of this fact.) To show that we get a graph coloring, suppose that u and v are neighbors in G. Could they get the same color? The easiest way to see that they could not is to define the distance $d(a, b)$ between two vertices a and b in a connected graph to be the length of the shortest chain between them. Then one can show that vertex z gets colored red if $d(x, z)$ is odd and blue if $d(x, z)$ is even.[16] (The proof is left as Exercise 31.) Now if two neighbors u and v are both colored red, there is a shortest chain C_1 from x to u of odd length and a shortest chain C_2 from x to v of odd length. It follows that C_1 plus edge $\{u, v\}$ plus C_2 (followed backwards) forms a closed chain from x to x of odd length. But if G has an odd-length closed chain, it must have an odd-length circuit (Exercise 32). Thus, we have a contradiction. We reach a similar contradiction if u and v are both colored blue.

Remember that Algorithm 3.1 and Theorem 3.4 apply only to 2-colorings. The general problem of graph coloring is an NP-complete problem; there is no known polynomial algorithm that determines k-colorability, for any fixed $k \geq 3$, let alone finding an actual optimal coloring.

3.3.5 Graph-Coloring Variants

There are many variations of graph (vertex) coloring. The following three examples only begin to scratch the surface of this burgeoning area in graph theory.

Example 3.20 Channel Assignment and the *T*-Coloring Problem Recall the problem of channel assignment from Example 3.13. The graph associated with this problem had the transmitters as vertices and an edge between vertices if the corresponding transmitters were within 100 miles of one another. Coloring this graph with as few colors as possible is tantamount to solving the channel assignment problem.

However, in certain situations not only can't "close" transmitters get the same channel but they can't get channels that differ by certain values. We call this set of

[16] The shortest proof of Theorem 3.4 is simply to *define* the coloring this way.

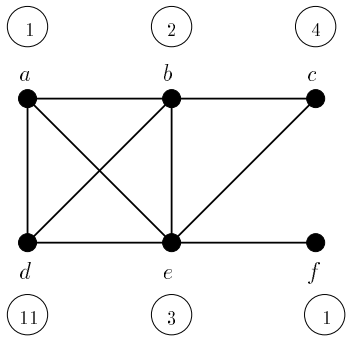


Figure 3.42: A T-coloring of a graph with T-set $T = \{0, 3, 4, 7\}$.

"forbidden" differences a *T-set* and denote it by T. For example, in UHF television, transmitters within 55 miles of one another can't have channels that differ by any value in the T-set $T = \{0, 1, 7, 14, 15\}$. Given a T-set T and a graph G, a vertex coloring of G in which adjacent vertices don't have colors whose absolute difference is in T is called a *T-coloring* of G. (*Absolute difference* refers to $|c_i - c_j|$ where c_i, c_j are colors.) Figure 3.42 gives an example of a T-coloring of a graph with a T-set $T = \{0, 3, 4, 7\}$. Notice that absolute differences in the colors of adjacent vertices equal 1, 2, 8, 9, and 10, none of which are in the T-set. We make the assumption that all T-sets contain 0. Otherwise, the very uninteresting case of all vertices being colored the same color would constitute a T-coloring. ■

In Example 3.20, we could have just as easily colored the vertices of Figure 3.42 with the colors 1, 100, 500, 1000, and 2000 (replacing 1, 2, 3, 4, and 11, respectively) to produce a T-coloring. However, this would not have been an efficient T-coloring. So, what do we mean by an efficient T-coloring? Just as minimizing the number of colors is primarily used as the criterion for efficient graph coloring, we need criteria for efficient T-colorings. With regard to channel assignment, sometimes we are concerned with the total number of channels used and other times we are concerned with the range of the channels used. Thus, sometimes we are interested in the total number of colors used in a T-coloring and sometimes we are more concerned with the range of colors used.

The *order* of a T-coloring refers to the total number of colors used. The order of the T-coloring in Figure 3.42 equals 5 since 5 distinct colors are used; 1, 2, 3, 4, and 11. The minimum number of colors needed to T-color a graph G (i.e., the minimum order) is called the *T-chromatic number of G* and is denoted $\chi_T(G)$. The T-coloring used in Figure 3.42 is not most efficient in this sense since G can be colored using 4 colors (but not 3). Vertices a, b, d, and e all need different colors since any two are joined by an edge. Thus, $\chi_T(G) \geq 4$. Vertices c and f can each be colored the same as a (or d), producing an order 4 T-coloring. So if G is the graph of Figure 3.42 and $T = \{0, 3, 4, 7\}$, then $\chi_T(G) = 4$.

The minimum order of a T-coloring of a graph G, $\chi_T(G)$, is not a new parameter, as the next theorem shows.

Theorem 3.5 (Cozzens and Roberts [1982]) For all graphs G and any T-set T,

$$\chi_T(G) = \chi(G).$$

Proof. Since we have assumed that 0 is contained in every T-set, any T-coloring of G will be (at the very least) a graph coloring. Thus, $\chi_T(G) \geq \chi(G)$.

Next, any graph coloring of G using j colors can be turned into a T-coloring using j colors in the following way. Without loss of generality, we can assume that the j colors in the graph coloring are the colors $1, 2, \ldots, j$. Replace color i with $i \cdot (t+1)$, where t is the largest element in the T-set T. These new colors form a T-coloring of G (see Exercise 37). Therefore, a graph coloring using $\chi(G)$ colors can be turned into a T-coloring using the same number of colors. Thus, $\chi_T(G) \leq \chi(G)$.

We have shown that $\chi_T(G) \geq \chi(G)$ and $\chi_T(G) \leq \chi(G)$. Therefore,

$$\chi_T(G) = \chi(G). \qquad \text{Q.E.D.}$$

A more natural criterion for efficiency has to do with the range of colors used in a T-coloring. Many times in channel assignment it is not how many channels are assigned that's important but whether or not all requests for channels can fit into the allocated bandwidth. We define the *span* of a T-coloring to be the difference between the largest and smallest colors used. The span of the T-coloring in Figure 3.42 equals 10, which comes from $11 - 1$, 11 being the largest color and 1 being the smallest color used in the T-coloring. The smallest span over all T-colorings of a graph G is called the *T-span of G* and is denoted $sp_T(G)$. The T-coloring used in Figure 3.42 is most efficient with regard to span. Assuming that 1 is the smallest color used in any T-coloring, restricting yourself to the colors $1, 2, \ldots, 10$ will not allow for a T-coloring of this graph (see Exercise 38).

Both the T-chromatic number and T-span of a graph are not easy values to ascertain. In the notation of Section 2.18, calculation of each is in the class of NP-complete problems. However, for certain classes of graphs and specific T-sets, exact values for the T-span have been found. (See, for example, Bonias [1991], Liu [1991], Raychaudhuri [1985], Tesman [1993], or Wang [1985].)

Example 3.21 Task Assignment and the Set Coloring Problem[17] A large and complicated task, such as building an airplane, is actually made up of many subtasks. Some of these subtasks are incompatible and may not be performed at the same time. For example, some of the subtasks may require the same tools, resources, hangar space, and so on. The task assignment problem is the problem of scheduling subtasks so that only compatible subtasks may be scheduled in overlapping time periods.

To formulate this problem graph-theoretically, we let each vertex represent one of the subtasks of the larger task. Put an edge between two subtasks if they are

[17] From Opsut and Roberts [1981].

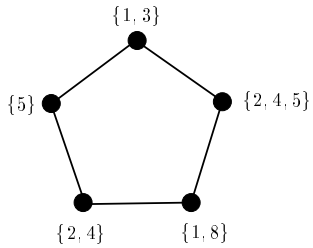


Figure 3.43: A set coloring for Z_5.

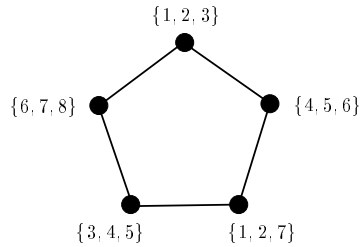


Figure 3.44: A 3-tuple coloring of Z_5.

incompatible. Then since each subtask will need a "time period," instead of assigning just one color to each vertex, we assign a set of colors which represent the times that the subtask will need. If x is a vertex, we denote the set assigned to x by $S(x)$. As before, we will require that adjacent vertices not receive the same colors. By this we mean that if x and y are adjacent, then $S(x)$ and $S(y)$ must not have any common members, that is,

$$S(x) \cap S(y) = \emptyset.$$

Such a coloring will be called a *set coloring* of the graph. Figure 3.43 gives an example of a set coloring for the graph Z_5. Note that the sets do not have to have the same size, which they don't in this example. As with vertex coloring, minimizing the total number of colors used will be the criterion for efficiency in set colorings. ■

To give a little more structure to set colorings, we consider the special case when all assigned sets are the same size. (Gilbert [1972] introduced this idea in connection with the frequency assignment problem involving mobile radios.) A *k-tuple coloring* will refer to a set coloring where each vertex is assigned a set of k colors. (If each set contains only one element, i.e., if we have a 1-tuple coloring, the set coloring is an ordinary graph coloring.) Figure 3.44 gives an example of a 3-tuple coloring of the graph Z_5 using 8 colors.

How many colors are needed to k-tuple color a graph? If the graph has n vertices, a crude upper bound would be kn colors. (Give each vertex a completely different set of k colors.) We can do much better than this, however. The following theorem gives an upper bound based on the chromatic number of the graph as opposed to the size of its vertex set. The proof of Theorem 3.6 is left as an exercise (Exercise 59).

Theorem 3.6 A graph G can be k-tuple colored in at most $k \cdot \chi(G)$ colors.

The minimum number of colors needed to k-tuple color a graph G is called the *k-tuple chromatic number* of G and is denoted by $\chi_k(G)$. From Theorem 3.6 we know that $\chi_k(G) \leq k \cdot \chi(G)$. This upper bound on $\chi_k(G)$ is, in fact, the exact value of $\chi_k(G)$ for many different classes of graphs, but it certainly does not hold in general. We have shown that we can 3-tuple color Z_5 using 8 colors, which is already better than $3 \cdot \chi(Z_5) = 3 \cdot 3 = 9$. (Irving [1983] has shown that finding

Table 3.3: A List Coloring for the List Assignment and Graph in Figure 3.45(b)

Vertex:	a	b	c	d	e	f
List:	$\{1,2\}$	$\{1,3\}$	$\{2,4\}$	$\{1,2\}$	$\{2,3\}$	$\{3,4\}$
Choose:	1	1	4	2	3	3

$\chi_k(G)$ is an NP-complete problem.) In fact, 8 is the 3-tuple chromatic number for Z_5. Let a and b be any pair of adjacent vertices. Then a and b cannot have any common colors. Without loss of generality, color vertex a with the colors 1, 2, 3 and color vertex b with the colors 4, 5, 6. It is then easy to show that the remaining vertices cannot be 3-tuple colored by using only one more additional color.

One class of graphs for which the bound in Theorem 3.6 is correct for determining the k-tuple chromatic number is the class of complete graphs. Since any two vertices are adjacent in a complete graph, no color can be repeated in a k-tuple coloring. Thus, we have the following theorem.

Theorem 3.7 For any n and k, $\chi_k(K_n) = k \cdot \chi(K_n) = kn$.

For other references to k-tuple colorings, see Brigham and Dutton [1982], Geller [1976], Roberts [1991], and Scott [1975].

Example 3.22 Scheduling Meetings of Legislative Committees (Example 1.4 Revisited): List Colorings In Example 1.4 we considered a scheduling problem where each committee chair provides a list of acceptable meeting times. The scheduling problem without this extra constraint was translated into a graph coloring problem in Example 3.11. With this constraint, it becomes the problem of finding a graph coloring in which the color assigned to a vertex is chosen from a list of acceptable colors. The same kind of problem arises in many other applications. For example, in channel assignment, the user of a given transmitter might specify a list of acceptable channels. Let $L(x)$ denote the list of colors assigned to vertex x. L is called a *list assignment* of G. A vertex coloring for G such that the color that you assign to a vertex x comes from the list $L(x)$ is called an *L-list coloring*.

Figure 3.45(a) shows an example of the graph $K_{3,3}$ with a list assignment. Can you find a list coloring for this assignment? Vertex a must be colored either 1 or 2. If it is colored 1, then vertex d must be colored 3 since there is an edge between vertex a and vertex d. Similarly, vertex f must be colored 2. But then there is no color left for vertex c. A similar problem occurs if vertex a is colored 2. Therefore, we have shown that this graph is not list colorable for this list assignment. However, if the lists assigned to the vertices were as in Figure 3.45(b) then choosing colors as in Table 3.3 gives selections that would work as a list coloring. ■

Example 3.23 List Colorings with Modified Lists In the committee scheduling example (Example 3.22), what would we do if there were no list coloring? We might ask some people to accept colors not on their original lists. One simple way

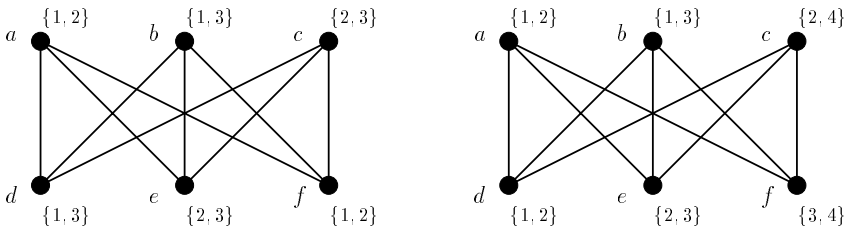


Figure 3.45: Two list assignments for $K_{3,3}$.

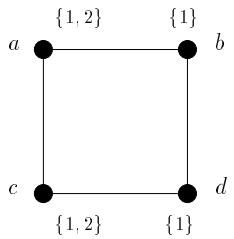


Figure 3.46: A 1-addable graph G.

to think of this is to allow some people x to expand their lists $L(x)$ by adding an additional color (from the available colors). What is the smallest number of people that have to do this? We say that G with list assignment L is *p-addable* if we can identify p distinct vertices $x_1, x_2, \ldots, x_p$ in G and (not necessarily distinct) colors $c_1, c_2, \ldots, c_p$ in $\bigcup L(x)$ so that if $L'(x_i) = L(x_i) \cup \{c_i\}$ for $i = 1, 2, \ldots, p$ and $L'(x) = L(x)$ otherwise, then there is a list coloring of G with list assignment L'. We are interested in calculating $I(G, L)$, the smallest p for which G with L is p-addable. To give a simple example, consider the graph G and list assignment L shown in Figure 3.46. Then there is no list coloring because vertices b and d must get different colors in any list coloring. However, adding color 2 to $L(d)$ makes this list colorable. Thus, G with L is 1-addable and $I(G, L) = 1$.

Consider the graph $G = K_{10,10}$, which has two classes, A and B, with 10 vertices each and edges between every vertex x in A and every vertex y in B. On vertices of A, use the ten 2-element subsets of $\{1, 2, 3, 4, 5\}$ as sets $L(x)$, and do the same on B. We shall show that $I(K_{10,10}, L) = 4$. Suppose we add a color to some sets $L(x)$ to get $L'(x)$ so there is a list coloring for $K_{10,10}$ with L'. Suppose the list coloring uses r colors on A and s colors on B. Then, of course, $r + s \leq 5$. Now $\binom{5-r}{2}$ sets on A do not use these r colors, so at least $\binom{5-r}{2}$ sets on A need a color added. Similarly, at least $\binom{5-s}{2}$ sets on B need a color added. This number of additions

will work since all other sets on A have one of the r colors and similarly for B. It follows that

$$I(K_{10,10}, L) \leq \binom{5-r}{2} + \binom{5-s}{2},$$

with equality for some r and s. In fact, $r = 3$ and $s = 2$ give equality. Thus,

$$I(K_{10,10}, L) = \binom{5-3}{2} + \binom{5-2}{2} = 4.$$

Mahadev and Roberts [2003] prove that there are G and L so that $I(G, L)/|V(G)|$ is arbitrarily close to 1. This theorem has the interpretation that there are situations where almost everyone has to accept a color not on their original list! ■

Although the lists $L(x)$ may vary in size, let us consider the situation where the lists are all the same size. Then an important question for list coloring is: Given a graph G, what is the smallest c so that no matter what lists of size c are assigned to the vertices, you can always find a list coloring? If a graph G can be L-list colored for any lists of size c, we say that G is *c-choosable*. The smallest c for which G is c-choosable is defined to be the *choice number* of G and is denoted by $ch(G)$. As we saw in Figure 3.45, the graph $K_{3,3}$ is not 2-choosable. Even though the list assignment in Figure 3.45(b) can produce a list coloring, the list assignment in Figure 3.45(a) cannot. In Exercise 60 we ask the reader to show that $K_{3,3}$ is 3-choosable. Thus, $ch(K_{3,3}) = 3$.

Calculating the choice number of a graph is, like many of the other graph coloring problems we have studied, NP-complete. This was proven by Gravier [1996]. But like other coloring problems, bounds for the choice number in general and exact values for special classes of graphs can be obtained.

By definition, if a graph G is c-choosable, a list coloring exists for any list assignment of c colors to each vertex. In particular, a list coloring (which must also be a proper graph coloring) must exist if we assign the set $\{1, 2, \ldots, c\}$ to every vertex. Therefore, the chromatic number of G is at most c. Thus, we have shown that

$$ch(G) \geq \chi(G). \tag{3.10}$$

Equality in Equation (3.10) is sometimes attained for certain graphs. Consider the complete graph with n vertices, K_n. Recall that $\chi(K_n) = n$; every vertex is adjacent to every other vertex, thus requiring n colors. What is the choice number of K_n, $ch(K_n)$? It is easy to see that $ch(K_n) \leq n$. For when coloring a vertex by choosing an element from its list, the only colors not allowed are those that have been assigned to adjacent vertices, and there are $n - 1$ of these. Combining the observation that $ch(K_n) \leq n$ with Equation (3.10) shows that

$$ch(K_n) = n = \chi(K_n).$$

However, we certainly don't always attain equality in Equation (3.10). In fact, Erdős, Rubin, and Taylor [1979] have shown that the difference between the choice number and chromatic number can be arbitrarily large. See Alon [1993] and Kratochvíl, Tuza, and Voigt [1999] for further reading in this area.

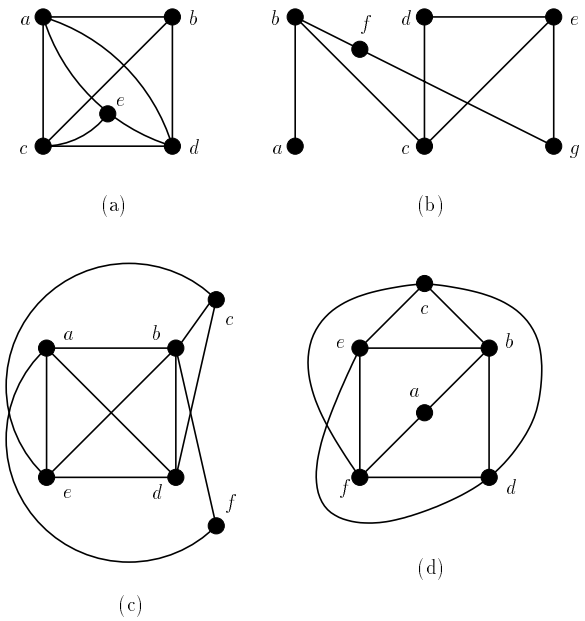


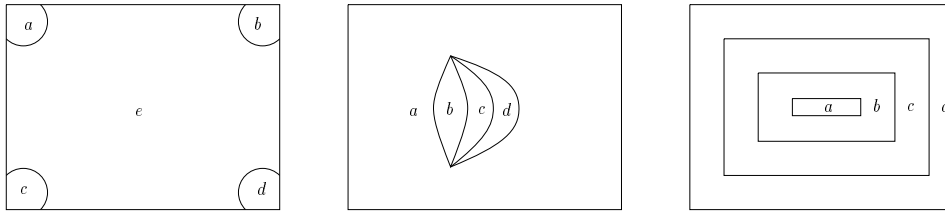
Figure 3.47: Graphs for exercises of Section 3.3.

EXERCISES FOR SECTION 3.3

1. Consider the following four tours of garbage trucks on the West Side of New York City. Tour 1 visits sites from 21st to 30th Streets, tour 2 visits sites from 28th to 40th Streets, tour 3 visits sites from 35th to 50th Streets, and tour 4 visits sites from 80th to 110th Streets. Draw the corresponding tour graph.
2. In Exercise 1, can the tours each be scheduled on Monday or Tuesday in such a way that no site is visited twice on the same day?
3. For each graph of Figure 3.47:

   (a) Determine if it is 3-colorable. (b) Determine its chromatic number $\chi(G)$.

4. A local zoo wants to take visitors on animal feeding tours and has decided on the following tours. Tour 1 visits the lions, elephants, and ostriches; tour 2 the monkeys, birds, and deer; tour 3 the elephants, zebras, and giraffes; tour 4 the birds, reptiles, and bears; and tour 5 the kangaroos, monkeys, and seals. If animals should not get fed more than once a day, can these tours be scheduled using only Monday, Wednesday, and Friday?
5. The following tours of garbage trucks in New York City are being considered (behind the mayor's back). Tour 1 picks up garbage at the Empire State Building, Madison Square Garden, and Pier 42 on the Hudson River. Tour 2 visits Greenwich Village, Pier 42, the Empire State Building, and the Metropolitan Opera House. Tour 3 visits Shea Stadium, the Bronx Zoo, and the Brooklyn Botanical Garden. Tour 4 goes to the Statue of Liberty and Pier 42; tour 5 to the Statue of Liberty, the



(a) (b) (c)

Figure 3.48: Maps.

New York Stock Exchange, and the Empire State Building; tour 6 to Shea Stadium, Yankee Stadium, and the Bronx Zoo; and tour 7 to the New York Stock Exchange, Columbia University, and the Bronx Zoo. Assuming that sanitation workers refuse to work more than three days a week, can these tours be partitioned so that no site is visited more than once on a given day?

6. The following committees need to have meetings scheduled.

$$\begin{aligned} A &= \{\text{Smith, Jones, Brown, Green}\} \\ B &= \{\text{Jones, Wagner, Chase}\} \\ C &= \{\text{Harris, Oliver}\} \\ D &= \{\text{Harris, Jones, Mason}\} \\ E &= \{\text{Oliver, Cummings, Larson}\}. \end{aligned}$$

Are three meeting times sufficient to schedule the committees so that no member has to be at two meetings simultaneously? Why?

7. In assigning frequencies to mobile radio telephones, a "zone" gets a frequency to be used by all vehicles in that zone. Two zones that interfere (because of proximity or for meteorological reasons) must get different frequencies. How many different frequencies are required if there are 6 zones, a, b, c, d, e, and f, and zone a interferes with zone b only; b with a, c, and d; c with b, d, and e; d with b, c, and e; e with c, d, and f; and f with e only?

8. In assigning work areas to workers, we want to be sure that if two such workers will interfere with each other, they will get different work areas. How many work areas are required if there are six workers, a, b, c, d, e, and f, and worker a interferes with workers b, e, and f; worker b with workers a, c, and f; worker c with b, d, and f; worker d with c, e, and f; e with a, d, and f; and f with all other workers?

9. In a given loop of a program, six variables arise. Variable A must be stored in steps 1 through 4, variable B in steps 3 through 6, variable C in steps 4 through 7, variable D in steps 6 through 9, variable E in steps 8 and 9, and variable F in steps 9 and 10. How many index registers are required for storage?

10. Find the graphs corresponding to the maps of Figure 3.48. Note that a single common point does not qualify as a common boundary.

11. Translate the map of Figure 3.49 into a graph G and calculate $\chi(G)$.

12. For each graph of Figure 3.50:

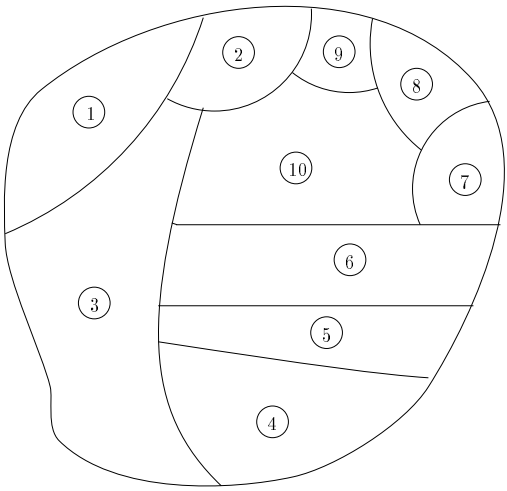
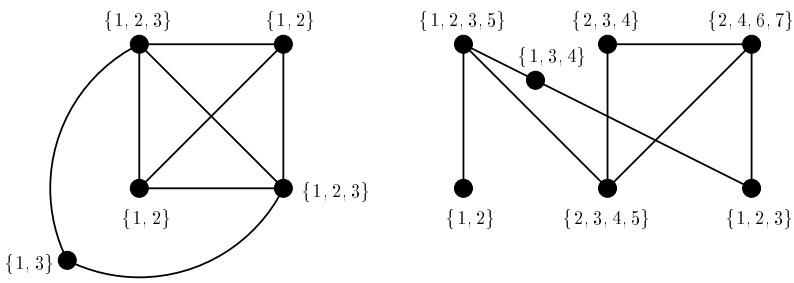


Figure 3.49: Map for exercises of Section 3.3.



(a) (b)

Figure 3.50: Graphs for exercises.

(a) Find $\omega(G)$. (b) Find $\alpha(G)$.

(c) Find $\chi(G)$ and verify the inequality (3.9).

13. Let G be any graph of 17 vertices and chromatic number 4.

    (a) Does G necessarily have an independent set of size 4?

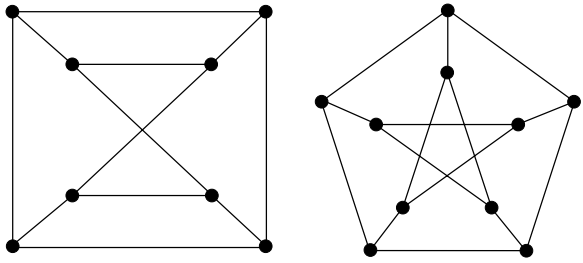
    (b) Does G necessarily have an independent set of size 5?

14. If the vertices of the circuit of length 11, Z_{11}, are colored in four colors, what can you say about the size of the largest set of vertices each of which gets the same color?

15. Give examples of graphs G so that:

    (a) $\chi(G) = \omega(G)$ (b) $\chi(G) > \omega(G)$

16. Let $\theta(G)$ be the size of the smallest set S of cliques of G so that every vertex is in some clique of S. What is the relation between $\theta(G)$ and $\chi(G)$?



(a) (b)

Figure 3.51: Two nonplanar graphs.

17. Can any graph containing K_5 as a subgraph be planar? Why?
18. Suppose that there are four houses and four utilities, and each house is joined by an edge to each utility. Is the resulting graph planar? Why?
19. Could K_6 come from a map? Why?
20. (a) Can K_3 be obtained by subdivision from K_2?
    (b) Are K_2 and K_3 homeomorphic?
21. Which of the graphs of Figure 3.47 are planar?
22. Prove that graph (a) of Figure 3.51 is nonplanar using:
    (a) Kuratowski's Theorem (Theorem 3.2) (b) Theorem 3.3
23. Repeat Exercise 22 for graph (b) of Figure 3.51.
24. For all graphs of Figure 3.47, find the graph obtained by contracting the edge $\{c, e\}$.
25. For all graphs of Figure 3.47, find the graph obtained by contracting edges $\{a, b\}$, $\{b, c\}$, and $\{c, d\}$ in that order.
26. Let $\deg(u)$, the *degree of vertex* u in a graph G, be defined to be the number of neighbors of u. Let $\Delta(G)$ be the maximum over all vertex degrees in G. Show that $\chi(G) \leq 1 + \Delta(G)$.
27. Give an example of a graph G such that $\chi(G) < 1 + \Delta(G)$, where $\Delta(G)$ is as defined in Exercise 26.
28. A graph G in which $\chi(G) = \omega(G)$ is called *weakly γ-perfect*. Give an example of a graph that is weakly γ-perfect and a graph that is not weakly γ-perfect.
29. Show that every 2-colorable graph is weakly γ-perfect.
30. Illustrate Algorithm 3.1 on graphs (c) and (g) of Figure 3.23.
31. Show that under Algorithm 3.1, vertex z gets colored red if $d(x, z)$ is odd and blue if $d(x, z)$ is even where x is the starting vertex.
32. Show that if a graph has an odd-length closed chain, it has an odd-length circuit.
33. Prove that $\chi(Z_p) = 3$ if and only if p is odd.
34. (a) Find a T-coloring for K_4 using $T = \{0, 1, 2\}$.
    (b) Find a T-coloring for Z_6 using $T = \{0, 2, 4\}$.
    (c) Find a T-coloring for the graph in Figure 3.52 using $T = \{0, 1, 2, 4, 8, 15\}$ and having order 6.

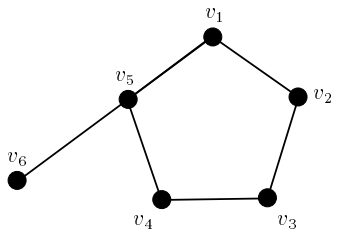


Figure 3.52: Graph for Section 3.3 exercises.

(d) Find a T-coloring for K_4 using $T = \{0, 1, 3\}$ and so that the span of the T-coloring is 7.

35. Find the T-span of K_4 when using the T-set $\{0, 1, 3, 4\}$.

36. When finding an efficient T-coloring, minimizing the order and minimizing the span may require different T-colorings. Recall that $\chi(Z_5) = 3$ and by Theorem 3.8, $\chi_T(Z_5) = 3$ for any T-set.

    (a) Find a T-coloring of Z_5 with the T-set $\{0, 1, 4, 5\}$ using 3 colors that has smallest possible span.

    (b) Find a T-coloring of Z_5 with the T-set $\{0, 1, 4, 5\}$ using 4 colors that has smallest possible span.

    (c) Find a T-coloring of Z_5 with the T-set $\{0, 1, 4, 5\}$ using 5 colors that has smallest possible span.

    (d) Is there *one* T-coloring of Z_5 with the T-set $\{0, 1, 4, 5\}$ that can minimize both order and span?

37. Prove that from a vertex coloring of a graph G using the colors $1, 2, \ldots, j$, a T-coloring will be produced by replacing color i with $i \cdot (t+1)$, where t represents the largest color in the given T-set.

38. Prove that the T-coloring in Figure 3.42 is most efficient with regard to span. That is, prove that the graph of Figure 3.42 cannot be T-colored, with $T = \{0, 3, 4, 7\}$, using the colors $1, 2, \ldots, n$ where $n \leq 10$.

39. Find a minimum set coloring for the graph in Figure 3.52 which has two colors assigned to each vertex of even subscript and three colors assigned to each vertex of odd subscript.

40. Find a 2-tuple coloring of Z_4 using 5 colors.

41. Recall that $\chi_k(G)$ is the smallest m so that G has a k-tuple coloring using m colors. Find:

    (a) $\chi_2(Z_4)$ (b) $\chi_2(Z_3)$ (c) $\chi_2(K_4)$

    (d) $\chi_3(Z_4)$ (e) $\chi_3(Z_3)$ (f) $\chi_3(K_4)$

42. Find $\chi_2(G)$ for G the graph of Figure 3.52.

43. If G is 2-colorable and has at least one edge, show that $\chi_m(G) = 2m$.

44. Prove that $\chi_3(Z_5) = 8$.

45. For each graph of Figure 3.50, determine if it is list colorable with the given list assignment.

46. Show that (G, L) is p-addable for some p if and only if

$$\left|\bigcup\{L(x) : x \in V\}\right| \geq \chi(G).$$

47. The graph $K_{\binom{m}{2},\binom{m}{2}}$ has two classes of $\binom{m}{2}$ vertices, A and B, and every vertex x in A is adjacent to every vertex y in B. Let L give all 2-element subsets of $\{1, 2, \ldots, m\}$ to vertices of A and similarly for vertices of B. Find

$$I\left(K_{\binom{m}{2},\binom{m}{2}}, L\right).$$

48. Let $K_{7,7}$ be defined analogously to $K_{10,10}$ in Example 3.23. Let $|L(x)| = 3$ for all x and $|\bigcup L(x)| = 6$. Show that $K_{7,7}$ with L is 1-addable.

49. Show that:

    (a) Z_3 is not 2-choosable.

    (b) Z_3 is 3-choosable.

50. (a) Determine the choice number for Z_4.

    (b) Determine the choice number for Z_n, n even.

51. Suppose that a k-tuple coloring of a graph with n vertices is given. If we consider the k-tuple coloring as a list assignment of the graph, how many different list colorings are possible?

52. A graph G is *k-edge-colorable* if you can color the edges with k colors so that two edges with a common vertex get different colors. Let $\chi'(G)$, the *edge chromatic number*, be the smallest k so that G is k-edge-colorable. State a relation between $\chi'(G)$ and the number $\Delta(G)$ defined in Exercise 26. (For applications of edge coloring, see Fiorini and Wilson [1977].)

53. If G has n vertices, show that

$$\frac{n}{\alpha(G)} \leq \chi(G) \leq n - \alpha(G) + 1.$$

54. A graph G is called *k-critical* if $\chi(G) = k$ but $\chi(G-u) < k$ for each vertex $u \in V(G)$.

    (a) Find all 2-critical graphs.

    (b) Give an example of a 3-critical graph.

    (c) Can you identify *all* 3-critical graphs?

55. If $G = (V, E)$ is a graph, its *complement* G^c is the graph with vertex set V and an edge between $x \neq y$ in V if and only if $\{x, y\} \notin E$.

    (a) Comment on the relationship between the clique number $\omega(G)$ and the vertex independence number $\alpha(G^c)$.

    (b) Recall that $\theta(G)$ is the smallest number of cliques which cover all vertices of G. Show that $\chi(G) = \theta(G^c)$.

(c) Say that G is *weakly α-perfect* if $\theta(G) = \alpha(G)$. Give an example of a graph that is weakly α-perfect and an example of a graph that is not weakly α-perfect.

56. G is said to be *γ-perfect* (*α-perfect*) if every generated subgraph of G is weakly γ-perfect (weakly α-perfect). Give examples of graphs that are:
    (a) γ-perfect (b) weakly γ-perfect but not γ-perfect
    (c) α-perfect (d) weakly α-perfect but not α-perfect

57. Lovász [1972a,b] shows that a graph G is γ-perfect if and only if it is α-perfect. Hence, a graph that is γ-perfect (or α-perfect) is called *perfect*. For more on perfect graphs and their many applications, see Golumbic [1980].
    (a) Show that it is not true that G is weakly γ-perfect if and only if G is weakly α-perfect.
    (b) Show that G is γ-perfect if and only if G^c is γ-perfect. (You may use Lovász's result.)

58. (Tutte [1954], Kelly and Kelly [1954], Zykov [1949]) Show that for any integer $k > 1$, there is a graph G such that $\omega(G) = 2$ and $\chi(G) = k$.

59. Prove Theorem 3.6.

60. Figure 3.45(a) gave a list assignment to $K_{3,3}$ showing that $K_{3,3}$ was not 2-choosable. This exercise gives a proof that $K_{3,3}$ is 3-choosable. Let L be a list assignment of $K_{3,3}$ where each list is of size 3.
    (a) Suppose that two nonadjacent vertices' lists share a common color. Show that an L-list coloring exists by using the common color.
    (b) Suppose that every pair of nonadjacent vertices' lists do not share a common color. Show that an L-list coloring exists.
    (c) Show why $ch(K_{3,3}) = 3$.

3.4 CHROMATIC POLYNOMIALS[18]

3.4.1 Definitions and Examples

Suppose that G is a graph and $P(G, x)$ counts the number of ways to color G in at most x colors. The related idea of counting the number of ways to color a map (see Exercise 3) was introduced by Birkhoff [1912] in an attack on the four-color conjecture—we discuss this below. The numbers $P(G, x)$ were introduced by Birkhoff and Lewis [1946]. Note that $P(G, x)$ is 0 if it is not possible to color G using x colors. Now $\chi(G)$ is the smallest positive integer x such that $P(G, x) \neq 0$. One of the primary reasons for studying the numbers $P(G, x)$ is to learn something about $\chi(G)$. In this section we study the numbers $P(G, x)$ in some detail, making heavy use of the counting techniques of Chapter 2.

[18] This section is not needed for what follows, except in Section 7.1.6. However, the reader is strongly encouraged to include it, for it provides many applications of the counting techniques of Chapter 2.

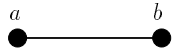


Figure 3.53: The graph K_2.

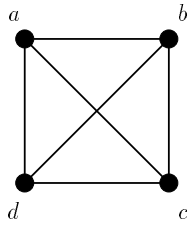


Figure 3.54: The graph K_4.

Table 3.4: Colorings of Graph K_2 of Figure 3.53 with Colors Red (R), Green (G), Blue (B), and Yellow (Y)

a	R	R	R	G	G	G	B	B	B	Y	Y	Y
b	G	B	Y	R	B	Y	R	G	Y	R	G	B

Consider first the graph K_2 shown in Figure 3.53. If x colors are available, any one of them can be used to color vertex a, and any one of the remaining $x-1$ colors can be used to color vertex b. Hence, by the product rule,

$$P(K_2, x) = x(x-1) = x^2 - x.$$

In particular,

$$P(K_2, 4) = 16 - 4 = 12.$$

The 12 ways of coloring K_2 in at most 4 colors are shown in Table 3.4.

Consider next the graph K_4 of Figure 3.54. If x colors are available, there are x choices of color for vertex a. For each of these choices, there are $x-1$ choices for vertex b, since b has an edge to a; for each of these there are $x-2$ choices for vertex c, since c has edges to both a and b; for each of these there are $x-3$ choices for vertex d, since d has edges to each of a, b, and c. Hence,

$$P(K_4, x) = x(x-1)(x-2)(x-3) = x^4 - 6x^3 + 11x^2 - 6x.$$

To color the graph of Figure 3.55 in x or fewer colors, there are x choices for vertex f. Then there are $x-1$ choices left for vertex a, and for each of these also $x-1$ choices for vertex b (since b may get the same color as a), and for each of these $x-1$ choices for vertex c, and for each of these $x-1$ choices for vertex d, and for each of these $x-1$ choices for vertex e. Hence,

$$P(G, x) = x(x-1)^5 = x^6 - 5x^5 + 10x^4 - 10x^3 + 5x^2 - x.$$

Let us turn now to the graph Z_4 of Figure 3.56. Here a and b must get different colors, but a and c could get the same color. Similarly, b and d could get the same color. And so on. If x colors are available, there are x choices of color for a. Then b and d can get either the same color or a different one. It is convenient to think of two cases.

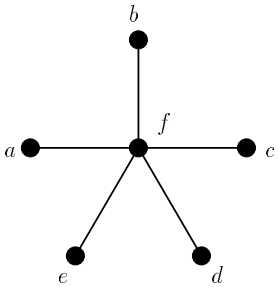


Figure 3.55: A graph.

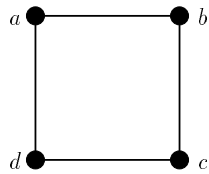


Figure 3.56: The graph Z_4.

Case 1. b and d get the same color.

Case 2. b and d get different colors.

In Case 1, c can get any of the colors not used for b and d. Hence, there are x choices for a, for each of these $x-1$ choices for the common color for b and d, and for each of these $x-1$ choices for the color for c. Hence, the number of colorings with x or fewer colors in which b and d get the same color is

$$x(x-1)^2.$$

In Case 2, there are x choices for a, $x-1$ for b, $x-2$ for d (since it must get a different color than b), and then $x-2$ choices for c (since it cannot receive either of the distinct colors used on b and d, but it can receive a's color). Hence, the number of colorings in which b and d get different colors is

$$x(x-1)(x-2)^2.$$

Since either Case 1 or Case 2 holds, the sum rule gives us

$$P(Z_4,x) = x(x-1)^2 + x(x-1)(x-2)^2 = x^4 - 4x^3 + 6x^2 - 3x. \tag{3.11}$$

The reader will notice that in each of the examples we have given, $P(G,x)$ is a polynomial in x. This will always be the case. Hence, it makes sense to call $P(G,x)$ the *chromatic polynomial*.

Theorem 3.8 $P(G,x)$ is always a polynomial.[19]

We will prove this theorem below. Recall that $\chi(G)$ is the smallest positive integer x such that $P(G,x) \neq 0$, that is, such that x is not a root of the polynomial $P(G,x)$. Thus, the chromatic number can be estimated by finding roots of a polynomial. Birkhoff's approach to the four-color problem was based on the idea of trying to characterize what polynomials were chromatic polynomials, in particular, of maps (or planar graphs), and then seeing if 4 was a root of any of these polynomials. To this day, the problem of characterizing the chromatic polynomials is not yet solved. We return to this problem below.

[19] This theorem was discovered for maps (equivalently, for graphs arising from maps) by Birkhoff [1912]. For all graphs, it is due to Birkhoff and Lewis [1946].

Example 3.24 Scheduling Meetings of Legislative Committees (Example 3.11 Revisited) Let us count the number of colorings of the graph G of Figure 3.28 using three or fewer colors. We start by computing $P(G, x)$. If there are x choices for the color of Education, there are $x - 1$ for the color of Housing, and then $x - 2$ for the color of Health. This leaves $x - 2$ choices for the color of Transportation, and $x - 2$ choices for the color of Environment, and finally, $x - 1$ choices for the color of Finance. Hence,

$$P(G, x) = x(x-1)^2(x-2)^3$$

and

$$P(G, 3) = 12.$$

This result agrees with the conclusion in our discussion of Example 1.4. There we described in Table 1.7 the 12 possible colorings in three or fewer colors. ■

Next, we state two simple but fundamental results about chromatic polynomials. One of these is about the graph I_n with n vertices and no edges, the *empty graph*.

Theorem 3.9 (a) If G is K_n, then

$$P(G, x) = x(x-1)(x-2)\cdots(x-n+1). \tag{3.12}$$

(b) If G is I_n, then $P(G, x) = x^n$.

Proof. (a) There are x choices for the color of the first vertex, $x - 1$ choices for the color of the second vertex, and so on.

(b) There are x choices for the color of each of the n vertices. Q.E.D.

The expression on the right-hand side of (3.12) will occur so frequently that it is convenient to give it a name. We call it $x^{(n)}$.

3.4.2 Reduction Theorems

A very common technique in combinatorics is to reduce large computations to a set of smaller ones. We employ this technique often in this book, and in particular in Chapter 6 when we study recurrence relations and reduce calculation of x_n to values of x_k for k smaller than n. This turns out to be a very useful technique for the computation of chromatic polynomials. In this subsection we develop and apply a number of reduction theorems that can be used to reduce the computation of any chromatic polynomial to that of a chromatic polynomial for a graph with fewer edges, and eventually down to the computation of chromatic polynomials of complete graphs and/or empty graphs.

We are now ready to state the first theorem.

Theorem 3.10 (The Two-Pieces Theorem) Let the vertex set of G be partitioned into disjoint sets W_1 and W_2, and let G_1 and G_2 be the subgraphs generated by W_1 and W_2, respectively. Suppose that in G, no edge joins a vertex of W_1 to a vertex of W_2. Then

$$P(G, x) = P(G_1, x)P(G_2, x).$$

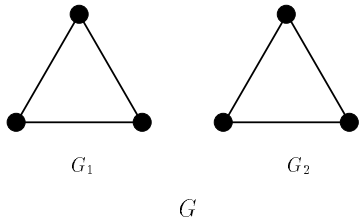


Figure 3.57: A graph with two pieces.

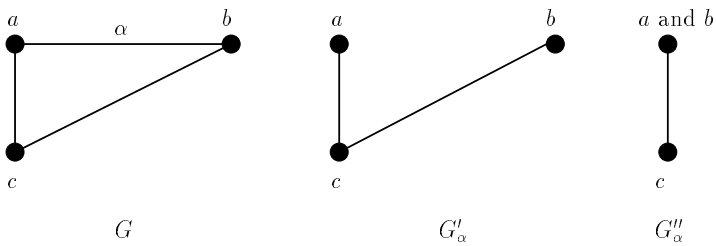


Figure 3.58: The graphs G'_α and G''_α.

Proof. If x colors are available, there are $P(G_1, x)$ colorings of G_1; for each of these there are $P(G_2, x)$ colorings of G_2. This is because a coloring of G_1 does not affect a coloring of G_2, since there are no edges joining the two pieces. The theorem follows by the product rule. Q.E.D.

To illustrate the theorem, we note that if G is the graph shown in Figure 3.57, then

$$P(G, x) = (x^{(3)})(x^{(3)}) = (x^{(3)})^2 = [x(x-1)(x-2)]^2.$$

To state our next reduction theorem, the crucial one, suppose that α is an edge of the graph G, joining vertices a and b. We define two new graphs from G. The graph G'_α is obtained by deleting the edge α but retaining the vertices a and b. The graph G''_α is obtained by identifying the two vertices a and b. In this case, the new combined vertex is joined to all those vertices to which either a or b were joined. (If both a and b were joined to a vertex c, only one of the edges from the combined vertex is included.) (In Section 3.3.2 we said that G''_α is obtained from G by contracting the edge α.) Figures 3.58 and 3.59 illustrate the two new graphs.

Theorem 3.11 (Fundamental Reduction Theorem[20])

$$P(G, x) = P(G'_\alpha, x) - P(G''_\alpha, x). \tag{3.13}$$

Proof. Suppose that we use up to x colors to color G'_α where edge α joins vertices a and b in G. Then either a and b receive different colors or a and b receive the same color. The number of ways of coloring G'_α so that a and b receive different

[20] This theorem is due to Birkhoff and Lewis [1946].

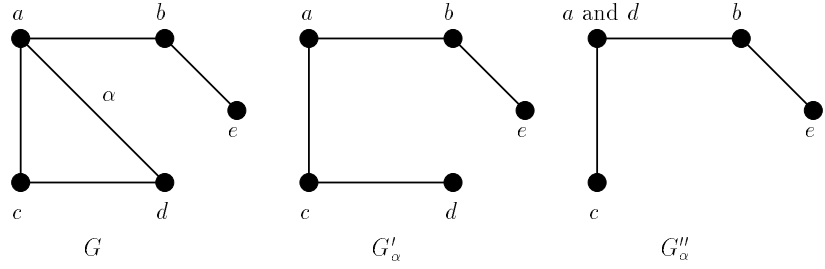


Figure 3.59: Another example of graphs G'_α and G''_α.

colors is simply the same as the number of ways of coloring G, that is, $P(G,x)$. The number of ways of coloring G'_α so that a and b get the same color is the same as the number of ways of coloring G''_α, namely $P(G''_\alpha,x)$. For we know that in G''_α, a and b would get the same color, and moreover the joint vertex a and b is forced to get a different color from that given to a vertex c if and only if one of a and b, and hence both, is forced to get a different color from that given to vertex c. The result now follows by the sum rule:

$$P(G'_\alpha,x) = P(G,x) + P(G''_\alpha,x). \qquad \text{Q.E.D.}$$

To illustrate the theorem, let us consider the graph G of Figure 3.60 and use the Fundamental Reduction Theorem to calculate its chromatic polynomial. We choose the edge between vertices 1 and 3 to use as α and so obtain G'_α and G''_α as shown in the figure. Now the graph G''_α of Figure 3.60 is the complete graph K_2, and hence by Theorem 3.9 we know that

$$P(G''_\alpha,x) = x(x-1). \tag{3.14}$$

The graph G'_α has two pieces, K_1 and K_2. By the Two-Pieces Theorem,

$$P(G'_\alpha,x) = P(K_1,x)P(K_2,x). \tag{3.15}$$

By Theorem 3.9, the first expression on the right-hand side of (3.15) is x and the second expression is $x(x-1)$. Hence,

$$P(G'_\alpha,x) = x \cdot x(x-1) = x^2(x-1). \tag{3.16}$$

Substituting (3.14) and (3.16) into (3.13), we obtain

$$\begin{aligned} P(G,x) &= x^2(x-1) - x(x-1) \\ &= x(x-1)(x-1) \\ &= x(x-1)^2. \end{aligned}$$

This expression could of course have been derived directly. However, it is a good illustration of the use of the Fundamental Reduction Theorem. Incidentally, applying the Fundamental Reduction Theorem again to $G''_\alpha = K_2$, we have

$$P(G''_\alpha,x) = P(K_2,x) = P(I_2,x) - P(I_1,x), \tag{3.17}$$

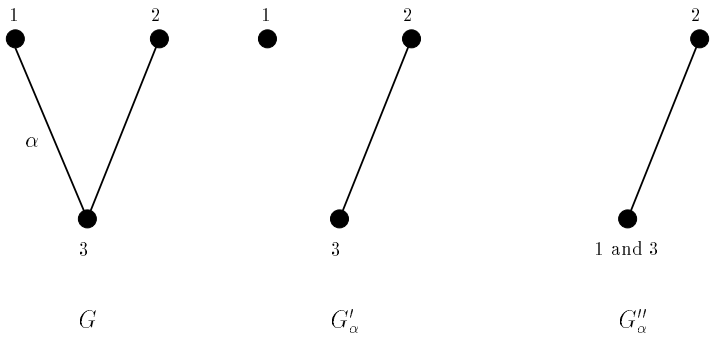


Figure 3.60: An application of the Fundamental Reduction Theorem.

as is easy to verify. Also, since $K_1 = I_1$, (3.15) and (3.17) give us

$$P(G'_\alpha, x) = P(I_1, x)[P(I_2, x) - P(I_1, x)]. \tag{3.18}$$

Finally, plugging (3.17) and (3.18) into (3.13) gives us

$$P(G, x) = P(I_1, x)[P(I_2, x) - P(I_1, x)] - [P(I_2, x) - P(I_1, x)]. \tag{3.19}$$

We have reduced $P(G, x)$ to an expression (3.19) that requires only knowledge of the polynomials $P(I_k, x)$ for different values of k.

As a second example, let us use the Fundamental Reduction Theorem to calculate $P(K_3, x)$. Note from Figure 3.61 that if $H = K_3$, then H'_α is the graph G of Figure 3.60 and H''_α is K_2. Thus,

$$\begin{aligned} P(K_3, x) &= P(G, x) - P(K_2, x) \\ &= x(x-1)^2 - x(x-1), \end{aligned}$$

where the first expression arises from our previous computation, and the second from the formula for $P(K_2, x)$. Simplifying, we obtain

$$\begin{aligned} P(K_3, x) &= x(x-1)[(x-1) - 1] \\ &= x(x-1)(x-2) \\ &= x^{(3)}, \end{aligned}$$

which agrees with Theorem 3.9.

The reader should note that each application of the Fundamental Reduction Theorem reduces the number of edges in each graph that remains. Hence, by repeated uses of the Fundamental Reduction Theorem, we must eventually end up with graphs with no edges, namely graphs of the form I_k. We illustrated this point with our first example. In any case, this shows that Theorem 3.8 must indeed hold, that is, that $P(G, x)$ is always a polynomial in x. For $P(I_k, x) = x^k$. Hence, we eventually reduce $P(G, x)$ to an expression that is a sum, difference, or product of terms each of which is of the form x^k, for some k. (The proof may be formalized by arguing by induction on the number of edges in the graph.)

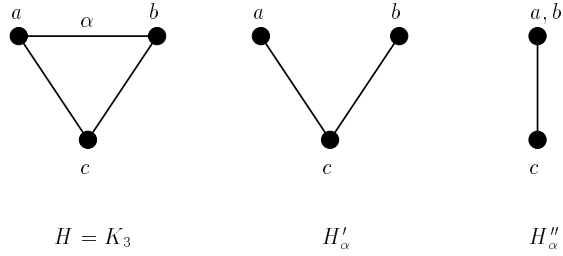


Figure 3.61: A second application of the Fundamental Reduction Theorem.

Figure 3.62 gives one final illustration of the Fundamental Reduction Theorem. In that figure, we simply draw a graph to stand for the chromatic polynomial of the graph. The edge α is indicated at each step.

3.4.3 Properties of Chromatic Polynomials[21]

We have already pointed out that one of Birkhoff's hopes in introducing chromatic polynomials was to be able to tell what polynomials were chromatic and then to study the roots of those polynomials that were chromatic. In this section we study the properties of the chromatic polynomials. We shall discover that we can learn a great deal about a graph by knowing its chromatic polynomial. Exercises 13–16 outline the proofs of the theorems stated here and present further properties of chromatic polynomials.

The first theorem summarizes some elementary properties of chromatic polynomials. These can be checked for all the examples of Sections 3.4.1 and 3.4.2.

Theorem 3.12 (Read [1968]) Suppose that G is a graph of n vertices and

$$P(G,x) = a_p x^p + a_{p-1}x^{p-1} + \cdots + a_1 x + a_0.$$

Then:

(a) The degree of $P(G,x)$ is n, that is, $p = n$.

(b) The coefficient of x^n is 1, that is, $a_n = 1$.

(c) The constant term is 0, that is, $a_0 = 0$.

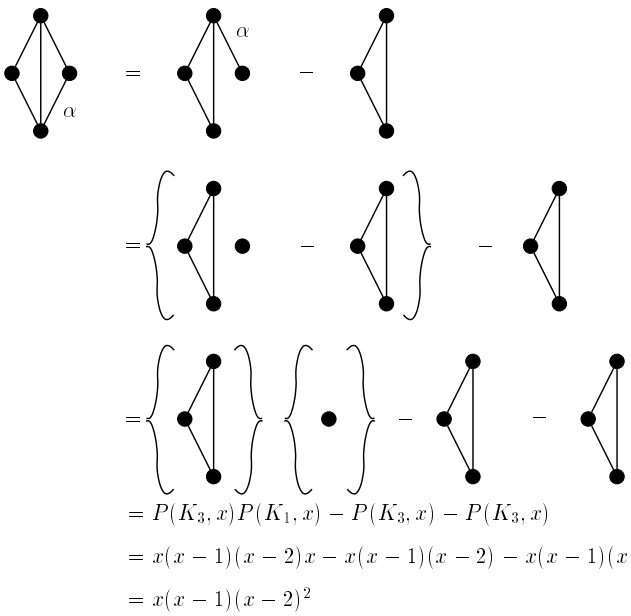
(d) Either $P(G,x) = x^n$ or the sum of the coefficients in $P(G,x)$ is 0.

Theorem 3.13 (Whitney [1932]) $P(G,x)$ is the sum of consecutive powers of x and the coefficients of these powers alternate in sign. That is, for some I,

$$P(G,x) = x^n - \alpha_{n-1}x^{n-1} + \alpha_{n-2}x^{n-2} \mp \cdots \pm \alpha_0, \tag{3.20}$$

with $\alpha_i > 0$ for $i \geq I$ and $\alpha_i = 0$ for $i < I$.

[20] This subsection may be omitted.



$$= P(K_3, x)P(K_1, x) - P(K_3, x) - P(K_3, x)$$
$$= x(x-1)(x-2)x - x(x-1)(x-2) - x(x-1)(x-2)$$
$$= x(x-1)(x-2)^2$$

Figure 3.62: Another application of the Fundamental Reduction Theorem.

Theorem 3.14 (Read [1968]) In $P(G, x)$, the absolute value of the coefficient of x^{n-1} is the number of edges of G.

Unfortunately, the properties of chromatic polynomials that we have listed in Theorems 3.12 and 3.13 do not characterize chromatic polynomials. There are polynomials $P(x)$ that satisfy all these conditions but that are not chromatic polynomials of any graph. For instance, consider

$$P(x) = x^4 - 4x^3 + 3x^2.$$

Note that the coefficient of x^n is 1, the constant term is 0, the sum of the coefficients is 0, and the coefficients alternate in sign until, from the coefficient of x^1 and on, they are 0. However, $P(x)$ is not a chromatic polynomial of any graph. If it were, the number of vertices of the graph would have to be 4, by part (a) of Theorem 3.12. The number of edges would also have to be 4, by Theorem 3.14. No graph with four vertices and four edges has this polynomial as its chromatic polynomial, as is easy to check. Or see Lehmer [1985], who found the chromatic polynomials for all graphs on 6 or fewer vertices.

For further results on chromatic polynomials, see, for example, Jensen and Toft [1995], Liu [1972], or Read [1968].

EXERCISES FOR SECTION 3.4

1. Find the chromatic polynomial of each graph in Figure 3.63.

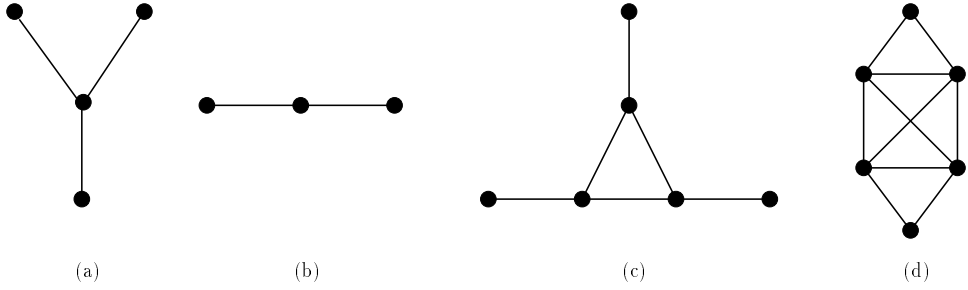


Figure 3.63: Graphs for exercises of Section 3.4.

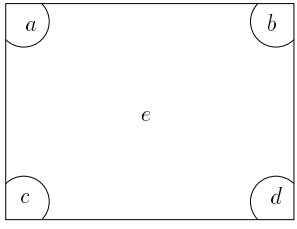


Figure 3.64: Map for exercises of Section 3.4.

2. For each graph in Figure 3.63, find the number of ways to color the graph in at most three colors.
3. The chromatic polynomial $P(M, x)$ of a map M is the number of ways to color M in x or fewer colors. Find $P(M, x)$ for the map of Figure 3.64.
4. Let L_n be the graph consisting of a simple chain of n vertices. Find a formula for $P(L_n, x)$.
5. For each of the graphs of Figure 3.65, find the chromatic polynomial using reduction theorems. (You may reduce to graphs with previously known chromatic polynomials.)
6. If G is the graph of Figure 3.65(a), express $P(G, x)$ in terms of polynomials $P(I_k, x)$ for various k.
7. If L_n is as defined in Exercise 4, what is the relation among $P(Z_n, x), P(Z_{n-1}, x)$, and $P(L_n, x)$?
8. Use reduction theorems to find the chromatic polynomial of the map of Figure 3.66 (see Exercise 3). You may use the result of Exercise 4.
9. Let $N(G, x)$ be the number of ways of coloring G in exactly x colors. Find $N(G, x)$ for each of the following graphs G and the given values of x.

   (a) $Z_5, x = 4$ (b) $K_5, x = 6$ (c) $L_5, x = 3$

   (d) Find an expression for $P(G, x)$ in terms of the numbers $N(G, r)$ for $r \leq x$.

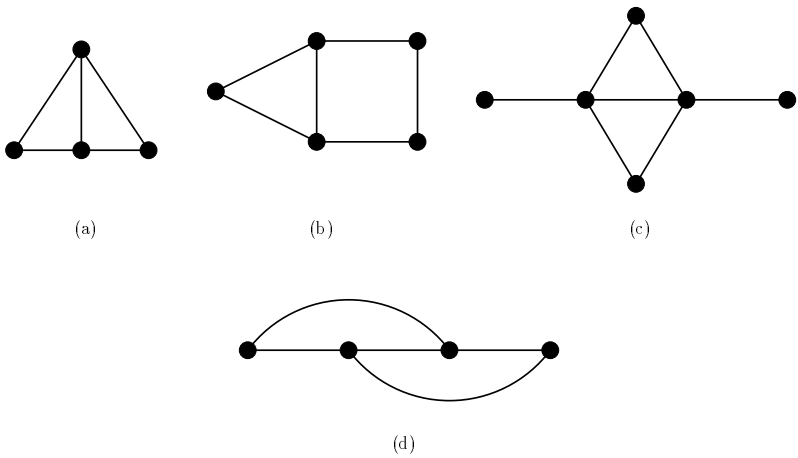


Figure 3.65: Graphs for exercises of Section 3.4.

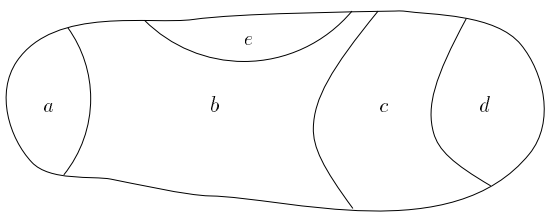


Figure 3.66: Map for exercises of Section 3.4.

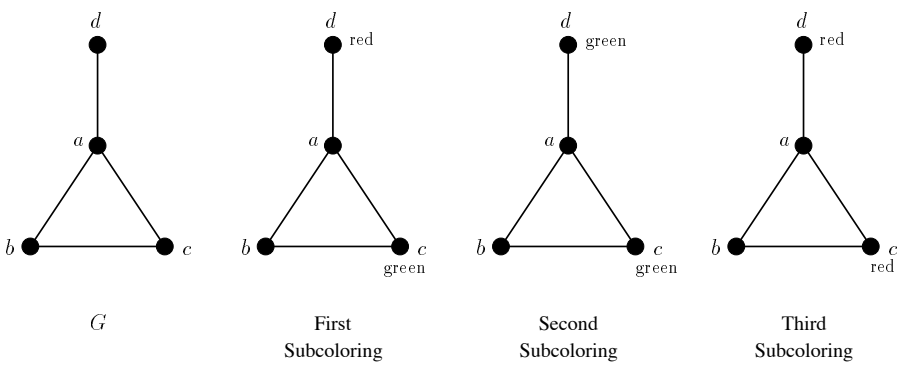


Figure 3.67: A graph G and three subcolorings of G.

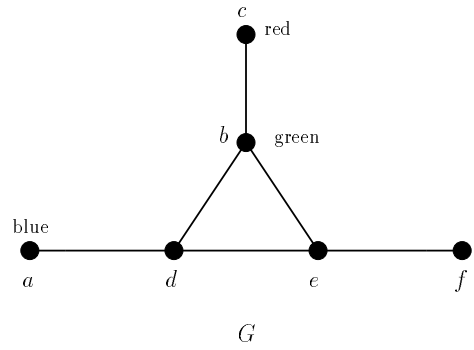


Figure 3.68: A graph and a subcoloring.

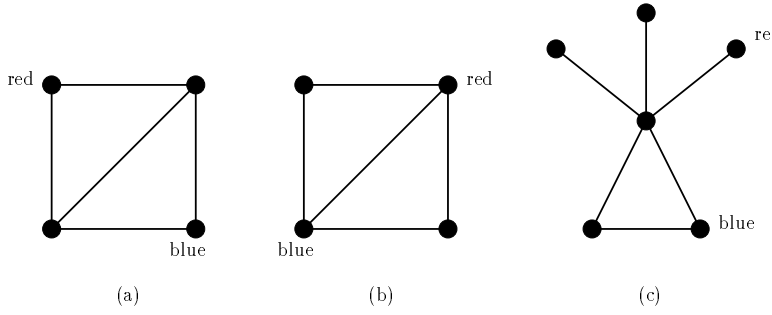


Figure 3.69: Graphs and subcolorings.

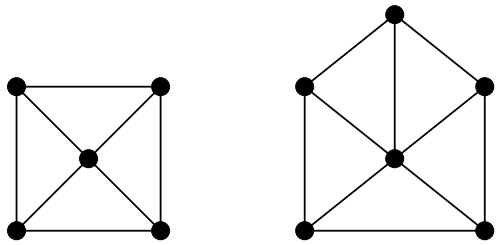
10. If we have a coloring of some vertices of G, we call this a *subcoloring* of G. A coloring of all the vertices of G that agrees with a subcoloring of some of the vertices of G is called an *extension* of the subcoloring. Figure 3.67 shows a graph G and three subcolorings of G. If there is just one more color available, say blue, then the first subcoloring can be extended to G in just one way, namely by coloring vertex a blue and vertex b red. However, the second subcoloring can be extended to a subcoloring of G in two ways, by coloring a blue and b red, or by coloring a red and b blue.

    (a) How many extensions are there of the third subcoloring shown in Figure 3.67?

    (b) Consider the graph G of Figure 3.68 and the subcoloring of the vertices a, b, and c shown in that figure. How many extensions are there of this subcoloring to all of G if only the colors green, red, blue, and brown are available?

    (c) Consider the graphs of Figure 3.69 and the subcolorings shown in that figure. Find the number of extensions of each subcoloring to a coloring of the whole graph in three or fewer colors, if red, blue, and green are the three colors available.

11. Repeat Exercise 10(c), finding the number of extensions using at most x colors, $x \geq 3$.



W_4 W_5

Figure 3.70: The wheels W_4 and W_5.

12. Show that the following could not be chromatic polynomials.
    (a) $P(x) = x^8 - 1$
    (b) $P(x) = x^5 - x^3 + 2x$
    (c) $P(x) = 2x^3 - 3x^2$
    (d) $P(x) = x^3 + x^2 + x$
    (e) $P(x) = x^3 - x^2 + x$
    (f) $P(x) = x^4 - 3x^3 + 3x^2$
    (g) $P(x) = x^9 + x^8 - x^7 - x^6$

13. Prove parts (c) and (d) of Theorem 3.12.

14. Prove parts (a) and (b) of Theorem 3.12 together, by induction on the number e of edges and by use of the Fundamental Reduction Theorem.

15. Prove Theorem 3.13 by induction on the number e of edges and by use of the Fundamental Reduction Theorem.

16. Prove Theorem 3.14 from Theorem 3.13, by induction on the number e of edges.

17. Prove that $P(G, q) \leq q(q-1)^{n-1}$ for any positive integer q, if G is connected with n vertices.

18. Prove that $P(G, \lambda) \neq 0$ for any $\lambda < 0$.

19. (a) If G has k connected components, show that the smallest i such that x^i has a nonzero coefficient in $P(G, x)$ is at least k.
    (b) Is this smallest i necessarily equal to k? Why?
    (c) Prove that if $P(G, x) = x(x-1)^{n-1}$, then G is connected.

20. Suppose that W_n is the *wheel* of $n+1$ vertices, that is, the graph obtained from Z_n by adding one vertex and joining it to all vertices of Z_n. W_4 and W_5 are shown in Figure 3.70. Find $P(W_n, x)$. You may leave your answer in terms of $P(Z_n, x)$.

21. If Z_n is the circuit of length n:
    (a) Show that for $n \geq 3, (-1)^n[P(Z_n, x) - (x-1)^n]$ is constant, independent of n.
    (b) Solve for $P(Z_n, x)$ by evaluating the constant in part (a).

22. Suppose that H is a clique of G and that we have two different subcolorings of H in at most x colors. Show that the number of extensions to a coloring of G in at most x colors is the same for each subcoloring.

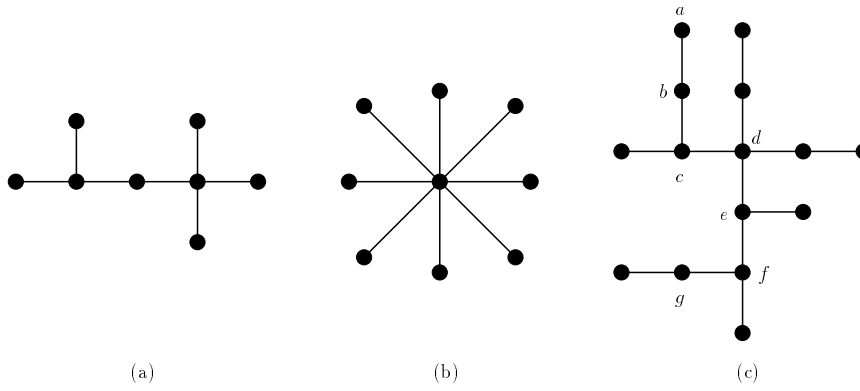


Figure 3.71: Some trees.

23. The following is another reduction theorem. Suppose that H and K are generated subgraphs of G, with $V(G) = V(H) \cup V(K)$ and $E(G) = E(H) \cup E(K)$, and that $V(H) \cap V(K)$ is a clique of G of p vertices. Then

$$P(G,x) = \frac{P(H,x)P(K,x)}{x^{(p)}}.$$

(a) Illustrate this result on the graph G of Figure 3.68 if H is the subgraph generated by $\{a,d,e,f\}$ and K the subgraph generated by $\{c,b,d,e\}$. (Disregard the subcoloring.)

(b) Make use of the result of Exercise 22 to prove the theorem.

24. If the chromatic polynomial $P(K_n,x)$ is expanded out, the coefficient of x^k is denoted $s(n,k)$ and called a *Stirling number of the first kind*. Exercises 24–26 will explore these numbers. Find:

(a) $s(n,0)$ (b) $s(n,n)$ (c) $s(n,1)$ (d) $s(n,n-1)$

25. Show that

$$|s(n,k)| = (n-1)|s(n-1,k)| + |s(n-1,k-1)|.$$

26. Use the result in Exercise 25 to describe how to compute Stirling numbers of the first kind by a method similar to Pascal's triangle and apply your ideas to compute $s(6,3)$.

3.5 TREES

3.5.1 Definition of a Tree and Examples

In this section and the next we consider one of the most useful concepts in graph theory, that of a tree. A *tree* is a graph T that is connected and has no circuits. Figure 3.71 shows some trees.

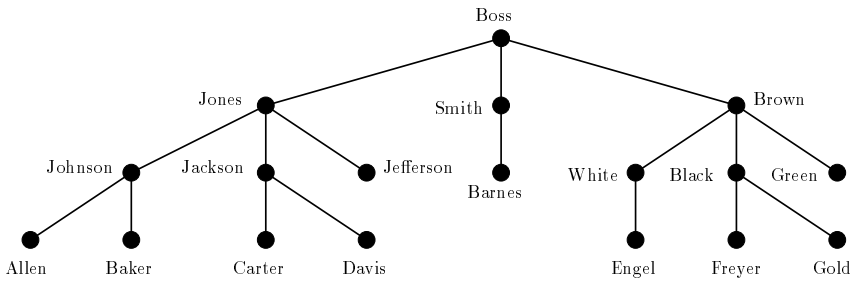


Figure 3.72: A telephone chain.

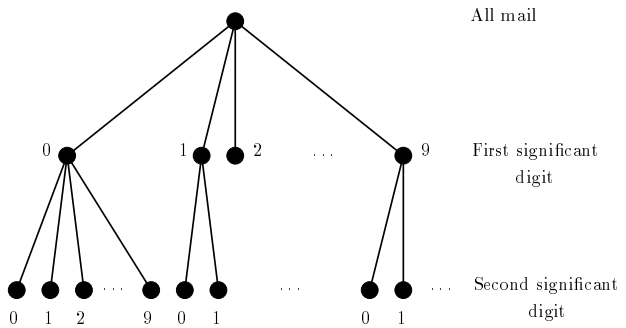


Figure 3.73: Part of the sort tree for sorting mail by ZIP code.

Example 3.25 Telephone Trees Many companies and other organizations have prearranged telephone chains to notify their employees in case of an emergency, such as a snowstorm that will shut down the company. In such a telephone chain, a person in charge makes a decision (e.g., to close because of snow) and calls several designated people, who each call several designated people, who each call several designated people, and so on. We let the people in the company be vertices of a graph and include an edge from a to b if a calls b (we include an undirected edge even though calling is not symmetric). The resulting graph is a tree, such as that shown in Figure 3.72. ■

Example 3.26 Sorting Mail[22] Mail intended for delivery in the United States carries on it, if the sender follows Postal Service instructions, a ZIP code consisting of a certain number of decimal digits. Mail arriving at a post office is first sorted into 10 piles by the most significant digit. Then each pile is divided into 10 piles by the next most significant digit. And so on. The sorting procedure can be summarized by a tree, part of which is shown in Figure 3.73. To give a simpler example, suppose that we sort mail within a large organization by giving a mail code not unlike a ZIP code, but consisting of only three digits, each being 0 or 1. Then the sort tree is shown in Figure 3.74. ■

[22] This example is based on Deo [1974].

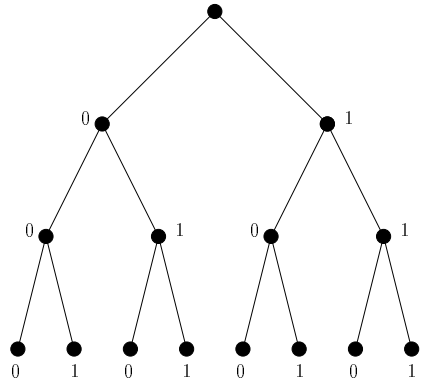


Figure 3.74: The sort tree for mail coded by three binary digits.

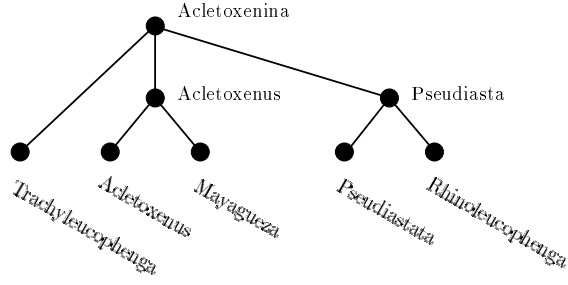


Figure 3.75: A (partial) phylogenetic tree of genera in the Drosophilidae (Diptera). (Based on data in Grimaldi [1990].)

Example 3.27 Phylogenetic Trees A widely held view in modern biology is that all existing organisms derive from common ancestors and that new species arise when a given population splits into two or more populations by some process, such as mutation. Trees are often used to model the evolutionary process in which species evolve into new species. The vertices of the tree are the species and an edge leads from a species to an immediate descendant. Figure 3.75 shows an evolutionary tree. Evolutionary trees also arise in subjects other than biology. For example, in linguistics, we study the evolution of words over time, through "mutations" in spelling. A common problem in phylogeny, the scientific discipline that studies evolution, is to reconstruct an evolutionary or *phylogenetic tree* from information about presently existing species. We have more to say about this in Section 3.5.7. ■

In this section and the next we consider a variety of other applications of trees, in particular emphasizing applications to organic chemistry, phylogenetic tree reconstruction, and to searching and sorting problems in computer science.

3.5.2 Properties of Trees

A fundamental property of trees is obtained by noting the relationship between the number of vertices and the number of edges of a tree.

Theorem 3.15 If T is a tree of n vertices and e edges, then $n = e + 1$.

Theorem 3.15 is illustrated by any of the trees we have drawn. It will be proved in Section 3.5.3.

Note that the property $n = e + 1$ does not characterize trees. There are graphs that have $n = e + 1$ and are not trees (Exercise 6). However, we have the following result, which will be proved in Section 3.5.5.

Theorem 3.16 Suppose that G is a graph of n vertices and e edges. Then G is a tree if and only if G is connected and $n = e + 1$.

We next note an interesting consequence of Theorem 3.15. The result will be used in counting the number of trees.

Theorem 3.17 If T is a tree with more than one vertex, there are at least two vertices of degree 1.

Proof. Since T is connected, every vertex must have degree ≥ 1. (Why?) Now by Theorems 3.15 and 3.1, $\sum \deg(u) = 2e = 2n - 2$. If $n - 1$ vertices had degree ≥ 2, the sum of the degrees would have to be at least $2(n-1) + 1 = 2n - 1$, which is greater than $2n - 2$. Thus, no more than $n - 2$ vertices can have degree ≥ 2. Q.E.D.

3.5.3 Proof of Theorem 3.15[23]

Theorem 3.18 In a tree T, if x and y are any two vertices, then there is one and only one simple chain joining x and y.

Proof. We know there is a chain between x and y, since T is connected. Recall that a simple chain has no repeated vertices. A shortest chain between x and y must be a simple chain. For if $x, u, \ldots, v, \ldots, v, \ldots, w, y$ is a shortest chain between x and y with a repeated vertex v, we can skip the part of the chain between repetitions of v, thus obtaining a shorter chain from x to y. Hence, there is a simple chain joining x and y. Suppose next that $C_1 = x, x_1, x_2, \ldots, x_k, y$ is a shortest simple chain joining x and y. We show that there can be no other simple chain joining x and y. Suppose that C_2 is such a chain. Let x_{p+1} be the first vertex of C_1 on which C_1 and C_2 differ, and let x_q be the next vertex of C_1 following x_p on which C_1 and C_2 agree. Then we obtain a circuit by following C_1 from x_p to x_q and C_2 back from x_q to x_p, which contradicts the fact that T is a tree. Q.E.D.

[23] This subsection may be omitted.

To illustrate this theorem, we note that in tree (c) of Figure 3.71, the unique simple chain joining vertices a and g is given by a, b, c, d, e, f, g. Now we return to the proof of Theorem 3.15.

Proof of Theorem 3.15. The proof is by induction on n. If $n = 1$, the result is trivial. The only tree with one vertex has no edges. Now suppose that the result is true for all trees of fewer than n vertices and that tree T has n vertices. Pick any edge $\{u, v\}$ in T. By Theorem 3.18, u, v is the only simple chain between u and v. If we take edge $\{u, v\}$ away from G (but leave vertices u and v), we get a new graph H. Now in H, there can be no chain between u and v. For if there is, it is easy to find a simple chain between u and v. But this is also a simple chain in G, and G has only one simple chain between u and v.

Now since H has no chain between u and v, H is disconnected. It is not difficult to show (Exercise 22) that H has exactly two connected components. Call these H_1 and H_2. Since each of these is connected and can have no circuits (why?), each is a tree. Moreover, each has fewer vertices than G. By the inductive hypothesis, if n_i and e_i are the number of vertices and edges of H_i, we have $n_1 = e_1 + 1$ and $n_2 = e_2 + 1$. Now $n = n_1 + n_2$ and $e = e_1 + e_2 + 1$ (add edge $\{u, v\}$). We conclude that

$$n = n_1 + n_2 = (e_1 + 1) + (e_2 + 1) = (e_1 + e_2 + 1) + 1 = e + 1. \qquad \text{Q.E.D.}$$

3.5.4 Spanning Trees[24]

Suppose that $G = (V, E)$ is a graph and $H = (W, F)$ is a subgraph. We say H is a *spanning subgraph* if $W = V$. A spanning subgraph that is a tree is called a *spanning tree*. For example, in the graph G of Figure 3.76, H and K as shown in the figure are spanning subgraphs because they have the same vertices as G. K is a spanning tree. Spanning trees have a wide variety of applications in combinatorics, which we shall investigate shortly. Analysis of an electrical network reduces to finding all spanning trees of the corresponding graph (see Deo [1974]). Spanning trees can also be used, through the program digraph (see Example 3.6), to estimate program running time (Deo [1974], p. 442]). They arise in connection with seriation problems in political science and archaeology (Roberts [1979], Wilkinson [1971]). They also form the basis for a large number of algorithms in network flows and the solution of minimal cost problems in operations research (Chapter 13). Graham and Hell [1985] mention applications of spanning trees to design of computer and communication networks, power networks, leased-line telephone networks, wiring connections, links in a transportation network, piping in a flow network, network reliability, picture processing, automatic speech recognition, clustering and classification problems, and so on. Their paper also gives many references to the literature of spanning trees and their applications.

We now give several applications of spanning trees in more detail.

[24] This subsection may be omitted until just before Section 11.1 or Section 13.1.

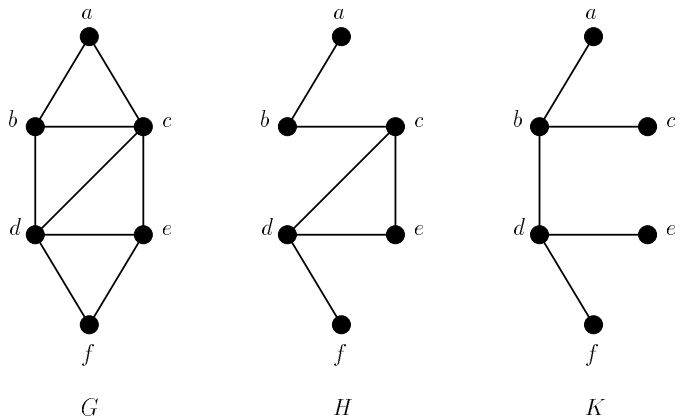


Figure 3.76: H and K are spanning subgraphs of G, and K is also a spanning tree.

Example 3.28 Highway Construction Suppose that a number of towns spring up in a remote region where there are no good highways. It is desired to build enough highways so that it is possible to travel from any of the towns to any other of the towns by highway either directly or by going through another town. Let the towns be vertices of a graph G and join each pair of towns by an edge. We wish to choose a subset F of the edges of G representing highways to be built, so that if we use the vertices of G and the edges of F, we have a connected graph. Thus, we seek a connected, spanning subgraph of G. If we wish to minimize costs, then certainly the highways we choose to build correspond to a connected spanning subgraph H of G with the property that removal of any edge leads to a spanning subgraph that is no longer connected. But in any connected spanning subgraph H of G, if there is a circuit, removal of any edge of the circuit cannot disconnect H. (Why?) Thus, H is connected and has no circuits; that is, H is a spanning tree. If, in addition, each edge of G has a weight or number on it, representing the cost of building the corresponding highway, then we wish to find a spanning tree F which is *minimum* in the sense that the sum of the weights on its edges is no larger than that on any other spanning tree. We study algorithms for finding minimum spanning trees in Section 13.1. We shall sometimes also be interested in finding maximum spanning trees. The procedure for finding them is analogous to the procedure for finding minimum spanning trees. It should be noted that a similar application of minimum spanning trees arises if the links are gas pipelines, electrical wire connections, railroads, sewer lines, and so on. ■

The approach taken in Example 3.28 assumes that we do not allow highways to link up at points other than the towns in question. This is implicit in the assumption that we want to go either directly between two towns or link them up through another town. If we allowed highways to link up at points other than the towns in question, additional savings could possibly be realized when minimizing

costs. These cost-saving "additional" points are called *Steiner points*. For more information on Steiner points, see Bern and Graham [1989] or Cieslik [1998].

Example 3.29 Telephone Lines In a remote region, isolated villages are linked by road, but there is not yet any telephone service. We wish to put in telephone lines so that every pair of villages is linked by a (not necessarily direct) telephone connection. It is cheapest to put the lines in along existing roads. Along which stretches of road should we put telephone lines so as to make sure that every pair of villages is linked and the total number of miles of telephone line (which is probably proportional to the total cost of installation) is minimized? Again we seek a minimum spanning tree. (Why?) ■

Example 3.30 Computer Hardware (Ahuja, Magnanti, and Orlin [1993]) A digital computer has a variety of components to be connected by high-frequency circuitry. To reduce capacitance and delay line effects, it is important to minimize the length of wires between components. We need to connect all components and so seek a minimum spanning tree. ■

Example 3.31 Data Storage (Ahuja, Magnanti, and Orlin [1993], Kang, *et al.* [1977]) In many applied problems, data are stored in a two-dimensional array. When the rows of the array have many similar entries and differ only in a few places, we can save memory by storing only one row completely and then storing the differences of the other rows from this *reference row*. Build a complete graph G with the weight on the edge between rows i and j being the number of changes in entries required to switch from row i to row j, or vice versa. We can minimize data storage by choosing as the reference row that row with the minimum amount of data to store and then finding a minimum spanning tree in the graph G. Why? ■

Example 3.32 Measuring the Homogeneity of Bimetallic Objects[25] Minimum spanning trees have found applications at the U.S. National Bureau of Standards and elsewhere, in determining to what extent a bimetallic object is homogeneous in composition. Given such an object, build a graph by taking as vertices a set of sample points in the material. Measure the composition at each sample point and join physically adjacent sample points by an edge. Place on the edge $\{x, y\}$ a weight equal to the physical distance between sample points x and y multiplied by a homogeneity factor between 0 and 1. This factor is 0 if the composition of the two points is exactly alike and 1 if it is dramatically opposite, and otherwise is a number in between. In the resulting graph, find a minimum spanning tree. The sum of the weights in this tree is 0 if all the sample points have exactly the same composition. A high value says that the material is quite inhomogeneous. This statistic, the sum of the weights of the edges in a minimum spanning tree, is a very promising statistic in measuring homogeneity. According to Goldman [1981], it will probably come into standard use over an (appropriately extended) period of time. ■

[25] From Filliben, Kafadar, and Shier [1983], Goldman [1981], and Shier [1982].

We now present one main result about spanning trees.

Theorem 3.19 A graph G is connected if and only if it has a spanning tree.

Proof. Suppose that G is connected. Then there is a connected spanning subgraph H of G with a minimum number of edges. Now H can have no circuits. For if C is a circuit of H, removal of any edge of C (without removing the corresponding vertices) leaves a spanning subgraph of G that is still connected and has one less edge than H. This is impossible by choice of H. Thus, H has no circuits. Also, by choice, H is connected. Thus, H is a spanning tree.

Conversely, if G has a spanning tree, it is clearly connected. Q.E.D.

The proof of Theorem 3.19 can be reworded as follows. Suppose that we start with a connected graph G. If G has no circuits, it is already a tree. If it has a circuit, remove any edge of the circuit and a connected graph remains. If there is still a circuit, remove any edge of the circuit and a connected graph remains, and so on. Note that Theorem 3.19 gives us a method for determining if a graph G is connected: Simply test if G has a spanning tree. Algorithms for doing this are discussed in Section 13.1.

3.5.5 Proof of Theorem 3.16 and a Related Result[26]

We now present a proof of Theorem 3.16 and then give a related theorem.

Proof of Theorem 3.16. We have already shown, in Theorem 3.15, one direction of this equivalence. To prove the other direction, suppose that G is connected and $n = e + 1$. By Theorem 3.19, G has a spanning tree T. Then T and G have the same number of vertices. By Theorem 3.15, T has $n - 1$ edges. Since G also has $n - 1$ edges, and since all edges of T are edges of G, $T = G$. Q.E.D.

We also have another result, similar to Theorem 3.16, which will be needed in Chapter 13.

Theorem 3.20 Suppose that G is a graph with n vertices and e edges. Then G is a tree if and only if G has no circuits and $n = e + 1$.

Proof. Again it remains to prove one direction of this theorem. Suppose that G has no circuits and $n = e + 1$. If G is not connected, let $K_1, K_2, \ldots, K_p$ be its connected components, $p > 1$. Let K_i have n_i vertices. Then K_i is connected and has no circuits, so K_i is a tree. Thus, by Theorem 3.15, K_i has $n_i - 1$ edges. We conclude that the number of edges of G is given by

$$(n_1 - 1) + (n_2 - 1) + \cdots + (n_p - 1) = \sum n_i - p = n - p.$$

But since $p > 1, n - p < n - 1$, so $n \neq e + 1$. This is a contradiction. Q.E.D.

[26] This subsection may be omitted.

3.5.6 Chemical Bonds and the Number of Trees

In 1857, Arthur Cayley discovered trees while he was trying to enumerate the isomers of the saturated hydrocarbons, chemical compounds of the form C_kH_{2k+2}. This work was the forerunner of a large amount of graph-theoretical work in chemistry and biochemistry. We present Cayley's approach here.

Combinatorial methods are used in chemistry to represent molecules, clusters, and reaction paths. Coding theory (to be studied in Chapter 9) is basic in systematizing the enormous amount of chemical data available and in enumerating molecules, isomers, and families having various properties. Graph theory is used to understand the structure of chemical compounds, proteins, and so on, and symmetries in chemical structures, and it is also useful in developing systematic approaches to chemical nomenclature. All of these techniques are used increasingly by industry in the rapidly expanding fields of computer-assisted molecular design and combinatorial chemistry. For good references on graph theory and chemistry, see Balaban [1976], Hansen, Fowler, and Zheng [2000], McKee and Beineke [1997], and Rouvray and Balaban [1979]. See also the September 1992 issue of *The Journal of Chemical Education*, which featured graph theory in chemistry. We give other applications of combinatorics to chemistry in Section 6.4 and Chapter 8.

The isomers C_kH_{2k+2} can be represented by representing a carbon atom with the letter C and a hydrogen atom with the letter H, and linking two atoms if they are bonded in the given compound. For example, methane and ethane are shown in Figure 3.77. We can replace these diagrams with graphs by replacing each letter with a vertex, as shown in Figure 3.78. We call these graphs *bond graphs*. Note that given a bond graph of a saturated hydrocarbon, the vertices can be relabeled with letters C and H in an unambiguous fashion: A vertex is labeled C if it is bonded to four other vertices (carbon has chemical valence 4), and H if it is bonded to one other vertex (hydrogen has valence 1). Every vertex of one of our bond graphs is bonded to either one or to four other vertices. The bond graphs of some other saturated hydrocarbons are shown in Figure 3.79.

The graphs of Figures 3.78 and 3.79 are all trees. We shall show that this is no accident, that is, that the bond graph of every saturated hydrocarbon is a tree.

Recall that the degree of a vertex in a graph is the number of its neighbors, and by Theorem 3.1, the sum of the degrees of the vertices is twice the number of edges. Now in the bond graph for C_kH_{2k+2}, there are

$$k + 2k + 2 = 3k + 2$$

vertices. Moreover, each carbon vertex has degree 4 and each hydrogen vertex has degree 1. Thus, the sum of the degrees is

$$4k + 1(2k + 2) = 6k + 2.$$

The number of edges is half this number, or $3k + 1$. Thus, the number of edges is exactly one less than the number of vertices. Since the bond graph for a chemical compound C_kH_{2k+2} must be connected, it follows by Theorem 3.16 that it is a tree.

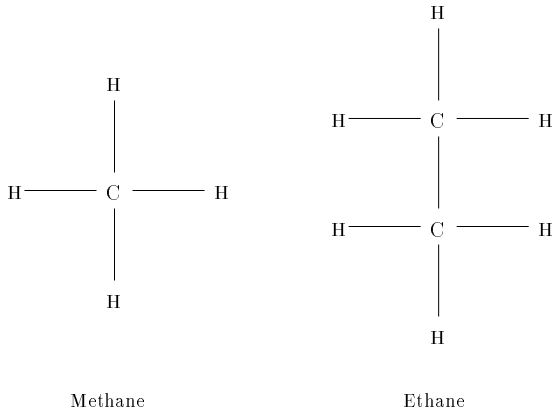


Figure 3.77: Two saturated hydrocarbons.

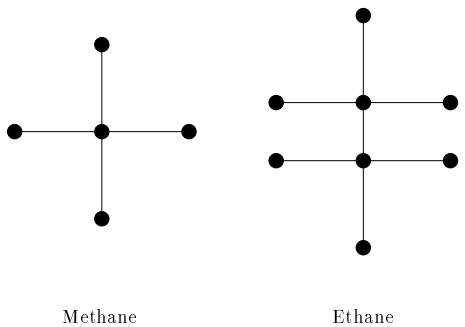


Figure 3.78: Bond graphs for the saturated hydrocarbons of Figure 3.77.

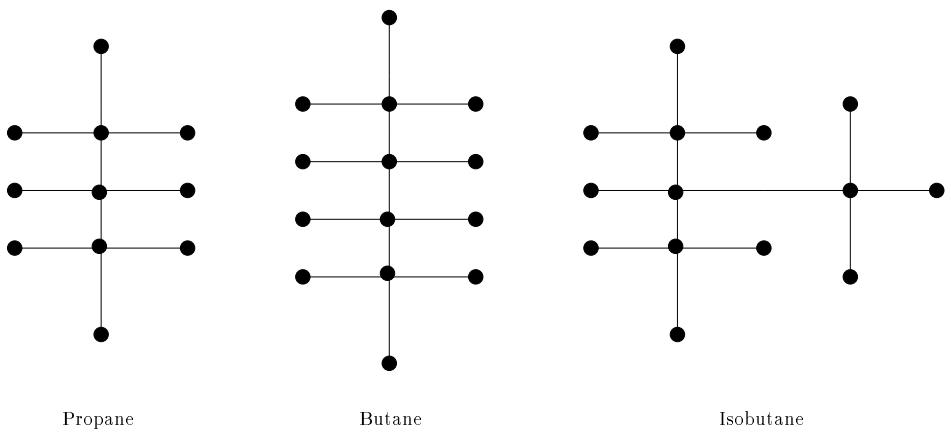


Figure 3.79: More bond graphs.

Now Cayley abstracted the problem of enumerating all possible saturated hydrocarbons to the problem of enumerating all trees in which every vertex has degree 1 or 4. He found it easier to begin by enumerating all trees. In the process, he discovered abstractly the bond graphs of some saturated hydrocarbons which were previously unknown, and predicted their existence. They were later discovered.

It clearly makes a big difference in counting the number of distinct graphs of a certain type whether or not we consider these graphs as labeled (see the discussion in Section 3.1.3). In particular, Cayley discovered that for $n \geq 2$, there were n^{n-2} distinct labeled trees with n vertices. We shall present one proof of this result here. For a survey of other proofs, see Moon [1967] (see also Harary and Palmer [1973], Shor [1995], and Takács [1990]).

Theorem 3.21 (Cayley [1889]) If $n \geq 2$, there are n^{n-2} distinct labeled trees of n vertices.

Let $N(d_1, d_2, \ldots, d_n)$ count the number of labeled trees with n vertices with the vertex labeled i having degree $d_i + 1$. We first note the following result, which is proved in Exercise 34.

Theorem 3.22 If $n \geq 2$ and all d_i are nonnegative integers, then

$$N(d_1, d_2, \ldots, d_n) = \begin{cases} 0 & \text{if } \sum_{i=1}^{n} d_i \neq n-2 \\ C(n-2; d_1, d_2, \ldots, d_n) & \text{if } \sum_{i=1}^{n} d_i = n-2. \end{cases} \tag{3.21}$$

In this theorem,

$$C(n-2; d_1, d_2, \ldots, d_n) = \frac{(n-2)!}{d_1! d_2! \cdots d_n!}$$

is the multinomial coefficient studied in Section 2.11.

To illustrate Theorem 3.22, note that

$$N(1,3,0,0,0,0) = C(4; 1,3,0,0,0,0) = \frac{4!}{1!3!0!0!0!0!} = 4.$$

There are four labeled trees of six vertices with vertex 1 having degree 2, vertex 2 having degree 4, and the remaining vertices having degree 1. Figure 3.80 shows the four trees.

Cayley's Theorem now follows as a corollary of Theorem 3.22. For the number $T(n)$ of labeled trees of n vertices is given by

$$T(n) = \sum \left\{ N(d_1, d_2, \ldots, d_n) : d_i \geq 0, \sum_{i=1}^{n} d_i = n-2 \right\},$$

which is the same as

$$T(n) = \sum \left\{ C(n-2; d_1, d_2, \ldots, d_n) : d_i \geq 0, \sum_{i=1}^{n} d_i = n-2 \right\}.$$

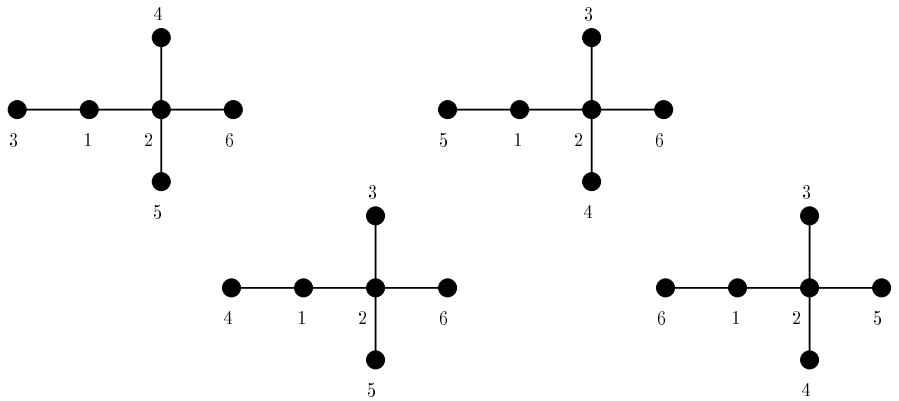


Figure 3.80: The four labeled trees with degrees 1+1, 3+1, 0+1, 0+1, 0+1, 0+1, respectively.

Now it is easy to see that the binomial expansion (Theorem 2.7) generalizes to the following *multinomial expansion*: If p is a positive integer, then

$$(a_1 + a_2 + \cdots + a_k)^p = \sum \left\{ C(p; d_1, d_2, \ldots, d_k) a_1^{d_1} a_2^{d_2} \cdots a_k^{d_k} \ : d_i \geq 0, \ \sum_{i=1}^{k} d_i = p \right\}.$$

Thus, taking $k = n$ and $p = n - 2$ and $a_1 = a_2 = \cdots = a_k = 1$, we have

$$T(n) = (1 + 1 + \cdots + 1)^{n-2} = n^{n-2}.$$

This is Cayley's Theorem.

3.5.7 Phylogenetic Tree Reconstruction

In Example 3.27, we introduced the concept of an evolutionary or phylogenetic tree. A more general phylogenetic tree has weights of real numbers on its edges, representing the passage of time between two evolutionary events. An important problem in biology (and in other areas such as linguistics) is to reconstruct a phylogenetic tree from some information about present species. A general introduction to this topic from the biological point of view can be found in Fitch [1999] and an introduction from a combinatorial point of view can be found in Gusfield [1997]. Common methods for phylogenetic tree reconstruction start with the DNA sequences of presently existing species. *Sequence-based methods* use these sequences directly to reconstruct the tree. *Distance-based methods*, by contrast, first compute distances between all pairs of sequences and then use these distances to reconstruct the tree. More particularly, they start with n species and an $n \times n$ symmetric matrix D whose i, j entry $D(i, j)$ gives the calculated distance between the two species i and j. They then seek a tree in which the given species are the *leaves*, i.e., the vertices with only one neighbor, and $D(i, j)$ is equal to or closely approximated by the distance $d(i, j)$

$$D = \begin{pmatrix} 0 & 5 & 9 & 6 \\ 5 & 0 & 10 & 7 \\ 9 & 10 & 0 & 5 \\ 6 & 7 & 5 & 0 \end{pmatrix}$$

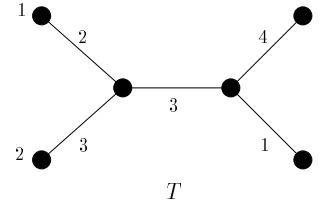


Figure 3.81: An additive distance matrix D and corresponding weighted tree T.

between i and j in the tree. Here, we measure distance $d(i,j)$ between vertices i and j in a tree with edge weights by summing up the weights in the unique simple chain from i to j in the tree. (Theorem 3.18 says that in any tree, there is always a unique simple chain between two given vertices.)

If D is a symmetric $n \times n$ matrix with 0's on the diagonal, we say that D is an *additive distance matrix* if we can find an edge-weighted tree with n leaves, corresponding to the given species, so that $d(i,j) = D(i,j)$. For instance, the matrix D of Figure 3.81 is an additive distance matrix and T of that figure is the corresponding weighted tree. For example, in the tree, the weights on the unique simple chain from 1 to 4 are 2, 3, and 1, so $d(1,4) = 6$, which corresponds to $D(1,4)$. How can we tell if a matrix is an additive distance matrix?

Theorem 3.23 (Buneman [1971]) A symmetric matrix D with 0's on the diagonal is an additive distance matrix if and only if for all i,j,k,l, of the three pairwise sums $D(i,j)+D(k,l), D(i,k)+D(j,l), D(i,l)+D(j,k)$, the largest two are identical.

To illustrate this theorem, we note that in Figure 3.81, for example, $D(1,2) + D(3,4) = 9, D(1,3) + D(2,4) = 15, D(1,4) + D(2,3) = 15$. There are good algorithms, in fact $O(n^2)$ algorithms, for reconstructing, if possible, an edge-weighted tree corresponding exactly to an additive distance matrix. See Gusfield [1997] for a variety of references.

If a matrix is not an additive distance matrix, we might wish to find a "closest" edge-weighted tree. This problem can be made precise in a variety of ways and almost all of them have been shown to be difficult to solve exactly (more precisely, NP-hard). One such problem is to find an additive distance matrix D' so that $\max_{i,j} |D(i,j) - D'(i,j)|$ is as small as possible. (This problem is NP-hard.) Agarwala, *et al.* [1996] found an algorithm that approximates an optimal solution, finding an edge-weighted tree so that $\max_{i,j} |D(i,j) - D'(i,j)|$ is guaranteed to be at most three times the optimal. This algorithm was improved by Cohen and Farach [1997]. Other algorithms, based on the theorem of Buneman (Theorem 3.23), are described in Erdős, *et al.* [1997, 1999].

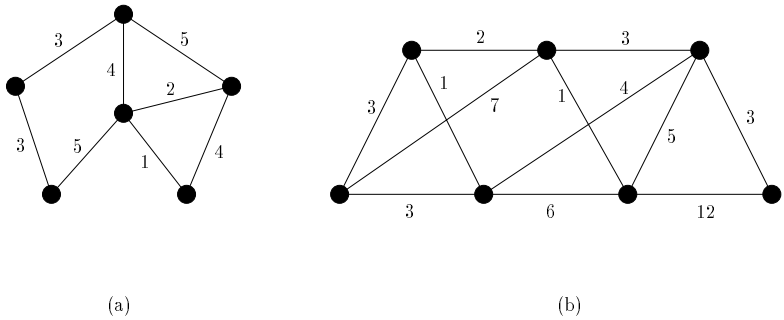


Figure 3.82: Graphs with weights on edges.

EXERCISES FOR SECTION 3.5

1. Draw the sort tree for sorting mail if the "ZIP code" consists of four digits, each being 0, 1, or 2.
2. Find all nonisomorphic trees of four vertices.
3. Find all nonisomorphic trees of five vertices.
4. Find:
   (a) The number of vertices in a tree of 10 edges
   (b) The number of edges in a tree of 10 vertices
5. Check Theorem 3.1 for every graph of Figure 3.76.
6. Give an example of a graph G with $n = e + 1$ but such that G is not a tree.
7. For each graph of Figure 3.23, either find a spanning tree or argue that none exists.
8. In each graph with weights on edges shown in Figure 3.82, find a spanning tree with minimum total weight.
9. A *forest* is a graph each of whose connected components is a tree. If a forest has n vertices and k components, how many edges does it have?
10. A simpler "proof" of uniqueness in Theorem 3.18 would be as follows. Suppose that C_1 and C_2 are two distinct simple chains joining x and y. Then C_1 followed by C_2 is a closed chain. But if a graph has a closed chain, it must have a circuit. Show that the latter statement is false.
11. In a connected graph with 15 edges, what is the maximum possible number of vertices?
12. In a connected graph with 25 vertices, what is the minimum possible number of edges?
13. What is the maximum number of vertices in a graph with 15 edges and three components?
14. Prove the converse of Theorem 3.18; that is, if G is any graph and any two vertices are joined by exactly one simple chain, then G is a tree.

15. Prove that if two nonadjacent vertices of a tree are joined by an edge, the resulting graph will have a circuit.
16. Prove that if any edge is deleted from a tree, the resulting graph will be disconnected.
17. If we have an (connected) electrical network with e elements (edges) and n nodes (vertices), what is the minimum number of elements we have to remove to eliminate all circuits in the network?
18. If G is a tree of n vertices, show that its chromatic polynomial is given by
$$P(G, x) = x(x-1)^{n-1}.$$
19. Use the result of Exercise 18 to determine the chromatic number of a tree.
20. Find the chromatic number of a tree by showing that every tree is bipartite.
21. Show that the converse of the result in Exercise 18 is true, that is, if
$$P(G, x) = x(x-1)^{n-1},$$
then G is a tree.
22. Suppose that G is a tree, $\{u, v\}$ is an edge of G, and H is obtained from G by deleting edge $\{u, v\}$, but not vertices u and v. Show that H has exactly two connected components. (You may not assume any of the theorems of this section except possibly Theorem 3.18.)
23. (Peschon and Ross [1982]) In an electrical distribution system, certain locations are joined by connecting electrical lines. A system of switches is used to open or close these lines. The collection of open lines has to have two properties: (1) every location has to be on an open line, and (2) there can be no circuits of open lines, for a short on one open line in an open circuit would shut down all lines in the circuit. Discuss the mathematical problem of finding which switches to open.
24. (Ahuja, Magnanti, and Orlin [1993], Prim [1957]) Agents of an intelligence agency each know how to contact each other. However, a message passed between agents i and j has a certain probability p_{ij} of being intercepted. If we want to make sure that all agents get a message but minimize the probability of the message being intercepted, which agents should pass the message to which agents? Formulate this as a spanning tree problem. (*Hint:* You will need to use logarithms.)
25. Here is an algorithm for finding a spanning tree of a connected graph G. Pick any vertex and mark it. Include any edge to an unmarked neighboring vertex and mark that vertex. At each step, continue adding an edge from the last marked vertex to an unmarked vertex until there is no way to continue. Then go back to the most recent marked vertex from which it is possible to continue. Stop when all vertices have been marked. (This procedure is called *depth first search* and we return to it in Section 11.1.)
    (a) Illustrate the algorithm on the graphs of Figure 3.82 (disregarding the weights).
    (b) Show that there is a spanning tree of a graph that cannot be found this way.
26. Suppose that a chemical compound C_kH_m has a bond graph that is connected and has no circuits. Show that m must be $2k + 2$.
27. Find the number of spanning trees of K_n.

28. Check Cayley's Theorem by finding all labeled trees of:

    (a) Three vertices (b) Four vertices

29. Is there a tree of seven vertices:

    (a) With each vertex having degree 1?

    (b) With two vertices having degree 1 and five vertices having degree 2?

    (c) With five vertices having degree 1 and two vertices having degree 2?

    (d) With vertices having degrees 2, 2, 2, 3, 1, 1, 1?

30. Is there a tree of five vertices with two vertices of degree 3?

31. In each of the following cases, find the number of labeled trees satisfying the given degree conditions by our formula and draw the trees in question.

    (a) Vertices 1, 2, and 3 have degree 2, and vertices 4 and 5 have degree 1.

    (b) Vertex 1 has degree 2, vertex 2 has degree 3, and vertices 3, 4, and 5 have degree 1.

    (c) Vertex 1 has degree 3, vertices 2 and 3 have degree 2, and vertices 4, 5, and 6 have degree 1.

32. Find the number of labeled trees of:

    (a) Six vertices, four having degree 2

    (b) Eight vertices, six having degree 2

    (c) Five vertices, exactly three of them having degree 1

    (d) Six vertices, exactly three of them having degree 1

33. Prove that $N(d_1, d_2, \ldots, d_n) = 0$ if $\sum_{i=1}^{n} d_i \neq n-2$. (*Hint:* Count edges.)

34. This exercise sketches a proof of Theorem 3.22. Define $M(d_1, d_2, \ldots, d_n)$ by the right-hand side of Equation (3.21). It suffices to prove that if $n \geq 2$ and all d_i are nonnegative and $\sum_{i=1}^{n} d_i = n-2$, then

$$N(d_1, d_2, \ldots, d_n) = M(d_1, d_2, \ldots, d_n). \tag{3.22}$$

    (a) Under the given assumptions, verify (3.22) for $n = 2$.

    (b) Under the given assumptions, show that $d_i = 0$, for some i.

    (c) Suppose that i in part (b) is n. Show that

$$\begin{aligned} N(d_1, d_2, \ldots, d_{n-1}, 0) = {} & N(d_1 - 1, d_2, d_3, \ldots, d_{n-1}) + \\ & N(d_1, d_2 - 1, d_3, \ldots, d_{n-1}) + \cdots + N(d_1, d_2, d_3, \ldots, d_{n-2}, d_{n-1} - 1), \end{aligned} \tag{3.23}$$

    where a term $N(d_1, d_2, \ldots, d_{k-1}, d_k - 1, d_{k+1}, \ldots, d_{n-1})$ appears on the right-hand side of (3.23) if and only if $d_k > 0$.

    (d) Show that M also satisfies (3.23).

    (e) Verify (3.22) by induction on n. (In the language of Chapter 6, the argument essentially amounts to showing that if M and N satisfy the same recurrence and the same initial condition, then $M = N$.)

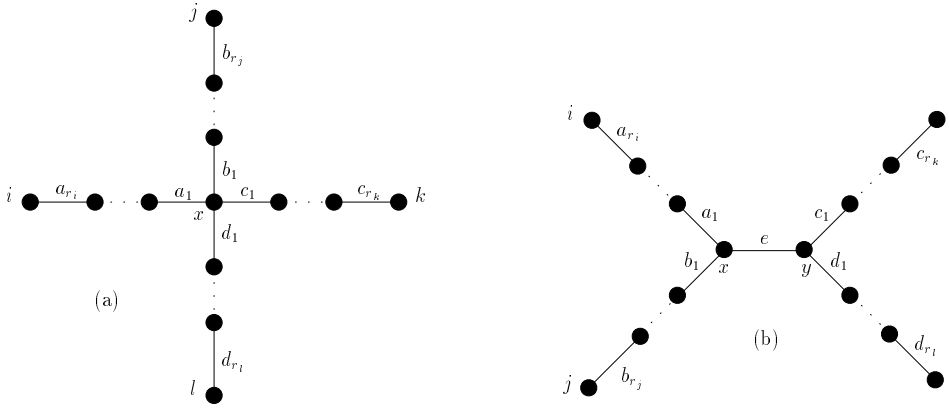


Figure 3.83: Edge-weighted trees. In these trees, the lengths of the chains from x to i, to j, to k, and to l in (a), from x to i and to j, and y to k and to l in (b) are arbitrary. Also, arbitrary positive weights a_i, b_i, c_i, d_i, e can be assigned to the edges.

35. Determine which of the following are additive distance matrices.

(a) $\begin{pmatrix} 0 & 1 & 3 \\ 1 & 0 & 2 \\ 3 & 2 & 0 \end{pmatrix}$ (b) $\begin{pmatrix} 0 & 1 & 4 \\ 1 & 0 & 2 \\ 4 & 2 & 0 \end{pmatrix}$

(c) $\begin{pmatrix} 0 & 1 & 3 & 9 \\ 1 & 0 & 2 & 2 \\ 3 & 2 & 0 & 1 \\ 9 & 2 & 1 & 0 \end{pmatrix}$

36. (a) Show that in tree (a) of Figure 3.83, all three sums in Theorem 3.23 are equal.

    (b) Show that this conclusion fails for tree (b) of Figure 3.83.

37. Suppose that T is an edge-weighted tree and

$$d(i,j) + d(k,l) \leq d(i,k) + d(j,l) = d(i,l) + d(j,k).$$

    Show that the vertices i, j, k, l are located in either configuration (a) or configuration (b) of Figure 3.83.

38. Suppose that T is a phylogenetic tree and each vertex has at most three neighbors. This occurs when each "evolutionary event" involves the split of a population into two new ones. Conclude from Exercise 37 that for all i, j, k, l, one of the three sums in Theorem 3.23 is strictly less than the other two.

3.6 APPLICATIONS OF ROOTED TREES TO SEARCHING, SORTING, AND PHYLOGENY RECONSTRUCTION[27]

3.6.1 Definitions

Using trees to search through a table or a file is one of the most important operations in computer science. In Example 2.18 we discussed the problem of searching through a file to find the key (identification number) of a particular person, and pointed out that there were more efficient ways to search than to simply go through the list of keys from beginning to end. In this section we show how to do this using search trees. Then we show how trees can be used for another important problem in computer science: sorting a collection into its natural order, given a list of its members. Finally, we use keyword ideas to formulate a different approach to the phylogenetic tree reconstruction problem discussed in Section 3.5.7.

Before defining search trees, let us note that each of the examples of trees in Figures 3.72–3.74 is drawn in the following way. Each vertex has a *level*, $0, 1, 2, \ldots, k$. There is exactly one vertex, the *root*, which is at level 0. All adjacent vertices differ by exactly one level and each vertex at level $i + 1$ is adjacent to exactly one vertex at level i. Such a tree is called a *rooted tree*. (It is not hard to show that every tree, after designating one vertex as the root, can be considered a rooted tree; see Exercise 5.) The number k is called the *height* of the rooted tree. In the example of Figure 3.72, the level 0 (root) vertex is Boss; the level 1 vertices are Jones, Smith, and Brown; the level 2 vertices are Johnson, Jackson, and so on; and the level 3 vertices are Allen, Baker, and so on. The height is 3. In a rooted tree, all vertices adjacent to vertex u and at a level below u's are called the *children* of u. For instance, in Figure 3.72, the children of Brown are White, Black, and Green. All vertices that are joined to u by a chain of vertices at levels below u's in the tree are called *descendants* of u. Thus, in our example, the descendants of Brown are White, Black, Green, Engel, Freyer, and Gold.

A rooted tree is called *m-ary* if every vertex has m or fewer children. A 2-ary rooted tree is called a *binary tree*. A rooted tree is called *complete m-ary* if every vertex has either 0 or m children. Figure 3.84 shows a complete binary tree. In a binary tree, we shall assume that any child of a vertex is designated as a *left child* or a *right child*.

Example 3.33 Code Trees When data need to be transmitted, each symbol is *encoded* or assigned a binary string. Suppose that the data consist of grades: 15 A's, 27 B's, 13 C's, 4 D's, and 1 F. If each bit string is the same length, then we need bit strings of length 3 (or more) to represent these five symbols. Therefore, this dataset requires at least

$$(15 + 27 + 13 + 4 + 1) \cdot 3 = 180 \text{ bits.}$$

[27] This section may be omitted.

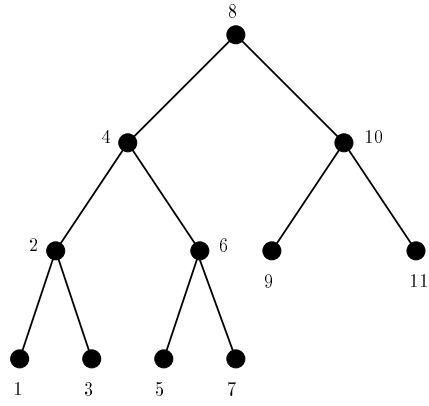


Figure 3.84: A complete binary search tree.

If we were trying to minimize the number of bits to transmit, bit strings of length 4 or more would not be used.

If we are allowed to use variable-length bit strings, substantial savings can be reaped. Suppose that we represent the grades as follows: A: 01, B: 1, C: 111, D: 10, F: 010. We now require only

$$(15 \cdot 2) + (27 \cdot 1) + (13 \cdot 3) + (4 \cdot 2) + (1 \cdot 3) = 107 \text{ bits.}$$

Although this is a savings in bits sent, there is a problem with this encoding. Consider the following transmission: $01011\cdots$. When the receiver begins to decode, what is the first grade? An A might be assumed since the transmission begins with 01. However, an F could also be construed since the transmission also begins with 010. This problem arises since a bit string is the prefix of some other bit string. We need to find an encoding where this doesn't happen. Such an encoding is called a *prefix code*.

Prefix codes are easy to find. (In fact, the length 3 bit string encoding from above is a prefix code.) Consider the binary tree in Figure 3.85. Left branches are labeled with 0 and right branches with 1. This generates a bit string to associate with each vertex by looking at the unique chain from the root to that vertex. The vertices representing the various grades are labeled accordingly. Notice that the unique chain to the vertex labeled B (010) starts at the root and goes left (0), then right (1), then left (0).

This prefix code yields a savings in bits over the code assigning bit strings of length 3 to each grade. But is this the best that we could do? The trees associated, as above, with prefix codes are called *code trees*. Figure 3.85 is but one example of a *code tree* for this example. The problem of minimizing the number of bits to transmit data is equivalent to finding an optimal code tree. Given a set of symbols and their frequencies, Huffman [1952] provides an algorithm for the construction of an optimal code tree. The optimal code tree output of the following algorithm is sometimes called a *Huffman tree*.

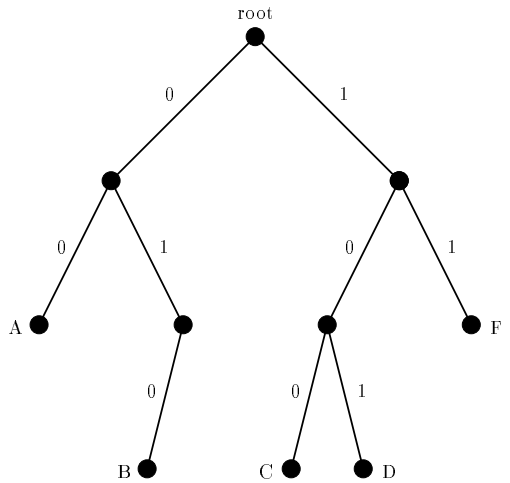


Figure 3.85: A code tree.

Algorithm 3.2: Optimal Code Tree

Input: A forest S of n isolated vertices (rooted trees) with labels the symbols $\{s_1, s_2, \ldots, s_n\}$ and weights the frequencies $\{f_1, f_2, \ldots, f_n\}$, respectively.
Output: An optimal code tree.

Step 1. Find two trees T, T' in S with the two smallest root weights f_i, f_j.

Step 2. Create a new complete binary tree T'' with root s_{ij} and weight $f_{ij} = f_i + f_j$ and having T and T' as its children.

Step 3. Replace T and T' in S with T''. If S is a tree, stop and output S. If not, return to step 1.

To give an example, suppose that the following data need to be transmitted: 15 A's, 4 B's, 10 C's, 15 D's, 11 Passes, and 3 Fails. Figure 3.86 shows the construction of the optimal code tree using Algorithm 3.2. Note that 6 distinct pieces of information (A, B, C, D, Pass, Fail) would need, at least, length 3 bit strings if all bit strings must be of the same length. The total number of bits for transmission would thus equal $(15+4+10+15+11+3)\cdot 3 = 174$. Using Algorithm 3.2, we find that only

$$(15\cdot 2) + (4\cdot 4) + (10\cdot 3) + (15\cdot 2) + (11\cdot 2) + (3\cdot 4) = 140$$

bits are needed, a savings of almost 20 percent. The optimality of the algorithm is proven in the exercises (see Exercise 20). For more on codes see Chapter 10. ■

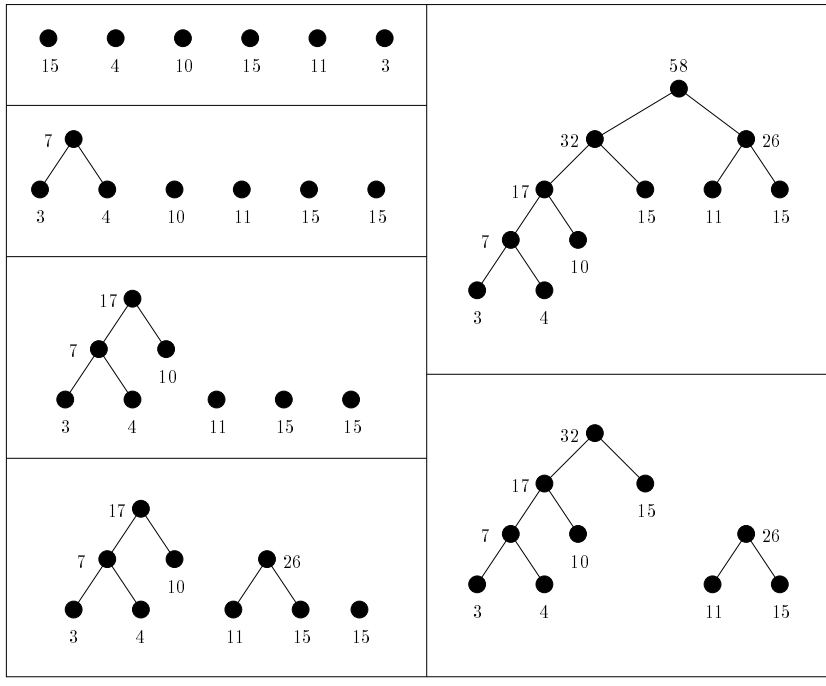


Figure 3.86: Optimal code tree using Algorithm 3.2 for transmitting 15 A's, 4 B's, 10 C's, 15 D's, 11 Passes, and 3 Fails. (Figure/algorithm starts in upper left-hand corner and goes counterclockwise.)

3.6.2 Search Trees

Suppose that we have a file of n names. Let T be any (rooted) binary tree. Label each vertex u of T with a key (a real number) in such a way that if u has a child, its left child and all descendants of its left child get keys that are lower numbers than the key of u, and its right child and all descendants of its right child get keys that are higher numbers than the key of u. A binary tree with such a labeling will be called a *binary search tree*. (Such a labeling can be found for every binary tree; see below.) Figure 3.84 shows a complete binary search tree, where the keys are the integers $1, 2, \ldots, 11$. Now given a particular key k, to locate it we search through the binary search tree, starting with the root. At any stage we look at the key of the vertex x being examined. At the start, x is the root. We ask if k is equal to, less than, or greater than the key of x. If equal, we have found the desired file. If less, we look next at the left child of x; if greater, we look next at the right child of x. We continue the same procedure. For instance, to find the key 7 using the binary search tree of Figure 3.84, we start with key 8, that of the root. Then we go to key 4 (since 7 is less than 8), then to key 6 (since 7 is higher than 4), then to key 7 (since 7 is higher than 6). Notice that our search took four steps rather than the seven steps it would have taken to go through the list $1, 2, \ldots, 11$ in order. See

Exercise 23 for another application of binary search trees.

Suppose that we can find a binary search tree. What is the computational complexity of a file search in the worst case? It is the number of vertices in the longest chain descending from the root to a vertex with no children; that is, it is one more than the height of the binary search tree. Obviously, the computational complexity is minimized if we find a binary search tree of minimum height. Now any binary tree can be made into a binary search tree. For a proof and an algorithm, see, for example, Reingold, Nievergelt, and Deo [1977]. Hence, we are left with the following question. Given n, what is the smallest h such that there is a binary tree on n vertices with height h? We shall answer this question through the following theorem, which is proved in Section 3.6.3. Recall that $\lceil a \rceil$ is the least integer greater than or equal to a.

Theorem 3.24 The minimum height of a binary tree on n vertices is equal to $\lceil \log_2 (n+1) \rceil - 1$.

The binary tree of Figure 3.84 is a binary tree on 11 vertices that has the minimum height 3 since

$$\lceil \log_2 (n+1) \rceil = \lceil \log_2 12 \rceil = \lceil 3.58 \rceil = 4.$$

In sum, Theorem 3.24 gives a logarithmic bound

$$\lceil \log_2 (n+1) \rceil - 1 + 1 = \lceil \log_2 (n+1) \rceil$$

on the computational complexity of file search using binary search trees. This bound can of course be attained by finding a binary search tree of minimum height, which can always be done (see the proof of Theorem 3.24). To summarize:

Corollary 3.24.1 The computational complexity of file search using binary search trees is $\lceil \log_2 (n+1) \rceil$.

This logarithmic complexity is in general a much better complexity than the complexity n we obtained in Example 2.18 for file search by looking at the entries in a list in order. For $\lceil \log_2 (n+1) \rceil$ becomes much less than n as n increases.

3.6.3 Proof of Theorem 3.24[28]

To prove Theorem 3.24, we first prove the following.

Theorem 3.25 If T is a binary tree of n vertices and height h, then $n \leq 2^{h+1} - 1$.

Proof. There is one vertex at level 0, there are at most $2^1 = 2$ vertices at level 1, there are at most $2^2 = 4$ vertices at level 2, there are at most $2^3 = 8$ vertices at level 3, ..., and there are at most 2^h vertices at level h. Hence,

$$n \leq 1 + 2^1 + 2^2 + 2^3 + \cdots + 2^h. \tag{3.24}$$

[28] This subsection may be omitted.

Now we use the general formula

$$1 + x + x^2 + \cdots + x^h = \frac{1 - x^{h+1}}{1 - x}, \quad x \neq 1, \tag{3.25}$$

which will be a very useful tool throughout this book. Substituting $x = 2$ into (3.25) and using (3.24), we obtain

$$n \leq \frac{1 - 2^{h+1}}{1 - 2} = 2^{h+1} - 1.$$

Q.E.D.

Proof of Theorem 3.24. We have

$$\begin{aligned} 2^{h+1} &\geq n + 1, \\ h + 1 &\geq \log_2(n + 1), \\ h &\geq \log_2(n + 1) - 1. \end{aligned}$$

Since h is an integer,

$$h \geq \lceil \log_2(n + 1) \rceil - 1.$$

Thus, every binary tree on n vertices has height at least $\lceil \log_2(n + 1) \rceil - 1$. It is straightforward to show that there is always a binary tree of n vertices whose height is exactly $p = \lceil \log_2(n+1) \rceil - 1$. Indeed, any complete binary tree in which the only vertices with no children are at level p or $p - 1$ will suffice to demonstrate this.[29]

Q.E.D.

3.6.4 Sorting

A basic problem in computer science is the problem of placing a set of items in their natural order, usually according to some numerical value. We shall call this problem the *sorting problem*. We shall study the problem of sorting by making comparisons of pairs of items.

Any algorithm for sorting by comparisons can be represented by a (complete) binary tree called a *decision tree*. Figure 3.87 shows a decision tree for a computer program that would sort the three numbers a, b, and c. In each case, a vertex of the tree corresponds to a test question or an output. At a test question vertex, control moves to the left child if the question is answered "yes" and to the right child if "no." Output vertices are shown by squares, test vertices by circles. The complexity of the algorithm represented by the decision tree T of Figure 3.87 is the number of steps (comparisons) required to reach a decision in the worst case. Since outputs correspond to vertices with no children, the complexity is obtained by finding one less than the number of vertices in the longest chain from the root to a vertex of T with no children, that is, by finding the height of the binary tree T. In our example, the height is 3.

In a rooted tree, let us call vertices with no children *leaves*. (These correspond to the leaves defined in Section 3.5.7.) Note that to sort a set of p (distinct) items,

[29] Such a binary tree is called *balanced*. Figure 3.84 is an example of a balanced binary tree.

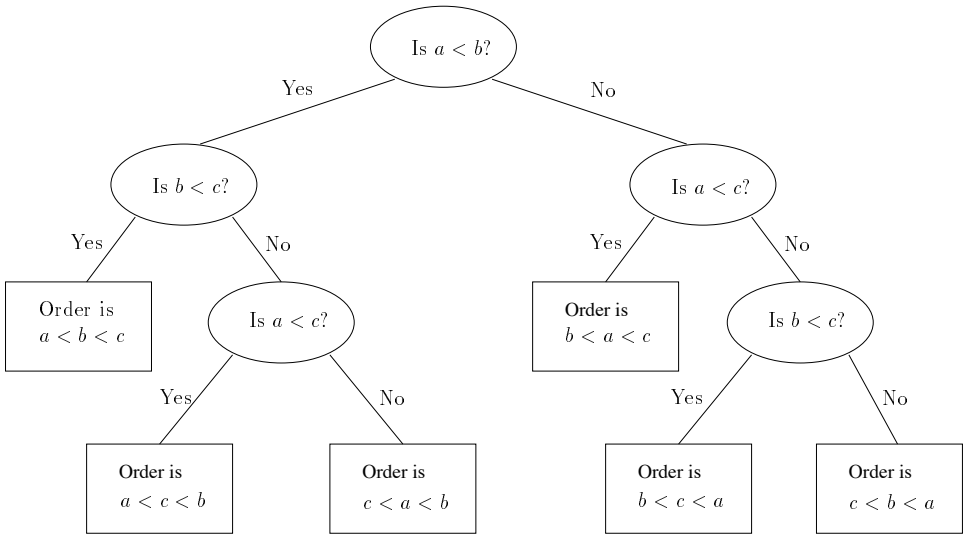


Figure 3.87: A decision tree T for sorting the set of numbers $\{a, b, c\}$.

there are $p!$ possible orders, so a decision tree for the sorting will have at least $p!$ leaves. (We say "at least" because several chains may lead from a root to the same order.) Now we have the following theorem, whose proof is left as an exercise (Exercise 33).

Theorem 3.26 A binary tree of height h has at most 2^h leaves.

This theorem is easily illustrated by the binary trees of Figure 3.74 or 3.84.

Theorem 3.27 Any algorithm for sorting $p \geq 4$ items by pairwise comparisons requires in the worst case at least $cp \log_2 p$ comparisons, where c is a positive constant.

Proof. We have already observed that any decision tree T that sorts p items must have at least $p!$ leaves. Thus, the number of comparisons in the worst case, which is the height of the tree T, has to be at least $\log_2 p!$ (for $2^{\log_2 p!} = p!$). Now for $p \geq 1$,

$$p! \geq p(p-1)(p-2)\cdots\left(\left\lceil \frac{p}{2} \right\rceil\right) \geq \left(\frac{p}{2}\right)^{p/2}.$$

Thus, for $p \geq 4$,

$$\log_2 p! \geq \log_2 \left(\frac{p}{2}\right)^{p/2} = \frac{p}{2} \log_2 \left(\frac{p}{2}\right) \geq \frac{p}{4} \log_2 p = \frac{1}{4} p \log_2 p. \qquad \text{Q.E.D.}$$

There are a variety of sorting algorithms that actually achieve the bound in Theorem 3.27, that is, can be carried out in a constant times $p \log_2 p$ steps. Among

the better-known ones is heap sort. In the text and exercises we discuss two well-known sorting algorithms, bubble sort and quik sort, which do not achieve the bound. For careful descriptions of all three of these algorithms, see, for example, Baase [1992], Brassard and Bratley [1995], or Manber [1989]. Note that $cp \log_2 p \leq cp^2$, so an algorithm that takes $cp \log_2 p$ steps is certainly a polynomially bounded algorithm.

In the algorithm known as *bubble sort*, we begin with an ordered set of p (distinct) items. We wish to put them in their proper (increasing) order. We successively compare the ith item to the $(i+1)$st item in the list, interchanging them if the ith item is larger than the $(i+1)$st. The algorithm is called bubble sort because the larger items rise to the top much like the bubbles in a glass of champagne. Here is a more formal statement of the algorithm.

Algorithm 3.3: Bubble Sort

Input: An ordered list $a_1 a_2 \ldots a_p$ of p items.
Output: A listing of $a_1, a_2, \ldots, a_p$ in increasing order.

Step 1. Set $m = p - 1$.

Step 2. For $i = 1, 2, \ldots, m$, if $a_i > a_{i+1}$, interchange a_i and a_{i+1}.

Step 3. Decrease m by 1. If m is now 0, stop and output the order. If not, return to step 2.

To illustrate bubble sort, suppose that we start with the order 516423. We first set $m = p - 1 = 5$. We compare 5 to 1, and interchange them, getting 156423. We then compare 5 to 6, leaving this order as is. We compare 6 to 4, and interchange them, getting 154623. Next, we compare 6 to 2, and interchange them, getting 154263. Finally, we compare 6 to 3, interchange them, and get 154236. We decrease m to 4 and repeat the process, getting successively 154236, 145236, 142536, and 142356. Note that we do not have to compare 5 to 6, since m is now 4. We decrease m to 3 and repeat the process, getting 142356, 124356, and 123456. Then m is set equal to 2 and we get 123456 and 123456. Note that no more interchanges are needed. Next, we set $m = 1$. No interchanges are needed. Finally, we set $m = 0$, and we output the order 123456.

Part of the decision tree for bubble sort on an order a, b, c, d is shown in Figure 3.88.

Note that bubble sort requires $p(p-1)/2$ comparisons. For at the mth iteration or repetition of the procedure, m comparisons are required, and m takes on the values $p-1, p-2, \ldots, 1$. Thus, using the standard formula for the sum of an arithmetic progression, we see that a total of

$$(p-1) + (p-2) + \cdots + 1 = \frac{p(p-1)}{2}$$

steps are needed. In the language of Section 2.18, the algorithm bubble sort is not as efficient as an algorithm that requires $cp \log_2 p$ steps. For bubble sort is an $O(p^2)$ algorithm and p^2 is not $O(p \log_2 p)$.

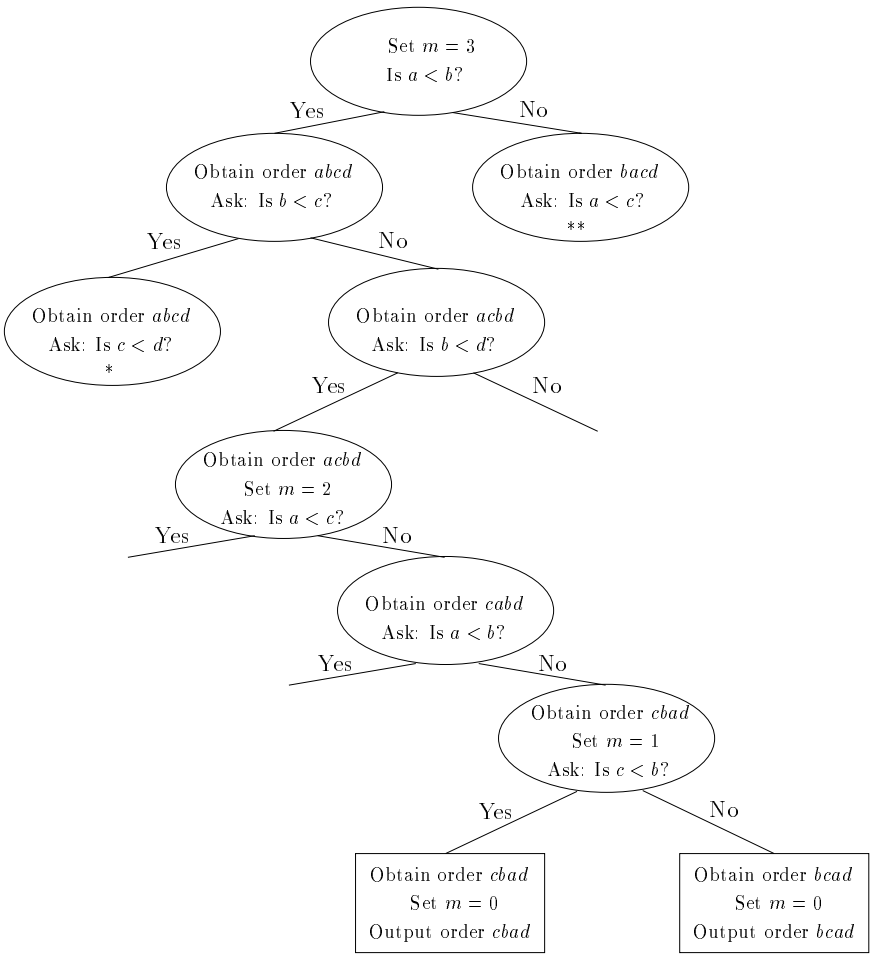


Figure 3.88: Part of the decision tree for bubble sort on an ordered set of four items, $abcd$. (The labels * and ** are for Exercise 28.)

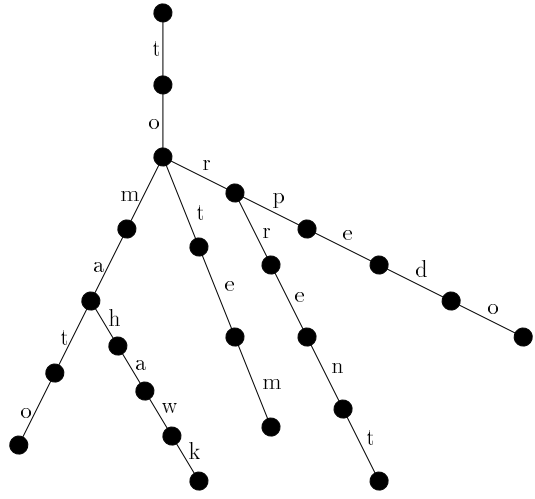


Figure 3.89: A keyword tree.

3.6.5 The Perfect Phylogeny Problem

In this section we describe a different model than that given in Section 3.5.7 for reconstructing the phylogenetic tree corresponding to a given set of presently existing species.

Let Σ be an alphabet (set of "letters"). A *pattern* from Σ is a sequence of letters from the alphabet. For example, if Σ consists of the 26 letters of the alphabet, "tomato" is a pattern, and if $\Sigma = \{0, 1\}$, a bit string is a pattern. Let P be a finite set of patterns from Σ. A *keyword tree* for P is a rooted tree satisfying the following conditions:

- Every edge is labeled with exactly one letter from Σ.
- Any two edges out of a given vertex have distinct labels.
- Every pattern in P corresponds to exactly one leaf so that the letters on the path from the root to the leaf, in that order, give the pattern.

To give an example, Figure 3.89 shows the keyword tree for the set of patterns {tomato, tomahawk, totem, torrent, torpedo}. Algorithms for constructing keyword trees are described in Gusfield [1997].

A way to reconstruct evolutionary history is to study characters that biological objects, such as species, might or might not have. These can be features such as possessing a backbone or having feathers or walking upright. Or they can be more subtle things such as "protein A enhances the expression of protein B" or that the species has a given nucleotide (e.g., adenine, A) in the pth position in its DNA sequence. In all of these cases, we say that we have *binary characters* where objects, in particular species, either have the character or don't. Suppose that we

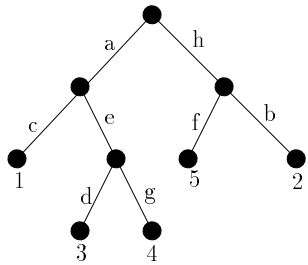


Figure 3.90: A phylogenetic tree.

are interested in a set S of n species and a set C of m characters. We can think of evolution as taking place from an ancestor (the root of the tree) that has none of the given characters in the set C. At some point, a character becomes evident. We then label the edge leading from a given vertex without a given character to its child by putting the name of the character on the edge. For the sake of simplicity, we assume that once a character becomes evident, it never disappears. Then by the time we get to a leaf of the tree, the path from the root to that leaf gives the names of all of the characters displayed by that species. Figure 3.90 gives an example. Here the species 1 has the characters a and c, the species 3 has the characters a, e, d, and so on.

In practice, we want to reconstruct a tree like that in Figure 3.90 given information about what characters each species has. To formulate this problem precisely, let us take M to be an $n \times m$ matrix of 0's and 1's, with its (i, j) entry equal to 1 if and only if species i has character j. We say that a *phylogenetic tree* for M is a rooted tree with exactly n leaves and exactly m edges with the following properties:

- Each of the n species in the set S labels exactly one leaf.
- Each of the m characters in the set C labels exactly one edge.
- For any species, the characters that label the edges along the path from the root to the leaf labeled i specify all of the characters j such that $M(i, j) = 1$.

We would like to know whether or not, given a matrix M, we can construct a phylogenetic tree for M. This is known as the *perfect phylogeny problem*. For instance, consider the matrix M given by

$$M = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccccc} a & b & c & d & e & f & g & h \end{array} \\ \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{pmatrix} \end{array}.$$

Then a phylogenetic tree for M is given by the tree of Figure 3.90. In particular, a phylogenetic tree for M is a kind of keyword tree for the sets of patterns defined by

the characters that a given species has. Not every M has a phylogenetic tree. For instance, the following matrix does not:

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix} \end{array}.$$

We will see from the next theorem why there is no phylogenetic tree. However, the reader might wish to experiment with this small example to try to construct such a tree before reading the theorem.

To solve the perfect phylogeny problem, let O_j be the set of species with a 1 in column j of M.

Theorem 3.28 (Estabrook, Johnson, and McMorris [1975, 1976a,b]) A matrix M has a phylogenetic tree if and only if for j and k, either O_j and O_k are disjoint or one contains the other.

An algorithm for constructing a phylogenetic tree in $O(mn)$ steps was given by Gusfield [1991]. See Gusfield [1997] for more details. The problem becomes more complicated if each character can take on more than two possible states. For instance, we might say that the pth position in the DNA chain can take on states A, T, C, or G. The perfect phylogeny problem reformulated in this more general setting is studied by, among others, Agarwala and Fernandez-Baca [1994], Bodlaender, Fellows, and Warnow [1992], Kannan and Warnow [1994, 1995], and Steel [1992].

EXERCISES FOR SECTION 3.6

1. In the rooted tree of Figure 3.91, find the level of each vertex.
2. In each rooted tree of Figure 3.92, find the level of each vertex.
3. In each rooted tree of Figure 3.92, find the height of the tree.
4. In each rooted tree of Figure 3.92, find all descendants of the vertex b.
5. For each tree in Figure 3.71, label each vertex and select a vertex for the root. Determine the level of the remaining vertices and the height of the tree.
6. Is the tree of Figure 3.91 a binary search tree? If so, describe how to find the key 7.
7. Find a balanced binary tree (as defined in footnote 29) with n vertices where:

   (a) $n = 5$ (b) $n = 8$ (c) $n = 12$ (d) $n = 15$

8. Find a complete binary tree of:

   (a) Height 3 and 8 leaves (b) Height 3 and fewer than 8 leaves

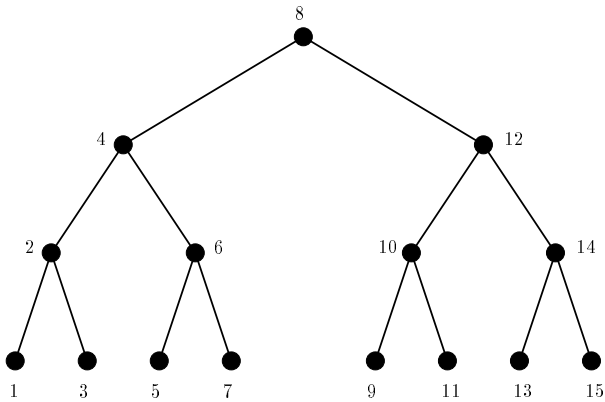


Figure 3.91: Rooted tree for Exercises of Section 3.6.

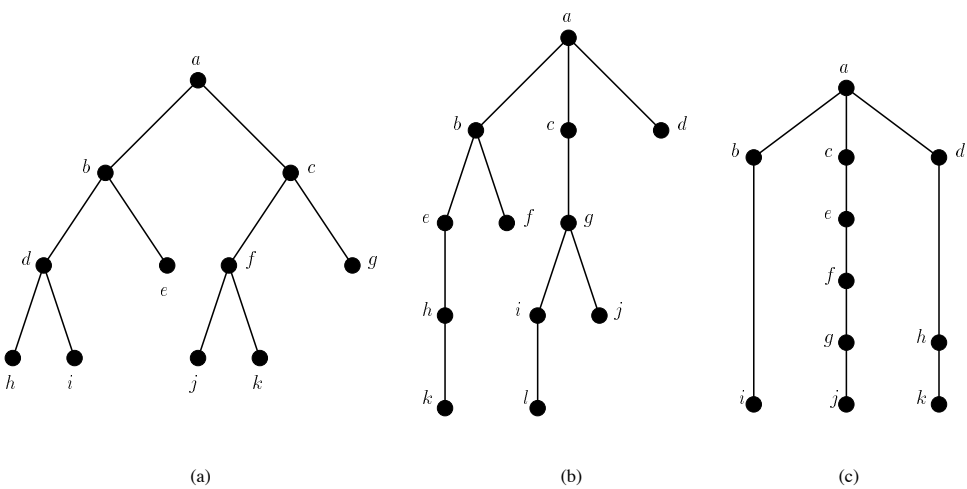


Figure 3.92: Rooted trees for Exercises of Section 3.6.

9. Find a complete binary tree of 11 vertices with:

   (a) As large a height as possible (b) As small a height as possible

10. Find a binary search tree with n vertices where n is:

    (a) 10 (b) 14 (c) 18 (d) 20

11. Find the minimum height of a binary tree of n vertices where n is:

    (a) 74 (b) 512 (c) 4095

12. A complete binary tree can be used to encode bit strings of n bits as follows. At any vertex, its left child is labeled 0 and its right child is labeled 1. A bit string then corresponds to a simple chain from the root to a vertex with no children. Draw such a tree for $n = 4$ and identify the simple chain corresponding to the string 1011.

13. Of all ternary (3-ary) trees on n vertices, what is the least possible height?

14. Using the code tree in Figure 3.85, decode the following transmissions:

    (a) 0100100010110011 (b) 000101110010100010111100101

15. Suppose that a dataset D consists of the four suits in a deck of cards.

    (a) Find a prefix code for D whose longest bit string is length 2. Draw the associated code tree.

    (b) Find a prefix code for D with one bit string of length 3 and the rest shorter. Draw the associated code tree.

16. Find bit strings of length 4 or less for g and h to produce a prefix code for $\{a, b, c, d, e, f, g, h\}$, where the following bit strings have already been assigned:

$$a : 00 \quad b : 011 \quad c : 10 \quad d : 1100 \quad e : 1101 \quad f : 111.$$

    (*Hint:* Draw the code tree.)

17. Using Algorithm 3.2, find an optimal code tree for transmitting the following grades:

    (a) 4 A's, 4 B's, 4 B−'s, 4 C's, 4 C−'s, 4 D's, 4 D−'s, 4 F's

    (b) 8 A's, 6 A−'s, 4 B+'s, 1 B, 5 B−'s, 9 C+'s, 11 C's, 5 C−'s, 3 D+'s, 3 D's, 5 D−'s, 1 F

18. Consider the following data: 3 A's, 4 B's, 7 C's, 14 D's, 14 F's.

    (a) Show that Algorithm 3.2 can produce optimal code trees of different heights.

    (b) How can you modify Algorithm 3.2 to produce minimum-height optimal code trees?

19. Find a set of data for which the tree of Figure 3.93 is an optimal code tree.

20. This exercise proves the optimality of Algorithm 3.2's output.

    (a) Show that an optimal code tree for a given set of data is always a complete binary tree.

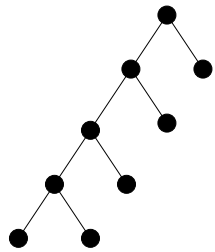


Figure 3.93: An optimal code tree.

(b) Show that Algorithm 3.2's output is optimal for datasets containing two symbols.

(c) Assume that Algorithm 3.2's output is optimal for datasets containing $n-1$ symbols, $n > 2$. Then, given a dataset D containing n symbols, remove the two symbols s_i and s_j with smallest frequencies (weights) f_i and f_j, and replace them with a single symbol s_{ij} with frequency $f_{ij} = f_i + f_j$. What can you say about Algorithm 3.2's output on these new data?

(d) Attach to vertex s_{ij} two children s_i and s_j with weights f_i and f_j, respectively. Explain why this new tree would be produced by Algorithm 3.2.

(e) Explain why this new tree is an optimal code tree for a dataset of n symbols.

21. Suppose that we have a *fully balanced binary search tree*, that is, a balanced binary search tree (see footnote 29 on page 207) in which every vertex with no children is at the same level. Assume that a file is equally likely to be at any of the n vertices of T. What is the computational complexity of file search using T if we measure complexity using the average number of steps to find a file rather than the largest number of steps to find one.

22. In a complete m-ary rooted tree, if a vertex is chosen at random, show that the probability that it has a child is about $1/m$.

23. (Tucker [1984]) In a compiler, a control word is stored as a number. Suppose that the possible control words are GET, DO, ADD, FILL, STORE, REPLACE, and WAIT, and these are represented by the numbers 1, 2, 3, 4, 5, 6, and 7 (in binary notation), respectively. Given an unknown control word X, we wish to test it against the possible control words on this list until we find which word it is. One approach is to compare X's number in order to the numbers $1, 2, \ldots, 7$ corresponding to the control words. Another approach is to build a binary search tree. Describe how the latter approach would work and build such a tree.

24. The *binary search algorithm* searches through an ordered file to see if a key x is in the file. The entries of the file are n numbers, $x_1 < x_2 < \cdots < x_n$. The algorithm compares x to the middle entry x_i in the file, the $\lceil n/2 \rceil$th entry. If $x = x_i$, the search is done. If $x < x_i$, then $x_i, x_{i+1}, \ldots, x_n$ are eliminated from consideration, and the algorithm searches through the file $x_1, x_2, \ldots, x_{i-1}$, starting with the middle entry. If $x > x_i$, then $x_1, x_2, \ldots, x_i$ are eliminated from consideration, and the algorithm searches through the file $x_{i+1}, x_{i+2}, \ldots, x_n$, starting with the middle entry. The procedure is repeated iteratively. Table 3.5 shows two examples. (See Knuth [1973]

Table 3.5: Two Binary Searches[a]

Searching for 180	
71 97 164 180 285 <u>436</u> 513 519 522 622 663 687	Compare 180 to 436
71 97 <u>164</u> 180 285	Compare 180 to 164
<u>180</u> 285	Compare 180 to 180
	180 has been found
Searching for 515	
71 97 164 180 285 <u>436</u> 513 519 522 622 663 687	Compare 515 to 436
513 519 <u>522</u> 622 663 687	Compare 515 to 522
<u>513</u> 519	Compare 515 to 513
<u>519</u>	Compare 515 to 519
	Conclude that 515 is not in the file

[a] In each case, the first list contains the whole file, the underlined number is the middle entry, and each subsequent line shows the remaining file to be searched.

for details.) Apply the binary search algorithm to search for 150 in the following ordered files:

(a) 22 36 83 148 150 190 201 225 216 221 301

(b) 18 29 149 301 399 625 648 833 901 949

25. Compute the computational complexity of the binary search algorithm defined in Exercise 24. (*Hint:* Show that a binary search tree is being used.)

26. Apply bubble sort to the order 4312 and illustrate the steps required to put this in proper order.

27. Apply bubble sort to the following orders.

(a) 516324 (b) 346152

28. In the partial decision tree for bubble sort in Figure 3.88, fill in the part beginning at the vertex labeled:

(a) * (b) **

29. Compare the worst-case complexity of the bubble sort algorithm to the bound in Theorem 3.27 for p equal to:

(a) 7 (b) 15

30. In the sorting algorithm known as *quik sort*, we start with an ordered list of p items and try to find the proper order. We select the first item from the list and divide the remaining items into two groups, those less than the item selected and those greater than it. We then put the item chosen in between the two groups and repeat the process on each group. We eventually get down to groups of one element, and we stop. For instance, given the order 31564827, we select item 3. Then the items before 3 are listed as 12, those after 3 as 56487. We now apply the algorithm to

sort these two groups. For instance, choosing 5 from the second group gives us the two subgroups 4 and 687. We now order these. And so on. Apply quik sort to the following ordered lists:

(a) 5176324 (b) 941258376

31. How many steps (comparisons) does the algorithm quik sort (Exercise 30) require in the worst case if we start with a list $123\cdots p$ and $p = 5$?
32. Repeat Exercise 31 for arbitrary p.
33. Prove Theorem 3.26 by induction on h.
34. Find the keyword tree corresponding to the set of patterns {sentry, seldom, spackle, spanking, spanning, seller}.
35. Find the keyword tree corresponding to the set of patterns {ATTCG, AATGC, ATGCC, AATTT, AATGTG} on the alphabet $\Sigma = \{\text{A, T, G, C}\}$.
36. For each of the following matrices M of 0's and 1's, determine if the perfect phylogeny problem has a solution and, if so, find a phylogenetic tree for M.

(a) $M = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{array}$

(b) $M = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \end{pmatrix} \end{array}$

(c) $M = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix} \end{array}$

37. Prove that if the matrix M of 0's and 1's has a phylogenetic tree, then for j and k, either O_j and O_k are disjoint or one contains the other.
38. In the *dictionary problem*, a set of patterns forming a dictionary is known. When a pattern is presented, we want to find out if it is in the dictionary. Explain how keyword trees might help with this problem.
39. Suppose that a character can take any one of the states $0, 1, 2, \ldots, s$. Let M be an $n \times m$ matrix with entries from $\{0, 1, 2, \ldots, s\}$ and $M(i, j) = p$ if species i has character j in state p. A *perfect phylogeny* for M is a rooted tree where each species labels exactly one leaf, and edges are labeled with ordered triples (j, p, q), meaning that along that edge, character j changes state from p to q. Assume that the starting state for each character at the root is given and that for any character j and any y in $\{0, 1, 2, \ldots, s\}$, there is at most one edge on any chain from the root to a leaf that has the form (j, x, y) for some y. (A character may only change to state y once on this chain.)

(a) If $s = 2$ and $M = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cc} a & b \end{array} \\ \begin{pmatrix} 2 & 0 \\ 1 & 2 \\ 1 & 1 \\ 2 & 2 \end{pmatrix} \end{array}$, find a perfect phylogeny for M.

(b) If $s = 2$ and $M = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccc} a & b & c \end{array} \\ \begin{pmatrix} 1 & 0 & 0 \\ 2 & 2 & 2 \\ 0 & 0 & 0 \\ 0 & 1 & 2 \\ 2 & 2 & 0 \end{pmatrix} \end{array}$, find a perfect phylogeny for M.

40. Suppose that D is a digraph. A *level assignment* is an assignment of a level L_i to each vertex i so that if (i, j) is an arc, then $L_i < L_j$. Show that a digraph has a level assignment if and only if it has no cycles.

41. A *graded level assignment* for a digraph D is a level assignment (Exercise 40) such that if there is an arc (i, j) in D, then $L_j = L_i + 1$. Show that if $D = (V, A)$ has a graded level assignment, then it is *equipathic*; that is, for all u, v in V, all simple paths from u to v have the same length.

42. Suppose that T is a rooted tree. Orient T by directing each edge $\{u, v\}$ from a lower level to a higher level. The resulting digraph is called a *directed rooted tree*.

    (a) Show that every directed rooted tree has exactly one vertex from which every other vertex is reachable by a path.

    (b) Can a directed rooted tree be unilaterally connected?

    (c) Show that every directed rooted tree has a level assignment.

    (d) Suppose that D is a directed rooted tree. Show that D has a graded level assignment if and only if D is equipathic.

43. Find the number of rooted, labeled trees of:

    (a) Five vertices, two having degree 2

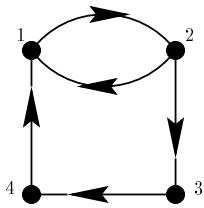
    (b) Five vertices in which the root has degree 2

    (c) Four vertices in which the root has degree 1

3.7 REPRESENTING A GRAPH IN THE COMPUTER

Many problems in graph theory cannot be solved except on the computer. Indeed, the development of high-speed computing machines has significantly aided the solution of graph-theoretical problems—for instance, the four-color problem—and it has also aided the applications of graph theory to other disciplines. At the same time, the development of computer science has provided graph theorists with a large number of important and challenging problems to solve. In this section we discuss various ways to represent a digraph or graph as input to a computer program. It also allows us to demonstrate how another area of mathematics, matrix theory, can aid in graph-theoretic problems.

Note that a diagram of a digraph or graph such as the diagrams we have used is not very practical for large digraphs or graphs and also not very amenable for input into a computer. Rather, some alternative means of entering digraphs and graphs are required. The best way to input a digraph or graph depends on the properties



D

$$\mathbf{A}(D) = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

Figure 3.94: A digraph with its adjacency matrix $\mathbf{A}(D)$.

of the digraph or graph and the use that will be made of it. The efficiency of an algorithm depends on the choice of method for entering a digraph or graph, and the memory storage required depends on this as well. Here, we shall mention a number of different ways to input a digraph or graph.

Let us concentrate on entering digraphs, recalling that graphs can be thought of as special cases. One of the most common approaches to entering a digraph D into a computer is to give its *adjacency matrix* $\mathbf{A} = \mathbf{A}(D)$. This matrix is obtained by labeling the vertices of D as $1, 2, \ldots, n$, and letting the i, j entry of $\mathbf{A}$ be 1 if there is an arc from i to j in D, and 0 if there is no such arc. Figure 3.94 shows a digraph and its associated adjacency matrix.[30] Note that the adjacency matrix can be entered as a two-dimensional array or as a bit string with n^2 bits, by listing one row after another. Thus, n^2 bits of storage are required, $n(n-1)$ if the diagonal elements are assumed to be 0, which is the case for loopless digraphs. The storage requirements are less for a graph, thought of as a symmetric digraph, since its adjacency matrix is symmetric. Specifically, if there are no loops, we require

$$(n-1) + (n-2) + \cdots + 1 = \frac{n(n-1)}{2}$$

bits of storage. For we only need to encode the entries above the diagonal. If a digraph is very sparse, that is, has few arcs, the adjacency matrix is not a very useful representation.

Another matrix that is useful for entering graphs, although not digraphs, is called the incidence matrix. In general, suppose that S is a set and $\mathcal{F}$ is a family of subsets of S. Define the *point-set incidence matrix* $\mathbf{M}$ as follows. Label the elements of S as $1, 2, \ldots, n$ and the sets in $\mathcal{F}$ as $S_1, S_2, \ldots, S_m$. Then $\mathbf{M}$ is an $n \times m$ matrix and its i, j entry is 1 if element i is in set S_j and 0 if element i is not in set S_j. For example, if $S = \{1, 2, 3, 4\}$ and

$$\mathcal{F} = \{\{1, 2\}, \{2, 3\}, \{1, 4\}\},$$

[30] Of course, the specific matrix obtained depends on the way we list the vertices. But by an abuse of language, we call any of these matrices *the* adjacency matrix.

Figure 3.95: The graph whose incidence matrix is given by (3.26).

then $\mathbf{M}$ is given by

$$\mathbf{M} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{ccc} \{1,2\} & \{2,3\} & \{1,4\} \end{array} \\ \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \end{array}. \tag{3.26}$$

If G is a graph, its *incidence matrix* is the point-set incidence matrix for the set $S = V(G)$ and the set $\mathcal{F} = E(G)$. The matrix $\mathbf{M}$ of (3.26) is the incidence matrix for the graph of Figure 3.95. An incidence matrix requires $n \times e$ bits of storage, where e is the number of edges. This tends to be larger than $\frac{1}{2}n(n-1)$, the storage required for the adjacency matrix, and it is always at least $n(n-1) = n^2 - n$ for connected graphs. (Why?) Incidence matrices are widely used in inputting electrical networks and switching networks. It is possible to define an incidence matrix for digraphs as well (see Exercise 20).

Still another approach to inputting a digraph D is simply to give an *arc list*, a list of its arcs. How much storage is required here? Suppose that each vertex label is encoded by a bit string of length t. Then if D has a arcs, since each arc is encoded as a pair of codewords, $2at$ bits of storage are required. Note that t must be large enough so that 2^t, the number of bit strings of length t, is at least n, the number of vertices. Thus, $t \geq \log_2 n$. In fact,

$$t \geq \lceil \log_2 n \rceil,$$

where $\lceil x \rceil$ is the least integer greater than or equal to x. Hence, we require at least $2a\lceil \log_2 n \rceil$ bits of storage. If there are not many arcs, that is, if the adjacency matrix is sparse, this can be a number smaller than n^2 and so require less storage than the adjacency matrix.

There are two variants on the arc list. The first is to input two linear orders or arrays, each having a codewords where a is the size of the arc set. If these arrays are $(h_1, h_2, \ldots, h_a)$ and $(k_1, k_2, \ldots, k_a)$, and h_i encodes for the vertex x and k_i for the vertex y, then (x, y) is the ith arc on the arc list. The storage requirements are the same as for an arc list.

Another variant on the arc list is to give n *adjacency lists*, one for each vertex x of D, listing the vertices y such that (x, y) is an arc. The n lists are called an *adjacency structure*. An adjacency structure requires $(n+a)t$ bits of storage, where t is as for the arc list. This is because the adjacency list for a vertex x must be encoded by encoding x and then encoding all the y's so that x is adjacent to y.

As we have already pointed out, the best way to input a digraph or a graph into a computer will depend on the application for which we use the input. To give a simple example, suppose that we ask the question: Is (u_i, u_j) an arc? This question

Table 3.6: An Adjacency Structure for a Digraph

Vertex x:	1	2	3	4	5	6
Vertices adjacent to x:	2, 4	1, 3, 4	2	1, 5, 6	3	

can be answered in one step from an adjacency matrix (look at the i, j entry). To answer it from an adjacency structure requires $\deg(u_i)$ steps in the worst case if u_j is the last vertex to which there is an arc from u_i. On the other hand, let us consider an algorithm that requires marking all vertices to which there is an arc from u_i. This requires n steps from an adjacency matrix (look at all entries in the ith row), but only $\deg(u_i)$ steps from an adjacency structure.

EXERCISES FOR SECTION 3.7

1. For each digraph of Figure 3.7, find:
   (a) Its adjacency matrix
   (b) An arc list
   (c) Two linear arrays that can be used to input the arcs
   (d) An adjacency structure
2. For each graph of Figure 3.23, find an incidence matrix.
3. Draw the digraph whose adjacency matrix is given by

$$\text{(a) } \mathbf{A} = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 \end{pmatrix}; \qquad \text{(b) } \mathbf{A} = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

4. Find the digraph whose vertices are a, b, c, d, e, f and whose arc list is

$$\{(b, a), (f, a), (b, d), (e, c), (e, b), (c, a), (c, e), (a, d)\}.$$

5. Find the digraph whose vertices are a, b, c, d, e and whose arcs are encoded by the two linear arrays (a, b, c, c, d, d, e), (b, d, a, d, c, e, a).
6. Find the digraph whose adjacency structure is given by Table 3.6.
7. Calculate the adjacency matrix for each of the graphs of Figure 3.9. (The adjacency matrix is the adjacency matrix of the corresponding digraph.)
8. Find the point-set incidence matrix for graph G of Figure 3.76 if:
   (a) $S = V(G)$ and $\mathcal{F}$ = all spanning trees of G
   (b) $S = E(G)$ and $\mathcal{F}$ = all spanning trees of G

9. If D^c is the complementary digraph of D (Exercise 34, Section 3.2), what is $\mathbf{A}(D) + \mathbf{A}(D^c)$?

10. Suppose that it requires one step to scan an entry in a list or an array. Suppose that a digraph is stored as an adjacency matrix.

    (a) How many steps are required to mark all vertices x adjacent to a particular vertex y, that is, such that (x, y) is an arc?

    (b) How many steps are required to mark or count all arcs?

11. Repeat Exercise 10 if the digraph is stored as an arc list.

12. Repeat Exercise 10 if the digraph is stored as an adjacency structure.

13. If D is a digraph of n vertices, its *reachability matrix* is an $n \times n$ matrix $\mathbf{R}$ whose i, j entry r_{ij} is 1 if vertex j is reachable from vertex i by a path, and 0 otherwise. For each digraph of Figure 3.7, find its reachability matrix. (Note that i is always reachable from i.)

14. What is the relationship between the reachability matrix of D, $\mathbf{R}(D)$, and $\mathbf{R}(D^c)$?

15. Show that D is strongly connected if and only if its reachability matrix (Exercise 13) is $\mathbf{J}$, the matrix of all 1's.

16. If D is a digraph with adjacency matrix $\mathbf{A}$, show by induction on k that the i, j entry of $\mathbf{A}^k$ gives the number of paths of length k in D that lead from i to j.

17. If G is a graph with adjacency matrix $\mathbf{A}$ (Exercise 7), what is the interpretation in graph (as opposed to digraph) language of the i, j entry of $\mathbf{A}^k$ (Exercise 16)?

18. Suppose that a square 0-1 symmetric matrix has 0's on its diagonal. Is it necessarily the adjacency matrix of some graph?

19. For digraphs D_1, D_2, and D_7 of Figure 3.7, use the result of Exercise 16 to find the number of paths of length 3 from u to v. Identify the paths.

20. If D is a digraph, its *incidence matrix* has rows corresponding to vertices and columns to arcs, with i, j entry equal to 1 if j is the arc (i, k) for some k, -1 if j is the arc (k, i) for some k, and 0 otherwise.

    (a) Find the incidence matrix for each digraph of Figure 3.7.

    (b) How many bits of storage are required for the incidence matrix?

    (c) If $\mathbf{M} = (m_{ij})$ is the incidence matrix of digraph D, what is the significance of the matrix $\mathbf{N} = (n_{ij})$, where $n_{ij} = \sum_k m_{ik} m_{jk}$?

    (d) If D is strongly connected, can we ever get away with fewer bits of storage than are needed for the adjacency matrix?

21. If $\mathbf{M} = (m_{ij})$ is any matrix of nonnegative entries, let $B(\mathbf{M})$ be the matrix whose i, j entry is 1 if $m_{ij} > 0$ and 0 if $m_{ij} = 0$. Show that if D is a digraph of n vertices with reachability matrix $\mathbf{R}$ and adjacency matrix $\mathbf{A}$, and $\mathbf{I}$ is the identity matrix, then:

    (a) $\mathbf{R} = B[\mathbf{I} + \mathbf{A} + \mathbf{A}^2 + \cdots + \mathbf{A}^{n-1}]$ (b) $\mathbf{R} = B[(\mathbf{I} + \mathbf{A})^{n-1}]$

22. Check the results of Exercise 21 on the digraphs D_1 and D_7 of Figure 3.7.

23. (a) Show that D is unilaterally connected (Exercise 9, Section 3.2) if and only if $B(\mathbf{R}+\mathbf{R}^T)=\mathbf{J}$, where B is defined in Exercise 21, $\mathbf{R}$ in Exercise 13, and $\mathbf{J}$ in Exercise 15, and where $\mathbf{R}^T$ is the transpose of $\mathbf{R}$.

    (b) Show that D is weakly connected (Exercise 10, Section 3.2) if and only if we have $B[(\mathbf{I}+\mathbf{A}+\mathbf{A}^T)^{n-1}]=\mathbf{J}$.

24. Suppose that D is a digraph with reachability matrix (Exercise 13) $\mathbf{R}=(r_{ij})$ and that $\mathbf{R}^2$ is the matrix (s_{ij}). Show that:

    (a) The strong component (Exercise 13, Section 3.2) containing vertex i is given by the entries of 1 in the ith row of $\mathbf{T}=(t_{ij})$, where $t_{ij}=r_{ij}\times r_{ij}^{(T)}$ and $r_{ij}^{(T)}$ is the i,j entry of the transpose of $\mathbf{R}$.

    (b) The number of vertices in the strong component containing i is s_{ii}.

25. For each digraph of Figure 3.7, use the results of Exercise 24 to find the strong components.

26. If $\mathbf{R}$ is the reachability matrix of a digraph and $c(i)$ is the ith column sum of $\mathbf{R}$, what is the interpretation of $c(i)$?

27. If G is a graph, how would you define directly its reachability matrix $\mathbf{R}(G)$?

28. If G is a graph, $\mathbf{R}=\mathbf{R}(G)$ is its reachability matrix (Exercise 27), and $\mathbf{T}$ is as defined in Exercise 24:

    (a) Show that $\mathbf{T}=\mathbf{R}$.

    (b) What is the interpretation of the 1, 1 entry of $\mathbf{R}^2$?

29. Suppose that $\mathbf{R}$ is a matrix of 0's and 1's with 1's down the diagonal (and perhaps elsewhere). Is $\mathbf{R}$ necessarily the reachability matrix of some digraph? (Give a proof or counterexample.)

30. (Harary [1969]) Suppose that $\mathbf{B}$ is the incidence matrix of a graph G and $\mathbf{B}^T$ is the transpose of $\mathbf{B}$. What is the significance of the i,j entry of the matrix $\mathbf{B}^T\mathbf{B}$?

31. (Harary [1969]) Let G be a graph. The *circuit matrix* $\mathbf{C}$ of G is the point-set incidence matrix with S the set of edges of G and $\mathcal{F}$ the family of circuits of G. Let $\mathbf{B}$ be the incidence matrix of G. Show that every entry of $\mathbf{BC}$ is $\equiv 0 \pmod 2$.

32. Can two graphs have the same incidence matrix and be nonisomorphic? Why?

33. Can two graphs have the same circuit matrix and be nonisomorphic? Why? What if every edge is on a circuit?

3.8 RAMSEY NUMBERS REVISITED

Ramsey theory and, in particular, Ramsey numbers $R(p,q)$ were introduced in Section 2.19.3. To study Ramsey numbers, it is convenient to look at them using graph theory. If G is a graph, its *complement* G^c is the graph with the same vertex set as G, and so that for all $a\neq b$ in $V(G)$, $\{a,b\}\in E(G^c)$ if and only if $\{a,b\}\notin E(G)$. In studying the Ramsey numbers $R(p,q)$, we shall think of a set S as the vertex set of a graph, and of a set of 2-element subsets of S as the edge set

of this graph. Then, to say that a number N has the (p, q) Ramsey property means that whenever S is a set of N elements and we have a graph G with vertex set S, then if we divide the 2-element subsets of S into edges of G and edges of G^c (edges not in G), either there are p vertices all of which are joined by edges in G or there are q vertices all of which are joined by edges in G^c. Put another way, we have the following theorem.

Theorem 3.29 A number N has the (p, q) Ramsey property if and only if for every graph G of N vertices, either G has a complete p-gon (complete subgraph of p vertices, K_p) or G^c has a complete q-gon.

Corollary 3.29.1 $R(p, 2) = p$ and $R(2, q) = q$.

Proof. For every graph of p vertices, either it is complete or its complement has an edge. This shows that $R(p, 2) \leq p$. Certainly, $R(p, 2) \geq p$. (Why?) Q.E.D.

In the language of Theorem 3.29, Theorem 2.18 says that if G is a graph of 6 (or more) vertices, then either G has a triangle or G^c has a triangle. (The reader should try this out on a number of graphs of his or her choice.) More generally, every graph of (at least) $R(p, q)$ vertices has either a complete p-gon (K_p) or its complement has a complete q-gon (K_q).

Consider now the graph $G = Z_5$, the circuit of length 5. Now G^c is again (isomorphic to) Z_5. Thus, neither G nor G^c has a triangle. We conclude that the number 5 does not have the $(3, 3)$ Ramsey property, so $R(3, 3) > 5$. This completes the proof of the following theorem.

Theorem 3.30 $R(3, 3) = 6$.

To use the terminology of Section 3.3.1, a complete p-gon in G is a clique of p vertices. A complete q-gon in G^c corresponds to an independent set of q vertices in G. Thus, we can restate Theorem 3.29 as follows.

Theorem 3.31 A number N has the (p, q) Ramsey property if and only if for every graph G of N vertices, either G has a clique of p vertices or G has an independent set of q vertices.

There is still another way to restate Theorem 3.29, which is as follows.

Theorem 3.32 A number N has the (p, q) Ramsey property if and only if whenever we color the *edges* of K_N, the complete graph on N vertices, with each edge being colored either red or blue, then K_N has a complete red p-gon or K_N has a complete blue q-gon.

Proof. Given an edge coloring, let G be the graph whose vertices are the same as those of K_N and whose edges are the red edges. Q.E.D.

We have already observed that Ramsey numbers are difficult to compute. The few Ramsey numbers that have been determined are given in Table 2.12. To verify (at least partly) some of the numbers in Table 2.12, one can derive some bounds

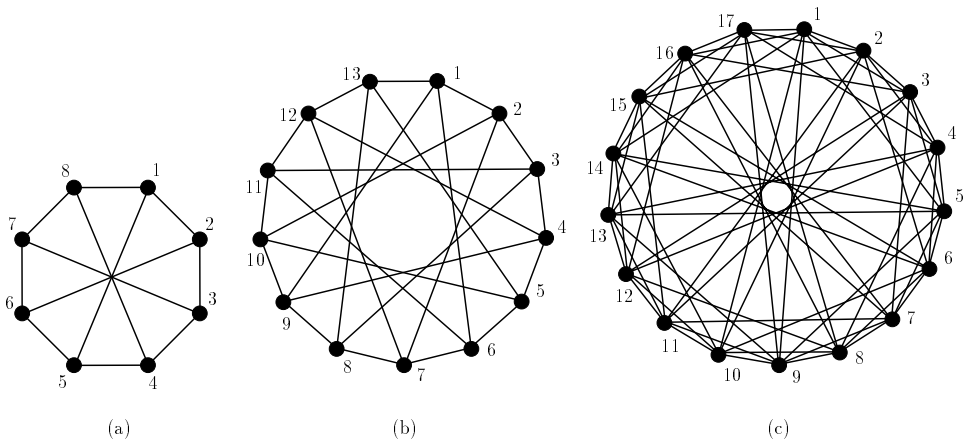


Figure 3.96: Graphs that demonstrate (a) $R(3,4) \geq 9$, (b) $R(3,5) \geq 14$, and (c) $R(4,4) \geq 18$.

on the Ramsey numbers. Consider graph (a) of Figure 3.96. This graph has 8 vertices. It also has no triangle (3-gon) and no independent set of 4 vertices. Thus, by Theorem 3.31, the number 8 does not have the $(3,4)$ Ramsey property. It follows that $R(3,4) \geq 9$. Similarly, graphs (b) and (c) of Figure 3.96 show that $R(3,5) \geq 14$ and $R(4,4) \geq 18$ (Exercises 2 and 3).

Finally, variants of Ramsey numbers present difficult challenges in graph theory. For discussion of one such topic, that of graph Ramsey numbers, see Exercises 11–13.

EXERCISES FOR SECTION 3.8

1. Show that $R(p,2) \geq p$.
2. Use Figure 3.96(b) to show that $R(3,5) \geq 14$.
3. Use Figure 3.96(c) to show that $R(4,4) \geq 18$.
4. Let G be a complete graph on 25 vertices and let the edges of G be colored either brown or green. If there is no green triangle, what is the largest complete brown m-gon you can be sure G has?
5. For each of the graphs of Figure 3.97, either find a clique of 3 vertices or an independent set of 3 vertices, or conclude that neither of these can be found.
6. Let G be any graph of 11 vertices and chromatic number 3.
   (a) Does G necessarily have either a clique of 3 vertices or an independent set of 3 vertices?
   (b) Does G necessarily have either a clique of 4 vertices or an independent set of 3 vertices?
7. Let G be a graph of 16 vertices and largest clique of size 3.

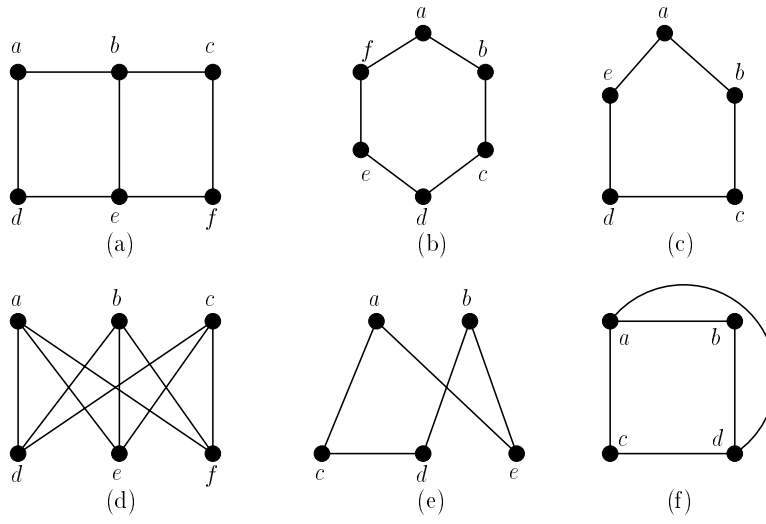


Figure 3.97: Graphs for Exercise 5, Section 3.8.

(a) Does G necessarily have an independent set of 4 vertices?

(b) Of 5 vertices?

8. Color the edges of K_{10} either red or blue.

   (a) Show that if there are at least 4 red edges from one vertex, there are 3 vertices all joined by red edges or 4 vertices all joined by blue edges.

   (b) Similarly, if there are at least 6 blue edges from a vertex, show that either there are 3 vertices all joined by red edges or 4 vertices all joined by blue edges.

   (c) Show that by parts (a) and (b), K_{10} has in any coloring of its edges with red or blue colors either 3 vertices all joined by red edges or 4 vertices all joined by blue edges.

   (d) Does part (c) tell you anything about a Ramsey number?

9. Let G be a tree of 20 vertices.

   (a) Does G necessarily have an independent set of 5 vertices?

   (b) Of 6 vertices?

10. Color the edges of the graph K_{17} in red, white, or blue. This exercise will argue that there are 3 vertices all joined by edges of the same color.

    (a) Fix one vertex a. Show that of the edges joining this vertex, at least 6 must have the same color.

    (b) Suppose that the 6 edges in (a) are all red. These lead from a to 6 vertices, b, c, d, e, f, and g. Argue from here that either K_{17} has a red triangle, a blue triangle, or a white triangle.

(c) What does the result say about the Ramsey numbers $R(p, q; r)$ defined in Exercise 33 of Section 2.19.

11. Let G_1 and G_2 be graphs. An integer N is said to have the *graph Ramsey property* (G_1, G_2) if every coloring of the edges of the complete graph K_N in the colors 1 and 2 gives rise, for some i, to a subgraph that is (isomorphic to) G_i and is colored all in color i, that is, to a monochromatic G_i, for $i = 1$ or 2. The *graph Ramsey number* $R(G_1, G_2)$ is the smallest N with the graph Ramsey property (G_1, G_2). (It is not hard to show that this is well defined. See Chartrand and Lesniak [1996] or Graham, Rothschild, and Spencer [1990].) If L_p is the chain of p vertices and Z_q is the circuit of q vertices:

(a) Show that $R(L_3, L_3) = 3$. (b) Show that $R(L_4, L_4) = 5$.

(c) Find $R(L_3, L_4)$. (d) Find $R(L_3, Z_4)$.

(e) Find $R(L_4, Z_4)$. (f) Find $R(Z_4, Z_4)$.

12. (Chvátal and Harary [1972]) Let $c(G)$ be the size of the largest connected component of G. Show that

$$R(G, H) \geq (\chi(G) - 1)\,(c(H) - 1) + 1.$$

13. (Chvátal [1977]) If T_m is a tree on m vertices, show that

$$R(T_m, K_n) = 1 + (m - 1)(n - 1).$$

REFERENCES FOR CHAPTER 3

Agarwala, R., and Fernandez-Baca, D., "A Polynomial-time Algorithm for the Perfect Phylogeny Problem When the Number of Character States Is Fixed," *SIAM J. Comput., 23* (1994), 1216–1224.

Agarwala, R., Bafna, V., Farach, M., Narayanan, B., Paterson, M., and Thorup, M., "On the Approximability of Numerical Taxonomy: Fitting Distances by Tree Metrics," *Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 1996.

Aho, A. V., Hopcroft, J. E., and Ullman, J. D., *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.

Ahuja, R. K., Magnanti, T. L., and Orlin, J. B., *Network Flows: Theory, Algorithms, and Applications*, Prentice Hall, Englewood Cliffs, NJ, 1993.

Alon, N., "Restricted Colorings of Graphs," in K. Walker (ed.), *Surveys in Combinatorics*, London Mathematical Society Lecture Note Series, 187, Cambridge University Press, Cambridge, 1993, 1–33.

Appel, K., and Haken, W., "Every Planar Map Is Four Colorable. Part I: Discharging," *Ill. J. Math., 21* (1977), 429–490.

Appel, K., Haken, W., and Koch, J., "Every Planar Map Is Four Colorable. Part II: Reducibility," *Ill. J. Math., 21* (1977), 491–567.

Baase, S., *Computer Algorithms*, 2nd ed., Addison-Wesley Longman, Reading, MA, 1992.

Balaban, A. T. (ed.), *Chemical Applications of Graph Theory*, Academic Press, New York, 1976.

Beltrami, E. J., and Bodin, L. D., "Networks and Vehicle Routing for Municipal Waste Collection," *Networks, 4* (1973), 65–94.

BERGE, C., "Färbung von Graphen, deren sämtliche bzw. deren ungerade Kreise starr sind," *Wiss. Z. Martin-Luther-Univ. Halle-Wittenberg, Math.-Naturwiss. Reihe, 10* (1961), 114.

BERGE, C., "Sur une conjecture relative au problème des codes optimaux," *Commun. 13ème Assemblèe Gènèrale de l'URSI* (International Scientific Radio Union), Tokyo, 1962.

BERN, M. W., and GRAHAM, R. L., "The Shortest-Network Problem," *Scientific American, 260* (1989), 84–89.

BIRKHOFF, G. D., "A Determinant Formula for the Number of Ways of Coloring a Map," *Ann. Math., 14* (1912), 42–46.

BIRKHOFF, G. D., and LEWIS, D. C., "Chromatic Polynomials," *Trans. Amer. Math. Soc., 60* (1946), 355–451.

BODLAENDER, H. L., "Polynomial Algorithms for Graph Isomorphism and Chromatic Index on Partial k-Trees," *J. Algorithms, 11* (1990), 631–643.

BODLAENDER, H. L., FELLOWS, M., and WARNOW, T., "Two Strikes Against Perfect Phylogeny," *Proc. of the 19th Int. Colloq. on Automata, Languages, and Programming*, (1992), 273–283.

BONDY, J. A., and MURTY, U. S. R., *Graph Theory with Applications*, Elsevier, New York, 1976.

BONIAS, I., "T-colorings of Complete Graphs," Ph.D. thesis, Northeastern University, 1991.

BOOTH, K. S., and LUEKER, G. S., "Testing for the Consecutive Ones Property, Interval Graphs, and Graph Planarity Using $P\,Q$-tree Algorithms," *J. Comp. Syst. Sci., 13* (1976), 335–379.

BRASSARD, G., and BRATLEY, P., *Algorithmics: Theory and Practice*, Prentice Hall, Upper Saddle River, NJ, 1995.

BRIGHAM, R. C., and DUTTON, R. D., "Generalized k-Tuple Colorings of Cycles and Other Graphs," *J. Comb. Theory, Series B, 32* (1982), 90–94.

BUNEMAN, P., "The Recovery of Trees from Measures of Dissimilarity," in F. R. Hodson, D. G. Kendall, and P. Tautu (eds.), *Mathematics in the Archaeological and Historical Sciences*, Edinburgh University Press, Edinburgh, 1971, 387–395.

CAYLEY, A., "On the Theory of the Analytical Forms Called Trees," *Philos. Mag., 13* (1857), 172–176. [Also *Math. Papers*, Cambridge, *3* (1891), 242–246.]

CAYLEY, A., "On the Mathematical Theory of Isomers," *Philos. Mag., 67* (1874), 444–446. [Also *Math. Papers*, Cambridge, *9* (1895), 202–204.]

CAYLEY, A., "A Theorem on Trees," *Quart. J. Math., 23* (1889), 376–378. [Also *Math. Papers*, Cambridge, *13* (1897), 26–28.]

CHARTRAND, G., and LESNIAK, L., *Graphs and Digraphs*, 3rd ed., CRC Press, Boca Raton, 1996.

CHUDNOVSKY, M., ROBERTSON, N., SEYMOUR, P. D., and THOMAS, R., "The Strong Perfect Graph Theorem," manuscript, (2002).

CHVÁTAL, V., "Tree-Complete Graph Ramsey Numbers," *J. Graph Theory, 1* (1977), 93.

CHVÁTAL, V., and HARARY, F., "Generalized Ramsey Theory for Graphs," *Bull. Amer. Math. Soc., 78* (1972), 423–426.

CIESLIK, D., *Steiner Minimal Trees*, Kluwer Academic Publishers, Norwell, MA, 1998.

COHEN, J. E., *Food Webs and Niche Space*, Princeton University Press, Princeton, NJ, 1978.

COHEN, J., and FARACH, M., "Numerical Taxonomy on Data: Experimental Results,"

J. Comput. Biol., 4 (1997), 547–558.

Cozzens, M. B., and Roberts, F. S., "T-Colorings of Graphs and the Channel Assignment Problem," *Congressus Numerantium, 35* (1982), 191–208.

Demourcron, G., Malgrance, V., and Pertuiset, R., "Graphes planaires: reconnaissance et construction des représentations planaires topologiques," *Rev. Française Recherche Opérationnelle, 8* (1964), 33–47.

Deo, N., *Graph Theory with Applications to Engineering and Computer Science*, Prentice Hall, Englewood Cliffs, NJ, 1974.

Erdős, P., Rubin, A. L., and Taylor, H., "Choosability in Graphs," *Congressus Numerantium, 26* (1979), 125–157.

Erdős, P. L., Steel, M. A., Székely, L. A., and Warnow, T. J., "Constructing Big Trees from Short Sequences," in P. Degano, R. Gorrieri, and A. Marchetti-Spaccamela (eds.), *ICALP'97, 24th International Colloquium on Automata, Languages, and Programming (Silver Jubilee of EATCS)*, Bologna, Italy, July 7–11, 1997, *Lecture Notes in Computer Science*, Vol. 1256, Springer-Verlag, Berlin, 1997, 827–837.

Erdős, P. L., Steel, M. A., Székely, L. A., and Warnow, T. J., "A few logs suffice to build (almost) all trees. I," *Random Structures Algorithms, 14* (1999), 153–184.

Estabrook, G., Johnson, C., and McMorris, F. R., "An Idealized Concept of the True Cladistic Character," *Math. Bioscience, 23* (1975), 263–272.

Estabrook, G., Johnson, C. and McMorris, F. R., "A Mathematical Foundation for the Analysis of Cladistic Character Compatibility," *Math. Bioscience, 29* (1976), 181–187. (a)

Estabrook, G., Johnson, C., and McMorris, F. R., "An Algebraic Analysis of Cladistic Characters," *Discrete Math., 16* (1976), 141–147. (b)

Euler, L., "Solutio Problematis ad Geometriam Situs Pertinentis," *Comment. Acad. Sci. 1 Petropolitanae, 8* (1736), 128–140. [Reprinted in *Opera Omnia*, Series 1–7 (1766), 1–10.]

Even, S., *Graph Algorithms*, Computer Science Press, Potomac, MD, 1979.

Filliben, J. J., Kafadar, K., and Shier, D. R., "Testing for Homogeneity of Two-Dimensional Surfaces," *Math Modeling, 4* (1983), 167–189.

Fiorini, S., and Wilson, R. J., *Edge Colorings of Graphs*, Pitman, London, 1977.

Fitch, W. M., "An Introduction to Molecular Biology for Mathematicians and Computer Programmers," in M. Farach-Colton, F. S. Roberts, M. Vingron, and M. S. Waterman (eds.), *Mathematical Support for Moleclar Biology*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 47, American Mathematical Society, Providence, RI, 1999, 1–31.

Frank, A., "Connectivity and Network Flows," in R. L. Graham, M. Grötschel, and L. Lovász (eds.), *Handbook of Combinatorics*, Elsevier, Amsterdam, 1995, 111–177.

Garey, M. R., and Johnson, D. S., *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, 1979.

Geller, D. P., "r-Tuple Colorings of Uniquely Colorable Graphs," *Discrete Mathematics, 16* (1976), 9–12.

Gibbons, A., *Algorithmic Graph Theory*, Cambridge University Press, Cambridge, 1985.

Gilbert, E. N., Unpublished Technical Memorandum, Bell Telephone Labs, Murray Hill, NJ, 1972.

Goldman, A. J., "Discrete Mathematics in Government," lecture presented at SIAM Symposium on Applications of Discrete Mathematics, Troy, NY, June 1981.

Golumbic, M. C., *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New

York, 1980.

GRAHAM, R. L., and HELL, P., "On the History of the Minimum Spanning Tree Problem," *Annals of the History of Computing, 7* (1985), 43–57.

GRAHAM, R. L., ROTHSCHILD, B. L., and SPENCER, J. H., *Ramsey Theory*, 2nd ed., Wiley, New York, 1990.

GRAVIER, S., "Coloration et Produits de Graphes," Ph.D. thesis, Université Joseph Fourier, Grenoble, France, 1996.

GRIMALDI, D. A., *A Phylogenetic, Revised Classification of Genera in the Drosophilidae (Diptera)*, Bulletin of the American Museum of Natural History, American Museum of Natural History, New York, 1990.

GUSFIELD, D., "Efficient Algorithms for Inferring Evolutionary History," *Networks, 21* (1991), 19–28.

GUSFIELD, D., *Algorithms on Strings, Trees, and Sequences*, Cambridge University Press, New York, 1997.

HALE, W. K., "Frequency Assignment: Theory and Applications," *Proc. IEEE, 68* (1980), 1497–1514.

HALIN, R., "Bemerkungen Über Ebene Graphen," *Math. Ann., 53* (1964), 38–46.

HANSEN, P., FOWLER, P., and ZHENG, M. (eds.), *Discrete Mathematical Chemistry*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 51, American Mathematical Society, Providence, RI, 2000.

HARARY, F., *Graph Theory*, Addison-Wesley, Reading, MA, 1969.

HARARY, F., NORMAN, R. Z., and CARTWRIGHT, D., *Structural Models: An Introduction to the Theory of Directed Graphs*, Wiley, New York, 1965.

HARARY, F., and PALMER, E. M., *Graphical Enumeration*, Academic Press, New York, 1973.

HARARY, F., and TUTTE, W. T., "A Dual Form of Kuratowski's Theorem," *Canad. Math. Bull., 8* (1965), 17–20.

HARRISON, J. L., "The Distribution of Feeding Habits among Animals in a Tropical Rain Forest," *J. Anim. Ecol., 31* (1962), 53–63.

HOPCROFT, J. E., and TARJAN, R. E., "Efficient Planarity Testing," *J. ACM, 21* (1974), 549–568.

HUFFMAN, D. A., "A Method for the Construction of Minimum Redundancy Codes," *Proc. Inst. Rail. Eng., 40* (1952), 1098–1101.

IRVING, R. W., "NP-completeness of a Family of Graph-Colouring Problems," *Discrete Appl. Math., 5* (1983), 111–117.

JENSEN, T. R., and TOFT, B., *Graph Coloring Problems*, Wiley, New York, 1995.

KANG, A. N. C., LEE, R. C. T., CHANG, C. L., and CHANG, S. K., "Storage Reduction through Minimal Spanning Trees and Spanning Forests," *IEEE Trans. Comput., C-26* (1977), 425–434.

KANNAN, S., and WARNOW, T., "Inferring Evolutionary History from DNA Sequences," *SIAM J. Comput., 23* (1994), 713–737.

KANNAN, S., and WARNOW, T., "A Fast Algorithm for the Computation and Enumeration of Perfect Phylogenies When the Number of Character States Is Fixed," *6th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 1995, 595–603.

KELLY, J. B., and KELLY, L. M., "Paths and Circuits in Critical Graphs," *Amer. J. Math., 76* (1954), 786–792.

KEMENY, J. G., and SNELL, J. L., *Mathematical Models in the Social Sciences*, Blaisdell, New York, 1962. (Reprinted by MIT Press, Cambridge, MA, 1972.)

KIRCHHOFF, G., "Über die Auflösung der Gleichungen, auf welche man bei der Un-

tersuchung der linearen Verteilung galvanischer Ströme geführt wird," *Ann. Phys. Chem., 72* (1847), 497–508.

KLOTZ, W., "A Constructive Proof of Kuratowski's Theorem," *Ars Combinatoria, 28* (1989), 51–54.

KNUTH, D. E., *The Art of Computer Programming*, Vol. 3: *Sorting and Searching*, Addison-Wesley, Reading, MA, 1973.

KÖNIG, D., *Theorie des endlichen und unendlichen Graphen*, Akademische Verlagsgesellschaft, Leipzig, 1936. (Reprinted by Chelsea, New York, 1950.)

KRATOCHVÍL, J., TUZA, Z., and VOIGT, M., "New Trends in the Theory of Graph Colorings: Choosability and List Coloring," in R. L. Graham, J. Kratochvíl, J. Nešetřil, and F. S. Roberts (eds.), *Contemporary Trends in Discrete Mathematics*, DIMACS Series, 49, American Mathematical Society, Providence, RI, 1999, 183–195.

KREHER, D. L., and STINSON, D. R., *Combinatorial Algorithms: Generation, Enumeration, and Search*, CRC Press, Boca Raton, FL, 1998.

KURATOWSKI, K., "Sur le Problème des Courbes Gauches en Topologie," *Fund. Math., 15* (1930), 271–283.

LEHMER, D. H., "The Chromatic Polynomial of a Graph," *Pacific J. Math., 118* (1985), 463–469.

LIU, C. L., *Topics in Combinatorial Mathematics*, Mathematical Association of America, Washington, DC, 1972.

LIU, D.-F., "Graph Homomorphisms and the Channel Assignment Problem," Ph.D. thesis, University of South Carolina, 1991.

LOVÁSZ, L., "Normal Hypergraphs and the Perfect Graph Conjecture," *Discrete Math., 2* (1972), 253–267. (a)

LOVÁSZ, L., "A Characterization of Perfect Graphs," *J. Comb. Theory B, 13* (1972), 95–98. (b)

LUKS, E. M., "Isomorphism of Graphs of Bounded Valence Can Be Tested in Polynomial Time," *J. Comput. System Sci., 25* (1982), 42–65.

MACKENZIE, D., "Graph Theory Uncovers the Roots of Perfection," *Science, 297* (2002), 38.

MAHADEV, N. V. R., and ROBERTS, F. S., "Consensus List Colorings of Graphs and Physical Mapping of DNA," in M. Janowitz, F. R. McMorris, B. Mirkin, and F. S. Roberts (eds.), *Bioconsensus*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 61, American Mathematical Society, Providence, RI, 2003, 83–95.

MAKARYCHEV, Y., "A Short Proof of Kuratowski's Graph Planarity Criterion," *J. Graph Theory, 25* (1997), 129–131.

MANBER, U., *Introduction to Algorithms: A Creative Approach*, Addison-Wesley Longman, Reading, MA, 1989.

MCKEE, T. A., and BEINEKE, L. W., *Graph Theory in Computer Science, Chemistry, and Other Fields*, Pergamon Press, Exeter, UK, 1997.

MOON, J. W., "Various Proofs of Cayley's Formula for Counting Trees," in F. Harary (ed.), *A Seminar on Graph Theory*, Holt, Rinehart and Winston, New York, 1967, 70–78.

OPSUT, R. J., and ROBERTS, F. S., "On the Fleet Maintenance, Mobile Radio Frequency, Task Assignment, and Traffic Phasing Problems," in G. Chartrand, Y. Alavi, D. L. Goldsmith, L. Lesniak-Foster, and D. R. Lick (eds.), *The Theory and Applications of Graphs*, Wiley, New York, 1981, 479–492.

PESCHON, J., and ROSS, D., "New Methods for Evaluating Distribution, Automation,

and Control (DAC) Systems Benefits," *SIAM J. Algebraic Discrete Methods, 3* (1982), 439–452.

PONOMARENKO, I. N., "A Polynomial Isomorphism Algorithm for Graphs Not Contractible to $K_{3,g}$ (Russian; English summary)," *Zap. Nauchn. Sem. Leningrad. Otdel. Mat. Inst. Steklov. (LOMI), 137* (1984), 99–114.

PONOMARENKO, I. N., "Polynomial Time Algorithms for Recognizing and Isomorphism Testing of Cyclic Tournaments," *Acta Appl. Math., 29* (1992), 139–160.

PRIM, R. C., "Shortest Connection Networks and Some Generalizations," *Bell Syst. Tech. J., 36* (1957), 1389–1401.

RAYCHAUDHURI, A., "Intersection Assignments, T-Coloring, and Powers of Graphs," Ph.D. thesis, Rutgers University, 1985.

READ, R. C., "An Introduction to Chromatic Polynomials," *J. Comb. Theory, 4* (1968), 52–71.

REINGOLD, E. M., NIEVERGELT, J., and DEO, N., *Combinatorial Algorithms: Theory and Practice*, Prentice Hall, Englewood Cliffs, NJ, 1977.

ROBERTS, F. S., *Discrete Mathematical Models, with Applications to Social, Biological, and Environmental Problems*, Prentice Hall, Englewood Cliffs, NJ, 1976.

ROBERTS, F. S., *Graph Theory and Its Applications to Problems of Society*, NSF-CBMS Monograph No. 29, SIAM, Philadelphia, 1978.

ROBERTS, F. S., "Indifference and Seriation," *Ann. N.Y. Acad. Sci., 328* (1979), 173–182.

ROBERTS, F. S., "From Garbage to Rainbows: Generalizations of Graph Coloring and their Applications," in Y. Alavi, G. Chartrand, O. R. Oellermann, and A. J. Schwenk (eds.), *Graph Theory, Combinatorics, and Applications*, Vol. 2, Wiley, New York, 1991, 1031–1052.

ROBERTSON, N., SANDERS, D. P., SEYMOUR, P. D., and THOMAS, R., "The Four Colour Theorem," *J. Comb. Theory, Series B, 70* (1997), 2–44.

ROUVRAY, D. H., and BALABAN, A. T., "Chemical Applications of Graph Theory," in R. J. Wilson and L. W. Beinecke (eds.), *Applications of Graph Theory*, Academic Press, London, 1979, 177–221.

SCOTT, S. H., "Multiple Node Colourings of Finite Graphs," doctoral dissertation, University of Reading, England, March 1975.

SHIER, D. R., "Testing for Homogeneity using Minimum Spanning Trees," *UMAP J., 3* (1982), 273–283.

SHOR, P. W., "A New Proof of Cayley's Formula for Counting Labeled Trees," *J. Combin. Theory, Ser. A, 71* (1995), 154–158.

STEEL, M. A., "The Complexity of Reconstructing Trees from Qualitative Characters and Subtrees," *J. Classification, 9* (1992), 91–116.

TAKÁCS, L., "On Cayley's Formula for Counting Forests," *J. Combin. Theory, Ser. A, 53* (1990), 321–323.

TESMAN, B. A., "Complete Graph T-Spans," *Congressus Num., 35-A* (1993), 161–173.

TUCKER, A. C., "Perfect Graphs and an Application to Optimizing Municipal Services," *SIAM Rev., 15* (1973), 585–590.

TUCKER, A. C., *Applied Combinatorics*, 2nd ed., Wiley, New York, 1984.

TUTTE, W. T. (alias B. DESCARTES), "Solution to Advanced Problem No. 4526," *Amer. Math. Monthly, 61* (1954), 352.

VAN DEN HEUVEL, J., LEESE, R. A., and SHEPHERD, M. A., "Graph Labeling and Radio Channel Assignment," *J. Graph Theory, 29* (1998), 263–283.

WAGNER, K., "Über Eine Eigenschaft der Ebene Komplexe," *Math. Ann., 114* (1937),

570–590.

Wang, D.-I., "The Channel Assignment Problem and Closed Neighborhood Containment Graphs," Ph.D. thesis, Northeastern University, 1985.

Welsh, D., and Whittle, G. P., "Arrangements, Channel Assignments, and Associated Polynomials," *Adv. Appl. Math., 23* (1999), 375–406.

West, D. B., *Introduction to Graph Theory*, Prentice Hall, Upper Saddle River, NJ, 2001.

Whitney, H., "The Coloring of Graphs," *Ann. Math., 33* (1932), 688–718.

Wilkinson, E. M., "Archaeological Seriation and the Traveling Salesman Problem," in F. R. Hodson, *et al.* (eds.), *Mathematics in the Archaeological and Historical Sciences*, Edinburgh University Press, Edinburgh, 1971.

Zykov, A. A., "On Some Properties of Linear Complexes (Russian)," *Mat. Sbornik N.S., 24* (1949), 163–188. (English transl.: *Amer. Math. Soc. Transl., 1952*, (1952), 33 pp.)