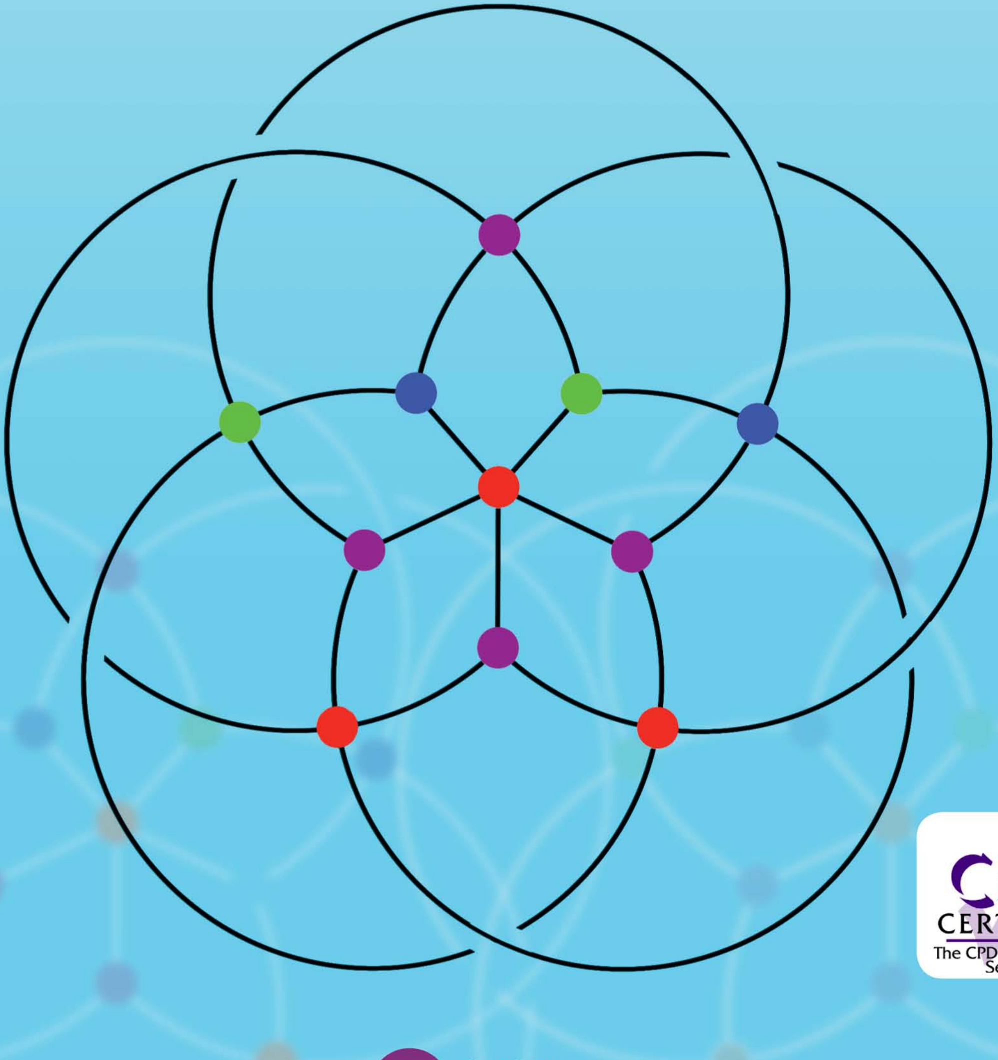
TEXTBOOKS IN MATHEMATICS

GRAPH THEORY AND ITS APPLICATIONS

THIRD EDITION

Jonathan L. Gross
Jay Yellen
Mark Anderson



CRC Press
Taylor & Francis Group
A CHAPMAN & HALL BOOK

Graph Theory and Its Applications, Third Edition

Graph Theory and Its Applications, Third Edition

Jonathan L. Gross
Jay Yellen
Mark Anderson

CRC Press
Taylor & Francis Group
Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an informa business

CRC Press
Taylor & Francis Group
6000 Broken Sound Parkway NW, Suite 300
Boca Raton, FL 33487-2742

Printed on acid-free paper
Version Date: 20191011

International Standard Book Number-13: 978-1-4822-4948-4 (Hardback)

Jonathan dedicates this book to Alisa.

Jay dedicates this book to the memory of his brother Marty.

Mark dedicates this book to Libertad.

CONTENTS

PREFACE

Graphs have proven to be useful models for a wide variety of problems, arising in computer science, operations research, and in the natural and social sciences. This text targets the need for a comprehensive approach to the theory, integrating a careful exposition of classical developments with emerging methods, models, and practical needs. It is suitable for classroom presentation at the introductory graduate or advanced undergraduate level, or for self-study and reference by working professionals.

Graph theory has evolved as a collection of seemingly disparate topics. The intent of the authors is to present this material in a more cohesive framework. In the process, important techniques and analytic tools are transformed into a unified mathematical methodology.

Emphasis throughout is conceptual, with more than 600 graph drawings included to strengthen intuition and more than 1500 exercises ranging from routine drill to more challenging problem solving. Applications and concrete examples are designed to stimulate interest and to demonstrate the relevance of new concepts.

Algorithms are presented in a concise format, and computer science students will find numerous projects inviting them to convert algorithms to computer programs. Software design issues are addressed throughout the book in the form of computational notes, which can be skipped by those not interested in implementation. These design issues include the efficient use of computational resources, software reusability, and the user interface.

This book is a collection of topics drawn from the second edition of *Graph Theory and Its Applications*, written by the first two authors of this book. The topics chosen provide a strong foundation in graph theory.

Summary of Contents

Chapters 1 through 6 concentrate on graph representation, basic properties, modeling, and applications. Graphical constructions here are concrete and primarily spatial. When necessary, we introduce abstractions in a supportive role.

Chapter 7, which is devoted to planarity and Kuratowski's theorem, presents some of the underpinnings of topological graph theory. Chapter 8 is about graph colorings, including vertex- and edge-colorings, map-colorings, and the related topics of cliques, independence numbers, and graph factorization.

The material in Chapters 9 and 10, special digraph models and network flows, overlaps with various areas in operations research and computer science.

Chapter 11 explores the connections between a graph's automorphisms (its symmetries) and its colorings.

The appendix provides a review of a variety of topics which a student would typically encounter in earlier math classes.

To the Instructor

The book is well-suited for a variety of one-semester courses. A *general course* in graph theory might include many of the topics covered in the first eight chapters. A course oriented toward *operations research/optimization* should include most or all of the material in Chapters 4 (spanning trees), 5 (connectivity), 6 (traversability), 8 (colorings), 9 (digraph models), and 10 (flows). A course emphasizing the role of *data structures and algorithms* might include more material from Chapter 3 (trees). A course designed to have strong ties to *geometry, linear algebra, and group theory*, might include parts of Chapters 4 (graphs and vector spaces), 7 (planarity), 8 (graph colorings), and 11 (graph symmetry).

The definitions and results from all of Chapter 1, most of Chapter 2, and from §3.1 are used throughout the text, and we recommend that they be covered in any course that uses the book. The remaining chapters are largely independent of each other, with the following exceptions:

- §3.2 is used in §4.1.
- §4.1 and §4.2 are used in §5.4 and §9.5.
- §4.5 is referred to in §5.3 and §6.1.
- Parts of Chapter 5 are used in §10.3.

Features of This Book

- *Applications.* The Index of Applications lists 70 applications, distributed throughout the text.
- *Algorithms.* The Index of Algorithms points to 51 algorithms, outlined in the text with enough detail to make software implementation feasible.
- *Foreshadowing.* The first three chapters now preview a number of concepts, mostly via the exercises, that are more fully developed in later chapters. This makes it easier to encourage students to take earlier excursions in research areas that may be of particular interest to the instructor.
- *Solutions* and *Hints.* Each exercise marked with a superscript$^{\mathrm{S}}$ has a solution or hint appearing in the back of the book.
- *Supplementary Exercises.* In addition to the section exercises, each chapter concludes with a section of supplementary exercises, which are intended to develop the problem-solving skills of students and test whether they can go beyond what has been explicitly taught. Most of these exercises were designed as examination questions for students at Columbia University.

Websites

Suggestions and comments from readers are welcomed and may be sent to the authors' website at *www.graphtheory.com*. Thanks to our founding webmaster Aaron Gross for his years of maintaining this website. The general website for CRC Press is *www.crcpress.com*. As with our previous graph theory books, we will post the corrections to all known errors on our website.

Jonathan L. Gross, Jay Yellen, and Mark Anderson

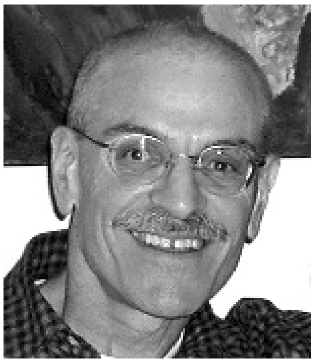
ABOUT THE AUTHORS

Jonathan L. Gross is Professor of Computer Science at Columbia University. His research in topology, graph theory, and cultural sociometry has earned him an Alfred P. Sloan Fellowship, an IBM Postdoctoral Fellowship, and various research grants from the Office of Naval Research, the National Science Foundation, and the Russell Sage Foundation.

Professor Gross has created and delivered numerous software development short courses for Bell Laboratories and for IBM. These include mathematical methods for performance evaluation at the advanced level and for developing reusable software at a basic level. He has received several awards for outstanding teaching at Columbia University, including the career Great Teacher Award from the Society of Columbia Graduates. His peak semester enrollment in his graph theory course at Columbia was 101 students.

His previous books include *Topological Graph Theory*, coauthored with Thomas W. Tucker; *Measuring Culture*, coauthored with Steve Rayner, constructs network-theoretic tools for measuring sociological phenomena; and *Combinatorial Methods with Computer Applications.*

Prior to Columbia University, Professor Gross was in the Mathematics Department at Princeton University. His undergraduate work was at M.I.T., and he wrote his Ph.D. thesis on 3-dimensional topology at Dartmouth College.



Jay Yellen is Professor of Mathematics at Rollins College. He received his B.S. and M.S. in Mathematics at Polytechnic University of New York and did his doctoral work in finite group theory at Colorado State University. Dr. Yellen has had regular faculty appointments at Allegheny College, State University of New York at Fredonia, and Florida Institute of Technology, where he was Chair of Operations Research from 1995 to 1999. He has had visiting appointments at Emory University, Georgia Institute of Technology, and Columbia University, University of Nottingham, UK, and KU Leuven, Belgium.

In addition to the *Handbook* of *Graph Theory*, which he co-edited with Professor Gross, Professor Yellen has written manuscripts used at IBM for two courses in discrete mathematics within the Principles of Computer Science Series and has contributed two sections to the *Handbook* of *Discrete and Combinatorial Mathematics.* He also has designed and conducted several summer workshops on creative problem solving for secondary-school mathematics teachers, which were funded by the National Science Foundation and New York State. He has been a recipient of a Student's Choice Professor Award at Rollins College.

Dr. Yellen has published research articles in character theory of finite groups, graph theory, power-system scheduling, and timetabling. His current research interests include graph theory, discrete optimization, and the development of course-scheduling software based on graph coloring.

Mark Anderson is Professor of Mathematics and Computer Science at Rollins College. He received his doctorate from the University of Virginia for work in algebraic and differential topology. After arriving at Rollins College, he switched his emphasis to graph theory and also earned a masters degree in Computer Science from the University of Central Florida. He has publications in the areas of algebraic topology (cobordism), topological graph theory (crossing number and Cayley maps), graph coloring (using voltage graphs), as well as graph security (eternal domination). He is currently working on problems in robust graph coloring, which combines ideas from graph coloring with those of graph security. Undergraduate students have done collaborative research with Dr. Anderson in topological graph theory, graph coloring, and software testing.

Methods included in this book appear in a wide range of courses that Professor Anderson has designed. In his Artificial Intelligence course, students explore graph search algorithms; in Data Compression, they discover Huffman codes; in Coding Theory, they work with the geometry of the n-dimensional hypercube. Courses that use algebra to answer geometric questions include Algebraic Topology and Transformational Geometry. The Geometry of Islamic Patterns involves imbeddings of graphs on surfaces as a way of characterizing planar tilings. Courses he has developed for students with majors outside math and computer science include The Mathematics of Games and Gaming and Zero, A Revolutionary Idea.

Chapter 1

INTRODUCTION TO GRAPH MODELS

INTRODUCTION

Configurations of nodes and connections occur in a great diversity of applications. They may represent physical networks, such as electrical circuits, roadways, or organic molecules. And they are also used in representing less tangible interactions as might occur in ecosystems, sociological relationships, databases, or the flow of control in a computer program.

Formally, such configurations are modeled by combinatorial structures called *graphs*, consisting of two sets called *vertices* and *edges* and an incidence relation between them. The vertices and edges may have additional attributes, such as color or weight, or anything else useful to a particular model. Graph models tend to fall into a handful of categories. For instance, the network of one-way streets of a city requires a model in which each edge is assigned a direction; two atoms in an organic molecule may have more than one bond between them; and a computer program is likely to have loop structures. These examples require graphs with directions on their edges, with multiple connections between vertices, or with connections from a vertex to itself.

In the past these different types of graphs were often regarded as separate entities, each with its own set of definitions and properties. We have adopted a unified approach by introducing all of these various graphs at once. This allows us to establish properties that are shared by several classes of graphs without having to repeat arguments. Moreover, this broader perspective has an added bonus: it inspires computer representations that lead to the design of fully reusable software for applications.

We begin by introducing the basic terminology needed to view graphs both as configurations in space and as combinatorial objects that lend themselves to computer representation. Pure and applied examples illustrate how different kinds of graphs arise as models.

1.1 GRAPHS AND DIGRAPHS

We think of a *graph* as a set of points in a plane or in 3-space and a set of line segments (possibly curved), each of which either joins two points or joins a point to itself.

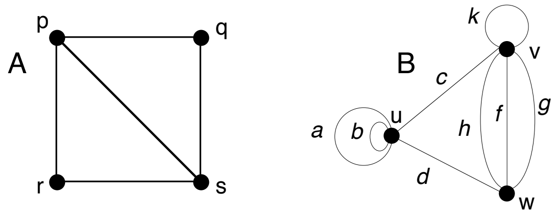


Figure 1.1.1 Line drawings of a graph A and a graph B.

Graphs are highly versatile models for analyzing a wide range of practical problems in which points and connections between them have some physical or conceptual interpretation. Placing such analysis on solid footing requires precise definitions, terminology, and notation.

DEFINITION: A ***graph*** $G = (V, E)$ is a mathematical structure consisting of two finite sets V and E. The elements of V are called ***vertices*** (or ***nodes***), and the elements of E are called ***edges***. Each edge has a set of one or two vertices associated to it, which are called its ***endpoints***.

TERMINOLOGY: An edge is said to ***join*** its endpoints. A vertex joined by an edge to a vertex v is said to be a ***neighbor*** of v.

DEFINITION: The (***open***) ***neighborhood*** of a vertex v in a graph G, denoted $N(v)$, is the set of all the neighbors of v. The ***closed neighborhood*** of v is given by $N[v] = N(v) \cup \{v\}$.

NOTATION: When G is not the only graph under consideration, the notations V_G and E_G (or $V(G)$ and $E(G)$) are used for the vertex- and edge-sets of G, and the notations $N_G(v)$ and $N_G[v]$ are used for the neighborhoods of v.

Example 1.1.1: The vertex- and edge-sets of graph A in Figure 1.1.1 are given by

$$V_A = \{p, q, r, s\} \quad \text{and} \quad E_A = \{pq, pr, ps, rs, qs\}$$

and the vertex- and edge-sets of graph B are given by

$$V_B = \{u, v, w\} \quad \text{and} \quad E_B = \{a, b, c, d, f, g, h, k\}$$

Notice that in graph A, we are able to denote each edge simply by juxtaposing its endpoints, because those endpoints are unique to that edge. On the other hand, in graph B, where some edges have the same set of endpoints, we use explicit names to denote the edges.

Simple Graphs and General Graphs

In certain applications of graph theory and in some theoretical contexts, there are frequent instances in which an edge joins a vertex to itself or multiple edges have the same set of endpoints. In other applications or theoretical contexts, such instances are absent.

DEFINITION: A ***proper edge*** is an edge that joins two distinct vertices.

DEFINITION: A ***self-loop*** is an edge that joins a single endpoint to itself.[†]

DEFINITION: A ***multi-edge*** is a collection of two or more edges having identical endpoints. The ***edge multiplicity*** is the number of edges within the multi-edge.

DEFINITION: A ***simple graph*** has neither self-loops nor multi-edges.

DEFINITION: A ***loopless graph*** (or ***multi-graph***) may have multi-edges but no self-loops.

DEFINITION: A (***general***) ***graph*** may have self-loops and/or multi-edges.

Example 1.1.1 continued: Graph A in Figure 1.1.1 is simple. Graph B is not simple; the edges a, b, and k are self-loops, and the edge-sets $\{f, g, h\}$ and $\{a, b\}$ are multi-edges.

TERMINOLOGY: When we use the term *graph* without a modifier, we mean a *general graph.* An exception to this convention occurs when an entire section concerns simple graphs only, in which case, we make an explicit declaration at the beginning of that section.

TERMINOLOGY NOTE: Some authors use the term *graph* without a modifier to mean simple graph, and they use *pseudograph* to mean general graph.

Null and Trivial Graphs

DEFINITION: A ***null graph*** is a graph whose vertex- and edge-sets are empty.

DEFINITION: A ***trivial graph*** is a graph consisting of one vertex and no edges.

Edge Directions

An edge between two vertices creates a connection in two opposite senses at once. Assigning a direction makes one of these senses *forward* and the other *backward.* In a line drawing, the choice of forward direction is indicated by placing an arrow on an edge.

DEFINITION: A ***directed edge*** (or ***arc***) is an edge, one of whose endpoints is designated as the ***tail***, and whose other endpoint is designated as the ***head***.

TERMINOLOGY: An arc is said to be ***directed from*** its tail to its head.

NOTATION: In a general digraph, the head and tail of an arc e may be denoted $head(e)$ and $tail(e)$, respectively.

DEFINITION: Two arcs between a pair of vertices are said to be ***oppositely directed*** if they do not have the same head and tail.

DEFINITION: A ***multi-arc*** is a set of two or more arcs having the same tail and same head. The ***arc multiplicity*** is the number of arcs within the multi-arc.

DEFINITION: A ***directed graph*** (or ***digraph***) is a graph each of whose edges is directed.

[†]We use the term "self-loop" instead of the more commonly used term "loop" because loop means something else in many applications.

DEFINITION: A digraph is ***simple*** if it has neither self-loops nor multi-arcs.

NOTATION: In a simple digraph, an arc from vertex u to vertex v may be denoted by uv or by the ordered pair $[u, v]$.

Example 1.1.2: The digraph in Figure 1.1.2 is simple. Its arcs are uv, vu, and vw.

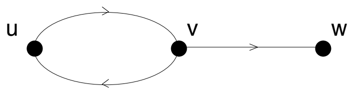


Figure 1.1.2 A simple digraph with a pair of oppositely directed arcs.

DEFINITION: A ***mixed graph*** (or ***partially directed graph***) is a graph that has both undirected and directed edges.

DEFINITION: The ***underlying graph*** of a directed or mixed graph G is the graph that results from removing all the designations of *head* and *tail* from the directed edges of G (i.e., deleting all the edge-directions).

Example 1.1.3: The digraph D in Figure 1.1.3 has the graph G as its underlying graph.

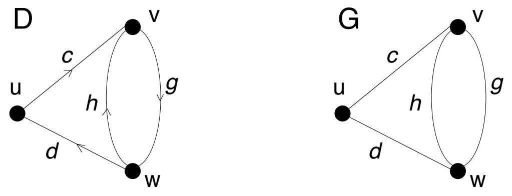


Figure 1.1.3 A digraph and its underlying graph.

Simple and non-simple graphs and digraphs all commonly arise as models; the numerous and varied examples in §1.3 illustrate the robustness of our comprehensive graph model.

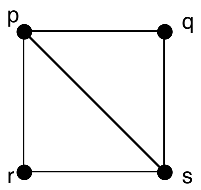
Formal Specification of Graphs and Digraphs

Except for the smallest graphs, line drawings are inadequate for describing a graph; imagine trying to draw a graph to represent a telephone network for a small city. Since many applications involve computations on graphs having hundreds, or even thousands, of vertices, another, more formal kind of specification of a graph is often needed.

The specification must include (implicitly or explicitly) a function *endpts* that specifies, for each edge, the subset of vertices on which that edge is incident (i.e., its endpoint set). In a simple graph, the juxtaposition notation for each edge implicitly specifies its endpoints, making formal specification simpler for simple graphs than for general graphs.

DEFINITION: A ***formal specification of a simple graph*** is given by an ***adjacency table*** with a row for each vertex, containing the list of neighbors of that vertex.

Example 1.1.4: Figure 1.1.4 shows a line drawing and a formal specification for a simple graph.

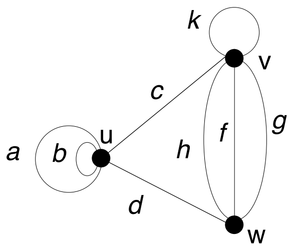


$p:$	q	r	s
$q:$	p	s	
$r:$	p	s	
$s:$	p	q	r

Figure 1.1.4 A simple graph and its formal specification.

DEFINITION: A ***formal specification of a general graph*** $G = (V, E, endpts)$ consists of a list of its vertices, a list of its edges, and a two-row ***incidence table*** (specifying the *endpts* function) whose columns are indexed by the edges. The entries in the column corresponding to edge e are the endpoints of e. The same endpoint appears twice if e is a self-loop. (An isolated vertex will appear only in the vertex list.)

Example 1.1.5: Figure 1.1.5 shows a line drawing and a formal specification for a general graph.



$V = \{u, v, w\}$ and $E = \{a, b, c, d, f, g, h, k\}$

edge	a	b	c	d	f	g	h	k
endpts	u	u	u	w	v	v	w	v
	u	u	v	u	w	w	v	v

Figure 1.1.5 A general graph and its formal specification.

DEFINITION: A ***formal specification of a general digraph*** or ***a mixed graph*** $D = (V, E, endpts, head, tail)$ is obtained from the formal specification of the underlying graph by adding the functions $head : E_G \to V_G$ and $tail : E_G \to V_G$, which designate the *head* vertex and *tail* vertex of each arc.

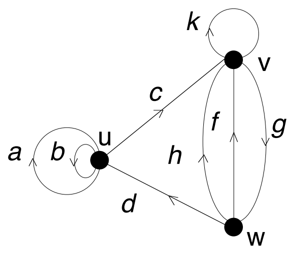
One way to specify these designations in the incidence table is to mark in each column the endpoint that is the head of the corresponding arc.

Remark: A mixed graph is specified by simply restricting the functions *head* and *tail* to a proper subset of E_G. In this case, a column of the incidence table that has no mark means that the corresponding edge is undirected.

Example 1.1.6: Figure 1.1.6 gives the formal specification for the digraph shown, including the corresponding values of the functions *head* and *tail.* A superscript "h" is used to indicate the *head* vertex.

Remark: Our approach treats a digraph as an *augmented* type of graph, where each edge e of a digraph is still associated with a subset $endpts(e)$, but which now also includes a mark on one of the endpoints to specify the head of the directed edge.

This viewpoint is partly motivated by its impact on computer implementations of graph algorithms (see the computational notes), but it has some advantages from a mathematical perspective as well. Regarding digraphs as augmented graphs makes it easier to view certain results that tend to be established separately for graphs and for digraphs as a single result that applies to both.



edge	a	b	c	d	f	g	h	k
endpts	u	u^h	u	w	v^h	v	w	v
	u^h	u	v^h	u^h	w	w^h	v^h	v^h

$$head(a) = tail(a) = head(b) = tail(b) = head(d) = tail(c) = u;$$
$$head(c) = head(h) = head(f) = tail(g) = head(k) = tail(k) = v;$$
$$head(g) = tail(d) = tail(h) = tail(f) = w.$$

Figure 1.1.6 A general digraph and its formal specification.

Also, our formal incidence specification permits us to reverse the direction of an edge e at any time, just by reversing the values of $head(e)$ and $tail(e)$. This amounts to switching the h mark in the relevant column of the incidence table or reversing the arrowhead in the digraph drawing.

COMPUTATIONAL NOTE 1: These formal specifications for a graph and a digraph can easily be implemented with a variety of programmer-defined data structures, whatever is most appropriate to the application. A discussion of the comparative advantages and disadvantages of a few of the most common computer information structures for graphs and digraphs appears at the end of Chapter 2.

COMPUTATIONAL NOTE 2: (*A caution to software designers*) From the perspective of object-oriented software design, the ordered-pair representation of arcs in a digraph treats digraphs as a different class of objects from graphs. This could seriously undermine *software reuse.* Large portions of computer code might have to be rewritten in order to adapt an algorithm that was originally designed for a digraph to work on an undirected graph.

The ordered-pair representation could also prove awkward in implementing algorithms for which the graphs or digraphs are *dynamic* structures (i.e., they change during the algorithm). Whenever the direction on a particular edge must be reversed, the associated ordered pair has to be deleted and replaced by its reverse. Even worse, if a directed edge is to become undirected, then an ordered pair must be replaced with an unordered pair. Similarly, the undirected and directed edges of a mixed graph would require two different types of objects.

COMPUTATIONAL NOTE 3: For some applications (network layouts on a surface, for instance), the direction of flow around a self-loop has practical importance, and distinguishing between the ends of a self-loop becomes necessary. This distinction is made in Chapters 8 and 16 but not elsewhere.

Mathematical Modeling with Graphs

To bring the power of mathematics to bear on real-world problems, one must first *model* the problem mathematically. Graphs are remarkably versatile tools for modeling, and their wide-ranging versatility is a central focus throughout the text.

Example 1.1.7: The mixed graph in Figure 1.1.7 is a model for a roadmap. The vertices represent landmarks, and the directed and undirected edges represent the one-way and two-way streets, respectively.

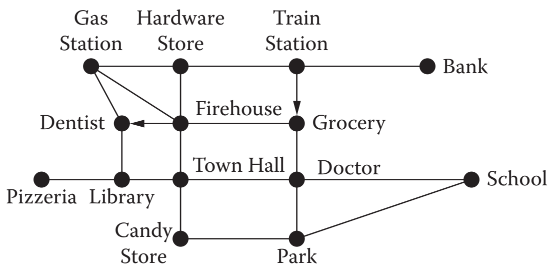


Figure 1.1.7 Road-map of landmarks in a small town.

Example 1.1.8: The digraph in Figure 1.1.8 represents the hierarchy within a company. This illustrates how, beyond physical networks, graphs and digraphs are used to model social relationships.

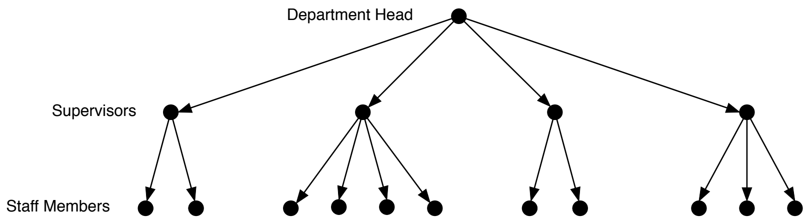


Figure 1.1.8 A corporate hierarchy.

Degree of a Vertex

DEFINITION: ***Adjacent vertices*** are two vertices that are joined by an edge.

DEFINITION: ***Adjacent edges*** are two distinct edges that have an endpoint in common.

DEFINITION: If vertex v is an endpoint of edge e, then v is said to be ***incident*** on e, and e is incident on v.

DEFINITION: The ***degree*** (or ***valence***) of a vertex v in a graph G, denoted $deg(v)$, is the number of proper edges incident on v plus twice the number of self-loops.[†]

TERMINOLOGY: A vertex of degree d is also called a d-***valent vertex***.

NOTATION: The smallest and largest degrees in a graph G are denoted $\delta_{\min}$ and $\delta_{\max}$ (or $\delta_{\min}(G)$ and $\delta_{\max}(G)$ when there is more than one graph under discussion). Some authors use δ instead of $\delta_{\min}$ and Δ instead of $\delta_{\max}$.

DEFINITION: The ***degree sequence*** of a graph is the sequence formed by arranging the vertex degrees in non-increasing order.

[†]Applications of graph theory to physical chemistry motivate the use of the term *valence*.

Example 1.1.9: Figure 1.1.9 shows a graph and its degree sequence.

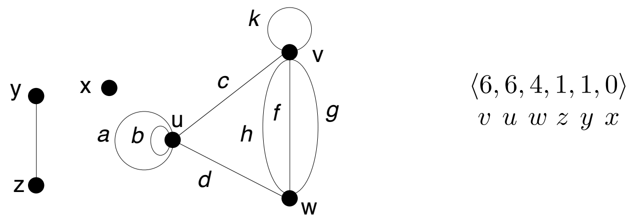


Figure 1.1.9 A graph and its degree sequence.

Although each graph has a unique degree sequence, two structurally different graphs can have identical degree sequences.

Example 1.1.10: Figure 1.1.10 shows two different graphs, G and H, with the same degree sequence.

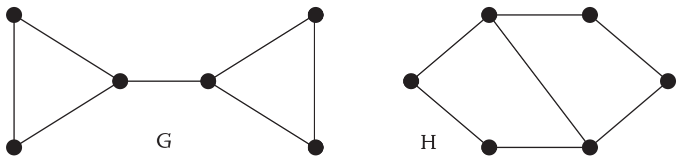


Figure 1.1.10 Two graphs whose degree sequences are both $\langle 3,3,2,2,2,2\rangle$.

The following result shows that the degree sequence of a simple graph must have at least two equal terms. This has an interesting interpretation in a sociological model that appears in Section 1.3 (see Application 1.3.2).

Proposition 1.1.1: *A non-trivial simple graph G must have at least one pair of vertices whose degrees are equal.*

Proof: Suppose that the graph G has n vertices. Then there appear to be n possible degree values, namely $0,\ldots,n-1$. However, there cannot be both a vertex of degree 0 and a vertex of degree $n-1$, since the presence of a vertex of degree 0 implies that each of the remaining $n-1$ vertices is adjacent to at most $n-2$ other vertices. Hence, the n vertices of G can realize at most $n-1$ possible values for their degrees. Thus, the *pigeonhole principle* implies that at least two of the n vertices have equal degree. ♢

The work of Leonhard Euler (1707–1783) is regarded as the beginning of graph theory as a mathematical discipline. The following theorem of Euler establishes a fundamental relationship between the vertices and edges of a graph.

Theorem 1.1.2: [***Euler's Degree-Sum Theorem***] *The sum of the degrees of the vertices of a graph is twice the number of edges.*

Proof: Each edge contributes two to the degree sum. ♢

Corollary 1.1.3: *In a graph, there is an even number of vertices having odd degree.*

Proof: Consider separately, the sum of the degrees that are odd and the sum of those that are even. The combined sum is even by Theorem 1.1.2, and since the sum of the even degrees is even, the sum of the odd degrees must also be even. Hence, there must be an even number of vertices of odd degree. ♢

Remark: By Theorem 1.1.2, the sum of the terms of a degree sequence of a graph is even. Theorem 1.1.4 establishes the following converse: any non-increasing, nonnegative sequence of integers whose sum is even is the degree sequence of some graph. Example 1.1.11 illustrates the construction used in its proof.

Example 1.1.11: To construct a graph whose degree sequence is $\langle 5,4,3,3,2,1,0\rangle$, start with seven isolated vertices $v_1, v_2, \ldots, v_7$. For the even-valued terms of the sequence, draw the appropriate number of self-loops on the corresponding vertices. Thus, v_2 gets two self-loops, v_5 gets one self-loop, and v_7 remains isolated. For the four remaining odd-valued terms, group the corresponding vertices into any two pairs, for instance, v_1, v_3 and v_4, v_6. Then join each pair by a single edge and add to each vertex the appropriate number of self-loops. The resulting graph is shown in Figure 1.1.11.

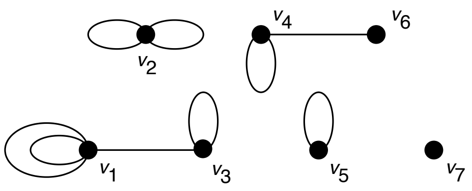


Figure 1.1.11 Constructing a graph with degree sequence $\langle 5,4,3,3,2,1,0\rangle$.

Theorem 1.1.4: *Suppose that $\langle d_1, d_2, \ldots, d_n\rangle$ is a sequence of nonnegative integers whose sum is even. Then there exists a graph with vertices $v_1, v_2, \ldots, v_n$ such that $deg(v_i) = d_i$, for $i = 1, \ldots, n$.*

Proof: Start with n isolated vertices $v_1, v_2, \ldots, v_n$. For each i, if d_i is even, draw $\frac{d_1}{2}$ self-loops on vertex v_i, and if d_i is odd, draw $\frac{d_1 - 1}{2}$ self-loops. By Corollary 1.1.3, there is an even number of odd d_i's. Thus, the construction can be completed by grouping the vertices associated with the odd terms into pairs and then joining each pair by a single edge. ♢

Graphic Sequences

The construction in Theorem 1.1.4 is straightforward but hinges on allowing the graph to be non-simple. A more interesting problem is determining when a sequence is the degree sequence of a *simple* graph.

DEFINITION: A non-increasing sequence $\langle d_1, d_2, \ldots, d_n\rangle$ is said to be ***graphic*** if it is the degree sequence of some simple graph. That simple graph is said to ***realize*** the sequence.

Remark: If $\langle d_1, d_2, \ldots, d_n\rangle$ is the degree sequence of a simple graph, then, clearly, $d_1 \leq n - 1$.

Theorem 1.1.5: *Let $\langle d_1, d_2, \ldots, d_n\rangle$ be a graphic sequence, with $d_1 \geq d_2 \geq \ldots \geq d_n$. Then there is a simple graph with vertex-set $\{v_1, \ldots, v_n\}$ satisfying deg $(v_i) = d_i$ for $i = 1, 2, \ldots, n$, such that v_1 is adjacent to vertices $v_2, \ldots, v_{d_1+1}$.*

Proof: Among all simple graphs with vertex-set $\{v_1, v_2, \ldots, v_n\}$ and $deg(v_i) = d_i$, $i = 1, 2, \ldots, n$, let G be one for which $r = |N_G(v_1) \cap \{v_2, \ldots, v_{d_1+1}\}|$ is maximum. If $r = d_1$, then the conclusion follows. If $r < d_1$, then there exists a vertex $v_s, 2 \leq s \leq d_1 + 1$, such that v_1 is not adjacent to v_s, and there exists a vertex $v_t, t > d_1 + 1$ such that v_1 is adjacent to v_t (since $deg(v_1) = d_1$). Moreover, since $deg(v_s) \geq deg(v_t)$, there exists a vertex v_k such that v_k is adjacent to v_s but not to v_t. Let $\tilde{G}$ be the graph obtained from G by replacing the edges v_1v_t and v_sv_k with the edges v_1v_s and v_tv_k (as shown in Figure 1.1.12). Then the degrees are all preserved and $v_s \in N_{\tilde{G}}(v_1) \cap \{v_3, \ldots, v_{d_1+1}\}$. Thus, $|N_{\tilde{G}}(v_1) \cap \{v_2, \ldots, v_{d_1+1}\}| = r + 1$, which contradicts the choice of G and completes the proof. ♢

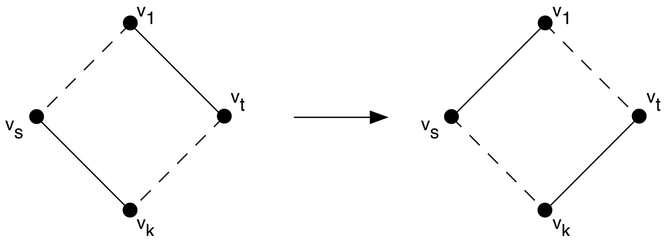


Figure 1.1.12 Switching adjacencies while preserving all degrees.

Corollary 1.1.6: [***Havel (1955) and Hakimi (1961)***] *A sequence $\langle d_1, d_2, \ldots, d_n \rangle$ of nonnegative integers such that $d_1 \leq n-1$ and $d_1 \geq d_2 \geq \ldots \geq d_n$ is graphic if and only if the sequence $\langle d_2 - 1, \ldots, d_{d_1+1} - 1, d_{d_1+2}, \ldots, d_n \rangle$ is graphic.*

♢ *(Exercises)*

Remark: Corollary 1.1.6 yields a recursive algorithm that decides whether a non-increasing sequence is graphic.

```
Algorithm 1.1.1:    GraphicSequence  ($\langle d_1, d_2, \ldots, d_n \rangle$)
Input: a non-increasing sequence $\langle d_1, d_2, \ldots, d_n \rangle$.
Output: TRUE if the sequence is graphic, FALSE if it is not.
    If $d_1 \geq n$ or $d_n < 0$
        Return FALSE
    Else
        If $d_1 = 0$
            Return TRUE
        Else
            Let $\langle a_1, a_2, \ldots, a_{n-1} \rangle$ be a non-increasing permutation
              of $\langle d_2 - 1, \ldots, d_{d_1+1} - 1, d_{d_1+2}, \ldots, d_n \rangle$.
            Return GraphicSequence($\langle a_1, a_2, \ldots, a_{n-1} \rangle$)
```

Remark: An iterative version of the algorithm GraphicSequence based on repeated application of Corollary 1.1.6 can also be written and is left as an exercise (see Exercises). Also, given a graphic sequence, the steps of the iterative version can be reversed to construct a graph realizing the sequence. However many zeroes you get at the end of the forward pass, start with that many isolated vertices. Then backtrack the algorithm, adding a vertex each time. The following example illustrates these ideas.

Example 1.1.12: We start with the sequence $\langle 3, 3, 2, 2, 1, 1 \rangle$. Figure 1.1.13 illustrates an iterative version of the algorithm GraphicSequence and then illustrates the backtracking steps leading to a graph that realizes the original sequence. The hollow vertex shown in each backtracking step is the new vertex added at that step.

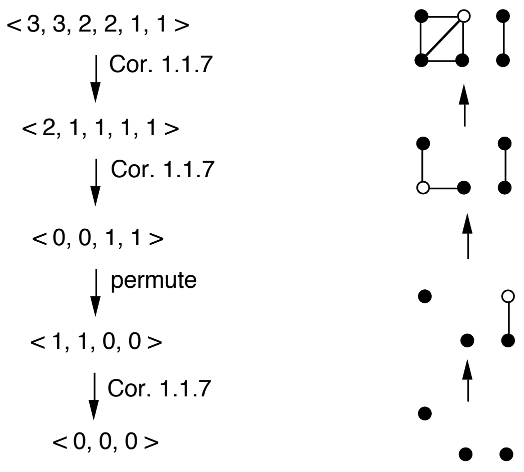
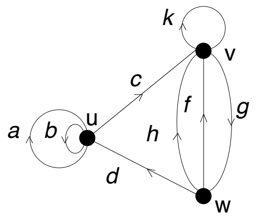


Figure 1.1.13 Testing and realizing the sequence $\langle 3,3,2,2,1,1\rangle$.

Indegree and Outdegree in a Digraph

The definition of vertex degree is slightly refined for digraphs.

DEFINITION: The ***indegree*** of a vertex v in a digraph is the number of arcs directed to v; the ***outdegree*** of vertex v is the number of arcs directed from v. Each self-loop at v counts one toward the indegree of v and one toward the outdegree.



vertex	u	v	w
indegree	3	4	1
outdegree	3	2	3

Figure 1.1.14 The indegrees and outdegrees of the vertices of a digraph.

The next theorem is the digraph version of Euler's Degree-Sum Theorem 1.1.2.

Theorem 1.1.7: *In a digraph, the sum of the indegrees and the sum the outdegrees both equal the number of edges.*

Proof: Each directed edge e contributes one to the indegree at $head(e)$ and one to the outdegree at $tail(e)$. ♢

EXERCISES for Section 1.1

In Exercises 1.1.1 through 1.1.3, construct a line drawing, an incidence table, and the degree sequence of the graph with the given formal specification.

1.1.**1**^{S}
$$V = \{u, w, x, z\}\,; \quad E = \{e, f, g\}$$
$$endpts(e) = \{w\};\ endpts(f) = \{x, w\};\ \ endpts(g) = \{x, z\}$$

1.1.**2**
$$V = \{u, v, x, y, z\}\,; \quad E = \{a, b, c, d\}$$
$$endpts(e) = \{u, v\};\ endpts(b) = \{x, v\};\ endpts(c) = \{u, v\};\ endpts(d) = \{x\}$$

1.1.**3**
$$V = \{u, v, x, y, z\}\,; \quad E = \{e, f, g, h, k\}$$
$$endpts(e) = endpts(f) = \{u, v\}; endpts(g) = \{x, z\}; endpts(h) = endpts(k) = \{y\}$$

In Exercises 1.1.4 through 1.1.6, construct a line drawing for the digraph or mixed graph with vertex-set $V = \{u, v, w, x, y\}$, edge-set $E = \{e, f, g, h\}$, and the given incidence table.

1.1.**4**^{S}

edges	e	f	g	h
endpts	y	x^h	v^h	v^h
	w^h	u	x	u

1.1.**5**

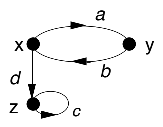
edges	e	f	g	h
endpts	x	v^h	v	v
	w^h	u	u^h	u^h

1.1.**6**

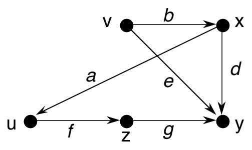
edges	e	f	g	h
endpts	u	x^h	v	v
	w^h	u	y	u

In Exercises 1.1.7 through 1.1.9, give a formal specification for the given digraph.

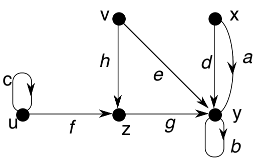
1.1.**7**^{S}



1.1.**8**



1.1.**9**



In Exercises 1.1.10 through 1.1.12, give a formal specification for the underlying graph of the digraph indicated.

1.1.**10**^{S} The digraph of Exercise 1.1.7.

1.1.**11** The digraph of Exercise 1.1.8.

1.1.**12** The digraph of Exercise 1.1.9.

1.1.**13** Draw a graph with the given degree sequence.

a. $\langle 8, 7, 3\rangle$ b. $\langle 9, 8, 8, 6, 5, 3, 1\rangle$

1.1.**14** Draw a simple graph with the given degree sequence.

a. $\langle 6, 4, 4, 3, 3, 2, 1, 1\rangle$ b. $\langle 5, 5, 5, 3, 3, 3, 3, 3\rangle$

For each of the number sequences in Exercises 1.1.15 through 1.1.18, either draw a simple graph that realizes it, or explain, without resorting to Corollary 1.1.6 or Algorithm 1.1.1, why no such graph can exist.

1.1.**15**$^{\mathbf{S}}$ a. $\langle 2, 2, 1, 0, 0\rangle$ b. $\langle 4, 3, 2, 1, 0\rangle$

1.1.**16** a. $\langle 4, 2, 2, 1, 1\rangle$ b. $\langle 2, 2, 2, 2\rangle$

1.1.**17** a. $\langle 4, 3, 2, 2, 1\rangle$ b. $\langle 4, 3, 3, 3, 1\rangle$

1.1.**18** a. $\langle 4, 4, 4, 4, 3, 3, 3, 3\rangle$ b. $\langle 3, 2, 2, 1, 0\rangle$

1.1.**19** Apply Algorithm 1.1.1 to each of the following sequences to determine whether it is graphic. If the sequence is graphic, then draw a simple graph that realizes it.

a. $\langle 7, 6, 6, 5, 4, 3, 2, 1\rangle$ b. $\langle 5, 5, 5, 4, 2, 1, 1, 1\rangle$

c. $\langle 7, 7, 6, 5, 4, 4, 3, 2\rangle$ d. $\langle 5, 5, 4, 4, 2, 2, 1, 1\rangle$

1.1.**20** Use Theorem 1.1.5 to prove Corollary 1.1.6.

1.1.**21** Write an iterative version of Algorithm 1.1.1 that applies Corollary 1.1.6 repeatedly until a sequence of all zeros or a sequence with a negative term results.

1.1.**22**$^{\mathbf{S}}$ Given a group of nine people, is it possible for each person to shake hands with exactly three other people?

1.1.**23** Draw a graph whose degree sequence has no duplicate terms.

1.1.**24**$^{\mathbf{S}}$ What special property of a function must the *endpts* function have for a graph to have no multi-edges?

1.1.**25** Draw a digraph for each of the following indegree and outdegree sequences, such that the indegree and outdegree of each vertex occupy the same position in both sequences.

a. in: $\langle 1, 1, 1\rangle$ out: $\langle 1, 1, 1\rangle$

b. in: $\langle 2, 1\rangle$ out: $\langle 3, 0\rangle$

DEFINITION: A pair of sequences $\langle a_1, a_2, \ldots, a_n\rangle$ and $\langle b_1, b_2, \ldots, b_n\rangle$ is called ***digraphic*** if there exists a simple digraph with vertex-set $\{v_1, v_2, \ldots, v_n\}$ such that $outdegree(v_i) = a_i$ and $indegree(v_i) = b_i$ for $i = 1, 2, \ldots, n$.

1.1.**26** Determine whether the pair of sequences $\langle 3, 1, 1, 0\rangle$ and $\langle 1, 1, 1, 2\rangle$ is digraphic.

1.1.**27** Establish a result like Corollary 1.1.6 for a pair sequences to be digraphic.

1.1.**28** How many different degree sequences can be realized for a graph having three vertices and three edges?

1.1.**29** Given a list of three vertices and a list of seven edges, show that 3^7 different formal specifications for simple graphs are possible.

1.1.**30** Given a list of four vertices and a list of seven edges, show that $\binom{7}{2}5^6 2^{10}$ different formal specifications are possible if there are exactly two self-loops.

1.1.**31** Given a list of three vertices and a list of seven edges, how many different formal specifications are possible if exactly three of the edges are directed?

1.1.**32**$^{\mathbf{S}}$ Does there exist a simple graph with five vertices, such that every vertex is incident with at least one edge, but no two edges are adjacent?

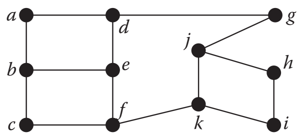
1.1.**33** Prove or disprove: There exists a simple graph with 13 vertices, 31 edges, three 1-valent vertices, and seven 4-valent vertices.

DEFINITION: Let $G = (V, E)$ be a graph and let $W \subseteq V$. Then W is a ***vertex cover*** of G if every edge is incident on at least one vertex in W. (See §10.4.)

1.1.**34** Find upper and lower bounds for the size of a minimum (smallest) vertex cover of an n-vertex connected simple graph G. Then draw three 8-vertex graphs, one that achieves the lower bound, one that achieves the upper bound, and one that achieves neither.

1.1.**35** Find a minimum vertex cover for the underlying graph of the mixed graph shown in Figure 1.1.7.

1.1.**36**$^{\mathbf{S}}$ The graph shown below represents a network of tunnels, where the edges are sections of tunnel and the vertices are junction points. Suppose that a guard placed at a junction can monitor every section of tunnel leaving it. Determine the minimum number of guards and a placement for them so that every section of tunnel is monitored.



DEFINITION: An ***independent set of vertices*** in a graph G is a set of mutually non-adjacent vertices. (See §2.3.)

1.1.**37** Find upper and lower bounds for the size of a maximum (largest) independent set of vertices in an n-vertex connected graph. Then draw three 8-vertex graphs, one that achieves the lower bound, one that achieves the upper bound, and one that achieves neither.

1.1.**38** Find a maximum independent set of vertices in the underlying graph of the mixed graph shown in Figure 1.1.7.

1.1.**39**$^{\mathbf{S}}$ Find a maximum independent set of vertices in the graph of Exercise 1.1.36.

DEFINITION: A ***matching*** in a graph G is a set of mutually non-adjacent edges in G. (See §8.3 and §10.4.)

1.1.**40** Find upper and lower bounds for the size of a maximum (largest) matching in an n-vertex connected graph. Then draw three 8-vertex graphs, one that achieves the lower bound, one that achieves the upper bound, and one that achieves neither.

1.1.**41** Find a maximum matching in the underlying graph of the mixed graph shown in Figure 1.1.7.

1.1.**42**$^{\mathbf{S}}$ Find a maximum matching in the graph of Exercise 1.1.36.

DEFINITION: Let $G = (V, E)$ be a graph and let $W \subseteq V$. Then W ***dominates*** G (or is a ***dominating set*** of G) if every vertex in V is in W or is adjacent to at least one vertex in W. That is, $\forall v \in V, \exists w \in W, v \in N[w]$.

1.1.**43** Find upper and lower bounds for the size of a minimum (smallest) dominating set of an n-vertex graph. Then draw three 8-vertex graphs, one that achieves the lower bound, one that achieves the upper bound, and one that achieves neither.

1.1.**44** Find a minimum dominating set for the underlying graph of the mixed graph shown in Figure 1.1.7.

1.1.**45**$^{\mathbf{S}}$ Find a minimum dominating set for the graph of Exercise 1.1.36.

1.2 COMMON FAMILIES OF GRAPHS

There is a multitude of standard examples that recur throughout graph theory.

Complete Graphs

DEFINITION: A ***complete graph*** is a simple graph such that every pair of vertices is joined by an edge. Any complete graph on n vertices is denoted K_n.

Example 1.2.1: Complete graphs on one, two, three, four, and five vertices are shown in Figure 1.2.1.

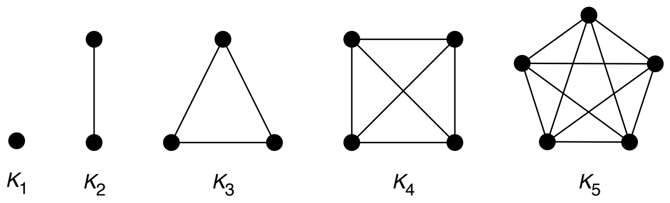


Figure 1.2.1 The first five complete graphs.

Bipartite Graphs

DEFINITION: A ***bipartite*** graph G is a graph whose vertex-set V can be partitioned into two subsets U and W, such that each edge of G has one endpoint in U and one endpoint in W. The pair U, W is called a ***(vertex) bipartition*** of G, and U and W are called the ***bipartition subsets***.

Example 1.2.2: Two bipartite graphs are shown in Figure 1.2.2. The bipartition subsets are indicated by the solid and hollow vertices.

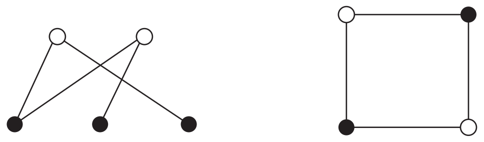


Figure 1.2.2 Two bipartite graphs.

Proposition 1.2.1: *A bipartite graph cannot have any self-loops.*

Proof: This is an immediate consequence of the definition. ♢

Example 1.2.3: The smallest possible simple graph that is not bipartite is the complete graph K_3, shown in Figure 1.2.3.

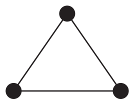


Figure 1.2.3 The smallest non-bipartite simple graph.

DEFINITION: A ***complete bipartite graph*** is a simple bipartite graph such that every vertex in one of the bipartition subsets is joined to every vertex in the other bipartition subset. Any complete bipartite graph that has m vertices in one of its bipartition subsets and n vertices in the other is denoted $K_{m,n}$.[†]

Example 1.2.4: The complete bipartite graph $K_{3,4}$ is shown in Figure 1.2.4.

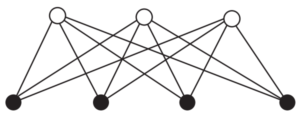


Figure 1.2.4 The complete bipartite graph $K_{3,4}$.

Regular Graphs

DEFINITION: A ***regular*** graph is a graph whose vertices all have equal degree. A ***k-regular*** graph is a regular graph whose common degree is k.

DEFINITION: The five regular polyhedra illustrated in Figure 1.2.5 are known as the ***platonic solids***. Their vertex and edge configurations form regular graphs called the ***platonic graphs***.

DEFINITION: The ***Petersen graph*** is the 3-regular graph represented by the line drawing in Figure 1.2.6. Because it possesses a number of interesting graph-theoretic properties, the Petersen graph is frequently used both to illustrate established theorems and to test conjectures.

[†]The sense in which $K_{m,n}$ is a unique object is described in §2.1.

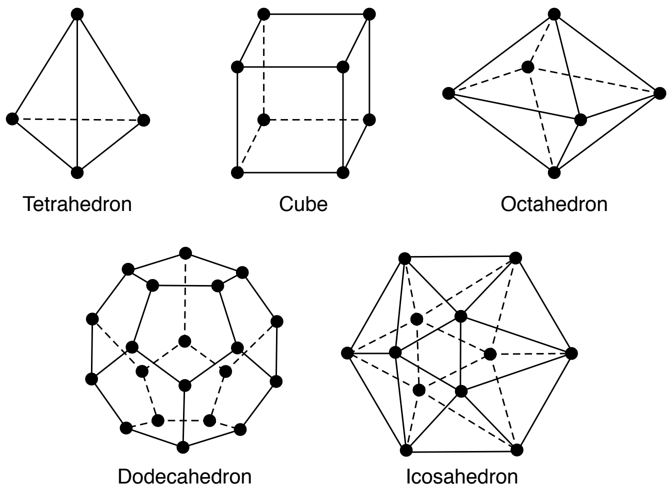


Figure 1.2.5 The five platonic graphs.

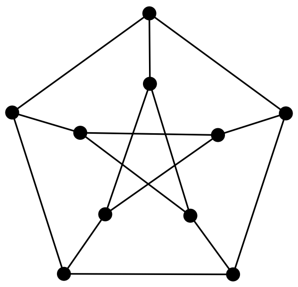


Figure 1.2.6 The Petersen graph.

Example 1.2.5: The oxygen molecule O_2, made up of two oxygen atoms linked by a double bond, can be represented by the 2-regular graph shown in Figure 1.2.7.

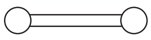


Figure 1.2.7 A 2-regular graph representing the oxygen molecule O_2.

Bouquets and Dipoles

One-vertex and two-vertex (non-simple) graphs often serve as building blocks for various interconnection networks, including certain parallel architectures.

DEFINITION: A graph consisting of a single vertex with n self-loops is called a ***bouquet*** and is denoted B_n.

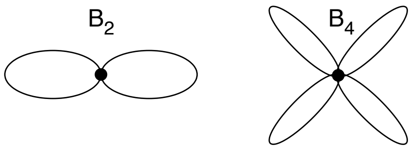


Figure 1.2.8 Bouquets B_2 and B_4.

DEFINITION: A graph consisting of two vertices and n edges joining them is called a ***dipole*** and is denoted D_n.

Example 1.2.6: The graph representation of the oxygen molecule in Figure 1.2.7 is an instance of the dipole D_2. Figure 1.2.9 shows the dipoles D_3 and D_4.

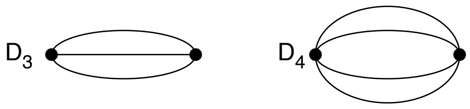


Figure 1.2.9 Dipoles D_3 and D_4.

Path Graphs and Cycle Graphs

DEFINITION: A ***path graph*** P is a simple graph with $|V_P| = |E_P| + 1$ that can be drawn so that all of its vertices and edges lie on a single straight line. A path graph with n vertices and $n - 1$ edges is denoted P_n.

Example 1.2.7: Path graphs P_2 and P_4 are shown in Figure 1.2.10.

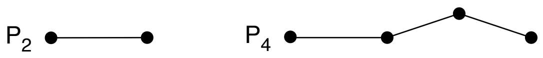


Figure 1.2.10 Path graphs P_2 and P_4.

DEFINITION: A ***cycle graph*** is a single vertex with a self-loop or a simple graph C with $|V_C| = |E_C|$ that can be drawn so that all of its vertices and edges lie on a single circle. An n-vertex cycle graph is denoted C_n.

Example 1.2.8: The cycle graphs C_1, C_2, and C_4 are shown in Figure 1.2.11.

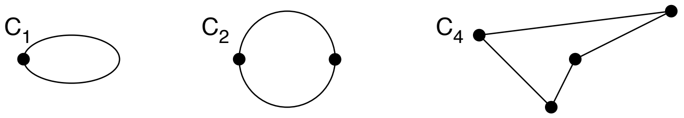


Figure 1.2.11 Cycle graphs C_1, C_2, and C_4.

Remark: The terms *path* and *cycle* also refer to special sequences of vertices and edges within a graph and are defined in §1.4 and §1.5.

Hypercubes and Circular Ladders

DEFINITION: The ***hypercube graph*** Q_n is the n-regular graph whose vertex-set is the set of bitstrings of length n, such that there is an edge between two vertices if and only if they differ in exactly one bit.

Example 1.2.9: The 8-vertex cube graph that appeared in Figure 1.2.5 is a hypercube graph Q_3 (see Exercises).

DEFINITION: The ***circular ladder graph*** CLn is visualized as two concentric n-cycles in which each of the n pairs of corresponding vertices is joined by an edge.

Example 1.2.10: The circular ladder graph CL_4 is shown in Figure 1.2.12.

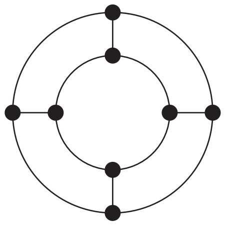


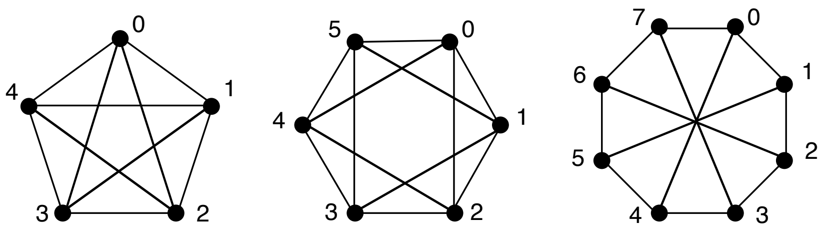
Figure 1.2.12 Circular ladder graph CL_4.

Circulant Graphs

DEFINITION: To the group of integers $\mathbb{Z}_n = \{0, 1, \ldots, n-1\}$ under addition modulo n and a set $S \subseteq \{1, \ldots, n-1\}$, we associate the ***circulant graph*** $circ(n : S)$ whose vertex set is $\mathbb{Z}_n$, such that two vertices i and j are adjacent if and only if there is a number $s \in S$ such that $i + s = j \bmod n$ or $j + s = i \bmod n$. In this regard, the elements of the set S are called ***connections***.

NOTATION: It is often convenient to specify the connection set $S = \{s_1, \ldots, s_r\}$ without the braces, and to write $circ\,(n : s_1, \ldots, s_r)$.

Example 1.2.11: Figure 1.2.13 shows three circulant graphs.



$circ(5 : 1, 2)$ $circ(6 : 1, 2)$ $circ(8 : 1, 4)$

Figure 1.2.13 Three circulant graphs.

Remark: Notice that circulant graphs are simple graphs. Circulant graphs are a special case of ***Cayley graphs***, which are themselves derived from a special case of *voltage graphs.*

Intersection and Interval Graphs

DEFINITION: A simple graph G with vertex-set $V_G = \{v_1, v_2 \ldots, v_n\}$ is an ***intersection graph*** if there exists a family of sets $F = \{S_1, S_2, \ldots, S_n\}$ such that vertex v_i is adjacent to v_j if and only if $i \neq j$ and $S_i \cap S_j \neq \emptyset$.

DEFINITION: A simple graph is an ***interval graph*** if it is an intersection graph corresponding to a family of intervals on the real line.

Example 1.2.12: The graph in Figure 1.2.14 is an interval graph for the following family of intervals:

$$a \leftrightarrow (1,3) \quad b \leftrightarrow (2,6) \quad c \leftrightarrow (5,8) \quad d \leftrightarrow (4,7)$$

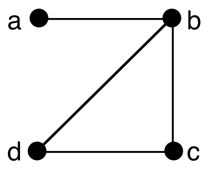


Figure 1.2.14 An interval graph.

Application 1.2.1: *Archeology* Suppose that a collection of artifacts was found at the site of a town known to have existed from 1000 BC to AD 1000. In the graph shown below, the vertices correspond to artifact types, and two vertices are adjacent if some grave contains artifacts of both types. It is reasonable to assume that artifacts found in the same grave have overlapping time intervals during which they were in use. If the graph is an interval graph, then there is an assignment of subintervals of the interval $(-1000, 1000)$ (by suitable scaling, if necessary) that is consistent with the archeological find.

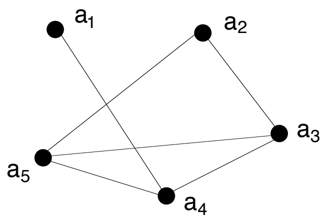


Figure 1.2.15 A graph model of an archeological find.

Line Graphs

Line graphs are a special case of intersection graphs.

DEFINITION: The ***line graph*** $L(G)$ of a graph G has a vertex for each edge of G, and two vertices in $L(G)$ are adjacent if and only if the corresponding edges in G have a vertex in common.

Thus, the line graph $L(G)$ is the intersection graph corresponding to the endpoint sets of the edges of G.

Example 1.2.13: Figure 1.2.16 shows a graph G and its line graph $L(G)$.

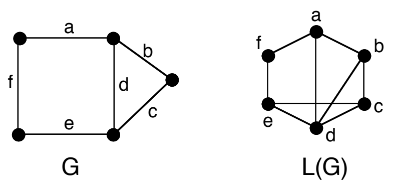


Figure 1.2.16 A graph and its line graph.

EXERCISES for Section 1.2

1.2.**1**^{S} Find the number of edges for each of the following graphs.

a. K_n b. $K_{m,n}$

1.2.**2** What is the maximum possible number of edges in a simple bipartite graph on m vertices?

1.2.**3** Draw the smallest possible non-bipartite graph.

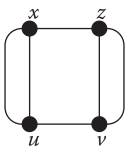
1.2.**4**^{S} Determine the values of n for which the given graph is bipartite.

a. K_n b. C_n c. P_n

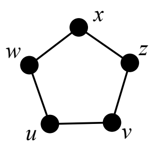
1.2.**5** Draw a 3-regular bipartite graph that is not $K_{3,3}$.

In Exercises 1.2.6 to 1.2.8, determine whether the given graph is bipartite. In each case, give a vertex bipartition or explain why the graph is not bipartite.

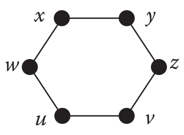
1.2.**6**^{S}



1.2.**7**



1.2.**8**



1.2.**9** Label the vertices of the cube graph in Figure 1.2.5 with 3-bit binary strings so that the labels on adjacent vertices differ in exactly one bit.

1.2.**10** Prove that the graph Q_n is bipartite.

1.2.**11** For each of the platonic graphs, is it possible to trace a *tour* of all the vertices by starting at one vertex, traveling only along edges, never revisiting a vertex, and never lifting the pen off the paper? Is it possible to make the tour return to the starting vertex?

1.2.**12**^{S} Prove or disprove: There does not exist a 5-regular graph on 11 vertices.

DEFINITION: A ***tournament*** is a digraph whose underlying graph is a complete graph.

1.2.**13** a. Draw all the 3-vertex tournaments whose vertices are u , v, x.
b. Determine the number of 4-vertex tournaments whose vertices are u, v, x, y.

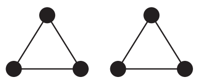
1.2.**14** Prove that every tournament has at most one vertex of indegree 0 and at most one vertex of outdegree 0.

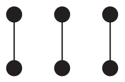
1.2.**15**^{S} Suppose that n vertices $v_1, v_2, \ldots, v_n$ are drawn in the plane. How many different n-vertex tournaments can be drawn on those vertices?

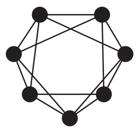
1.2.**16** Chartrand and Lesniak [ChLe04] define a pair of sequences of nonnegative integers $\langle a_1, a_2, \ldots a_r\rangle$ and $\langle b_1, b_2, \ldots b_t\rangle$ to be ***bigraphical*** if there exists a bipartite graph G with bipartition subsets $U = \{u_1, u_2, \ldots u_r\}$ and $W = \{w_1, w_2, \ldots w_t\}$ such that $deg\,(u_i) = a_i, i = 1, 2, \ldots, r$, and $deg(w_i) = b_i, i = 1, 2, \ldots, t$. Prove that a pair of non-increasing sequences of nonnegative integers $\langle a_1, a_2, \ldots, a_r\rangle$ and $\langle b_1, b_2, \ldots, b_t\rangle$ with $r \geq 2$, $0 < a_1 \leq t$, and $0 < a_1 \leq r$ is bigraphical if and only if the pair $\langle a_2, \ldots, a_r\rangle$ and $\langle b_1 - 1, b_2 - 1, \ldots, b_{a_1} - 1, b_{a_1+1}, b_{a_1+2}, \ldots, b_t\rangle$ is bigraphical.

1.2.**17** Find all the 4-vertex circulant graphs.

1.2.**18** Show that each of the following graphs is a circulant graph.

a.


b.


c.


1.2.**19** State a necessary and sufficient condition on the positive integers n and k for $circ(\,n : k)$ to be the cycle graph C_n.

1.2.**20** Find necessary and sufficient conditions on the positive integers n and k for $circ(n : k)$ to be the graph consisting of $n/2$ mutually non-adjacent edges.

1.2.**21** Determine the size of a smallest dominating set (defined in §1.1 exercises) in the graph indicated.

a. K_n b. $K_{m,n}$ c. C_n d. P_n e. CL_n

1.2.**22** Determine the size of a smallest vertex cover (defined in §1.1 exercises) in the graph indicated.

a. K_n b. $K_{m,n}$ c. C_n d. P_n e. CL_n

1.2.**23** Determine the size of a largest independent set of vertices (defined in §1.1 exercises) in the graph indicated.

a. K_n b. $K_{m,n}$ c. C_n d. P_n e. CL_n

1.2.**24** Determine the size of a maximum matching (defined in §1.1 exercises) in the graph indicated.

a. K_n b. $K_{m,n}$ c. C_n d. P_n e. CL_n

1.2.**25**$^{\mathbf{S}}$ Show that the complete graph K_n is an interval graph for all $n \geq 1$.

1.2.**26** Draw the interval graph for the intervals $(0,2),(3,8),(1,4),(3,4),(2,5),(7,9)$.

1.2.**27**$^{\mathbf{S}}$ Show that the graph modeling the archeological find in Application 1.2.1 is an interval graph by using a family of subintervals of the interval $(-1000,1000)$.

1.2.**28** Prove that the cycle graph C_n is not an interval graph for any $n \geq 4$.

1.2.**29** Draw the intersection graph for the family of all subsets of the set $\{1, 2, 3\}$.

1.2.**30** Prove that every simple graph is an intersection graph by describing how to construct a suitable family of sets.

1.3 GRAPH MODELING APPLICATIONS

Different kinds of graphs arise in modeling real-world problems. Sometimes, simple graphs are adequate; other times, non-simple graphs are needed. The analysis of some of the applications considered in this section is deferred to later chapters, where the necessary theoretical methods are fully developed.

Models That Use Simple Graphs

Application 1.3.1: *Personnel-Assignment Problem* Suppose that a company requires a number of different types of jobs, and suppose each employee is suited for some of these jobs, but not others. Assuming that each person can perform at most one job at a time, how should the jobs be assigned so that the maximum number of jobs can be performed simultaneously? In the bipartite graph of Figure 1.3.1, the hollow vertices represent the employees, the solid vertices the jobs, and each edge represents a suitable job assignment. The bold edges represent a largest possible set of suitable assignments. Chapter 10 provides a fast algorithm to solve large instances of this classical problem in *operations research.*

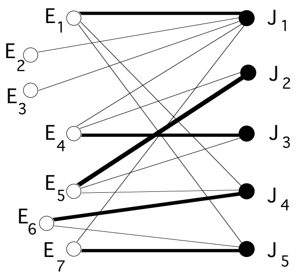


Figure 1.3.1 An optimal assignment of employees to jobs.

Application 1.3.2: *Sociological-Acquaintance Networks* In an ***acquaintance network***, the vertices represent persons, such as the students in a college class. An edge joining two vertices indicates that the corresponding pair of persons knew each other when the course began. The simple graph in Figure 1.3.2 shows a typical acquaintance network. Including the Socratic concept of *self-knowledge* would require the model to allow self-loops. For instance, a self-loop drawn at the vertex representing Slim might mean that she was "in touch" with herself.

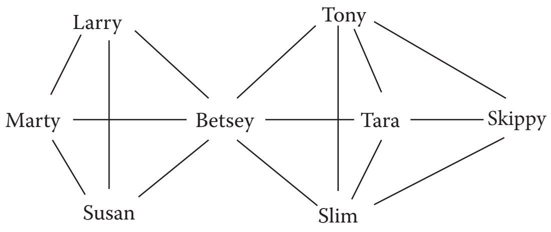


Figure 1.3.2 An acquaintance network.

By Proposition 1.1.1, every group of two or more persons must contain at least two who know the same number of persons in the group. The acquaintance network of Figure 1.3.2 has degree sequence $\langle 6, 4, 4, 4, 3, 3, 3, 3\rangle$.

Application 1.3.3: *Geographic Adjacency* In the geographical model in Figure 1.3.3, each vertex represents a northeastern state, and each adjacency represents sharing a border.

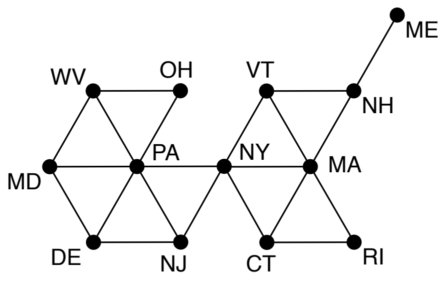


Figure 1.3.3 Geographic adjacency of the northeastern states.

Application 1.3.4: *Geometric Polyhedra* The vertex and edge configuration of any polyhedron in 3-space forms a simple graph, which topologists call its ***1-skeleton***. The 1-skeletons of the platonic solids, appearing in the previous section, are regular graphs. Figure 1.3.4 shows a polyhedron whose 1-skeleton is not regular.

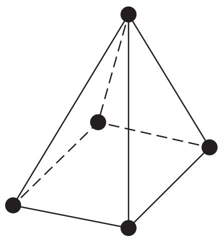


Figure 1.3.4 A non-regular 1-skeleton of a polyhedron.

Application 1.3.5: *Interconnection Networks for Parallel Architectures*
Numerous processors can be linked together on a single chip for a multi-processor computer that can execute parallel algorithms. In a graph model for such an interconnection network, each vertex represents an individual processor, and each edge represents a direct link between two processors. Figure 1.3.5 illustrates the underlying graph structure of one such interconnection network, called a *wrapped butterfly.*

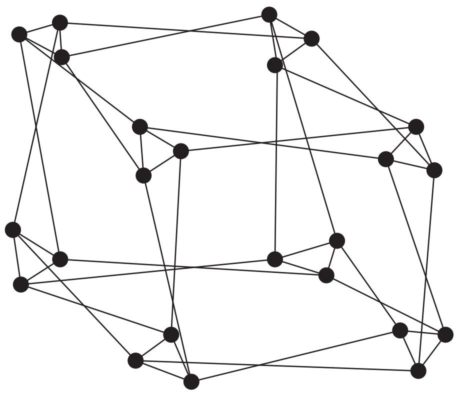


Figure 1.3.5 A wrapped-butterfly-interconnection-network model.

Application 1.3.6: *Assigning Broadcasting Frequencies* When the radio transmitters in a geographical region are assigned broadcasting frequencies, some pairs of transmitters require different frequencies to avoid interference. A graph model can be used for the problem of minimizing the number of different frequencies assigned.

Suppose that the seven radio transmitters, $A, B, \ldots, G$, must be assigned frequencies. For simplicity, assume that if two transmitters are less than 100 miles apart, they must broadcast at different frequencies. Consider a graph whose vertices represent the transmitters, and whose edges indicate those pairs that are less than 100 miles apart. Figure 1.3.6 shows a table of distances for the seven transmitters and the corresponding graph on seven vertices.

	B	C	D	E	F	G
A	55	110	108	60	150	88
B		87	142	133	98	139
C			77	91	85	93
D				75	114	82
E					107	41
F						123

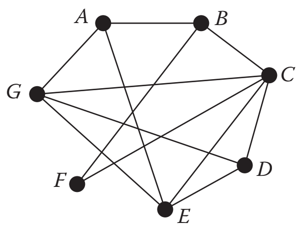


Figure 1.3.6 A simple graph for a radio-frequency-assignment problem.

The problem of assigning radio frequencies to avoid interference is equivalent to the problem of *coloring* the vertices of the graph so that adjacent vertices get different colors. The minimum number of frequencies will equal the minimum number of different colors required for such a coloring. This and several other graph-coloring problems and applications are discussed in Chapter 8.

Models Requiring Non-Simple Graphs

Application 1.3.7: *Roadways between States* If in the Geographic-Adjacency Application 1.3.3, each edge joining two vertices represented a road that crosses a border between the corresponding two states, then the graph would be non-simple, since pairs of bordering states have more than one road joining them.

Application 1.3.8: *Chemical Molecules* The benzene molecule shown in Figure 1.3.7 has double bonds for some pairs of its atoms, so it is modeled by a non-simple graph. Since each carbon atom has valence 4, corresponding to four electrons in its outer shell, it is represented by a vertex of degree 4; and since each hydrogen atom has one electron in its only shell, it is represented by a vertex of degree 1.

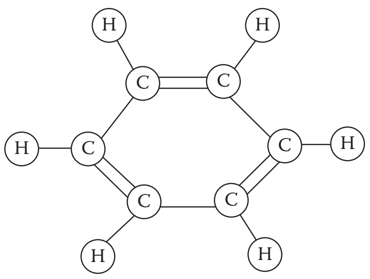


Figure 1.3.7 The benzene molecule.

Models That Use Simple Digraphs

For each of the next series of applications, a link in one direction does not imply a link in the opposite direction.

Application 1.3.9: *Ecosystems* The feeding relationships among the plant and animal species of an ecosystem are called a *food web* and may be modeled by a simple digraph. The food web for a Canadian willow forest is illustrated in Figure 1.3.8.[†] Each species in the system is represented by a vertex, and a directed edge from vertex u to vertex v means that the species corresponding to u feeds on the species corresponding to v.

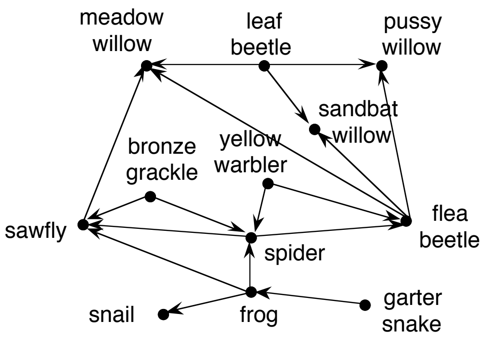


Figure 1.3.8 The food web in a Canadian willow forest.

Application 1.3.10: *Activity-Scheduling Networks* In large projects, often there are some tasks that cannot start until certain others are completed. Figure 1.3.9 shows a digraph model of the *precedence relationships* among some tasks for building a house. Vertices correspond to tasks. An arc from vertex u to vertex v means that task v cannot start until task u is completed. To simplify the drawing, arcs that are implied by transitivity are not drawn. This digraph is the *cover diagram* of a partial ordering of the tasks. A different model, in which the tasks are represented by the arcs of a digraph, is studied in Chapter 9.

	Activities
1	Foundation
2	Walls and Ceilings
3	Roof
4	Electrical wiring
5	Windows
6	Siding
7	Paint interior
8	Paint exterior

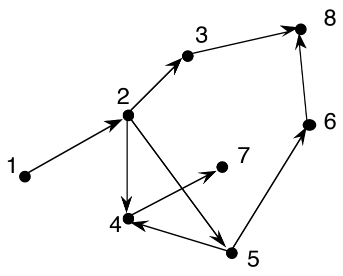


Figure 1.3.9 An activity digraph for building a house.

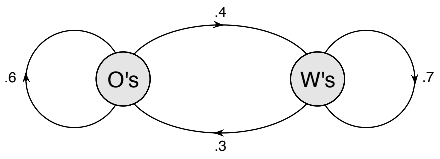
[†]This illustration was adapted from [WiWa90], p. 69.

Application 1.3.11: *Flow Diagrams for Computer Programs* A computer program is often designed as a collection of *program blocks*, with appropriate flow control. A digraph is a natural model for this decomposition. Each program block is associated with a vertex, and if control at the last instruction of block u can transfer to the first instruction of block v, then an arc is drawn from vertex u to vertex v. Computer flow diagrams do not usually have multi-arcs. Unless a single block is permitted both to change values of some variables and to retest those values, a flow diagram with a self-loop would mean that the program has an infinite loop.

Models Requiring Non-Simple Digraphs

Application 1.3.12: *Markov Diagrams* Suppose that the inhabitants of some remote area purchase only two brands of breakfast cereal, O's and W's. The consumption patterns of the two brands are encapsulated by the *transition matrix* shown in Figure 1.3.10. For instance, if someone just bought O's, there is a 0.4 chance that the person's next purchase will be W's and a 0.6 chance it will be O's.

In a *Markov process*, the *transition probability* of going from one state to another depends only on the current state. Here, states "O" and "W" correspond to whether the most recent purchase was O's or W's, respectively. The digraph model for this Markov process, called a *Markov diagram*, is shown in Figure 1.3.10. Each arc is labeled with the transition probability of moving from the state at the tail vertex to the state at the head. Thus, the probabilities on the outgoing edges from each vertex must sum to 1. This Markov diagram is an example of a *weighted graph* (see §1.6). Other examples of Markov diagrams appear in Chapter 9.



	O's	W's
O's	.6	.4
W's	.3	.7

Figure 1.3.10 A Markov diagram and its transition matrix.

Application 1.3.13: *Lexical Scanners* The source code of a computer program may be regarded as a string of symbols. A *lexical scanner* must scan these symbols, one at a time, and recognize which symbols "go together" to form a syntactic *token* or *lexeme*. We now consider a single-purpose scanner whose task is to recognize whether an input string of characters is a valid *identifier* in the C programming language. Such a scanner is a special case of a *finite-state recognizer* and can be modeled by a labeled digraph, as in Figure 1.3.11. One vertex represents the *start* state, in effect before any symbols have been scanned. Another represents the *accept* state, in which the substring of symbols scanned so far forms a valid C identifier. The third vertex is the *reject* state, indicating that the substring has been discarded because it is not a valid C identifier. The arc labels tell what kinds of symbols cause a transition from the tail state to the head state. If the final state after the input string is completely scanned is the accept state, then the string is a valid C identifier.

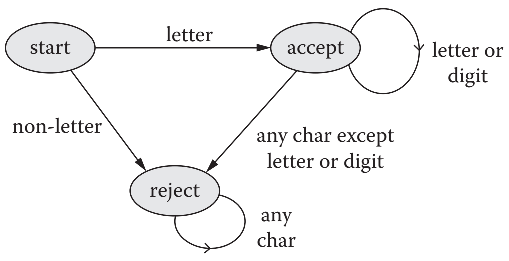
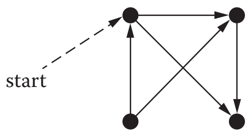


Figure 1.3.11 Finite-state recognizer for identifiers.

EXERCISES for Section 1.3

1.3.1^S Solve the radio-frequency-assignment problem in Application 1.3.6 by determining the minimum number of colors needed to color the vertices of the associated graph. Argue why the graph cannot be colored with fewer colors.

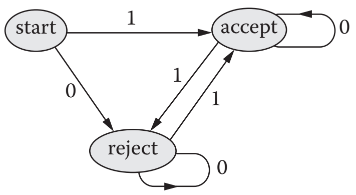
1.3.2 What is wrong with a computer program having the following abstract flow pattern?



1.3.3^S Referring to the Markov diagram in Application 1.3.12, suppose that someone just purchased a box of O's. What is the probability that his next three purchases are W's, O's, and then W's?

1.3.4 Modify the finite-state recognizer in Application 1.3.13 so that it accepts only those strings that begin with two letters and whose remaining characters, if any, are digits. (Hint: Consider adding one or more new states.)

1.3.5^S Which strings are accepted by the following finite-state recognizer?



In Exercises 1.3.6 through 1.3.13, design appropriate graph or digraph models and problems for the given situation. That is, specify the vertices and edges and any additional features.

1.3.6^S Two-person rescue teams are being formed from a pool of n volunteers from several countries. The only requirement is that both members of a team must have a language in common.

1.3.7 Suppose that meetings must be scheduled for several committees. Two committees must be assigned different meeting times if there is at least one person on both committees.

1.3.8 Represent the "relative strength" of a finite collection of logical propositional forms. For example, the proposition $p \wedge q$ is at least as strong as $p \vee q$ since the first implies the second (i.e., $(p \wedge q) \Rightarrow (p \vee q)$ is a *tautology*).

1.3.**9** Suppose there are three power generators to be built in three of the seven most populated cities of a certain country. The distances between each pair of cities is given in the table shown. One would like to situate the generators so that each city is within 50 miles of at least one generator.

	B	C	D	E	F	G
A	80	110	15	60	100	80
B		40	45	55	70	90
C			65	20	50	80
D				35	55	40
E					25	60
F						70

1.3.**10** Suppose there are k machines and l jobs, and each machine can do only a subset of the jobs. a) Draw a graph to model this situation. b) Express in terms of your graph model, the problem of assigning jobs to machines so that the maximum number of jobs can be performed simultaneously.

1.3.**11** A bridge tournament for five teams is to be scheduled so that each team plays two other teams.

1.3.**12** Let R be a binary *relation* on a set S. (Relations are discussed in Appendix A.2.)

a. Describe a digraph model for a binary relation on a finite set.

b. Draw the digraph for the relation R on the set $S = \{1, 2, 3, 4, 5\}$, given by

$$R = \{(1,2),(2,1),(1,1),(1,5),(4,5),(3,3)\}.$$

1.3.**13**$^{\mathbf{S}}$ Describe in graph-theory terms, the digraph properties corresponding to each of the following possible properties of binary relations.

a. reflexive; b. symmetric; c. transitive; d. antisymmetric.

1.4 WALKS AND DISTANCE

Many applications call for graph models that can represent traversal and distance. For instance, the number of node-links traversed by an email message on its route from sender to recipient is a form of distance. Beyond physical distance, another example is that a sequence of tasks in an activity-scheduling network forms a *critical path* if a delay in any one of the tasks would cause a delay in the overall project completion. This section and the following one clarify the notion of walk and related terminology.

Walks and Directed Walks

In proceeding continuously from a starting vertex to a destination vertex of a physical representation of a graph, one would alternately encounter vertices and edges. Accordingly, a *walk* in a graph is modeled by such a sequence.

DEFINITION: In a graph G, a ***walk*** from vertex v_0 to vertex v_n is an alternating sequence

$$W = \langle v_0, e_1, v_1, e_2, \ldots, v_{n-1}, e_n, v_n \rangle$$

of vertices and edges, such that $endpts(e_i) = \{v_{i-1}, v_i\}$, for $i = 1, \ldots, n$. If G is a digraph (or mixed graph), then W is a ***directed walk*** if each edge e_i is directed from vertex v_{i-1} to vertex v_i, i.e., $tail\,(e_i) = v_{i-1}$ and $head(e_i) = v_i$.

In a simple graph, there is only one edge between two consecutive vertices of a walk, so one could abbreviate the representation as a vertex sequence

$$W = \langle v_0, v_1, \ldots, v_n \rangle$$

In a general graph, one might abbreviate the representation as an edge sequence from the starting vertex to the destination vertex

$$W = \langle v_0, e_1, e_2, \ldots, e_n, v_n \rangle$$

TERMINOLOGY: A walk (or directed walk) from a vertex x to a vertex y is also called an ***x-y walk*** (or ***x-y directed walk***).

DEFINITION: The ***length*** of a walk or directed walk is the number of edge-steps in the walk sequence.

DEFINITION: A walk of length 0, i.e., with one vertex and no edges, is called a ***trivial walk***.

DEFINITION: A ***closed walk*** (or ***closed directed walk***) is a walk (or directed walk) that begins and ends at the same vertex. An ***open walk*** (or ***open directed walk***) begins and ends at different vertices.

Example 1.4.1: In Figure 1.4.1, there is an open walk of length 6,

$$< \text{OH, PA, NY, VT, MA, NY, PA} >$$

that starts at Ohio and ends at Pennsylvania. Notice that the given walk is an inefficient route, since it contains two repeated vertices and retraces an edge. §1.5 establishes the terminology necessary for distinguishing between walks that repeat vertices and/or edges and those that do not.

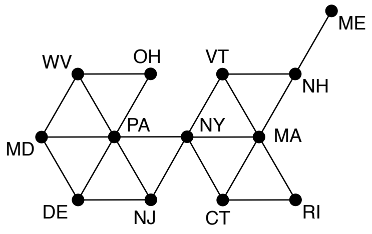
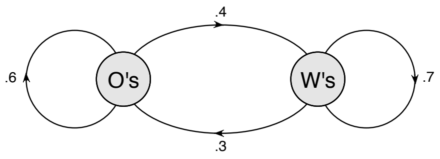


Figure 1.4.1 Geographic adjacency of the northeastern states.

Example 1.4.2: In the Markov diagram from Application 1.3.12, shown in Figure 1.4.2, the choice sequence of a cereal eater who buys O's, switches to W's, sticks with W's for two more boxes, and then switches back to O's is represented by the closed directed walk

$$< \text{O, W, W, W, O} >$$



	O's	W's
O's	.6	.4
W's	.3	.7

Figure 1.4.2 Markov process from Application 1.3.12.

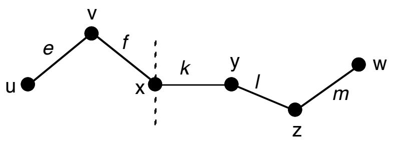
The product of the transition probabilities along a walk in any Markov diagram equals the probability that the process will follow that walk during an experimental trial. Thus, the probability that this walk occurs, when starting from O's equals $.4 \times .7 \times .7 \times .3 = 0.0588$.

Example 1.4.3: In the lexical scanner of Application 1.3.13, the identifier *counter12* would generate an open directed walk of length 9 from the start vertex to the accept vertex.

DEFINITION: The ***concatenation*** of two walks $W_1 = \langle v_0, e_1, \ldots, v_{k-1}, e_k, v_k \rangle$ and $W_2 = \langle v_k, e_{k+1}, v_{k+1}, e_{k+2}, \ldots, v_{n-1}, e_n, v_n \rangle$ such that walk W_2 begins where walk W_1 ends, is the walk

$$W_1 \circ W_2 = \langle v_0, e_1, \ldots, v_{k-1}, e_k, v_k, e_{k+1}, \ldots, v_{n-1}, e_n, v_n \rangle$$

Example 1.4.4: Figure 1.4.3 shows the concatenation of a walk of length 2 with a walk of length 3.

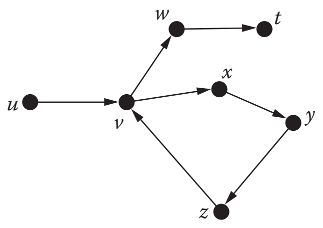


$$W_1 =< u, e, v, f, x >$$
$$W_2 =< x, k, y, l, z, m, w >$$
$$W_1 \circ W_2 =< u, e, v, f, x, k, y, l, z, m, w >$$

Figure 1.4.3 Concatenation of two walks.

DEFINITION: A ***subwalk*** of a walk $W = \langle v_0, e_1, v_1, e_2, \ldots, v_{n-1}, e_n, v_n \rangle$ is a subsequence of consecutive entries $S = \langle v_j, e_{j+1}, v_{j+1}, \ldots, e_k, v_k \rangle$ such that $0 \leq j \leq k \leq n$, that begins and ends at a vertex. Thus, the subwalk is itself a walk.

Example 1.4.5: In the figure below, the closed directed walk $\langle v, x, y, z, v \rangle$ is a subwalk of the open directed walk $\langle u, v, x, y, z, v, w, t \rangle$.



Distance

DEFINITION: The ***distance*** $d(s,t)$ from a vertex s to a vertex t in a graph G is the length of a shortest $s-t$ walk if one exists; otherwise, $d(s,\, t) = \infty$. For digraphs, the ***directed distance*** $\vec{d}(s,t)$ is the length of a shortest directed walk from s to t.

Example 1.4.6: In Figure 1.4.4, the distance from West Virginia to Maine is five. That is, starting in West Virginia, one cannot get to Maine without crossing at least five state borders.

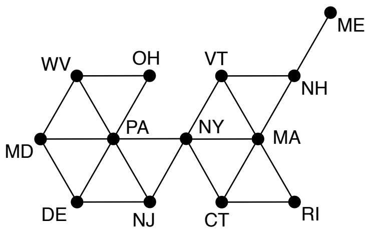


Figure 1.4.4 Geographic adjacency of the northeastern states.

A shortest walk (or directed walk) contains no repeated vertices or edges (see Exercises). It is instructive to think about how one might find a shortest walk. Ad hoc approaches are adequate for small graphs, but a systematic algorithm is essential for larger graphs (see §4.3).

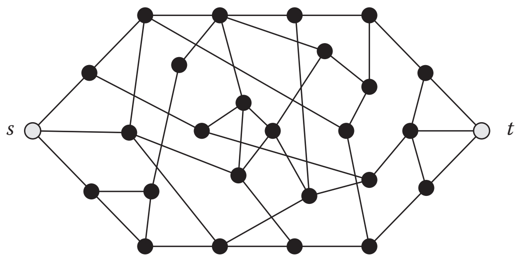


Figure 1.4.5 How might you find a shortest walk from s to t in this graph?

Eccentricity, Diameter, and Radius

DEFINITION: The ***eccentricity*** of a vertex v in a graph G, denoted $ecc(v)$, is the distance from v to a vertex farthest from v. That is,

$$ecc\,(v) = \max_{x \in V_G} \{d\,(v, x)\}$$

DEFINITION: The ***diameter*** of a graph G, denoted $diam(G)$, is the maximum of the vertex eccentricities in G or, equivalently, the maximum distance between two vertices in G. That is,

$$diam\,(G) = \max_{x \in V_G} \{ecc\,(x)\} = \max_{x,y \in V_G} \{d\,(x, y)\}$$

DEFINITION: The ***radius*** of a graph G, denoted $rad(G)$, is the minimum of the vertex eccentricities. That is,

$$rad\,(G) = \min_{x \in V_G} \{ecc\,(x)\}$$

DEFINITION: A ***central vertex*** v of a graph G is a vertex with minimum eccentricity. Thus, $ecc(v) = rad(G)$.

Example 1.4.7: The graph of Figure 1.4.6 has diameter 4, achieved by the vertex pairs u, v and u, w. The vertices x and y have eccentricity 2 and all other vertices have greater eccentricity. Thus, the graph has radius 2 and central vertices x and y.

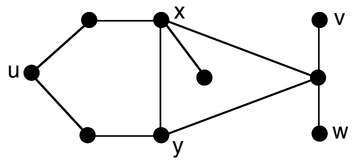


Figure 1.4.6 A graph with diameter 4 and radius 2.

Example 1.4.8: Let G be the graph whose vertex-set is the set of all people on the planet and whose edges correspond to the pairs of people who are acquainted. Then according to the *six degrees of separation* conjecture, the graph G has diameter 6.

Connectedness

DEFINITION: Vertex v is ***reachable from*** vertex u if there is a walk from u to v.

DEFINITION: A graph is ***connected*** if for every pair of vertices u and v, there is a walk from u to v.

DEFINITION: A digraph is ***connected*** if its underlying graph is connected.

TERMINOLOGY NOTE: Some authors use the term *weakly connected* digraph instead of connected digraph.

Example 1.4.9: The non-connected graph in Figure 1.4.7 is made up of connected pieces called *components*, and each component consists of vertices that are all reachable from one another. This concept is defined formally in §2.3.

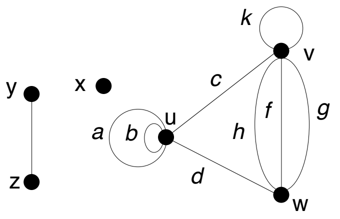


Figure 1.4.7 A non-connected graph with three components.

Strongly Connected Digraphs

DEFINITION: Two vertices u and v in a digraph D are said to be ***mutually reachable*** if D contains both a directed u-v walk and a directed v-u walk.

DEFINITION: A digraph D is ***strongly connected*** if every two of its vertices are mutually reachable.

Example 1.4.10: The digraph shown in Figure 1.4.8 is strongly connected.

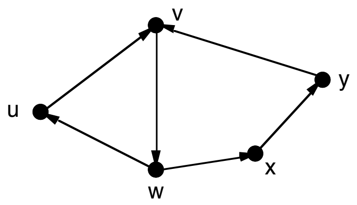


Figure 1.4.8 A strongly connected digraph.

Example 1.4.11: Suppose the graph in Figure 1.4.9 represents the network of roads in a certain national park. The park officials would like to make all of the roads one-way, but only if visitors will still be able to drive from anyone intersection to any other.

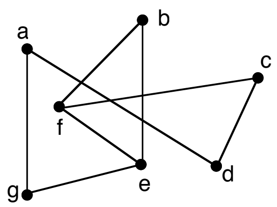


Figure 1.4.9 A national park's system of roads.

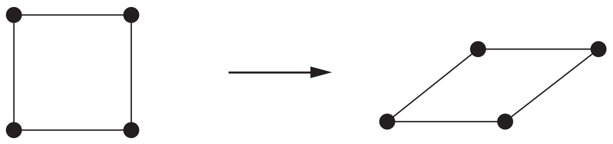
This problem can be expressed in graph-theoretic terms with the following definition.

DEFINITION: A graph is said to be ***strongly orientable*** if there is an assignment of directions to its edges so that the resulting digraph is strongly connected. Such an assignment is called a ***strong orientation***.

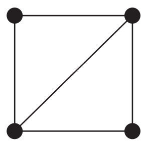
Chapter 9 includes a characterization of strongly orientable graphs, which leads to the design of a polynomial-time algorithm for determining whether a graph is strongly orientable.

Application of Connectedness to Rigidity

Application 1.4.1: *Rigidity of Rectangular Frameworks*[†] Consider a 2-dimensional framework of steel beams connected by joints that allow each beam to swivel in the plane. The framework is said to be ***rigid*** if none of its beams can swivel. The rectangle shown below is not rigid, since it can be deformed into a parallelogram.



Adding one diagonal brace would make the framework rigid, since the brace keeps the horizontal and vertical edges perpendicular to each other.



[†] No subsequent material depends on this application.

A rectangular framework with some diagonal braces might be rigid or non-rigid. For example, framework B in Figure 1.4.10 is rigid, but framework A is not.

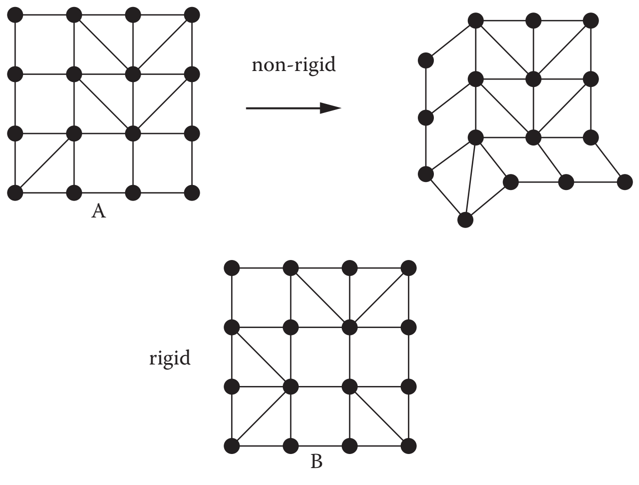


Figure 1.4.10 Two rectangular frameworks.

Surprisingly, the problem of determining whether a given framework is rigid can be reduced to a simple problem concerning connectivity in a bipartite graph.

Transforming the Problem: Suppose that an imaginary line is drawn through the middle of each row and each column of rectangles in the framework, as illustrated in the figure below. The framework is rigid if and only if each row-line r_i stays perpendicular to each column-line $c_j, 1, 2 \ldots$.

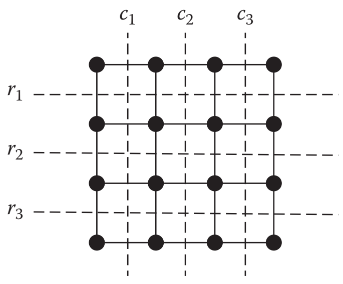
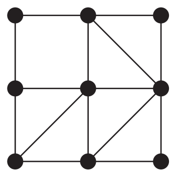


Illustration: Consider the following rigid 2-by-2 framework.



The brace in row 1, column 2 keeps $r_1 \perp c_2$. Similarly, the other two braces keep $c_2 \perp r_2$ and $r_2 \perp c_1$. Thus, each pair of consecutive entries in the sequence r_1, c_2, r_2, c_1 is perpendicular. This implies, by simple plane geometry, that $r_1 \perp c_1$, even though the (1, 1) rectangle is not braced. Therefore, the framework is rigid.

More generally, a framework is rigid if for each i and j there is a perpendicularity sequence that begins with r_i and ends with c_j. This observation leads to a simple method for determining whether a given rectangular framework is rigid.

Construct a bipartite graph by first drawing two sets of vertices, $r_1, r_2, \ldots$ and $c_1, c_2, \ldots,$ one set corresponding to the rows of the framework, and the other set corresponding to the columns. Then draw an edge between vertices r_i and c_j if there is a brace in the row i, column j rectangle of the framework.

By the observation above regarding perpendicularity sequences, the framework is rigid if and only if there is a walk in the bipartite graph between every pair of vertices, r_i and c_j. This simple criterion takes the form of the following theorem.

Theorem 1.4.1: *A rectangular framework is rigid if and only if its associated bipartite graph is connected.* ♢

Illustration: Figure 1.4.11 shows the two frameworks A and B from Figure 1.4.11 along with their associated bipartite graphs. Notice that for the non-rigid framework A, each deformable rectangle corresponds to a pair of vertices in the bipartite graph that has no walk between them. On the other hand, for the rigid framework B, the bipartite graph is connected.

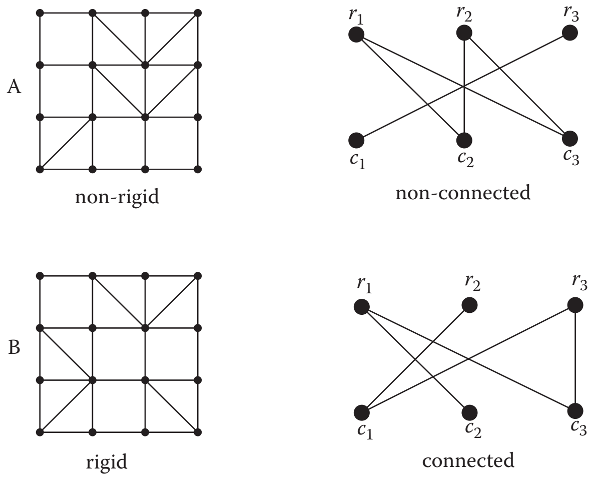
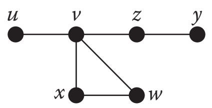


Figure 1.4.11 Sample applications for Theorem 1.4.1 on rigid frameworks.

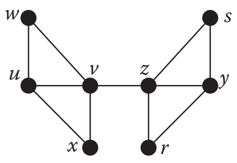
EXERCISES for Section 1.4

1.4.**1**^{S} Determine which of the following vertex sequences represent walks in the graph below.

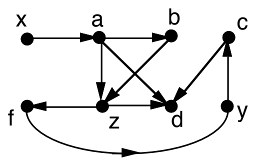
a. $\langle u, v \rangle$; b. $\langle v \rangle$; c. $\langle u, z, v \rangle$; d. $\langle u, v, w, v \rangle$



1.4.**2** Find all walks of length 4 or 5 from vertex w to vertex r in the following graph.



1.4.**3** Find all directed walks from x to d in the following digraph.



1.4.**4**$^{\mathbf{S}}$ Which of the following incidence tables represent connected graphs?

edge	a	b	c	d
endpts	u	v	w	x
	u	w	x	v

edge	a	b	c	d
endpts	u	v	v	x
	v	w	x	v

edge	a	b	c	d
endpts	u	v	w	x
	v	v	x	x

1.4.**5** Express the walk $\langle u, v, w, x, v, z, y, z, v\rangle$ as the concatenation of a series of walks such that each walk has no repeated vertices.

In Exercises 1.4.6 through 1.4.8, determine whether the given incidence table represents a strongly connected digraph. Assume that there are no isolated vertices.

1.4.**6**$^{\mathbf{S}}$

edges	a	b	c	d
endpts	u	v	v	x
	v^h	w^h	x^h	v^h

1.4.**7**

edges	a	b	c	d
endpts	u	v^h	w	x
	v^h	w	x^h	v^h

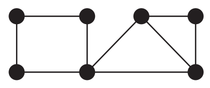
1.4.**8**

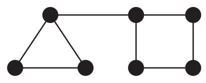
edges	a	b	c	d	e
endpts	u	v	w^h	v	w
	v^h	u^h	x	x^h	v^h

1.4.**9** Explain why all graphs having at least one edge have walks of arbitrarily large length. Can you make the same statement for digraphs?

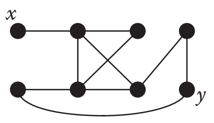
1.4.**10**$^{\mathbf{S}}$ Draw a simple digraph that has directed walks of arbitrarily large length.

1.4.**11** For each of the following graphs, either find a strong orientation or argue why no such orientation exists.

a. 

b. 

1.4.**12**$^{\mathbf{S}}$ Find the distance between vertices x and y in the following graph.



1.4.**13** Is there an even-length walk between two antipodal vertices (i.e., endpoints of a long diagonal) of a cube?

1.4.**14** Draw an 8-vertex connected graph with as few edges as possible.

1.4.**15**$^{\mathbf{S}}$ Draw a 7-vertex connected graph such that the removal of any one edge results in a non-connected graph.

1.4.**16** Draw an 8-vertex connected graph with no closed walks, except those that retrace at least one edge.

1.4.**17** Draw a 5-vertex connected graph that remains connected after the removal of any two of its vertices.

In Exercises 1.4.18 through 1.4.28, determine the diameter, radius, and central vertices of the graph indicated.

1.4.**18**$^{\mathbf{S}}$ The graph in Exercise 1.4.1.

1.4.**19** The graph in Exercise 1.4.2.

1.4.**20** Path graph $P_n, n \geq 3$ (from §1.2).

1.4.**21** Cycle graph $C_n, n \geq 4$ (from §1.2).

1.4.**22** Complete graph K_n , $n \geq 3$.

1.4.**23** Complete bipartite graph $K_{m,n}$, $m \geq n \geq 3$ (from §1.2).

1.4.**24** The Petersen graph (from §1.2).

1.4.**25** Hypercube graph Q_3; can you generalize to Q_n? (from §1.2).

1.4.**26** Circular ladder graph CL_n , $n \geq 4$ (from §1.2).

1.4.**27** Dodecahedral graph (from §1.2).

1.4.**28** Icosahedral graph (from §1.2).

1.4.**29** Determine the radius and diameter of each of the following circulant graphs (from §1.2).

a. $circ(5 : 1, 2)$; b. $circ(6 : 1, 2)$; c. $circ(8 : 1, 2)$.

1.4.**30** Describe the diameter and radius of the circulant graph $circ(n : m)$ in terms of integer n and connection m.

1.4.**31** Describe the diameter and radius of the circulant graph $circ(n : a, b)$ in terms of integer n and the connections a and b.

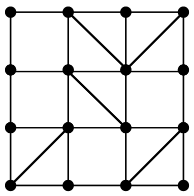
1.4.**32**$^{\mathbf{S}}$ Let G be a connected graph. Prove that

$$rad\,(G) \leq diam\,(G) \leq 2 \cdot rad\,(G)$$

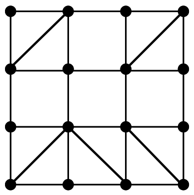
1.4.**33** Prove that a digraph must have a closed directed walk if each of its vertices has nonzero outdegree.

In Exercises 1.4.34 through 1.4.36, use a bipartite graph model to determine whether the given rectangular framework is rigid.

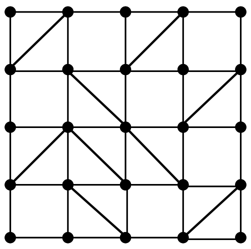
1.4.**34**$^{\mathbf{S}}$



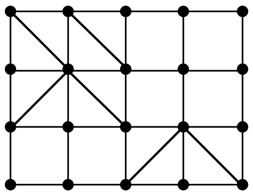
1.4.**35**



1.4.**36**



1.4.**37** Show that the framework shown below is not rigid. Can moving one brace make the framework rigid?



1.4.**38**$^{\mathbf{S}}$ Let u and v be any two vertices of a connected graph G. Prove that there exists a $u - v$ walk containing all the vertices of G.

1.4.**39** Prove that a shortest walk between two vertices cannot repeat a vertex or an edge.

1.4.**40** Let D be a digraph. Prove that mutual reachability is an equivalence relation on V_D.

1.4.**41**$^{\mathbf{S}}$ Let x and y be two different vertices in the complete graph K_4. Find the number of x-y walks of length 2 and of length 3.

1.4.**42** Let x and y be two different adjacent vertices in the complete bipartite graph $K_{3,3}$. Find the number of x-y walks of length 2 and of length 3.

1.4.**43** Let x and y be two different non-adjacent vertices in the complete bipartite graph $K_{3,3}$. Find the number of x-y walks of length 2 and of length 3.

1.4.**44** Prove that the distance function d on a graph G satisfies the *triangle inequality.* That is,

$$\text{For all } x, y, z \in V_G, \quad d(x,z) \le d(x,y) + d(y,z).$$

1.4.**45** Let G be a connected graph. Prove that if $d(x,y) \ge 2$, then there exists a vertex w such that $d(x, y) = d(x, w) + d(w, y)$.

1.4.**46**$^{\mathbf{S}}$ Let G be an n-vertex simple graph such that $\deg(v) \ge \frac{(n-1)}{2}$ for every vertex $v \in V_G$. Prove that graph G is connected.

1.5 PATHS, CYCLES, AND TREES

By stripping away its extraneous *detours*, an x-y walk can eventually be reduced to one that has no repetition of vertices or edges. Working with these shorter x-y walks simplifies discussion in many situations.

Trails and Paths

DEFINITION: A ***trail*** is a walk with no repeated edges.

DEFINITION: A ***path*** is a trail with no repeated vertices (except possibly the initial and final vertices).

DEFINITION: A walk, trail, or path is ***trivial*** if it has only one vertex and no edges.

TERMINOLOGY NOTE: Unfortunately, there is no universally agreed-upon terminology for walks, trails, and paths. For instance, some authors use the term *path* instead of walk and *simple path* instead of path. Others use the term *path* instead of trail. It is best to check the terminology when reading articles about this material.

Example 1.5.1: For the graph shown in Figure 1.5.1, $W = \langle v, a, e, f, a, d, z\rangle$ is the edge sequence of a walk but not a trail, because edge a is repeated, and $T = \langle v, a, b, c, d, e, u\rangle$ is a trail but not a path, because vertex x is repeated.

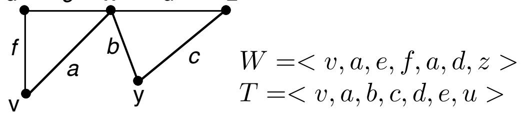


Figure 1.5.1 Walk W is not a trail, and trail T is not a path.

Remark: Directed trails and directed paths are directed walks that satisfy conditions analogous to those of their undirected counterparts. For example, a directed trail is a directed walk in which no directed edge appears more than once. When it is clear from the context that the objects under discussion are directed, then the modifier *directed* will be omitted.

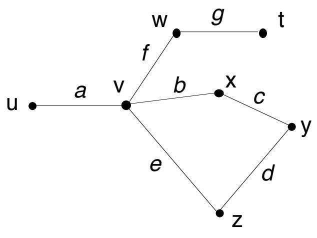
Deleting Closed Subwalks from Walks

DEFINITION: Given a walk $W = \langle v_0, e_1, \ldots, v_{n-1}, e_n, v_n \rangle$ that contains a nontrivial, closed subwalk $W' = \langle v_k, e_{k+1}, \ldots, v_{m-1}, e_m, v_k \rangle$, the ***reduction of walk*** W ***by subwalk*** W', denoted $W - W'$, is the walk

$$W - W' = \langle v_0, e_1, \ldots, v_{k-1}, e_k, v_k, e_{m+1}, v_{m+1}, \ldots, v_{n-1}, e_n \rangle$$

Thus, $W - W'$ is obtained by deleting from W all of the vertices and edges of W' except v_k. Or, less formally, $W - W'$ is obtained by ***deleting*** W' ***from*** W.

Example 1.5.2: Figure 1.5.2 shows a simple graph and the edge sequences of three walks, W, W', and $W - W'$. Observe that $W - W'$ is traversed by starting a traversal of walk W at vertex u and avoiding the detour W' at vertex v by continuing directly to vertex w.



$$W = < u, a, b, c, d, e, f, g, t >$$
$$W' = < v, b, c, d, e, v >$$
$$W - W' = < u, a, f, g, t >$$

Figure 1.5.2 Walk W is not a trail, and trail T is not a path.

Remark: The reduction of a walk by a (nontrivial) closed subwalk yields a walk that is obviously shorter than the original walk.

DEFINITION: Given two walks A and B, the walk B is said to be a ***reduced walk*** of A if there exists a sequence of walks $A = W_1, W_2, \ldots, W_r = B$ such that for each $i = 1, \ldots, r-1$, walk W_{i+1} is the reduction of W_i by some closed subwalk of W_i.

The next three results show that the concepts of distance and reachability in a graph can be defined in terms of paths instead of walks. This often simplifies arguments in which the detours of a walk can be ignored.

Lemma 1.5.1: *Every open x-y walk W is either an x-y path or contains a closed subwalk.*

Proof: If W is not an x-y path, then the subsequence of W between repeated vertices defines a closed subwalk of W. ♢

Theorem 1.5.2: *Let W be an open x-y walk. Then either W is an x-y path or there is an x-y path that is a reduced walk of W.*

Proof: If W is not an x-y path, then delete a closed subwalk from W to obtain a shorter x-y walk. Repeat the process until the resulting x-y walk contains no closed subwalks and, hence, is an x-y path, by Lemma 1.5.1. ♢

Corollary 1.5.3: *The distance from a vertex x to a reachable vertex y is always realizable by an x-y path.*

Proof: A shortest x-y walk must be an x-y path, by Theorem 1.5.2. ♢

Cycles

DEFINITION: A nontrivial closed path is called a ***cycle***.

DEFINITION: An ***acyclic graph*** is a graph that has no cycles.

DEFINITION: A cycle that includes every vertex of a graph is call a ***Hamiltonian cycle***.[†]

DEFINITION: A ***Hamiltonian graph*** is a graph that has a Hamiltonian cycle. (§6.3 elaborates on Hamiltonian graphs.)

Example 1.5.3: The graph in Figure 1.5.3 is Hamiltonian. The edges of the Hamiltonian cycle $\langle u, z, y, x, w, t, v, u\rangle$ are shown in bold.

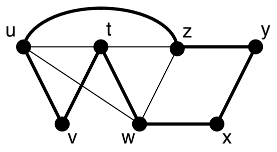


Figure 1.5.3 A Hamiltonian graph.

The following result gives an important characterization of bipartite graphs.

Theorem 1.5.4: *A graph G is bipartite if and only if it has no cycles of odd length.*

Proof: *Necessity* ($\Rightarrow$): Suppose that G is bipartite. Since traversing each edge in a walk switches sides of the bipartition, it requires an even number of steps for a walk to return to the side from which it started. Thus, a cycle must have even length.

Sufficiency ($\Leftarrow$): Let G be a graph with $n \geq 2$ vertices and no cycles of odd length. Without loss of generality, assume that G is connected. Pick any vertex u of G, and define a partition (X, Y) of V as follows:

$$X = \{x \mid d(u,x) \text{ is even}\}$$
$$Y = \{y \mid d(u,y) \text{ is odd}\}$$

If (X, Y) is not a bipartition of G, then there are two vertices in one of the sets, say v and w, that are joined by an edge, say e. Let P_1 be a shortest $u - v$ path, and let P_2 be a shortest $u - w$ path. By definition of the sets X and Y, the lengths of these paths are both even or both odd. Starting from vertex u, let z be the last vertex common to both paths (see Figure 1.5.4).

Since P_1 and P_2 are both shortest paths, their $u \to z$ sections have equal length. Thus, the lengths of the $z \to v$ section of P_1 and the $z \to w$ section of P_2 are either both even or both odd. But then the concatenation of those two sections with edge e forms a cycle of odd length, contradicting the hypothesis. Hence, (X, Y) is a bipartition of G. $\diamondsuit$

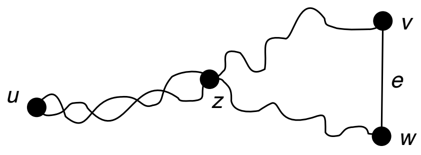


Figure 1.5.4 Figure for sufficiency part of Theorem 1.5.4 proof.

[†]The term "Hamiltonian" is in honor of the Irish mathematician William Rowan Hamilton.

The following proposition is similar to Theorem 1.5.2.

Proposition 1.5.5: *Every nontrivial, closed trail T contains a subwalk that is a cycle.*

Proof: Let T' be a minimum-length, nontrivial, closed subwalk of T. Its minimum length implies that T' has no proper closed subwalks, and, hence, its only repeated vertices are its first and last. Thus, T' is a cycle. ♢

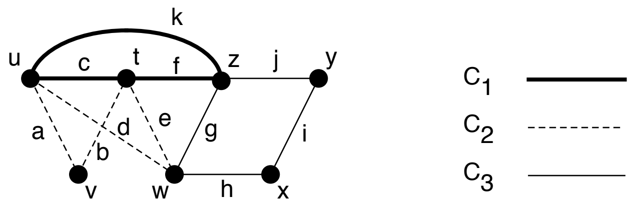
Remark: The assertion of Proposition 1.5.5 is no longer true if T is merely a closed walk (see Exercises).

DEFINITION: A collection of edge-disjoint cycles, $C_1, C_2, \ldots, C_m$, is called a ***decomposition*** of a closed trail T if each cycle C_i is either a subwalk or a reduced walk of T and the edge-sets of the cycles *partition* the edge-set of trail T.

Theorem 1.5.6: *A closed trail can be decomposed into edge-disjoint cycles.*

Proof: A closed trail having only one edge is itself a cycle. By way of induction, assume that the theorem is true for all closed trails having m or fewer edges. Next, let T be a closed trail with $m + 1$ edges. By Proposition 1.5.5, the trail T contains a cycle, say C, and $T - C$ is a closed trail having m or fewer edges. By the induction hypothesis, the trail $T - C$ can be decomposed into edge-disjoint cycles. Therefore, these cycles together with C form an edge-decomposition of trail T. ♢

Example 1.5.4: The three cycles C_1, C_2, and C_3 form a decomposition of the closed trail T in Figure 1.5.5. The edge-based abbreviated representation of walks is used here to emphasize the edge partition of the decomposition.



$$T = < u, a, b, c, d, e, f, g, h, i, j, k, u >$$

$$C_1 = < z, k, c, f, z >; \;\; C_2 = < u, a, b, e, d, u >; \;\; C_3 = < w, h, i, j, g, w >$$

Figure 1.5.5 A closed trail T decomposed into three cycles.

Eulerian Trails

DEFINITION: An ***Eulerian trail*** in a graph is a trail that contains every edge of that graph.

DEFINITION: An ***Eulerian tour*** is a closed Eulerian trail.

DEFINITION: An ***Eulerian graph*** is a connected graph that has an Eulerian tour.[†]

[†]The term "Eulerian" is in honor of the Swiss mathematician Leonhard Euler.

Example 1.5.5: The trail $T = \langle u, a, b, c, d, e, f, g, h, i, j, k, u \rangle$ in Figure 1.5.6 is an Eulerian tour. Its decomposition into edge-disjoint cycles, as in Figure 1.5.5, is a property that holds for all Eulerian tours and is part of a classical characterization of Eulerian graphs given in §4.5.

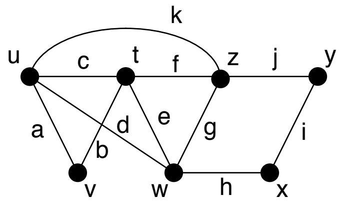


Figure 1.5.6 An Eulerian graph.

Application 1.5.1: *Traversing the Edges of a Network* Suppose that the graph shown in Figure 1.5.6 represents a network of railroad tracks. A special cart equipped with a sensing device will traverse every section of the network, checking for imperfections. Can the cart be routed so that it traverses each section of track exactly once and then returns to its starting point?

The problem is equivalent to determining whether the graph is Eulerian, and as seen from Example 1.5.5, trail T does indeed provide the desired routing.

Algorithms to construct Eulerian tours and several other applications involving Eulerian graphs appear in Chapter 6.

Girth

DEFINITION: The ***girth*** of a graph G with at least one cycle is the length of a shortest cycle in G. The girth of an acyclic graph is undefined.

Example 1.5.6: The girth of the graph in Figure 1.5.7 below is 3 since there is a 3-cycle but no 2-cycle or 1-cycle.

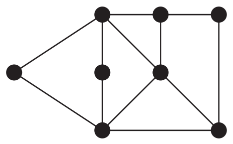


Figure 1.5.7 A graph with girth 3.

Trees

DEFINITION: A ***tree*** is a connected graph that has no cycles.

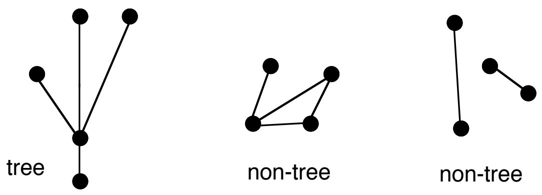


Figure 1.5.8 A tree and two non-trees.

Trees are central to the structural understanding of graphs, and they have a wide range of applications, including information processing. Trees often occur with additional attributes such as roots and vertex orderings. Chapters 3 and 4 are devoted entirely to establishing and applying many of the properties of trees. The following examples provide a preview of some of the material on trees and illustrate how they arise naturally.

Application 1.5.2: *Physical Chemistry* A hydrocarbon is a chemical molecule composed of hydrogen and carbon atoms. It is said to be *saturated* if it includes the maximum number of hydrogen atoms for its number of carbon atoms. Saturation occurs when no two carbon atoms have a multiple bond, that is, when the hydrocarbon's structural model is a simple graph. We may recall from elementary chemistry that a hydrogen atom has one electron, and thus it is always 1-valent in a molecule. Also, a carbon atom has four electrons in its outer orbit, making it 4-valent. The saturated hydrocarbons butane and isobutane both have the same chemical formula, i.e., C_4H_{10}, as shown in Figure 1.5.9, so they are said to be *isomers*.

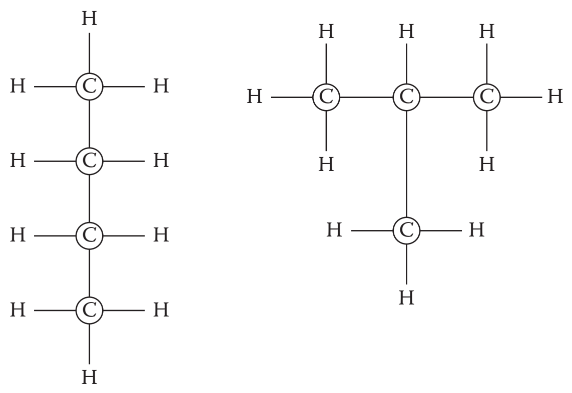


Figure 1.5.9 Two saturated hydrocarbons: butane and isobutane.

Application 1.5.3: *Operating-System Directories* Many information structures of computer science are based on trees. The directories and subdirectories (or folders and subfolders) containing a computer user's files are typically represented by the operating system as vertices in a *rooted tree* (see §3.2), as illustrated in Figure 1.5.10.

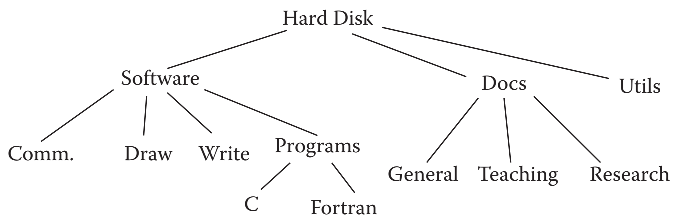


Figure 1.5.10 A directory structure modeled by a rooted tree.

Application 1.5.4: *Information Retrieval* *Binary trees* (see §3.2) store ordered data in a manner that permits an efficient search. In particular, if n nodes of a binary tree are used appropriately to store the *keys* of n data elements, then a search can be completed in running time $O(\log n)$. Figure 1.5.11 shows a set of 3-digit keys that are stored in a binary tree. Notice that all the keys in the *left subtree* of each node are smaller than the key at that node, and that all the keys in the *right subtree* are larger. This is called the *binary-search-tree property.*

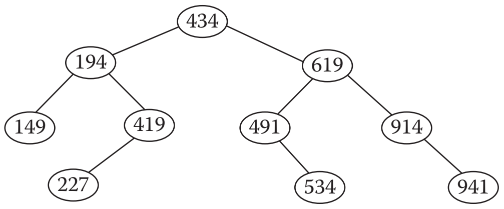


Figure 1.5.11 A binary-search tree storing 3-digit keys.

Application 1.5.5: *Minimal Connected Networks* Suppose a network is to be created from n computers. There are $\binom{n}{2} = \frac{n^2-n}{2}$ possible pairs of computers that can be directly joined. If all of these pairs are linked, then the n computers will certainly be connected, but many of these links will be unnecessary. Figure 1.5.12 shows a network connected with a minimum number of edges. Notice that these edges form a tree, and that the number of edges in the tree is one less than the number of vertices.

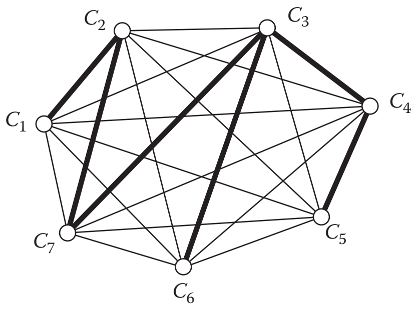
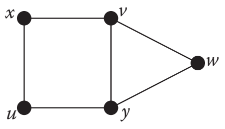


Figure 1.5.12 Connecting seven computers using a minimum number of edges.

EXERCISES for Section 1.5

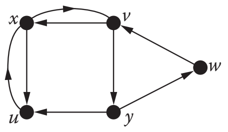
1.5.1^S Which of the following vertex sequences represent a walk in the graph shown below? Which walks are paths? Which walks are cycles? What are the lengths of those that are walks?

a. $\langle u, y, v, w, v \rangle$
b. $\langle u, y, u, x, v, w, u \rangle$
c. $\langle y, v, u, x, v, y \rangle$
d. $\langle w, v, x, u, y, w \rangle$



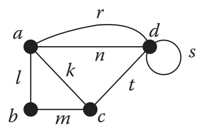
1.5.**2** Which of the following vertex sequences represent a directed walk in the graph shown below? Which directed walks are directed paths? Which directed walks are directed cycles? What are the lengths of those that are directed walks?

a. $\langle x, v, y, w, v\rangle$
b. $\langle x, u, x, u, x\rangle$
c. $\langle x, u, v, y, x\rangle$
d. $\langle x, v, y, w, v, u, x\rangle$



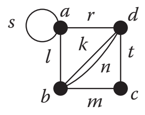
1.5.**3**$^{\mathbf{S}}$ In the graph shown at right, find

a. a closed walk that is not a trail.
b. a closed trail that is not a cycle.
c. all the cycles of length 5 or less.

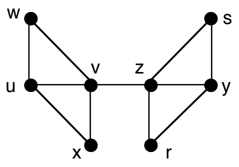


1.5.**4** In the graph shown at right, find

a. a walk of length 7 between vertices b and d.
b. a path of maximum length.



1.5.**5**$^{\mathbf{S}}$ Determine the number of paths from w to r in the following graph.

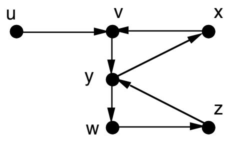


1.5.**6** Draw a copy of the Petersen graph (from §1.2) with vertex labels. Then find

a. a trail of length 5.
b. a path length 9.
c. cycles of lengths 5, 6, 8, and 9.

1.5.**7** Consider the digraph shown at the right. Find

a. an open directed walk that is not a directed trail.
b. an open directed trail that is not a directed path.
c. a closed directed walk that is not a directed cycle.



1.5.**8**$^{\mathbf{S}}$ Give an example of a nontrivial, closed walk that does not contain a cycle.

1.5.**9** Let x and y be two different vertices in the complete graph K_5. Find the number of x-y paths of length 2 and of length 3.

1.5.**10** Let x and y be two different vertices in the complete graph K_n, $n \geq 5$. Find the number of x-y paths of length 4.

1.5.**11**$^{\mathbf{S}}$ Let x and y be two adjacent vertices in the complete bipartite graph $K_{3,3}$. Find the number of x-y paths of length 2, or length 3, and of length 4.

1.5.**12** Let x and y be two different non-adjacent vertices in the complete bipartite graph $K_{3,3}$. Find the number of x-y paths of length 2, of length 3, and of length 4.

1.5.**13** Let x and y be two adjacent vertices in the complete bipartite graph $K_{n,n}$, $n \geq 3$. Find the number of x-y paths of length 2, of length 3, and of length 4.

1.5.**14** Let x and y be two different non-adjacent vertices in the complete bipartite graph $K_{n,n}, n \geq 3$. Find the number of x-y paths of length 2, of length 3, and of length 4.

In Exercises 1.5.15 through 1.5.22, determine the girth of the graph indicated.

1.5.**15**$^{\mathbf{S}}$ Complete bipartite graph $K_{3,7}$.

1.5.**16** Complete bipartite graph $K_{m,n}$, $m \geq n \geq 3$.

1.5.**17** Complete graph K_n.

1.5.**18** Hypercube graph Q_5. Can you generalize to Q_n?

1.5.**19** Circular ladder graph CL_6. Can you generalize to CL_n?

1.5.**20** The Petersen graph.

1.5.**21** Dodecahedral graph.

1.5.**22** Icosahedral graph.

1.5.**23** Determine the girth of each of the following circulant graphs (from §1.2).

a. $circ(5:1,2)$; b. $circ(6:1,2)$; c. $circ(8:1,2)$.

1.5.**24** Describe the girth of the circulant graph $circ(n:m)$ in terms of integer n and connection m.

1.5.**25** Describe the girth of the circulant graph $circ(n:a,b)$ in terms of integer n and the connections a and b.

1.5.**26** Find necessary and sufficient conditions for the circulant graph $circ(n:a,b)$ to be Hamiltonian.

1.5.**27** Determine whether the hypercube graph Q_3 is Hamiltonian.

1.5.**28** Determine whether the Petersen graph is Hamiltonian.

1.5.**29** Determine whether the circular ladder graph CL_n, $n \geq 3$, is Hamiltonian.

1.5.**30**$^{\mathbf{S}}$ Prove that if v is a vertex on a nontrivial, closed trail, then v lies on a cycle.

1.5.**31** Prove or disprove: Every graph that has a closed walk of odd length has a cycle.

1.5.**32**$^{\mathbf{S}}$ Prove that in a digraph, a shortest directed walk from a vertex x to a vertex y is a directed path from x to y.

1.5.**33** Suppose G is a simple graph whose vertices all have degree at least k.

a. Prove that G contains a path of length k.

b. Prove that if $k \geq 2$, then G contains a cycle of length at least k.

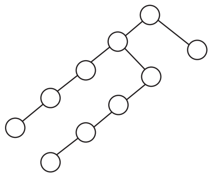
1.5.**34** State and prove the digraph version of Theorem 1.5.2.

1.5.**35** State and prove the digraph version of Proposition 1.5.5.

1.5.**36** State and prove the digraph version of Theorem 1.5.6.

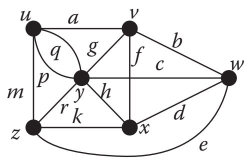
1.5.**37** Find a collection of cycles whose edge-sets partition the edge-set of the closed trail in Example 1.5.4 but that is not a decomposition of the closed trail (i.e., at least one of the cycles in the collection is neither a subwalk nor a reduced walk of the closed trail).

1.5.**38** a. In the following binary tree, store the ten 3-digit keys of Figure 1.5.11, in adherence to the binary-search-tree property.



b. Is this tree easier or harder to search than the one in Figure 1.5.11?

1.5.**39** Find an Eulerian tour in the following graph.



1.5.**40**$^{\mathbf{S}}$ Describe how to use a rooted tree to represent all possible moves of a game of tic-tac-toe, where each node in the tree corresponds to a different board configuration.

1.5.**41** Which of the platonic graphs are bipartite?

1.5.**42** Prove that if G is a digraph such that every vertex has positive indegree, then G contains a directed cycle.

1.5.**43**$^{\mathbf{S}}$ Prove that if G is a digraph such that every vertex has positive outdegree, then G contains a directed cycle.

1.6 VERTEX AND EDGE ATTRIBUTES: MORE APPLICATIONS

The all-purpose model of a graph includes attribute lists for edges and vertices. In viewing digraphs as a species of graphs, direction is an edge attribute. The Markov-diagram and lexical-scanner applications, introduced in §1.3, gave a first glimpse of graph models that require edge labels, another kind of edge attribute.

Four Classical Edge-Weight Problems in Combinatorial Optimization

DEFINITION: A ***weighted graph*** is a graph in which each edge is assigned a number, called its ***edge-weight***.

Edge-weights are among the most commonly used attributes of graphs, especially in combinatorial optimization. For instance, the edge-weight might represent transportation cost, travel time, spatial distance, power loss, upper bounds on flow, or inputs and outputs for transitions in a finite state machine.

The definitions of length and distance given in Section §1.4 extend naturally to weighted graphs.

DEFINITION: The ***length*** of a path in a weighted graph is the sum of the edge-weights of the path. The ***distance*** between two vertices s and t in a weighted graph is the length of a shortest s-t path.

Application 1.6.1: *Shortest-Path Problem* Suppose that each edge-weight in the digraph of Figure 1.6.1 represents the time it takes to traverse that edge. Find the shortest path (in time) from vertex s to vertex t (see Chapter 4).

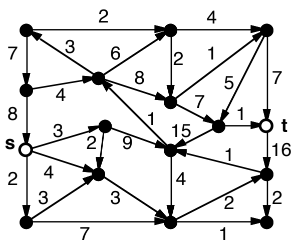


Figure 1.6.1 A weighted directed graph for a shortest-path problem.

Application 1.6.2: *Minimum-Weight Spanning-Tree Problem* Suppose that several computers in fixed locations are to be linked to form a computer network. The cost of creating a direct connection between a pair of computers is known for each possible pair and is represented by the edge weights given in Figure 1.6.2. Determine which direct links to construct so that the total networking cost is minimized (see Chapter 4).

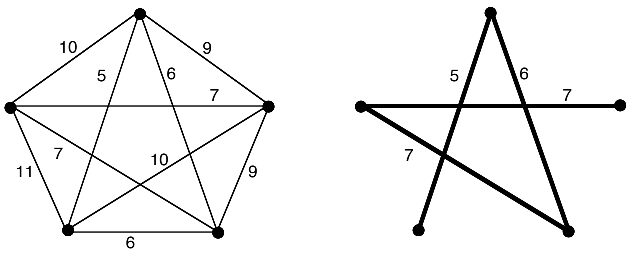


Figure 1.6.2 A weighted graph and its minimum spanning tree.

Application 1.6.3: *Traveling Salesman Problem* Suppose that a salesman must visit several cities during the next month. The edge-weights shown in Figure 1.6.3 represent the travel costs between each pair of cities. The problem is to sequence the visits so that the salesman's total travel cost is minimized. In other words, find a Hamiltonian cycle whose total edge-weight is a minimum (see Chapter 6).

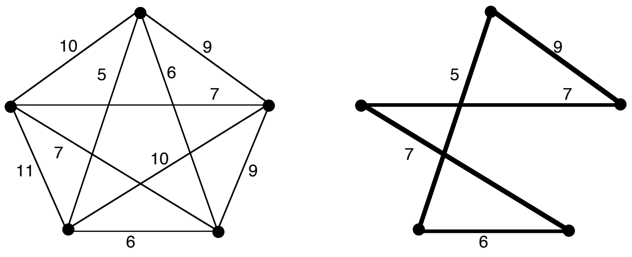


Figure 1.6.3 A weighted graph and its optimal traveling salesman tour.