

DISCRETE MATHEMATICS AND ITS APPLICATIONS  
Series Editor KENNETH H. ROSEN

# Graph Theory and ITS APPLICATIONS

*Second Edition*

Jonathan L. Gross  
Jay Yellen



Chapman & Hall/CRC  
Taylor & Francis Group

DISCRETE MATHEMATICS AND ITS APPLICATIONS  
Series Editor KENNETH H. ROSEN

# **GRAPH THEORY AND ITS APPLICATIONS**

## **SECOND EDITION**

# DISCRETE MATHEMATICS AND ITS APPLICATIONS

Series Editor  
Kenneth H. Rosen, Ph.D.

- Juergen Bierbrauer, Introduction to Coding Theory*
- Kun-Mao Chao and Bang Ye Wu, Spanning Trees and Optimization Problems*
- Charalambos A. Charalambides, Enumerative Combinatorics*
- Henri Cohen, Gerhard Frey, et al., Handbook of Elliptic and Hyperelliptic Curve Cryptography*
- Charles J. Colbourn and Jeffrey H. Dinitz, The CRC Handbook of Combinatorial Designs*
- Steven Furino, Ying Miao, and Jianxing Yin, Frames and Resolvable Designs: Uses, Constructions, and Existence*
- Randy Goldberg and Lance Riek, A Practical Handbook of Speech Coders*
- Jacob E. Goodman and Joseph O'Rourke, Handbook of Discrete and Computational Geometry, Second Edition*
- Jonathan L. Gross and Jay Yellen, Graph Theory and Its Applications, Second Edition*
- Jonathan L. Gross and Jay Yellen, Handbook of Graph Theory*
- Darrel R. Hankerson, Greg A. Harris, and Peter D. Johnson, Introduction to Information Theory and Data Compression, Second Edition*
- Daryl D. Harms, Miroslav Kraetzl, Charles J. Colbourn, and John S. Devitt, Network Reliability: Experiments with a Symbolic Algebra Environment*
- Derek F. Holt with Bettina Eick and Eamonn A. O'Brien, Handbook of Computational Group Theory*
- David M. Jackson and Terry I. Visentin, An Atlas of Smaller Maps in Orientable and Nonorientable Surfaces*
- Richard E. Klima, Ernest Stitzinger, and Neil P. Sigmon, Abstract Algebra Applications with Maple*
- Patrick Knupp and Kambiz Salari, Verification of Computer Codes in Computational Science and Engineering*
- William Kocay and Donald L. Kreher, Graphs, Algorithms, and Optimization*
- Donald L. Kreher and Douglas R. Stinson, Combinatorial Algorithms: Generation Enumeration and Search*
- Charles C. Lindner and Christopher A. Rodgers, Design Theory*
- Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone, Handbook of Applied Cryptography*

### ***Continued Titles***

*Richard A. Mollin, Algebraic Number Theory*

*Richard A. Mollin, Codes: The Guide to Secrecy from Ancient to Modern Times*

*Richard A. Mollin, Fundamental Number Theory with Applications*

*Richard A. Mollin, An Introduction to Cryptography*

*Richard A. Mollin, Quadratics*

*Richard A. Mollin, RSA and Public-Key Cryptography*

*Kenneth H. Rosen, Handbook of Discrete and Combinatorial Mathematics*

*Douglas R. Shier and K.T. Wallenius, Applied Mathematical Modeling: A Multidisciplinary Approach*

*Jörn Steuding, Diophantine Analysis*

*Douglas R. Stinson, Cryptography: Theory and Practice, Second Edition*

*Roberto Togneri and Christopher J. deSilva, Fundamentals of Information Theory and Coding Design*

*Lawrence C. Washington, Elliptic Curves: Number Theory and Cryptography*



DISCRETE MATHEMATICS AND ITS APPLICATIONS  
Series Editor KENNETH H. ROSEN

# **GRAPH THEORY AND ITS APPLICATIONS**

## **SECOND EDITION**

**JONATHAN L. GROSS  
JAY YELLEN**



**Chapman & Hall/CRC**  
Taylor & Francis Group

Boca Raton London New York

Copyright Jonathan L. Gross and Jay Yellen

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

© 2006 by Taylor & Francis Group, LLC  
CRC Press is an imprint of Taylor & Francis Group, an Informa business

No claim to original U.S. Government works  
Version Date: 20110713

International Standard Book Number-13: 978-1-4200-5714-0 (eBook - PDF)

This book contains information obtained from authentic and highly regarded sources. Reasonable efforts have been made to publish reliable data and information, but the author and publisher cannot assume responsibility for the validity of all materials or the consequences of their use. The authors and publishers have attempted to trace the copyright holders of all material reproduced in this publication and apologize to copyright holders if permission to publish in this form has not been obtained. If any copyright material has not been acknowledged please write and let us know so we may rectify in any future reprint.

Except as permitted under U.S. Copyright Law, no part of this book may be reprinted, reproduced, transmitted, or utilized in any form by any electronic, mechanical, or other means, now known or hereafter invented, including photocopying, microfilming, and recording, or in any information storage or retrieval system, without written permission from the publishers.

For permission to photocopy or use material electronically from this work, please access [www.copyright.com](http://www.copyright.com) (<http://www.copyright.com/>) or contact the Copyright Clearance Center, Inc. (CCC), 222 Rosewood Drive, Danvers, MA 01923, 978-750-8400. CCC is a not-for-profit organization that provides licenses and registration for a variety of users. For organizations that have been granted a photocopy license by the CCC, a separate system of payment has been arranged.

**Trademark Notice:** Product or corporate names may be trademarks or registered trademarks, and are used only for identification and explanation without intent to infringe.

Visit the Taylor & Francis Web site at  
<http://www.taylorandfrancis.com>

and the CRC Press Web site at  
<http://www.crcpress.com>

# PREFACE

Interest in graphs and their applications continues to grow rapidly, largely due to the usefulness of graphs as models for computation and optimization. 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, characteristic of mathematical areas with longer traditions, such as linear algebra and group theory. In the process, important techniques and analytic tools are transformed into a unified mathematical methodology.

Emphasis throughout is conceptual, with more than 800 graph drawings included to strengthen intuition and more than 2000 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, shorn of the details of computer implementation. 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.

## 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.

Chapters 7 and 8 present some of the underpinnings of topological graph theory. Chapter 7 is devoted to planarity and Kuratowski's theorem, and in Chapter 8, the scope of graph drawings is expanded in multiple directions, including the topological model for drawings, drawings on higher-order surfaces, and computer drawings. Chapter 9 is about graph colorings, including vertex- and edge-colorings, map-colorings, and the related topics of cliques, independence numbers, and graph factorization.

Chapter 10 discusses graph measurement, including measurements of graph mappings. Chapter 11 provides a brief introduction to analytic graph theory, which comprises three of the most extensively developed branches of graph theory – Ramsey theory, extremal graphs, and random graphs. The material in Chapters 12 and 13, special digraph models and network flows, overlaps with various areas in operations research and computer science.

Chapters 14 through 16, the most algebraic chapters, concern enumeration, specification by voltage graphs, and constructing nonplanar layouts.

Most of the material in Chapters 1 through 6 assumes no prerequisite. However, some familiarity with topics typically found in an undergraduate course in discrete mathematics would be useful, and those topics appear briefly in the appendix (along

with some linear algebra, which is used in §4.6). The appendix also provides a quick review (including permutation groups) for some of the more advanced topics in the later chapters.

### To the Instructor

The book has ample material for a two-semester course, and a variety of one-semester courses can be designed with different slants using various combinations of chapters. An introductory one-semester general course in graph theory would typically include most of the topics covered in the first nine chapters, except possibly parts of Chapter 8. However, some instructors of a fast-paced one-semester course might consider leaving various sections of earlier chapters for self-study by students. This would allow more time for selections from later chapters.

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 several notable exceptions, as follows:

- §3.2 is used in §4.1.
- §4.1 and §4.2 are used in §5.4 and §12.5.
- §4.5 is referred to in §5.3 and §6.1.
- Parts of Chapter 5 are used in §13.3.
- Chapter 8 is used in §16.1 and §16.2.
- §15.1, §15.2, and §15.3 are used in §16.3, §16.4, and §16.5.

A course oriented toward *operations research/optimization* should include most or all of the material in Chapters 4 (spanning trees), 5 (connectivity), 6 (traversability), 9 (colorings), 12 (digraph models), and 13 (flows), along with various other sections of the instructor's choice, depending on the time available. A course emphasizing the role of *data structures and algorithms* might add to the above topics more material from Chapter 3 (trees). A more algebraic and topological course in graph theory might replace some of these selections with parts or all of Chapter 7 (planarity), Chapter 8 (graph drawings), Chapter 14 (enumeration), Chapter 15 (voltage graphs), and Chapter 16 (graphs on general surfaces).

### New Features in the Second Edition

- *Solutions and Hints.* Each exercise marked with a superscript<sup>s</sup> has a solution or hint appearing in the back of the book.
- *Supplementary Exercises.* In addition to the section exercises, each chapter now 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.
- *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.

## New Material in the Second Edition

We were gratified to see the first edition of this book used successfully at many colleges and universities in North America, Europe, Asia, and Oceania. Suggestions from instructors at these institutions, and from their students, led to several improvements in this second edition, including the addition of much new material. The inclusion of other new material was inspired by contributions to our *Handbook of Graph Theory*. Almost nothing has been deleted from the first edition, and well over 100 pages of new or expanded material have been added to the second edition. The following 9 sections are all new:

- §8.6 Geometric Drawings
- §9.4 Factorization
- §10.1 Distance in Graphs
- §10.2 Domination in Graphs
- §10.3 Bandwidth
- §10.4 Intersection Graphs
- §11.1 Ramsey Graph Theory
- §11.2 Extremal Graph Theory
- §11.3 Random Graphs

## Highlights of Changes in the Second Edition

Now, at the start of Chapter 1, we introduce the notation  $uv$  for an edge between vertices  $u$  and  $v$  of a simple graph. Our move to this more convenient notation was motivated by comments from our colleagues who work primarily in simple graphs. Of course, we continue to use explicit edge names in contexts where multiple edges may be present. We have expanded Chapter 2 with increased discussion of isomorphism testing, automorphisms and symmetry, vertex and edge orbits, graph reconstruction, and some graph operations that appeared much later in the first edition.

We revised and reorganized some of the material in Chapters 3 and 4 to create a smoother flow and to make the unifying treatment of the tree-growing algorithms more transparent. The main change in Chapter 5 is the inclusion in §5.2 of a short proof of the Whitney-Robbins characterization of 2-edge-connected graphs.

With the new Chapter 7, Kuratowski's Theorem and its proof (which were in Chapter 9 of the first edition) can now be studied without having to first cover the more demanding topology material in Chapter 8. Chapter 7 now also includes crossing numbers and thickness (from Chapter 15 of the first edition), which are more closely related to planarity than to higher order surfaces. Most of the content of the old Chapter 7 (in the first edition) was redistributed to other chapters. Chapter 9 on graph colorings is essentially the old Chapter 10 together with the new section on factorization (§9.4).

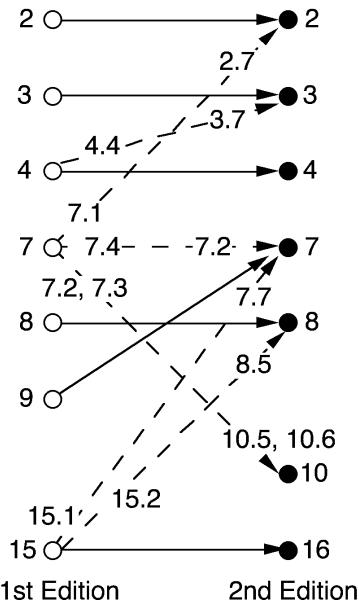
Chapters 10 and 11 are entirely new, except for the two sections on graph mappings (§10.5 and §10.6), which migrated from the old Chapter 7. Except for some reorganization within each chapter, Chapter 12 is the old Chapter 11, and Chapter 13 is the old Chapter 12.

The new Chapters 14, 15, and 16 were the old Chapters 13, 14, and 15, respectively. Chapter 14 (graphical enumeration) has several improved proofs, especially of Burnside's Lemma. Chapter 16 (non-planar layouts) no longer includes the sections on crossing numbers and thickness, now in Chapter 7, or the section on generalizing planar drawings to higher-order surfaces, now in Chapter 8.

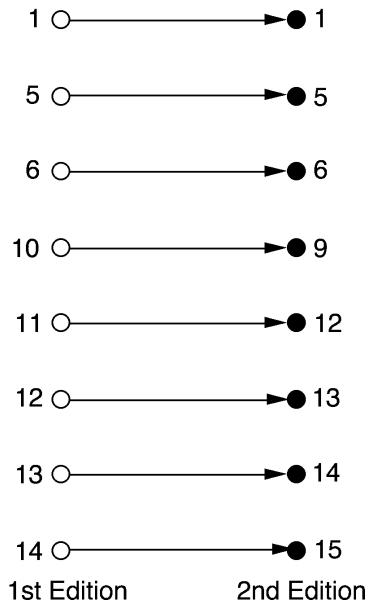
### First Edition to Second Edition at a Glance

The section migrations are shown in the figure on the left in the diagram below. For instance, §4.4 in the first edition became §3.7 in the second edition.

Substantial chapter rearrangements.



Intact chapter transfers



### Websites

Suggestions and comments from readers are welcomed and may be sent to the authors' website at [www.graphtheory.com](http://www.graphtheory.com), which, thanks mostly to the efforts of our colleague Dan Sanders and our webmaster Aaron Gross, also maintains extensive graph theory informational resources. The general website for CRC Press is [www.crcpress.com](http://www.crcpress.com).

In advance, we thank our students, colleagues, and other readers for notifying us of any errors that they may find. As with the first edition, we will post the corrections to all known errors on our website.

### Acknowledgements

Several readers of our manuscript at various stages offered many helpful suggestions regarding the mathematical content. In particular, we would like to thank Bob Brigham, Jianer Chen, Lynn Kiaer, Ward Klein, Ben Manvel, Buck McMorris, Ken Rosen, Greg Starling, Joe Straight, Tom Tucker, and Dav Zimak. We also thank Betsey Maupin for her proofreading and for her many stylistic suggestions. Special thanks to Ward Klein for his considerable assistance with proofreading the manuscript for both editions.

Jonathan Gross and Jay Yellen

## ABOUT THE AUTHORS

Jonathan 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.

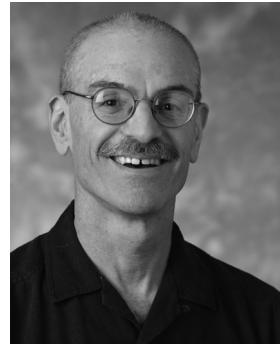


Professor Gross has created and delivered numerous software-development short courses for Bell Laboratories and for IBM. These include mathematical methods for performance evaluation at the advanced level and for developing reusable software at a basic level. He has received several awards for outstanding 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. Another previous book, *Measuring Culture*, coauthored with Steve Rayner, constructs network-theoretic tools for measuring sociological phenomena.

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.



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 graph algorithms for software testing and course timetabling.



Jonathan dedicates this book to Alisa.

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



# CONTENTS

## *Preface*

<b>1. INTRODUCTION to GRAPH MODELS</b>	<b>1</b>
1.1 Graphs and Digraphs .....	2
1.2 Common Families of Graphs .....	15
1.3 Graph Modeling Applications .....	22
1.4 Walks and Distance .....	28
1.5 Paths, Cycles, and Trees .....	39
1.6 Vertex and Edge Attributes: More Applications .....	48
1.7 Supplementary Exercises .....	51
Glossary .....	53
<b>2. STRUCTURE and REPRESENTATION</b>	<b>57</b>
2.1 Graph Isomorphism .....	58
2.2 Automorphisms and Symmetry .....	65
2.3 Subgraphs .....	72
2.4 Some Graph Operations .....	80
2.5 Tests for Non-Isomorphism .....	89
2.6 Matrix Representations .....	95
2.7 More Graph Operations .....	101
2.8 Supplementary Exercises .....	107
Glossary .....	110
<b>3. TREES</b>	<b>115</b>
3.1 Characterizations and Properties of Trees .....	116
3.2 Rooted Trees, Ordered Trees, and Binary Trees .....	124
3.3 Binary-Tree Traversals .....	132
3.4 Binary-Search Trees .....	137
3.5 Huffman Trees and Optimal Prefix Codes .....	141
3.6 Priority Trees .....	146
3.7 Counting Labeled Trees: Prüfer Encoding .....	151
3.8 Counting Binary Trees: Catalan Recursion .....	156
3.9 Supplementary Exercises .....	158
Glossary .....	160
<b>4. SPANNING TREES</b>	<b>163</b>
4.1 Tree Growing .....	164
4.2 Depth-First and Breadth-First Search .....	171
4.3 Minimum Spanning Trees and Shortest Paths .....	176
4.4 Applications of Depth-First Search .....	182
4.5 Cycles, Edge-Cuts, and Spanning Trees .....	190
4.6 Graphs and Vector Spaces .....	197
4.7 Matroids and the Greedy Algorithm .....	207
4.8 Supplementary Exercises .....	212
Glossary .....	213

<b>5. CONNECTIVITY</b>	<b>217</b>
5.1 Vertex- and Edge-Connectivity .....	218
5.2 Constructing Reliable Networks .....	223
5.3 Max-Min Duality and Menger's Theorems .....	231
5.4 Block Decompositions .....	241
5.5 Supplementary Exercises .....	244
Glossary .....	245
<b>6. OPTIMAL GRAPH TRAVERSALS</b>	<b>247</b>
6.1 Eulerian Trails and Tours .....	248
6.2 DeBruijn Sequences and Postman Problems .....	252
6.3 Hamiltonian Paths and Cycles .....	267
6.4 Gray Codes and Traveling Salesman Problems .....	273
6.5 Supplementary Exercises .....	282
Glossary .....	283
<b>7. PLANARITY AND KURATOWSKI'S THEOREM</b>	<b>285</b>
7.1 Planar Drawings and Some Basic Surfaces .....	286
7.2 Subdivision and Homeomorphism .....	292
7.3 Extending Planar Drawings .....	297
7.4 Kuratowski's Theorem .....	304
7.5 Algebraic Tests for Planarity .....	311
7.6 Planarity Algorithm .....	324
7.7 Crossing Numbers and Thickness .....	327
7.8 Supplementary Exercises .....	331
Glossary .....	334
<b>8. DRAWING GRAPHS AND MAPS</b>	<b>337</b>
8.1 The Topology of Low Dimensions .....	338
8.2 Higher-Order Surfaces .....	341
8.3 Mathematical Model for Drawing Graphs .....	346
8.4 Regular Maps on a Sphere .....	349
8.5 Imbeddings on Higher-Order Surfaces .....	354
8.6 Geometric Drawings of Graphs .....	361
8.7 Supplementary Exercises .....	365
Glossary .....	366
<b>9. GRAPH COLORINGS</b>	<b>371</b>
9.1 Vertex-Colorings .....	372
9.2 Map-Colorings .....	386
9.3 Edge-Colorings .....	393
9.4 Factorization .....	407
9.5 Supplementary Exercises .....	411
Glossary .....	413

<b>10. MEASUREMENT AND MAPPINGS</b>	<b>417</b>
10.1 Distance in Graphs .....	418
10.2 Domination in Graphs .....	424
10.3 Bandwidth .....	430
10.4 Intersection Graphs .....	435
10.5 Linear Graph Mappings .....	444
10.6 Modeling Network Emulation .....	453
10.7 Supplementary Exercises .....	462
Glossary .....	463
<b>11. ANALYTIC GRAPH THEORY</b>	<b>469</b>
11.1 Ramsey Graph Theory .....	470
11.2 Extremal Graph Theory .....	475
11.3 Random Graphs .....	480
11.4 Supplementary Exercises .....	489
Glossary .....	490
<b>12. SPECIAL DIGRAPH MODELS</b>	<b>493</b>
12.1 Directed Paths and Mutual Reachability .....	494
12.2 Digraphs as Models for Relations .....	505
12.3 Tournaments .....	511
12.4 Project Scheduling and Critical Paths .....	516
12.5 Finding the Strong Components of a Digraph .....	523
12.6 Supplementary Exercises .....	528
Glossary .....	529
<b>13. NETWORK FLOWS and APPLICATIONS</b>	<b>533</b>
13.1 Flows and Cuts in Networks .....	534
13.2 Solving the Maximum-Flow Problem .....	542
13.3 Flows and Connectivity .....	551
13.4 Matchings, Transversals, and Vertex Covers .....	560
13.5 Supplementary Exercises .....	573
Glossary .....	574
<b>14. GRAPHICAL ENUMERATION</b>	<b>577</b>
14.1 Automorphisms of Simple Graphs .....	578
14.2 Graph Colorings and Symmetry .....	583
14.3 Burnside's Lemma .....	589
14.4 Cycle-Index Polynomial of a Permutation Group .....	595
14.5 More Counting, Including Simple Graphs .....	600
14.6 Pólya-Burnside Enumeration .....	606
14.7 Supplementary Exercises .....	609
Glossary .....	610

<b>15. ALGEBRAIC SPECIFICATION of GRAPHS</b>	<b>613</b>
15.1 Cyclic Voltages .....	614
15.2 Cayley Graphs and Regular Voltages.....	623
15.3 Permutation Voltages.....	632
15.4 Symmetric Graphs and Parallel Architectures .....	637
15.5 Interconnection-Network Performance.....	644
15.6 Supplementary Exercises.....	646
Glossary.....	649
<b>16. NONPLANAR LAYOUTS</b>	<b>651</b>
16.1 Representing Imbeddings by Rotations.....	652
16.2 Genus Distribution of a Graph .....	659
16.3 Voltage-Graph Specification of Graph Layouts .....	664
16.4 Non-KVL Imbedded Voltage Graphs .....	670
16.5 Heawood Map-Coloring Problem.....	672
16.6 Supplementary Exercises.....	677
Glossary.....	678
<b>APPENDIX</b>	<b>681</b>
A.1 Logic Fundamentals.....	681
A.2 Relations and Functions.....	683
A.3 Some Basic Combinatorics .....	686
A.4 Algebraic Structures .....	687
A.5 Algorithmic Complexity .....	692
A.6 Supplementary Reading .....	694
<b>BIBLIOGRAPHY</b>	<b>695</b>
B.1 General Reading .....	695
B.2 References .....	697
<b>SOLUTIONS and HINTS</b>	<b>709</b>
<b>INDEXES</b>	<b>757</b>
I.1 Index of Applications .....	757
I.2 Index of Algorithms .....	759
I.3 Index of Notations .....	761
I.4 General Index .....	767

# Chapter 1

---

## INTRODUCTION TO GRAPH MODELS

- 1.1 Graphs and Digraphs**
  - 1.2 Common Families of Graphs**
  - 1.3 Graph Modeling Applications**
  - 1.4 Walks and Distance**
  - 1.5 Paths, Cycles, and Trees**
  - 1.6 Vertex and Edge Attributes: More Applications**
- 

### 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.

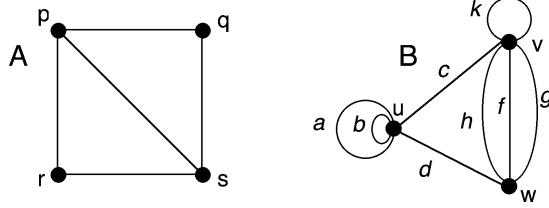
Formally, such configurations are modeled by combinatorial structures called *graphs*, consisting of two sets called *vertices* and *edges* and an incidence relation between them. The vertices and edges may have additional attributes, such as color or weight, or anything else useful to a particular model. 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 two or more vertices 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.<sup>†</sup>

**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 general; 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.

---

<sup>†</sup> We use the term “self-loop” instead of the more commonly used term “loop” because loop means something else in many applications.

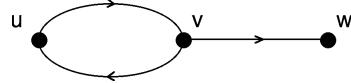
DEFINITION: A **multi-arc** is a set of two or more arcs having the same tail and same head. The **arc multiplicity** is the number of arcs within the multi-arc.

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

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  $vw$ .

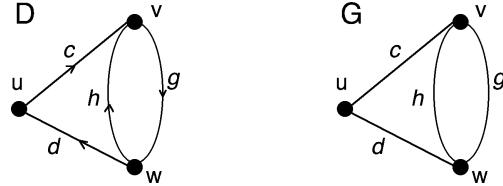


**Figure 1.1.2** A simple digraph with a pair of oppositely directed arcs.

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.

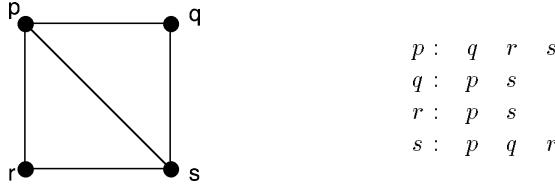
### Formal Specification of Graphs and Digraphs

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.

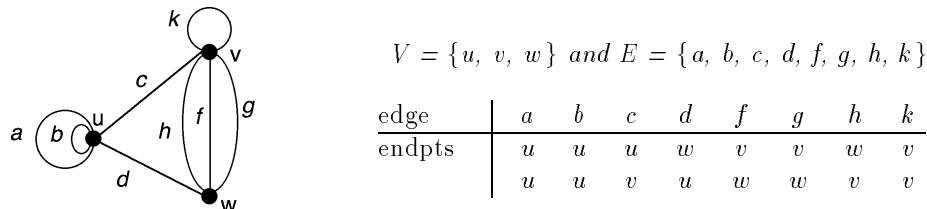
**Example 1.1.4:** Figure 1.1.4 shows a line drawing and a formal specification for a simple graph.



**Figure 1.1.4** A simple graph and its formal specification.

**DEFINITION:** A **formal specification of a general graph**  $G = (V, E, \text{endpts})$  consists of a list of its vertices, a list of its edges, and a two-row **incidence table** (specifying the  $\text{endpts}$  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.

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



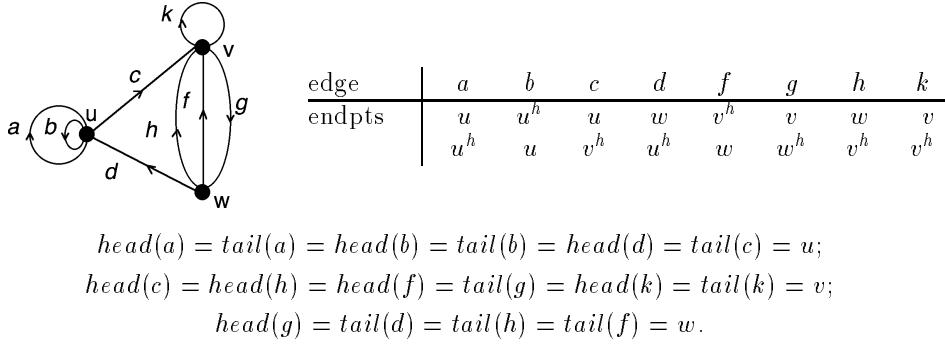
**Figure 1.1.5** A general graph and its formal specification.

**DEFINITION:** A **formal specification of a general digraph or a mixed graph**  $D = (V, E, \text{endpts}, \text{head}, \text{tail})$  is obtained from the formal specification of the underlying graph by adding the functions  $\text{head} : E_G \rightarrow V_G$  and  $\text{tail} : E_G \rightarrow V_G$ , 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  $\text{head}$  and  $\text{tail}$  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  $\text{head}$  and  $\text{tail}$ . A superscript “h” is used to indicate the **head** vertex.



**Figure 1.1.6** A general digraph and its formal specification.

**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  $\text{endpts}(e)$ , 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 below), 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.

Also, our formal incidence specification permits us to reverse the direction of an edge  $e$  at any time, just by reversing the values of  $\text{head}(e)$  and  $\text{tail}(e)$ . 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.

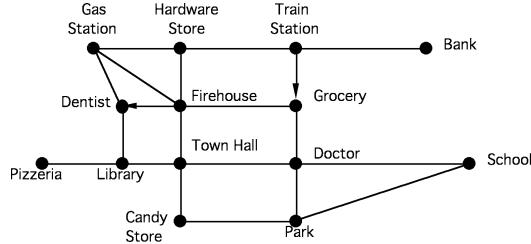
The ordered-pair representation could also prove awkward in implementing algorithms for which the graphs or digraphs are *dynamic* structures (i.e., they change during the algorithm). Whenever the direction on a particular edge must be reversed, the associated ordered pair has to be deleted and replaced 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

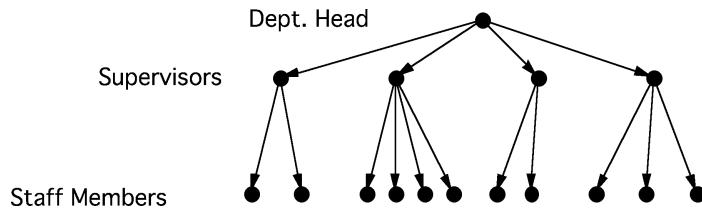
To bring the power of mathematics to bear on real-world problems, one must first *model* the problem mathematically. Graphs are remarkably versatile tools for modeling, and their wide-ranging versatility is a central focus throughout the text.

**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 of decision-making 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 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.<sup>†</sup>

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

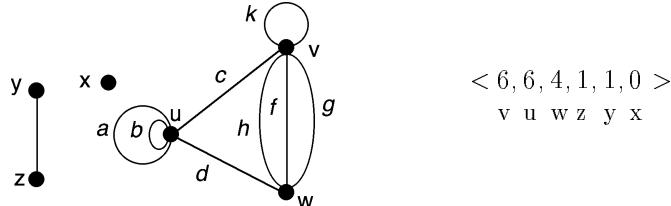
NOTATION: The smallest and largest degrees in a graph  $G$  are denoted  $\delta_{\min}$  and  $\delta_{\max}$  (or  $\delta_{\min}(G)$  and  $\delta_{\max}(G)$  when there is more than one graph under discussion). Some authors use  $\delta$  instead of  $\delta_{\min}$  and  $\Delta$  instead of  $\delta_{\max}$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

---

<sup>†</sup> Applications of graph theory to physical chemistry motivate the use of the term *valence*.

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

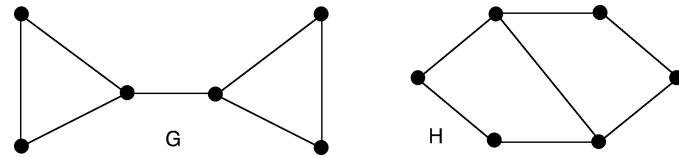
**Example 1.1.9:** Figure 1.1.9 shows a graph and its degree sequence.



**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  $\langle 3, 3, 2, 2, 2 \rangle$ .

The following theorem 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  $0, \dots, n - 1$ . 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.  $\diamond$

The work of Leonhard Euler (1707–1783) is regarded as the beginning of graph theory as a mathematical discipline. The following result 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.  $\diamond$

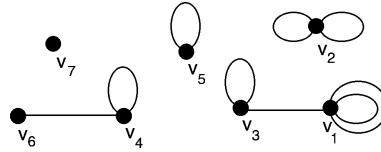
**Corollary 1.1.3.** *In a graph, there is an even number of vertices having odd degree.*

**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.  $\diamond$

**Corollary 1.1.4.** *The degree sequence of a graph is a finite, non-increasing sequence of nonnegative integers whose sum is even.*  $\diamond$

Conversely, any non-increasing, nonnegative sequence of integers whose sum is even is the degree sequence of some graph. Theorem 1.1.5 prescribes how to construct such a graph. The following preliminary example illustrates the construction.

**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  $v_1, v_2, \dots, v_7$ . 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 1.1.5.** *Suppose that  $\langle d_1, d_2, \dots, d_n \rangle$  is a sequence of nonnegative integers whose sum is even. Then there exists a graph with vertices  $v_1, v_2, \dots, v_n$  such that  $\deg(v_i) = d_i$ , for  $i = 1, \dots, n$ .*

**Proof:** Start with  $n$  isolated vertices  $v_1, v_2, \dots, v_n$ . For each  $i$ , if  $d_i$  is even, draw  $\frac{d_i}{2}$  self-loops on vertex  $v_i$ , and if  $d_i$  is odd, draw  $\frac{d_i-1}{2}$  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.  $\diamond$

### Graphic Sequences

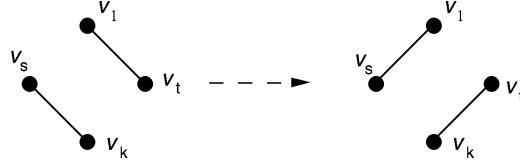
The construction in Theorem 1.1.5 is straightforward but hinges on allowing the graph to be non-simple. A more interesting problem is determining when a sequence is the degree sequence of a *simple* graph.

**DEFINITION:** A sequence  $\langle d_1, d_2, \dots, d_n \rangle$  is said to be **graphic** if there is a permutation of it that is the degree sequence of some simple graph. Such a simple graph is said to **realize** the sequence.

**Theorem 1.1.6.** *Let  $\langle d_1, d_2, \dots, d_n \rangle$  be a graphic sequence, with  $d_1 \geq d_2 \geq \dots \geq d_n$ . Then there is a simple graph with vertex-set  $\{v_1, \dots, v_n\}$  satisfying  $\deg(v_i) = d_i$  for  $i = 1, 2, \dots, n$ , such that  $v_1$  is adjacent to vertices  $v_2, \dots, v_{d_1+1}$ .*

**Proof:** Among all simple graphs with vertex-set  $\{v_1, v_2, \dots, v_n\}$  and  $\deg(v_i) = d_i$ ,  $i = 1, 2, \dots, n$ , let  $G$  be one for which  $r = |N_G(v_1) \cap \{v_2, \dots, v_{d_1+1}\}|$  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  $v_s \in N_{\tilde{G}}(v_1) \cap \{v_2, \dots, v_{d_1+1}\}$ . Thus,  $|N_{\tilde{G}}(v_1) \cap \{v_2, \dots, v_{d_1+1}\}| = r + 1$ , which contradicts the choice of  $G$  and completes the proof.  $\diamondsuit$



**Figure 1.1.12** Switching adjacencies while preserving all degrees.

**Corollary 1.1.7 [Havel (1955) and Hakimi (1961)].** A sequence  $\langle d_1, d_2, \dots, d_n \rangle$  of nonnegative integers such that  $d_1 \geq d_2 \geq \dots \geq d_n$  is graphic if and only if the sequence  $\langle d_2 - 1, \dots, d_{d_1+1} - 1, d_{d_1+2}, \dots, d_n \rangle$  is graphic.  $\diamondsuit$  (Exercises)

**Remark:** Corollary 1.1.7 yields a recursive algorithm that decides whether a non-increasing sequence is graphic.

**Algorithm 1.1.1: Recursive GraphicSequence( $\langle d_1, d_2, \dots, d_n \rangle$ )**

*Input:* a non-increasing sequence  $\langle d_1, d_2, \dots, d_n \rangle$ .

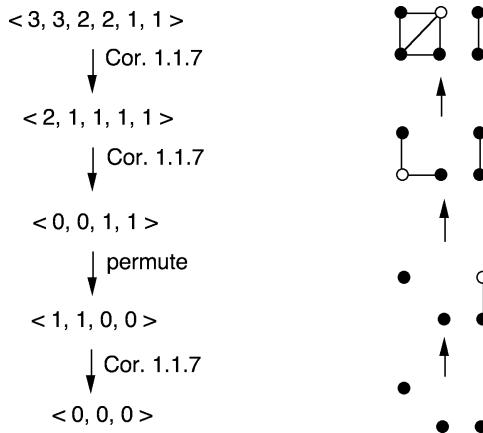
*Output:* TRUE if the sequence is graphic; FALSE if it is not.

```

If  $d_1 = 0$ 
    Return TRUE
Else
    If  $d_n < 0$ 
        Return FALSE
    Else
        Let  $\langle a_1, a_2, \dots, a_{n-1} \rangle$  be a non-increasing permutation
            of  $\langle d_2 - 1, \dots, d_{d_1+1} - 1, d_{d_1+2}, \dots, d_n \rangle$ .
        Return GraphicSequence( $\langle a_1, a_2, \dots, a_{n-1} \rangle$ )
    
```

**Remark:** An iterative version of the algorithm GraphicSequence based on repeated application of Corollary 1.1.7 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.

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

**Proof:** Each directed edge  $e$  contributes one to the indegree at  $\text{head}(e)$  and one to the outdegree at  $\text{tail}(e)$ .  $\diamond$

**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<sup>s</sup>**

$$V = \{u, w, x, z\}; E = \{e, f, g\}$$

$$\text{endpts}(e) = \{w\}; \text{endpts}(f) = \{x, w\}; \text{endpts}(g) = \{x, z\}$$

1.1.2

$$V = \{u, v, x, y, z\}; E = \{a, b, c, d\}$$

$$\text{endpts}(a) = \{u, v\}; \text{endpts}(b) = \{x, v\}; \text{endpts}(c) = \{u, v\}; \text{endpts}(d) = \{x\}$$

1.1.3

$$V = \{u, v, x, y, z\}; E = \{e, f, g, h, k\}$$

$$\text{endpts}(e) = \text{endpts}(f) = \{u, v\}; \text{endpts}(g) = \{x, z\}; \text{endpts}(h) = \text{endpts}(k) = \{y\}$$

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<sup>s</sup>

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

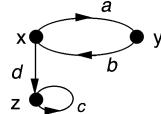
1.1.5

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

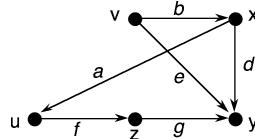
1.1.6

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

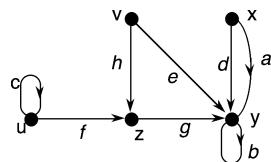
In Exercises 1.1.7 through 1.1.9, give a formal specification for the given digraph.

1.1.7<sup>s</sup>

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<sup>s</sup> 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 1.1.7 or Algorithm 1.1.1, why no such graph can exist.

1.1.15<sup>s</sup> 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.  $\langle 5, 5, 4, 4, 2, 1, 1 \rangle$

1.1.20 Use Theorem 1.1.6 to prove Corollary 1.1.7.

1.1.21 Write an iterative version of Algorithm 1.1.1 that applies Corollary 1.1.7 repeatedly until a sequence of all zeros or a sequence with a negative term results.

1.1.22<sup>s</sup> 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<sup>s</sup> 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  $\langle a_1, a_2, \dots, a_n \rangle$  and  $\langle b_1, b_2, \dots, b_n \rangle$  is called **digraphic** if there exists a simple digraph with vertex-set  $\{v_1, v_2, \dots, v_n\}$  such that  $\text{outdegree}(v_i) = a_i$  and  $\text{indegree}(v_i) = b_i$  for  $i = 1, 2, \dots, n$ .

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 1.1.7 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  $\binom{7}{2}5^62^{10}$  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<sup>s</sup> 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?

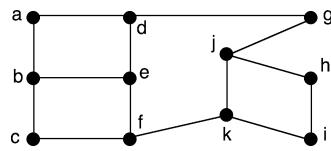
**1.1.33** Prove or disprove: There exists a simple graph with 13 vertices, 31 edges, three 1-valent vertices, and seven 4-valent vertices.

**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 §13.4.)

**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<sup>s</sup>** 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 and §9.1.)

**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<sup>s</sup>** 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 §9.3 and §13.4.)

**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<sup>s</sup>** 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,  $\forall v \in V, \exists w \in W, v \in N[w]$ . (See §10.2.)

**1.1.43** Find upper and lower bounds for the size of a minimum (smallest) dominating set of 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.

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

1.1.45<sup>s</sup> Find a minimum dominating set for the graph of Exercise 1.1.36.

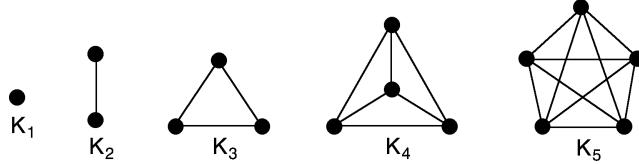
## 1.2 COMMON FAMILIES OF GRAPHS

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.

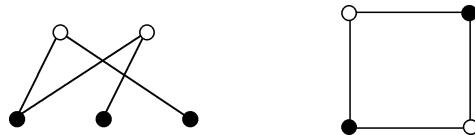


**Figure 1.2.1** The first five complete graphs.

### 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.

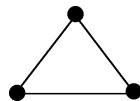


**Figure 1.2.2** Two bipartite graphs.

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

**Proof:** This is an immediate consequence of the definition. ◊

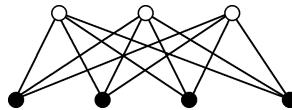
**Example 1.2.3:** The smallest possible simple graph that is not bipartite is the complete graph  $K_3$ , shown in Figure 1.2.3.



**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  $K_{m,n}$ .<sup>†</sup>

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

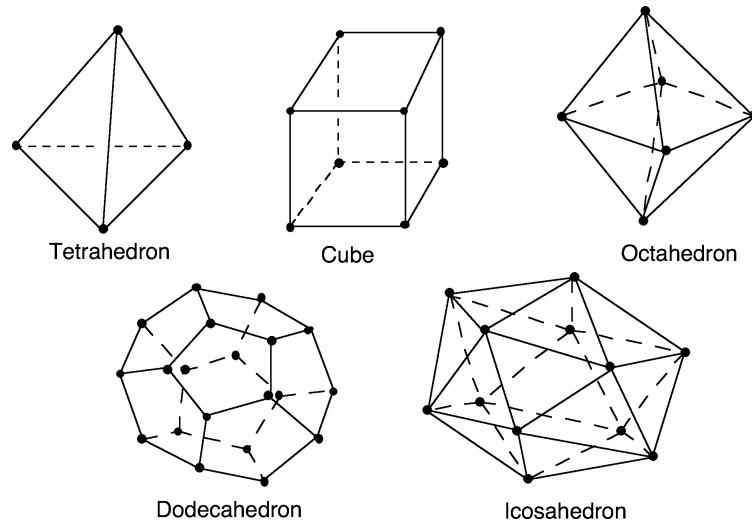


**Figure 1.2.4** The complete bipartite graph  $K_{3,4}$ .

### 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**.

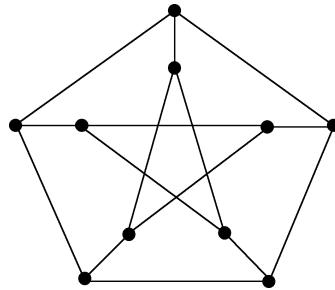


**Figure 1.2.5** The five platonic graphs.

---

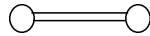
<sup>†</sup> The sense in which  $K_{m,n}$  is a unique object is described in §2.1.

**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.



**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.

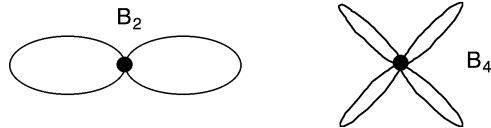


**Figure 1.2.7** A 2-regular graph representing the oxygen molecule  $O_2$ .

### Bouquets and Dipoles

One-vertex and two-vertex (non-simple) graphs often serve as building blocks for various interconnection networks, including certain parallel architectures (see Chapter 15).

**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  $D_4$ .

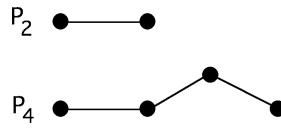


**Figure 1.2.9** Dipoles  $D_3$  and  $D_4$ .

### 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  $P_n$ .

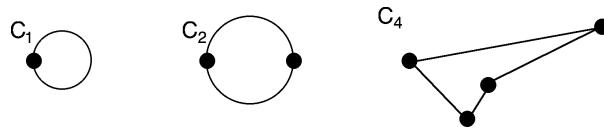
**Example 1.2.7:** Path graphs  $P_2$  and  $P_4$  are shown in Figure 1.2.10.



**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.

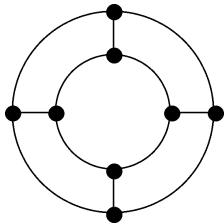
### Hypercubes and Circular Ladders

**DEFINITION:** The **hypercube graph**  $Q_n$  is the  $n$ -regular graph whose vertex-set is the set of bitstrings of length  $n$ , and 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**  $CL_n$  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.



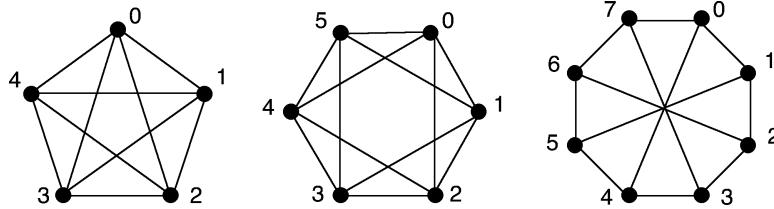
**Figure 1.2.12** Circular ladder graph  $CL_4$ .

### Circulant Graphs

**DEFINITION:** To the group of integers  $\mathbb{Z}_n = \{0, 1, \dots, n - 1\}$  under addition modulo  $n$  and a set  $S \subseteq \{1, \dots, n - 1\}$ , we associate the **circulant graph**  $\text{circ}(n : S)$  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  $i + s = j \pmod{n}$  or  $j + s = i \pmod{n}$ . In this regard, the elements of the set  $S$  are called **connections**.

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

**Example 1.2.11:** Figure 1.2.13 shows three circulant graphs.



**Figure 1.2.13** The circulant graphs  $\text{circ}(5 : 1, 2)$ ,  $\text{circ}(6 : 1, 2)$ , and  $\text{circ}(8 : 1, 4)$ .

**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*. Cayley graphs and voltage graphs, with applications, appear in Chapter 15.

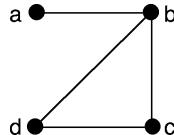
### Intersection and Interval Graphs

**DEFINITION:** A simple graph  $G$  with vertex-set  $V_G = \{v_1, v_2, \dots, v_n\}$  is an **intersection graph** if there exists a family of sets  $\mathcal{F} = \{S_1, S_2, \dots, S_n\}$  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:

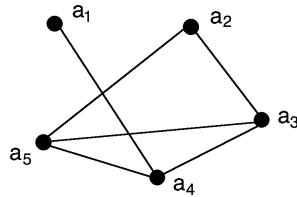
$$a \leftrightarrow (1, 3) \quad b \leftrightarrow (2, 6) \quad c \leftrightarrow (5, 8) \quad d \leftrightarrow (4, 7)$$



**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 1000 AD. In the graph shown below, the vertices correspond to the artifacts, and two vertices are adjacent if the corresponding artifacts appeared in the same grave. 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.

**Remark:** Intersection graphs have been generalized to *tolerance graphs*, which are discussed in §10.4.

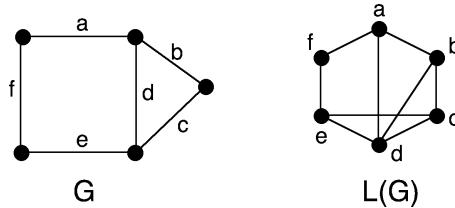
### Line Graphs

*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<sup>s</sup> 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?

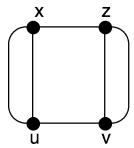
1.2.3 Draw the smallest possible non-bipartite graph.

1.2.4<sup>s</sup> Determine the values of  $n$  for which the given graph is bipartite.

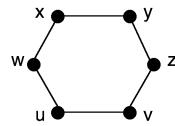
- a.  $K_n$
- b.  $C_n$
- c.  $P_n$

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

In Exercises 1.2.6 and 1.2.7, 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<sup>s</sup>

1.2.7



1.2.8 Label the vertices of the cube graph in Figure 1.2.5 with 3-digit binary strings so that the labels on adjacent vertices differ in exactly one digit.

1.2.9 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?

1.2.10<sup>s</sup> 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.

1.2.11 a. Draw all the 3-vertex tournaments whose vertices are  $u, v, x$ .

b. Determine the number of 4-vertex tournaments whose vertices are  $u, v, x, y$ .

1.2.12 Prove that every tournament has at most one vertex of indegree 0 and at most one vertex of outdegree 0.

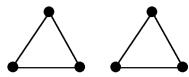
1.2.13<sup>s</sup> Suppose that  $n$  vertices  $v_1, v_2, \dots, v_n$  are drawn in the plane. How many different  $n$ -vertex tournaments can be drawn on those vertices?

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

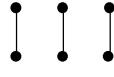
1.2.15 Find all the 4-vertex circulant graphs.

1.2.16 Show that each of the following graphs is a circulant graph.

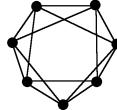
a.



b.



c.



1.2.17 State a necessary and sufficient condition on the positive integers  $n$  and  $k$  for  $\text{circ}(n : k)$  to be the cycle graph  $C_n$ .

1.2.18 Find necessary and sufficient conditions on the positive integers  $n$  and  $k$  for  $\text{circ}(n : k)$  to be the graph consisting of  $n/2$  mutually non-adjacent edges.

1.2.19 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.  $CL_n$

1.2.20 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.  $CL_n$

**1.2.21** 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.  $CL_n$

**1.2.22** 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.  $CL_n$

**1.2.23<sup>s</sup>** Show that the complete graph  $K_n$  is an interval graph for all  $n \geq 1$ .

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

**1.2.25<sup>s</sup>** 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  $(-1000, 1000)$ .

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

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

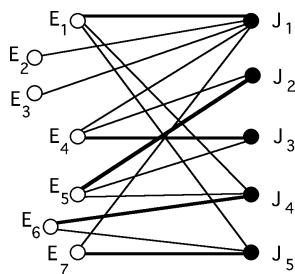
**1.2.28** Prove that every simple graph is an intersection graph by describing how to construct a suitable family of sets.

## 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.

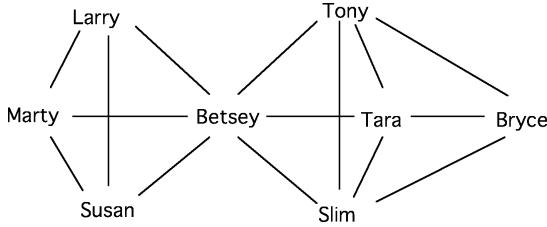
### Models That Use Simple Graphs

**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 13 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.

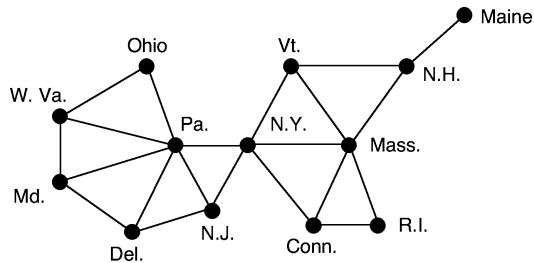
**Application 1.3.2 Sociological-Acquaintance Networks:** In an **acquaintance network**, the vertices represent persons, such as the students in a college class. An edge joining two vertices indicates that the corresponding pair of persons knew each other when the course began. The simple graph in Figure 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.

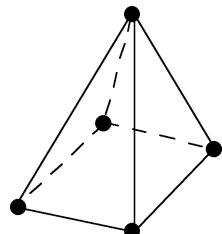
By Proposition 1.1.1, every group of two or more persons must contain at least two who know the same number of persons in the group. The acquaintance network of Figure 1.3.2 has degree sequence  $\langle 3, 3, 3, 3, 4, 4, 4, 6 \rangle$ .

**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.



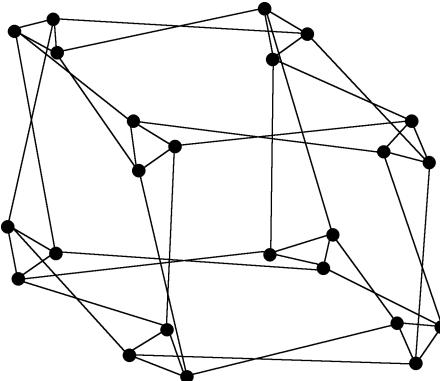
**Figure 1.3.3** Geographic adjacency of the northeastern states.

**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-skeletoins 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*. Chapter 15 offers a glimpse of several parallel architectures, including the wrapped butterfly. Their specification illustrates some of the beautiful interplay between graph theory and abstract algebra.

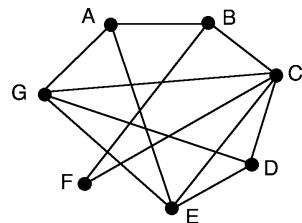


**Figure 1.3.5** A wrapped-butterfly-interconnection-network model.

**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,  $A, B, \dots, G$ , 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



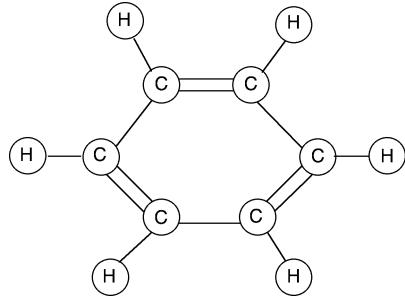
**Figure 1.3.6** A simple graph for a radio-frequency-assignment problem.

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 9.

### 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.

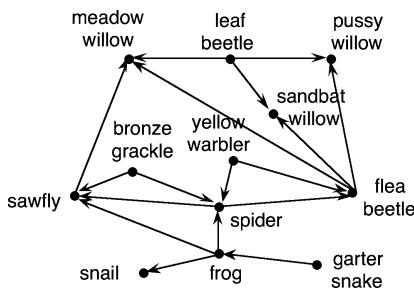


**Figure 1.3.7** The benzene molecule.

### 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 is 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.<sup>†</sup> 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.

---

<sup>†</sup> This illustration was adapted from [WiWa90], p.69.

**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 12.

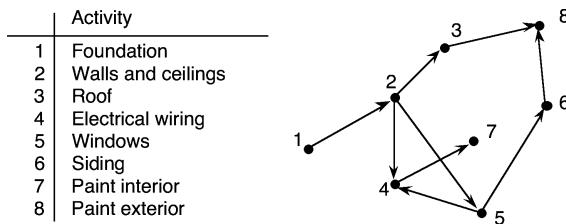


Figure 1.3.9 An activity digraph for building a house.

**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 12.

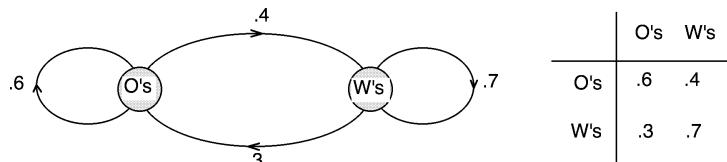


Figure 1.3.10 A Markov diagram and its transition matrix.

**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. Each arc label tells what kind of symbols causes 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.

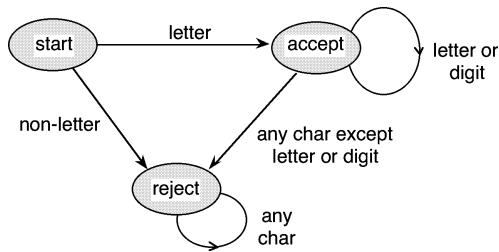
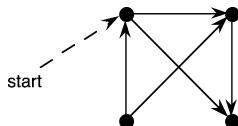


Figure 1.3.11 Finite-state recognizer for identifiers.

### EXERCISES for Section 1.3

1.3.1<sup>s</sup> 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.

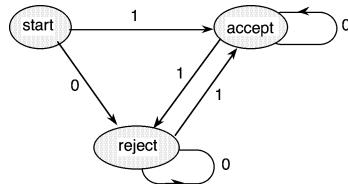
1.3.2 What is wrong with a computer program having the following abstract flow pattern?



1.3.3<sup>s</sup> 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<sup>s</sup> 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<sup>s</sup>** 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. 1) Draw a graph to model this situation. 2) 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<sup>s</sup>** 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

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

of vertices and edges, such that  $\text{endpts}(e_i) = \{v_{i-1}, v_i\}$ , for  $i = 1, \dots, n$ . 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.,  $\text{tail}(e_i) = v_{i-1}$  and  $\text{head}(e_i) = v_i$ .

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

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

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

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

**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.

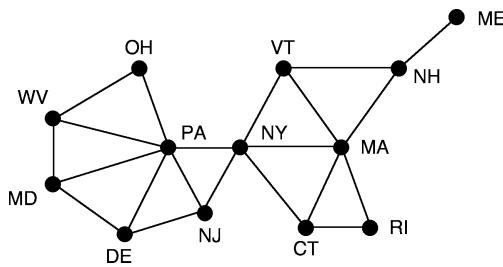
**DEFINITION:** A walk of length 0, i.e., with one vertex and no edges, is called a **trivial walk**.

**DEFINITION:** A **closed walk** (or **closed directed walk**) is a nontrivial 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 below, there is an open walk of length 6,

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

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.

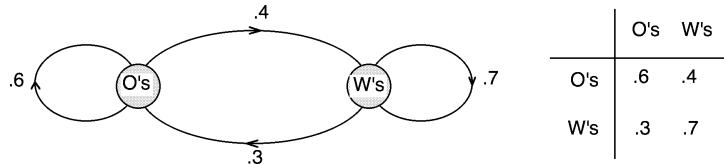


**Figure 1.4.1** Geographic adjacency of the northeastern states.

**Example 1.4.2:** In the Markov diagram below (from Application 1.3.12), 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

$\langle O, W, W, W, O \rangle$

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  $.4 \times .7 \times .7 \times .3 = 0.0588$ .

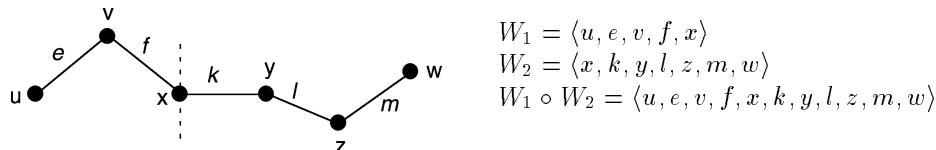


**Figure 1.4.2** Markov process from Application 1.3.12.

**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  $W_1 = \langle v_0, e_1, \dots, v_{k-1}, e_k, v_k \rangle$  and  $W_2 = \langle v_k, e_{k+1}, v_{k+1}, e_{k+2}, \dots, v_{n-1}, e_n, v_n \rangle$  such that walk  $W_2$  begins where walk  $W_1$  ends, is the walk  $W_1 \circ W_2 = \langle v_0, e_1, \dots, v_{k-1}, e_k, v_k, e_{k+1}, \dots, v_{n-1}, e_n, v_n \rangle$ .

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



**Figure 1.4.3** Concatenation of two walks.

**DEFINITION:** A **subwalk** of a walk  $W = \langle v_0, e_1, v_1, e_2, \dots, v_{n-1}, e_n, v_n \rangle$  is a subsequence of consecutive entries  $S = \langle v_j, e_{j+1}, v_{j+1}, \dots, e_k, v_k \rangle$  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 Figure 1.4.4, the closed directed walk  $\langle v, x, y, z, v \rangle$  is a subwalk of the open directed walk  $\langle u, v, x, y, z, v, w, t \rangle$ .

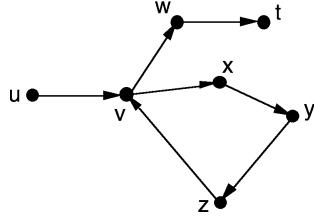


Figure 1.4.4

### 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**  $\bar{d}(s, t)$  is the length of a shortest directed walk from  $s$  to  $t$ .

**Example 1.4.6:** In Figure 1.4.5, 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.

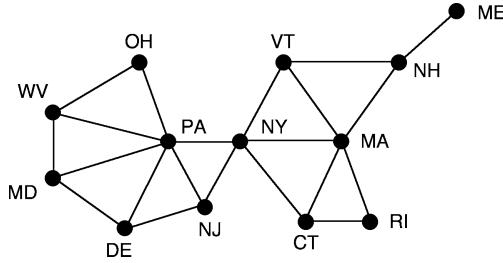
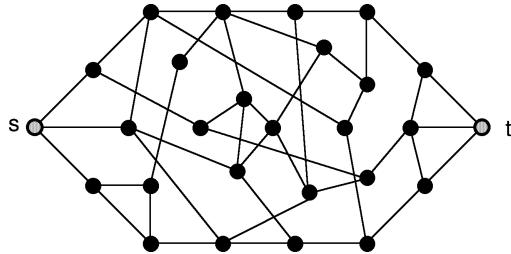


Figure 1.4.5 Geographic adjacency of the northeastern states.

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 1.4.6 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  $\text{diam}(G)$ , is the maximum of the vertex eccentricities in  $G$  or, equivalently, the maximum distance between two vertices in  $G$ . That is,

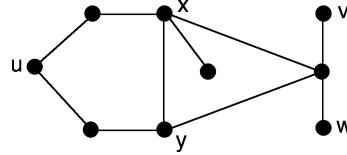
$$\text{diam}(G) = \max_{x \in V_G} \{ecc(x)\} = \max_{x,y \in V_G} \{d(x,y)\}$$

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

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

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

**Example 1.4.7:** The graph of Figure 1.4.7 below 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 1.4.7** 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.

**Remark:** Diameter, radius, and other distance-related concepts are discussed in §10.1.

### Connectedness

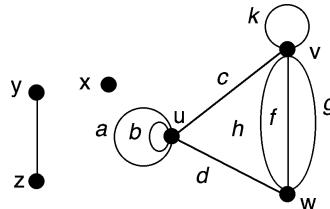
**DEFINITION:** Vertex  $v$  is **reachable from** vertex  $u$  if there is a walk from  $u$  to  $v$ .

**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 1.4.8 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.



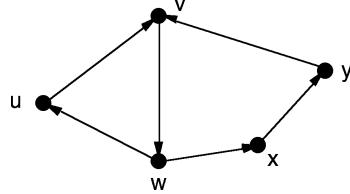
**Figure 1.4.8** A non-connected graph with three components.

### 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  $u\text{-}v$  walk and a directed  $v\text{-}u$  walk.

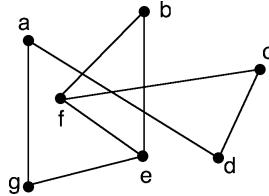
DEFINITION: A digraph  $D$  is **strongly connected** if every two of its vertices are mutually reachable.

**Example 1.4.10:** The digraph shown in Figure 1.4.9 is strongly connected.



**Figure 1.4.9** A strongly connected digraph.

**Example 1.4.11:** Suppose the graph in Figure 1.4.10 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 any one intersection to any other.



**Figure 1.4.10** A national park's system of roads.

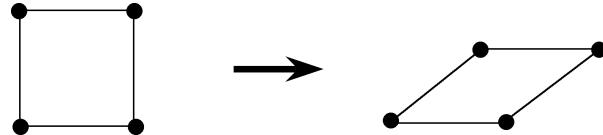
This problem can be expressed in graph-theoretic terms with the following definition.

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 12 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

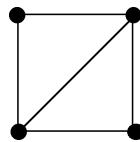
**Application 1.4.1 Rigid of Rectangular Frameworks<sup>†</sup>:** Consider a 2-dimensional framework of steel beams connected by joints that allow each beam to swivel in the plane. The framework is said to be **rigid** if none of its beams can swivel. The rectangle shown below is not rigid, since it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



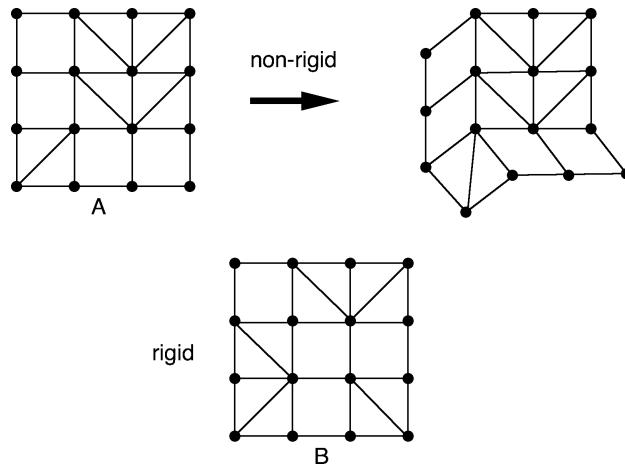

---

<sup>†</sup> No subsequent material depends on this application.

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



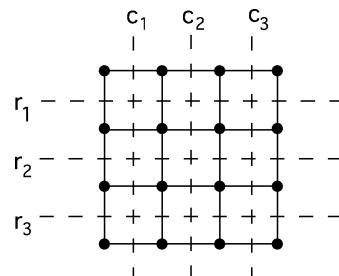
A rectangular framework with some diagonal braces might be rigid or non-rigid. For example, framework  $B$  in Figure 1.4.11 is rigid, but framework  $A$  is not.



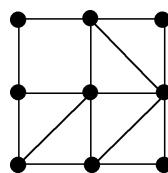
**Figure 1.4.11** Two rectangular frameworks.

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  $c_j$ ,  $1, 2 \dots$



**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.

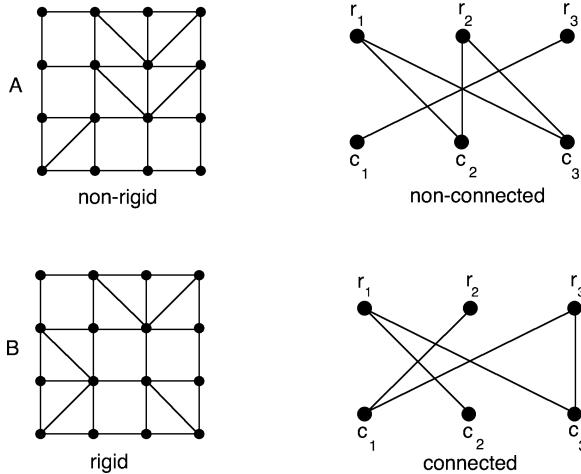
More generally, a framework is rigid if for each  $i$  and  $j$  there is a perpendicularity sequence that begins with  $r_i$  and ends with  $c_j$ . This observation leads to a simple method for determining whether a given rectangular framework is rigid.

Construct a bipartite graph by first drawing two sets of vertices,  $r_1, r_2, \dots$  and  $c_1, c_2, \dots$ , 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.*  $\diamond$

**Illustration:** Figure 1.4.12 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.

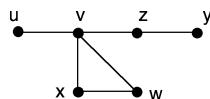


**Figure 1.4.12** Sample applications for Theorem 1.4.1 on rigid frameworks.

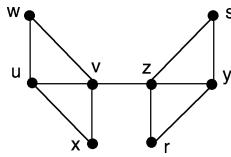
### EXERCISES for Section 1.4

1.4.1<sup>s</sup> Determine which of the following vertex sequences represent walks in the graph below.

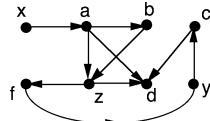
- a.  $\langle u, v \rangle$ ; b.  $\langle v \rangle$ ; c.  $\langle u, z, v \rangle$ ; d.  $\langle u, v, w, v \rangle$



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<sup>s</sup> Which of the following incidence tables represent connected graphs?

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

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

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

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<sup>s</sup>

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

1.4.7

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

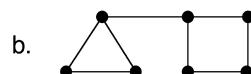
1.4.8

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

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<sup>s</sup> 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.



1.4.12<sup>s</sup> 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<sup>s</sup>** 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<sup>s</sup>** 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.  $\text{circ}(5 : 1, 2)$ ;      b.  $\text{circ}(6 : 1, 2)$ ;      c.  $\text{circ}(8 : 1, 2)$ .

**1.4.30** Describe the diameter and radius of the circulant graph  $\text{circ}(n : m)$  in terms of integer  $n$  and connection  $m$ .

**1.4.31** Describe the diameter and radius of the circulant graph  $\text{circ}(n : a, b)$  in terms of integer  $n$  and the connections  $a$  and  $b$ .

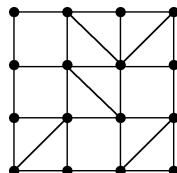
**1.4.32<sup>s</sup>** Let  $G$  be a connected graph. Prove that

$$\text{rad}(G) \leq \text{diam}(G) \leq 2 \cdot \text{rad}(G)$$

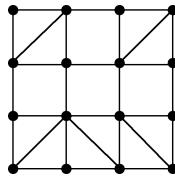
**1.4.33** Prove that a digraph must have a closed directed walk if each of its vertices has nonzero outdegree.

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

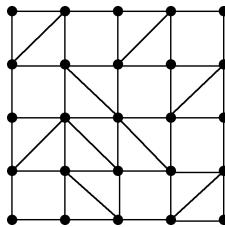
**1.4.34<sup>s</sup>**



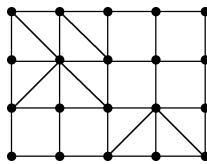
1.4.35



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<sup>s</sup> 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<sup>s</sup> 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,

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

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

1.4.46<sup>s</sup> Let  $G$  be an  $n$ -vertex simple graph such that  $\deg(v) \geq \frac{(n-1)}{2}$  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

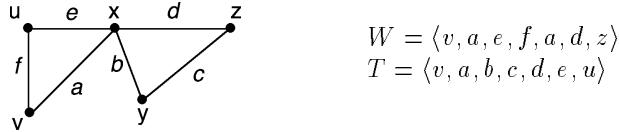
**DEFINITION:** A **trail** is a walk with no repeated edges.

**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

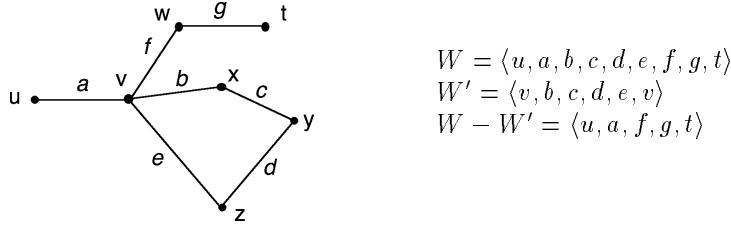
**DEFINITION:** Given a walk  $W = \langle v_0, e_1, \dots, v_{n-1}, e_n, v_n \rangle$  that contains a nontrivial, closed subwalk  $W' = \langle v_k, e_{k+1}, \dots, v_{m-1}, e_m, v_k \rangle$ , the **reduction of walk  $W$  by subwalk  $W'$** , denoted  $W - W'$ , is the walk

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

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  $w$ .



**Figure 1.5.2** A walk  $W$  reduced by a closed subwalk  $W'$ .

**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  $A = W_1, W_2, \dots, W_r = B$  such that for each  $i = 1, \dots, r - 1$ , 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  $W$ .  $\diamond$

**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.  $\diamond$

**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.  $\diamond$

## 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 called a **hamiltonian cycle**.<sup>†</sup>

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

---

<sup>†</sup> The term “hamiltonian” is in honor of the Irish mathematician William Rowan Hamilton.

**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.

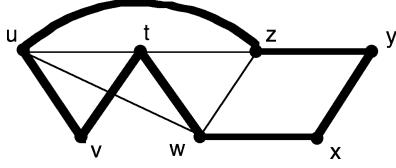


Figure 1.5.3 A hamiltonian graph.

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:

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

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  $x$  be the last vertex common to both paths (see Figure 1.5.4).

Since  $P_1$  and  $P_2$  are both shortest paths, their  $u \rightarrow x$  sections have equal length. Thus, the lengths of the  $x \rightarrow v$  section of  $P_1$  and the  $x \rightarrow w$  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  $G$ .  $\diamond$

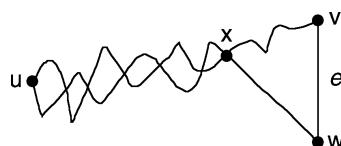


Figure 1.5.4 Figure for sufficiency part of Theorem 1.5.4 proof.

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.  $\diamond$

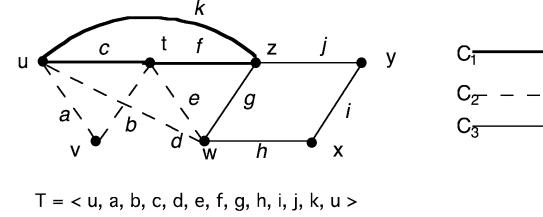
**Remark:** The assertion of Proposition 1.5.5 is no longer true if  $T$  is merely a closed walk (see Exercises).

**DEFINITION:** A collection of edge-disjoint cycles,  $C_1, C_2, \dots, C_m$ , 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  $T$ .  $\diamond$

**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.



**Figure 1.5.5** A closed trail  $T$  decomposed into three cycles.

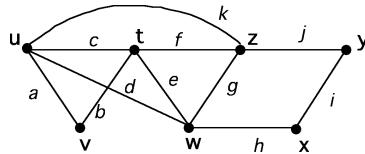
### Eulerian Trails

**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 graph that has an eulerian tour.<sup>†</sup>

**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 1.5.4, 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.

---

<sup>†</sup> The term “eulerian” is in honor of the Swiss mathematician Leonhard Euler.

**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.

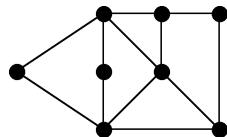


Figure 1.5.7 A graph with girth 3.

### Trees

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

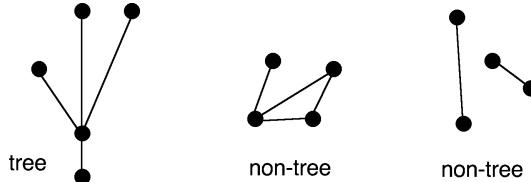
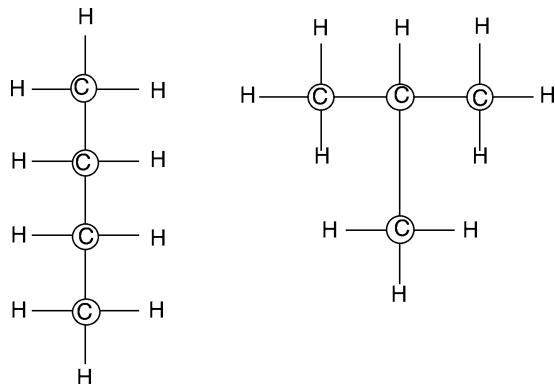


Figure 1.5.8 A tree and two non-trees.

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.

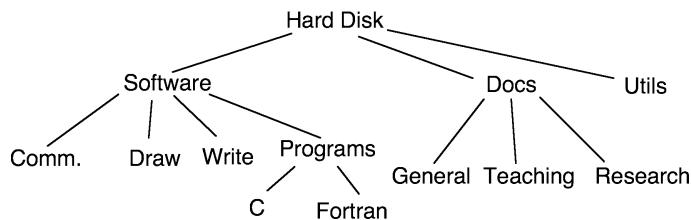
**Application 1.5.2** *Physical Chemistry:* A hydrocarbon is a chemical molecule composed of hydrogen and carbon atoms. It is said to be *saturated* if it includes the maximum number of hydrogen atoms for its number of carbon atoms. Saturation occurs when no two carbon atoms have a multiple bond, that 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*.



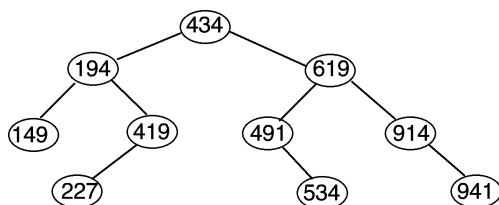
**Figure 1.5.9** Two saturated hydrocarbons: butane and isobutane.

**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.



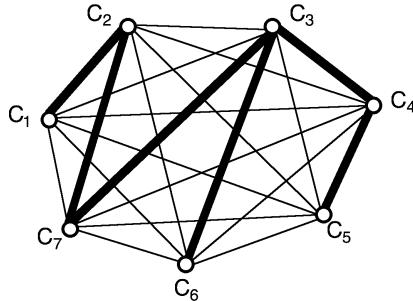
**Figure 1.5.10** A directory structure modeled by a rooted tree.

**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 minimal 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.

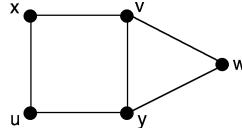


**Figure 1.5.12** Connecting seven computers using a minimal number of edges.

### EXERCISES for Section 1.5

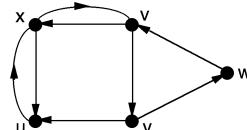
1.5.1<sup>s</sup> Which of the following vertex sequences represent a walk in the graph shown below? Which walks are paths? Which walks are cycles? What are the lengths of those that are walks?

- 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.  $\langle w, v, x, u, y, w \rangle$



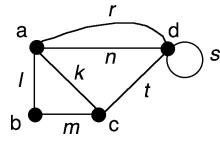
1.5.2 Which of the following vertex sequences represent a directed walk in the graph shown below? Which directed walks are directed paths? Which directed walks are directed cycles? What are the lengths of those that are directed walks?

- 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.  $\langle x, v, y, w, v, u, x \rangle$



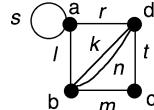
1.5.3<sup>s</sup> In the graph shown below, find

- 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.

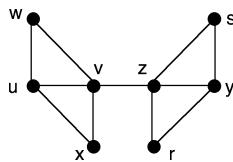


1.5.4 In the graph shown below, find

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



- 1.5.5<sup>s</sup> Determine the number of paths from  $w$  to  $r$  in the following graph.

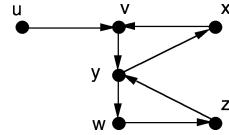


- 1.5.6 Draw a copy of the Petersen graph (from §1.2) with vertex labels. Then find

- 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<sup>s</sup> 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<sup>s</sup> 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, of 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<sup>s</sup> 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.  $\text{circ}(5 : 1, 2)$ ;
- b.  $\text{circ}(6 : 1, 2)$ ;
- c.  $\text{circ}(8 : 1, 2)$ .

- 1.5.24 Describe the girth of the circulant graph  $\text{circ}(n : m)$  in terms of integer  $n$  and connection  $m$ .

- 1.5.25 Describe the girth of the circulant graph  $\text{circ}(n : a, b)$  in terms of integer  $n$  and the connections  $a$  and  $b$ .

1.5.26 Find necessary and sufficient conditions for the circulant graph  $\text{circ}(n : a, b)$  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<sup>s</sup> 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 closed walk of odd length contains a cycle.

1.5.32<sup>s</sup> 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  $k$ .

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

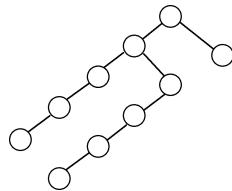
1.5.34 State and prove the digraph version of Theorem 1.5.2.

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.

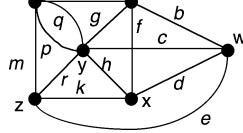
1.5.37 Find a collection of cycles whose edge-sets partition the edge-set of the closed trail in Example 1.5.4 but that is not a decomposition of the closed trail (i.e., at least one of the cycles in the collection is neither a subwalk nor a reduced walk of the closed trail).

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.



- 1.5.39 Find an eulerian tour in the following graph.

- u a v



- 1.5.40<sup>s</sup> 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<sup>s</sup> 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

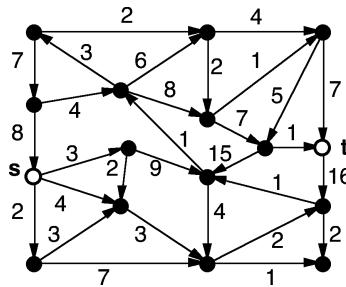
The all-purpose model of a graph includes attribute lists for edges and vertices. In viewing digraphs as a species of graphs, direction is an edge attribute. The Markov-diagram and lexical-scanner applications, introduced in §1.3, gave a first glimpse of graph models that require edge labels, another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

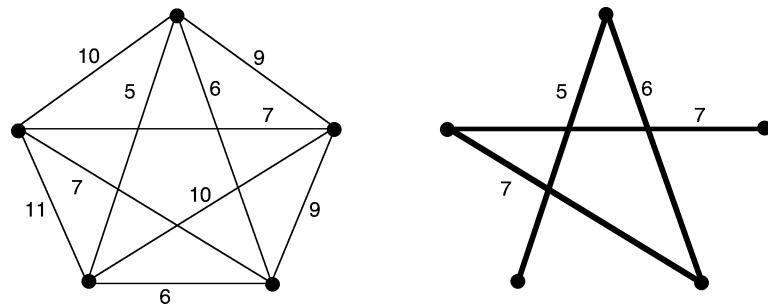
Edge-weights are among the most commonly used attributes of graphs, especially in combinatorial optimization. For instance, the edge-weight might represent transportation cost, travel time, spatial distance, power loss, upper bounds on flow, or inputs and outputs for transitions in a finite state machine.

**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).



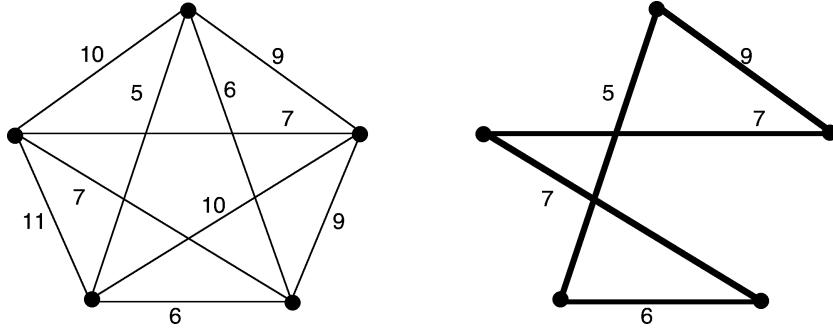
**Figure 1.6.1** A weighted directed graph for a shortest-path problem.

**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).



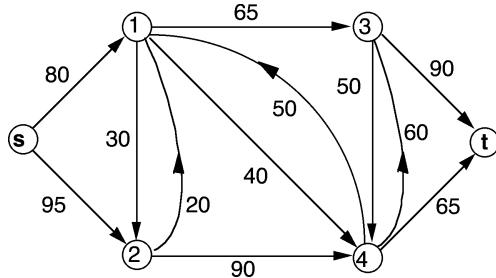
**Figure 1.6.2** A weighted graph and its minimum spanning tree.

**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.

**Application 1.6.4 Maximum-Flow Problem:** Suppose that water is being pumped through a network of pipelines, from a *source*  $s$  to a *sink*  $t$ . Figure 1.6.4 represents the network, and each edge-weight is the upper bound on the flow through the corresponding pipeline. Find the maximum flow from  $s$  to  $t$  (see Chapter 13).



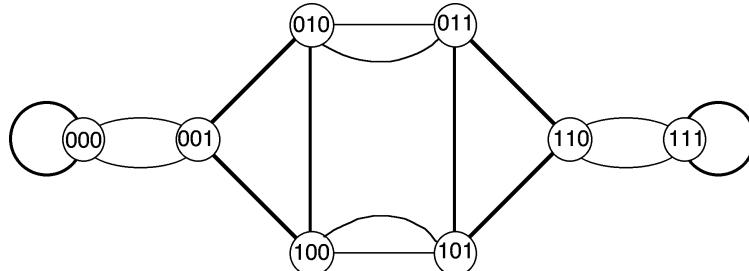
**Figure 1.6.4** A maximum-flow problem.

It is interesting to note that there are *polynomial-time* algorithms for three of the problems, but the traveling salesman problem is notoriously difficult, belonging to the class *NP-Hard*. It has motivated a number of different approaches, including cutting-plane methods in integer programming, neural nets, simulated annealing, and the design of various heuristics.

### Vertex-Weights and Labels

There are also many important problems requiring vertex attributes in the graph model. For example, a vertex-weight might represent production cost at an associated manufacturing site, and a vertex color might be a timeslot for a timetabling problem. Also, a vertex label might identify a state for a Markov process or a node in an interconnection network.

**Application 1.6.5 Parallel Architectures:** *Shuffle-exchange graphs* serve as models for parallel architectures suitable for the execution of various distributed algorithms, including *card-shuffling* and the *Fast Fourier Transform*. The vertices of the shuffle-exchange graph  $SE_n$  are the bitstrings of length  $n$ . These and other interconnection networks are presented in Chapter 15.



**Figure 1.6.5** The shuffle-exchange graph  $SE_3$ .

**Application 1.6.6 Software Testing and the Chinese Postman Problem:** During execution, a computer program's flow moves between various *states*, and the *transitions* from one state to another depend on the input. In testing software, one would like to generate input data that forces the program to test all possible transitions.

**DEFINITION:** A **directed postman tour** is a closed directed walk that uses each arc *at least* once.

**Digraph Model:** The program's execution flow is modeled as a digraph, where the states of the program are represented by vertices, the transitions are represented by arcs, and each of the arcs is assigned a label indicating the input that forces the corresponding transition. Then the problem of finding an input sequence for which the program invokes all transitions and minimizes the total number of transitions is equivalent to finding a directed postman tour of minimum length. This is a variation of the *Chinese Postman Problem*, which is discussed in §6.2.

**Remark:** Under certain reasonable assumptions, the flow digraph modeling a program's execution can be assumed to be strongly connected, which guarantees the existence of a postman tour.

### EXERCISES for Section 1.6

1.6.1 Formulate the personnel-assignment problem (Application 1.3.1) as a maximum-flow problem. (Hint: add an artificial source and sink to the bipartite graph.)

1.6.2<sup>s</sup> Let  $x$  be a vertex of a complete graph  $K_n$ . How many different Hamiltonian cycles are there that begin at  $x$ ?

*In Exercises 1.6.3 through 1.6.8, describe an appropriate graph or digraph model, including vertex and/or edge attributes, for the given situation.*

1.6.3<sup>s</sup> Suppose that  $n$  final exams are to be scheduled in  $k$  non-overlapping timeslots. The undesirability of assigning certain pairs to the same timeslot varies from pair to pair. How should the assignments be made?

**1.6.4** Suppose that a city owns ten snowplows. How should the snowplows be routed so that all the paved roads are plowed and the total distance that the snowplows travel is minimized?

**1.6.5<sup>s</sup>** There are  $n$  employees and  $n$  tasks. Each employee-task pair has an efficiency rating, based on past performance. What is the most efficient matching of employees to tasks?

**1.6.6** A manufacturer owns  $m$  warehouses,  $w_1, w_2, \dots, w_m$ , from which she can ship units of a product to  $n$  customers. Warehouse  $w_i$  has  $s_i$  units of the product, and the  $j$ th customer requires  $d_j$  units. The cost of shipping from  $w_i$  to customer  $j$  is  $c_{ij}$ . How should the product be distributed so that demand is met while the total shipping cost is minimized?

**1.6.7<sup>s</sup>** Suppose that an electric utility owns  $n$  power generators that serve a given region. The production cost for the  $i$ th generator is  $\$p_i$  per KWH, and its maximum output per day is  $M$  KWH. Each generator has a separate transmission line to each of  $m$  holding stations. Each holding station requires  $d_j$  KWH each day. The percentage loss of power during transmission across the line from the  $i$ th generator to the  $j$ th station is  $l_{ij}$ . How much power should each generator produce, and how should that power be distributed?

**1.6.8<sup>s</sup>** Suppose that a school owns  $m$  buses to transport  $n$  children to and from  $k$  bus stops. There are  $d_i$  children at the  $i$ th bus stop. The capacity of each bus is 50, and the distance from the  $i$ th stop to the  $j$ th stop is  $c_{ij}$ . How should the children be assigned to the buses, and how should the buses be routed through the stops, so that the total bus mileage is minimized?

## 1.7 SUPPLEMENTARY EXERCISES

**1.7.1** A 20-vertex graph has 62 edges. Every vertex has degree 3 or 7. How many vertices have degree 3?

**1.7.2** Either draw a 3-regular 7-vertex graph or prove that none exists.

**1.7.3** Prove that no 5-vertex, 7-edge simple graph has diameter greater than 2.

**1.7.4** Use the Havel-Hakimi algorithm to decide whether a simple graph can have the degree sequence 533322. If so, then construct the graph.

**1.7.5** Draw a connected simple graph  $G$  whose line graph  $L(G)$  has the degree sequence 1122233.

**1.7.6** Draw a simple connected graph  $H$  whose line graph  $L(H)$  has the degree sequence 1122233, such that  $H$  is *not* isomorphic to the graph  $G$  of Exercise 1.7.5.

**1.7.7** How many edges are in the icosahedral graph?

**1.7.8** How many edges are in the hypercube graph  $Q_4$ ?

**1.7.9** What is the diameter of the Petersen graph? Explain.

**1.7.10** In the circulant graph  $\text{circ}(24 : 1, 5)$ , what vertices are in the open neighborhood  $N(3)$ ?

1.7.11 In the circulant graph  $\text{circ}(24 : 1, 5)$ , what vertices are at distance 2 from vertex 3?

1.7.12 Is  $\text{circ}(24 : 4, 9)$  connected? What is a general condition for the connectedness of a circulant graph? How would you calculate the number of components?

1.7.13 What vertex of  $\text{circ}(13 : 1, 5)$  is most distant from 0?

1.7.14 Calculate the girth of  $\text{circ}(13 : 1, 5)$ .

1.7.15 Two opposite corners are removed from an 8-by-8 checkerboard. Prove that it is impossible to cover the remaining 62 squares with 31 dominoes, such that each domino covers two adjacent squares.

1.7.16 Prove that a graph  $G$  is bipartite if and only if it has the following property: every subgraph  $H$  contains an independent set of vertices whose cardinality is at least half the cardinality of  $V_H$ .

1.7.17 Prove that in a connected graph, any two longest paths must have a vertex in common.

1.7.18 Count the number of different eulerian tours in  $K_4$ .

1.7.19 Let  $G$  be a simple  $n$ -vertex graph with an independent set of vertices of cardinality  $c \leq n$ . What is the maximum number of edges that  $G$  may have?

DEF: The **edge-complement** of a simple graph  $G$  is the simple graph  $\overline{G}$  on the same vertex set such that two vertices of  $\overline{G}$  are adjacent if and only if they are *not* adjacent in  $G$ .

1.7.20 Let  $G$  be a simple bipartite graph with at least 5 vertices. Prove that  $\overline{G}$  is not bipartite. (See §2.4.)

1.7.21 Let  $G$  be a graph whose vertices are the bitstrings of length 4 such that two bitstrings are adjacent if they differ either in exactly one bit or in all four bits.

- Write the vertex sequence of a minimum length cycle in the edge-complement  $\overline{G}$ .
- What is the diameter of the graph  $\overline{G}$ ? Explain.

DEF: A simple graph  $G$  is **self-complementary** if  $G = \overline{G}$ .

1.7.22 Prove that if  $G$  is an  $n$ -vertex self-complementary graph, then either  $n \equiv 0 \pmod{4}$  or  $n \equiv 1 \pmod{4}$ .

1.7.23 Can a self-complementary graph be non-connected?

1.7.24 Suppose that a self-complementary graph has  $4k + 1$  vertices. Prove that it must have at least one vertex of degree  $2k$ .

DEF: An  $n$ -vertex,  $m$ -edge simple graph  $G$  is **graceful** if there is a one-to-one function  $g : V_G \rightarrow \{0, 1, \dots, m\}$  such that the function  $g' : E_G \rightarrow \{0, 1, \dots, m\}$  given by the rule  $uv \mapsto |g(u) - g(v)|$  is one-to-one.

1.7.25 Prove that the cycle  $C_5$  is not graceful.

1.7.26 Prove that the cycle  $C_8$  is graceful.

---

## GLOSSARY

**acyclic:** having no cycles.

**adjacency table:** a table with a row for each vertex, each containing the list of neighbors of that vertex.

**adjacent edges:** two edges that have an endpoint in common.

**adjacent vertices:** two vertices joined by an edge.

**arc:** an edge, one of whose endpoints is designated as the **tail**, and whose other endpoint is designated as the **head**; a synonym for *directed edge*.

**arc multiplicity:** the number of arcs within a multi-arc.

**bigraphical** pair of sequences of nonnegative integers  $\langle a_1, a_2, \dots, a_r \rangle$  and  $\langle b_1, b_2, \dots, b_t \rangle$ : there exists a bipartite graph  $G$  with bipartition subsets  $U = \{u_1, u_2, \dots, u_r\}$  and  $W = \{w_1, w_2, \dots, w_t\}$  such that  $\deg(u_i) = a_i$ ,  $i = 1, 2, \dots, r$ , and  $\deg(w_i) = b_i$ ,  $i = 1, 2, \dots, t$ .

**bipartite** graph: a graph whose vertex-set can be partitioned into two subsets (parts) such that every edge has one endpoint in one part and one endpoint in the other part.

**bipartition** of a bipartite graph  $G$ : the two subsets into which  $V_G$  is partitioned.

**bouquet:** a one-vertex graph with one or more self-loops.

**central vertex** of a graph  $G$ : a vertex  $v$  with minimum *eccentricity*, i.e.,  $\text{ecc}(v) = \text{rad}(G)$ .

**circulant graph**  $\text{circ}(n : S)$  with **connections** set  $S \subseteq \{1, \dots, n - 1\}$ : the graph whose vertex-set is the group of integers  $\mathbb{Z}_n = \{0, 1, \dots, n-1\}$  under addition modulo  $n$ , 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  $j + s = i \bmod n$ .

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

**clique:** a subgraph that is a complete graph.

**closed (directed) walk:** a (directed) walk whose initial and final vertices are identical.

**complete bipartite graph:** a simple bipartite graph such that each pair of vertices in different sides of the partition is joined by an edge.

**complete graph:** a simple graph such that every pair of vertices is joined by an edge.

**concatenation** of walks  $W_1$  and  $W_2$ : a walk whose traversal consists of a traversal of  $W_1$  followed by a traversal of  $W_2$  such that  $W_2$  begins where  $W_1$  ends.

**connected** graph: a graph in which every pair of distinct vertices has a walk between them.

**connected** digraph: a digraph whose underlying graph is connected.

**cycle:** a closed path with at least one edge.

**cycle graph:** a 1-vertex *bouquet* or a simple connected graph with  $n$  vertices and  $n$  edges that can be drawn so that all of its vertices and edges lie on a single circle.

**decomposition** of a closed trail  $T$ : a collection  $C_1, C_2, \dots, C_m$  of edge-disjoint cycles, such that 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  $T$ .

**degree** of a vertex: the number of proper edges incident on that vertex plus twice the number of self-loops.

**degree sequence**: a list of the degrees of all the vertices in ascending order.

**diameter** of a graph  $G$ , denoted  $\text{diam}(G)$ : given by  $\text{diam}(G) = \max_{x,y \in V_G} \{d(x,y)\}$ .

**digraph**: abbreviated name for *directed graph*.

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

**dipole**: a two-vertex graph with no self-loops.

**directed distance** from vertex  $u$  to  $v$ : the length of a shortest directed walk from  $u$  to  $v$ .

**directed edge**: a synonym for *arc*.

**directed graph**: a graph in which every edge is a directed edge.

**directed walk** from vertex  $u$  to vertex  $v$ : an alternating sequence of vertices and arcs representing a continuous traversal from  $u$  to  $v$ , where each arc is traversed from tail to head.

**direction** on an edge: an optional attribute that assigns the edge a one-way restriction or preference, said to be from *tail* to *head*; in a drawing the tail is the end behind the arrow and the head is the end in front.

**distance** between vertices  $u$  and  $v$ : the length of a shortest walk between  $u$  and  $v$ .

**dominating set** of vertices in a graph  $G$ : a vertex subset  $W$  such that every vertex of  $G$  is in  $W$  or is adjacent to at least one vertex in  $W$ .

**eccentricity** of a vertex  $v$  in a graph  $G$ : the distance from  $v$  to a vertex farthest from  $v$ ; denoted  $\text{ecc}(v)$ .

**edge**: a connection between one or two vertices of a graph.

**edge-complement** of a simple graph  $G$ : the simple graph  $\overline{G}$  on the same vertex set such that two vertices of  $\overline{G}$  are adjacent if and only if they are *not* adjacent in  $G$ .

**edge multiplicity**: the number of edges within a *multi-edge*.

**edge-weight**: a number assigned to an edge (optional attribute).

**endpoints** of an edge: the one or two vertices that are associated with that edge.

**eulerian graph**: a graph that has an eulerian tour.

**eulerian tour**: a closed eulerian trail.

**eulerian trail** in a graph: a trail that contains every edge of that graph.

**formal specification** of a graph or digraph: a triple  $(V, E, \text{endpts})$  comprising a vertex list, an edge list, and an incidence table.

**girth** of a graph  $G$ : the length of a shortest cycle in  $G$ .

**graceful graph**: an  $n$ -vertex,  $m$ -edge simple graph  $G$  for which there is a one-to-one function  $g : V_G \rightarrow \{0, 1, \dots, m\}$  such that the function  $g' : E_G \rightarrow \{0, 1, \dots, m\}$  given by the rule  $uv \mapsto |g(u) - g(v)|$  is one-to-one.

**graph**  $G = (V, E)$ : a mathematical structure consisting of two 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**.

**graphic sequence**: a sequence  $\langle d_1, d_2, \dots, d_n \rangle$  such that some permutation of it is the degree sequence of a simple graph.

**hamiltonian cycle** in a graph: a cycle that uses every vertex of that graph.

**head** of an arc: the endpoint to which that arc is directed.

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

**incidence**: the relationship between an edge and its endpoints.

**incidence table** of a graph or digraph: a table that specifies the endpoints of every edge and, for a digraph, which endpoint is the head.

**indegree** of a vertex  $v$ : the number of arcs directed to  $v$ .

**independent set** of vertices: a set of mutually non-adjacent vertices.

**intersection graph**: a graph whose vertices are associated with a collection of sets, such that two vertices are adjacent if and only if their corresponding sets overlap.

**interval graph**: an intersection graph whose vertex-set corresponds to a collection of intervals on the real line.

**length** of a walk: the number of edge-steps in the walk sequence.

**line graph**  $L(G)$  of a graph  $G$ : the graph that 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.

**matching** in a graph  $G$ : a set of mutually non-adjacent edges in  $G$ .

**mixed graph**: a graph that has undirected edges as well as directed edges.

**multi-arc** in a digraph: a set of two or more arcs such that each arc has the same head and the same tail.

**multi-edge** in a graph: a set of two or more edges such that each edge has the same set of endpoints.

**mutual reachability** of two vertices  $u$  and  $v$  in a digraph  $D$ : the existence of both a directed  $u$ - $v$  walk and a directed  $v$ - $u$  walk.

**node**: synonym for *vertex*.

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

**open (directed) walk**: a walk whose initial and final vertex are different.

**outdegree** of a vertex  $v$ : the number of arcs directed from  $v$ .

**partially directed graph**: synonym of *mixed graph*.

**path**: a walk with no repeated edges and no repeated vertices (except possibly the initial and final vertex); a trail with no repeated vertices.

**path graph**: a connected graph that can be drawn so that all of its vertices and edges lie on a single straight line.

**Petersen graph:** a special 3-regular, 10-vertex graph first described by Julius Petersen.

**platonic graph:** a vertex-and-edge configuration of a platonic solid.

**platonic solid:** any one of the five regular 3-dimensional polyhedrons – tetrahedron, cube, octahedron, dodecahedron, or icosahedron.

**proper edge:** an edge that is not a self-loop.

**radius** of a graph  $G$ , denoted  $\text{rad}(G)$ : the minimum of the vertex eccentricities.

**reachability** of a vertex  $v$  from a vertex  $u$ : the existence of a walk from  $u$  to  $v$ .

**reduction of a walk  $W$  by a walk  $W'$ :** the walk that results when all of the vertices and edges of  $W'$ , except for one of the vertices common to both walks, are deleted from  $W$ .

**regular graph:** a graph whose vertices all have equal degree.

**self-complementary graph:** a simple graph  $G$  such that  $G = \overline{G}$ .

**self-loop:** an edge whose endpoints are the same vertex.

**simple graph (digraph):** a graph (digraph) with no multi-edges or self-loops.

**1-skeleton** of a polyhedron: the graph consisting of the vertices and edges of that polyhedron, retaining their incidence relation from the polyhedron.

**strongly connected** digraph: a digraph in which every two vertices are mutually reachable.

**strong orientation** of a graph: an assignment of directions to the edges of a graph so that the resulting digraph is strongly connected.

**subwalk** of a walk  $W$ : a walk consisting of a subsequence of consecutive entries of the walk sequence of  $W$ .

**tail** of an arc: the endpoint from which that arc is directed.

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

**trail:** a walk with no edge that occurs more than once.

**tree:** a connected graph that has no cycles.

**trivial** walk, trail, path, or graph: a walk, trail, path, or graph consisting of one vertex and no edges.

**underlying graph** of a directed or mixed graph  $G$ : 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).

**valence:** a synonym for *degree*.

**vertex:** an element of the first constituent set of a graph; a synonym for *node*.

**vertex cover** of a graph  $G$ : a vertex subset  $W$  such that every edge of  $G$  is incident on at least one vertex in  $W$ .

**walk** from vertex  $u$  to vertex  $v$ : an alternating sequence of vertices and edges, representing a continuous traversal from vertex  $u$  to vertex  $v$ .

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

# Chapter 2

---

## STRUCTURE AND REPRESENTATION

- 2.1 Graph Isomorphism**
  - 2.2 Automorphisms and Symmetry**
  - 2.3 Subgraphs**
  - 2.4 Some Graph Operations**
  - 2.5 Tests for Non-Isomorphism**
  - 2.6 Matrix Representations**
  - 2.7 More Graph Operations**
- 

### INTRODUCTION

The possible representations of a graph include drawings, incidence tables, formal specifications, and many other concrete descriptions of the abstract structure. Structure is what characterizes a graph itself and is independent of the way the graph is represented. The concepts and language needed for analyzing a graph's structure, and for understanding the distinction between the structure and its many forms of representation, are the main concern of this chapter.

Two different-seeming graph representations might actually be alternative descriptions of structurally equivalent graphs. Developing a universally applicable method for deciding structural equivalence is called the *graph isomorphism problem*. Although the analogous problem for vector spaces reduces simply to checking if the dimensions are equal, no such simple test is available in graph theory. Some strategies and tests work well in many instances, but no one has developed a practical method to handle all cases.

One prominent aspect of the structure is the system of smaller graphs inside a graph, which are called its *subgraphs*. Subgraphs arise explicitly or implicitly in almost any discussion of graphs. For instance, in the process of building a graph  $G$  from scratch, vertex by vertex, and edge by edge, the entire sequence of graphs formed along the way is a nested chain of subgraphs in  $G$ .

The basic operations of adding a vertex or an edge to a graph, together with the operations of deleting an edge or a vertex, provide a mechanism for transforming any graph, step by step, into any other graph. Computer scientists perceive of a graph as a variable, and they include these *primary operations* in the definition of a graph, in much the same way that mathematicians include the operations of vector addition and scalar multiplication in the definition of a vector space.

Upon the primary graph operations are built subsequent layers of graph algorithms, just as certain algorithms in elementary linear algebra (e.g., *Gaussian elimination*) are

developed by combining the operations of vector addition and scalar multiplication. This layering perspective provides an organizational scheme that rescues graph theory from what might otherwise seem to be a grab bag of concepts and methods. Moreover, this layering approach is consistent with the established principles of software engineering.

Beyond pictures, adjacency lists, and incidence tables, the possible descriptions of a graph include various kinds of matrices. In view of the high level of computational success that matrix representations have achieved for vector spaces, it is not surprising that some forms of matrix representations were also introduced for graphs. The *incidence matrix* and the *adjacency matrix* are two classical matrix representations for graphs, introduced in §2.6, that allow us to establish certain graph properties using matrix-theoretic methods. Even though these matrix representations are in themselves rather inefficient for most graph computations, they do serve as a good start toward developing more efficient *information structures*.

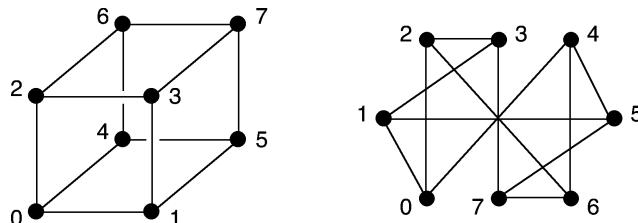
## 2.1 GRAPH ISOMORPHISM

Deciding when two line drawings represent the same graph can be quite difficult for graphs containing more than a few vertices and edges. A related task is deciding when two graphs with different specifications are *structurally equivalent*, that is, whether they have the same pattern of connections. Designing a practical algorithm to make these decisions is a famous unsolved problem, called the *graph-isomorphism problem*.

### Structurally Equivalent Graphs

Since the shape or length of an edge and its position in space are not part of a graph's specification, and neither are edge-crossings or other artifacts of a drawing, each graph has infinitely many spatial representations and drawings.

**Example 2.1.1:** The two line drawings in Figure 2.1.1 both depict the same graph.

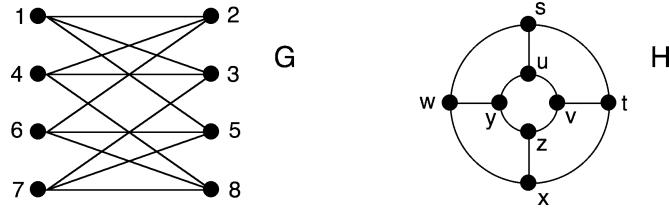


**Figure 2.1.1** Two different drawings of the same graph.

The vertices and edges of the two drawings have matched labels, and they each have only eight vertices and twelve edges. It is easy to verify for every pair  $\{i, j\}$  of vertices that they are adjacent in one graph if and only if they are adjacent in the other. We recognize, therefore, that these two drawings both represent the same graph.

If the graphs are unlabeled, or if the vertices of one drawing are labeled differently from the vertices of another, there are circumstances under which the graphs are nonetheless regarded as virtually the same.

**Example 2.1.2:** The names of the vertices and edges of graphs  $G$  and  $H$  in Figure 2.1.2 differ, but these two graphs are strikingly similar.



**Figure 2.1.2** Two drawings of essentially the same graph.

In particular, the specification for graph  $G$  could be transformed into the specification for graph  $H$  by the following bijection  $f$  on the vertex names.

$$\begin{array}{ll} 1 \rightarrow s & 5 \rightarrow w \\ 2 \rightarrow t & 6 \rightarrow x \\ 3 \rightarrow u & 7 \rightarrow y \\ 4 \rightarrow v & 8 \rightarrow z \end{array}$$

For instance, in graph  $G$ , we see that vertex 1 is adjacent to vertices 2, 3, and 5, but to no others. In graph  $H$ , we see that vertex  $s = f(1)$  is adjacent to vertices  $t = f(2)$ ,  $u = f(3)$ , and  $w = f(5)$ , but to no others. In this sense, the given vertex bijection  $f : V_G \rightarrow V_H$  also acts bijectively on the adjacencies. Our immediate goal is to develop this notion.

### Formalizing Structural Equivalence for Simple Graphs

**DEFINITION:** Let  $G$  and  $H$  be two simple graphs. A vertex bijection  $f : V_G \rightarrow V_H$  **preserves adjacency** if for every pair of adjacent vertices  $u$  and  $v$  in graph  $G$ , the vertices  $f(u)$  and  $f(v)$  are adjacent in graph  $H$ . Similarly,  $f$  **preserves non-adjacency** if  $f(u)$  and  $f(v)$  are non-adjacent whenever  $u$  and  $v$  are non-adjacent.

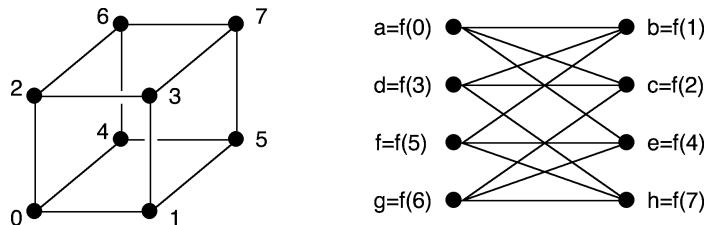
**DEFINITION:** A vertex bijection  $f : V_G \rightarrow V_H$  between the vertex-sets of two simple graphs  $G$  and  $H$  is **structure-preserving** if it preserves adjacency and non-adjacency. That is, for every pair of vertices in  $G$ ,

$$u \text{ and } v \text{ are adjacent in } G \iff f(u) \text{ and } f(v) \text{ are adjacent in } H$$

**DEFINITION:** Two simple graphs  $G$  and  $H$  are **isomorphic**, denoted  $G \cong H$ , if there exists a structure-preserving vertex bijection  $f : V_G \rightarrow V_H$ . Such a function  $f$  between the vertex-sets of  $G$  and  $H$  is called an **isomorphism** from  $G$  to  $H$ .

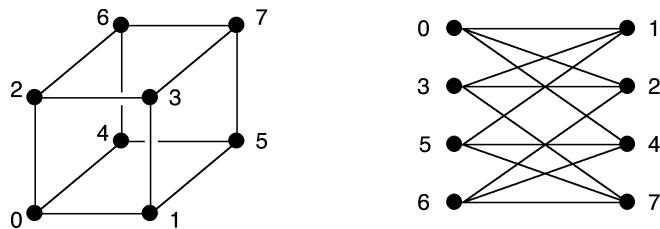
**NOTATION:** When we think of a vertex function  $f : V_G \rightarrow V_H$  as a mapping from one graph to another, we often write  $f : G \rightarrow H$ .

**Example 2.1.3:** Figure 2.1.3 specifies an isomorphism between the two simple graphs shown. Checking that the given vertex bijection is structure-preserving is left to the reader.



**Figure 2.1.3** Specifying an isomorphism between two simple graphs.

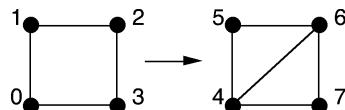
Alternatively, one may simply relabel the vertices of the second graph with the names of the vertices in the first graph, as shown in Figure 2.1.4.



**Figure 2.1.4** Another way of depicting an isomorphism.

**Remark:** A generalization of graph isomorphism, called a *linear graph mapping*, is an adjacency-preserving mapping between the vertex-sets that is not required to preserve non-adjacency and is not necessarily bijective. The reader familiar with *group theory* will notice this analogy: linear graph mapping is to graph isomorphism as group homomorphism is to group isomorphism. In fact, some authors use the term *graph homomorphism* instead of linear graph mapping. Linear graph mappings are introduced in Chapter 10.

**Example 2.1.4:** The vertex function  $j \mapsto j + 4$  depicted in Figure 2.1.5 is bijective and adjacency-preserving, but it is not an isomorphism since it does not preserve non-adjacency. In particular, the non-adjacent pair  $\{0, 2\}$  maps to the adjacent pair  $\{4, 6\}$ .



**Figure 2.1.5** Bijective and adjacency-preserving, but not an isomorphism.

**Example 2.1.5:** The vertex function  $j \mapsto j \bmod 2$  depicted in Figure 2.1.6 preserves adjacency and non-adjacency, but it is not an isomorphism, since it is not bijective.

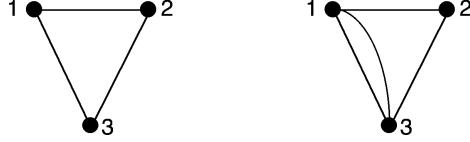


**Figure 2.1.6** Preserves adjacency and non-adjacency, but is not bijective.

### Extending the Definition of Isomorphism to General Graphs

As the following example illustrates, the concept of isomorphism for general graphs requires a more general definition of *structure-preserving* for a vertex bijection.

**Example 2.1.6:** The mapping  $f : V_G \rightarrow V_H$  between the vertex-sets of the two graphs shown in Figure 2.1.7 given by  $f(i) = i$ ,  $i = 1, 2, 3$ , preserves adjacency and non-adjacency, but the two graphs are clearly not structurally equivalent.



**Figure 2.1.7** Two graphs that are not structurally equivalent.

**DEFINITION:** A vertex bijection  $f : V_G \rightarrow V_H$  between the vertex-sets of two graphs  $G$  and  $H$ , simple or general, is **structure-preserving** if

- (i) the number of edges (even if 0) between every pair of distinct vertices  $u$  and  $v$  in graph  $G$  equals the number of edges between their images  $f(u)$  and  $f(v)$  in graph  $H$ , and
- (ii) the number of self-loops at each vertex  $x$  in  $G$  equals the number of self-loops at the vertex  $f(x)$  in  $H$ .

This new definition, which reduces to our original definition of *structure-preserving* for simple graphs, allows us to extend the concept of isomorphism to general graphs.

**DEFINITION:** Two graphs  $G$  and  $H$  (simple or general) are **isomorphic graphs** if there exists a structure-preserving vertex bijection  $f : V_G \rightarrow V_H$ . This relationship is denoted  $G \cong H$ .

### Specifying an Isomorphism Between Graphs Having Multi-Edges

**DEFINITION:** Let  $G$  and  $H$  be two isomorphic graphs. A vertex bijection  $f_V : V_G \rightarrow V_H$  and an edge bijection  $f_E : E_G \rightarrow E_H$  are **consistent** if for every edge  $e \in E_G$ , the function  $f_V$  maps the endpoints of  $e$  to the endpoints of edge  $f_E(e)$ . A consistent mapping pair  $(f_V : V_G \rightarrow V_H, f_E : E_G \rightarrow E_H)$  is often written shorthand as  $f : G \rightarrow H$ .

**Proposition 2.1.1.** *Let  $G$  and  $H$  be any two graphs. Then  $G \cong H$  if and only if there is a vertex bijection  $f_V : V_G \rightarrow V_H$  and an edge bijection  $f_E : E_G \rightarrow E_H$  that are consistent.*

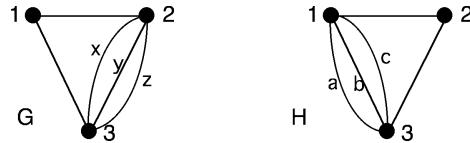
**Proof:** If vertex bijection  $f_V$  and edge bijection  $f_E$  are consistent, then  $f_V$  is structure-preserving. Conversely, if  $G \cong H$ , then, by definition, there is a structure-preserving vertex bijection  $f_V : V_G \rightarrow V_H$ . We may construct a consistent edge bijection as follows: for each pair of distinct vertices  $u, v \in V_G$ , map the set of edges between  $u$  and  $v$  bijectively to the set of edges between vertices  $f_V(u)$  and  $f_V(v)$  (this is possible because these two edge sets have the same number of elements).  $\diamond$

**Remark:** If  $G$  and  $H$  are isomorphic simple graphs, then every structure-preserving vertex bijection  $f : V_G \rightarrow V_H$  induces a unique consistent edge bijection, implicitly given by the rule:  $uv \mapsto f(u)f(v)$ .

However, for a given structure-preserving bijection between the vertex-sets of two graphs with multi-edges, there is more than one *consistent* edge bijection.

**DEFINITION:** If  $G$  and  $H$  are graphs *with multi-edges*, then an **isomorphism** from  $G$  to  $H$  is specified by giving a vertex bijection  $f_V : V_G \rightarrow V_H$  and an edge bijection  $f_E : E_G \rightarrow E_H$  that are consistent.

**Example 2.1.7:** There are two structure-preserving vertex bijections for the isomorphic graphs  $G$  and  $H$  shown in Figure 2.1.8. Each of these vertex bijections has six consistent edge-bijections. Hence there are 12 distinct isomorphisms from  $G$  to  $H$ .



**Figure 2.1.8** There are 12 distinct isomorphisms from  $G$  to  $H$ .

### Isomorphic Graph Pairs

The following properties of isomorphisms provide some preliminary criteria to check when considering the existence or possible construction of a graph isomorphism. Although the examples given involve simple graphs, the properties apply to general graphs as well.

**Theorem 2.1.2.** Let  $G$  and  $H$  be isomorphic graphs. Then they have the same number of vertices and the same number of edges.

**Proof:** Let  $f : G \rightarrow H$  be an isomorphism. The graphs  $G$  and  $H$  must have the same number of vertices and the same number of edges, because an isomorphism maps both sets bijectively.  $\diamond$

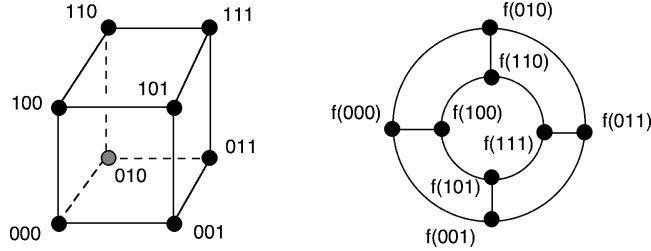
**Theorem 2.1.3.** Let  $f : G \rightarrow H$  be a graph isomorphism and let  $v \in V_G$ . Then  $\deg(f(v)) = \deg(v)$ .

**Proof:** Since  $f$  is structure-preserving, the number of proper edges and the number of self-loops incident on vertex  $v$  equal the corresponding numbers for vertex  $f(v)$ . Thus,  $\deg(f(v)) = \deg(v)$ .  $\diamond$

**Corollary 2.1.4.** Let  $G$  and  $H$  be isomorphic graphs. Then they have the same degree sequence.  $\diamond$

**Corollary 2.1.5.** Let  $f : G \rightarrow H$  be a graph isomorphism and let  $e \in E_G$ . Then the endpoints of edge  $f(e)$  have the same degrees as the endpoints of  $e$ .  $\diamond$

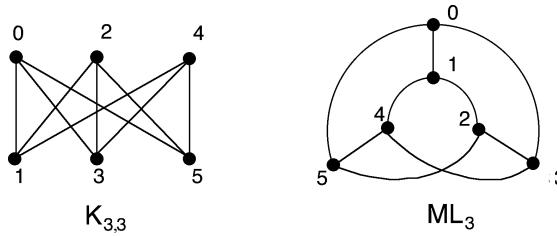
**Example 2.1.8:** In Figure 2.1.9, we observe that both graphs have 8 vertices and 12 edges, and that both are 3-regular. The vertex labelings for the hypercube  $Q_3$  and for the circular ladder  $CL_4$  specify a vertex bijection. A careful examination reveals that this vertex bijection is structure-preserving. It follows that  $Q_3$  and  $CL_4$  are isomorphic graphs.



**Figure 2.1.9** Hypercube graph  $Q_3$  and circular ladder  $CL_4$  are isomorphic.

**DEFINITION:** The **Möbius ladder**  $ML_n$  is a graph obtained from the circular ladder  $CL_n$  by deleting from the circular ladder two of its parallel curved edges and replacing them with two edges that cross-match their endpoints.

**Example 2.1.9:** The complete bipartite graph  $K_{3,3}$  is shown on the left in Figure 2.1.10, and the Möbius ladder  $ML_3$  is shown on the right. Both graphs have 6 vertices and 9 edges, and both are 3-regular. The vertex labelings for the two drawings specify an isomorphism.



**Figure 2.1.10** Bipartite graph  $K_{3,3}$  and Möbius ladder  $ML_3$  are isomorphic.

### Isomorphism Type of a Graph

If  $f = (f_V, f_E)$  is an isomorphism from  $G$  to  $H$ , then  $f^{-1} = (f_V^{-1}, f_E^{-1})$  is an isomorphism from  $H$  to  $G$ . Thus, the relation  $\cong$  (“isomorphic to”) is a symmetric relation on the set of all graphs. Reflexivity and transitivity are also easy to establish, and thus, the relation  $\cong$  is an equivalence relation.

**DEFINITION:** Each equivalence class under  $\cong$  is called an **isomorphism type**.

Isomorphic graphs are graphs of the same isomorphism type. The graphs in Figure 2.1.11 represent all four possible isomorphism types for a simple graph on three vertices. Counting isomorphism types of graphs generally involves the algebra of permutation groups (see Chapter 14).



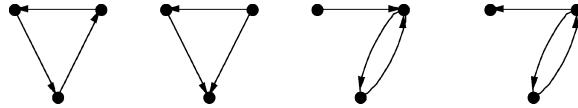
**Figure 2.1.11** The four isomorphism types for a simple 3-vertex graph.

### Isomorphism of Digraphs

The definition of graph isomorphism is easily extended to digraphs.

**DEFINITION:** Two digraphs are **isomorphic** if there is an isomorphism  $f$  between their underlying graphs that preserves the direction of each edge. That is,  $e$  is directed from  $u$  to  $v$  if and only if  $f(e)$  is directed from  $f(u)$  to  $f(v)$ .

**Example 2.1.10:** The four different isomorphism types of a simple digraph with three vertices and three arcs are shown in Figure 2.1.12. Notice that non-isomorphic digraphs can have underlying graphs that are isomorphic.



**Figure 2.1.12** Four non-isomorphic digraphs.

### The Isomorphism Problem

It would be very useful to have a reasonably fast general algorithm that accepts as input any two  $n$ -vertex graphs and that produces as output either an isomorphism between them or a report that the graphs are not isomorphic. Checking whether a given vertex bijection is an isomorphism would require an examination of all vertex pairs, which is not in itself overwhelming. However, since there are  $n!$  vertex bijections to check, this brute-force approach is feasible only for very small graphs.

**DEFINITION:** The **graph-isomorphism problem** is to devise a practical general algorithm to decide graph isomorphism, or, alternatively, to prove that no such algorithm exists.

**Application 2.1.1 Computer Chip Intellectual Property Rights:** Suppose that not long after ABC Corporation develops and markets a computer chip, it happens that the DEF Corporation markets a chip with striking operational similarities. If ABC could prove that DEF's circuitry is merely a rearrangement of the ABC circuitry (i.e., that the circuitries are isomorphic), they might have the basis for a patent-infringement suit. If ABC had to check structure preservation for each of the permutations of the nodes of the DEF chip, the task would take prohibitively long. However, knowledge of the organization of the chips might enable the ABC engineers to take a shortcut.

### EXERCISES for Section 2.1

*In Exercises 2.1.1 through 2.1.6, find all possible isomorphism types of the given kind of simple graph.*

- 2.1.1<sup>s</sup> A 4-vertex tree.
- 2.1.2 A 4-vertex connected graph.
- 2.1.3 A 5-vertex tree.
- 2.1.4 A 6-vertex tree.
- 2.1.5 A 5-vertex graph with exactly three edges.
- 2.1.6 A 6-vertex graph with exactly four edges.

*In Exercises 2.1.7 through 2.1.10, find all possible isomorphism types of the given kind of simple digraph.*

- 2.1.7<sup>s</sup> A simple 4-vertex digraph with exactly three arcs.

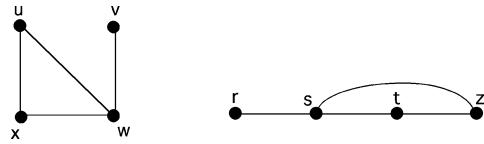
- 2.1.8 A simple 4-vertex digraph with exactly four arcs.  
 2.1.9 A simple 3-vertex strongly connected digraph.  
 2.1.10 A simple 3-vertex digraph with no directed cycles.

*In Exercises 2.1.11 through 2.1.14, find all possible isomorphism types of the given kind of general graph.*

- 2.1.11<sup>s</sup> A graph with two vertices and three edges.  
 2.1.12 A graph with three vertices and two edges.  
 2.1.13 A graph with three vertices and three edges.  
 2.1.14 A 4-vertex graph with exactly four edges including exactly one self-loop and a multi-edge of size 2.

*In Exercises 2.1.15 through 2.1.18, find a vertex-bijection that specifies an isomorphism between the two graphs or digraphs shown.*

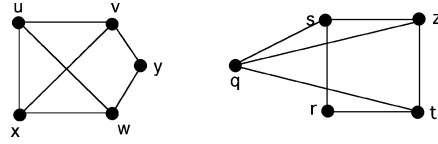
2.1.15<sup>s</sup>



2.1.16



2.1.17



2.1.18



2.1.19 Give an example of two non-isomorphic 5-vertex digraphs whose underlying graphs are isomorphic.

2.1.20 Give an example of two non-isomorphic 9-vertex graphs with the same degree sequence.

## 2.2 AUTOMORPHISMS AND SYMMETRY

**DEFINITION:** An isomorphism from a graph  $G$  to itself is called an **automorphism**.

Thus, an automorphism  $\pi$  of graph  $G$  is a structure-preserving *permutation*  $\pi_V$  on the vertex-set of  $G$ , along with a (consistent) permutation  $\pi_E$  on the edge-set of  $G$ . We may write  $\pi = (\pi_V, \pi_E)$ .

**NOTATION:** When the context is clear, the subscripts that distinguish the vertex and edge actions of an automorphism are suppressed. Thus, we may simply write  $\pi$  in place of  $\pi_V$  or of  $\pi_E$ .

**Remark:** Any structure-preserving vertex-permutation is associated with one (if simple) or more (if there are any multi-edges) automorphisms of  $G$ . The proportion of vertex-permutations of  $V_G$  that are structure-preserving is a measure of the *symmetry* of  $G$ .

### Permutations and Cycle Notation

The most convenient representation of a permutation, for our present purposes, is as a *product of disjoint cycles*.

**NOTATION:** In specifying a permutation, we use the notation  $(x)$  to show that the object  $x$  is fixed, i.e., permuted to itself; the notation  $(x\ y)$  means that objects  $x$  and  $y$  are swapped; and the notation  $(x_0\ x_1\ \cdots\ x_{n-1})$  means that the objects  $x_0, x_1, \dots, x_{n-1}$  are cyclically permuted, so that  $x_j \mapsto x_{j+1} \pmod{n}$ , for  $j = 0, 1, \dots, n-1$ . As explained in Appendix A4, every permutation can be written as a composition of disjoint cycles.

**Example 2.2.1:** The permutation

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 7 & 4 & 1 & 8 & 5 & 2 & 9 & 6 & 3 \end{pmatrix}$$

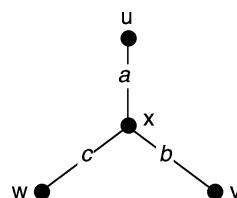
which maps 1 to 7, 2 to 4, and so on, has the **disjoint cycle form**

$$\pi = (1\ 7\ 9\ 3)(2\ 4\ 8\ 6)(5)$$

### Geometric Symmetry

A geometric symmetry on a graph drawing can be used to represent an automorphism on the graph.

**Example 2.2.2:** The graph  $K_{1,3}$  has six automorphisms. Each of them is realizable by a rotation or reflection of the drawing in Figure 2.2.1.



**Figure 2.2.1** The graph  $K_{1,3}$ .

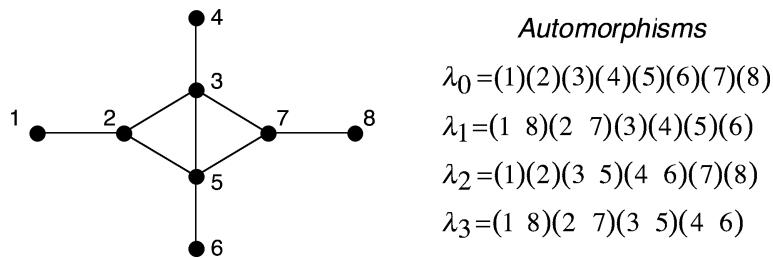
For instance, a  $120^\circ$  clockwise rotation of the figure corresponds to the graph automorphism with vertex-permutation  $(x)(u\ v\ w)$  and edge-permutation  $(a\ b\ c)$ . Also, reflection through the vertical axis corresponds to the the graph automorphism with vertex-permutation  $(x)(u\ v\ w)$  and edge-permutation  $(a)(b\ c)$ . The following table lists all the automorphisms of  $K_{1,3}$  and their corresponding vertex- and edge-permutations.

Symmetry	Vertex permutation	Edge permutation
identity	$(u)(v)(w)(x)$	$(a)(b)(c)$
$120^\circ$ rotation	$(x)(u\ v\ w)$	$(a\ b\ c)$
$240^\circ$ rotation	$(x)(u\ w\ v)$	$(a\ c\ b)$
refl. thru $a$	$(x)(u)(v\ w)$	$(a)(b\ c)$
refl. thru $b$	$(x)(v)(u\ w)$	$(b)(a\ c)$
refl. thru $c$	$(x)(w)(u\ v)$	$(c)(a\ b)$

Since the graph  $K_{1,3}$  has four vertices, the total number of permutations on its vertex-set is 24. By Theorem 2.1.3, every automorphism on  $K_{1,3}$  must fix the 3-valent vertex. Since there are only six permutations of four objects that fix one designated object, it follows that there can be no more than six automorphisms of  $K_{1,3}$ .

**Remark:** Except for a few exercises, the focus of this section is on simple graphs. Since each automorphism of a simple graph  $G$  is completely specified by a structure-preserving vertex-permutation, the automorphism and its corresponding vertex-permutation are often regarded as the same object.

**Example 2.2.3:** Figure 2.2.2 shows a graph with four vertex-permutations, each written in disjoint cyclic notation. It is easy to verify that these vertex-permutations are structure-preserving, so they are all graph automorphisms. We observe that the automorphisms  $\lambda_0, \lambda_1, \lambda_2$ , and  $\lambda_3$  correspond, respectively, to the identity, vertical reflection, horizontal reflection, and  $180^\circ$  rotation. Proving that there are no other automorphisms is left as an exercise.

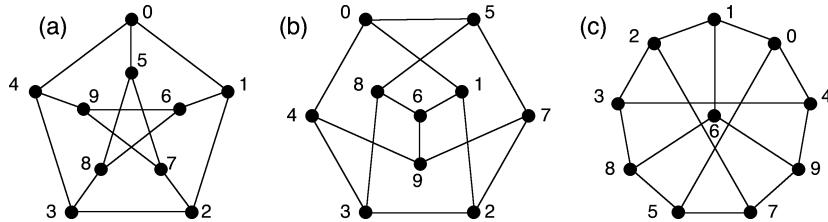


**Figure 2.2.2** A graph with four automorphisms.

### Limitations of Geometric Symmetry

Although looking for geometric symmetry within a drawing may help in discovering automorphisms, there may be automorphisms that are not realizable as reflections or rotations of a particular drawing.

**Example 2.2.4:** Figure 2.2.3 shows three different drawings of the same-labeled Petersen graph. The leftmost drawing has 5-fold rotational symmetry that corresponds to the automorphism  $(0\ 1\ 2\ 3\ 4)(5\ 6\ 7\ 8\ 9)$ , but this automorphism does not correspond to any geometric symmetry of either of the other two drawings. The automorphism  $(0\ 5)(1\ 8)(4\ 7)(2\ 3)(6)(9)$  is realized by 2-fold reflectional symmetry in the middle and rightmost drawings (about the axis through vertices 6 and 9) but is not realizable by any geometric symmetry in the leftmost drawing. There are several other automorphisms that are realizable by geometric symmetry in at least one of the drawings but not realizable in at least one of the other ones. (See Exercises.)



**Figure 2.2.3** Three drawings of the Petersen graph.

### Vertex- and Edge-Transitive Graphs

DEFINITION: A graph  $G$  is **vertex-transitive** if for every vertex pair  $u, v \in V_G$ , there is an automorphism that maps  $u$  to  $v$ .

DEFINITION: A graph  $G$  is **edge-transitive** if for every edge pair  $d, e \in E_G$ , there is an automorphism that maps  $d$  to  $e$ .

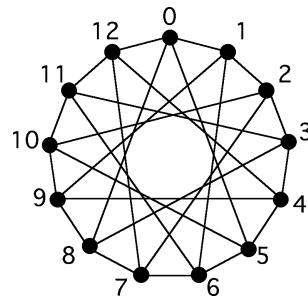
**Example 2.2.5:** The graph  $K_{1,3}$ , discussed in Example 2.2.2, is edge-transitive, but not vertex-transitive, since every automorphism must map the 3-valent vertex to itself.

**Example 2.2.6:** The complete graph  $K_n$  is vertex-transitive and edge-transitive for every  $n$ . (See Exercises.)

**Example 2.2.7:** The hypercube graph  $Q_n$  is vertex-transitive and edge-transitive for every  $n$ . (See Exercises.)

**Example 2.2.8:** The Petersen graph is vertex-transitive and edge-transitive. (See Exercises.)

**Example 2.2.9:** Every circulant graph  $\text{circ}(n; S)$  is vertex-transitive. In particular, the vertex function  $i \mapsto i + k \bmod n$  is an automorphism. In effect, it rotates a drawing of the circulant graph, as in Figure 2.2.4,  $\frac{k}{n}$  of the way around. For instance, vertex 3 can be mapped to vertex 7 by rotating  $\frac{4}{13}$  of the way around. Although  $\text{circ}(13 : 1, 5)$  is edge-transitive, some circulant graphs are not. (See Exercises.)



**Figure 2.2.4** The circulant graph  $\text{circ}(13 : 1, 5)$ .

### Vertex Orbits and Edge Orbits

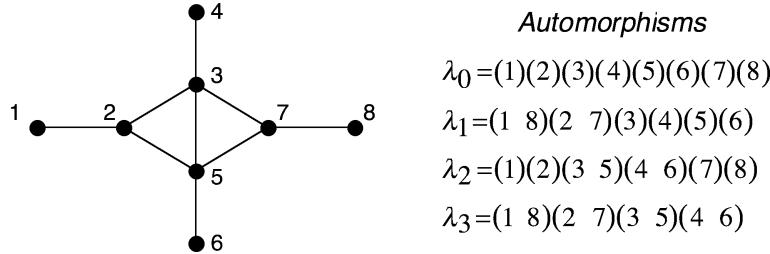
Suppose that two vertices  $u$  and  $v$  of a graph  $G$  are to be considered related if there is an automorphism that maps  $u$  to  $v$ . This is clearly an *equivalence relation* on

the vertices of  $G$ . There is a similar equivalence relation on the edges. (Equivalence relations are discussed in Appendix A.2.)

**DEFINITION:** The equivalence classes of the vertices of a graph under the action of the automorphisms are called **vertex orbits**. The equivalence classes of the edges are called **edge orbits**.

**Example 2.2.10:** The graph of Example 2.2.3 (repeated below in Figure 2.2.5) has the following vertex and edge orbits.

$$\begin{aligned} \text{vertex orbits : } & \{1, 8\}, \{4, 6\}, \{2, 7\}, \{3, 5\} \\ \text{edge orbits : } & \{12, 78\}, \{34, 56\}, \{23, 25, 37, 57\}, \{35\} \end{aligned}$$



**Figure 2.2.5** Graph of Example 2.2.3.

**Theorem 2.2.1.** All vertices in the same orbit have the same degree.

**Proof:** This follows immediately from Theorem 2.1.3.  $\diamond$

**Theorem 2.2.2.** All edges in the same orbit have the same pair of degrees at their endpoints.

**Proof:** This follows immediately from Corollary 2.1.5.  $\diamond$

**Remark:** A vertex-transitive graph is a graph with only one vertex orbit, and an edge-transitive graph is a graph with only one edge orbit.

**Example 2.2.11:** The complete graph  $K_n$  has one vertex orbit and one edge orbit.

**Example 2.2.12:** Each of the two partite sets of the complete bipartite graph  $K_{m,n}$  is a vertex orbit. The graph is vertex-transitive if and only if  $m = n$ ; otherwise it has two vertex orbits. However,  $K_{m,n}$  is always edge-transitive (see Exercises).

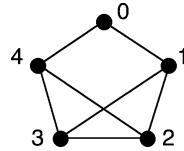
**Remark:** Automorphism theory is one of several graph topics involving an interaction between graph theory and group theory. Chapters 14 and 15 explore various aspects of this interaction.

### How to Find the Orbits

We illustrate how to find orbits by consideration of two examples. (It is not known whether there exists a polynomial-time algorithm for finding orbits. Testing all  $n!$  vertex-permutations for the adjacency preservation property is too tedious an approach.) In addition to using Theorems 2.2.1 and 2.2.2, we observe that if an automorphism maps vertex  $u$  to vertex  $v$ , then it maps the neighbors of  $u$  to the neighbors of  $v$ .

**Example 2.2.13:** In the graph of Figure 2.2.6, vertex 0 is the only vertex of degree 2, so it is in an orbit by itself. The vertical reflection  $(0)(1\ 4)(2\ 3)$  establishes that vertices 1 and 4 are co-orbital and that 2 and 3 are co-orbital. Moreover, since vertices 1 and 4 have a 2-valent neighbor but vertices 2 and 3 do not, the orbit of vertices 1 and 4 must be different from that of 2 and 3. Thus, the vertex orbits are

$$\{0\}, \{1, 4\}, \text{ and } \{2, 3\}$$

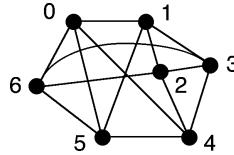


**Figure 2.2.6** Find the vertex orbits and the edge orbits.

Since edge 23 is the only edge, both of whose endpoints have three 3-valent neighbors, it is in an orbit by itself. The edges 01 and 04 are the only edges with a 2-valent endpoint and a 3-valent endpoint, so they could not be co-orbital with any edges except each other. The vertical reflection establishes that the pair  $\{01, 04\}$  is indeed co-orbital, as are the pair  $\{12, 34\}$  and the pair  $\{13, 24\}$ . The latter two pairs combine into a single edge orbit, because of the automorphism  $(0)(1)(4)(2\ 3)$ , which swaps edges 12 and 13 as well as edges 24 and 34. Thus, the edge orbits are

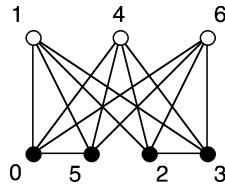
$$\{23\}, \{01, 04\}, \text{ and } \{12, 13, 24, 34\}$$

**Example 2.2.14:** We could approach the 4-regular graph of Figure 2.2.7 by recognizing the symmetry  $(0\ 5)(1\ 4)(2\ 3)(6)$  and seeking to find others. However, it is possible to expedite the determination of orbits.



**Figure 2.2.7** Find the vertex orbits and the edge orbits.

When we look at vertices 0, 2, 3, and 5, we discover that each of them has a set of 3 neighbors that are independent, while vertices 1, 4, and 6 each have two pairs of adjacent vertices. This motivates us to redraw the graph as in Figure 2.2.8.

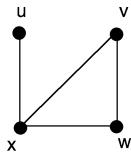


**Figure 2.2.8** Find the vertex orbits and the edge orbits.

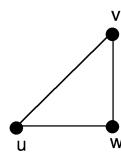
In that form, we see immediately that there are two vertex orbits, namely  $\{0, 2, 3, 5\}$  and  $\{1, 4, 6\}$ . One of the two edge orbits is  $\{05, 23\}$ , and the other contains all the other edges.

**EXERCISES for Section 2.2**

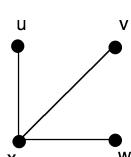
In Exercises 2.2.1 through 2.2.4, write down all the automorphisms of the given simple graph as vertex-permutations.

2.2.1<sup>s</sup>

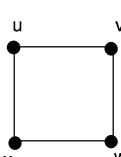
2.2.2



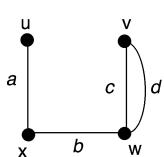
2.2.3



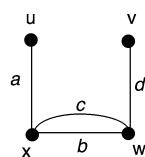
2.2.4



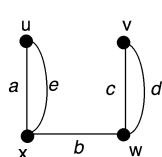
In Exercises 2.2.5 through 2.2.7, specify all the automorphisms of the given graph.

2.2.5<sup>s</sup>

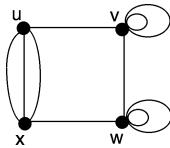
2.2.6



2.2.7



2.2.8 Determine the number of distinct automorphisms of the graph shown below.



In Exercises 2.2.9 through 2.2.12, prove that the specified graph is vertex-transitive.

2.2.9<sup>s</sup>  $K_n$ 2.2.10  $Q_n$ 2.2.11  $\text{circ}(n : S)$ 

2.2.12 The Petersen graph

2.2.13 Prove that the complete bipartite graph  $K_{m,n}$  is vertex-transitive if and only if  $m = n$ .

In Exercises 2.2.14 through 2.2.17, decide for which values of  $n$  (and/or  $m$ ) the specified graph is edge-transitive.

2.2.14<sup>s</sup>  $\text{circ}(9 : 1, m)$ 2.2.15  $\text{circ}(13 : 1, m)$ 2.2.16  $K_{m,n}$ 2.2.17  $Q_n$ 

In Exercises 2.2.18 through 2.2.27, find the vertex and edge orbits of the given graph.

2.2.18<sup>s</sup> The graph of Exercise 2.2.1.

2.2.19 The graph of Exercise 2.2.2.

2.2.20 The graph of Exercise 2.2.3.

2.2.21 The graph of Exercise 2.2.4.

2.2.22 The graph of Exercise 2.2.5.

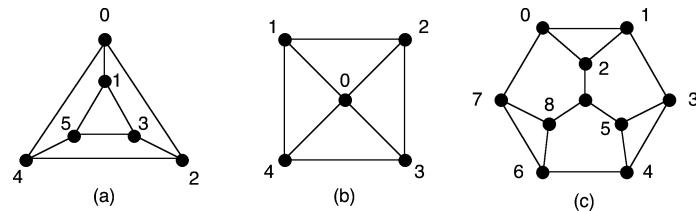
2.2.23 The graph of Exercise 2.2.6.

2.2.24 The graph of Exercise 2.2.7.

2.2.25 The graph of Figure 2.2.9(a).

2.2.26 The graph of Figure 2.2.9(b).

2.2.27 The graph of Figure 2.2.9(c).



**Figure 2.2.9** Three graphs with symmetries.

**2.2.28** Each of the following refers to the three drawings of the Petersen graph that appear in Example 2.2.4.

- (a) Specify an automorphism of the Petersen graph that is realizable by 2-fold reflectional symmetry in the leftmost drawing but not realizable by any geometric symmetry in the other two drawings.
  - (b) Specify an automorphism of the Petersen graph that is realizable by some geometric symmetry in the middle drawing but not realizable by any in the other two drawings.
  - (c) Specify an automorphism of the Petersen graph that is realizable by some geometric symmetry in the rightmost drawing but not realizable by any in the other two drawings.

**2.2.29** Prove that the given list of automorphisms in Example 2.2.3 is complete.

## 2.3 SUBGRAPHS

Properties of a given graph are often determined by the existence or non-existence of certain types of smaller graphs inside it. For instance, Theorem 1.5.3 asserts that a graph is bipartite if and only if it contains no odd cycle. Kuratowski's theorem, proved in Chapter 7, shows that a graph can be drawn in the plane without any edge-crossings if and only if the graph contains (in a more general sense) neither the complete graph  $K_5$  nor the complete bipartite graph  $K_{3,3}$ . This section concentrates on basic kinds of structural containment.

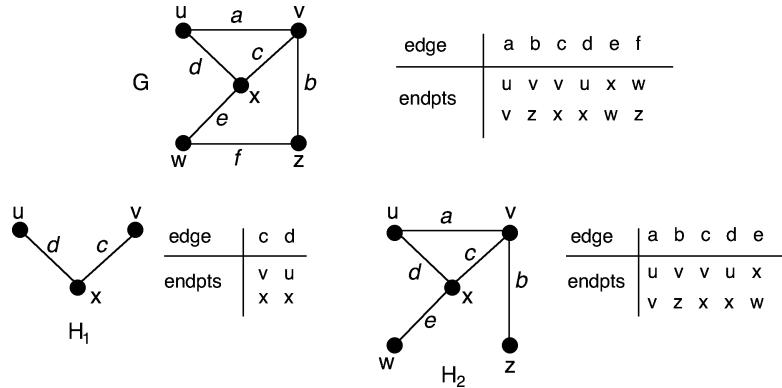
**DEFINITION:** A **subgraph** of a graph  $G$  is a graph  $H$  whose vertices and edges are all in  $G$ . If  $H$  is subgraph of  $G$ , we may also say that  $G$  is a **supergraph** of  $H$ .

**DEFINITION:** A **subdigraph** of a digraph  $G$  is a digraph  $H$  whose vertices and arcs are all in  $G$ .

Equivalently, one can say that a subgraph of a graph  $G$  comprises a subset  $V'$  of  $V_G$  and an subset  $E'$  of  $E_G$  such that the endpoints of every edge in  $E'$  are members of  $V'$ . Thus, the incidence table for a subgraph  $H$  can be obtained simply by deleting from the incidence table of  $G$  each column that does not correspond to an edge in  $H$ .

**DEFINITION:** A **proper** subgraph  $H$  of  $G$  is a subgraph such that  $V_H$  is a proper subset of  $V_G$  or  $E_H$  is a proper subset of  $E_G$ .

**Example 2.3.1:** Figure 2.3.1 shows the line drawings and corresponding incidence tables for two proper subgraphs of a graph.



**Figure 2.3.1** A graph  $G$  and two (proper) subgraphs  $H_1$  and  $H_2$ .

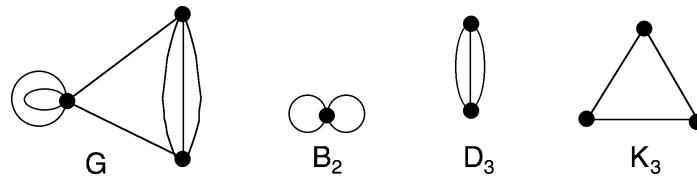
### A Broader Use of the Term “Subgraph”

The usual meaning of the phrase “ $H$  is a subgraph of  $G$ ” is that  $H$  is merely isomorphic to a subgraph of  $G$ .

**Example 2.3.2:** The graph  $G$  of Figure 2.3.1 contains as subgraphs exactly one copy each of  $C_3$ ,  $C_4$ , and  $C_5$ .

**Example 2.3.3:** A graph with  $n$  vertices is hamiltonian if and only if it contains a graph isomorphic to the  $n$ -cycle  $C_n$ .

**Example 2.3.4:** The graph  $G$  shown in Figure 2.3.2 has among its subgraphs a  $B_2$  (bouquet), a  $D_3$  (dipole), and three copies of  $K_3$ .



**Figure 2.3.2** A graph and three of its subgraphs.

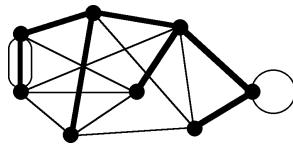
### Spanning Subgraphs

DEFINITION: A subgraph  $H$  is said to **span** a graph  $G$  if  $V_H = V_G$ .

**Example 2.3.5:** In Figure 2.3.1, the subgraph  $H_2$  spans  $G$ , but the subgraph  $H_1$  does not.

DEFINITION: A **spanning tree** of a graph is a spanning subgraph that is a tree.

**Example 2.3.6:** The non-simple graph in Figure 2.3.3 has several different spanning trees. The edges of one of them are drawn in bold.



**Figure 2.3.3** A spanning tree.

A number of important properties related to spanning trees are established in Chapters 3 and 4. Two of these properties are previewed here.

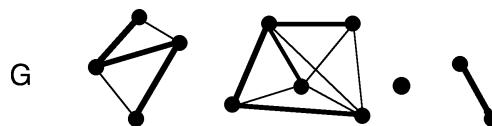
**Proposition 2.3.1.** *A graph is connected if and only if it contains a spanning tree.*  $\diamond$   
(See Exercises or §4.5.)

**Proposition 2.3.2.** *Every acyclic subgraph of a connected graph  $G$  is contained in at least one spanning tree of  $G$ .*  $\diamond$  (See Exercises or §4.5.)

Since each of the components of an acyclic graph is a tree, the following terminology should not be surprising.

DEFINITION: An acyclic graph is called a **forest**.

**Example 2.3.7:** In Figure 2.3.4, the isolated vertex and the bold edges with their endpoints form a forest that spans a 12-vertex graph  $G$ . Each component of the forest is a spanning tree of the corresponding component of  $G$ .



**Figure 2.3.4** A spanning forest  $H$  of graph  $G$ .

### Cliques and Independent Sets

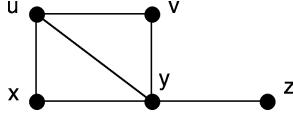
DEFINITION: A subset  $S$  of  $V_G$  is called a **clique** if every pair of vertices in  $S$  is joined by at least one edge, and no proper superset of  $S$  has this property.

Thus, a clique of a graph  $G$  is a maximal subset of mutually adjacent vertices in  $G$ .

TERMINOLOGY NOTE: The graph-theory community seems to be split on whether to require that a clique be maximal with respect to the pairwise-adjacency property. The prose sense of “clique” as an *exclusive* group of people motivates our decision to include maximality in the mathematical definition.

DEFINITION: The **clique number** of a graph  $G$  is the number  $\omega(G)$  of vertices in a largest clique in  $G$ .

**Example 2.3.8:** There are three cliques in the graph shown in Figure 2.3.5. In particular, each of the vertex subsets,  $\{u, v, y\}$ ,  $\{u, x, y\}$ , and  $\{y, z\}$  induce complete subgraphs, and no proper superset of any of them induces a complete subgraph. The clique number is 3, the cardinality of the largest clique.



**Figure 2.3.5** A graph with three cliques.

**DEFINITION:** A subset  $S$  of  $V_G$  is said to be an **independent set** if no pair of vertices in  $S$  is joined by an edge. That is,  $S$  is a subset of mutually non-adjacent vertices of  $G$ .

**DEFINITION:** The **independence number** of a graph  $G$  is the number  $\alpha(G)$  of vertices in a largest independent set in  $G$ .

**Remark:** Thus, the clique number  $\omega(G)$  and the independence number  $\alpha(G)$  of a graph may be regarded as complementary concepts. In §2.4, a precise notion of complementation is introduced.

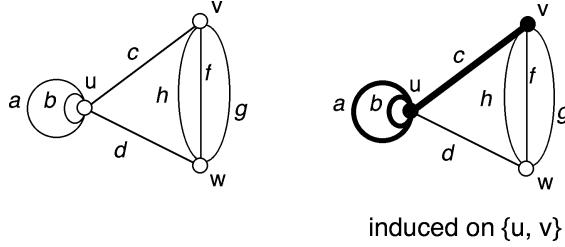
### Induced Subgraphs

**DEFINITION:** For a given graph  $G$ , the **subgraph induced on a vertex subset**  $U$  of  $V_G$ , denoted  $G(U)$ , is the subgraph of  $G$  whose vertex-set is  $U$  and whose edge-set consists of all edges in  $G$  that have both endpoints in  $U$ . That is,

$$V_{G(U)} = U \quad \text{and} \quad E_{G(U)} = \{e \in E_G \mid \text{endpts}(e) \subseteq U\}$$

**Example 2.3.9:** If  $G$  is a simple graph, then it follows from the definition of a clique that the subgraph induced on a clique is a complete graph. Some authors refer to this complete graph as a clique.

**Example 2.3.10:** A 3-vertex graph is shown on the left in Figure 2.3.6, and the subgraph induced on the vertex subset  $\{u, v\}$  is shown in bold on the right.



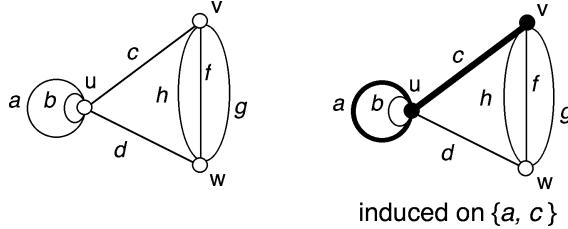
induced on  $\{u, v\}$

**Figure 2.3.6** A subgraph induced on a subset of vertices.

**DEFINITION:** For a given graph  $G$ , the **subgraph induced on an edge subset**  $D$  of  $E_G$ , denoted  $G(D)$ , is the subgraph of  $G$  whose edge-set is  $D$  and whose vertex-set consists of all vertices that are incident with at least one edge in  $D$ . That is,

$$V_{G(D)} = \{v \in V_G \mid v \in \text{endpts}(e), \text{ for some } e \in D\} \quad \text{and} \quad E_{G(D)} = D$$

**Example 2.3.11:** In Figure 2.3.7, the induced subgraph on the edge subset  $\{a, c\}$  is shown on the right in bold.

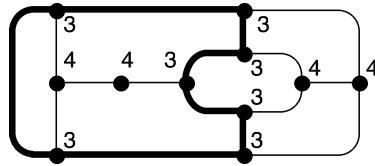


**Figure 2.3.7** A subgraph induced on a subset of edges.

The subdigraphs induced on a vertex subset or directed-edge subset are analogously defined.

**DEFINITION:** The **center of a graph**  $G$ , denoted  $Z(G)$ , is the subgraph induced on the set of central vertices of  $G$  (see §1.4).

**Example 2.3.12:** The vertices of the graph in Figure 2.3.8 are labeled by their eccentricities. Since the minimum eccentricity is 3, the vertices of eccentricity 3 lie in the center.



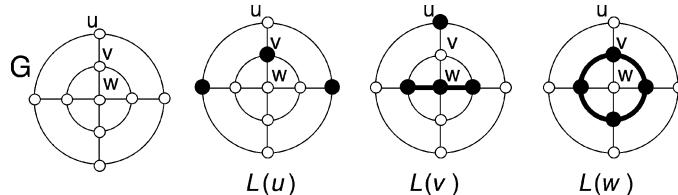
**Figure 2.3.8** A graph whose center is a 7-cycle.

**Remark:** In §2.4, we show that any graph can be the center of a graph.

### Local Subgraphs

**DEFINITION:** The (*open*) **local subgraph** (or (*open*) **neighborhood subgraph**) of a vertex  $v$  is the subgraph  $L(v)$  induced on the neighbors of  $v$ .

**Example 2.3.13:** Figure 2.3.9 shows a graph and the local subgraphs of three of its vertices.

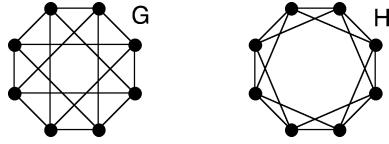


**Figure 2.3.9** A graph  $G$  and three of its local subgraphs.

**Theorem 2.3.3.** Let  $f : G \rightarrow H$  be a graph isomorphism and  $u \in V_G$ . Then  $f$  maps the local subgraph  $L(u)$  of  $G$  isomorphically to the local subgraph  $L(f(u))$  of  $H$ .

**Proof:** This follows since  $f$  maps the vertices of  $L(u)$  bijectively to the vertices of  $L(f(u))$ , and since  $f$  is edge-multiplicity preserving.  $\diamond$

**Example 2.3.14:** The local subgraphs for graph  $G$  of Figure 2.3.10 are all isomorphic to  $4K_1$ . The local subgraphs for graph  $H$  are all isomorphic to  $P_3$ . Thus, the two graphs are not isomorphic. Alternatively, we observe that  $\alpha(G) = 4$ , but  $\alpha(H) = 2$ , and also that  $\omega(G) = 2$ , but  $\omega(H) = 3$ .



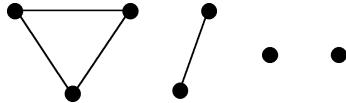
**Figure 2.3.10** Two 4-regular 8-vertex graphs.

## Components

**DEFINITION:** A **component** of a graph  $G$  is a *maximal* connected subgraph of  $G$ . In other words, a connected subgraph  $H$  is a component of  $G$  if  $H$  is not a proper subgraph of any connected subgraph of  $G$ .

The only component of a connected graph is the entire graph. Intuitively, the components of a non-connected graph are the “whole pieces” it comprises.

**Example 2.3.15:** The 7-vertex graph in Figure 2.3.11 has four components.



**Figure 2.3.11** A graph with four components.

**REVIEW FROM §1.4:** A vertex  $v$  is said to be **reachable** from vertex  $u$  if there is a walk from  $u$  to  $v$ .

Notice that each vertex in a component is reachable from every other vertex in that component. This is because membership in the same component is an *equivalence relation* on  $V_G$ , called the *reachability relation*.<sup>†</sup>

**DEFINITION:** In a graph, the **component of a vertex**  $v$ , denoted  $C(v)$ , is the subgraph induced by the subset of all vertices reachable from  $v$ .

Observe that if  $w$  is any vertex in  $C(v)$ , then  $C(w) = C(v)$ . This suggests the following alternate definition of component.

**ALTERNATIVE DEFINITION:** A **component** of a graph  $G$  is a subgraph induced by an equivalence class of the reachability relation on  $V_G$ .

**COMPUTATIONAL NOTE:** Identifying the components of a small graph is trivial. But larger graphs that are specified by some computer representation require a computer algorithm. These “component-finding” algorithms are developed in Chapter 4 as straightforward applications of certain graph-traversal procedures.

---

<sup>†</sup> Equivalence relations are discussed in Appendix A.2.

### Application to Parallel-Computer Computation

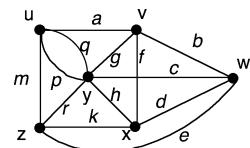
The following application shows how subgraph analysis might occur within a scaled-down model for porting a *distributed algorithm* from one kind of parallel architecture to another. A distributed algorithm is an algorithm that has been designed so that different steps are executed simultaneously on different processing units within a parallel computer.

**Application 2.3.1 Porting an Algorithm:** Suppose that a distributed algorithm has been designed to run on an interconnection network whose architecture is an  $8 \times 8 \times 32$  array. Suppose also that the only immediately available interconnection network is a 13-dimensional hypercube graph  $Q_{13}$  (see §1.2). If the 13-dimensional hypercube graph contains an  $8 \times 8 \times 32$  mesh as a subgraph, then the algorithm could be directly ported to that hypercube.

**Remark:** In general, the ideal situation in which the available network (the *host*) is a supergraph of the required network (the *guest*) is not likely to occur. More realistically, one would like to associate the nodes of the guest with the nodes of the host in such a way as to minimize the decrease in performance of the algorithm. The concept of *graph mapping*, introduced in Chapter 10, addresses this problem. In Chapter 15, a number of architectures are shown to be subgraphs or “nearly” subgraphs of the hypercube, demonstrating the robustness of the hypercube as an interconnection network.

### EXERCISES for Section 2.3

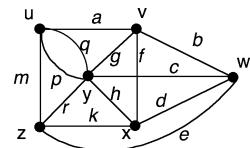
In Exercises 2.3.1 and 2.3.2, determine whether each of the given vertex and edge subsets,  $W$  and  $D$ , respectively, form a subgraph of the graph at the right.



- 2.3.1<sup>s</sup> a.  $W = \{u, v, y\}$ ;  $D = \{q, g\}$   
 b.  $W = \{u, v, y\}$ ;  $D = \{p, q\}$   
 c.  $W = \{u, v, y\}$ ;  $D = \{p, r, h\}$

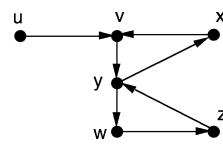
- 2.3.2 a.  $W = \{u, w, y\}$ ;  $D = \{a, b, c\}$   
 b.  $W = \{u, w, y\}$ ;  $D = \{c, q\}$   
 c.  $W = \{u, w, y\}$ ;  $D = \{g, r, h\}$

In Exercises 2.3.3 to 2.3.6, find the subgraphs  $G(W)$  and  $G(D)$  induced on the given vertex subset  $W$  and on the edge subset  $D$ , respectively, of the graph at the right.



- 2.3.3<sup>s</sup>  $W = \{u, v, y\}$ ;  $D = \{p, r, e\}$       2.3.4  $W = \{u, v, x\}$ ;  $D = \{g, c, h\}$   
 2.3.5  $W = \{u, w, y\}$ ;  $D = \{b, c, d\}$       2.3.6  $W = \{x, y, z, w\}$ ;  $D = \{f, c, h\}$

In Exercises 2.3.7 to 2.3.12, find the subdigraph  $G(U)$  induced on the given vertex subset  $U$  of the digraph at the right.



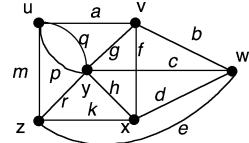
- 2.3.7<sup>s</sup>  $U = \{u, v, x\}$       2.3.8  $U = \{y\}$       2.3.9  $U = \{u, v, y\}$   
 2.3.10  $U = \{x, y, z\}$       2.3.11  $U = \{u, w, y, z\}$       2.3.12  $U = \{u, x, z\}$

In Exercises 2.3.13 to 2.3.15, find the local subgraphs of the given vertex in the graph  $G$  at the right.

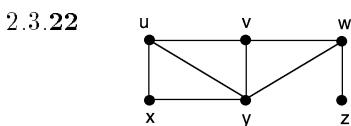
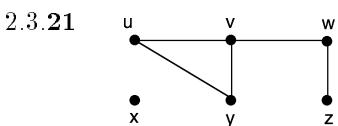
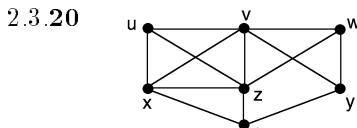
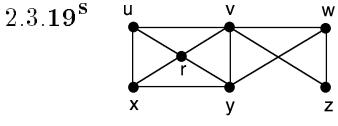
2.3.13<sup>s</sup>  $u$   
2.3.16  $x$

2.3.14  $v$   
2.3.17  $y$

2.3.15  $w$   
2.3.18  $z$



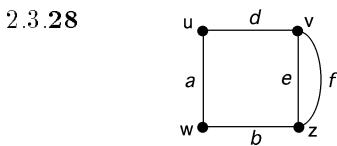
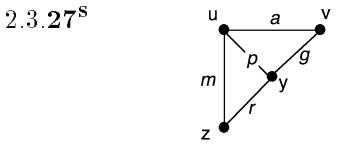
In Exercises 2.3.19 to 2.3.22, do all of the following. (a) Find all of the cliques in the given graph. (b) Give the clique number. (c) Find all the maximal independent sets. (d) Give the independence number. (e) Find the center.



In Exercises 2.3.23 through 2.3.26, determine in the given graph the largest cardinality of a set of mutually adjacent vertices that is not a clique, and find one such set.

- 2.3.23<sup>s</sup> The graph of Exercise 2.3.19.  
 2.3.24 The graph of Exercise 2.3.20.  
 2.3.25 The graph of Exercise 2.3.21.  
 2.3.26 The graph of Exercise 2.3.22.

In Exercises 2.3.27 and 2.3.28, find the edge-sets of all spanning trees of the given graph.



In Exercises 2.3.29 to 2.3.32, find all possible isomorphism types of the given kind of graph.

- 2.3.29 A 6-vertex forest with exactly two components.  
 2.3.30 A simple 4-vertex graph with exactly two components.  
 2.3.31 A simple 5-vertex graph with exactly three components.  
 2.3.32 A simple 6-vertex graph with exactly three components.  
 2.3.33 How many of the subgraphs of  $K_n$  are complete graphs?  
 2.3.34 Prove that the reachability relation is an equivalence relation on  $V_G$ .  
 2.3.35 Prove that the subgraph induced by an equivalence class of the vertex-reachability relation on a graph  $G$  is a component of  $G$ .  
 2.3.36 Draw a forest having ten vertices, seven edges, and three components.  
 2.3.37<sup>s</sup> Given four vertices  $x, y, z, w$  drawn in the plane, how many 2-component forests can be drawn having those four vertices?  
 2.3.38 Prove or disprove: Every subgraph of a bipartite graph is a bipartite graph.

**2.3.39** Prove or disprove: For every subgraph  $H$  of any graph  $G$ , there exists a vertex subset  $W$  such that  $H = G(W)$ .

**2.3.40** Prove or disprove: For every subgraph  $H$  of any graph  $G$ , there exists an edge subset  $D$  such that  $H = G(D)$ .

**2.3.41** Let  $U$  and  $W$  be vertex subsets of a graph  $G$ . Under what conditions will  $G(U) \cup G(W) = G(U \cup W)$ ?

**2.3.42** Let  $D$  and  $F$  be edge subsets of a graph  $G$ . Under what conditions will  $G(D) \cup G(F) = G(D \cup F)$ ?

**2.3.43** Characterize those graphs with the property that the local subgraph of each vertex is a complete graph.

**2.3.44** Characterize the class of graphs that have this property: every induced subgraph is connected.

**2.3.45** Prove Proposition 2.3.1. (Hint: Prove that an edge-minimal connected spanning subgraph of  $G$  must be a spanning tree of  $G$ .)

**2.3.46** Prove Proposition 2.3.2. (Hint: Let  $H$  be an acyclic subgraph of a connected graph  $G$ , and let  $S$  be the set of all acyclic supergraphs of  $H$  that are spanning subgraphs of  $G$ . Then show that an edge-minimal element of the set  $S$  must be a spanning tree of  $G$  that contains  $H$ .)

## 2.4 SOME GRAPH OPERATIONS

Computer scientists often regard a graph as a variable. Accordingly, the configuration that results when a vertex or edge is added to or deleted from a graph  $G$  is considered to be a new value of  $G$ . These primary operations are part of the datatype *graph*, just as the operations of addition and scalar multiplication are part of the definition of a vector space. In Chapter 4, other primary operations, intended for information retrieval, are constructed with the aid of spanning trees. In general, all non-primary operations can be constructed by combining and/or iterating the primary operations.

### Deleting Vertices or Edges

**DEFINITION:** If  $v$  is a vertex of a graph  $G$ , then the **vertex-deletion subgraph**  $G - v$  is the subgraph induced by the vertex-set  $V_G - \{v\}$ . That is,

$$V_{G-v} = V_G - \{v\} \quad \text{and} \quad E_{G-v} = \{e \in E_G \mid v \notin \text{endpts}(e)\}$$

More generally, if  $U \subseteq V_G$ , then the result of iteratively deleting all the vertices in  $U$  is denoted  $G - U$ .

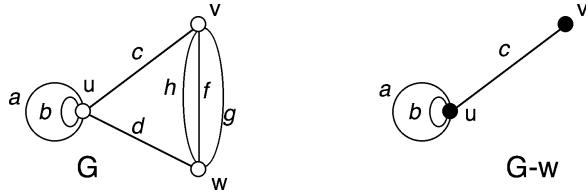
**DEFINITION:** If  $e$  is an edge of a graph  $G$ , then the **edge-deletion subgraph**  $G - e$  is the subgraph induced by the edge-set  $E_G - \{e\}$ . That is,

$$V_{G-e} = V_G \quad \text{and} \quad E_{G-e} = E_G - \{e\}$$

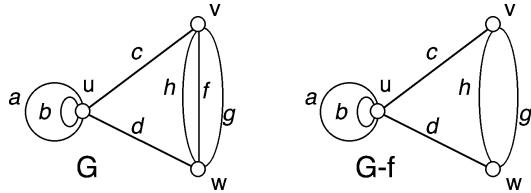
More generally, if  $D \subseteq E_G$ , then the result of iteratively deleting all the edges in  $D$  is denoted  $G - D$ .

The vertex-deletion and edge-deletion subdigraphs are analogously defined.

**Example 2.4.1:** The next two figures show one of the vertex-deletion subgraphs of a graph  $G$  and one of the edge-deletion subgraphs.



**Figure 2.4.1** The result of deleting the vertex  $w$  from graph  $G$ .



**Figure 2.4.2** The result of deleting the edge  $f$  from graph  $G$ .

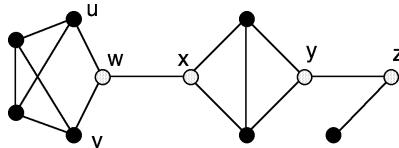
### Network Vulnerability

The following definitions identify the most vulnerable parts of a network. These definitions lead to different characterizations of a graph's connectivity, which appear in Chapter 5.

**DEFINITION:** A **vertex-cut** in a graph  $G$  is a vertex-set  $U$  such that  $G - U$  has more components than  $G$ .

**DEFINITION:** A **cut-vertex** (or **cutpoint**) is a vertex-cut consisting of a single vertex.

**Example 2.4.2:** If the graph in Figure 2.4.3 represents a communications network, then a breakdown at any of the four cut-vertices  $w$ ,  $x$ ,  $y$ , or  $z$  would destroy the connectedness of the network. Also,  $\{u, v\}$  and  $\{w, x\}$  are both vertex-cuts, and  $\{u, v\}$  is a *minimal* vertex-cut (i.e., no proper subset of  $\{u, v\}$  is a vertex-cut).

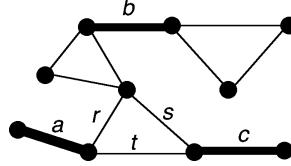


**Figure 2.4.3** A graph with four cut-vertices.

**DEFINITION:** An **edge-cut** in a graph  $G$  is a set of edges  $D$  such that  $G - D$  has more components than  $G$ .

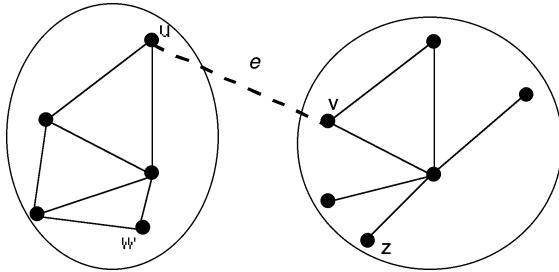
**DEFINITION:** A **cut-edge** (or **bridge**) is an edge-cut consisting of a single edge.

**Example 2.4.3:** For the graph shown in Figure 2.4.4, edges  $a$ ,  $b$ , and  $c$  are cut-edges;  $\{r, s, t\}$  is an edge-cut; and  $\{r, s\}$  is a *minimal* edge-cut.



**Figure 2.4.4** A graph with three cut-edges.

**Observation:** In a connected graph  $G$ , let  $e$  be any edge, with endpoints  $u$  and  $v$ . Then in the edge-deletion graph  $G - e$ , every vertex is reachable either from  $u$  or from  $v$  (or possibly from both). Thus, the only possible components of  $G - e$  are the component  $C_{G-e}[u]$  that contains the vertex  $u$  and the component  $C_{G-e}[v]$ . These two components coincide if and only if there is a path in  $G$  from  $u$  to  $v$  that avoids the edge  $e$  (in which case  $G - e$  is connected). For example, in Figure 2.4.5, the graph  $G - e$  has two components,  $C_{G-e}[u]$  and  $C_{G-e}[v]$ . Notice that had there been an edge in the original graph joining the vertices  $w$  and  $z$ , then  $G - e$  would have been connected.



**Figure 2.4.5** The components  $C_{G-e}[u]$  and  $C_{G-e}[v]$ .

**DEFINITION:** An edge  $e$  of a graph is called a **cycle-edge** if  $e$  lies in some cycle of that graph.

**Proposition 2.4.1.** *Let  $e$  be an edge of a connected graph  $G$ . Then  $G - e$  is connected if and only if  $e$  is a cycle-edge of  $G$ .*

**Proof:** Let  $u$  and  $v$  be the endpoints of the edge  $e$ . If  $e$  lies in some cycle of  $G$ , then the long way around that cycle is a path between vertices  $u$  and  $v$ , and by the observation above, the graph  $G - e$  is connected. Conversely, if  $G - e$  is connected, then there is a path  $P$  from  $u$  to  $v$  that avoids  $e$ . Thus,  $P + e$  is a cycle in  $G$  containing  $e$ . ◇

The first of the following two corollaries restates Proposition 2.4.1 as a characterization of cut-edges, and the second combines the result of the proposition with the observation above.

**Corollary 2.4.2.** *An edge of a graph is a cut-edge if and only if it is not a cycle-edge.*

**Proof:** Apply Proposition 2.4.1 to the component that contains that edge. ◇

**NOTATION:** The number of components of a graph  $G$  is denoted  $c(G)$ .

**Corollary 2.4.3.** *Let  $e$  be any edge of a graph  $G$ . Then*

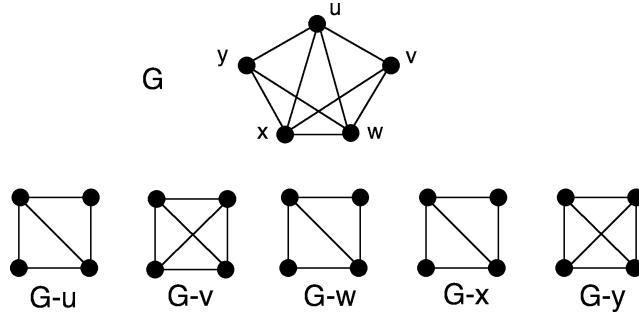
$$c(G - e) = \begin{cases} c(G), & \text{if } e \text{ is a cycle-edge} \\ c(G) + 1, & \text{otherwise} \end{cases}$$
◇

### The Graph-Reconstruction Problem

DEFINITION: Let  $G$  be a graph with  $V_G = \{v_1, v_2, \dots, v_n\}$ . Then the **vertex-deletion subgraph list** of  $G$  is the list of the subgraphs  $G - v_1, \dots, G - v_n$ .

DEFINITION: The **reconstruction deck** of a graph is its vertex-deletion subgraph list, with no labels on the vertices. We regard each individual vertex-deletion subgraph as being a **card** in the deck.

**Example 2.4.4:** A graph  $G$  and its labeled vertex-deletion subgraph list is shown in Figure 2.4.6.



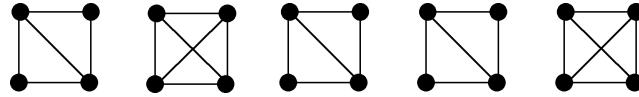
**Figure 2.4.6** A graph and its vertex-deletion subgraph list.

DEFINITION: The **graph-reconstruction problem** is to decide whether two non-isomorphic simple graphs with three or more vertices can have the same reconstruction deck.

**Remark:** We observe that the 2-vertex graphs  $K_2$  and  $2K_1$  (two non-adjacent vertices) have the same deck.

**Remark:** The graph-reconstruction problem would be easy to solve if the vertex-deletion subgraphs included the vertex and edge names. However, the problem is concerned with a list of unlabeled graphs, and it is among the foremost unsolved problems in graph theory. The problem was originally formulated by P.J. Kelly and S.M. Ulam in 1941.

**Example 2.4.5:** In terms of Example 2.4.4, the graph-reconstruction problem asks whether the graph  $G$  in Figure 2.4.6 is the only graph (up to isomorphism) that has the following deck.



**Figure 2.4.7** Is  $G$  the only graph with this reconstruction deck?

**Reconstruction Conjecture:** Let  $G$  and  $H$  be two graphs with at least three vertices and with  $V_G = \{v_1, v_2, \dots, v_n\}$  and  $V_H = \{w_1, w_2, \dots, w_n\}$ , such that  $G - v_i \cong H - w_i$ , for each  $i = 1 \dots, n$ . Then  $G \cong H$ .

The following results indicate that some information about the original graph can be derived from its vertex-deletion subgraph list. This information is a big help in solving small reconstruction problems.

**Theorem 2.4.4.** *The number of vertices and the number of edges of a graph  $G$  can be calculated from its vertex-deletion subgraph list.*

**Proof:** If the length of the vertex-deletion subgraph list is  $n$ , then clearly  $|V_G| = n$ . Moreover, since each edge appears only in the  $n - 2$  subgraphs that do not contain either of its endpoints, it follows that the sum  $\sum_v |E_{G-v}|$  counts each edge  $n - 2$  times. Thus,

$$|E_G| = \frac{1}{n-2} \sum_v |E_{G-v}| \quad \diamond$$

**Corollary 2.4.5.** *The degree sequence of a graph  $G$  can be calculated from its reconstruction deck.*

**Proof:** First calculate  $|E_G|$ . For each card, the degree of the missing vertex is the difference between  $|E|$  and the number of edges on that card.  $\diamond$

**Corollary 2.4.6.** *A regular graph can be reconstructed from its reconstruction deck.*

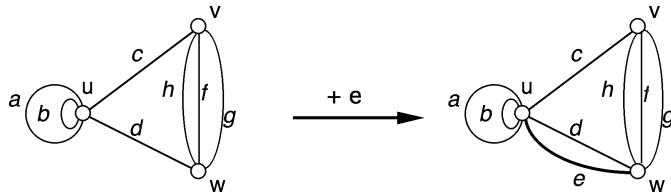
**Proof:** By Corollary 2.4.5, we could determine from the deck that the graph is regular and its degree  $d$  of regularity. On any card in the deck, there would be  $d$  vertices of degree  $d - 1$ . To reconstruct the graph, join those vertices to the missing vertex.  $\diamond$

**Remark:** B. McKay [Mc77] and A. Nijenhuis [Ni77] have shown, with the aid of computers, that a counterexample to the conjecture would have to have at least 10 vertices.

### Adding Edges or Vertices

**DEFINITION:** **Adding an edge** between two vertices  $u$  and  $w$  of a graph  $G$  means creating a supergraph, denoted  $G \cup \{e\}$ , with vertex-set  $V_G$  and edge-set  $E_G \cup \{e\}$ , where  $e$  is a new edge with endpoints  $u$  and  $w$ .

**Example 2.4.6:** Figure 2.4.8 shows the effect of the operation of adding an edge.



**Figure 2.4.8** Adding an edge  $e$  with endpoints  $u$  and  $w$ .

**DEFINITION:** **Adding a vertex**  $v$  to a graph  $G$ , where  $v$  is a new vertex not already in  $V_G$ , means creating a supergraph, denoted  $G \cup \{v\}$ , with vertex-set  $V_G \cup \{v\}$  and edge-set  $E_G$ .

**DEFINITION:** Any one of the operations of adding or deleting a vertex or adding or deleting an edge is called a **primary maintenance operation**.

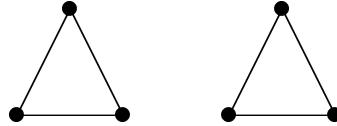
**Remark:** *Secondary operations* on a graph are those that are realizable by a combination and/or repetition of one or more of the primary graph operations. Whether or not one is designing software, it is mathematically interesting to analyze or synthesize a graph construction as a sequence of primary operations.

### Graph Union

The iterative application of vertex addition results in the secondary graph operation called *graph union*.

**DEFINITION:** The **(graph) union** of two graphs  $G = (V, E)$  and  $G' = (V', E')$  is the graph  $G \cup G'$  whose vertex-set and edge-set are the disjoint unions, respectively, of the vertex-sets and edge-sets of  $G$  and  $G'$ .

**Example 2.4.7:** The graph union  $K_3 \cup K_3$  is two disjoint copies of  $K_3$ , as shown in Figure 2.4.9. This illustrates clearly that the graph union of two graphs is *not* given by the set-theoretic union of their vertex-sets and the set-theoretic union of their edge-sets, which in this example would be a single copy of  $K_3$ .



**Figure 2.4.9** The graph union  $K_3 \cup K_3$  of two copies of  $K_3$ .

**DEFINITION:** The  **$n$ -fold self-union**  $nG$  is the iterated disjoint union  $G \cup \dots \cup G$  of  $n$  copies of the graph  $G$ .

**Example 2.4.8:** The graph in Figure 2.4.9 is the 2-fold self-union  $2K_3$ .

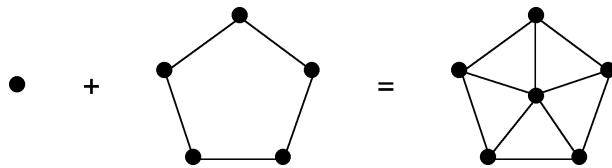
### Joining a Vertex to a Graph

Another secondary operation is the *join* of a new vertex to an existing graph. A more general operation joining two graphs is defined in §2.7.

**DEFINITION:** If a new vertex  $v$  is joined to each of the pre-existing vertices of a graph  $G$ , then the resulting graph is called the **join of  $v$  to  $G$**  or the **suspension of  $G$  from  $v$** , and is denoted  $G + v$ .

Thus, to join a new vertex  $v$  to a graph  $G$ , using only primary operations, first add the vertex  $v$  to  $G$ . Then add one new edge between  $v$  and each pre-existing vertex of  $G$ .

**DEFINITION:** The  **$n$ -wheel**  $W_n$  is the join  $K_1 + C_n$  of a single vertex and an  $n$ -cycle. (The  $n$ -cycle forms the rim of the wheel, and the additional vertex is its hub.) If  $n$  is even, then  $W_n$  is called an **even wheel**; if odd, then  $W_n$  is called an **odd wheel**.



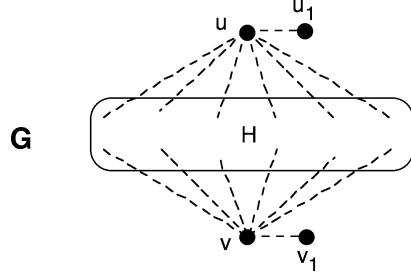
**Figure 2.4.10** The 5-wheel  $W_5 = K_1 + C_5$ .

The following proposition hints at the general usefulness of the join construction.

**Proposition 2.4.7.** Let  $H$  be a graph of diameter  $d$ . Then there is a graph  $G$  of radius  $d$  of which  $H$  is the center.

**Proof:** First construct a supergraph  $\hat{H}$  by joining two new vertices  $u$  and  $v$  to every vertex of  $H$ , but not to each other; that is,  $\hat{H} = ((H + u) + v) - uv$ . Then form the

supergraph  $G$  by attaching new vertices  $u_1$  and  $v_1$  to vertices  $u$  and  $v$ , respectively, with edges  $uu_1$  and  $vv_1$ , as in Figure 2.4.11. Thus,  $G = (\hat{H} \cup 2K_1) + uu_1 + vv_1$ , where the two copies of  $K_1$  are the two trivial graphs consisting of vertices  $u_1$  and  $v_1$ . It is easy to verify that, in graph  $G$ , each vertex of  $H$  has eccentricity 2, vertices  $u$  and  $v$  have eccentricity 3, and  $u_1$  and  $v_1$  have eccentricity 4. Thus,  $H$  is the center of  $G$ .  $\diamond$

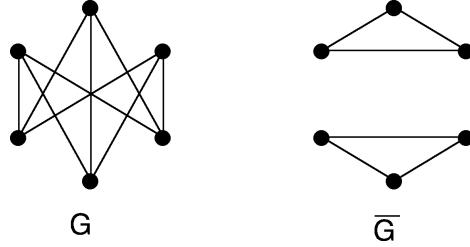


**Figure 2.4.11** Making graph  $H$  the center of a graph  $G$ .

### Edge-Complementation

**DEFINITION:** Let  $G$  be a simple graph. Its **edge-complement** (or **complement**)  $\overline{G}$  is the graph on the same vertex-set, such that two vertices are adjacent in  $\overline{G}$  if and only if they are not adjacent in  $G$ .

**Example 2.4.9:** A graph and its edge-complement are shown in Figure 2.4.12.



**Figure 2.4.12** A graph and its complement.

To construct the edge-complement of the graph variable  $G$  from primary operations, initialize the graph variable  $H$  as the null graph (i.e.,  $V_H = E_H = \emptyset$ ). Next, for each vertex of  $G$ , add a vertex to  $H$ . Then for each non-adjacent pair of vertices in  $G$ , add an edge to  $H$ . The final value of the graph variable  $H$  is then assigned to  $\overline{G}$ .

**Remark:** Observe that the edge-complement of the edge-complement always equals the original graph, i.e.,  $\overline{\overline{G}} = G$ .

The following theorem formulates precisely the sense in which the independence number of a graph and the clique number are complementary.

**Theorem 2.4.8.** *Let  $G$  be a simple graph. Then*

$$\omega(G) = \alpha(\overline{G}) \quad \text{and} \quad \alpha(G) = \omega(\overline{G})$$

$\diamond$

**Remark:** Other graph operations are defined in §2.7.

**EXERCISES for Section 2.4**

In Exercises 2.4.1 through 2.4.5, find the indicated vertex-deletion subgraph and edge-deletion subgraph of the graph  $G$ .

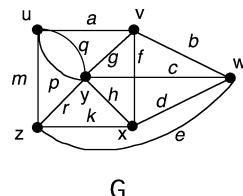
2.4.1<sup>s</sup>  $G - u$ ;  $G - q$ .

2.4.2  $G - y$ ;  $G - e$ .

2.4.3  $G - \{w, z\}$ ;  $G - \{m, p, q, a\}$ .

2.4.4<sup>s</sup>  $G - \{w, v\}$ ;  $G - \{c, a\}$ .

2.4.5  $G - \{w, v, y\}$ ;  $G - \{k, h, d\}$ .



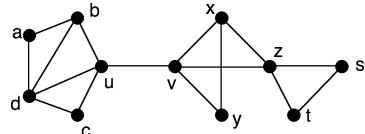
$G$

2.4.6 Draw a list of graphs that represents all possible isomorphism types of the vertex-deletion subgraphs of the Petersen graph (defined in §1.2).

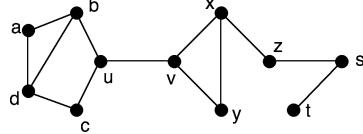
2.4.7 Draw a list of graphs that represents all possible isomorphism types of the edge-deletion subgraphs of the Petersen graph.

In Exercises 2.4.8 through 2.4.11, find all the cut-vertices and cut-edges.

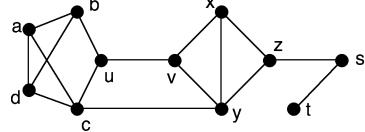
2.4.8



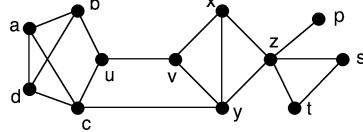
2.4.9



2.4.10



2.4.11



In Exercises 2.4.12 through 2.4.15, determine the largest number of vertices in a minimal vertex-cut for the specified graph, and find one such vertex-cut.

2.4.12<sup>s</sup> The graph of Exercise 2.4.8.

2.4.13 The graph of Exercise 2.4.9.

2.4.14 The graph of Exercise 2.4.10.

2.4.15 The graph of Exercise 2.4.11.

In Exercises 2.4.16 through 2.4.19, determine the largest number of edges in a minimal edge-cut for the specified graph, and find one such edge-cut.

2.4.16<sup>s</sup> The graph of Exercise 2.4.8.

2.4.17 The graph of Exercise 2.4.9.

2.4.18 The graph of Exercise 2.4.10.

2.4.19 The graph of Exercise 2.4.11.

In Exercises 2.4.20 and 2.4.21, find all possible isomorphism types of the given kind of graph.

2.4.20<sup>s</sup> A 4-vertex connected simple graph with no cut-vertices.

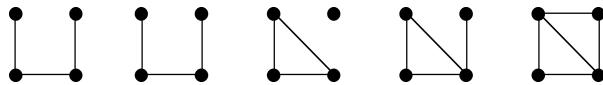
2.4.21 A 4-vertex connected simple graph with no cut-edges.

In Exercises 2.4.22 through 2.4.25, find a graph with the given vertex-deletion subgraph list.

2.4.22<sup>s</sup>



2.4.23



2.4.24



2.4.25



2.4.26 How would you recognize from the reconstruction deck of a graph whether it is bipartite? (Give a proof that your method works.)

2.4.27 How would you recognize from the reconstruction deck of a graph whether it is connected? (Give a proof that your method works.)

2.4.28 Characterize those graphs  $G$  that satisfy the equality  $(G - v) + v = G$  for every vertex  $v$  in  $G$ .

2.4.29 Prove that an edge is a cut-edge of a graph if and only if it is not part of any cycle in that graph.

2.4.30 Let  $e$  be a cut-edge of a connected graph  $G$ . Prove that  $G - e$  has exactly two components.

2.4.31 Let  $v$  be one of the  $n$  vertices of a connected graph  $G$ . Find an upper bound for the number of components of  $G - v$ , and give an example that achieves the upper bound.

2.4.32 Draw a 5-vertex connected graph  $G$  that has no cut-vertices, and then verify that  $G$  satisfies each of the following properties.

- Given any two vertices, there exists a cycle containing both.
- For any vertex  $v$  and any edge  $e$  of  $G$ , there exists a cycle containing  $v$  and  $e$ .
- Given any two vertices  $x$  and  $y$ , and any edge  $e$ , there exists a path from  $x$  to  $y$  that contains  $e$ .
- Given any two edges, there exists a cycle containing both.
- Given any three distinct vertices  $u$ ,  $v$ , and  $w$ , there exists a  $u$ - $v$  path that contains  $w$ .
- Given any three distinct vertices  $u$ ,  $v$ , and  $w$ , there exists a  $u$ - $v$  path that does not contain  $w$ .

2.4.33 Draw a 5-vertex graph  $G$  that has no cut-edges and such that the suspension  $G + v$  has exactly five cut-edges.

2.4.34 Let  $G$  be an  $n$ -vertex graph with no cut-edges. Prove or disprove: The suspension  $G + v$  has exactly  $n$  cut-edges.

2.4.35<sup>s</sup> Prove or disprove: For every graph  $G$ , the suspension  $G + v$  has no cut-vertices.

2.4.36 State a necessary and sufficient condition for an  $n$ -vertex graph  $G$  to be a suspension of some subgraph  $H$  from a vertex in  $V_G - V_H$ .

2.4.37 Give a recursive definition for the complete graph  $K_n$ , using the join operation.

**2.4.38** Prove that for any simple graph  $G$  on 6 vertices, a copy of  $K_3$  appears as a subgraph of either  $G$  or  $\overline{G}$  (or both).

**2.4.39** Prove that no vertex of a graph  $G$  can be a cut-vertex of both  $G$  and  $\overline{G}$ .

**2.4.40<sup>s</sup>** Let  $G$  be a simple graph. Prove that at least one of the graphs  $G$  and  $\overline{G}$  is connected.

## 2.5 TESTS FOR NON-ISOMORPHISM

In §2.1, we derived the most basic necessary conditions for two graphs to be isomorphic: same number of vertices, same number of edges, and same degree sequence. We noticed also that two isomorphic graphs could differ in various artifacts of their representations, including vertex names and some kinds of differences in drawings. And we observed that a brute-force approach of considering all  $n$  possible vertex bijections for two  $n$ -vertex graphs would be too labor-intensive.

The fact is, there is no known short list of easily applied tests to decide for any possible pair of graphs whether they are isomorphic. Thus, the isomorphism problem is formidable, but not all is bleak. We will establish a collection of graph properties that are preserved under isomorphism. These will be used to show that various pairs of graphs are not isomorphic.

**DEFINITION:** A **graph invariant** (or digraph invariant) is a property of graphs (di-graphs) that is preserved by isomorphisms.

**Example 2.5.1:** We established in §2.1 that the number of vertices, the number of edges, and the degree sequence are the same for any two isomorphic graphs, so they are graph invariants.

In this section, we establish several other graph invariants that are useful for isomorphism testing and in the construction of isomorphisms.

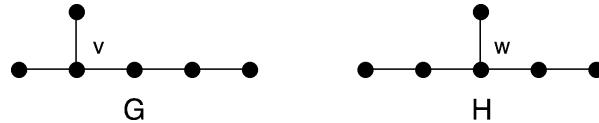
### A Local Invariant

The following theorem provides additional necessary criteria for isomorphism.

**Theorem 2.5.1.** *Let  $f : G \rightarrow H$  be a graph isomorphism, and let  $v \in V_G$ . Then the multiset of degrees of the neighbors of  $v$  equals the multiset of degrees of the neighbors of  $f(v)$ .*

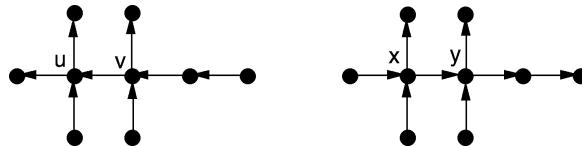
**Proof:** This is an immediate consequence of Corollary 2.1.5. ◊

**Example 2.5.2:** Figure 2.5.1 shows two non-isomorphic graphs with the same number of vertices, the same number of edges, and the same degree sequence:  $\langle 1, 1, 1, 2, 2, 3 \rangle$ . By Theorem 2.1.2, an isomorphism would have to map vertex  $v$  in graph  $G$  to vertex  $w$  in graph  $H$  since they are the only vertices of degree 3 in their respective graphs. However, the three neighbors of  $v$  have degrees 1, 1, and 2, and the continuity rule implies that they would have to be mapped bijectively to the three neighbors of  $w$ , with degrees 1, 2, and 2. This would violate Theorem 2.5.1.



**Figure 2.5.1** Non-isomorphic graphs with the same degree sequence.

**Example 2.5.3:** Although the indegree and outdegree sequences of the digraphs in Figure 2.5.2 are identical, these digraphs are not isomorphic. To see this, first observe that vertices  $u, v, x$ , and  $y$  are the only vertices in their respective digraphs that have indegree 2. Since indegree is a local isomorphism invariant (by a digraph analogy to Theorem 2.5.1),  $u$  and  $v$  must map to  $y$  and  $x$ , respectively. But the directed path of length 3 that ends at  $u$  would have to map to a directed path of length 3 that ends at  $y$ . Since there is no such path, the digraphs are not isomorphic.



**Figure 2.5.2** Non-isomorphic digraphs with identical degree sequences.

### Distance Invariants

**DEFINITION:** Let  $W = \langle v_0, e_1, v_1, \dots, e_n, v_n \rangle$  be a walk in the domain  $G$  of a graph isomorphism (or more generally, under any graph mapping)  $f : G \rightarrow H$ . Then the **image of the walk**  $W$  is the walk  $f(W) = \langle f(v_0), f(e_1), f(v_1), \dots, f(e_n), f(v_n) \rangle$  in graph  $H$ .

**Theorem 2.5.2.** *The isomorphic image of a graph walk is a walk of the same length.*

**Proof:** This follows directly from the edge-multiplicity-preserving property of an isomorphism.  $\diamond$

**Corollary 2.5.3.** *The isomorphic image of a trail, path, or cycle is a trail, path, or cycle, respectively, of the same length.*

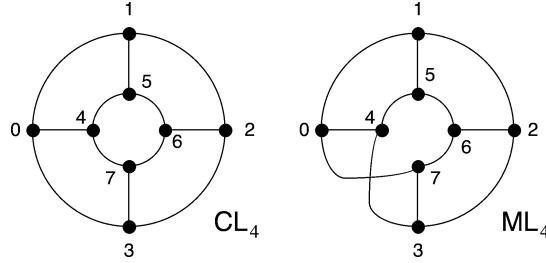
**Proof:** This is an immediate consequence of Theorem 2.5.2 and the bijectivity of the isomorphism.  $\diamond$

**Corollary 2.5.4.** *For each integer  $l$ , two isomorphic graphs must have the same number of trails (paths) (cycles) of length  $l$ .*  $\diamond$

**Corollary 2.5.5.** *The diameter, the radius, and the girth are graph invariants.*

**Proof:** This is an immediate consequence of Corollary 2.5.3 and the bijectivity of the isomorphism.  $\diamond$

**Example 2.5.4:** Figure 2.5.3 shows the circular ladder  $CL_4$  and the Möbius ladder  $ML_4$ , which are 8-vertex graphs. Since both graphs are 3-regular, it is pointless to apply isomorphism tests based on vertex degree.



**Figure 2.5.3** Non-isomorphic graphs with the same degree sequence.

To reduce the calculation of diameter from checking all  $\binom{8}{2} = 28$  vertex pairs to checking the maximum distance from any one vertex, we first establish that both graphs are vertex-transitive. For the circular ladder we observe the symmetries of rotation and the isomorphism that swaps the inner cycle with the outer cycle. For the Möbius ladder, we observe that  $j \mapsto j + 1 \bmod 8$  is an automorphism, whose iteration establishes vertex-transitivity.

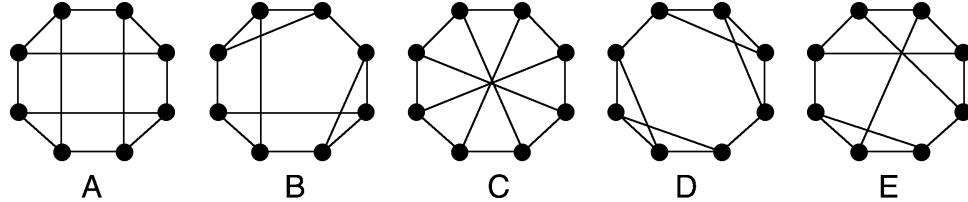
The maximum distance from vertex 0 in  $CL_4$  is 3, for vertex 6. The maximum distance from vertex 0 in  $ML_4$  is 2. Thus, they are not isomorphic. We may also observe that  $\alpha(CL_4) = 4$ , but  $\alpha(ML_4) = 3$ .

### Subgraph Presence

**Theorem 2.5.6.** *For each graph-isomorphism type, the number of distinct subgraphs in a graph having that isomorphism type is a graph invariant.*

**Proof:** Let  $f : G_1 \rightarrow G_2$  be a graph isomorphism and  $H$  a subgraph of  $G_1$ . Then  $f(H)$  is a subgraph of  $G_2$ , of the same isomorphism type as  $H$ . The bijectivity of  $f$  establishes the invariant.  $\diamond$

**Example 2.5.5:** The five graphs in Figure 2.5.4 are all 3-regular, even though they have the same degree sequence. Whereas  $A$  and  $C$  have no  $K_3$  subgraphs,  $B$  has two,  $D$  has four, and  $E$  has one. Thus, Theorem 2.5.6 implies that the only possible isomorphic pair is  $A$  and  $C$ . However, graph  $C$  has a 5-cycle, but graph  $A$  does not, because it is bipartite.



**Figure 2.5.4** Five mutually non-isomorphic, 8-vertex, 3-regular graphs.

Alternatively, we may observe that graphs  $A$  and  $C$  are the only pair with the same multiset of local subgraphs. We could distinguish this pair by observing that  $\alpha(A) = 4$  and  $\alpha(C) = 3$ .

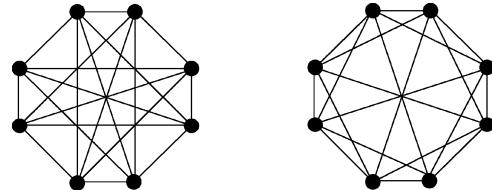
### Edge-Complementation

The next invariant is particularly useful when analyzing simple graphs that are *dense* (i.e., most of the vertex pairs are adjacent).

**Theorem 2.5.7.** Let  $G$  and  $H$  both be simple graphs. They are isomorphic if and only if their edge-complements are isomorphic.

**Proof:** By definition, a graph isomorphism necessarily preserves non-adjacency as well as adjacency.  $\diamond$

**Example 2.5.6:** The two graphs in Figure 2.5.5 are relatively dense, simple graphs (both with 20 out of 28 possible edges). The edge-complement of the left graph consists of two disjoint 4-cycles, and the edge-complement of the right graph is an 8-cycle. Since these edge-complements are non-isomorphic, the original graphs must be non-isomorphic.



**Figure 2.5.5** Two relatively dense, non-isomorphic 5-regular graphs.

### Summary

The following table summarizes these results on graph-isomorphism invariants. It is straightforward to establish digraph versions (see Exercises).

---

**Table 2.5.1** Some graph invariants.

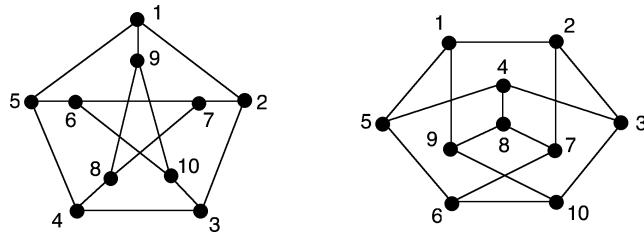
1. The number of vertices
  2. The number of edges
  3. The degree sequence
  4. The multiset of local graphs
  5. Degrees of neighbors of a forced match
  6. Diameter, radius, girth
  7. Independence number, clique number
  8. For any possible subgraph, the number of distinct copies
  9. For a simple graph, the edge-complement
- 

### Using Invariants to Construct an Isomorphism

The last example in this section illustrates how invariants can guide the construction of an isomorphism.

**Example 2.5.7:** The two graphs in Figure 2.5.6 both have open paths of length 9, indicated by consecutive vertex numbering  $1, \dots, 10$ . By Corollary 2.5.3, any isomorphism must map the length-9 path in the left graph to some length-9 path in the right graph. The label-preserving bijection that maps the left length-9 path to the right length-9 path is a reasonable candidate for such an isomorphism. Adjacency is clearly preserved for the nine pairs of consecutively numbered vertices,  $(i, i + 1)$ , since they represent edges along the paths. Also, it is easy to see that adjacency is preserved for the pairs of vertices that are not consecutively numbered, since each of the six remaining edges

in the left graph has a corresponding edge in the right graph with matching endpoints. Thus, the label-preserving bijection is an isomorphism.



**Figure 2.5.6** Two copies of the Petersen graph.

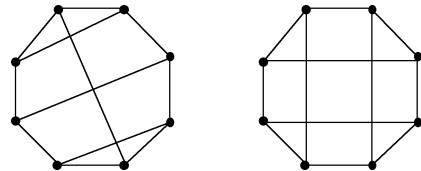
### EXERCISES for Section 2.5

**2.5.1<sup>s</sup>** For each of the following graphs, either show that it is isomorphic to one of the five regular graphs of Example 2.5.5, or argue why it is not.

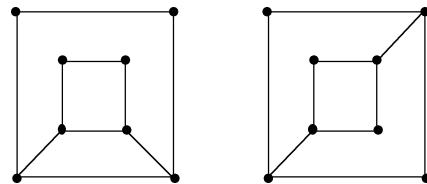


*In Exercises 2.5.2 and 2.5.3, explain why the graphs in the given graph pair are not isomorphic.*

**2.5.2<sup>s</sup>**



**2.5.3**

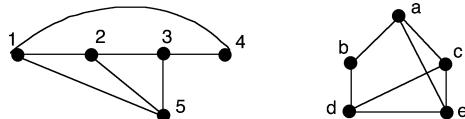


*In Exercises 2.5.4 through 2.5.11, determine whether the graphs in the given pair are isomorphic.*

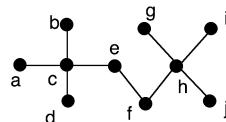
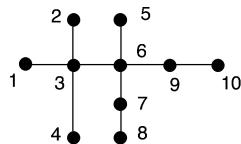
**2.5.4**



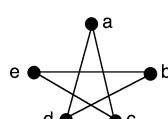
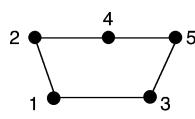
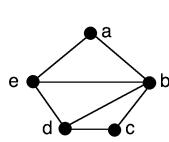
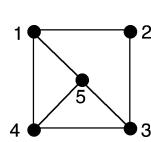
**2.5.5**



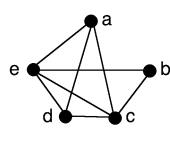
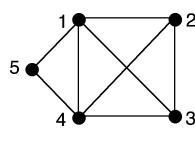
2.5.6



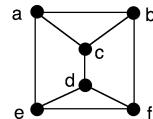
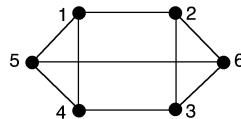
2.5.7

2.5.8<sup>s</sup>

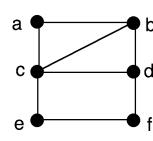
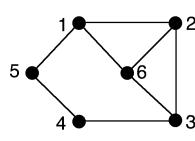
2.5.9



2.5.10



2.5.11



**2.5.12** Suppose that  $f : G \rightarrow H$  is a graph isomorphism. Establish each of the following isomorphism properties.

- a. For each  $v \in V_G$ , the local subgraph of  $v$  is isomorphic to the local subgraph of  $f(v)$ .
- b. For each  $v \in V_G$ ,  $G - v$  is isomorphic to  $H - f(v)$ .

*In Exercises 2.5.13 through 2.5.18, state and prove digraph versions of the specified assertion.*

2.5.13<sup>s</sup> Theorem 2.5.1.

2.5.14 Theorem 2.5.2.

2.5.15 Corollary 2.5.4.

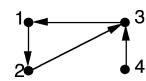
2.5.16 Corollary 2.5.5.

2.5.17 Theorem 2.5.6.

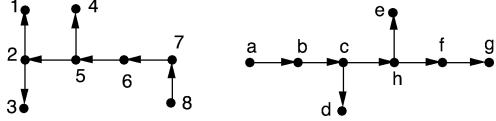
2.5.18 Theorem 2.5.7.

**2.5.19** Compile a table of digraph-isomorphism invariants analogous to Table 2.5.1.

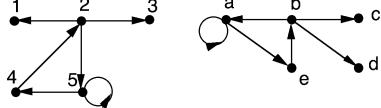
*In Exercises 2.5.20 through 2.5.25, determine whether the digraphs in the given pair are isomorphic.*

2.5.20<sup>s</sup>

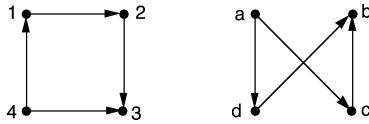
2.5.21



2.5.22



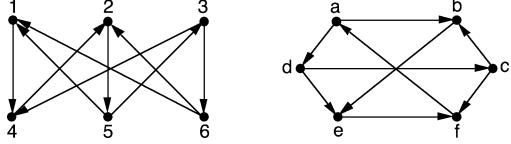
2.5.23



2.5.24



2.5.25



## 2.6 MATRIX REPRESENTATIONS

Representing graphs by matrices has conceptual and theoretical importance. It helps bring the power of linear algebra to graph theory.

### Adjacency Matrices

**DEFINITION:** The **adjacency matrix of a simple graph**  $G$ , denoted  $A_G$ , is the symmetric matrix whose rows and columns are both indexed by identical orderings of  $V_G$ , such that

$$A_G[u, v] = \begin{cases} 1 & \text{if } u \text{ and } v \text{ are adjacent} \\ 0 & \text{otherwise} \end{cases}$$

**Example 2.6.1:** Figure 2.6.1 shows the adjacency matrix of a graph, with respect to the vertex ordering  $u, v, w, x$ .

$$A_G = \begin{pmatrix} u & v & w & x \\ u & 0 & 1 & 1 & 0 \\ v & 1 & 0 & 1 & 0 \\ w & 1 & 1 & 0 & 1 \\ x & 0 & 0 & 1 & 0 \end{pmatrix}$$

**Figure 2.6.1** A graph and its adjacency matrix.

Usually the vertex order is implicit from context, in which case the adjacency matrix  $A_G$  can be written as a matrix without explicit row or column labels.

**Proposition 2.6.1.** Let  $G$  be a graph with adjacency matrix  $A_G$ . Then the value of element  $A_G^r[u, v]$  of the  $r^{th}$  power of matrix  $A_G$  equals the number of  $u-v$  walks of length  $r$ .

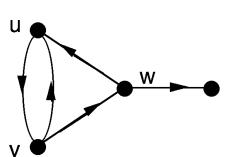
**Proof:** The assertion holds for  $r = 1$ , by the definition of the adjacency matrix and the fact that walks of length 1 are the edges of the graph. The inductive step follows from the definition of matrix multiplication (see Exercises).  $\diamond$

The adjacency matrix of a simple digraph and the relationship between its powers and the number of directed walks in the digraph is analogous to the result for graphs.

**DEFINITION:** The **adjacency matrix of a simple digraph**  $D$ , denoted  $A_D$ , is the matrix whose rows and columns are both indexed by identical orderings of  $V_G$ , such that

$$A_D[u, v] = \begin{cases} 1 & \text{if there is a edge from } u \text{ to } v \\ 0 & \text{otherwise} \end{cases}$$

**Example 2.6.2:** The adjacency matrix of the digraph in Figure 2.6.2 uses the vertex ordering  $u, v, w, x$ .



$$A_D = \begin{matrix} & \begin{matrix} u & v & w & x \end{matrix} \\ \begin{matrix} u \\ v \\ w \\ x \end{matrix} & \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{matrix}$$

**Figure 2.6.2** A digraph and its adjacency matrix.

**Proposition 2.6.2.** Let  $D$  be a digraph with  $V_D = v_1, v_2, \dots, v_n$ . Then the sum of the elements of row  $i$  of the adjacency matrix  $A_D$  equals the outdegree of vertex  $v_i$ , and the sum of the elements of column  $j$  equals the indegree of vertex  $v_j$ .  $\diamond$  (Exercises)

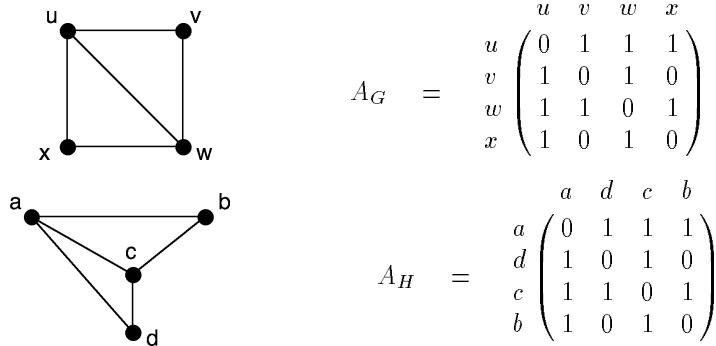
**Proposition 2.6.3.** Let  $D$  be a digraph with adjacency matrix  $A_D$ . Then the value of the entry  $A_D^r[u, v]$  of the  $r^{th}$  power of matrix  $A_D$  equals the number of directed  $u-v$  walks of length  $r$ .  $\diamond$  (Exercises)

**Remark:** To extend the definition of adjacency matrix to a general graph (or digraph), one lets  $A_G[u, v]$  equal the number of edges between vertex  $u$  and vertex  $v$  (or from vertex  $u$  to vertex  $v$ ).

### Brute-Force Graph-Isomorphism Testing

The definition of graph isomorphism implies that two graphs are isomorphic if it is possible to order their respective vertex-sets so that their adjacency matrices are identical.

**Example 2.6.3:** The graphs in Figure 2.6.3 are isomorphic, because under the orderings  $u, v, w, x$  and  $a, d, c, b$ , they have identical adjacency matrices.



**Figure 2.6.3** Establishing isomorphism using adjacency matrices.

**Algorithm 2.6.1: Brute-force test for graph isomorphism**

*Input:* graphs  $G$  and  $H$ .

*Output:* Return YES or NO, according to whether  $G$  is isomorphic to  $H$ .

If  $|V_G| \neq |V_H|$

    Return NO.

If degree sequences are not equal

    Return NO.

Fix a vertex ordering for graph  $G$ .

Write the adjacency matrix  $A_G$  with respect to that ordering.

For each vertex ordering  $\tau$  of graph  $H$

    Write  $A_H$  with respect to ordering  $\tau$

    If  $A_H$  (w.r.t.  $\tau$ ) =  $A_G$

        Return YES.

Return NO.

**COMPUTATIONAL NOTE:** For all but the smallest graphs, this exhaustive method is hopelessly inefficient. This should not be surprising in light of the discussion of the *isomorphism problem* in §2.1.

### Incidence Matrices for Undirected Graphs

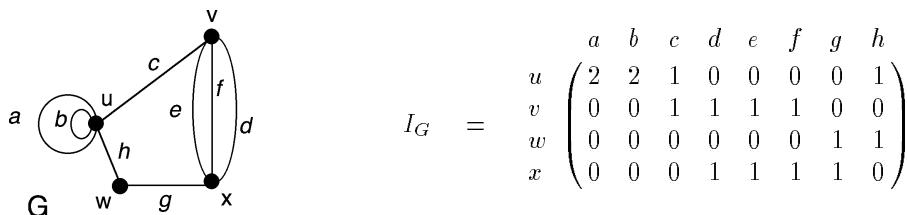
**DEFINITION:** The **incidence matrix** of a graph  $G$  is the matrix  $I_G$  whose rows and columns are indexed by some orderings of  $V_G$  and  $E_G$ , respectively, such that

$$I_G[v, e] = \begin{cases} 0 & \text{if } v \text{ is not an endpoint of } e \\ 1 & \text{if } v \text{ is an endpoint of } e \\ 2 & \text{if } e \text{ is a self-loop at } v \end{cases} \dagger$$

---

<sup>†</sup> An alternative definition seen elsewhere, which uses 1 for a self-loop instead of 2, takes more effort to find the self-loops. It also has the theoretical inconsistency that row-sums are not necessarily equal to vertex-degrees and column-sums are not necessarily equal to 2.

**Example 2.6.4:** Figure 2.6.4 shows a graph and its incidence matrix.



**Figure 2.6.4** A graph and its incidence matrix.

Observe that changing the orderings of  $V_G$  and  $E_G$  permutes the rows and columns of  $I_G$ . The next two propositions follow immediately from the definition of the incidence matrix.

**Proposition 2.6.4.** *The sum of the entries in any row of an incidence matrix is the degree of the corresponding vertex.*  $\diamondsuit$  (Exercises)

**Proposition 2.6.5.** *The sum of the entries in any column of an incidence matrix is equal to 2.*  $\diamondsuit$  (Exercises)

These two propositions lead to another simple proof of Euler's theorem that the sum of the degrees equals twice the number of edges. In particular, the sum of the degrees is simply the sum of the row-sums of the incidence matrix, and the sum of the column-sums equals twice the number of edges. The result follows since these two sums both equal the sum of all the entries of the matrix.

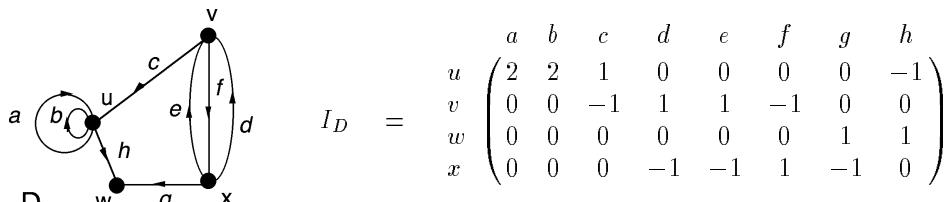
### Incidence Matrices for Digraphs

The incidence matrices for digraphs are analogously defined.

**DEFINITION:** The **incidence matrix** of a digraph  $D$  is the matrix whose rows and columns are indexed by some orderings of  $V_D$  and  $E_D$ , respectively, such that

$$I_D[v, e] = \begin{cases} 0 & \text{if } v \text{ is not an endpoint of } e \\ 1 & \text{if } v \text{ is the head of } e \\ -1 & \text{if } v \text{ is the tail of } e \\ 2 & \text{if } e \text{ is a self-loop at } v \end{cases}$$

**Example 2.6.5:** Figure 2.6.5 shows a digraph and its incidence matrix.



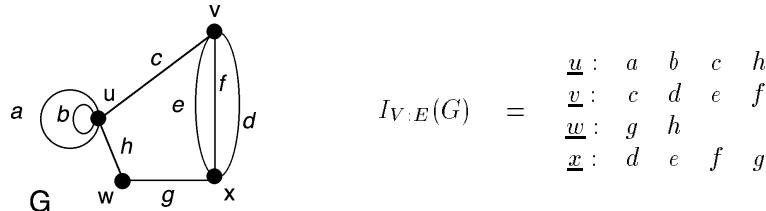
**Figure 2.6.5** A digraph and its incidence matrix.

### Using Incidence Tables to Save Space

The incidence matrix and the adjacency matrix of most graphs have the undesirable feature of consisting mostly of zeros. The following definition gives an alternative representation of a graph that is more space-efficient.

**DEFINITION:** The **table of incident edges** for a graph  $G$  is an incidence table that lists, for each vertex  $v$ , all the edges incident on  $v$ . This table is denoted  $I_{V:E}(G)$ .

**Example 2.6.6:** The table of incident edges for the graph of Example 2.6.4 (repeated below) is as follows:



**COMPUTATIONAL NOTE:** A “big  $O$ ” comparison of the space required by the three types of graph representation reveals the advantage that a table of incident edges holds over the two matrix representations.<sup>†</sup> In particular, the total space requirements of the incidence matrix, the adjacency matrix, and the table of incident-edges are, respectively,  $O(|V| \cdot |E|)$ ,  $O(|V|^2)$ , and  $O(|E|)$ .

**DEFINITION:** For a digraph  $D$ , the **table of outgoing arcs**, denoted  $out_{V:E}(D)$ , lists, for each vertex  $v$ , all arcs that are directed from  $v$ . The **table of incoming arcs**, denoted  $in_{V:E}(D)$ , is defined similarly.

**Example 2.6.7:** Here are  $in_{V:E}(D)$  and  $out_{V:E}(D)$  for the digraph in Figure 2.6.5.

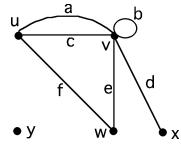
$$in_{V:E}(G) = \begin{array}{l} \underline{u} : a \quad b \quad c \\ \underline{v} : d \quad e \\ \underline{w} : g \quad h \\ \underline{x} : f \end{array} \quad out_{V:E}(G) = \begin{array}{l} \underline{u} : a \quad b \quad h \\ \underline{v} : c \quad f \\ \underline{w} : \\ \underline{x} : d \quad e \quad g \end{array}$$

**Remark:** Since software requirements vary from application to application, it is not possible to prescribe a universally optimal representation of the incidence structure of a graph. The objective here is to introduce software-performance criteria along with the environment required for achieving good performance.

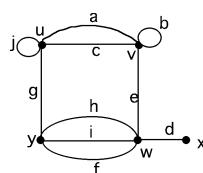
### EXERCISES for Section 2.6

In Exercises 2.6.1 through 2.6.3, determine the matrices  $A_G$ ,  $I_G$ , and the table  $I_{V:E}(G)$  for the given graph  $G$ .

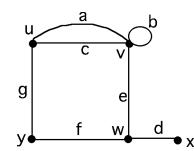
2.6.1<sup>s</sup>



2.6.2

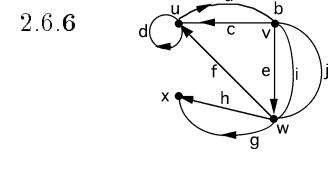
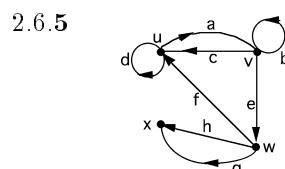
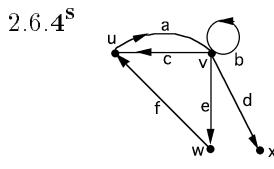


2.6.3



<sup>†</sup> A brief discussion of algorithmic complexity and “big  $O$ ” notation appears in Appendix A.5.

In Exercises 2.6.4 through 2.6.6, determine the matrices  $A_D$ ,  $I_D$ ,  $\text{out}_{V:E}(D)$ , and the table  $\text{in}_{V:E}(D)$  for the given digraph  $D$ .



In Exercises 2.6.7 and 2.6.8, draw a graph that has the given adjacency matrix.

2.6.7

$$A_G = \begin{pmatrix} a & b & c & d \\ a & 0 & 1 & 1 & 1 \\ b & 1 & 0 & 1 & 0 \\ c & 1 & 1 & 0 & 1 \\ d & 1 & 0 & 1 & 0 \end{pmatrix}$$

2.6.8

$$A_G = \begin{pmatrix} a & b & c & d \\ a & 2 & 1 & 1 & 1 \\ b & 1 & 0 & 1 & 0 \\ c & 1 & 1 & 0 & 1 \\ d & 1 & 0 & 1 & 0 \end{pmatrix}$$

In Exercises 2.6.9 through 2.6.13, describe the adjacency matrix of each of the following graph families.

2.6.9<sup>s</sup>  $K_{m,n}$     2.6.10  $K_n$     2.6.11  $P_n$     2.6.12  $C_n$     2.6.13  $Q_n$

In Exercises 2.6.14 and 2.6.15, draw a digraph that has the given adjacency matrix.

2.6.14<sup>s</sup>

$$A_D = \begin{pmatrix} a & b & c & d \\ a & 0 & 1 & 1 & 1 \\ b & 0 & 0 & 0 & 0 \\ c & 0 & 1 & 0 & 0 \\ d & 0 & 0 & 1 & 0 \end{pmatrix}$$

2.6.15

$$A_D = \begin{pmatrix} a & b & c & d \\ a & 0 & 1 & 1 & 1 \\ b & 0 & 0 & 1 & 0 \\ c & 1 & 1 & 0 & 1 \\ d & 1 & 0 & 1 & 0 \end{pmatrix}$$

In Exercises 2.6.16 and 2.6.17, verify that Proposition 2.6.1 holds for the entries  $A_G^2[a, a]$  and  $A_G^2[d, a]$  for the indicated graph and adjacency matrix.

2.6.16<sup>s</sup> The graph and adjacency matrix of Exercise 2.6.7.

2.6.17 The graph and adjacency matrix of Exercise 2.6.8.

In Exercises 2.6.18 and 2.6.19, verify that Proposition 2.6.3 holds for the entries  $A_D^2[a, b]$  and  $A_D^2[d, a]$  for the indicated digraph and adjacency matrix.

2.6.18<sup>s</sup> The digraph and adjacency matrix of Exercise 2.6.14.

2.6.19 The digraph and adjacency matrix of Exercise 2.6.15.

In Exercises 2.6.20 and 2.6.21, draw a graph that has the given incidence matrix.

2.6.20<sup>s</sup>

$$I_G = \begin{pmatrix} a & b & c & d & e \\ u & 0 & 1 & 0 & 0 & 0 \\ v & 1 & 0 & 2 & 1 & 1 \\ w & 1 & 0 & 0 & 0 & 0 \\ x & 0 & 1 & 0 & 1 & 1 \end{pmatrix}$$

2.6.21

$$I_G = \begin{pmatrix} a & b & c & d & e \\ u & 1 & 2 & 1 & 0 & 0 \\ v & 0 & 0 & 1 & 1 & 1 \\ w & 0 & 0 & 0 & 0 & 0 \\ x & 1 & 0 & 0 & 1 & 1 \end{pmatrix}$$

In Exercises 2.6.22 and 2.6.23, draw a digraph that has the given incidence matrix.

2.6.22<sup>s</sup>

$$I_D = \begin{pmatrix} a & b & c & d & e \\ u & 1 & 2 & 1 & 0 & 0 \\ v & 0 & 0 & -1 & 1 & -1 \\ w & 0 & 0 & 0 & 0 & 0 \\ x & -1 & 0 & 0 & -1 & 1 \end{pmatrix}$$

2.6.23

$$I_D = \begin{pmatrix} a & b & c & d & e \\ u & -1 & 0 & 1 & 0 & 0 \\ v & 0 & 0 & 0 & 0 & 1 \\ w & 0 & 0 & -1 & 1 & 0 \\ x & 1 & 2 & 0 & -1 & -1 \end{pmatrix}$$

- 2.6.24 Complete the proof of Proposition 2.6.1 by establishing the inductive step.
- 2.6.25 Prove Proposition 2.6.2.
- 2.6.26 Prove Proposition 2.6.3.
- 2.6.27 Prove Proposition 2.6.4.
- 2.6.28 Prove Proposition 2.6.5.
- 2.6.29 [Computer Project] Implement Algorithm 2.6.1 and test it on at least four of the graph pairs in Exercises 2.5.4 through 2.5.11.

## 2.7 MORE GRAPH OPERATIONS

The graph-theoretic binary operations of product and join are constructed by iteratively applying the primary operations of adding or deleting vertices and edges. Thus, in a computer-science sense, both these operations may be regarded as secondary operations. The operation of product is adapted from set theory, but the join operation has no set-theoretic counterpart. Another binary operation of interest is amalgamation, which means pasting two graphs together.

### Cartesian Product

**DEFINITION:** The (*cartesian*) **product**  $G \times H$  of the graphs  $G$  and  $H$  has as its vertex-set the cartesian product

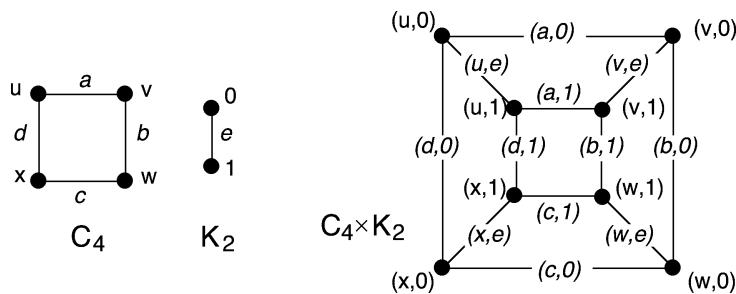
$$V_{G \times H} = V_G \times V_H$$

and as its edges a union of two products:

$$E_{G \times H} = (V_G \times E_H) \cup (E_G \times V_H)$$

The endpoints of the edge  $(u, d)$  are the vertices  $(u, x)$  and  $(u, y)$ , where  $x$  and  $y$  are the endpoints of edge  $d$  in graph  $H$ . The endpoints of the edge  $(e, w)$  are  $(u, w)$  and  $(v, w)$ , where  $u$  and  $v$  are the endpoints of edge  $e$  in graph  $G$ .

**Example 2.7.1:** Figure 2.7.1 illustrates the cartesian product of the 4-cycle graph  $C_4$  and the complete graph  $K_2$ . Observe that  $C_4 \times K_2$  is isomorphic to the hypercube graph  $Q_3$  defined in §1.2.



**Figure 2.7.1** The labeled cartesian product  $C_4 \times K_2$ .

**Example 2.7.2:** Figure 2.7.2 illustrates the cartesian product of the complete graph  $K_3$  and the path graph  $P_3$ .

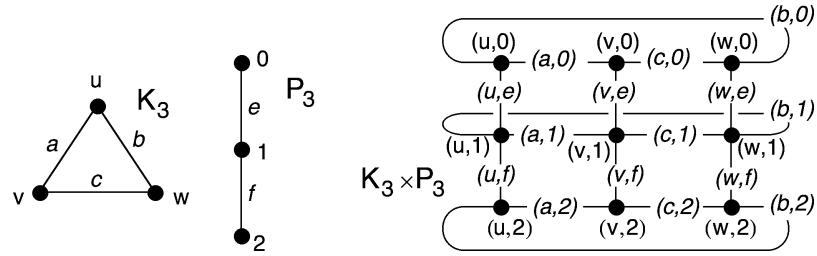


Figure 2.7.2 The labeled cartesian product  $K_3 \times P_3$ .

Often there are many ways to visualize a spatial model of the product of two graphs. For instance, two alternative ways to draw the product  $K_3 \times P_3$  would be as three concentric triangles, or as a row of triangles, both illustrated in Figure 2.7.3.

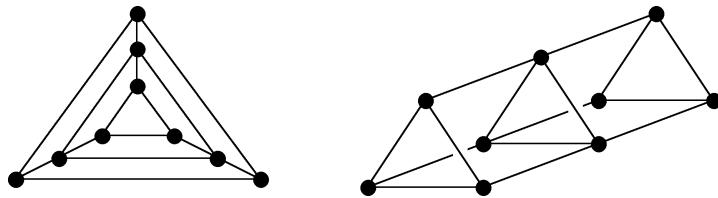


Figure 2.7.3 Two alternative views of  $K_3 \times P_3$ .

**DEFINITION:** The product  $K_2 \times C_n$  is called a **circular ladder with  $n$  rungs** and often denoted  $CL_n$ .

**Remark:** This formal definition is consistent with the informal definition of circular ladders in §1.2.

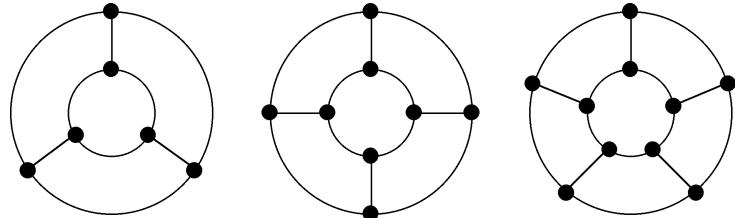
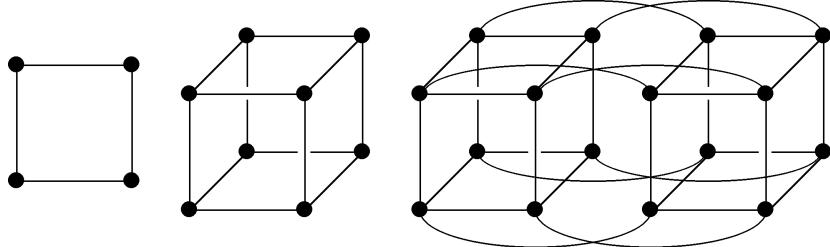


Figure 2.7.4 The circular ladders  $CL_3$ ,  $CL_4$ , and  $CL_5$ .

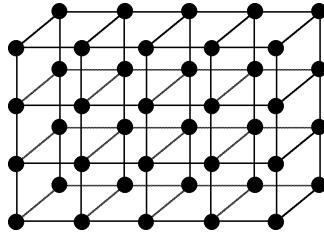
**DEFINITION:** The iterated product  $K_2 \times \cdots \times K_2$  of  $n$  copies of  $K_2$  is called either the **hypercube graph of dimension  $n$**  or the  **$n$ -hypercube graph**. It is denoted  $Q_n$ .

**Remark:** The 3-hypercube graph is isomorphic to the 1-skeleton of the cube, which was considered in §1.2.



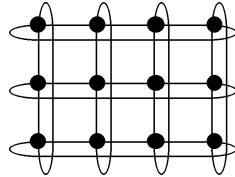
**Figure 2.7.5** The hypercube graphs  $Q_2$ ,  $Q_3$ , and  $Q_4$ .

DEFINITION: The  $m_1 \times m_2 \times \cdots \times m_n$ -mesh is the iterated product  $P_{m_1} \times P_{m_2} \times \cdots \times P_{m_n}$  of paths.



**Figure 2.7.6** A  $5 \times 4 \times 2$  mesh.

DEFINITION: The **wraparound**  $m_1 \times m_2 \times \cdots \times m_n$ -mesh is the iterated product  $C_{m_1} \times C_{m_2} \times \cdots \times C_{m_n}$  of cycles.

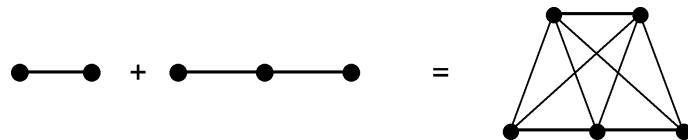


**Figure 2.7.7** A wraparound  $4 \times 3$  mesh.

**Remark:** The product operation is both commutative and associative.

### Join

DEFINITION: The **join**  $G + H$  of the graphs  $G$  and  $H$  is obtained from the graph union  $G \cup H$  by adding an edge between each vertex of  $G$  and each vertex of  $H$ .



**Figure 2.7.8** The join  $P_2 + P_3$ .

**Example 2.7.3:** The join  $K_m + K_n$  is isomorphic to the complete graph  $K_{m+n}$ . To see this, consider any two vertices in  $K_m + K_n$ . If both are from  $K_m$  or if both are from  $K_n$ , then they are obviously still adjacent in  $K_m + K_n$ . Moreover, if one is from  $K_m$  and the other from  $K_n$ , then the join construction places an edge between them.

**Example 2.7.4:** The join  $mK_1 + nK_1$  is isomorphic to the complete bipartite graph  $K_{m,n}$ . One part of the bipartition is the vertices from  $mK_1$  and the other part is the vertices from  $nK_1$ .

DEFINITION: The  **$n$ -dimensional octahedral graph**  $\mathcal{O}_n$ , is defined recursively, using the join operation.

$$\mathcal{O}_n = \begin{cases} 2K_1 & \text{if } n = 0 \\ 2K_1 + \mathcal{O}_{n-1} & \text{if } n \geq 1 \end{cases}$$

It is also called the  **$n$ -octahedron graph** or, when  $n = 3$ , the **octahedral graph**, because it is the 1-skeleton of the octahedron, a platonic solid.

**Example 2.7.5:** Figure 2.7.9 illustrates the first four octahedral graphs.

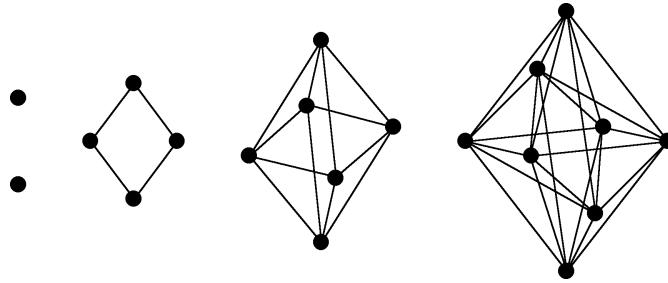


Figure 2.7.9 The octahedral graphs  $\mathcal{O}_1, \mathcal{O}_2, \mathcal{O}_3$ , and  $\mathcal{O}_4$ .

**Example 2.7.6:** Figure 2.7.10 illustrates that the edge-complement of the graph  $nK_2$  (in  $K_{2n}$ ) is isomorphic to  $\mathcal{O}_n$ . This follows recursively, because the two non-adjacent vertices from  $2K_1$  and the  $n - 1$  pairs of non-adjacent vertices from  $\mathcal{O}_{n-1}$  remain non-adjacent in the join  $2K_1 + \mathcal{O}_{n-1}$ , yielding  $n$  pairs of non-adjacent vertices in  $2K_1 + \mathcal{O}_{n-1}$ . The joining construction guarantees that apart from these  $n$  non-adjacencies, all other vertices are adjacent.

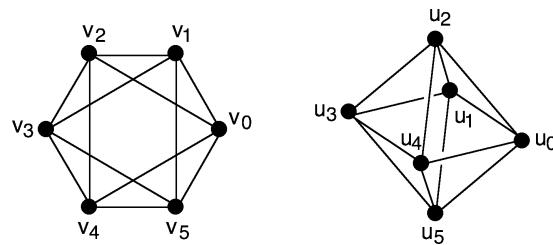


Figure 2.7.10 The graph  $K_6 - 3K_2$  is isomorphic to the octahedral graph  $\mathcal{O}_3$ .

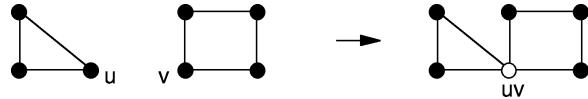
### Amalgamations

One way to construct new graphs is to paste together a few standard recognizable “graph parts”, which become subgraphs of the resulting big new graph. Pasting on vertices leads directly to pasting on arbitrary subgraphs, via an isomorphism between them.

DEFINITION: Let  $G$  and  $H$  be disjoint graphs, with  $u \in V_G$  and  $v \in V_H$ . The **vertex amalgamation**  $(G \cup H)/\{u = v\}$  is the graph obtained from the union  $G \cup H$  by merging (or amalgamating) vertex  $u$  of graph  $G$  and vertex  $v$  of graph  $H$  into a single

vertex, called  $uv$ .<sup>†</sup> The vertex-set of this new graph is  $(V_G - \{u\}) \cup (V_H - \{v\}) \cup \{uv\}$ , and the edge-set is  $E_G \cup E_H$ , except that any edge that had  $u$  or  $v$  as an endpoint now has the amalgamated vertex  $uv$  as an endpoint instead.

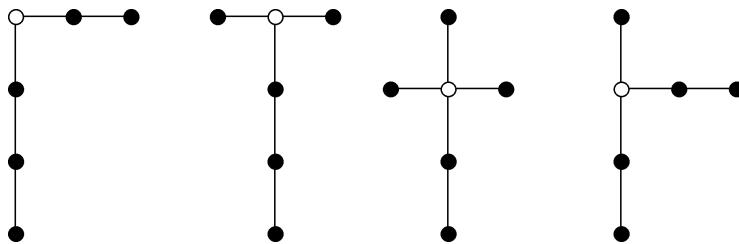
**Example 2.7.7:** Figure 2.7.11 illustrates an amalgamation of a 3-cycle and a 4-cycle, in which vertex  $u$  of the 3-cycle is identified with vertex  $v$  of the 4-cycle.



**Figure 2.7.11** A vertex amalgamation of a 3-cycle and a 4-cycle.

In this example, no matter which vertices are chosen in the 3-cycle and the 4-cycle, respectively, the isomorphism type of the amalgamated graph is the same. This is due to the symmetry of the two cycles.

**Example 2.7.8:** Figure 2.7.12 illustrates the four different isomorphism types of graphs that can be obtained by amalgamating  $P_3$  and  $P_4$  at a vertex.



**Figure 2.7.12** Four different vertex amalgamations of  $P_3$  and  $P_4$ .

When pasting on an arbitrary pair of isomorphic subgraphs, the isomorphism type of the resulting graph may depend on exactly how the vertices and edges of the two subgraphs are matched together. The matching of subgraph to subgraph is achieved by an isomorphism.

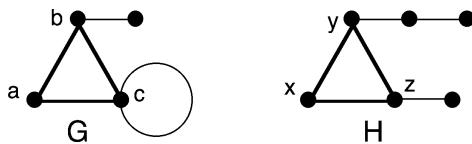
**DEFINITION:** Let  $G$  and  $H$  be disjoint graphs, with  $X$  a subgraph of  $G$  and  $Y$  a subgraph of  $H$ . Let  $f : X \rightarrow Y$  be an isomorphism between these subgraphs. The **amalgamation of  $G$  and  $H$  modulo the isomorphism  $f : X \rightarrow Y$**  is the graph obtained from the union  $G \cup H$  by merging each vertex  $u$  and each edge  $e$  of subgraph  $X$  with their images  $f(u)$  and  $f(e)$  in subgraph  $Y$ . The amalgamated vertex is generically denoted  $uf(u)$ , and the amalgamated edge is generically denoted  $ef(e)$ . The vertex-set of this new graph is  $(V_G - V_X) \cup (V_H - V_Y) \cup \{uf(u) \mid u \in V_X\}$ , and the edge-set is  $(E_G - E_X) \cup (E_H - E_Y) \cup \{ef(e) \mid e \in E_X\}$ , except that any edge that had  $u \in V_X$  or  $f(u) \in V_Y$  as an endpoint now has the amalgamated vertex  $uf(u)$  as an endpoint instead. This general amalgamation is denoted  $(G \cup H)/f : X \rightarrow Y$ .

**DEFINITION:** In an amalgamated graph  $(G \cup H)/f : X \rightarrow Y$ , the image of the pasted subgraphs  $X$  and  $Y$  is called the **subgraph of amalgamation**.

---

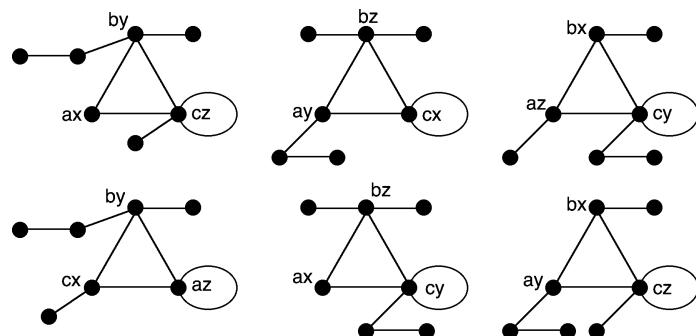
<sup>†</sup> In other contexts, the juxtaposition notation  $uv$  often denotes an edge with endpoints  $u$  and  $v$ . In this section, we use juxtaposition only for amalgamation.

**Example 2.7.9:** Each of the six different isomorphisms from the 3-cycle in graph  $G$  to the 3-cycle in graph  $H$  in Figure 2.7.13 leads to a different amalgamated graph.



**Figure 2.7.13** Two graphs that each contain a 3-cycle.

The six different possible results are illustrated in Figure 2.7.14.



**Figure 2.7.14** Six different possible amalgamated graphs.

**Remark:** The operation of amalgamating graphs  $G$  and  $H$  by pasting vertex  $u \in V_G$  to vertex  $v \in V_H$  is equivalent to amalgamating the graphs  $G$  and  $H$  modulo an isomorphism from the one-vertex subgraph  $u$  to the one-vertex subgraph  $v$ .

### EXERCISES for Section 2.7

For Exercises 2.7.1 through 2.7.4, draw the indicated cartesian product.

2.7.1<sup>s</sup>  $K_2 \times C_5$ .

2.7.2  $P_3 \times C_5$ .

2.7.3  $W_5 \times P_3$ .

2.7.4  $P_3 \times 2K_3$ .

2.7.5 Prove that the cartesian product of two bipartite graphs is always a bipartite graph.

2.7.6 Prove that the cartesian product  $C_4 \times C_4$  of two 4-cycles is isomorphic to the hypercube graph  $Q_4$ .

For Exercises 2.7.7 through 2.7.10, draw the indicated join.

2.7.7<sup>s</sup>  $K_3 + K_3$ .

2.7.8  $B_2 + K_4$ .

2.7.9  $W_6 + P_2$ .

2.7.10  $P_3 + K_3 + C_2$ .

2.7.11 Give a necessary and sufficient condition that the join of two bipartite graphs be non-bipartite.

For Exercises 2.7.12 through 2.7.15, draw all the different isomorphism types of graph that can be obtained by a vertex amalgamation of the indicated graphs.

2.7.12<sup>s</sup>  $P_3$  and  $W_4$ .

2.7.13  $P_3$  and  $P_5$ .

2.7.14  $K_4 - K_2$  and  $P_3$ .

2.7.15  $K_4 - P_3$  and  $C_3$ .

For Exercises 2.7.16 through 2.7.19, draw all the different isomorphism types of graph that can be obtained by an amalgamation across an edge (i.e., formally, across a  $K_2$ -subgraph) of the indicated graphs.

2.7.16<sup>s</sup>  $P_3$  and  $W_4$ .

2.7.18  $K_4 - K_2$  and  $P_3$ .

2.7.17  $P_3$  and  $P_5$ .

2.7.19  $K_4 - P_3$  and  $C_3$ .

For Exercises 2.7.20 through 2.7.23, draw all the different isomorphism types of graph that can be obtained by an amalgamation across a pair of vertices of the indicated graphs.

2.7.20<sup>s</sup>  $P_3$  and  $W_4$ .

2.7.22  $K_4 - K_2$  and  $P_3$ .

2.7.21  $P_3$  and  $P_5$ .

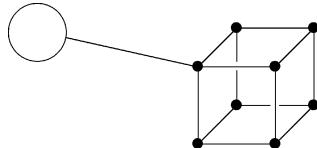
2.7.23  $K_4 - P_3$  and  $C_3$ .

2.7.24 It is possible to amalgamate  $2C_4$  to  $2C_4$  modulo an isomorphism into the circular ladder  $CL_4$ . Describe the subgraph of amalgamation.

2.7.25 It is possible to amalgamate the join  $K_1 + P_r$  to the join  $2K_1 + P_s$  modulo an isomorphism into the wheel  $W_{r+s-2}$ . Draw a representation of this amalgamation for  $r = 3$  and  $s = 4$ , clearly identifying all parts of the drawing.

2.7.26 Prove that  $K_{2n} - 2K_n$  is isomorphic to  $K_{n,n}$ .

2.7.27 As an entertaining way to obtain a physical model of  $Q_4$ , construct from flexible wires (such as soft paper clips) a frame in the shape of the cube graph  $Q_3$ , with an attached “wand”, as illustrated.



Then dip the frame into a solution of water and high quality liquid dish soap. A cubic bubble will cling to the frame, and inside there will appear a smaller cube, attached only by sheets of soapfilm surface to the outer cube. The edges within this total configuration form a physical model of  $Q_4$ .

[Computer Projects] For Exercises 2.7.28 through 2.7.31, write an algorithm to construct the indicated graph operation, using only the primary graph operations of additions and deletions of vertices and edges. Test your algorithm on the pair  $(P_4, W_5)$  and on the pair  $(K_4 - K_2, C_4)$ .

2.7.28 Cartesian product of two graphs.

2.7.29 Join of two graphs.

2.7.30 Vertex amalgamation of two graphs.

2.7.31 Amalgamation of two graphs modulo an isomorphism.

## 2.8 SUPPLEMENTARY EXERCISES

2.8.1 Draw all isomorphism types of general graphs with

- a. 2 edges and no isolated vertices.
- b. 2 vertices and 3 edges.

2.8.2 Draw all isomorphism types of digraphs with

- a. 2 edges and no isolated vertices.
- b. 2 vertices and 3 arcs.

2.8.3 Draw all the isomorphism types of simple graphs with

- a. 6 vertices and 3 edges.
- b. 6 vertices and 4 edges.
- c. 5 vertices and 5 edges.
- d. 7 vertices and 4 edges.

2.8.4 Draw the isomorphism types of simple graphs with degree sequence

- a. 433222.
- b. 133333.

2.8.5 Draw two non-isomorphic 4-regular, 7-vertex simple graphs, and prove that every such graph is isomorphic to one of them. Hint: consider edge complements.

2.8.6 Either draw two non-isomorphic 10-vertex, 4-regular, bipartite graphs, or prove that there is only one such graph.

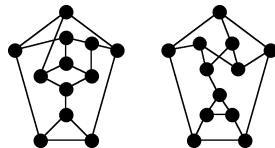
2.8.7 Show that the graphs  $\text{circ}(13 : 1, 4)$  and  $\text{circ}(13 : 1, 5)$  are not isomorphic.

2.8.8 Prove that the complete bipartite graph  $K_{4,4}$  is not isomorphic to the cartesian product graph  $K_4 \times K_2$ .

2.8.9 Indicate the isomorphic pairs of graphs in this set:

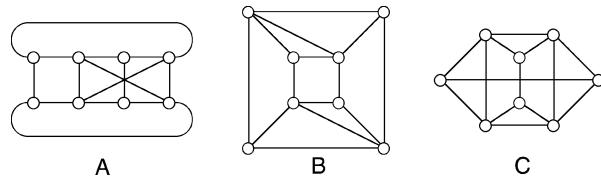
$$\{\text{circ}(8 : 1, 2), \text{circ}(8 : 1, 3), \text{circ}(8 : 1, 4), \text{circ}(8 : 2, 3), \text{circ}(8 : 2, 4), \text{circ}(8 : 3, 4)\}$$

2.8.10 Show that the two graphs in Figure 2.8.1 are not isomorphic.



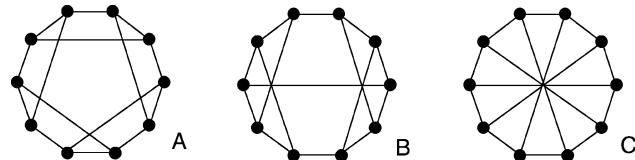
**Figure 2.8.1**

2.8.11 Decide which pairs of these three graphs are isomorphic.



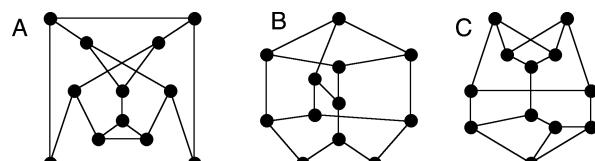
**Figure 2.8.2**

2.8.12 Decide which pairs of these three graphs are isomorphic.



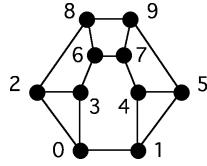
**Figure 2.8.3**

2.8.13 Decide which pairs of these three graphs are isomorphic.



**Figure 2.8.4**

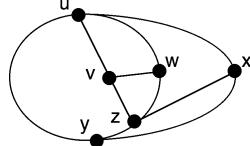
2.8.14 List the vertex orbits and the edge orbits of the graph of Figure 2.8.5.



**Figure 2.8.5**

2.8.15 Calculate the independence number of the graph in Figure 2.8.5.

2.8.16 List the vertex orbits and the edge orbits of the graph of Figure 2.8.6.



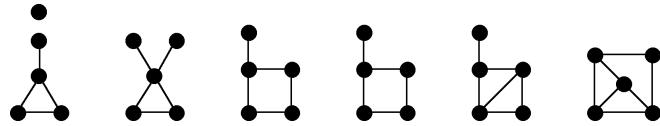
**Figure 2.8.6**

2.8.17 Some of the 4-vertex, simple graphs have exactly two vertex orbits. Draw an illustration of each such isomorphism type.

DEF: A **rigid graph** is a graph whose only automorphism is the identity automorphism.

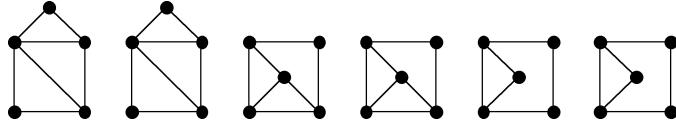
2.8.18 Draw a non-trivial rigid graph.

2.8.19 Reconstruct the graph whose deck appears in Figure 2.8.7.



**Figure 2.8.7**

2.8.20 Reconstruct the graph whose deck appears in Figure 2.8.8.



**Figure 2.8.8**

2.8.21 Suppose that  $v$  is a cut-vertex in an  $n$ -vertex graph  $G$ . Determine the maximum possible number of components of the vertex-deletion subgraph  $G - v$ , and then describe a graph that achieves that maximum.

2.8.22 Suppose that a graph  $G$  is obtained from the hypercube  $Q_5$  by joining each vertex to the vertex that differs from it in all five bits. (For instance, 01101 is joined to 10010.)

a. Prove that the diameter of the graph  $G$  equals 3.

b. Prove that the graph  $G$  does not contain a 3-cycle.

2.8.23 Characterize those graphs with the property that every connected subgraph is an induced subgraph.

- 2.8.24 Prove that the cube-graph  $Q_3$  is a subgraph of the product graph  $K_4 \times K_2$ .
- 2.8.25 A graph  $G$  is *edge-critical* for a property  $P$  if for every edge  $e$ , the graph  $G - e$  does *not* have property  $P$ . We consider diameter-related properties.
- The graphs  $K_{m,n}$  clearly have diameter 2 (for  $m, n \geq 2$ ). Prove they are edge-critical graph for this property.
  - What is the maximum increase in diameter that can be caused by deleting an edge from a 2-connected bipartite graph that is edge-critical with respect to its diameter? Explain.
- 2.8.26 Find the diameter of the cartesian product  $C_m \times C_n$  of two cycle graphs.
- 2.8.27 Calculate the diameter of the Möbius ladder  $ML_n$ .
- 

## GLOSSARY

**adding an edge** between two vertices  $u$  and  $v$  of a graph  $G$ : creating a supergraph with vertex-set  $V_G$  and edge-set  $E_G \cup \{e\}$ .

**adding a vertex**  $v$  to a graph  $G$ : creating a supergraph, denoted  $G \cup \{v\}$ , with vertex-set  $V_G \cup \{v\}$  and edge-set  $E_G$ .

**adjacency matrix of a simple digraph**  $D$ , denoted  $A_D$ : the matrix whose rows and columns are both indexed by identical orderings of  $V_G$ , such that

$$A_D[u, v] = \begin{cases} 1 & \text{if there is a edge from } u \text{ to } v \\ 0 & \text{otherwise} \end{cases}$$

**adjacency matrix of a simple graph**  $G$ , denoted  $A_G$ : the matrix whose rows and columns are both indexed by identical orderings of  $V_G$ , such that

$$A_G[u, v] = \begin{cases} 1 & \text{if } u \text{ and } v \text{ are adjacent} \\ 0 & \text{otherwise} \end{cases}$$

**adjacency-preserving** vertex bijection  $f : V_G \rightarrow V_H$  between two simple graphs  $G$  and  $H$ : for every pair of adjacent vertices  $u$  and  $v$  in graph  $G$ , the vertices  $f(u)$  and  $f(v)$  are adjacent in graph  $H$ .

—, **non-**:  $f(u)$  and  $f(v)$  are non-adjacent in  $H$  whenever  $u$  and  $v$  are non-adjacent in  $G$ .

**amalgamation, vertex** of two disjoint graphs  $G$  and  $H$ : the graph  $(G \cup H)/\{u = v\}$  obtained from the union  $G \cup H$  by merging (or amalgamating) a vertex  $u$  of graph  $G$  and a vertex  $v$  of graph  $H$  into a single vertex called  $uv$ . The vertex-set of this new graph is  $(V_G - \{u\}) \cup (V_H - \{v\}) \cup \{uv\}$  and the edge-set is  $E_G \cup E_H$ , except that any edge that had  $u$  or  $v$  as an endpoint now has the amalgamated vertex  $uv$  as an endpoint instead.

**amalgamation modulo an isomorphism**  $f : X \rightarrow Y$  of disjoint graphs  $G$  and  $H$  with  $X$  a subgraph of  $G$  and  $Y$  a subgraph of  $H$ : the graph  $(G \cup H)/f : X \rightarrow Y$  obtained from the union  $G \cup H$  by merging each vertex  $u$  and each edge  $e$  of subgraph  $X$  with their images  $f(u)$  and  $f(e)$  in subgraph  $Y$ . See §2.7.

—, **subgraph of**: the image of the pasted subgraphs  $X$  and  $Y$  in the amalgamated graph  $(G \cup H)/f : X \rightarrow Y$ .

**automorphism** of a graph  $G$ : an isomorphism from the graph to itself, that is, a structure-preserving permutation  $\pi_V$  on  $V_G$  and a consistent permutation  $\pi_E$  on  $E_G$ ; often written as  $\pi = (\pi_V, \pi_E)$ .

**bridge**: a synonym for *cut-edge*.

**card** in a reconstruction deck: one of the vertex-deletion subgraphs in the reconstruction deck.

**cartesian product**  $G \times H$  of graphs  $G$  and  $H$ : the graph whose vertex-set is the cartesian product  $V_{G \times H} = V_G \times V_H$  and whose edge-set is the union  $E_{G \times H} = (V_G \times E_H) \cup (E_G \times V_H)$ . The endpoints of the edge  $(u, d)$  are the vertices  $(u, x)$  and  $(u, y)$ , where  $x$  and  $y$  are the endpoints of edge  $d$  in graph  $H$ . The endpoints of the edge  $(e, w)$  are  $(u, w)$  and  $(v, w)$ , where  $u$  and  $v$  are the endpoints of edge  $e$  in graph  $G$ .

**center of a graph**  $G$ : the subgraph induced on the set of *central vertices* of  $G$  (see §1.4); denoted  $Z(G)$ .

**circular ladder** with  $n$  rungs: the product  $K_2 \times C_n$ ; denoted  $CL_n$  (defined informally in §1.2).

**clique** in a graph  $G$ : a maximal subset of mutually adjacent vertices in  $G$ .

**clique number** of a graph  $G$ : the number of vertices in a largest clique in  $G$ ; denoted  $\omega(G)$ .

**complement** of a graph: shortened term for *edge-complement*.

**component** of a graph: a maximal connected subgraph; that is, a connected subgraph which is not contained in any larger connected subgraph.

**component of a vertex**  $v$ : a subgraph induced on the subset of all vertices reachable from  $v$ .

**consistent** vertex and edge bijections: see *isomorphism* between *general* graphs.

**cut-edge** of a graph: an edge whose removal increases the number of components.

**cutpoint**: a synonym for *cut-vertex*.

**cut-vertex** of a graph: a vertex whose removal increases the number of components.

**digraph invariant**: a property of a digraph that is preserved by isomorphisms.

**disjoint cycle form** of a permutation: a notation for specifying a permutation; see Example 2.2.1.

**edge-complement** of a simple graph  $G$ : a graph  $\overline{G}$  with the same vertex-set as  $G$ , such that two vertices are in adjacent in  $\overline{G}$  if and only if they are *not* adjacent in  $G$ .

**edge-cut** of a graph: a subset of edges whose removal increases the number of components.

**edge-deletion subgraph**  $G - e$ : the subgraph of  $G$  induced on the edge subset  $E_G - \{e\}$ .

**edge-multiplicity**: the number of edges joining a given pair of vertices or the number of self-loops at a given vertex.

**edge orbit** of a graph  $G$ : an edge subset  $F \subseteq E_G$  such that for every pair of edges  $d, e \in F$ , there is an automorphism of  $G$  that maps  $d$  to  $e$ . Thus, an edge orbit is an equivalence class of  $E_G$  under the action of the automorphisms of  $G$ .

**edge-transitive graph:** a graph  $G$  such that for every edge pair  $d, e \in E_G$ , there is an automorphism of  $G$  that maps  $d$  to  $e$ .

**forest:** an acyclic graph.

**graph invariant:** a property of a graph that is preserved by isomorphisms.

**( $n$ -)hypercube graph**  $Q_n$  of dimension  $n$ : the iterated product  $K_2 \times \cdots \times K_2$  of  $n$  copies of  $K_2$ . Equivalently, the graph whose vertices correspond to the  $2^n$  bitstrings of length  $n$  and whose edges correspond to the pairs of bitstrings that differ in exactly one coordinate.

**incidence matrix**  $I_D$  of a **digraph**  $D$ : a matrix whose rows and columns are indexed by  $V_D$  and  $E_D$ , respectively, such that

$$I_D[v, e] = \begin{cases} 0 & \text{if } v \text{ is not an endpoint of } e \\ 1 & \text{if } v \text{ is the head of } e \\ -1 & \text{if } v \text{ is the tail of } e \\ 2 & \text{if } e \text{ is a self-loop at } v \end{cases}$$

**incidence matrix**  $I_G$  of a **graph**  $G$ : a matrix whose rows and columns are indexed by  $V_G$  and  $E_G$ , respectively, such that

$$I_G[v, e] = \begin{cases} 0 & \text{if } v \text{ is not an endpoint of } e \\ 1 & \text{if } v \text{ is an endpoint of } e \\ 2 & \text{if } e \text{ is a self-loop at } v \end{cases}$$

**independence number** of a graph  $G$ : the number of vertices in a largest independent set in  $G$ ; denoted  $\alpha(G)$ .

**independent set** of vertices in a graph  $G$ : a vertex subset  $W \subseteq V_G$  such that no pair of vertices in  $S$  is joined by an edge, i.e.,  $S$  is a subset of mutually non-adjacent vertices of  $G$ .

**induced subgraph on an edge set**  $D$ : the subgraph with edge-set  $D$  and with vertex-set consisting of the endpoints of all edges in  $D$ .

**induced subgraph on a vertex set**  $W$ : the subgraph with vertex-set  $W$  and edge-set consisting of all edges whose endpoints are in  $W$ .

**invariant:** a shortened term for graph (or digraph) invariant.

**isomorphic digraphs:** two digraphs that have an isomorphism between their underlying graphs that preserves the direction of each edge.

**isomorphic graphs:** two graphs  $G$  and  $H$  that have a structure-preserving vertex bijection between them; denoted  $G \cong H$ .

**isomorphism** between two **general** graphs  $G$  and  $H$ : a structure-preserving vertex bijection  $f_V : V_G \rightarrow V_H$  and an edge bijection  $f_E : E_G \rightarrow E_H$  such that for every edge  $e \in E_G$ , the function  $f_V$  maps the endpoints of  $e$  to the endpoints of the edge  $f_E(e)$ . Such a mapping pair  $(f_V : V_G \rightarrow V_H, f_E : E_G \rightarrow E_H)$  is often written shorthand as  $f : G \rightarrow H$ .

**isomorphism** between two **simple** graphs  $G$  and  $H$ : a structure-preserving vertex bijection  $f : V_G \rightarrow V_H$ .

**isomorphism problem** for graphs: the unsolved problem of devising a practical general algorithm to decide graph isomorphism, or, alternatively, to prove that no such algorithm exists.

**isomorphism type** of a graph  $G$ : the class of all graphs  $H$  isomorphic to  $G$ , i.e., such that there is an isomorphism of  $G$  and  $H$ .

**join** of a new vertex  $v$  to a graph  $G$ : the graph that results when each of the pre-existing vertices of  $G$  is joined to vertex  $v$ ; denoted  $G + v$ .

**join** of two graphs  $G$  and  $H$ : the graph obtained from the graph union  $G \cup H$  by adding an edge between each vertex of  $G$  and each vertex of  $H$ ; denoted  $G + H$ .

**local subgraph** of a vertex  $v$ : synonym of *neighborhood subgraph* of  $v$ ; denoted  $L(v)$ .

**mesh**,  $m_1 \times m_2 \times \cdots \times m_n$ : the iterated product  $P_{m_1} \times P_{m_2} \times \cdots \times P_{m_n}$  of paths.

**—, wraparound**: the iterated product  $C_{m_1} \times C_{m_2} \times \cdots \times C_{m_n}$  of paths.

**Möbius ladder graph**  $ML_n$ : a graph obtained from the circular ladder  $CL_n$  by deleting from the circular ladder two of its parallel curved edges and replacing them with two edges that cross; analogous to the relationship between a Möbius band and a cylindrical band.

**neighbor** of a vertex  $v$ : any vertex adjacent to  $v$ .

**(open) neighborhood subgraph** of a vertex  $v$ : the subgraph induced on the neighbors of  $v$ ; denoted  $L(v)$ . Also called *local subgraph*.

**octahedral graph,  $n$ -dimensional**: the graph  $\mathcal{O}_n$  defined recursively, using the join operation, as

$$\mathcal{O}_n = \begin{cases} 2K_1 & \text{if } n = 0 \\ 2K_1 + \mathcal{O}_{n-1} & \text{if } n \geq 1 \end{cases}$$

**preserves adjacency**: see *adjacency-preserving*.

**preserves non-adjacency**: see *adjacency-preserving, non-*.

**primary maintenance operation** on a graph: adding or deleting a vertex or an edge.

**product**  $G \times H$  of graphs  $G$  and  $H$ : shortened term for *cartesian product*.

**proper subgraph** of a graph  $G$ : a subgraph of  $G$  that is neither  $G$  nor an isomorphic copy of  $G$ .

**reachable** from a vertex  $v$ : said of a vertex for which there is a walk from  $v$ .

**reachability relation** for a graph  $G$ : the equivalence relation on  $V_G$ , where two vertices are related if one is reachable from the other.

**reconstruction deck** of a graph  $G$ : the vertex-deletion subgraph list of  $G$ , with no labels on the vertices.

**reconstruction problem** for graphs: the unsolved problem of deciding whether two non-isomorphic graphs with three or more vertices can have the same vertex-deletion subgraph list.

**rigid graph**: a graph whose only automorphism is the identity automorphism.

**spanning subgraph** of a graph  $G$ : a subgraph  $H$  of  $G$  with  $V_H = V_G$ .

**spanning tree**: a spanning subgraph that is a tree.

**structure-preserving** vertex bijection between two **general** graphs  $G$  and  $H$ : a vertex bijection  $f : V_G \rightarrow V_H$  such that

(i) the number of edges (even if 0) joining each pair of distinct vertices  $u$  and  $v$  in  $G$  equals the number of edges joining their images  $f(u)$  and  $f(v)$  in  $H$ , and

(ii) the number of self-loops at each vertex  $x$  in  $G$  equals the number of self-loops at the vertex  $f(x)$  in  $H$ .

**structure-preserving** vertex bijection between two **simple** graphs  $G$  and  $H$ : a vertex bijection  $f : V_G \rightarrow V_H$  that preserves adjacency and non-adjacency, i.e., for every pair of vertices in  $G$ ,

$$u \text{ and } v \text{ are adjacent in } G \iff f(u) \text{ and } f(v) \text{ are adjacent in } H$$

**subdigraph** of a digraph  $D$ : a digraph whose underlying graph is a subgraph of the underlying graph of  $D$ , and whose edge directions are inherited from  $D$ .

**subgraph** of a graph  $G$ : a graph  $H$  whose vertices and edges are all in  $G$ , or any graph isomorphic to such a graph.

—, **proper**: a subgraph that is neither  $G$  nor an isomorphic copy of  $G$ .

—, **spanning**: a subgraph  $H$  with  $V_H = V_G$ .

**supergraph**: the “opposite” of subgraph; that is,  $H$  is a supergraph of  $G$  if and only if  $G$  is a subgraph of  $H$ .

**suspension** of a graph  $G$  from a new vertex  $v$ : a synonym for *join*.

**table of incident edges** for a graph  $G$ : a table  $I_{V:E}(G)$  that lists, for each vertex  $v$ , all the edges incident on  $v$ .

**table of incoming arcs** for a digraph  $D$ : a table  $inv_{V:E}(D)$  that lists, for each vertex  $v$ , all arcs that are directed to  $v$ .

**table of outgoing arcs** for a digraph  $D$ : a table  $out_{V:E}(D)$  that lists, for each vertex  $v$ , all arcs that are directed from  $v$ .

**(graph) union** of two graphs  $G = (V, E)$  and  $G' = (V', E')$ : the graph  $G \cup G'$  whose vertex-set and edge-set are the disjoint unions, respectively, of the vertex-sets and edge-sets of  $G$  and  $G'$ .

—,  **$n$ -fold self-**: the iterated disjoint union  $G \cup \dots \cup G$  of  $n$  copies of the graph  $G$ ; denoted  $nG$ .

**vertex-cut** of a graph: a subset of vertices whose removal increases the number of components.

**vertex-deletion subgraph**  $G - v$ : the subgraph of  $G$  induced on the vertex subset  $V_G - \{v\}$ .

**vertex-deletion subgraph list**: a list of the isomorphism types of the collection of vertex-deletion subgraphs.

**vertex orbit** of a graph  $G$ : a vertex subset  $W \subseteq V_G$  such that for every pair of vertices  $u, v \in W$ , there is an automorphism of  $G$  that maps  $u$  to  $v$ . Thus, a vertex orbit is an equivalence class of  $V_G$  under the action of the automorphisms of  $G$ .

**vertex-transitive** graph: a graph  $G$  such that for every vertex pair  $u, v \in V_G$ , there is an automorphism of  $G$  that maps  $u$  to  $v$ .

**$n$ -wheel**  $W_n$ : the join  $K_1 + C_n$  of a single vertex and an  $n$ -cycle.

—, **even**: a wheel for  $n$  even.

—, **odd**: a wheel for  $n$  odd.

# Chapter 3

---

## TREES

- 3.1 Characterizations and Properties of Trees**
  - 3.2 Rooted Trees, Ordered Trees, and Binary Trees**
  - 3.3 Binary-Tree Traversals**
  - 3.4 Binary-Search Trees**
  - 3.5 Huffman Trees and Optimal Prefix Codes**
  - 3.6 Priority Trees**
  - 3.7 Counting Labeled Trees: Prüfer Encoding**
  - 3.8 Counting Binary Trees: Catalan Recursion**
- 

### INTRODUCTION

Trees are important to the structural understanding of graphs and to the algorithms of information processing, and they play a central role in the design and analysis of connected networks. In fact, trees are the backbone of optimally connected networks.

A main task in information management is deciding how to store data in space-efficient ways that also allow their retrieval and modification to be time-efficient. Tree-based structures are often the best way of balancing these competing goals. For example, the binary-search-tree structure introduced in §3.4 leads to an optimally efficient search algorithm. Several other applications of binary trees are given in §3.3 through §3.6.

The final two sections include a brief excursion into *enumerative combinatorics*. In §3.7, Cayley's formula for the number of labeled  $n$ -vertex trees is derived using *Prüfer Encoding*. In §3.8, a recurrence relation for the number of different binary trees is established and then solved to obtain a closed formula.

## 3.1 CHARACTERIZATIONS AND PROPERTIES OF TREES

REVIEW FROM §1.5: A *tree* is a connected graph with no cycles.

Characterizing trees in a variety of ways provides flexibility for their application. The first part of §3.1 establishes some basic properties of trees that culminate in Theorem 3.1.8, where six different but equivalent characterizations of a tree are given.

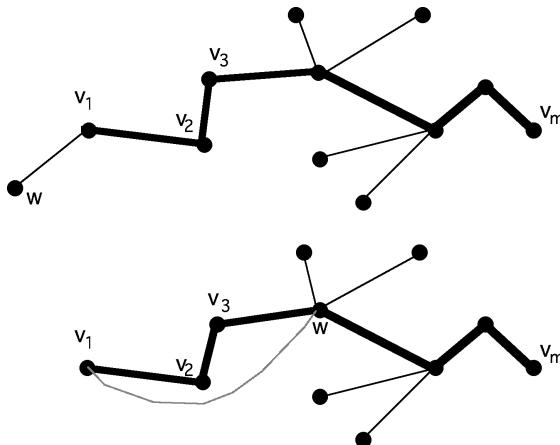
### Basic Properties of Trees

DEFINITION: In an undirected tree, a *leaf* is a vertex of degree 1.

If a leaf is deleted from a tree, then the resulting graph is a tree having one vertex fewer. Thus, induction is a natural approach to proving tree properties, provided one can always find a leaf. The following proposition guarantees the existence of such a vertex.

**Proposition 3.1.1.** *Every tree with at least one edge has at least two leaves.*

**Proof:** Let  $P = \langle v_1, v_2, \dots, v_m \rangle$  be a path of maximum length in a tree  $T$ . Suppose one of its endpoints, say  $v_1$ , has degree greater than 1. Then  $v_1$  is adjacent to vertex  $v_2$  on path  $P$  and also to some other vertex  $w$  (see Figure 3.1.1). If  $w$  is different from all of the vertices  $v_i$ , then  $P$  could be extended, contradicting its maximality. On the other hand, if  $w$  is one of the vertices  $v_i$  on the path, then the acyclic property of  $T$  would be contradicted. Thus, both endpoints of path  $P$  must be leaves in tree  $T$ .  $\diamond$



**Figure 3.1.1** The two cases in the proof of Proposition 3.1.1.

**Corollary 3.1.2.** *If the degree of every vertex of a graph is at least 2, then that graph must contain a cycle.*

**Proof:** Apply Proposition 3.1.1 to any one of the components of that graph.  $\diamond$

The next proposition establishes a fundamental property of trees. Its proof is the first of several instances that demonstrate the effectiveness of the inductive approach to proving assertions about trees.

**Proposition 3.1.3.** Every tree on  $n$  vertices contains exactly  $n - 1$  edges.

**Proof:** A tree on one vertex is the trivial tree, which has no edges.

Assume for some number  $k \geq 1$ , as an induction hypothesis, that every tree on  $k$  vertices has exactly  $k - 1$  edges. Next consider any tree  $T$  on  $k + 1$  vertices. By Proposition 3.1.1,  $T$  contains a leaf, say  $v$ . Then the graph  $T - v$  is acyclic, since deleting pieces from an acyclic graph cannot create a cycle. Moreover,  $T - v$  is connected, since the vertex  $v$  has degree 1 in  $T$ . Thus,  $T - v$  is a tree on  $k$  vertices, and, hence,  $T - v$  has  $k - 1$  edges, by the induction hypothesis. But since  $\deg(v) = 1$ , it follows that  $T - v$  has one edge fewer than  $T$ . Therefore,  $T$  has  $k$  edges, which completes the proof.  $\diamond$

REVIEW FROM §2.3: An acyclic graph is called a **forest**.

REVIEW FROM §2.4: The number of components of a graph  $G$  is denoted  $c(G)$ .

**Corollary 3.1.4.** A forest  $G$  on  $n$  vertices has  $n - c(G)$  edges.

**Proof:** Apply Proposition 3.1.3 to each of the components of  $G$ .  $\diamond$

**Corollary 3.1.5.** Any graph  $G$  on  $n$  vertices has at least  $n - c(G)$  edges.

**Proof:** If  $G$  has cycle-edges, then remove them one at a time until the resulting graph  $\hat{G}$  is acyclic. Then  $\hat{G}$  has  $n - c(\hat{G})$  edges, by Corollary 3.1.4; and  $c(\hat{G}) = c(G)$ , by Corollary 2.4.3.  $\diamond$

Corollary 3.1.5 provides a lower bound for the number of edges in a graph. The next two results establish an upper bound for certain simple graphs. This kind of result is typically found in *extremal graph theory* (see §11.2).

**Proposition 3.1.6.** If  $G$  is a simple graph with  $n$  vertices and  $k$  components, then

$$|E_G| \leq \frac{(n - k)(n - k + 1)}{2}$$

**Proof:** Let  $C_1, \dots, C_k$  be the  $k$  components of  $G$ . If component  $C_i$  has  $n_i$  vertices, then  $\sum_{i=1}^k n_i = n$  and  $n_i \geq 1$ . Since the maximum number of edges occurs when each of the  $k$  components is a complete graph, we may assume that  $|E(C_i)| = \frac{n_i^2 - n_i}{2}$ . Thus,  $|E_G| = \sum_{i=1}^k \frac{n_i^2 - n_i}{2}$ .

To establish the upper bound, it suffices to show that the maximum number of edges is attained by a graph having exactly one nontrivial component. Suppose, to the contrary, that the maximum occurs in a graph with  $C_i = K_r$  and  $C_j = K_s$ , for  $i \neq j$  and  $r \geq s \geq 2$ . Then the total number of edges contained in these two components is  $\frac{r^2 + s^2 - (r+s)}{2}$ . But then the graph that results from replacing  $C_i$  and  $C_j$  by  $K_{r+1}$  and  $K_{s-1}$ , respectively, has  $\frac{(r+1)^2 + (s-1)^2 - (r+s)}{2}$  edges in those two components, contradicting the maximality of the first graph.  $\diamond$

**Corollary 3.1.7.** A simple  $n$ -vertex graph with more than  $\frac{1}{2}(n - 1)(n - 2)$  edges must be connected.  $\diamond$

### Six Different Characterizations of a Tree

Trees have many possible characterizations, and each contributes to the structural understanding of graphs in a different way. The following theorem establishes some of the most useful characterizations.

**Theorem 3.1.8.** *Let  $T$  be a graph with  $n$  vertices. Then the following statements are equivalent.*

1.  $T$  is a tree.
2.  $T$  contains no cycles and has  $n - 1$  edges.
3.  $T$  is connected and has  $n - 1$  edges.
4.  $T$  is connected, and every edge is a cut-edge.
5. Any two vertices of  $T$  are connected by exactly one path.
6.  $T$  contains no cycles, and for any new edge  $e$ , the graph  $T + e$  has exactly one cycle.

**Proof:** If  $n = 1$ , then all six statements are trivially true. So assume  $n \geq 2$ .

1  $\Rightarrow$  2 : By Proposition 3.1.3.

2  $\Rightarrow$  3 : Suppose that  $T$  has  $k$  components. Then, by Corollary 3.1.4, the forest  $T$  has  $n - k$  edges. Hence,  $k = 1$ .

3  $\Rightarrow$  4 : Let  $e$  be an edge of  $T$ . Since  $T - e$  has  $n - 2$  edges, Corollary 3.1.5 implies that  $n - 2 \geq n - c(T - e)$ . So  $T - e$  has at least two components.

4  $\Rightarrow$  5 : By way of contradiction, suppose that Statement 4 is true, and let  $x$  and  $y$  be two vertices that have two different paths between them, say  $P_1$  and  $P_2$ . Let  $u$  be the first vertex from which the two paths diverge (this vertex might be  $x$ ), and let  $v$  be the first vertex at which the paths meet again (see Figure 3.1.2). Then these two  $u - v$  paths taken together form a cycle, and any edge on this cycle is not a cut-edge, which contradicts Statement 4.

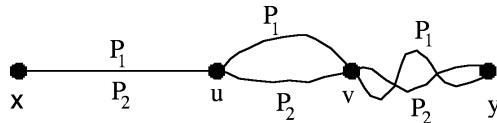


Figure 3.1.2

5  $\Rightarrow$  6 :  $T$  contains no cycles since any two vertices on a cycle have two different paths between them, consisting of the opposite routes around the cycle. Furthermore, the addition of any new edge  $e$  to  $T$  will create a cycle, since the endpoints of  $e$ , say  $u$  and  $v$ , are already connected by a path in  $T$ . To show that this cycle is unique, suppose two cycles were created. They both would contain edge  $e$ , and the long way around each of these cycles would then be two different  $u - v$  paths in  $T$ , which would contradict Statement 5.

6  $\Rightarrow$  1 : By way of contradiction, assume that  $T$  is not connected. Then the addition of an edge joining a vertex in one component to a vertex in a different component would not create a cycle, which would contradict Statement 6.  $\diamond$

The fundamental properties of a tree are summarized below for easy reference.

**Table 3.1.1** Summary of Basic Properties of a Tree  $T$  on  $n$  vertices

1.  $T$  is connected.
2.  $T$  contains no cycles.
3. Given any two vertices  $u$  and  $v$  of  $T$ , there is a unique  $u$ - $v$  path.
4. Every edge in  $T$  is a cut-edge.
5.  $T$  contains  $n - 1$  edges.
6.  $T$  contains at least two vertices of degree 1 if  $n \geq 2$ .
7. Adding an edge between two vertices of  $T$  yields a graph with exactly one cycle.

### The Center of a Tree

REVIEW FROM §1.4 AND §2.3:

- The **eccentricity** of a vertex  $v$  in a graph  $G$ , denoted  $\text{ecc}(v)$ , is the distance from  $v$  to a vertex farthest from  $v$ . That is,  $\text{ecc}(v) = \max_{x \in V_G} \{d(v, x)\}$ .
- A **central vertex** of a graph is a vertex with minimum eccentricity.
- The **center of a graph**  $G$ , denoted  $Z(G)$ , is the subgraph induced on the set of central vertices of  $G$ .

In an arbitrary graph  $G$ , the center  $Z(G)$  can be anything from a single vertex to all of  $G$ . However, C. Jordan showed in 1869 that the center of a tree has only two possible cases. We begin with some preliminary results concerning the eccentricity of vertices in a tree.

**Lemma 3.1.9.** *Let  $T$  be a tree with at least three vertices.*

- (a) *If  $v$  is a leaf of  $T$  and  $w$  is its neighbor, then  $\text{ecc}(v) = \text{ecc}(w) + 1$ .*
- (b) *If  $v$  is a central vertex of  $T$ , then  $\deg(v) \geq 2$ .*

**Proof:** (a) Since  $T$  has at least three vertices,  $\deg(w) \geq 2$ . Then there exists a vertex  $z \neq v$  such that  $d(w, z) = \text{ecc}(w)$ . But  $z$  is also a vertex farthest from  $v$ , and hence  $\text{ecc}(v) = d(v, z) = d(w, z) + 1 = \text{ecc}(w) + 1$ .

(b) By Part (a), a vertex of degree 1 cannot have minimum eccentricity in  $T$  and hence, cannot be a central vertex of  $T$ .  $\diamond$

**Lemma 3.1.10.** *Let  $v$  and  $w$  be two vertices in a tree  $T$  such that  $w$  is of maximum distance from  $v$  (i.e.,  $\text{ecc}(v) = d(v, w)$ ). Then  $w$  is a leaf.*

**Proof:** Let  $P$  be the unique  $v$ - $w$  path in tree  $T$ . If  $\deg(w) \geq 2$ , then  $w$  would have a neighbor whose distance from  $v$  would equal  $d(v, w) + 1$ , contradicting the premise that  $w$  is at maximum distance.  $\diamond$

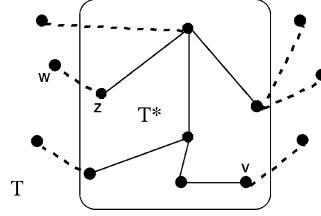
**Lemma 3.1.11.** *Let  $T$  be a tree with at least three vertices, and let  $T^*$  be the subtree of  $T$  obtained by deleting from  $T$  all its leaves. If  $v$  is a vertex of  $T^*$ , then  $\text{ecc}_T(v) = \text{ecc}_{T^*}(v) + 1$ .*

**Proof:** Let  $w$  be a vertex of  $T$  such that  $\text{ecc}_T(v) = d(v, w)$ . By Lemma 3.1.10,  $w$  is a leaf of  $T$  and hence,  $w \notin V_{T^*}$  (as illustrated in Figure 3.1.3). It follows that the

neighbor of  $w$ , say  $z$ , is a vertex of  $T^*$  that is farthest from  $v$  among all vertices in  $T^*$ , that is,  $\text{ecc}_{T^*}(v) = d(v, z)$ . Thus,

$$\text{ecc}_T(v) = d(v, w) = d(v, z) + 1 = \text{ecc}_{T^*}(v) + 1$$

◊



**Figure 3.1.3**

**Proposition 3.1.12.** *Let  $T$  be a tree with at least three vertices, and let  $T^*$  be the subtree of  $T$  obtained by deleting from  $T$  all its leaves. Then*

$$Z(T) = Z(T^*)$$

**Proof:** From the two preceding lemmas, we see that deleting all the leaves decreases all the remaining vertex eccentricities by 1. It follows that the resulting tree has the same center. ◊

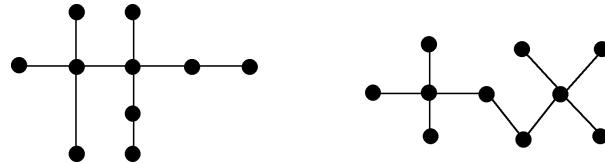
**Corollary 3.1.13 [Jordan, 1869].** *Let  $T$  be an  $n$ -vertex tree. Then the center  $Z(G)$  is either a single vertex or a single edge.*

**Proof:** The assertion is trivially true for  $n = 1$  and  $n = 2$ . The result follows by induction, using Proposition 3.1.12. ◊

### Tree Isomorphisms and Automorphisms

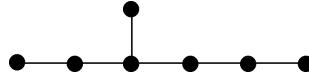
As we saw in §2.5, graph invariants are helpful both in isomorphism testing and in the construction of isomorphisms. For instance, the center of a graph must map to the center of the image graph, and a leaf and its image leaf must be the same distance from their respective centers. The next two examples illustrate these ideas for trees.

**Example 3.1.1:** The two graphs in Figure 3.1.4 have the same degree sequence, but they can be readily seen to be non-isomorphic in several ways. For instance, the center of the left graph is a single vertex, but the center of the right graph is a single edge. Another way is to observe that the two graphs have unequal diameters.



**Figure 3.1.4** Why are these trees non-isomorphic?

**Example 3.1.2:** The graph shown in Figure 3.1.5 below does not have a non-trivial automorphism because the three leaves are all different distances from the center, and hence, an automorphism must map each of them to itself.



**Figure 3.1.5** A tree that has no non-trivial automorphisms.

**Remark:** There is a linear-time algorithm for testing the isomorphism of two trees (see [AhHoUl74, p84]).

### Tree-Graphic Sequences

The next result characterizes those sequences that are degree sequences of trees.

**DEFINITION:** A sequence  $\langle d_1, d_2, \dots, d_n \rangle$  is said to be **tree-graphic** if there is a permutation of it that is the degree sequence of some  $n$ -vertex tree.

**Theorem 3.1.14.** A sequence  $\langle d_1, d_2, \dots, d_n \rangle$  of  $n \geq 2$  positive integers is tree-graphic if and only if  $\sum_{i=1}^n d_i = 2n - 2$ .

**Proof:** If some permutation of the sequence  $\langle d_1, d_2, \dots, d_n \rangle$  is the degree sequence of an  $n$ -vertex tree  $T$ , then, by Euler's Degree-Sum Theorem (§1.1) and Proposition 3.1.3, we have  $\sum_{i=1}^n d_i = 2|E_T| = 2n - 2$ .

To prove the sufficiency assertion, we use induction on  $n$ . The assertion is trivially true for  $n = 2$ . Assume that the assertion is true for some  $n \geq 2$  and let  $\langle d_1, d_2, \dots, d_{n+1} \rangle$  be a sequence of positive integers satisfying  $\sum_{i=1}^{n+1} d_i = 2(n+1) - 2 = 2n$ . We may assume without loss of generality that  $d_1 \geq d_2 \geq \dots \geq d_{n+1}$ . It follows from a simple counting argument that  $2 \leq d_1 \leq n$  and  $d_n = d_{n+1} = 1$ . Thus, the terms of the sequence  $\langle d_1 - 1, d_2, d_3, \dots, d_n \rangle$  are positive and sum to  $2n - 2$ , and hence, by the induction hypothesis, there is an  $n$ -vertex tree  $T$  whose degree sequence is a permutation of  $\langle d_1 - 1, d_2, d_3, \dots, d_n \rangle$ . Let  $\hat{T}$  be the tree obtained by joining a new vertex to a vertex of  $T$  of degree  $d_1 - 1$ . Then the degree sequence of  $\hat{T}$  is a permutation of the sequence  $\langle d_1, d_2, \dots, d_{n+1} \rangle$ .  $\diamond$

### Trees as Subgraphs

An arbitrary graph is likely to contain a number of different trees as subgraphs. The following theorem shows that if a simple  $n$ -vertex graph is sufficiently *dense* (i.e., it has sufficiently many edges), then it will contain every type of tree up to a certain order.

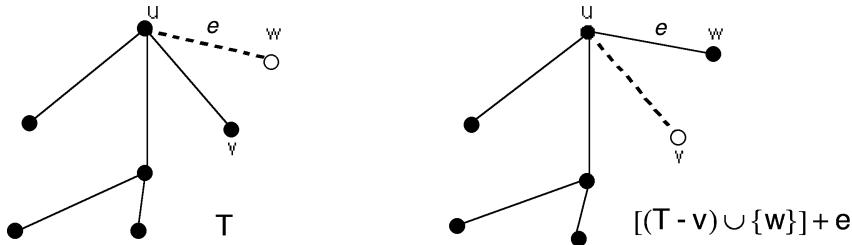
**NOTATION:** The minimum degree of the vertices of a graph  $G$  is denoted  $\delta_{\min}(G)$ .

**Theorem 3.1.15.** Let  $T$  be any tree on  $n$  vertices, and let  $G$  be a simple graph such that  $\delta_{\min}(G) \geq n - 1$ . Then  $T$  is a subgraph of  $G$ .

**Proof:** The result is clearly true if  $n = 1$  or  $n = 2$ , since  $K_1$  and  $K_2$  are subgraphs of every graph having at least one edge.

Assume that the result is true for some  $n \geq 2$ . Let  $T$  be a tree on  $n + 1$  vertices, and let  $G$  be a graph with  $\delta_{\min}(G) \geq n$ . We show that  $T$  is a subgraph of  $G$ .

Let  $v$  be a leaf of  $T$ , and let vertex  $u$  be its only neighbor in  $T$ . The vertex-deletion subgraph  $T - v$  is a tree on  $n$  vertices, so, by the induction hypothesis,  $T - v$  is a subgraph of  $G$ . Since  $\deg_G(u) \geq n$ , there is some vertex  $w$  in  $G$  but not in  $T - v$  that is adjacent to  $u$ . Let  $e$  be the edge joining vertices  $u$  and  $w$  (see Figure 3.1.6). Then  $T - v$  together with vertex  $w$  and edge  $e$  form a tree that is isomorphic to  $T$  and is a subgraph of  $G$ .  $\diamond$



**Figure 3.1.6**  $T$  is isomorphic to  $[(T - v) \cup \{w\}] + e$ .

### EXERCISES for Section 3.1

- 3.1.1 Draw a 6-vertex connected graph that has exactly seven edges and exactly three cycles.
- 3.1.2 Draw a 12-vertex forest that has exactly 10 edges.

*In Exercises 3.1.3 through 3.1.12, either draw the required graph or explain why no such graph exists.*

- 3.1.3<sup>s</sup> A 7-vertex, 3-component, simple graph with 10 edges.
- 3.1.4 A 6-vertex, 2-component, simple graph with 11 edges.
- 3.1.5 An 8-vertex, 2-component, simple graph with exactly nine edges and three cycles.
- 3.1.6 An 8-vertex, 2-component, simple graph with exactly 10 edges and three cycles.
- 3.1.7 A 9-vertex, 2-component, simple graph with exactly 10 edges and two cycles.
- 3.1.8 A 9-vertex, simple, connected graph with exactly 10 edges and three cycles.
- 3.1.9<sup>s</sup> A 9-vertex, simple, connected graph with exactly 12 edges and three cycles.
- 3.1.10 A 10-vertex, 2-component, forest with exactly nine edges.
- 3.1.11 A 10-vertex, 3-component, forest with exactly nine edges.
- 3.1.12 An 11-vertex, simple, connected graph with exactly 14 edges that contains five edge-disjoint cycles.

3.1.13<sup>s</sup> Let  $G$  be a connected simple graph on  $n$  vertices. Determine a lower bound on the average degree of a vertex, and characterize those graphs that achieve the lower bound.

- 3.1.14 Prove that if  $G$  is a tree having an even number of edges, then  $G$  must contain at least one vertex having even degree.

- 3.1.15 Suppose the average degree of the vertices of a connected graph is exactly 2. How many cycles does  $G$  have?

**3.1.16** Prove or disprove: If a simple graph  $G$  has no cut-edges, then every vertex of  $G$  has even degree.

**3.1.17<sup>s</sup>** Prove or disprove: A connected  $n$ -vertex simple graph with  $n$  edges must contain exactly one cycle.

**3.1.18** Prove or disprove: There exists a connected  $n$ -vertex simple graph with  $n + 1$  edges that contains exactly two cycles.

**3.1.19** Prove or disprove: There does not exist a connected  $n$ -vertex simple graph with  $n + 2$  edges that contains four edge-disjoint cycles.

**3.1.20<sup>s</sup>** Prove that  $H$  is a maximal acyclic subgraph of a connected graph  $G$  if and only if  $H$  is a spanning tree of  $G$ . What analogous statement can be made for graphs that are not necessarily connected?

**3.1.21** Prove that if a graph has exactly two vertices of odd degree, then there must be a path between them.

**3.1.22** Show that any nontrivial simple graph contains at least two vertices that are not cut-vertices.

**3.1.23** Prove that if two distinct closed paths have an edge  $e$  in common, then there must be a closed path that does not contain  $e$ .

**3.1.24<sup>s</sup>** Let  $G$  be a simple graph on  $n$  vertices. Prove that if  $\delta_{\min}(G) \geq \frac{n-1}{2}$ , then  $G$  is connected.

**3.1.25** Prove that if any single edge is added to a connected graph  $G$ , then at least one cycle is created.

**3.1.26** Prove that a graph  $G$  is a forest if and only if every induced subgraph of  $G$  contains a vertex of degree 0 or 1.

**3.1.27<sup>s</sup>** Characterize those graphs with the property that every connected subgraph is an induced subgraph.

**3.1.28** Let  $T$  be a tree with at least three vertices, and let  $T^*$  be the subtree of  $T$  obtained by deleting from  $T$  all its vertices of degree 1. Prove that  $diam(T^*) = diam(T) - 2$  and  $rad(T^*) = rad(T) - 1$ .

**3.1.29** Let  $T$  be a tree with at least two vertices. Prove that the center  $Z(T)$  is a single edge if and only  $diam(T) = 2rad(T) - 1$ .

**3.1.30** Determine the smallest integer  $n \geq 2$  for which there exists an  $n$ -vertex tree whose only automorphism is the trivial one.

**3.1.31** Determine the smallest integer  $n \geq 2$  for which there exists an  $n$ -vertex tree whose center is a single edge and whose only automorphism is the trivial one.

**DEFINITION:** A tree  $T$  with  $m + 1$  vertices and  $m$  edges is **graceful** if it is possible to label the vertices of  $T$  with distinct elements from the set  $\{0, 1, \dots, m\}$  in such a way that the induced edge-labeling, which assigns the integer  $|i - j|$  to the edge joining the vertices labeled  $i$  and  $j$ , assigns the labels  $1, 2, \dots, m$  to the  $m$  edges of  $G$ .

- 3.1.32**
  - a. Prove that every path is graceful.
  - b. Prove that  $K_{1,n}$  (also called a *star*) is graceful for all  $n \geq 2$ .
  - c. Show that every tree on 6 vertices is graceful.
  - d. Prove or disprove: Every tree is graceful. (This is an open problem.)

**DEFINITION:** The **distance sum** (or **Wiener index**<sup>†</sup>) of a graph  $G$ , denoted  $ds(G)$ , is the sum of the distances between all pairs of vertices in  $G$ ; that is,  $ds(G) = \sum_{u,v \in V_G} d(u,v)$ .

**3.1.33** Determine the distance sum for the  $n$ -vertex star  $K_{1,n-1}$ .

**3.1.34** Let  $T$  be an  $n$ -vertex tree different from the star  $K_{1,n}$ . Prove that  $ds(T) > ds(K_{1,n-1})$ .

**3.1.35** Determine the distance sum for the  $n$ -vertex path graph  $P_n$ .

**3.1.36** Let  $T$  be an  $n$ -vertex tree different from the path graph  $P_n$ . Prove that  $ds(T) < ds(P_n)$ .

## 3.2 ROOTED TREES, ORDERED TREES, AND BINARY TREES

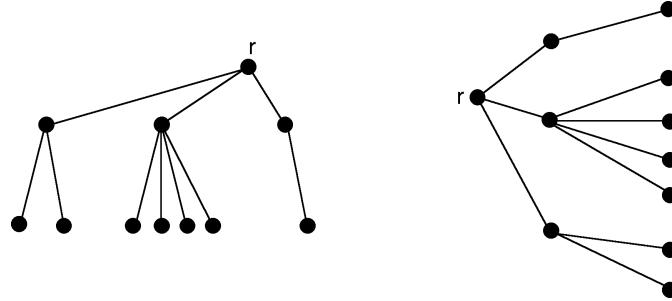
### Rooted Trees

**DEFINITION:** A **directed tree** is a directed graph whose underlying graph is a tree.

**DEFINITION:** A **rooted tree** is a tree with a designated vertex called the **root**. Each edge is considered to be directed away from the root.

Thus, a rooted tree is a directed tree such that the root has indegree 0 and all other vertices have indegree 1.

In drawing a rooted tree, the arrows are usually omitted from the arcs, since their direction is always away from the root. Figure 3.2.1 shows the two most common ways of drawing a rooted tree.

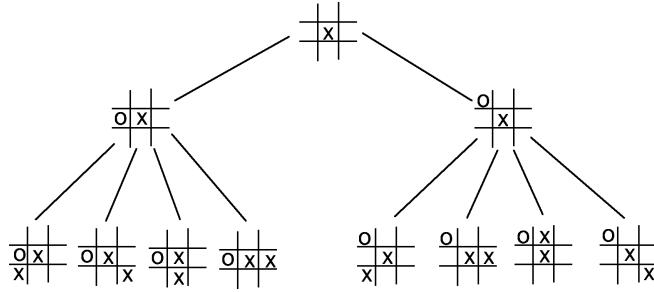


**Figure 3.2.1** Two common ways of drawing a rooted tree.

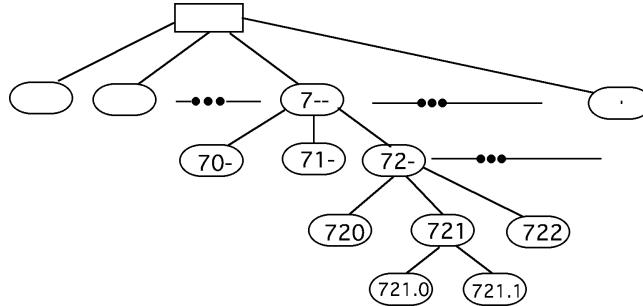
### Some Applications of Rooted Trees

**Application 3.2.1 Decision Trees:** The organization of large computer programs that perform complex decision-making strategies is often based on rooted trees. For example, Figure 3.2.2 below shows some of the sequences for the first three moves of a game of tic-tac-toe, starting with an  $x$  in the center. In fact, if we take into account symmetry, then the figure actually shows *all* the possible sequences.

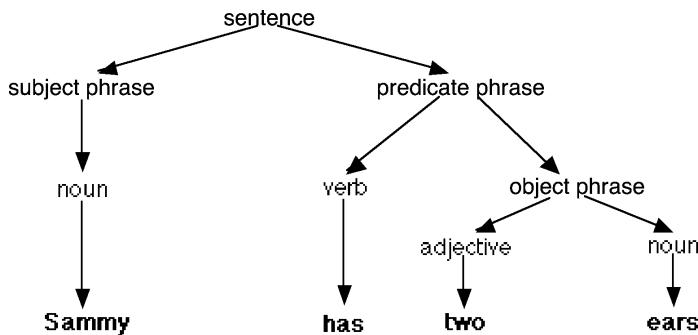
<sup>†</sup> Named for the chemist H. Wiener, who introduced this measure in 1947.

**Figure 3.2.2** The first three moves of tic-tac-toe.

**Application 3.2.2 Data Organization:** Rooted trees are used to store information that is organized into categories and subcategories. The *Dewey Decimal Classification System* for libraries is one such example. The system starts with ten broad subject areas, each assigned a range of numbers. More specialized areas within a given area are assigned subranges. Eventually single numbers are assigned to the specific subjects within an area, and these subjects are further broken down by using decimal fractions. The rooted tree shown in Figure 3.2.3 shows how books in the fine arts are classified in the Dewey decimal system.

**Figure 3.2.3** Fine arts subtree under Dewey decimal system.

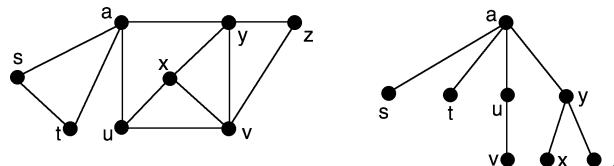
**Application 3.2.3 Sentence Parsing:** A rooted tree can be used to parse a sentence in a natural language, such as English. The tree in Figure 3.2.4 diagrams the underlying syntactic structure of the words and phrases in the sentence “Sammy has two ears.”

**Figure 3.2.4** Parsing a sentence.

**Application 3.2.4 Shortest-Path Trees:** For a given connected graph  $G$  with a vertex  $v$ , a rooted tree provides a compact way of exhibiting for each  $w \in V_G$ , a shortest path from  $v$  to  $w$ .

**DEFINITION:** Let  $v$  be a vertex in a connected graph  $G$ . A **shortest-path tree** for  $G$  from  $v$  is a rooted tree  $T$  with vertex-set  $V_G$  and root  $v$  such that the unique path in  $T$  from  $v$  to each vertex  $w$  is a shortest path in  $G$  from  $v$  to  $w$ .

**Example 3.2.1:** Figure 3.2.5 shows an 8-vertex graph and a shortest-path tree from vertex  $a$



**Figure 3.2.5 A graph and a shortest-path tree.**

**Remark:** The *breadth-first search* produces a shortest-path tree for an unweighted graph, and *Dijkstra's algorithm* produces one for a weighted graph. These and other *tree-growing* algorithms are discussed in Chapter 4.

### Rooted Tree Terminology

Designating a root imposes a hierarchy on the vertices of a rooted tree, according to their distance from that root.

**DEFINITION:** In a rooted tree, the **depth** or **level** of a vertex  $v$  is its distance from the root, i.e., the length of the unique path from the root to  $v$ . Thus, the root has depth 0.

**Remark:** Typically, a rooted tree is drawn so that the root is at the top, and the vertices at each level are horizontally aligned.

**DEFINITION:** The **height** of a rooted tree is the length of a longest path from the root (or the greatest depth in the tree).

**DEFINITION:** If vertex  $v$  immediately precedes vertex  $w$  on the path from the root to  $w$ , then  $v$  is the **parent** of  $w$  and  $w$  is the **child** of  $v$ .

**DEFINITION:** Vertices having the same parent are called **siblings**.

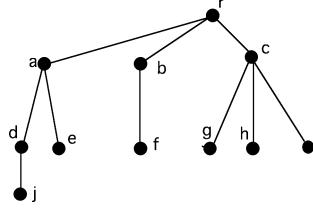
**DEFINITION:** A vertex  $w$  is called a **descendant** of a vertex  $v$  (and  $v$  is called an **ancestor** of  $w$ ), if  $w$  is on the unique path from the root to  $w$ . If, in addition,  $w \neq v$ , then  $w$  is a **proper** descendant of  $v$  (and  $v$  is a proper ancestor of  $w$ ).

**DEFINITION:** A **leaf** in a rooted tree is any vertex having no children.

**DEFINITION:** An **internal vertex** in a rooted tree is any vertex that has at least one child. The root is internal, unless the tree is trivial (i.e., a single vertex).

**Example 3.2.2:** A rooted tree is shown in Figure 3.2.6. The height of this tree is 3, and vertex  $j$  is the only vertex whose depth achieves this value. Also,  $r, a, b, c$ , and  $d$

are the internal vertices; vertices  $e, f, g, h, i$ , and  $j$  are the leaves; vertices  $g, h$ , and  $i$  are siblings; vertex  $a$  is an ancestor of  $j$ , and  $j$  is a descendant of  $a$ .



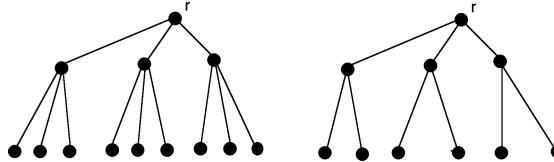
**Figure 3.2.6**

Many applications impose an upper bound on the number of children that a given vertex can have.

**DEFINITION:** An ***m*-ary tree** ( $m \geq 2$ ) is a rooted tree in which every vertex has  $m$  or fewer children.

**DEFINITION:** A ***complete m*-ary tree** is an *m*-ary tree in which every internal vertex has exactly  $m$  children and all leaves have the same depth.

**Example 3.2.3:** Figure 3.2.7 shows two *ternary* (3-ary) trees; the one on the left is complete; the other one is not.



**Figure 3.2.7** Two 3-ary trees, one complete and the other not complete.

**Proposition 3.2.1.** A complete *m*-ary tree has  $m^k$  vertices at level  $k$ .

**Proof:** The statement is trivially true for  $k = 1$ .

Assume as an induction hypothesis that there are  $m^l$  vertices at level  $l$ , for some  $l \geq 1$ . Since each of these vertices has  $m$  children, there are  $m \cdot m^l = m^{l+1}$  children at level  $l + 1$ .  $\diamond$

**Corollary 3.2.2.** An *m*-ary tree has at most  $m^k$  vertices at level  $k$ .  $\diamond$

**Theorem 3.2.3.** Let  $T$  be an  $n$ -vertex *m*-ary tree of height  $h$ . Then

$$h + 1 \leq n \leq \frac{m^{h+1} - 1}{m - 1}$$

**Proof:** Let  $n_k$  be the number of vertices at level  $k$ , so that  $1 \leq n_k \leq m^k$ , by Corollary 3.2.2. Thus,

$$h + 1 = \sum_{k=0}^h 1 \leq \sum_{k=0}^h n_k \leq \sum_{k=0}^h m^k = \frac{m^{h+1} - 1}{m - 1}$$

The result follows since  $\sum_{k=0}^h n_k = n$ .  $\diamond$

**Corollary 3.2.4.** The complete *m*-ary tree of height  $h$  has  $\frac{m^{h+1} - 1}{m - 1}$  vertices.  $\diamond$

### Isomorphism of Rooted Trees

**DEFINITION:** Two rooted trees are said to be **isomorphic as rooted trees** if there is a graph isomorphism between them that maps root to root.

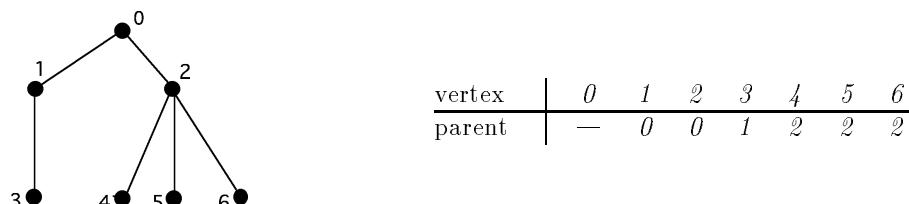
**Example 3.2.4:** Figure 3.2.8 illustrates two pairs of rooted trees. The two on the left are isomorphic as rooted trees; the two on the right are isomorphic trees, but they are not isomorphic as rooted trees. This shows that there are more isomorphism types of rooted trees than there are of trees.



**Figure 3.2.8** Isomorphic trees need not be isomorphic as rooted trees.

**COMPUTATIONAL NOTE: Representing a Rooted Tree:** If the  $n$  vertices of a rooted tree are labeled  $0, 1, \dots, n - 1$ , with the root assigned label 0, then a list whose  $k$ th entry is the parent of vertex  $k$  is called an **array-of-parents** representation. It fully specifies the rooted tree.

**Example 3.2.5:** Figure 3.2.9 shows the array-of-parents representation for a rooted tree with seven vertices.



**Figure 3.2.9** Representing a rooted tree with an array of parents.

### Ordered Trees

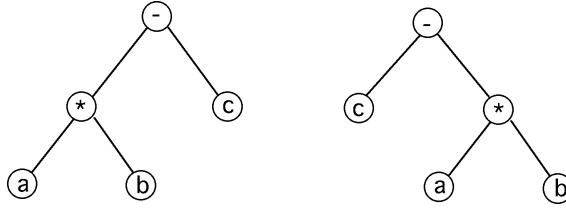
**DEFINITION:** An **ordered tree** is a rooted tree in which the children of each vertex are assigned a fixed ordering.

**DEFINITION:** In a **standard plane drawing** of an ordered tree, the root is at the top, the vertices at each level are horizontally aligned, and the left-to-right order of the vertices agrees with their prescribed order.

**Remark:** In an ordered tree, the prescribed *local ordering* of the children of each vertex extends to several possible *global orderings* of the vertices of the tree. One of them, the *level order*, is equivalent to reading the vertex names top to bottom, left to right in a standard plane drawing. Level order and three other global orderings, *pre-order*, *post-order*, and *in-order*, are explored in §3.3.

**Example 3.2.6:** Standard plane drawings of two different ordered trees are shown in Figure 3.2.10 below. As rooted trees, they are isomorphic, and one of the rooted-tree

isomorphisms preserves the vertex labeling. The ordered tree on the left stores the expression  $a * b - c$ , whereas the one on the right stores  $c - a * b$ .



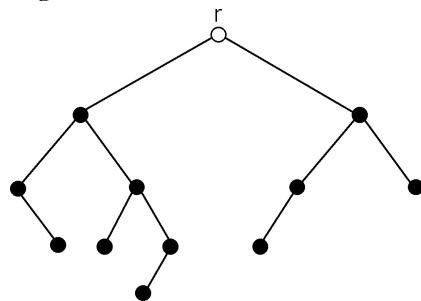
**Figure 3.2.10** Two different ordered trees that are isomorphic rooted trees.

**Remark:** The two ordered trees in Example 3.2.6 are a special type of 2-ary ordered tree known as a *binary tree*. The example previews an application that appears in §3.3, in which binary trees are used to store and produce arithmetic expressions.

### Binary Trees

Binary trees are among the most frequently used information structures in computer science. A variety of applications are given in §3.3 through §3.6.

**DEFINITION:** A **binary tree** is an ordered 2-ary tree in which each child is designated either a **left-child** or a **right-child**.

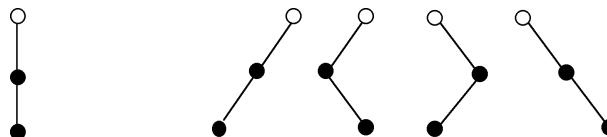


**Figure 3.2.11** A binary tree of height 4.

**DEFINITION:** The **left (right) subtree** of a vertex  $v$  in a binary tree is the binary subtree spanning the left (right)-child of  $v$  and all of its descendants.

The mandatory designation of left-child or right-child means that two different binary trees may be indistinguishable when regarded as ordered trees.

**Example 3.2.7:** In Figure 3.2.12, the four different binary trees shown on the right are all realizations of the ordered tree on the left. A formula for the number of different binary trees with a given number of vertices is developed in §3.8.

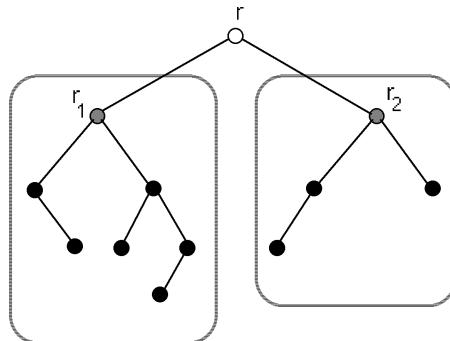


**Figure 3.2.12** Four different binary trees that are the same ordered tree.

A binary tree consists of a root and its left and right subtrees, which are themselves smaller binary trees. This *recursive* view makes mathematical induction the method of choice for proving many important facts about binary trees. Typically, the induction is on the height of the binary tree, because of the following recursive property.

**RECURSIVE PROPERTY OF A BINARY TREE:** If  $T$  is a binary tree of height  $h$ , then its left and right subtrees both have height less than or equal to  $h - 1$ , and equality holds for at least one of them.

**Example 3.2.8:** The binary tree shown in Figure 3.2.13 has height 4, and its left and right subtrees have heights 3 and 2, respectively.



**Figure 3.2.13** Recursive view of a binary tree.

The next two results provide an upper bound on the number of vertices in a binary tree of fixed height. The upper bound is fundamental to certain results in algorithmic analysis. Although Theorem 3.2.5 is an immediate consequence of Corollary 3.2.4, the inductive proof given here is instructive in its use of the recursive property given above.

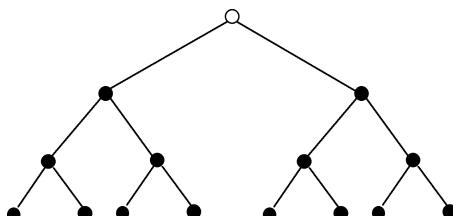
**Theorem 3.2.5.** *The complete binary tree of height  $h$  has  $2^{h+1} - 1$  vertices.*

**Proof:** The assertion is trivially true if  $h = 0$ . Assume for some  $k \geq 0$ , that a complete binary tree of height  $k$  has  $2^{k+1} - 1$  vertices, and let  $T$  be a complete binary tree of height  $k + 1$ . Since  $T$  is complete, its left and right subtrees, say  $T_1$  and  $T_2$ , must also be complete. Furthermore, trees  $T_1$  and  $T_2$  are both of height  $k$ , so by the induction hypothesis, they each contain  $2^{k+1} - 1$  vertices. Thus, the number of vertices of  $T$  is

$$1 + 2^{k+1} - 1 + 2^{k+1} - 1 = 2^{k+2} - 1 \quad \diamond$$

**Corollary 3.2.6.** *Every binary tree of height  $h$  has at most  $2^{h+1} - 1$  vertices.*  $\diamond$

**Example 3.2.9:** Figure 3.2.14 illustrates Theorem 3.2.5 for the case  $h = 3$ .



**Figure 3.2.14** The complete binary tree of height 3 has 15 vertices.

**EXERCISES for Section 3.2**

**3.2.1<sup>s</sup>** Draw all the rooted tree types with 3 or 4 vertices. How many different graph isomorphism types do they represent?

**3.2.2** Draw all the rooted tree types with 5 vertices. How many different graph isomorphism types do they represent?

**3.2.3<sup>s</sup>** Draw all possible binary trees of height 2, and group these into classes of isomorphic rooted trees.

**3.2.4** Draw all possible binary trees of height 3 whose internal vertices have exactly two children. Group these into classes of isomorphic rooted trees.

*In Exercises 3.2.5 through 3.2.13, draw the specified tree(s) or explain why no such tree(s) can exist.*

**3.2.5<sup>s</sup>** A 12-vertex binary tree of height 5 that has exactly five leaf vertices.

**3.2.6** A 12-vertex binary tree of height 4.

**3.2.7** A 14-vertex binary tree of height 3.

**3.2.8<sup>s</sup>** A 14-vertex tree such that every leaf vertex has exactly three proper ancestors.

**3.2.9** Three different 6-vertex rooted trees whose underlying graphs are isomorphic.

**3.2.10** A ternary tree of height 3 with exactly 10 vertices.

**3.2.11** A ternary tree of height 3 with exactly 28 leaf vertices.

**3.2.12** A ternary tree of height 3 with exactly four vertices.

**3.2.13<sup>s</sup>** A ternary tree of height 3 with exactly 13 vertices, such that every internal vertex has exactly three children.

**3.2.14** Prove that a directed tree that has more than one vertex with indegree 0 cannot be a rooted tree.

**3.2.15** Represent the folder and subfolder organization in your computer as a rooted tree.

**3.2.16** What is the relationship between the depth of a vertex in a rooted tree and the number of ancestors of  $v$ ? Prove your answer.

**3.2.17<sup>s</sup>** How many different non-complete ternary trees (i.e., non-isomorphic as rooted trees) are there of height 3 such that every internal vertex has exactly three children?

**3.2.18** Demonstrate the semantic power of the comma by using a rooted tree to parse the sentence “The Republicans, say the Democrats, are crooks.” in two ways, one with the commas and one without.

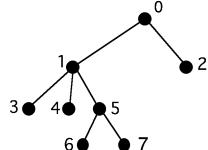
*In Exercises 3.2.19 and 3.2.20, draw the rooted tree specified by the given array of parents.*

**3.2.19**  $- , 0, 1, 1, 1, 1, 2, 2, 2.$

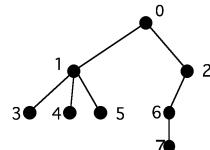
**3.2.20**  $- , 0, 1, 1, 2, 2, 3, 4, 4.$

*In Exercises 3.2.21 and 3.2.22, specify the given rooted tree with an array of parents.*

**3.2.21<sup>s</sup>**



**3.2.22**



**3.2.23<sup>s</sup>** Let  $f(h)$  be the number of nodes in a complete binary tree of height  $h$ . Express  $f(h)$  in terms of  $f(h - 1)$  without appealing to the result of Theorem 3.2.5. Then use this *recurrence relation* to simplify the induction step in the proof of Theorem 3.2.5.

**3.2.24** Instead of using Theorem 3.2.5, prove Corollary 3.2.6 directly with the “strong form” of induction, using the following weaker induction hypothesis: For some  $h > 0$  and all  $k \leq h - 1$ , a binary tree of height  $k$  has at most  $2^{k+1} - 1$  vertices.

### 3.3 BINARY-TREE TRAVERSALS

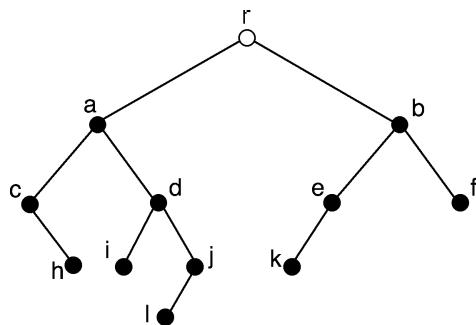
A systematic visit of each vertex of a tree is called a *tree traversal*. Thus, a traversal of a binary tree is a global ordering of its vertices. Four kinds of traversals are presented in this section: *level-order*, *pre-order*, *post-order*, and *in-order*.

#### Level-Order Traversal

**DEFINITION:** The *level-order* of an ordered tree is a listing of the vertices in the top-to-bottom, left-to-right order of a standard plane drawing of that tree.

Thus, the level-order is consistent with the prescribed local ordering of the children of each vertex in the ordered tree.

**Example 3.3.1:** The level-order of the labeled binary tree in Figure 3.3.1 is  $r, a, b, c, d, e, f, h, i, j, k, l$ .



**Figure 3.3.1** Level-order:  $r, a, b, c, d, e, f, h, i, j, k, l$ .

#### Pre-Order, Post-Order, and In-Order Traversals

The pre-order, post-order, and in-order traversals are defined using the recursive view of a binary tree introduced in §3.2 (p.130). Then a geometric description is given for each one, and at the end of the section, algorithmic descriptions using stacks are presented. The vertex sequence given for each of the traversals refers to the binary tree in Figure 3.3.1.

**DEFINITION:** The **left pre-order traversal** of a binary tree  $T$  is defined recursively as follows:

- i. List (process) the root of  $T$ .
- ii. Perform a left pre-order traversal of the left subtree of  $T$ .
- iii. Perform a left pre-order traversal of the right subtree of  $T$ .

**Example 3.3.1, continued: Left Pre-Order:**  $r, a, c, h, d, i, j, l, b, e, k, f$

**DEFINITION:** The **post-order traversal** of a binary tree  $T$  is defined recursively as follows:

- i. Perform a post-order traversal of the left subtree of  $T$ .
- ii. Perform a post-order traversal of the right subtree of  $T$ .
- iii. List (process) the root of  $T$ .

**Example 3.3.1, continued: Post-Order:**  $h, c, i, l, j, d, a, k, e, f, b, r$

**DEFINITION:** The **in-order traversal** of a binary tree  $T$  is defined recursively as follows:

- i. Perform an in-order traversal of the left subtree of  $T$ .
- ii. List (process) the root of  $T$ .
- iii. Perform an in-order traversal of the right subtree of  $T$ .

**Example 3.3.1, continued: In-Order:**  $c, h, a, i, d, l, j, r, k, e, b, f$

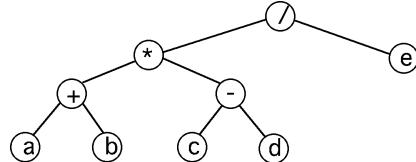
### Processing Arithmetic Expressions

An expression tree is the most natural information structure for storing and processing arithmetic expressions.

**DEFINITION:** An **expression tree** for an arithmetic expression is either a single vertex labeled by an identifier, or is described by the following recursive rule:

- The operator to be executed during an evaluation is at the root.
- The left subtree of the root is an expression tree representing the left operand.
- The right subtree is an expression tree representing the right operand.

**Example 3.3.2:** The expression tree in Figure 3.3.2 stores the expression  $((a + b) * (c - d))/e$ .



**Figure 3.3.2** An expression tree for  $((a + b) * (c - d))/e$ .

**Application 3.3.1 Prefix, Postfix, and Infix Notation:** When an expression tree is traversed, the resulting expression will appear in either *prefix*, *postfix*, or *infix* notation, according to which of the traversals is used.

**Example 3.3.3:** For the expression tree in Figure 3.3.2, the pre-order, post-order, and in-order traversals yield, respectively, the following.

prefix:  $/ * + ab - cde$

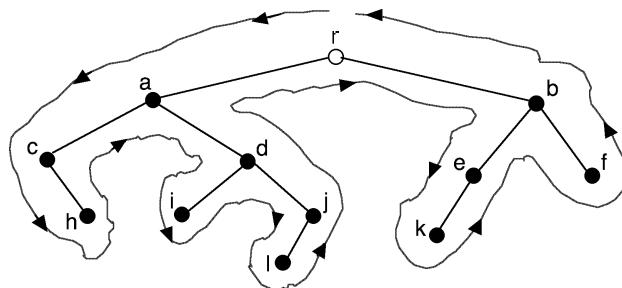
postfix:  $ab + cd - *e /$

infix:  $((a + b) * (c - d)) / e$

The infix expression requires parentheses, which can be incorporated into the recursive in-order traversal by producing a left parenthesis at the beginning of each subtree traversal and producing a right parenthesis at the end of each subtree traversal.

### Geometric Descriptions of Pre-order, Post-Order, and In-Order Traversals

The three traversals can be described geometrically in terms of a *left-first walk* around a standard plane drawing of the binary tree. A left-first walk may be imagined by starting at the root, and walking alongside the edges of the tree so that the edge you are passing along is always on your left (see Figure 3.3.3).



**Figure 3.3.3** A left-first walk around the tree.

**Pre-Order Traversal:**  $r, a, c, h, d, i, j, l, b, e, k, f$

As you take a left-first walk around the tree, list each vertex the first time you see it, but not again thereafter.

**Post-Order Traversal:**  $h, c, i, l, j, d, a, k, e, f, b, r$

While taking a left-first walk around the tree, list each vertex the last time you see it and not beforehand.

**In-Order Traversal:**  $c, h, a, i, d, l, j, r, k, e, b, f$

While taking a left-first walk around the tree, list each node with a left-child the second time you see it, and list each other node the first time you see it.

### Stack Implementations of Pre-Order, Post-Order, and In-Order Traversals

**DEFINITION:** A **stack** is a sequence of elements such that each new element is added (or **pushed**) onto one end, called the *top*, and an element is removed (or **popped**) from that same end.

These iterative versions of the three traversals use the functions *pop(stack)* and *top(stack)*. Both functions return the value at the top of the stack, but *pop* removes that value from the stack, whereas *top* does not.

**Algorithm 3.3.1: Left Pre-Order**

*Input:* a binary tree.

*Output:* a list of the vertices in left pre-order.

```

Push root onto stack.
While stack is not empty
    Pop a vertex off stack, and write it on the output list.
    Push its children right-to-left onto stack.

```

**Algorithm 3.3.2: Post-Order**

*Input:* a binary tree.

*Output:* a list of the vertices in post-order.

```

Push root onto stack.
While stack is not empty
    If top(stack) is unmarked
        Mark it, and push its children right-to-left onto stack.
    Else
        v := pop(stack).
        List v.

```

**Algorithm 3.3.3: In-Order**

*Input:* a binary tree.

*Output:* a list of the vertices in in-order.

```

Push root onto stack.
While stack is not empty
    v := top(stack).
    While v has a left-child
        Push leftchild(v) onto stack.
        v := leftchild(v).
    v := pop(stack).
    List v.
    If v has a right-child,
        Push rightchild(v) onto stack.
        v := rightchild(v).
    Else
        While stack not empty and v has no right-child
            v := pop(stack).
            List v.
        If v has a right-child,
            Push rightchild(v) onto stack.
            v := rightchild(v).

```

**Queue Implementation of Level-Order Traversal**

DEFINITION: A **queue** is a sequence of elements such that each new element is added (**enqueued**) to one end, called the *back* of the queue, and an element is removed (**dequeued**) from the other end, called the *front*.

**Algorithm 3.3.4: Level-Order: Top-to-Bottom, Left-to-Right***Input:* A binary tree.*Output:* A list of the vertices in level-order.Enqueue *root*.

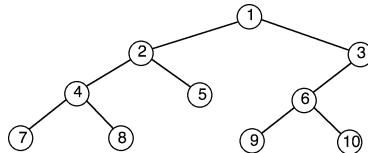
While queue is not empty

Dequeue a vertex and write it on the output list.

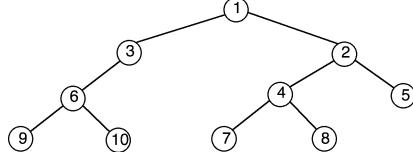
Enqueue its children left-to-right.

**EXERCISES for Section 3.3**

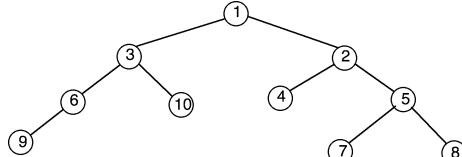
*In Exercises 3.3.1 through 3.3.3, give the level-order, pre-order, in-order, and post-order traversals of the given binary tree.*

3.3.1<sup>s</sup>

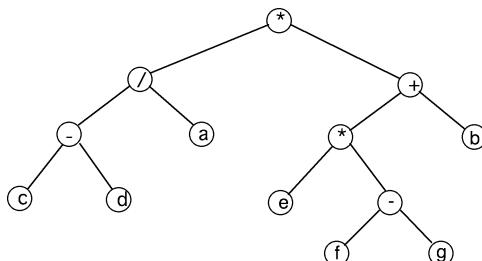
3.3.2



3.3.3



3.3.4 Give the prefix, infix, and postfix notations for the arithmetic expression represented by the following expression tree.



*In Exercises 3.3.5 through 3.3.9, represent the given arithmetic expression by an expression tree. Then give the prefix and postfix notations for the arithmetic expression by performing the appropriate tree traversal.*

3.3.5<sup>s</sup>  $(a + b) \times (c - (d + e))$ 3.3.6  $((a + (b \times c)) - d)/g$ 3.3.7  $((a + b) \times (c - (d + e))) \times (((a + (b \times c)) - d)/g)$ 3.3.8  $((b \times (c + a) \times (g - d))/(b + d)) \times ((g + c) \times h)$ 3.3.9  $((((a + b) \times c) - (d + e)) \times (((a + (b \times c)) - d)/g)$

**3.3.10** Give an example of two 4-vertex binary trees whose level-order and pre-order traversals are  $a, b, c, d$  but whose post-order is not.

**3.3.11** Prove that two binary trees that have the same level-order and the same post-order must be equivalent as 2-ary trees. (Hint: show that the information obtained from a post-order traversal uniquely determines the parent of each vertex.)

**3.3.12** Show that a post-order traversal is equivalent to taking a *right pre-order* traversal (i.e., taking a right-first walk around the tree) and reversing the order of the output list.

**3.3.13** Reconcile the symmetry suggested by Exercise 3.3.12 with the asymmetry suggested by Exercises 3.3.10 and 3.3.11.

**3.3.14** [Computer Project] Implement Algorithm 3.3.1 and test it on each of the binary trees in Exercises 3.3.1 through 3.3.3.

**3.3.15** [Computer Project] Implement Algorithm 3.3.2 and test it on each of the binary trees in Exercises 3.3.1 through 3.3.3.

**3.3.16** [Computer Project] Implement Algorithm 3.3.3 and test it on each of the binary trees in Exercises 3.3.1 through 3.3.3.

**3.3.17** [Computer Project] Implement recursive versions of the pre-order, post-order, and in-order traversal algorithms based on the definitions given at the beginning of the section. Test them on each of the binary trees in Exercises 3.3.1 through 3.3.3.

## 3.4 BINARY-SEARCH TREES

**DEFINITION:** A **random-access table** is a database whose domain is a sequence of entries, where each entry consists of two fields. One field is for the actual data element, and the other one is for the **key**, whose value determines that entry's position in the database.

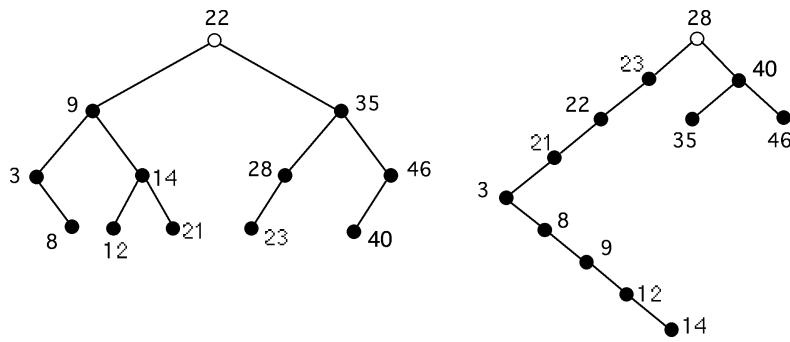
Thus, an entry is located in a random-access table by searching for its key. The most generally useful implementation of a random-access table uses the following information structure.

**DEFINITION:** A **binary-search tree** (BST) is a binary tree, each of whose vertices is assigned a key, such that the key assigned to any vertex  $v$  is greater than the key at each vertex in the left subtree of  $v$ , and is less than the key at each vertex in the right subtree of  $v$ .

**Example 3.4.1:** Both of the binary-search trees in Figure 3.4.1 store the keys:

$$3, 8, 9, 12, 14, 21, 22, 23, 28, 35, 40, 46$$

Notice that the smallest key in either BST can be found by starting at the root and proceeding to the left until you reach a vertex with no left-child. Similarly, the largest key can be found by proceeding from the root iteratively to the right as long as possible. A straightforward inductive proof can be used to show that these two properties hold for an arbitrary binary-search tree (see Exercises).



**Figure 3.4.1** Two binary-search trees storing the same set of keys.

Iterative and recursive versions of the binary-search algorithm are shown below. The algorithm is based on the following simple strategy.

- In each iteration, exclude either the left or right subtree from the rest of the search, depending on whether the target key is less than or greater than the key at the current vertex.

**Remark:** An apology is due to the computer scientists because the versions below blur the distinction between a variable and a pointer to that variable. For instance, the variable  $T$  is typically a pointer to the root of the binary tree that is called  $T$ . Here, instead,  $\text{root}(T)$  is used to refer to the root of the binary tree  $T$ . Of course, any implementation requires a careful treatment of the pointer datatype, but for the purposes here, pointers are avoided for fear of obscuring the essence of a fundamentally important graph algorithm.

**Algorithm 3.4.1: Binary-Search-Tree Search**

*Input:* a binary-search tree  $T$  and a target key  $t$ .  
*Output:* a vertex  $v$  of  $T$  such that  $\text{key}(v) = t$  if  $t$  is found,  
 or  $\text{NULL}$  if  $t$  is not found.

```

 $v := \text{root}(T)$ 
While ( $v \neq \text{NULL}$ ) and ( $t \neq \text{key}(v)$ )
    If  $t > \text{key}(v)$ 
         $v := \text{rightchild}(v)$ 
    Else  $v := \text{leftchild}(v)$ 
Return  $v$ .
```

**Algorithm 3.4.2: Recursive BSTsearch( $T, t$ )**

*Input:* a binary-search tree  $T$  and a target key  $t$ .  
*Output:* a vertex  $v$  of  $T$  such that  $\text{key}(v) = t$  if  $t$  is found,  
 or  $\text{NULL}$  if  $t$  is not found.

```

If  $\text{root}(T) = \text{NULL}$ 
    Return  $\text{NULL}$ 
Else If  $t = \text{key}(\text{root})$ 
    Return  $\text{root}$ 
Else If  $t > \text{key}(\text{root})$ 
    BSTsearch( $\text{rightsubtree}(T), t$ )
Else BSTsearch( $\text{leftsubtree}(T), t$ )
```

### Balanced Binary-Search Trees

If each of the vertices of a binary-search tree has left and right subtrees of approximately equal size, then each comparison in the algorithm will rule out roughly half of the vertices remaining to be searched. This property motivates the following definition, which is a generalization of a complete binary tree.

**DEFINITION:** A binary tree is **balanced** if for every vertex, the number of vertices in its left and right subtrees differ by at most one.

**Example 3.4.2:** In Figure 3.4.1, the binary tree on the left is balanced; the one on the right is quite unbalanced. Observe that the binary-search algorithm performs faster on the balanced one. For instance, it takes four comparisons to determine that 20 is not one of the keys stored in the tree on the left, but it takes nine comparisons for the tree on the right.

**COMPUTATIONAL NOTE:** The **search** operation is one of the *primary operations* of the *random-access table* datatype. Two others are the **insert** and **delete** operations. An evaluation of a particular implementation of the random-access table should include an analysis of the computation required for each of these three primary operations.

Three different software implementations of the random-access table of  $n$  elements are as follows:

- Sorted array of records
- Sorted linked list
- Binary-search tree

Searching a sorted array using a binary search requires at worst  $O(\log_2 n)$  computations, since each comparison eliminates half of the elements. A search through a linked list must be sequential and therefore is, in the worst case,  $O(n)$ .

Since each comparison of a binary search performed on a binary-search tree moves the search down to the next level, the number of comparisons is at most the height of the tree plus one. If the tree is balanced, then it is not hard to show that the number of vertices  $n$  is between  $2^h$  and  $2^{h+1}$ . Hence, the worst-case performance of the binary search on a perfectly balanced binary-search tree is  $O(\log_2 n)$ . The other extreme occurs when each internal vertex of the binary tree has only one child. Such a binary tree is actually an ordinary linked list, and therefore the performance of the search degenerates to  $O(n)$ .

If the random-access table is relatively *static*, that is, if insertions or deletions are infrequent, then the array implementation is the clearcut winner. However, if insertions and deletions occur regularly, then the computational cost of these two operations must be taken into account.

For the array implementation, it will require  $O(\log_2 n)$  comparisons to find where the insertion or deletion must occur, but then  $O(n)$  movements of data are needed either to make room for the new element or to close the gap for the deleted element. The linked-list implementation requires  $O(n)$  comparisons to find where the insertion or deletion must occur, but then a reassignment of a few pointer values ( $O(1)$ ) is all that is needed to perform the actual insertion or deletion. Thus, both the array and the linked-list implementations perform the insertion and deletion in  $O(n)$  instruction executions.

The binary-search tree combines the best characteristics of the other two implementations to achieve  $O(\log_2 n)$  execution time for the search.

### Insertions and Deletions in a Binary-Search Tree

A recursive description of the insert operation is shown below. From an iterative view, a binary search is performed until it terminates at a leaf  $w$ . Then the new key  $k$  is assigned to a new vertex  $v$ , which becomes the left-child or right-child of  $w$ , depending on the comparison of  $k$  with  $\text{key}(w)$ .

**Algorithm 3.4.3: Binary-Search-Tree Insert( $T, x$ )**

*Input:* a binary-search tree  $T$  and an entry  $x$   
*Output:* tree  $T$  with a new leaf  $v$  such that  $\text{entry}(v) = x$

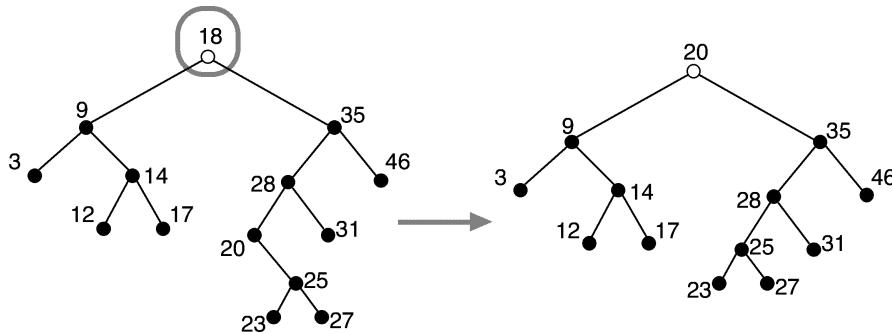
```
If  $\text{root}(T) = \text{NULL}$ 
    Create a vertex  $v$  with  $\text{entry}(v) = x$ .
     $\text{root}(T) := v$ 
Else If  $\text{key}(x) < \text{key}(\text{root})$ 
    Insert( $\text{leftsubtree}(T), x$ )
Else Insert( $\text{rightsubtree}(T), x$ )
```

The deletion operation requires a bit more care. If an internal vertex  $v$  is deleted, then the way in which the two resulting pieces are pasted back together falls into one of three cases.

Case 1: If  $v$  has no left-child, then  $v$  is replaced by the right subtree of  $v$ .

Case 2: If  $v$  has no right-child, then  $v$  is replaced by the left subtree of  $v$ .

Case 3: If  $v$  has both a left-child and a right-child, then  $v$  is replaced by the leftmost (minimum) element  $l$  of the right subtree of  $v$ , and  $l$  is replaced by its right subtree (it cannot have a left subtree). It is straightforward to show that the resulting tree still satisfies the binary-search-tree property.



**Figure 3.4.2** Deleting a vertex having two children.

### EXERCISES for Section 3.4

In Exercises 3.4.1 through 3.4.4, draw three different binary-search trees for the given list of keys, two that are balanced and one that is not.

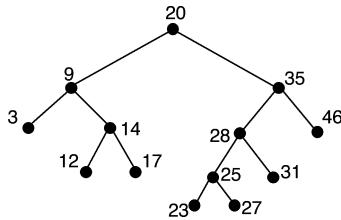
3.4.1<sup>s</sup> 1, 4, 9, 19, 41, 49.

3.4.2 2, 3, 5, 7, 15, 20, 30, 40, 45, 50.

3.4.3 12, 23, 35, 37, 45, 50, 54, 60, 66.

3.4.4 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144.

In Exercises 3.4.5 through 3.4.10, perform the specified sequence of insertions and deletions on the following search tree. After each individual insertion or deletion, draw the resulting search tree.



3.4.5<sup>s</sup> Insert 10, insert 15, delete 20.

3.4.6 Insert 10, delete 20, insert 15.

3.4.7 Delete 35, delete 28, insert 35.

3.4.8 Delete 28, insert 35, delete 9.

3.4.9 Insert 7, delete 9, insert 5.

3.4.10 Delete 9, insert 37, insert 5.

3.4.11<sup>s</sup> Find a sequence of the keys 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144 such that their insertion by Algorithm 3.4.3, in that order, results in a balanced binary-search tree. Begin the sequence of insertions with the null tree.

3.4.12 How many different insertion sequences of the keys 1, 4, 9, 19, 41, 49 are there that result in a balanced search tree?

3.4.13<sup>s</sup> When will an insertion sequence for a set of  $n$  keys result in the worst possible search tree?

3.4.14 Give an iterative description of the insert operation.

3.4.15 Show that Case 3 of the deletion operation can also be accomplished by replacing  $v$  by the rightmost element  $r$  of the left subtree of  $v$ , and by then replacing  $r$  by the left subtree of  $r$ .

3.4.16 Use induction to prove that the smallest key in a binary-search tree can be found by starting at the root and proceeding to the left until you reach a vertex with no left-child.

3.4.17 [Computer Project] Implement Algorithm 3.4.2, and test it on the balanced search tree in Figure 3.4.1 by searching for 23 and then for 25.

## 3.5 HUFFMAN TREES AND OPTIMAL PREFIX CODES

### Binary Codes

The existence of fast two-state materials makes sequences of 0's and 1's (*bitstrings*) the most natural way of encoding information for computers. Every upper- and lower-case English letter, punctuation mark, and mathematical symbol that can occur (as well as blank, \$, etc.) is encoded as a different bitstring. The most common such encoding is

known as ASCII code. Each of the letters and other symbols is regarded as the *meaning* of its ASCII code.

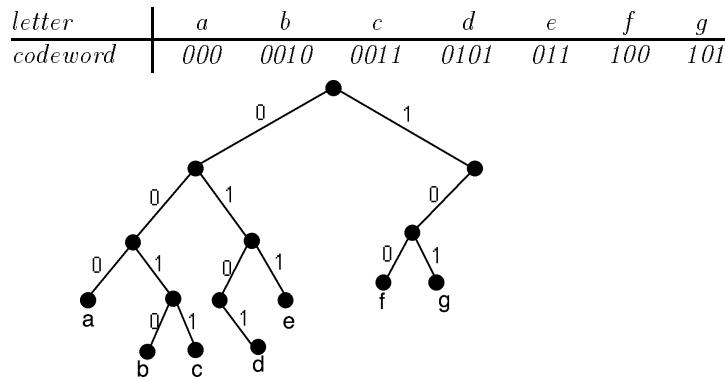
**DEFINITION:** A **binary code** is an assignment of symbols or other meanings to a set of bitstrings. Each bitstring is referred to as a **codeword**.

For some applications, it may be desired to allow codewords to vary in length. Confusion might result, in scanning a string from left to right, if one codeword were also part of another codeword. The following property, referred to as the *prefix property*, avoids this kind of ambiguity.

**DEFINITION:** A **prefix code** is a binary code with the property that no codeword is an initial substring of any other codeword.

**Application 3.5.1** *Constructing Prefix Codes:* A binary tree can be used to construct a prefix code for a given set of symbols. First, draw an arbitrary binary tree whose leaves are bijectively labeled by the symbols (i.e., each leaf gets a different symbol). Next, label every edge that goes to a left-child with a 0, and label every edge to a right-child with a 1. Then each symbol corresponds to the codeword formed by the sequence of edge labels on the path from the root to the leaf labeled by that symbol. It is not hard to show that the resulting set of codewords satisfies the prefix property.

**Example 3.5.1:** Suppose that each relevant message can be expressed as a string of letters (repetitions allowed) drawn from the restricted alphabet  $\{a, b, c, d, e, f, g\}$ . The binary tree shown in Figure 3.5.1 represents the prefix code whose seven codewords correspond to the unique paths from the root to each of the seven leaves. The resulting encoding scheme is shown below.



**Figure 3.5.1** A binary tree used to construct a prefix code.

### Huffman Codes

In a prefix code that uses its shorter codewords to encode the more frequently occurring symbols, the messages will tend to require fewer bits than in a code that does not. For instance, in a prefix code for ordinary English prose, it would make sense to use a short codeword to represent the letter “e” and a longer codeword to represent “x”.

This suggests that one measure of a code’s efficiency be the *average weighted length* of its codewords, where the length of each codeword is multiplied by the frequency of the symbol it encodes.

**Example 3.5.2:** Suppose that the frequency for each letter of the restricted alphabet of the previous example is given by the following table.

letter	a	b	c	d	e	f	g
frequency	.2	.05	.1	.1	.25	.15	.15

Then the average weighted length of a codeword for the prefix code in Figure 3.5.1 is

$$3 \times .2 + 4 \times .05 + 4 \times .1 + 4 \times .1 + 3 \times .25 + 3 \times .15 + 3 \times .15 = 3.25$$

For a prefix code constructed from a binary tree, as described above, the length of each codeword is simply the depth of its corresponding leaf. This motivates the next definition, which is used in the discussion that follows.

**DEFINITION:** Let  $T$  be a binary tree with leaves  $s_1, s_2, \dots, s_l$ , such that each leaf  $s_i$  is assigned a weight  $w_i$ . Then the **average weighted depth** of the binary tree  $T$ , denoted  $wt(T)$ , is given by

$$wt(T) = \sum_{i=1}^l depth(s_i) \cdot w_i$$

Thus, if the weight assigned to a leaf is the frequency of the symbol that labels that leaf, then the average weighted length of a codeword equals the average weighted depth of the binary tree.

**Application 3.5.2 Constructing Efficient Codes — the Huffman Algorithm:** The following algorithm, developed by David Huffman [Hu52], constructs a prefix code whose codewords have the smallest possible average weighted length.

**Algorithm 3.5.1: Huffman Prefix Code**

*Input:* a set  $\{s_1, \dots, s_l\}$  of symbols; a list  $\{w_1, \dots, w_l\}$  of weights, where  $w_i$  is the weight associated with symbol  $s_i$ .

*Output:* a binary tree representing a prefix code for a set of symbols whose codewords have minimum average weighted length.

Initialize  $F$  to be a forest of isolated vertices, labeled  $s_1, \dots, s_l$ , with respective weights  $w_1, \dots, w_l$ .

For  $i = 1$  to  $l - 1$

    Choose from forest  $F$  two trees,  $T$  and  $T'$ , of smallest weights in  $F$ .

    Create a new binary tree whose root has  $T$  and  $T'$  as its left and right subtrees, respectively.

    Label the edge to  $T$  with a 0 and the edge to  $T'$  with a 1.

    Assign to the new tree the weight  $w(T) + w(T')$ .

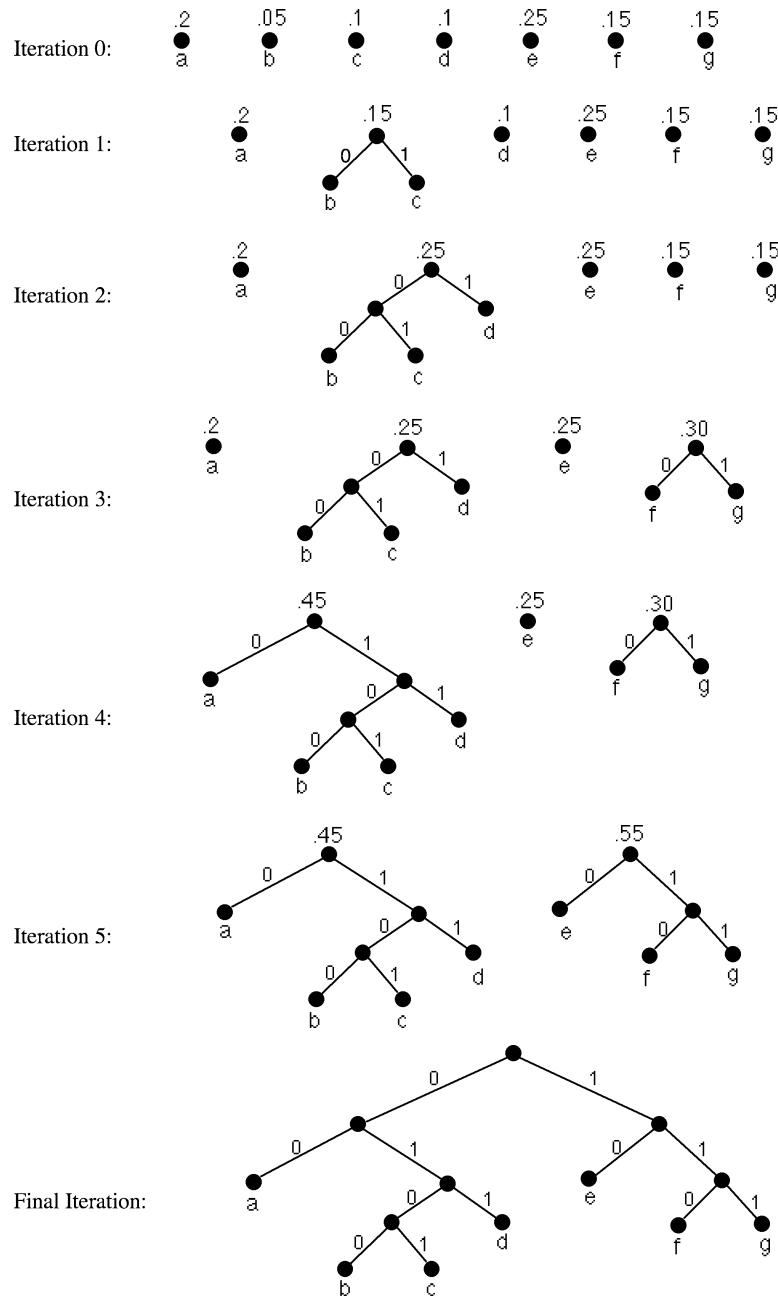
    Replace trees  $T$  and  $T'$  in forest  $F$  by the new tree.

Return  $F$ .

**COMPUTATIONAL NOTE:** In choosing the two trees of smallest weights, ties are resolved by some default ordering of the trees in the forest. In Example 3.3.6, we use a left-to-right ordering.

**DEFINITION:** The binary tree produced from Algorithm 3.5.1 is called the **Huffman tree** for the list of symbols, and its corresponding prefix code is called the **Huffman code**.

**Example 3.5.3:** The Huffman algorithm is applied to the symbols and frequencies given in Example 3.5.2 to obtain an optimal prefix code. For each iteration, the current set of binary trees is displayed, and the root of each binary tree is labeled with the weight assigned to that tree.



The resulting Huffman code has the following encoding scheme.

letter	a	b	c	d	e	f	g
codeword	00	0100	0101	011	10	110	111

The average length of a codeword for this prefix code is

$$2 \times .2 + 4 \times .05 + 4 \times .1 + 3 \times .1 + 2 \times .25 + 3 \times .15 + 3 \times .15 = 2.7$$

The Huffman tree also provides an efficient decoding scheme. A given codeword determines the unique path from the root to the leaf that stores the corresponding symbol. As the codeword is scanned from left to right, the path is traced downward from the root by traversing 0-edges or 1-edges, according to each bit. When a leaf is reached, its corresponding symbol is recorded. The next bit begins a new path from the root to the next symbol. The process continues until all the bits in the codeword have been read.

The next assertion, which can be proved by induction, is used in showing that the Huffman algorithm produces an optimal prefix code.

**Lemma 3.5.1.** *If the leaves of a binary tree are assigned weights, and if each internal vertex is assigned a weight equal to the sum of its children's weights, then the tree's average weighted depth equals the sum of the weights of its internal vertices.*

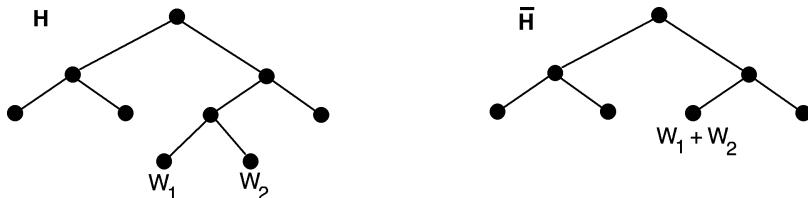
◇ (See Exercises.)

**Theorem 3.5.2.** *For a given list of weights  $w_1, w_2, \dots, w_l$ , a Huffman tree has the smallest possible average weighted depth among all binary trees whose leaves are assigned those weights.*

**Proof:** The proof uses induction on the number  $l$  of weights. If  $l = 2$ , then the Huffman tree consists only of a root with weight  $w_1 + w_2$  and a left-child and a right-child. Any other binary tree with two leaves has at least two internal vertices, one with weight  $w_1 + w_2$ , and by Lemma 3.5.1, has greater average weighted depth.

Assume for some  $l \geq 2$  that the Huffman algorithm produces a Huffman tree of minimum average weighted depth for any list of  $l$  weights. Let  $w_1, w_2, \dots, w_{l+1}$  be any list of  $l+1$  weights, and assume that  $w_1$  and  $w_2$  are two of the smallest ones, chosen first by the algorithm. Then the Huffman tree  $H$  that results consists of a Huffman tree  $\bar{H}$  for the weights  $w_1 + w_2, w_3, \dots, w_{l+1}$  where the leaf of weight  $w_1 + w_2$  is replaced by a vertex with leaves assigned weights  $w_1$  and  $w_2$  (see Figure 3.5.2). Lemma 3.5.1 implies  $wt(H) = wt(\bar{H}) + w_1 + w_2$ . By the induction hypothesis,  $\bar{H}$  is optimal among all binary trees whose leaves are assigned weights  $w_1 + w_2, w_3, \dots, w_{l+1}$ .

Now suppose  $T^*$  is an optimal binary tree for the weights  $w_1, w_2, \dots, w_{l+1}$ . Let  $x$  be an internal vertex of  $T^*$  of greatest depth and suppose that  $y$  and  $z$  are its left-child and right-child, respectively. Without loss of generality, we can assume that  $y$  and  $z$  have weight  $w_1$  and  $w_2$ , since otherwise we could swap their weights with  $w_1$  and  $w_2$  to produce a tree with smaller average weighted depth. Consider the tree  $\bar{T}$  formed by deleting the leaves  $y$  and  $z$  from  $T^*$ . Then  $wt(T^*) = wt(\bar{T}) + w_1 + w_2$ . But  $\bar{T}$  is a binary tree with leaves of weight  $w_1 + w_2, w_3, \dots, w_{l+1}$ , and, hence,  $wt(\bar{T}) \geq wt(\bar{H})$ . Thus,  $wt(T^*) \geq wt(H)$ , proving  $H$  is optimal. ◇



**Figure 3.5.2** Huffman trees  $H$  and  $\bar{H}$  in proof.

### EXERCISES for Section 3.5

*In Exercises 3.5.1 through 3.5.3, use the Huffman tree constructed in Example 3.5.3 to decode the given string.*

3.5.1<sup>s</sup> 1100001010001110.

3.5.2 010000111110011110.

3.5.3 0111011000010110011.

*In Exercises 3.5.4 through 3.5.6, construct a Huffman code for the given list of symbols and weights, calculate its average weighted length, and encode the strings “defaced” and “baggage”. Use the left-to-right ordering to break ties.*

3.5.4	<table style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left; padding-bottom: 2px;">letter</th><th style="padding-bottom: 2px;">a</th><th style="padding-bottom: 2px;">b</th><th style="padding-bottom: 2px;">c</th><th style="padding-bottom: 2px;">d</th><th style="padding-bottom: 2px;">e</th><th style="padding-bottom: 2px;">f</th><th style="padding-bottom: 2px;">g</th><th style="padding-bottom: 2px;">h</th></tr> </thead> <tbody> <tr> <td style="text-align: left; vertical-align: bottom;">frequency</td><td style="text-align: center; vertical-align: bottom;">.2</td><td style="text-align: center; vertical-align: bottom;">.05</td><td style="text-align: center; vertical-align: bottom;">.1</td><td style="text-align: center; vertical-align: bottom;">.1</td><td style="text-align: center; vertical-align: bottom;">.18</td><td style="text-align: center; vertical-align: bottom;">.15</td><td style="text-align: center; vertical-align: bottom;">.15</td><td style="text-align: center; vertical-align: bottom;">.07</td></tr> </tbody> </table>	letter	a	b	c	d	e	f	g	h	frequency	.2	.05	.1	.1	.18	.15	.15	.07
letter	a	b	c	d	e	f	g	h											
frequency	.2	.05	.1	.1	.18	.15	.15	.07											

3.5.5	<table style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left; padding-bottom: 2px;">letter</th><th style="padding-bottom: 2px;">a</th><th style="padding-bottom: 2px;">b</th><th style="padding-bottom: 2px;">c</th><th style="padding-bottom: 2px;">d</th><th style="padding-bottom: 2px;">e</th><th style="padding-bottom: 2px;">f</th><th style="padding-bottom: 2px;">g</th><th style="padding-bottom: 2px;">h</th></tr> </thead> <tbody> <tr> <td style="text-align: left; vertical-align: bottom;">frequency</td><td style="text-align: center; vertical-align: bottom;">.1</td><td style="text-align: center; vertical-align: bottom;">.15</td><td style="text-align: center; vertical-align: bottom;">.2</td><td style="text-align: center; vertical-align: bottom;">.17</td><td style="text-align: center; vertical-align: bottom;">.13</td><td style="text-align: center; vertical-align: bottom;">.15</td><td style="text-align: center; vertical-align: bottom;">.05</td><td style="text-align: center; vertical-align: bottom;">.05</td></tr> </tbody> </table>	letter	a	b	c	d	e	f	g	h	frequency	.1	.15	.2	.17	.13	.15	.05	.05
letter	a	b	c	d	e	f	g	h											
frequency	.1	.15	.2	.17	.13	.15	.05	.05											

3.5.6	<table style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left; padding-bottom: 2px;">letter</th><th style="padding-bottom: 2px;">a</th><th style="padding-bottom: 2px;">b</th><th style="padding-bottom: 2px;">c</th><th style="padding-bottom: 2px;">d</th><th style="padding-bottom: 2px;">e</th><th style="padding-bottom: 2px;">f</th><th style="padding-bottom: 2px;">g</th><th style="padding-bottom: 2px;">h</th></tr> </thead> <tbody> <tr> <td style="text-align: left; vertical-align: bottom;">frequency</td><td style="text-align: center; vertical-align: bottom;">.15</td><td style="text-align: center; vertical-align: bottom;">.1</td><td style="text-align: center; vertical-align: bottom;">.15</td><td style="text-align: center; vertical-align: bottom;">.12</td><td style="text-align: center; vertical-align: bottom;">.08</td><td style="text-align: center; vertical-align: bottom;">.25</td><td style="text-align: center; vertical-align: bottom;">.05</td><td style="text-align: center; vertical-align: bottom;">.1</td></tr> </tbody> </table>	letter	a	b	c	d	e	f	g	h	frequency	.15	.1	.15	.12	.08	.25	.05	.1
letter	a	b	c	d	e	f	g	h											
frequency	.15	.1	.15	.12	.08	.25	.05	.1											

3.5.7 Use the Huffman algorithm to construct a prefix code so that the following line is encoded using the shortest possible bitstring. Remember to encode the blank.

*meandering with a mazy motion*

3.5.8 Explain why the output forest  $F$  in Algorithm 3.5.1 is actually a tree.

3.5.9 Prove Lemma 3.5.1.

3.5.10 [Computer Project] Implement the Huffman Code algorithm and test the computer program on Exercises 3.5.4 through 3.5.6.

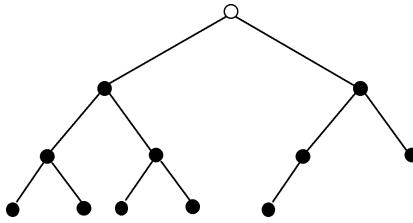
## 3.6 PRIORITY TREES

A *priority tree* is a special type of binary tree used in implementing the abstract datatype called a *priority queue*, an important structure for information processing.

**DEFINITION:** A binary tree of height  $h$  is called **left-complete** if the bottom level has no gaps as one traverses from left to right. More precisely, it must satisfy the following three conditions.

- Every vertex of depth  $h - 2$  or less has two children.
- There is at most one vertex  $v$  at depth  $h - 1$  that has only one child (a left one).
- No vertex at depth  $h - 1$  has fewer children than another vertex at depth  $h - 1$  to its right.

Figure 3.6.1 shows a left-complete binary tree of height 3.

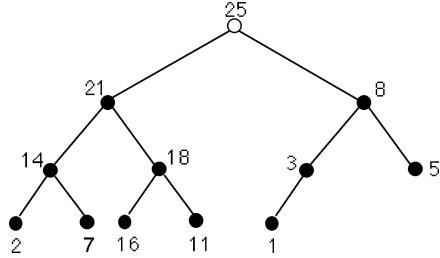


**Figure 3.6.1** A left-complete binary tree of height 3.

**DEFINITION:** A **priority tree** is a left-complete binary tree whose vertices have labels (called *priorities*) from an ordered set (or sometimes, a *partially ordered set*), such that no vertex has higher priority than its parent.

Although no vertex can have higher priority than any of its ancestors, a vertex can have higher priority than a sibling of an ancestor.

**Example 3.6.1:** In the priority tree shown in Figure 3.6.2, all of the vertices in the right subtree of the vertex labeled 21 have higher priority than the sibling of that vertex.



**Figure 3.6.2** A priority tree of height 3.

**DEFINITION:** A **priority queue** is a set of entries, each of which is assigned a *priority*. When an entry is to be removed, or *dequeued*, from the queue, an entry with the highest priority is selected.

The stack and queue structures are extreme cases of the priority queue, and are opposites to each other, because the newest entry to a stack gets the highest priority (*Last-In-First-Out*), whereas the newest entry to a queue gets the lowest priority (*First-In-First-Out*).

The most obvious implementation of a priority queue is as a linked list, sorted by descending priority. Each insertion must be “bubbled” to its correct position, and, hence, its worst-case behavior is  $O(n)$ , where  $n$  is the number of items. The deletion operation is  $O(1)$ , since deletion from a priority queue occurs at the beginning of the sorted list, where the highest priority item is stored.

A less obvious and generally more efficient implementation of the priority queue uses a priority tree.

### Inserting an Entry Into a Priority Tree

The enqueue operation for a priority queue is implemented by the following algorithm, which inserts an entry  $x$  into a priority tree  $T$  representing that priority queue.

The idea is to start by appending  $x$  at the first vacant spot in  $T$ , and then to bubble  $x$  upward toward the root until the priority of  $x$  is less than or equal to the priority of its parent.

**Algorithm 3.6.1: PriorityTreeInsert**

*Input:* a priority tree  $T$  and a new entry  $x$ .

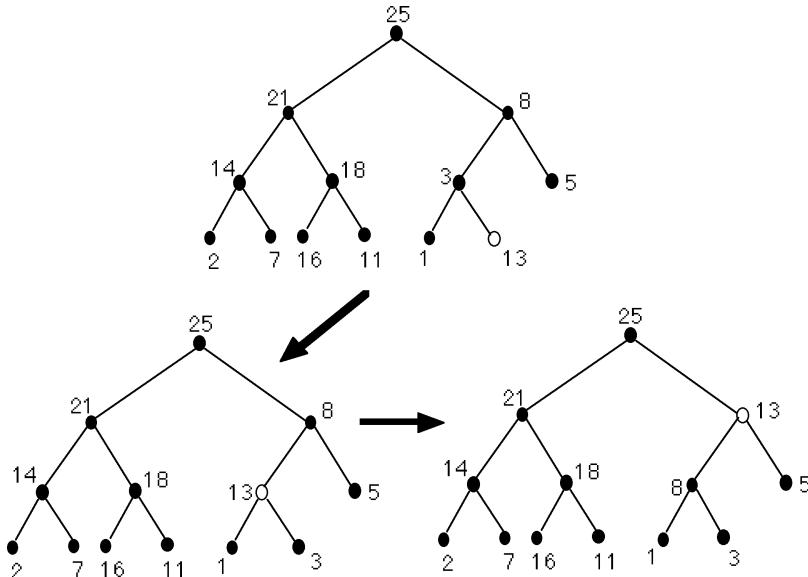
*Output:* tree  $T$  with  $x$  inserted so that  $T$  is still a priority tree.

```

Append entry  $x$  to the first vacant spot  $v$  in the left-complete tree  $T$ .
While  $v \neq \text{root}(T)$  AND  $\text{priority}(v) > \text{priority}(\text{parent}(v))$ 
    Swap  $v$  with  $\text{parent}(v)$ .

```

**Example 3.6.2:** Figure 3.6.3 shows the sequence of exchanges that result when an entry with priority 13 is inserted into the priority tree from Figure 3.6.2.



**Figure 3.6.3** Inserting the entry with priority 13 into the priority tree.

**COMPUTATIONAL NOTE:** The worst case occurs if entry  $x$  must be bubbled all the way up to the root. The number of exchanges this would require is the height of priority tree  $T$ , and thus, the worst-case running time of PriorityTreeInsert is  $O(\log_2 n)$ .

### Deleting an Entry From a Priority Tree

The dequeue operation on a priority queue is implemented by the following algorithm, which deletes an entry  $x$  from a priority tree  $T$ . First, entry  $x$  is replaced by the entry  $y$  that occupies the rightmost spot at the bottom level of the priority tree. Then  $y$  is trickled iteratively downward, each time swapped with the larger of its two children, until it exceeds the values of both its children.

**Algorithm 3.6.2: PriorityTreeDelete**

*Input:* a priority tree  $T$  and an entry  $x$  in  $T$ .

*Output:* tree  $T$  with  $x$  deleted so that it remains a priority tree.

Replace  $x$  by the entry  $y$  that occupies the rightmost spot at the bottom level of  $T$ .

While  $y$  is not a leaf AND [ $\text{priority}(y) \leq \text{priority}(\text{leftchild}(y))$

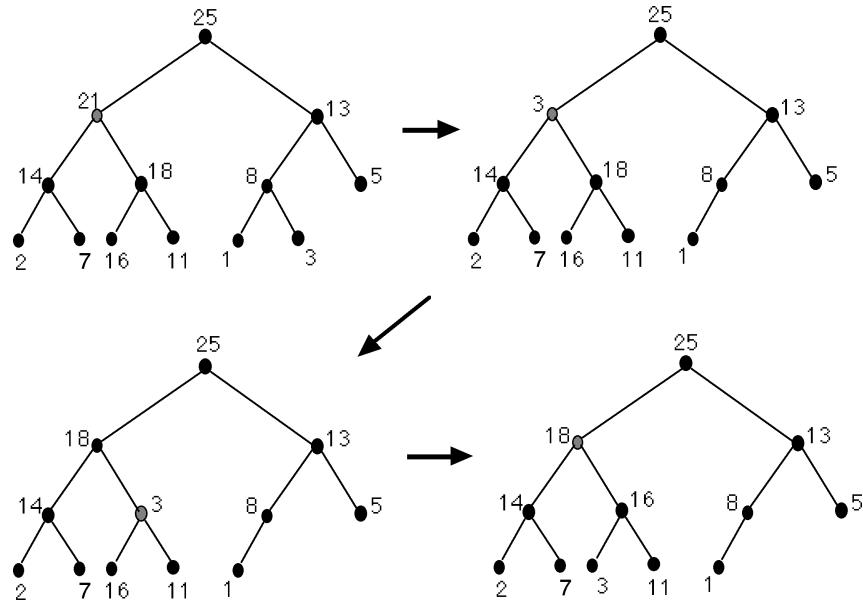
    OR  $\text{priority}(y) \leq \text{priority}(\text{rightchild}(y))$ ]

    If  $\text{priority}(\text{leftchild}(y)) > \text{priority}(\text{rightchild}(y))$

        Swap  $y$  with  $\text{leftchild}(y)$ .

    Else Swap  $y$  with  $\text{rightchild}(y)$ .

**Example 3.6.3:** Figure 3.6.4 illustrates this process when the entry with priority 21 is deleted from the given priority tree.



**Figure 3.6.4** Deleting the entry 21 from the priority tree.

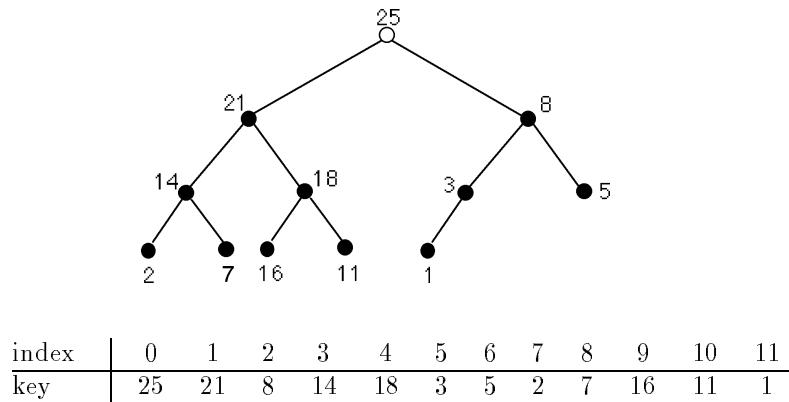
## Heaps

**DEFINITION:** A **heap** is a representation of a priority tree as an array, having the following address pattern.

- $\text{index}(\text{root}) = 0$
- $\text{index}(\text{leftchild}(v)) = 2 \times \text{index}(v) + 1$
- $\text{index}(\text{rightchild}(v)) = 2 \times \text{index}(v) + 2$
- $\text{index}(\text{parent}(v)) = \lfloor \frac{\text{index}(v)-1}{2} \rfloor$

A heap saves space because the address pattern eliminates the need to store pointers to children and to parents.

**Example 3.6.4:** Figure 3.6.5 shows a priority tree and the heap that represents it.



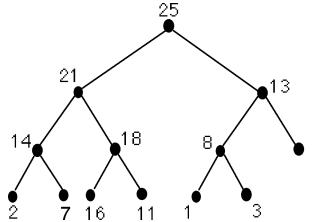
**Figure 3.6.5** Representing a priority tree using a heap.

### EXERCISES for Section 3.6

3.6.1 Draw all possible left-complete binary trees of height 2.

3.6.2<sup>s</sup> How many different left-complete binary trees of height  $h$  are there?

*In Exercises 3.6.3 through 3.6.8, perform the specified sequence of insertions and deletions on the following priority tree. After each individual insertion or deletion, draw the resulting priority tree.*



3.6.3 Insert 10, insert 15, delete 21.

3.6.4 Insert 10, delete 21, insert 15.

3.6.5 Delete 14, delete 3, insert 14.

3.6.6<sup>s</sup> Delete 18, insert 35, delete 25.

3.6.7 Insert 17, delete 8, insert 15.

3.6.8 Delete 8, insert 17, insert 22.

*In Exercises 3.6.9 through 3.6.12, either draw the priority tree represented by the heap whose entries are as shown, or explain why the given sequence does not represent a priority tree.*

3.6.9<sup>s</sup> 80, 60, 40, 50, 25, 10, 5, 40, 25, 10, 15.

3.6.10 45, 30, 32, 20, 28, 25, 16, 15, 18.

3.6.11<sup>s</sup> 45, 32, 37, 20, 25, 16, 29, 10, 22.

3.6.12 50, 20, 42, 15, 16, 25, 37, 11, 5, 8, 3.

3.6.13 Prove or disprove: The level-order of a priority tree is the same as the order of the heap that represents it.

### 3.7 COUNTING LABELED TREES: PRÜFER ENCODING

In 1875, Arthur Cayley presented a paper to the British Association describing a method for counting certain hydrocarbons containing a given number of carbon atoms. In the same paper, Cayley also counted the number of  $n$ -vertex trees with the standard vertex labels  $1, 2, \dots, n$ . Two labeled trees are considered the same if their respective edge-sets are identical. For example, in Figure 3.7.1, the two labeled 4-vertex trees are different, even though their underlying unlabeled trees are both isomorphic to the path graph  $P_4$ .



**Figure 3.7.1** Two different labeled trees.

The number of  $n$ -vertex labeled trees is  $n^{n-2}$ , for  $n \geq 2$ , and is known as **Cayley's Formula**. A number of different proofs have been given for this result, and the one presented here, due to H. Prüfer, is considered among the most elegant. The strategy of the proof is to establish a one-to-one correspondence between the set of standard-labeled trees with  $n$  vertices and certain finite sequences of numbers.

**Remark:** Counting the number of isomorphically distinct labeled  $n$ -vertex trees is much more difficult. The Pólya-Burnside enumeration method, which is presented in Chapter 14, can be used to solve this kind of problem.

#### Prüfer Encoding

**DEFINITION:** A **Prüfer sequence** of length  $n - 2$ , for  $n \geq 2$ , is any sequence of integers between 1 and  $n$ , with repetitions allowed.

The following encoding procedure constructs a Prüfer sequence from a given standard labeled tree, and thus, defines a function  $f_e : \mathcal{T}_n \rightarrow \mathcal{P}_{n-2}$  from the set  $\mathcal{T}_n$  of trees on  $n$  labeled vertices to the set  $\mathcal{P}_{n-2}$  of Prüfer sequences of length  $n - 2$ .

#### Algorithm 3.7.1: Prüfer Encoding

*Input:* an  $n$ -vertex tree with a standard 1-based vertex-labeling.

*Output:* a Prüfer sequence of length  $n - 2$ .

Initialize  $T$  to be the given tree.

For  $i = 1$  to  $n - 2$

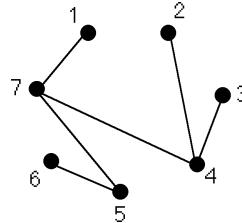
    Let  $v$  be the 1-valent vertex with the smallest label.

    Let  $s_i$  be the label of the only neighbor of  $v$ .

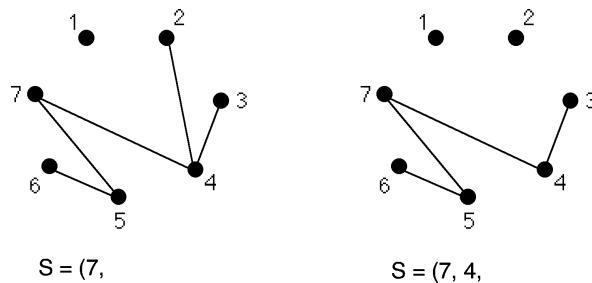
$T := T - v$ .

Return sequence  $\langle s_1, s_2, \dots, s_{n-2} \rangle$ .

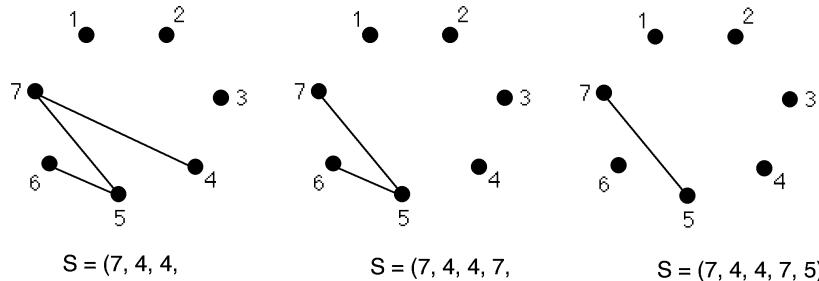
**Example 3.7.1:** The encoding procedure for the tree shown in Figure 3.7.2 is illustrated with the two figures that follow. The first figure shows the first two iterations of the construction, and the second figure shows the last three iterations. The portion of the Prüfer sequence constructed after each iteration is also shown.



**Figure 3.7.2** A labeled tree to be encoded into a Prüfer sequence  $S$ .



**Figure 3.7.3** First two iterations of the Prüfer encoding.



**Figure 3.7.4** Last three iterations of the Prüfer encoding.

Notice for the above example that the degree of each vertex is one more than the number of times that its label appears in the Prüfer sequence. The next result shows this is true in general. In the proof,  $l(u)$  denotes the label on vertex  $u$ .

**Proposition 3.7.1.** Let  $d_k$  be the number of occurrences of the number  $k$  in a Prüfer encoding sequence for a labeled tree  $T$ . Then the degree of vertex  $k$  in  $T$  equals  $d_k + 1$ .

**Proof:** The assertion is true for any tree on 3 vertices, because the Prüfer sequence for such a tree consists of a single occurrence of the label of the vertex of degree 2.

Assume that the assertion is true for all labeled trees on  $n$  vertices for some  $n \geq 3$ , and let  $T$  be a standard labeled tree on  $n + 1$  vertices. Let  $v$  be the 1-valent vertex with

the smallest label, and suppose that vertex  $w$  is the neighbor of  $v$ . Then the Prüfer sequence  $S$  for  $T$  consists of the label  $l(w)$  followed by the Prüfer sequence  $S^*$  of the tree  $T^* = T - v$ . By the induction hypothesis,  $\deg_{T^*}(u)$  equals the number of occurrences of  $l(u)$  in  $S^*$ . But for all  $u \neq w$ , the number of occurrences of the label  $l(u)$  in  $S^*$  is the same as in  $S$ , and  $\deg_T(u) = \deg_{T^*}(u)$ . Furthermore,  $\deg_T(w) = \deg_{T^*}(w) + 1$ , and  $l(w)$  has one more occurrence in  $S$  than in  $S^*$ . Thus, the condition is true for every vertex in  $T$ .  $\diamond$

## Prüfer Encoding

The following decoding procedure maps a given Prüfer sequence to a standard labeled tree.

### Algorithm 3.7.2: Prüfer Decoding

*Input:* a Prüfer sequence of length  $n - 2$ .

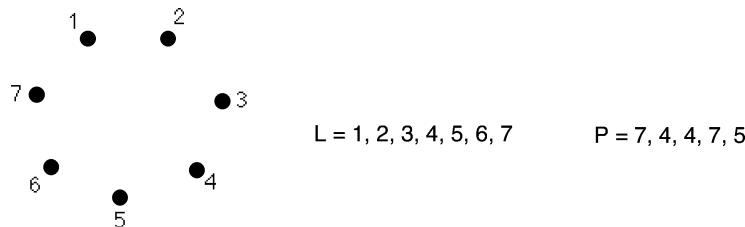
*Output:* an  $n$ -vertex tree with a standard 1-based vertex-labeling.

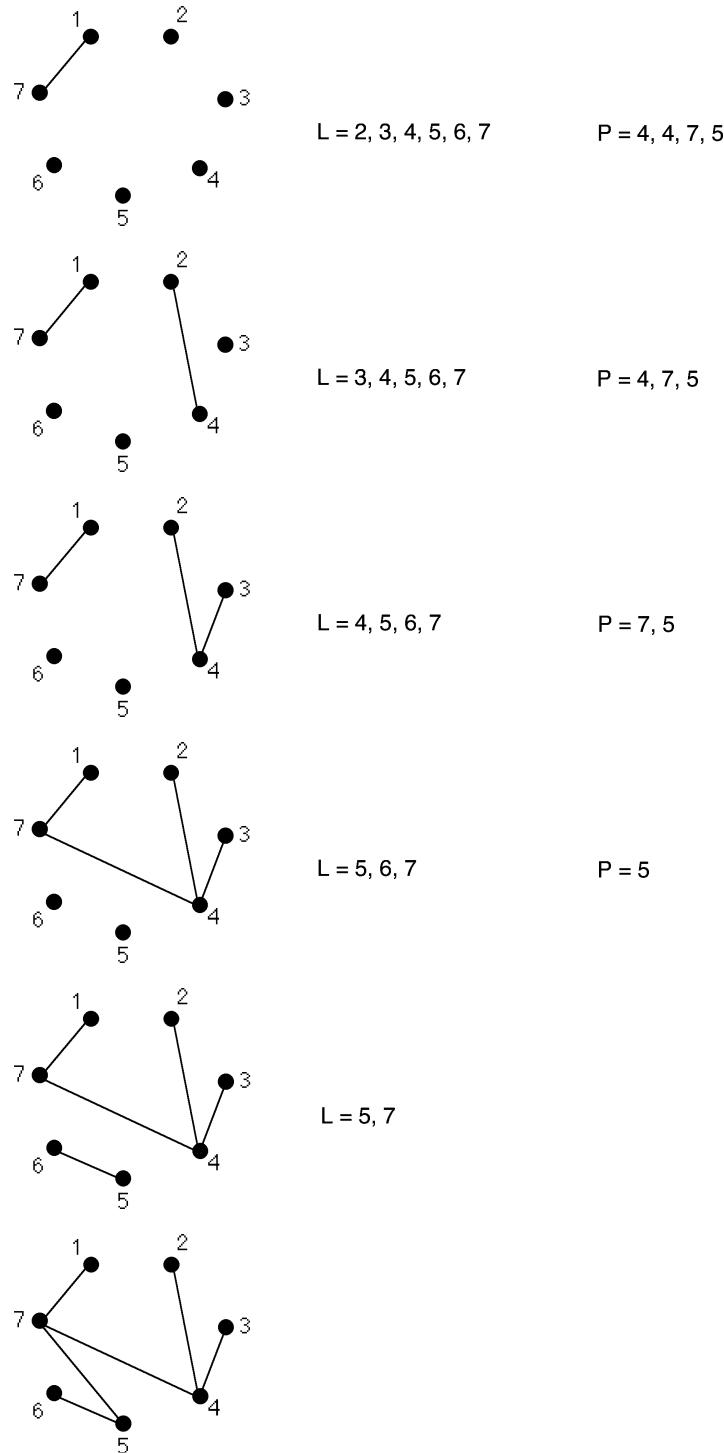
```

Initialize list  $P$  as the Prüfer input sequence.
Initialize list  $L$  as  $1, \dots, n$ .
Initialize forest  $F$  as  $n$  isolated vertices, labeled 1 to  $n$ .
For  $i = 1$  to  $n - 2$ 
    Let  $k$  be the smallest number in list  $L$  that is not in list  $P$ .
    Let  $j$  be the first number in list  $P$ .
    Add an edge joining the vertices labeled  $k$  and  $j$ .
    Remove  $k$  from list  $L$ .
    Remove the first occurrence of  $j$  from list  $P$ .
Add an edge joining the vertices labeled with the two remaining numbers
in list  $L$ .
Return  $F$  with its vertex-labeling.

```

**Example 3.7.2:** To illustrate the decoding procedure, start with the input sequence 7, 4, 4, 7, 5. Proposition 3.7.1 implies that the corresponding tree has:  $\deg(1) = \deg(2) = \deg(3) = \deg(6) = 1$ ;  $\deg(4) = \deg(7) = 3$ ; and  $\deg(5) = 2$ . Among the 1-valent vertices, vertex 1 has the smallest label, and its neighbor must be vertex 7. Thus, an edge is drawn joining vertices 1 and 7. The number 1 is removed from the list, and the first occurrence of label 7 is removed from the sequence. The sequence of figures that follows shows each iteration of the decoding procedure. Shown in each figure, from left to right, are: the construction up to that point, the numbers remaining in the list, and the remaining part of the Prüfer sequence.





**Proposition 3.7.2.** *The decoding procedure defines a function  $f_d : \mathcal{P}_{n-2} \rightarrow \mathcal{T}_n$  from the set of Prüfer sequences of length  $n - 2$  to the set of labeled trees on  $n$  vertices.*

**Proof:** First observe that at each step of the procedure, there is never any choice as to which edge must be drawn. Thus, the procedure defines a function from the Prüfer

sequences to the set of graphs on  $n$  vertices. Therefore, proving the following assertion will complete the proof.

*Assertion:* When the decoding procedure is applied to a Prüfer sequence of length  $n - 2$ , the graph produced is a tree.

The assertion is trivially true for  $n = 2$ , since the procedure produces a single edge. Assume that the assertion is true for some  $n \geq 2$ , and consider a Prüfer sequence  $\langle p_1, p_2, \dots, p_{n-1} \rangle$  and a set of vertices  $\{1, 2, \dots, n+1\}$ . The first iteration of the procedure draws an edge from  $b$  to  $p_1$ , where  $b$  is the smallest vertex not appearing among the  $p_i$ 's. None of the  $n - 1$  edges that are produced in iterations 2 through  $n$  will be incident with  $b$ . Thus, continuing the procedure from iteration 2 is equivalent to applying the procedure to the Prüfer sequence  $\langle p_2, \dots, p_{n-1} \rangle$  for the set of vertices  $\{1, \dots, b-1, b+1, \dots, n+1\}$ . By the induction hypothesis, the edges produced form a tree on these vertices. This tree, together with the edge from  $b$  to  $p_1$ , forms a tree on the vertices  $\{1, 2, \dots, n+1\}$ .  $\diamond$

Notice that the tree obtained in Example 3.7.2 by the Prüfer decoding of the sequence  $\langle 7, 4, 4, 7, 5 \rangle$  is the same as the tree in Example 3.7.1 that was Prüfer-encoded as  $\langle 7, 4, 4, 7, 5 \rangle$ . This inverse relationship between the encoding and decoding functions holds in general, as the following proposition asserts.

**Proposition 3.7.3.** The decoding function  $f_d : \mathcal{P}_{n-2} \rightarrow \mathcal{T}_n$  is the inverse of the encoding function  $f_e : \mathcal{T}_n \rightarrow \mathcal{P}_{n-2}$ .  $\diamond$  (Exercises)

**Theorem 3.7.4 [Cayley's Tree Formula].** The number of different trees on  $n$  labeled vertices is  $n^{n-2}$ .

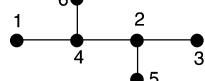
**Proof:** Proposition 3.7.3 establishes a one-to-one correspondence between the trees in  $\mathcal{T}_n$  and the sequences in  $\mathcal{P}_{n-2}$ , and, by the Rule of Product, there are  $n^{n-2}$  such sequences.  $\diamond$

**Remark:** A slightly different view of Cayley's Tree Formula is that it gives us the number of different spanning trees of the complete graph  $K_n$ . The next chapter is devoted to spanning trees.

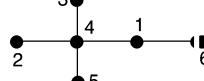
### EXERCISES for Section 3.7

In Exercises 3.7.1 through 3.7.6, encode the given labeled tree as a Prüfer sequence. Then decode the resulting sequence, to demonstrate that Proposition 3.7.3 holds.

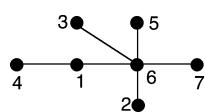
3.7.1<sup>s</sup>



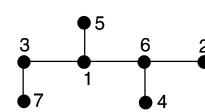
3.7.2



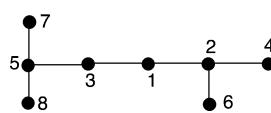
3.7.3



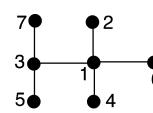
3.7.4



3.7.5



3.7.6<sup>s</sup>



In Exercises 3.7.7 through 3.7.12, construct the labeled tree corresponding to the given Prüfer sequence.

3.7.7  $\langle 6, 7, 4, 4, 4, 2 \rangle$ .

3.7.8  $\langle 2, 1, 1, 3, 5, 5 \rangle$ .

3.7.9  $\langle 1, 3, 7, 2, 1 \rangle$ .

3.7.10<sup>s</sup>  $\langle 1, 3, 2, 3, 5 \rangle$ .

3.7.11  $\langle 1, 1, 5, 1, 5 \rangle$ .

3.7.12  $\langle 1, 1, 5, 2, 5 \rangle$ .

3.7.13 Draw the  $4^{4-2}$  labeled trees on four vertices.

3.7.14<sup>s</sup> How many spanning trees are there of the labeled complete bipartite graph  $K_{m,2}$ ?

3.7.15 Prove Proposition 3.7.3, by showing that the Prüfer encoding of an arbitrary  $n$ -vertex labeled tree, followed by the Prüfer decoding of the resulting Prüfer sequence, recaptures the original tree.

## 3.8 COUNTING BINARY TREES: CATALAN RECURSION

In this section, we derive a formula for the number of different binary trees on  $n$  vertices. As is frequently the case in *enumerative combinatorics*, the derivation begins by establishing a recursive formula.

Let  $b_n$  denote the number of binary trees on  $n$  vertices. The recursion can be expressed more conveniently by artificially defining  $b_0 = 1$ . Since the only binary tree on one vertex is a root with no children, it follows that  $b_1 = 1$ . For  $n > 1$ , a binary tree  $T$  on  $n$  vertices has a left subtree with  $j$  vertices and a right subtree with  $n - 1 - j$  vertices, for some  $j$  between 0 and  $n - 1$ . Pairing the  $b_j$  possible left subtrees with the  $b_{n-1-j}$  possible right subtrees results in  $b_j \times b_{n-1-j}$  different combinations of left and right subtrees for  $T$ . Hence,

$$b_n = b_0 b_{n-1} + b_1 b_{n-2} + \dots + b_{n-1} b_0$$

This *recurrence relation* is known as the **Catalan recursion**, and the quantity  $b_n$  is called the  **$n$ th Catalan number**. The Catalan numbers occurs in many different applications besides tree-counting (see Exercises).

**Example 3.8.1:** Applying the Catalan recursion to the cases  $n = 2$  and  $n = 3$  yields  $b_2 = 2$  and  $b_3 = 5$ . Figure 3.8.1 shows the five different binary trees on 3 vertices.

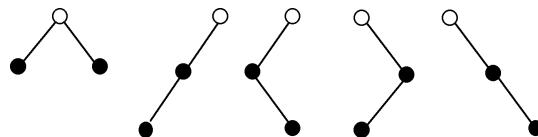


Figure 3.8.1 The five different binary trees on 3 vertices.

With the aid of generating functions, it is possible to derive the following *closed formula* for  $b_n$ .

**Theorem 3.8.1.** *The number  $b_n$  of different binary trees on  $n$  vertices is given by*

$$b_n = \frac{1}{n+1} \binom{2n}{n}$$

#### Outline of Proof:

- (a) Let  $g(x) = \sum_{k=0}^{\infty} b_k x^k$  be the generating function for the number  $b_n$  of different binary trees on  $n$  vertices. Then  $1 + x[g(x)]^2 = g(x)$ .
- (b) Applying the quadratic formula to step (a) implies that  $g(x) = \frac{1-\sqrt{1-4x}}{2x}$ .
- (c) The *Generalized Binomial Theorem* states that  $(1+z)^r = \sum_{k=0}^{\infty} \binom{r}{k} z^k$ , for any real number  $r$ , where the generalized binomial coefficient  $\binom{r}{k}$  is defined by

$$\binom{r}{k} = \begin{cases} 1, & \text{if } k = 0; \\ \frac{r(r-1)(r-2)\cdots(r-k+1)}{k!}, & \text{if } k \geq 1. \end{cases}$$

Using the Generalized Binomial Theorem and the result from part (b) we obtain

$$g(x) = \frac{1}{2x} \left( 1 - \sum_{k=0}^{\infty} \binom{1/2}{k} (-4x)^k \right).$$

- (d) A change of variable for the index of summation in part (c) yields

$$g(x) = \sum_{n=0}^{\infty} \binom{1/2}{n+1} (-1)^n 2^{2n+1} x^n,$$

from which we conclude that  $b_n = \binom{1/2}{n+1} (-1)^n 2^{2n+1}$ .

- (e) The definition of  $\binom{1/2}{n+1}$  implies that

$$\binom{1/2}{n+1} = \frac{(-1)^n}{2^{n+1}} \cdot \frac{(1)(3)(5)\cdots(2n-1)}{(1)(2)(3)\cdots(n+1)},$$

and, hence,

$$b_n = \frac{1}{n+1} \binom{2n}{n}$$

◇

#### EXERCISES for Section 3.8

**3.8.1<sup>s</sup>** Use the Catalan recursion to determine the number of different binary trees on four vertices. Compare your answer with that of the closed formula.

**3.8.2** Use the Catalan recursion to determine the number of different binary trees on five vertices. Compare your answer with that of the closed formula.

**3.8.3** Illustrate the Catalan recursion for  $n = 4$  by drawing the different binary trees on four vertices and grouping them according to the size of their left and right subtrees.

**3.8.4<sup>s</sup>** Let  $p_n$  be the number of ways to place parentheses to multiply the  $n$  numbers  $a_1 \times a_2 \times \cdots \times a_n$  on a calculator. For instance,  $p_3 = 2$ , because there are two ways to multiply  $a_1 \times a_2 \times a_3$ , namely,  $(a_1 \times a_2) \times a_3$  and  $a_1 \times (a_2 \times a_3)$ . Show that  $p_n$  equals the  $(n-1)$ st Catalan number  $b_{n-1}$ .

- 3.8.5 Fill in the details of steps (a) and (b) of the proof outline for Theorem 3.8.1.
- 3.8.6 Fill in the details of steps (c) and (d) of the proof outline for Theorem 3.8.1.
- 3.8.7<sup>s</sup> Fill in the details of step (e) of the proof outline for Theorem 3.8.1.
- 3.8.8 [Computer Project] Write a computer program that uses a recursive function  $Catalan(n)$  whose value is the  $n$ th Catalan number, and write a second program that uses an iterative version of  $Catalan(n)$ , based on the closed formula established in Theorem 3.8.1. Then use both programs to produce the first 15 Catalan numbers, and if your computing environment allows it, compare the execution time of both programs.

### 3.9 SUPPLEMENTARY EXERCISES

- 3.9.1 Draw every tree  $T$  such that the edge-complement  $\overline{T}$  is a tree.
- 3.9.2 What are the minimum and maximum independence numbers of an  $n$ -vertex tree?
- 3.9.3 What are the minimum and maximum number of vertex orbits in an  $n$ -vertex tree?
- 3.9.4 What are the minimum and maximum number of rooted  $n$ -vertex trees that have the same underlying tree?
- 3.9.5 Reconstruct the tree whose deck appears in Figure 3.9.1.

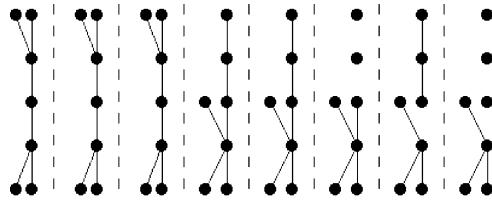


Figure 3.9.1

- 3.9.6 Characterize the class of connected graphs  $G$  such that for every edge  $e \in E_G$ , the graph  $G - e$  is a tree.
- 3.9.7 List all tree-graphic sequences for a 7-vertex tree.
- 3.9.8 Draw all 11 isomorphism types of trees with 7 vertices.
- 3.9.9 List all the possible degree sequences for an 8-vertex tree with maximum degree 3.
- 3.9.10 For each of the possible degree sequences of Exercise 3.9.9, draw all the different isomorphism types of tree with that degree sequence.
- 3.9.11 List all tree-graphic sequences for an 8-vertex tree.
- 3.9.12 Draw all five isomorphism types of trees with 9 vertices, such that no vertex has degree two.
- 3.9.13 Draw all the isomorphism types (without repetition) of trees with degree sequence 33221111.

**DEF:** A **rigid tree** is a tree whose only automorphism is the identity automorphism.

- 3.9.14 Prove that for all  $n \geq 7$ , there exists an  $n$ -vertex rigid tree.  
 3.9.15 Draw two non-isomorphic rigid 9-vertex trees.  
 3.9.16 Prove that there is no 5-vertex rigid tree.  
 3.9.17 Prove that there is no 6-vertex rigid tree.  
 3.9.18 Write the post-order for the following plane tree? Hint: Be carefl. :-)

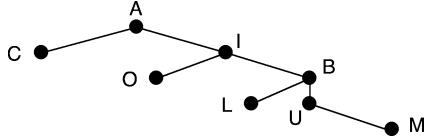


Figure 3.9.2

- 3.9.19 Draw the BST that results from deleting node 41 from the BST in Figure 3.9.3.  
 3.9.20 Draw the BST that results from inserting 44 into the BST in Figure 3.9.3.

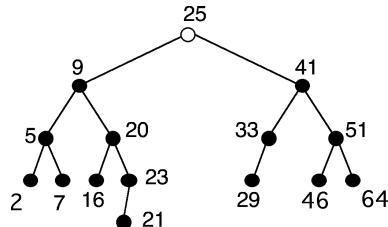


Figure 3.9.3

- 3.9.21 Draw the result of dequeuing highest priority node 25 from the heap in Figure 3.9.4 below. Then draw the result of reinserting 25.

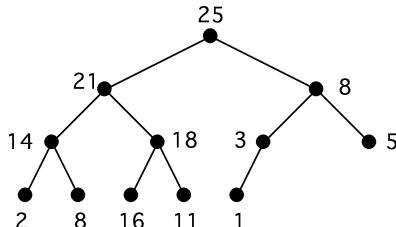


Figure 3.9.4

- 3.9.22 Write the Prüfer code for the following labeled tree.

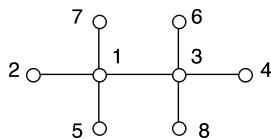


Figure 3.9.5

In Exercises 3.9.23 through 3.9.28, draw the tree with the given Prüfer code.

- |        |        |        |        |        |          |
|--------|--------|--------|--------|--------|----------|
| 3.9.23 | 112050 | 3.9.24 | 316313 | 3.9.25 | 512256   |
| 3.9.26 | 61661  | 3.9.27 | 713328 | 3.9.28 | 01201201 |

## GLOSSARY

**ancestor** of a vertex  $w$  in a rooted tree: a vertex on the path from the root to vertex  $w$ ; vertex  $v$  is an ancestor of vertex  $w$  if and only if  $w$  is a descendant of  $v$ .

**array-of-parents** representation of a rooted tree whose vertices are labeled  $0, 1, \dots, n - 1$ : a list whose  $k$ th entry is the parent of vertex  $k$ .

**average weighted depth** of a binary tree  $T$  with leaves  $s_1, s_2, \dots, s_l$  assigned weights  $w_1, w_2, \dots, w_l$ : the quantity, denoted  $wt(T)$ , given by  $wt(T) = \sum_{i=1}^l \text{depth}(s_i) \cdot w_i$ .

**balanced** binary tree: a binary tree such that for every vertex, the number of vertices in its left and right subtrees differ by at most one.

**binary code**: an assignment of symbols or other meanings to a set of bitstrings.

**binary-search tree (BST)**: a binary tree, each of whose vertices is assigned a key, such that the key assigned to any vertex  $v$  is greater than the key at each vertex in the left subtree of  $v$ , and is less than the key at each vertex in the right subtree of  $v$ .

**binary tree**: a rooted tree in which each vertex has no more than two children, each designated as either a *left-child* or *right-child*; an ordered 2-ary tree.

**Catalan number,  $n$ th**: the quantity  $b_n$  in the Catalan recursion.

**Catalan recursion**: the recurrence relation for the number  $b_n$  of different binary trees on  $n$  vertices, given by  $b_n = b_0 b_{n-1} + b_1 b_{n-2} + \dots + b_{n-1} b_0$ .

**center of a graph**  $G$ , denoted  $Z(G)$ : the subgraph induced on the set of central vertices of  $G$ .

**central vertex** of a graph: a vertex of minimum eccentricity.

**child** of a vertex  $v$  in a rooted tree: a vertex  $w$  that immediately succeeds vertex  $v$  on the path from the root to  $w$ ;  $w$  is the child of  $v$  if and only if  $v$  is the parent of  $w$ .

**codeword**: one of the bitstrings in a binary code.

**complete  $n$ -ary tree**: an  $n$ -ary tree in which every internal vertex has exactly  $n$  children, and all leaves have the same depth.

**cycle-edge**: an edge that lies on a cycle.

**depth** of a vertex  $v$  in a rooted tree: the length of the unique path from the root to  $v$ .

**descendant** of a vertex  $v$  in a rooted tree: a vertex  $w$  whose path from the root contains vertex  $v$ ;

**directed tree**: a digraph whose underlying graph is a tree.

**distance sum** of a graph  $G$ , denoted  $ds(G)$ : the sum of the distances between all pairs of vertices in  $G$ ; that is,  $ds(G) = \sum_{u, v \in V_G} d(u, v)$ .

**eccentricity** of a vertex  $v$  in a graph  $G$ , denoted  $\text{ecc}(v)$ : the distance from  $v$  to a vertex farthest from  $v$ ; that is,  $\text{ecc}(v) = \max_{x \in V_G} \{d(v, x)\}$ .

**expression tree** for an arithmetic expression: either a single vertex labeled by an identifier, or a binary tree described by the following recursive rule:

- The operator to be executed during an evaluation is at the root.
- The left subtree of the root is an expression tree representing the left operand.
- The right subtree is an expression tree representing the right operand.

**forest**: an acyclic graph.

**graceful tree:** a tree  $T$  with  $m + 1$  vertices and  $m$  edges such that it is possible to label the vertices of  $T$  with distinct elements from the set  $\{0, 1, \dots, m\}$  in such a way that the induced edge-labeling, which assigns the integer  $|i - j|$  to the edge joining the vertices labeled  $i$  and  $j$ , assigns the labels  $1, 2, \dots, m$  to the  $m$  edges of  $G$ .

**heap:** a representation of a priority tree as an array, having the following address pattern:

- $\text{index}(\text{root}) = 0$
- $\text{index}(\text{leftchild}(v)) = 2 \times \text{index}(v) + 1$
- $\text{index}(\text{rightchild}(v)) = 2 \times \text{index}(v) + 2$
- $\text{index}(\text{parent}(v)) = \lfloor \frac{\text{index}(v)-1}{2} \rfloor$

**height** of a rooted tree: the length of a longest path from the root; the greatest depth in the tree.

**Huffman code:** the prefix code corresponding to a Huffman tree.

**Huffman tree:** the binary tree that results from an application of the Huffman algorithm (Algorithm 3.5.1).

**internal vertex** in a rooted tree: a vertex that is not a leaf.

**isomorphic rooted trees:** two rooted trees that have a graph isomorphism between them that maps root to root.

**leaf** in a rooted tree: a vertex having no children.

**leaf** in an undirected tree: a vertex of degree 1.

**left-child:** one of two possible designations for a child of a vertex in a binary tree.

**left-complete** binary tree: a binary tree such that the bottom level has no gaps as one traverses from left to right.

**level** of a vertex in a rooted tree: synonym for *depth*.

**level-order** of an ordered tree: a listing of the vertices in the top-to-bottom, left-to-right order of a standard plane drawing of that tree.

**$m$ -ary tree:** a rooted tree in which every vertex has  $m$  or fewer children.

**ordered tree:** a rooted tree in which the children of each vertex are assigned a fixed ordering.

**parent** of a vertex  $w$  in a rooted tree: the vertex that immediately precedes vertex  $w$  on the path from the root to  $w$ .

**prefix code:** a binary code with the property that no codeword is an initial substring of any other codeword.

**priority queue:** a set of entries, each of which is assigned a *priority*. When an entry is to be removed, an entry with the highest priority is selected.

**priority tree:** a left-complete binary tree whose vertices have labels (called *priorities*) from an ordered set, such that no vertex has higher priority than its parent.

**proper descendant** of a vertex  $v$ : a descendant of  $v$  that is different from  $v$ .

**Prüfer sequence** of length  $n - 2$ , for  $n \geq 2$ : any sequence of integers between 1 and  $n$ , with repetitions allowed; used to encode trees (see Algorithm 3.7.1).

**queue:** a sequence of elements such that each new element is added (*enqueued*) to one end, called the *back* of the queue, and an element is removed (*dequeued*) from the other end, called the *front*.

**random-access table:** a database whose domain is a sequence of entries, where each entry consists of two fields. One field is for the actual data element, and the other one is for the key, whose value determines that entry's position in the database.

**right-child:** one of two possible designations for a child of a vertex in a binary tree.

**rigid tree:** a tree whose only automorphism is the identity automorphism.

**root:** the distinguished vertex of a rooted tree.

**rooted tree:** a directed tree having a distinguished vertex  $r$  called the *root* such that for every other vertex  $v$ , there is a directed path from  $r$  to  $v$ .

**shortest-path tree** for a connected graph  $G$  from a vertex  $v$ : a rooted tree  $T$  with vertex-set  $V_G$  and root  $v$  such that the unique path in  $T$  from  $v$  to each vertex  $w$  is a shortest path in  $G$  from  $v$  to  $w$ .

**siblings** in a rooted tree: vertices that have the same parent.

**stack:** a sequence of elements such that each new element is added (*pushed*) onto one end, called the *top*, and an element is removed (*popped*) from that same end.

**standard plane drawing** of a rooted tree: a plane drawing of the rooted tree such that the root is at the top, and the vertices at each level are horizontally aligned.

**standard plane representation** of an ordered tree: a standard plane drawing of the ordered tree such that at each level, the left-to-right order of the vertices agrees with their prescribed order.

**ternary tree:** synonym for  $\beta$ -ary tree.

**tree-graphic sequence:** a sequence for which there is a permutation that is the degree sequence of some tree.

**tree traversal** of a binary tree: a systematic visit of each vertex of the tree.

—, **in-order:** defined recursively as follows:

- i. Perform an in-order traversal of the left subtree of  $T$ .
- ii. List (process) the root of  $T$ .
- iii. Perform an in-order traversal of the right subtree of  $T$ .

—, **left pre-order:** defined recursively as follows:

- i. List (process) the root of  $T$ .
- ii. Perform a left pre-order traversal of the left subtree of  $T$ .
- iii. Perform a left pre-order traversal of the right subtree of  $T$ .

—, **level-order:** a visit of the vertices in the top-to-bottom, left-to-right order of a standard plane drawing of that binary tree.

—, **post-order:** defined recursively as follows:

- i. Perform a post-order traversal of the left subtree of  $T$ .
- ii. Perform a post-order traversal of the right subtree of  $T$ .
- iii. List (process) the root of  $T$ .

**Wiener index:** synonym for *distance sum*.

# Chapter 4

---

## SPANNING TREES

- 4.1 Tree Growing**
  - 4.2 Depth-First and Breadth-First Search**
  - 4.3 Minimum Spanning Trees and Shortest Paths**
  - 4.4 Applications of Depth-First Search**
  - 4.5 Cycles, Edge-Cuts, and Spanning Trees**
  - 4.6 Graphs and Vector Spaces**
  - 4.7 Matroids and the Greedy Algorithm**
- 

### INTRODUCTION

Systematic processing of the vertices and edges of a graph often involves the strategy of growing a spanning tree. This chapter demonstrates that several classical algorithms, including depth-first and breadth-first searches, Prim's minimum-spanning-tree method, and Dijkstra's shortest-path method, are special cases or extensions of this strategy. Sketches of these different algorithms highlight their similarities and provide an overview from which a deeper study can be made. That there is potentially an exponentially large number of spanning trees (established by Cayley's Formula in §3.7) makes the low-order-polynomial-time algorithms that find a spanning tree of minimum weight all the more impressive.

Spanning trees also provide a foundation for a systematic analysis of the cycle structure of a graph, as described in the last three sections of this chapter. Mathematicians regard the algebraic structures underlying the collection of cycles and edge-cuts of a graph as beautiful in their own right. Their use in various applications here offers further evidence of the merger of mathematical elegance with techniques of practical importance.

## 4.1 TREE-GROWING

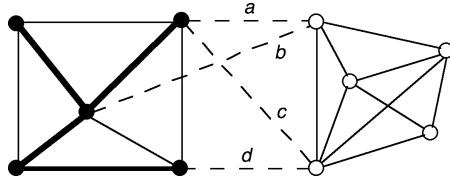
Several different problem-solving algorithms involve growing a spanning tree, one edge and one vertex at a time. All these techniques are refinements and extensions of the same basic tree-growing scheme given in this section.

**TERMINOLOGY:** For a given tree  $T$  in a graph  $G$ , the edges and vertices of  $T$  are called **tree edges** and **tree vertices**, and the edges and vertices of  $G$  that are not in  $T$  are called **non-tree edges** and **non-tree vertices**.

### Frontier Edges

**DEFINITION:** A **frontier edge** for a given tree  $T$  in a graph is a non-tree edge with one endpoint in  $T$ , called its **tree endpoint**, and one endpoint not in  $T$ , its **non-tree endpoint**.

**Example 4.1.1:** For the graph in Figure 4.1.1, the tree edges of a tree  $T$  are drawn in bold. The tree vertices are black, and the non-tree vertices are white. The frontier edges for  $T$ , appearing as dashed lines, are edges  $a$ ,  $b$ ,  $c$ , and  $d$ . The plain edges are the non-tree edges that are not frontier edges for  $T$ .



**Figure 4.1.1** A tree with frontier edges  $a$ ,  $b$ ,  $c$ , and  $d$ .

Observe that when any one of the frontier edges in Figure 4.1.1 is added to the tree  $T$ , the resulting subgraph is still a tree. This property holds in general, as we see in the next proposition, and its iterative application forms the core of the tree-growing scheme.

**Proposition 4.1.1.** *Let  $T$  be a tree in a graph  $G$ , and let  $e$  be a frontier edge for  $T$ . Then the subgraph of  $G$  formed by adding edge  $e$  to tree  $T$  is a tree.*

**Proof:** The addition of the frontier edge  $e$  to the tree  $T$  cannot create a cycle, since one of its endpoints is outside of  $T$ . Moreover the vertex that was added to the tree is clearly reachable from any other vertex in the resulting tree. ◇

**Remark:** Formally, adding a frontier edge to a tree involves adding a new vertex to the tree, as well as the primary operation of *adding an edge*, defined in §2.4.

### Choosing a Frontier Edge

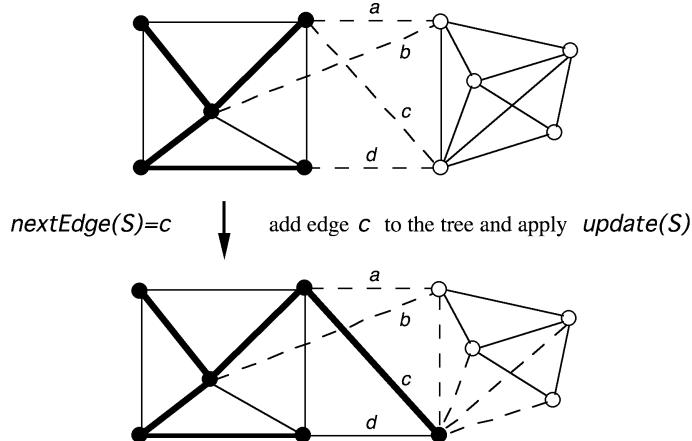
An essential component of our tree-growing scheme is a function called *nextEdge*, which chooses a frontier edge to add to the current tree.

**DEFINITION:** Let  $T$  be a tree subgraph of a graph  $G$ , and let  $S$  be the set of frontier edges for  $T$ . The function **nextEdge**( $G, S$ ) chooses and returns as its value the frontier edge in  $S$  that is to be added to tree  $T$ .

**Remark:** Ordinarily, the function  $nextEdge$  is deterministic; however, it may also be randomized. In either case, the full specification of  $nextEdge$  must ensure that it always returns a frontier edge.

**DEFINITION:** After a frontier edge is added to the current tree, the procedure **updateFrontier**( $G, S$ ) removes from  $S$  those edges that are no longer frontier edges and adds to  $S$  those that have become frontier edges.

**Example 4.1.1, continued:** The current set  $S$  of frontier edges shown in the top half of Figure 4.1.2 is  $S = \{a, b, c, d\}$ . Suppose that the value of  $nextEdge(G, S)$  is edge  $c$ . The bottom half of Figure 4.1.2 shows the result of adding edge  $c$  to tree  $T$  and applying  $updateFrontier(G, S)$ . Notice that  $updateFrontier(G, S)$  added four new frontier edges to  $S$  and removed edges  $c$  and  $d$ , which are no longer frontier edges.



**Figure 4.1.2** Result after adding edge  $c$  to the tree.

**Algorithm 4.1.1: Tree-Growing**

*Input:* a connected graph  $G$ , a starting vertex  $v \in V_G$ , and a function  $nextEdge$ .  
*Output:* an ordered spanning tree  $T$  of  $G$  with root  $v$ .

```

Initialize tree  $T$  as vertex  $v$ .
Initialize  $S$  as the set of proper edges incident on  $v$ .
While  $S \neq \emptyset$ 
    Let  $e = nextEdge(G, S)$ .
    Let  $w$  be the non-tree endpoint of edge  $e$ .
    Add edge  $e$  and vertex  $w$  to tree  $T$ .
     $updateFrontier(G, S)$ .
Return tree  $T$ .

```

**Remark:** Each different version of  $nextEdge$ , based on how its selection rule is defined, creates a different instance of Tree-Growing.<sup>†</sup> In §4.2 and §4.3, we will see that the four classical algorithms, *depth-first search*, *breadth-first search*, *Prim's spanning-tree*, and *Dijkstra's shortest-path*, are four different instances of Tree-Growing.

---

<sup>†</sup> That is,  $nextEdge$  is regarded as a function-valued variable whose instantiation turns Tree-Growing into a specific algorithm.

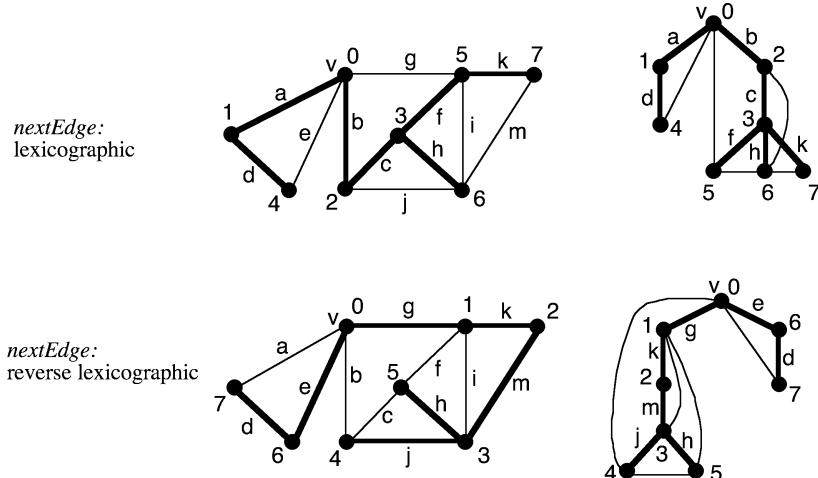
### Discovery Order of the Vertices

**DEFINITION:** Let  $T$  be the spanning tree produced by any instance of Tree-Growing in a graph  $G$ . The **discovery order** is a listing of the vertices of  $G$  in the order in which they are added (discovered) as tree  $T$  is grown. The position of a vertex in this list, starting with 0 for the start vertex, is called the **discovery number** of that vertex.

**Proposition 4.1.2.** *Let  $T$  be the output tree produced by any instance of Tree-Growing in a graph. Then  $T$  is an ordered tree with respect to the discovery order of its vertices.*  $\diamond$  (Exercises)

**Example 4.1.2:** Figure 4.1.3 shows the two output trees that result when two different instances of Tree-Growing are applied to the graph shown with starting vertex  $v$ . For the first instance, the function *nextEdge* selects the frontier edge whose name is lexicographically (alphabetically) first, and for the second instance, *nextEdge* uses reverse lexicographic order.

Each output tree is depicted in two ways: as a spanning tree whose edges are drawn in bold, and as an ordered tree with root  $v$ . The discovery number appears next to each vertex. Notice that the left-to-right order of the children of each vertex is consistent with the discovery order, as asserted by Proposition 4.1.2.



**Figure 4.1.3** Output produced by two instances of Tree-Growing.

**Example 4.1.3:** *Cycle Detection in a Connected Graph.* Observe that a connected graph  $G$  is acyclic if and only if the output tree produced by Tree-Growing for the input graph  $G$  contains all the edges of  $G$ . Moreover, if graph  $G$  does have cycles, then Tree-Growing can be used to find a special subcollection of them that forms a basis in a certain vector subspace associated with graph (see §4.5 and §4.6). In particular, as we saw in §3.1 and as illustrated in Figure 4.1.3 above, each non-tree edge creates a unique cycle when that edge alone is added to the tree. In §4.5 and §4.6, we see how these *fundamental cycles* generate *all* the cycles of the graph.

### Two Kinds of Non-Tree Edges

The non-tree edges that appear in both output trees in Figure 4.1.3 above, or any other output tree, fall into two categories.

**DEFINITION:** Two vertices in a rooted tree are **related** if one is a descendant of the other.

**DEFINITION:** For a given output tree grown by Tree-Growing, a **skip-edge** is a non-tree edge whose endpoints are related; a **cross-edge** is a non-tree edge whose endpoints are not related.

**Example 4.1.3, continued:** For the output tree shown in the top half of Figure 4.1.3, there are three skip-edges and two cross-edges. For the other output tree, there are four skip-edges and one cross-edge.

### Depth-First and Breadth-First Search as Tree-Growing

We preview the two classical search algorithms presented in the next section by specifying two different selection rules for the function *nextEdge*. In fact, the *depth-first* and *breadth-first* searches use opposite versions of *nextEdge*.

- **depth-first:** *nextEdge* selects a frontier edge whose tree endpoint was most recently discovered.
- **breadth-first:** *nextEdge* selects a frontier edge whose tree endpoint was discovered earliest.

### Resolving Ties When Choosing the Next Frontier Edge

It must be emphasized that these last two selection criteria do *not* fully specify the function *nextEdge*, because, in general, they will produce ties. Typically, ties are resolved by some *default priority* that is likely to be part of the implementation of the data structures involved. For instance, if the graph is stored using a *table of incident edges* (§2.6), the table imposes a global ordering of the vertices and a local ordering of the incident edges on each vertex. These orderings can then be used to complete the specification of the function *nextEdge* by incorporating a tie-breaking rule.

To avoid getting bogged down in implementation details when we illustrate the different instances of Tree-Growing in this and the next two sections, we will often rely on the somewhat artificial lexicographic (alphabetical) order of the edge names and/or vertex names to resolve ties in choosing the next frontier edge. In practice, it is highly unlikely that lexicographic order would be used.

### Tree-Growing Applied to a Non-Connected Graph

**REVIEW FROM §2.3:** The **component of a vertex**  $v$  in a graph  $G$ , denoted  $C_G(v)$ , is the subgraph of  $G$  induced on the set of vertices that are reachable from  $v$ .

**Proposition 4.1.3.** *Let  $T$  be the output tree produced when an instance of Tree-Growing (Algorithm 4.1.1) is applied to a graph  $G$  (not necessarily connected), starting at a vertex  $v \in V_G$ . Then  $T$  is a spanning tree of the component  $C_G(v)$ .*

**Proof:** The result follows by induction, using Proposition 4.1.1. ◊

**Corollary 4.1.4.** *A graph  $G$  is connected if and only if the output tree produced when any instance of Tree-Growing is applied to  $G$  is a spanning tree of  $G$ .* ◊

### Tree-Growing in a Digraph

**DEFINITION:** A **frontier arc** for a rooted tree  $T$  in a digraph is an arc whose tail is in  $T$  and whose head is not in  $T$ .

Tree-growing for digraphs looks the same as for undirected graphs: in each iteration, a frontier arc is selected and added to the growing tree. However, there are notable differences. For instance, Corollary 4.1.4 provides an easy way of determining whether a given undirected graph is connected, but determining whether a digraph is strongly connected is more involved. In contrast with undirected graphs, the number of vertices in the output tree for a digraph depends on the choice of a starting vertex, as the next example shows.

**Example 4.1.4:** If Tree-Growing is applied to the 5-vertex digraph in Figure 4.1.4, then the number of vertices in the output tree ranges between 1 and 5, depending on the starting vertex. For instance, the output tree spans the digraph if the tree is grown starting at vertex  $u$ , but the output tree contains only one vertex if the starting vertex is  $v$ .



**Figure 4.1.4** The output tree for a digraph depends on the starting vertex.

**DEFINITION:** For tree-growing in digraphs, the non-tree edges (arcs) fall into three categories:

- A **back-arc** is directed from a vertex to one of its ancestors.
- A **forward-arc** is directed from a vertex to one of its descendants.
- A **cross-arc** is directed from a vertex to another vertex that is unrelated.

**TERMINOLOGY:** There are two kinds of cross-arcs: a **left-to-right cross-arc** is directed from smaller discovery number to larger one; a **right-to-left cross-arc** is the opposite.

**Remark:** In Chapter 12, a digraph version of tree-growing is adapted to determine whether a given digraph is strongly connected and to finding the strong components if it is not. As the last example suggests, the solution requires more than just checking to see whether all the vertices are discovered.

### Forest-Growing

The forest-growing scheme shown below is simply the repeated application of Tree-Growing. Each iteration grows a spanning tree for a new component by restarting the tree-growing at a vertex that had not been discovered previously. Forest-Growing uses a function  $nextVertex$ , which selects (deterministically) a restart vertex from the set of undiscovered vertices.

**DEFINITION:** A **full spanning forest** of a graph  $G$  is a spanning forest consisting of a collection of trees, such that each tree is a spanning tree of a different component of  $G$ .

**Algorithm 4.1.2: Forest-Growing and Component-Counting**

*Input:* a graph  $G$

*Output:* a full spanning forest  $F$  of  $G$  and the number of components  $c(G)$ .

Initialize forest  $F$  as the empty graph.

Initialize component counter  $t := 1$

While forest  $F$  does not yet span graph  $G$

    Let  $v = \text{nextVertex}(V_G - V_F)$ .

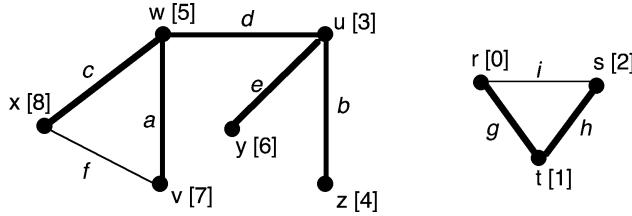
    Use Tree-Growing to produce a spanning tree  $T_t$  of  $C_G(v)$ .

    Add tree  $T_t$  to forest  $F$ .

$t := t + 1$

Return forest  $F$  and component count  $c(G) = t$ .

**Example 4.1.5:** Figure 4.1.5 shows the forest (with edges in bold) that results when the function *nextVertex* uses lexicographic order of vertex names and *nextEdge* uses lexicographic order of edge names. The bracketed numbers give the discovery order of the vertices. Notice that the forest consists of two trees that are spanning trees of their respective components.



**Figure 4.1.5** Growing a 2-component spanning forest of a graph.

### Some Implementation Considerations

COMPUTATIONAL NOTE 1: Any implementation of Tree-Growing (Algorithm 4.1.1) requires concrete data structures and operations. One of the options for the implementation of the procedure *updateFrontier* is to have it immediately discard at each iteration all old frontier edges that end at the newly discovered vertex  $w$ . However, certain applications may require some additional bookkeeping involving the old frontier edges, making their immediate removal premature.

Deciding how best to manage the frontier set is just one of several implementation details that depend on how the algorithm is to be used. But the goal here is to unify several classical algorithms in computer science and operations research, and sidestepping these issues makes it easier to view each of these algorithms as essentially the same tree-growing scheme, except for the rules by which the function *nextEdge* is defined. The reader who wants to learn more about the data structures necessary for a full implementation is encouraged to consult a standard computer science text on algorithms and data structures (e.g., [AhHoUl83], [Ba83], [Se88]).

### COMPUTATIONAL NOTE 2: Algorithmic Analysis of Tree-Growing

A complete analysis of the computational requirements of Tree-Growing (Algorithm 4.1.1) is not possible without fully specifying data structures and other implementation details. Instead, we provide a rough estimate.

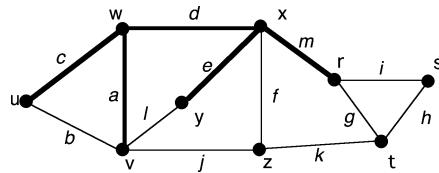
Suppose that the input graph has  $n$  vertices and  $m$  edges. Even if the adjacency-matrix representation (§2.6) is used to represent incidence (a possibility for simple graphs), Tree-Growing is implementable in  $O(n^2)$  running time. However, if a table of incident edges is used, then Tree-Growing is essentially  $O(m)$ , since each edge is examined once. Thus, for *sparse* graphs (graphs with relatively few edges, i.e.,  $m$  much less than  $\binom{n}{2}$ ), the tree-growing runs faster when the incidence-table representation is used.

One possibility is to store the frontier edges in a *priority queue*, leading to an algorithm having  $O(m \log m)$  running time.

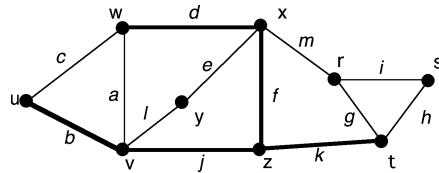
### EXERCISES for Section 4.1

*In Exercises 4.1.1 through 4.1.3, indicate the set of frontier edges for the given tree.*

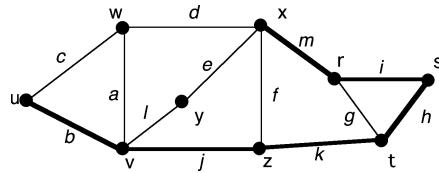
4.1.1<sup>s</sup>



4.1.2



4.1.3



*In Exercises 4.1.4 through 4.1.8, draw the output tree, including the discovery numbers, that results when Algorithm 4.1.1 is applied to the graph in Exercise 4.1.1. Start at the specified vertex and use: (a) lexicographic order for nextEdge; (b) reverse lexicographic order for nextEdge.*

4.1.4<sup>s</sup> Vertex  $w$

4.1.7 Vertex  $y$

4.1.5 Vertex  $u$

4.1.8<sup>s</sup> Vertex  $t$

4.1.6 Vertex  $x$

4.1.9 Apply Algorithm 4.1.2 to the graph in Figure 4.1.6, using reverse lexicographic order as the default priority for vertices and for edges.

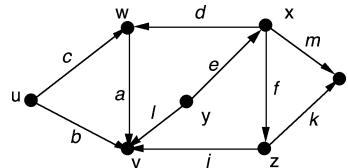
*In Exercises 4.1.10 through 4.1.13, draw the output tree that results when the digraph version of Algorithm 4.1.1 is applied to the digraph shown, starting at the specified vertex and using lexicographic order for nextEdge. Include the discovery numbers.*

4.1.10<sup>s</sup> Vertex  $u$ .

4.1.11 Vertex  $x$ .

4.1.12 Vertex  $w$ .

4.1.13 Vertex  $y$ .



**4.1.14** Add an edge  $q$  to the graph in Figure 4.1.3 so that the outputs produced by two instances of Algorithm 4.1.1, one using lexicographic order for  $\text{nextEdge}$ , and the other using reverse lexicographic order, both contain at least one cross-edge whose endpoints are at different levels of the output (rooted) tree.

**4.1.15** Prove Proposition 4.1.2.

**4.1.16<sup>s</sup>** Characterize those connected graphs for which there is a starting vertex such that the discovery order produced by Algorithm 4.1.1 is independent of the default priority of the edges.

**4.1.17** Use induction to prove that for any application of Algorithm 4.1.1, the discovered vertices are all reachable from the starting vertex.

**4.1.18** [Computer Project] Implement Algorithm 4.1.1 and compare its output with that obtained by hand for Exercises 4.1.4 through 4.1.8.

**4.1.19** [Computer Project] Implement Algorithm 4.1.2, and test the program on a 10-vertex 2-component graph and a 10-vertex 3-component graph.

## 4.2 DEPTH-FIRST AND BREADTH-FIRST SEARCH

*Depth-first search* and *breadth-first search* are integral parts of numerous algorithms used in computer science, operations research, and other engineering disciplines. To mention just a few of these, depth-first search is central to algorithms for finding the cut-vertices and blocks of a graph (§4.4 and §5.4), for finding the strong components of a digraph (§12.5), and for determining whether a graph can be drawn in the plane without edge-crossings (§7.6); breadth-first search is used for finding flow-augmenting paths as part of a flow-maximization algorithm for capacitated networks (§13.2).

**TERMINOLOGY:** The output trees produced by the depth-first and breadth-first searches of a graph are called the ***depth-first tree*** (or ***dfs-tree***) and the ***breadth-first tree*** (or  ***bfs-tree***).

**TERMINOLOGY NOTE:** The use of the word “search” here is traditional. Although one does occasionally use these algorithms to search for something, they are tree-growing algorithms with much wider use.

As previewed in §4.1, depth-first search and breadth-first search use two opposite priority rules for the function  $\text{nextEdge}$ .

### Depth-First Search

The idea of depth-first search is to select a frontier edge incident on the most recently discovered vertex of the tree grown so far. When that is not possible, the algorithm *backtracks* to the next most recently discovered vertex and tries again. Thus, the search moves deeper into the graph (hence, the name “depth-first”). In implementing Tree-Growing, depth-first search uses the following version of the function  $\text{nextEdge}$ .

**DEFINITION:** Let  $S$  be the current set of frontier edges. The function ***dfs-nextEdge*** is defined as follows:  $\text{dfs-nextEdge}(G, S)$  selects and returns as its value the frontier edge whose tree-endpoint has the largest discovery number. If there is more than one such edge, then  $\text{dfs-nextEdge}(G, S)$  selects the one determined by the default priority.

**Algorithm 4.2.1: Depth-First Search**

*Input:* a connected graph  $G$ , a starting vertex  $v \in V_G$ .  
*Output:* an ordered spanning tree  $T$  of  $G$  with root  $v$ .

```

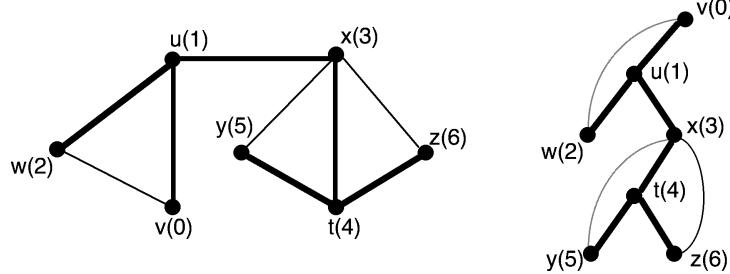
Initialize tree  $T$  as vertex  $v$ .
Initialize  $S$  as the set of proper edges incident on  $v$ .
While  $S \neq \emptyset$ 
    Let  $e = \text{dfs-nextEdge}(G, S)$ .
    Let  $w$  be the non-tree endpoint of edge  $e$ .
    Add edge  $e$  and vertex  $w$  to tree  $T$ .
     $\text{updateFrontier}(G, S)$ .
Return tree  $T$ .

```

**TERMINOLOGY:** The discovery number of each vertex  $w$  for a depth-first search is called the ***dfnumber*** of  $w$  and is denoted  $\text{dfnumber}(w)$ .

**Remark:** The term *dfnumber* is used instead of discovery number to underscore its special role in a number of applications of depth-first search. For instance, in §4.4 and §5.4, dfnumbers are used to determine the cut-vertices and blocks of a graph. In Chapter 12, they are used to find the strong components of a digraph.

**Example 4.2.1:** Figure 4.2.1 shows the output when the depth-first search is applied to the connected graph on the left, starting at vertex  $v$ . The function *nextEdge* resolves ties using lexicographic order of the non-tree endpoints of the frontier edges. The *dfnumbers* are shown in parentheses, and the tree edges are drawn in bold. Notice that there are no cross-edges. More specifically, for each non-tree edge, the endpoint with the smaller *dfnumber* is an ancestor of the other endpoint. The next proposition shows that this is always the case for a depth-first search of an *undirected* graph. In Chapter 12, it is shown that this property does not hold for digraphs.



**Figure 4.2.1** Depth-first search using lexicographic order for ties.

**Proposition 4.2.1.** *Depth-first search trees have no cross-edges.*

**Proof:** Let  $T$  be the output tree produced by a depth-first search, and let  $e$  be a non-tree edge whose endpoints  $x$  and  $y$  satisfy  $\text{dfnumber}(x) < \text{dfnumber}(y)$ . At the point when the depth-first search discovered vertex  $x$ , edge  $e$  became a frontier edge. Since  $e$  never became a tree edge, the search had to have discovered vertex  $y$  before backtracking to vertex  $x$  for the last time. Thus,  $y$  is in the subtree rooted at  $x$  and, hence, is a descendant of  $x$ .  $\diamond$

**Remark:** A preorder traversal (§3.3) of the depth-first tree reproduces the discovery order generated by the depth-first search (see Exercises).

### Recursive Depth-First Search

A depth-first search, starting at a vertex  $v$ , has the following informal recursive description.

$\text{DFS}(v)$ :

```
While  $v$  has an undiscovered neighbor,  
    Let  $w$  be an undiscovered neighbor of  $v$ .  
     $\text{DFS}(w)$   
Return
```

**COMPUTATIONAL NOTE:** We mentioned in §4.1 that for tree-growing, the natural way to store the frontier edges is to use a *priority queue*. For the particular case of depth-first search, the priority queue emulates the simpler *stack* data structure (Last-In-First-Out), because the newest frontier edges are given the highest priority by being pushed onto the stack (in increasing default priority order if there is more than newest frontier edge). The recursive aspect of depth-first search also suggests the feasibility of implementation as a stack.

### Depth-First Search in a Digraph

The depth-first search in a digraph is Algorithm 4.2.1 with the function  $dfs\text{-}nextArc$  replacing  $dfs\text{-}nextEdge$ . Also, we do not assume that the input digraph is strongly connected, so the dfs-tree produced will not necessarily be a spanning tree.

**DEFINITION:** The function ***dfs*-nextArc** selects and returns as its value the frontier arc whose tree-endpoint has the largest dfnumber.

**Proposition 4.2.2.** When a depth-first search is executed on a digraph, the only kind of non-tree arc that cannot occur is a left-to-right cross arc.  $\diamondsuit$  (Exercises)

**Remark:** Applications of depth-first search in a digraph appear in §4.4 and in Chapter 12.

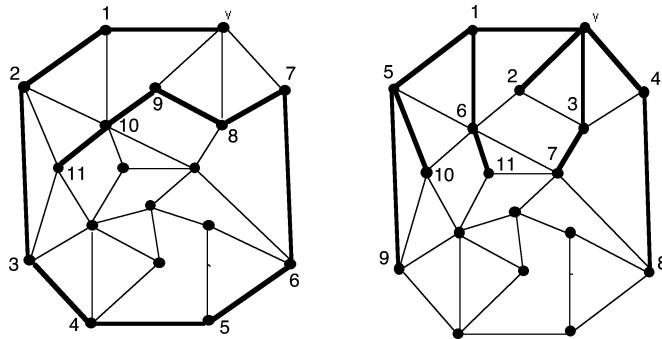
### Breadth-First Search

In the breadth-first search, the search “fans out” from the starting vertex and grows the tree by selecting frontier edges incident on vertices as close to the starting vertex as possible.

**Example 4.2.2:** Figure 4.2.2 below compares the results of depth-first and breadth-first searches by displaying typical possibilities for their partial output trees after 11 iterations, starting at vertex  $v$ . Of course, the exact partial trees depend on the exact rule for choosing a frontier edge. Assume that the respective  $nextEdge$  functions resolve ties by the same default priority.

Algorithm 4.2.2 below uses the following version of the function  $nextEdge$ .

**DEFINITION:** Let  $S$  be the current set of frontier edges. The function ***bfs*-nextEdge** is defined as follows:  $bfs\text{-}nextEdge(G, S)$  selects and returns as its value the frontier edge whose tree-endpoint has the smallest discovery number. If there is more than one such edge, then  $bfs\text{-}nextEdge(G, S)$  selects the one determined by the default priority.



**Figure 4.2.2** Depth-first and breadth-first search after 11 iterations.

**Algorithm 4.2.2: Breadth-First Search**

*Input:* a connected graph  $G$ , a starting vertex  $v \in V_G$ .

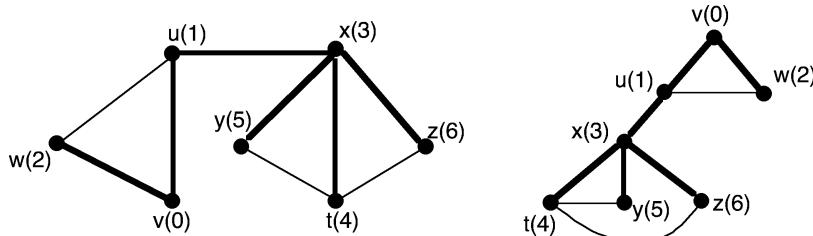
*Output:* an ordered spanning tree  $T$  of  $G$  with root  $v$ .

```

Initialize tree  $T$  as vertex  $v$ .
Initialize  $S$  as the set of proper edges incident on  $v$ .
While  $S \neq \emptyset$ 
    Let  $e = bfs\text{-}nextEdge(G, S)$ .
    Let  $w$  be the non-tree (undiscovered) endpoint of edge  $e$ .
    Add edge  $e$  and vertex  $w$  to tree  $T$ .
     $updateFrontier(G, S)$ .
Return tree  $T$ .

```

**Example 4.2.3:** Figure 4.2.3 shows the result of a breadth-first search applied to the same graph as in Example 4.2.1, again starting at vertex  $v$ . The function  $nextEdge$  resolves ties using lexicographic order of the non-tree endpoints of the frontier edges. Observe that the non-tree edges are all cross-edges, which is opposite from the result of depth-first search. Also notice that the breadth-first tree is a shortest-path tree (§3.2). Both these properties hold in general, as asserted by the next two propositions.



**Figure 4.2.3** Breadth-first search using lexicographic order for ties.

**Proposition 4.2.3.** When breadth-first search is applied to an undirected graph, the endpoints of each non-tree edge are either at the same level or at consecutive levels.

**Proof:** The result uses an argument analogous to the one given in the proof of Proposition 4.2.1.  $\diamond$  (Exercises)

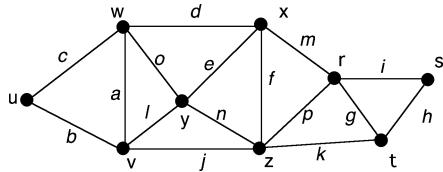
**Proposition 4.2.4.** The breadth-first tree produced by any application of Algorithm 4.2.2 is a shortest-path tree for the input graph.  $\diamond$  (Exercises)

**COMPUTATIONAL NOTE:** Whereas the stack (Last-In-First-Out) is the appropriate data structure to store the frontier edges in a depth-first search, the *queue* (First-In-First-Out) is most appropriate for the breadth-first search, since the frontier edges that are oldest have the highest priority.

## EXERCISES for Section 4.2

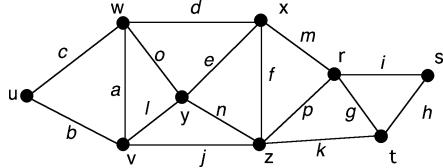
In Exercises 4.2.1 through 4.2.7, draw the depth-first tree that results when Algorithm 4.2.1 is applied to the graph shown below, starting at the specified vertex. Include the dfnumbers and use: (a) lexicographic order as the default priority; (b) reverse lexicographic order as the default priority.

- 4.2.1<sup>s</sup> Vertex  $w$ .
  - 4.2.2 Vertex  $u$ .
  - 4.2.3 Vertex  $x$ .
  - 4.2.4 Vertex  $y$ .
  - 4.2.5<sup>s</sup> Vertex  $t$ .
  - 4.2.6 Vertex  $z$ .
  - 4.2.7 Vertex  $s$ .



In Exercises 4.2.8 through 4.2.14, draw the breadth-first tree, including the discovery numbers, that results when Algorithm 4.2.2 is applied to the following graph. Start at the specified vertex and use: (a) lexicographic order as the default priority; (b) reverse lexicographic order as the default priority.

- 4.2.8** Vertex  $w$ .
  - 4.2.9<sup>s</sup>** Vertex  $u$ .
  - 4.2.10** Vertex  $x$ .
  - 4.2.11** Vertex  $y$ .
  - 4.2.12** Vertex  $t$ .
  - 4.2.13<sup>s</sup>** Vertex  $z$ .
  - 4.2.14** Vertex  $s$ .



**4.2.15** Characterize those graphs for which the depth-first and breadth-first trees that result are identical, no matter what the default priority and starting vertex are.

**4.2.16<sup>s</sup>** What kinds of graphs have a unique depth-first spanning tree that is independent of the starting vertex?

4.2.17 Verify that a preorder traversal of a depth-first search tree recreates the discovery order.

**4.2.18** Is there a traversal from Chapter 3 that can be applied to a breadth-first tree to recreate the discovery order of the breadth-first search? Justify your answer.

#### 4.2.19 Prove Proposition 4.2.2.

#### 4.2.20 Prove Proposition 4.2.3.

#### 4.2.21 Prove Proposition 4.2.4.

**4.2.22 [Computer Project]** Implement a recursive depth-first search and use an adjacency matrix (§2.6) to store the input graph. Compare the output from the computer program with the results obtained by hand in Exercises 4.2.1 through 4.2.7.

**4.2.23 [Computer Project]** Implement a recursive depth-first search and use an incidence matrix (§2.6) to store the input graph. Compare the output from the computer program with the results obtained by hand in Exercises 4.2.8 through 4.2.14.

**4.2.24 [Computer Project]** Implement Algorithm 4.2.1 using stacks, and use an adjacency matrix (§2.6) to store the input graph. Compare the output from the computer program with the results obtained by hand in Exercises 4.2.1 through 4.2.7.

**4.2.25 [Computer Project]** Implement Algorithm 4.2.1 using stacks, and use an incidence matrix (§2.6) to store the input graph. Compare the output from the computer program with the results obtained by hand in Exercises 4.2.8 through 4.2.14.

**4.2.26 [Computer Project]** Implement Algorithm 4.2.2 using queues, and use an adjacency matrix to store the input graph. Compare the output from the computer program with the results obtained by hand in Exercises 4.2.1 through 4.2.7.

**4.2.27 [Computer Project]** Implement Algorithm 4.2.2 using queues, and use an incidence matrix to store the input graph. Compare the output from the computer program with the results obtained by hand in Exercises 4.2.8 through 4.2.14.

**4.2.28 [Computer Project]** Modify Algorithm 4.2.1 so that it halts as soon as a cycle is detected. Then implement the algorithm and test the program on at least two acyclic graphs and at least two with cycles.

## 4.3 MINIMUM SPANNING TREES AND SHORTEST PATHS

The two classical algorithms presented in this section are instances of the generic tree-growing algorithm given in §4.1, applied on a weighted graph.

**REVIEW FROM §1.6:** A **weighted graph** is a graph in which each edge is assigned a number, called its **edge-weight**.

### Finding the Minimum Spanning Tree: Prim's Algorithm

*Minimum-Spanning-Tree Problem:* Let  $G$  be a connected weighted graph. Find a spanning tree of  $G$  whose total edge-weight is a minimum.

We introduced this problem in §1.6. By Cayley's formula (§3.7), the number of different spanning trees of an  $n$ -vertex graph could be as many as  $n^{n-2}$ . Thus, an exhaustive examination of all the spanning trees is clearly impractical for large graphs.

The following algorithm, attributed to R. C. Prim [Pr57], finds a minimum spanning tree without explicitly examining all of the spanning trees. The basic idea is to start at any vertex and “grow” a **Prim tree** by adding the frontier edge of smallest edge-weight at each iteration. Thus, Prim's algorithm is an instance of Tree-Growing, that uses the following version of the function *nextEdge*. Since *Prim-nextEdge*( $G, S$ ) and its associated procedure *updateFrontier* run in low-order polynomial-time, so does Prim's algorithm.

**DEFINITION:** Let  $S$  be the current set of frontier edges. The function **Prim-nextEdge** is defined as follows:  $\text{Prim-nextEdge}(G, S)$  selects and returns as its value the frontier edge with smallest edge-weight. If there is more than one such edge, then  $\text{Prim-nextEdge}(G, S)$  selects the one determined by the default priority.

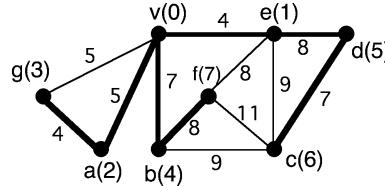
**Algorithm 4.3.1: Prim's Minimum Spanning Tree**

*Input:* a weighted connected graph  $G$  and starting vertex  $v$ .  
*Output:* a minimum spanning tree  $T$ .

```

Initialize tree  $T$  as vertex  $v$ .
Initialize  $S$  as the set of proper edges incident on  $v$ .
While  $S \neq \emptyset$ 
    Let  $e = \text{Prim-nextEdge}(G, S)$ .
    Let  $w$  be the non-tree endpoint of edge  $e$ .
    Add edge  $e$  and vertex  $w$  to tree  $T$ .
     $\text{updateFrontier}(G, S)$ .
Return tree  $T$ .
```

**Example 4.3.1:** Figure 4.3.1 shows the minimum spanning tree produced by Prim's algorithm for the input graph shown, starting at vertex  $v$ . The function  $\text{nextEdge}$  uses lexicographic order: of the names of the non-tree endpoints of frontier edges as a first tie-breaker and of the tree-endpoints as a second tie-breaker. The discovery numbers appear in parentheses.



**Figure 4.3.1** Minimum spanning tree produced by Prim's algorithm.

The correctness of Prim's algorithm is an immediate consequence of the next proposition.

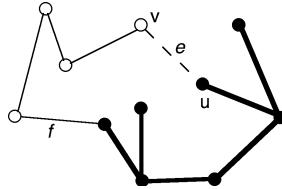
**Proposition 4.3.1.** *Let  $T_k$  be the Prim tree after  $k$  iterations of Prim's algorithm on a connected graph  $G$ , for  $0 \leq k \leq |V_G| - 1$ . Then  $T_k$  is a subtree of a minimum spanning tree of  $G$ .*

**Proof:** The assertion is trivially true for  $k = 0$ , since  $T_0$  is the trivial tree consisting of the starting vertex.

By way of induction, assume for some  $k$ ,  $0 \leq k \leq |V_G| - 2$ , that  $T_k$  is a subtree of a minimum spanning tree  $T$  of  $G$ , and consider tree  $T_{k+1}$ . The algorithm grew  $T_{k+1}$  during the  $(k+1)$ st iteration by adding to  $T_k$  a frontier edge  $e$  of smallest weight. Let  $u$  and  $v$  be the endpoints of edge  $e$ , such that  $u$  is in tree  $T_k$  and  $v$  is not.

If spanning tree  $T$  contains edge  $e$ , then  $T_{k+1}$  is a subtree of  $T$ . If  $e$  is not an edge in tree  $T$ , then  $e$  is part of the unique cycle contained in  $T + e$ . Consider the path in  $T$  from  $u$  to  $v$  consisting of the “long way around the cycle”. On this path, let  $f$  be the first edge that joins a vertex in  $T_k$  to a vertex not in  $T_k$ . The situation is illustrated in Figure 4.3.2; the black vertices and bold edges make up Prim tree  $T_k$ , the spanning tree  $T$  consists of everything except edge  $e$ , and Prim tree  $T_{k+1} = (T_k \cup v) + e$ .

Since  $f$  was a frontier edge at the beginning of the  $(k+1)$ st iteration,  $w(e) \leq w(f)$  (since the algorithm chose  $e$ ). Tree  $\hat{T} = T + e - f$  clearly spans  $G$ , and  $T_{k+1}$  is a subtree of  $\hat{T}$  (since  $f$  was not part of  $T_k$ ). Finally,  $w(\hat{T}) = w(T) + w(e) - w(f) \leq w(T)$ , which shows that  $\hat{T}$  is a minimum spanning tree of  $G$ .  $\diamond$



**Figure 4.3.2**

**Corollary 4.3.2.** When Prim's algorithm is applied to a connected graph, it produces a minimum spanning tree.

**Proof:** Proposition 4.3.1 implies that the Prim tree resulting from the final iteration is a minimum spanning tree.  $\diamond$

At the end of this chapter, a different (i.e., not an instance of Tree-Growing) but comparably fast algorithm for finding a minimum spanning tree, known as *Kruskal's algorithm*, is presented.

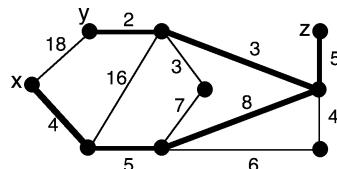
### The Steiner-Tree Problem

If instead of requiring that the minimum-weight tree *span* the given graph, we require that it simply include a prescribed *subset* of the vertices, we get a generalization of the minimum-spanning-tree problem, called the *Steiner-tree problem*.

**DEFINITION:** Let  $U$  be a subset of vertices in a connected edge-weighted graph  $G$ . The **Steiner-tree problem** is to find a minimum-weight tree subgraph of  $G$  that contains all the vertices of  $U$ .

Observe that if  $U = V_G$ , then the Steiner-tree problem reduces to the minimum-spanning tree problem.

**Example 4.3.2:** Figure 4.3.3 shows a weighted graph and a Steiner tree (with bold edges) for the vertex subset  $U = \{x, y, z\}$ .



**Figure 4.3.3** Steiner tree for  $U = \{x, y, z\}$ .

**Remark:** The Steiner-tree problem often arises in network-design and wiring-layout problems. It does not lend itself to the tree-growing strategy that we used for the minimum-spanning tree problem. A version of the Steiner-tree problem is discussed in §8.6 in the context of *graph drawings*. The case in which all the edges have equal weight (or are unweighted) gives rise to a generalization of distance, the *Steiner distance* of a subset of vertices (see §10.1).

### Finding the Shortest Path: Dijkstra's Algorithm

*Shortest-Path Problem:* Let  $s$  and  $t$  be two vertices of a connected weighted graph. Find a path from  $s$  to  $t$  whose total edge-weight is minimum, i.e., a shortest  $s$ - $t$  path.

As the name of the problem suggests, edge-weight is taken to mean *distance*, but the word is used loosely and can represent some other measurable quantity (e.g., time or cost).

**Remark:** If the edge-weights are all equal, then the problem reduces to one that can be solved by breadth-first search (Algorithm 4.2.2).

The algorithm presented here is another instance of generic tree-growing and is attributed to E. Dijkstra [Di59]. The algorithm does slightly more than the problem requires: it finds a shortest path from vertex  $s$  to *each* of the vertices of the graph. That is, it produces a shortest-path tree for the input (weighted) graph. We assume that all edge-weights are nonnegative. Dijkstra's algorithm is not guaranteed to work for graphs that have edges with negative weight, but a good alternative in this case is Floyd's algorithm [Fl62].

The strategy for Dijkstra's algorithm is to grow a **Dijkstra tree**, starting at vertex  $s$ , by adding, at each iteration, a frontier edge whose non-tree endpoint is as close as possible to  $s$ . Thus, Dijkstra's algorithm is an instance of Tree-Growing that uses the following version of the function *nextEdge*.

**DEFINITION:** The function **Dijkstra-nextEdge** is defined as follows: Let  $S$  be the current set of frontier edges. *Dijkstra-nextEdge*( $G, S$ ) selects and returns as its value the frontier edge whose non-tree endpoint is closest to the start vertex  $s$ . If there is more than one such edge, then *Dijkstra-nextEdge*( $G, S$ ) selects the one determined by the default priority.

#### Algorithm 4.3.2: Dijkstra's Shortest Path

*Input:* a weighted connected graph  $G$  and starting vertex  $s$ .

*Output:* a shortest-path tree  $T$  with root  $s$ .

```

Initialize tree  $T$  as vertex  $s$ .
Initialize  $S$  as the set of proper edges incident on  $s$ .
While  $S \neq \emptyset$ 
    Let  $e := \text{Dijkstra-nextEdge}(G, S)$ .
    Let  $w$  be the non-tree endpoint of edge  $e$ .
    Add edge  $e$  and vertex  $w$  to tree  $T$ .
     $\text{updateFrontier}(G, S)$ .
Return tree  $T$ .

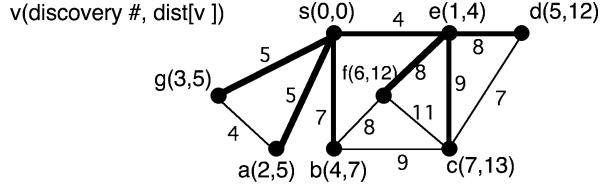
```

**REVIEW FROM §1.4:** The **distance** between two vertices  $s$  and  $t$  in a weighted graph is the length of a shortest  $s$ - $t$  path.

**NOTATION:** For each tree vertex  $x$ , let  $\text{dist}[x]$  denote the distance from vertex  $s$  to  $x$ .

**Example 4.3.3:** Figure 4.3.4 shows the shortest-path tree produced by Dijkstra's algorithm for the input graph shown, starting at vertex  $s$ . The function *nextEdge* uses lexicographic order: of the names of the non-tree endpoints of frontier edges as a first

tie-breaker and of the tree-endpoints as a second tie-breaker. In the parentheses at each vertex  $v$ , the discovery number appears first and  $dist[v]$  appears second.



**Figure 4.3.4** The shortest-path tree produced by Dijkstra's algorithm.

### Calculating Distances as the Dijkstra Tree Grows

NOTATION: For each frontier edge  $e$  in the weighted graph,  $w(e)$  denotes its edge-weight.

Notice that each tree edge  $q$  in Figure 4.3.4 has the following property: if  $x$  is the endpoint with the smaller discovery number and  $y$  is the other endpoint, then  $dist[y] = dist[x] + w(q)$ . Thus, when  $q$  was the frontier edge selected by *Dijkstra-nextEdge*, the value of  $dist[x] + w(q)$  must have been a minimum over all frontier edges in that iteration.

This suggests the following definition, which enables the function *Dijkstra-nextEdge* to be efficiently calculated. It is also used in the proof of correctness of Dijkstra's algorithm (Theorem 4.3.3 below).

DEFINITION: Let  $e$  be a frontier edge of the Dijkstra tree grown so far, and let  $x$  be the tree endpoint of  $e$ . The **P-value** of edge  $e$ , denoted  $P(e)$ , is given by

$$P(e) = dist[x] + w(e)$$

Thus, *Dijkstra-nextEdge*( $G, S$ ) selects and returns as its value the edge  $e^*$  such that  $P(e^*) = \min_{e \in S} \{P(e)\}$ . (As usual, if there is more than one such edge, then *Dijkstra-nextEdge*( $G, S$ ) selects the one determined by the default priority.)

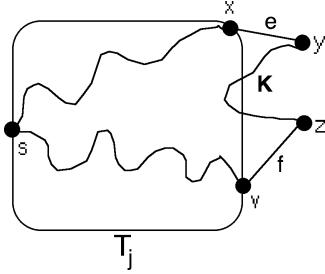
### Correctness of Dijkstra's Algorithm

The following theorem establishes the correctness of Dijkstra's algorithm. Its proof is similar to the one used to show the correctness of Prim's algorithm.

**Theorem 4.3.3.** *Let  $T_j$  be the Dijkstra tree after  $j$  iterations of Dijkstra's algorithm on a connected graph  $G$ , for  $0 \leq j \leq |V_G| - 1$ . Then for each  $v$  in  $T_j$ , the unique  $s-v$  path in  $T_j$  is a shortest  $s-v$  path in  $G$ .*

**Proof:** The assertion is trivially true for  $T_0$ . By way of induction, assume for some  $j$ ,  $0 \leq j \leq |V_G| - 2$ , that  $T_j$  satisfies the property. Let  $e$ , with labeled endpoint  $x$  and unlabeled endpoint  $y$ , be the frontier edge added to  $T_j$  in the  $(j+1)^{st}$  iteration. Since  $y$  is the only new vertex in  $T_{j+1}$ , it suffices to show that the  $s-y$  path  $Q$  in  $T_{j+1}$  is a shortest  $s-y$  path in  $G$ . The algorithm formed  $Q$  by concatenating the  $s-x$  path in  $T_j$  with edge  $e$ , and  $\text{length}(Q) = P(e)$ .

Now let  $R$  be any  $s-y$  path in  $G$ . We must show that  $\text{length}(R) \geq \text{length}(Q)$ . Suppose that edge  $f$ , with endpoints  $v$  and  $z$ , is the first edge of path  $R$  that is not in  $T_j$ , and let  $K$  be the portion of  $R$  from  $z$  to  $y$  (see Figure 4.3.5 below).



**Figure 4.3.5 Proof of correctness of Dijkstra's algorithm.**

Since the edge  $e$  was the frontier edge selected in the  $(j + 1)^{st}$  iteration, it follows that  $P(e) \leq P(f)$ . Thus,

$$\begin{aligned} \text{length}(R) &= \text{dist}[v] + w(f) + \text{length}(K) = P(f) + \text{length}(K) \\ &\geq P(e) + \text{length}(K) \geq P(e) = \text{length}(Q) \end{aligned}$$

where the rightmost inequality follows by the nonnegativity of the edge-weights.  $\diamond$

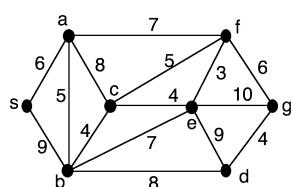
**COMPUTATIONAL NOTE:** Each time a new vertex  $y$  is added to the Dijkstra tree, the set of frontier edges can be updated without recomputing any  $P$ -values. In particular, delete each previous frontier edge incident on vertex  $y$  (because both its endpoints are now in the tree), and compute the priorities of the new frontier edges only (i.e., those that are incident on vertex  $y$ ). This modification significantly decreases the running time.

Also, if there are two frontier edges  $e$  and  $f$  having the same undiscovered endpoint, then the edge with the larger  $P$ -value will never be selected and therefore need never be added to the set of frontier edges. A full-blown implementation of Dijkstra's algorithm might include these strategies for maintaining the set of frontier edges.

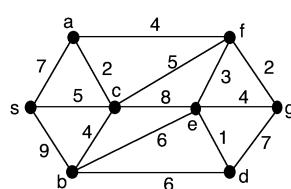
### EXERCISES for Section 4.3

In Exercises 4.3.1 through 4.3.4, apply Prim's algorithm (Algorithm 4.3.1) to the given weighted graph, starting at vertex  $s$  and resolving ties as in Example 4.3.1. Draw the minimum spanning tree that results, and indicate its total weight. Also give the discovery number at each vertex.

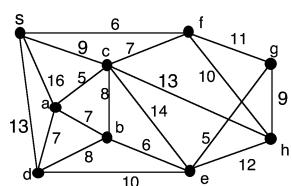
4.3.1<sup>s</sup>



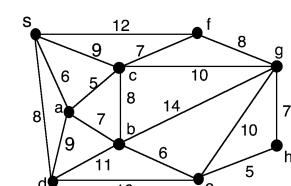
4.3.2



4.3.3



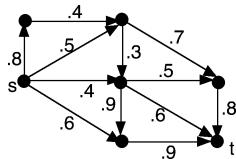
4.3.4



In Exercises 4.3.5 through 4.3.8, apply Dijkstra's algorithm (Algorithm 4.3.2) to the given weighted graph, starting at vertex  $s$  and resolving ties as in Example 4.3.3. Draw the shortest-path tree that results, and indicate for each vertex  $v$ , the discovery number and the distance from vertex  $s$  to  $v$ .

- |                    |                              |       |                              |
|--------------------|------------------------------|-------|------------------------------|
| 4.3.5 <sup>s</sup> | The graph of Exercise 4.3.1. | 4.3.6 | The graph of Exercise 4.3.2. |
| 4.3.7              | The graph of Exercise 4.3.3. | 4.3.8 | The graph of Exercise 4.3.4. |

4.3.9 Suppose that the weighted graph shown below represents a communication network, where the weight  $p_{ij}$  on arc  $ij$  is the probability that the link from  $i$  to  $j$  does not fail. If the link failures are independent of one another, then the probability that a path does not fail is the product of the link probabilities for that path. Under this assumption, find the most reliable path from  $s$  to  $t$ . (Hint: Consider  $-\log_{10} p_{ij}$ .)



- 4.3.10<sup>s</sup> Give an example of a weighted graph that has a unique minimum spanning tree, even though it contains a cycle having two edges of equal weight.

4.3.11 State a sufficient condition to guarantee that a given weighted graph does not have a unique minimum spanning tree. Is the condition also necessary?

4.3.12 Suppose that a breadth-first search is executed starting at each vertex of the input graph  $G$ . How would you identify the central vertices of  $G$ ?

4.3.13 Draw a graph  $G$  and a shortest-path tree for  $G$  that cannot be a breadth-first tree regardless of how ties are resolved.

4.3.14 [Computer Project] Implement Prim's spanning-tree algorithm (Algorithm 4.3.1), and run the program on each of the instances of the problem given in Exercises 4.3.1 through 4.3.4.

4.3.15 [Computer Project] Implement Dijkstra's shortest-path algorithm (Algorithm 4.3.2), and run the program on each of the instances of the problem given in Exercises 4.3.1 through 4.3.4.

## 4.4 APPLICATIONS OF DEPTH-FIRST SEARCH

In this section, depth-first search is applied to a connected graph. The results can easily be extended to non-connected graphs by considering each component separately.

### The Finish Order of a Depth-First Search

**DEFINITION:** During a depth-first search, a discovered vertex  $v$  is **finished** when all of its neighbors have been discovered.

Thus, a discovered vertex becomes finished at the instant there are no frontier edges incident on it. When the depth-first search reaches such a vertex, it will have to backtrack in order to continue the search.

**DEFINITION:** The **finish order** of a depth-first search is the order in which the vertices are finished.

**Remark:** In §4.2, we observed that a pre-order traversal of a dfs-tree reproduces the discovery order (Exercise 4.2.17). Regarding finish order, observe that the root is the last vertex finished in a depth-first search. In fact, a post-order traversal (§3.3) of the dfs-tree reproduces the finish order generated by the depth-first search (see Exercises).

### Growing a DFS-Path

A key strategy in many applications of depth-first search is to execute the search only until you have to backtrack for the first time.

**DEFINITION:** A **dfs-path** a path is the depth-first tree produced by executing a depth-first search and stopping before you backtrack for the first time.

**Proposition 4.4.1.** *Let  $G$  be a graph, and let  $P$  be the dfs-path produced by executing a depth-first search until the first vertex  $t$  becomes finished. Then all of the neighbors of  $t$  lie on path  $P$ .*

**Proof:** If a neighbor of  $t$  does not lie on the path  $P$ , then there is a frontier edge incident on  $t$ , contradicting the premise that  $t$  is finished.  $\diamond$

**Corollary 4.4.2.** *Let  $G$  be a connected graph with at least two vertices. If a dfs-path is grown until the first vertex  $t$  is finished, then either  $\deg(t) = 1$  or  $t$  and all its neighbors lie on a cycle of  $G$ .*

**Proof:** Let  $w$  be the neighbor of  $t$  that has the smallest dfnumber. By Proposition 4.4.1, the subpath of  $P$  from  $w$  to  $t$  contains at least  $\deg(t)$  vertices. If  $\deg(t) > 1$ , then vertices  $w$  and  $t$  are joined by a back edge, which completes a cycle that contains  $t$  and all of its neighbors.  $\diamond$

**Corollary 4.4.3.** *Let  $G$  be a connected simple graph with minimum degree  $\delta$ . Then  $G$  contains a cycle of length greater than  $\delta$ .*  $\diamond$

### Finding the Cut-Edges of a Connected Graph

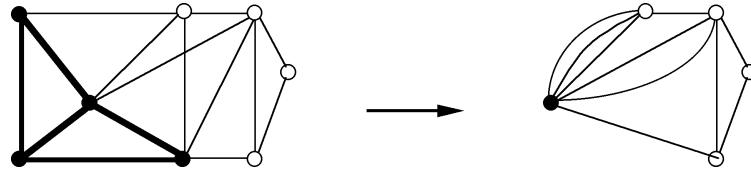
REVIEW FROM §2.4:

- A **cut-edge** (or **bridge**) of a connected graph  $G$  is an edge  $e$  such that the edge-deletion subgraph  $G - e$  has two components.
- An edge in a connected graph is a cut-edge if and only if it is a cycle edge.

**DEFINITION:** Let  $B$  be the set of cut-edges (bridges) of a connected graph  $G$ . A **bridge component (BC)** of  $G$  is a component of the subgraph  $G - B$  (§2.4).

**DEFINITION:** Let  $H$  be a subgraph of a graph  $G$ . The **contraction of  $H$  to a vertex** is the replacement of  $H$  by a single vertex  $k$ . Each edge that joined a vertex  $v \in V_G - V_H$  to a vertex in  $H$  is replaced by an edge with endpoints  $v$  and  $k$ .

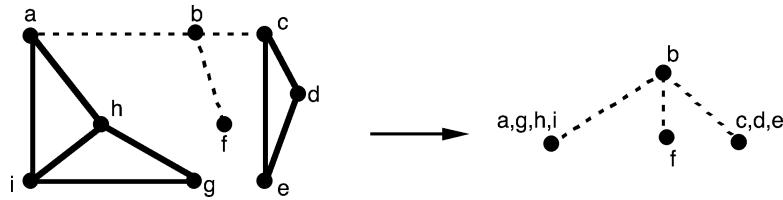
**Example 4.4.1:** The graph on the left in Figure 4.4.1 below has a subgraph  $H$  shown with bold edges. The graph on the right is the result of contracting  $H$  to a vertex.

**Figure 4.4.1** Contracting  $H$  to a vertex.

**Proposition 4.4.4.** Let  $G$  be a connected graph. All vertices on a cycle are in the same bridge component of  $G$ .  $\diamondsuit$  (Exercises)

**Proposition 4.4.5.** Let  $G$  be a connected graph. The graph that results from contracting each bridge component of  $G$  to a vertex is a tree.  $\diamondsuit$  (Exercises)

**Example 4.4.2:** The graph on the left in Figure 4.4.2, with its three cut-edges drawn with dashed lines, has four bridge components (two of them are single vertices). The contraction of each of them to a vertex results in the tree on the right.

**Figure 4.4.2** Contracting each bridge component to a vertex.

**Proposition 4.4.6.** Let  $v$  be a vertex of a connected graph  $G$  with  $\deg(v) \leq 1$ . Then the bridge components of  $G$  are the trivial subgraph  $x$  and the bridge components of  $G - x$ .  $\diamondsuit$  (Exercises)

The last three propositions together with Corollary 4.4.2 justify the following high-level algorithm for finding the cut-edges of a graph.

**Algorithm 4.4.1: Finding Cut-Edges**

*Input:* a connected graph  $G$ .

*Output:* the cut-edges of  $G$ .

```

Initialize graph  $H$  as graph  $G$ .
While  $|V_H| > 1$ 
    Grow a dfs-path to the first vertex  $t$  that becomes finished.
    If  $\deg(t) = 1$ 
        Mark the edge incident on  $t$  as a bridge.
         $H := H - t$ .
    Else /* vertex  $t$  and all its neighbors lie on a cycle  $C$  */
        Let  $H$  be the result of contracting cycle  $C$  to a vertex.
Return

```

**COMPUTATIONAL NOTE:** For efficiency, the dfs-path grown in each iteration should start at the same vertex and use as much of the previous dfs-path as possible.

### Topological Sorting by Depth-First Search

As we saw in §4.2, depth-first search extends easily to digraphs. Moreover, the strategy of growing a dfs-path is applicable to digraphs and is adapted here to perform a *topological sort* on a directed acyclic graph.

**DEFINITION:** A **dag** is a directed acyclic graph, i.e., it has no directed cycles.

**DEFINITION:** A **topological sort** (or **topological order**) of an  $n$ -vertex dag  $G$  is a bijection  $ts$  from the set  $\{1, 2, \dots, n\}$  to the vertex-set  $V_G$  such that every arc  $e$  is directed from a smaller number to a larger one, i.e.,  $ts(tail(e)) < ts(head(e))$ .

**DEFINITION:** A **source** in a dag is a vertex with indegree 0, and a **sink** is a vertex with outdegree 0.

**Proposition 4.4.7.** Every dag has at least one source and at least one sink.  $\diamond$   
(Exercises)

**Remark:** Simply using the discovery order (i.e., the dfnumbers) produced by a depth-first search will not necessarily be a topological sort because right-to-left cross arcs may occur (Proposition 4.2.2). However, the reverse finish order can be used.

**Proposition 4.4.8.** Let  $G$  be a dag, and let  $t$  be the vertex that becomes finished during an execution of a depth-first search. Then  $t$  is a sink.

**Proof:** If vertex  $t$  were not a sink, then  $G$  would contain a directed cycle.  $\diamond$

This last proposition leads to the following high-level algorithm for a topological sort. The algorithm grows a dfs-path, assigns  $n$  to the vertex that finishes first, deletes that vertex, and grows another dfs-path on the digraph that remains, etc.

#### Algorithm 4.4.2: Topological Sort

*Input:* an  $n$ -vertex dag  $G$ .

*Output:* a topological sort  $ts$  of the vertices of  $G$ .

Initialize graph  $H$  as graph  $G$ .

Initialize  $k := n$ .

While  $V_H \neq \emptyset$

    Start at a source and grow a dfs-path until the first vertex  $t$  is finished.

    Set  $ts(t) = k$

$H := H - t$ .

$k := k - 1$ .

Return the function  $ts$ .

**COMPUTATIONAL NOTE:** Again, as in Algorithm 4.4.1, efficiency dictates that the dfs-path grown in each iteration should start at the same vertex and use as much of the previous dfs-path as possible.

**Remark:** Topological sorting is discussed in the context of partial orders in Chapter 12.

### Finding the Cut-Vertices of a Connected Graph

The next two characterizations of a cut-vertex lead to a method for finding cut-vertices.

**Proposition 4.4.9.** *A vertex  $v$  in a connected graph is a cut-vertex if and only if there exist two distinct vertices  $u$  and  $w$ , both different from  $v$ , such that  $v$  is on every  $u-w$  path in the graph.*

**Proof:** *Necessity ( $\Rightarrow$ )* If  $v$  is a cut-vertex of a connected graph  $G$ , then there are vertices, say  $u$  and  $w$ , in separate components of  $G - v$ . It follows that every  $u-w$  path in  $G$  must contain  $v$ .

*Sufficiency ( $\Leftarrow$ )* If  $u$  and  $w$  are vertices of a connected graph  $G$ , such that every  $u-w$  path in  $G$  contains  $v$ , then  $G - v$  contains no  $u-w$  path. Thus,  $u$  and  $w$  are in different components of  $G - v$ .  $\diamond$

**Example 4.4.3:** In graph  $G$  of Figure 4.4.3, vertices  $v$  and  $x$  are the only cut-vertices. The rooted tree on the left represents one of the possible depth-first trees (depending on how ties are resolved) that could result when the search starts at cut-vertex  $v$ . The rooted tree on the right is generated by starting the search at vertex  $w$ , which is not a cut-vertex. Observe that cut-vertex  $w$  as the root has only one child, but  $v$  as the root has two children. Also notice that when either of the cut-vertices  $v$  or  $x$  is not the root, none of its descendants is joined to any of its proper ancestors. This last property does not hold for any other internal vertex of either search tree.

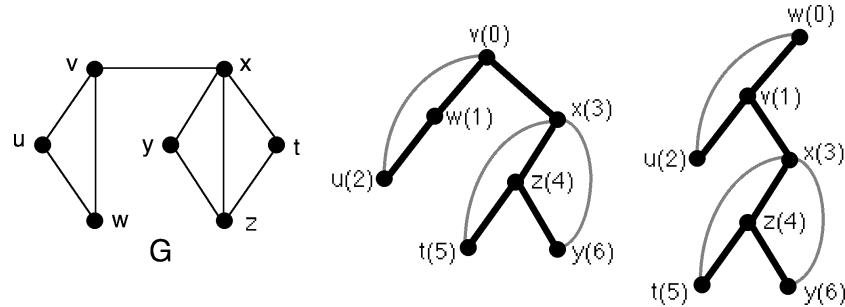


Figure 4.4.3 Depth-first trees with two different roots.

These observations lead to the next two propositions, which show how a depth-first tree and its non-tree edges can be used to determine whether a given vertex is a cut-vertex.

**Proposition 4.4.10.** *Let tree  $T$  be the result of applying depth-first search to a connected graph  $G$ . Then the root  $r$  of  $T$  is a cut-vertex of  $G$  if and only if  $r$  has more than one child in  $T$ .*

**Proof:** *Necessity ( $\Rightarrow$ )* By way of contrapositive, suppose that the root  $r$  has only one child  $w$  in  $T$ . Then all other vertices are descendants of  $w$ . Thus, the subtree rooted at  $w$  is a spanning tree of  $G - r$ , and, hence,  $G - r$  is connected.

*Sufficiency ( $\Leftarrow$ )* Suppose that  $x$  and  $y$  are two of the children of  $r$ . Neither  $x$  nor  $y$  is an ancestor of the other; furthermore, no descendant of  $x$  is an ancestor or descendant of any descendant of  $y$  (see Figure 4.4.4). Thus, by Proposition 4.2.1, there is no non-tree

edge joining any vertex in the subtree rooted at  $x$  to any vertex in the subtree rooted at  $y$ . This implies that every  $x-y$  path in  $G$  must go through  $r$ , and, hence, by Proposition 4.4.9,  $r$  is a cut-vertex.  $\diamond$

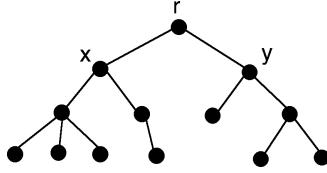


Figure 4.4.4

**Proposition 4.4.11.** *Let tree  $T$  be the result of applying depth-first search to a connected graph  $G$ . Then a non-root  $v$  of  $T$  is a cut-vertex of  $G$  if and only if  $v$  has a child  $w$  such that no descendant of  $w$  is joined to a proper ancestor of  $v$  by a non-tree edge.*

**Proof:** *Necessity ( $\Rightarrow$ )* By way of contrapositive, suppose that every child of  $v$  has a descendant joined to a proper ancestor of  $v$  (see Figure 4.4.5). First suppose that  $x$  and  $y$  are any two proper descendants of  $v$ . We claim that there exists an  $x-y$  path that does not contain  $v$ . If  $x$  and  $y$  are descendants of the same child of  $v$ , then the claim is trivially true. Otherwise,  $x$  and  $y$  are descendants of two different children of  $v$ , say  $w_1$  and  $w_2$ , respectively. Vertex  $w_1$  has a descendant  $d_1$  joined by a non-tree edge to a proper ancestor of  $v$ , say  $a_1$ . Similarly,  $w_2$  has a descendant  $d_2$  joined by a non-tree edge to a proper ancestor of  $v$ , say  $a_2$  (see Figure 4.4.5). Then the concatenation of the  $x-w_1$  path, the  $w_1-d_1$  path, edge  $d_1a_1$ , the  $a_1-a_2$  path, edge  $a_2d_2$ , the  $d_2-w_2$  path, and the  $w_2-y$  path forms an  $x-y$  path that does not contain  $v$ .

A similar argument shows that for any proper ancestor of  $v$  and any proper descendant of  $v$ , there is a path between them that avoids  $v$  (see Exercises). It follows that there is a path from any proper descendant of  $v$  to any vertex in a different subtree of  $r$ . Hence, by Proposition 4.4.9,  $v$  is not a cut-vertex.

*Sufficiency ( $\Leftarrow$ )* Suppose that  $v$  has a child  $w$  such that no descendant of  $w$  is joined to an ancestor of  $v$  by a non-tree edge. Then every path in  $G$  from  $w$  to  $r$  must pass through  $v$ , and, hence,  $v$  is a cut-vertex.  $\diamond$

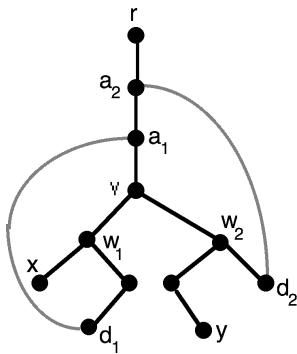


Figure 4.4.5

### Characterizing Cut-Vertices in Terms of the *dfnumbers*

By Propositions 4.2.1 and 4.4.11, it follows that a non-root  $v$  of a depth-first search tree is a cut-vertex if and only if  $v$  has a child  $w$  such that no descendant of  $w$  is joined by a non-tree edge to a vertex whose *dfnumber* is smaller than *dfnumber*( $v$ ).

**NOTATION:** Let  $low(w)$  denote the smaller of *dfnumber*( $w$ ) and the smallest *dfnumber* among all vertices joined by a non-tree edge to some descendant of  $w$ .

Proposition 4.4.11 may now be restated as the following corollary, on which a scheme for finding the cut-vertices (Algorithm 4.4.3) is based.

**Corollary 4.4.12.** *Let tree  $T$  be the result of applying depth-first search to a connected graph. Then a non-root  $v$  of  $T$  is a cut-vertex if and only if  $v$  has a child  $w$  such that  $low(w) \geq dfnumber(v)$ .*  $\diamond$

#### Algorithm 4.4.3: Finding Cut-Vertices

*Input:* a connected graph  $G$ .

*Output:* a set  $K$  of the cut-vertices of graph  $G$ .

Initialize the set  $K$  of cut-vertices as empty.

Choose an arbitrary vertex  $r$  of graph  $G$ .

Execute a depth-first search of graph  $G$ , starting at vertex  $r$ .

Let  $T$  be the output tree.

If root  $r$  has more than one child in  $T$

    Add vertex  $r$  to set  $K$ .

For each vertex  $w$

    Compute  $low(w)$ .

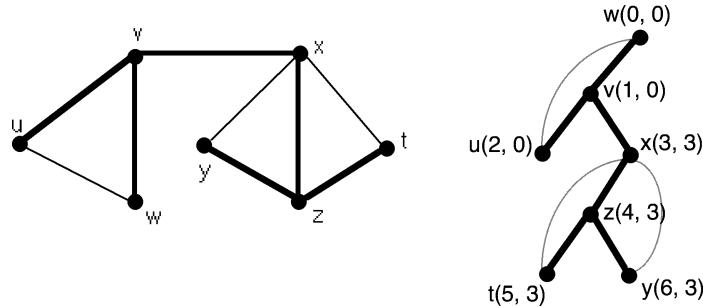
For each non-root  $v$

    If there is a child  $w$  of  $v$  such that  $low(w) \geq dfnumber(v)$

        Add vertex  $v$  to set  $K$ .

Return set  $K$ .

**Example 4.4.4:** The *dfnumber* and *low* values are computed for the example graph in Figure 4.4.6 and are shown in parentheses with the *dfnumber* listed first.



**Figure 4.4.6** The *dfnumber* and *low* values for a depth-first search.

**COMPUTATIONAL NOTE:** It is not hard to show that for any vertex  $v$ ,  $\text{low}(v)$  is the minimum of the following values:

- a.  $\text{dfnumber}(v)$ ,
- b.  $\text{dfnumber}(z)$  for any  $z$  joined to  $v$  by a non-tree edge, and
- c.  $\text{low}(y)$  for any child  $y$  of  $v$ .

This relationship allows the computation of the  $\text{low}$  values to take place *during* the depth-first search. In particular, when the vertex  $v$  has been backtracked to for the last time, the  $\text{low}$  values of each of the children of  $v$  will have already been computed, allowing  $\text{low}(v)$  then to be computed. (For further details, see [AhHoUl83].)

We close this section with an example illustrating how depth-first search can arise in unexpected ways.

### Escaping From a Maze: Tarry's Algorithm

Suppose you wake up inside a room in a maze consisting of a network of tunnels and rooms. At the end of each tunnel is an open doorway through which you can enter or exit a room. Your goal is to find your way to a *freedom* room from which you can exit the maze, if such a room exists, or determine that no such room exists. Each room has a piece of marking chalk and nothing else.

Gaston Tarry published a maze-tracing algorithm in 1895, now known as **Tarry's algorithm**, that avoids the danger of cycling forever. The algorithm is based on the following strategy: *never backtrack through a tunnel unless there is no alternative, and never go through a tunnel a second time in the same direction*. Thus, you must always traverse an unexplored tunnel before you resort to backtracking, which makes the strategy a variation of a depth-first search.

The chief difficulty here is in keeping track of which tunnels have already been traversed in a given direction. For an ordinary computer implementation of the corresponding graph, this problem is easily solved with the kind of routine bookkeeping that we have alluded to earlier. But without a map, there is no graph to store, and hence, all you can do is keep *local information* at each room. Tarry's solution involves marking the doorways according to the following rules:

- a. If you are in a room that is not the freedom room, and it has at least one unmarked doorway, then pass through one of the unmarked doorways to a tunnel, mark that doorway OUT, traverse the tunnel, pass through the doorway at the other end, and enter the room there.
- b. If you pass through a doorway and enter a room that is not the freedom room, and it has all doorways unmarked, mark the doorway you passed through IN.
- c. If you pass through a doorway and enter a room that is not the freedom room, and it has all doorways marked, pass through a doorway marked IN, if one exists, traverse the tunnel, pass through the doorway at the other end, and enter the room there.
- d. If you pass through a doorway and enter a room with all doorways marked OUT, stop.

**Proposition 4.4.13.** *If there is no freedom room in the maze, Tarry's algorithm will stop after each tunnel is traversed exactly twice, once in each direction.*  $\diamondsuit$  (Exercises)

**EXERCISES for Section 4.4**

In Exercises 4.4.1 through 4.4.7, (a) give the *dfnumber* and *low* values for each vertex that result from a depth-first search on the graph shown, starting at the specified vertex. Use the alphabetical order of the vertex names as the default priority; (b) verify the characterization given by Corollary 4.4.12 for the calculations of part (a).

4.4.1<sup>s</sup> Vertex *a*.

4.4.2 Vertex *b*.

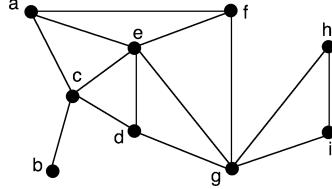
4.4.3 Vertex *c*.

4.4.4 Vertex *e*.

4.4.5<sup>s</sup> Vertex *g*.

4.4.6 Vertex *i*.

4.4.7 Vertex *f*.



4.4.8 Prove that a postorder traversal of a depth-first tree reproduces the finish order of the depth-first search.

4.4.9 Show that the assertion in Corollary 4.4.3 is not true for all non-simple graphs.

4.4.10 Prove Proposition 4.4.4.

4.4.11 Prove Proposition 4.4.5.

4.4.12 Prove Proposition 4.4.6.

4.4.13 Prove Proposition 4.4.7.

4.4.14 Complete the *Necessity* part of the proof of Proposition 4.4.11 by showing that for any ancestor of vertex *v* and any descendant of *v*, there is a path between them that avoids *v*.

4.4.15 Prove Proposition 4.4.13.

4.4.16 [Computer Project] Implement Algorithm 4.4.1, and test it on the graph used for Exercise 4.4.1.

4.4.17 [Computer Project] Implement Algorithm 4.4.3, and test it on the graph used for Exercises 4.4.1 through 4.4.7. Execute the program seven times, one for each of the starting vertices specified in those exercises.

## 4.5 CYCLES, EDGE-CUTS, AND SPANNING TREES

This section lays the foundation for the chapter's final two sections, where an algebraic structure for the cycles, edge-cuts, and spanning trees of a graph is revealed. Toward that end, we establish here a number of properties that highlight the relationship and a certain duality between the edge-cuts and cycles of a graph.

The first two propositions can be proved using properties of tree-growing (e.g., Corollary 4.1.4). However, the proofs presented here are more in keeping with the focus of these final sections.

**Proposition 4.5.1.** *A graph  $G$  is connected if and only if it has a spanning tree.*

**Proof:** If  $G$  is connected, then among the connected, spanning subgraphs of  $G$ , there is at least one, say  $T$ , with the least number of edges. If subgraph  $T$  contained a cycle, then the deletion of one of its cycle-edges would create a smaller connected, spanning subgraph, contradicting the minimality of  $T$ . Thus,  $T$  is acyclic and therefore, a spanning tree of graph  $G$ .

Conversely, if  $G$  contains a spanning tree, then every pair of vertices is connected by a path in the tree, and, hence,  $G$  is connected.  $\diamond$

**Proposition 4.5.2.** *A subgraph  $H$  of a connected graph  $G$  is a subgraph of some spanning tree if and only if  $H$  is acyclic.*

**Proof:** *Necessity* ( $\Rightarrow$ ) A subgraph of a tree is acyclic by definition.

*Sufficiency* ( $\Leftarrow$ ) Let  $H$  be an acyclic subgraph of  $G$ , and let  $T$  be any spanning tree of  $G$ . Consider the connected, spanning subgraph  $G_1$ , where  $V_{G_1} = V_T \cup V_H$  and  $E_{G_1} = E_T \cup E_H$ . If  $G_1$  is acyclic, then every edge of  $H$  must already be an edge of the spanning tree  $T$ , since otherwise a cycle would have been created (by the characterization theorem in §3.1). Thus, subgraph  $H$  is contained in tree  $T$ , and we are done. So suppose alternatively that  $G_1$  has a cycle  $C_1$ . Since  $H$  is acyclic, it follows that some edge  $e_1$  of cycle  $C_1$  is not in  $H$ . Then the graph  $G_2 = G_1 - e_1$  is still a connected, spanning subgraph of  $G$  and still contains  $H$  as a subgraph. If  $G_2$  is acyclic, then it is the required spanning tree. Otherwise, repeat the process of removing a cycle-edge not in  $H$  until a spanning tree is obtained.  $\diamond$

### Partition-Cuts and Minimal Edge-Cuts

REVIEW FROM §2.4: An **edge-cut**  $S$  in a graph  $G$  is a set of edges such that the edge-deletion subgraph  $G - S$  has more components than  $G$ .

The following definition is closely linked to the concept of a minimal edge-cut. It is used in this and the next section, and it also plays a central role in the study of *network flows*, which appears in Chapter 13.

**DEFINITION:** Let  $G$  be a graph, and let  $X_1$  and  $X_2$  form a partition of  $V_G$ . The set of all edges of  $G$  having one endpoint in  $X_1$  and the other endpoint in  $X_2$  is called a **partition-cut** of  $G$  and is denoted  $\langle X_1, X_2 \rangle$ .

The next two propositions make explicit the relationship between partition-cuts and minimal edge-cuts.

**Proposition 4.5.3.** *Let  $\langle X_1, X_2 \rangle$  be a partition-cut of a connected graph  $G$ . If the subgraphs of  $G$  induced by the vertex sets  $X_1$  and  $X_2$  are connected, then  $\langle X_1, X_2 \rangle$  is a minimal edge-cut.*

**Proof:** The partition-cut  $\langle X_1, X_2 \rangle$  is an edge-cut of  $G$ , since  $X_1$  and  $X_2$  lie in different components of  $G - \langle X_1, X_2 \rangle$ . Let  $S$  be a proper subset of  $\langle X_1, X_2 \rangle$ , and let edge  $e \in \langle X_1, X_2 \rangle - S$ . By definition of  $\langle X_1, X_2 \rangle$ , one endpoint of  $e$  is in  $X_1$  and the other endpoint is in  $X_2$ . Thus, if the subgraphs induced by the vertex sets  $X_1$  and  $X_2$  are connected, then  $G - S$  is connected. Therefore,  $S$  is not an edge-cut of  $G$ , which implies that  $\langle X_1, X_2 \rangle$  is a minimal edge-cut.  $\diamond$

**Proposition 4.5.4.** Let  $S$  be a minimal edge-cut of a connected graph  $G$ , and let  $X_1$  and  $X_2$  be the vertex-sets of the two components of  $G - S$ . Then  $S = \langle X_1, X_2 \rangle$ .

**Proof:** Clearly,  $S \subset \langle X_1, X_2 \rangle$ . If  $e \in \langle X_1, X_2 \rangle - S$ , then its endpoints would lie in the same component of  $G - S$ , contradicting the definition of  $X_1$  and  $X_2$ .  $\diamond$

**Remark:** The premise of Proposition 4.5.4 assumes that the removal of a minimal edge-cut from a connected graph creates *exactly* two components. This is a generalization of Corollary 2.4.3 and can be argued similarly, using the minimality condition (see Exercises).

**Proposition 4.5.5.** A partition-cut  $\langle X_1, X_2 \rangle$  in a connected graph  $G$  is a minimal edge-cut of  $G$  or a union of edge-disjoint minimal edge-cuts.

**Proof:** Since  $\langle X_1, X_2 \rangle$  is an edge-cut of  $G$ , it must contain a minimal edge-cut, say  $S$ . If  $\langle X_1, X_2 \rangle \neq S$ , then let  $e \in \langle X_1, X_2 \rangle - S$ , where the endpoints  $v_1$  and  $v_2$  of  $e$  lie in  $X_1$  and  $X_2$ , respectively. Since  $S$  is a minimal edge-cut, the  $X_1$ -endpoints of  $S$  are in one of the components of  $G - S$ , and the  $X_2$ -endpoints of  $S$  are in the other component. Furthermore,  $v_1$  and  $v_2$  are in the same component of  $G - S$  (since  $e \in G - S$ ). Suppose, without loss of generality, that  $v_1$  and  $v_2$  are in the same component as the  $X_1$ -endpoints of  $S$  (see Figure 4.5.1).

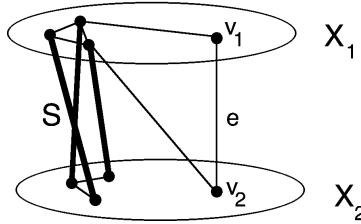


Figure 4.5.1

Then every path in  $G$  from  $v_1$  to  $v_2$  must use at least one edge of  $\langle X_1, X_2 \rangle - S$ . Thus,  $\langle X_1, X_2 \rangle - S$  is an edge-cut of  $G$  and, hence, contains a minimal edge-cut  $R$ . Applying the same argument,  $\langle X_1, X_2 \rangle - (S \cup R)$  either is empty or is an edge-cut of  $G$ . Eventually, the process ends with  $\langle X_1, X_2 \rangle - (S_1 \cup S_2 \cup \dots \cup S_r) = \emptyset$ , where the  $S_i$  are edge-disjoint minimal edge-cuts of  $G$ .  $\diamond$

### Fundamental Cycles and Fundamental Edge-Cuts

**DEFINITION:** Let  $G$  be a graph with  $c(G)$  components. The **edge-cut rank** of  $G$  is the number of edges in a full spanning forest of  $G$ . Thus, by Corollary 3.1.4, the edge-cut rank equals  $|V_G| - c(G)$ .

**REVIEW FROM §2.4:** Let  $H$  be a fixed subgraph of a graph  $G$ . The **relative complement of  $H$  (in  $G$ )**, denoted  $G - H$ , is the edge-deletion subgraph  $G - E(H)$ .

**DEFINITION:** Let  $G$  be a graph with  $c(G)$  components. The **cycle rank** (or **Betti number**) of  $G$ , denoted  $\beta(G)$ , is the number of edges in the relative complement of a full spanning forest of  $G$ . Thus, the cycle rank is  $\beta(G) = |E_G| - |V_G| + c(G)$ .

**Remark:** Observe that *all* of the edges in the relative complement of a spanning forest could be removed without increasing the number of components. Thus, the cycle rank  $\beta(G)$  equals the maximum number of edges that can be removed from  $G$  without increasing the number of components. Therefore,  $\beta(G)$  is a measure of the *edge redundancy* with respect to the graph's connectedness.

**REVIEW FROM §4.1:** A **full spanning forest** of a graph  $G$  is a spanning forest consisting of a collection of trees, such that each tree is a spanning tree of a different component of  $G$ .

**DEFINITION:** Let  $F$  be a full spanning forest of a graph  $G$ , and let  $e$  be any edge in the relative complement of forest  $F$ . The cycle in the subgraph  $F + e$  (existence and uniqueness guaranteed by the characterization theorem in §3.1) is called a **fundamental cycle of  $G$  (associated with the spanning forest  $F$ )**.

**Remark:** Each of the edges in the relative complement of a full spanning forest  $F$  gives rise to a *different* fundamental cycle.

**DEFINITION:** The **fundamental system of cycles** associated with a full spanning forest  $F$  of a graph  $G$  is the set of all fundamental cycles of  $G$  associated with  $F$ .

By the remark above, the cardinality of the fundamental system of cycles of  $G$  associated with a given full spanning forest of  $G$  is the cycle rank  $\beta(G)$ .

**DEFINITION:** Let  $F$  be a full spanning forest of a graph  $G$ , and let  $e$  be any edge of  $F$ . Let  $V_1$  and  $V_2$  be the vertex-sets of the two new components of the edge-deletion subgraph  $F - e$ . Then the partition-cut  $\langle V_1, V_2 \rangle$ , which is a minimal edge-cut of  $G$  by Proposition 4.5.3, is called a **fundamental edge-cut (associated with  $F$ )**.

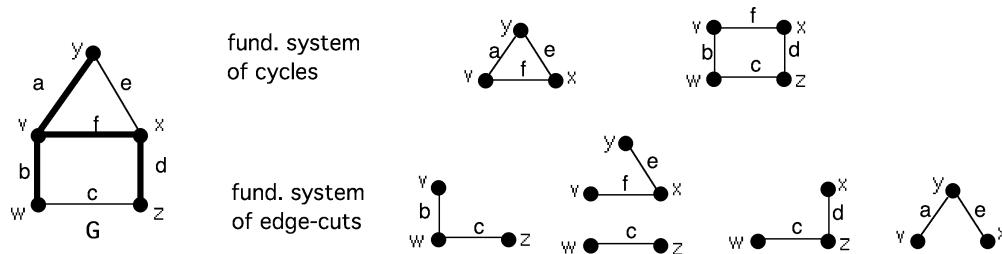
**Remark:** For each edge of  $F$ , its deletion gives rise to a different fundamental edge-cut.

**DEFINITION:** The **fundamental system of edge-cuts** associated with a full spanning forest  $F$  is the set of all fundamental edge-cuts associated with  $F$ .

Thus, the cardinality of the fundamental system of edge-cuts associated with a given full spanning forest of  $G$  is the edge-cut rank of  $G$ .

**Remark:** If  $F$  is a full spanning forest of a graph  $G$ , then each of the components of  $F$  is a spanning tree of the corresponding component of  $G$ . Since the removal or addition of an edge in a general graph affects only one of its components, the definitions of fundamental cycle and fundamental edge-cut are sometimes given in terms of a spanning tree of a connected graph. All of the remaining assertions in this section are stated in terms of a connected graph but can easily be restated for graphs having two or more components.

**Example 4.5.1:** Figure 4.5.2 below shows a fundamental system of cycles and a fundamental system of edge-cuts for a graph  $G$ . Both systems are associated with the spanning tree whose edges are drawn in bold. Notice that the fundamental system of edge-cuts does not contain every minimal edge-cut of graph  $G$ . For instance, the edge-cut consisting of the three edges incident on vertex  $v$  is a minimal one but is not in the fundamental system.



**Figure 4.5.2** Fundamental system of cycles and of edge-cuts.

### Relationship Between Cycles and Edge-Cuts

The next series of propositions reveals a dual relationship between the cycles and minimal edge-cuts of a graph. This duality is an instance of a more general property that holds for certain abstract structures called *matroids*, which are introduced in §4.7.

**Proposition 4.5.6.** *Let  $S$  be a set of edges in a connected graph  $G$ . Then  $S$  is an edge-cut of  $G$  if and only if every spanning tree of  $G$  has at least one edge in common with  $S$ .*

**Proof:** By Proposition 4.5.1,  $S$  is an edge-cut if and only if  $G - S$  contains no spanning tree of  $G$ , which means that every spanning tree of  $G$  has at least one edge in common with  $S$ .  $\diamond$

**Proposition 4.5.7.** *Let  $C$  be a set of edges in a connected graph  $G$ . Then  $C$  contains a cycle if and only if the relative complement of every spanning tree of  $G$  has at least one edge in common with  $C$ .*

**Proof:** By Proposition 4.5.2, edge set  $C$  contains a cycle if and only if  $C$  is not contained in any spanning tree of  $G$ , which means that the relative complement of every spanning tree of  $G$  has at least one edge in common with  $C$ .  $\diamond$

**Proposition 4.5.8.** *A cycle and a minimal edge-cut of a connected graph have an even number of edges in common.*

**Proof:** Let  $C$  be a cycle and  $S$  be a minimal edge-cut of a connected graph  $G$ . Let  $V_1$  and  $V_2$  be the vertex-sets of the two components  $G_1$  and  $G_2$  of  $G - S$ . Then each edge of  $S$  joins a vertex in  $V_1$  to a vertex in  $V_2$ . Now consider a traversal of the edges of the cycle  $C$ . Without loss of generality, assume that the traversal begins at some vertex in  $V_1$ . Then each time the traversal uses an edge in  $S$  in moving from  $V_1$  to  $V_2$ , it will have to return to  $V_1$  by traversing another edge of  $S$ . This is possible only if  $C$  and  $S$  have an even number of edges in common.  $\diamond$

**Example 4.5.1 continued:** It is easy but tedious to check that each of the three cycles in graph  $G$  of Figure 4.5.2 has either zero or two edges in common with each minimal edge-cut of  $G$ .

**Proposition 4.5.9.** *Let  $T$  be a spanning tree of a connected graph, and let  $C$  be a fundamental cycle with respect to an edge  $e^*$  in the relative complement of  $T$ . Then the edge-set of cycle  $C$  consists of edge  $e^*$  and those edges of tree  $T$  whose fundamental edge-cuts contain  $e^*$ .*

**Proof:** Let  $e_1, e_2, \dots, e_k$  be the edges of  $T$  that, with  $e^*$ , make up the cycle  $C$ , and let  $S_i$  be the fundamental edge-cut with respect to  $e_i$ ,  $1 \leq i \leq k$ .

Edge  $e_i$  is the only edge of  $T$  common to both  $C$  and  $S_i$  (by the definitions of  $C$  and  $S_i$ ). By Proposition 4.5.8,  $C$  and  $S_i$  must contain an even number of edges, and, hence, there must be an edge in the relative complement of  $T$  that is also common to both  $C$  and  $S_i$ . But  $e^*$  is the only edge in the complement of  $T$  that is in  $C$ . Thus, the fundamental edge-cut  $S_i$  must contain  $e^*$ ,  $1 \leq i \leq k$ .

To complete the proof, we must show that no other fundamental edge-cuts associated with  $T$  contain  $e^*$ . So let  $S$  be the fundamental edge-cut with respect to some edge  $b$  of  $T$ , different from  $e_1, e_2, \dots, e_k$ . Then  $S$  does not contain any of the edges  $e_1, e_2, \dots, e_k$  (by definition of  $S$ ). The only other edge of cycle  $C$  is  $e^*$ , so by Proposition 4.5.8, edge-cut  $S$  cannot contain  $e^*$ .  $\diamond$

**Example 4.5.1 continued:** The 3-cycle in Figure 4.5.2 is the fundamental cycle obtained by adding edge  $e$  to the given spanning tree. Of the four fundamental edge-cuts associated with that spanning tree, only the second and fourth ones contain edge  $e$ . The tree edges in these two edge-cuts, namely  $f$  and  $a$ , are the other two edges of the 3-cycle.

The proof of the next proposition uses an argument similar to the one just given and is left as an exercise.

**Proposition 4.5.10.** *The fundamental edge-cut with respect to an edge  $e$  of a spanning tree  $T$  consists of  $e$  and exactly those edges in the relative complement of  $T$  whose fundamental cycles contain  $e$ .*  $\diamond$  (Exercises)

This section closes with a characterization of eulerian graphs that dates back to as early as 1736, when Euler solved and generalized the Königsberg Bridge Problem (§6.1). The result is used in the next section, but it is important in its own right and is essential for parts of Chapter 6.

**REVIEW FROM §1.5:** An **eulerian tour** in a graph is a closed trail that contains every edge of that graph. An **eulerian graph** is a graph that has an eulerian tour.

**Theorem 4.5.11 [Eulerian-Graph Characterization].** *The following statements are equivalent for a connected graph  $G$ .*

1.  $G$  is eulerian.
2. The degree of every vertex in  $G$  is even.
3.  $E_G$  is the union of the edge-sets of a set of edge-disjoint cycles of  $G$ .

**Proof:** (1  $\Rightarrow$  2) Let  $C$  be an eulerian tour of  $G$ , and let  $v$  be the starting point of some traversal of  $C$ . The initial edge and final edge of the traversal contribute 2 toward the degree of  $v$ , and each time the traversal passes through a vertex, a contribution of 2 toward that vertex's degree also results. Thus, there is an even number of traversed edges incident with each vertex, and since each edge of  $G$  is traversed exactly once, this number is the degree of that vertex.

$(2 \Rightarrow 3)$  Since  $G$  is connected and every vertex has even degree,  $G$  cannot be a tree and therefore contains a cycle, say  $C_1$ . If  $C_1$  includes all the edges of  $G$ , then statement 3 is established. Otherwise, the graph  $G_1 = G - E_{C_1}$  has at least one nontrivial component. Furthermore, since the edges that were deleted from  $G$  form a cycle, the degree in  $G_1$  of each vertex on  $C_1$  is reduced by 2, and, hence, every vertex of  $G_1$  still has even degree. It follows that  $G_1$  contains a cycle  $C_2$ . If all of the edges have been exhausted, then  $E_G = E_{C_1} \cup E_{C_2}$ . Otherwise, consider  $G_2 = G - E_{C_1} - E_{C_2}$ , and continue the procedure until all the edges are exhausted. If  $C_n$  is the cycle obtained at the last step, then  $E_G = E_{C_1} \cup E_{C_2} \cup \dots \cup E_{C_n}$ , which completes the proof of the implication  $2 \Rightarrow 3$ .

$(3 \Rightarrow 1)$  Assume that  $E_G$  is the union of the edge-sets of  $m$  edge-disjoint cycles of  $G$ . Start at any vertex  $v_1$  on one of these cycles, say  $C_1$ , and consider  $T_1 = C_1$  as our first closed trail. There must be some vertex of  $T_1$ , say  $v_2$ , that is also a vertex on some other cycle, say  $C_2$ . Form a closed trail  $T_2$  by *splicing* cycle  $C_2$  into  $T_1$  at vertex  $v_2$ . That is, trail  $T_2$  is formed by starting at vertex  $v_1$ , traversing the edges of  $T_1$  until  $v_2$  is reached, traversing all the edges of cycle  $C_2$ , and completing the closed trail  $T_2$  by traversing the remaining edges of trail  $T_1$ . The process continues until all the cycles have been spliced in, at which point  $T_m$  is an eulerian tour of  $G$ .  $\diamond$

### EXERCISES for Section 4.5

In Exercises 4.5.1 through 4.5.5, consider the spanning tree  $T$  with the specified edge subset of the following graph  $G$ . (a) Find the fundamental system of cycles associated with  $T$ . (b) Find the fundamental system of edge-cuts associated with  $T$ .

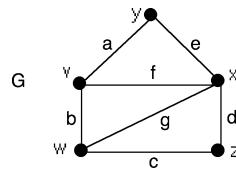
4.5.1<sup>s</sup> { $a, e, c, d$ }

4.5.2 { $a, e, g, d$ }

4.5.3 { $e, f, c, d$ }

4.5.4 { $a, b, g, c$ }

4.5.5 { $e, f, g, d$ }



In Exercises 4.5.6 through 4.5.9, consider the fundamental systems of cycles and edge-cuts obtained in the specified exercise. (a) Show that Proposition 4.5.6 holds for one of the fundamental edge-cuts. (b) Show that Proposition 4.5.7 holds for one of the fundamental cycles. (c) Show that the 5-cycle in graph  $G$  has an even number of edges in common with each of the fundamental edge-cuts.

4.5.6<sup>s</sup> Exercise 4.5.1.

4.5.7 Exercise 4.5.2.

4.5.8 Exercise 4.5.3.

4.5.9 Exercise 4.5.4.

In Exercises 4.5.10 through 4.5.13, consider the fundamental systems of cycles and edge-cuts obtained in the specified exercise. (a) Show that Proposition 4.5.9 holds for one of the fundamental cycles. (b) Show that Proposition 4.5.10 holds for one of the fundamental edge-cuts.

4.5.10<sup>s</sup> Exercise 4.5.1.

4.5.11 Exercise 4.5.2.

4.5.12 Exercise 4.5.3.

4.5.13 Exercise 4.5.4.

4.5.14 Prove that each edge of a connected graph  $G$  lies in a spanning tree of  $G$ .

4.5.15 Give an alternative proof of Proposition 4.5.2 that avoids the “repeat the process” phrase by considering a connected spanning subgraph of  $G$  that contains  $H$ , and that has the least number of edges among all such subgraphs.

- 4.5.16<sup>s</sup> Prove that in a tree, every vertex of degree greater than 1 is a cut-vertex.
- 4.5.17 Prove that every nontrivial connected graph contains a minimal edge-cut.
- 4.5.18 Prove that the removal of a minimal edge-cut from any graph increases the number of components by exactly 1.
- 4.5.19 Let  $T_1$  and  $T_2$  be two different spanning trees of a graph. Show that if  $e$  is any edge in tree  $T_1$ , then there exists an edge  $f$  in tree  $T_2$  such that  $T_1 - e + f$  is also a spanning tree. (Hint: Apply Proposition 4.5.6 to the fundamental edge-cut associated with  $T_1$  and  $e$ .)
- 4.5.20 Prove that a subgraph of a connected graph  $G$  is a subgraph of the relative complement of some spanning tree if and only if it contains no edge-cuts of  $G$ .
- 4.5.21 Prove Proposition 4.5.10.
- 4.5.22<sup>s</sup> Give a counterexample to show that the assertion of Proposition 4.5.8 no longer holds if the minimality condition is removed.
- 

## 4.6 GRAPHS AND VECTOR SPACES

The results of the previous section are used here to define a vector space associated with a graph. Two important subspaces of this vector space, the *edge-cut space* and the *cycle space*, are also identified and studied in detail. Understanding the algebraic structure underlying a graph's cycles and edge-cuts makes it possible to apply powerful and elegant results from linear algebra that lead to a deeper understanding of graphs and of certain graph algorithms. Supporting this claim is an application, appearing later this section, to a problem concerning electrical circuits.

### Vector Space of Edge Subsets

NOTATION: For a graph  $G$ , let  $W_E(G)$  denote the set of all subsets of  $E_G$ .

DEFINITION: The **ring sum** of two elements of  $W_E(G)$ , say  $E_1$  and  $E_2$ , is defined by

$$E_1 \oplus E_2 = (E_1 - E_2) \cup (E_2 - E_1)$$

The next proposition asserts that  $W_E(G)$  under the ring-sum operation forms a vector space over the field of scalars  $GF(2)$ , where the scalar multiplication  $*$  is defined by  $1 * S = S$  and  $0 * S = \emptyset$  for any  $S$  in  $W_E(G)$ . Its proof is a straightforward verification of each of the defining properties of a vector space and is left as an exercise. (The basic definitions and properties of a vector space and of the finite field  $GF(2)$  appear in Appendix A.4.)

**Proposition 4.6.1.**  $W_E(G)$  is a vector space over  $GF(2)$ .

◊ (Exercises)

TERMINOLOGY: The vector space  $W_E(G)$  is called the **edge space of  $G$** .

**Proposition 4.6.2.** Let  $E_G = \{e_1, e_2, \dots, e_m\}$  be the edge-set of a graph  $G$ . Then the subsets  $\{e_1\}, \{e_2\}, \dots, \{e_m\}$  form a basis for the edge space  $W_E(G)$ . Thus,  $W_E(G)$  is an  $m$ -dimensional vector space over  $GF(2)$ .

**Proof:** If  $H = \{e_{i_1}, e_{i_2}, \dots, e_{i_r}\}$  is any vector in  $W_E(G)$ , then  $H = \{e_{i_1}\} \oplus \{e_{i_2}\} \oplus \dots \oplus \{e_{i_r}\}$ . Clearly, the elements  $\{e_1\}, \{e_2\}, \dots, \{e_m\}$  are also linearly independent.  $\diamond$

**DEFINITION:** Let  $s_1, s_2, \dots, s_n$  be any sequence of objects, and let  $A$  be a subset of  $S = \{s_1, s_2, \dots, s_n\}$ . The **characteristic vector** of the subset  $A$  is the  $n$ -tuple whose  $j$ th component is 1 if  $s_j \in A$  and 0 otherwise.

A general result from linear algebra states that every  $m$ -dimensional vector space over a given field  $F$  is isomorphic to the vector space of  $m$ -tuples over  $F$ . For the vector space  $W_E(G)$ , this result may be realized in the following way. If  $E_G = \{e_1, e_2, \dots, e_m\}$ , then the mapping *charvec* that assigns to each subset of  $E_G$  its *characteristic vector* is an isomorphism from  $W_E(G)$  to the vector space of  $m$ -tuples over  $GF(2)$ . Proving this assertion first requires showing that the mapping *preserves the vector-space operations*. If  $E_1$  and  $E_2$  are two subsets of  $E_G$ , then the definitions of the ring-sum operator  $\oplus$  and mod 2 component-wise addition  $+_2$  imply

$$\text{charvec}(E_1 \oplus E_2) = \text{charvec}(E_1) +_2 \text{charvec}(E_2)$$

The remaining details are left to the reader.

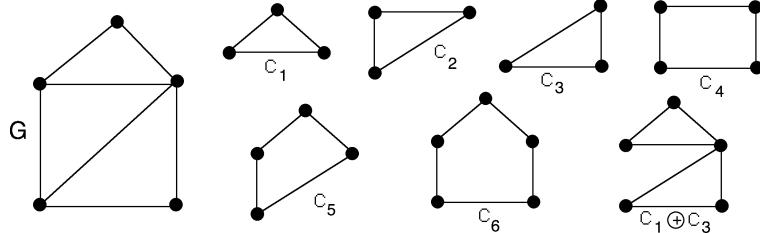
**Remark:** Each subset  $E_i$  of  $E_G$  uniquely determines a subgraph of  $G$ , namely, the edge-induced subgraph  $G_i = G(E_i)$ . Thus, the vectors of  $W_E(G)$  may be viewed as the edge-induced subgraphs  $G_i$  instead of as the edge subsets  $E_i$ . Accordingly,  $1 * G_i = G_i$  and  $0 * G_i = \emptyset$ , where  $G_i$  is any edge-induced subgraph of  $G$  and where  $\emptyset$  now refers to the *null graph* (with no vertices and no edges). This vector space will still be denoted  $W_E(G)$ .

Furthermore, the only subgraphs of a graph that are *not* edge-induced subgraphs are those that contain isolated vertices. Since isolated vertices play no role in the discussions in this section and in the section that follows, the adjective “edge-induced” will no longer be used when referring to the elements of the edge space  $W_E(G)$ .

### The Cycle Space of a Graph

**DEFINITION:** The **cycle space** of a graph  $G$ , denoted  $W_C(G)$ , is the subset of the edge space  $W_E(G)$  consisting of the null set (graph)  $\emptyset$ , all cycles in  $G$ , and all unions of edge-disjoint cycles of  $G$ .

**Example 4.6.1:** Figure 4.6.1 shows a graph  $G$  and the seven non-null elements of the cycle space  $W_C(G)$ .



**Figure 4.6.1** A graph  $G$  and the non-null elements of cycle space  $W_C(G)$ .

Notice that each vector of  $W_C(G)$  is a subgraph having no vertices of odd degree, and that the sum of any two of the vectors of  $W_C(G)$  is again a vector of  $W_C(G)$ . The first of these observations follows directly from the characterization of eulerian graphs from the previous section (Theorem 4.5.11). The next result shows that the second property of  $W_C(G)$  is also true in general.

**Proposition 4.6.3.** *Given a graph  $G$ , the cycle space  $W_C(G)$  is a subspace of the edge space  $W_E(G)$ .*

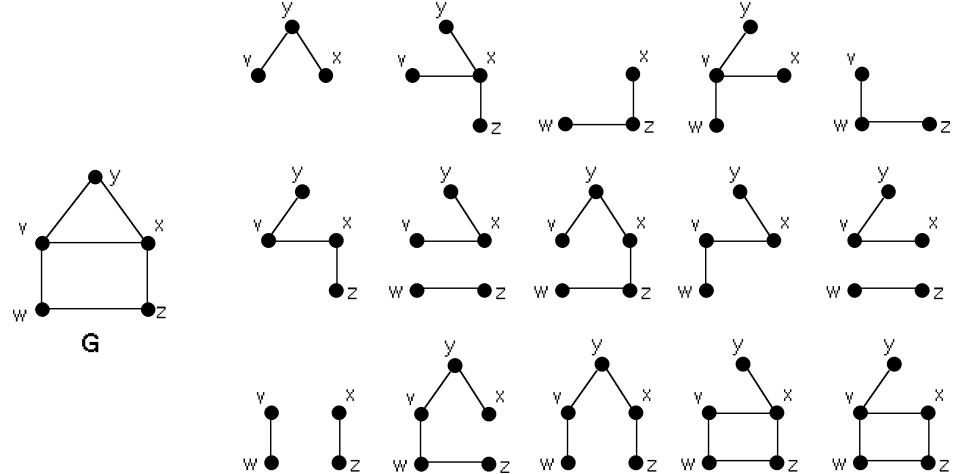
**Proof:** It suffices to show that the elements of  $W_C(G)$  are closed under  $\oplus$ . So consider any two distinct members  $C_1$  and  $C_2$  of  $W_C(G)$  and let  $C_3 = C_1 \oplus C_2$ . By Theorem 4.5.11, it must be shown that  $\deg_{C_3}(v)$  is even for each vertex  $v$  in  $C_3$ .

Consider any vertex  $v$  in  $C_3$ , and let  $X_i$  denote the set of edges incident with  $v$  in  $C_i$  for  $i = 1, 2, 3$ . Since  $|X_i|$  is the degree of  $v$  in  $C_i$ ,  $|X_1|$  and  $|X_2|$  are both even, and  $|X_3|$  is nonzero. Since  $C_3 = C_1 \oplus C_2$ ,  $X_3 = X_1 \oplus X_2$ . But this implies that  $|X_3| = |X_1| + |X_2| - 2|X_1 \cap X_2|$ , which shows that  $|X_3|$  must be even.  $\diamond$

### The Edge-Cut Subspace of a Graph

**DEFINITION:** The **edge-cut space** of a graph  $G$ , denoted  $W_S(G)$ , is the subset of the edge space  $W_E(G)$  consisting of the null graph  $\emptyset$ , all minimal edge-cuts in  $G$ , and all unions of edge-disjoint minimal edge-cuts of  $G$ .

**Example 4.6.2:** Figure 4.6.2 shows a graph  $G$  and the 15 non-null elements of the edge-cut space  $W_S(G)$ .



**Figure 4.6.2** A graph  $G$  and the non-null elements of edge-cut space  $W_S(G)$ .

**Proposition 4.6.4.** *Given a graph  $G$ , the edge-cut space  $W_S(G)$  is closed under the ring-sum operation  $\oplus$  and, hence, is a subspace of the edge space  $W_E(G)$ .*

**Proof:** By Proposition 4.5.3, it suffices to show that the ring sum of any two partition-cuts in a graph  $G$  is also a partition-cut in  $G$ . So let  $S_1 = \langle X_1, X_2 \rangle$  and  $S_2 = \langle X_3, X_4 \rangle$  be any two partition-cuts in  $G$ , and let  $V_{ij} = X_i \cap X_j$ , for  $i = 1, 2$  and  $j = 3, 4$ .

Then the  $V_{ij}$  are mutually disjoint, with

$$\begin{aligned} S_1 &= \langle V_{13} \cup V_{14}, V_{23} \cup V_{24} \rangle = \langle V_{13}, V_{23} \rangle \cup \langle V_{13}, V_{24} \rangle \cup \langle V_{14}, V_{23} \rangle \cup \langle V_{14}, V_{24} \rangle \\ \text{and } S_2 &= \langle V_{13} \cup V_{23}, V_{14} \cup V_{24} \rangle = \langle V_{13}, V_{14} \rangle \cup \langle V_{13}, V_{24} \rangle \cup \langle V_{23}, V_{14} \rangle \cup \langle V_{23}, V_{24} \rangle \end{aligned}$$

Hence,

$$S_1 \oplus S_2 = \langle V_{13}, V_{23} \rangle \cup \langle V_{14}, V_{24} \rangle \cup \langle V_{13}, V_{14} \rangle \cup \langle V_{23}, V_{24} \rangle$$

But

$$\langle V_{13}, V_{23} \rangle \cup \langle V_{14}, V_{24} \rangle \cup \langle V_{13}, V_{14} \rangle \cup \langle V_{23}, V_{24} \rangle = \langle V_{13} \cup V_{24}, V_{14} \cup V_{23} \rangle$$

which is a partition-cut in  $G$ , and the proof is complete.  $\diamond$

**Example 4.6.3:** To illustrate Proposition 4.6.4, it is easy to check that the elements of  $W_S(G)$  in Figure 4.6.2 are closed under ring sum  $\oplus$  and that each element is a partition-cut. For instance, if  $S_1$  denotes the last element in the second row of five elements, and  $S_2$  and  $S_3$  are the first and last elements of the third row, then  $S_1 = \langle \{v, w\}, \{x, y, z\} \rangle$ ;  $S_2 = \langle \{v, x, y\}, \{w, z\} \rangle$ ; and  $S_3 = S_1 \oplus S_2 = \langle \{x, y, w\}, \{v, z\} \rangle$ .

### Bases for the Cycle and Edge-Cut Spaces

**Theorem 4.6.5.** Let  $T$  be a spanning tree of a connected graph  $G$ . Then the fundamental system of cycles associated with  $T$  is a basis for the cycle space  $W_C(G)$ .

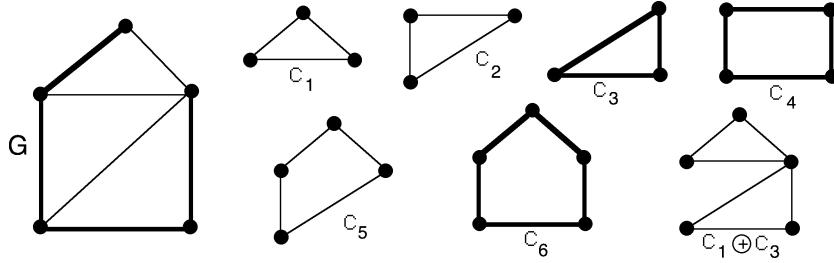
**Proof:** By the construction, each fundamental cycle associated with  $T$  contains exactly one non-tree edge that is not part of any other fundamental cycle associated with  $T$ . Thus, no fundamental cycle can be expressed as a ring sum of some or all of the other fundamental cycles. Hence, the fundamental system of cycles is a linearly independent set. To show the fundamental system of cycles spans  $W_C(G)$ , suppose  $H$  is any element of  $W_C(G)$ .

Now let  $e_1, e_2, \dots, e_r$  be the non-tree edges of  $H$ , and let  $C_i$  be the fundamental cycle in  $T + e_i$ ,  $i = 1, \dots, r$ . The completion of the proof requires showing that  $H = C_1 \oplus C_2 \oplus \dots \oplus C_r$ , or equivalently, that  $B = H \oplus C_1 \oplus C_2 \oplus \dots \oplus C_r$  is the null graph.

Since each  $e_i$  appears only in  $C_i$ , and  $e_i$  is the only non-tree edge in  $C_i$ ,  $B$  contains no non-tree edges. Thus,  $B$  is a subgraph of  $T$  and is therefore acyclic. But  $B$  is an element of  $W_C(G)$  (since  $W_C(G)$  is closed under ring sum), and the only element of  $W_C(G)$  that is acyclic is the null graph.  $\diamond$

**Example 4.6.1 continued:** Figure 4.6.3 shows a spanning tree and the associated fundamental system of cycles  $\{C_3, C_4, C_6\}$  for the graph of Figure 4.6.1. Each of the other four non-null elements of cycle space  $W_C(G)$  can be expressed as the mod 2 sum of some or all of the fundamental cycles. In particular,

$$\begin{aligned} C_1 &= C_4 \oplus C_6 \\ C_2 &= C_3 \oplus C_4 \\ C_5 &= C_3 \oplus C_6 \\ C_1 \oplus C_3 &= C_3 \oplus C_4 \oplus C_6 \end{aligned}$$

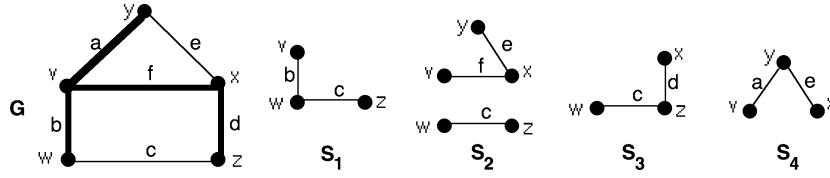


**Figure 4.6.3** Fundamental system  $\{C_3, C_4, C_6\}$  is a basis for  $W_C(G)$ .

The next theorem and its proof are analogous to Theorem 4.6.5 and its proof.

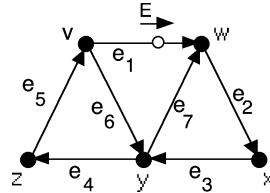
**Theorem 4.6.6.** Let  $T$  be a spanning tree of a connected graph  $G$ . Then the fundamental system of edge-cuts associated with  $T$  is a basis for the edge-cut space  $W_S(G)$ .  
 $\diamond$  (Exercises)

**Example 4.6.2 continued:** Figure 4.6.4 shows a spanning tree and the associated fundamental system of edge-cuts  $\{S_1, S_2, S_3, S_4\}$  for the graph of Figure 4.6.2. It is easy to verify that each of the 15 non-null elements of the edge-cut space  $W_S(G)$  can be expressed as the mod 2 sum of some or all of the fundamental edge-cuts  $S_1, S_2, S_3, S_4$  (see Exercises). For instance, the edge-cut appearing on the lower right in Figure 4.6.2 is equal to  $S_1 \oplus S_2 \oplus S_3 \oplus S_4$ .



**Figure 4.6.4** Fundamental system  $\{S_1, S_2, S_3, S_4\}$  is a basis for  $W_S(G)$ .

**Application 4.6.1 Applying Ohm's and Kirchhoff's Laws:** Suppose that the graph shown in Figure 4.6.5 represents an electrical network with a given voltage  $E$  on wire  $e_1$ , oriented as shown. Also let  $R_j$  be the resistance on  $e_j$ .



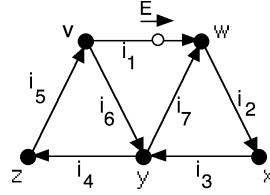
**Figure 4.6.5** An electrical network.

The problem is to determine the electric current  $i_j$  for wire  $e_j$ , using Ohm's Law, Kirchhoff's current law (KCL), and Kirchhoff's voltage law (KVL), given by:

- Ohm's Law For a current  $i$  flowing through a resistance  $r$ , the voltage drop  $v$  across the resistance satisfies  $v = ir$ .
- KCL The algebraic sum of the currents at each vertex is zero.
- KVL The algebraic sum of voltage drops around any cycle is zero.

To apply these laws, a direction must be assigned to the current in each wire. These directions are arbitrary and do not affect the final solution, in the sense that a negative value for an  $i_j$  simply means that the direction of flow is opposite to the direction assigned for  $e_j$ .

**Illustration:** Figure 4.6.6 shows an arbitrary assignment of directions for the wires in the example network.



**Figure 4.6.6** An electrical network.

The five (KCL)-equations corresponding to the five vertices are

$$\begin{aligned} -i_1 + i_5 - i_6 &= 0 \\ i_2 - i_3 &= 0 \\ i_4 - i_5 &= 0 \\ i_3 - i_4 + i_6 - i_7 &= 0 \\ i_1 - i_2 + i_7 &= 0 \end{aligned}$$

Notice that the sum of these equations is the equation  $0 = 0$ , which indicates that one of them is redundant. Furthermore, if one circuit is the sum of other circuits, then its (KVL)-equation is redundant. For instance, the (KVL)-equations for the circuits  $\langle v, w, y, v \rangle$ ,  $\langle w, x, y, w \rangle$ , and  $\langle v, w, x, y, v \rangle$  are, respectively,

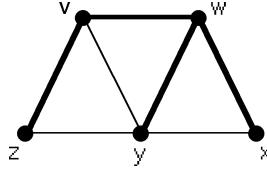
$$\begin{aligned} i_1 R_1 - i_7 R_7 - i_6 R_6 - E &= 0 \\ i_2 R_2 + i_3 R_3 + i_7 R_7 &= 0 \\ i_1 R_1 + i_2 R_2 + i_3 R_3 - i_6 R_6 - E &= 0 \end{aligned}$$

The third equation is the sum of the first two, since the third circuit is the sum of the first two circuits.

A large network is likely to have a huge number of these redundancies. The objective for an efficient solution strategy is to find a minimal set of circuits whose corresponding equations, together with the equations from Kirchhoff's circuit law, are just enough to solve for the  $i_j$ 's. Since a fundamental system of cycles is a basis for the cycle space, their corresponding equations will constitute a full set of linearly independent equations and will meet the objective.

**Illustration:** One of the spanning trees of the example network is shown in Figure 4.6.7 below. The fundamental system of cycles and their corresponding equations for this spanning tree are as follows:

$$\begin{aligned} \langle z, v, w, y, z \rangle : \quad i_5 R_5 + i_1 R_1 - i_7 R_7 + i_4 R_4 - E &= 0 \\ \langle v, w, y, v \rangle : \quad i_1 R_1 - i_7 R_7 - i_6 R_6 - E &= 0 \\ \langle w, x, y, w \rangle : \quad i_2 R_2 + i_3 R_3 + i_7 R_7 &= 0 \end{aligned}$$



**Figure 4.6.7** A spanning tree for the electrical network.

These three equations, together with any four of the five (KCL)-equations will determine the  $i_j$ 's. If, for example, the voltage  $E = 28$  and all the  $R_j$ 's are taken to be 1, then it is easy to verify that the solution for the currents is

$$i_1 = 12; i_2 = i_3 = 4; i_4 = i_5 = 4; i_6 = i_7 = -8$$

### Dimension of the Cycle Space and of the Edge-Cut Space

In light of the discussion and remark preceding Example 4.5.1 in §4.5, Theorems 4.6.5 and 4.6.6 extend easily to non-connected graphs. For such graphs, the fundamental systems are associated with a full spanning forest instead of a spanning tree. The resulting corollary establishes the dimensions of the cycle space and the edge-cut space, thereby justifying the use of the terms *cycle rank* and *edge-cut rank*.

**Corollary 4.6.7.** *Let  $G$  be a graph with  $c(G)$  components. Then the dimension of the cycle space  $W_C(G)$  is the cycle rank  $\beta(G) = |E_G| - |V_G| + c(G)$ , and the dimension of the edge-cut space  $W_S(G)$  is the edge-cut rank  $|V_G| - c(G)$ .*

**Proof:** This follows directly from Theorems 4.6.5 and 4.6.6 and the definitions of cycle rank and edge-cut rank.  $\diamond$

### Relationship Between the Cycle and Edge-Cut Spaces

The next two propositions characterize the elements of the cycle space and the edge-cut space of a graph in terms of each other. The proof of the second characterization is similar to that of the first and is left as an exercise.

**Proposition 4.6.8.** *A subgraph  $H$  of a graph  $G$  is in the cycle space  $W_C(G)$  if and only if it has an even number of edges in common with every subgraph in the edge-cut space  $W_S(G)$ .*

**Proof:** *Necessity ( $\Rightarrow$ )* Each subgraph in the cycle space is a union of edge-disjoint cycles, and each subgraph in the edge-cut space is a union of edge-disjoint edge-cuts. Thus, necessity follows from Proposition 4.5.8.

*Sufficiency ( $\Leftarrow$ )* It may be assumed without loss of generality that  $G$  is connected, since the argument that follows may be applied separately to each of the components of  $G$  if  $G$  is not connected.

Suppose  $H$  has an even number of edges in common with each subgraph in the edge-cut space of  $G$ , and let  $T$  be a spanning tree of  $G$ . Let  $e_1, e_2, \dots, e_r$  be the non-tree edges of  $H$ , and consider  $C = C_1 \oplus C_2 \oplus \dots \oplus C_r$ , where each  $C_i$  is the fundamental cycle in  $T + e_i$ ,  $i = 1, \dots, r$ . Arguing as in the proof of Theorem 4.6.5,  $H \oplus C$  has no non-tree edges. Thus, the only possible edges in  $H \oplus C$  are edges of  $T$ . So suppose  $b$  is an edge

in both  $T$  and  $H \oplus C$ . Now if  $S$  is the fundamental edge-cut associated with  $b$ , then  $b$  is the only edge in  $H \oplus C \oplus S$ . But, since  $C \in W_C(G)$ ,  $C$  has an even number of edges in common with each subgraph in the edge-cut space of  $G$ , as does  $H$ . This implies that  $H \oplus C$  has an even number of edges in common with each subgraph in the edge-cut space. In particular,  $H \oplus C \oplus S$  must have an even number of edges. This contradiction shows that  $H \oplus C$  must be the null graph, that is,  $H = C$ . Hence,  $H$  is a subgraph in the cycle space of  $G$ .  $\diamond$

**Proposition 4.6.9.** *A subgraph  $H$  of a graph  $G$  is an element of the edge-cut space of  $G$  if and only if it has an even number of edges in common with every subgraph in the cycle space of  $G$ .*  $\diamond$  (Exercises)

### Orthogonality of the Cycle and Edge-Cut Spaces

**DEFINITION:** Two vectors in the edge space  $W_E(G)$  are said to be **orthogonal** if the dot product of their characteristic vectors equals 0 in  $GF(2)$ . Thus, two subsets of edges are orthogonal if they have an even number of edges in common.

**Theorem 4.6.10.** *Given a graph  $G$ , the cycle space  $W_C(G)$  and the edge-cut space  $W_S(G)$  are orthogonal subspaces of  $W_E(G)$ .*

**Proof:** Let  $H$  be a subgraph in the cycle space, and let  $K$  be a subgraph in the edge-cut space. By either Proposition 4.6.8 or Proposition 4.6.9,  $H$  and  $K$  have an even number of edges in common and, hence, are orthogonal.  $\diamond$

**Theorem 4.6.11.** *Given a graph  $G$ , the cycle space  $W_C(G)$  and the edge-cut space  $W_S(G)$  are orthogonal complements in  $W_E(G)$  if and only if  $W_C(G) \cap W_S(G) = \emptyset$ .*

**Proof:** If  $W_C(G) \oplus W_S(G)$  denotes the direct sum of the subspaces  $W_C(G)$  and  $W_S(G)$ , then

$$\dim(W_C(G) \oplus W_S(G)) = \dim(W_C(G)) + \dim(W_S(G)) - \dim(W_C(G) \cap W_S(G))$$

By Corollary 4.6.7,  $\dim(W_C(G)) + \dim(W_S(G)) = |E_G|$ . Thus,  $\dim(W_C(G) \oplus W_S(G)) = |E_G| = \dim(W_E(G))$  if and only if  $\dim(W_C(G) \cap W_S(G)) = 0$ .  $\diamond$

### Vector Space of Vertex Subsets

This section concludes by showing that a spanning tree of a simple graph  $G$  corresponds to a basis of the column space of the incidence matrix of  $G$ .

**FROM LINEAR ALGEBRA:** The column space of a matrix  $M$  is the set of column vectors that are linear combinations of the columns of  $M$ . When the entries of  $M$  come from  $GF(2)$ , linear combinations are simply mod 2 sums. (Also see Appendix A.4.)

Let  $G$  be a simple graph with  $n$  vertices and  $m$  edges, and let  $\langle e_1, e_2, \dots, e_m \rangle$  and  $\langle v_1, v_2, \dots, v_n \rangle$  be fixed orderings of  $E_G$  and  $V_G$ , respectively.

**REVIEW FROM §2.6:** The incidence matrix  $I_G$  of  $G$  is the matrix whose  $(i, j)$ th entry is given by

$$I_G[i, j] = \begin{cases} 0, & \text{if } v_i \text{ is not an endpoint of } e_j \\ 1, & \text{otherwise} \end{cases}$$

Analogous to the edge space  $W_E(G)$ , the collection of vertex subsets of  $V_G$  under ring sum forms a vector space over  $GF(2)$ , which is called the **vertex space of  $G$**  and is denoted  $W_V(G)$ . Each element of the vertex space  $W_V(G)$  may be viewed as an  $n$ -tuple over  $GF(2)$ .

In this setting,  $I_G$  represents a linear transformation from edge space  $W_E(G)$  to vertex space  $W_V(G)$ , mapping the characteristic vectors of edge subsets to characteristic vectors of vertex subsets.

**Example 4.6.4:** Consider the graph  $G$  and its corresponding incidence matrix  $I_G$  shown in Figure 4.6.8.

$$I_G = \begin{pmatrix} & a & b & c & d & e \\ x & 1 & 0 & 0 & 1 & 1 \\ y & 1 & 1 & 0 & 0 & 0 \\ z & 0 & 1 & 1 & 0 & 1 \\ w & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

**Figure 4.6.8** A graph  $G$  and its incidence matrix  $I_G$ .

The characteristic vector of the image of the subset  $E_1 = \{a, c, e\}$  under the mapping is obtained by multiplying the characteristic vector of  $E_1$  by  $I_G$  (mod 2), as follows:

$$\begin{pmatrix} 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \end{pmatrix}$$

Thus,  $E_1$  is mapped to the vertex subset  $V_1 = \{y, w\}$ . Notice that  $V_1$  consists of the endpoints of the path formed by the edges of  $E_1$ . It is not hard to show that every open path in  $G$  is mapped to the vertex subset consisting of the path's initial and terminal vertices. For this reason,  $I_G$  is sometimes called a *boundary operator*.

**TERMINOLOGY NOTE:** Algebraists and topologists sometimes refer to the edge subsets of  $W_E(G)$  and the vertex subsets of  $W_V(G)$  as *1-chains* and *0-chains*, respectively, and denote these vector spaces  $C_1(G)$  and  $C_0(G)$ . In this context, the chains are typically represented as sums of edges or vertices. For example, the expression  $e_1 + e_2 + e_3$  would represent the edge subset  $\{e_1, e_2, e_3\}$ .

In general, the image under  $I_G$  of any edge subset is obtained by simply computing the mod 2 sum of the corresponding set of columns of  $I_G$ . For instance, in Example 4.6.4,  $I_G$  maps the edge subset  $E_2 = \{c, d, e\}$  to  $\emptyset$  (which is the zero element of the vertex space  $W_V(G)$ ), since the mod 2 sum of the third, fourth, and fifth columns of  $I_G$  is the column vector of all zeros. It is not a coincidence that  $E_2$  comprises the edges of a cycle, as the next proposition confirms.

**NOTATION:** For any edge subset  $D$  of a simple graph  $G$ , let  $C_D$  denote the corresponding set of columns of  $I_G$ .

**Proposition 4.6.12.** *Let  $D$  be the edge-set of a cycle in a simple graph  $G$ . Then the mod 2 sum of the columns in  $C_D$  equals zero.*

**Proof:** For a suitable ordering of the edges and vertices of  $G$ , a cycle of length  $k$  corresponds to the following submatrix of  $I_G$ .

$$\begin{pmatrix} 1 & 0 & \cdots & 0 & 1 \\ 1 & 1 & \ddots & \vdots & 0 \\ 0 & 1 & \ddots & 0 & \vdots \\ \vdots & \ddots & \ddots & 1 & 0 \\ 0 & \cdots & 0 & 1 & 1 \end{pmatrix} \quad \diamond$$

**Corollary 4.6.13.** A set  $D$  of edges of a simple graph  $G$  forms a cycle or a union of edge-disjoint cycles if and only if the mod 2 sum of the columns in  $C_D$  is zero.

$\diamond$  (Exercises)

Each column of  $I_G$  may be regarded as an element of the vector space of all  $n$ -tuples over  $GF(2)$ . Therefore, a subset  $S$  of the columns of  $I_G$  is *linearly independent* over  $GF(2)$  if no nonempty subset of  $S$  sums to the zero vector. Using this terminology, the following characterization of acyclic subgraphs is an immediate consequence of Corollary 4.6.13.

**Proposition 4.6.14.** An edge subset  $D$  of a simple graph  $G$  forms an acyclic subgraph of  $G$  if and only if the column set  $C_D$  is linearly independent over  $GF(2)$ .  $\diamond$

The next proposition is needed to complete the characterization of spanning trees promised earlier.

**Proposition 4.6.15.** Let  $D$  be a subset of edges of a connected, simple graph  $G$ . Then  $D$  forms (induces) a connected spanning subgraph of  $G$  if and only if column set  $C_D$  spans the column space of  $I_G$ .

**Proof:** The column set  $C_D$  spans the column space of  $I_G$  if and only if each column of  $I_G$  can be expressed as the sum of the columns in a subset of  $C_D$ . But each column of  $I_G$  corresponds to some edge  $e = xy$  of  $G$ , and that column is a sum of the columns in some subset of  $C_D$  if and only if there is a path from  $x$  to  $y$  whose edges are in  $D$ .  $\diamond$

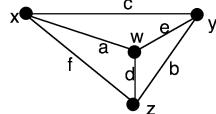
**Proposition 4.6.16.** Let  $G$  be a connected, simple graph. Then an edge subset  $D$  induces a spanning tree of  $G$  if and only if the columns corresponding to the edges of  $D$  form a basis for the column space of  $I_G$  over  $GF(2)$ .

**Proof:** This follows directly from Propositions 4.6.14 and 4.6.15.  $\diamond$

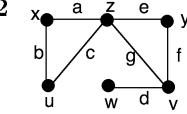
### EXERCISES for Section 4.6

In Exercises 4.6.1 through 4.6.3, find (a) the non-null elements of the cycle space  $W_C(G)$  for the given graph  $G$ ; and (b) the non-null elements of the edge-cut space  $W_S(G)$ .

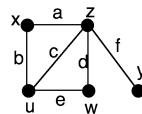
4.6.1<sup>s</sup>



4.6.2



4.6.3



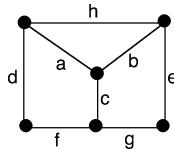
In Exercises 4.6.4 through 4.6.6, consider the spanning tree  $T$  defined by the specified edge subset of the graph  $G$  in the indicated exercise. (a) Show that the fundamental system of cycles associated with  $T$  is a basis for the cycle-space  $W_C(G)$ . (b) Show that the fundamental system of edge-cuts associated with  $T$  is a basis for the edge-cut space  $W_S(G)$ .

4.6.4<sup>s</sup> { $a, e, d$ } in Exercise 4.6.1.

4.6.5 { $a, c, g, e, d$ } in Exercise 4.6.2.

4.6.6 { $a, b, e, f$ } in Exercise 4.6.3.

4.6.7 a. Show that the collection  $\{\{a, c, d, f\}, \{b, c, e, g\}, \{a, b, h\}\}$  of edge subsets of  $E_G$  forms a basis for the cycle space  $W_C(G)$  of the graph  $G$  shown. b. Find a different basis by choosing some spanning tree and using the associated fundamental system of cycles.

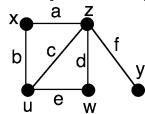


4.6.8 Express each of the 15 non-null elements of  $W_S(G)$  in Figure 4.6.2 as the mod 2 sum of some or all of the four fundamental edge-cuts in Figure 4.6.4.

4.6.9 For Application 4.6.1, reverse the directions assigned to  $i_1, i_2$ , and  $i_3$ , and show that the solution for the actual currents does not change.

In Exercises 4.6.10 through 4.6.12, verify that Corollary 4.6.13 and Proposition 4.6.16 hold for the given edge subset of the following graph.

4.6.10<sup>s</sup> { $a, b, c, d$ }.



4.6.11 { $a, b, e, f$ }.

4.6.12 { $a, b, e, d$ }.

4.6.13 Prove Proposition 4.6.1.

4.6.14 Prove Theorem 4.6.6.

4.6.15 Prove Proposition 4.6.9.

4.6.16 Prove Corollary 4.6.13.

4.6.17<sup>s</sup> Test whether the cycle and edge-cut spaces for the graph in Figure 4.6.2 are orthogonal complements of the edge space  $W_E(G)$ .

4.6.18 Express the graph of Figure 4.6.2 as the ring sum of a subgraph in the cycle space with a subgraph in the edge-cut space of  $G$ .

## 4.7 MATROIDS AND THE GREEDY ALGORITHM

**DEFINITION:** An **hereditary subset system** (or **independence system**)  $S = (E, \mathcal{I})$  is a finite set  $E$  together with a collection  $\mathcal{I}$  of subsets of  $E$  closed under *inclusion* (i.e., if  $A \in \mathcal{I}$  and  $A' \subseteq A$ , then  $A' \in \mathcal{I}$ ). The subsets in the collection  $\mathcal{I}$  are called **independent**, and any subset that is not in the collection is called **dependent**.

**Example 4.7.1:** Let  $\mathcal{V}$  be the set of vectors from a vector space and  $\mathcal{B}$  the set of subsets of linearly independent vectors. For example, if the vector space is  $\mathbb{R}^3$ , then one of the elements of  $\mathcal{B}$  is  $\{(1, 0, 0), (0, 1, 0), (0, 0, 1)\}$ . Then  $S = (\mathcal{V}, \mathcal{B})$  is an hereditary subset system since any subset of a linearly independent set of vectors is a linear independent set (possibly empty).

**Example 4.7.2:** Let  $G$  be a graph and  $\mathcal{I}$  the set of subsets of  $E_G$  whose induced subgraphs are acyclic subgraphs of  $G$ . Then  $S = (E_G, \mathcal{I})$  is an hereditary subset system.

**Remark:** For hereditary subset systems that consist of edge subsets of a graph  $G$ , no distinction will be made between the edge subset and the subgraph induced by that edge subset. In this context, if  $H$  is a subgraph, then  $e \in H$  will mean  $e \in E_H$ .

Many combinatorial optimization problems are instances of the following maximization problem for hereditary subset systems.

**DEFINITION:** The **maximum-weight problem** associated with an hereditary subset system  $S = (E, \mathcal{I})$  is the following: Let  $w$  be a *weight function* that assigns a nonnegative real number to each  $e \in E$ . Find an independent subset  $I \in \mathcal{I}$  such that  $\sum_{e \in I} w(e)$  is maximum.

**Remark:** The minimization version of the above problem is trivial, since the empty set is a minimum-weight independent subset. But since the nonnegativity of the weight function implies that a maximum independent subset is maximal, a truer analogue of the maximum-weight problem is to find a minimum-weight *maximal* independent subset.

One approach to trying to solve the maximum-weight problem associated with an hereditary subset system  $(E, \mathcal{I})$  is the following simple algorithm known as the **greedy algorithm**. To simplify the notation, let  $I + e$  denote  $I \cup \{e\}$ .

**Algorithm 4.7.1: Greedy**

*Input:* an hereditary subset system  $(E, \mathcal{I})$  and a nonnegative weight function  $w$ .  
*Output:* an independent set  $I \in \mathcal{I}$  such that  $\sum_{e \in I} w(e)$  is maximum.

```

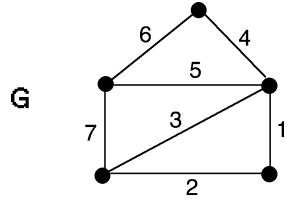
Initialize  $I := \emptyset$ .
Initialize  $A := E$ 
While  $A \neq \emptyset$ 
  Choose  $e \in A$  of largest weight.
   $A := A - e$ 
  If  $I + e \in \mathcal{I}$  then  $I := I + e$ .
Return  $I$ .

```

**Remark:** Substituting the word “largest” for “smallest” in the while-loop results in a greedy algorithm for trying to solve the minimum-weight problem.

The next two examples illustrate the greedy algorithm on two different graph optimization problems. The algorithm produces an optimal solution for the first problem but fails to do so for the second one. An examination of the underlying algebraic structure will reveal why.

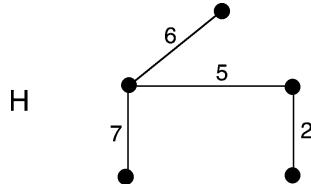
**Example 4.7.3:** Consider the problem of finding a maximum-weight spanning tree for the weighted graph  $G$  shown in Figure 4.7.1.



**Figure 4.7.1** A weighted graph.

Since a spanning tree is an acyclic subgraph, the problem may be viewed as a maximum-weight problem associated with the hereditary subset system of acyclic subgraphs. Thus, the problem is to find an element  $H$  of  $\mathcal{I}$  (i.e., an edge subset whose induced subgraph is acyclic) whose total edge-weight is maximum.

The greedy algorithm begins by choosing the edge of weight 7 and continues by choosing the edge of largest weight that does not create a cycle. It is easy to see that the resulting spanning tree, shown in Figure 4.7.2, has the maximum weight.



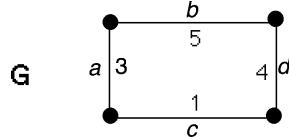
**Figure 4.7.2** The greedy algorithm produces a maximum spanning tree.

**Remark:** This version of the greedy algorithm is known as **Kruskal's algorithm**, named for J. B. Kruskal [Kr56]. Since a spanning tree is an edge-maximal acyclic subgraph, the *minimum-spanning-tree* problem is an instance of the minimum-weight problem for an hereditary subset system, and as will be seen later, Kruskal's algorithm succeeds for both spanning-tree problems.

**DEFINITION:** A **matching** in a graph is a set of edges that have no endpoints in common.

The *maximum-matching problem* is to find a matching in a weighted graph, whose total edge-weight is a maximum. The hereditary subset system that corresponds to this maximum-weight problem is  $(E_G, \mathcal{I})$ , where  $\mathcal{I}$  is the set of matchings in the graph  $G$ .

**Example 4.7.4:** Consider the maximum-matching problem for the graph  $G$  shown in Figure 4.7.3.



**Figure 4.7.3**

The greedy algorithm chooses the edge of weight 5 first and then must choose the edge of weight 1, resulting in a non-optimal solution. Thus, the greedy algorithm does *not* solve this version of the maximum-weight problem.

### For Which Problems Will the Greedy Algorithm Perform Optimally?

**DEFINITION:** An hereditary subset system  $M = (E, \mathcal{I})$  is called a **matroid** if it satisfies the following condition, which we refer to as the **augmentation property**.

*Augmentation property:* If  $I, J \in \mathcal{I}$  and  $|I| < |J|$ , then there is an element  $e \in J - I$  such that  $I + e \in \mathcal{I}$ .

The next proposition shows that the hereditary subset system associated with the maximum-spanning-tree problem is a matroid.

**Proposition 4.7.1.** *Let  $G$  be a graph. Then the hereditary subset system  $M(G) = (E_G, \mathcal{I})$ , where  $\mathcal{I}$  is the collection of edge subsets whose induced subgraphs are acyclic, is a matroid.*

**Proof:** Suppose that  $I, J \in \mathcal{I}$ , with  $|I| < |J|$ . Let  $H$  be the subgraph induced from  $I \cup J$ , and let  $F$  be a full spanning forest of  $H$  that contains  $I$  ( $F$  exists, by Proposition 4.5.2). Then  $F$  has at least as many edges as  $J$ , since  $J$  is acyclic. So there exists an edge  $e \in F - I$ . By the construction of  $H$ ,  $e \in J$ . Furthermore,  $I + e$  is a subgraph of  $F$  and, hence, is acyclic. This shows that the hereditary subset system satisfies the augmentation property, and the proof is complete.  $\diamond$

**Remark:** One of the several alternative ways of defining a matroid is to specify its minimal dependent subsets, called *cycles*, and to require them to satisfy an additional property analogous to the augmentation property (see Exercises). The cycles of the matroid  $M(G)$  are precisely the cycles in the graph  $G$  (see Exercises), which is why this matroid is called the **cycle matroid**.

**Example 4.7.5:** The hereditary subset system associated with the maximum matching problem is *not* a matroid. To see that it does not satisfy the augmentation property, consider the graph in Figure 4.7.3. The subsets  $I = \{a\}$  and  $J = \{b, c\}$  violate the augmentation property.

The final result of this chapter characterizes those problems for which the greedy algorithm produces an optimal solution for the maximum-weight problem.

**Theorem 4.7.2.** *Let  $S = (E, \mathcal{I})$  be an hereditary subset system. Then the greedy algorithm solves every instance of the maximum-weight problem associated with  $S$  if and only if  $S$  is a matroid.*

**Proof:** *Necessity ( $\Rightarrow$ )* Suppose that the augmentation property is not satisfied, and consider  $I, J \in \mathcal{I}$  with  $|I| < |J|$  such that there is no  $e \in J - I$  for which  $I + e \in \mathcal{I}$ . Since  $S$  is an hereditary subset system, it can be assumed without loss of generality that  $|J| = |I| + 1$ . Let  $|I| = p$ , and consider the following weight function on  $E$ .

$$w(e) = \begin{cases} p+2, & \text{if } e \in I; \\ p+1, & \text{if } e \in J - I; \\ 0, & \text{otherwise.} \end{cases}$$

After the greedy algorithm chooses all the elements of  $I$ , it cannot increase the total weight, since the only edges left that have nonzero weight belong to  $J$ . Thus, the greedy algorithm ends with an independent subset whose weight is  $p(p+2)$ . But the weight of subset  $J$  is at least  $(p+1)^2$ , which shows that the greedy algorithm did not find a maximum-weight independent subset.

*Sufficiency ( $\Leftarrow$ )* Let  $I = \{e_1, e_2, \dots, e_n\}$  be the independent subset obtained when the greedy algorithm is applied, and let  $J = \{f_1, f_2, \dots, f_l\}$  be any independent subset in  $\mathcal{I}$ . Also assume that the  $e_i$ 's and  $f_i$ 's are subscripted in descending order by weight. The augmentation property implies that  $l \leq n$ , since otherwise there would be an  $f_i$  that should have been added to  $I$  by the greedy algorithm but was not. Now let  $I_k = \{e_1, e_2, \dots, e_k\}$  and  $J_k = \{f_1, f_2, \dots, f_k\}$ ,  $k = 1, \dots, l$ . To complete the proof, it suffices to show that

$$\sum_{i=1}^k w(e_i) \doteq w(I_k) \geq w(J_k), \quad k = 1, \dots, l$$

Observe that  $w(I_1) \geq w(J_1)$ , since the algorithm starts by choosing a largest element. By way of induction, assume  $w(I_k) \geq w(J_k)$  for some  $1 \leq k < l$ . By the augmentation property, there is an  $x \in J_{k+1} - I_k$  such that  $I_k + x \in \mathcal{I}$ . Thus,  $w(e_{k+1}) \geq w(x)$ , since otherwise the algorithm would have chosen  $x$  instead of  $e_{k+1}$ . Also,  $w(x) \geq w(f_{k+1})$ , since the  $f_i$ 's are subscripted in descending order. Therefore,

$$w(I_{k+1}) = w(I_k) + w(e_{k+1}) \geq w(I_k) + w(f_{k+1}) \geq w(J_k) + w(f_{k+1}) = w(J_{k+1}),$$

which completes the proof of the claim. Finally,

$$w(I) = w(I_n) \geq w(I_l) \geq w(J_l) = w(J)$$

which shows that  $I$  is a maximum-weight independent subset and completes the proof.  $\diamond$

**Remark:** An immediate consequence of Proposition 4.7.1 and Theorem 4.7.2 is that Kruskal's algorithm always finds the maximum-weight spanning tree. Using an argument similar to the one given for the sufficiency part of Theorem 4.7.2, it is not hard to show that the version of Kruskal's algorithm that chooses edges of *smallest* possible weight finds the minimum-weight spanning tree (see Exercises). The algorithmic analysis and comparisons with Prim's algorithm may be found in one of the standard texts in data structures and algorithms (e.g., [AhUlHo83], [Se88]).

Using matroid theory, several results in combinatorial optimization can be unified, often resulting in simpler proofs. For those wanting to learn more about matroids, there are several excellent texts (e.g., [Ox92], [Wi96], [ThSw92], [La76], [We76]).

## EXERCISES for Section 4.7

*In Exercises 4.7.1 through 4.7.4, use Kruskal's algorithm to find a minimum-weight spanning tree of the specified graph.*

- |                    |                              |       |                              |
|--------------------|------------------------------|-------|------------------------------|
| 4.7.1 <sup>s</sup> | The graph of Exercise 4.3.1. | 4.7.2 | The graph of Exercise 4.3.2. |
| 4.7.3              | The graph of Exercise 4.3.3. | 4.7.4 | The graph of Exercise 4.3.4. |

*In Exercises 4.7.5 through 4.7.9, determine whether the given family of subsets  $(E, \mathcal{I})$  is an hereditary subset system, and justify your answer. If the family is an hereditary system, then determine whether it is a matroid.*

- 4.7.5<sup>s</sup>  $E = V_G$  for an arbitrary graph  $G$ , and  $\mathcal{I}$  is the collection of all subsets of mutually non-adjacent vertices in graph  $G$ .

- 4.7.6  $E = V_G$  for an arbitrary graph  $G$ , and  $\mathcal{I}$  is the collection of all subsets of mutually adjacent vertices in graph  $G$ .

- 4.7.7  $E = E_G$  for an arbitrary graph  $G$ , and  $\mathcal{I}$  is the collection of all subsets of mutually non-adjacent edges in graph  $G$ .

**4.7.8**  $E = E_G$  for an arbitrary graph  $G$ , and  $\mathcal{I}$  is the collection of all subsets of mutually adjacent edges in graph  $G$ .

**4.7.9<sup>s</sup>**  $E$  is any set of  $n$  elements, and  $\mathcal{I}$  is the collection of all subsets of  $k$  elements, where  $1 \leq k \leq n$ .

**4.7.10** Show that when the greedy algorithm (Algorithm 4.7.1) has chosen  $k$  elements, this subset of  $k$  elements is of maximum weight among all independent subsets of  $k$  or fewer elements.

**4.7.11** Let  $M = (E, \mathcal{I})$  be a matroid. Show that all maximal independent subsets in  $M$  have the same number of elements.

**4.7.12** Let  $G$  be a graph, and let  $\mathcal{I}$  be the collection of subsets of  $E_G$  that do *not* contain an edge-cut.

a. Show that  $H = (E_G, \mathcal{I})$  is an hereditary subset system.

b. Show that  $H$  is a matroid, by showing that it satisfies the augmentation property.

**4.7.13<sup>s</sup>** Let  $M = (E, \mathcal{I})$  be a matroid. Suppose that  $C_1$  and  $C_2$  are two different minimal *dependent* subsets in  $M$  and that both contain an element  $e \in E$ . Show that there exists a minimal dependent subset  $C_3$  such that in  $C_3 \subset C_1 \cup C_2$  and  $e \notin C_3$ .

**4.7.14** Let  $E$  be a non-empty finite set, and let  $\mathcal{C}$  be a collection of non-empty subsets of  $E$  satisfying the following two properties:

(1) If  $C \in \mathcal{C}$  and  $A$  is a proper subset of  $C$ , then  $A \notin \mathcal{C}$ ;

(2) If  $C_1$  and  $C_2$  are two different members of  $\mathcal{C}$  and  $e \in C_1 \cap C_2$ , for some  $e \in E$ , then there exists a subset  $C_3 \in \mathcal{C}$  such that  $C_3 \subset C_1 \cup C_2$  and  $e \notin C_3$ .

Prove that the subsets of  $E$  that are *not* members of  $\mathcal{C}$  are the independent subsets of a matroid.

**4.7.15** Show that the cycles of a graph  $G$  are the minimal dependent subsets of the cycle matroid  $M(G)$ .

**4.7.16<sup>s</sup>** Let  $M = (E, \mathcal{I})$  be a matroid. Prove that the “minimization” version of the greedy algorithm solves every instance of the minimum-weight problem associated with matroid  $M$ . This will confirm our claim that Kruskal’s algorithm always finds a minimum-weight spanning tree for any graph (with nonnegative edge-weights).

**4.7.17** Suppose that a set  $E$  of jobs are to be processed by a single machine. All jobs require the same processing time, and each job has a deadline. A subset of jobs is said to be **manageable** if there is a processing sequence for the jobs in the subset, such that each job is completed on time. Let  $\mathcal{I}$  be the collection of all manageable subsets of jobs. Show that  $(E, \mathcal{I})$  is a matroid.

**4.7.18** [Computer Project] a. Rewrite Algorithm 4.7.1 as Kruskal’s algorithm for finding a minimum spanning tree. b. Implement Kruskal’s algorithm, and run the program on each of the instances of the problem given in Exercise 4.3.1 through 4.3.4 from §4.3.

## 4.8 SUPPLEMENTARY EXERCISES

**4.8.1** Draw the isomorphism types of 6-vertex trees that cannot occur as spanning trees of  $K_{3,3}$ , and prove that they cannot.

**4.8.2** In how many different spanning trees of the complete graph  $K_n$  does a given edge  $e$  lie?

**4.8.3** Prove that the number of spanning trees in the wheel  $W_5$  equals 45. (Hint: partition into cases.)

**4.8.4** Draw an example of a connected, simple graph  $G$  with a spanning tree  $T$  indicated by thickened edges, such that no matter what root or local ordering is selected,  $T$  could not possibly be either the BFS-tree or the DFS-tree. Explain briefly why not.

**4.8.5** What is the maximum number of edges in a simple graph with a DFS-tree isomorphic to  $K_{1,5}$ . Prove your answer.

**4.8.6** Show that if no two edges of a connected graph  $G$  have the same weight, then the minimum spanning tree is unique.

**4.8.7** In the graph of Figure 4.8.1,

- use Prim's algorithm to find a minimum spanning tree;
- use Dijkstra's algorithm to find a shortest s-t path.

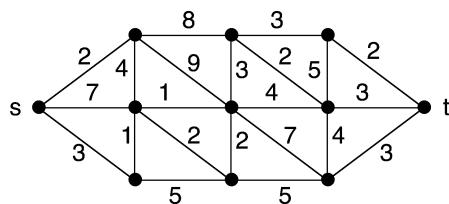


Figure 4.8.1

## GLOSSARY

**augmentation property:** see *matroid*.

**back-arc** relative to a rooted tree produced by an instance of Tree-Growing in a digraph: a non-tree arc directed from a vertex to one of its ancestors.

**Betti number of a graph:** synonym for *cycle rank*.

**bfs-tree:** shortened name for *breadth-first tree*.

**breadth-first search:** the version of Tree-Growing that selects, at each iteration, a frontier edge incident on the vertex having the smallest discovery number possible.

**breadth-first tree:** the output tree produced by a breadth-first search of a graph.

**bridge:** synonym for *cut-edge*.

**bridge component BC** of a connected  $G$  whose set of cut-edges bridges is  $B$ : a component of the subgraph  $G - B$  (§2.4).

**component of a vertex  $v$**  in a graph  $G$ , denoted  $C_G(v)$ : the subgraph of  $G$  induced on the set of vertices that are reachable from  $v$ .

**contraction of a subgraph  $H$  of a graph  $G$  to a vertex:** the replacement of  $H$  by a single vertex  $k$ . Each edge that joined a vertex  $v \in V_G - V_H$  to a vertex in  $H$  is replaced by an edge with endpoints  $v$  and  $k$ .

**cross-arc** relative to a rooted tree produced by an instance of Tree-Growing in a digraph: a non-tree arc directed from a vertex to a vertex to which it is not *related*.

—, **left-to-right**: directed from smaller discovery number to larger.

—, **right-to-left**: directed from larger discovery number to smaller.

**cross-edge** relative to a tree produced by an instance of Tree-Growing: a non-tree edge whose endpoints are not *related*.

**cut-edge** of a connected graph  $G$ : an edge  $e$  such that the edge-deletion subgraph  $G - e$  has two components.

**cycle-rank** of a graph  $G$ : the number of edges in the complement of a full spanning forest of  $G$ ; the quantity  $|E_G| - |V_G|$  plus the number of components of  $G$ .

**cycle space** of a graph  $G$ : the subspace  $W_C(G)$  consisting of the null graph  $\emptyset$ , all cycles in  $G$ , and all unions of edge-disjoint cycles of  $G$ .

**dag**: acronym for *directed acyclic graph*, i.e., it has no directed cycles.

**depth-first tree**: the output tree produced by a depth-first search of a graph.

**depth-first search**: the version of Tree-Growing that selects, at each iteration, a frontier edge incident on the vertex having the largest discovery number possible.

**dfnumber** of a vertex  $w$  for a depth-first search: the discovery number of vertex  $w$ ; denoted  $dfnumber(w)$ .

**dfs-path**: a path in the depth-first tree produced by executing a depth-first search and stopping before you backtrack for the first time.

**dfs-tree**: shortened name for *depth-first tree*.

**Dijkstra tree**: the tree produced by Dijkstra's Shortest Path Algorithm 4.3.2.

**discovery number** of a vertex for an instance of Tree-Growing: the position of that vertex in the discovery order, starting with 0 for the start vertex.

**discovery order** for an instance of Tree-Growing: the order in which the vertices of  $G$  are added (discovered) as the tree is grown.

**distance** between two vertices  $s$  and  $t$  in a weighted graph: the length of a shortest  $s$ - $t$  path.

**edge-cut** in a graph  $G$ : a set  $S$  of edges such that the edge-deletion subgraph  $G - S$  has more components than  $G$ .

**edge-cut rank** of a graph  $G$ : the quantity  $|V_G|$  minus the number of components of  $G$ ; the number of edges in a spanning forest of  $G$ .

**edge-cut space** of a graph  $G$ : the subspace  $W_S(G)$  consisting of the null graph  $\emptyset$ , all minimal edge-cuts of  $G$ , and all unions of edge-disjoint minimal edge-cuts of  $G$ .

**edge space**  $W_E(G)$  of a graph  $G$ : the vector space over  $GF(2)$  consisting of the collection of edge subsets of  $E_G$  under ring sum.

**edge-weight** of an edge in a weighted graph: the number assigned to that edge.

**eulerian graph**: a graph that has an eulerian tour.

**eulerian tour** in a graph: a closed trail that contains every edge of that graph.

**finish order** of a depth-first search: the order in which the vertices are finished.

**finished vertex** during a depth-first search: a discovered vertex whose neighbors have all been discovered.

**forward-arc** relative to a rooted tree produced by an instance of Tree-Growing in a digraph: a non-tree arc directed from a vertex to one of its descendants.

**frontier arc** relative to a rooted tree  $T$  in a digraph: an arc whose tail is in  $T$  and whose head is not in  $T$ .

**frontier edge** for a given tree  $T$ : an edge with one endpoint in  $T$  and one endpoint not in  $T$ .

**full spanning forest** of a graph  $G$ : a spanning forest consisting of a collection of trees, such that each tree is a spanning tree of a different component of  $G$ .

**fundamental cycle** of a graph  $G$  associated with a full spanning forest  $F$  and an edge  $e$  not in  $F$ : the unique cycle that is created when the edge  $e$  is added to the forest  $F$ .

**fundamental edge-cut** of a graph  $G$  associated with a full spanning forest  $F$  and an edge  $e$  in  $F$ : the unique partition-cut  $\langle V_1, V_2 \rangle$  of  $G$ , where  $V_1$  and  $V_2$  are the vertex-sets of the two components of the subgraph  $F - e$ .

**fundamental system of cycles** of a graph  $G$  associated with a full spanning forest  $F$ : the collection of fundamental cycles of the graph  $G$  that result from each addition of an edge to the spanning forest  $F$ .

**fundamental system of edge-cuts** of a graph  $G$  associated with a full spanning forest  $F$ : the collection of fundamental edge-cuts of the graph  $G$  that result from each removal of an edge from the spanning forest  $F$ .

**greedy algorithm**: an algorithm based on shortsighted greed; see Algorithm 4.7.1.

**hereditary subset system**  $S = (E, \mathcal{I})$ : a finite set  $E$  together with a collection  $\mathcal{I}$  of subsets of  $E$  closed under set inclusion (i.e., if  $A \in \mathcal{I}$  and  $A' \subseteq A$ , then  $A' \in \mathcal{I}$ ).

**independent subset** of a subset system  $S = (E, \mathcal{I})$ : a subset in the collection  $\mathcal{I}$ .

**matching** in a graph: a set of edges that have no endpoints in common.

**matroid**: an hereditary subset system  $M = (E, \mathcal{I})$  that satisfies the *augmentation property*: if  $I, J \in \mathcal{I}$  and  $|I| < |J|$ , then there is an element  $e \in J - I$  such that  $I + e \in \mathcal{I}$ .

—, **cycle**: of a graph  $G$ : the collection of edge subsets whose induced subgraphs are acyclic (see Proposition 4.7.1).

**maximum-weight problem** associated with an hereditary subset system  $S = (E, \mathcal{I})$ : for a given weight function that assigns a nonnegative real number (weight) to each  $e \in E$ , the problem of finding an independent subset in the subset system  $S$  that has the largest total weight.

**nextArc**: the digraph analog of the function *nextEdge*.

—, **dfs-**: the digraph analog of *dfs-nextEdge*.

**nextEdge(G,S)** for a set  $S$  of frontier edges for a tree  $T$  in a graph  $G$ : a function that chooses and returns as its value the frontier edge in  $S$  that is to be added to tree  $T$ ; used in various versions of Tree-Growing Algorithm 4.1.1.

—, **bfs-**: selects and returns as its value the frontier edge whose tree-endpoint has the smallest discovery number. If there is more than one such edge, then *bfs-nextEdge*( $G, S$ ) selects the one determined by the default priority.

—, **dfs-**: selects and returns as its value the frontier edge whose tree-endpoint has the largest discovery number. If there is more than one such edge, then *dfs-nextEdge*( $G, S$ ) selects the one determined by the default priority.

—, **Dijkstra-**: selects and returns as its value the frontier edge whose non-tree

endpoint is closest to the start vertex  $s$ . If there is more than one such edge, then  $Dijkstra\text{-}nextEdge(G, S)$  selects the one determined by the default priority.

—, **Prim-**: selects and returns as its value the frontier edge with smallest edge-weight. If there is more than one such edge, then  $Prim\text{-}nextEdge(G, S)$  selects the one determined by the default priority.

**non-tree edge**: see *tree edge, non-*.

**P-value**  $P(e)$  of a frontier edge  $e$  with tree endpoint  $x$  for a partial Dijkstra tree: the quantity given by  $P(e) = dist[x] + w(e)$ .

**partition-cut**  $\langle X_1, X_2 \rangle$  of a graph  $G$ : the set of all edges of  $G$  having one endpoint in  $X_1$  and the other endpoint in  $X_2$ , where  $X_1$  and  $X_2$  form a partition of  $V_G$ .

**Prim tree**: the tree produced by Prim's Minimim Spanning Tree Algorithm 4.3.1.

**related vertices** in a rooted tree: two vertices such that one is a (proper) descendant of the other.

**relative complement of a subgraph**  $H$  in a graph  $G$ , denoted  $G - H$ : the edge-deletion subgraph  $G - E(H)$ .

**ring sum** of two sets  $A$  and  $B$ : the set  $(A - B) \cup (B - A)$ , denoted  $A \oplus B$ .

**sink** in a dag: a vertex with outdegree 0.

**skip-edge** relative to a tree produced by an instance of Tree-Growing: a non-tree edge whose endpoints are *related*.

**spanning forest** of a graph  $G$ : an acyclic spanning subgraph of  $G$ .

—, **full**: a spanning forest consisting of a collection of trees, such that each tree is a spanning tree of a different component of  $G$ .

**spanning tree** of a (connected) graph: a spanning subgraph that is a tree.

**source** in a dag: a vertex with indegree 0.

**Steiner-tree problem** for a subset  $U$  of vertices in a connected edge-weighted graph  $G$ : the problem of finding a minimum-weight tree subgraph of  $G$  that contains all the vertices of  $U$ .

**topological sort** (or **topological order**) of an  $n$ -vertex dag  $G$ : a bijection  $ts$  from the set  $\{1, 2, \dots, n\}$  to the vertex-set  $V_G$  such that every arc  $e$  is directed from a smaller number to a larger one, i.e.,  $ts(tail(e)) < ts(head(e))$ .

**tree edge** relative to a given tree  $T$  in a graph  $G$ : an edge in tree  $T$ .

—, **non-**: an edge of  $G$  that is not in tree  $T$ .

**tree endpoint** of a frontier edge relative to a given tree  $T$ : the endpoint of that edge that is in tree  $T$ .

—, **non-**: the endpoint of that edge that is not in tree  $T$ .

**updateFrontier( $G, S$ )** after a frontier edge is added to the current tree: the procedure that removes from  $S$  those edges that are no longer frontier edges and adds to  $S$  those that have become frontier edges; used in various versions of Tree-Growing Algorithm 4.1.1.

**vertex space**  $W_V(G)$  of a graph  $G$ : the vector space over  $GF(2)$  consisting of the collection of vertex subsets of  $V_G$  under ring sum.

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

# Chapter 5

---

## CONNECTIVITY

- 
- 5.1 Vertex- and Edge-Connectivity**
  - 5.2 Constructing Reliable Networks**
  - 5.3 Max-Min Duality and Menger's Theorems**
  - 5.4 Block Decompositions**
- 

### INTRODUCTION

Some connected graphs are “more connected” than others. For instance, some connected graphs can be disconnected by the removal of a single vertex or a single edge, whereas others remain connected unless more vertices or more edges are removed. Two numerical parameters, *vertex-connectivity* and *edge-connectivity*, are useful in measuring a graph’s connectedness.

Determining the number of edges or vertices that must be removed to disconnect a given connected graph applies directly to analyzing the vulnerability of existing or proposed telecommunications networks, road systems, and other networks. Intuitively, a network’s vulnerability should be closely related also to the number of alternative paths between each pair of nodes. There is a rich body of mathematical results concerning this relationship, many of which are variations of a classical result of Menger, and some of these extend well beyond graph theory.

## 5.1 VERTEX- AND EDGE-CONNECTIVITY

**TERMINOLOGY:** Let  $S$  be a subset of vertices or edges in a connected graph  $G$ . The removal of  $S$  is said to **disconnect**  $G$  if the deletion subgraph  $G - S$  is not connected.

REVIEW FROM §2.4:

- A **vertex-cut** in a graph  $G$  is a vertex-set  $U$  such that  $G - U$  has more components than  $G$ .
- A **cut-vertex** (or **cutpoint**) is a vertex-cut consisting of a single vertex.
- An **edge-cut** in a graph  $G$  is a set of edges  $D$  such that  $G - D$  has more components than  $G$ .
- A **cut-edge** (or **bridge**) is an edge-cut consisting of a single edge.
- An edge is a cut-edge if and only if it is not a cycle-edge.

**DEFINITION:** The **vertex-connectivity** of a connected graph  $G$ , denoted  $\kappa_v(G)$ , is the minimum number of vertices whose removal can either disconnect  $G$  or reduce it to a 1-vertex graph.

Thus, if  $G$  has at least one pair of non-adjacent vertices, then  $\kappa_v(G)$  is the size of a smallest vertex-cut.

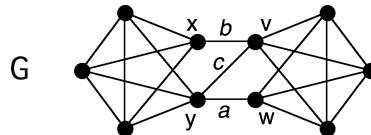
**DEFINITION:** A graph  $G$  is  **$k$ -connected** if  $G$  is connected and  $\kappa_v(G) \geq k$ . If  $G$  has non-adjacent vertices, then  $G$  is  $k$ -connected if every vertex-cut has at least  $k$  vertices.

**DEFINITION:** The **edge-connectivity** of a connected graph  $G$ , denoted  $\kappa_e(G)$ , is the minimum number of edges whose removal can disconnect  $G$ .

Thus, if  $G$  is a connected graph, the edge-connectivity  $\kappa_e(G)$  is the size of a smallest edge-cut.

**DEFINITION:** A graph  $G$  is  **$k$ -edge-connected** if  $G$  is connected and every edge-cut has at least  $k$  edges (i.e.,  $\kappa_e(G) \geq k$ ).

**Example 5.1.1:** In the graph  $G$  shown in Figure 5.1.1, the vertex set  $\{x, y\}$  is one of three different 2-element vertex-cuts, and it is easy to see that there is no cut-vertex. Thus,  $\kappa_v(G) = 2$ . The edge set  $\{a, b, c\}$  is the unique 3-element edge-cut of graph  $G$ , and there is no edge-cut with fewer than three edges. Therefore,  $\kappa_e(G) = 3$ .



**Figure 5.1.1** A graph  $G$  with  $\kappa_v(G) = 2$  and  $\kappa_e(G) = 3$ .

**Application 5.1.1 Network Survivability:** The connectivity measures  $\kappa_v$  and  $\kappa_e$  are used in a quantified model of **network survivability**, which is the capacity of a network to retain connections among its nodes after some edges or nodes are removed.

**Remark:** Since neither the vertex-connectivity nor the edge-connectivity of a graph is affected by the existence or absence of self-loops, we will assume that all graphs under consideration throughout this chapter are loopless, unless otherwise specified.

**Proposition 5.1.1.** *Let  $G$  be a graph. Then the edge-connectivity  $\kappa_e(G)$  is less than or equal to the minimum degree  $\delta_{\min}(G)$ .*

**Proof:** Let  $v$  be a vertex of graph  $G$ , with degree  $k = \delta_{\min}(G)$ . Then the deletion of the  $k$  edges that are incident on vertex  $v$  separates  $v$  from the other vertices of  $G$ .  $\diamond$

REVIEW FROM §4.5: A **partition-cut**  $\langle X_1, X_2 \rangle$  is an edge-cut each of whose edges has one endpoint in each of the vertex bipartition sets  $X_1$  and  $X_2$ .

The following proposition characterizes the edge-connectivity of a graph in terms of the size of its partition-cuts. The digraph version of this result is closely related to the network flow properties established in Chapter 13.

**Proposition 5.1.2.** *A graph  $G$  is  $k$ -edge-connected if and only if every partition-cut contains at least  $k$  edges.*

**Proof:** ( $\Rightarrow$ ) Suppose that graph  $G$  is  $k$ -edge-connected. Then every partition-cut of  $G$  has at least  $k$  edges, since a partition-cut is an edge-cut.

( $\Leftarrow$ ) Suppose that every partition-cut contains at least  $k$  edges. By Proposition 4.5.4, every minimal edge-cut is a partition-cut. Thus, every edge-cut contains at least  $k$  edges.  $\diamond$

### Relationship Between Vertex- and Edge-Connectivity

The next few results concern the relationship between vertex-connectivity and edge-connectivity. They lead to a characterization of 2-connected graphs, first proved by Hassler Whitney in 1932.

**Proposition 5.1.3.** *Let  $e$  be any edge of a  $k$ -connected graph  $G$ , for  $k \geq 3$ . Then the edge-deletion subgraph  $G - e$  is  $(k - 1)$ -connected.*

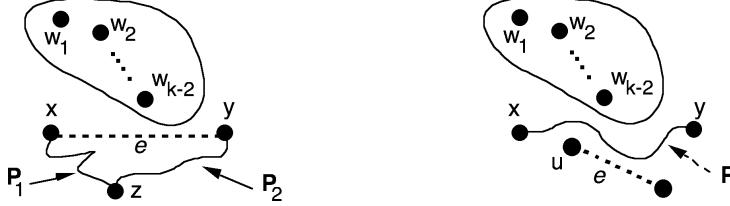
**Proof:** Let  $W = \{w_1, w_2, \dots, w_{k-2}\}$  be any set of  $k - 2$  vertices in  $G - e$ , and let  $x$  and  $y$  be any two different vertices in  $(G - e) - W$ . It suffices to show the existence of an  $x-y$  walk in  $(G - e) - W$ .

First, suppose that at least one of the endpoints of edge  $e$  is contained in set  $W$ . Since the vertex-deletion subgraph  $G - W$  is connected (in fact, 2-connected), there is an  $x-y$  path in  $G - W$ . This path cannot contain edge  $e$  and, hence, it is an  $x-y$  path in the subgraph  $(G - e) - W$ . Next, suppose that neither endpoint of edge  $e$  is in set  $W$ . Then there are two cases to consider.

*Case 1:* Vertices  $x$  and  $y$  are the endpoints of edge  $e$ . Graph  $G$  has at least  $k + 1$  vertices (since  $G$  is  $k$ -connected). So there exists some vertex  $z \in G - \{w_1, w_2, \dots, w_{k-2}, x, y\}$ . Since graph  $G$  is  $k$ -connected, there exists an  $x-z$  path  $P_1$  in the vertex-deletion subgraph  $G - \{w_1, w_2, \dots, w_{k-2}, y\}$  and a  $z-y$  path  $P_2$  in the subgraph  $G - \{w_1, w_2, \dots, w_{k-2}, x\}$  (shown on the left in Figure 5.1.2). Neither of these paths contains edge  $e$ , and, therefore, their concatenation is an  $x-y$  walk in the subgraph  $G - \{w_1, w_2, \dots, w_{k-2}\}$ .

*Case 2:* At least one of the vertices  $x$  and  $y$ , say  $x$ , is not an endpoint of edge  $e$ . Let  $u$  be an endpoint of edge  $e$  that is different from vertex  $x$ . Since graph  $G$  is  $k$ -connected,

the subgraph  $G - \{w_1, w_2, \dots, w_{k-2}, u\}$  is connected. Hence, there is an  $x-y$  path  $P$  in  $G - \{w_1, w_2, \dots, w_{k-2}, u\}$  (shown on the right in Figure 5.1.2). It follows that  $P$  is an  $x-y$  path in  $G - \{w_1, w_2, \dots, w_{k-2}\}$  that does not contain vertex  $u$  and, hence, excludes edge  $e$  (even if  $P$  contains the other endpoint of  $e$ , which it could). Therefore,  $P$  is an  $x-y$  path in  $(G - e) - \{w_1, w_2, \dots, w_{k-2}\}$ .  $\diamond$



**Figure 5.1.2** The existence of an  $x-y$  walk in  $(G - e) - \{w_1, w_2, \dots, w_{k-2}\}$ .

**Corollary 5.1.4.** Let  $G$  be a  $k$ -connected graph, and let  $D$  be any set of  $m$  edges of  $G$ , for  $m \leq k - 1$ . Then the edge-deletion subgraph  $G - D$  is  $(k - m)$ -connected.

**Proof:** The result follows by the iterative application of Proposition 5.1.3.  $\diamond$

**Corollary 5.1.5.** Let  $G$  be a connected graph. Then  $\kappa_e(G) \geq \kappa_v(G)$ .

**Proof:** Let  $k = \kappa_v(G)$ , and let  $S$  be any set of  $k - 1$  edges in graph  $G$ . Since  $G$  is  $k$ -connected, the graph  $G - S$  is 1-connected, by Corollary 5.1.4. Thus, edge subset  $S$  is not an edge-cut of graph  $G$ , which implies that  $\kappa_e(G) \geq k$ .  $\diamond$

**Corollary 5.1.6.** Let  $G$  be a connected graph. Then  $\kappa_v(G) \leq \kappa_e(G) \leq \delta_{\min}(G)$ .

**Proof:** The assertion simply combines Proposition 5.1.1 and Corollary 5.1.5.  $\diamond$

### Internally Disjoint Paths and Vertex-Connectivity: Whitney's Theorem

A communications network is said to be *fault-tolerant* if it has at least two alternative paths between each pair of vertices. This notion actually characterizes 2-connected graphs, as the next theorem demonstrates. The theorem was proved by Hassler Whitney in 1932 and is a prelude to his more general result for  $k$ -connected graphs, which appears in §5.3.

**TERMINOLOGY:** A vertex of a path  $P$  is an **internal vertex** of  $P$  if it is neither the initial nor the final vertex of that path.

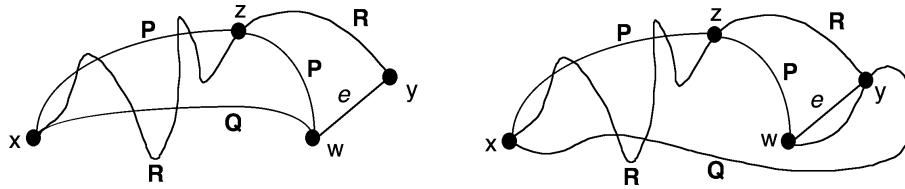
**DEFINITION:** Let  $u$  and  $v$  be two vertices in a graph  $G$ . A collection of  $u-v$  paths in  $G$  is said to be **internally disjoint** if no two paths in the collection have an internal vertex in common.

**Theorem 5.1.7 [Whitney's 2-Connected Characterization].** Let  $G$  be a connected graph with three or more vertices. Then  $G$  is 2-connected if and only if for each pair of vertices in  $G$ , there are two internally disjoint paths between them.

**Proof:** ( $\Leftarrow$ ) Arguing by contrapositive, suppose that graph  $G$  is not 2-connected. Then let  $v$  be a cut-vertex of  $G$ . Since  $G - v$  is not connected, there must be two vertices  $x$  and  $y$  such that there is no  $x-y$  path in  $G - v$ . It follows that  $v$  is an internal vertex of every  $x-y$  path in  $G$ .

( $\Rightarrow$ ) Suppose that graph  $G$  is 2-connected, and let  $x$  and  $y$  be any two vertices in  $G$ . We use induction on the distance  $d(x, y)$  to prove that there are at least two vertex-disjoint  $x$ - $y$  paths in  $G$ . If there is an edge  $e$  joining vertices  $x$  and  $y$ , (i.e.,  $d(x, y) = 1$ ), then the edge-deletion subgraph  $G - e$  is connected, by Corollary 5.1.4. Thus, there is an  $x$ - $y$  path  $P$  in  $G - e$ . It follows that path  $P$  and edge  $e$  are two internally disjoint  $x$ - $y$  paths in  $G$ .

Next, assume for some  $k \geq 2$  that the assertion holds for every pair of vertices whose distance apart is less than  $k$ . Let  $x$  and  $y$  be vertices such that distance  $d(x, y) = k$ , and consider an  $x$ - $y$  path of length  $k$ . Let  $w$  be the vertex that immediately precedes vertex  $y$  on this path, and let  $e$  be the edge between vertices  $w$  and  $y$ . Since  $d(x, w) < k$ , the induction hypothesis implies that there are two internally disjoint  $x$ - $w$  paths in  $G$ , say  $P$  and  $Q$ . Also, since  $G$  is 2-connected, there exists an  $x$ - $y$  path  $R$  in  $G$  that avoids vertex  $w$ . This is illustrated in Figure 5.1.3 for the two possibilities for path  $Q$ : either it contains vertex  $y$  (as shown on the right), or it does not (as on the left).



**Figure 5.1.3**

Let  $z$  be the last vertex on path  $R$  that precedes vertex  $y$  and is also on one of the paths  $P$  or  $Q$  ( $z$  might be vertex  $x$ ). Assume, without loss of generality, that  $z$  is on path  $P$ . Then  $G$  has two internally disjoint  $x$ - $y$  paths. One of these paths is the concatenation of the subpath of  $P$  from  $x$  to  $z$  with the subpath of  $R$  from  $z$  to  $y$ . If vertex  $y$  is not on path  $Q$ , then a second  $x$ - $y$  path, internally disjoint from the first one, is the concatenation of path  $Q$  with the edge joining vertex  $w$  to vertex  $y$ . If  $y$  is on path  $Q$ , then the subpath of  $Q$  from  $x$  to  $y$  can be used as the second path.  $\diamond$

**Corollary 5.1.8.** *Let  $G$  be a graph with at least three vertices. Then  $G$  is 2-connected if and only if any two vertices of  $G$  lie on a common cycle.*

**Proof:** This follows from Theorem 5.1.7, since two vertices  $x$  and  $y$  lie on a common cycle if and only if there are two internally disjoint  $x$ - $y$  paths.  $\diamond$

**Remark:** Theorem 5.1.7 is a prelude to Whitney's more general result for  $k$ -connected graphs, which appears in §5.3. Corollary 5.1.8 is used in Chapter 7 in the proof of Kuratowski's characterization of graph planarity.

The following theorem extends the list of characterizations of 2-connected graphs. Its proof uses reasoning similar to that used in the proof of the last two results (see Exercises).

**Theorem 5.1.9 [Characterization of 2-Connected Graphs].** Let  $G$  be a connected graph with at least three vertices. Then the following statements are equivalent.

1. Graph  $G$  is 2-connected.
2. For any two vertices of  $G$ , there is a cycle containing both.
3. For any vertex and any edge of  $G$ , there is a cycle containing both.
4. For any two edges of  $G$ , there is a cycle containing both.
5. For any two vertices and one edge of  $G$ , there is a path containing all three.
6. For any three distinct vertices of  $G$ , there is a path containing all three.
7. For any three distinct vertices of  $G$ , there is a path containing any two of them which does not contain the third.  $\diamond$

### EXERCISES for Section 5.1

5.1.1<sup>s</sup> Find the other two 2-element vertex-cuts in the graph of Example 5.1.1.

In Exercises 5.1.2 through 5.1.5, either draw a graph meeting the specifications or explain why no such graph exists.

5.1.2 A 6-vertex graph  $G$  such that  $\kappa_v(G) = 2$  and  $\kappa_e(G) = 2$ .

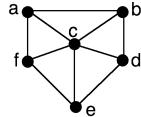
5.1.3 A connected graph with 11 vertices and 10 edges and no cut-vertices.

5.1.4 A 3-connected graph with exactly one bridge.

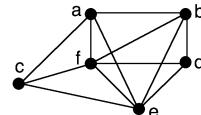
5.1.5<sup>s</sup> A 2-connected 8-vertex graph with exactly two bridges.

In Exercises 5.1.6 through 5.1.11, determine the vertex- and edge-connectivity of the given graph.

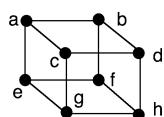
5.1.6



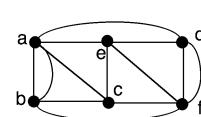
5.1.7



5.1.8



5.1.9<sup>s</sup>



5.1.10 Complete bipartite graph  $K_{4,7}$ .

5.1.11 Cube graph  $Q_4$ .

5.1.12 Determine the vertex- and edge-connectivity of the complete bipartite graph  $K_{m,n}$ .

5.1.13<sup>s</sup> Determine the vertex-connectivity and edge-connectivity of the Petersen graph (§1.2), and justify your answer. (Hint: Use the graph's symmetry to reduce the number of cases to consider.)

In Exercises 5.1.14 through 5.1.17, give an example of a graph  $G$  satisfying the given conditions.

5.1.14  $\kappa_v(G) = \kappa_e(G) = \delta_{\min}(G)$

5.1.15  $\kappa_v(G) = \kappa_e(G) < \delta_{\min}(G)$

5.1.16  $\kappa_v(G) < \kappa_e(G) = \delta_{\min}(G)$

5.1.17<sup>s</sup>  $\kappa_v(G) < \kappa_e(G) < \delta_{\min}(G)$

**5.1.18** Let  $v_1, v_2, \dots, v_k$  be  $k$  distinct vertices of a  $k$ -connected graph  $G$ , and let  $G^w$  be the graph formed from  $G$  by joining a new vertex  $w$  to each of the  $v_i$ 's. Show that  $\kappa_v(G^w) = k$ .

**5.1.19** Let  $G$  be a  $k$ -connected graph, and let  $v$  be a vertex not in  $G$ . Prove that the suspension  $H = G + v$  (§2.4) is  $(k+1)$ -connected.

**5.1.20** Prove that there exists no 3-connected simple graph with exactly seven edges.

**5.1.21<sup>s</sup>** Let  $a$ ,  $b$ , and  $c$  be positive integers with  $a \leq b \leq c$ . Show that there exists a graph  $G$  with  $\kappa_v(G) = a$ ,  $\kappa_e(G) = b$ ,  $\delta_{\min}(G) = c$ .

DEFINITION: A **unicyclic** graph is a connected graph with exactly one cycle.

**5.1.22** Show that the edge-connectivity of a unicyclic graph is no greater than 2.

**5.1.23<sup>s</sup>** Characterize those unicyclic graphs whose vertex-connectivity equals 2.

**5.1.24** Prove the characterization of 2-connected graphs given by Theorem 5.1.0.

**5.1.25** Prove that if  $G$  is a connected graph, then  $\kappa_v(G) = 1 + \min_{v \in V} \{\kappa_v(G - v)\}$ .

**5.1.26<sup>s</sup>** Find a lower bound on the number of vertices in a  $k$ -connected graph with diameter  $d$  (§1.4) and a graph that achieves that lower bound (thereby showing that the lower bound is *sharp*).

## 5.2 CONSTRUCTING RELIABLE NETWORKS

In this section we examine methods for constructing graphs with a prescribed vertex-connectivity. In light of earlier remarks, these graphs amount to blueprints for reliable networks.

### Whitney's Synthesis of 2-Connected Graphs and 2-Edge-Connected Graphs

DEFINITION: A **path addition** to a graph  $G$  is the addition to  $G$  of a path between two existing vertices of  $G$ , such that the edges and internal vertices of the path are not in  $G$ . A **cycle addition** is the addition to  $G$  of a cycle that has exactly one vertex in common with  $G$ .

DEFINITION: A **Whitney-Robbins synthesis** of a graph  $G$  from a graph  $H$  is a sequence of graphs,  $G_0, G_1, \dots, G_l$ , where  $G_0 = H$ ,  $G_l = G$ , and  $G_i$  is the result of a path or cycle addition to  $G_{i-1}$ , for  $i = 1, \dots, l$ . If each  $G_i$  is the result of a path addition *only*, then the sequence is called a **Whitney synthesis**.

**Example 5.2.1:** Figure 5.2.1 shows a 4-step Whitney synthesis of the cube graph  $Q_3$ , starting from the cycle graph  $C_4$ .

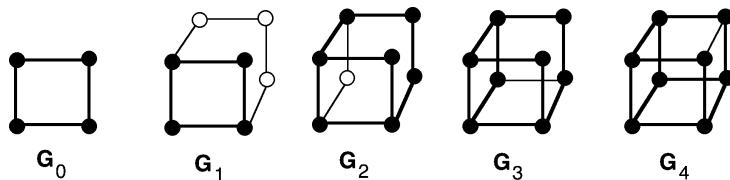


Figure 5.2.1 A Whitney synthesis of the cube graph  $Q_3$ .

**Lemma 5.2.1.** *Let  $H$  be a 2-connected graph. Then the graph  $G$  that results from a path addition to  $H$  is 2-connected.*

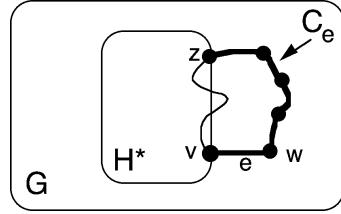
**Proof:** The property that every two vertices lie on a common cycle is preserved under path addition. Thus, by Corollary 5.1.8, graph  $G$  is 2-connected.  $\diamond$

**Theorem 5.2.2 [Whitney Synthesis Theorem].** *A graph  $G$  is 2-connected if and only if  $G$  is a cycle or a Whitney synthesis from a cycle.*

**Proof:** ( $\Leftarrow$ ) Suppose that  $C = G_0, G_1, \dots, G_l = G$  is a Whitney synthesis from a cycle  $C$ . Since a cycle is 2-connected, iterative application of Lemma 5.2.1 implies that graph  $G_i$  is 2-connected for  $i = 1, \dots, l$ . In particular,  $G = G_l$  is 2-connected.

( $\Rightarrow$ ) Suppose that  $G$  is a 2-connected graph, and let  $C$  be any cycle in  $G$ . Consider the collection  $\mathcal{H}$  of all subgraphs of  $G$  that are Whitney syntheses from cycle  $C$ . Since the collection  $\mathcal{H}$  is nonempty ( $C \in \mathcal{H}$ ), there exists a subgraph  $H^* \in \mathcal{H}$  with the maximum number of edges.

Suppose that  $H^* \neq G$ . Then, the connectedness of  $G$  implies that there exists an edge  $e = vw \in E_G - E_{H^*}$  whose endpoint  $v$  lies in  $H^*$ . Since  $G$  is 2-connected, every edge is a cycle-edge, from which it follows that there exists a cycle containing edge  $e$ . Moreover, since endpoint  $v$  is not a cut-vertex, there must be at least one such cycle, say  $C_e$ , that meets subgraph  $H^*$  at a vertex other than  $v$ . Let  $z$  be the first vertex on  $C_e$  at which the cycle returns to  $H^*$  (see Figure 5.2.2). Then the portion of  $C_e$  from  $v$  to  $z$  that includes edge  $e$  is a path addition to  $H^*$ . Thus,  $H^*$  is extendible by a path addition, contradicting the maximality of  $H^*$ . Therefore,  $H^* = G$ .  $\diamond$



**Figure 5.2.2** A path addition to  $H^*$ .

Using a similar strategy, we now establish Robbins' analogous characterization of 2-edge-connected graphs.

**Lemma 5.2.3.** *Let  $H$  be a 2-edge-connected (i.e., bridgeless) graph. Then the graph that results from a path or cycle addition to  $H$  is 2-edge-connected.*

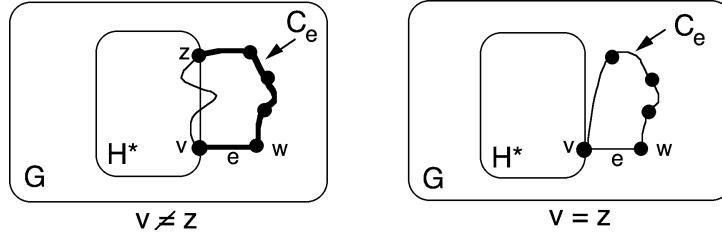
**Proof:** The property that every edge is a cycle-edge is preserved under both path and cycle addition.  $\diamond$

**Theorem 5.2.4 [Whitney-Robbins Synthesis Theorem].** *A graph  $G$  is 2-edge-connected if and only if  $G$  is a cycle or a Whitney-Robbins synthesis from a cycle.*

**Proof:** ( $\Leftarrow$ ) Suppose that  $C = G_0, G_1, \dots, G_l = G$  is a Whitney-Robbins synthesis from a cycle  $C$ . Since a cycle is 2-edge-connected, iterative application of Lemma 5.2.3 implies that  $G$  is 2-edge-connected.

( $\Rightarrow$ ) Suppose that  $G$  is a 2-edge-connected graph, and let  $C$  be any cycle in  $G$ . Among all subgraphs of  $G$  that are Whitney-Robbins syntheses from cycle  $C$ , let  $H^*$  be one with the maximum number of edges.

Suppose that  $H^* \neq G$ . As in the proof of Theorem 5.2.2, there exists an edge  $e = vw \in E_G - E_{H^*}$  whose endpoint  $v$  lies in  $H^*$ . Moreover, edge  $e$  must be part of some cycle  $C_e$  (because  $G$  is 2-edge-connected). Again, let  $z$  be the first vertex at which the cycle returns to subgraph  $H^*$ . Because  $v$  can be a cut-vertex, there are now two possibilities, as shown in Figure 5.2.3. Thus,  $H^*$  is extendible by a path addition or a cycle addition, contradicting the maximality of  $H^*$ . Therefore,  $H^* = G$ .  $\diamond$



**Figure 5.2.3** A path or cycle addition to  $H^*$ .

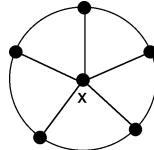
**TERMINOLOGY:** Path additions and cycle additions are often called, respectively, **open-ear** and **closed-ear additions**, since the new paths in drawings like Figure 5.2.3 are imagined to look like human ears. Moreover, the Whitney and Whitney-Robbins syntheses are called **ear decompositions**.

### Tutte's Synthesis of 3-Connected Graphs

The next theorem is a constructive characterization of 3-connected graphs, analogous to Whitney's synthesis of 2-connected graphs. The result is due to W.T. Tutte, and a proof may be found in [Tu61]. Tutte's synthesis starts with a *wheel graph*.

**REVIEW FROM §2.4:** The ***n-spoke wheel*** (*or n-wheel*)  $W_n$  is the join  $K_1 + C_n$  of a single vertex and an  $n$ -cycle. (The  $n$ -cycle forms the rim of the wheel, and the additional vertex is its hub.)

**Example 5.2.2:** The 5-spoke wheel  $W_5$  is shown in Figure 5.2.4. It has six vertices.

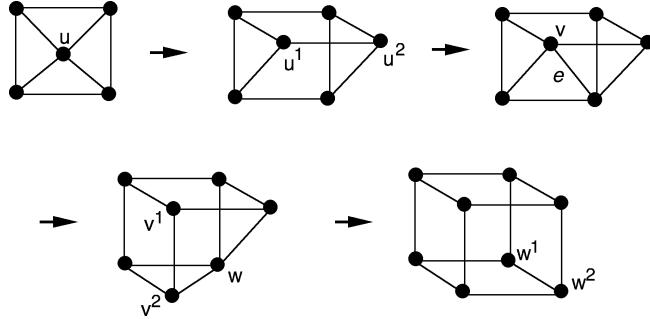


**Figure 5.2.4** The 5-spoke wheel  $W_5$ .

**Theorem 5.2.5 [Tutte Synthesis Theorem].** A graph is 3-connected if and only if it is a wheel or can be obtained from a wheel by a sequence of operations of the following two types.

1. Adding an edge between two non-adjacent vertices.
2. Replacing a vertex  $v$  with degree at least 4 by two new vertices  $v^1$  and  $v^2$ , joined by a new edge; each vertex that was adjacent to  $v$  in  $G$  is joined by an edge to exactly one of  $v^1$  and  $v^2$  so that  $\deg(v^1) \geq 3$  and  $\deg(v^2) \geq 3$ .  $\diamond$  ([Tu61])

**Example 5.2.3:** As an illustration of Tutte's synthesis, the cube graph  $Q_3$  is synthesized from the 4-spoke wheel  $W_4$  in four steps. All but the second step are operations of type 2.



**Figure 5.2.5** A 4-step Tutte synthesis of the cube graph  $Q_3$ .

**Application 5.2.1** *Construction of a Class of Reliable Networks:* In a communication network, the breakdown (deletion) of a vertex or edge from the graph representing that network is called a *fault*. Thus, the greater the vertex-connectivity and edge-connectivity, the more reliable the network. Naturally, the fewer the links used to achieve a given connectivity, the less costly the network is. This suggests the following optimization problem.

*Given positive integers  $n$  and  $k$ , with  $k < n$ , find a  $k$ -connected  $n$ -vertex graph having the smallest possible number of edges.*

Let  $h_k(n)$  denote the minimum number of edges that a  $k$ -connected graph on  $n$  vertices must have. Observe that  $h_1(n) = n - 1$ , since a spanning tree is a connected subgraph with the least number of edges. The following proposition establishes a lower bound for  $h_k(n)$ .

**DEFINITION:** For any real number  $x$ , the **floor** of  $x$ , denoted  $\lfloor x \rfloor$ , is the greatest integer less than or equal to  $x$ , and  $\lceil x \rceil$ , the **ceiling** of  $x$ , is the smallest integer greater than or equal to  $x$ .

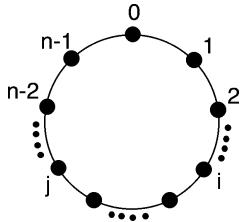
**Proposition 5.2.6.** *Let  $G$  be a  $k$ -connected graph on  $n$  vertices. Then the number of edges in  $G$  is at least  $\lceil \frac{kn}{2} \rceil$ . That is,  $h_k(n) \geq \lceil \frac{kn}{2} \rceil$ .*

**Proof:** Euler's Degree-Sum Theorem (§1.1) implies that  $2|E_G| \geq n\delta_{min}(G)$ . The result now follows by Corollary 5.1.6.  $\diamond$

### Harary's Construction of an Optimal $k$ -Connected Graph

Frank Harary gave a procedure for constructing a  $k$ -connected graph  $H_{k,n}$  on  $n$  vertices that has exactly  $\lceil \frac{kn}{2} \rceil$  edges for  $k \geq 2$  ([Ha62]). The construction of the **Harary graph**  $H_{k,n}$  begins with an  $n$ -cycle graph, whose vertices are consecutively numbered  $0, 1, 2, \dots, n - 1$  clockwise around its perimeter (see Figure 5.2.6 below).

Adjacency between two vertices  $i$  and  $j$  in the graph  $H_{k,n}$  is determined by the distance between  $i$  and  $j$  along the perimeter of the  $n$ -cycle. This distance is the length of the shorter of the two  $i-j$  paths on the perimeter and, hence, is the smaller of the two values  $|j - i|$  and  $n - |j - i|$ .



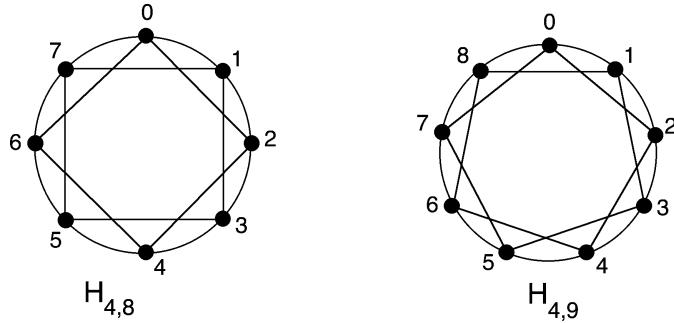
**Figure 5.2.6** Harary construction starts with an  $n$ -cycle graph.

**DEFINITION:** Let  $i$  and  $j$  be any two integers from the set  $\{0, 1, \dots, n - 1\}$ . The **mod  $n$  distance** between  $i$  and  $j$ , denoted  $|j - i|_n$ , is the smaller of the two values  $|j - i|$  and  $n - |j - i|$ .

The construction of  $H_{k,n}$  depends on the parity of  $k$  and  $n$  and falls into three cases. The construction for the first case ( $k$  even) is needed for the last two cases. Immediately following this description is an algorithm that encapsulates the construction for all three cases.

*Case 1:  $k$  even*

Let  $k = 2r$ . Vertices  $i$  and  $j$  are joined by an edge if  $|j - i|_n \leq r$ . Figure 5.2.7 shows the graphs  $H_{4,8}$  and  $H_{4,9}$ .



**Figure 5.2.7** Harary graphs  $H_{4,8}$  and  $H_{4,9}$ .

It is not hard to show that  $H_{2r,n}$  has exactly  $rn$  edges (see Exercises). Thus,  $H_{2r,n}$  has the desired number of edges, since  $rn = \frac{kn}{2} = \lceil \frac{kn}{2} \rceil$  ( $kn$  is even).

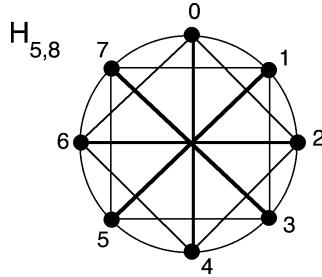
*Case 2:  $k$  odd and  $n$  even*

Let  $k = 2r+1$ . Start with graph  $H_{2r,n}$ , and add the  $\frac{n}{2}$  diameters of the original  $n$ -cycle. That is, an edge is drawn between vertices  $i$  and  $i + \frac{n}{2}$ , for  $i = 0, \dots, \frac{n}{2} - 1$ . Thus, the total number of edges in  $H_{2r+1,n}$  equals  $rn + \frac{n}{2} = \frac{(2r+1)n}{2} = \frac{kn}{2} = \lceil \frac{kn}{2} \rceil$ .

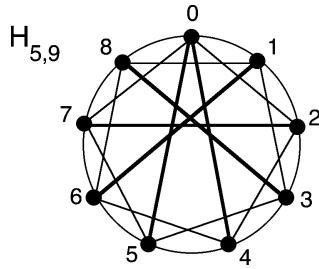
Graph  $H_{5,8}$ , shown in Figure 5.2.8 below, is obtained from  $H_{4,8}$  by adding the four diameters.

*Case 3:  $k$  and  $n$  both odd*

Let  $k = 2r+1$ . Start with graph  $H_{2r,n}$ , and add  $\frac{n+1}{2}$  quasi-diameters as follows. First, draw an edge from vertex 0 to vertex  $\frac{n-1}{2}$  and from vertex 0 to vertex  $\frac{n+1}{2}$ . Then draw an edge from vertex  $i$  to vertex  $(i + \frac{n+1}{2})$ , for  $i = 1, \dots, \frac{n-3}{2}$ . The total number of edges in  $H_{2r+1,n}$  equals  $rn + \frac{n+1}{2} = \frac{(2r+1)n+1}{2} = \lceil \frac{kn}{2} \rceil$  (since  $kn$  is odd).



**Figure 5.2.8** Harary graph  $H_{5,8}$  is  $H_{4,8}$  plus four diameters.



**Figure 5.2.9** Harary graph  $H_{5,9}$  is  $H_{4,9}$  plus five quasi-diameters.

**Remark:** An algorithm that realizes the Harary construction appears at the end of this section.

**Theorem 5.2.7.** *The Harary graph  $H_{k,n}$  is  $k$ -connected.*

**Proof:** *Case 1:  $k = 2r$*

Suppose that  $2r - 1$  vertices  $i_1, i_2, \dots, i_{2r-1}$  are deleted from graph  $H_{2r,n}$ , and let  $x$  and  $y$  be any two of the remaining vertices. It suffices to show that there is an  $x-y$  path in the vertex-deletion subgraph  $H_{2r,n} - \{i_1, i_2, \dots, i_{2r-1}\}$ .

The vertices  $x$  and  $y$  divide the perimeter of the original  $n$ -cycle into two sectors. One of these sectors contains the clockwise sequence of vertices from  $x$  to  $y$ , and the other sector contains the clockwise sequence from  $y$  to  $x$ . From one of these sequences, no more than  $r - 1$  of the vertices  $i_1, i_2, \dots, i_{2r-1}$  were deleted. Assume, without loss of generality, that this is the sector that extends clockwise from  $x$  to  $y$ , and let  $S$  be the subsequence of vertices that remain between  $x$  and  $y$  in that sector after the deletions. The gap created between two consecutive vertices in subsequence  $S$  is the largest possible when the  $r - 1$  deleted vertices are consecutively numbered, say, from  $a$  to  $a + r - 2$ , using addition modulo  $n$  (see Figure 5.2.10 below). But even in this extreme case, the resulting gap is no bigger than  $r$ . That is,  $|j - i|_n \leq r$ , for any two consecutive vertices  $i$  and  $j$  in subsequence  $S$ .

Thus, by the construction, every pair of consecutive vertices in  $S$  is joined by an edge. Therefore, subsequence  $S$  is the vertex sequence of an  $x-y$  path in  $H_{2r,n} - \{i_1, i_2, \dots, i_{2r-1}\}$ .  $\diamondsuit$  (Case 1)

*Case 2:  $k = 2r + 1$  and  $n$  even*

Suppose that  $D$  is a set of  $2r$  vertices that are deleted from graph  $H_{2r+1,n}$ , and let  $x$  and  $y$  be any two of the remaining vertices. It suffices to show that there is an  $x-y$  path in the vertex-deletion subgraph  $H_{2r+1,n} - D$ .

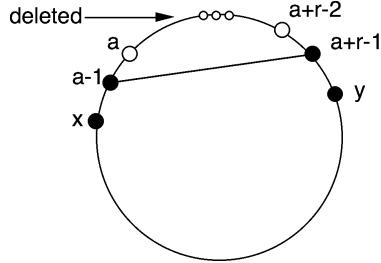


Figure 5.2.10

If one of the sectors contains fewer than  $r$  of the deleted vertices, then, as in Case 1, the subsequence of vertices left in that sector forms a path between vertices  $x$  and  $y$ . Therefore, assume both of the sectors contain exactly  $r$  of the deleted vertices. Further assume that both sets of deleted vertices are consecutively numbered, thereby creating the largest possible gap between consecutive vertices in each of the remaining subsequences.

Using addition modulo  $n$ , let  $A = \{a, a+1, \dots, a+r-1\}$  be the set of vertices that were deleted from one sector, and let  $B = \{b, b+1, \dots, b+r-1\}$  be the other set of deleted vertices. Each of the two subsets of remaining vertices is also consecutively numbered. One of these subsets starts with vertex  $a+r$  and ends with vertex  $b-1$ , and the other starts with  $b+r$  and ends with  $a-1$  (see Figure 5.2.11).

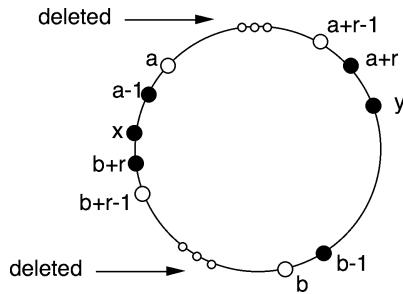


Figure 5.2.11 The four subsets of consecutively numbered vertices.

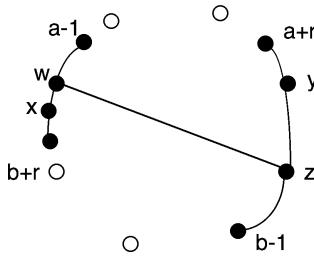
Let  $l = |(b+r) - (a-1)|_n$  and let  $w = (b+r) + \lfloor \frac{l}{2} \rfloor$ . Then  $w$  is “halfway” between vertices  $b+r$  and  $a-1$ , moving clockwise from  $b+r$  to  $a-1$ . Now let  $z = w + \frac{n}{2}$ . Then  $z$  is halfway between  $b+r$  and  $a-1$ , moving counterclockwise from  $b+r$  to  $a-1$ . But both subsets of deleted vertices are of equal size  $r$ , which implies that  $z$  is in the other subset of remaining vertices (see Figure 5.2.12). Moreover, since  $|z-w|_n = \frac{n}{2}$ , there is an edge joining  $w$  and  $z$ , by the definition of  $H_{2r+1,n}$  for Case 2.

Finally, there is a path from  $w$  to  $x$ , since both vertices are on a sector of the  $n$ -cycle left intact by the vertex deletions. By the same argument, there is a path from  $z$  to  $y$ . These two paths, together with the edge between  $w$  and  $z$ , show that there is an  $x-y$  path in the vertex-deletion subgraph  $H_{2r+1,n} - D$ .  $\diamondsuit$  (Case 2)

*Case 3:*  $k = 2r + 1$  and  $n$  odd

The argument for Case 3 is similar to the one used for Case 2.

$\diamondsuit$  (Exercises)



**Figure 5.2.12** A diameter connecting the two remaining sectors.

The following result states that the Harary graphs are also optimal with respect to achieving a given edge-connectivity with a minimum number of edges.

**Corollary 5.2.8.** *The Harary graph  $H(k, n)$  is a  $k$ -edge-connected,  $n$ -vertex graph with the fewest possible edges.*  $\diamondsuit$  (Exercises)

**Algorithm 5.2.1: Constructing an Optimal  $k$ -Connected  $n$ -Vertex Graph**

*Input:* positive integers  $k$  and  $n$ , with  $k < n$ .

*Output:* Harary graph  $H_{k,n}$  with the standard 0-based labeling.

```

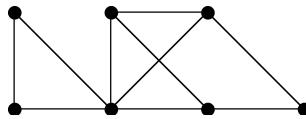
Initialize graph  $H$  to be  $n$  isolated vertices, with labels  $0, \dots, n$ .
Let  $r = \lfloor \frac{k}{2} \rfloor$ .
For  $i = 0$  to  $n - 2$ 
    For  $j = i + 1$  to  $n - 1$ 
        If  $j - i \leq r$  OR  $n + i - j \leq r$ 
            Create an edge between vertices  $i$  and  $j$ .
    [This completes the construction of  $H_{2r,n}$ .]
    If  $k$  is even
        Return graph  $H$ .
    Else
        If  $n$  is even
            For  $i = 0$  to  $\frac{n}{2} - 1$ 
                Create an edge between vertex  $i$  and vertex  $(i + \frac{n}{2})$ 
        Else
            Create an edge from vertex  $0$  to vertex  $\frac{n-1}{2}$ .
            Create an edge from vertex  $0$  to vertex  $\frac{n+1}{2}$ .
            For  $i = 1$  to  $\frac{n-3}{2}$ 
                Create an edge between vertex  $i$  and vertex  $(i + \frac{n+1}{2})$ .
    Return graph  $H$ .

```

### EXERCISES for Section 5.2

- 5.2.1<sup>s</sup> Find a Whitney synthesis of the 7-vertex wheelgraph  $W_6$ .
- 5.2.2 Find a Whitney synthesis of the complete graph  $K_5$ .
- 5.2.3 Find a 3-step Whitney synthesis of the cube graph  $Q_3$ , starting with a 4-cycle.

5.2.4 For the 2-edge-connected graph shown, find a 3-step Whitney-Robbins synthesis from a cycle.



5.2.5<sup>s</sup> Find a 1-step Tutte synthesis of the complete bipartite graph  $K_{3,3}$ .

5.2.6 Use Tutte Synthesis Theorem 5.2.5 to show that the cube graph  $Q_3$  is 3-connected.

5.2.7<sup>s</sup> Use Tutte Synthesis Theorem 5.2.5 to show that the Harary graph  $H_{3,n}$  is 3-connected for all  $n \geq 4$ .

5.2.8 Show that the number of edges in the Harary graph  $H_{2r,n}$  is  $rn$ .

*In Exercises 5.2.9 through 5.2.12, use Algorithm 5.2.1 to construct the specified Harary graph.*

5.2.9  $H_{4,7}$

5.2.10  $H_{5,7}$

5.2.11<sup>s</sup>  $H_{6,8}$

5.2.12  $H_{6,9}$

5.2.13 Show that Algorithm 5.2.1 correctly handles the three cases of the Harary construction.

5.2.14<sup>s</sup> Prove Corollary 5.2.8.

5.2.15 Prove Case 3 of Theorem 5.2.7.

5.2.16 [Computer Project] Write a computer program whose inputs are two positive integers  $k$  and  $n$ , with  $k < n$ , and whose output is the adjacency matrix of the Harary graph  $H_{k,n}$  with the vertices labeled  $0, 1, \dots, n - 1$ .

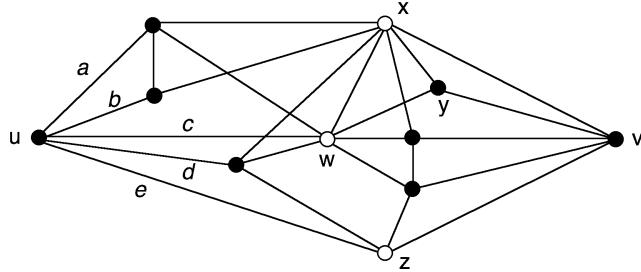
## 5.3 MAX-MIN DUALITY AND MENGER'S THEOREMS

Borrowing from operations research terminology, we consider certain *primal-dual pairs* of optimization problems that are intimately related. Usually, one of these problems involves the maximization of some objective function, while the other is a minimization problem. A feasible solution to one of the problems provides a bound for the optimal value of the other problem (this is sometimes referred to as *weak duality*), and the optimal value of one problem is equal to the optimal value of the other (*strong duality*). Menger's Theorems and their many variations epitomize this primal-dual relationship. The following terminology is used throughout this section.

**DEFINITION:** Let  $u$  and  $v$  be distinct vertices in a connected graph  $G$ . A vertex subset (or edge subset)  $S$  is  **$u$ - $v$  separating** (or **separates**  $u$  and  $v$ ), if the vertices  $u$  and  $v$  lie in different components of the deletion subgraph  $G - S$ .

Thus, a  $u$ - $v$  separating vertex set is a vertex-cut, and a  $u$ - $v$  separating edge set is an edge-cut. When the context is clear, the term  **$u$ - $v$  separating set** will refer either to a  $u$ - $v$  separating vertex set or to a  $u$ - $v$  separating edge set.

**Example 5.3.1:** For the graph in Figure 5.3.1, the vertex-cut  $\{x, w, z\}$  is a  $u-v$  separating set of vertices of minimum size, and the edge-cut  $\{a, b, c, d, e\}$  is a  $u-v$  separating set of edges of minimum size. Notice that a minimize-size  $u-v$  separating set of edges (vertices) need not be a minimum-size edge-cut (vertex-cut). For instance, the set  $\{a, b, c, d, e\}$  is not a minimum-size edge-cut, because the set of edges incident on the 3-valent vertex  $y$  is an edge-cut of size 3.



**Figure 5.3.1** Vertex- and edge-cuts that are  $u-v$  separating sets.

### A Primal-Dual Pair of Optimization Problems

The discussion in §5.1 suggests two different interpretations of a graph's connectivity. One interpretation is the number of vertices or edges it takes to disconnect the graph, and the other is the number of alternative paths joining any two given vertices of the graph. Corresponding to these two perspectives are the following two optimization problems for two non-adjacent vertices  $u$  and  $v$  of a connected graph  $G$ .

*Maximization Problem:* Determine the maximum number of internally disjoint  $u-v$  paths in graph  $G$ .

*Minimization Problem:* Determine the minimum number of vertices of graph  $G$  needed to separate the vertices  $u$  and  $v$ .

**Proposition 5.3.1.** (Weak Duality) Let  $u$  and  $v$  be any two non-adjacent vertices of a connected graph  $G$ . Let  $\mathcal{P}_{uv}$  be a collection of internally disjoint  $u-v$  paths in  $G$ , and let  $S_{uv}$  be a  $u-v$  separating set of vertices in  $G$ . Then  $|\mathcal{P}_{uv}| \leq |S_{uv}|$ .

**Proof:** Since  $S_{uv}$  is a  $u-v$  separating set, each  $u-v$  path in  $\mathcal{P}_{uv}$  must include at least one vertex of  $S_{uv}$ . Since the paths in  $\mathcal{P}_{uv}$  are internally disjoint, no two of them can include the same vertex. Thus, the number of internally disjoint  $u-v$  paths in  $G$  is at most  $|S_{uv}|$  (by the pigeonhole principle).  $\diamond$

**Corollary 5.3.2.** Let  $u$  and  $v$  be any two non-adjacent vertices of a connected graph  $G$ . Then the maximum number of internally disjoint  $u-v$  paths in  $G$  is less than or equal to the minimum size of a  $u-v$  separating set of vertices in  $G$ .  $\diamond$

The main result of this section is Menger's Theorem, which states that these two quantities are in fact equal. But even the weak duality result by itself provides *certificates of optimality*, as the following corollary shows. It follows directly from Proposition 5.3.1.

**Corollary 5.3.3 [Certificate of Optimality].** Let  $u$  and  $v$  be any two non-adjacent vertices of a connected graph  $G$ . Suppose that  $\mathcal{P}_{uv}$  is a collection of internally disjoint  $u$ - $v$  paths in  $G$ , and that  $S_{uv}$  is a  $u$ - $v$  separating set of vertices in  $G$ , such that  $|\mathcal{P}_{uv}| = |S_{uv}|$ . Then  $\mathcal{P}_{uv}$  is a maximum-size collection of internally disjoint  $u$ - $v$  paths, and  $S_{uv}$  is a minimum-size  $u$ - $v$  separating set.  $\diamondsuit$  (Exercises)

**Example 5.3.2:** Consider the graph  $G$  shown in Figure 5.3.2. The vertex sequences  $\langle u, x, y, t, v \rangle$ ,  $\langle u, z, v \rangle$ , and  $\langle u, r, s, v \rangle$  represent a collection  $\mathcal{P}$  of three internally disjoint  $u$ - $v$  paths in  $G$ , and the set  $S = \{y, s, z\}$  is a  $u$ - $v$  separating set of size 3. Therefore, by Corollary 5.3.3,  $\mathcal{P}$  is a maximum-size collection of internally disjoint  $u$ - $v$  paths, and  $S$  is a minimum-size  $u$ - $v$  separating set.

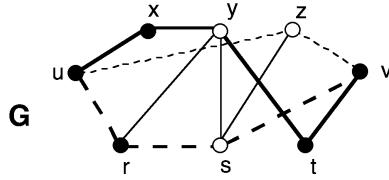


Figure 5.3.2

### Menger's Theorem

The next theorem, proved by K. Menger in 1927, establishes a *strong duality* between the two optimization problems introduced earlier. The theorem and its variations are closely related (and in many cases, equivalent) to several other *max-min* duality results for graphs and directed networks and to several results outside graph theory as well. Some of these are stated in this section, and others are established in Chapter 13, using network flows.

The proof, which involves several steps, appears at the end of this section so that we may first present a number of consequences and related results.

**Theorem 5.3.4 [Menger's Theorem].** Let  $u$  and  $v$  be distinct, non-adjacent vertices in a connected graph  $G$ . Then the maximum number of internally disjoint  $u$ - $v$  paths in  $G$  equals the minimum number of vertices needed to separate  $u$  and  $v$ .

### Variations and Consequences of Menger's Theorem

The vertex-connectivity of a graph can be expressed in terms of the *local connectivity* between a given pair of vertices, and this relationship is used in the proof of Whitney's Theorem given below.

**DEFINITION:** Let  $s$  and  $t$  be non-adjacent vertices of a connected graph  $G$ . Then the **local vertex-connectivity** between  $s$  and  $t$ , denoted  $\kappa_v(s, t)$ , is the size of a smallest  $s$ - $t$  separating vertex set in  $G$ .

**Lemma 5.3.5.** Let  $G$  be a connected graph containing at least one pair of non-adjacent vertices. Then the vertex-connectivity  $\kappa_v(G)$  is the minimum of the local vertex-connectivity  $\kappa_v(s, t)$ , taken over all pairs of non-adjacent vertices  $s$  and  $t$ .

**Proof:** Since each  $s$ - $t$  separating vertex set of the graph  $G$  is a vertex-cut, it follows that  $\kappa_v(G) \leq \kappa_v(s, t)$ , for all pairs of non-adjacent vertices  $s$  and  $t$ . Thus,  $\kappa_v(G)$  is less

than or equal to the minimum of  $\kappa_v(s, t)$  over all non-adjacent  $s$  and  $t$ . On the other hand, if  $S$  is a vertex-cut of size  $\kappa_v(G)$ , then there are at least two vertices, say  $s$  and  $t$ , that lie in different components of the vertex-deletion subgraph  $G - S$ . But then  $\kappa_v(s, t) \leq \kappa_v(G)$ , which implies that the minimum of  $\kappa_v(s, t)$  over all non-adjacent  $s$  and  $t$  is less than or equal to  $\kappa_v(G)$ .  $\diamond$

The following variation of Menger's Theorem was published by Whitney in 1932. It generalizes the characterization of 2-connected graphs given by Theorem 5.1.7.

**Theorem 5.3.6 [Whitney's  $k$ -Connected Characterization].** *A nontrivial graph  $G$  is  $k$ -connected if and only if for each pair  $u, v$  of vertices, there are at least  $k$  internally disjoint  $u-v$  paths in  $G$ .*

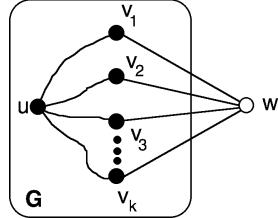
**Proof:** If every two vertices in  $G$  are adjacent, then the special case of the vertex-connectivity definition applies, and the theorem assertion is immediately true. So assume that  $G$  has at least two non-adjacent vertices.

If  $G$  is  $k$ -connected, then there are at least  $k$  vertices in any vertex-cut of  $G$ . Thus, there are at least  $k$  vertices in any  $u-v$  separating set. Theorem 5.3.4 implies that the maximum number of internally disjoint  $u-v$  paths is at least  $k$ . Hence, there are at least  $k$  internally disjoint  $u-v$  paths.

Conversely, if for each pair of vertices  $u$  and  $v$ , there are at least  $k$  internally disjoint  $u-v$  paths, then Proposition 5.3.1 implies that  $\kappa_v(u, v) \geq k$ , for each pair  $u, v$  of non-adjacent vertices. Therefore,  $\kappa_v(G) \geq k$ , by Lemma 5.3.5.  $\diamond$

**Corollary 5.3.7.** *Let  $G$  be a  $k$ -connected graph and let  $u, v_1, v_2, \dots, v_k$  be any  $k+1$  distinct vertices of  $G$ . Then there are paths  $P_i$  from  $u$  to  $v_i$ , for  $i = 1, \dots, k$ , such that the collection  $\{P_i\}$  is internally disjoint.*

**Proof:** Construct a new graph  $G^w$  from graph  $G$  by adding a new vertex  $w$  to  $G$  together with an edge joining  $w$  to  $v_i$ , for  $i = 1, \dots, k$ , as in Figure 5.3.3.



**Figure 5.3.3** The graph  $G^w$  is constructed from  $G$ .

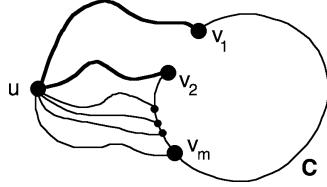
Since graph  $G$  is  $k$ -connected, it follows that graph  $G^w$  is also  $k$ -connected (by Exercise 5.1.18). By Theorem 5.3.6, there are  $k$  internally disjoint  $u-w$  paths in  $G^w$ . The  $u-v_i$  portions of these paths are  $k$  internally disjoint paths in  $G$ .  $\diamond$

The following theorem, proved by Dirac in 1960, generalizes one half of the characterization of 2-connected graphs given in Corollary 5.1.8.

**Theorem 5.3.8 [Dirac Cycle Theorem].** *Let  $G$  be a  $k$ -connected graph with at least  $k+1$  vertices, for  $k \geq 3$ , and let  $U$  be any set of  $k$  vertices in  $G$ . Then there is a cycle in  $G$  containing all the vertices in  $U$ .*

**Proof:** Let  $C$  be a cycle in  $G$  that contains the maximum possible number of vertices of set  $U$ , and suppose that  $\{v_1, \dots, v_m\}$  is the subset of vertices of  $U$  that lie on  $C$ . By

Corollary 5.1.8,  $m \geq 2$ . If there were a vertex  $u \in U$  not on cycle  $C$ , then by Corollary 5.3.7, there would exist a set of internally disjoint paths  $P_i$  from  $u$  to  $v_i$ , one for each  $i = 1, \dots, m$ . But then cycle  $C$  could be extended to include vertex  $u$ , by replacing the cycle edge between  $v_1$  and  $v_2$  by the paths  $P_1$  and  $P_2$  (see Figure 5.3.4), and this extended cycle would contradict the maximality of cycle  $C$ .  $\diamond$



**Figure 5.3.4** Extending cycle  $C$  to include vertex  $u$ .

### Analogues of Menger's Theorem

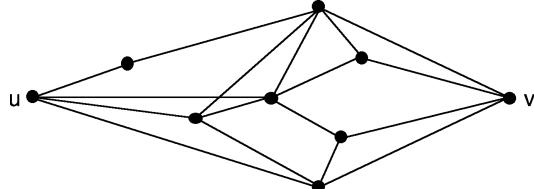
Some of the results given at the beginning of this section have the following edge analogues. Each can be proved by mimicking the proof of the corresponding vertex version. The proofs are left as exercises here but are proved in Chapter 13.

**Proposition 5.3.9 [Edge Form of Certificate of Optimality].** Let  $u$  and  $v$  be any two vertices of a connected graph  $G$ . Suppose  $\mathcal{P}_{uv}$  is a collection of edge-disjoint  $u-v$  paths in  $G$ , and  $S_{uv}$  is a  $u-v$  separating set of edges in  $G$ , such that  $|\mathcal{P}_{uv}| = |S_{uv}|$ . Then  $\mathcal{P}_{uv}$  is the largest possible collection of edge-disjoint  $u-v$  paths, and  $S_{uv}$  is the smallest possible  $u-v$  separating set. In other words, each is an optimal solution to its respective problem.  $\diamond$  (Exercises)

**Theorem 5.3.10 [Edge Form of Menger's Theorem].** Let  $u$  and  $v$  be any two distinct vertices in a graph  $G$ . Then the minimum number of edges of  $G$  needed to separate  $u$  and  $v$  equals the maximum size of a set of edge-disjoint  $u-v$  paths in  $G$ .  $\diamond$  (Exercises)

**Theorem 5.3.11 [Whitney's  $k$ -Edge-Connected Characterization].** A nontrivial graph  $G$  is  $k$ -edge-connected if and only if for every two distinct vertices  $u$  and  $v$  of  $G$ , there are at least  $k$  edge-disjoint  $u-v$  paths in  $G$ .  $\diamond$  (Exercises)

**Example 5.3.3:** For the graph shown in Figure 5.3.5, it is easy to find four edge-disjoint  $u-v$  paths and a  $u-v$  separating edge set of size 4. Thus, the maximum number of edge-disjoint  $u-v$  paths and the minimum size of a  $u-v$  separating set are both 4.



**Figure 5.3.5**

### Proof of Menger's Theorem 5.3.4

The proof of Menger's Theorem given here is illustrative of a traditional-style proof in graph theory. A different proof, based on the theory of network flows, is given in Chapter 13, where various analogues of Menger's Theorem are also proved.

**DEFINITION:** Let  $W$  be a set of vertices in a graph  $G$  and  $x$  another vertex not in  $W$ . A **strict  $x$ - $W$  path** is a path joining vertex  $x$  to a vertex in  $W$  and containing no other vertex of  $W$ . A **strict  $W$ - $x$  path** is the reverse of a strict  $x$ - $W$  path (i.e., its sequence of vertices and edges is in reverse order).

**Example 5.3.4:** Corresponding to the  $u$ - $v$  separating set  $W = \{y, s, z\}$  in the graph shown in Figure 5.3.6, the vertex sequences  $\langle u, x, y \rangle$ ,  $\langle u, r, y \rangle$ ,  $\langle u, r, s \rangle$ , and  $\langle u, z \rangle$  represent the four strict  $u$ - $W$  paths, and the three strict  $W$ - $v$  paths are given by  $\langle z, v \rangle$ ,  $\langle y, t, v \rangle$ , and  $\langle s, v \rangle$ .

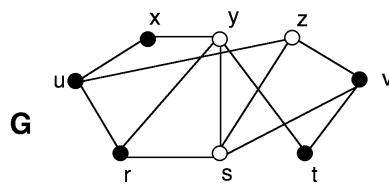


Figure 5.3.6

**Proof:** The proof uses induction on the number of edges. The smallest graph that satisfies the premises of the theorem is the path graph from  $u$  to  $v$  of length 2, and the theorem is trivially true for this graph. Assume that the theorem is true for all connected graphs having fewer than  $m$  edges, for some  $m \geq 3$ .

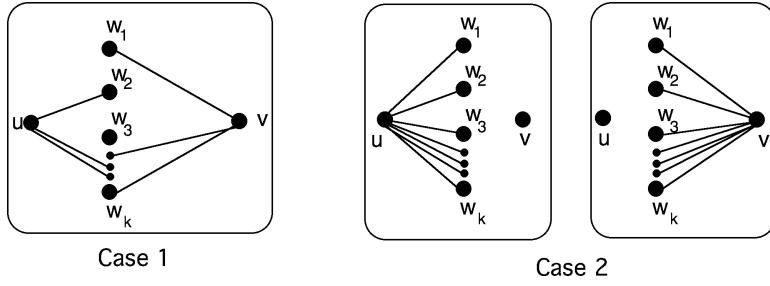
Now suppose that  $G$  is a connected graph with  $m$  edges, and let  $k$  be the minimum number of vertices needed to separate the vertices  $u$  and  $v$ . By Corollary 5.3.2, it suffices to show that there exist  $k$  internally disjoint  $u$ - $v$  paths in  $G$ . Since this is clearly true if  $k = 1$  (since  $G$  is connected), we may assume that  $k \geq 2$ .

**Assertion 5.3.4a:** If  $G$  contains a  $u$ - $v$  path of length 2, then  $G$  contains  $k$  internally disjoint  $u$ - $v$  paths.

**Proof of 5.3.4a:** Suppose that  $P = \langle u, e_1, x, e_2, v \rangle$  is a path in  $G$  of length 2. Let  $W$  be a smallest  $u$ - $v$  separating set for the vertex-deletion subgraph  $G - x$ . Since  $W \cup \{x\}$  is a  $u$ - $v$  separating set for  $G$ , the minimality of  $k$  implies that  $|W| \geq k - 1$ . By the induction hypothesis, there are at least  $k - 1$  internally disjoint  $u$ - $v$  paths in  $G - x$ . Path  $P$  is internally disjoint from any of these, and, hence, there are  $k$  internally disjoint  $u$ - $v$  paths in  $G$ .  $\diamond$  (Assertion 5.3.4a)

If there is a  $u$ - $v$  separating set that contains a vertex adjacent to *both* vertices  $u$  and  $v$ , then Assertion 5.3.4a guarantees the existence of  $k$  internally disjoint  $u$ - $v$  paths in  $G$ . The argument for  $distance(u, v) \geq 3$  is broken into two cases, according to the kinds of  $u$ - $v$  separating sets that exist in  $G$ .

In Case 1, there exists a  $u$ - $v$  separating set  $W$ , as depicted on the left in Figure 5.3.7, where neither  $u$  nor  $v$  is adjacent to every vertex of  $W$ . In Case 2, no such separating set exists. Thus, in every  $u$ - $v$  separating set for Case 2, either every vertex is adjacent to  $u$  or every vertex is adjacent to  $v$ , as shown on the right in the figure.



**Figure 5.3.7** The two cases remaining in the proof of Menger's Theorem.

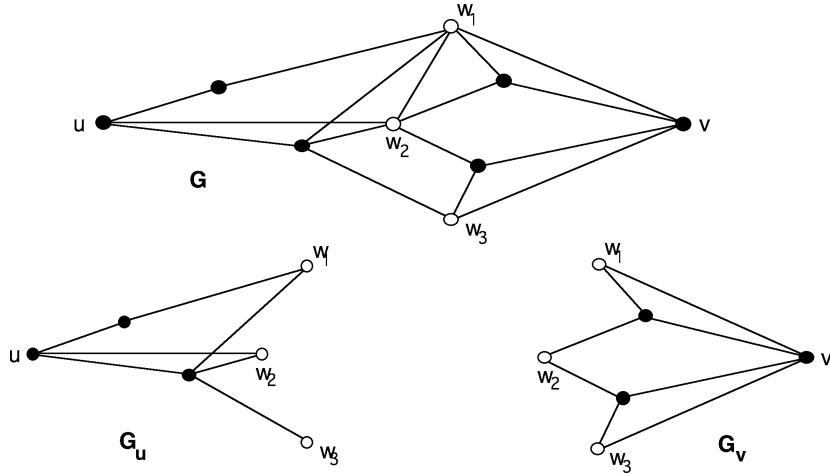
**Case 1:** There exists a  $u-v$  separating set  $W = \{w_1, w_2, \dots, w_k\}$  of vertices in  $G$  of minimum size  $k$ , such that neither  $u$  nor  $v$  is adjacent to every vertex in  $W$ .

Let  $G_u$  be the subgraph induced on the union of the edge-sets of all strict  $u-W$  paths in  $G$ , and let  $G_v$  be the subgraph induced on the union of edge-sets of all strict  $W-v$  paths (see Figure 5.3.8 below).

**Assertion 5.3.4b:** Both of the subgraphs  $G_u$  and  $G_v$  have more than  $k$  edges.

**Proof of 5.3.4b:** For each  $w_i \in W$ , there is a  $u-v$  path  $P_{w_i}$  in  $G$  on which  $w_i$  is the only vertex of  $W$  (otherwise,  $W - \{w_i\}$  would still be a  $u-v$  separating set, contradicting the minimality of  $W$ ). The  $u-w_i$  subpath of  $P_{w_i}$  is a strict  $u-W$  path that ends at  $w_i$ . Thus, the final edge of this strict  $u-W$  path is different for each  $w_i$ . Hence,  $G_u$  has at least  $k$  edges.

The only way  $G_u$  could have exactly  $k$  edges would be if each of these strict  $u-W$  paths consisted of a single edge joining  $u$  and  $w_i$ ,  $i = 1, \dots, k$ . But this is ruled out by the condition for Case 1. Therefore,  $G_u$  has more than  $k$  edges. A similar argument shows that  $G_v$  also has more than  $k$  edges.  $\diamond$  (Assertion 5.3.4b)

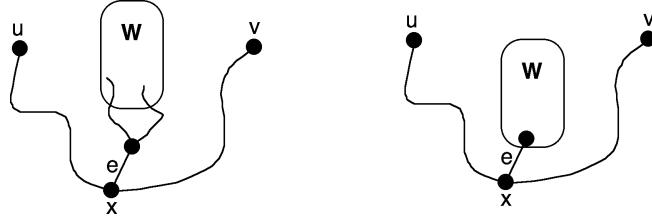


**Figure 5.3.8** An example illustrating the subgraphs  $G_u$  and  $G_v$ .

**Assertion 5.3.4c:** The subgraphs  $G_u$  and  $G_v$  have no edges in common.

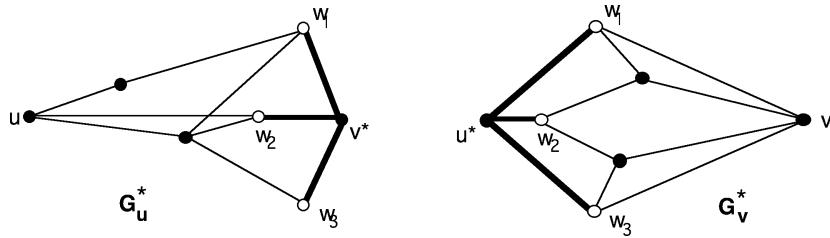
**Proof of 5.3.4c:** By way of contradiction, suppose that the subgraphs  $G_u$  and  $G_v$  have an edge  $e$  in common. By the definitions of  $G_u$  and  $G_v$ , edge  $e$  is an edge of both a strict  $u-W$  path and a strict  $W-v$  path. Hence, at least one of its endpoints, say  $x$ , is not a

vertex in the  $u-v$  separating set  $W$  (see Figure 5.3.9). But this implies the existence of a  $u-v$  path in  $G - W$ , which contradicts the definition of  $W$ .  $\diamond$  (Assertion 5.3.4c)



**Figure 5.3.9** At least one of the endpoints of edge  $e$  lies outside  $W$ .

We now define two auxiliary graphs  $G_u^*$  and  $G_v^*$ :  $G_u^*$  is obtained from  $G$  by replacing the subgraph  $G_v$  with a new vertex  $v^*$  and drawing an edge from each vertex in  $W$  to  $v^*$ , and  $G_v^*$  is obtained by replacing  $G_u$  with a new vertex  $u^*$  and drawing an edge from  $u^*$  to each vertex in  $W$  (see Figure 5.3.10).



**Figure 5.3.10** Illustration for the construction of graphs  $G_u^*$  and  $G_v^*$ .

**Assertion 5.3.4d:** Both of the auxiliary graphs  $G_u^*$  and  $G_v^*$  have fewer edges than  $G$ .

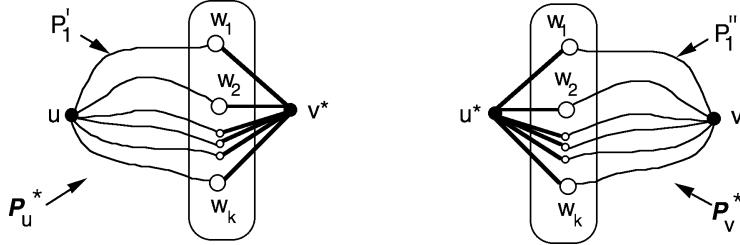
**Proof of 5.3.4d:** The following chain of inequalities shows that graph  $G_u^*$  has fewer edges than  $G$ .

$$\begin{aligned} |E_G| &\geq |E_{G_u \cup G_v}| && (\text{since } G_u \cup G_v \text{ is a subgraph of } G) \\ &= |E_{G_u}| + |E_{G_v}| && (\text{by Assertion 5.3.4c}) \\ &> |E_{G_u}| + k && (\text{by Assertion 5.3.4b}) \\ &= |E_{G_u^*}| && (\text{by the construction of } G_u^*) \end{aligned}$$

A similar argument shows that  $G_v^*$  also has fewer edges than  $G$ .  $\diamond$  (Assertion 5.3.4d)

By the construction of graphs  $G_u^*$  and  $G_v^*$ , every  $u-v^*$  separating set in graph  $G_u^*$  and every  $u^*-v$  separating set in graph  $G_v^*$  is a  $u-v$  separating set in graph  $G$ . Hence, the set  $W$  is a smallest  $u-v^*$  separating set in  $G_u^*$  and a smallest  $u^*-v$  separating set in  $G_v^*$ . Since  $G_u^*$  and  $G_v^*$  have fewer edges than  $G$ , the induction hypothesis implies the existence of two collections,  $\mathcal{P}_u^*$  and  $\mathcal{P}_v^*$ , of  $k$  internally disjoint  $u-v^*$  paths in  $G_u^*$  and  $k$  internally disjoint  $u^*-v$  paths in  $G_v^*$ , respectively (see Figure 5.3.11 below). For each  $w_i$ , one of the paths in  $\mathcal{P}_u^*$  consists of a  $u-w_i$  path  $P'_i$  in  $G$  plus the new edge from  $w_i$  to  $v^*$ , and one of the paths in  $\mathcal{P}_v^*$  consists of the new edge from  $u^*$  to  $w_i$  followed by a  $w_i-v$  path  $P''_i$  in  $G$ .

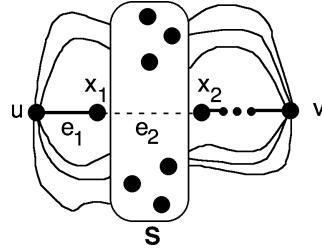
Let  $P_i$  be the concatenation of paths  $P'_i$  and  $P''_i$ , for  $i = 1, \dots, k$ . Then the set  $\{P_i\}$  is a collection of  $k$  internally disjoint  $u-v$  paths in  $G$ .  $\diamond$  (Case 1)



**Figure 5.3.11** Each of the graphs  $G_u^*$  and  $G_v^*$  has  $k$  internally disjoint paths.

*Case 2:* Suppose that for each  $u-v$  separating set of size  $k$ , one of the vertices  $u$  or  $v$  is adjacent to all the vertices in that separating set.

Let  $P = \langle u, e_1, x_1, e_2, x_2, \dots, v \rangle$  be a shortest  $u-v$  path in  $G$ . By Assertion 5.3.4a, we can assume that  $P$  has length at least 3 and that vertex  $x_1$  is not adjacent to vertex  $v$ . By Proposition 5.1.3, the edge-deletion subgraph  $G - e_2$  is connected. Let  $S$  be a smallest  $u-v$  separating set in subgraph  $G - e_2$  (see Figure 5.3.12). Then  $S$  is a  $u-v$  separating set in the vertex-deletion subgraph  $G - x_1$  (since  $G - x_1$  is a subgraph of  $G - e_2$ ). Thus,  $S \cup \{x_1\}$  is a  $u-v$  separating set in  $G$ , which implies that  $|S| \geq k-1$ , by the minimality of  $k$ . On the other hand, the minimality of  $|S|$  in  $G - e_2$  implies that  $|S| \leq k$ , since every  $u-v$  separating set in  $G$  is also a  $u-v$  separating set in  $G - e_2$ .



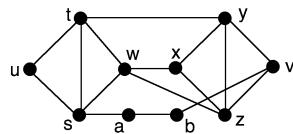
**Figure 5.3.12** Completing Case 2 of Menger's Theorem.

If  $|S| = k$ , then, by the induction hypothesis, there are  $k$  internally disjoint  $u-v$  paths in  $G - e_2$  and, hence, in  $G$ . If  $|S| = k-1$ , then  $x_i \notin S$ ,  $i = 1, 2$  (otherwise  $S - \{x_i\}$  would be a  $u-v$  separating set in  $G$ , contradicting the minimality of  $k$ ). Thus, the sets  $S \cup \{x_1\}$  and  $S \cup \{x_2\}$  are both of size  $k$  and both  $u-v$  separating sets of  $G$ . The condition for Case 2 and the fact that vertex  $x_1$  is not adjacent to  $v$  imply that every vertex in  $S$  is adjacent to vertex  $u$ . Hence, no vertex in  $S$  is adjacent to  $v$  (lest there be a  $u-v$  path of length 2). But then the condition for Case 2 applied to  $S \cup \{x_2\}$  implies that vertex  $x_2$  is adjacent to vertex  $u$ , which contradicts the minimality of path  $P$  and completes the proof.  $\diamond$

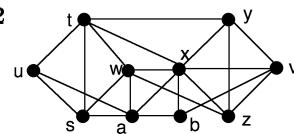
**Remark:** Digraph versions of Menger's Theorem are also proved in Chapter 13. The assertions are the same, except that all paths are directed paths. The edge form and arc form of Menger's Theorem were proved by Ford and Fulkerson in 1955. Their approach, which is presented in Chapter 13, uses an algorithm to maximize the flow through a *capacitated network*, and a minimum cut for the network is a byproduct of the algorithm.

**EXERCISES for Section 5.3**

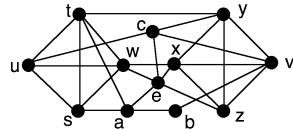
In Exercises 5.3.1 through 5.3.4, find the maximum number of internally disjoint  $u$ - $v$  paths for the given graph, and use Certificate of Optimality (Corollary 5.3.3) to justify your answer.

5.3.1<sup>s</sup>

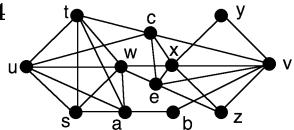
5.3.2



5.3.3



5.3.4



In Exercises 5.3.5 through 5.3.8, find the maximum number of edge-disjoint  $u$ - $v$  paths for the specified graph, and use Edge Form of Certificate of Optimality (Proposition 5.3.9) to justify your answer.

5.3.5<sup>s</sup> The graph of Exercise 5.3.1.

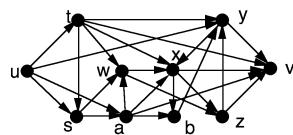
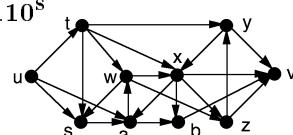
5.3.6 The graph of Exercise 5.3.2.

5.3.7 The graph of Exercise 5.3.3.

5.3.8 The graph of Exercise 5.3.4.

In Exercises 5.3.9 and 5.3.10, find the maximum number of arc-disjoint directed  $u$ - $v$  paths for the given graph, and use the digraph version of Proposition 5.3.9 to justify your answer. That is, find  $k$  arc-disjoint  $u$ - $v$  paths and a set of  $k$  arcs that separate vertices  $u$  and  $v$ , for some integer  $k$ .

5.3.9

5.3.10<sup>s</sup>

5.3.11 Prove Certificate of Optimality (Corollary 5.3.3).

5.3.12 Prove Edge Form of Certificate of Optimality (Proposition 5.3.9).

5.3.13 Prove Edge Form of Menger's Theorem 5.3.10.

5.3.14 Prove Whitney's  $k$ -Edge-Connected Characterization Theorem 5.3.11.5.3.15<sup>s</sup> Prove that Tutte Synthesis Theorem 5.2.5 always produces a 3-connected graph. (Hint: Develop an induction that uses Whitney's  $k$ -Connected Characterization Theorem 5.3.6.)5.3.16 Let  $G$  be a simple  $k$ -connected graph with  $k \geq 2$ . Let  $S$  be a set of two edges and  $W$  a set of  $k - 2$  vertices. Prove that there exists a cycle in  $G$  containing the elements of  $S$  and  $W$ .5.3.17 Let  $G$  be a simple  $k$ -connected graph. Let  $W = \{w_1, w_2, \dots, w_k\}$  be a set of  $k$  vertices and  $v \in V_G - W$ . Prove that there exists a  $v$ - $w_i$  path  $P_i$ ,  $i = 1, \dots, k$ , such that the collection  $\{P_i\}$  of  $k$  paths is internally disjoint.

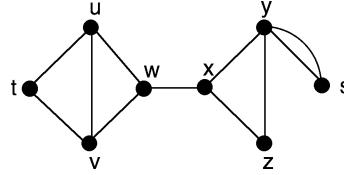
## 5.4 BLOCK DECOMPOSITIONS

**DEFINITION:** A **block** of a loopless graph is a maximal connected subgraph  $H$  such that no vertex of  $H$  is a cut-vertex of  $H$ .

Thus, if a block has at least three vertices, then it is a maximal 2-connected subgraph. The only other types of blocks (in a loopless graph) are isolated vertices or dipoles (2-vertex graphs with a single edge or a multi-edge).

**Remark:** The blocks of a graph  $G$  are the blocks of the components of  $G$  and can therefore be identified one component of  $G$  at a time. Also, self-loops (or their absence) have no effect on the connectivity of a graph. For these reasons we assume throughout this section (except for the final subsection) that all graphs under consideration are loopless and connected.

**Example 5.4.1:** The graph in Figure 5.4.1 has four blocks; they are the subgraphs induced on the vertex subsets  $\{t, u, w, v\}$ ,  $\{w, x\}$ ,  $\{x, y, z\}$ , and  $\{y, s\}$ .



**Figure 5.4.1** A graph with four blocks.

**Remark:** By definition, a block  $H$  of a graph  $G$  has no cut-vertices (of  $H$ ), but  $H$  may contain vertices that are cut-vertices of  $G$ . For instance, in the above figure, the vertices  $w$ ,  $x$ , and  $y$  are cut vertices of  $G$ .

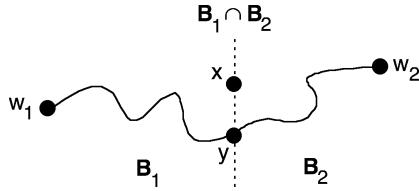
The complete graphs  $K_n$  have no cut-vertices. The next result concerns the other extreme.

**Proposition 5.4.1.** Every nontrivial connected graph  $G$  contains two or more vertices that are not cut-vertices.

**Proof:** Choose two 1-valent vertices of a spanning tree of graph  $G$ . ◊

**Proposition 5.4.2.** Two different blocks of a graph can have at most one vertex in common.

**Proof:** Let  $B_1$  and  $B_2$  be two different blocks of a graph  $G$ , and suppose that  $x$  and  $y$  are vertices in  $B_1 \cap B_2$ . Since the vertex-deletion subgraph  $B_1 - x$  is a connected subgraph of  $B_1$ , there is a path in  $B_1 - x$  between any given vertex  $w_1 \in B_1 - x$  and vertex  $y$ . Similarly, there is a path in  $B_2 - x$  from vertex  $y$  to any given vertex  $w_2 \in B_2 - x$  (see Figure 5.4.2). The concatenation of these two paths is a  $w_1-w_2$  walk in the vertex-deletion subgraph  $(B_1 \cup B_2) - x$ , which shows that  $x$  is not a cut-vertex of the subgraph  $B_1 \cup B_2$ . The same argument shows that no other vertex in  $B_1 \cap B_2$  is a cut-vertex of  $B_1 \cup B_2$ . Moreover, none of the vertices that are in exactly one of the  $B_i$ 's is a cut-vertex of  $B_1 \cup B_2$ , since such a vertex would be a cut-vertex of that block  $B_i$ . Thus, the subgraph  $B_1 \cup B_2$  has no cut-vertices, which contradicts the maximality of blocks  $B_1$  and  $B_2$ . ◊



**Figure 5.4.2** Two blocks cannot have more than one vertex in common.

The following assertions are immediate consequences of Proposition 5.4.2.

**Corollary 5.4.3.** The edge-sets of the blocks of a graph  $G$  partition  $E_G$ .

◇ (Exercises)

**Corollary 5.4.4.** Let  $x$  be a vertex in a graph  $G$ . Then  $x$  is a cut-vertex of  $G$  if and only if  $x$  is in two different blocks of  $G$ .

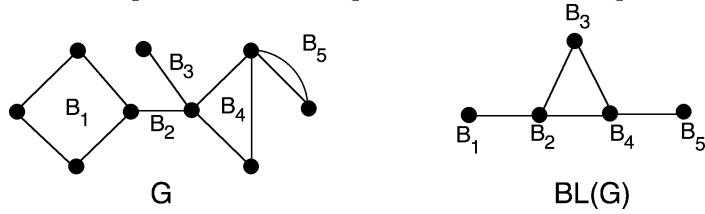
◇ (Exercises)

**Corollary 5.4.5.** Let  $B_1$  and  $B_2$  be distinct blocks of a connected graph  $G$ . Let  $y_1$  and  $y_2$  be vertices in  $B_1$  and  $B_2$ , respectively, such that neither is a cut-vertex of  $G$ . Then vertex  $y_1$  is not adjacent to vertex  $y_2$ .

◇ (Exercises)

**DEFINITION:** The **block graph** of a graph  $G$ , denoted  $BL(G)$ , is the graph whose vertices correspond to the blocks of  $G$ , such that two vertices of  $BL(G)$  are joined by a single edge whenever the corresponding blocks have a vertex in common.

**Example 5.4.2:** Figure 5.4.3 shows a graph  $G$  and its block graph  $BL(G)$ .



**Figure 5.4.3** A graph and its block graph.

**DEFINITION:** A **leaf block** of a graph  $G$  is a block that contains exactly one cut-vertex of  $G$ .

The following result is used in §9.1 to prove *Brooks's Theorem* concerning the *chromatic number* of graph.

**Proposition 5.4.6.** Let  $G$  be a connected graph with at least one cut-vertex. Then  $G$  has at least two leaf blocks.

◇ (Exercises)

### Finding the Blocks of a Graph

In §4.4, depth-first search was used to find the cut-vertices of a connected graph (Algorithm 4.4.3). The following algorithm, which uses Algorithm 4.4.3 as a subroutine, finds the blocks of a connected graph. Recall from §4.4 that  $low(w)$  is the smallest *dfnumber* among all vertices in the depth-first tree that are joined to some descendant of vertex  $w$  by a non-tree edge.

**Algorithm 5.4.1: Block-Finding**

*Input:* a connected graph  $G$ .

*Output:* the vertex-sets  $B_1, B_2, \dots, B_l$  of the blocks of  $G$ .

Apply Algorithm 4.4.3 to find the set  $K$  of cut-vertices of graph  $G$ .

Initialize the block counter  $i := 0$ .

For each cut-vertex  $v$  in set  $K$  (in order of decreasing  $dfnumber$ )

    For each child  $w$  of  $v$  in depth-first search tree  $T$

        If  $low(w) \geq dfnumber(v)$

            Let  $T^w$  be the subtree of  $T$  rooted at  $w$ .

$i := i + 1$

$B_i := V_{T^w} \cup \{v\}$

$T := T - V_{T^w}$

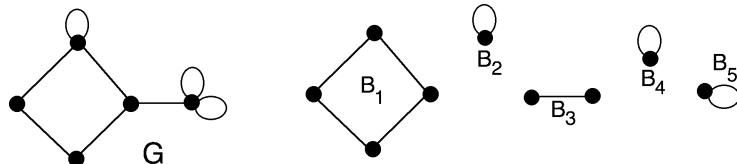
Return sets  $B_1, B_2, \dots, B_i$ .

**COMPUTATIONAL NOTE:** With some relatively minor modifications of Algorithm 5.4.1, the cut-vertices and blocks of a graph can be found in one pass of a depth-first search (see e.g., [AhHoUl83], [Ba83], [ThSw92]). A similar one-pass algorithm for finding the strongly connected components of a digraph is given in §12.5.

**Block Decomposition of Graphs With Self-Loops**

In a graph with self-loops, each self-loop and its endpoint are regarded as a distinct block, isomorphic to the bouquet  $B_1$ . The other blocks of such a graph are exactly the same as if the self-loops were not present. This extended concept of block decomposition preserves the property that the blocks partition the edge-set.

**Example 5.4.3:** The block decomposition of the graph  $G$  shown in Figure 5.4.4 contains five blocks, three of which are self-loops.

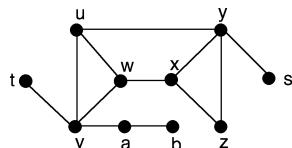


**Figure 5.4.4** A graph  $G$  and its five blocks.

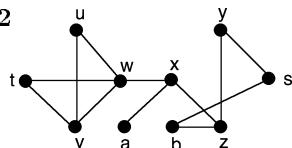
**EXERCISES for Section 5.4**

*In Exercises 5.4.1 through 5.4.4, identify the blocks in the given graph and draw the block graph.*

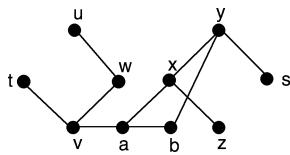
5.4.1<sup>s</sup>



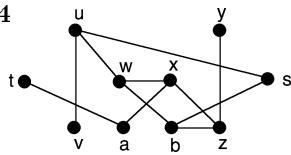
5.4.2



5.4.3



5.4.4

5.4.5<sup>s</sup> Draw a graph whose block graph is the complete graph  $K_3$ .

5.4.6 Find two non-isomorphic connected graphs with six vertices, six edges, and three blocks.

5.4.7 Find a graph whose block graph is the  $n$ -cycle graph  $C_n$ .5.4.8<sup>s</sup> How many non-isomorphic simple connected graphs are there that have seven vertices, seven edges, and three blocks?*In Exercises 5.4.9 through 5.4.12, apply Algorithm 5.4.1 to the specified graph.*

5.4.9 The graph of Exercise 5.4.1.      5.4.10 The graph of Exercise 5.4.2.

5.4.11 The graph of Exercise 5.4.3.      5.4.12 The graph of Exercise 5.4.4.

5.4.13 Prove or disprove: Every simple graph is the block graph of some graph.

5.4.14<sup>s</sup> Prove or disprove: Two graphs are isomorphic if and only if their block graphs are isomorphic.

5.4.15 Prove Corollary 5.4.3.

5.4.16 Prove Corollary 5.4.4.

5.4.17 Prove Corollary 5.4.5.

**DEFINITION:** Let  $G$  be a simple connected graph with at least two blocks. The **block-cutpoint graph**  $bc(G)$  of  $G$  is the bipartite graph with vertex bipartition  $\{V_b, V_c\}$ , where the vertices in  $V_b$  bijectively correspond to the blocks of  $G$  and the vertices in  $V_c$  bijectively correspond to the cut-vertices of  $G$ , and where vertex  $v_b$  is adjacent to vertex  $v_c$  if cut-vertex  $c$  is a vertex of block  $b$ .

*In Exercises 5.4.18 through 5.4.21, draw the block-cutpoint graph  $bc(G)$  of the specified graph  $G$ , and identify the vertices in  $bc(G)$  that correspond to leaf blocks of  $G$ .*5.4.18<sup>s</sup> The graph of Exercise 5.4.1.      5.4.19 The graph of Exercise 5.4.2.

5.4.20 The graph of Exercise 5.4.3.      5.4.21 The graph of Exercise 5.4.4.

5.4.22<sup>s</sup> Let  $G$  be a simple connected graph with at least two blocks. Prove that the block-cutpoint graph  $bc(G)$  is a tree.

5.4.23 Prove Proposition 5.4.6. (Hint: See Exercise 5.4.22.)

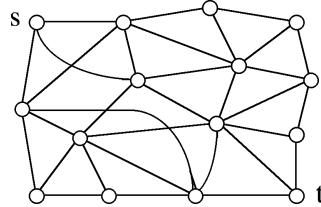
5.4.24 [Computer Project] Implement Algorithm 5.4.1 and run the program, using each of the graphs in Exercises 5.4.-2 through 5.4.1 as input.

## 5.5 SUPPLEMENTARY EXERCISES

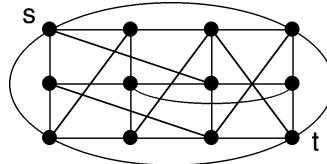
5.5.1 Calculate the vertex-connectivity of  $K_{4,7}$ .5.5.2 Prove that the complete bipartite graph  $K_{m,m}$  is  $m$ -connected.5.5.3 Prove that the vertex-connectivity of the hypercube graph  $Q_n$  is  $n$ .

**5.5.4** In the  $n$ -dimensional hypercube graph  $Q_n$ , find  $n$  edge-disjoint paths from  $(0, 0, \dots, 0)$  to  $(1, 1, \dots, 1)$ .

**5.5.5** Prove that deleting two vertices from the graph below is insufficient to separate vertex  $s$  from vertex  $t$ .



**5.5.6** How many vertices must be removed from the graph below to separate vertex  $s$  from vertex  $t$ ?



## GLOSSARY

**block** of a loopless graph  $G$ : a maximal connected subgraph  $H$  of  $G$  such that no vertex of  $H$  is a cut-vertex of  $H$ ; in a graph with self-loops, each self-loop and its endpoint are regarded as a distinct block, isomorphic to the bouquet  $B_1$ .

—, **leaf**: a block that contains exactly one cut-vertex.

**block-cutpoint graph**  $bc(G)$  of a simple connected graph  $G$  with at least two blocks: the bipartite graph with vertex bipartition  $\langle V_b, V_c \rangle$ , where the vertices in  $V_b$  bijectively correspond to the blocks of  $G$  and the vertices in  $V_c$  bijectively correspond to the cut-vertices of  $G$ , and where vertex  $v_b$  is adjacent to vertex  $v_c$  if cut-vertex  $c$  is a vertex of block  $b$ .

**block graph**  $BL(G)$  of a graph  $G$ : the graph whose vertices correspond to the blocks of  $G$ , such that two vertices of  $BL(G)$  are joined by a single edge whenever the corresponding blocks have a vertex in common.

**bridge**: synonym for *cut-edge*.

**ceiling**  $[x]$  of a real number  $x$ : the smallest integer greater than or equal to  $x$ .

**closed-ear addition**: synonym for *cycle addition*.

**$k$ -connected graph**  $G$ : a connected graph with  $\kappa_v(G) \geq k$ .

**cut-edge** of a graph: an edge whose removal increases the number of components.

**cutpoint**: a synonym for *cut-vertex*.

**cut-vertex** of a graph: a vertex whose removal increases the number of components.

**cycle addition** to a graph  $G$ : the addition of a cycle that has exactly one vertex in common with  $G$ .

**disconnects** a connected graph  $G$ : said of a subset of vertices or edges whose deletion from  $G$  results in a non-connected graph.

**ear decomposition:** a *Whitney* or *Whitney-Robbins synthesis*.

—, **closed-**: a *Whitney-Robbins synthesis*.

—, **open-**: a *Whitney synthesis*.

**$k$ -edge-connected graph**  $G$ : a connected graph such that every edge-cut has at least  $k$  edges (i.e.,  $\kappa_e(G) \geq k$ ).

**edge-connectivity** of a connected graph  $G$ : the minimum number of edges whose removal can disconnect  $G$ ; denoted  $\kappa_e(G)$ .

**edge-cut** in a graph  $G$ : a subset  $D$  of edges such that  $G - D$  has more components than  $G$ .

**floor**  $[x]$  of a real number  $x$ : the greatest integer less than or equal to  $x$ .

**Harary graph**  $H_{k,n}$ : a  $k$ -connected,  $n$ -vertex simple graph with the fewest edges possible, whose construction is due to Frank Harary (see §5.2).

**internally disjoint paths**: paths that have no internal vertex in common.

**leaf block**: see *block*, *leaf*.

**local vertex-connectivity** between non-adjacent vertices  $s$  and  $t$ : the size of a smallest  $s-t$  separating vertex set; denoted  $\kappa_v(s,t)$ .

**mod  $n$  distance**  $|j - i|_n$  between integers  $i, j \in \{0, 1, \dots, n - 1\}$ : the smaller of the two values  $|j - i|$  and  $n - |j - i|$ ; used in Harary's construction of optimal  $k$ -connected graphs (§5.2).

**open-ear addition**: synonym for *path addition*.

**path addition** to a graph  $G$ : the addition to  $G$  of a path between two existing vertices of  $G$ , such that the edges and internal vertices of the path are new.

**$u-v$  separating set**  $S$  of vertices or of edges: a set  $S$  such that the vertices  $u$  and  $v$  lie in different components of the deletion subgraph  $G - S$ .

**strict  $x-W$  path**: a path joining vertex  $x$  to a vertex in vertex set  $W$  and containing no other vertex of  $W$ . A strict  $W-x$  path is the reverse of a strict  $x-W$  path.

**Tutte synthesis**: a constructive characterization of 3-connected graphs that starts with a wheel graph; see Tutte Synthesis Theorem 5.2.5.

**unicyclic graph**: a connected graph with exactly one cycle.

**vertex-connectivity** of a connected graph  $G$ : the minimum number of vertices whose removal can disconnect  $G$  or reduce it to a 1-vertex graph; denoted  $\kappa_v(G)$ .

—, **local** between non-adjacent vertices  $s$  and  $t$ : the size of a smallest  $s-t$  separating vertex set; denoted  $\kappa_v(s,t)$ .

**vertex-cut** in a graph  $G$ : a subset  $U$  of vertices such that  $G - U$  has more components than  $G$ .

**( $n$ -spoked) wheel graph**  $W_n$ : the suspension of the  $(n - 1)$ -vertex cycle graph from a new vertex  $x$ . That is,  $W_n = C_{n-1} + x$ .

**Whitney synthesis**: a Whitney-Robbins synthesis that uses path additions *only*.

**Whitney-Robbins synthesis** of a graph  $G$  from a graph  $H$ : a sequence of graphs,  $G_0, G_1, \dots, G_l$ , where  $G_0 = H$ ,  $G_l = G$ , and  $G_i$  is the result of a path or cycle addition to  $G_{i-1}$ , for  $i = 1, \dots, l$ .

# Chapter 6

---

## OPTIMAL GRAPH TRAVERSALS

### 6.1 Eulerian Trails and Tours

### 6.2 DeBruijn Sequences and Postman Problems

### 6.3 Hamiltonian Paths and Cycles

### 6.4 Gray Codes and Traveling Salesman Problems

---

#### INTRODUCTION

Walks in a graph that use all the edges or all the vertices are often called *graph traversals*. A variety of practical problems can be posed in terms of graph traversals, and they tend to fall into one of two categories. The first half of the chapter deals with *eulerian-type* problems, which require that each edge be traversed at least once. The second half is concerned with *hamiltonian-type* problems, which involve traversals that must visit each vertex at least once.

Sometimes lurking beneath the surface of an application is some version of an eulerian or hamiltonian problem. Recognizing such a problem allows a host of strategies and algorithms to be brought to bear on its solution. A number of applications where this occurs are given. The classical optimization applications are known as the *Chinese Postman Problem* and the *Traveling Salesman Problem*.

On the surface, the two types of problems are similar, one involving an edge-based condition, the other an analogous, vertex-based condition. But the similarity ends there. Most eulerian-type problems yield to polynomial-time algorithms, stemming from a simple characterization of eulerian graphs in terms of vertex degrees. In sharp contrast, no simple characterization for hamiltonian graphs is known, and most hamiltonian-type problems are notoriously time-consuming to solve, classified as NP-hard. Practical problems where cost of solution is important call for strategies called *heuristics* that produce answers quickly, forfeiting the guarantee of optimality. Some of the most commonly used heuristics are presented in §6.4.

## 6.1 EULERIAN TRAILS AND TOURS

REVIEW FROM §1.5:

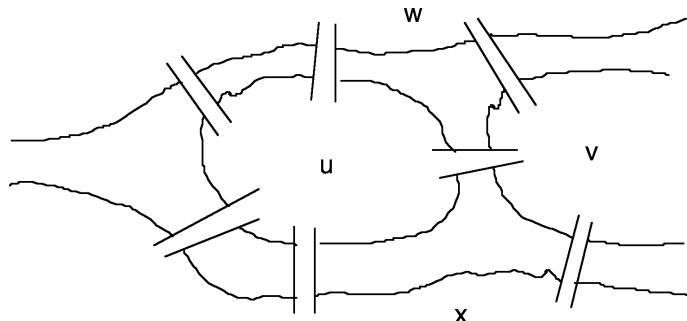
- An **eulerian trail** in a graph is a trail that contains every edge of that graph.
- An **eulerian tour** is a closed eulerian trail.
- An **eulerian graph** is a graph that has an eulerian tour.

REVIEW FROM §4.5: [Eulerian-Graph Characterization] *The following statements are equivalent for a connected graph  $G$ .*

1.  $G$  is eulerian.
2. The degree of every vertex in  $G$  is even.
3.  $E_G$  is the union of the edge-sets of a set of edge-disjoint cycles in  $G$ .

### Königsberg Bridges Problem

In the town of Königsberg in what was once East Prussia, the two branches of the River Pregel converge and flow through to the Baltic Sea. Parts of the town were on an island and a headland that were joined to the outer river banks and to each other by seven bridges, as shown in Figure 6.1.1. The townspeople wanted to know if it was possible to take a walk that crossed each of the bridges exactly once before returning to the starting point. The Prussian emperor, Frederick the Great, brought the problem to the attention of the famous Swiss mathematician Leonhard Euler, and Euler proved that no such walk was possible. His solution in 1736 is generally regarded as the origin of graph theory.

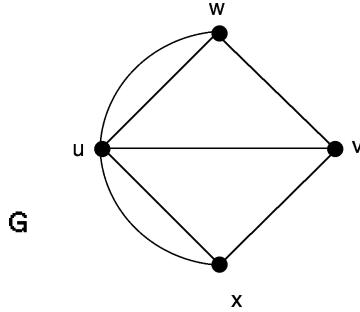


**Figure 6.1.1** The seven bridges of Königsberg.

Euler's model for the problem was a graph with four vertices, one for each of the land masses  $u, v, w$ , and  $x$ , and seven edges, one for each of the bridges, as shown in Figure 6.1.2 below. In modern terminology, the problem was to determine whether that graph is eulerian.

Since the graph has vertices of odd degree, there is no eulerian tour, by the Eulerian-Graph Characterization (§4.5). The argument given in the first part of the proof of that theorem is essentially the one that Euler used back in 1736.

The second and third parts of that proof show that a graph whose vertices all have even degree must be eulerian. The essence of those arguments is embodied in the

**Figure 6.1.2** Graph representation of Königsberg.

Eulerian Tour Algorithm given below, developed by Hierholzer in 1873, which constructs an eulerian tour for any connected graph whose vertices are all of even degree.

**Algorithm 6.1.1: Eulerian Tour**

*Input:* a connected graph  $G$  whose vertices all have even degree.

*Output:* an eulerian tour  $T$ .

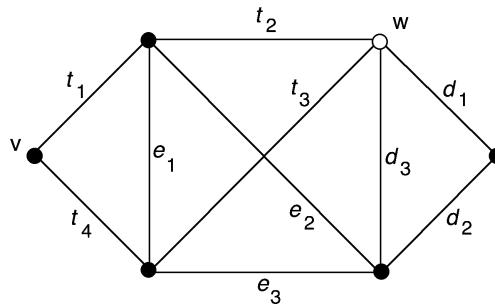
```

Start at any vertex  $v$ , and construct a closed trail  $T$  in  $G$ .
While there are edges of  $G$  not already in trail  $T$ 
    Choose any vertex  $w$  in  $T$  that is incident with an unused edge.
    Starting at vertex  $w$ , construct a closed trail  $D$  of unused edges.
    Enlarge trail  $T$  by splicing trail  $D$  into  $T$  at vertex  $w$ .
Return  $T$ .

```

**COMPUTATIONAL NOTE:** A modified *depth-first search* (§4.2), in which every unused edge remains in the stack, can be used to construct the closed trails.

**Example 6.1.1:** The key step in Algorithm 6.1.1 is enlarging a closed trail by combining it with a second closed trail — the *detour*. To illustrate, consider the closed trails  $T = \langle t_1, t_2, t_3, t_4 \rangle$  and  $D = \langle d_1, d_2, d_3 \rangle$  in the graph shown in Figure 6.1.3. The closed trail that results when detour  $D$  is spliced into trail  $T$  at vertex  $w$  is given by  $T' = \langle t_1, t_2, d_1, d_2, d_3, t_3, t_4 \rangle$ . At the next iteration, the trail  $\langle e_1, e_2, e_3 \rangle$  is spliced into trail  $T'$ , resulting in an eulerian tour of the entire graph.

**Figure 6.1.3**

### Open Eulerian Trails

Certain applications require *open* eulerian trails, which do not end at the starting vertex. The following characterization of graphs having an open eulerian trail is obtained directly from the characterization of eulerian graphs.

**Theorem 6.1.1.** *A connected graph  $G$  has an open eulerian trail if and only if it has exactly two vertices of odd degree. Furthermore, the initial and final vertices of an open eulerian trail must be the two vertices of odd degree.*

**Proof:** ( $\Rightarrow$ ) Suppose that  $\langle x, e_1, v_1, e_2, \dots, e_m, y \rangle$  is an open eulerian trail in  $G$ . Adding a new edge  $e$  joining vertices  $x$  and  $y$  creates a new graph  $G^* = G + e$  with an eulerian tour  $\langle x, e_1, v_1, e_2, \dots, e_m, y, e, x \rangle$ . By the eulerian-graph characterization, the degree in the eulerian graph  $G^*$  of every vertex must be even, and thus, the degree in graph  $G$  of every vertex except  $x$  and  $y$  is even.

( $\Leftarrow$ ) Suppose that  $x$  and  $y$  are the only two vertices of graph  $G$  with odd degree. If  $e$  is a new edge joining  $x$  and  $y$ , then all the vertices of the resulting graph  $G^*$  have even degree. It follows from the eulerian-graph characterization that graph  $G^*$  has an eulerian tour  $T$ . Hence, the trail  $T - e$  obtained by deleting edge  $e$  from tour  $T$  is an open eulerian trail of  $G^* - e = G$ .  $\diamond$

### Eulerian Trails in Digraphs

The use of the term *eulerian* is identical for digraphs, except that all trails are directed. The characterizations of graphs that have eulerian tours or open eulerian trails apply to digraphs as well. Their proofs are essentially the same as the ones for undirected graphs and are left as exercises.

**Theorem 6.1.2 [Eulerian-Digraph Characterization].** *The following statements are equivalent for a connected digraph  $D$ .*

1. *Digraph  $D$  is eulerian.*
2. *For every vertex  $v$  in  $D$ ,  $\text{indegree}(v) = \text{outdegree}(v)$ .*
3. *The edge-set  $E_D$  is the union of the edge-sets of a set of edge-disjoint directed cycles in digraph  $D$ .*  $\diamond$  (Exercises)

**Theorem 6.1.3.** *A connected digraph has an open eulerian trail from vertex  $x$  to vertex  $y$  if and only if  $\text{indegree}(x) + 1 = \text{outdegree}(x)$ ,  $\text{indegree}(y) = \text{outdegree}(y) + 1$ , and all vertices except  $x$  and  $y$  have equal indegree and outdegree.*  $\diamond$  (Exercises)

### Tarry's Maze-Searching Algorithm Revisited

In §4.4, we recognized that Tarry's maze-searching algorithm follows a depth-first search strategy. In the context of this section, the algorithm produces an eulerian tour of the graph obtained by duplicating each edge of the input graph. Tarry's algorithm is just one of several maze-searching algorithms. For a more extensive study, see [Fl91].

**DEFINITION:** A **double tracing** is a closed walk that traverses every edge exactly twice. A double tracing is **bidirectional** if every edge is used once in each of its two directions.

**Proposition 6.1.4.** Every connected graph has a bidirectional double tracing. In a tree, every double tracing is bidirectional.  $\diamond$  (Exercises)

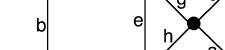
## EXERCISES for Section 6.1

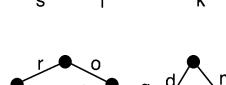
In Exercises 6.1.1 through 6.1.4, determine which graphs in the given graph family are eulerian.

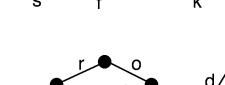
- 6.1.1<sup>s</sup> The complete graph  $K_n$ .      6.1.2 The complete bipartite graph  $K_{m,n}$ .  
 6.1.3 The  $n$ -vertex wheel  $W_n$ .      6.1.4 The hypercube graph  $Q_n$ .  
 6.1.5<sup>s</sup> Which platonic graphs (§1.2) are eulerian?

In Exercises 6.1.6 through 6.1.9, apply Algorithm 6.1.1 to construct an eulerian tour of the given graph. Begin the construction at vertex  $s$ .

- 6.1.6 

6.1.7<sup>s</sup> 

6.1.8 

6.1.9 

- 6.1.10<sup>s</sup>** Use the strategy of the second half of the proof of Theorem 6.1.1 to design a variation of Algorithm 6.1.1 that constructs an open eulerian trail in any graph that has exactly two vertices of odd degree.

In Exercises 6.1.11 through 6.1.14, use the modified version of Algorithm 6.1.1 from Exercise 6.1.10 to construct an open eulerian trail in the given graph.

- 6.1.11

6.1.12

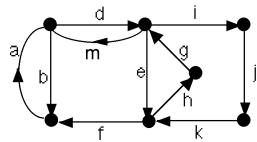
6.1.13<sup>s</sup>

6.1.14

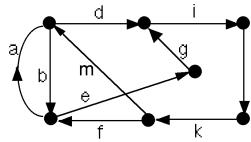
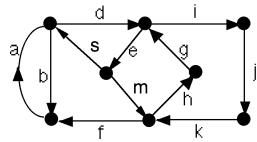
- 6.1.15<sup>s</sup>** Design a variation of Algorithm 6.1.1 that constructs an eulerian tour of any eulerian digraph.

In Exercises 6.1.16 through 6.1.19, use an appropriate modification of Algorithm 6.1.1 to construct an eulerian tour or open eulerian trail in the given digraph.

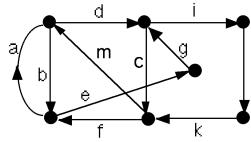
6.1.16



6.1.17

6.1.18<sup>s</sup>

6.1.19



6.1.20 Referring to the initial step of Algorithm 6.1.1, why is it always possible to construct a closed trail, regardless of the choice of starting vertex?

6.1.21<sup>s</sup> Referring to the while-loop in Algorithm 6.1.1, why is it always possible to find a vertex on trail  $T$  that is incident on an unused edge?

6.1.22 Prove Theorem 6.1.2. (Hint: See proof of eulerian-graph characterization in §4.5.)

6.1.23 Prove Theorem 6.1.3. (Hint: Use Theorem 6.1.2.)

6.1.24 Prove Proposition 6.1.0.

6.1.25<sup>s</sup> Prove that if a simple graph  $G$  is eulerian, then its line graph  $L(G)$  (§1.2) is eulerian.

6.1.26 Prove or disprove: The line graph of *any* eulerian graph is eulerian.

6.1.27<sup>s</sup> Prove or disprove: If the line graph of a graph  $G$  is eulerian, then  $G$  is eulerian.

6.1.28 Prove that if a connected graph  $G$  has  $2k$  vertices of odd degree, then there is a set of  $k$  edge-disjoint trails that use all the edges of  $G$ .

6.1.29 [Computer Project] Implement Algorithm 6.1.1 and run the program on each of the graphs in Exercises 6.1.6 through 6.1.9.

6.1.30 [Computer Project] Implement a modified version of Algorithm 6.1.1 so that it finds an open eulerian trail in a graph that has exactly two vertices of odd degree. Run the program on the graphs in Exercises 6.1.11 through 6.1.14.

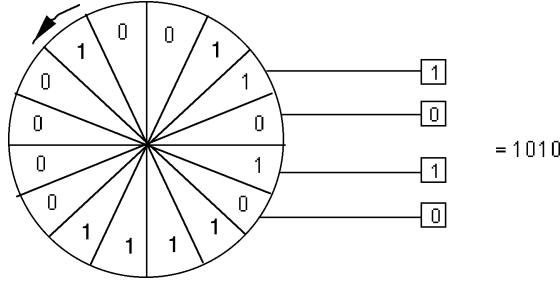
## 6.2 DEBRUIJN SEQUENCES AND POSTMAN PROBLEMS

**DEFINITION:** A bitstring of length  $2^n$  is called a **( $2, n$ )-deBruijn sequence** if each of the  $2^n$  possible bitstrings of length  $n$  occurs *exactly once* as a substring, where wraparound is allowed.

**Example 6.2.1:** The following four bitstrings are  $(2, n)$ -deBruijn sequences for the cases  $n = 1, 2, 3, 4$ , respectively.

01      0110      01110100      0000100110101111

**Application 6.2.1 Identifying the Position of a Rotating Drum:** Suppose that a rotating drum has 16 different sectors. We can assign a 0 or 1 to each of the 16 sectors by putting conducting material in some of the sectors and nonconducting material in others. Sensors can then be placed in a fixed position to “read” a 4-bit string corresponding to the four sectors that are positioned there. For example, in Figure 6.2.1, the sensors read 1010.



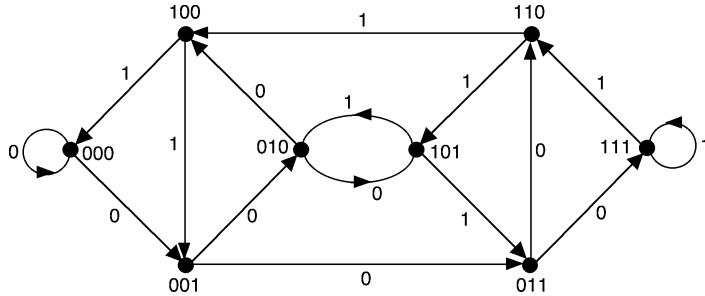
**Figure 6.2.1** A rotating drum with 16 sectors and 4 sensors.

Observe that the bitstring of length 16 formed by starting at any sector of the drum and moving clockwise (or counterclockwise) around the drum is a  $(2, 4)$ -deBruijn sequence. This means that the position of the drum can be automatically determined by the 4-bit substring that the sensors pick up. This is more space-efficient than writing a 4-bit identifying code in each sector.

In general,  $(2, n)$ -deBruijn sequences can be constructed for any positive integer  $n$ . In fact, the construction outlined below can be generalized to produce a  $(p, n)$ -deBruijn sequence for any positive integers  $p$  and  $n$ , where a  $(p, n)$ -sequence consists of strings of characters from a set of  $p$  characters instead of just a binary set. The Dutch mathematician N. DeBruijn used a digraph to construct such sequences, based on an eulerian tour.

**DEFINITION:** The  $(2, n)$ -**deBruijn digraph**  $D_{2,n}$  consists of  $2^{n-1}$  vertices, labeled by the bitstrings of length  $n - 1$ , and  $2^n$  arcs, labeled by the bitstrings of length  $n$ . The arc from vertex  $b_1 b_2 \dots b_{n-1}$  to vertex  $b_2 \dots b_{n-1} b_n$  is labeled  $b_1 b_2 \dots b_n$ .

**Example 6.2.2:** Figure 6.2.2 shows the  $(2, 4)$ -deBruijn digraph. To simplify the subsequent discussion, each arc in the figure is labeled with only the *leftmost bit* of its full label. Thus, the actual arc label is the string consisting of the single bit that appears on that arc, followed by the label of the vertex at the head of the arc. For instance, the actual label for the arc directed from 000 to 001 is 0001.



**Figure 6.2.2** The  $(2, 4)$ -deBruijn digraph  $D_{2,4}$ .

**Proposition 6.2.1.** *The  $(2, n)$ -deBruijn digraph  $D_{2,n}$  is eulerian.*

**Proof:** The deBruijn graph  $D_{2,n}$  is strongly connected, since if  $a_1a_2 \cdots a_{n-1}$  and  $b_1b_2 \cdots b_{n-1}$  are any two vertices of  $D_{2,n}$ , then the vertex sequence:

$$a_1a_2 \cdots a_{n-1}; \quad a_2 \cdots a_{n-1}b_1; \quad a_3 \cdots a_{n-1}b_1b_2; \quad \dots; \quad b_1b_2 \cdots b_{n-1}$$

defines a directed trail from  $a_1a_2 \cdots a_{n-1}$  to  $b_1b_2 \cdots b_{n-1}$ . Moreover, for every vertex  $b_1b_2 \cdots b_{n-1}$  of  $D_{2,n}$ , the only outgoing arcs from  $b_1b_2 \cdots b_{n-1}$  are  $b_1b_2 \cdots b_{n-1}0$  and  $b_1b_2 \cdots b_{n-1}1$ , and the only incoming arcs to  $b_1b_2 \cdots b_{n-1}$  are  $0b_1b_2 \cdots b_{n-1}$  and  $1b_1b_2 \cdots b_{n-1}$ . Thus,  $\text{indegree}(b_1b_2 \cdots b_{n-1}) = \text{outdegree}(b_1b_2 \cdots b_{n-1}) = 2$ , which implies, by Theorem 6.1.2, that  $D_{2,n}$  is eulerian.  $\diamond$

**Algorithm 6.2.1: Constructing a  $(2, n)$ -deBruijn Sequence**

*Input:* a positive integer  $n$ .

*Output:* a  $(2, n)$ -deBruijn sequence  $S$ .

Construct the  $(2, n)$ -deBruijn digraph  $D_{2,n}$ .

Choose a vertex  $v$ .

Construct a directed eulerian tour  $T$  of  $D_{2,n}$  starting at  $v$ .

Initialize sequence  $S$  as the empty sequence  $\langle \rangle$ .

For each arc  $e$  in tour  $T$  (taken in order of the tour sequence)

Append the single-bit label of arc  $e$  to the right of sequence  $S$ .

**Example 6.2.3:** The construction of a  $(2, 4)$ -deBruijn sequence can begin with the  $(2, 4)$ -deBruijn digraph shown in Figure 6.2.2. It is easy to verify that the following sequence of vertices and arcs is an eulerian tour.

$$\begin{aligned} 000 &\xrightarrow{0} 000 & \xrightarrow{0} 001 &\xrightarrow{0} 010 &\xrightarrow{0} 100 &\xrightarrow{1} 001 &\xrightarrow{0} 011 &\xrightarrow{0} 110 &\xrightarrow{1} 101 \\ &&&&&&&& \\ &\xrightarrow{1} 010 &\xrightarrow{0} 101 &\xrightarrow{1} 011 &\xrightarrow{0} 111 &\xrightarrow{1} 111 &\xrightarrow{1} 110 &\xrightarrow{1} 100 &\xrightarrow{1} 000 \end{aligned}$$

The  $(2, 4)$ -deBruijn sequence 0000100110101111 is the sequence of the arc labels.

**Remark:** Traversing the tour starting from a vertex different from 000 would result in a  $(2, 4)$ -deBruijn sequence that is merely a *cyclic shift* of the sequence above. A different  $(2, 4)$ -deBruijn sequence (i.e., one that is not a cyclic shift) can be obtained by constructing a different eulerian tour in  $D_{2,4}$  (see Exercises).

The following proposition establishes the correctness of the construction of Algorithm 6.2.1.

**Proposition 6.2.2.** *The sequence of single-bit arc labels of any directed trail of length  $k$ ,  $1 \leq k \leq n$ , in the  $(2, n)$ -deBruijn digraph, is precisely the string of the leftmost  $k$  bits of the full label of the initial arc of that trail.*

**Proof:** The definition of  $D_{2,n}$  implies the assertion for  $k = 1$ . Assume for some  $k$ ,  $1 \leq k \leq n - 1$ , that the assertion is true for any directed trail of length  $k$ , and let  $\langle v_0, v_1, \dots, v_{k+1} \rangle$  be any directed trail of length  $k + 1$ . By the induction hypothesis, the single-bit arc labels associated with the subtrail  $\langle v_1, v_2, \dots, v_{k+1} \rangle$  match the first  $k$  bits of the full label of the arc from  $v_1$  to  $v_2$ . By the definition of the deBruijn digraph, these  $k$  bits are the first  $k$  bits of the label on vertex  $v_1$ , which means that they are also

bits 2 through  $k + 1$  of the full label of the arc from  $v_0$  to  $v_1$ . But the single-bit label on that arc is the leftmost bit of its full label, which completes the induction step.  $\diamond$

**Example 6.2.4:** To illustrate Proposition 6.2.2 for the digraph  $D_{2,4}$  in Figure 6.2.2, consider the directed subtrail given by

$$100 \xrightarrow{1} 001 \xrightarrow{0} 011 \xrightarrow{0} 110 \xrightarrow{1} 101$$

The bitstring 1001 formed from the single-bit arc labels is the full label for the arc from 100 to 001.

By the construction of  $D_{2,4}$ , each arc is labeled with a different bitstring of length 4. Thus, since the eulerian tour uses each arc exactly once, Proposition 6.2.2 implies that the resulting sequence contains all possible bitstrings of length 4 and is, therefore, a  $(2, 4)$ -deBruijn sequence.

### Guan's Postman Problem

The Chinese mathematician Meigu Guan [Gu62] introduced the problem of finding a shortest closed walk to traverse every edge of a graph at least once. Guan envisioned a letter carrier who wants to deliver the mail through a network of streets and return to the post office as quickly as possible. Jack Edmonds [Ed65a] dubbed this problem the **Chinese Postman Problem**.

**DEFINITION:** A **postman tour** in a graph  $G$  is a closed walk that uses each edge of  $G$  at least once.

**DEFINITION:** In a weighted graph, an **optimal postman tour** is a postman tour whose total edge-weight is a minimum.

The objective of the Chinese Postman Problem is to find an optimal postman tour in a given weighted graph, where the edge-weights represent some measurable quantity that depends on the application (e.g., distance, time, cost, etc.).

Of course, if each vertex of the graph has even degree, then any eulerian tour is an optimal postman tour. Otherwise, some edges must be retraced (or *deadheaded*). Thus, the goal is to find a postman tour whose deadheaded edges have minimum total edge-weight. It is easy to see that every postman tour of a graph  $G$  corresponds to an eulerian tour of a graph  $G^*$  formed from  $G$  by adding to  $G$  as many additional copies of an edge as the number of times it was deadheaded during the postman tour.

Edmonds and Johnson [EdJo73] solved the Chinese Postman Problem by using a polynomial-time algorithm, outlined below as Algorithm 6.2.2.

**REVIEW FROM §4.7:** A **matching** in a graph  $G$  is a subset  $M$  of  $E_G$  such that no two edges in  $M$  have an endpoint in common.

**DEFINITION:** A **perfect matching** in a graph is a matching in which every vertex is an endpoint of one of the edges.

Matchings appear in a variety of applications, like machine scheduling, job assignment, interview pairings, etc. They are discussed in Chapter 13.

**Algorithm 6.2.2: Optimal Postman Tour**

*Input:* a connected weighted graph  $G$ .

*Output:* an optimal postman tour  $W$ .

Find the set  $S$  of odd-degree vertices of  $G$ .

For each pair of odd-degree vertices  $u$  and  $v$  in  $S$

    Find a shortest path  $P$  in  $G$  between vertices  $u$  and  $v$ .

    Let  $d_{uv}$  be the length of path  $P$ .

Form a complete graph  $K$  on the vertices of  $S$ .

For each edge  $e$  of the complete graph  $K$

    Assign to edge  $e$  the weight  $d_{uv}$ , where  $u$  and  $v$  are the endpoints of  $e$ .

Find a perfect matching  $M$  in  $K$  whose total edge-weight is minimum.

For each edge  $e$  in the perfect matching  $M$

    Let  $P$  be the corresponding shortest path in  $G$  between the endpoints of edge  $e$ .

    For each edge  $f$  on path  $P$

        Add to graph  $G$  a duplicate copy of edge  $f$ , including its edge-weight.

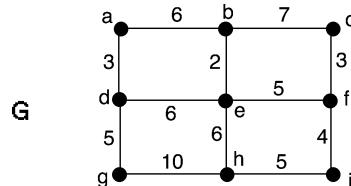
Let  $G^*$  be the eulerian graph formed by adding to graph  $G$  the edge duplications from the previous step.

Construct an eulerian tour  $W$  in  $G^*$ .

The eulerian tour  $W$  corresponds to an optimal postman tour of the original graph  $G$ .

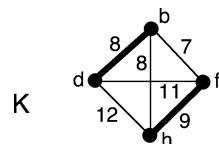
The idea behind the Edmonds and Johnson algorithm is to retrace the edges on certain shortest paths between odd-degree vertices, where the edge-weights are regarded as distances. If the edges of a path between two odd-degree vertices are duplicated, then the degrees of those two vertices become even and the parity of the degree of each internal vertex on the path remains the same.

**Example 6.2.5:** The next few figures illustrate the application of Algorithm 6.2.2 to the weighted graph  $G$  in Figure 6.2.3.



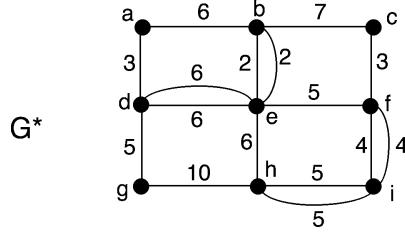
**Figure 6.2.3** Weighted graph for the Chinese Postman Problem.

Figure 6.2.4 shows the weighted complete graph on the odd-degree vertices in  $G$ . For instance, the shortest path between odd-degree vertices  $b$  and  $d$  has length 8. The minimum-weight perfect matching is shown in bold.



**Figure 6.2.4** Minimum-weight perfect matching of complete graph  $K$ .

Each edge in the perfect matching obtained in Algorithm 6.2.2 represents a path in  $G$ . The edges on this path are the ones that are duplicated to obtain the eulerian graph  $G^*$ , shown in Figure 6.2.5.



**Figure 6.2.5** Duplicating edges to obtain an eulerian graph  $G^*$ .

Finally, an eulerian tour for graph  $G^*$  is an optimal postman tour for  $G$ . One such tour is given by the vertex sequence

$$\langle a, b, c, f, e, b, e, d, e, h, i, f, i, h, g, d, a \rangle$$

All but one of the steps of Algorithm 6.2.2 involve algorithms that have already been discussed. The exception is the step that finds an optimal perfect matching of the complete graph  $K$ . This problem can be transformed into the problem of finding an optimal perfect matching of an associated complete bipartite graph. Methods developed in Chapter 13 lead to a polynomial-time algorithm for this type of problem.

The optimality of the postman tour follows from having chosen a minimum-weight perfect matching of edges that correspond to shortest paths in  $G$ . (See Exercises.)

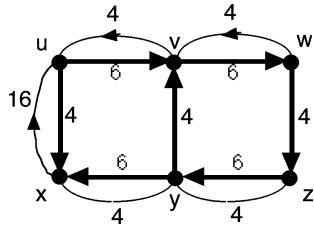
**Remark:** Edmonds and Johnson's polynomial-time solution applies equally well to the digraph version of the Chinese Postman Problem, but for *mixed graphs* (having both undirected and directed edges), the problem becomes NP-hard ([Pa76]). In their same paper, Edmonds and Johnson gave an approximate algorithm for the mixed-graph problem, and G. Frederickson [Fr79] showed that the solution obtained is never worse than twice the optimum. He also proposed some modifications that improved the worst-case performance.

### Other Eulerian-Type Applications

Variations of the Chinese Postman Problem arise in numerous applications. These include garbage collection and street sweeping; snow-plowing; line-painting down the center of each street; police-car patrolling; census taking; and computer-driven plotting of a network.

**Application 6.2.2 Street Sweeping:** Suppose that the digraph in Figure 6.2.6 below represents a network of one-way streets, with the bold arcs representing streets to be swept. Each edge-weight gives the time required to traverse that street *without* sweeping (i.e., deadheading the street). The time required to sweep a street is estimated as twice the deadheading time. What route minimizes the total time to sweep all the required streets, starting and ending at vertex  $z$ ?

This is a digraph variation of the Chinese Postman Problem studied by Tucker and Bodin [TuBo83]. As in the postman problem for undirected graphs, if the subgraph to be swept (called the *sweep subgraph*) is eulerian, then an eulerian tour will be an optimal sweeping tour. Otherwise, a subset of arcs must be added to the sweep subgraph so that the resulting digraph is eulerian; furthermore, the sum of the deadheading

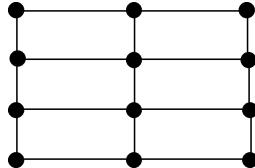


**Figure 6.2.6** Edges in bold represent streets to be swept.

times of the additional arcs should be minimum. Adapting Edmonds and Johnson's algorithm, Tucker and Bodin modeled the problem of finding the minimum-weight set of deadheaded arcs as a special case of the *minimum-cost network flow problem* (network flows are discussed in Chapter 13). For the example here, it turns out that the trail given by  $\langle z, y, x, y, v, u, v, u, x, y, v, w, z \rangle$  is an optimal route for the sweeper. The total sweeping time is 72, and the total deadheading time is 20.

They also studied the more complex problem of having to route a fleet of sweepers for a large-scale network (they considered the streets of New York City). Their approach was to find an optimal postman tour and then to partition the edges of the solution into subtours, one for each sweeper.

**Application 6.2.3 Mechanical Plotters:** Suppose that a mechanical plotter is to plot several thousand copies of the grid shown in Figure 6.2.7, and suppose that it takes twice as long to plot a horizontal edge as it takes to plot a vertical edge. The problem of routing the plotter so that the total time is minimized can be modeled as a postman problem [ReTa81]. (See Exercises).

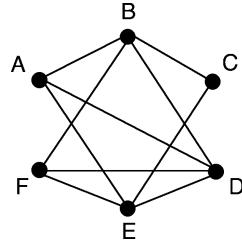


**Figure 6.2.7**

**Application 6.2.4 Sequencing Two-Person Conferences:** Suppose certain pairs in a department of six people,  $\{A, B, C, D, E, F\}$ , must meet privately in a single available conference room. The matrix below indicates with a 1 each pair that must meet. Is it possible to sequence the two-person conferences so that one of the participants in each conference (except the last one) also participates in the next conference, but no one participates in three consecutive conferences?

$$\begin{array}{cccccc}
 & A & B & C & D & E & F \\
 A & - & 1 & 0 & 1 & 1 & 0 \\
 B & 1 & - & 1 & 1 & 0 & 1 \\
 C & 0 & 1 & - & 0 & 1 & 0 \\
 D & 1 & 1 & 0 & - & 1 & 1 \\
 E & 1 & 0 & 1 & 1 & - & 1 \\
 F & 0 & 1 & 0 & 1 & 1 & -
 \end{array}$$

To solve this problem, first create a graph  $G$  with a vertex for each department member and an edge for each two-person conference (thus, the matrix is the adjacency matrix of graph  $G$ ). Figure 6.2.8 shows this graph.



**Figure 6.2.8** Graph model for two-person conferences.

Then an open eulerian trail in  $G$  will correspond to a sequencing of conferences that meets the conditions. Since the graph has exactly two odd-degree vertices, Theorem 6.1.1 implies that the sequencing is possible, and a modified version of Algorithm 6.1.1 can be used to construct it.

**Application 6.2.5 Determining an RNA Chain From Its Fragments:** In an RNA (ribonucleic acid) chain, such as CGAGUGUACGAA, each link is one of four possible *nucleotides*: adenine (A), cytosine (C), guanine (G), or uracil (U). In trying to identify the RNA chain in an unknown sample, present technology does not permit direct identification of long chains. Our description of the method of fragmentation and sub-fragmentation of a long RNA chain into identifiable subchains is adapted from the work of George Hutchinson [Hu69].

**Fragmentation:** Fortunately, there are available two types of enzymes that can be used to break an RNA chain into identifiable subchains. A ***G-enzyme*** breaks an RNA chain after each G-link. A ***UC-enzyme*** breaks an RNA chain after each U-link and after each C-link. The resulting fragments of the original RNA chain are called ***G-fragments*** and ***UC-fragments***, according to which enzyme created them.

An ***abnormal G-fragment*** is a G-fragment that does not end with a G nucleotide. Abnormal fragments can occur only at the end of a chain, because the only place the G-enzyme splits an RNA-chain is after a G-link. Similarly, an ***abnormal UC-fragment*** is a UC-fragment that does not end with a U or a C. In the description here, an abnormal fragment is indicated by underscoring its final link.

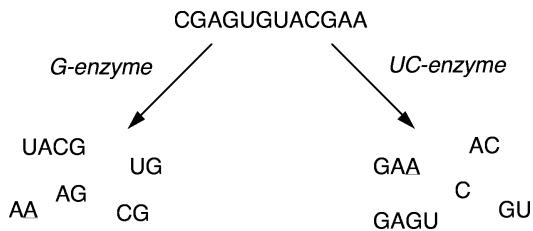
**Illustration:** If a sample of the chain CGAGUGUACGAA is subjected to the G-enzyme, then the result is the following ***G-set*** (i.e., of G-fragments):

CG, AG, UG, UACG, AA

If another sample of the same chain is subjected to the UC-enzyme, then the result is the following ***UC-set***:

C, GAGU, GU, AC, GAA

Figure 6.2.9 below illustrates the two enzymatic fragmentations of the given RNA chain.



**Figure 6.2.9** RNA-chain fragmentation under G-enzyme and UC-enzyme.

**Problem:** Given the unordered set of G-fragments and the unordered set of UC-fragments from the same RNA-chain, reconstruct the RNA-chain, if unique; otherwise, specify the complete set of possible RNA-chains from which the G-set and the UC-set could have arisen.

**Small-scale Approach:** The abnormal G-fragment AA must go last in any plausible reconstruction of the RNA-chain from the G-fragments. Since there are four other fragments, there is a **G-list** of  $24 = 4!$  possible chains (all cases of a permutation of the four normal G-fragments followed by AA) from which they might have arisen, starting alphabetically with AGCGUACGUGAA.

Similarly, the abnormal UC-fragment GAA must be last, and thus, there is a **UC-list** of  $24 = 4!$  possible chains from which these fragments could have arisen, starting alphabetically with ACCGAGUGUGGAA.

Any chain that appears in both lists is a candidate for the original RNA chain. One could sort both the G-list of 24 possible reconstructions and the UC-list of 24 possible reconstructions into alphabetical order, and then scan the two lists to find all chains that appear in both.

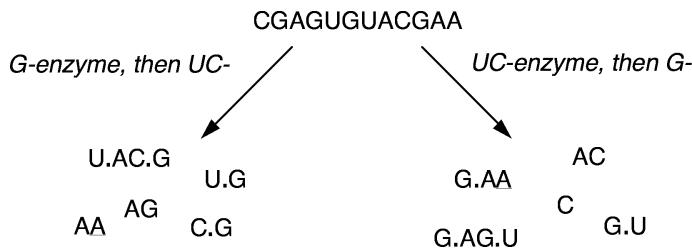
However, if a long RNA chain had 71 G-fragments and 82 UC-fragments, then there would be  $70! \approx 10^{100}$  possible chains in the G-list, and  $81! \approx 10^{118}$  in the UC-list. There is no feasible way to scan lists of such length.

Hutchinson's strategy is to construct an eulerian digraph whose arcs are labeled by fragments, such that each eulerian tour corresponds to an RNA chain that could be the unknown sample. Some additional terminology is needed before describing his construction.

**Subfragmentation:** A **subfragment** is a nonempty subchain that results either when a UC-enzyme is applied to a G-fragment or when a G-enzyme is applied to a UC-fragment. A fragment is **irreducible** if it consists of a single subfragment. A subfragment within a G-fragment or within a UC-fragment is **internal** if it has subfragments to its left and right. Figure 6.2.10 below illustrates the two enzymatic subfragmentations of the given RNA chain. The G-fragments appear on the left, the UC-fragments are on the right, and dots are used as separators between subfragments.

From Figure 6.2.10, it is clear that the irreducible G-fragments are AG and AA and that the irreducible UC-fragments are C and AC.

Also, the internal subfragment AC in the G-set is an irreducible UC-fragment, and the internal subfragment AG in the UC-set is an irreducible G-fragment. Moreover, the only irreducible fragments in either set that do not appear as internal subfragments in the other set are the first and last subfragments of the entire chain. Both of these



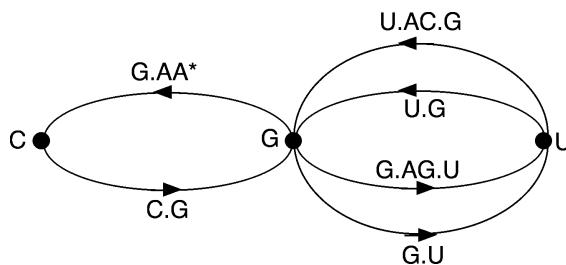
**Figure 6.2.10 RNA-chain subfragmentation under G-enzyme and UC-enzyme.**

properties hold in general for any chain that has at least two G-fragments and at least two UC-fragments (see Exercises). It implies for the example that the chain must begin with C and end with AA.

**Constructing the eulerian digraph:** The vertices of the digraph correspond to a subset of the subfragments, and each vertex is labeled with a different subfragment. Each arc corresponds to a fragment and is labeled by that fragment. The correspondence is prescribed as follows:

1. Identify the first subfragment of the entire RNA chain, and draw a vertex labeled with that subfragment. (In the example, this is subfragment C.)
2. Identify the longest abnormal fragment, and draw a vertex labeled with the first subfragment of that abnormal fragment. (In the example, this is subfragment G.)
3. Identify all normal fragments that contain at least two subfragments.
4. Draw a vertex for each subfragment that appears at the beginning or at the end of at least one of the normal fragments identified in step 3.
5. For each fragment  $F$  identified in step 3, draw an arc from the vertex labeled with the first subfragment of  $F$  to the vertex labeled with the last subfragment of  $F$ ; label the arc with  $F$ .
6. Draw an arc from the vertex drawn in step 2 to the vertex drawn in step 1, and label this arc with the longest abnormal fragment. Append an asterisk (\*) to that label to indicate that it appears at the end of the entire chain and that it is to be the last arc in any corresponding eulerian tour.

**Illustration:** The digraph for the example is shown in Figure 6.2.11.



**Figure 6.2.11 Digraph for determining the possible RNA chains.**

**Generating an RNA chain from a given eulerian tour:**

1. Starting at the vertex labeled with the subfragment known to be the first one of the entire chain, write down the entire fragment that labels the first arc of the eulerian tour.
2. For each subsequent arc, including the last one, write down the fragment labeling that arc, excluding that fragment's first subfragment.

**Illustration:** There are four different eulerian tours in the digraph of Figure 6.2.11, and their corresponding RNA chains are:

CGAGUGUACGAA  
CGAGUACGUGAA  
CGUACGAGUGAA  
CGUGAGUACGAA

It is easy to verify that each of these chains will give rise to the same set of G-fragments and UC-fragments.

A number of details have been omitted. For example, why will the construction always produce an eulerian digraph, and why are the eulerian tours in one-to-one correspondence with the candidate RNA chains? These and other details may be found in Hutchinson's paper.

**Application 6.2.6 Information Encoding:** In the last application, an RNA chain is treated like a string, where the individual nucleotides are its *letters*. More generally and under certain assumptions, the information carried by molecules depends only on the number of letters of each type and the frequency of each letter pair. J. Hutchinson and H. Wilf [HuWi75] considered a purely combinatorial problem that arises naturally from this perspective.

**NOTATION:** For a given string, let  $f_i$  be the number of occurrences of  $i$ , and let  $f_{ij}$  be the number of times  $j$  follows  $i$ .

**Illustration:** For the string 12321212, the nonzero values are:

$$f_1 = 3; f_2 = 4; f_3 = 1; f_{12} = 3; f_{21} = 2; f_{23} = f_{32} = 1$$

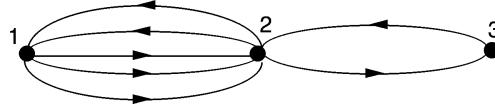
**Problem:** For a given set of  $f_i$ 's and  $f_{ij}$ 's,  $i, j = 1 \dots, n$ , how many different strings satisfy these prescribed frequency requirements?

It is also assumed that  $f_{ii} = 0$  for all  $i$ , since a solution to the original problem can easily be deduced from this reduced problem (see [HuWi75]).

**Strategy:** Draw the digraph  $G$  whose adjacency matrix is  $(f_{ij})$ , and determine the number of eulerian trails.

**Observation:** A string  $i_1 i_2 \dots i_l$  will meet the requirements if and only if it corresponds to an eulerian trail in  $G$ . The eulerian trail is closed or open, depending on whether  $i_1 = i_l$ .

**Illustration:** The digraph shown in Figure 6.2.12 has an adjacency matrix whose entries match the  $f_{ij}$ 's above. It is easy to check that the digraph contains an open eulerian trail from vertex 1 to vertex 2 and that the string 12321212 corresponds to the vertex sequence of one of the eulerian trails. Moreover, there are two other eulerian trails whose corresponding vertex sequences represent the only other strings satisfying the requirements.



**Figure 6.2.12** Digraph obtained from the  $f_{ij}$ 's.

The solution for the general case  $i_1 \neq i_l$  is outlined below. The analysis for the case  $i_1 = i_l$  is similar and left for the reader.

By Theorem 6.1.3, there are no solution-strings unless the corresponding digraph is connected (up to isolated vertices), and the  $f_i$ 's and  $f_{ij}$ 's satisfy the following conditions.

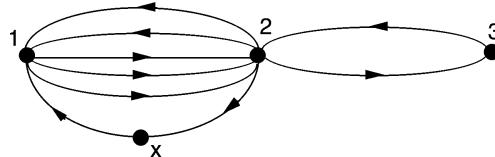
$$\sum_{k=1}^n f_{ik} = \begin{cases} \sum_{k=1}^n f_{ki} + 1, & \text{if } i = i_1 \\ \sum_{k=1}^n f_{ki} - 1, & \text{if } i = i_l \\ \sum_{k=1}^n f_{ki}, & \text{otherwise} \end{cases}$$

$$f_i = \begin{cases} \sum_{k=1}^n f_{ki} + 1, & \text{if } i = i_1 \\ \sum_{k=1}^n f_{ki}, & \text{otherwise} \end{cases}$$

Each open eulerian trail corresponds to a unique solution-string  $w$ , namely, the vertex sequence of the trail. Conversely, a solution-string  $w$  corresponds to the vertex sequence of many eulerian trails. In particular, if  $T$  is an eulerian trail corresponding to  $w$ , and  $i, j$  is fixed, then each permutation of the  $f_{ij}$  arcs from  $i$  to  $j$  results in a different eulerian trail corresponding to  $w$ . Thus,  $w$  corresponds to precisely  $\prod_{i \neq j} (f_{ij}!)$  different eulerian trails.

It remains to count the open eulerian trails of  $G$ . Let  $G_x$  be the digraph formed by adjoining a new vertex  $x$  to  $G$ , with arcs from  $x$  to  $i_1$  and  $i_l$  to  $x$ .

**Illustration:** Figure 6.2.13 shows the digraph  $G_x$  for the example.



**Figure 6.2.13** The eulerian digraph  $G_x$ .

There is a one-to-one correspondence between eulerian trails of  $G$  and eulerian tours of  $G_x$ . The number of eulerian tours in a directed graph has been determined by de Bruijn and Ehrenfest [DeEh51] and by Smith and Tutte [SmTu41], and for  $G_x$ , is given by

$$\prod_{i=1}^n (f_i - 1)! \det([a_{ij}])$$

where

$$a_{ij} = \begin{cases} f_i, & \text{if } i = j \\ -f_{ij}, & \text{otherwise} \end{cases}$$

Thus, the total number of solution-strings is

$$N = \left\{ \prod_{i=1}^n (f_i - 1)! \right\} \left\{ \prod_{i,j=1}^n (f_{ij}!) \right\}^{-1} \det([a_{ij}])$$

For the example string, this yields

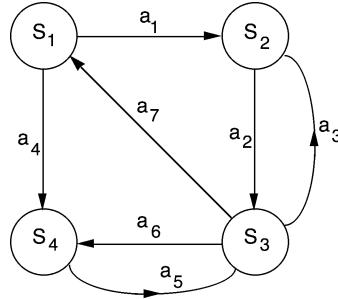
$$2!3!0!\{3!2!\}^{-1} \det \begin{pmatrix} 3 & -3 & 0 \\ -2 & 4 & -1 \\ 0 & -1 & 1 \end{pmatrix} = 3$$

The three solution-strings are: 12321212; 12121232; 12123212.

### Eulerian Digraphs and Software Testing<sup>†</sup>

In §1.6, we observed that a directed postman tour can be used in designing test input sequences to a computer program, that involve invoking all the transitions (actions) of a finite-state model of the program. Here, we give a small example to illustrate that software-testing method, after which, we consider the related problem of testing combinations of actions.

**Example 6.2.6:** Suppose that the digraph shown is the behavioral model of some computer program to be tested. The states,  $\{S_1, S_2, S_3, S_4\}$ , and actions,  $\{a_1, a_2, \dots, a_7\}$ , of the program are represented by the vertices and arcs, respectively. A shortest directed postman tour is given by the arc sequence  $\langle a_1, a_2, a_3, a_2, a_6, a_5, a_5, a_7, a_4, a_5, a_7 \rangle$ . This was obtained by constructing a directed eulerian tour of the *eulerized* digraph that results from duplicating the arcs  $a_2$ ,  $a_5$ , and  $a_7$  of the original digraph.



**Figure 6.2.14** Finite-state model.

Now suppose we want to test all pairs of consecutive transitions that are possible. In the digraph model, each pair of consecutive transitions corresponds to a pair of arcs, say  $e_1$  and  $e_2$ , such that  $\text{head}(e_1) = \text{tail}(e_2)$ . For instance, in the digraph in Figure 6.2.14, we seek a shortest closed directed walk that includes  $a_3a_2$ ,  $a_2a_3$ ,  $a_2a_6$ , etc.

---

<sup>†</sup> The material in this subsection is adapted from H. Robinson, [Ro99].

**TERMINOLOGY:** For a given finite-state machine (digraph), a sequence of arcs that includes all pairs of adjacent arcs is called a **length-2 switch-cover** for the digraph.

Our solution uses the following digraph version of a line graph (§1.2).

**DEFINITION:** The **line graph** of a digraph  $G$  is the digraph  $L(G)$  whose vertex-set corresponds to the arc-set of  $G$ ; an arc in  $L(G)$  is directed from vertex  $e_1$  to vertex  $e_2$  if, in  $G$ ,  $\text{head}(e_1) = \text{tail}(e_2)$ . (See Figure 6.2.15.)

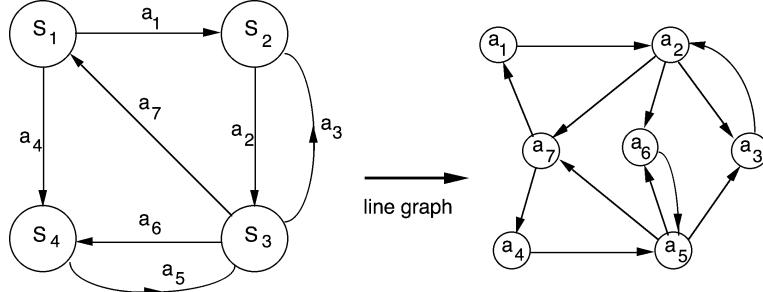
A length-2 switch-cover is constructed as follows:

1. Create the line graph  $L(G)$  of the original digraph  $G$ .
2. Construct an optimal directed postman tour of the line graph  $L(G)$ .
3. The vertex sequence of this tour corresponds to the desired arc sequence in the original digraph.

**Example 6.2.7:** Figure 6.2.15 shows the digraph from Example 6.2.6 on the left and its line graph on the right. An optimal postman tour of the line graph is given by the vertex sequence

$$\langle a_1, a_2, a_3, a_2, a_6, a_5, a_3, a_2, a_7, a_4, a_5, a_6, a_5, a_7 \rangle$$

Notice that this sequence of arcs in the original digraph is a length-2 switch-cover.



**Figure 6.2.15** Finite-state model and its line graph.

### EXERCISES for Section 6.2

6.2.1<sup>s</sup> Construct a  $(2, 4)$ -deBruijn sequence that is different from the one obtained in Application 6.2.1 by finding a different eulerian tour of  $D_{2,4}$ .

6.2.2 Draw the  $(2, 3)$ -deBruijn digraph and use it to construct two different  $(2, 3)$ -deBruijn sequences.

6.2.3 Which vertices of the deBruijn digraph have self-loops? Justify your answer.

6.2.4 Find an appropriate extension of the line-graph definition (§1.2) to digraphs, and show that the line graph of the deBruijn digraph  $D_{2,3}$  is the deBruijn digraph  $D_{2,4}$ .

6.2.5<sup>s</sup> Prove that  $L(D_{2,n}) = D_{2,n+1}$ .

6.2.6 Give an alternative proof of Proposition 6.2.1 using line graphs.

6.2.7 A  $(p, n)$ -deBruijn sequence is a string of length  $p^n$ , containing as substrings (allowing wraparound) all possible sequences of length  $n$  of characters drawn from a  $p$ -element set. Generalize the procedure outlined in this section, and use it to construct the  $(3, 2)$ -deBruijn digraph  $D_{3,2}$  and a  $(3, 2)$ -deBruijn sequence.

**6.2.8<sup>s</sup>** Construct the deBruijn digraph  $D_{3,3}$  and use it to construct a  $(3, 3)$ -deBruijn sequence.

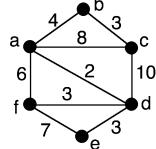
**6.2.9** Show that the line graph  $L(D_{3,2})$  of the deBruijn digraph  $D_{3,2}$  is the deBruijn digraph  $D_{3,3}$ .

**6.2.10** Prove that  $L(D_{p,n}) = D_{p,n+1}$ .

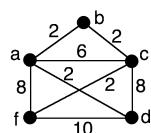
**6.2.11** Which vertices of the deBruijn digraph  $D_{2,n}$  have self-loops? Justify your answer.

*In Exercises 6.2.12 through 6.2.15, apply Algorithm 6.2.2 to find a minimum-weight postman tour for the given weighted graph. Determine whether the solution is unique.*

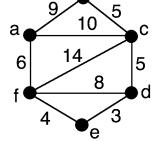
**6.2.12**



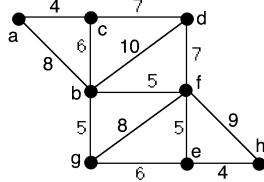
**6.2.13**



**6.2.14<sup>s</sup>**



**6.2.15**



*Exercises 6.2.16 through 6.2.18 refer to Algorithm 6.2.2.*

**6.2.16<sup>s</sup>** Prove that a perfect matching of  $K$  is guaranteed to exist.

**6.2.17** Prove that  $G^*$  is eulerian.

**6.2.18** Show how the optimality of the postman tour follows from having chosen a minimum-weight perfect matching of edges that correspond to shortest paths in  $G$ .

**6.2.19** Use a modified version of Algorithm 6.2.2 to solve the example problem in Application 6.2.3.

**6.2.20** Solve the problem posed in Application 6.2.4.

*In Exercises 6.2.21 through 6.2.24, use the method of Application 6.2.5 to find an RNA chain whose G- and UC-fragments are as given.*

**6.2.21** G-fragments: CCG, G, UCCG, AAAG;  
UC-fragments: GGAAAG, GU, C, C, C, C.

**6.2.22<sup>s</sup>** G-fragments: CUG, CAAG, G, UC;  
UC-fragments: C, C, U, AAGC, GGU.

**6.2.23** G-fragments: G, UCG, G, G, UU;  
UC-fragments: GGGU, U, GU, C.

**6.2.24** G-fragments: G, G, CC, CUG, G;  
UC-fragments: GGGU, U, C, GC.

**6.2.25** Referring to Application 6.2.5, construct a chain that contains no internal subfragments and is not irreducible.

6.2.26 Referring to Application 6.2.5, show that for any chain,

a. every interior subfragment also appears as an irreducible fragment.

b. the only irreducible fragments that do not appear as internal subfragments are the first and last subfragments of the entire chain.

*In Exercises 6.2.27 through 6.2.30, use the observation in Application 6.2.6 to determine whether there is a sequence of the numbers 1, 2, and 3 that satisfies the given frequency and adjacency matrix ( $f_{ij}$ ). If there is such a sequence, use an appropriate version of Algorithm 6.2.1 to find one.*

$$6.2.27^s \quad f_1 = 3; f_2 = 5; f_3 = 3.$$

$$6.2.28 \quad f_1 = 4; f_2 = 4; f_3 = 4.$$

$$\begin{pmatrix} 0 & 3 & 0 \\ 2 & 0 & 1 \\ 1 & 2 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 1 & 3 \\ 3 & 0 & 1 \\ 2 & 1 & 0 \end{pmatrix}$$

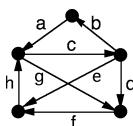
$$6.2.29 \quad f_1 = 5; f_2 = 4; f_3 = 3.$$

$$6.2.30 \quad f_1 = 3; f_2 = 3; f_3 = 4; f_4 = 4.$$

$$\begin{pmatrix} 0 & 3 & 1 \\ 3 & 0 & 1 \\ 2 & 1 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 0 & 3 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 3 \\ 1 & 2 & 0 & 0 \end{pmatrix}$$

6.2.31 Apply the method illustrated in Example 6.2.7 to find a closed directed walk for the digraph shown, such that every pair of adjacent arcs is included in the arc sequence of the walk.



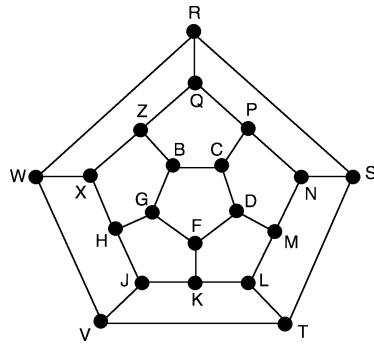
## 6.3 HAMILTONIAN PATHS AND CYCLES

**DEFINITION:** A **hamiltonian path** (*cycle*) of a graph is a path (*cycle*) that contains all the vertices. For digraphs, the hamiltonian path or cycle is directed.

**DEFINITION:** A **hamiltonian graph** is a graph that has a hamiltonian cycle.

**Remark:** Adding or deleting self-loops or extra adjacencies between two vertices does not change a graph from hamiltonian to non-hamiltonian.

Among the many contributions of the Irish mathematician Sir William Rowan Hamilton (1805–1865) was some of the earliest study of hamiltonian graphs. A byproduct of this work was his invention of a puzzle known as the *Icosian Game*. One of several aspects of the Icosian Game was to find a hamiltonian cycle on the dodecahedral graph, depicted in Figure 6.3.1 below.



**Figure 6.3.1** Dodecahedral graph for the Icosian Game.

Since a hamiltonian cycle is analogous to an eulerian tour, one might hope for a characterization of hamiltonian graphs as convenient as the even-degree characterization of eulerian graphs. No such characterization is known, nor is there a quick way of determining whether a given graph is hamiltonian. In fact, the problem is NP-complete, making it extremely unlikely that there exists a polynomial algorithm that can answer the question “Is  $G$  hamiltonian?” for an arbitrary graph  $G$ .

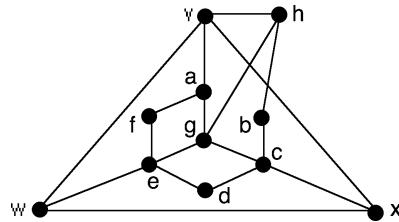
The absence of a polynomial-time algorithm that works for all graphs does not mean the situation is hopeless. There are sufficient conditions for a graph to be hamiltonian that apply to large classes of graphs. Also, there are some basic rules that help show certain graphs are not hamiltonian.

### Showing That a Graph Is Not Hamiltonian

The following rules are based on the simple observation that any hamiltonian cycle must contain exactly two edges incident on each vertex. The strategy for applying these rules is to begin a construction of a hamiltonian cycle and show at some point during the construction that it is impossible to proceed any further. The path that this kind of argument takes depends on the particular graph. There are three rules:

1. If a vertex  $v$  has degree 2, then both of its incident edges must be part of any hamiltonian cycle.
2. During the construction of a hamiltonian cycle, no cycle can be formed until all the vertices have been visited.
3. If during the construction of a hamiltonian cycle two of the edges incident on a vertex  $v$  are shown to be required, then all other incident edges can be deleted.

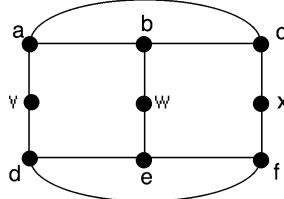
**Example 6.3.1:** The graph in Figure 6.3.2 is not hamiltonian.



**Figure 6.3.2** A non-hamiltonian graph.

**Proof:** Rule 1 applied to vertices  $b$ ,  $d$ , and  $f$  implies that edges  $bh$ ,  $bc$ ,  $dc$ ,  $de$ ,  $fe$ , and  $fa$  must be on every hamiltonian cycle. Rule 3 applied to vertices  $c$ , and  $e$  eliminates edges  $cg$ ,  $cx$ ,  $eg$ , and  $ew$ . In the resulting graph, Rule 1 implies that edges  $wv$ ,  $vx$ , and  $xw$  must be on every hamiltonian cycle. But these form a 3-cycle, which violates Rule 2. This implies the existence of two disjoint subcycles (the inner hexagon and the outer triangle), which violates Rule 2.  $\diamond$

**Example 6.3.2:** The graph in Figure 6.3.3 is not hamiltonian.



**Figure 6.3.3** A non-hamiltonian graph.

**Proof:** Rule 1 applied to vertices  $v$ ,  $w$ , and  $x$  implies that all six vertical edges must be part of any hamiltonian cycle. Rules 1 and 3 applied to vertex  $b$  imply that exactly one of the edges  $ab$  and  $bc$  is part of the hamiltonian cycle. If  $ab$  is on the cycle and  $bc$  is not, then Rule 3 applied to vertex  $a$  implies that  $ac$  is not on the cycle. But then  $cx$  is the only edge on the cycle that is incident on  $c$ . Thus, by Rule 1, no hamiltonian cycle exists in this case. By symmetry, a similar contradiction results if  $bc$  is on the cycle and  $ab$  is not.  $\diamond$

### Sufficient Conditions for a Graph to be Hamiltonian

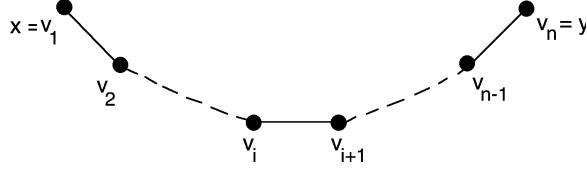
The next two results lend precision to the notion that the more edges a simple graph has, the more likely it is hamiltonian. The chronologically earlier of the two results, first proved by G.A. Dirac in 1952 ([Di52]), can now be deduced from the more recent result proved by O. Ore ([Or60]).

**Theorem 6.3.1 [Ore, 1960].** *Let  $G$  be a simple  $n$ -vertex graph, where  $n \geq 3$ , such that  $\deg(x) + \deg(y) \geq n$  for each pair of non-adjacent vertices  $x$  and  $y$ . Then  $G$  is hamiltonian.*

**Proof:** By way of contradiction, assume that the theorem is false, and let  $G$  be a maximal counterexample. That is,  $G$  is non-hamiltonian and satisfies the conditions of the theorem, and the addition of any edge joining two non-adjacent vertices of  $G$  results in a hamiltonian graph.

Let  $x$  and  $y$  be two non-adjacent vertices of  $G$  ( $G$  is not complete, since  $n \geq 3$ ). To reach a contradiction, it suffices to show that  $\deg(x) + \deg(y) \leq n - 1$ .

Since graph  $G + xy$  contains a hamiltonian cycle,  $G$  contains a hamiltonian path whose endpoints are  $x$  and  $y$ . Let  $\langle x = v_1, v_2, \dots, v_n = y \rangle$  be such a path (see Figure 6.3.4).

Figure 6.3.4 A hamiltonian path in  $G$ .

For each  $i = 2, \dots, n - 1$ , at least one of the pairs  $v_1, v_{i+1}$  and  $v_i, v_n$  is non-adjacent, since otherwise,  $\langle v_1, v_2, \dots, v_i, v_n, v_{n-1}, \dots, v_{i+1}, v_1 \rangle$  would be a hamiltonian cycle in  $G$  (see Figure 6.3.5). This means that if  $(a_{i,j})$  is the adjacency matrix for  $G$ , then  $a_{1,i+1} + a_{i,n} \leq 1$ , for  $i = 2, \dots, n - 2$ .

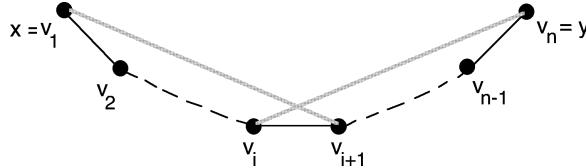


Figure 6.3.5

Thus,

$$\begin{aligned} \deg(x) + \deg(y) &= \sum_{i=2}^{n-1} a_{1,i} + \sum_{i=2}^{n-1} a_{i,n} \\ &= a_{1,2} + \sum_{i=3}^{n-1} a_{1,i} + \sum_{i=2}^{n-2} a_{i,n} + a_{n-1,n} \\ &= 1 + \sum_{i=2}^{n-2} a_{1,i+1} + \sum_{i=2}^{n-2} a_{i,n} + 1 \\ &= 2 + \sum_{i=2}^{n-2} (a_{1,i+1} + a_{i,n}) \\ &\leq 2 + n - 3 = n - 1 \end{aligned}$$

which establishes the desired contradiction.  $\diamond$

**Corollary 6.3.2 [Dirac, 1952].** Let  $G$  be a simple  $n$ -vertex graph, where  $n \geq 3$ , such that  $\deg(v) \geq \frac{n}{2}$  for each vertex  $v$ . Then  $G$  is hamiltonian.  $\diamond$

The following two theorems are digraph versions of the results of Ore and Dirac. The proof of the first is considerably more difficult than its undirected-graph counterpart and may be found in [Wo72].

**Theorem 6.3.3.** Let  $D$  be a simple  $n$ -vertex digraph. Suppose that for every pair of vertices  $v$  and  $w$  for which there is no arc from  $v$  to  $w$ ,  $\text{outdeg}(v) + \text{indeg}(w) \geq n$ . Then  $D$  is hamiltonian.  $\diamond ([Wo72])$

**Corollary 6.3.4.** Let  $D$  be a simple  $n$ -vertex digraph such that for every vertex  $v$ ,  $\text{outdeg}(v) \geq \frac{n}{2}$  and  $\text{indeg}(v) \geq \frac{n}{2}$ . Then  $D$  is hamiltonian.  $\diamond$

**EXERCISES for Section 6.3**

*In Exercises 6.3.1 through 6.3.4, determine which graphs in the given graph family are hamiltonian.*

- 6.3.1 The complete graph  $K_n$ .      6.3.2<sup>s</sup> The complete bipartite graph  $K_{m,n}$ .  
 6.3.3 The  $n$ -vertex wheel  $W_n$ .      6.3.4 Trees on  $n$  vertices.

- 6.3.5 Which platonic graphs are hamiltonian?

*In Exercises 6.3.6 through 6.3.10, draw the specified graph or prove that it does not exist.*

- 6.3.6<sup>s</sup> An 8-vertex simple graph with more than 8 edges that is both eulerian and hamiltonian.

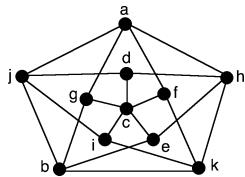
- 6.3.7 An 8-vertex simple graph with more than 8 edges that is eulerian but not hamiltonian.

- 6.3.8 An 8-vertex simple graph with more than 8 edges that is hamiltonian but not eulerian.

- 6.3.9 An 8-vertex simple hamiltonian graph that does not satisfy the conditions of Ore's theorem.

- 6.3.10 A 6-vertex simple graph with 10 edges that is not hamiltonian.

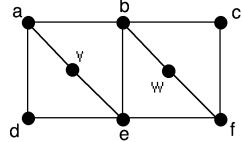
- 6.3.11<sup>s</sup> Prove that the **Grötzsch graph**, shown below, is hamiltonian.



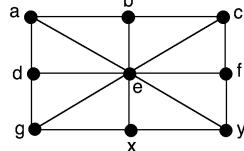
- 6.3.12 Prove that a bipartite graph that is hamiltonian must have an even number of vertices.

*In Exercises 6.3.13 through 6.3.18, either construct a hamiltonian cycle in the given graph, or prove that the graph is not hamiltonian.*

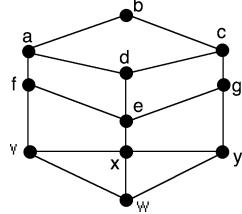
- 6.3.13<sup>s</sup>



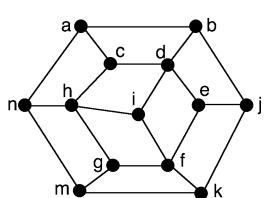
- 6.3.14



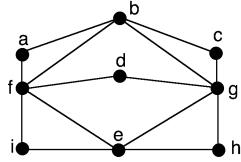
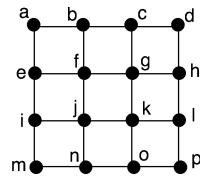
- 6.3.15



- 6.3.16



6.3.17

6.3.18<sup>s</sup>

6.3.19 Show that the Petersen graph is not hamiltonian.

6.3.20 One version of the Icosian Game was to find a hamiltonian cycle that started with a given five letters (see Figure 6.3.1). Find all hamiltonian cycles that begin with the letters BCPNM.

6.3.21<sup>s</sup> Characterize the graphs for which the following properties hold.

- a. a depth-first search produces a hamiltonian path, regardless of the starting vertex.
- b. a breadth-first search produces a hamiltonian path, regardless of the starting vertex.

**DEFINITION:** A **knight's tour** of a chessboard is a sequence of knight moves that visits each square exactly once and returns to its starting square with one more move.

6.3.22 The problem of determining whether a knight's tour exists actually predates the work of Hamilton. Pose the knight's tour problem as one of determining whether a certain graph is hamiltonian.

6.3.23 Show that if the requirement in Dirac's result (Corollary 6.3) is relaxed to " $\deg(v) \geq \frac{n-1}{2}$ " (from " $\deg(v) \geq \frac{n}{2}$ "), then the assertion of the theorem is false.6.3.24<sup>s</sup> Show that the sufficient condition in Ore's Theorem 6.2 is not a necessary condition, by giving an example of a hamiltonian graph that does not satisfy the conditions of the theorem.6.3.25 Consider the complete graph  $K_n$  with vertices labeled  $1, 2, \dots, n$ .

- a. Find the number of different hamiltonian cycles. Two cycles that differ only in where they start and end should be counted as the same cycle.
- b. Find the number of hamiltonian paths from vertex 1 to vertex 2.
- c. Find the number of open hamiltonian paths.

6.3.26 Prove or disprove each of the following statements.

- a. There exists a 6-vertex eulerian graph that is not hamiltonian.
- b. There exists a 6-vertex hamiltonian graph that is not eulerian.

6.3.27 Show that for any odd prime  $n$ , the edges of  $K_n$  can be partitioned into  $\frac{n-1}{2}$  edge-disjoint hamiltonian cycles. (Hint: Arrange the vertices  $1, 2, \dots, n$  around a circle.)

6.3.28 Suppose that 19 world leaders are to dine together at a circular table during a conference. It is desired that each leader sit next to a pair of different leaders for each dinner. How many consecutive dinners can be scheduled? (Hint: See the preceding exercise.)

6.3.29 A mouse eats its way through a  $3 \times 3 \times 3$  cube of cheese by tunneling through all of the 27  $1 \times 1 \times 1$  subcubes. If the mouse starts at one corner and always moves on to an uneaten subcube, can it finish at the center of the cube?

## 6.4 GRAY CODES AND TRAVELING SALESMAN PROBLEMS

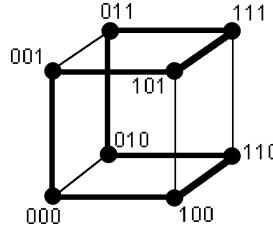
Initially, the word “code” suggested secrecy. But with the advances in digital techniques, now applied routinely to represent analog signals in digital form (e.g., music on compact disks), the word “code” has acquired much broader meaning.

**DEFINITION:** A **Gray code of order  $n$**  is an ordering of the  $2^n$  length- $n$  bitstrings such that consecutive bitstrings (and the first and last bitstring) differ in precisely one bit position.

**Example 6.4.1:** The sequence  $\langle 000, 100, 110, 010, 011, 111, 101, 001 \rangle$  is a Gray code of order 3.

**REVIEW FROM §1.2:** The  **$n$ -dimensional hypercube  $Q_n$**  is the graph whose vertex-set is the set of length- $n$  bitstrings, such that there is an edge between two vertices if and only if they differ in exactly one bit.

Thus, a Gray code of order  $n$  corresponds to a hamiltonian cycle in the hypercube graph  $Q_n$ . The hypercube  $Q_3$  is shown in Figure 6.4.1, and the edges drawn in bold form a hamiltonian cycle.



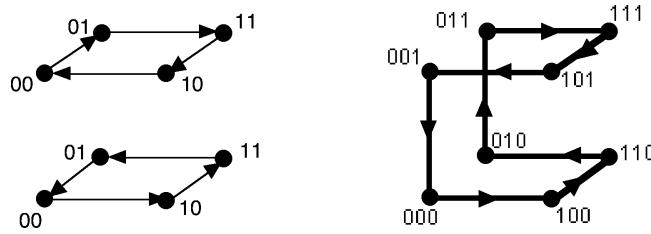
**Figure 6.4.1** A hamiltonian cycle in the hypercube  $Q_3$ .

The next result shows that  $Q_n$  is hamiltonian for all  $n$ , thereby proving that Gray codes of all orders exist. The proof relies on the following inductive construction of the hypercube. The  $(n + 1)$ -dimensional hypercube  $Q_{n+1}$  can be obtained from two copies of the  $n$ -dimensional hypercube  $Q_n$ , as follows: adjoin a 0 to the right of each  $n$ -bit sequence of one of the copies of  $Q_n$ , adjoin a 1 to the right of each sequence of the other copy, and join by an edge the two vertices labeled by corresponding sequences.

**Remark:** This construction is a special case of the *cartesian product* of two graphs, which is defined in §2.7.

A closer look at the hamiltonian cycle shown in Figure 6.4.1 previews the inductive step in the proof. Figure 6.4.2 suggests how a hamiltonian cycle in  $Q_3$  can be constructed from two oppositely oriented hamiltonian cycles in two copies of  $Q_2$ . The vertices of  $Q_3$  are obtained by adding a 0 or 1 as a third bit to the corresponding vertex of the bottom and top  $Q_2$  graphs, respectively.

Starting at 000, traverse all but the last edge of the hamiltonian cycle in the bottom  $Q_2$ ; traverse the vertical edge to get to the corresponding vertex in the top  $Q_2$  (011); then follow the top hamiltonian cycle in the opposite direction until reaching 001; finally, complete the hamiltonian cycle for  $Q_3$  by returning to 000 via the vertical edge.



**Figure 6.4.2** Gray code of order 3 from a Gray code of order 2.

**Theorem 6.4.1.** *The  $n$ -dimensional hypercube  $Q_n$  is hamiltonian for all  $n \geq 2$ .*

**Proof:**  $Q_2$  is a 4-cycle and, hence, hamiltonian. Assume for some  $n \geq 2$  that there exists a hamiltonian cycle  $\langle b_1, b_2, \dots, b_{2^n}, b_1 \rangle$  in  $Q_n$ . Then

$$\langle b_1 0, b_2 0, \dots, b_{2^n} 0, b_{2^n} 1, b_{2^n-1} 1, \dots, b_1 1, b_1 0 \rangle$$

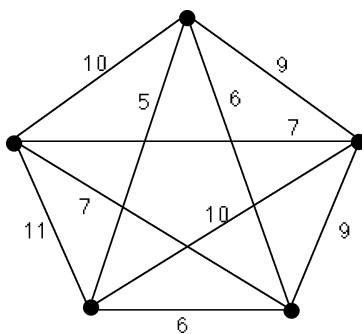
is a hamiltonian cycle in  $Q_{n+1}$ .  $\diamond$

#### Application 6.4.1 Transmitting Photographs from a Spacecraft:

A spacecraft transmits a picture back to earth using long sequences of numbers, where each number is a darkness value for one of the dots in the picture. The advantage of using a Gray code to encode the picture is that if an error from “cosmic noise” causes one binary digit in a sequence to be misread by the receiver, then the mistaken sequence will often be interpreted as a darkness value that is almost the same as the true darkness number. For example, if the Gray code given in Example 6.4.1 were used to encode the darkness values 1 through 8, respectively, and if 011 ( $= 5$ ) were transmitted, then the sequence that results from an error in the first or last bit (reading right to left) would be interpreted as a darkness value of 4 or 6, respectively.

#### Traveling Salesman Problem

One version of the **Traveling Salesman Problem (TSP)** is to minimize the total airfare for a traveling salesman who wants to make a tour of  $n$  cities, visiting each city exactly once before returning home. Figure 6.4.3 shows a weighted graph model for the problem; the vertices represent the cities and the edge-weights give the airfare between each pair of cities. A solution requires finding a minimum-weight hamiltonian cycle. The graph can be assumed to be complete by assigning arbitrarily large weight to edges that do not actually exist.



**Figure 6.4.3** Weighted graph for a TSP.

The TSP has a long history that has stimulated considerable research in *combinatorial optimization*. The earliest work related to the TSP dates back to 1759, when Euler published a solution to the Knight's Tour Problem (see Exercises). Other early efforts were made by A.T. Vandermonde (1771), T.P. Kirkman (1856), and of course W.R. Hamilton (1856).

The problem of finding a minimum-weight hamiltonian cycle appears to have been first posed as a TSP by H. Whitney in 1934. M. Flood of Rand Corporation recognized its importance in the context of the then young field of *operations research*, and in 1954, three of his colleagues at Rand, G.B. Dantzig, D.R. Fulkerson, and S.M. Johnson ([DaFuJo54]), achieved the first major breakthrough by finding a provably optimal tour of 49 cities (Washington, D.C. and the capitals of the 48 contiguous states). Their landmark paper used a combination of linear programming and graph theory, and it was probably the earliest application of what are now two of the standard tools in integer programming, *branch-and-bound* and *cutting planes*.

The next dramatic success occurred in 1980 with the publication by Crowder and Padberg of a provably optimal solution to a 318-city problem ([CrPa80]). To enumerate the problem's approximately  $10^{655}$  tours at the rate of 1 billion tours per second, it would take a computer  $10^{639}$  years. The Crowder-Padberg solution took about 6 minutes by computer, using a combination of branch-and-bound and *facet-defining inequalities*.

## Heuristics and Approximate Algorithms for the TSP

**DEFINITION:** A **heuristic** is a guideline that helps in choosing from among several possible alternatives for a decision step.

Heuristics are what human experts apply when it is difficult or impossible to evaluate every possibility. In chess, the stronger the player, the more effective that player's heuristics in eliminating all but a few of the possible moves, without evaluating each legal move. Of course, this has the risk of sometimes missing what may be the best move.

**DEFINITION:** A **heuristic algorithm** is an algorithm whose steps are guided by heuristics. In effect, the heuristic algorithm is forfeiting the guarantee of finding the best solution, so that it can terminate quickly.

Since the TSP is  $\mathcal{NP}$ -hard, there is a trade-off between heuristic algorithms that run quickly and those that guarantee finding an optimal solution. Time constraints in many applications usually force practitioners to opt for the former. An excellent survey and detailed analysis of heuristics are found in [LaLeKaSh85]. A few of the more commonly used ones are given here.

The simplest TSP heuristic is **nearest neighbor**. Its philosophy is one of short-sighted greed: from wherever you are, pick the cheapest way to go somewhere else. Thus, the nearest-neighbor algorithm is simply a depth-first traversal where ties are broken by choosing the edge of smallest weight.

**Algorithm 6.4.1:** Nearest Neighbor

*Input:* a weighted complete graph.

*Output:* a sequence of labeled vertices that forms a hamiltonian cycle.

Start at any vertex  $v$ .

Initialize  $l(v) = 0$ .

Initialize  $i = 0$ .

While there are unlabeled vertices

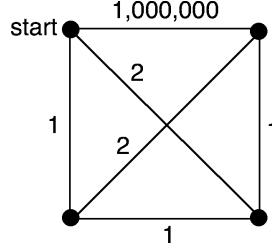
$i := i + 1$

Traverse the cheapest edge that joins  $v$  to an unlabeled vertex, say  $w$ .

Set  $l(w) = i$ .

$v := w$

As is typical of greedy algorithms, the nearest-neighbor heuristic is very fast, and it is easy to implement. The algorithm sometimes performs quite well; for instance, for the weighted graph in Figure 6.4.3, it produces the optimal solution if it starts at the top vertex. However, in general, it can produce arbitrarily bad (costly) hamiltonian cycles, as it does for the graph in Figure 6.4.4.



**Figure 6.4.4** The nearest-neighbor heuristic can be arbitrarily bad.

The following theorem shows that performance guarantees for approximate algorithms for the general TSP are extremely unlikely.

**Theorem 6.4.2.** *If there exists a polynomial-time approximate algorithm whose solution to every instance of the general TSP is never worse than some constant  $r$  times the optimum, then  $P = NP$ .*  $\diamond ([SaGo76])$

However, for TSPs satisfying the *triangle inequality*, there are such algorithms. But the algorithm based on the nearest-neighbor heuristic is not one of them, as Theorem 6.4.3 demonstrates.

**DEFINITION:** Let  $G$  be a weighted simple graph with vertices labeled  $1, 2, \dots, n$ , such that edge  $ij$  has weight  $c_{ij}$ . Then  $G$  is said to satisfy the **triangle inequality** if  $c_{ij} \leq c_{ik} + c_{kj}$  for all  $i, j$ , and  $k$ .

**Theorem 6.4.3.** *For every  $r > 1$ , there exists an instance of the TSP, obeying the triangle inequality, for which the solution obtained by the nearest-neighbor algorithm is at least  $r$  times the optimal value.*  $\diamond ([RoStLe77])$

## Two Heuristic Algorithms That Have Performance Guarantees

The next two heuristic algorithms have low-order polynomial complexity, and both have performance guarantees for graphs satisfying the triangle inequality. Both algorithms find a minimum spanning tree, create an eulerian tour of an associated graph, and then extract a hamiltonian cycle from the eulerian tour by taking shortcuts.

### Algorithm 6.4.2: Double the Tree

*Input:* a weighted complete graph  $G$ .

*Output:* a sequence of vertices and edges that forms a hamiltonian cycle.

Find a minimum spanning tree  $T^*$  of  $G$ .

Create an eulerian graph  $H$  by using two copies of each edge of  $T^*$ .

Construct an eulerian tour  $W$  of  $H$ .

Construct a hamiltonian cycle in  $G$  from  $W$  as follows:

Follow the sequence of edges and vertices of  $W$  until the next edge in the sequence is joined to an already visited vertex. At that point, skip to the next unvisited vertex by taking a shortcut, using an edge that is not part of  $W$ . Resume the traversal of  $W$ , taking shortcuts whenever necessary, until all the vertices have been visited. Complete the cycle by returning to the starting vertex via the edge joining it to the last vertex.

**Example 6.4.2:** Suppose that the tree shown in Figure 6.4.5(i) is a minimum spanning tree for a 7-vertex weighted graph. The eulerian tour of the doubled edges of the tree is shown in Figure 6.4.5(ii), and the vertex labels indicate the order of their first visits. Figure 6.4.5(iii) shows the hamiltonian cycle that results when the eulerian tour is modified by shortcuts via non-tree edges. The triangle inequality implies that these shortcuts are indeed shortcuts.

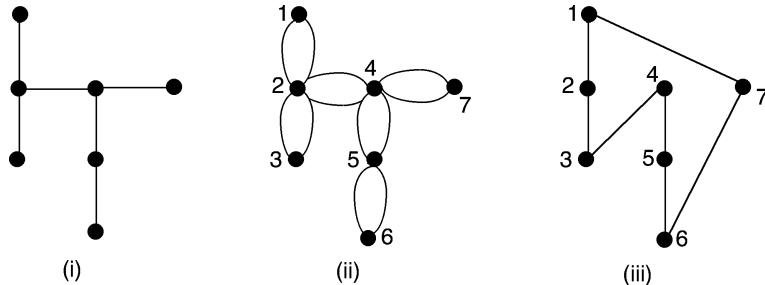


Figure 6.4.5 Illustration of Algorithm 6.4.2.

**Theorem 6.4.4.** For all instances of the TSP that obey the triangle inequality, the solution produced by Algorithm 6.4.2 is never worse than twice the optimal value.

**Proof:** Let  $C$  be the hamiltonian cycle produced by Algorithm 6.4.2, and let  $C^*$  and  $T^*$  be a minimum-weight hamiltonian cycle and a minimum-weight spanning tree, respectively. The total edge-weight of the eulerian tour is  $2 \times wt(T^*)$ , and since each shortcut is an edge that joins the initial and terminal points of a path of length at

least 2, the triangle inequality implies that  $wt(C) \leq 2 \times wt(T^*)$ . But  $C^*$  minus one of its edges is a spanning tree, which implies  $2 \times wt(T^*) \leq 2 \times wt(C^*)$ .  $\diamond$

One of the key steps of Algorithm 6.4.2 is the creation of an eulerian graph by duplicating each edge of the minimum spanning tree. N. Christofides took this idea one step further by recognizing that an eulerian graph can be created without having to duplicate all of the tree's edges. His idea is based on the strategy Edmonds and Johnson used in their solution of the Chinese Postman Problem (§6.2). That is, find a minimum-weight matching of the odd-degree vertices of  $T^*$ , and add those edges to  $T^*$  to obtain an eulerian graph. Christofides' algorithm, outlined below, achieves the best known performance guarantee of any approximate algorithm for the TSP.

**Algorithm 6.4.3: Tree and Matching**

*Input:* a weighted complete graph  $G$ .

*Output:* a sequence of vertices and edges that forms a hamiltonian cycle.

Find a minimum spanning tree  $T^*$  of  $G$ .

Let  $O$  be the subgraph of  $G$  induced on the odd-degree vertices in  $T^*$ .

Find a minimum-weight perfect matching  $M^*$  in  $O$ .

Create an eulerian graph  $H$  by adding the edges in  $M^*$  to  $T^*$ .

Construct an eulerian tour  $W$  of  $H$ , as in Algorithm 6.4.2.

Construct a hamiltonian cycle in  $G$  from  $W$ , as in Algorithm 6.4.2.

The details of Christofides' algorithm, including a proof of the following performance bound, can be found in [LaLeKaSh85].

**Theorem 6.4.5.** *For all instances of the TSP that obey the triangle inequality, the solution produced by Christofides' algorithm is never worse than  $\frac{3}{2}$  times the optimal.*

$\diamond$

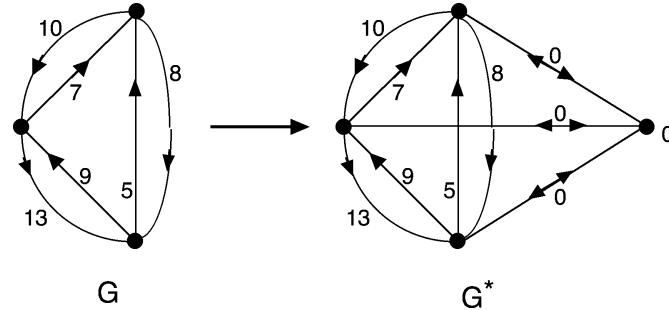
**Remark:** Performance guarantees must consider the worst-case behavior of a heuristic, and they may not reflect how well the heuristic actually performs in practice. Thus, performance guarantees should not be the only criterion in evaluating a heuristic. Runtime, ease of implementation, and empirical analysis are at least as important for the practitioner.

### TSP's in Disguise

Recognizing when a given problem can be transformed to the TSP does not make the problem's difficulty vanish, but it does make available a variety of methods developed over the last century. Here are three commonly encountered variations of the TSP that can be transformed to a standard TSP. Assume a weighted digraph  $G$  on vertices  $1, 2, \dots, n$ , where  $c_{ij}$  is the weight of arc  $ij$ .

**Variation 1.** Find a minimum-weight hamiltonian path.

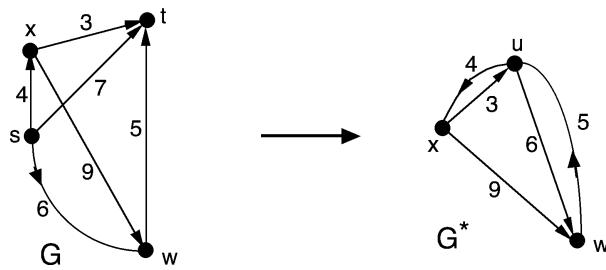
**Transformation 1:** Form a new graph  $G^*$  by adding to  $G$  an artificial vertex 0 and arcs to and from each of the other vertices, with  $c_{0j} = c_{j0} = 0$  for  $j = 1, 2, \dots, n$  (see Figure 6.4.6). Then an optimal hamiltonian path is obtained by deleting vertex 0 from an optimal hamiltonian cycle of  $G^*$ .

**Figure 6.4.6** Transformation 1.

**Variation 2.** Find a minimum-weight hamiltonian path in digraph  $G$ , from a specified vertex  $s$  to a specified vertex  $t$ .

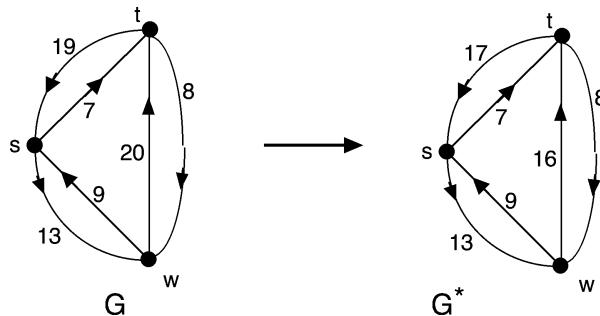
**Transformation 2:** Solve the standard TSP for a digraph  $G^*$ , formed from digraph  $G$  by replacing vertices  $s$  and  $t$  by a single vertex  $u$  and defining the edge-weights for  $G^*$  by

$$c_{ij}^* = \begin{cases} c_{sj}, & \text{if } i = u, \\ c_{it}, & \text{if } j = u, \\ c_{ij}, & \text{otherwise} \end{cases}$$

**Figure 6.4.7** Transformation 2.

**Variation 3.** Find a minimum-weight closed walk in digraph  $G$  that visits each vertex *at least* once.

**Transformation 3:** Replace each  $c_{ij}$  by the length of a shortest path from  $i$  to  $j$  (in the original graph).

**Figure 6.4.8** Transformation 3.

**Application 6.4.2 Job Sequencing on a Single Machine:** Suppose there are  $n$  jobs that must be processed on a single machine. The time required to process job  $j$  immediately after job  $i$  is  $c_{ij}$ . How should the jobs be sequenced so that the total time is minimized?

Draw an  $n$ -vertex digraph whose vertices correspond to the jobs; the arc directed from vertex  $i$  to vertex  $j$  is assigned the weight  $c_{ij}$ . Then an optimal sequencing of the jobs corresponds to a minimum-weight hamiltonian path.

**Application 6.4.3 Circuit Design:** A computer or other digital system consists of a number of modules and several pins located on each module. The physical position of each module has already been determined, and a given subset of pins has to be connected by wires. Because of the small size of the pins, at most two wires are to be attached to any pin. Furthermore, to minimize *noise* (stray signals from external sources) and to improve the ease of wiring, the total wire length should be minimized. An optimal wiring design will correspond to a minimum-weight hamiltonian path.

The next application is a special instance of *vehicle routing*. The general vehicle routing problem is to determine which vehicles should serve which customers and in what order. Typically, constraints include capacities of the vehicles as well as time windows for the customers.

**Application 6.4.4 School Bus Routing:** Each weekend, a private school transports  $n$  children to  $m$  bus stops across the state. The parents then meet their children at the bus stops. The school owns  $k$  buses, and the  $j$ th bus has seating capacity  $c_j$ ,  $j = 1, \dots, k$ . How should the buses be routed so that the total cost is minimized?

The  $i$ th bus stop is associated with a subset  $S_i$  of children,  $i = 1, \dots, m$ . Let the vertices  $v_1, \dots, v_m$  represent the  $m$  bus stops, and let vertex  $v_0$  represent the school location. An assignment  $\{v_{j_1}, v_{j_2}, \dots, v_{j_l}\}$  of  $l$  bus stops to the  $j$ th bus is *feasible* if  $\sum_{k=1}^l |S_{j_k}| \leq c_j$ . The cost of a feasible assignment is the cost of a minimum-weight hamiltonian cycle on the vertices  $\{v_0, v_{j_1}, v_{j_2}, \dots, v_{j_l}\}$ , where edge-weight represents distance. Thus, the problem is to partition the  $m$  bus stops into  $k$  subsets, such that the sum of all the buses' optimal tours is minimized.

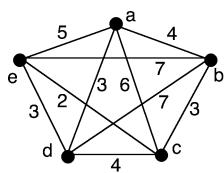
Notice that calculating the cost of a given partition requires solving  $k$  TSP subproblems. But a number of details have been omitted that actually make the problem even more complicated. For example, the distances that parents must travel to get to their children's bus stops are not being considered. Also, the operating costs of the chartered buses are likely to be more than those of the school-owned buses. Furthermore, there are limitations on how late each bus can make its last stop. If these are handicapped students, then there may be an additional requirement of one adult chaperon per bus. A final complication is that the  $m$  bus stops need not be predetermined. That is, the school may be able to choose which and how many bus stops from among hundreds of potential ones.

## EXERCISES for Section 6.4

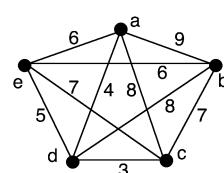
6.4.1<sup>s</sup> Use the inductive step in the proof of Theorem 6.4.1 to obtain a Gray code of order 4 from the Gray code of order 3 in Example 6.4.1.

In Exercises 6.4.2 through 6.4.5, apply each of Algorithms 6.4.1 through 6.4.3 to the given graph, starting at vertex  $a$  and resolving ties alphabetically. Indicate the vertex sequence and total edge-weight for each of the three outputs.

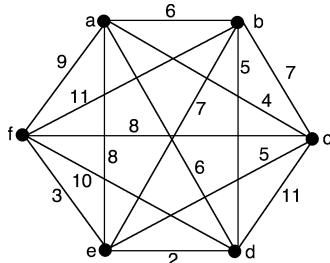
### 6.4.2



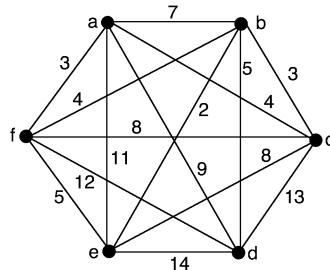
### 6.4.3<sup>s</sup>



#### 6.4.4



#### 6.4.5



In Exercises 6.4.6 through 6.4.9, apply a modified version of each of Algorithms 6.4.1 through 6.4.3 to the specified graph to try to find a minimum-weight open hamiltonian path that starts at vertex  $a$ . Base your modification on Transformation 1. Indicate the vertex sequence and total edge-weight for each of the three outputs.

#### 6.4.6 The graph of Exercise 6.4.2.

**6.4.7<sup>s</sup>** The graph of Exercise 6.4.3.

#### 6.4.8 The graph of Exercise 6.4.4.

#### 6.4.9 The graph of Exercise 6.4.5.

**6.4.10** Prove or disprove: In the  $n$ -dimensional hypercube  $Q_n$  (§1.2), the initial and terminal vertices of every open hamiltonian path are adjacent.

In Exercises 6.4.11 through 6.4.14, apply a modified version of each of Algorithms 6.4.1 through 6.4.3 to the specified graph to try to find a minimum-weight hamiltonian path from vertex  $a$  to vertex  $d$ . Base your modification on Transformation 2. Indicate the vertex sequence and total edge-weight for each of the three outputs.

#### 6.4.11 The graph of Exercise 6.4.2.

**6.4.12<sup>s</sup>** The graph of Exercise 6.4.3.

### 6.4.13 The graph of Exercise 6.4.4.

#### 6.4.14 The graph of Exercise 6.4.5.

In Exercises 6.4.15 and 6.4.16, apply a modified version of each of Algorithms 6.4.1 through 6.4.3 to the specified graph to try to find a minimum-weight closed walk that starts at vertex  $a$  and visits each vertex at least once. Base your modification on Transformation 3. Indicate the vertex sequence and total edge-weight for each of the three outputs.

#### 6.4.15 The graph of Exercise 6.4.2.

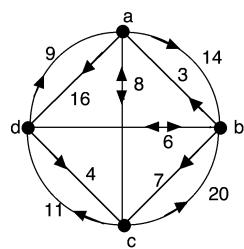
**6.4.16<sup>s</sup>** The graph of Exercise 6.4.3.

In Exercises 6.4.17 and 6.4.18, do each of the following for the given digraph.

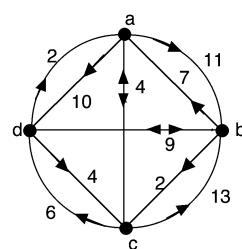
- a. Draw the digraph obtained by applying each of Transformations 1, 2, and 3 to the given digraph.
  - b. Give the vertex sequence and total edge-weight of the hamiltonian path (in the original digraph) from vertex  $d$  to vertex  $a$  that is obtained by applying each of Algorithms 6.4.1 and 6.4.2 to the digraph from part (a), and use exhaustive analysis to determine if either of these is a minimum-weight one.

- c. Give the vertex sequence and total edge-weight of the closed walk (in the original digraph) that uses each vertex at least once that is obtained by applying each of Algorithms 6.4.1 and 6.4.2 to the digraph from part (a), and use exhaustive analysis to determine if either of these is a minimum-weight one.

6.4.17



6.4.18



- 6.4.19 Suppose each of  $n$  jobs requires processing by  $m$  machines in the same order. Each machine can work on at most one job at a time, and once it begins work on a job, it must work on it to completion, without interruption. The amount of processing time that job  $i$  requires on machine  $h$  is  $p_{hi}$ . Also, once the processing of a job is complete on machine  $h$ , its processing must begin immediately on machine  $h + 1$  (this is called a *flow shop with no wait in process*). Show that the problem of sequencing the jobs so that the last job is completed as early as possible can be formulated as a standard  $(n + 1)$ -vertex TSP. (Hint: Create a dummy job requiring zero units of processing, and let  $c_{ij}$  represent the amount of idle time required on machine 1 if job  $j$  immediately follows job  $i$ .)

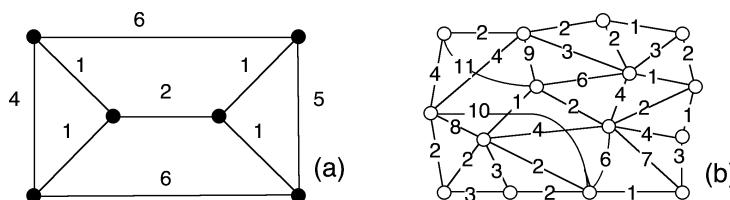
- 6.4.20 [Computer Project]** Implement Algorithm 6.4.1 and run the program on each of the graphs in Exercises 6.4.0 through 6.4.5.

## 6.5 SUPPLEMENTARY EXERCISES

DEF: A **deadhead path** in a postman tour is one that is retraced.

- 6.5.1 Draw a minimum-length postman tour in the graph of Figure 6.5.1(a), and show that the set of deadhead paths is of minimum cost.

- 6.5.2 Draw a minimum-length postman tour in the graph of Figure 6.5.1(b), and show that the set of deadhead paths is of minimum cost.



**Figure 6.5.1**

- 6.5.3 Calculate the number of eulerian tours in the complete graph  $K_5$ .

- 6.5.4 Give a pair of graphs one of which is eulerian, the other non-eulerian, with the same number of vertices, such that all the vertex-deleted subgraphs of both graphs are non-eulerian. This would establish the non-reconstructibility of the eulerian property.

**6.5.5** Two eulerian graphs  $A$  and  $B$  are amalgamated across a subgraph  $C$ . Prove that the resulting graph  $G$  is eulerian if and only if the subgraph  $C$  is eulerian.

**6.5.6** Decide whether the join  $2K_1 + K_{1,4}$  is hamiltonian.

**6.5.7** Decide whether the join  $3K_1 + K_{1,4}$  is hamiltonian.

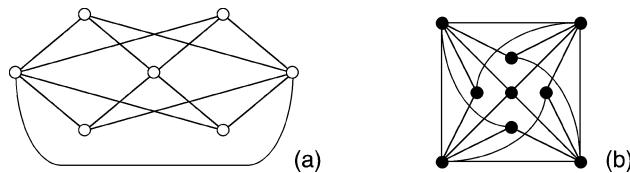
**6.5.8** Decide whether  $K_{3,4}$  is hamiltonian.

**6.5.9** Give an example of two connected graphs, each with at least two vertices, whose join is non-hamiltonian.

**6.5.10** Draw a hamiltonian graph  $H$  such that two disjoint copies of  $H$  can be amalgamated across  $K_3$ , so that the result is non-hamiltonian. Explain why the result is non-hamiltonian.

**6.5.11** Show that the graph of Figure 6.5.2(a) has a hamiltonian path but no hamiltonian cycle.

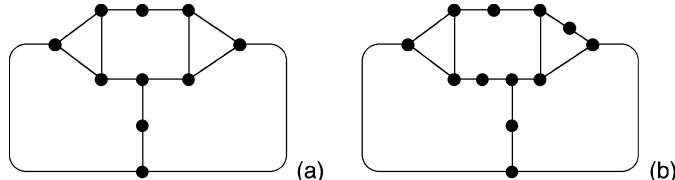
**6.5.12** Show that the graph of Figure 6.5.2(b) has a hamiltonian path but no hamiltonian cycle.



**Figure 6.5.2**

**6.5.13** Decide whether the graph in Figure 6.5.3(a) is hamiltonian.

**6.5.14** Decide whether the graph in Figure 6.5.3(b) is hamiltonian.



**Figure 6.5.3**

**6.5.15** Suppose that two non-adjacent vertices are chosen in a hamiltonian graph  $H$ , and two disjoint copies of  $H$  are amalgamated across this vertex pair. Give examples to show that the resulting graph can be hamiltonian or non-hamiltonian.

**6.5.16** Give a pair of graphs one of which is hamiltonian, the other non-hamiltonian, with the same number of vertices, such that all the vertex-deleted subgraphs of both graphs are non-hamiltonian. This would establish the non-reconstructibility of the hamiltonian property.

## GLOSSARY

**deadhead path** in a postman tour: a path that is retraced.

**deBruijn digraph**  $D_{2,n}$ : the digraph consisting of  $2^{n-1}$  vertices, each labeled by a different bitstring of length  $n - 1$ , and  $2^n$  arcs such that the arc from vertex  $b_1 b_2 \cdots b_{n-1}$  to vertex  $b_2 \cdots b_{n-1} b_n$  is labeled by the bitstring  $b_1 b_2 \cdots b_n$ ; referred to as the  $(2, n)$ -deBruijn digraph.

**deBruijn sequence** of length  $2^n$ : a bitstring of length  $2^n$  in which each of the  $2^n$  possible bitstrings of length  $n$  occurs *exactly once* as a substring, where wraparound is allowed; referred to as a  $(2, n)$ -deBruijn sequence.

**double tracing** in a graph: a closed walk that traverses every edge exactly twice.

—, **bidirectional**: a double tracing such that every edge is traversed in both directions.

**eulerian graph**: a graph that has an eulerian tour.

**eulerian tour**: a closed eulerian trail.

**eulerian trail** in a graph: a trail that contains every edge of that graph.

**Gray code of order  $n$** : an ordering of the  $2^n$  length- $n$  bitstrings such that consecutive bitstrings differ in precisely one bit position.

**hamiltonian graph**: a graph that has a hamiltonian cycle.

**hamiltonian path (cycle)** in a graph or digraph: a path (cycle) that contains all the vertices; for digraphs, the hamiltonian path or cycle is directed.

**heuristic**: a guideline that helps in choosing from among several possible alternatives for a decision step.

**heuristic algorithm**: an algorithm whose steps are guided by heuristics.

**knight's tour** of a chessboard: a sequence of knight moves that visits each square exactly once and returns to its starting square with one more move.

**line graph of a digraph  $G$** : the digraph  $L(G)$  whose vertex-set corresponds to the arc-set of  $G$ ; an arc in  $L(G)$  is directed from vertex  $e_1$  to vertex  $e_2$  if, in  $G$ ,  $\text{head}(e_1) = \text{tail}(e_2)$ .

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

**matching** in a graph  $G$ : a subset  $M$  of  $E_G$  such that no two edges in  $M$  have an endpoint in common.

—, **perfect**: a matching in which every vertex of the graph is an endpoint of one of the edges.

**postman tour** in a graph  $G$ : a closed walk that uses each edge of  $G$  at least once.

—, **optimal**: a postman tour whose total edge-weight is a minimum.

**switch cover, length 2** for a digraph: a sequence of arcs that includes all pairs of adjacent arcs.

**triangle inequality** in a weighted simple graph whose vertices are labeled  $1, 2, \dots, n$ : a condition on the edge-weights  $c_{ij}$ , given by  $c_{ij} \leq c_{ik} + c_{kj}$  for all  $i, j$ , and  $k$ .

# Chapter 7

---

## PLANARITY AND KURATOWSKI'S THEOREM

- 7.1 Planar Drawings and Some Basic Surfaces**
  - 7.2 Subdivision and Homeomorphism**
  - 7.3 Extending Planar Drawings**
  - 7.4 Kuratowski's Theorem**
  - 7.5 Algebraic Tests for Planarity**
  - 7.6 Planarity Algorithm**
  - 7.7 Crossing Numbers and Thickness**
- 

### INTRODUCTION

The central theme of this chapter is the topological problem of deciding whether a given graph can be drawn in the plane or sphere with no edge-crossings. Some planarity tests for graphs are in the form of algebraic formulas (see §7.5) based on the numbers of vertices and edges. These are the easiest tests to apply, yet they are one-way tests, and there are difficult cases in which they are inconclusive.

A celebrated result of the Polish mathematician Kasimir Kuratowski transforms the planarity decision problem into the combinatorial problem of calculating whether the given graph contains a subgraph *homeomorphic* (defined in §7.2) to the complete graph  $K_5$  or to the complete bipartite graph  $K_{3,3}$ . Directly searching for  $K_5$  and  $K_{3,3}$  would be quite inefficient, but this chapter includes a simple, practical algorithm (see §7.6) to test for planarity.

The relationship of planarity to topological graph theory is something like the relationship of plane geometry to what geometers call geometry, where mathematicians long ago began to develop concepts and methods to progress far beyond the plane. Whereas planarity consists mostly of relatively accessible ideas that were well understood several decades ago, topological graph theorists use newer methods to progress to all the other surfaces. Chapters 8, 15, and 16 provide an entry to modern topological graph theory.

## 7.1 PLANAR DRAWINGS AND SOME BASIC SURFACES

Our approach to drawings of graphs on surfaces begins with our intuitive notions of what we mean by a drawing and a surface. This chapter provides some precise examples of a surface, and Chapter 8 gives precision to the definitions of surface and drawing.

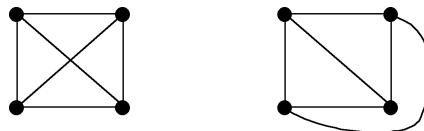
### Planar Drawings

Consistent with our temporary informality, we introduce the main topic of this chapter with the following definition.

**DEFINITION:** A **planar drawing** of a graph is a drawing of the graph in the plane without edge-crossings.

**DEFINITION:** A graph is said to be **planar** if there exists a planar drawing of it.

**Example 7.1.1:** Two drawings of the complete graph  $K_4$  are shown in Figure 7.1.1. The planar drawing on the right shows that  $K_4$  is a planar graph.



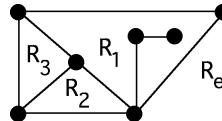
**Figure 7.1.1** A nonplanar drawing and a planar drawing of  $K_4$ .

**Example 7.1.2:** An instance of the problem of determining whether a given graph is planar occurs in the form of a well-known puzzle, called the **utilities problem**, in which three houses are on one side of a street and three utilities (electricity, gas, and water) are on the other. The objective of the puzzle is to join each of the three houses to each of the three utilities without having any crossings of the utility lines. Later in this section, we show that this is impossible, by proving that  $K_{3,3}$  is nonplanar.

**Remark:** A graph  $G$  and a given drawing of  $G$  are categorically different objects. That is, a graph is combinatorial and a drawing is topological. In particular, the *vertices* and *edges* in a drawing of a graph are actually *images* of the vertices and edges in that graph. Yet, to avoid excessive formal phrasing, these distinctions are relaxed when it is discernable from context what is intended.

**TERMINOLOGY:** Intuitively, we see that in a planar drawing of a graph, there is exactly one *exterior* (or *infinite*) region whose area is infinite.

**Example 7.1.3:** The exterior region,  $R_e$ , and the three *finite* regions of a planar drawing of a graph are shown in Figure 7.1.2.



**Figure 7.1.2** The four regions of a planar drawing of a graph.

**Remark:** If we consider a planar drawing of a graph on a piece of paper, then the intuitive notion of *region* corresponds to the pieces of paper that result from cutting the paper along the length of every edge in the drawing. If one adds new edges to a graph without crossing an existing edge, then a region may be subdivided into more regions.

**TERMINOLOGY NOTE:** We restrict the usage of the word “regions” to the case of crossing-free drawings, since many assertions that are true in that case may be untrue when there are edge-crossings.

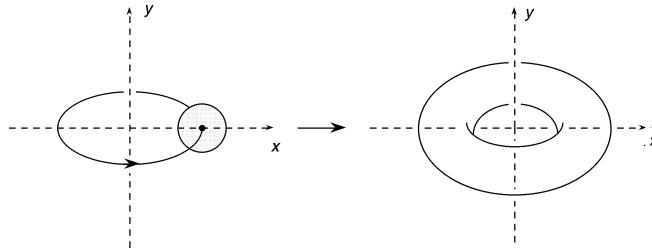
### Three Basic Surfaces

All of the surfaces under consideration in this chapter are subsets of Euclidean 3-space. Although we assume that the reader is familiar with the following surfaces, it is helpful to think about their mathematical models.

**DEFINITION:** A **plane** in Euclidean 3-space  $\mathbb{R}^3$  is a set of points  $(x, y, z)$  such that there are numbers  $a, b, c$ , and  $d$  with  $ax + by + cz = d$ .

**DEFINITION:** A **sphere** is a set of points in  $\mathbb{R}^3$  equidistant from a fixed point.

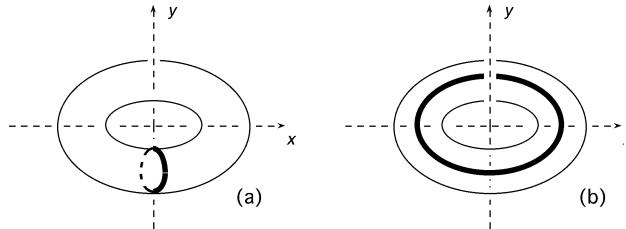
**DEFINITION:** The **standard torus** is the surface of revolution obtained by revolving a circle of radius 1 centered at  $(2,0)$  in the  $xy$ -plane disk around the  $y$ -axis in 3-space, as depicted in Figure 7.1.3. The solid inside is called the **standard donut**.



**Figure 7.1.3** Creating a torus.

**DEFINITION:** The circle of intersection, as in Figure 7.1.4(a), of the standard torus with the half-plane  $\{(x, y, z) \mid x = 0, y \leq 0\}$  is called the **standard meridian**. We observe that the standard meridian bounds a disk inside the standard donut.

**DEFINITION:** The circle of tangent intersection of the standard torus with the plane  $y = 1$  is called the **standard longitude**. Figure 7.1.4(b) illustrates the standard longitude. We observe that the standard longitude bounds a disk in the plane  $y = 1$  that lies outside the standard donut.



**Figure 7.1.4** (a) Standard meridian and (b) standard longitude on a torus.

**TERMINOLOGY:** Any closed curve that circles the torus once in the meridian direction (the “short” direction) without circling in the longitude direction (the “long” direction) is called a **meridian**. Any closed curve that circles the torus once in the longitude direction without circling in the meridian direction is called a **longitude**.

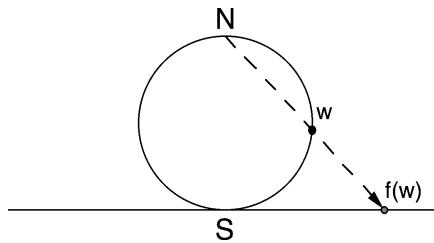
**Remark:** The three surfaces described above are all relatively uncomplicated. In §8.2, the *Möbius band* and the *Klein bottle* are defined, and it is described how the surfaces generally fall into two infinite sequences.

### Riemann Stereographic Projection

Riemann observed that deleting a single point from a sphere yields a surface that is equivalent to the plane for many purposes, including that of drawing graphs.<sup>†</sup>

**DEFINITION:** The **Riemann stereographic projection** is the function  $\rho$  that maps each point  $w$  of the unit-diameter sphere (tangent at the origin  $(0, 0, 0)$ ) to the point  $\rho(w)$  where the ray from the north pole  $(0, 1, 0)$  through point  $w$  intersects the  $xz$ -plane.

Under the Riemann projection, the “southern hemisphere” of the sphere is mapped continuously onto the unit disk. The “northern hemisphere” (minus the north pole) is mapped continuously onto the rest of the plane. The points nearest to the north pole are mapped to the points farthest from the origin. Figure 7.1.5 illustrates the construction.



**Figure 7.1.5** The Riemann stereographic projection.

**Remark:** Whereas a planar drawing of a graph has one exterior region (containing the “point at infinity”), a crossing-free graph drawn on the sphere has one region that contains the north pole. Given a graph drawn on the sphere, the Riemann stereographic projection enables us to move the drawing to the plane so that the point at infinity is deleted from whatever region we choose. That is, we simply rotate the sphere so that a point in the designated region is at the north pole. Accordingly, the choice of exterior region is usually irrelevant to understanding anything about the drawings of a particular graph in the plane. Moreover, a graph drawing on a flat piece of paper may be conceptualized as a drawing on a sphere whose radius is so large that its curvature is imperceptible.

**Proposition 7.1.1.** *A graph is planar if and only if it can be drawn without edge-crossings on the sphere.*

**Proof.** This is an immediate consequence of the Riemann stereographic projection.  $\diamond$

---

<sup>†</sup> The notion of *topological equivalence* is made precise in §8.1.

### Jordan Separation Property

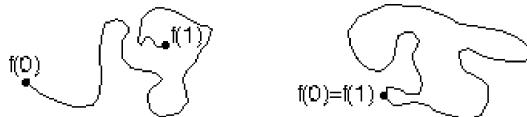
The objective of the definitions in this subsection is to lend precision to the intuitive notion of using a closed curve to separate a surface. Most of these definitions can be generalized.

**DEFINITION:** By a **Euclidean set**, we mean a subset of any Euclidean space  $\mathbb{R}^n$ .

**DEFINITION:** An **open path from  $s$  to  $t$**  in a Euclidean set  $X$  is the image of a continuous bijection  $f$  from the unit interval  $[0, 1]$  to a subset of  $X$  such that  $f(0) = s$  and  $f(1) = t$ . (One may visualize a path as the trace of a particle traveling through space for a fixed length of time.)

**DEFINITION:** A **closed path** or **closed curve** in a Euclidean set is the image of a continuous function  $f$  from the unit interval  $[0, 1]$  to a subset of that space such that  $f(0) = f(1)$ , but which is otherwise a bijection. (For instance, this would include a “knotted circle” in space.)

**Example 7.1.4:** Figure 7.1.6 shows an open path and a closed curve in the plane.



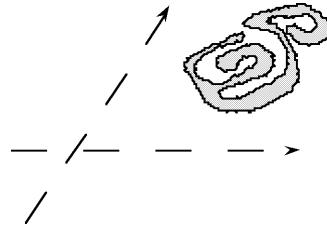
**Figure 7.1.6** Open path and closed curve (= closed path).

**Example 7.1.5:** In a crossing-free drawing of a graph on a surface, a cycle of the graph is a closed curve.

**DEFINITION:** A Euclidean set  $X$  is **connected**<sup>†</sup> if for every pair of points  $s, t \in X$ , there exists a path within  $X$  from  $s$  to  $t$ .

**DEFINITION:** The Euclidean set  $X$  **separates** the connected Euclidean set  $Y$  if there exist a pair of points  $s$  and  $t$  in  $Y - X$ , such that every path in  $Y$  from  $s$  to  $t$  intersects the set  $X$ .

**Example 7.1.6:** Figure 7.1.7 shows a meandering closed curve in the plane. It separates the white part from the shaded part.



**Figure 7.1.7** A closed curve separating the plane.

What makes the plane and the sphere the simplest surfaces for drawing graphs is the *Jordan separation property*. By invoking this property, we can prove that  $K_5$  and  $K_{3,3}$  cannot be drawn without edge-crossings in the plane or the sphere.

---

<sup>†</sup> A topologist would say *path-connected*.

DEFINITION: A Euclidean set  $X$  has the **Jordan separation property** if every closed curve in  $X$  separates  $X$ .

**Theorem 7.1.2 (Jordan Curve Theorem).** *Every closed curve in the sphere (plane) has the Jordan separation property, that is, it separates the sphere (plane) into two regions, one of which contains the north pole (contains “infinity”).* <sup>†</sup>  $\diamond$  (proof omitted)

**Corollary 7.1.3.** *A path from one point on the boundary of a disk through the interior to another point on the boundary separates the disk.*  $\diamond$  (proof omitted)

We again emphasize that the Jordan separation property distinguishes the plane and sphere from other surfaces. For instance, although it is possible to draw a closed curve on a torus that separates the surface into two parts (e.g., just draw a little circle around a point), a meridian does not, nor does a longitude, as one sees clearly in Figure 7.1.4.

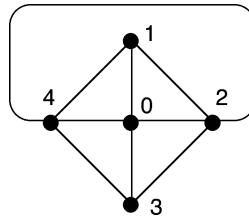
The Jordan Curve Theorem is quite difficult to prove in full generality; in fact, Jordan himself did it incorrectly in 1887. The first correct proof was by Veblen in 1905. A proof for the greatly simplified case in which the closed curve consists entirely of straight-line segments, so that it is a closed polygon, is given by Courant and Robbins in *What Is Mathematics?*

### Applying The Jordan Curve Theorem to the Nonplanarity of $K_5$ and $K_{3,3}$

It is possible to prove that a particular graph is nonplanar directly from the Jordan Curve Theorem. In what follows, we continue to rely temporarily on an intuitive notion of the *regions* of a crossing-free drawing of a graph on a sphere or plane. A more formal definition of *region* is given at the beginning of §7.3.

**Theorem 7.1.4.** *Every drawing of the complete graph  $K_5$  in the sphere (or plane) contains at least one edge-crossing.*

**Proof:** Label the vertices  $0, \dots, 4$ . By the Jordan Curve Theorem, any drawing of the cycle  $\langle 1, 2, 3, 4, 1 \rangle$  separates the sphere into two regions. Consider the region with vertex 0 in its interior as the “inside” of the cycle. By the Jordan Curve Theorem, the edges joining vertex 0 to each of the vertices 1, 2, 3, and 4 must also lie entirely inside the cycle, as illustrated in Figure 7.1.8. Moreover, each of the 3-cycles  $\langle 0, 1, 2, 0 \rangle$ ,  $\langle 0, 2, 3, 0 \rangle$ ,  $\langle 0, 3, 4, 0 \rangle$ , and  $\langle 0, 4, 1, 0 \rangle$  also separates the sphere, and, hence, edge 24 must lie to the exterior of the cycle  $\langle 1, 2, 3, 4, 1 \rangle$ , as shown.



**Figure 7.1.8** Drawing most of  $K_5$  in the sphere.

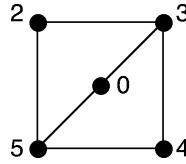
---

<sup>†</sup> By the Schönflies theorem, both regions on a sphere and the interior region in a plane are topologically equivalent to open disks. For a proof, see [Ne54].

It follows that the cycle formed by edges 24, 40, and 02 separates vertices 1 and 3, again by the Jordan Curve Theorem. Thus, it is impossible to draw edge 13 without crossing an edge of that cycle.  $\diamond$

**Theorem 7.1.5.** *Every drawing of the complete bipartite graph  $K_{3,3}$  in the sphere (or plane) contains at least one edge-crossing.*

**Proof:** Label the vertices of one partite set 0, 2, 4, and of the other 1, 3, 5. By the Jordan Curve Theorem, cycle  $\langle 2, 3, 4, 5, 2 \rangle$  separates the sphere into two regions, and, as in the previous proof, we regard the region containing vertex 0 as the “inside” of the cycle. By the Jordan Curve Theorem, the edges joining vertex 0 to each of the vertices 3 and 5 lie entirely inside that cycle, and each of the cycles  $\langle 0, 3, 2, 5, 0 \rangle$  and  $\langle 0, 3, 4, 5, 0 \rangle$  separates the sphere, as illustrated in Figure 7.1.9.

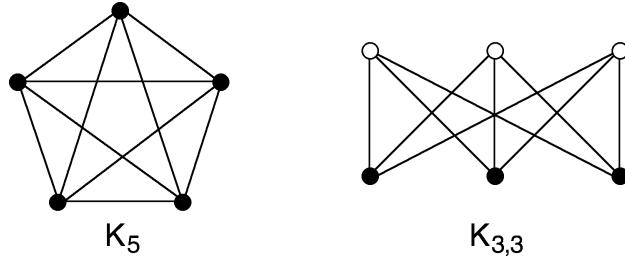


**Figure 7.1.9** Drawing most of  $K_{3,3}$  in the sphere.

Thus, there are three regions: the exterior of cycle  $\langle 2, 3, 4, 5, 2 \rangle$ , and the inside of each of the other two cycles. It follows that no matter which region contains vertex 1, there must be some even-numbered vertex that is not in that region, and, hence, the edge from vertex 1 to that even-numbered vertex would have to cross some cycle edge.  $\diamond$

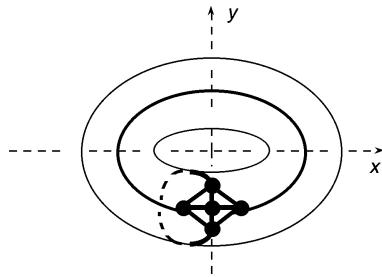
**Corollary 7.1.6.** *If either  $K_5$  or  $K_{3,3}$  is a subgraph of a graph  $G$  then every drawing of  $G$  in the sphere (or plane) contains at least one edge-crossing.*  $\diamond$

**DEFINITION:** The complete graph  $K_5$  and the complete bipartite graph  $K_{3,3}$  are called the **Kuratowski graphs**.



**Figure 7.1.10** The Kuratowski graphs.

**Example 7.1.7:** Figure 7.1.11 illustrates that  $K_5$  can be drawn without edge-crossings on the torus, even though this is impossible on the sphere. It becomes clear in Chapter 8 that for every graph there is a surface on which a crossing-free drawing is possible.



**Figure 7.1.11** A crossing-free drawing of  $K_5$  on the torus.

### EXERCISES for Section 7.1

Each of the Exercises 7.1.1 through 7.1.4 gives a point  $w$  on the sphere that serves as the domain of the Riemann stereographic projection. Calculate the coordinates of the point  $\rho(w)$  in the  $xz$ -plane to which it is projected. (Hint: Write the locus of the line through the north pole and  $w$ .)

$$7.1.1^s \quad \left(\frac{1}{2}, \frac{1}{2}, 0\right).$$

$$7.1.2 \quad \left(\frac{1}{4}, \frac{3}{4}, \frac{\sqrt{2}}{4}\right).$$

$$7.1.3 \quad \left(\frac{1}{5}, \frac{9}{10}, \frac{\sqrt{5}}{10}\right).$$

$$7.1.4 \quad \left(\frac{1}{3}, \frac{3}{4}, \frac{\sqrt{11}}{12}\right).$$

Each of the Exercises 7.1.5 through 7.1.8 gives a point  $w$  in the  $xz$ -plane. Calculate the coordinates of the point  $\rho^{-1}(w)$  in the unit sphere under the inverse Riemann stereographic projection. (Hint: Use analytic geometry, as in the previous four exercises.)

$$7.1.5^s \quad \left(\frac{\sqrt{3}}{3}, 0, 0\right).$$

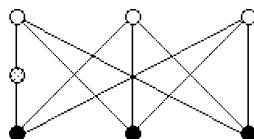
$$7.1.6 \quad (3, 0, 4).$$

$$7.1.7 \quad (1, 0, 1).$$

$$7.1.8 \quad \left(\frac{1}{2}, 0, 0\right).$$

## 7.2 SUBDIVISION AND HOMEOMORPHISM

The graph in Figure 7.2.1 looks a lot like the Kuratowski graph  $K_{3,3}$ , but one of the “edges” has an intermediate vertex. Clearly, if this graph could be drawn on a surface without any edge-crossings, then so could  $K_{3,3}$ . Intuitively, it would be an easy matter of erasing the intermediate vertex and splicing the “loose ends” together.



**Figure 7.2.1** Subdividing an edge.

Placing one or more intermediate vertices on an edge is formally known as *subdivision*. The relationship between the two graphs is known as *homeomorphism*. The concept of homeomorphism originated as an equivalence relation of topological spaces. It is defined in topology to be a one-to-one, onto, continuous function with a continuous inverse. This section introduces the graph-theoretic analogue of this topological notion.

### Graph Subdivision

**DEFINITION:** Let  $e$  be an edge with endpoints  $\{u, v\}$  in a graph  $G$ . **Subdividing the edge  $e$**  means that a new vertex  $w$  is added to  $V_G$ , and that edge  $e$  is replaced in  $E_G$  by an edge  $e'$  with endpoints  $\{u, w\}$  and an edge  $e''$  with endpoints  $\{w, v\}$ .

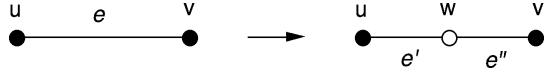


Figure 7.2.2 Subdividing an edge.

**DEFINITION:** Let  $w$  be a 2-valent vertex in a graph  $G$ , such that two proper edges  $e'$  and  $e''$  meet at  $w$ . **Smoothing away** or (**smoothing out**) vertex  $w$  means replacing edges  $e'$  and  $e''$  by a new edge  $e$  that joins the other endpoints of  $e'$  and  $e''$ .



Figure 7.2.3 Smoothing away a vertex.

**Example 7.2.1:** The  $(n + 1)$ -cycle graph can be obtained from the  $n$ -cycle graph by subdividing any edge, as illustrated in Figure 7.2.4. Inversely, smoothing away any vertex on the  $(n + 1)$ -cycle, for  $n \geq 1$ , yields the  $n$ -cycle. It is not permitted to smooth away the only vertex of the 1-cycle  $C_1$  (= the bouquet  $B_1$ ).

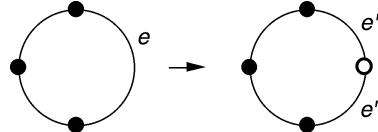


Figure 7.2.4 Subdividing an edge of the 3-cycle yields the 4-cycle.

**DEFINITION:** **Subdividing a graph  $G$**  means performing a sequence of edge-subdivision operations. The resulting graph is called a **subdivision of the graph  $G$** .

**Example 7.2.2:** Performing any  $k$  subdivisions on the  $n$ -cycle graph  $C_n$  yields the  $(n + k)$ -cycle graph  $C_{n+k}$ , and  $C_n$  can be obtained by any  $k$  smoothing operations on  $C_{n+k}$ . Thus,  $C_{n+k}$  is a subdivision of  $C_n$ , for all  $n \geq 1, k \geq 1$ .

**Proposition 7.2.1.** A subdivision of a graph can be drawn without edge-crossings on a surface if and only if the graph itself can be drawn without edge-crossings on that surface.

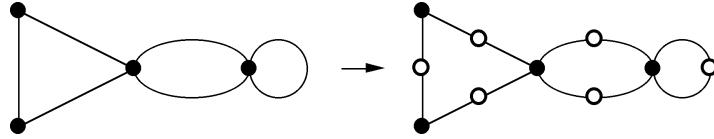
**Proof:** When the operations of subdivision and smoothing are performed on a copy of the graph already drawn on the surface, they neither introduce nor remove edge-crossings.  $\diamond$

### Barycentric Subdivision

The operation of subdivision can be used to convert a general graph into a simple graph. This justifies a brief digression on our path toward a proof of Kuratowski's theorem.

**DEFINITION:** The (*first*) **barycentric subdivision** of a graph is the subdivision in which one new vertex is inserted in the interior of each edge.

**Example 7.2.3:** The concept of barycentric subdivision and the next three propositions are illustrated by Figure 7.2.5.



**Figure 7.2.5** A graph and its barycentric subdivision.

**Proposition 7.2.2.** *The barycentric subdivision of any graph is a bipartite graph.*

**Proof:** Let  $G'$  denote the barycentric division of graph  $G$ . One endpoint of each edge in  $G'$  is an “old” vertex (i.e., from  $V_G$ ), and the other endpoint is “new” (i.e., from subdividing).  $\diamond$

**Proposition 7.2.3.** *Barycentric subdivision of any graph yields a loopless graph.*

**Proof:** By Proposition 7.2.2, a barycentric subdivision of a graph is bipartite, and a bipartite graph has no self-loops.  $\diamond$

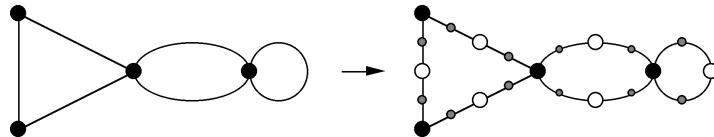
**Proposition 7.2.4.** *Barycentric subdivision of any loopless graph yields a simple graph.*

**Proof:** Clearly, barycentric subdivision of a loopless graph cannot create loops. If two edges of a barycentric subdivision graph have the same “new” vertex  $w$  as an endpoint, then the other endpoints of these two edges must be the distinct “old” vertices that were endpoints of the old edge subdivided by  $w$ .  $\diamond$

**DEFINITION:** The  $n^{\text{th}}$  **barycentric subdivision** of a graph is the first barycentric subdivision of the  $(n - 1)^{\text{st}}$  barycentric subdivision.

**Proposition 7.2.5.** *The second barycentric subdivision of any graph is a simple graph.*

**Proof:** By Proposition 7.2.3, the first barycentric subdivision is loopless, and thus, by Proposition 7.2.4, the second barycentric subdivision is simple. See Figure 7.2.6.  $\diamond$

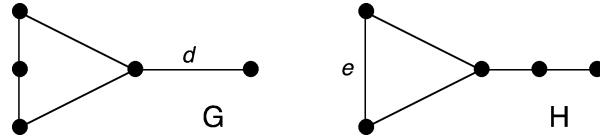


**Figure 7.2.6** A graph and its second barycentric subdivision.

### Graph Homeomorphism

**DEFINITION:** The graphs  $G$  and  $H$  are **homeomorphic graphs** if there is an isomorphism from a subdivision of  $G$  to a subdivision of  $H$ .

**Example 7.2.4:** Graphs  $G$  and  $H$  in Figure 7.2.7 cannot be isomorphic, since graph  $G$  is bipartite and graph  $H$  contains a 3-cycle. However, they are homeomorphic, because if edge  $d$  in graph  $G$  and edge  $e$  in graph  $H$  are both subdivided, then the resulting graphs are isomorphic.



**Figure 7.2.7** Two non-isomorphic graphs that are homeomorphic.

**Remark:** Notice in Example 7.2.4 that no subdivision of graph  $G$  is isomorphic to graph  $H$ , and that no subdivision of graph  $H$  is isomorphic to graph  $G$ .

**Proposition 7.2.6.** *Let  $G$  and  $H$  be homeomorphic graphs. Then  $G$  can be drawn without edge-crossings on a surface  $S$  if and only if  $H$  can be drawn on  $S$  without edge-crossings.*

**Proof:** This follows from iterated application of Proposition 7.2.1.  $\diamond$

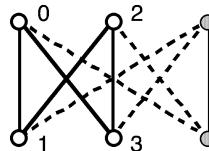
**Proposition 7.2.7.** *Every graph is homeomorphic to a bipartite graph.*

**Proof:** By Proposition 7.2.2, the barycentric subdivision of a graph is bipartite. Of course, a graph is homeomorphic to a subdivision of itself.  $\diamond$

### Subgraph Homeomorphism Problem

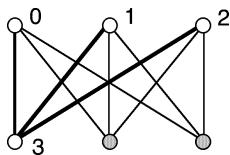
Deciding whether a graph  $G$  contains a subgraph that is homeomorphic to a target graph  $H$  is a common problem in graph theory.

**Example 7.2.5:** Figure 7.2.8 shows that the complete bipartite graph  $K_{3,3}$  contains a subgraph that is homeomorphic to the complete graph  $K_4$ . Four of the edges of  $K_4$  are represented by the cycle 0-1-2-3-0, and the other two are represented by the two paths of length 2 indicated by broken lines.



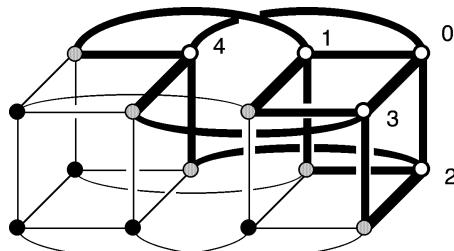
**Figure 7.2.8** A homeomorphic copy of  $K_4$  in  $K_{3,3}$ .

Notice in Figure 7.2.8 that each side of the bipartition of  $K_{3,3}$  contains two of the images of vertices of  $K_4$ . A homeomorphic copy of  $K_4$  in  $K_{3,3}$  cannot have three vertex images on one side of the bipartition. To see this, suppose (without loss of generality) that vertex images 0, 1, and 2 are on one side, and that vertex 3 is on the other side, as illustrated in Figure 7.2.9 below. A homeomorphic copy of  $K_4$  would require three internally disjoint paths joining the three possible pairs of vertices 0, 1, and 2. But this is impossible, since there are only two remaining vertices on the other side that can be used as internal vertices of these paths.



**Figure 7.2.9** An impossible way to place  $K_4$  into  $K_{3,3}$ .

**Example 7.2.6:** Figure 7.2.10 shows that the hypercube graph  $Q_4$  contains a subgraph that is homeomorphic to the complete graph  $K_5$ . The five labeled white vertices represent vertices of  $K_5$ . The ten internally disjoint paths joining the ten different pairs of these vertices are given bold edges. Gray vertices are internal vertices along such paths.

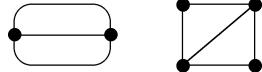


**Figure 7.2.10** A homeomorphic copy of  $K_5$  in  $Q_4$ .

### EXERCISES for Section 7.2

For Exercises 7.2.1 through 7.2.4, for the given pair of graphs, either prove that they are homeomorphic or prove that they are not.

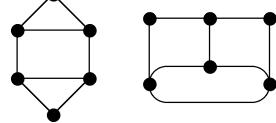
7.2.1<sup>s</sup>



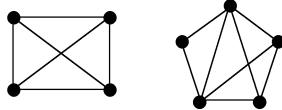
7.2.2



7.2.3



7.2.4



7.2.5<sup>s</sup> What is the minimum number of subdivision vertices required to make the bouquet  $B_n$  simple? (Hint: The second barycentric subdivision is not the minimum such subdivision.)

7.2.6 Show that two graphs are homeomorphic if and only if they are isomorphic to subdivisions of the same graph.

For Exercises 7.2.7 through 7.2.10, prove that a homeomorphic copy of the first of the given graphs is contained in the second graph.

7.2.7<sup>s</sup>  $K_4$  in  $K_{3,4}$ .

7.2.8  $K_{3,3}$  in  $K_{4,5}$ .

7.2.9  $W_4$  in  $K_{4,5}$ .

7.2.10  $K_5$  in  $K_{4,5}$ .

For Exercises 7.2.11 through 7.2.14, prove that the second given graph does NOT contain a homeomorphic copy of the first graph.

7.2.11<sup>s</sup>  $K_4$  in  $K_{2,3}$ .

7.2.13  $W_5$  in  $K_{4,4}$ .

7.2.12  $K_{3,3}$  in  $K_{4,4} - 4K_2$ .

7.2.14  $Q_3$  in  $K_{4,4}$ .

## 7.3 EXTENDING PLANAR DRAWINGS

We continue to regard a *drawing* of a graph on a surface as an intuitive notion. A more precise definition appears in the next chapter. The following definitions become mathematically precise as soon as the notion of a *drawing* is precise.

**DEFINITION:** An **imbedding of a graph**  $G$  on a surface  $S$  is a drawing without any edge-crossings. We may denote the imbedding  $\iota : G \rightarrow S$ .

Thus, a planar drawing of a graph  $G$  is an imbedding of  $G$  on the plane.

**DEFINITION:** A **region** of a graph imbedding  $G \rightarrow S$  is a component of the Euclidean set that results from deleting the image of  $G$  from the surface  $S$ .

**DEFINITION:** The **boundary of a region**  $r$  of a graph imbedding  $\iota : G \rightarrow S$  is the subgraph of  $G$  that comprises all vertices that abut the region and all edges whose interiors abut the region.

**Remark:** When a 2-connected graph is imbedded on the sphere, the boundary of every region is a cycle on the perimeter of the region. However, it is a dangerous misconception to imagine that this is the general case.

**DEFINITION:** A **face** of a graph imbedding  $\iota : G \rightarrow S$  is the union of a region and its boundary.

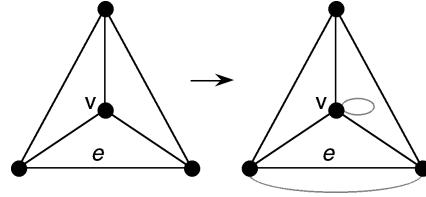
**Remark:** A drawing does *not* have faces unless it is an imbedding.

### Planar Extensions of a Planar Subgraph

A standard way to construct a planar drawing of a graph is to draw a subgraph in the plane, and then to extend the drawing by adding the remaining parts of the graph. This section gives the terminology and some basic results about the planar extensions of a subgraph. These results are helpful in proving Kuratowski's theorem and in specifying a planarity algorithm in the final two sections of this chapter.

**Proposition 7.3.1.** *A planar graph  $G$  remains planar if a multiple edge or self-loop is added to it.*

**Proof:** Draw graph  $G$  in the plane. Alongside any existing edge  $e$  of the drawing, another edge can be drawn between the endpoints of  $e$ , sufficiently close to  $e$  that it does not intersect any other edge. Moreover, at any vertex  $v$ , inside any region with  $v$  on its boundary, it is possible to draw a self-loop with endpoint  $v$ , sufficiently small that it does not intersect another edge. ◇

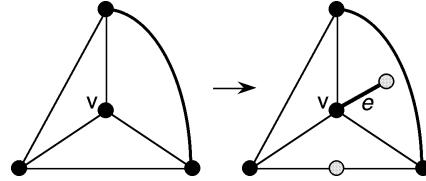


**Figure 7.3.1** Adding a self-loop or a multi-edge preserves planarity.

**Corollary 7.3.2.** A nonplanar graph remains nonplanar if a self-loop is deleted or if one edge of a multi-edge is deleted.  $\diamond$

**Proposition 7.3.3.** A planar graph  $G$  remains planar if any edge is subdivided or if a new edge  $e$  is attached to a vertex  $v \in V_G$  (with the other endpoint of  $e$  added as a new vertex).

**Proof:** Clearly, placing a dot in the middle of an edge in a planar drawing of  $G$  to indicate subdivision does not create edge-crossings. Moreover, it is easy enough to insert edge  $e$  into any region that is incident on vertex  $v$ .  $\diamond$

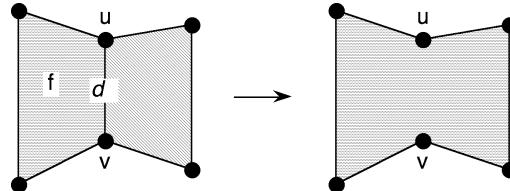


**Figure 7.3.2** Subdividing an edge or adding a spike preserves planarity.

The following proposition applies to imbeddings in all surfaces, not just to imbeddings in the sphere.

**Proposition 7.3.4.** Let  $\iota : G \rightarrow S$  be a graph imbedding on a sphere or on any other surface. Let  $d$  be an edge of  $G$  with endpoints  $u$  and  $v$ . Then the imbedding of the graph  $G - d$  obtained by deleting edge  $d$  from the imbedding  $\iota : G \rightarrow S$  has a face whose boundary contains both of the vertices  $u$  and  $v$ .

**Proof:** In the imbedding  $\iota : G \rightarrow S$ , let  $f$  be a face whose boundary contains edge  $d$ . When edge  $d$  is deleted, face  $f$  is merged with whatever face lies on the other side of edge  $d$ , as illustrated in Figure 7.3.3. On surface  $S$ , the boundary of the merged face is the union of the boundaries of the faces containing edge  $d$ , minus the interior of edge  $d$ . (This is true even when the same face  $f$  lies on both sides of edge  $d$ .) Vertices  $u$  and  $v$  both lie on the boundary of that merged face, exactly as illustrated.  $\diamond$



**Figure 7.3.3** Merging two regions by deleting an edge.

### Amalgamating Planar Graphs

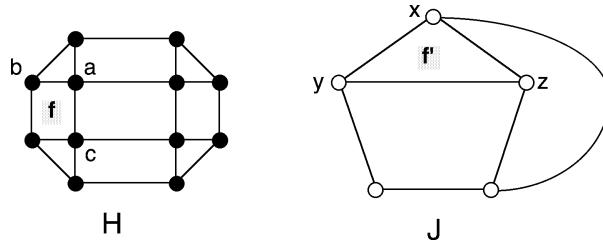
We recall from §2.7 that amalgamating two graphs means pasting a subgraph in one graph to an isomorphic subgraph in the other.

**Proposition 7.3.5.** *Let  $f$  be a face of a planar drawing of a connected graph  $G$ . Then there is a planar drawing of  $G$  in which the boundary walk of face  $f$  bounds a disk in the plane that contains the entire graph  $G$ . That is, in the new drawing, face  $f$  is the “outer” face.*

**Proof:** Copy the planar drawing of  $G$  onto the sphere so that the north pole lies in the interior of face  $f$ . Then apply the Riemann stereographic projection.  $\diamond$

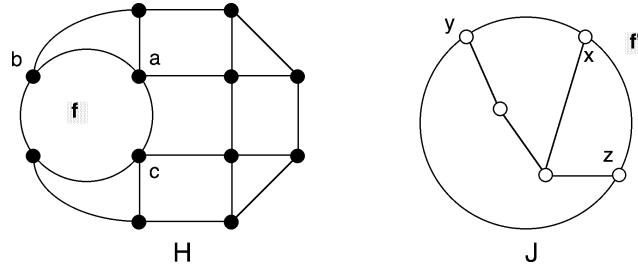
**Proposition 7.3.6.** *Let  $f$  be a face of a planar drawing of a graph  $H$ , and let  $u_1, \dots, u_n$  be a subsequence of vertices in the boundary walk of  $f$ . Let  $f'$  be a face of a planar drawing of a graph  $J$ , and let  $w_1, \dots, w_n$  be a subsequence of vertices in the boundary walk of  $f'$ . Then the amalgamated graph  $(H \cup J)/\{u_1 = w_1, \dots, u_n = w_n\}$  is planar.*

**Proof:** Planar drawings of graphs  $H$  and  $J$  with  $n = 3$  are illustrated in Figure 7.3.4.



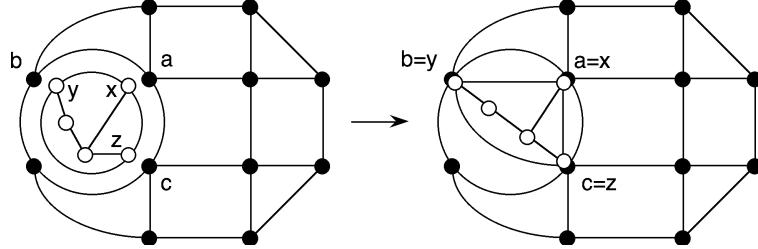
**Figure 7.3.4** Two planar graph drawings.

First redraw the plane imbedding of  $H$  so that the unit disk lies wholly inside face  $f$ . Next redraw graph  $J$  so that the boundary walk of face  $f'$  surrounds the rest of graph  $J$ , which is possible according to Proposition 7.3.5. Figure 7.3.5 shows these redrawings of graphs  $H$  and  $J$ .



**Figure 7.3.5** Redrawings of the two planar graph imbeddings.

We may assume that the cyclic orderings of the vertex sequences  $\{u_1, \dots, u_n\}$  and  $\{w_1, \dots, w_n\}$  are consistent with each other, since the drawing of graph  $J$  can be reflected, if necessary, to obtain cyclic consistency. Now shrink the drawing of graph  $J$  so that it fits inside the unit disk in face  $f$ , as shown on the left in Figure 7.3.6 below. Then stretch the small copy of  $J$  outward, as illustrated on the right side of that figure, thereby obtaining a crossing-free drawing of the amalgamation  $(H \cup J)/\{a = x, b = y, c = z\}$ .  $\diamond$

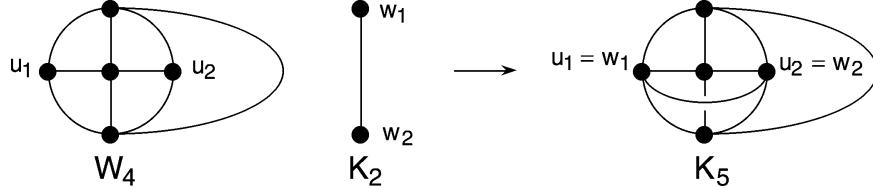


**Figure 7.3.6** Position the two graphs and then amalgamate by stretching.

**Corollary 7.3.7.** Let  $H$  and  $J$  be planar graphs. Let  $U$  be a set of one, two, or three vertices in the boundary of a face  $f$  of the drawing of  $H$ , and let  $W$  be a set of the same number of vertices in the boundary of a face  $f'$  of the drawing of  $J$ . Then the amalgamated graph  $(H \cup J)/\{U = W\}$  is planar.

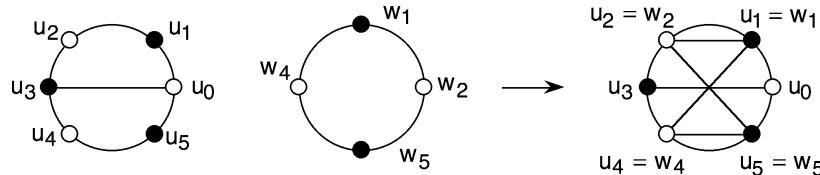
**Proof:** Whenever there are at most three vertices in the vertex subsequences to be amalgamated, there are only two possible cyclic orderings. Since reflection of either of the drawings is possible, the vertices of sets  $U$  and  $W$  in the boundaries of faces  $f$  and  $f'$ , respectively, can be aligned to correspond to any bijection  $U \rightarrow W$ .  $\diamond$

**Remark:** The requirement that vertices of amalgamation be selected in their respective graphs from the boundary of a single face cannot be relaxed. Amalgamating two planar graphs across two arbitrarily selected vertices may yield a nonplanar graph. For instance, Figure 7.3.7 shows how the nonplanar graph  $K_5$  can be derived as the amalgamation of the planar graphs  $W_4$  and  $K_2$ . This does not contradict Corollary 7.3.7, because the two vertices of amalgamation in  $W_4$  do not lie on the same face of any planar drawing of  $W_4$ .



**Figure 7.3.7** A 2-vertex amalgamation of two planar graphs into  $K_5$ .

**Remark:** Amalgamating two planar graphs across sets of four or more vertices per face may yield a nonplanar graph, as shown in Figure 7.3.8. The resulting graph shown is  $K_{3,3}$  with two doubled edges. The bipartition of  $K_{3,3}$  is shown in black and white.



**Figure 7.3.8** A 4-vertex amalgamation of two planar graphs into  $K_{3,3}$ .

### Appendages to a Subgraph

In an intuitive sense, subgraph  $H$  of a graph  $G$  *separates* two edges if it is impossible to get from one edge to the other without going through  $H$ . The following definition makes this precise.

**DEFINITION:** Let  $H$  be a subgraph of a connected graph  $G$ . Two edges  $e_1$  and  $e_2$  of  $E_G - E_H$  are **unseparated by subgraph  $H$**  if there exists a walk in  $G$  that contains both  $e_1$  and  $e_2$ , but whose internal vertices are not in  $H$ .

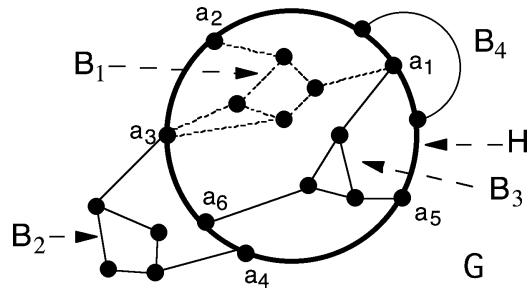
**Remark:** The relation *unseparated by subgraph  $H$*  is an equivalence relation on  $E_G - E_H$ ; that is, it is reflexive, symmetric, and transitive.

**DEFINITION:** Let  $H$  be a subgraph of a graph  $G$ . Then an **appendage to subgraph  $H$**  is the induced subgraph on an equivalence class of edges of  $E_G - E_H$  under the relation *unseparated by  $H$* .

**DEFINITION:** Let  $H$  be a subgraph of a graph. An appendage to  $H$  is called a **chord** if it contains only one edge. Thus, a chord joins two vertices of  $H$ , but does not lie in the subgraph  $H$  itself.

**DEFINITION:** Let  $H$  be a subgraph of a graph, and let  $B$  be an appendage to  $H$ . Then a **contact point** of  $B$  is a vertex of  $B \cap H$ .

**Example 7.3.1:** In the graph  $G$  of Figure 7.3.9, subgraph  $H$  is the bold cycle. Appendage  $B_1$  is the broken-edge subgraph with contact points  $a_1, a_2$ , and  $a_3$ . Appendage  $B_2$  is the exterior subgraph with contact points  $a_3$  and  $a_4$ . Appendage  $B_3$  is the interior subgraph with contact points  $a_1, a_5$ , and  $a_6$ . Appendage  $B_4$  is an exterior chord. Notice that within each of the three non-chord appendages, it is possible to get from any edge to any other edge by a walk in which none of the internal vertices is a contact point.



**Figure 7.3.9** A subgraph  $H$  and its appendages  $B_1, B_2, B_3$ , and  $B_4$ .

Also notice that each non-chord appendage contains a single component of the deletion subgraph  $G - V_H$ . In addition to a component of  $G - V_H$ , a non-chord appendage also contains every edge extending from that component to a contact point, and the contact point as well.

**Remark:** Every subgraph of a graph (not just cycles) has appendages. Even a subgraph comprising a set of vertices and no edges would have appendages.

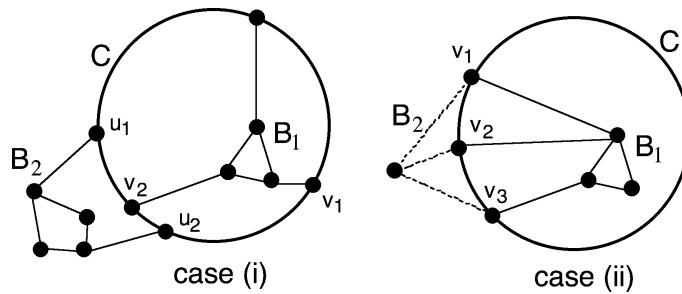
### Overlapping Appendages

The construction of a planar drawing of a connected graph  $G$  commonly begins with the selection of a “large” cycle to be the subgraph whose appendages constitute the rest of  $G$ . In this case, some special terminology describes the relationship between two appendages, in terms of their contact points.

**DEFINITION:** Let  $C$  be a cycle in a graph. The appendages  $B_1$  and  $B_2$  of  $C$  **overlap** if either of these conditions holds:

- i. Two contact points of  $B_1$  alternate with two contact points of  $B_2$  on cycle  $C$ .
- ii.  $B_1$  and  $B_2$  have three contact points in common.

**Example 7.3.2:** Both possibilities (i) and (ii) for overlapping appendages are illustrated in Figure 7.3.10.



**Figure 7.3.10** The two ways that appendages can overlap.

**Proposition 7.3.8.** Let  $C$  be a cycle in a planar drawing of a graph, and let  $B_1$  and  $B_2$  be overlapping appendages of  $C$ . Then one appendage lies inside cycle  $C$  and the other outside.

**Proof:** Overlapping appendages on the same side of cycle  $C$  would cross, by Corollary 7.1.3 to the Jordan Curve Theorem. ◇

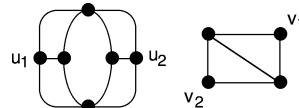
**TERMINOLOGY:** Let  $C$  be a cycle of a connected graph, and suppose that  $C$  has been drawn in the plane. Relative to that drawing, an appendage of  $C$  is said to be *inner* or *outer*, according to whether that appendage is drawn inside or outside of  $C$ .

**Example 7.3.3:** In both parts of Figure 7.3.10, appendage  $B_1$  is an inner appendage and appendage  $B_2$  is an outer appendage.

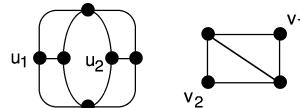
### EXERCISES for Section 7.3

For Exercises 7.3.1 through 7.3.4, the given graphs are to be amalgamated so that each vertex labeled  $u_i$  is matched to the vertex  $v_i$  with the same subscript. Draw a planar imbedding of the resulting graph.

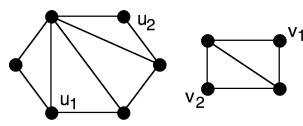
7.3.1<sup>s</sup>



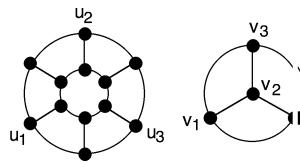
7.3.2



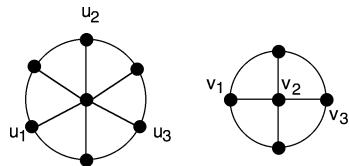
7.3.3



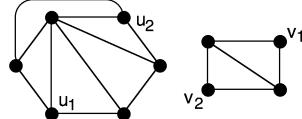
7.3.4



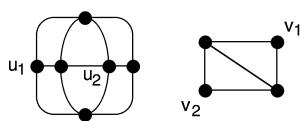
For Exercises 7.3.5 through 7.3.8, the given graphs are to be amalgamated so that each vertex labeled  $u_i$  is matched to the vertex  $v_i$  with the same subscript. Either draw a planar imbedding of the resulting graph or prove that it is nonplanar.

7.3.5<sup>s</sup>

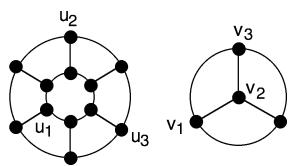
7.3.6



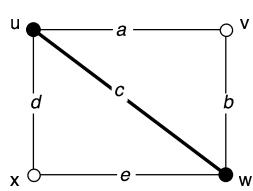
7.3.7



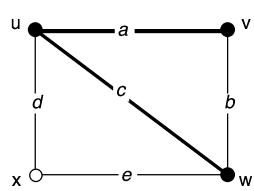
7.3.8



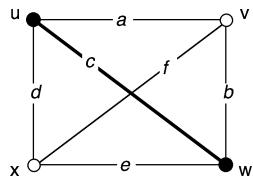
For Exercises 7.3.9 through 7.3.12, relative to the bold subgraph, specify each appendage by listing its vertex-set and its edge-set.

7.3.9<sup>s</sup>

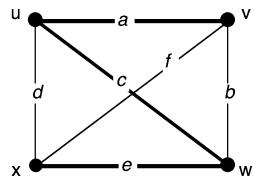
7.3.10



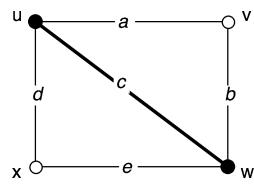
7.3.11



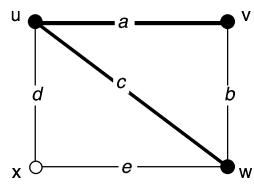
7.3.12



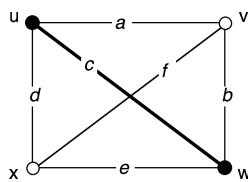
For Exercises 7.3.13 through 7.3.16, a different subgraph of the same graph is indicated by bold vertices and edges. List the contact-point sets of every appendage for each subgraph, and determine which appendages are overlapping.

7.3.13<sup>s</sup>

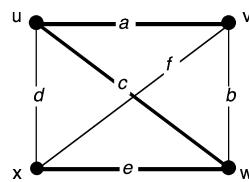
7.3.14



7.3.15



7.3.16



7.3.17 Given two graph imbeddings  $\iota : G \rightarrow S_0$  and  $\iota' : H \rightarrow S_0$  in the sphere, suppose that two vertices  $u_1$  and  $u_2$  are selected from  $G$  that do not lie on the same face boundary of the imbedding of  $G$ , and two vertices  $v_1$  and  $v_2$  are selected from  $H$  that do not lie on the same face boundary of the imbedding of  $H$ . Decide whether the amalgamation of  $G$  and  $H$  across these two pairs of vertices must be nonplanar. Explain your answer.

7.3.18 Show that the operation of subdividing an edge  $e$  of a graph  $G$  can be achieved by adding a new edge between the endpoints of the dual edge  $e^*$  to the dual graph  $G^*$  and then dualizing the result. Show also that the operation of attaching a new edge at a vertex  $v$  of graph  $G$  is dual to the operation of adding a self-loop to  $G^*$ . Then derive Proposition 7.3.3 as a corollary of Proposition 7.3.1.

## 7.4 KURATOWSKI'S THEOREM

One of the landmarks of graph theory is Kuratowski's characterization of planarity in terms of two forbidden subgraphs,  $K_5$  and  $K_{3,3}$ .

**TERMINOLOGY:** Any graph homeomorphic either to  $K_5$  or to  $K_{3,3}$  is called a **Kuratowski subgraph**.

**Theorem 7.4.1. Kuratowski's Theorem [1930]** *A graph is planar if and only if it contains no subgraph homeomorphic to  $K_5$  or to  $K_{3,3}$ .*

**Proof:** Theorems 7.1.4 and 7.1.5 have established that  $K_5$  and  $K_{3,3}$  are both nonplanar. Proposition 7.2.6 implies that a planar graph cannot contain a homeomorphic copy of either. Thus, containing no Kuratowski subgraphs is necessary for planarity. The rest of this section is devoted to proving sufficiency.

If the absence of Kuratowski subgraphs were not sufficient, then there would exist nonplanar graphs with no Kuratowski subgraphs. If there were any such counterexamples, then some counterexample graph would have the minimum number of edges among all counterexamples. The strategy is to derive some properties that this minimum counterexample would have to have, which ultimately establish that it could not exist. The main steps within this strategy are proofs of three statements:

*Step 1:* The minimum counterexample would be simple and 3-connected.

*Step 2:* The minimum counterexample would contain a cycle with three mutually overlapping appendages.

*Step 3:* Any configuration comprising a cycle and three mutually overlapping appendages must contain a Kuratowski subgraph.

### Step 1: A Minimum Counterexample would be Simple and 3-Connected

**Assertion 1.1.** Let  $G$  be a nonplanar connected graph with no Kuratowski subgraph and with the minimum number of edges for any such graph. Then  $G$  is a simple graph.

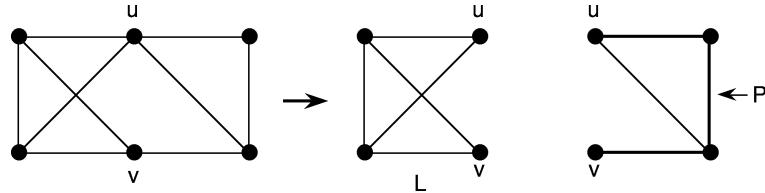
**Proof of Assertion 1.1:** Suppose that  $G$  is not simple. By Corollary 7.3.2, deleting a self-loop or one edge of a multi-edge would result in a smaller nonplanar graph, still with no Kuratowski subgraph, contradicting the minimality of  $G$ .  $\diamond$  (Assertion 1.1)

**Assertion 1.2.** Let  $G$  be a nonplanar connected graph with no Kuratowski subgraph and with the minimum number of edges for any such graph. Then graph  $G$  has no cut-vertex.

**Proof of Assertion 1.2:** If graph  $G$  had a cut-vertex  $v$ , then every appendage of  $v$  would be planar, by minimality of  $G$ . By iterative application of Corollary 7.3.7, graph  $G$  itself would be planar, a contradiction.  $\diamond$  (Assertion 1.2)

**Assertion 1.3.** Let  $G$  be a nonplanar connected graph containing no Kuratowski subgraph with the minimum number of edges for any such graph. Let  $\{u, v\}$  be a vertex-cut in  $G$ , and let  $L$  be a non-chord appendage of  $\{u, v\}$ . Then there is a planar drawing of  $L$  with a face whose boundary contains both vertices  $u$  and  $v$ .

**Proof of Assertion 1.3:** Since  $\{u, v\}$  is a vertex-cut of graph  $G$ , the graph  $G - \{u, v\}$  has at least two components. Thus, the vertex set  $\{u, v\}$  must have at least one more non-chord appendage besides  $L$ . Since, by Assertion 1.2,  $\{u, v\}$  is a minimal cut, it follows that both  $u$  and  $v$  must be contact points of this other non-chord appendage. Since this other non-chord appendage is connected and has no edge joining  $u$  and  $v$ , it must contain a  $u-v$  path  $P$  of length at least 2, as illustrated in Figure 7.4.1.



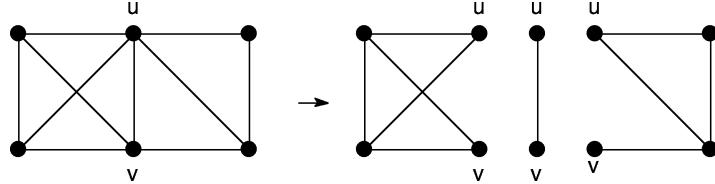
**Figure 7.4.1** A  $u-v$  path  $P$  in an appendage other than  $L$ .

The subgraph  $H = (V_L \cup V_P, E_L \cup E_P)$  (obtained by adding path  $P$  to subgraph  $L$ ) has no Kuratowski subgraph, because it is contained in graph  $G$ , which by premise contains no Kuratowski subgraphs. Let  $d$  be the edge obtained from path  $P$  by smoothing away all the internal vertices. Then the graph  $L + d$  contains no Kuratowski subgraphs (because  $L + d$  is homeomorphic to subgraph  $H$ ). Moreover, the graph  $L + d$  has fewer edges than the minimal counterexample  $G$  (because  $P$  has at least one internal vertex). Thus, the graph  $L + d$  is planar.

In every planar drawing of the graph  $L + d$ , vertices  $u$  and  $v$  both lie on the boundary of each face containing edge  $d$ . Discarding edge  $d$  from any such planar drawing of  $L + d$  yields a planar drawing of appendage  $L$  such that vertices  $u$  and  $v$  lie on the same face (by Proposition 7.3.4).  $\diamond$  (Assertion 1.3)

**Assertion 1.4.** Let  $G$  be a nonplanar connected graph containing no Kuratowski subgraph, with the minimum number of edges for any such graph. Then graph  $G$  has no vertex-cut with exactly two vertices.

**Proof of Assertion 1.4:** Suppose that graph  $G$  has a minimal vertex-cut  $\{u, v\}$ . Clearly, a chord appendage of  $\{u, v\}$  would have a planar drawing with only one face, whose boundary contains both the vertices  $u$  and  $v$ . By Assertion 1.3, every non-chord appendage of  $\{u, v\}$  would also have a planar drawing with vertices  $u$  and  $v$  on the same face boundary. Figure 7.4.2 illustrates the possible decomposition of graph  $G$  into appendages.



**Figure 7.4.2** Decomposition of a graph at vertices  $u$  and  $v$ .

By Proposition 7.3.6, when graph  $G$  is reassembled by iteratively amalgamating these planar appendages at vertices  $u$  and  $v$ , the result is a planar graph.

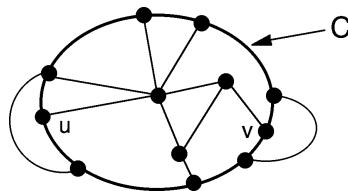
◇ (Assertion 1.4)

**Completion of Step 1.** Let  $G$  be a nonplanar connected graph containing no Kuratowski subgraph and with the minimum number of edges for any such graph. Then graph  $G$  is simple and 3-connected.

**Proof:** This summarizes the four preceding assertions. The connectedness premise serves only to avoid isolated vertices. ◇ (Step 1)

### Step 2: Finding a Cycle with Three Mutually Overlapping Appendages

Let  $G$  be a nonplanar graph containing no Kuratowski subgraph and with the minimum number of edges for any such graph. Let  $e$  be any edge of graph  $G$ , say with endpoints  $u$  and  $v$ , and consider a planar drawing of  $G - e$ . Since graph  $G$  is 3-connected (by Step 1), it follows (see §5.1) that  $G - e$  is 2-connected. This implies (see §5.1) that there is a cycle in  $G - e$  through vertices  $u$  and  $v$ . Among all such cycles, choose cycle  $C$ , as illustrated in Figure 7.4.3, so that the number of edges “inside”  $C$  is as large as possible. The next few assertions establish that cycle  $C$  must have two overlapping appendages in  $G - e$  that both overlap edge  $e$ .



**Figure 7.4.3** Cycle  $C$  has the maximum number of edges inside it.

**Assertion 2.1.** Cycle  $C$  has at least one outer appendage.

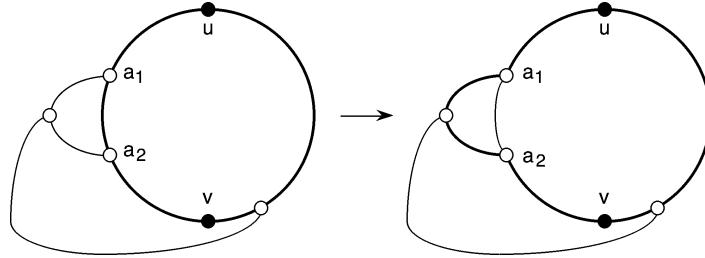
**Proof of Assertion 2.1:** Otherwise, edge  $e$  could be drawn in the outer region, thereby completing a planar drawing of  $G$ . ◇ (Assertion 2.1)

**Assertion 2.2.** Let  $B$  be an appendage of  $C$  that has only two contact points, neither of which is  $u$  or  $v$ . Then appendage  $B$  is a chord.

**Proof of Assertion 2.2:** If  $B$  were a non-chord appendage, then those two contact points of  $B$  would separate vertices  $u$  and  $v$  in  $G$  from the other vertices of  $B$ , which would contradict the 3-connectivity of  $G$ .  $\diamond$  (Assertion 2.2)

**Assertion 2.3.** Let  $d$  be an outer appendage of cycle  $C$ . Then  $d$  is a chord, and its endpoints alternate on  $C$  with  $u$  and  $v$ , so that edges  $e$  and  $d$  are overlapping chords of cycle  $C$ .

**Proof of Assertion 2.3:** Suppose that two contact points  $a_1$  and  $a_2$  of appendage  $d$  do not alternate with vertices  $u$  and  $v$  on cycle  $C$ . Then appendage  $d$  would contain a path  $P$  between vertices  $a_1$  and  $a_2$  with no contact point in the interior of  $P$ . Under such a circumstance, cycle  $C$  could be “enlarged” by replacing its arc between  $a_1$  and  $a_2$  by path  $P$ . The enlarged cycle would still pass through vertices  $u$  and  $v$  and would have more edges inside it than cycle  $C$ , as shown in Figure 7.4.4, thereby contradicting the choice of cycle  $C$ . Moreover, if outer appendage  $d$  had three or more contact points, then there would be at least one pair of them that does not alternate with  $u$  and  $v$ . Thus, appendage  $d$  has only two contact points, and they alternate with  $u$  and  $v$ . By Assertion 2.2, such an appendage is a chord.  $\diamond$  (Assertion 2.3)



**Figure 7.4.4** Using a hypothetical outer appendage to extend cycle  $C$ .

**Assertion 2.4.** There is an inner appendage of cycle  $C$  that overlaps edge  $e$  and also overlaps some outer chord.

**Proof of Assertion 2.4:** One or more inner appendages must overlap chord  $e$ , because otherwise edge  $e$  could be drawn inside cycle  $C$  without crossing any inner appendages, thereby completing a planar drawing of graph  $G$ . Moreover, at least one inner appendage that overlaps  $e$  also overlaps some outer appendage  $d$ . Otherwise, every such inner appendage could be redrawn outside cycle  $C$ , which would permit edge  $e$  to be drawn inside, thereby completing the drawing of  $G$ . By Assertion 2.3, outer appendage  $d$  is necessarily a chord.  $\diamond$  (Assertion 2.4)

**Completion of Step 2.** Let  $G$  be a nonplanar graph containing no Kuratowski subgraph and with the minimum number of edges for any such graph. Then graph  $G$  contains a cycle that has three mutually overlapping appendages, two of which are chords.

**Proof:** The cycle  $C$  selected at the start of Step 2 meets the requirements of this concluding assertion. In particular, one of the appendages of cycle  $C$  is the chord  $e$  designated at the start of Step 2. Assertion 2.4 guarantees the existence of a second

appendage  $B$  that not only overlaps chord  $e$ , but also overlaps some outer chord  $d$ . By Assertion 2.3, chord  $d$  also overlaps chord  $e$ . Thus, the three appendages  $e$ ,  $B$ , and  $d$  are mutually overlapping.  $\diamondsuit$  (Step 2)

### Step 3: Analyzing the Cycle-and-Appendages Configuration

The concluding step in the proof of Kuratowski's theorem is to show that the cycle-and-appendages configuration whose existence is guaranteed by Step 2 must contain a Kuratowski subgraph.

**Step 3.** Let  $C$  be a cycle in a connected graph  $G$ . Let edge  $e$  be an inner chord, edge  $d$  an outer chord, and  $B$  an inner appendage, such that  $e$ ,  $d$ , and  $B$  are mutually overlapping. Then graph  $G$  has a Kuratowski subgraph.

**Proof:** Let  $u$  and  $v$  be the contact points of inner chord  $e$ , and let  $x$  and  $y$  be the contact points of outer chord  $d$ . These pairs of contact points alternate on cycle  $C$ , as shown in Figure 7.4.5. Observe that the union of cycle  $C$ , chord  $e$ , and chord  $d$  forms the complete graph  $K_4$ . There are two cases to consider, according to the location of the contact points of appendage  $B$ .

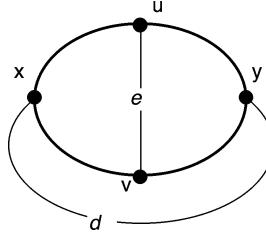


Figure 7.4.5 Forming  $K_4$  with cycle  $C$  and chords  $e$  and  $d$ .

**Case 1.** Suppose that appendage  $B$  has at least one contact point  $s$  that differs from  $u$ ,  $v$ ,  $x$ , and  $y$ .

By symmetry of the  $C-e-d$  configuration, it suffices to assume that contact point  $s$  lies between vertices  $u$  and  $x$  on cycle  $C$ . In order to overlap chord  $e$ , appendage  $B$  must have a contact point  $t$  on the other side of  $u$  and  $v$  from vertex  $s$ . In order to overlap chord  $d$ , appendage  $B$  must have a contact point  $t'$  on the other side of  $x$  and  $y$  from vertex  $s$ .

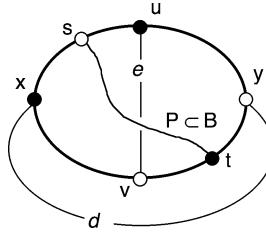
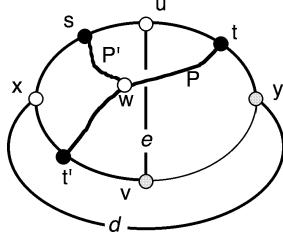


Figure 7.4.6 Subcase 1a.

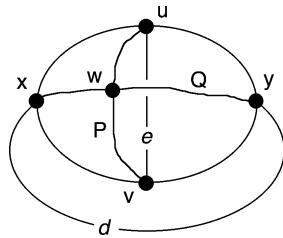
**Subcase 1a.** Contact point  $t$  lies in the interior of the arc between  $v$  and  $y$  on cycle  $C$ , in which case it may be considered that  $t' = t$ , as in Figure 7.4.6. In this subcase, let  $P$  be a path in appendage  $B$  between contact points  $s$  and  $t$ . Then the union of cycle  $C$ , path  $P$ , and chords  $e$  and  $d$  forms a homeomorphic copy of  $K_{3,3}$ .  $\diamondsuit$  (Case 1a)

**Figure 7.4.7** Subcase 1b.

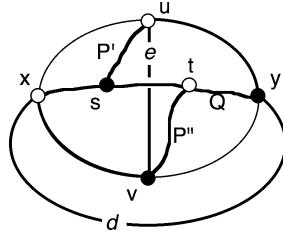
**Subcase 1b.** Contact point  $t$  lies on arc  $uy$ , with  $t \neq u$  (so that appendage  $B$  overlaps chord  $e$ ), and contact point  $t'$  lies on arc  $xv$  with  $t' \neq v$  (so that  $B$  overlaps chord  $d$ ), as in Figure 7.4.7. In this subcase, let  $P$  be a path in appendage  $B$  between contact points  $t'$  and  $t$ . Let  $w$  be an internal vertex on path  $P$  such that there is a  $w-s$  path  $P'$  in  $B$  with no internal vertices in  $P$ . (Such a path  $P'$  exists, because appendage  $B$  is connected.) Then this configuration contains a homeomorph of  $K_{3,3}$  in which each of the vertices  $w$ ,  $x$ , and  $u$  is joined to each of the vertices  $s$ ,  $t$ , and  $t'$ . As is apparent in Figure 7.4.7, it does not matter if  $t = y$  or if  $t' = v$ .  $\diamondsuit$  (Case 1b)

**Case 2.** Appendage  $B$  has no contact points other than  $u, v, x$ , and  $y$ .

Vertices  $x$  and  $y$  must be contact points of  $B$  so that it overlaps chord  $e$ . Vertices  $u$  and  $v$  must be contact points of  $B$ , so that it overlaps chord  $d$ . Thus, all four vertices  $u, v, x$ , and  $y$  must be contact points of appendage  $B$ . Appendage  $B$  has a  $u-v$  path  $P$  and an  $x-y$  path  $Q$  whose internal vertices are not contact points of  $B$ .

**Figure 7.4.8** Subcase 2a.

**Subcase 2a.** Paths  $P$  and  $Q$  intersect in a single vertex  $w$ , as shown in Figure 7.4.8. Then the union of the  $C-e-d$  configuration with paths  $P$  and  $Q$  yields a configuration homeomorphic to  $K_5$ , as illustrated, in which the five mutually linked vertices are  $u, v, y, x$ , and  $w$ .  $\diamondsuit$  (Case 2a)

**Figure 7.4.9** Subcase 2b.

**Subcase 2b.** Paths  $P$  and  $Q$  intersect in more than one vertex.

Let  $s$  be the intersection nearest on path  $P$  to contact point  $u$ , and  $t$  the intersection nearest on  $P$  to contact point  $v$ , as in Figure 7.4.9. Also, let  $P'$  be the  $us$ -subpath of  $P$  and  $P''$  the  $tv$ -subpath of  $P$ . Then the union of chords  $e$  and  $d$  with the paths  $Q, P', P''$ ,

and  $P''$  and with arcs  $vx$  and  $uy$  on cycle  $C$  form a subgraph homeomorphic to  $K_{3,3}$ , with bipartition  $(\{u, x, t\}, \{v, y, s\})$ .  $\diamondsuit$  (Case 2b)  $\diamondsuit$  (Theorem 7.4.1)

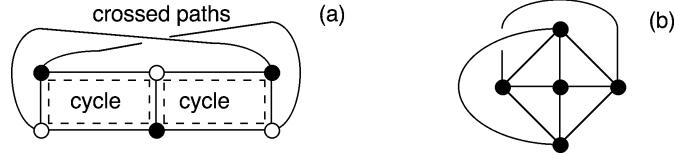
The following Corollary sometimes expedites planarity testing.

**Corollary 7.4.2.** *Let  $G$  be a nonplanar graph formed by the amalgamation of subgraphs  $H$  and  $J$  at vertices  $u$  and  $v$ , such that  $J$  is planar. Then the graph obtained from graph  $H$  by joining vertices  $u$  and  $v$  is nonplanar.*

**Proof:** By Theorem 7.4.1, graph  $G$  must contain a Kuratowski subgraph  $K$ . All the vertices of the underlying Kuratowski graph would have to lie on the same side of the cut  $\{u, v\}$ , because  $K$  is 3-connected. At most, a subdivided edge of the Kuratowski subgraph crosses through  $u$  and back through  $v$ . The conclusion follows.  $\diamondsuit$

### Finding $K_{3,3}$ or $K_5$ in Small Nonplanar Graphs

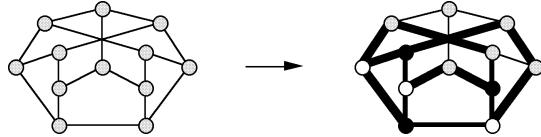
A simple way to find a homeomorphic copy of  $K_{3,3}$  in a small graph is to identify two cycles  $C$  and  $C'$  (necessarily of length at least 4) that meet on a path  $P$ , such that there is a pair of paths between  $C - P$  and  $C' - P$  that “cross”, thereby forming a subdivided Möbius ladder  $ML_3$ , as illustrated in Figure 7.4.10(a).



**Figure 7.4.10** Finding a  $K_{3,3}$  (a) or a  $K_5$  (b) by visual inspection.

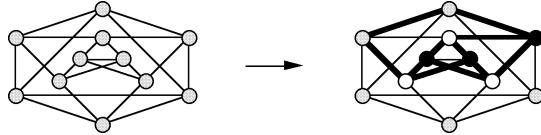
To find a homeomorphic copy of  $K_5$ , look for a 4-wheel with a pair of disjoint paths joining two pairs of vertices that alternate on the rim, as illustrated in Figure 7.4.10(b).

**Example 7.4.1:** The bold edges form paths joining the three white vertices with the three black vertices of Figure 7.4.11, thereby forming a subdivided  $K_{3,3}$ .



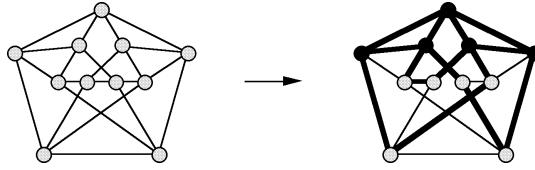
**Figure 7.4.11** Finding a  $K_{3,3}$ , example.

**Example 7.4.2:** The bold edges form paths joining the three white vertices with the three black vertices of Figure 7.4.12.



**Figure 7.4.12** Finding a  $K_{3,3}$ , example.

**Example 7.4.3:** The bold edges form paths joining the five black vertices of Figure 7.4.13, thereby forming a subdivided  $K_5$ .

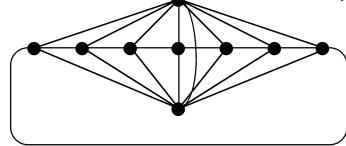


**Figure 7.4.13** Finding a  $K_5$ , example.

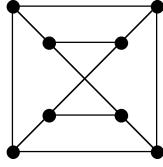
### EXERCISES for Section 7.4

In all of the following exercises, find a Kuratowski subgraph in the given graph.

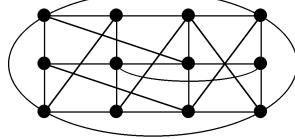
7.4.1



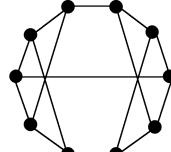
7.4.2<sup>s</sup>



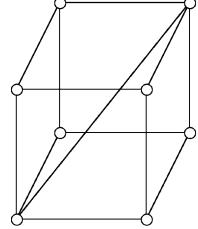
7.4.3



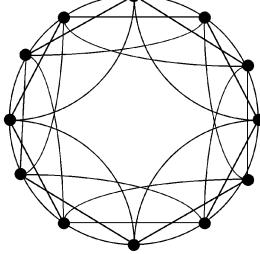
7.4.4



7.4.5



7.4.6



7.4.7<sup>s</sup>  $K_9 - K_{4,5}$ .

7.4.8  $(K_4 - K_2) \times P_3$ .

7.4.9  $Q_3 + K_1$ .

7.4.10  $(K_5 - K_3) \times P_2$ .

7.4.11  $K_2 \times K_4$ .

7.4.12  $C_5 \times C_5$ .

7.4.13  $Q_3 + K_1$ .

7.4.14  $K_7 - C_7$ .

## 7.5 ALGEBRAIC TESTS FOR PLANARITY

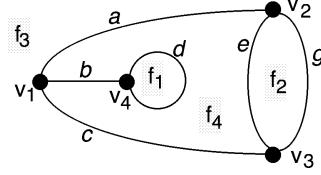
The most basic type of graph-imbedding problem asks whether a particular graph can be imbedded in a particular surface. In this chapter, the surface of particular concern is the sphere (or plane). Transforming such problems into algebraic problems is the inspired approach of algebraic topology. This section derives the fundamental relations used in algebraic analysis of graph-imbedding problems.

The algebraic methods now introduced apply to all surfaces, except that our present version of the Euler polyhedral equation is only for the sphere. This version serves in §8.5 as the base case for an induction that generalizes the equation to all surfaces.

### About Faces

**DEFINITION:** The **face-set of a graph imbedding**  $\iota : G \rightarrow S$  is the set of all faces of the imbedding, formally denoted  $F_\iota$ . Informally, when there is no ambiguity regarding the imbedding, the face-set is often denoted  $F_G$  or  $F$ .

**Example 7.5.1:** The graph imbedding in Figure 7.5.1 has four faces, labeled  $f_1, f_2, f_3$ , and  $f_4$ . Observe that the boundary of face  $f_4$  is not a cycle.



**Figure 7.5.1** A graph imbedding in the sphere.

**DEFINITION:** The **boundary walk of a face**  $f$  of a graph imbedding  $\iota : G \rightarrow S$  is a closed walk in graph  $G$  that corresponds to a complete traversal of the perimeter of the polygonal region within the face.

**Remark:** Vertices and edges can reoccur in a boundary walk. One visualizes one's hand tracing along the edges and vertices in the walk, from just slightly within the region. Thus, if both sides of an edge lie on a single region, the edge is retraced on the boundary walk.

**DEFINITION:** The **size of a face** or **face-size** means the number of edge-steps in the boundary walk (which may be more than the number of edges on the face boundary, due to retracings). A face of size  $n$  is said to be  **$n$ -sided**.

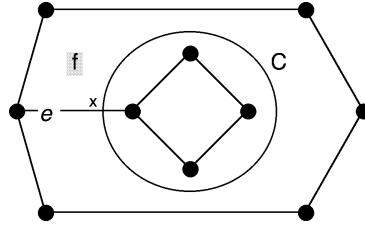
**DEFINITION:** A **monogon** is a 1-sided region.

**DEFINITION:** A **digon** is a 2-sided region.

**Example 7.5.2:** In Figure 7.5.1, face  $f_1$  is 1-sided (a monogon) with boundary walk  $\langle d \rangle$ , face  $f_2$  is 2-sided (a digon) with boundary walk  $\langle e, g \rangle$ , and face  $f_3$  is 3-sided with boundary walk  $\langle a, g, c \rangle$ . Even though only five edges lie on the boundary of face  $f_4$ , face  $f_4$  is 6-sided with boundary walk  $\langle a, b, d, b, c, e \rangle$ . Two sides of face  $f_4$  are pasted together across edge  $b$ , and that edge is a cut-edge of the graph.

**Proposition 7.5.1.** Let  $\iota : G \rightarrow S$  be a planar graph imbedding. Let  $e$  be an edge of  $G$  that occurs twice on the boundary walk of some face  $f$ . Then  $e$  is a cut-edge of  $G$ .

**Proof:** Choose any point  $x$  in the interior of edge  $e$ . Let  $C$  be a curve in face  $f$  from the instance of point  $x$  on one occurrence of edge  $e$  to the instance of  $x$  on the other occurrence of edge  $e$ , as illustrated in Figure 7.5.2.

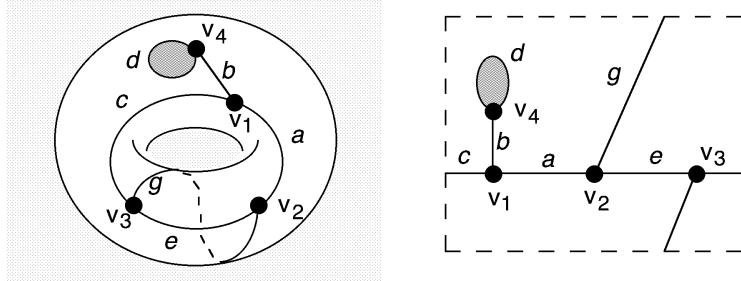


**Figure 7.5.2** A curve  $C$  from one instance of point  $x$  to the other.

Since curve  $C$  starts and stops at the same point  $x$ , it is a closed curve, so it separates the plane (by the Jordan curve theorem). Thus, curve  $C$  separates whatever subgraph of  $G$  lies on one side of curve  $C$  from whatever subgraph lies on the other side. Since curve  $C$  intersects graph  $G$  only in edge  $e$ , it follows that deleting edge  $e$  would separate graph  $G$ . In other words, edge  $e$  is a cut-edge.  $\diamond$

The number of faces of a drawing of  $G$  can vary from surface to surface, as we observe in the following example.

**Example 7.5.3:** When the graph of Figure 7.5.1 is imbedded in the torus, as shown in Figure 7.5.3, the number of faces drops from four to two. One face is the monogon  $\langle d \rangle$  shaded with wavy lines. The other face is 11-sided, with boundary walk  $\langle a, b, d, b, c, e, g, c, a, e, g \rangle$ . The surface is not the plane, so it should not be surprising that even though edge  $a$  occurs twice on the boundary of the 11-sided face, edge  $a$  is not a cut-edge.



**Figure 7.5.3** Two views of the same graph imbedding in the torus.

### Face-Size Equation

**Theorem 7.5.2. Face-Size Equation:** Let  $\iota : G \rightarrow S$  be an imbedding of graph  $G$  into surface  $S$ . Then

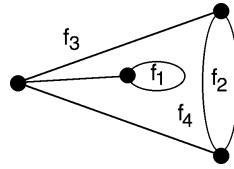
$$2|E_G| = \sum_{f \in F} \text{size}(f)$$

**Proof:** Each edge either occurs once in each of two different face boundary walks or occurs twice in the same boundary walk. Thus, by definition of face-size, each edge contributes two sides to the sum.  $\diamond$

**Example 7.5.4:** The graph in Figure 7.5.4 has six edges, so the value of the left side of the equation is  $2|E| = 12$ . The value of the right side of the equation is

$$\sum_{f \in F} \text{size}(f) = \text{size}(f_1) + \text{size}(f_2) + \text{size}(f_3) + \text{size}(f_4)$$

$$= 1 + 2 + 3 + 6 = 12$$



**Figure 7.5.4** A graph imbedding in the sphere.

### Edge-Face Inequality

**DEFINITION:** A **proper walk** in a graph is a walk in which no edge occurs immediately after itself.

**Proposition 7.5.3.** Every proper closed walk  $W$  in a graph contains a subwalk (possibly  $W$  itself) that is a cycle.

**Proof:** If  $\text{length}(W) = 1$ , then walk  $W$  itself is a cycle. If  $\text{length}(W) = 2$ , then walk  $W$  has the form

$$W = v, d, v', e, v$$

where  $d \neq e$ , since walk  $W$  is proper. If  $v' = v$ , then the subwalk  $v, d, v$  of length 1 is a cycle. Otherwise, walk  $W$  itself is a 2-cycle.

Now suppose that  $\text{length}(W) \geq 3$ . If walk  $W$  contains no repeated vertex, then walk  $W$  itself is a cycle. Otherwise, let  $v$  be a vertex with two occurrences on walk  $W$  such that the subwalk between them contains no repeated occurrences of any vertex and no additional occurrences of  $v$ . Then that subwalk is a cycle.  $\diamond$

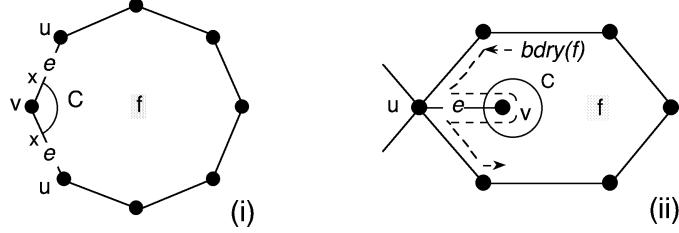
**Proposition 7.5.4.** Let  $f$  be a face of an imbedding  $\iota : G \rightarrow S$  whose boundary walk  $W$  contains no 1-valent vertices and no self-loops. Then  $W$  is a proper closed walk.

**Proof:** Suppose that the improper subwalk

$$\dots, u, e, v, e, w, \dots$$

occurs in boundary walk  $W$ . This means that face  $f$  meets itself on edge  $e$ , and that the two occurrences of  $e$  are consecutive sides of  $f$  when  $f$  is regarded as a polygon. Clearly, the direction of the second traversal of edge  $e$  is opposite to that of the immediately preceding traversal, since  $e$  is not a self-loop. Moreover, since edge  $e$  has only one endpoint other than  $v$ , it follows that  $u = w$ . This situation is illustrated by Figure 7.5.5(i) below.

Consider a closed curve  $C$  in surface  $S$  that begins at some point  $x$  on edge  $e$  near vertex  $v$ , and runs down the end of  $e$  around vertex  $v$  and back out the other side of that end of  $e$  to that same point  $x$  on the other side of edge  $e$ . Since all of curve  $C$  except point  $x$

**Figure 7.5.5** Surface configuration of an improper boundary walk.

lies in the interior of face  $f$ , it follows that no edge other than  $e$  intersects curve  $C$ . Since curve  $C$  separates surface  $S$  (one side is a topological neighborhood of an end of edge  $e$ , and the other is the rest of surface  $S$ ), it follows that no edge except  $e$  is incident on  $v$ , as shown in Figure 7.5.5(ii). Thus, vertex  $v$  is 1-valent, which contradicts the premise.  $\diamond$

We recall that the **girth** of a non-acyclic graph  $G$  is defined to be the length of a smallest cycle in  $G$ .

**Proposition 7.5.5.** *The number of sides of each face  $f$  of an imbedding  $\iota : G \rightarrow S$  of a connected graph  $G$  that is not a tree is at least as large as  $\text{girth}(G)$ .*

**Proof:** (Case 1) If the boundary walk of face  $f$  is a proper walk, then its length is at least as large as the girth, by Proposition 7.5.3.

(Case 2) If the boundary walk of face  $f$  contains a self-loop, then

$$\text{girth}(G) = 1 \leq \text{size}(f)$$

since 1 is the minimum possible size of any face.

(Case 3) By Proposition 7.5.4, the remaining case to consider is when the boundary walk contains one or more 1-valent vertices and no self-loop. In this case, consider the imbedding  $\iota' : G' \rightarrow S'$  that results from iteratively contracting every edge with a 1-valent endpoint, until no such edges remain, with  $f'$  denoting the face that results from deleting all these edges that “spike” into face  $f$ . Then

$$\begin{aligned} \text{size}(f) &\geq \text{size}(f') \\ &\geq \text{girth}(G') \quad \text{by case 1} \\ &= \text{girth}(G) \quad \text{since contracting a cut-edge preserves the girth} \end{aligned}$$

$\diamond$

**Theorem 7.5.6. Edge-Face Inequality:** *Let  $G$  be a connected graph that is not a tree, and let  $\iota : G \rightarrow S$  be an imbedding. Then*

$$2|E| \geq \text{girth}(G) \cdot |F|$$

**Proof:** In the Face-Size Equation

$$2|E| = \sum_{f \in F} \text{size}(f)$$

each term in the sum on the right side is at least  $\text{girth}(G)$ , and there are exactly  $|F|$  terms.  $\diamond$

### Euler Polyhedral Equation

We observe that a tetrahedron has 4 vertices, 6 edges, and 4 faces, and we calculate that  $4 - 6 + 4 = 2$ . We also observe that a cube has 8 vertices, 12 edges, and 6 faces, and that  $8 - 12 + 6 = 2$ .

**Remark:** Robin Wilson [Wi04] offers an historical account of the generalization of this observation. In 1750, Euler communicated the formula  $|V| - |E| + |F| = 2$  by letter to Goldbach. The first proof, in 1794, was by Legendre, who used metrical properties. Lhuilier gave a topological proof in 1811, and he also derived the formula  $|V| - |E| + |F| = 0$  for graphs on a torus. Poincaré generalized the formula to other surfaces in a series of papers of 1895-1904.

**Theorem 7.5.7. Euler Polyhedral Equation:** Let  $\iota : G \rightarrow S$  be an imbedding of a connected graph  $G$  into a sphere. Then

$$|V| - |E| + |F| = 2$$

**Proof:** If the cycle rank  $\beta(G)$  is zero, then graph  $G$  is a tree, because  $\beta(G)$  equals the number of non-tree edges for any spanning tree in  $G$ . Of course, a tree contains no cycles. Thus, in a drawing of graph  $G$  on the sphere, the original single region is not subdivided. It follows that  $|F| = 1$ . Thus,

$$\begin{aligned} |V| - |E| + |F| &= |V| - (|V| - 1) + |F| && \text{by Prop. 3.1.3} \\ &= |V| - (|V| - 1) + 1 &=& 2 \end{aligned}$$

This serves as the base case for an induction on cycle rank. Next suppose that the Euler Polyhedral Equation holds for  $\beta(G) = n$ , for some  $n \geq 0$ .

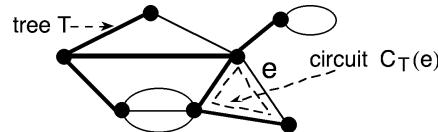
Then consider an imbedding in which graph  $G$  has cycle rank  $\beta(G) = n + 1$ . Let  $T$  be a spanning tree in  $G$ , and let  $e \in E_G - E_T$ , so that  $\beta(G - e) = n$ . Clearly,

$$(1) \quad |V_G| = |V_{G-e}|$$

and

$$(2) \quad |E_G| = |E_{G-e}| + 1$$

Figure 7.5.6 illustrates the relationship between the face sets.



**Figure 7.5.6** Erasing a non-tree edge in a sphere.

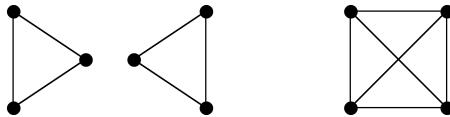
By Proposition 3.1.8, the subgraph  $T + e$  contains a unique cycle,  $C_T(e)$ . By the Jordan curve theorem, the cycle  $C_T(e)$  separates the sphere, which implies that two different faces meet at edge  $e$ . Thus, deleting edge  $e$  from the imbedding merges those two faces. This yields an imbedding of the connected graph  $G - e$  in the sphere. Therefore,

$$(3) \quad |F_G| = |F_{G-e}| + 1$$

Accordingly,

$$\begin{aligned}
 |V_G| - |E_G| + |F_G| &= |V_{G-e}| - |E_G| + |F_G| \quad \text{by (1)} \\
 &= |V_{G-e}| - (|E_{G-e}| + 1) + |F_G| \quad \text{by (2)} \\
 &= |V_{G-e}| - (|E_{G-e}| + 1) + (|F_{G-e}| + 1) \quad \text{by (3)} \\
 &= |V_{G-e}| - |E_{G-e}| + |F_{G-e}| \\
 &= 2 \quad (\text{by the induction hypothesis}) \quad \diamond
 \end{aligned}$$

Theorem 7.5.7 specifies the Euler formula only for an imbedding of a connected graph in the sphere, not for other kinds of drawings. Figure 7.5.7 illustrates two drawings for which the formula does not apply.



**Figure 7.5.7** (a) non-connected graph; (b) drawing with edge-crossings.

**Example 7.5.5:** The non-connected graph  $2K_3$  in Figure 7.5.7(a) has 6 vertices and 6 edges. When it is imbedded in the plane, there are 3 faces, so the Euler formula does not hold. We observe that there is a face (the exterior face) with two boundary components, which cannot happen with a connected graph in the plane. We also observe that the formula does not hold for the drawing in Figure 7.5.7(b), because it has edge-crossings and, thus, is not an imbedding.

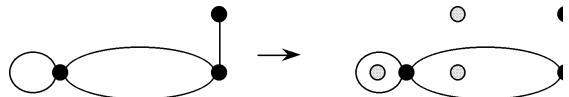
### Poincaré Duality

The French mathematician Henri Poincaré [1900] invented a method by which, for every (*cellular*) imbedding  $\iota : G \rightarrow S$  of any graph  $G$  in any (*closed*) surface  $S$ , one can construct a new imbedding  $\iota^* : G^* \rightarrow S$  of a new graph  $G^*$  in that same surface  $S$ . We defer the formal definition of **cellular imbedding** and **closed surface** to Chapter 8. The sphere is closed, and an imbedding of a connected graph on the sphere is cellular.

**DEFINITION:** Whatever graph imbedding  $\iota : G \rightarrow S$  is to be supplied as input to the duality process (whose definition follows below) is called the **primal graph imbedding**. Moreover, the graph  $G$  is called the **primal graph**, the vertices of  $G$  are called **primal vertices**, the edges of  $G$  are called **primal edges**, and the faces of the imbedding of  $G$  are called **primal faces**.

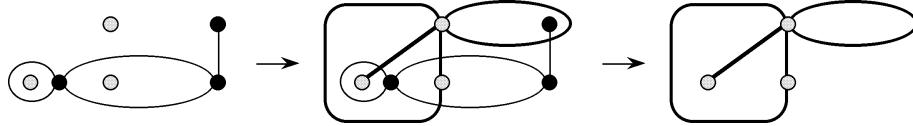
**DEFINITION:** Given a primal graph imbedding  $\iota : G \rightarrow S$ , the **(Poincaré) duality construction** of a new graph imbedding is a two-step process, one for the dual vertices and one for the dual edges:

**dual vertices:** Into the interior of each primal face  $f$ , insert a new vertex  $f^*$  (as in Figure 7.5.8), to be regarded as dual to face  $f$ . The set  $\{f^* \mid f \in F_\iota\}$  is denoted  $V^*$ .



**Figure 7.5.8** Inserting dual vertices into a primal imbedding.

**dual edges:** Through each primal edge  $e$ , draw a new edge  $e^*$  (as in Figure 7.5.9), joining the dual vertex in the primal face on one side of that edge to the dual vertex in the primal face on the other side, to be regarded as dual to edge  $e$ . If the same primal face  $f$  contains both sides of primal edge  $e$ , then dual edge  $e^*$  is a self-loop through primal edge  $e$  from the dual vertex  $f^*$  to itself. The set  $\{e^* \mid e \in E_G\}$  is denoted  $E^*$ .



**Figure 7.5.9** Inserting dual edges completes the dual imbedding.

**DEFINITION:** In Poincaré duality, the **dual graph** is the graph  $G^*$  with vertex-set  $V^*$  and edge-set  $E^*$ . (The vertices and edges of the primal graph are not part of the dual graph.)

**DEFINITION:** In Poincaré duality, the **dual imbedding** is the imbedding  $\iota^* : G^* \rightarrow S$  of the dual graph  $G^*$  that is constructed in the dualization process.

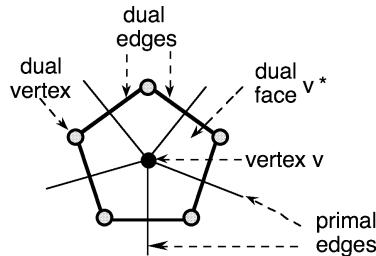
**DEFINITION:** In the Poincaré duality construction, the **dual faces** are the faces of the dual imbedding. The dual face containing the vertex  $v$  is denoted  $v^*$  (For each vertex  $v \in V_G$ , there is only one such face.) The set  $\{v^* \mid v \in V_G\}$  of all dual faces is denoted  $F^*$ .

**NOTATION:** Occasionally, a dual graph (or its vertex-set, edge-set, or face-set) is subscripted to distinguish it from the dual graph of other imbeddings of the same graph.

**Proposition 7.5.8.** *Numerics of Duality:* Let  $\iota : G \rightarrow S$  be a cellular graph imbedding. Then the following numerical relations hold.

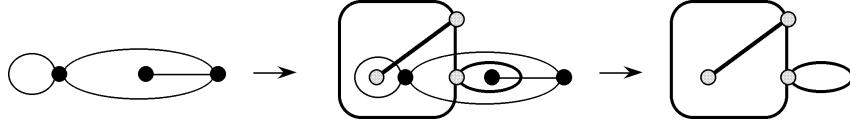
- (i)  $|V^*| = |F|$
- (ii)  $|E^*| = |E|$
- (iii)  $|F^*| = |V|$

**Proof:** Parts (i) and (ii) are immediate consequences of the definition of the Poincaré duality construction. Part (iii) holds because the dual edges that cross the cyclic sequence of edges emanating from a primal vertex  $v$  of graph  $G$  form a closed walk that serves as boundary of the dual face  $v^*$  that surrounds primal vertex  $v$ , as illustrated in Figure 7.5.10.  $\diamond$



**Figure 7.5.10** A dual face surrounding its primal vertex.

**Example 7.5.6:** The same graph may have many different cellular imbeddings in a given surface and many more over the full range of possible surfaces. Figure 7.5.11 shows a different imbedding of the primal graph from the imbedding of Figure 7.5.9. The dual graph of the previous imbedding has degree sequence 1, 2, 5, and the dual graph of Figure 7.5.11 has degree sequence 1, 3, 4, so the dual graphs are not isomorphic.



**Figure 7.5.11** A different dual of the same primal graph.

**Remark:** Applying the Poincaré duality construction to the dual graph imbedding simply reconstructs the primal graph and the primal imbedding. This restoration of the original object through redualization is an essential feature of any “duality” construction.

**Remark:** The Edge-Face Inequality (Theorem 7.5.6) is dual to Euler’s theorem on degree-sum (Theorem 1.1.2).

### Algebraic Proofs of Nonplanarity

The Euler polyhedral equation  $|V| - |E| + |F| = 2$  amounts to an algebraization of the Jordan property. Upon the Euler equation, we can construct inequalities that serve as quick tests for planarity, which we can apply to graphs far too large for ad hoc application of the Jordan property. These are one-way tests, and there are difficult cases in which they are inconclusive.

This algebraic approach to such non-imbeddability proofs illustrates one of the dominant themes of 20th-century mathematics — the transformation of various kinds of other problems, especially topological and geometric problems, into algebraic problems. The formula derived in the following theorem illustrates this theme, by providing a necessary algebraic condition for graph planarity.

**Theorem 7.5.9.** Let  $G$  be any connected simple graph with an imbedding  $\iota : G \rightarrow S_0$  in the sphere or plane. Then

$$|E_G| \leq 3|V_G| - 6$$

**Proof:** The Euler Polyhedral Equation is the starting point.

- |     |   |   |
|-----|---|---|
| (1) | $ V_G  -  E_G  +  F_\iota  = 2$               | Euler Polyhedral Equation                       |
| (2) | $\text{girth}(G) \cdot  F_\iota  \leq 2 E_G $ | Edge-Face Inequality                            |
| (3) | $3 F_\iota  \leq 2 E_G $                      | $G$ simple $\Rightarrow 3 \leq \text{girth}(G)$ |
| (4) | $ F_\iota  \leq \frac{2 E_G }{3}$             |   |
| (5) | $ V_G  -  E_G  + \frac{2 E_G }{3} \geq 2$     | subst. (4) into (1)                             |
| (6) | $ V_G  - \frac{ E_G }{3} \geq 2$              |   |
| (7) | $\frac{ E_G }{3} \leq  V_G  - 2$              |   |
| (8) | $ E_G  \leq 3 V_G  - 6$                       | ◊   |

**Theorem 7.5.10.** *The complete graph  $K_5$  is nonplanar.*

**Proof:** Since  $3|V| - 6 = 9$  and  $|E| = 10$ , the inequality  $|E| > 3|V| - 6$  is satisfied. According to Theorem 7.5.9, the graph  $K_5$  must be nonplanar.  $\diamond$

**Remark:** The Jordan Curve Theorem was used to derive the Euler Polyhedral Equation, which was used, in turn, to derive the planarity criterion  $|E_G| \leq 3|V_G| - 6$ . It remains the ultimate reason why  $K_5$  is not imbeddable in the sphere.

Whereas an ad hoc proof of the nonplanarity of  $K_5$  establishes that fact alone, the inequality of Theorem 7.5.9, derived by the algebraic approach, can be used to prove the nonplanarity of infinitely many different graphs. The next proposition is another application of that inequality.

**Proposition 7.5.11.** *Let  $G$  be any 8-vertex simple graph with  $|E_G| \geq 19$ . Then graph  $G$  has no imbeddings in the sphere or plane.*

**Proof:** Together, the calculation  $3|V_G| - 6 = 3 \cdot 8 - 6 = 18$  and the premise  $|E_G| \geq 19$  imply that  $|E_G| > 3|V_G| - 6$ . By Theorem 7.5.9, graph  $G$  is nonplanar.  $\diamond$

**Remark:** According to Table A1 of [Ha69], there are 838 connected simple 8-vertex graphs with 19 or more edges. Proposition 7.5.11 establishes their nonplanarity collectively with one short proof. An ad hoc approach could not so quickly prove the nonplanarity of even one such graph, much less all of them at once.

### A More Powerful Nonplanarity Condition for Bipartite Graphs

For a bipartite graph, the same algebraic methods used in Theorem 7.5.9 yield an inequality that can sometimes establish nonplanarity when the more general inequality is not strong enough.

**Example 7.5.7:** The graph  $K_{3,3}$  has 6 vertices and 9 edges. Thus,  $|E| \leq 3|V| - 6$ , even though  $K_{3,3}$  is not planar.

**Theorem 7.5.12.** *Let  $G$  be any connected bipartite simple graph with an imbedding  $\iota : G \rightarrow S_0$  in the sphere or plane. Then*

$$|E_G| \leq 2|V_G| - 4$$

**Proof:** This proof is exactly like that of the previous theorem.

- |     |   |  |
|-----|---|--|
| (1) | $ V_G  -  E_G  +  F_\iota  = 2$           | Euler Polyhedral Equation                          |
| (2) | $girth(G) \cdot  F_\iota  \leq 2 E_G $    | Edge-Face Inequality                               |
| (3) | $4 F_\iota  \leq 2 E_G $                  | $G$ simple bipartite $\Rightarrow 4 \leq girth(G)$ |
| (4) | $ F_\iota  \leq \frac{2 E_G }{4}$         |  |
| (5) | $ V_G  -  E_G  + \frac{2 E_G }{4} \geq 2$ | subst. (4) into (1)                                |
| (6) | $ V_G  - \frac{ E_G }{2} \geq 2$          |  |
| (7) | $\frac{ E_G }{2} \leq  V_G  - 2$          |  |
| (8) | $ E_G  \leq 2 V_G  - 4$                   | $\diamond$   |

**Theorem 7.5.13.** *The complete bipartite graph  $K_{3,3}$  is nonplanar.*

**Proof:** Since  $2|V| - 4 = 8$  and  $|E| = 9$ , the inequality  $|E| > 2|V| - 4$  is satisfied. According to Theorem 7.5.12, the graph  $K_{3,3}$  must be nonplanar.  $\diamond$

REVIEW FROM §7.1: The complete graph  $K_5$  and the complete bipartite graph  $K_{3,3}$  are called the **Kuratowski graphs**.

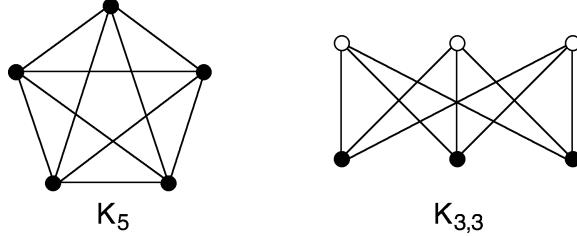


Figure 7.5.12 The Kuratowski graphs.

### Nonplanar Subgraphs

Another way to prove a graph is nonplanar, besides applying Theorem 7.5.9 or Theorem 7.5.12, is to establish that it contains a nonplanar subgraph.

**Theorem 7.5.14.** *If a graph contains a nonplanar subgraph, then that graph is nonplanar.*

**Proof:** Every planar drawing of a graph remains free of edge-crossings after the deletion (e.g., by erasure) of any set of edges and vertices. This establishes the contrapositive, that every subgraph of a planar graph  $G$  is planar.  $\diamond$

**Example 7.5.8:** The graph  $G = K_7 - E(K_4)$  is obtained by deleting the six edges joining any four designated vertices in  $K_7$ . Since  $|V_G| = 7$  and  $|E_G| = 21 - 6 = 15$ , it follows that

$$15 = |E_G| \leq 3|V_G| - 6 = 15$$

Thus, the planarity formula does not rule out the planarity of  $K_7 - E(K_4)$ . However,  $K_7 - E(K_4)$  contains the Kuratowski graph  $K_{3,3}$ , as indicated by Figure 7.5.13. Hence,  $K_7 - E(K_4)$  is nonplanar, by Theorem 7.5.14.

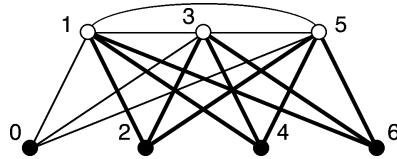


Figure 7.5.13  $K_7 - E(K_4)$  contains  $K_{3,3}$ .

**Proposition 7.5.15.** *Planarity is invariant under subdivision.*

**Proof:** Suppose that  $G$  is a planar graph and that graph  $H$  is homeomorphic to  $G$ . Then, in a planar drawing of  $G$ , smooth whatever vertices are needed and subdivide whatever edges are needed to transform  $G$  into  $H$ . These operations simultaneously transform the planar drawing of  $G$  into a planar drawing of  $H$ .  $\diamond$

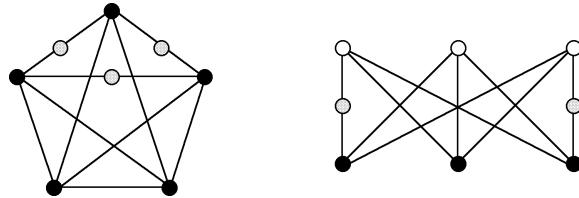
**Proposition 7.5.16.** Suppose that a graph  $G$  contains a subgraph  $H$  that is homeomorphic to a nonplanar graph. Then  $G$  is nonplanar.

**Proof:** This follows immediately from the preceding theorem and proposition.  $\diamond$

As important as the nonplanarity inequalities of Theorem 7.5.9 and Theorem 7.5.12 may be, they provide only one-sided tests. That is, if a graph has too many edges according to either theorem, then it is nonplanar; however, not having too many edges does not imply that the graph is planar. The usual ways to prove planarity are to exhibit a planar drawing or to apply a planarity algorithm, as in §7.6.

**Heuristic 7.5.17.** To construct a nonplanar, non-bipartite graph with a specified number of vertices and edges, start with a Kuratowski graph and add vertices, add edges, and subdivide edges.

**Example 7.5.9:** To construct an 8-vertex nonplanar, non-bipartite graph with at most 13 edges, simply subdivide  $K_5$  or  $K_{3,3}$ , as shown in Figure 7.5.14. Observe that the two graphs resulting from these subdivisions have too few edges to make a conclusive application of the numerical nonplanarity formulas.



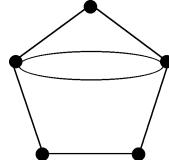
**Figure 7.5.14** Two graphs with too few edges for the nonplanarity formula.

### EXERCISES for Section 7.5

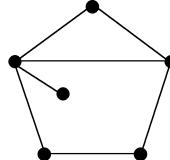
For Exercises 7.5.1 through 7.5.4, do the following for the given graph:

- Redraw the given graph.
- Write the face-size into each region.
- Show that the sum of the face sizes equals twice the number of edges.

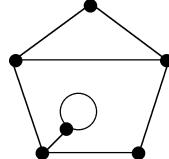
7.5.1



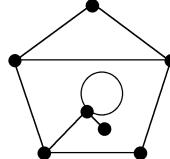
7.5.2<sup>s</sup>



7.5.3



7.5.4



For Exercises 7.5.5 through 7.5.8, do the following for the given graph:

a. Calculate the girth of the graph.

b. Show that the product of the girth and the number of faces is less than or equal to twice the number of edges.

7.5.5 The graph of Exercise 7.5.1.      7.5.6<sup>s</sup> The graph of Exercise 7.5.2.

7.5.7 The graph of Exercise 7.5.3.      7.5.8 The graph of Exercise 7.5.4.

For Exercises 7.5.9 through 7.5.12, do the following for the given graph:

a. Calculate  $|V|$ ,  $|E|$ , and  $|F|$ .

b. Show that the Euler polyhedral equation  $|V| - |E| + |F| = 2$  holds.

7.5.9 The graph of Exercise 7.5.1.      7.5.10<sup>s</sup> The graph of Exercise 7.5.2.

7.5.11 The graph of Exercise 7.5.3.      7.5.12 The graph of Exercise 7.5.4.

For Exercises 7.5.13 through 7.5.16, draw the Poincaré dual of the given planar graph imbedding.

7.5.13 The graph of Exercise 7.5.1.      7.5.14<sup>s</sup> The graph of Exercise 7.5.2.

7.5.15 The graph of Exercise 7.5.3.      7.5.16 The graph of Exercise 7.5.4.

For Exercises 7.5.17 through 7.5.20, draw two more additional imbeddings of the given imbedded graph, so that the degree sequences of all three duals (including the given imbedding) of the indicated graph are mutually distinct.

7.5.17 The graph of Exercise 7.5.1.      7.5.18<sup>s</sup> The graph of Exercise 7.5.2.

7.5.19 The graph of Exercise 7.5.3.      7.5.20 The graph of Exercise 7.5.4.

**DEFINITION:** A graph  $G$  is **self-dual** in the sphere if there is an imbedding of  $G$  such that the Poincaré dual graph  $G^*$  for that imbedding is isomorphic to  $G$ .

For Exercises 7.5.21 through 7.5.26, either draw an imbedding establishing self-duality of the given graph, or prove that no such imbedding is possible.

7.5.21  $K_4$ .      7.5.22<sup>s</sup>  $K_4 - K_2$ .      7.5.23  $D_2$ .

7.5.24  $K_5 - K_2$ .      7.5.25  $B_1 + K_1$ .      7.5.26  $K_6 - 3K_2$ .

For Exercises 7.5.27 through 7.5.32, prove that the given graph is nonplanar.

7.5.27  $K_8 - 4K_2$ .      7.5.28  $K_{3,6} - 3K_2$ .      7.5.29<sup>s</sup>  $K_8 - 2C_4$ .

7.5.30  $K_8 - K_{3,3}$ .      7.5.31  $K_2 + Q_3$ .      7.5.32  $K_{6,6} - C_{12}$ .

For Exercises 7.5.33 through 7.5.40, either draw a plane graph that meets the given description or prove that no such graph exists.

7.5.33<sup>s</sup> A simple graph with 6 vertices and 13 edges.

7.5.34 A non-simple graph with 6 vertices and 13 edges.

7.5.35 A bipartite simple graph with 7 vertices and 11 edges.

7.5.36 A bipartite non-simple graph with 7 vertices and 11 edges.

7.5.37 A simple planar graph with 144 vertices and 613 edges.

7.5.38 A simple planar graph with 1728 vertices and 5702 edges.

7.5.39 A bipartite simple graph with 36 vertices and 68 edges.

7.5.40 A planar simple graph with 36 vertices and 102 edges.

7.5.41 Draw a bipartite graph  $G$  with 15 vertices and 18 edges that is not planar, even though it satisfies the formula  $|E_G| \leq 2|V_G| - 4$ .

For Exercises 7.5.42 through 7.5.45, show that the given graph satisfies the planarity inequality  $|E_G| \leq 3|V_G| - 6$ , and that it is nonetheless nonplanar.

7.5.42<sup>s</sup>  $K_{3,3} + K_1$ .

7.5.44  $K_8 - (K_5 \cup K_3)$ .

7.5.43  $K_{3,3} \times K_2$ .

7.5.45  $W_5 \times C_4$ .

## 7.6 PLANARITY ALGORITHM

Several good planarity algorithms have been developed. The planarity algorithm described here (from [DeMaPe64]) has been chosen for simplicity of concept and of implementability. It starts by drawing a cycle, and adds to it until the drawing is completed or until further additions would force an edge-crossing in the drawing. This section first gives a prose description of this algorithm, then pseudocode for the algorithm.

### Blocked and Forced Appendages of a Subgraph

The body of the main loop of the planarity algorithm attempts to extend a subgraph already drawn in the plane by choosing a plausible appendage of that subgraph to be used to augment the drawing.

**DEFINITION:** Let  $H$  be a subgraph of a graph. In a planar drawing of  $H$ , an appendage of  $H$  is **undrawable in region**  $R$  if the boundary of region  $R$  does not contain all the contact points of that appendage.

**DEFINITION:** Let  $H$  be a subgraph of a graph. In a planar drawing of  $H$ , an appendage of  $H$  is **blocked** if that appendage is undrawable in every region.

**Example 7.6.1:** In Figure 7.6.1, a subgraph isomorphic to  $K_{2,3}$  is shown with black vertices and solid edges, and its only appendage is shown with a white vertex and broken edges. The appendage is blocked, because no region boundary contains all three contact points.

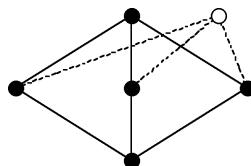


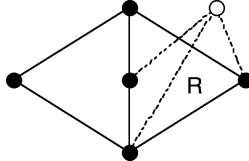
Figure 7.6.1 A subgraph with a blocked appendage.

**Proposition 7.6.1.** Let  $\iota : H \rightarrow S_0$  be a planar drawing of a subgraph  $H$  of a graph  $G$ , such that  $H$  has a blocked appendage  $B$ . Then it is impossible to extend  $\iota : H \rightarrow S_0$  to a planar drawing of  $G$ .

**Proof:** Let  $u$  and  $v$  be two contact points of  $B$  such that no face boundary contains them both. Then the first and last edges of a path in  $B$  joining  $u$  and  $v$  in the drawing of  $H$  would be in different regions of the drawing. By the Jordan curve theorem, the path would have to cross some edge of  $H$ .  $\diamond$

**DEFINITION:** Let  $H$  be a subgraph of a graph. In a drawing of  $H$  on any surface, an appendage of  $H$  is **forced into region  $R$**  if  $R$  is the only region whose boundary contains all the contact points of that appendage.

**Example 7.6.2:** In Figure 7.6.2, a subgraph isomorphic to  $K_{2,3}$  is shown with black vertices and solid edges, and its only appendage is shown with a white vertex and broken edges. The appendage is forced into region  $R$ , because it is the only region whose boundary contains all three contact points.



**Figure 7.6.2** A subgraph with a forced appendage.

### Algorithmic Preliminaries

By iterative application of Corollary 7.3.7, a graph is planar if and only if all its blocks are planar. Performing a prior decomposition into blocks (see §5.4) frees the main part of the algorithm from a few cluttersome details, and permits a focus on the most interesting case, in which the graph to be tested for planarity is 2-connected.

In testing the planarity of a 2-connected graph  $G$ , the basis for a Whitney synthesis (see §5.2) is to find a cycle and to draw it in the plane. (If there is no cycle, then the graph is a tree, and it is planar.) The cycle is designated as the basis subgraph  $G_0$ .

### Selecting the Next Path Addition

Before each iteration of the body of the main loop, there is a two-part exit test. One exit condition is that the entire graph is already drawn in the plane, in which case it is decided, of course, that the graph is planar. The other exit condition is that some appendage of subgraph  $G_j$  is blocked. If so, then graph  $G$  is declared nonplanar and the algorithm terminates. If neither exit is taken, then there are two possible cases of appendage selection.

In the main loop, graph  $G_{j+1}$  is obtained from graph  $G_j$  by choosing an appendage of  $G_j$  and adding a path joining two contact points of that appendage to the planar drawing of  $G_j$ . The main technical concern is that of choosing an appropriate appendage at each iteration.

In the first case, it is determined that some appendage is forced. If so, then a path between two of its contact points is drawn into the one region whose boundary contains all contact points of that forced appendage, thereby extending the drawing of  $G_j$  to a drawing of  $G_{j+1}$ .

In the second case, there are no forced appendages. Under this circumstance, an arbitrary appendage is chosen, and a path between two of its contact points is drawn into any region whose boundary contains all contact points of that appendage, thereby extending the drawing of  $G_j$  to a drawing of  $G_{j+1}$ .

After extension to a drawing of  $G_{j+1}$ , the loop returns to the two-fold exit test and possibly continues with the next attempt at extending the drawing.

### The Algorithm

Naively searching for a homeomorphic copy of  $K_5$  or of  $K_{3,3}$  in an  $n$ -vertex graph would require exponentially many steps. The planarity algorithm given here requires  $O(n^2)$  execution steps.

#### Algorithm 7.6.1: Easy Planarity-Testing for a 2-Connected Graph

*Input:* a 2-connected graph  $G$   
*Output:* a planar drawing of  $G$ , or the decision FALSE.  
{Initialize} Find an arbitrary cycle  $G_0$  in  $G$ , and draw it in the plane.  
While  $G_j \neq G$  {this exit implies that  $G$  is planar}  
    If any appendage is blocked, then return (FALSE).  
    If some appendage is forced, then  $B :=$  that appendage  
        else  $B :=$  any appendage whatever.  
     $R :=$  any region containing all contact points of  $B$  on  $bd(R)$ .  
    Select any path between two contact points of  $B$ .  
    Draw that path into region  $R$  to obtain  $G_{j+1}$ .  
    Continue with next iteration of while-loop.  
{End of while-loop body}  
Return (planar drawing of  $G$ )

### Outline of Correctness of Planarity Algorithm

To establish the correctness of this algorithm, there is a two-step proof that when no appendage is forced, the particular choice of an appendage to extend the subgraph  $G_j$  and a region in which to draw it does not affect the eventual decision of the algorithm. The first step, of identifying the configuration under which there is an appendage  $A$  with no forced choice between two regions  $R$  and  $R'$ , is illustrated in Figure 7.6.3.

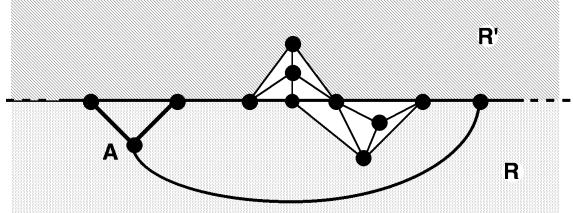


Figure 7.6.3 Appendage  $A$  has option of region  $R$  or region  $R'$ .

The second step is to demonstrate that regardless of which region is chosen, the eventual algorithmic decision on planarity is the same. For further details, see §9.8 of [BoMu76] or §6.3 of [We01].

Hopcroft and Tarjan [HoTa74] have published a linear-time planarity algorithm. Topological arguments of Gross and Rosen [GrRo81] enabled them to achieve a linear-time reduction of planarity-testing for *2-complexes* [GrRo79] to a planarity-test for graphs.

### EXERCISES for Section 7.6

For Exercises 7.6.1 through 7.6.12, apply Algorithm 7.6.1 to the designated graph.

- |        |   |                    |                              |
|--------|---|--------------------|------------------------------|
| 7.6.1  | The graph of Exercise 7.4.1.                                | 7.6.2 <sup>s</sup> | The graph of Exercise 7.4.2. |
| 7.6.3  | The graph of Exercise 7.4.3.                                | 7.6.4              | The graph of Exercise 7.4.4. |
| 7.6.5  | The graph of Exercise 7.4.5.                                | 7.6.6              | The graph of Exercise 7.4.6. |
| 7.6.7  | $K_{3,3}$ .   | 7.6.8              | $K_5$ .                      |
| 7.6.9  | $Q_3 + K_1$ .   | 7.6.10             | $(K_4 - K_2) \times P_3$ .   |
| 7.6.11 | $K_9 - K_{4,5}$ .   | 7.6.12             | $(K_5 - K_3) \times P_2$ .   |
| 7.6.13 | [Computer Project] Implement Algorithm 7.6.1 on a computer. |                    |                              |
- 

## 7.7 CROSSING NUMBERS AND THICKNESS

Two quantifications of the nonplanarity of a graph involve only the plane, without going to higher-order surfaces. One is to determine the minimum number of edge-crossings needed in a drawing of that graph in the plane. The other partitions the edge-set of a graph into “layers” of planar subgraphs and establishes the minimum number of layers needed.

### Crossing Numbers

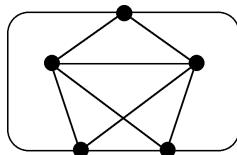
Although it is easy to count the edge-crossings in a graph drawing, it is usually very difficult to prove that the number of crossings in a particular drawing is the smallest possible. The context of most such calculations is that the drawing has been *normalized* (see §8.3) so that no edge crosses another more than once and that at most two edges cross at any point.

**DEFINITION:** Let  $G$  be a simple graph. The **crossing number**  $\nu(G)$  is the minimum number of edge-crossings that can occur in a normal drawing of  $G$  in the plane.

The usual way to calculate the crossing number of a graph  $G$  has two parts. A lower bound  $\nu(G) \geq b$  is established by a theoretical proof. An upper bound  $\nu(G) \leq b$  is established by exhibiting a drawing of  $G$  in the plane with  $b$  crossings. If one is able to draw the given graph with only one edge-crossing, then calculating the crossing number reduces to a planarity test.

**Proposition 7.7.1.**  $\nu(K_5) = 1$ .

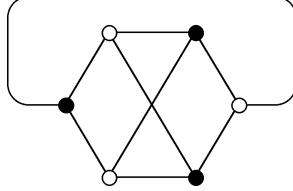
**Proof:** By Theorem 7.1.4, every drawing of  $K_5$  in the plane must have at least one crossing. Thus,  $\nu(K_5) \geq 1$ . Since the drawing of  $K_5$  in Figure 7.7.1 has only one crossing, it follows that  $\nu(K_5) \leq 1$ .  $\diamond$



**Figure 7.7.1** A drawing of  $K_5$  in the plane with only one crossing.

**Proposition 7.7.2.**  $\nu(K_{3,3}) = 1$ .

**Proof:** By Theorem 7.1.5, it follows that  $\nu(K_{3,3}) \geq 1$ . The drawing of  $K_{3,3}$  in Figure 7.7.2 with only one crossing implies that  $\nu(K_{3,3}) \leq 1$ .  $\diamond$



**Figure 7.7.2** A drawing of  $K_{3,3}$  in the plane with only one crossing.

### Lower Bounds for Crossing Numbers

The main interest in crossing numbers of graphs is for cases in which the minimum number is more than 1.

**Theorem 7.7.3.** Let  $G$  be a connected simple graph. Then

$$\nu(G) \geq |E_G| - 3|V_G| + 6$$

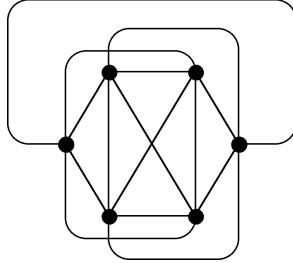
**Proof:** Let  $H$  be a planar spanning subgraph of  $G$  with the maximum number of edges. Theorem 7.5.9 implies that  $|E_H| \leq 3|V_H| - 6$ , from which it follows that  $|E_H| \leq 3|V_G| - 6$ , since  $V_H = V_G$ . Since  $H$  is maximum planar, we infer that no matter how each of the remaining  $|E_G| - |E_H|$  edges is added to a planar drawing of  $H$ , it crosses at least one edge of  $H$ . Thus,  $\nu(G) \geq |E_G| - |E_H| \geq |E_G| - 3|V_G| + 6$ .  $\diamond$

**Proposition 7.7.4.**  $\nu(K_6) = 3$ .

**Proof:** Since  $|E(K_6)| = 15$  and  $|V(K_6)| = 6$ , Theorem 7.7.3 implies that

$$\nu(K_6) \geq 15 - 3 \cdot 6 + 6 = 3$$

The drawing of  $K_6$  in Figure 7.7.3 with three crossings establishes that  $\nu(K_6) \leq 3$ .  $\diamond$



**Figure 7.7.3** A drawing of  $K_6$  in the plane with three crossings.

**Theorem 7.7.5.** Let  $G$  be a connected simple bipartite graph. Then

$$\nu(G) \geq |E_G| - 2|V_G| + 4$$

**Proof:** Let  $H$  be a planar spanning subgraph of  $G$  with the maximum number of edges. Theorem 7.5.12 implies that  $|E_H| \leq 2|V_H| - 4$ , from which it follows that  $|E_H| \leq 2|V_G| - 4$ , since  $V_H = V_G$ . Since  $H$  is maximum planar, it follows that no

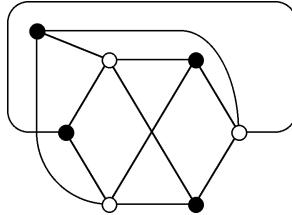
matter how each of the remaining  $|E_G| - |E_H|$  edges is added to a planar drawing of  $H$ , it crosses at least one edge of  $H$ . Thus,  $\nu(G) \geq |E_G| - |E_H| \geq |E_G| - 2|V_G| + 4$ .  $\diamond$

**Proposition 7.7.6.**  $\nu(K_{3,4}) = 2$ .

**Proof:** Since  $|E(K_{3,4})| = 12$  and  $|V(K_{3,4})| = 7$ , Theorem 7.7.5 implies that

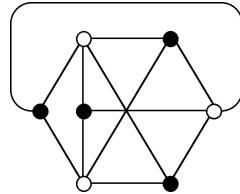
$$\nu(K_{3,4}) \geq 12 - 2 \cdot 7 + 4 = 2$$

The drawing of  $K_{3,4}$  in Figure 7.7.4 with two crossings implies that  $\nu(K_{3,4}) \leq 2$ .  $\diamond$



**Figure 7.7.4** A drawing of  $K_{3,4}$  in the plane with two crossings.

**Example 7.7.1:** The requirement that no more than two edges cross at the same intersection is necessary for these results to hold. Figure 7.7.5 shows that if a triple-intersection is permitted, then  $K_{3,4}$  can be drawn in the plane with only one crossing.



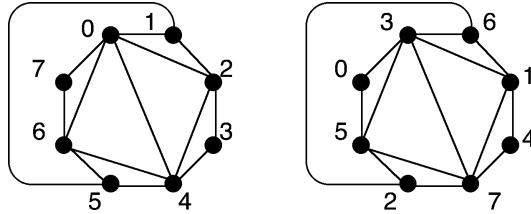
**Figure 7.7.5** A drawing of  $K_{3,4}$  in the plane with one triple-crossing.

### Thickness

**Application 7.7.1 Partitioning a Network into Planar Layers:** One technique involved in miniaturizing a nonplanar electronic network is to sandwich layers of insulation between planar layers of uninsulated wires, which connect nodes that pierce through all the layers. Minimizing the number of layers tends to reduce the size of the chip. Moreover, in applying the technique to mass production, minimizing the number of layers tends to reduce the cost of fabrication.

**DEFINITION:** The **thickness** of a simple graph  $G$ , denoted  $\theta(G)$ , is the smallest cardinality of a set of planar spanning subgraphs of  $G$  whose edge-sets partition  $E_G$ .

**Example 7.7.2:** The two planar graphs in Figure 7.7.6 below both span  $K_8$ . The union of their edge-sets is  $E_{K_8}$ . Thus, a chip implementing  $K_8$  would need only two planar layers.



**Figure 7.7.6** Two planar spanning subgraphs of  $K_8$ .

**Theorem 7.7.7.** Let  $G$  be a connected simple graph. Then

$$\theta(G) \geq \left\lceil \frac{|E_G|}{3|V_G| - 6} \right\rceil$$

**Proof:** By Theorem 7.5.9, at most  $3|V_G| - 6$  edges lie in a planar subgraph of  $G$ .  $\diamond$

**Proposition 7.7.8.**  $\theta(K_8) = 2$ .

**Proof:** By Theorem 7.7.7,  $\theta(K_8) \geq 2$ . Figure 7.7.6 shows two planar graphs whose union is  $K_8$ , which implies that  $\theta(K_8) \leq 2$ .  $\diamond$

**Theorem 7.7.9.** Let  $G$  be a connected simple bipartite graph. Then

$$\theta(G) \geq \left\lceil \frac{|E_G|}{2|V_G| - 4} \right\rceil$$

**Proof:** By Theorem 7.5.12, at most  $2|V_G| - 4$  edges lie in a planar subgraph of  $G$ .  $\diamond$

### Straight-Line Drawings of Graphs

In circuit design (either single-layer or multi-layer), it is convenient for wires to be straight-line segments, rather than having curves or bends.

**DEFINITION:** A **straight-line drawing** of a graph is a planar drawing in which every edge is represented by a straight-line segment.

Wagner [Wa36] and Fary [Fa48] proved the following result, which is known as Fary's theorem. Thomassen [Th80, Th81] developed a proof of Kuratowski's theorem that simultaneously yields Fary's Theorem.

**Theorem 7.7.10.** Let  $G$  be a simple planar graph. Then  $G$  has a straight-line drawing without crossings.  $\diamond$

### EXERCISES for Section 7.7

**7.7.1<sup>s</sup>** Derive this lower bound for the crossing number of the complete graph  $K_n$ .

$$\nu(K_n) \geq \left\lceil \frac{(n-3)(n-4)}{2} \right\rceil$$

**7.7.2** Derive this lower bound for the crossing number of the complete bipartite graph  $K_{m,n}$ .

$$\nu(K_{m,n}) \geq (m-2)(n-2)$$

7.7.3 Draw the Petersen graph in the plane with 2 (normal) crossings.

7.7.4 Draw  $K_7$  in the plane with 9 (normal) crossings.

*Exercises 7.7.5 through 7.7.8 require calculating the crossing number of a given graph.*

7.7.5<sup>s</sup>  $\nu(K_{4,4})$ .

7.7.6  $\nu(K_{4,5})$ .

7.7.7  $\nu(Q_3 + K_1)$ .

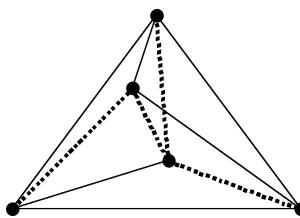
7.7.8  $\nu(C_4 + K_2)$ .

7.7.9 Prove that  $\theta(K_{m,n}) \geq \left\lceil \frac{mn}{2(m+n-2)} \right\rceil$ .

7.7.10<sup>s</sup> Prove that  $\theta(K_{6,6}) = 2$ .

7.7.11 Prove that  $\theta(K_9) = 3$ .

**Application 7.7.1, continued:** A straightforward approach to multi-layer circuit design is to use “simultaneous straight-line drawings” of the spanning subgraphs that induce the edge-set partition. This means that the nodes in each layer are in a fixed location in the plane, so that each layer is given a straight-line drawing. Figure 7.7.7 shows a two-layer partition of the complete graph  $K_5$  by simultaneous straight-line drawings.



**Figure 7.7.7** Two-layer straight-line partition of  $K_5$ .

*Exercises 7.7.12 through 7.7.15 require partitioning the given graph by simultaneous straight-line drawings of spanning subgraphs into the prescribed number of layers.*

7.7.12<sup>s</sup> Partition  $K_6$  into two simultaneous straight-line layers.

7.7.13 Partition  $K_{4,4}$  into two simultaneous straight-line layers.

7.7.14 Partition  $K_8 - K_4$  into three simultaneous straight-line layers.

7.7.15 Partition  $K_8$  into four simultaneous straight-line layers.

## 7.8 SUPPLEMENTARY EXERCISES

7.8.1 Consider a plane drawing of an  $n$ -vertex simple graph  $G$  with fewer than  $3n - 6$  edges.  
a. Use the Euler Polyhedral Equation and the Edge-Face Inequality to prove that there exists a face  $f$  with more than 3 sides.  
b. Use the Jordan curve theorem to prove that there are two non-consecutive vertices on the boundary of face  $f$  that are not adjacent in graph  $G$ .

7.8.2 Draw a self-dual imbedding of a planar simple graph with degree sequence  $\langle 3, 3, 3, 3, 4, 4, 4 \rangle$ .

7.8.3 Draw a non-self-dual imbedding of a planar simple graph with degree sequence  $\langle 3, 3, 3, 3, 4, 4, 4 \rangle$ .

**7.8.4** A connected graph  $G$  with  $p$  vertices and  $q$  edges is imbedded in the sphere. Its dual is drawn. The dual vertex in each primal face is joined to every vertex on the face boundary, and every crossing of a primal and dual edge is converted into a new vertex, so that the result is a graph  $G^\#$  imbedded in the sphere. a. How many vertices and edges does  $G^\#$  have? b. For  $G = C_3$ , draw the edge-complement of  $G^\#$ .

**7.8.5** Draw a 4-regular simple 9-vertex planar graph.

**7.8.6** Suppose that a simple graph has degree sequence  $\langle 5, 5, 5, 5, 5, 5, 4, 4 \rangle$ . Prove that it cannot be planar.

**7.8.7** Prove that every simple planar graph with at least 4 vertices has at least 4 vertices of degree less than 6.

**7.8.8** Can a 3-connected simple graph of girth 3 be drawn in the plane so that every face has at least four sides? Either give an example or prove it is impossible.

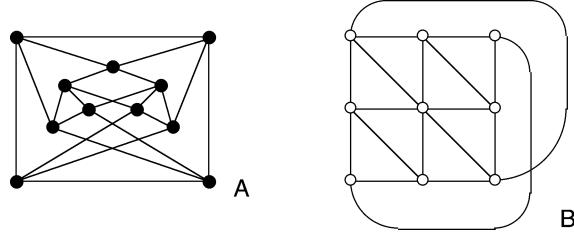
**7.8.9** Prove that the Petersen graph has no 3-cycles or 4-cycles. Then use the Euler Polyhedral Equation and the Edge-Face Inequality to show that the Petersen graph is nonplanar.

**7.8.10** Let  $H$  be a 4-regular simple graph, and let  $G$  be the graph obtained by joining  $H$  to  $K_1$ . Decide whether  $G$  can be planar. Either give an example or prove impossibility.

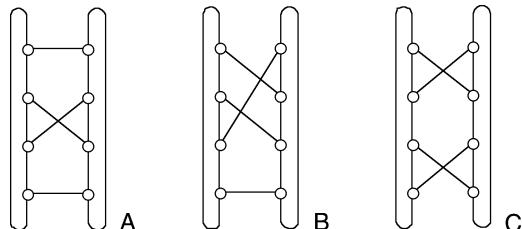
**7.8.11** Draw each of the four isomorphism types of simple 6-vertex graph that contains  $K_5$  homeomorphically, but does not contain  $K_{3,3}$ .

**7.8.12** a. Draw a 9-vertex tree  $T$  such that  $K_2 + T$  is planar. b. Prove that there is only one such tree.

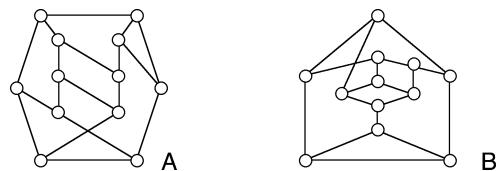
**7.8.13** Decide whether the following graphs contain a homeomorphic copy of  $K_{3,3}$ , of  $K_5$ , of both, or of neither.



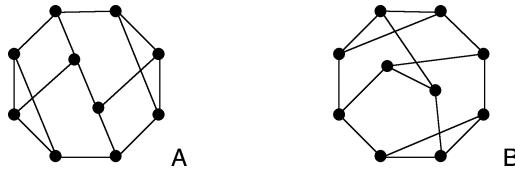
**7.8.14** Decide which of the graphs below are planar.



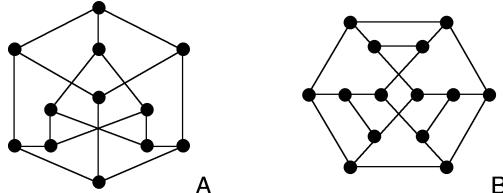
**7.8.15** Decide which of the graphs below are planar.



- 7.8.16** a. Show how to add one edge to graph A so that the resulting graph is nonplanar (and prove nonplanarity). b. Show how to delete one edge from graph B so that the resulting graph is nonplanar.



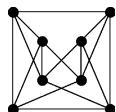
- 7.8.17** Decide which of the graphs below are planar.



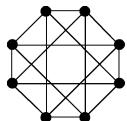
- 7.8.18** Use the planarity algorithm to show that the circulant graph  $\text{circ}(7 : 1, 2)$  is nonplanar. Find a Kuratowski subgraph.

- 7.8.19** Use the planarity algorithm to show that the circulant graph  $\text{circ}(9 : 1, 3)$  is nonplanar. Find a Kuratowski subgraph.

- 7.8.20** Redraw the following graph with only two crossings.



- 7.8.21** Calculate the crossing number of the following graph.



- 7.8.22** Calculate the crossing number of  $K_6 - K_2$ .

- 7.8.23** In a planar drawing of  $K_6 - K_2$ , in which (as usual) at most two edges cross at a point, what is the minimum number of edges that must cross at least one other edge. (Hint: be careful; for  $K_{3,3}$ , the answer is two – not one, since two edges cross at the single crossing.)

- 7.8.24** Suppose that a triple edge-crossing were permitted. Prove that even then, it would still be impossible to draw  $K_6 - K_2$  in the plane with only one crossing. (Hint: The graph obtained by inserting a new vertex at the triple-intersection point would have to be planar.)

- 7.8.25** Prove that  $\theta(K_n) \geq \lfloor \frac{n+7}{6} \rfloor$ .

- 7.8.26** Calculate  $\theta(K_{12})$ .

- 7.8.27** Redraw the graph at the left in Figure 7.7.6 so that its vertices are arranged in the plane as shown at the right in Figure 7.7.6. (Suggestion: First read about *rotations* in §16.1. In redrawing the graph, be careful to preserve the rotation at each vertex. This requires a lot of winding about.)

---

**GLOSSARY**

**appendage to a subgraph  $H$ :** the induced subgraph on a set of edges such that each pair lies on a path with no internal vertices in  $H$  and is maximal with respect to this property.

—, **blocked:** for a subgraph  $H$  drawn on a surface, an appendage that is undrawable in every region.

—, **forced** into region  $R$ : an appendage  $B$  such that  $R$  is the only region whose boundary contains all the contact points of  $B$ .

—, **overlapping pair** for a cycle  $C$ : appendages  $B_1$  and  $B_2$  such that either of these conditions holds:

(i) Two contact points of  $B_1$  alternate with two contact points of  $B_2$  on cycle  $C$ .

(ii)  $B_1$  and  $B_2$  have three contact points in common.

—, **undrawable** for region  $R$ : for a subgraph  $H$  drawn on a surface, an appendage  $B$  such that the boundary of region  $R$  does not contain all the contact points of  $B$ .

**barycentric subdivision, (first)** of a graph  $G$ : the subdivision of  $G$  such that every edge of  $G$  is subdivided exactly once.

—,  $n^{\text{th}}$ : the first barycentric subdivision of the  $(n - 1)^{\text{st}}$  barycentric subdivision.

**boundary of a region  $r$**  of an imbedding of a graph  $G$ : the subgraph of  $G$  that comprises all vertices that abut the region and all edges whose interiors abut the region.

**boundary walk of a face** of an imbedding of a graph  $G$ : a closed walk in graph  $G$  that corresponds to a complete traversal of the perimeter of the polygonal region within the face.

**cellular imbedding**  $\iota : G \rightarrow S$ : a graph imbedding such that every region is topologically equivalent to an open disk.

**chord of a subgraph  $H$ :** an appendage of  $H$  that consists of a single edge.

**connected Euclidean set:** see *Euclidean set*.

**contact point of an appendage  $B$  to a subgraph  $H$ :** a vertex of  $B \cap H$ .

**crossing number  $\nu(G)$**  of a simple graph  $G$ : the minimum number of edge-crossings that can occur in a normal (see §8.3) planar drawing of  $G$ .

**curve, closed or open** in a Euclidean set: see *path*.

**digon:** a 2-sided region.

**dilation of a semilinear mapping:** the maximum dilation of any edge in the domain graph.

**dual edges:** the edges of the dual graph.

**dual faces:** the faces of the dual imbedding.

**dual graph and dual imbedding:** the new graph and its imbedding derived, starting with a cellular imbedding of a graph, by inserting a new vertex into each existing face, and by then drawing through each existing edge a new edge that joins the new vertex in the region on one side to the new vertex in the region on the other side.

**dual vertices:** the vertices of the dual graph.

**edges separated in a graph  $G$  by the subgraph  $H$ :** two edges  $e_1$  and  $e_2$  of  $E_G - E_H$  such that every walk in  $G$  from an endpoint of  $e_1$  to an endpoint of  $e_2$  contains a vertex of  $H$ .

**edges unseparated in a graph  $G$  by the subgraph  $H$ :** two edges  $e_1$  and  $e_2$  of  $E_G - E_H$  that are not separated.

**Euclidean set  $X$ :** a subset of any Euclidean space  $\mathbb{R}^n$ .

—, **connected:** a Euclidean set  $X$  such that for every pair of points  $s, t \in X$ , there exists a path within  $X$  from  $s$  to  $t$ .

—, **separation of** by a subset  $W \subset X$ : the condition that there exists a pair of points  $s$  and  $t$  in  $X - W$ , such that every path in  $X$  from  $s$  to  $t$  intersects the set  $W$ .

**face** of a graph imbedding  $\iota : G \rightarrow S$ : the union of a region and its boundary. (A drawing does *not* have faces unless it is an imbedding.)

**face-set of a graph imbedding  $\iota : G \rightarrow S$ :** the set of all faces of the imbedding, denoted  $F_\iota$ ,  $F_G$ , or  $F$ , depending on the context.

**face-size:** the number of edge-steps in the closed walk around its boundary (in which some edges may count twice). A face of size  $n$  is said to be *n-sided*.

**girth** of a non-acyclic graph: the length of the smallest closed cycle (undefined for an acyclic graph).

**homeomorphic graphs:** a pair of graphs  $G$  and  $H$  such that there is an isomorphism from a *subdivision* of  $G$  to a *subdivision* of  $H$ .

**imbedding of a graph  $G$  on a surface  $S$ :** a drawing without any edge-crossings. See Chapter 8.

**Jordan separation property:** for a Euclidean set  $X$ , the property that every closed curve in  $X$  separates  $X$ .

**Kuratowski graphs:** the complete graph  $K_5$  and the complete bipartite graph  $K_{3,3}$ .

**Kuratowski subgraph of a graph:** a subgraph homeomorphic either to  $K_5$  or to  $K_{3,3}$ .

**longitude on the standard torus:** a closed curve that bounds a disk in the space exterior to the solid donut (and goes around in the “long” direction).

**meridian on the standard torus:** a closed curve that bounds a disk inside the solid donut it surrounds (and goes around the donut in the “short” direction).

**monogon:** a 1-sided region.

**path** from  $s$  to  $t$  in a Euclidean set: the image of a continuous function  $f$  from the unit interval  $[0, 1]$  to a subset of that space such that such that  $f(0) = s$  and  $f(1) = t$ , that is a bijection on the interior of  $[0, 1]$ . (One may visualize a path as the trace of a particle traveling through space for a fixed length of time.)

—, **closed:** a path such that  $f(0) = f(1)$ , i.e., in which  $s = t$ . (For instance, this would include a “knotted circle” in space.)

—, **open:** a path such that  $f(0) \neq f(1)$ , i.e., in which  $s \neq t$ .

**planar drawing** of a graph: a drawing of the graph in the plane without edge-crossings.

**planar graph:** a graph such that there exists a planar drawing of it.

**plane** in Euclidean 3-space  $\mathbb{R}^3$ : a set of points  $(x, y, z)$  such that there are numbers  $a$ ,  $b$ ,  $c$ , and  $d$  with  $ax + by + cz = d$ .

**Poincaré dual**: see dual graph imbedding.

**primal graph and primal imbedding**: whatever graph and imbedding are supplied as input to the Poincaré duality process.

**primal vertices, primal edges, and primal faces**: the vertices, edges, and faces of the primal graph and primal imbedding.

**proper walk**: a walk in which no edge occurs immediately after itself.

**region of a graph imbedding**  $\iota : G \rightarrow S$ : a component of the Euclidean set  $S - \iota(G)$ , which does *not* contain its boundary.

**Riemann stereographic projection**: a function  $\rho$  that maps each point  $w$  of the unit-diameter sphere (tangent at the origin  $(0, 0, 0)$  to the  $xz$ -plane in Euclidean 3-space) to the point  $\rho(x)$  where the ray from the north pole  $(0, 1, 0)$  through the point  $w$  intersects the  $xz$ -plane.

**self-dual graph**: a graph  $G$  with an imbedding in the sphere (or sometimes another surface) such that the Poincaré dual graph  $G^*$  for that imbedding is isomorphic to  $G$ .

**separated Euclidean set**: see *Euclidean set*.

**size of a face**: see *face-size*.

**smoothing out a 2-valent vertex**  $v$ : replacing two different edges that meet at  $v$  by a new edge that joins their other endpoints.

**sphere**: a set of points in  $\mathbb{R}^3$  equidistant from a fixed point.

**standard donut**: the surface of revolution obtained by revolving a disk of radius 1 centered at  $(2, 0)$  in the  $xy$ -plane around the  $y$ -axis in 3-space.

**standard torus**: the surface of revolution obtained by revolving a circle of radius 1 centered at  $(2, 0)$  in the  $xy$ -plane around the  $y$ -axis in 3-space. It is the surface of the standard donut.

**straight-line drawing** of a graph: a planar drawing in which every edge is represented by a straight-line segment.

**subdividing an edge  $e$  with endpoints  $\{u, v\}$** : replacing that edge  $e$  by a path  $u, e', w, e'', v$ , where edges  $e'$  and  $e''$  and vertex  $w$  are new to the graph.

**subdividing a graph**: performing a sequence of edge-subdivision operations.

**thickness**  $\theta(G)$  of a simple graph  $G$ : the smallest cardinality of a set of planar spanning subgraphs of  $G$  whose edge-sets partition  $E_G$ .

# Chapter 8

---

## DRAWING GRAPHS AND MAPS

- 8.1 The Topology of Low Dimensions**
  - 8.2 Higher-Order Surfaces**
  - 8.3 Mathematical Model for Drawing Graphs**
  - 8.4 Regular Maps on a Sphere**
  - 8.5 Imbeddings on Higher-Order Surfaces**
  - 8.6 Geometric Drawings of Graphs**
- 

### INTRODUCTION

Eight chapters deep into a book filled with graph drawings, we are ready to say with more precision what it means to *draw* a graph on a surface. For that purpose, it is necessary to have spatial models for surfaces and graphs, in which the edges of a graph have length, just as they do in drawings.

In practical applications, most graph drawings are on the plane or the sphere. Both these surfaces have the simplifying feature described in §7.1, called the *Jordan separation property*, that every closed curve separates it into two parts. As described there, applying the Jordan separation property provides an elementary method for solving the problem of deciding which graphs can be drawn on the plane or sphere.

However, some elementary methods of *algebraic topology* for solving this drawability problem, once mastered, yield correct results with far less effort, and they also apply to the more complicated surfaces. Chapter 7 presented the most basic principles for such endeavors, in particular, the Euler polyhedral equation, the face-size equation, the edge-face inequality, and Poincaré duality. These principles are applied in §8.4 to the problem of drawing highly symmetric maps on the sphere. They are subsequently generalized to the higher-order surfaces in §8.5, and then applied to the problem of drawing graphs on higher-order surfaces, where the Jordan curve theorem is of no avail. This chapter may be regarded as the starting point of topological graph theory.

## 8.1 THE TOPOLOGY OF LOW DIMENSIONS

The formal definition of a drawing as a continuous function depends on a spatial model for a graph (the domain) and a spatial model for a surface (the codomain), in which the edges of a graph have length, just as they do in drawings. Surfaces are defined intrinsically, starting here and continuing into §8.2, with the aid of *topological equivalence*, and they need not be attached to some solid that they bound.

### Some Subsets of Euclidean 2-Space and 3-Space

**DEFINITION:** **Euclidean  $n$ -space**  $\mathbb{R}^n$  is the set of  $n$ -tuples  $(x_1, \dots, x_n)$  with the usual Euclidean distance metric

$$\delta((x_1, \dots, x_n), (y_1, \dots, y_n)) = \sqrt{(x_1 - y_1)^2 + \dots + (x_n - y_n)^2}$$

**DEFINITION:** A **Euclidean set** is a subset of a Euclidean space.

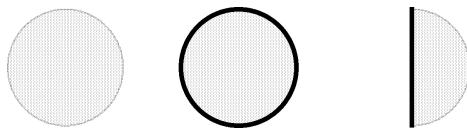
**DEFINITION:** The **plane** is another name for Euclidean 2-space  $\mathbb{R}^2$ . (This is consistent with the definition in §7.1 of a plane in  $\mathbb{R}^3$ .)

**DEFINITION:** The **open unit disk** is the plane Euclidean set containing the points  $\{(x_1, x_2) \mid x_1^2 + x_2^2 < 1\}$ , i.e., the points inside the unit circle.

**DEFINITION:** The **closed unit disk** is the plane Euclidean set containing the points  $\{(x_1, x_2) \mid x_1^2 + x_2^2 \leq 1\}$ , i.e., the points inside and on the unit circle.

**DEFINITION:** The **standard half-disk** is the plane Euclidean set containing the points  $\{(x_1, x_2) \mid x_1 \geq 0 \wedge x_1^2 + x_2^2 < 1\}$ .

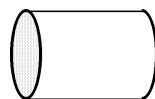
**Example 8.1.1:** As illustrated in Figure 8.1.1, the open unit disk and the the half-disk are subsets of the closed unit disk. The half-disk contains the segment of its frontier that is an open interval on the  $x_2$ -axis, but it does not contain the semi-circle that is its other frontier segment in  $\mathbb{R}^2$ .



**Figure 8.1.1** An open disk, a closed disk, and a standard half-disk.

**DEFINITION:** The **unit sphere** is the 3-dimensional Euclidean set containing the points  $\{(x_1, x_2, x_3) \mid x_1^2 + x_2^2 + x_3^2 = 1\}$ .

**DEFINITION:** The **unit cylinder** is the 3-dimensional Euclidean set containing the points  $\{(x_1, x_2, x_3) \mid 0 \leq x_1 \leq 1 \wedge x_2^2 + x_3^2 = 1\}$ .



**Figure 8.1.2** The unit cylinder.

### Topological Equivalences of Euclidean Sets

Intuitively, a function from one Euclidean set to another is continuous if it always maps “nearby” points of the domain to “nearby” points of the codomain. In theoretical calculus and point-set topology, this can be expressed precisely with epsilons and deltas. It is possible to gain an elementary grasp of graph placements on surfaces without a digression into the technicalities of continuous functions.

**DEFINITION:** A continuous function  $f : X \rightarrow Y$  between two Euclidean sets is called a **topological equivalence** if it is one-to-one and onto, and if the inverse function  $f^{-1} : Y \rightarrow X$  is continuous.

**Remark:** Intuitively, a topological equivalence is a function that deforms its domain into its codomain, without tearing the domain and without compressing any set of two or more points down to a single point.

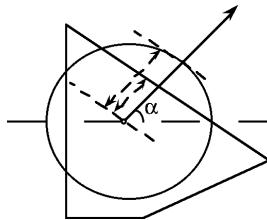
**Example 8.1.2:** The Riemann stereographic projection (see §7.1) is a topological equivalence between a sphere minus its north pole and the plane.

**Example 8.1.3:** A topological equivalence between the open unit disk and the entire plane, which illustrates how topological equivalence permits spaces to be stretched, is given by the function

$$f((x, y)) = \left( \frac{x}{1 - x^2 - y^2}, \frac{y}{1 - x^2 - y^2} \right)$$

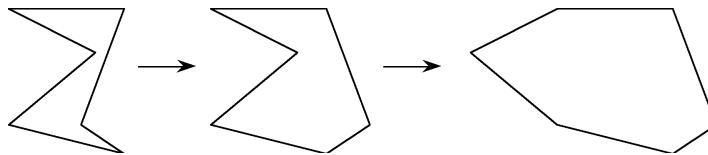
**Example 8.1.4:** The *interior* region  $R$  in the plane enclosed by any polygon but not including the polygon itself is topologically equivalent to the open unit disk. Writing out the continuous bijection with precise formulas is tedious. Instead, we describe the equivalence informally.

*Case 1: a convex polygon  $P$ .* Place the geometric center (or any other interior point) of the polygon at the origin. Next observe that the ray at angle  $\alpha$  from the origin to infinity intersects the open disk in a half-open segment  $I_D^\alpha$  and intersects region  $R$  in a half-open segment  $I_R^\alpha$ , as illustrated in Figure 8.1.3. Let the function  $f$  map  $I_D^\alpha$  to  $I_R^\alpha$  by linear stretching or compression. Then  $f$  is a topological equivalence from the unit open disk to the polygonal region  $R$ .



**Figure 8.1.3** Equivalence of a convex polygonal region to a unit disk.

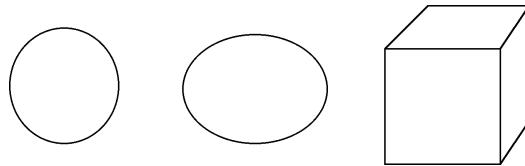
*Case 2: a non-convex polygon  $P$ .* First fill polygon  $P$  with “mathematical helium” that “inflates” it into a convex polygon with the same cycle of side-lengths, as illustrated in Figure 8.1.4. This inflation process is a topological equivalence. Then apply the method of the convex case.



**Figure 8.1.4** Mapping a non-convex region to a convex region.

DEFINITION: A **sphere** is any Euclidean set that is topologically equivalent to the unit sphere.

**Example 8.1.5:** The surface of any convex 3-dimensional solid is a sphere. This follows by a construction analogous to the convex-polygon example above. For instance, Figure 8.1.5 illustrates a ball, an ovoid, and a cube, each a 3-dimensional convex solid. Thus, the surface of each of these solids is a sphere.



**Figure 8.1.5** The surfaces of these convex solids are spheres.

### Topological Model of a Graph

To draw graphs on surfaces, except for simple graphs in the plane, we need a model of an edge.

DEFINITION: A **space curve** between two points  $x$  and  $y$  in 3-space is the image of a continuous function  $f : [0, 1] \rightarrow \mathbb{R}^3$  from the unit interval into 3-space such that  $f(0) = x$  and  $f(1) = y$ , which is one-to-one, except that possibly  $x = y$ . That is, the space curve is the set  $\{f(t) \mid t \in [0, 1]\}$ . The **interior of the space curve** is the subset  $\{f(t) \mid t \in (0, 1)\}$ .

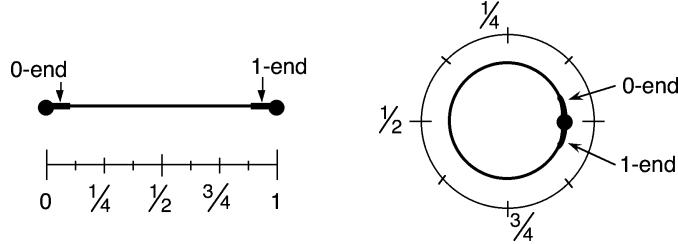
DEFINITION: In a **topological model** (or **carrier**) of a graph  $G = (V, E)$ , each vertex  $v \in V$  is represented by a point  $p_v$  in  $\mathbb{R}^3$  called a **space vertex**, and each proper edge  $e \in E$  is represented by a space curve  $q_e$  joining its endpoints, called a **space edge**. The interior of each space edge  $q_e$  of the carrier is disjoint from all the other spaces edges and also from all the space vertices. Except when confusion might result, the topological model of a graph  $G$  is also denoted  $G$ .

The topological model of a graph is a rich source of topological and geometric intuition. For instance, it immediately clarifies the distinction between the ends of an edge, which means intuitively the two minute parts of the edge “near” its endpoints, and the endpoints themselves, which are vertices.

DEFINITION: The **0-end of a space edge** of a graph is the part near the image of the 0-end of the unit interval  $[0, 1]$  in the space curve.

DEFINITION: The **1-end of a space edge** of a graph is the part near the image of the 1-end of the unit interval  $[0, 1]$  in the space curve.

**Remark:** In this topological sense, every edge has two distinct *ends*, even if it is a self-loop with only one *endpoint*. If a large segment  $\{f(t) \mid t \in (\epsilon, 1-\epsilon)\}$  of the interior of a space edge is discarded, what remains is the two whisker-like ends, as illustrated in Figure 8.1.6.



**Figure 8.1.6** The 0-end and the 1-end of a proper edge and of a self-loop.

**Remark:** The *direction* of a space edge may coincide with parametric increase or with parametric decrease. In the incidence table, a marker on the upper copy of the endpoint of a self-loop designates direction of parametric increase from the 0-end to the 1-end, and a marker on the lower copy of the endpoint designates direction of parametric decrease from the 1-end to the 0-end. Thus, even though a self-loop may have only one possible direction in a standard combinatorial model of a graph, it has two directions in a topological model.

### EXERCISES for Section 8.1

8.1.1<sup>s</sup> Specify a function from the open unit interval  $(0,1)$  to the entire real line that is a topological equivalence.

8.1.2 Specify a function from the annulus  $\{(x,y) \mid 1 \leq x^2 + y^2 \leq 2\}$  to the unit cylinder that is a topological equivalence.

8.1.3<sup>s</sup> How many topologically inequivalent Euclidean sets can be formed by identifying a single point of an open interval to a point in an open disk?

8.1.4 Four different Euclidean sets (i.e., topologically inequivalent) can be formed by identifying a single point of an open real interval to a point of a closed disk. Describe them.

## 8.2 HIGHER-ORDER SURFACES

Topology generalizes the concept of a surface, from the most elementary meaning, in which it surrounds some solid, to a more powerful meaning, in which it need not enclose anything.

**DEFINITION:** An  **$\epsilon$ -neighborhood** of a point in a Euclidean set, is the set of all points whose distance from that point is less than  $\epsilon$ , where  $\epsilon > 0$ .

**DEFINITION:** A **surface** is a Euclidean set in which every point has an  $\epsilon$ -neighborhood that is topologically equivalent either to the open unit disk or to the standard half-disk.

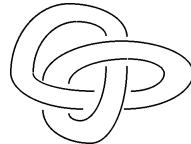
### Torus and Möbius Band

One possible kind of complication in a surface is apparent in the *torus*, and another kind is apparent in the *Möbius band*. Interestingly, these two kinds of complication are the only kinds of complication.

**REVIEW FROM §7.1:** The **standard torus** is the surface of revolution obtained by revolving a circle of radius 1 centered at  $(2,0)$  in the  $xy$ -plane disk around the  $y$ -axis in 3-space.

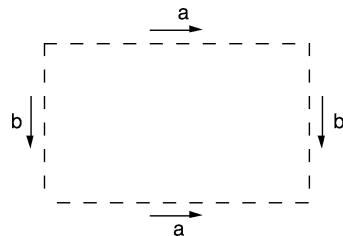
**DEFINITION:** A **torus** is any Euclidean set that is topologically equivalent to the standard torus.

**Example 8.2.1:** The surface of a knotted solid donut is a torus (as in Figure 8.2.1).



**Figure 8.2.1** A knotted donut in 3-space.

Akin to the way that the Riemann stereographic projection enables us to represent drawings on a sphere by drawings on a flat piece of paper, there is a way to represent toroidal drawings on a flat piece of paper. The rectangle in Figure 8.2.2 represents a torus.



**Figure 8.2.2** Representing a torus as a rectangle.

The top edge and the bottom edge of the rectangle in Figure 8.2.2 are both marked with the same letter, namely “a”. This indicates that these two edges are to be identified with each other, that is, pasted together. This pasting of edges creates a cylinder, exactly as would occur with a paper model, as illustrated in Figure 8.2.3 below.



**Figure 8.2.3** Identifying top and bottom edges yields a cylinder.

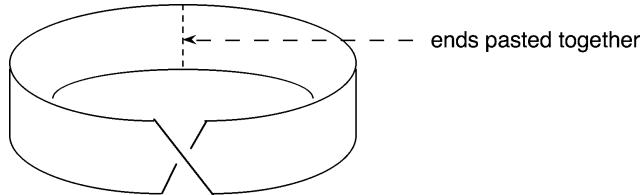
An effect of pasting the top edge to the bottom edge was to turn the left edge (marked “b”), and likewise the right edge (also marked “b”), into a closed curve. When the left end of the cylinder is pasted to the right end, the resulting surface is a torus. A paper cylinder would crumple, of course, but a plastic tube could be so converted into a torus.

**Remark:** Pasting the left edge to the right edge converts line “a” into a closed curve. Indeed, on the resulting torus, what was once edge “a” has become a longitude, and what was once edge “b” has become a meridian.

**DEFINITION:** A **Möbius band** is a space formed from a rectangular strip with a half-twist, as illustrated in Figure 8.2.4, by identifying the left edge with the right edge, to form a continuous band, as shown in Figure 8.2.5.

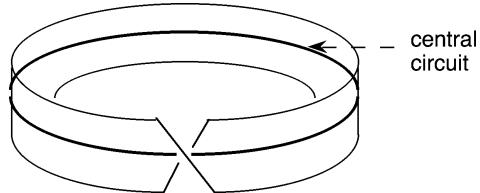


**Figure 8.2.4** A rectangular strip with a half-twist.



**Figure 8.2.5** A Möbius band.

**DEFINITION:** The **central circuit of a Möbius band** (Figure 8.2.6) is the result of matching one end of the line halfway between the top and bottom of the rectangular strip to the other end, as the right and left ends are identified.



**Figure 8.2.6** A Möbius band.

**Theorem 8.2.1.** *The Möbius band does not have the Jordan separation property.*

**Proof:** The central circuit does not separate a Möbius band into two parts.  $\diamond$

**Remark:** To illustrate that the Jordan curve theorem does not hold for the Möbius band, make a paper model of a Möbius band. Cut the Möbius band open along the central circuit. Notice that the result is a single connected surface, not two surfaces.

### Bounded and Boundaryless Surfaces

**DEFINITION:** An **interior point of a surface** is a point that has an  $\epsilon$ -neighborhood topologically equivalent to an open disk.

**DEFINITION:** A **boundary point of a surface** is a point that is not an interior point.

**DEFINITION:** A surface is **boundaryless** if every point is an interior point.

**Example 8.2.2:** The Euclidean plane is a boundaryless surface.

**Example 8.2.3:** A sphere is a boundaryless surface. Although a sphere may bound a solid ball, the sphere itself is boundaryless.

**Example 8.2.4:** The closed unit disk is not boundaryless. The points on the unit circle are its boundary points.

**Example 8.2.5:** The unit cylinder is not boundaryless. The points on the circles at either end are its boundary points.

**Example 8.2.6:** The Möbius band is not a boundaryless surface.

### Closed Surfaces

DEFINITION: A Euclidean set is **finite** if there is a real number  $M$  such that the maximum distance of any point from the origin is at most  $M$ .

DEFINITION: A connected surface is **closed** if it is a Euclidean set that satisfies these three conditions:

- (i). The surface  $S$  is finite.
- (ii). The surface  $S$  is boundaryless.
- (iii). The endpoints of every open arc in  $S$  are in surface  $S$  itself.

or if it is topologically equivalent to a Euclidean surface satisfying all three conditions.

**Example 8.2.7:** The sphere and the torus are closed surfaces.

**Example 8.2.8:** The open unit disk is not closed, because the endpoints of the open line segment  $\{(x_1, x_2) \mid x_1 = 0 \text{ and } -1 < x_2 < 1\}$  are on the unit circle, not in the open disk itself.

**Example 8.2.9:** The plane is not closed, because it is not finite.

**Example 8.2.10:** The Möbius band is not closed, because it has boundary points.

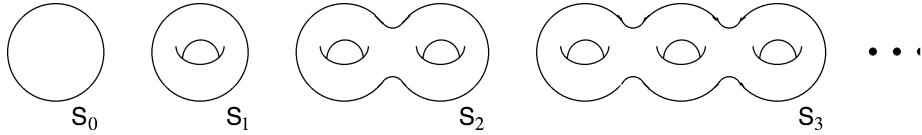
TERMINOLOGY NOTE: As readers familiar with topology may recognize, the present usage of “closed” in the geometric-topology sense of a closed surface differs from the usage of “closed” in point-set topology.

DEFINITION: A surface is **non-orientable** if it contains a subspace that is topologically equivalent to a Möbius band, and **orientable** otherwise.

**Remark:** Suppose that a small square is placed with its bottom edge on the central circuit of a Möbius band in 3-space. Suppose that the square is slid forward around the band exactly *once* (not twice). Then the top of the square will be positioned below the bottom. This reversability of top and bottom is the intuitive idea of “non-orientability”.

### Classification of Surfaces

DEFINITION: The **sequence of orientable surfaces**  $S_0, S_1, S_2, \dots$  is defined recursively.  $S_0$  is the sphere, and  $S_1$  is the torus. Surface  $S_{n+1}$  is obtained by attaching a handle to surface  $S_n$ . This sequence is illustrated in Figure 8.2.7 below.



**Figure 8.2.7** The sequence of closed orientable surfaces.

**Theorem 8.2.2.** *Classification of Closed Orientable Surfaces.* Every closed orientable surface is topologically equivalent to exactly one of the surfaces in the infinite sequence  $S_0, S_1, S_2, \dots$ .  $\diamond$  (proof omitted here; see [GrTu87])

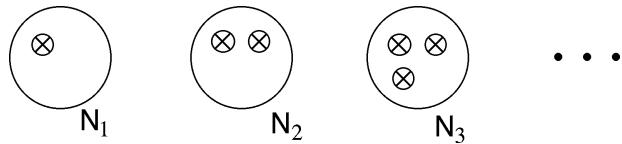
**DEFINITION:** The **genus of a closed orientable surface** is the subscript of the surface  $S_g$  to which it is topologically equivalent. That is, the genus is the “number of handles” on the surface.

**DEFINITION:** By **adding a crosscap to a surface**, we mean punching a hole in the surface and reclosing it by attaching a Möbius band. That is, the boundary of the Möbius band is identified with the boundary of the hole by a topological equivalence, which specifies exactly how to make the attachment.

**Remark:** The topological operation of replacing a disk on a sphere or other surface by a Möbius band can be readily performed in 4-space, but not in 3-space. Fortunately, there is a way to conceptualize the operation without thinking 4-dimensionally. The result of the operation is simply that whenever we cross through from the “main part” of the surface through the boundary of a removed disk, we cross into the Möbius band that replaced the disk.

**Remark:** Trying to draw a surface from 4-space in the plane would be a drop of two dimensions. (In other words, it would be a loss of detail comparable to representing a 3-dimensional figure by a subset of a single line.) The circle with the  $\times$  is the graphic device by which the result of the crosscap operation is represented.

**DEFINITION:** The **sequence of non-orientable surfaces**  $N_1, N_2, N_3, \dots$  is defined recursively, starting from the sphere, which in this context is denoted  $N_0$ , even though it is orientable. Surface  $N_{n+1}$  is obtained by adding a crosscap to surface  $N_n$ . Surface  $N_1$  is called the **projective plane** and surface  $N_2$  the **Klein bottle**.



**Figure 8.2.8** The sequence of closed non-orientable surfaces.

**Theorem 8.2.3.** *Classification of Closed Non-Orientable Surfaces.* Every closed non-orientable surface is topologically equivalent to exactly one of the surfaces in the infinite sequence  $N_1, N_2, N_3, \dots$ .  $\diamond$  (proof omitted here; see [GrTu87])

**DEFINITION:** The **crosscap number** (or **non-orientable genus**) of a **closed non-orientable surface** is the subscript of the surface  $N_k$  to which it is topologically equivalent. That is, it is the number of (disjoint) Möbius bands on the surface.

### EXERCISES for Section 8.2

*In each of the Exercises 8.2.1 through 8.2.5, give a reason why the Euclidean set described is not a closed surface.*

- 8.2.1<sup>s</sup> The result of deleting the origin  $(0,0)$  from the open unit disk.
- 8.2.2 The result of deleting the origin  $(0,0)$  from the closed unit disk.
- 8.2.3 The result of deleting the point  $(1,0)$  from the closed unit disk.
- 8.2.4 The result of deleting a single point from a torus in 3-space.
- 8.2.5 The result of deleting a meridian from the standard torus in 3-space.

*In each of the Exercises 8.2.6 through 8.2.10, determine whether the Euclidean set described is a boundaryless surface.*

- 8.2.6<sup>s</sup> The result of deleting the origin  $(0,0)$  from the open unit disk.
- 8.2.7 The result of deleting the origin  $(0,0)$  from the closed unit disk.
- 8.2.8 The result of deleting the point  $(1,0)$  from the closed unit disk.
- 8.2.9 The result of deleting a single point from a torus in 3-space.
- 8.2.10 The result of deleting a meridian from the standard torus in 3-space.

8.2.11 Cut a paper model of a Möbius band open along the central circuit, and see that the Jordan curve theorem does not hold for the Möbius band, thereby confirming Theorem 8.2.1.

8.2.12<sup>s</sup> Draw a closed circuit on a flat polygon representation of the torus that traverses the meridian direction once and the longitudinal direction once. Does this circuit separate the torus?

8.2.13 Draw a closed circuit on the torus that traverses the torus once in the meridian direction and once in the longitudinal direction.

8.2.14 Draw a closed circuit on the torus that traverses the torus three times in the meridian direction and twice in the longitudinal direction.

## 8.3 MATHEMATICAL MODEL FOR DRAWING GRAPHS

**REVIEW FROM ELEMENTARY SET THEORY:** The **image** of a point or subset of the domain of a function is the point or subset in the codomain onto which it is mapped.

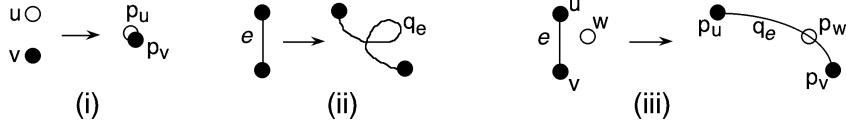
**REVIEW FROM §7.1:** An **open path from  $s$  to  $t$**  in a Euclidean set  $X$  is the image of a continuous bijection  $f$  from the unit interval  $[0, 1]$  to a subset of  $X$  such that  $f(0) = s$  and  $f(1) = t$ . (One may visualize a path as the trace of a particle traveling through space for a fixed length of time.)

**TERMINOLOGY NOTE:** A **singularity** of a continuous function is a point of the image where the function is not one-to-one.

Designing a mathematical model of a drawing of a graph on a surface starts by representing each vertex as a point on the surface and representing each edge as an open path in the surface between the images of its endpoints.

**DEFINITION:** A **drawing of a graph**  $G$  on a surface  $S$  is the union of a set of points of  $S$ , one for each vertex of  $G$ , and a set of open paths, one for each edge of  $G$ , which joins the images of the endpoints of that edge. Three types of **forbidden singularities** that occasionally occur in representations of graphs on surfaces are illustrated in Figure 8.3.1, but are otherwise absent from this book.

- (i) Two vertex images  $p_u$  and  $p_v$  occur at the same point of  $S$ .
- (ii) An edge image  $q_e$  has a self-intersection or other singularities.
- (iii) The interior of an edge image intersects a vertex image.



**Figure 8.3.1** Three forbidden singularity types for graph drawings.

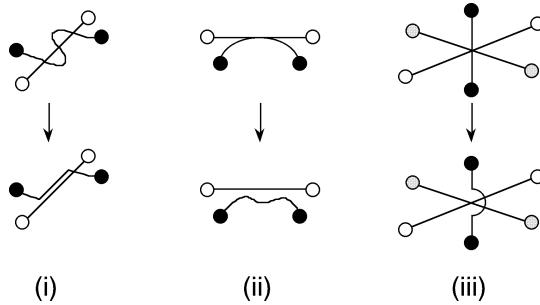
**DEFINITION:** An **edge-crossing** in a graph drawing is a singularity such that the images of two different edges meet.

### Normalized Drawings and Imbeddings

**DEFINITION:** An **abnormality in a drawing of a graph** on a surface is any of these three types of singularities:

- (i) The images of two edges meet at more than one point of the surface.
- (ii) The images of two different edges meet, but fail to cross each other (i.e., a tangency).
- (iii) The images of three or more edges meet at the same point of the surface.

Figure 8.3.2 illustrates the remedies for the three kinds of abnormalities: (i) an edge that crosses another several times can stay on the same side until it gets to the final crossing; (ii) if one edge just touches another, then it can be pulled away from the other edge; (iii) if an edge crosses a place where two other edges already cross, then it can be rerouted to go around that pre-existing crossing.



**Figure 8.3.2** Remedies for the three types of abnormalities in drawings.

**DEFINITION:** A **normal drawing of a graph** on a surface is a drawing that is free of all three types of abnormalities.

Unless explicitly specified otherwise in context, it is assumed throughout this book that a drawing is normalized. (We occasionally relax rule (iii).)

**DEFINITION:** An **imbedding of a graph**  $G$  on a surface  $S$  is a drawing with no edge-crossings at all; that is, it is the image of a topological equivalence  $\iota : G \rightarrow S$  from the topological model of  $G$  to a subset of  $S$ .

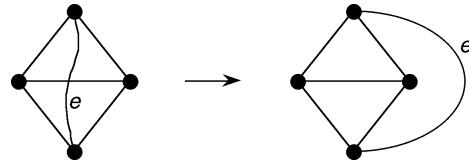
**TERMINOLOGY NOTE:** More generally, an *imbedding* is a one-to-one function from one topological space to another, for instance, between arbitrary Euclidean sets. In this sense, the *standard torus* is imbedded in  $\mathbb{R}^3$ .

**TERMINOLOGY NOTE:** One usually speaks of the image of a function  $f : X \rightarrow Y$  as lying *in* the codomain  $Y$ . However, in view of the notion that one draws *on* a surface, it is natural when the codomain  $Y$  is a surface to speak of the image as lying *on* the surface. This is not a rigid rule, and indeed, some inconsistency of usage from paragraph to paragraph is not unusual.

### Eliminating Edge-Crossings

Frequently, it is desirable to keep the number of edge-crossings to a minimum, which may involve some redrawing.

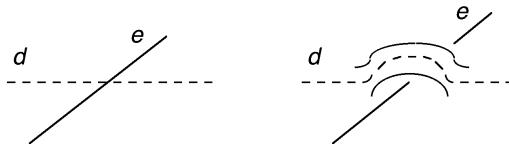
**Example 8.3.1:** Sometimes a drawing of a graph can be changed into an imbedding by rerouting the image of one or more edges to different positions. For instance, in Figure 8.3.3, a drawing of the complete graph  $K_4$  with one edge-crossing is changed into an imbedding by moving the image of edge  $e$ .



**Figure 8.3.3** Eliminating an edge-crossing by moving the image of an edge.

**DEFINITION:** By **attaching a handle to a surface**, we mean punching two separate holes in the surface and connecting one to the other with a tube.

**Example 8.3.2:** Adding a handle to a surface is another way to eliminate an edge-crossing from a drawing of a given graph, as illustrated in Figure 8.3.4. The handle is added from one side of edge  $e$  to the other, and then the image of edge  $d$  is rerouted so that it lies on the new handle, instead of crossing edge  $e$ . Of course, the handle-adding operation changes the surface on which the graph is drawn.



**Figure 8.3.4** Eliminating an edge-crossing by adding a handle.

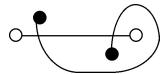
**Remark:** Every graph can be imbedded in 3-space. First draw the graph in the plane, and then eliminate the crossings by adding handles, as in Example 8.3.2. Different handles can be at different heights off the plane, so they do not intersect.

**Application 8.3.1 Printed Circuit Boards:** A planar electronic circuit can be printed onto a board. Since the wires of a planar circuit need not cross, no insulation is needed. When a circuit is non-planar, one practical approach, described in §7.7, is to partition its edges into a few layers.

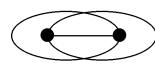
### EXERCISES for Section 8.3

In Exercises 8.3.1 through 8.3.6, show how to remove the abnormality from the drawing.

8.3.1<sup>s</sup>



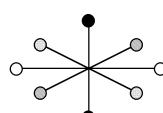
8.3.2



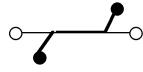
8.3.3



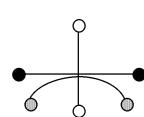
8.3.4



8.3.5



8.3.6



## 8.4 REGULAR MAPS ON A SPHERE

The difference between *imbedding* and *map* is largely nuance. The notation  $G \rightarrow S$  is the same. When one discusses the imbeddings of a graph  $G$ , the graph is fixed and the surface  $S$  may vary. When one discusses the maps on a surface  $S$ , the graph  $G$  may vary and the surface is fixed. Whereas the *imbedding theory* of a given graph studies all the various ways that graph can be imbedded in its range of possible closed surfaces, the *map theory* of a given surface, by way of contrast, studies all the various graphs that can be imbedded on that surface and all the ways they can be imbedded.

### Map Theory

**DEFINITION:** A **map on a surface** is an imbedding of a graph on that surface.

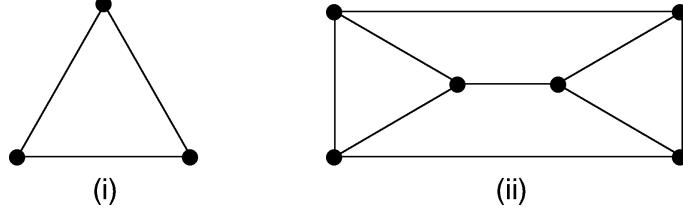
**TERMINOLOGY NOTE:** Whereas the term *mapping* is a generic synonym for function, the term *map* refers to a function from a graph to a surface.

**REVIEW FROM §7.5:** The (*Poincaré*) **dual graph** and the **dual imbedding** are derived, starting with a cellular imbedding of a graph, by inserting a new vertex into each existing face, and by then drawing through each existing edge a new edge that joins the new vertex in the region on one side to the new vertex in the region on the other side.

**DEFINITION:** A **simple map** on a surface is an imbedding of a simple graph on that surface, such that the dual graph is also a simple graph.

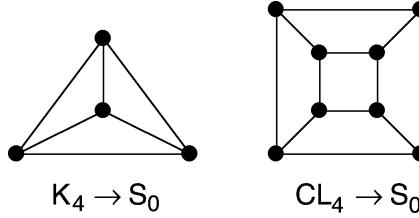
**DEFINITION:** A **regular map** on a surface is an imbedding of a regular graph such that the dual graph is also a regular graph. <sup>†</sup>

**Example 8.4.1:** The graphs for the *sphere maps* (i.e., maps on a sphere) in Figure 8.4.1 are both simple and regular. However, the dual graph of map (i) is the dipole  $D_3$ , which is regular but not simple. Furthermore, the dual graph of map (ii) is the graph  $K_5 - e$ , which is simple but not regular. These assertions are easily verified by drawing the dual graphs.



**Figure 8.4.1** Two sphere maps: (i) one non-simple, (ii) the other non-regular.

**Example 8.4.2:** The maps in Figure 8.4.2 are both regular and simple. The first map is of the 3-regular simple graph  $K_4$  on the sphere  $S_0$ , with all four faces 3-sided; the dual graph is  $K_4$ . The second map is of the 3-regular simple graph  $CL_4$  on the sphere  $S_0$ , with all six faces 4-sided; the dual graph is the octahedron graph  $O_3$  (see Exercises). These assertions are easily verified by drawing the dual graphs.



**Figure 8.4.2** Two regular simple maps on the sphere.

**Remark:** These definitions of regular and simple also apply to maps on surfaces other than the sphere.

**Proposition 8.4.1.** A map  $\iota : G \rightarrow S$  is regular if and only if graph  $G$  is regular and every face has the same number of sides.

**Proof:** ( $\Rightarrow$ ) Assume that the map  $\iota : G \rightarrow S$  is regular. Then graph  $G$  is regular. Moreover, since the dual graph  $G^*$  is regular, each dual vertex has the same degree. Since the degree of a dual vertex equals the number of sides of a primal face, each primal face has the same number of sides.

( $\Leftarrow$ ) Assume that graph  $G$  is regular and that every face has the same number of sides. The latter condition implies that the dual graph  $G^*$  is regular. Thus, the map  $\iota : G \rightarrow S$  is regular.  $\diamond$

---

<sup>†</sup> The algebraic map-regularity criterion prescribed by Coxeter and Moser [CoMo72] and others is equivalent for maps on the sphere and less inclusive for maps on other surfaces.

**DEFINITION:** A **self-dual map** on a surface is a map  $\iota : G \rightarrow S$  such that there is a topological equivalence of the surface  $S$  to itself that takes graph  $G$  onto the dual graph  $G^*$ .

**Example 8.4.3:** The map  $K_4 \rightarrow S_0$  in Figure 8.4.2 is self-dual, but the map  $CL_4 \rightarrow S_0$  is not.

### Degrees and Face-Sizes of Regular Maps

The following straightforward application of the numerical relations in the previous section yields an upper bound on the average degree  $\delta_{\text{avg}}(G)$  of a graph  $G$  that can be imbedded on the sphere.

**Theorem 8.4.2.** Let  $\iota : G \rightarrow S_0$  be an imbedding of a simple graph on the sphere. Then  $\delta_{\text{avg}}(G) < 6$ .

**Proof:**

- (1)  $\text{girth}(G) \cdot |F| \leq 2|E|$  edge-face ineq.
- (2)  $3 \leq \text{girth}(G)$  since  $G$  is simple
- (3)  $3|F| \leq 2|E|$  combine (1) and (2)
- (4)  $|F| \leq \frac{2|E|}{3}$
- (5)  $|V| - |E| + |F| = 2$  Euler poly. eq.
- (6)  $|V| - \frac{|E|}{3} \geq 2$  subst. (4) into (5)
- (7)  $6|V| - 2|E| \geq 12$
- (8)  $2|E| \leq 6|V| - 12$
- (9)  $\frac{2|E|}{|V|} \leq 6 - \frac{12}{|V|}$
- (10)  $\delta_{\text{avg}}(G) = \frac{\sum_{v \in V} \deg(v)}{|V|}$  by def. of avg.
- (11)  $\delta_{\text{avg}}(G) = \frac{2|E|}{|V|}$  by Theorem 1.1.2
- (12)  $\delta_{\text{avg}}(G) \leq 6 - \frac{12}{|V|}$  combine (9) and (11)
- (13)  $\delta_{\text{avg}}(G) < 6$  from (12), since  $\frac{12}{|V|} > 0$   $\diamond$

**Corollary 8.4.3.** The only possible degrees of a graph with a regular simple map on the sphere are 3, 4, and 5.

**Proof:** Theorem 8.4.2 precludes the possibility of any degree greater than 5.  $\diamond$

**Corollary 8.4.4.** The only possible face-sizes of a regular simple map on the sphere are 3, 4, and 5.

**Proof:** Apply Corollary 8.4.3 to the dual map.  $\diamond$

### Construction of the Regular Simple Maps

In view of Corollaries 8.4.3 and 8.4.4, a regular simple map on the sphere must have constant degree 3, 4, or 5 and constant face-size 3, 4, or 5. That seems to permit nine combinations. Theorem 8.4.5 provides further information, based on the numerical relations.

**Theorem 8.4.5.** *If a regular map on the sphere has  $d$  and  $r$  for its constant degree and constant face-size, respectively, then*

$$|V| = \frac{4r}{2(d+r)-dr} \quad |E| = \frac{2dr}{2(d+r)-dr} \quad |F| = \frac{4d}{2(d+r)-dr}$$

**Proof:** Solving the following three linear equations for the “unknowns”  $|V|$ ,  $|E|$ , and  $|F|$  in terms of  $d$  and  $r$  yields the result.

$$|V| - |E| + |F| = 2 \quad \text{Euler polyhedral equation}$$

$$2|E| = r|F| \quad \text{face-size equation}$$

$$2|E| = d|V| \quad \text{degree-sum equation}$$

◊

The following table gives the numbers of vertices, edges, and faces that would correspond to each of the nine possible combinations of degree  $d$  and face-size  $r$  in a regular simple map on the sphere. The common denominator in the conclusion of Theorem 8.4.5 is denoted  $Y$ . In the rightmost column, the table names the polyhedron whose 1-skeleton and surface realize the map if it exists.

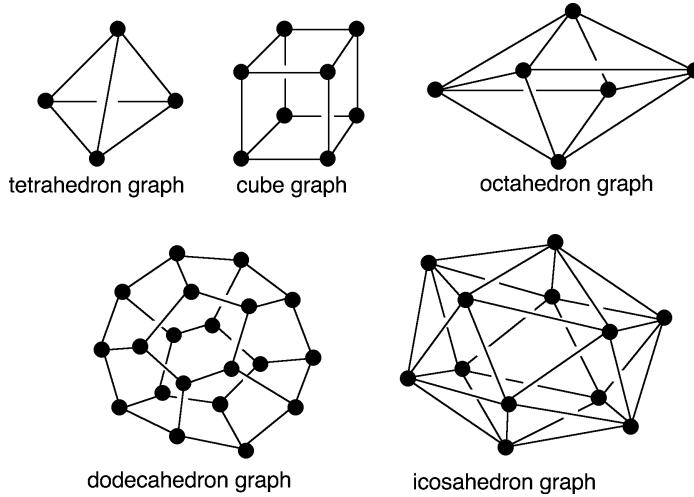
$d$	$r$	$Y$	$\frac{4r}{Y}$	$\frac{2dr}{Y}$	$\frac{4d}{Y}$	Name of polyhedron
3	3	3	4	6	4	Tetrahedron
3	4	2	8	12	6	Cube
3	5	1	20	30	12	Dodecahedron
4	3	2	6	12	8	Octahedron
4	4	0	undef.	undef.	undef.	no solution
4	5	-2	-10	-20	-8	no solution
5	3	1	12	30	20	Icosahedron
5	4	-2	-8	-20	-10	no solution
5	5	-5	-4	-10	-4	no solution

**DEFINITION:** A **platonic solid** is a *geometrically regular* 3-dimensional polyhedron. Geometrically regular means here that its 1-skeleton is a regular graph and also that each of the faces is geometrically a regular polygon.

**DEFINITION:** A **platonic graph** is the 1-skeleton of a platonic solid.

**DEFINITION:** A **platonic map** is the imbedding of the 1-skeleton of a platonic solid into its surface.

**Example 8.4.4:** The five platonic solids, graphs, and maps are illustrated in Figure 8.4.3.



**Figure 8.4.3** The five platonic graphs.

**Remark:** The five maps in Figure 8.4.3 are the only regular simple maps on the sphere. The illustration proves their existence.

**Remark:** Since the three numerical equations used in the proof of Theorem 8.4.5 are true for non-simple graphs, the resulting formulas can be used to determine possibilities for non-simple regular maps on the sphere.

### EXERCISES for Section 8.4

For Exercises 8.4.1 through 8.4.4, draw a regular map of a possibly non-simple graph on the sphere that conforms to the given description.

- 8.4.1 Regular of degree = 1.
- 8.4.2<sup>s</sup> Regular of degree = 2 with face-size = 4.
- 8.4.3 Regular of degree = 4 with face-size = 2.
- 8.4.4 Regular of degree = 2 with face-size = 3.
- 8.4.5<sup>s</sup> Find all non-simple regular maps on the sphere with graph degree  $\geq 3$ .
- 8.4.6 Draw the dual of the map  $CL_4 \rightarrow S_0$  (see Figure 8.4.2), and prove that the dual graph is isomorphic to the octahedron graph  $\mathcal{O}_3$ .
- 8.4.7 Prove that for graph degree = 2, there is a regular map on the sphere with any positive integer specified as the face-size.
- 8.4.8 Prove that for face-size = 2, there is a regular map on the sphere with any positive integer specified as the degree.

## 8.5 IMBEDDINGS ON HIGHER-ORDER SURFACES

To draw a graph on a higher surface, it sometimes helps to cut the surface open along a few strategic closed curves, so that the surface can be flattened out. Some algebraic relations previously derived, including the Euler polyhedral equation, can be generalized to drawings on higher surfaces.

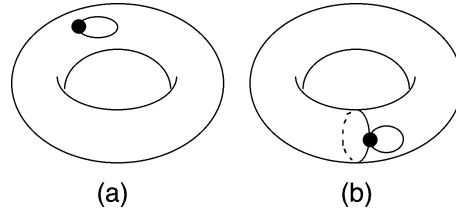
### Cellular Imbeddings

When a graph is drawn on a closed surface other than the sphere, perhaps there are handles or crosscaps in the interiors of one or more regions, but usually not.

**DEFINITION:** A region of a graph imbedding  $\iota : G \rightarrow S$  on a surface  $S$  is a **cellular region** if it is topologically equivalent to an open disk. (Topologists sometimes call a disk a “2-cell”.) It is a **strongly cellular region** if, moreover, the boundary walk is a cycle in the graph.

**DEFINITION:** A graph imbedding  $\iota : G \rightarrow S$  on a surface  $S$  is a **cellular imbedding** if every region is cellular. It is a **strongly cellular imbedding** if, moreover, every region is strongly cellular.

**Example 8.5.1:** In Figure 8.5.1, the imbeddings of the bouquets  $B_1$  and  $B_2$  are both non-cellular. In Figure 8.5.1(a), the “big” region is not even planar (because it does not have the Jordan separation property). In Figure 8.5.1(b), both regions are planar, but a meridian cycle does not bound a disk within the big region, which is topologically equivalent to the unit cylinder minus its two boundary components.



**Figure 8.5.1** Two non-cellular imbeddings on the torus.

Any non-cellular imbedding of a graph on a closed surface can be obtained by adding handles and crosscaps to a cellular imbedding. Largely for this reason, we concentrate on cellular imbeddings of graphs. For instance, the non-cellular imbedding in Figure 8.5.1(a) could be obtained by adding a handle to one region of a cellular imbedding of the bouquet  $B_1$  in the sphere. Moreover, the non-cellular imbedding in Figure 8.5.1(b) could be obtained from a cellular imbedding of the bouquet  $B_2$  on the sphere by adding a handle joining the digon to one of the monogons.

**Proposition 8.5.1.** *The boundary of a face  $f$  of a connected graph  $G$  imbedded on the sphere is connected.*

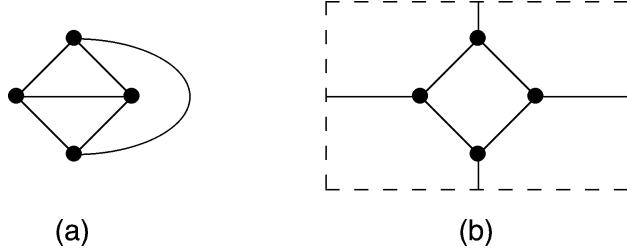
**Proof:** A closed loop drawn just inside one of the boundary components of face  $f$  would separate the sphere, by the Jordan curve theorem. Since graph  $G$  is connected, all of it lies on the same side of that closed loop as that boundary component. Thus, there is no other boundary component.  $\diamond$

**Proposition 8.5.2.** *Every imbedding of a connected graph on the sphere is a cellular imbedding.*

**Proof:** This follows from Proposition 8.5.1 and the Schoenflies theorem, that every closed curve on a sphere bounds a disk.  $\diamond$

**Example 8.5.2:** In the planar imbedding of  $K_4$  in Figure 8.5.2(a), there are four regions, all strongly cellular. In the toroidal imbedding of  $K_4$  in Figure 8.5.2(b), there are two regions. Centered in drawing (b) is a 4-sided strongly cellular region. The other region is cellular, but it “meets itself” along the lines used to paste the rectangle into a torus, so its boundary is not a cycle. Accordingly, that region is not strongly cellular.

Observe in drawing (b) that the partial edge incident on the topmost vertex meets the partial edge incident on the bottommost vertex. These two parts of the same edge are joined when the top broken edge of the rectangle is identified with the bottom broken edge. Similarly, the partial edge incident on the leftmost vertex is part of the same edge as the partial edge incident on the rightmost vertex.

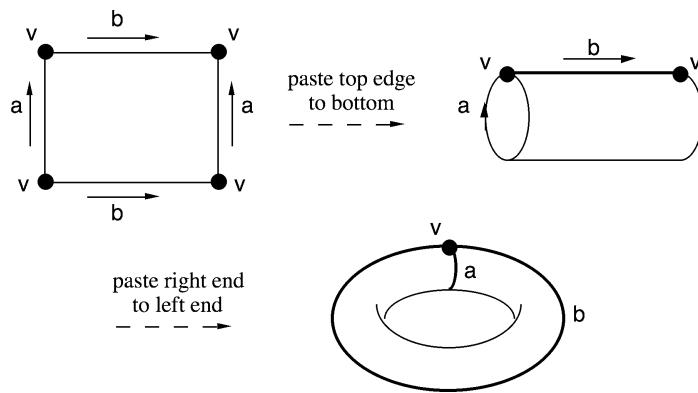


**Figure 8.5.2** Two imbeddings of  $K_4$ : (a) on the plane; (b) on the torus.

### Flat Polygon Drawings

**DEFINITION:** A **flat polygon representation** of a surface  $S$  is a drawing of a polygon with markings to match its sides in pairs, such that when the sides are pasted together as the markings indicate, the resulting surface obtained is topologically equivalent to  $S$ .

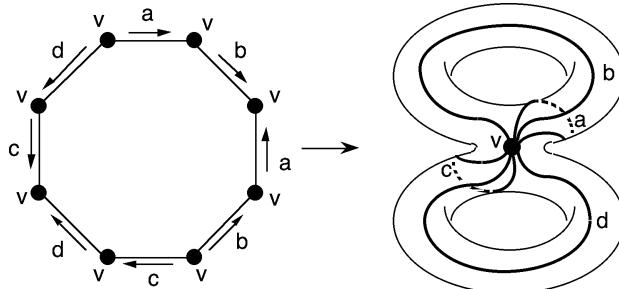
**Example 8.5.3:** Figure 8.5.3 recalls how the torus was represented in §8.2 as a rectangle with its sides identified in pairs. Another perspective on this representation is that cutting the torus open on its meridian and longitude permits it to be flattened into a rectangle.



**Figure 8.5.3** Representing a torus as a rectangle.

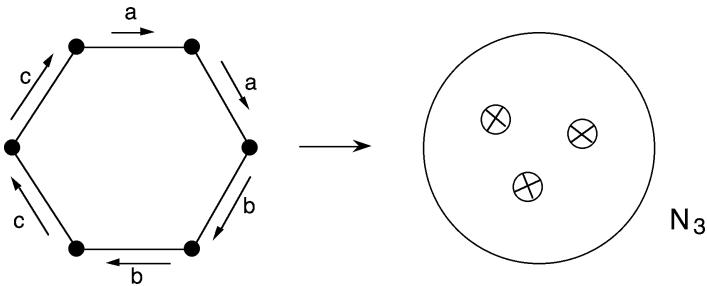
Every orientable surface  $S_g$  can be represented as a flat polygon with  $4g$  sides (for a detailed proof, see [Ma67]). As one traverses the boundary of the polygon, each handle is represented by a sequence of four consecutive sides, which are marked with the pasting pattern  $sts^{-1}t^{-1}$ .

**Example 8.5.4:** Figure 8.5.4 shows a representation of the double torus  $S_2$  as an octagon, from which one handle is formed by the four sides marked  $aba^{-1}b^{-1}$  and the other by the four sides marked  $cdc^{-1}d^{-1}$ .



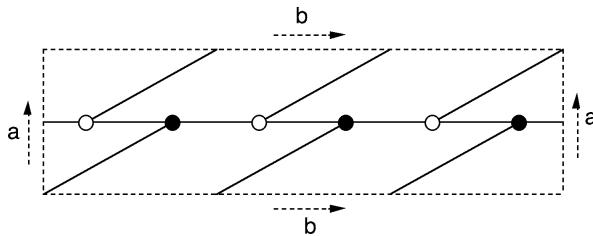
**Figure 8.5.4** Representing a double torus as an octagon.

Similarly, every non-orientable surface  $N_k$  can be represented as a flat polygon with  $2k$  sides (for further details, see [Ma67]). As one traverses the boundary of the polygon, each crosscap is represented by a sequence of two sides marked with the pasting pattern  $ss$ . Figure 8.5.5 shows a representation of the surface  $N_3$  as a hexagon.



**Figure 8.5.5** Representing the surface  $N_3$  as a hexagon.

In a drawing of a graph on the flat polygon representation of a surface, an edge may cross through one copy of a matched side to the other copy. For instance, Figure 8.5.6 shows an imbedding of  $K_{3,3}$  on the torus with three hexagonal regions. One edge crosses through the meridian cut  $a$ , two edges cross through the longitudinal cut  $b$ , and one edge cuts through the point where the meridian and longitude intersect.

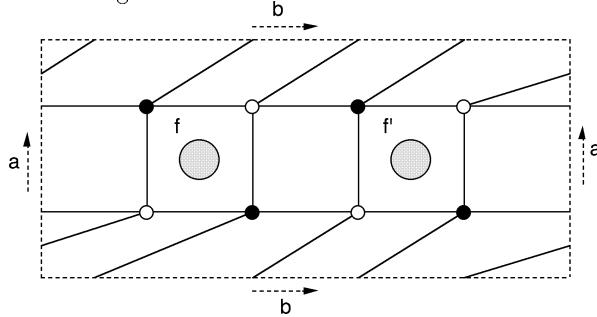


**Figure 8.5.6** Toroidal imbedding of  $K_{3,3}$ .

### Surgery on Imbeddings

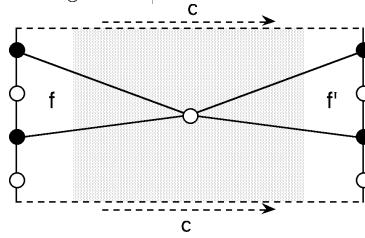
Trying to draw graph imbeddings on flat polygon representations of surfaces more complicated than a torus (the surface  $S_1$ ) or a Klein bottle (the surface  $N_2$ ) is often quite frustrating. It is frequently easier to draw most of the graph on a simpler surface and to complete the imbedding by attaching extra handles and drawing edges across the extra handles. Informally, such modifications are called **surgery**.

**Example 8.5.5:** Drawing  $K_{4,5}$  on a flat polygon representation of  $S_2$  is no easy task. (Try it and see!) A surgical approach to this imbedding problem is to start by drawing  $K_{4,4}$  on  $S_1$ , as shown in Figure 8.5.7.



**Figure 8.5.7** Toroidal imbedding of  $K_{4,4}$ , with holes punched in two regions.

Two regions  $f$  and  $f'$  of this imbedding are selected so that the union of their face-boundary walks contains all the black vertices. Next, a hole is punched in regions  $f$  and  $f'$ , shown by shaded disks in Figure 8.5.7. Then a tube is attached from one hole to the other, thereby reclosing the surface, so that a single new *non-cellular* region is formed from the two regions with the holes plus the tube. The resulting surface is  $S_2$ , and the new region has two boundary components, at opposite ends of the tube. Finally, a fifth white vertex is drawn, as shown in Figure 8.5.8, on the tube (depicted as shaded) and joined to both black vertices on the boundary components. The final result of this surgery is a cellular imbedding of  $K_{4,5}$  on  $S_2$ .



**Figure 8.5.8** Joining the fifth white vertex.

### Euler Polyhedral Equations for All Closed Surfaces

We observe that the **Face-Size Equation**

$$2|E_G| = \sum_{f \in F} \text{size}(f)$$

holds for every graph imbedding  $\iota : G \rightarrow S$ , cellular or not, in every surface. Similarly, the **Edge-Face Inequality**

$$2|E| \geq \text{girth}(G) \cdot |F|$$

holds whenever the graph is connected but not a tree (for which girth is undefined), in every surface, even if the imbedding is non-cellular. Moreover, for any cellular graph imbedding  $\iota : G \rightarrow S$ , the following Poincaré duality relations hold.

- (i)  $|V^*| = |F|$
- (ii)  $|E^*| = |E|$
- (iii)  $|F^*| = |V|$

However, the Euler polyhedral equation is restricted to cellular imbeddings. As the following theorems indicate, the value of this formula is unique on each orientable surface, and it is unique on each non-orientable surface.

**Lemma 8.5.3.** *Let  $G \rightarrow S$  be a cellular imbedding of a graph in a surface. Then the result of subdividing an edge of the imbedded copy of  $G$ , or of joining two vertices on the boundary of a face by a new edge is an imbedding whose Euler formula has the same value as for the imbedding  $G \rightarrow S$ .*

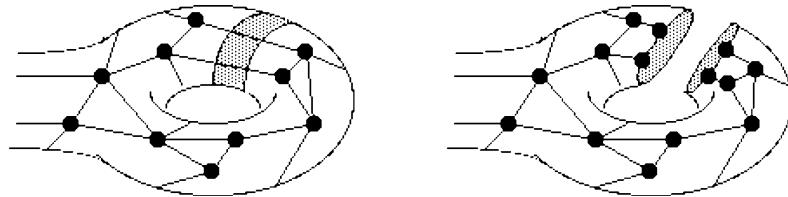
**Proof:** Subdividing an edge adds one new vertex and increases the number of edges by one. Drawing a new edge across a cellular face  $f$  separates  $f$  into two faces. In either case the changes have offsetting effects on the value of the Euler formula.  $\diamond$

**Theorem 8.5.4.** *Let a graph  $G$  be cellularly imbedded in the orientable surface  $S_g$ . Then  $|V| - |E| + |F| = 2 - 2g$ .*

**Proof:** For imbeddings on the sphere  $S_0$ , we proved in §7.5 that the value of the Euler formula  $|V| - |E| + |F|$  must be equal to 2. This serves as the base case for an induction. As an inductive hypothesis, assume that the equation is correct for every cellular imbedding in the surface  $S_g$ . We consider a cellular imbedding of a graph  $G$  in the surface  $S_{g+1}$ .

Draw a meridian on the rightmost handle of  $S_{g+1}$  so that it meets graph  $G$  in finitely many points, each in the interior of some edge of  $G$ . If necessary, we subdivide edges of  $G$  so that no edge crosses the meridian more than once. By Lemma 8.5.3, the value of the Euler formula in the resulting graph imbedding is unchanged from the original.

Next we thicken the meridian to an annulus  $A$ , as shown at the left of Figure 8.5.9, and we subdivide each edge of  $G$  that crosses the annulus at both of its intersection points with a boundary component of the annulus, which, by Lemma 8.5.3, also preserves the value of the Euler formula. Then we augment the edge-set of the graph by adding to it each segment of annulus boundary that joins two vertices on it. By Lemma 8.5.3, the resulting graph imbedding has the same value of the formula  $|V| - |E| + |F|$  as the original imbedding.



**Figure 8.5.9** Thickening a meridian and subsequent surgery.

In the interior of annulus  $A$ , edges and faces alternate, so there are equally many. Thus, excising the interior of the annulus preserves the value of the formula  $|V| - |E| + |F|$ . Finally, by capping the two resulting holes in the surface with disks, as shown at the right of Figure 8.5.9, we obtain a surface that is topologically equivalent to  $S_g$  and a cellular imbedding with vertex set  $V'$ , edge-set  $E'$ , and face set  $F'$ , such that  $|V'| - |E'| + |F'| = |V| - |E| + |F| + 2$ . By the induction hypothesis,  $|V'| - |E'| + |F'| = 2 - 2g$ . It follows that  $|V| - |E| + |F| = -2g = 2 - 2(g + 1)$ .  $\diamond$

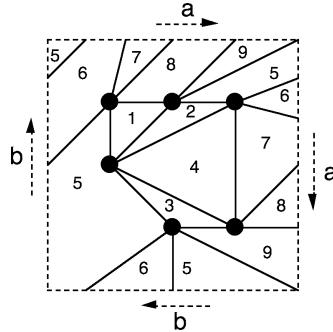
**Theorem 8.5.5.** *Let a graph  $G$  be cellularly imbedded in the non-orientable surface  $N_k$ . Then  $|V| - |E| + |F| = 2 - k$ .*

**Proof:** This proof follows the same pattern at that of Theorem 8.5.4, except that the role of the meridian is filled by the central cycle on a Möbius band.  $\diamond$

**Example 8.5.6:** The imbedding  $K_{3,3} \rightarrow S_1$  in Figure 8.5.6 above is cellular, and the value of the Euler polyhedral formula  $|V| - |E| + |F|$  is  $6 - 9 + 3 = 0 = 2 - 2g$ .

**Example 8.5.5, continued:** The imbedding  $K_{4,5} \rightarrow S_2$  in Figures 8.5.7 and 8.5.8 above is cellular, and the value of the Euler polyhedral formula  $|V| - |E| + |F|$  is  $9 - 20 + 9 = -2 = 2 - 2g$ .

**Example 8.5.7:** The imbedding of  $K_6$  into  $N_2$  in Figure 8.5.10 has 9 regions, each marked with a distinguishing numeral. Thus,  $|V| - |E| + |F| = 6 - 15 + 9 = 0 = 2 - k$ .



**Figure 8.5.10** An imbedding  $K_6 \rightarrow N_2$  with 9 regions.

**DEFINITION:** The **Euler characteristic** for a surface equals the value of the Euler polyhedral formula  $|V| - |E| + |F|$  for any cellular imbedding on that surface. It is denoted  $ec(S)$ . That is,

$$ec(S) = \begin{cases} 2 - 2g & \text{for the orientable surface } S_g \\ 2 - k & \text{for the non-orientable surface } N_k \end{cases}$$

**NOTATION:** Graph theorists usually denote the *chromatic number* of a graph  $G$  by  $\chi(G)$ , as we do in Chapter 9. Topologists and geometers usually denote the *Euler characteristic* of the surface  $S$  by  $\chi(S)$ . To avoid confusion when discussing topological graph theory, we use  $vcr(G)$  or  $cr(G)$  to denote the vertex chromatic number of a graph  $G$ , and we use  $ec(S)$  to denote the Euler characteristic of a surface  $S$ .

### Average Degree and General Surfaces

The following generalization of Theorem 8.4.2 is often useful in proving that a given graph cannot be imbedded in some given surface.

**Theorem 8.5.6.** *Let  $\iota : G \rightarrow S$  be an imbedding of a simple graph  $G$  on a surface  $S$  of Euler characteristic  $ec(S)$ . Then  $\delta_{\text{avg}}(G) \leq 6 - \frac{6 \cdot ec(S)}{|V|}$ .*

**Proof:** This follows the proof of Theorem 8.4.2 exactly, with details condensed.

$$\begin{aligned}
 (1) \quad & |F| \leq \frac{2|E|}{3} && (\text{from edge-face ineq.}) \\
 (2) \quad & |V| - |E| + |F| = ec(S) && (\text{Euler poly. eq.}) \\
 (3) \quad & |V| - \frac{|E|}{3} \geq ec(S) && (\text{subst. (1) into (2)}) \\
 (4) \quad & \frac{2|E|}{|V|} \leq 6 - \frac{6 \cdot ec(S)}{|V|} && (\text{multiply (3) by } \frac{6}{|V|}) \\
 (5) \quad & \delta_{\text{avg}}(G) = \frac{2|E|}{|V|} && (\text{by Theorem 1.1.2}) \\
 (6) \quad & \delta_{\text{avg}}(G) \leq 6 - \frac{6 \cdot ec(S)}{|V|} && (\text{combine (4) and (5)}) \quad \diamond
 \end{aligned}$$

**Example 8.5.8:** Since  $ec(N_1) = 1$ , the average degree of an imbedding in the non-orientable surface  $N_1$  is less than 6. Thus,  $K_7$  cannot be imbedded in  $N_1$ .

**Example 8.5.9:** The average degree of  $C_4 + C_5$  is  $6\frac{4}{9}$ , and  $ec(N_2) = 0$ . Theorem 8.5.6 implies that  $C_4 + C_5$  cannot be imbedded in  $N_2$ .

### EXERCISES for Section 8.5

*In Exercises 8.5.1 through 8.5.10, draw the given graph on a flat polygon representation of the torus.*

- |                    |                          |                    |                |
|--------------------|--------------------------|--------------------|----------------|
| 8.5.1 <sup>s</sup> | $Q_3 + K_1$ .            | 8.5.2              | $ML_3 + K_1$ . |
| 8.5.3              | $K_6$ .                  | 8.5.4 <sup>s</sup> | $K_{4,4}$ .    |
| 8.5.5 <sup>s</sup> | $K_7$ .                  | 8.5.6 <sup>s</sup> | $K_8 - 4K_2$ . |
| 8.5.7              | $K_8 - (K_3 \cup K_2)$ . | 8.5.8              | $K_8 - 2P_3$ . |
| 8.5.9              | $K_9 - C_9$ .            | 8.5.10             | $K_8 - C_5$ .  |

*In Exercises 8.5.11 through 8.5.14, draw the given graph on a flat polygon representation of the surface  $N_1$ .*

- |                     |          |        |             |
|---------------------|----------|--------|-------------|
| 8.5.11 <sup>s</sup> | $K_6$ .  | 8.5.12 | $K_{3,4}$ . |
| 8.5.13              | $ML_4$ . | 8.5.14 | $ML_5$ .    |

*In Exercises 8.5.15 through 8.5.18, draw the given graph on a flat polygon representation of the surface  $N_2$ .*

- |        |               |                     |               |
|--------|---------------|---------------------|---------------|
| 8.5.15 | $K_7 - K_2$ . | 8.5.16 <sup>s</sup> | $K_{4,4}$ .   |
| 8.5.17 | $K_9 - K_6$ . | 8.5.18              | $K_8 - C_5$ . |

For Exercises 8.5.19 through 8.5.26, draw a regular map of the designated graph on the torus.

- |                     |                    |        |             |
|---------------------|--------------------|--------|-------------|
| 8.5.19              | $B_2$ .            | 8.5.20 | $B_3$ .     |
| 8.5.21 <sup>s</sup> | $C_3 \times C_3$ . | 8.5.22 | $K_{3,3}$ . |
| 8.5.23              | $K_5$ .            | 8.5.24 | $K_7$ .     |
| 8.5.25              | $K_6 - 3K_2$ .     | 8.5.26 | $Q_4$ .     |
- 8.5.27<sup>s</sup> Prove that  $C_4 + C_4$  cannot be imbedded in  $N_1$ .  
 8.5.28 Prove that  $K_9 - 4K_2$  cannot be imbedded in  $S_1$  or in  $N_2$ .  
 8.5.29 Prove that  $K_{19}$  cannot be imbedded in  $S_{19}$ .
- 

## 8.6 GEOMETRIC DRAWINGS OF GRAPHS

Geometric drawing of graphs is a topic studied in computational geometry. Unlike topological graph drawings, geometric graph drawings are concerned with exact coordinates of the images of the vertices and edges of a graph, with lengths and areas, and with the angles formed in the drawings. Computer drawings are geometric drawings. Interest in geometric graphs is motivated in part by their many applications, which include computer graphics, pattern recognition, and communication networks. The intent of this brief, optional section is to introduce some types of geometric drawings, some properties of graph drawings, and some categories of problems involving geometric graph drawings.<sup>†</sup>

Geometric graph drawings avoid all three types of *forbidden singularities*. However, they need not be *normal drawings*. (See §8.3.)

### Some Types of Geometric Graph Drawings

Polyline drawings are piecewise-linear approximations of drawings with curved edges. The choice of polyline or straight-line drawings depends on the application.

DEFINITION: In a **straight-line drawing** of a graph, each edge is a single line segment.

DEFINITION: In a **polyline drawing of a graph** in a plane, each edge is a chain of line segments (see Figure 8.6.1(a)).

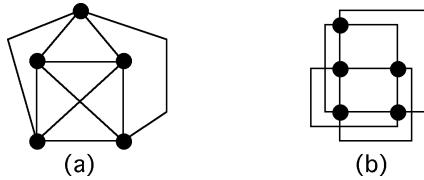
DEFINITION: A **bend in a polyline drawing** is a point where two segments belonging to the same edge meet.

DEFINITION: A **grid drawing** is a polyline drawing such that the vertices, crossings, and bends all have integer coordinates.

DEFINITION: An **orthogonal drawing**, in which each edge is a chain of horizontal and vertical segments (see Figure 8.6.1(b) below).

---

<sup>†</sup> No subsequent sections depend on this section.



**Figure 8.6.1** Drawings: (a) general polyline; (b) orthogonal grid.

### Some Properties of Graph Drawings

It is desirable to optimize various geometric properties of a graph, for instance, for the sake of miniaturization on a chip or of appearance on a video screen. One typically seeks to maximize the *angular resolution* and to minimize the other measures.

**DEFINITION:** The **area of a geometric drawing** is usually the area either of its convex hull or of the smallest rectangle with vertical and horizontal sides that covers the drawing.

**DEFINITION:** A **resolution rule for a drawing** is a constraint that prevents it from being arbitrarily scaled down, for instance, a minimum unit distance between vertices.

**DEFINITION:** The **aspect ratio of a drawing** is the ratio of the length of the longer side to the shorter side of the smallest rectangle covering the drawing.

**DEFINITION:** The **angular resolution** of a polyline drawing is the smallest angle formed by any two segments of an edge at a bend or between any two edges incident at the same vertex. It does not usually take into account the angle at which two edges cross.

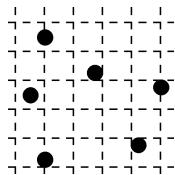
**DEFINITION:** The **total edge-length of a drawing** is the sum of the lengths of its edges.

### Minimizing Total Edge Length

**DEFINITION:** A **(geometric) spanning tree for a set  $S$  of points in  $\mathbb{R}^n$**  is a drawing of a tree that has  $S$  as its vertex set.

**DEFINITION:** A **(geometric) minimum spanning tree for a set of points in  $\mathbb{R}^n$**  is a geometric spanning tree that minimizes the total edge length. (This is provably a straight-line drawing.)

**DEFINITION:** The **Euclidean MST problem** is to produce a geometric minimum spanning tree for a set of points in  $\mathbb{R}^n$  (as in Figure 8.6.2), that is, to determine the pairs of points that are endpoints of the edges of the minimum spanning tree.



**Figure 8.6.2** Find the Euclidean minimum spanning tree.

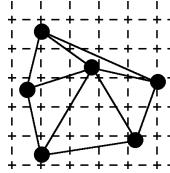
**Remark:** For a set of points in  $\mathbb{R}^2$ , knowing the locations of the endpoints permits a solution in  $O(n \log n)$  time, i.e., faster than Prim's algorithm.

**DEFINITION:** A **convex set**  $S$  in  $\mathbb{R}^n$  is a set such that for every pair of points  $x, y \in S$ , every point of the straight-line segment joining  $x$  and  $y$  is also in  $S$ .

**DEFINITION:** The **convex hull** of a set of points in  $\mathbb{R}^n$  is the smallest convex set that contains all the points.

**DEFINITION:** A **triangulation** of a set  $S$  of points in  $\mathbb{R}^2$  is a straight-line graph drawing whose vertices are the points in  $S$ , with the property that the graph subdivides the convex hull of  $S$  into triangular faces, as in Figure 8.6.3.

**DEFINITION:** A triangulation of a set of points in  $\mathbb{R}^2$  is a **minimum weight triangulation** if it minimizes the total edge length.



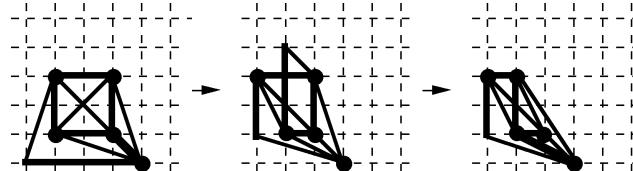
**Figure 8.6.3** Triangulation of a geometric set.

**Remark:** It is not presently known ([LiTa04]) whether the problem of finding minimum weight triangulations is NP-hard.

### Minimizing Area

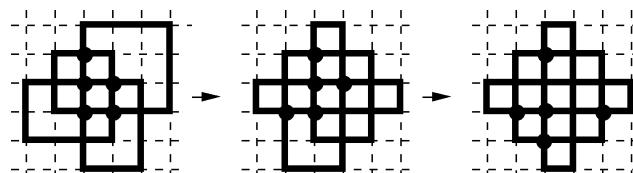
There are algorithms to shrink various kinds of polyline drawings. Doing it ad hoc is something of an art.

**Example 8.6.1:** Figure 8.6.4 shows three grid drawings of  $K_5$ , progressively reduced in area.



**Figure 8.6.4** Shrinking a grid drawing.

**Example 8.6.2:** Figure 8.6.5 shows three orthogonal grid drawings of  $K_5$ , progressively reduced in area.

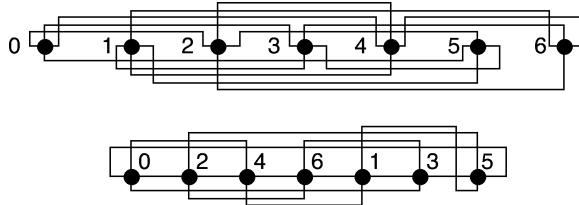


**Figure 8.6.5** Shrinking an orthogonal grid drawing.

### Representability

We seek combinatorial characterizations of graphs that are amenable to drawings of various types and with various geometric properties.

**BY ORTHOGONAL DRAWING.** Obviously, a graph with a vertex of degree more than 4 has no orthogonal drawing. To visualize how to construct an orthogonal drawing for an arbitrary graph with maximum degree at most 4, consider the orthogonal drawings of  $\text{circ}(7 : 2, 3)$  in Figure 8.6.6.



**Figure 8.6.6** Two orthogonal drawings of  $\text{circ}(7 : 2, 3)$ .

The lower drawing indicates savings in total edge-length and area if the vertices are placed judiciously.

**BY MINIMUM SPANNING TREE.** When drawing a tree  $T$  in the plane, quite possibly the geometric minimum spanning tree for the assigned locations of the vertices of  $T$  is not the tree  $T$  itself.

**DEFINITION:** A tree  $T$  is **minimum-weight drawable** if there exists a set of points whose geometric minimum spanning tree is isomorphic to  $T$ .

**Remark:** Each tree with maximum vertex degree at most 5 can be drawn as a minimum spanning tree of some set of vertices, by a linear-time algorithm, but no tree with maximum degree greater than 6 can be drawn as a minimum spanning tree ([MoSu92]). It is NP-hard to decide whether trees of maximum degree equal to 6 can be drawn as minimum spanning trees ([EaWh96]). No trees with maximum degree greater than 12 can be drawn as a Euclidean minimum spanning tree in  $\mathbb{R}^3$ , whereas all trees with vertex degree at most 9 are  $\mathbb{R}^3$ -drawable ([LiDi95]).

### Voronoi Diagrams and Delaunay Triangulations

Voronoi diagrams and Delaunay triangulations can be used to construct geometric minimum spanning trees.

**DEFINITION:** An  $n$ -dimensional **convex polytope** is the intersection of a finite number of half-spaces

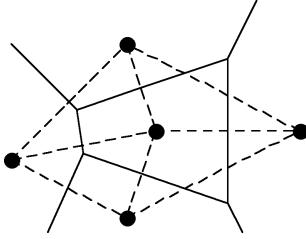
$$\{(x_1, \dots, x_n) \mid a_1x_1 + \dots + a_nx_n \geq b\}$$

It may be finite or infinite.

**DEFINITION:** The **Voronoi diagram** for a set  $S = \{p_1, \dots, p_n\}$  of points in  $\mathbb{R}^n$  is a partition of  $\mathbb{R}^n$  into  $n$  convex polytopes  $V(p_1), \dots, V(p_n)$  such that the interior of the region  $V(p_j)$  contains all points that are closer to  $p_j$  than to any other point in  $S$ .

**DEFINITION:** The **Delaunay graph** for a set  $S = \{p_1, \dots, p_n\}$  of points in the plane has the set  $S$  as its vertices. Two vertices  $p_i$  and  $p_j$  are joined by a straight-line (representing an edge) if and only if the Voronoi regions  $V(p_i)$  and  $V(p_j)$  share an edge.

Thus, the Delaunay graph of a set of points is isomorphic to a topological dual of their Voronoi diagram. Figure 8.6.7 shows the Voronoi diagram for a set of points in the plane and the corresponding Delaunay graph.



**Figure 8.6.7** Voronoi diagram and Delaunay graph.

Observe that their straight-line geometric specification may cause a Delaunay edge to traverse a region containing neither of its endpoints.

**DEFINITION:** A **Delaunay triangulation** of a point set  $P$  in  $\mathbb{R}^2$  is a straight-line triangulation with all internal faces triangles, such that three points are the vertices of a face if and only if their convex hull does not contain any other point of  $P$ .

**DEFINITION:** A graph is **Delaunay drawable** if it admits a drawing that is a Delaunay triangulation.

**Remark:** The Voronoi diagram of a point set can be constructed in  $O(n \log n)$ -time and used in an  $O(n \log n)$ -time algorithm to construct the geometric minimum spanning tree of the set. (See [Ch00].)

**Remark:** Finding a complete combinatorial characterization of Delaunay drawable graphs is an open problem. (See LiTa04.)

**Remark:** For a set of points in  $\mathbb{R}^2$ , knowing the locations of the endpoints permits a solution in  $\Omega(n \log n)$  time, i.e., faster than Prim's algorithm ([AgWe88].)

## 8.7 SUPPLEMENTARY EXERCISES

8.7.1 Draw an imbedding of the Möbius ladder  $ML_4$  in the torus.

8.7.2 Consider the following specifications for a graph imbedding: three 3-sided faces and four 4-sided faces. Either draw such an imbedding in the sphere or prove it is impossible.

8.7.3 Draw two cellular imbeddings of the following graph on a torus, such that the dual graphs are not isomorphic.



8.7.4 Draw the two dual graphs of Exercise 8.7.3 in the plane.

8.7.5 Draw a self-dual map of  $C_3 \times C_3$  on the torus.

**8.7.6** Either draw a self-dual imbedding of a regular simple graph in  $S_2$  or prove that such an imbedding cannot exist. (Hint: Use the Euler Polyhedral Equation and self-duality to determine the number of edges.)

**DEF:** A **2-complex**  $K = (V_K, E_K, R_K)$  is a generalization of a graph to a 2-dimensional object. It consists of a graph  $G = (V_K, E_K)$ , called the **1-skeleton of the complex**, and a set  $R_K$  of *regions*, to each of which is associated a closed walk in the graph that is the boundary walk of the polygon.

**8.7.7** Draw a 2-complex whose 1-skeleton is planar, but which cannot be drawn in the sphere with no overlap of edges or regions.

**8.7.8** What is the minimum number of vertices in a 2-complex that cannot be drawn in the sphere with no overlap of edges or regions.

## GLOSSARY

**abnormality in a drawing of a graph** on a surface: any of these three types of singularities:

- (i) The images of two edges meet at more than one point of the surface.
- (ii) The images of two different edges meet, but fail to actually cross each other (i.e., a tangency).
- (iii) Images of three or more edges meet at the same point of the surface.

**angular resolution** of a polyline drawing: the smallest angle formed by any two segments of an edge at a bend or between any two edges incident at the same vertex. It does not usually take into account the angle at which two edges cross.

**area of a geometric drawing:** usually the area either of its convex hull or of the smallest rectangle with vertical and horizontal sides that covers the drawing.

**aspect ratio** of a drawing: the ratio of the length of the longer side to the shorter side of the smallest rectangle covering the drawing.

**bend in a polyline drawing:** a point where two segments belonging to the same edge meet.

**boundary point of a surface:** a point that is not an interior point.

**boundary walk of a face  $f$ :** a closed walk that corresponds to a complete traversal of the perimeter of the face.

**boundaryless surface:** a surface such that every point is an interior point.

**carrier** of a graph: synonym for *topological model*.

**cellular imbedding**  $\iota : G \rightarrow S$ : a graph imbedding such that every region is topologically equivalent to an open disk.

—, **strongly**: a cellular imbedding whose regions are strongly cellular.

**cellular region** of a graph imbedding: a region that is topologically equivalent to an open disk.

—, **strongly**: a cellular region whose boundary walk is a cycle in the graph.

**central circuit of a Möbius band:** the circuit formed from the line halfway between the top and bottom of the rectangular strip when the right and left edges are identified.

**closed surface:** a connected surface that is finite and boundaryless, such that the endpoints of every open arc are in the surface itself.

**closed unit disk:** the plane set  $\{(x_1, x_2) \mid x_1^2 + x_2^2 \leq 1\}$ .

**2-complex  $K = (V_K, E_K, R_K)$ :** a generalization of a graph to a 2-dimensional object, comprising a graph  $G = (V_K, E_K)$ , called the *1-skeleton of the complex*, and a set  $R_K$  of *regions*, to each of which is associated a closed walk in the graph that is the boundary walk of the polygon.

**convex hull** of a set of points in  $\mathbb{R}^n$ : the smallest convex set that contains all the points.

**convex polytope:** the intersection of a finite number of half-spaces

$$\{(x_1, \dots, x_n) \mid a_1x_1 + \dots + a_nx_n \geq b\}$$

It may be of finite size or of infinite measure.

**convex set** in  $\mathbb{R}^n$ : a set  $S$  such that for every pair of points  $x, y \in S$ , every point of the straight-line segment joining  $x$  and  $y$  is also in  $S$ .

**crosscap** on a surface: a Möbius band that occurs as a subspace of that surface.

**—, adding a:** the operation on a surface of replacing a subspace that is homeomorphic to a closed disk by a Möbius band.

**crosscap number of a closed non-orientable surface:** the subscript of the surface  $N_k$  to which it is topologically equivalent, i.e., the number of Möbius bands on the surface.

**Delaunay drawable graph:** a graph that admits a drawing that is a *Delaunay triangulation*.

**Delaunay graph** for a set  $S = \{p_1, \dots, p_n\}$  of points in the plane: a graph whose vertex set is the set  $S$ , and such that two vertices  $p_i$  and  $p_j$  are joined by a straight-line (representing an edge) if and only if the *Voronoi regions*  $V(p_i)$  and  $V(p_j)$  share an edge.

**Delaunay triangulation** of a point set  $P$  in  $\mathbb{R}^2$ : a *straight-line triangulation* with all internal faces triangles, such that three points are the vertices of a face if and only if their convex hull does not contain any other point of  $P$ .

**donut, standard:** the solid obtained by revolving the  $xy$ -plane disk of radius 1 centered at  $(0,2)$  around the  $z$ -axis.

**drawing of a graph  $G = (V, E)$ :** the continuous image on a surface of a topological model of  $G$ , that is one-to-one everywhere except possibly the interior of one edge crossing the interior of another.

**—, grid drawing:** a polyline drawing such that the vertices, crossings, and bends all have integer coordinates.

**—, polyline:** a plane drawing in which each edge is a chain of line segments.

**—, orthogonal:** a drawing in which each edge is a chain of horizontal and vertical segments.

**—, straight-line:** a drawing such that each edge is a single line segment.

**dual graph and dual imbedding:** the new graph and its imbedding derived, starting with a cellular imbedding of a graph, by inserting a new vertex into each existing face, and by then drawing through each existing edge a new edge that joins the new vertex in the region on one side to the new vertex in the region on the other side.

**edge-crossing** in a graph drawing: a point on the surface where the images of two different edges meet.

**ends of a space edge:** the parts of the space edge nearest to the image of the endpoint(s).

—, **0-end:** the part of the space edge near the image of the 0-end of the unit interval  $[0, 1]$ .

—, **1-end:** the part of the space edge near the image of the 1-end of the unit interval  $[0, 1]$ .

**Euclidean metric:** the usual distance metric on real  $n$ -dimensional space

$$\delta((x_1, \dots, x_n), (y_1, \dots, y_n)) = \sqrt{(x_1 - y_1)^2 + \dots + (x_n - y_n)^2}$$

**Euclidean MST problem:** to produce a geometric minimum spanning tree for a set  $S$  of points in  $\mathbb{R}^n$ , that is, to determine the pairs of points that are endpoints of the edges of the minimum spanning tree.

**Euclidean set:** a subset  $X$  of a Euclidean space.

—, **connected:** a Euclidean set  $X$  such that for every pair of points  $s, t \in X$ , there exists a path in  $X$  from  $s$  to  $t$ .

—, **finite:** a Euclidean set in which the maximum distance of any point from the origin is at most  $M$ , for some real number  $M$ .

**Euclidean space:** the usual real vector space  $\mathbb{R}^n$ , for any positive integer  $n$ , with the Euclidean metric.

**Euler characteristic** of a surface: the value of the Euler polyhedral formula  $|V| - |E| + |F|$  for any cellular imbedding on that surface; it is usually denoted by  $\chi(S)$ . However, in the context of chromatic graph theory, it is denoted by  $ec(S)$ . Thus,

$$ec(S) = \begin{cases} 2 - 2g & \text{for the orientable surface } S_g \\ 2 - k & \text{for the non-orientable surface } N_k \end{cases}$$

**flat polygon representation** of a surface  $S$ : a drawing of a polygon with markings to match its sides in pairs, such that when the sides are pasted together as the markings indicate, the resulting surface obtained is topologically equivalent to  $S$ .

**forbidden types of singularities** in a graph drawing: illustrated in §8.3, but are otherwise absent from this book.

- (i) Two vertex images  $p_u$  and  $p_v$  occur at the same point of  $S$ .
- (ii) An edge image  $q_e$  has a self-intersection or other singularities.
- (iii) The interior of an edge image intersects a vertex image.

**genus of a closed orientable surface:** the subscript of the surface  $S_g$  to which it is topologically equivalent, i.e., the “number of handles” on the surface.

**geometric minimum spanning tree** for a set  $S$  of points in  $\mathbb{R}^n$ : a geometric spanning tree that minimizes the total edge length.

**geometric spanning tree** for a set  $S$  of points in  $\mathbb{R}^n$ : a drawing of a tree that has  $S$  as its vertex set.

**half-disk, standard:** the plane set  $\{(x_1, x_2) \mid x_1 \geq 0 \wedge x_1^2 + x_2^2 < 1\}$ .

**handle** on a surface: a subspace homeomorphic to the result of removing the interior of a closed disk from a torus.

—, **adding a handle**: the operation on a surface of replacing a subspace that is homeomorphic to a closed disk by a handle, or equivalently, by punching two separate holes in the surface  $S_n$  and then connecting them with a “tube”.

**imbedding of a graph  $G$** : a drawing with no edge-crossings at all.

**interior of a space curve**: all of the space curve except for its endpoints.

**interior point of a surface**: a point that has a  $\epsilon$ -neighborhood topologically equivalent to an open disk.

**Klein bottle**: the non-orientable surface  $N_2$  with two crosscaps.

**map on a surface**: an imbedding of a graph on that surface.

—, **regular**: an imbedding of a regular graph such that the dual graph is also a regular graph.

—, **self-dual**: an imbedding such that there is a topological equivalence from the imbedding surface to itself that takes the primal graph to the dual graph.

—, **simple**: an imbedding of a simple graph, such that the dual graph is also a simple graph.

**mapping**: a generic synonym for function.

**minimum-weight drawable tree**: a tree such that there exists a set  $S$  of points in  $\mathbb{R}^2$  whose geometric minimum spanning tree is isomorphic to  $T$ .

**Möbius band**: the space formed from a rectangular strip with a half-twist, by identifying the left edge to the right edge, to form a continuous band.

**neighborhood of a point  $y$  in a Euclidean set  $X$** : a subset  $N$  of  $X$  that contains the set  $\{z \in X \mid |z - y| < \epsilon\}$ , for some positive number  $\epsilon$ .

—, **open  $\epsilon$ -neighborhood**: the set of all points whose distance from  $y$  is less than  $\epsilon$ , where  $\epsilon > 0$ .

**non-orientable genus**: a synonym for *crosscap number*.

**non-orientable surface**: a surface that contains a Möbius band.

**non-orientable surfaces, classification sequence  $N_1, N_2, N_3, \dots$** : a recursively defined sequence starting from the sphere, which in this context is denoted  $N_0$ , even though it is orientable. The surface  $N_{n+1}$  is obtained from the surface  $N_n$  by *adding a crosscap*.

**normal drawing of a graph  $G$  in a surface  $S$** : a drawing such that (i) two different edges cross at most once; (ii) tangent edge touchings are not permitted; and (iii) images of three distinct edges never meet at the same point of the surface. It is usually assumed in graph theory that a drawing is normalized.

**open unit disk**: the plane set  $\{(x_1, x_2) \mid x_1^2 + x_2^2 < 1\}$ .

**orientable surface**: a surface that does not contain a Möbius band.

**orientable surfaces, classification sequence  $S_0, S_1, S_2, \dots$** : a recursively defined sequence such that  $S_0$  is the sphere,  $S_1$  is the torus, and surface  $S_{n+1}$  is obtained by *adding a handle*.

**path** from  $s$  to  $t$  in a Euclidean set: the image of a continuous function  $f$  from the unit interval  $[0, 1]$  to a subset of that space such that  $f(0) = s$  and  $f(1) = t$ , that is a bijection on the interior of  $[0, 1]$ . (One may visualize a path as the trace of a particle traveling through space for a fixed length of time.)

—, closed: a path such that  $f(0) = f(1)$ , i.e., in which  $s = t$ . (For instance, this would include a “knotted circle” in space.)

—, open: a path such that  $f(0) \neq f(1)$ , i.e., in which  $s \neq t$ .

**planar graph**: a graph that has an imbedding in the plane.

**plane**: Euclidean 2-space.

**platonic graph**: the 1-skeleton of a platonic solid.

**platonic map**: the imbedding of the 1-skeleton of a platonic solid on its surface.

**platonic solid**: any of the five *geometrically regular* 3-dimensional polyhedra: tetrahedron, octahedron, cube, dodecahedron, and icosahedron.

**projective plane**: the non-orientable surface  $N_1$  with one crosscap.

**resolution rule for a drawing**: a constraint that prevents it from being arbitrarily scaled down, for instance, a minimum unit distance between vertices.

**singularity in a graph drawing**: a place where the drawing is not one-to-one.

**1-skeleton** of a *2-complex*  $K = (V_K, E_K, R_K)$ : the graph  $G = (V_K, E_K)$ .

**space curve between two points  $x$  and  $y$  in 3-space**: the image of a continuous 1-to-1 function from the unit interval  $[0, 1]$  into 3-space that maps the endpoints 0 and 1 of  $[0, 1]$  to  $x$  and  $y$ .

**space edge**: see topological model of a graph.

**space vertex**: see topological model of a graph.

**sphere**: any Euclidean set that is topologically equivalent to the unit sphere.

**subspace of a Euclidean space**: any subset of the space, plus structure implicit in the Euclidean metric.

**surface**: a Euclidean set in which every point has a neighborhood that is topologically equivalent either to the open unit disk or to the standard half-disk.

—, boundaryless: a surface such that every point is an interior point.

**topological equivalence** between two Euclidean sets  $X$  and  $Y$ : a continuous function  $f : X \rightarrow Y$  that is one-to-one and onto, and whose inverse function  $f^{-1} : Y \rightarrow X$  is continuous.

**topological model of a graph  $G = (V, E)$** : a model of  $G$  in Euclidean 3-space  $\mathcal{R}^3$ , where each vertex  $v \in V$  is represented by a point  $p_v$  in 3-space, called a *space vertex*, and each proper edge  $e \in E$  is represented by a space curve  $q_e$  joining its endpoints, called a *space edge*. The topological model of a graph  $G$  is usually also denoted  $G$ .

**torus**: any Euclidean set that is topologically equivalent to the standard torus.

**total edge-length of a drawing**: the sum of the lengths of its edges.

**triangulation of a set  $S$  of points in  $\mathbb{R}^2$** : a straight-line graph drawing whose vertices are the points in  $S$ , with the property that the graph subdivides the convex hull of  $S$  into triangular faces.

—, minimum weight: a triangulation that minimizes the total edge length.

**unit cylinder**: the Euclidean set  $\{(x_1, x_2, x_3) \mid 0 \leq x_1 \leq 1 \wedge x_2^2 + x_3^2 = 1\}$ .

**unit sphere**: the Euclidean set  $\{(x_1, x_2, x_3) \mid x_1^2 + x_2^2 + x_3^2 = 1\}$ .

**Voronoi diagram** for a set  $S = \{p_1, \dots, p_n\}$  of points in  $\mathbb{R}^n$ : a partition of  $\mathbb{R}^n$  into  $n$  convex polytopes  $V(p_1), \dots, V(p_n)$  such that the interior of the region  $V(p_j)$  contains all points that are closer to  $p_j$  than to any other point in  $S$ .

# Chapter 9

---

## GRAPH COLORINGS

### **9.1 Vertex-Colorings**

### **9.2 Map-Colorings**

### **9.3 Edge-Colorings**

### **9.4 Factorization**

---

#### **INTRODUCTION**

The first known mention of coloring problems was in 1852, when Augustus De Morgan, Professor of Mathematics at University College, London, wrote Sir William Rowan Hamilton in Dublin about a problem posed to him by a former student, named Francis Guthrie. Guthrie noticed that it was possible to color the counties of England using four colors so that no two adjacent counties were assigned the same color. The question raised thereby was whether four colors would be sufficient for all possible decompositions of the plane into regions.

The Poincaré duality construction transforms this question into the problem of deciding whether it is possible to color the vertices of every planar graph with four colors so that no two adjacent vertices are assigned the same color. Wolfgang Haken and Kenneth Appel provided an affirmative solution in 1976.

Numerous practical applications involving graphs and some form of coloring are described in this chapter. Factorization is included here, because, like edge-coloring, it involves a partitioning of the edge-set of a graph.

## 9.1 VERTEX-COLORINGS

In the most common kind of graph coloring, colors are assigned to the vertices. From a standard mathematical perspective, the subset comprising all the vertices of a given color would be regarded a cell of a partition of the vertex-set. Drawing the graph with colors on the vertices is simply an intuitive way to represent such a partition.

**NOTATION:** The maximum degree of a vertex in a graph  $G$  is denoted  $\delta_{\max}(G)$ , or simply by  $\delta_{\max}$  when the graph of reference is evident from context.

### The Minimization Problem for Vertex-Colorings

Most applications involving *vertex-colorings* are concerned with determining the minimum number of colors required under the condition that the endpoints of an edge cannot have the same color.

**DEFINITION:** A **vertex  $k$ -coloring** is an assignment  $f : V_G \rightarrow C$  from its vertex-set onto the set  $C = \{1, \dots, k\}$ , or onto another set of cardinality  $k$ , whose elements are called *colors*. For any  $k$ , such an assignment is called a **vertex-coloring**.

**TERMINOLOGY:** Since vertex-colorings arise more frequently than edge-colorings or map-colorings, one often says *coloring*, instead of *vertex-coloring*, when the context is clear.

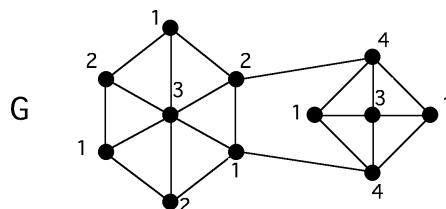
**DEFINITION:** A **color class** in a vertex-coloring of a graph  $G$  is a subset of  $V_G$  containing all the vertices of a given color.

**DEFINITION:** A **proper vertex-coloring** of a graph is a vertex-coloring such that the endpoints of each edge are assigned two different colors.

**Remark:** Quite commonly, it is implicit from context that the colorings under consideration are proper, in which case each color class is an independent set of vertices.

**DEFINITION:** A graph is said to be **vertex  $k$ -colorable** if it has a proper vertex  $k$ -coloring.

**Example 9.1.1:** The vertex-coloring shown in Figure 9.1.1 demonstrates that the graph is vertex 4-colorable.

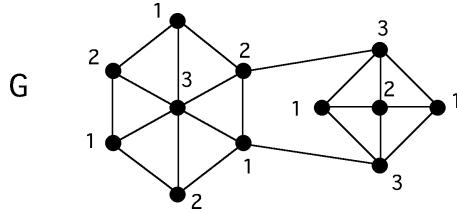


**Figure 9.1.1** A proper vertex 4-coloring of a graph.

**DEFINITION:** The (**vertex**) **chromatic number** of a graph  $G$ , denoted  $\chi(G)$ , is the minimum number of different colors required for a proper vertex-coloring of  $G$ . A graph  $G$  is (**vertex**)  **$k$ -chromatic** if  $\chi(G) = k$ .

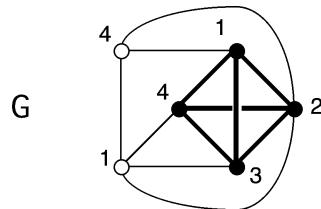
Thus,  $\chi(G) = k$ , if graph  $G$  is  $k$ -colorable but not  $(k - 1)$ -colorable.

**Example 9.1.1, continued:** The 3-coloring in Figure 9.1.2 shows that the graph  $G$  in Figure 9.1.1 is 3-colorable, which means that  $\chi(G) \leq 3$ . However, graph  $G$  contains three mutually adjacent vertices and hence is not 2-colorable. Thus,  $G$  is 3-chromatic.



**Figure 9.1.2** A proper 3-coloring of the graph from Example 9.1.1.

**Example 9.1.2:** The 4-coloring of the graph  $G$  shown in Figure 9.1.3 establishes that  $\chi(G) \leq 4$ , and the  $K_4$ -subgraph (drawn in bold) shows that  $\chi(G) \geq 4$ . Hence,  $\chi(G) = 4$ .



**Figure 9.1.3** Graph  $G$  is 4-colorable.

**Remark:** The study of vertex-colorings of graphs is customarily restricted to simple graphs. A graph with a self-loop is regarded as uncolorable, since the endpoint of the self-loop is adjacent to itself. Moreover, a multiple adjacency has no more effect on the colors of its endpoints than a single adjacency.

### Modeling Applications as Vertex-Coloring Problems

When an application is modeled as a vertex-coloring problem, the vertices in each color class typically represent individuals or items that do not compete or conflict with each other.

**Application 9.1.1 Assignment of Radio Frequencies:** Suppose that the vertices of a graph  $G$  represent transmitters for radio stations. In this model, two stations are considered adjacent when their broadcast areas overlap, which would result in interference if they broadcast at the same frequency. Then two “adjacent” stations should be assigned different transmission frequencies. Regarding the frequencies as colors transforms the situation into a graph-coloring problem, in which each color class contains vertices representing stations with no overlap. In this model,  $\chi(G)$  equals the minimum number of transmission frequencies required to avoid broadcast interference. (See Application 1.3.6.)

**Application 9.1.2 Separating Combustible Chemical Combinations:** Suppose that the vertices of a graph represent different kinds of chemicals needed in some manufacturing process. For each pair of chemicals that might explode if combined, there is an edge between the corresponding vertices. The chromatic number of this graph is the

number of different storage areas required so that no two chemicals that mix explosively are stored together.

**Application 9.1.3 University Course Scheduling:** Suppose that the vertices of a simple graph  $G$  represent the courses at a university. In this model, two vertices are adjacent if and only if at least one student preregisters for both of the corresponding classes. Clearly, it would be undesirable for two such courses to be scheduled at the same time. Then the vertex-chromatic number  $\chi(G)$  gives the minimum number of time periods in which to schedule the classes so that no student has a conflict between two courses.

**Application 9.1.4 Fast Register Allocation for Computer Programming:** In some computers, there are a limited number of special “registers” that permit faster execution of arithmetic operations than ordinary memory locations. The program variables that are used most often can be declared to have “register” storage class. Unfortunately, if the programmer declares more “register” variables than the number of hardware registers available, then the program execution may waste more time swapping variables between ordinary memory and the fast registers than is saved by using the fast registers. One solution is for the programmer to control the register designation, so that variables that are simultaneously active are assigned to different registers. The graph model has one vertex for each variable, and two vertices are adjacent if the corresponding variables can be simultaneously active. Then the chromatic number equals the number of registers needed to avoid the overswapping phenomenon.

### Sequential Vertex-Coloring Algorithm

There is a naive (brute-force) algorithm to decide, for some fixed  $k$ , whether a given  $n$ -vertex graph is  $k$ -colorable. Just check to see if any of the  $k^n$  possible  $k$ -colorings is proper and return YES if so. By iterating this decision procedure, starting with  $k = 1$ , until a YES is returned, one obtains an algorithm for calculating the exact value of the chromatic number. However, its running time is exponential in the number of vertices.

Alternatively, Algorithm 9.1.1 is a *sequential* algorithm that quickly produces a proper coloring of any graph; yet the coloring it produces is unlikely to be a minimum one. Moreover, it seems unlikely that *any* polynomial-time algorithm can accomplish this, since the problem of calculating the chromatic number of a graph is known to be NP-hard [GaJo79]. In fact, deciding whether a graph has a 3-coloring is an NP-complete problem.

**Algorithm 9.1.1: Sequential Vertex-Coloring**

*Input:* a graph  $G$  with vertex list  $v_1, v_2, \dots, v_p$ .

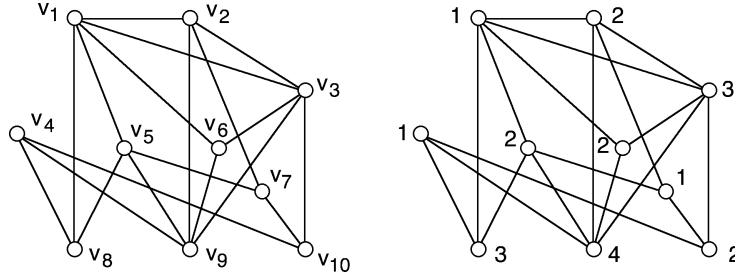
*Output:* a proper vertex-coloring  $f : V_G \rightarrow \{1, 2, \dots\}$ .

For  $i = 1, \dots, p$

Let  $f(v_i) :=$  the smallest color number not used on any  
of the smaller-subscripted neighbors of  $v_i$ .

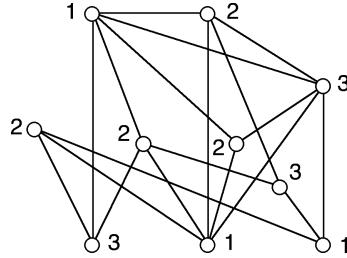
Return vertex-coloring  $f$ .

**Example 9.1.3:** When the sequential vertex-coloring algorithm is applied to the graph of Figure 9.1.4, the result is a 4-coloring.



**Figure 9.1.4** A graph and a sequential 4-coloring.

However, if the coloring assignment relaxes its obsessiveness with using the smallest possible color number at vertex  $v_4$  and at vertex  $v_7$ , then a 3-coloring can be obtained for the same graph, as shown in Figure 9.1.5.



**Figure 9.1.5** A nonsequential 3-coloring for the same graph.

**Remark:** Notice that if the vertices of the graph in Figure 9.1.5 are ordered so that  $v_1, v_2, v_3$  are the vertices with color 1, vertices  $v_4, v_5, v_6, v_7$  are the ones with color 2, and  $v_8, v_9, v_{10}$  are the vertices with color 3, then the sequential coloring algorithm would have produced that minimum coloring. More generally, for a given minimum coloring for a graph  $G$  (obtained by whatever means), if the vertices of  $G$  are linearly ordered so that all the vertices with color  $i$  precede all the vertices with color  $j$  whenever  $i < j$ , then, when  $G$  along with this vertex ordering is supplied as input to the sequential-coloring algorithm, the result is a minimum coloring. Thus, for any  $n$ -vertex graph, there is at least one vertex ordering (of the  $n!$  orderings) for which Algorithm 9.1.1 will produce a minimum coloring.

### Basic Principles for Calculating Chromatic Numbers

A few basic principles recur in many chromatic-number calculations. They combine to provide a direct approach involving two steps, as already seen for Examples 9.1.2 and 9.1.3.

- *Upper Bound:* Show  $\chi(G) \leq k$ , most often by exhibiting a proper  $k$ -coloring of  $G$ .
- *Lower Bound:* Show  $\chi(G) \geq k$ , most especially, by finding a subgraph that requires  $k$  colors.

The following result is an easy upper bound for  $\chi(G)$ ; it is complemented by the easy lower bound of clique number  $\omega(G)$ . Brooks's Theorem, appearing later in this section, sharpens this upper bound for a large class of graphs.

**Proposition 9.1.1.** Let  $G$  be a simple graph. Then  $\chi(G) \leq \delta_{\max}(G) + 1$ .

**Proof:** The sequential coloring algorithm never uses more than  $\delta_{\max}(G) + 1$  colors, no matter how the vertices are ordered, since a vertex cannot have more than  $\delta_{\max}(G)$  neighbors.  $\diamond$

**Proposition 9.1.2.** Let  $G$  be a graph that has  $k$  mutually adjacent vertices. Then  $\chi(G) \geq k$ .

**Proof:** Using fewer than  $k$  colors on graph  $G$  would result in a pair from the mutually adjacent set of  $k$  vertices being assigned the same color.  $\diamond$

REVIEW FROM §2.3: A **clique** in a graph  $G$  is a maximal subset of  $V_G$  whose vertices are mutually adjacent. The **clique number**  $\omega(G)$  of a graph  $G$  is the number of vertices in a largest clique in  $G$ .

**Corollary 9.1.3.** Let  $G$  be a graph. Then  $\chi(G) \geq \omega(G)$ .  $\diamond$

REVIEW FROM §2.3: The **independence number**  $\alpha(G)$  of a graph  $G$  is the number of vertices in an independent set in  $G$  of maximum cardinality.

**Proposition 9.1.4.** Let  $G$  be any graph. Then

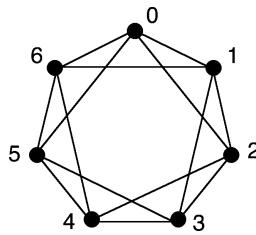
$$\chi(G) \geq \left\lceil \frac{|V_G|}{\alpha(G)} \right\rceil$$

**Proof:** Since each color class contains at most  $\alpha(G)$  vertices, the number of different color classes must be at least  $\left\lceil \frac{|V_G|}{\alpha(G)} \right\rceil$ .  $\diamond$

**Example 9.1.4:** Consider the graph  $G = \text{circ}(7 : 1, 2)$ , shown in Figure 9.1.6. No three vertices can be mutually non-adjacent, lest its edge-complement  $\overline{G} = \text{circ}(7 : 3)$ , a 7-cycle, contain a 3-cycle. Thus,  $\alpha(G) = 2$ , and, consequently, by Proposition 9.1.4,

$$\chi(G) \geq \left\lceil \frac{|V_G|}{\alpha(G)} \right\rceil = \left\lceil \frac{7}{2} \right\rceil = 4$$

There is a proper 4-coloring with color classes  $\{0, 3\}$ ,  $\{1, 4\}$ ,  $\{2, 5\}$ , and  $\{6\}$ .



**Figure 9.1.6**  $\text{Circ}(7 : 1, 2)$ .

**Proposition 9.1.5.** Let  $H$  be a subgraph of graph  $G$ . Then  $\chi(G) \geq \chi(H)$ .

**Proof:** Whatever colors are used on the vertices of subgraph  $H$  in a minimum coloring of graph  $G$  can also be used in a coloring of  $H$  by itself.  $\diamond$

REVIEW FROM §2.7: The **join**  $G + H$  of the graphs  $G$  and  $H$  is obtained from the graph union  $G \cup H$  by adding an edge between each vertex of  $G$  and each vertex of  $H$ .

**Proposition 9.1.6.** *The join of graphs  $G$  and  $H$  has chromatic number*

$$\chi(G + H) = \chi(G) + \chi(H)$$

**Proof:** *Lower Bound.* In the join  $G + H$ , no color used on the subgraph  $G$  can be the same as a color used on the subgraph  $H$ , since every vertex of  $G$  is adjacent to every vertex of  $H$ . Since  $\chi(G)$  colors are required for subgraph  $G$  and  $\chi(H)$  colors are required for subgraph  $H$ , it follows that  $\chi(G + H) \geq \chi(G) + \chi(H)$ .

*Upper Bound.* Just use any  $\chi(G)$  colors to properly color the subgraph  $G$  of  $G + H$ , and use  $\chi(H)$  different colors to color the subgraph  $H$ .  $\diamond$

### Chromatic Numbers for Common Graph Families

By using the basic principles given above, it is straightforward to establish the chromatic numbers of graphs in some of the most common graph families, which are summarized in Table 9.1.1.

**Table 9.1.1** Chromatic numbers for common graph families

Graph $G$	$\chi(G)$
trivial graph	1
bipartite graph	2
nontrivial path graph $P_n$	2
nontrivial tree $T$	2
cube graph $Q_n$	2
even cycle graph $C_{2n}$	2
odd cycle graph $C_{2n+1}$	3
even wheel $W_{2n}$	3
odd wheel $W_{2n+1}$	4
complete graph $K_n$	$n$

**Proposition 9.1.7.** A graph  $G$  has  $\chi(G) = 1$  if and only if  $G$  has no edges.

**Proof:** The endpoints of an edge must be colored differently.  $\diamond$

**Proposition 9.1.8.** A bipartite graph  $G$  has  $\chi(G) = 2$ , unless  $G$  is edgeless.

**Proof:** A 2-coloring is obtained by assigning one color to every vertex in one of the bipartition parts and another color to every vertex in the other part. If  $G$  is not edgeless, then  $\chi(G) \geq 2$ , by Proposition 9.1.2.  $\diamond$

**Corollary 9.1.9.** Path Graphs:  $\chi(P_n) = 2$ , for  $n \geq 2$ .

**Proof:**  $P_n$  is bipartite. Apply Proposition 9.1.8.  $\diamond$

**Remark:** Just as a bipartite graph is a graph whose vertices can be partitioned into two color classes, we define a **multipartite graph** for any number of color classes.

**DEFINITION:** A *k-partite graph* is a loopless graph whose vertices can be partitioned into  $k$  independent sets, which are sometimes called the **partite sets** of the partition.

**Corollary 9.1.10.** *Trees:  $\chi(T) = 2$ , for any nontrivial tree  $T$ .*

**Proof:** Trees are bipartite. ◊

**Corollary 9.1.11.** *Cube Graphs:  $\chi(Q_n) = 2$ .*

**Proof:** The cube graph  $Q_n$  is bipartite. ◊

**Corollary 9.1.12.** *Even Cycles:  $\chi(C_{2n}) = 2$ .*

**Proof:** An even cycle is bipartite. ◊

**Proposition 9.1.13.** *Odd Cycles:  $\chi(C_{2n+1}) = 3$ .*

**Proof:** Clearly,  $\alpha(C_{2n+1}) = n$ . Thus, by Proposition 9.1.4,

$$\chi(C_{2n+1}) \geq \left\lceil \frac{|V(C_{2n+1})|}{\alpha(V(C_{2n+1}))} \right\rceil = \left\lceil \frac{2n+1}{n} \right\rceil = 3 \quad \diamond$$

**REVIEW FROM §2.4:** The wheel graph  $W_n = K_1 + C_n$  is called an **odd wheel** if  $n$  is odd, and an **even wheel** if  $n$  is even.

**Proposition 9.1.14.** *Even Wheels:  $\chi(W_{2m}) = 3$ .*

**Proof:** Using the fact that  $W_{2n} = C_{2n} + K_1$ , Proposition 9.1.6 and Corollary 9.1.12 imply that  $\chi(W_{2m}) = \chi(C_{2m}) + \chi(K_1) = 2 + 1 = 3$ . ◊

**Proposition 9.1.15.** *Odd Wheels:  $\chi(W_{2m+1}) = 4$ , for all  $m \geq 1$ .*

**Proof:** Using the fact that the wheel graph  $W_{2m+1}$  is the join  $C_{2m+1} + K_1$ , Propositions 9.1.6 and 9.1.13 imply that  $\chi(W_{2m+1}) = \chi(C_{2m+1}) + \chi(K_1) = 3 + 1 = 4$ , for all  $m \geq 1$ . ◊

**Proposition 9.1.16.** *Complete Graphs:  $\chi(K_n) = n$ .*

**Proof:** By Proposition 9.1.2,  $\chi(K_n) \geq n$ . Moreover, any assignment of  $n$  colors to the  $n$  vertices of  $K_n$  is a proper  $n$ -coloring. ◊

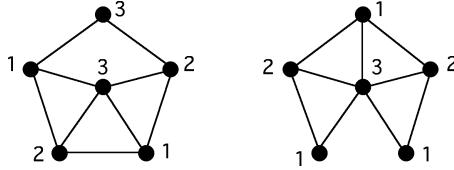
### Chromatically Critical Subgraphs

For a relatively small graph  $G$ , it is usually easier to calculate a plausible upper bound for  $\chi(G)$  than a lower bound. Finding a configuration, called an *obstruction*, that forces the number of colors to exceed some value is the general approach to establishing a lower bound. The presence of either a subgraph or a graph property can serve as an obstruction.

**DEFINITION:** A connected graph  $G$  is **(chromatically)  $k$ -critical** if  $\chi(G) = k$  and the edge-deletion subgraph  $G - e$  is  $(k-1)$ -colorable, for every edge  $e \in E_G$  (i.e., if  $G$  is an edge-minimal  $k$ -chromatic graph).

**Example 9.1.5:** An odd cycle is 3-critical, since deleting any edge yields a path, thereby reducing the chromatic number from 3 to 2.

**Example 9.1.6:** The odd wheel  $W_5$  is 4-chromatic, by Proposition 9.1.15, and Figure 9.1.7 illustrates how removing either a spoke-edge or a rim-edge reduces the chromatic number to 3. Thus,  $W_5$  is 4-critical. The argument that all odd wheels  $W_{2m+1}$  with  $m \geq 1$  are 4-critical is essentially the same (see Exercises).



**Figure 9.1.7** The graph  $W_5 - e$  is 3-colorable, for any edge  $e$ .

**Proposition 9.1.17.** Let  $G$  be a chromatically  $k$ -critical graph, and let  $v$  be any vertex of  $G$ . Then the vertex-deletion subgraph  $G - v$  is  $(k - 1)$ -colorable.  $\diamond$

**Theorem 9.1.18.** Let  $G$  be a chromatically  $k$ -critical graph. Then no vertex of  $G$  has degree less than  $k - 1$ .

**Proof:** By way of contradiction, suppose that  $v$  were a vertex of degree less than  $k - 1$ . Consider a  $(k - 1)$ -coloring of the vertex-deletion subgraph  $G - v$ , which exists by Proposition 9.1.17. The colors assigned to the neighbors of  $v$  would not include all the colors of the  $(k - 1)$ -coloring, because vertex  $v$  has fewer than  $k - 1$  neighbors. Thus, if  $v$  were restored to the graph, it could be colored with any one of the  $k - 1$  colors that was not used on any of its neighbors. This would achieve a  $(k - 1)$ -coloring of  $G$ , which is a contradiction.  $\diamond$

### Obstructions to $k$ -Chromaticity

**DEFINITION:** An **obstruction to  $k$ -chromaticity** (or  **$k$ -obstruction**) is a subgraph that forces every graph that contains it to have chromatic number greater than  $k$ .

**Example 9.1.7:** The complete graph  $K_{k+1}$  is an obstruction to  $k$ -chromaticity. (This is simply a restatement of Proposition 9.1.2.)

**Proposition 9.1.19.** Every  $(k + 1)$ -critical graph is an edge-minimal obstruction to  $k$ -chromaticity.

**Proof:** If any edge is deleted from a  $(k + 1)$ -critical graph, then, by definition, the resulting graph is not an obstruction to  $k$ -chromaticity.  $\diamond$

**DEFINITION:** A set  $\{G_j\}$  of chromatically  $(k + 1)$ -critical graphs is a **complete set of obstructions** if every  $(k + 1)$ -chromatic graph contains at least one member of  $\{G_j\}$  as a subgraph.

**Example 9.1.8:** The singleton set  $\{K_2\}$  is a complete set of obstructions to 1-chromaticity.

**REVIEW FROM §1.5:** [Characterization of bipartite graphs] A graph is bipartite if and only if it contains no cycles of odd length.

**Example 9.1.9:** The bipartite-graph characterization above implies that the family  $\{C_{2j+1} \mid j = 1, 2, \dots\}$  of all odd cycles is a complete set of 2-obstructions.

**Example 9.1.10:** Although the odd wheel graphs  $W_{2m+1}$  with  $m \geq 1$  are 4-critical, they do not form a complete set of 3-obstructions, since there are 4-chromatic graphs that contain no such wheel. The graph in Figure 9.1.8 does not contain an odd wheel, but its independence number is 2, which implies, by Proposition 9.1.4, that the graph is 4-chromatic.

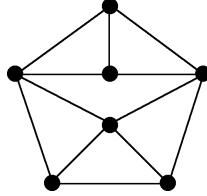


Figure 9.1.8 A 4-chromatic graph that contains no odd wheel.

### Brooks's Theorem

REVIEW FROM §5.4:

- A **block** of a loopless graph is a maximal connected subgraph  $H$  such that no vertex of  $H$  is a cut-vertex of  $H$ .
- A **leaf block** of a graph  $G$  is a block that contains exactly one cut-vertex of  $G$ .
- [1] A graph  $G$  with at least one cut-vertex has at least two leaf blocks.
- [2] Let  $B_1$  and  $B_2$  be distinct blocks of a connected graph  $G$ . Let  $y_1$  and  $y_2$  be vertices in  $B_1$  and  $B_2$ , respectively, such that neither is a cut-vertex of  $G$ . Then vertex  $y_1$  is not adjacent to vertex  $y_2$ .

**Lemma 9.1.20.** Let  $G$  be a non-complete,  $k$ -regular 2-connected graph with  $k \geq 3$ . Then  $G$  has a vertex  $x$  with two non-adjacent neighbors  $y$  and  $z$  such that  $G - \{y, z\}$  is a connected graph.

**Proof:** Let  $w$  be any vertex in graph  $G$ . First suppose that subgraph  $G - w$  is 2-connected. Then let  $z$  be any vertex at distance 2 from vertex  $w$  (which exists because the graph  $G$  is regular and non-complete). If  $x$  is the vertex between  $w$  and  $z$ , and if vertex  $y$  is taken to be  $w$ , then the conditions of the assertion are satisfied. Alternatively, suppose that  $\kappa_v(G - w) = 1$ . Then let  $B_1$  and  $B_2$  be two leaf blocks of  $G - w$  (which exist by review-result [1] above). Since graph  $G$  has no cut-vertices, it follows that vertex  $w$  is adjacent to some vertex  $y_1 \in B_1$  that is not a cut-vertex of  $G - w$  and, likewise, adjacent to some vertex  $y_2 \in B_2$  that is not a cut-vertex of  $G - w$ . By review-result [2], vertex  $y_1$  is not adjacent to vertex  $y_2$ . Hence, letting  $x$  be  $w$ ,  $y$  be  $y_1$ , and  $z$  be  $y_2$  completes the proof.  $\diamond$

**Theorem 9.1.21 [Brooks, 1941].** Let  $G$  be a non-complete, simple connected graph with maximum vertex degree  $\delta_{\max}(G) \geq 3$ . Then  $\chi(G) \leq \delta_{\max}(G)$ .

**Proof:** In this proof of [Lo75], the sequential coloring algorithm enables us to make some simplifying reductions. Suppose that  $|V_G| = n$ .

*Case 1.*  $G$  is not regular.

First choose vertex  $v_n$  to be any vertex of degree less than  $\delta_{\max}(G)$ . Next grow a

spanning tree (see §4.1) from  $v_n$ , assigning indices in decreasing order. In this ordering of  $V_G$ , every vertex except  $v_n$  has a higher-indexed neighbor along the tree path to  $v_n$ . Hence, each vertex has at most  $\delta_{\max}(G) - 1$  lower-indexed neighbors. It follows that the sequential coloring algorithm uses at most  $\delta_{\max}(G)$  colors.

*Case 2.*  $G$  is regular, and  $G$  has a cut-vertex  $x$ .

Let  $C_1, \dots, C_m$  be the components of the vertex-deletion subgraph  $G - x$ , and let  $G_i$  be the subgraph of  $G$  induced on the vertex-set  $V_{C_i} \cup x$ ,  $i = 1, \dots, m$ . Then the degree of vertex  $x$  in each subgraph  $G_i$  is clearly less than  $\delta_{\max}(G)$ . By Case 1, each subgraph  $G_i$  has a proper vertex coloring with  $\delta_{\max}(G)$  colors. By permuting the names of the colors in each such subgraph so that vertex  $x$  is always assigned color  $c_1$ , one can construct a proper coloring of  $G$  with  $\delta_{\max}(G)$  colors.

*Case 3.*  $G$  is regular and 2-connected.

By Lemma 9.1.20, graph  $G$  has a vertex, which we call  $v_n$ , with two non-adjacent neighbors, which we call  $v_1$  and  $v_2$ , such that  $G - \{v_1, v_2\}$  is connected. Grow a spanning tree from  $v_n$  in  $G - \{v_1, v_2\}$ , assigning indices in decreasing order. As in Case 1, each vertex except  $v_n$  has at most  $\delta_{\max}(G) - 1$  lower-indexed neighbors. It follows that the sequential coloring algorithm uses at most  $\delta_{\max}(G)$  colors on all vertices except  $v_n$ . Moreover, the sequential coloring algorithm assigns the same colors to  $v_1$  and  $v_2$  and at most  $k - 2$  colors on the other  $k - 2$  neighbors of  $v_n$ . Thus, one of the  $\delta_{\max}(G)$  colors is available for  $v_n$ .  $\diamond$

### Heuristics for Vertex-Coloring

Many vertex-coloring heuristics for graphs are based on the intuition that a vertex of large degree will be more difficult to color later than one of smaller degree [Ma81, Ca86, Br79], and that if two vertices have equal degree, then the one having the denser *neighborhood subgraph* will be harder to color later. These three references discuss variations and combinations of these ideas.

The following algorithm is one such combination. Assume that the colors are named  $1, 2, \dots$ . As the algorithm proceeds, the *colored degree* is the number of different colors used for the colored adjacent vertices of  $v$ .

**Algorithm 9.1.2: Vertex-Coloring: Largest Degree First**

*Input:* an  $n$ -vertex graph  $G$ .

*Output:* a vertex-coloring  $f$  of graph  $G$ .

While there are uncolored vertices of  $G$

Among the uncolored vertices with maximum degree,

choose vertex  $v$  with maximum colored degree.

Assign smallest possible color  $k$  to vertex  $v$ :  $f(v) := k$ .

Return graph  $G$  with vertex-coloring  $f$ .

**Example 9.1.11:** Descending degree order for the graph of Figure 9.1.9 (left) is

$$v_1, v_3, v_9, v_2, v_5, v_4, v_6, v_7, v_8, v_{10}$$

Applying the largest-degree-first algorithm yields a 3-coloring, as shown in Figure 9.1.9 (right).

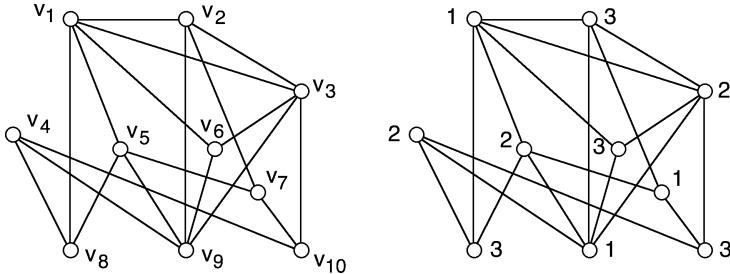


Figure 9.1.9 A largest-degree-first sequential coloring.

### Coloring the Vertices of an Edge-Weighted Graph

#### Application 9.1.5 Timetabling with Unavoidable Conflicts [KiYe92]

Suppose that the evening courses at a certain school must be scheduled in  $k$  timeslots,  $t_1, t_2, \dots, t_k$ . The school would like to schedule the courses so that, whenever at least three students are preregistered for both of a particular pair of courses, the two courses are assigned different timeslots. The graph model has a vertex-set corresponding to the set of courses, and has an edge between a pair of vertices if the corresponding pair of courses should be scheduled in different timeslots.

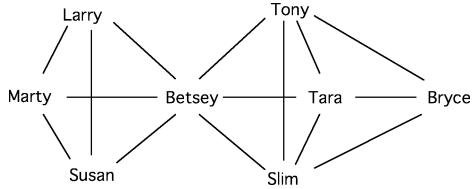
Typically, the chromatic number of such a graph is greater than  $k$ , which means that it is impossible to avoid *conflicts* (i.e., assigning the same timeslot to certain pairs of courses that should have been scheduled in different timeslots). A more realistic objective under such circumstances is to schedule the courses so that the total number of conflicts is minimized. Moreover, since some conflicts have greater impact than others, the number of students that preregistered for a given pair of courses is assigned as a weight to the corresponding edge. A further refinement of the model may adjust these edge-weights according to other factors not related to numbers of students. For instance, an edge corresponding to a pair of courses that are both required for graduation by some students or are to be taught by the same instructor should be assigned a large enough positive integer so as to preclude their being assigned to the same timeslot.

Suppose that  $w(e)$  denotes the positive integer indicating the deleterious impact of scheduling the two courses corresponding to the endpoints of edge  $e$  in the same timeslot. For a given  $k$ -coloring  $f$  (not necessarily *proper*), let  $z_f$  be the total edge-weight of those edges whose endpoints are assigned the same color. Then the objective is to find a  $k$ -coloring  $f$  for which  $z_f$  is as small as possible.

[KiYe92] develops a number of heuristics based on coloring the “most difficult” vertices as early as possible, where “most difficult” means having the greatest impact on the remaining vertices to be colored.

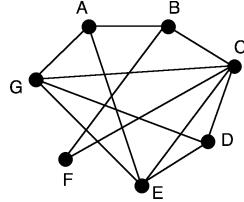
### EXERCISES for Section 9.1

- 9.1.1 The chromatic number of an acquaintance network tells the minimum number of groups into which the persons in that network must be partitioned so that no two persons in a group have prior acquaintance. Calculate the chromatic number of the following acquaintance network.

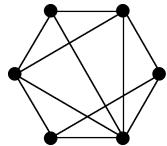


- 9.1.2** Calculate the number of different radio frequencies needed to avoid interference among the stations in the following configuration, from §1.3, in which two stations interfere if they are within 100 miles of each other.

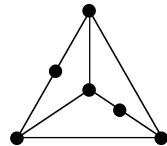
	B	C	D	E	F	G
A	55	110	108	60	150	88
B		87	142	133	98	139
C			77	91	85	93
D				75	114	82
E					107	41
F						123



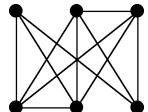
For Exercises 9.1.3 through 9.1.9, assign a minimum vertex-coloring to the given graph, and prove that it is a minimum coloring.

9.1.3<sup>s</sup>

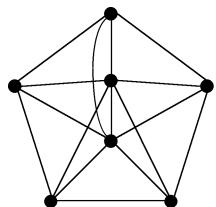
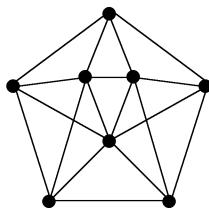
9.1.4



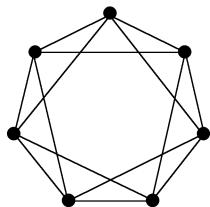
9.1.5



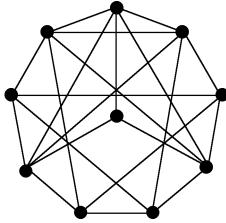
9.1.6

9.1.7<sup>s</sup>

9.1.8



9.1.9



For Exercises 9.1.10 through 9.1.16, apply the largest-degree-first heuristic to the given graph.

9.1.10<sup>s</sup> The graph of Exercise 9.1.3.

9.1.11 The graph of Exercise 9.1.4.

9.1.12 The graph of Exercise 9.1.5.

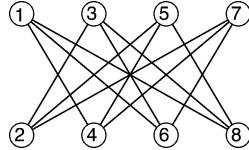
9.1.13 The graph of Exercise 9.1.6.

9.1.14<sup>s</sup> The graph of Exercise 9.1.7.

9.1.15 The graph of Exercise 9.1.8.

9.1.16 The graph of Exercise 9.1.9.

9.1.17 Apply the sequential vertex-coloring algorithm to this bipartite graph.



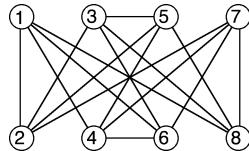
9.1.18 Give orderings to the vertices of the graphs of Figure 9.1.7 so that the sequential vertex-coloring algorithm will yield 3-colorings.

9.1.19<sup>s</sup> Prove that the sequential vertex-coloring algorithm always colors a complete bipartite graph with two colors, regardless of the order of its vertices in the input list.

9.1.20 Prove that adding an edge to a graph increases its chromatic number by at most one.

9.1.21 Prove that deleting a vertex from a graph decreases the chromatic number by at most one.

9.1.22<sup>s</sup> Apply the sequential vertex-coloring algorithm to this graph, with vertices in order of ascending index.



9.1.23 Apply the largest-degree-first heuristic to the graph of Exercise 9.1.22.

9.1.24 Prove that the chromatic number of an *interval graph* (§1.2) equals its clique number. (Hint: Apply the sequential vertex-coloring algorithm.)

9.1.25 Give an alternative proof of Theorem 9.1.17, using Theorem 9.1.15.

9.1.26 Generalize the result of the previous exercise to demonstrate that the number of colors used by the sequential algorithm can be arbitrarily larger than the chromatic number of a graph.

9.1.27<sup>s</sup> Prove that the wheel  $W_4$  is not chromatically critical.

9.1.28 Prove that all odd wheels are chromatically 4-critical.

9.1.29 Prove that  $K_7 - C_7$  is not 4-critical.

9.1.30 Prove that the join of two chromatically critical graphs is a chromatically critical graph.

For Exercises 9.1.31 through 9.1.35, calculate the independence number of the given graph.

9.1.31<sup>s</sup> The graph of Exercise 9.1.3.

9.1.32 The graph of Exercise 9.1.4.

9.1.33 The graph of Exercise 9.1.5.

9.1.34 The graph of Exercise 9.1.6.

9.1.35<sup>s</sup> The graph of Exercise 9.1.7.

9.1.36 Describe how to construct a connected graph  $G$  with independence number  $\alpha(G) = a$  and chromatic number  $\chi(G) = c$ , for arbitrary values  $a \geq 1$  and  $c \geq 2$ .

**DEFINITION:** The **domination number** of a graph  $G$ , denoted  $\text{dom}(G)$  is the cardinality of a minimum set  $S$  of vertices such that every vertex of  $G$  is either in  $S$  or a neighbor of a vertex in  $S$ . (This previews §10.2.)

For Exercises 9.1.37 through 9.1.41, calculate the domination number of the given graph.

9.1.37<sup>s</sup> The graph of Exercise 9.1.3.

9.1.38 The graph of Exercise 9.1.4.

9.1.39 The graph of Exercise 9.1.5.

9.1.40 The graph of Exercise 9.1.6.

9.1.41<sup>s</sup> The graph of Exercise 9.1.7.

9.1.42 Construct a graph with chromatic number 5 and domination number 2.

9.1.43 Construct a graph with domination number 5 and chromatic number 2.

9.1.44 Describe how to construct, for arbitrary values  $c \geq 2$  and  $m \geq 1$ , a connected graph  $G$  with chromatic number  $\chi(G) = c$  and domination number  $\text{dom}(G) = m$ .

**DEFINITION:** A graph  $G$  is **perfect** if for every induced subgraph (§2.3)  $H$  in  $G$ , the chromatic number  $\chi(H)$  equals the clique number  $\omega(H)$ .

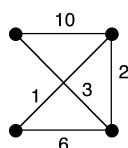
9.1.45<sup>s</sup> Prove that every bipartite graph is perfect.

9.1.46 Prove that the edge-complement of a perfect graph is a perfect graph.

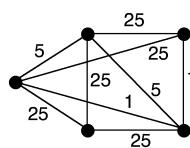
9.1.47 [Computer Project] Implement Algorithms 9.1.1 and 9.1.2 and compare their results on the graphs of Exercises 9.1.3 through 9.1.9.

For Exercises 9.1.48 and 9.1.49, find a vertex 2-coloring of the given weighted graph such that the total weight of the edges whose endpoints are assigned the same color is minimized.

9.1.48



9.1.49



**9.1.50 [Computer Project]** Design and write a computer program to construct a vertex  $k$ -coloring of a weighted graph that tries to minimize the total weight of the edges whose endpoints receive the same color. Use the heuristic discussed in the last subsection of this section. Test your program on various edge-weighted graphs, including the ones in Exercises 9.1.48 and 9.1.49, and compare its vertex-colorings to the ones you obtained by hand.

**9.1.51 Application 9.1.6 Examination Scheduling:** Suppose that four final exams are to be scheduled in three timeslots. It would be ideal if no two of the exams were assigned the same timeslot, but that would require four timeslots. The table shown below assigns a number to each exam pair, indicating the penalty for scheduling those two exams in the same timeslot. Find an examination schedule that minimizes the total penalty.

	$e_1$	$e_2$	$e_3$	$e_4$
$e_1$	—	4	16	4
$e_2$	4	—	4	16
$e_3$	1	16	—	4
$e_4$	4	1	4	—

## 9.2 MAP-COLORINGS

REVIEW FROM §8.4: A **map** on a surface is an imbedding of a graph on that surface.

For every closed surface  $S$ , there is a minimum number  $\text{chr}(S)$  of colors sufficient so that every map on  $S$  can be colored *properly* with  $\text{chr}(S)$  colors, which means that no color meets itself across an edge. The methods needed to establish a narrow range of possibilities for that minimum sufficient number are elementary enough to be presented in this book. Tightening the range to a single value was the substance of two of the outstanding mathematical problems solved in the 20th century, the Four-Color map problem for the plane (and sphere) and the Heawood map problem for all the other closed surfaces. In this section, the possibilities for  $\text{chr}(S_0)$  are narrowed down to four and five. The Heawood map problem is a principal topic in Chapter 16.

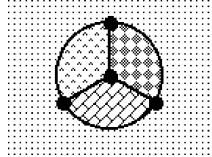
### Dualizing Map-Colorings into Vertex-Colorings

DEFINITION: A **map  $k$ -coloring** for an imbedding  $\iota : G \rightarrow S$  of a graph on a surface is an assignment  $f : F \rightarrow C$  from the face-set  $F$  onto the set  $C = \{1, \dots, k\}$ , or onto another set of cardinality  $k$ , whose elements are called *colors*. For any  $k$ , such an assignment is called a **map-coloring**.

DEFINITION: A map-coloring is **proper** if for each edge  $e \in E_G$ , the regions that meet on edge  $e$  are colored differently.

DEFINITION: The **chromatic number of a map**  $\iota : G \rightarrow S$  is the minimum number  $\text{chr}(\iota)$  of colors needed for a proper coloring.

**Example 9.2.1:** Figure 9.2.1 below shows a proper 4-coloring of a planar map. Observe that in a proper coloring of this map, no two regions can have the same color, since every pair of regions meets at an edge. Thus, this map requires four colors.

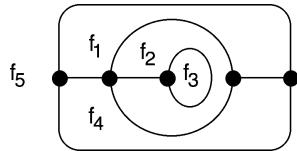


**Figure 9.2.1** A 4-colored planar map.

**TERMINOLOGY:** A region is said to **meet itself on edge**  $e$  if edge  $e$  occurs twice in the region's boundary walk. It is said to **meet itself on vertex**  $v$  if vertex  $v$  occurs more than once in the region's boundary walk.

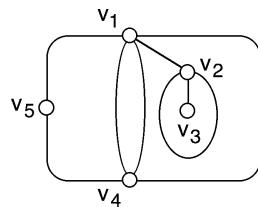
**Remark:** A map cannot be properly colored if a region meets itself.

**Example 9.2.2:** The map in Figure 9.2.2 has no proper coloring, since region  $f_2$  meets itself on an edge (which cannot occur in a geographic map). Observe also that region  $f_1$  meets region  $f_4$  in two distinct edges.



**Figure 9.2.2** A map with no proper coloring.

Long ago, it was realized that a coloring problem for the regions of a map on any closed surface can be converted by Poincaré duality (see §7.5) into a vertex-coloring problem for the dual graph. The dual of the map in Figure 9.2.2 is the graph in Figure 9.2.3, with dual vertex  $v_j$  corresponding to primal face  $f_j$ , for  $j = 1, \dots, 5$ . Thus, the self-adjacent region  $f_2$  dualizes to the self-adjacent vertex  $v_2$ .



**Figure 9.2.3** The dual graph for the map of Figure 9.2.2.

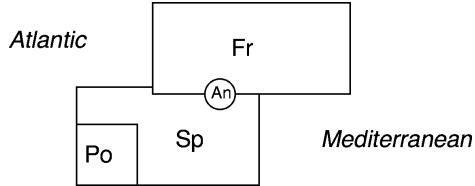
**Proposition 9.2.1.** *The chromatic number of a map equals the chromatic number of its dual graph.*  $\diamond$

### Geographic Maps

When the surface is the plane, sometimes a collection of contiguous regions are colored, and the other regions are ignored.

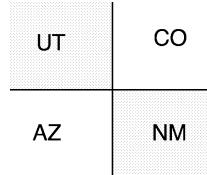
**Application 9.2.1 Political Cartography:** In the cartography of political maps, various interesting configurations arise. For instance, France and Spain meet on two distinct borders, one from Andorra to the Atlantic Ocean, the other from Andorra to the

Mediterranean Sea, as represented in Figure 9.2.4. Similar configurations occur where Switzerland meets Austria twice around Liechtenstein. Moreover, India and China have a triple adjacency around Nepal and Bhutan. Multiple adjacency of regions does not affect the rules for coloring a map.



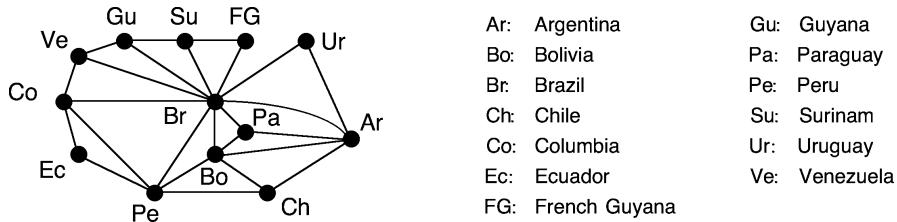
**Figure 9.2.4** Double adjacency of France and Spain around Andorra.

**Remark:** Two faces that meet at a vertex but not along an edge may have the same color in a proper map coloring. Thus, a checkerboard configuration such as the Four Corners, USA, representation in Figure 9.2.5, may be properly colored with only two colors.



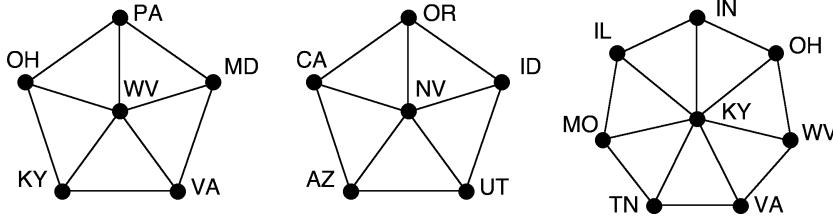
**Figure 9.2.5** A proper 2-coloring at Four Corners, USA.

**Example 9.2.3:** The chromatic number of the map of the countries of South America is equal to 4. In the dual graph, which appears in Figure 9.2.6, there is a 5-wheel with Bolivia as the hub and a 3-wheel with Paraguay as hub. Thus, by Proposition 9.1.15, South America requires at least four colors. We leave it as an Exercise to give a 4-coloring of South America (see Exercises).



**Figure 9.2.6** The dual graph of the map of South America.

**Example 9.2.4:** The chromatic number of the map of the United States of America is four. The graph of the USA contains three odd wheels, as illustrated in Figure 9.2.7 below. West Virginia and Nevada are each encircled by five neighbors, and Kentucky is encircled by seven neighbors.



**Figure 9.2.7** The three odd wheels in the map of the USA.

**Remark:** Utah meets five other states across an *edge* in the map, but these five do not quite encircle Utah, since Arizona and Colorado do not meet at an *edge*, even though they do meet at Four Corners. Thus, the Utah configuration is the join  $P_5 + K_1$ , and not the wheel  $W_5$ .

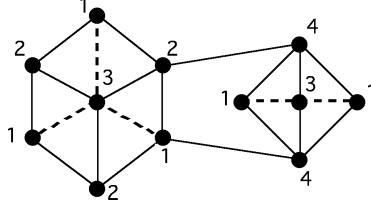
### Five-Color Theorem for Planar Graphs and Maps

An early investigation of the Four Color Problem by A. B. Kempe [1879] introduced a concept that enabled Heawood [1890] to prove without much difficulty that five colors are sufficient. Heawood's proof is the main concern of this section.

**DEFINITION:** The  $\{i, j\}$ -**subgraph** of a graph  $G$  with a vertex-coloring that has  $i$  and  $j$  in its color set is the subgraph of  $G$  induced on the subset of all vertices that are colored either  $i$  or  $j$ .

**DEFINITION:** A **Kempe  $i-j$  chain** for a vertex-coloring of a graph is a component of the  $\{i, j\}$ -subgraph.

**Example 9.2.5:** Figure 9.2.8 illustrates two Kempe 1-3 chains in a graph coloring. The edges in the Kempe chains are dashed.



**Figure 9.2.8** A graph coloring with two Kempe 1-3 chains.

**Theorem 9.2.2 [Heawood, 1890].** *The chromatic number of a planar simple graph is at most 5.*

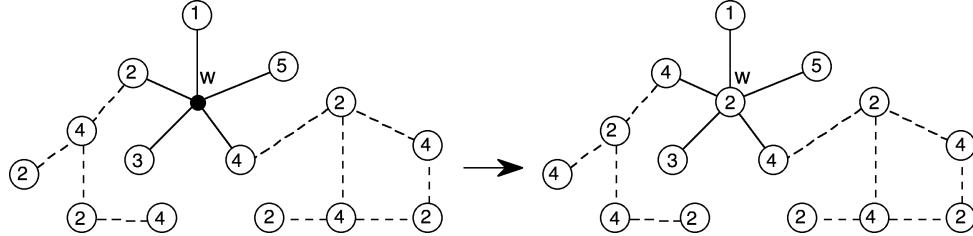
**Proof:** Starting with an arbitrary planar graph, edges are removed until a chromatically critical graph  $G$  is obtained. It suffices to prove that  $G$  is 5-colorable.

Since  $\delta_{\text{avg}}(G) < 6$  (by Theorem 8.4.2), there is a vertex  $w \in V_G$  of degree at most 5. Theorem 9.1.18 implies that  $\chi(G) \leq 6$ . Thus, the vertex-deletion subgraph  $G - w$  is 5-colorable, by Proposition 9.1.17.

Next, consider any 5-coloring of subgraph  $G - w$ . If not all five colors were used on the neighbors of vertex  $w$ , then the 5-coloring of  $G - w$  could be extended to graph  $G$  by assigning to  $w$  a color not used on the neighbors of  $w$ . Thus, we can assume that all five

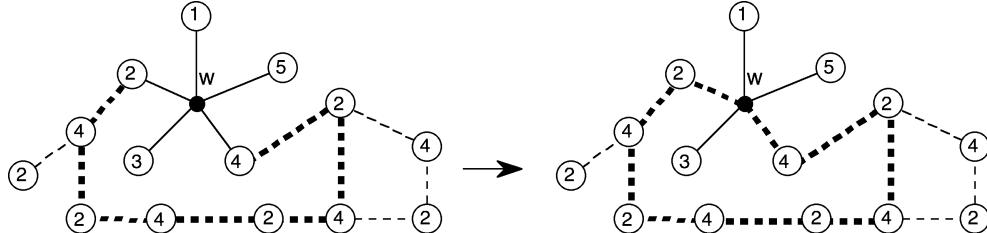
colors are assigned to the neighbors of vertex  $w$ . Moreover, there is no loss of generality in assuming that these colors are consecutive in counterclockwise order, as shown on the left in Figure 9.2.9. Consider the  $\{2, 4\}$ -subgraph shown on the left in Figure 9.2.9 with dashed edges, and let  $K$  be the Kempe 2-4 chain that contains the 2-neighbor of vertex  $w$  (i.e., the neighbor that was assigned color 2).

*Case 1.* Suppose that Kempe chain  $K$  does not also contain the 4-neighbor of vertex  $w$ . Then colors 2 and 4 can be swapped in Kempe chain  $K$ , as shown on the right in Figure 9.2.9. The result is a 5-coloring of  $G - w$  that does not use color 2 on any neighbor of  $w$ . This 5-coloring extends to a 5-coloring of graph  $G$  when color 2 is assigned to vertex  $w$ .



**Figure 9.2.9** Swapping colors in a Kempe 2-4 chain.

*Case 2.* Suppose that Kempe chain  $K$  contains both the 4-neighbor and the 2-neighbor of vertex  $w$ . Then there is a path in Kempe chain  $K$  from the 2-neighbor to the 4-neighbor, as illustrated with a bold broken path on the left in Figure 9.2.10. Appending the edges between vertex  $w$  and both these neighbors extends that path to a cycle, as depicted on the right in Figure 9.2.10. By the Jordan Curve Theorem (§7.1), this cycle separates the plane.



**Figure 9.2.10** Extending a path in a Kempe 2-4 chain to a cycle.

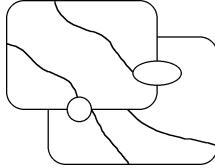
Since the 3-neighbor and 5-neighbor of  $w$  are on different sides of the separation, it follows that the Kempe 3-5 chain  $L$  containing the 5-neighbor cannot also contain the 3-neighbor. Thus, it is possible to swap colors in Kempe chain  $L$  and assign color 5 to vertex  $w$ , thereby completing a 5-coloring of  $G$ .  $\diamond$

**Theorem 9.2.3 [Appel and Haken, 1976].** *Every planar graph is 4-colorable.*  $\diamond$

**Remark:** The proof by Appel and Haken [ApHa76] of the Four Color Theorem is highly specialized, intricate, and long. Following an approach initiated by Heesch, Appel and Haken first reduced the seemingly infinite problem of considering every planar graph to checking a finite, unavoidable set of (over 1900) reducible configurations. Over 1200 hours of computer time were used. Eventually, a more concise proof was derived by Robertson, Sanders, Seymour, and Thomas [RoSaSeTh97].

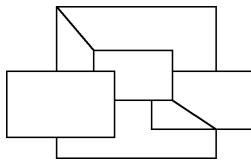
**EXERCISES for Section 9.2**

**9.2.1** Draw a minimum proper coloring of the following map, excluding the exterior region, and prove it is a minimum coloring.



**9.2.2<sup>s</sup>** Draw a minimum proper coloring of the map of Exercise 9.2.1, including the exterior region, and prove it is a minimum coloring.

**9.2.3** Draw a minimum proper coloring of the following map, excluding the exterior region, and prove it is a minimum coloring.

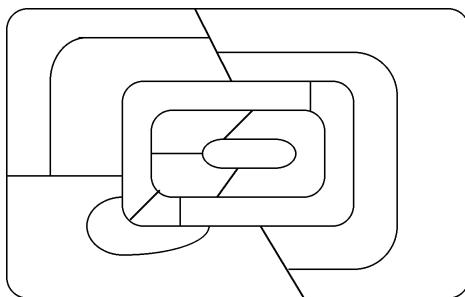


**9.2.4** Draw a minimum proper coloring of the map of Exercise 9.2.3, including the exterior region, and prove it is a minimum coloring.

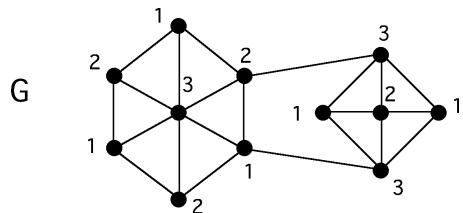
**9.2.5<sup>s</sup>** Is it possible for a map in the plane to be 4-chromatic when the exterior region is included, but 3-chromatic when it is excluded? Explain your answer.

**9.2.6** Draw the dual graph of the map of Exercise 9.2.3, and give it a minimum vertex-coloring.

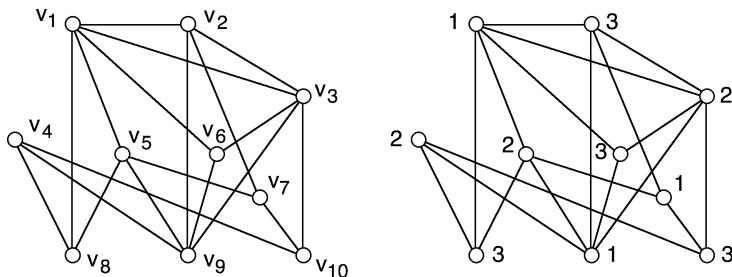
**9.2.7** Draw a minimum proper coloring of the following map, excluding the exterior region, and prove it is a minimum coloring.



**9.2.8<sup>s</sup>** Find the Kempe 1-4 chains in the following graph.

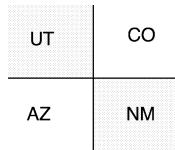


9.2.9 Find the Kempe 1-3 chains in the following graph.



REVIEW FROM §7.1: The *Riemann stereographic projection* provides a correspondence between the plane and the sphere.

9.2.10 Convert the Four Corners map to a sphere map, by redrawing it so that the “infinite lines” all meet at the restored *infinity* point.



9.2.11 Give a 4-coloring of the map (see Figure 9.2.6) of the countries of South America, or of the dual graph.

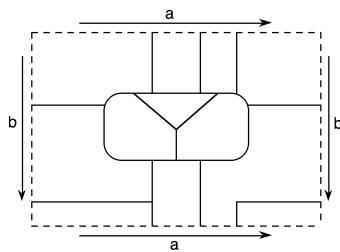
9.2.12 Consult a map of Europe, if necessary, and draw a map representing the following countries: France, Belgium, Netherlands, Luxembourg, Germany, and Switzerland. What is the chromatic number of this map?

9.2.13 Calculate the chromatic number of the map of the countries of North America.

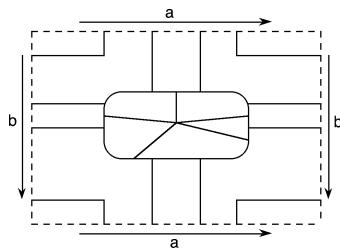
9.2.14<sup>s</sup> Calculate the chromatic number of the map of the countries of Africa.

9.2.15 Calculate the chromatic number of the map of the countries of Asia.

9.2.16<sup>s</sup> Calculate the chromatic number of the following map on the torus.



9.2.17 Calculate the chromatic number of the following map on the torus.



## 9.3 EDGE-COLORINGS

For certain problems, the most natural graph model for a problem might involve edge-colorings instead of vertex-colorings. Analogous to vertex-colorings, an edge-coloring partitions the edge-set of a graph into color classes. Although the *line-graph* transformation converts an edge-coloring problem into a vertex-coloring problem, the theory of edge-colorings has some special aspects.

### The Minimization Problem for Edge-Colorings

**REVIEW FROM §1.1:** Two different edges are **adjacent edges** if they have at least one endpoint in common.

**DEFINITION:** An **edge  $k$ -coloring** is an assignment  $f : E_G \rightarrow C$  from its edge-set onto the set  $C = \{1, \dots, k\}$ , or onto another set of cardinality  $k$ , whose elements are called *colors*. For any  $k$ , such an assignment is called an **edge-coloring**.

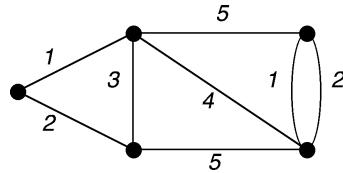
**DEFINITION:** An **edge color class** in a edge-coloring of a graph  $G$  is a subset of  $E_G$  containing all the edge of some color.

**DEFINITION:** A **proper edge-coloring** of a graph is an edge-coloring such that adjacent edges are assigned different colors.

**Remark:** Whereas multi-edges have no bearing on the proper vertex-colorings of a graph, they have an obvious effect on the proper edge-colorings and cannot be ignored. Graphs with self-loops are excluded from the present discussion.

**DEFINITION:** A graph is said to be **edge  $k$ -colorable** if it has a proper edge  $k$ -coloring.

**Example 9.3.1:** A proper edge 5-coloring of a graph is shown in Figure 9.3.1.

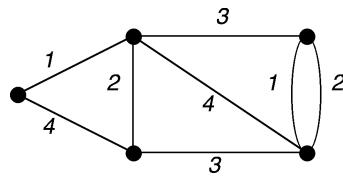


**Figure 9.3.1** A graph with a proper edge 5-coloring.

**DEFINITION:** The **edge-chromatic number** of a graph  $G$ , denoted  $\chi'(G)$ , is the minimum number of different colors required for a proper edge-coloring of  $G$ . A graph  $G$  is **edge  $k$ -chromatic** if  $\chi'(G) = k$ .

Thus,  $\chi'(G) = k$  if graph  $G$  is edge  $k$ -colorable but not edge  $(k - 1)$ -colorable.

**Example 9.3.1, continued:** The proper edge-coloring in Figure 9.3.2 improves on the one in Figure 9.3.1, since it uses only four colors. Moreover, the graph is not edge 3-colorable, since it contains four mutually adjacent edges. Thus,  $\chi'(G) = 4$ .

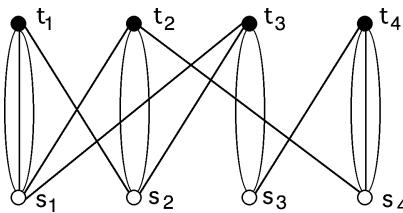


**Figure 9.3.2** A proper edge 4-coloring of the graph from Figure 9.3.1.

### Modeling Applications as Edge-Coloring Problems

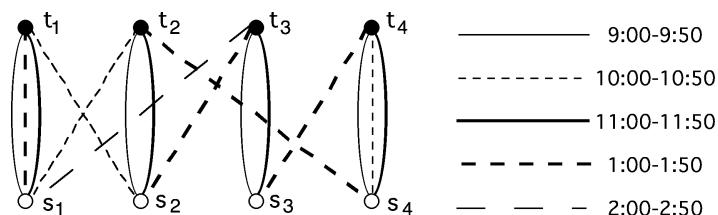
**Application 9.3.1** *Circuit Boards:* Some electronic devices  $x_1, x_2, \dots, x_n$  are on a board. The connecting wires emerging from each device must be colored differently, so that they can be distinguished. The least number of colors required is the edge-chromatic number of the associated network.

**Application 9.3.2** *Scheduling Class Times:* A high school has teachers  $t_1, \dots, t_m$  to teach courses  $s_1, \dots, s_n$ . In particular, teacher  $t_j$  must teach  $s_{j,k}$  sections of course  $s_k$ . **PROBLEM:** Calculate the minimum number of time periods required to schedule all the courses so that no two sections of the same course are taught at the same time. **SOLUTION:** Form a bipartite graph on the two sets  $\{t_1, \dots, t_m\}$  and  $\{s_1, \dots, s_n\}$  so that there are  $s_{j,k}$  edges joining  $t_j$  and  $s_k$ , for all  $j$  and  $k$ , as shown in Figure 9.3.3.



**Figure 9.3.3** Representing a scheduling problem by a bipartite graph.

A matching of teachers to courses can be realized in a time period. If each edge-color represents a timeslot in the schedule, then an edge-coloring of the bipartite graph represents a feasible timetable for sections of courses. A minimum edge-coloring, as shown in Figure 9.3.4, uses the smallest number of time periods.



**Figure 9.3.4** A minimum proper edge-coloring for the graph of Figure 9.3.3.

### Sequential Edge-Coloring Algorithm

There is a sequential edge-coloring algorithm analogous to the sequential vertex-coloring algorithm of §9.1.

**DEFINITION:** A **neighbor of an edge**  $e$  is another edge that shares one or both of its endpoints with  $e$ .

**Algorithm 9.3.1: Sequential Edge-Coloring**

*Input:* a graph with edge list  $e_1, e_2, \dots, e_p$ .

*Output:* a proper edge-coloring  $f$ , with positive integers as colors

For  $i = 1, \dots, p$

Let  $f(e_i) :=$  the smallest color number not used on any  
of the smaller-subscripted neighbors of  $e_i$ .

Return edge-coloring  $f$ .

### Basic Principles for Calculating Edge-Chromatic Numbers

Edge-chromatic-number calculations are largely based on a few simple principles, mostly analogous to those used in the two-step vertex-chromatic-number calculations.

- *Upper Bound:* Show  $\chi'(G) \leq k$  by exhibiting a proper edge  $k$ -coloring of  $G$ .
- *Lower Bound:* Show  $\chi'(G) \geq k$  by using properties of graph  $G$ .

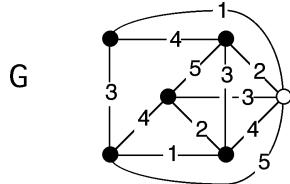
The next three results help in establishing a lower bound for the edge-chromatic number. They are immediate consequences of the definitions.

**Proposition 9.3.1.** Let  $G$  be a graph that has  $k$  mutually adjacent edges. Then  $\chi'(G) \geq k$ .  $\diamond$

**Corollary 9.3.2.** For any graph  $G$ ,  $\chi'(G) \geq \delta_{\max}(G)$ .  $\diamond$

**Proposition 9.3.3.** Let  $H$  be a subgraph of graph  $G$ . Then  $\chi'(G) \geq \chi'(H)$ .  $\diamond$

**Example 9.3.2:** The edge 5-coloring of the graph  $G$  in Figure 9.3.5 establishes that  $\chi'(G) \leq 5$ , and the existence of a 5-valent vertex shows that  $\chi'(G) \geq 5$ , by Corollary 9.3.2. Hence,  $\chi'(G) = 5$ .



**Figure 9.3.5** Graph  $G$  is edge 5-colorable.

### Matchings

A **matching** is the edge analogue of an independent vertex set. It can be used to obtain another lower bound for the edge-chromatic number.

**DEFINITION:** A **matching** (or **independent set of edges**) of a graph  $G$  is a subset of edges of  $G$  that are mutually non-adjacent.

**DEFINITION:** A **maximum matching** in a graph is a matching with the maximum number of edges.

**NOTATION:** The cardinality of a maximum matching in a graph  $G$  is denoted  $\alpha'(G)$ , analogous to the independence number  $\alpha(G)$  for the vertices.

**Remark:** It follows immediately from the definition that each color class of a proper edge-coloring of a graph  $G$  is a matching of  $G$ . The following proposition provides a lower bound on the edge-chromatic number that is based on the size of a maximum matching.

**Proposition 9.3.4.** *For any graph  $G$ ,  $\chi'(G) \geq \left\lceil \frac{|E_G|}{\alpha'(G)} \right\rceil$ .*  $\diamondsuit$  (Exercises)

The following algorithm constructs a proper edge-coloring by iteratively finding maximum matchings.

**Algorithm 9.3.2: Edge-Coloring by Maximum Matching**

*Input:* a graph  $G$ .

*Output:* a proper edge  $k$ -coloring  $f$ .

Initialize color number  $k := 0$ .

While  $E_G \neq \emptyset$

$k := k + 1$

    Find a maximum matching  $M$  of graph  $G$ .

    For each edge  $e \in M$

$f(e) := k$

$G := G - M$  (edge-deletion subgraph)

Return edge-coloring  $f$ .

**COMPUTATIONAL NOTE:** There are low-order polynomial-time algorithms that find maximum matchings, based largely on the work of Jack Edmonds. Bipartite matching is discussed in §13.4, and references to algorithms and applications of matchings in general graphs are given there.

### Edge-Chromatic Numbers for Common Graph Families

It is now possible to derive the edge-chromatic numbers for the same graph families, summarized below in Table 9.3.1, for which the vertex-chromatic numbers were derived in §9.1. The first of the following six results are analogous to their companion results for vertex-chromatic number and are left as exercises.

**Proposition 9.3.5.** *A graph  $G$  has  $\chi'(G) = 1$  if and only if  $\delta_{\max}(G) = 1$ .*

$\diamondsuit$  (Exercises)

**Proposition 9.3.6.** *Path Graphs:  $\chi'(P_n) = 2$ , for  $n \geq 3$ .*

$\diamondsuit$  (Exercises)

**Proposition 9.3.7.** *Even Cycle Graphs:  $\chi'(C_{2n}) = 2$ .*

$\diamondsuit$  (Exercises)

**Proposition 9.3.8.** *Odd Cycle Graphs:  $\chi'(C_{2n+1}) = 3$ .*

$\diamondsuit$  (Exercises)

**Proposition 9.3.9.** *Trees:  $\chi'(T) = \delta_{\max}(T)$ , for any tree  $T$ .*

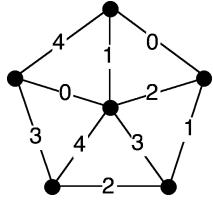
$\diamondsuit$  (Exercises)

**Proposition 9.3.10.** *Cube Graphs:  $\chi'(Q_n) = n$ .*

$\diamondsuit$  (Exercises)

**Proposition 9.3.11.** *Wheel Graphs:*  $\chi'(W_n) = n$ , for  $n \geq 3$ .  $\diamond$  (Exercises)

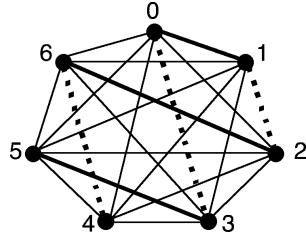
**Example 9.3.3:** Figure 9.3.6 illustrates an edge 5-coloring of the wheel  $W_5$ .



**Figure 9.3.6** A proper edge 5-coloring of the wheel  $W_5$ .

**Proposition 9.3.12.** *Odd Complete Graphs:*  $\chi'(K_n) = n$  for all odd  $n \geq 3$ .

**Proof:** *Upper bound:* Draw the complete graph  $K_n$  so that its vertices are the vertices of a regular  $n$ -gon, labeled  $0, 1, 2, \dots, n-1$  clockwise around the  $n$ -gon (illustrated in Figure 9.3.7 for  $n=7$ ). Observe that the edge joining vertices  $0$  and  $1$  is parallel to all edges whose endpoints also sum to  $1 \pmod{n}$ . Since no two of these parallel edges (depicted as bold edges in Figure 9.3.7) are adjacent, they all can be assigned color 1.



**Figure 9.3.7** Two sets of parallel edges in  $K_7$ .

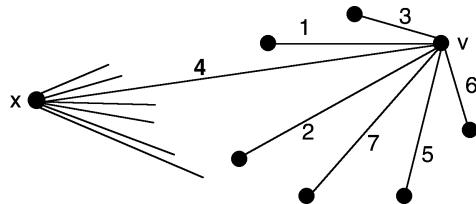
Similarly, the edges whose endpoints sum to  $3 \pmod{n}$  form a set of parallel edges (the dashed edges in Figure 9.3.7) that can be assigned the color 3. In all, there are  $n$  sets  $S_1, S_2, \dots, S_n$ , where  $S_k$  is the set of parallel edges whose endpoints sum to  $k \pmod{n}$ . Thus, if each of the edges in set  $S_k$  is assigned color  $c_k$ , then a proper edge  $n$ -coloring of  $K_n$  is obtained.

*Lower bound:* The size of a maximum matching in  $K_n$  with  $n$  odd is  $\frac{n-1}{2}$ . Since  $K_n$  contains  $\binom{n}{2} = \frac{n(n-1)}{2}$  edges, Proposition 9.3.4 implies that  $\chi'(K_n) \geq n$ .  $\diamond$

**Corollary 9.3.13.** *Even Complete Graphs:*  $\chi'(K_n) = n-1$  for all even  $n$ .

**Proof:** The even complete graph  $K_n$  is the join of the odd complete graph  $K_{n-1}$  with a single vertex  $x$ . The proof of Proposition 9.3.12 constructs a proper edge  $n$ -coloring of  $K_{n-1}$  in which each edge-color is missing at exactly one vertex. Thus, the edge-coloring of  $K_{n-1}$  can be extended to an edge-coloring of  $K_n$  by assigning the missing color at each vertex  $v \in K_{n-1}$  to the edge joining vertex  $v$  to vertex  $x$  (see Figure 9.3.8 below).  $\diamond$

Table 9.3.1 below summarizes the edge-chromatic numbers of the common graph families considered here, and also of the bipartite graphs, which we discuss later in this section.



**Figure 9.3.8** Extending the edge  $n$ -coloring from  $K_{n-1}$  to  $K_n$ .

**Table 9.3.1** Edge-chromatic numbers for common graph families

Graph $G$	$\chi'(G)$
graph $G$ with $\delta_{\max}(G) = 1$	1
path graph $P_n$ , $n \geq 3$	2
even cycle graph $C_{2n}$	2
odd cycle graph $C_{2n+1}$	3
bipartite graph $G$	$\delta_{\max}(G)$
tree $T$	$\delta_{\max}(T)$
cube graph $Q_n$	$n$
wheel $W_n$ , $n \geq 3$	$n$
even complete graph $K_{2n}$	$n - 1$
odd complete graph $K_{2n+1}$	$n$

### Chromatic Incidence

The next few definitions pertain to *all* edge-colorings, not just to proper ones.

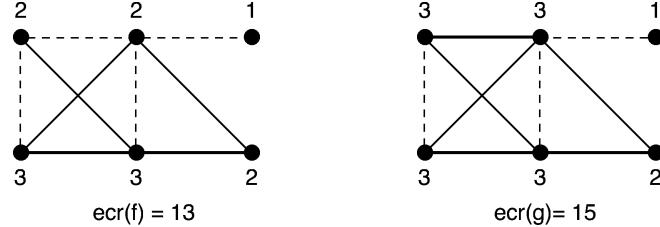
**DEFINITION:** For a given edge-coloring of a graph, color  $i$  is an **incident edge-color** on vertex  $v$  if some edge incident on  $v$  has been assigned color  $i$ . Otherwise, color  $i$  is an **absent edge-color** at vertex  $v$ .

**DEFINITION:** The **chromatic incidence at  $v$**  of a given edge-coloring  $f$  is the number of different edge-colors incident on vertex  $v$ . It is denoted  $ecr_v(f)$ .

**DEFINITION:** The **total chromatic incidence** for an edge-coloring  $f$  of a graph  $G$ , denoted  $ecr(f)$ , is the sum of the chromatic incidences of all the vertices. That is,

$$ecr(f) = \sum_{v \in V_G} ecr_v(f)$$

**Example 9.3.4:** For the edge-colorings shown in Figure 9.3.9 below, the three different edge-colors are represented by dashed, regular, and bold edges, instead of color numbers. This is to avoid confusion with the chromatic incidence numbers on the vertices. For the edge-coloring  $f$  on the left, the total chromatic incidence is 13, and for edge-coloring  $g$  on the right, the total chromatic incidence is 15.

**Figure 9.3.9** Total-chromatic-incidence calculations.

The following four assertions are immediate consequences of the definitions.

**Proposition 9.3.14.** Let  $f$  be any edge-coloring of a graph  $G$ . Then for every  $v \in G$ ,

$$\text{ecr}_v(f) \leq \deg(v)$$

◊

**Corollary 9.3.15.** Let  $f$  be any edge-coloring of a graph  $G$ . Then

$$\sum_{v \in V_G} \text{ecr}_v(f) \leq \sum_{v \in V_G} \deg(v)$$

◊

**Proposition 9.3.16.** An edge-coloring  $f$  of a graph  $G$  is proper if and only if for every vertex  $v \in V_G$ ,

$$\text{ecr}_v(f) = \deg(v)$$

◊

**Corollary 9.3.17.** An edge-coloring  $f$  of a graph  $G$  is proper if and only if

$$\sum_{v \in V_G} \text{ecr}_v(f) = \sum_{v \in V_G} \deg(v)$$

◊

### Edge-Coloring of Bipartite Graphs

Deriving a formula for the edge-chromatic number of a bipartite graph  $G$  is not quite as easy as for the vertex-chromatic number. Nonetheless, the eventual formula is uncomplicated:  $\chi'(G) = \delta_{\max}(G)$ . The characterization of bipartite graphs as the graphs without odd cycles is crucial to the derivation.

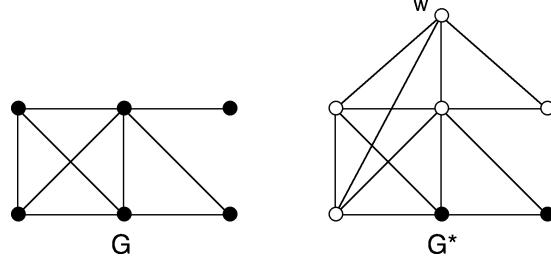
The following two lemmas establish facts about the chromatic degree that are used in the derivation. They involve edge-colorings that are *not* assumed to be proper. The first lemma makes use of the properties of an *eulerian graph*.

REVIEW FROM §1.5: An **eulerian tour** in a graph is a closed trail that contains every edge of that graph. An **eulerian graph** is a graph that has an eulerian tour.

**Lemma 9.3.18.** Let  $G$  be a connected graph with at least two edges that is not an odd-cycle graph. Then  $G$  has an edge 2-coloring such that both colors are incident on every vertex of degree at least 2.

**Proof:** *Case 1:*  $G$  is an even cycle. The edge 2-coloring obtained by assigning two edge-colors that alternate around the cycle meets the requirement.

*Case 2:*  $G$  is eulerian but not a cycle. Consider an eulerian tour that starts (and ends) at a vertex of degree at least 4. Assign color 1 to the edges that occur as odd terms in the edge sequence of the tour, and assign color 2 to the even-term edges. Then the two colors are incident at least once on each internal vertex of the tour, since each such vertex is an endpoint of both an odd-term edge and an even-term edge. Moreover, since the start vertex has degree at least 4, it also occurs on the tour as an internal vertex. Thus, both colors are incident on every vertex.

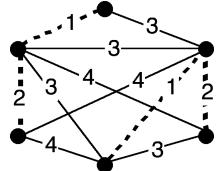


**Figure 9.3.10** Constructing the auxiliary graph for Case 3.

*Case 3:*  $G$  is not eulerian. Construct an auxiliary graph  $G^*$  by joining a new vertex  $w$  to every odd-degree vertex of  $G$ , thereby making each such vertex even-degree in  $G^*$  (see Figure 9.3.10 above). By a corollary to Euler's degree-sum equation (§1.1), every graph has an even number of odd-degree vertices, so vertex  $w$  has even degree. Thus, the auxiliary graph  $G^*$  is eulerian, by the eulerian-graph characterization (§4.5). Next, let  $f$  be an edge 2-coloring of graph  $G^*$ , as specified in Case 2. Then it is easy to verify that the edge-coloring  $f|_G$  of  $f$  restricted to the edges of graph  $G$  is an edge-coloring such that both colors are incident on each vertex of  $G$  of degree at least 2.  $\diamond$

**DEFINITION:** In a graph  $G$  with a (possibly improper) edge-coloring, a **Kempe  $i-j$  edge-chain** is a component of the subgraph of  $G$  induced on all the  $i$ -colored and  $j$ -colored edges.

**Example 9.3.5:** An edge 4-coloring is shown in Figure 9.3.11, and the two Kempe 1-2 edge-chains are shown as dashed edges.



**Figure 9.3.11** An edge 4-coloring with two Kempe 1-2 edge-chains.

**Lemma 9.3.19.** Let  $f$  be an edge  $k$ -coloring of a graph  $G$  with the largest possible total chromatic incidence. Let  $v$  be a vertex on which some color  $i$  is incident at least twice and on which some color  $j$  is not incident at all. Then the Kempe  $i-j$  edge-chain  $K$  containing vertex  $v$  is an odd cycle.

**Proof:** By Lemma 9.3.18, if the Kempe  $i-j$  edge-chain  $K$  incident on vertex  $v$  were not an odd cycle, then we could rearrange edge colors  $i$  and  $j$  within  $K$  so that the chromatic incidence of the coloring of  $K$  would be 2 at every vertex. The edge-coloring

for  $G$  thereby obtained would have higher chromatic incidence at vertex  $v$  and at-least-equal chromatic incidence at every other vertex of  $G$ . This would contradict the premise that edge-coloring  $f$  has the maximum possible total chromatic incidence.  $\diamond$

**Theorem 9.3.20 [König, 1916].** *Let  $G$  be a bipartite graph. Then  $\chi'(G) = \delta_{\max}(G)$ .*

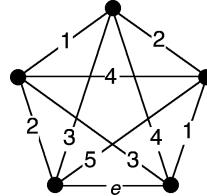
**Proof:** Let  $\Delta = \delta_{\max}(G)$ , and, by way of contradiction, suppose that  $\chi'(G) \neq \Delta$ . Then, by Corollary 9.3.2,  $\Delta < \chi'(G)$ . Next, let  $f$  be an edge  $\Delta$ -coloring of graph  $G$  for which the total chromatic incidence  $ecr_G(f)$  is maximum. Since  $f$  is not a proper edge-coloring, there is a vertex  $v$  such that  $ecr_v(f) < \deg(v)$  (by Proposition 9.3.16). Thus, some color occurs on at least two edges incident on  $v$ . But there are  $\Delta - 1$  other colors and at most  $\Delta - 2$  other edges incident on  $v$ , which means that some other color is not incident on vertex  $v$ . It follows by Lemma 9.3.19, that graph  $G$  contains an odd cycle, which contradicts the fact that  $G$  is bipartite.  $\diamond$

### Vizing's Theorem

Complementing the lower bound  $\chi'(G) \geq \delta_{\max}(G)$  for a simple graph, provided by Corollary 9.3.2, Vizing's theorem provides a sharp upper bound that narrows the range for  $\chi'(G)$  to two possible values.

**DEFINITION:** Let  $G$  be a graph, and let  $f$  be a proper edge  $k$ -coloring of a subset  $S$  of the edges of  $G$ . Then  $f$  is a **blocked partial edge  $k$ -coloring** if for each uncolored edge  $e$ , every color has already been assigned to the edges that are adjacent to  $e$ . Thus,  $f$  cannot be extended to any edge outside subset  $S$ .

**Example 9.3.6:** In Figure 9.3.12, the edge 5-coloring of all but one of the edges of  $K_5$  is blocked, since all five colors have been assigned to the neighbors of the uncolored edge  $e$ .



**Figure 9.3.12** Attempted edge 5-coloring of  $K_5$  that is blocked at edge  $e$ .

**Lemma 9.3.21.** *Let  $i$  and  $j$  be two of the colors used in a proper edge-coloring of a graph. Then every Kempe  $i-j$  edge-chain  $K$  is a path (open or closed).*

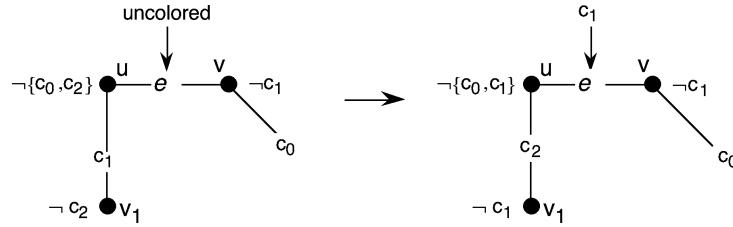
**Proof:** Every vertex of Kempe chain  $K$  has degree at most 2 (since the edge-coloring is proper), and, by definition,  $K$  is a connected subgraph.  $\diamond$

**Theorem 9.3.22 [Vizing, 1964, 1965] [Gupta, 1966].** *Let  $G$  be a simple graph. Then there exists a proper edge-coloring of  $G$  that uses at most  $\delta_{\max}(G) + 1$  colors.*

**Proof:** To construct such an edge-coloring, start by successively coloring edges, using any method (e.g., Algorithm 9.3.1) until the coloring is blocked or complete. If the set of uncolored edges is empty, then the construction is complete. Otherwise, there is some edge  $e$ , with endpoints  $u$  and  $v$ , that remains uncolored. It will be shown that

by recoloring some edges, the blocked coloring can be transformed into one that can be extended to edge  $e$ . The process can then be repeated until all edges have been colored.

Since the number of colors exceeds  $\delta_{\max}(G)$ , it follows that at each vertex, at least one of the colors is absent. Let  $c_0$  be a color absent at vertex  $u$ , and  $c_1$  a color absent at vertex  $v$ . Color  $c_1$  cannot also be absent at vertex  $u$ , since if it were, edge  $e$  would not have remained uncolored. (For the same reason, color  $c_0$  must occur at vertex  $v$ .) So let  $e_1$  be the  $c_1$ -edge incident on vertex  $u$ , and let  $v_1$  be its other endpoint. Next, let  $c_2$  be a color absent at  $v_1$ . If  $c_2$  is also absent at vertex  $u$ , then the color of edge  $e_1$  can be changed from  $c_1$  to  $c_2$ , thereby permitting the assignment of color  $c_1$  to edge  $e$ , as illustrated in Figure 9.3.13. A missing color  $c$  at a vertex is indicated by placing  $\neg c$  alongside that vertex. Several missing colors may be grouped in braces.

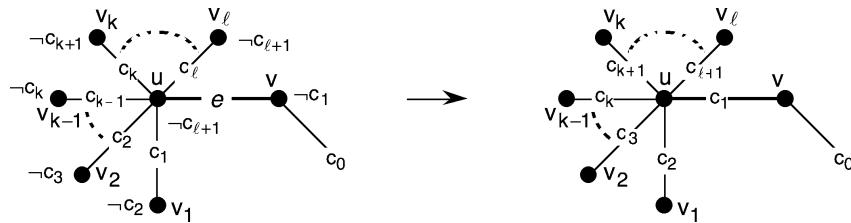


**Figure 9.3.13** Extending an edge-coloring to edge  $e$  by recoloring edge  $e_1$ .

If color  $c_2$  does occur at vertex  $u$ , then let  $e_2$  be the  $c_2$ -edge incident on vertex  $u$ , let  $v_2$  be its other endpoint, and let  $c_3$  be a color absent at vertex  $v_2$ . Continue iteratively in this way, so that at the  $j$ th iteration,  $e_j$  is the  $c_j$ -edge incident on vertex  $u$ ,  $v_j$  is its other endpoint, and  $c_{j+1}$  is the color absent at vertex  $v_j$ . Let  $\ell$  be the smallest  $j$  such that vertex  $v_\ell$  has a missing color  $c_{\ell+1}$  and that  $c_{\ell+1}$  is also absent at vertex  $u$  or is one of the colors in the list  $c_1, \dots, c_\ell$  (such an  $\ell$  exists, since the set of colors is finite).

*Case 1:* Color  $c_{\ell+1}$  is absent at both vertex  $v_\ell$  and vertex  $u$ . *Color Shift.*

Then perform the following *color shift*: for  $j = 1, \dots, \ell$ , change the color of edge  $e_j$  from  $c_j$  to  $c_{j+1}$ . This releases color  $c_1$  from edge  $e_1$ , so that it can be reassigned to edge  $e$ . The color shift is illustrated in Figure 9.3.14. Notice that it maintains a proper edge-coloring, because, by the construction, color  $c_{j+1}$  was absent at both endpoints of edge  $e_j$  before the shift.

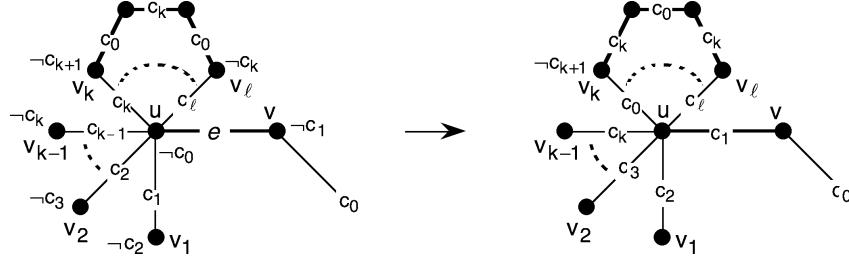


**Figure 9.3.14** Case 1: color shift to free color  $c_1$  for edge  $e$ .

*Case 2:* Color  $c_{\ell+1} = c_k$ , where  $1 \leq k \leq \ell$ . *Swap and Shift.*

Let  $K$  be the Kempe  $c_0-c_k$  edge-chain incident on vertex  $v_\ell$ . By definition,  $K$  includes the  $c_0$ -edge incident on  $v_\ell$ , but there is no  $c_k$ -edge incident on vertex  $v_\ell$  (by definition of  $\ell$ ). By Lemma 9.3.21, Kempe chain  $K$  is a path, and one end of this path is vertex  $v_\ell$ . There are three subcases to consider, according to where the other end of the path

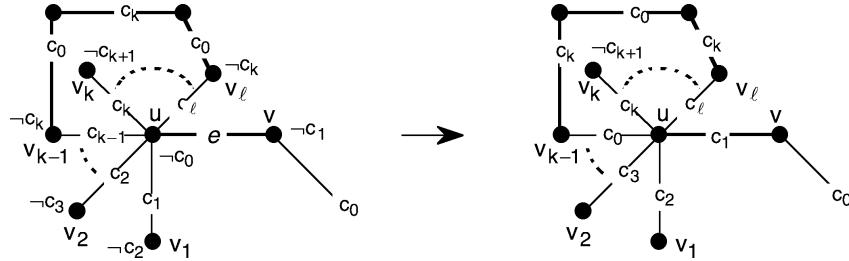
is. In each of the three subcases, the two colors are swapped so that a Case 1 color shift can then be performed.



**Figure 9.3.15** Case 2a: swap and shift.

*Case 2a.* Path  $K$  reaches vertex  $v_k$ .

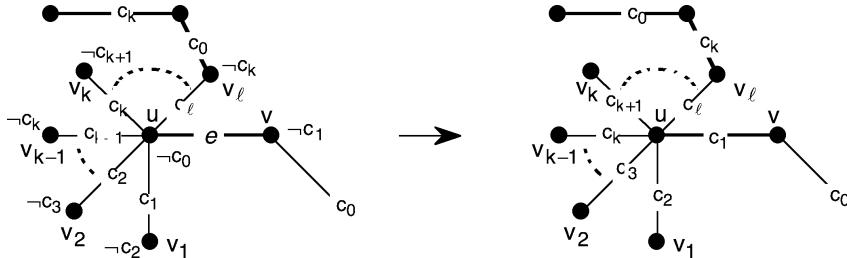
Then swap colors  $c_0$  and  $c_k$  along path  $K$ . As a result of the swap, color  $c_k$  no longer occurs at vertex  $u$ . This configuration permits a Case 1 color shift that releases color  $c_1$  for edge  $e$ . The swap and shift are illustrated in Figure 9.3.15 above.



**Figure 9.3.16** Case 2b: swap, recolor edge  $e_1$ , and then shift.

*Case 2b.* Path  $K$  reaches vertex  $v_{k-1}$ .

Then swap colors  $c_0$  and  $c_k$  along path  $K$ . As a result of the swap, color  $c_0$  no longer occurs at vertex  $v_{k-1}$ . Thus, edge  $e_{k-1}$  can be recolored  $c_0$ , as in Figure 9.3.16 above. A color shift can now be performed to release color  $c_1$  for edge  $e$ .



**Figure 9.3.17** Case 2c: swap and shift.

*Case 2c.* Path  $K$  never reaches vertex  $v_{k-1}$  or vertex  $v_k$ .

Since color  $c_0$  does not occur at vertex  $u$ , and since color  $c_k$  occurs at  $u$  only on the edge from  $v_k$ , it follows that path  $K$  does not reach vertex  $u$ . Then swap colors  $c_0$  and  $c_k$  along path  $K$ , so that color  $c_0$  no longer occurs at vertex  $v_l$ . Now perform a Case 1 color shift that releases color  $c_1$  for edge  $e$ , as in Figure 9.3.17 above.  $\diamond$

**Corollary 9.3.23.** Let  $G$  be a simple graph. Then either  $\chi'(G) = \delta_{\max}(G)$  or  $\chi'(G) = \delta_{\max}(G) + 1$ .

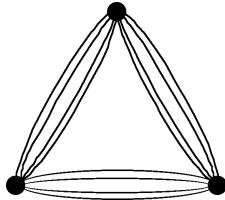
**Proof:** This follows immediately from Vizing's theorem and Corollary 9.3.2.  $\diamond$

**DEFINITION:** **Class 1** is the set of non-empty simple graphs  $G$  such that  $\chi'(G) = \delta_{\max}(G)$ . **Class 2** is the set of simple graphs  $G$  such that  $\chi'(G) = \delta_{\max}(G) + 1$ .

**COMPUTATIONAL NOTE:** Deciding whether a simple graph is in class 1 is an NP-complete problem [Ho81].

**DEFINITION:** The **multiplicity**  $\mu(G)$  of a graph  $G$  is the maximum number of edges joining two vertices of  $G$ .

**Remark:** A more general result of Vizing, beyond simple graphs, which applies to every loopless graph  $G$ , is that  $\delta_{\max}(G) \leq \chi'(G) \leq \delta_{\max}(G) + \mu(G)$ . The edge-chromatic number achieves the upper bound of Vizing's general formula when all three vertex pairs of a “fat triangle”, as illustrated by Figure 9.3.18, are joined by the same multiplicity of edges.



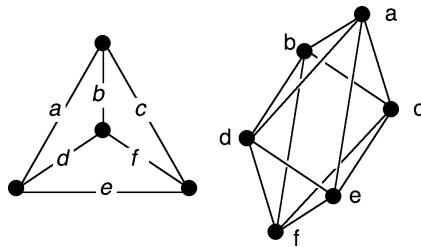
**Figure 9.3.18** A symmetric “fat triangle” requires  $\delta_{\max} + \mu$  edge colors.

### Line Graphs

A line graph can be used to convert an edge-coloring problem into a vertex-coloring problem.

**REVIEW FROM §1.2:** The **line graph** of a graph  $G$  is the graph  $L(G)$  whose vertices correspond bijectively to the edges of  $G$ , and such that two of these vertices are adjacent if and only if their corresponding edges in  $G$  have a vertex in common.

**Example 9.3.7:** The line graph of the complete graph  $K_4$  is the octahedron graph  $O_3$ , as illustrated in Figure 9.3.19.

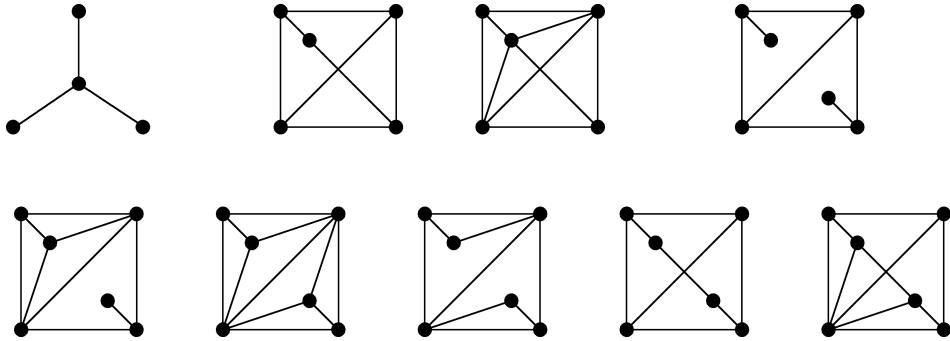


**Figure 9.3.19** The complete graph  $K_4$  and its line graph  $O_3$ .

**Proposition 9.3.24.** *The edge-chromatic number of a graph  $G$  equals the vertex-chromatic number of its line graph  $L(G)$ .*

**Proof:** This follows immediately from the definitions.  $\diamond$

**Remark:** Beineke [Be68] proved that a simple graph  $G$  is a line graph of some simple graph if and only if  $G$  does not contain any of the graphs in Figure 9.3.20 as an induced subgraph. Since it is an NP-complete problem to decide this subgraph problem, much of the theory of edge-colorings has prospered separately from the theory of vertex-colorings.

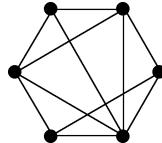


**Figure 9.3.20** The nine forbidden induced subgraphs of line graphs.

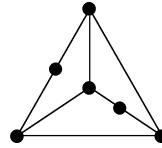
### EXERCISES for Section 9.3

For Exercises 9.3.1 through 9.3.12, assign a minimum edge-coloring and prove that it is a minimum coloring.

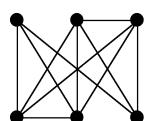
9.3.1



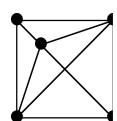
9.3.2<sup>s</sup>



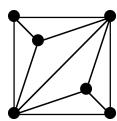
9.3.3



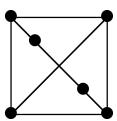
9.3.4



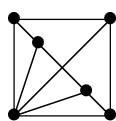
9.3.5



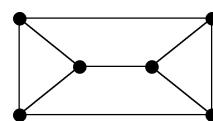
9.3.6



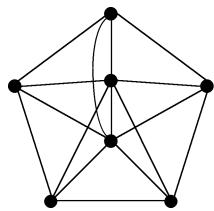
9.3.7



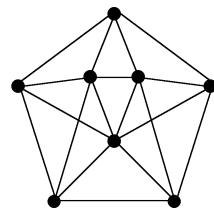
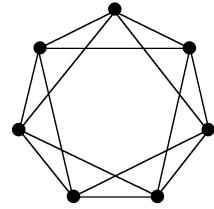
9.3.8



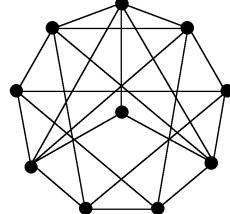
9.3.9



9.3.10

9.3.11<sup>s</sup>

9.3.12



**9.3.13** Prove Proposition 9.3.4. Let  $\alpha'(G)$  be the size of a maximum matching in a graph  $G$ . Then  $\chi'(G) \geq \left\lceil \frac{|E_G|}{\alpha'(G)} \right\rceil$ .

**9.3.14** Prove Proposition 9.3.5. A graph  $G$  has  $\chi'(G) = 1$  if and only if  $\delta_{\max}(G) = 1$ .

**9.3.15** Prove Proposition 9.3.6.  $\chi'(P_n) = 2$ , for  $n \geq 3$ .

**9.3.16<sup>s</sup>** Prove Proposition 9.3.7.  $\chi'(C_{2n}) = 2$ .

**9.3.17** Prove Proposition 9.3.8.  $\chi'(C_{2n+1}) = 3$ .

**9.3.18** Prove Proposition 9.3.9.  $\chi'(T) = \delta_{\max}(T)$ , for any tree  $T$ .

**9.3.19** Prove Proposition 9.3.10.  $\chi'(Q_n) = n$ .

**9.3.20** Prove Proposition 9.3.11.  $\chi'(W_n) = n$ , for  $n \geq 3$ .

**9.3.21<sup>s</sup>** Prove that adding an edge to a graph increases its edge-chromatic number by at most 1.

**9.3.22** Show that  $K_3$  and  $K_{1,3}$  have isomorphic line graphs.

**9.3.23** Show that the edge-complement of  $L(K_5)$  is isomorphic to the Petersen graph.

**9.3.24** Explain how iteratively subdividing graph  $G$  ultimately produces a graph whose edge-chromatic number equals  $\delta_{\max}(G)$ .

**9.3.25<sup>s</sup>** Give an example of a graph  $G$  and an edge  $e \in E_G$  such that subdividing  $e$  causes the edge-chromatic number to increase by 1.

**9.3.26** Let  $G$  be a graph such that  $|E_G| \geq \delta_{\max}(G) \cdot \alpha'(G)$ . Prove that this implies that  $G$  is in class 2.

**9.3.27** Let  $G$  be a regular graph with an odd number of vertices. Prove that  $G$  is in class 2.

**9.3.28<sup>s</sup>** Let  $G$  be a 3-regular graph with a cut-edge. Prove that  $G$  is in class 2.

**9.3.29** Give an example of a graph for which Algorithm 9.3.1 does not produce a minimum edge-coloring.

**9.3.30 [Computer Project]** Implement Algorithms 9.3.1 and 9.3.2 and compare their results on the graphs in Exercises 9.3.1 through 9.3.12.

## 9.4 FACTORIZATION

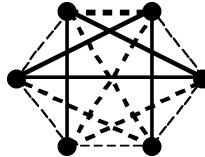
We observe that an edge-coloring of a graph is a partitioning of the edge-set into cells of mutually nonadjacent edges, and we now finish this chapter by considering an additional topic concerned with partitioning an edge-set, called *factorization*. In a factorization, each cell of the partition of the edge-set induces a spanning subgraph. We are able to prove a classical result of W. Tutte. However, since some of the key theorems on factorization are most easily proved with the aid of flows, we defer their proofs until Chapter 13.

### Factors

**DEFINITION:** A **factor** of a graph is a spanning subgraph.

**DEFINITION:** A **factorization** of a graph  $G$  is a set of factors whose edge-sets form a partition of the edge-set  $E_G$ .

**Example 9.4.1:** Figure 9.4.1 shows a factorization of the complete graph  $K_6$  into three spanning paths.

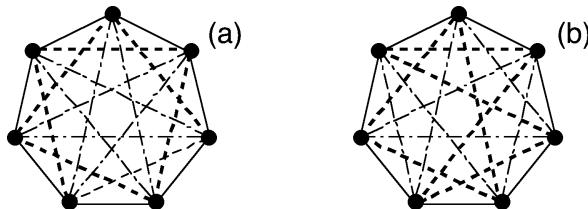


**Figure 9.4.1** Factorization of  $K_6$  into three paths.

**DEFINITION:** A  **$k$ -factor** of a graph  $G$  is a  $k$ -regular factor of  $G$ .

**DEFINITION:** A  **$k$ -factorization** of a graph  $G$  is a factorization into  $k$ -factors.

**Example 9.4.2:** Figure 9.4.2 shows two 2-factorizations of the complete graph  $K_7$ . The factors of factorization (a) are all spanning cycles. In factorization (b), two factors are spanning cycles, but the factor represented by bold dashes is the union of a 3-cycle and a 4-cycle.



**Figure 9.4.2** Two factorizations of  $K_7$  into 2-factors.

**TERMINOLOGY:** A 1-factor of a graph  $G$  is also called a **perfect matching**. Two vertices are said to be **matched** with respect to a 1-factor if they are the endpoints of an edge in the 1-factor.

### Tutte's 1-Factor Theorem

M. Plummer [Pl04] characterizes Tutte's 1-factor Theorem as the most influential theorem in the study of 1-factors. The definitions and lemmas that precede Tutte's theorem help to simplify the proof.

**DEFINITION:** An **odd component of a graph** is a component with an odd number of vertices.

**DEFINITION:** **Tutte's condition** on a graph  $G$  is that for every subset  $S \subset V_G$ , the number of odd components of  $G - S$  does not exceed  $|S|$ .

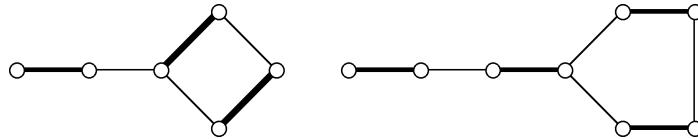
**Lemma 9.4.1.** *Tutte's condition is preserved under edge addition.*

**Proof:** Let  $G$  be a graph that satisfies Tutte's condition, and let  $e$  be an edge added to  $G$  between two nonadjacent vertices. To show that  $G + e$  satisfies Tutte's condition, let  $S \subset V_G$ . If either endpoint of  $e$  lies in  $S$ , then the components of the graph  $(G + e) - S$  are exactly the components of  $G - S$ . If both endpoints of  $e$  lie in the same component of  $G - S$ , then the number of odd components is unchanged. This reduces our consideration to the circumstance that  $e$  joins two components of  $G - S$ . If both endpoints of  $e$  are in even components of  $G - S$ , or if one endpoint of  $e$  is in an odd component and the other in an even component, then the number of odd components stays the same, that is, less than or equal to  $|S|$  (while the number of even components decreases by 1). If both endpoints of  $e$  are in odd components, then the number of odd components decreases by 2 (while the number of even components increases by 1).  $\diamond$

**Lemma 9.4.2.** *Let  $G$  be a connected graph with evenly many vertices and evenly many edges, with one vertex  $v$  of degree 3, one vertex  $u$  of degree 1, and all other vertices of degree 2. Then  $G$  has a 1-factor.*

**Proof:** By Theorem 6.1.1,  $G$  has an Eulerian trail from  $v$  to  $u$ . If we color the edges alternatingly red and blue, starting with red at vertex  $v$ , then the trail terminates with a blue edge at  $u$  (since there are evenly many edges). We observe that at every vertex except  $u$ , both colors are present, because by whatever color edge the trail enters a vertex, it leaves by an edge of the other color. (Thus, there are two red edges at  $v$  and one blue edge.) It follows that the blue edges form a 1-factor.  $\diamond$

**Example 9.4.3:** A graph that satisfies the premises of Lemma 9.4.2 looks something like a *polygon kite*, as illustrated in Figure 9.4.3. In order for the number of vertices in the graph to be even, the number of edges in the tail and the number of edges in the polygon must have the same parity. The dark edges indicate the 1-factor promised by Lemma 9.4.2.



**Figure 9.4.3** Two polygon kites with their 1-factors.

**DEFINITION:** Let  $M$  and  $N$  be spanning subgraphs of the same graph  $G$ . The **symmetric difference**  $M \Delta N$  is the spanning subgraph of  $G$  whose edge-set is  $(E_M \cup E_N) - (E_M \cap E_N)$ .

The following proof of Tutte's 1-Factor Theorem is due to Lovász [Lo75].

**Theorem 9.4.3 [Tutte's 1-Factor Theorem].** A nontrivial graph  $G$  has a 1-factor if and only if for every subset  $S \subset V_G$ , the number of odd components of  $G - S$  does not exceed  $|S|$ .

**Proof:** ( $\Rightarrow$ ) Suppose that  $G$  has a 1-factor. Then in each odd component of  $G - S$ , there is at least one vertex that is not matched to another vertex within that component, and hence, such a vertex is matched to a vertex of  $S$ . It follows that  $|S|$  is at least as large as the number of components of  $G - S$ .

( $\Leftarrow$ ) By way of contradiction, suppose that there exists a graph that satisfies Tutte's condition but has no 1-factor. By adding edges, one at a time, until it is impossible to do so without creating a 1-factor, we obtain a graph  $H$  that still satisfies Tutte's condition (by Lemma 9.4.1) and is edge maximal with respect to having no 1-factor. We now pursue the contradiction that  $H$  does contain a 1-factor.

Let  $W$  be the vertex subset given by

$$W = \{w \in V_H \mid w \text{ is adjacent to every other vertex of } H\}$$

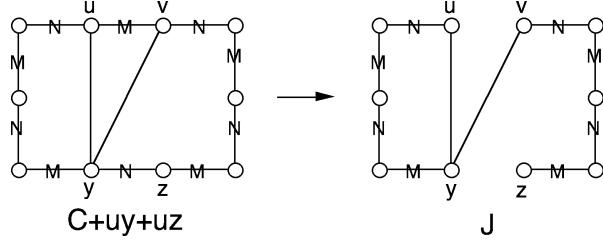
**Case 1.** Suppose that every component of  $H - W$  is a complete graph. Then we may match the vertices of  $H - W$  in pairs, except for one leftover vertex in each odd component of  $H - W$ . Every vertex in  $W$  is adjacent to every one of these leftover vertices (by the definition of set  $W$ ). Moreover, Tutte's condition implies that  $|W|$  is at least as large as the number of odd components of  $H - W$ . Hence, we may pair each of the leftover vertices from the odd components with a vertex of  $W$ . A 1-factor  $H$  will exist if the remaining (unpaired) vertices of  $W$  can be matched into pairs. These remaining vertices are mutually adjacent (again, by the definition of  $W$ ) and hence, can be matched if there are evenly many of them. Since we have previously matched evenly many vertices of  $H$ , it suffices to show that  $|V_H|$  is even. Tutte's condition, with  $S = \emptyset$ , implies that  $H$  has no odd components, which implies that  $H$  has evenly many vertices.

**Case 2.** Suppose that some component of  $H - W$  is not a complete graph. In this case, there is a pair of non-adjacent vertices  $u$  and  $v$  in that component with a common neighbor  $y \notin W$ . Moreover, since  $y \notin W$ , it follows that some vertex  $z$  of  $H$  is not adjacent to  $y$ . By the definition of graph  $H$ , adding an edge creates a 1-factor. In particular, let  $M$  and  $N$  be 1-factors in the graphs  $H + uv$  and  $H + yz$ , respectively. We shall show that the symmetric difference  $M \Delta N$  has a 1-factor that contains neither  $uv$  nor  $yz$ , and is, thus, also a 1-factor of  $H$ .

Since every vertex of  $H$  has degree 1 in  $M$  and degree 1 in  $N$ , it follows that every vertex of  $H$  has degree 0 or 2 in  $M \Delta N$ , which implies that the components of  $M \Delta N$  are cycles and isolated vertices. Moreover, since  $M$  and  $N$  are 1-factors, the  $M$ -edges and  $N$ -edges must alternate on each cycle component, implying that all the cycles are even.

Let  $C$  be the cycle of  $M \Delta N$  that contains the edge  $uv$ . (Of course, the 1-factor  $M$  contains  $uv$ , because we have specified that  $H$  has no 1-factor.) If cycle  $C$  does not also contain the edge  $yz$ , then the union of the set of the  $N$ -edges in cycle  $C$  with the set of  $M$ -edges not in cycle  $C$  forms a 1-factor of  $H$ .

If cycle  $C$  does contain the edge  $yz$ , then we consider the subgraph  $J = (C + uy + vy - uv - yz)$  of the graph  $H$ , now using the fact that we chose vertices  $u$  and  $v$  to have the common neighbor  $y$ . See Figure 9.4.4.



**Figure 9.4.4** Constructing the subgraph  $J$ .

Since  $C$  has evenly many vertices and edges, so does  $J$ . Moreover, in subgraph  $J$ , vertex  $y$  has degree 3, vertex  $z$  has degree 1, and all other vertices have degree 2. It follows from Lemma 9.4.2 that the subgraph  $J$  has a 1-factor. By combining the 1-factor in subgraph  $J$  with the edges of  $M$  that are not part of cycle  $C$ , we obtain a 1-factor of the graph  $H$ .  $\diamond$

### Petersen's 1-Factor Theorem

NOTATION: The number of odd components of a graph  $G$  is denoted  $oc(G)$ .

**Theorem 9.4.4 [Petersen's 1-Factor Theorem].** Every 2-edge-connected 3-regular graph  $G$  has a 1-factor.

**Proof:** By Tutte's 1-Factor Theorem, it suffices to show that  $G$  satisfies Tutte's condition. Let  $S$  be an arbitrary subset of vertices of  $G$ , and let  $k$  be the number of edges between  $S$  and the odd components of  $G - S$ . We first observe that

$$k \leq 3|S|$$

because each vertex of  $S$  has degree 3, since  $G$  is 3-regular. For any odd component  $H$  of  $G - S$ , let  $k_H$  be the number of edges joining  $H$  with  $S$ , so that

$$\sum_{\text{odd components } H} k_H = k$$

The sum of the vertex degrees in  $H$  is  $3|V_H| - k_H$ . By Euler's Theorem on Degree-Sum (§1.1), applied to the graph  $H$ , this sum is even. Since  $|V_H|$  is odd, so is  $3|V_H|$ , which implies that  $k_H$  is odd. Since  $G$  has no cut-edge, we have  $k_H > 1$ , from which it now follows that  $k_H \geq 3$ , and in turn, that

$$3oc(G - S) \leq \sum_{\text{odd components } H} k_H = k$$

Thus,

$$3oc(G - S) \leq 3|S|$$

which implies Tutte's condition  $oc(G - S) \leq |S|$ .  $\diamond$

**Corollary 9.4.5.** Every 2-edge-connected 3-regular graph  $G$  has a 2-factor.

**Proof:** By Petersen's 1-Factor Theorem, the graph  $G$  has a 1-factor. Since  $G$  is 3-regular, the edge-complement of that 1-factor is a 2-factor.  $\diamond$

**Remark:** F. Bäbler ([Bä38]) proved that every  $(r - 1)$ -edge-connected  $r$ -regular graph has a 1-factor, and also that every 2-edge-connected  $(2r + 1)$ -regular graph without self-loops has a 1-factor.

**Remark:** The following two results will be proved in Chapter 13. Their proofs use *Hall's Theorem for Bipartite Graphs*, which is also proved in Chapter 13 with the aid of network flows.

- **König's 1-Factorization Theorem** [Kö16] Every  $r$ -regular bipartite graph  $G$  with  $r > 0$  is 1-factorable.
- **Petersen's 2-Factorization Theorem** [Pe1891]. Every regular graph  $G$  of even degree is 2-factorable.

### EXERCISES for Section 9.4

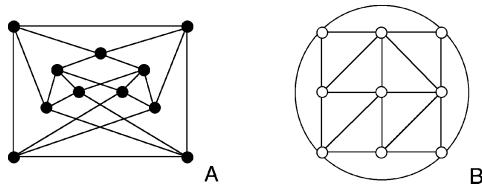
- 9.4.1 Draw a 3-regular simple graph that has no 1-factor or 2-factor.
- 9.4.2 Draw a 5-regular simple graph that has no 1-factor or 2-factor.
- 9.4.3<sup>s</sup> Prove that if two graphs  $G$  and  $H$  each have a  $k$ -factor, then their join has a  $k$ -factor.
- 9.4.4 Draw a connected simple graph that is decomposable into a 2-factor and a 1-factor, but is non-hamiltonian.
- 9.4.5 Prove that if a graph has a  $k$ -factor, then its cartesian product with any other graph has a  $k$ -factor.
- 9.4.6 Prove that the complete graph  $K_{2r}$  is 1-factorable. Hint: use induction on  $r$ .
- 9.4.7 Prove that an  $r$ -regular bipartite graph  $G$  can be decomposed into  $k$ -factors if and only if  $k$  divides  $r$ .
- 9.4.8<sup>s</sup> Describe a 2-factorable graph whose edge-complement contains no 2-factor.
- 9.4.9 Suppose that two graphs with a 1-factor are amalgamated across an edge. Does the resulting graph necessarily have a 1-factor?
- 9.4.10 Draw two graphs that have no 1-factor, but whose cartesian product does have a 1-factor.
- 9.4.11<sup>s</sup> Draw two graphs that have no 2-factor, but whose cartesian product does have a 2-factor.

## 9.5 SUPPLEMENTARY EXERCISES

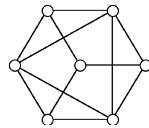
*Exercises 9.5.1 through 9.5.6 concern the set  $S$  of connected graphs with 8 vertices and 17 edges.*

- 9.5.1 Prove that no graph in  $S$  is 2-chromatic.
- 9.5.2 Draw a 3-chromatic graph from  $S$ , and prove it is 3-chromatic.
- 9.5.3 Draw a 4-chromatic graph from  $S$ , and prove it is 4-chromatic.
- 9.5.4 Draw a 5-chromatic graph from  $S$ , and prove it is 5-chromatic.
- 9.5.5 Draw a 6-chromatic graph from  $S$ , and prove it is 6-chromatic.

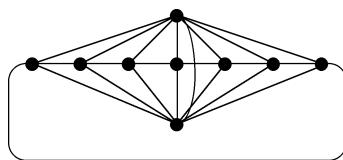
- 9.5.6 Prove that no graph in  $S$  is 7-chromatic.
- 9.5.7 Prove that for any two graphs  $G$  and  $H$ ,  $\chi(G + H) = \chi(G) + \chi(H)$ .
- 9.5.8 Draw a simple 6-vertex graph  $G$  such that  $\chi(G) + \chi(\overline{G}) = 5$ , where  $\overline{G}$  is the edge-complement of  $G$ .
- 9.5.9 Draw a simple 6-vertex graph  $G$  such that  $\chi(G) + \chi(\overline{G}) = 7$ .
- 9.5.10 Construct a non-complete graph of chromatic number 6 that is chromatically critical.
- 9.5.11 Draw a minimum vertex-coloring and a minimum edge-coloring of the graph  $A$  in Figure 9.5.1 and prove their minimality.
- 9.5.12 Draw a minimum vertex-coloring and a minimum edge-coloring of the graph  $B$  in Figure 9.5.1 and prove their minimality.

**Figure 9.5.1**

- 9.5.13 Calculate the vertex chromatic number and edge chromatic number of the graph below. Is it vertex chromatically critical? Is it edge chromatically critical?

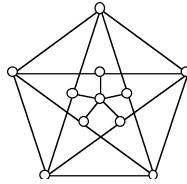
**Figure 9.5.2**

- 9.5.14 Prove that the minimum chromatic number among all 4-regular 9-vertex graphs is three. (Hint: first prove that chromatic number two is impossible, and then draw a 3-chromatic 4-regular 9-vertex graph.)
- 9.5.15 Calculate the maximum possible number of edges of a simple 3-colorable planar graph on 12 vertices. Be sure to prove that your number is achievable.
- 9.5.16 Calculate the chromatic number of the circulant graph  $\text{circ}(13 : 1, 5)$ .
- 9.5.17 a. Prove that the graph below has chromatic number 5. b. Mark two edges whose removal would make the graph 3-chromatic.

**Figure 9.5.3**

- 9.5.18 Calculate the chromatic number of the graph  $C_3 \times C_3 \times C_3 \times C_3 \times C_3$ .

**9.5.19** Give a proper 4-coloring of **Mycielski's graph**, shown below. Why must at least 3 different colors be used on the outer cycle in a proper coloring? Why must at least 3 different colors be used on the five hollow vertices? (This implies that a fourth color is needed for the central vertex.)



**Figure 9.5.4** Mycielski's graph.

**9.5.20** Calculate the chromatic number of the product graph  $K_4 \times K_4$ .

**9.5.21** Among all graphs with 8 vertices and independence number 4, determine the largest possible number of edges and draw such a graph.

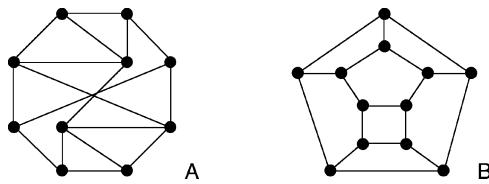
**9.5.22** Among all graphs with 8 vertices and clique number 4, draw a graph with the largest possible number of edges. Write the number.

**9.5.23** Calculate the independence number of  $C_5 \times C_5$ .

**9.5.24** Calculate the clique number of the circulant graph  $\text{circ}(9 : 1, 3, 4)$ .

**9.5.25** Calculate the independence number and chromatic number of the graph  $A$  in Figure 9.5.5.

**9.5.26** Calculate the independence number and chromatic number of the graph  $B$  in Figure 9.5.5.



**Figure 9.5.5**

**9.5.27** Prove that every hamiltonian 3-regular graph is 3-edge-colorable.

**9.5.28** Draw as many copies of the cube  $Q_3$  as needed, each with a different 1-factor, to give a complete repetition-free list of all the possible 1-factors.

## GLOSSARY

**adjacent edges:** different edges with at least one endpoint in common.

**block** of a loopless graph: a maximal connected subgraph  $H$  such that no vertex of  $H$  is a cut-vertex of  $H$ .

**$k$ -chromatic graph:** a graph whose vertex chromatic number is  $k$ .

**chromatic incidence** of an edge-coloring  $f$  at a vertex  $v$ : the number of different edge-colors present at  $v$ , denoted by  $\text{ecr}_v(f)$ .

**chromatic number:**

- , **of a graph  $G$ :** the minimum number of different colors required for a *proper vertex-coloring* of  $G$ , usually denoted by  $\chi(G)$ .
- , **of a map:** the minimum number of colors needed for a *proper map-coloring*.
- , **of a surface  $S$ :** the maximum of the chromatic numbers of the maps on  $S$ , or equivalently, of the graphs (without self-loops) that can be imbedded in  $S$ , denoted by  $\text{chr}(S)$ .

**chromatically  $k$ -critical graph:** a graph whose chromatic number would decrease if any edge were deleted.

**class one:** the class of graphs containing every non-empty graph whose edge-chromatic number equals its maximum degree.

**class two:** the class of graphs containing every graph whose edge-chromatic number is one more than its maximum degree.

**clique** in a graph: a maximal subset of mutually adjacent vertices.

**clique number** of a graph: the number of vertices in the largest clique.

**color class** in a vertex-coloring of a graph  $G$ : a subset of  $V_G$  containing all the vertices of some color.

**$k$ -colorable graph:** a graph that has a proper vertex  $k$ -coloring.

**coloring of a graph:** usually refers to a vertex-coloring.

**$k$ -coloring of a graph:** a vertex-coloring that uses exactly  $k$  different colors.

**$k$ -coloring of a map:** a map-coloring that uses exactly  $k$  different colors.

**colors** of vertices or faces: a set, usually of integers  $1, 2, \dots$ , to be assigned to the vertices of a graph or the regions of a map.

**complete set of obstructions to  $k$ -chromaticity:** a set  $\{G_j\}$  of chromatically  $(k+1)$ -critical graphs such that every  $(k+1)$ -chromatic graph contains at least one graph  $G_j$  as a subgraph.

**domination number  $\text{dom}(G)$  of a graph  $G$ :** the cardinality of a minimum set  $S$  of vertices such that every vertex of  $G$  is either in  $S$  or a neighbor of a vertex in  $S$ .

**edge  $k$ -chromatic graph  $G$ :** a graph with  $\chi'(G) = k$ .

**edge-chromatic number  $\chi'(G)$  of a graph:** the minimum number of different colors required for a proper edge-coloring of a graph  $G$ .

**edge  $k$ -colorable graph:** a graph that has a proper edge  $k$ -coloring.

**edge-coloring of a graph:** an assignment to its edges of “colors” from any set.

**edge  $k$ -coloring of a graph:** an edge-coloring that uses exactly  $k$  different colors.

- , **blocked partial:** the circumstance in which a proper subset of edges has a proper edge  $k$ -coloring, and every uncolored edge is adjacent to at least one edge of each of the  $k$  colors.

**edge-independence number  $\text{ind}_E(G)$  of a graph  $G$ :** the maximum cardinality of an independent set of edges.

**eulerian graph:** a graph that has an eulerian tour.

**eulerian tour** in a graph: a closed trail that contains every edge of that graph.

**even wheel:** a wheel graph  $W_n = K_1 + C_n$  such that  $n$  is even.

**factor** of a graph: a spanning subgraph.

**$k$ -factor** of a graph: a spanning subgraph of that is regular of degree  $k$ .

**factorization** of a graph  $G$ : a set of factors whose edge-sets form a partition of the edge-set  $E_G$ .

**$k$ -factorization** of a graph  $G$ : a factorization of  $G$  into  $k$ -factors.

**independence number** of a graph  $G$ : the maximum cardinality of an independent set of vertices, denoted by  $\alpha(G)$ .

**independent set of edges**: a set of mutually non-adjacent edges.

**independent set of vertices**: a set of mutually non-adjacent vertices.

**join of two graphs  $G$  and  $H$** : the graph  $G+H$  obtained from the graph union  $G \cup H$  by adding an edge between each vertex of  $G$  and each vertex of  $H$ .

**Kempe  $i$ - $j$  chain for a vertex-coloring of a graph**: a component of the subgraph induced on the set of all vertices colored either  $i$  or  $j$ .

**Kempe  $i$ - $j$  edge-chain in an edge-colored graph**: a component of the subgraph induced on all the  $i$ -colored and  $j$ -colored edges.

**leaf block** of a graph  $G$ : a block that contains exactly one cut-vertex of  $G$ .

**line graph** of a graph  $G$ : the graph  $L(G)$  whose vertices are the edges of  $G$ , such that edges with a common endpoint in  $G$  are adjacent in  $L(G)$ .

**map** on a surface: an imbedding of a graph on that surface.

**map-coloring** for an imbedding of a graph: a function from the set of faces to a set whose elements are regarded as *colors*.

**matching in a graph  $G$** : a subset of edges of  $G$  that are mutually non-adjacent.

—, **maximum**: a matching with the maximum number of edges.

**multipartite graph**: a loopless graph whose vertices can be partitioned into  $k$  independent sets, which are sometimes called the **partite sets**, is said to be  $k$ -partite.

**multiplicity  $\mu(G)$  of a graph**: the maximum number of edges joining two vertices.

**neighbor of an edge  $e$** : another edge that shares one or both of its endpoints with edge  $e$ .

**obstruction to  $k$ -chromaticity**: a graph whose presence as a subgraph forces the chromatic number to exceed  $k$ .

**odd component** of a graph: a component with an odd number of vertices.

**odd wheel**: a wheel graph  $W_n = K_1 + C_n$  such that  $n$  is odd.

**optimum  $k$ -edge coloring**: a  $k$ -edge-coloring with the highest total chromatic incidence among all  $k$ -edge-colorings.

**partite sets**: see *multipartite graph*.

**perfect graph**: a graph  $G$  such that every induced subgraph  $H$  has its chromatic number  $\chi(H)$  equal to its clique number  $\omega(H)$ .

**proper edge-coloring of a graph**: an edge-coloring such that if two edges have a common endpoint, then they are assigned two different colors.

**proper map-coloring**: a coloring such that if two regions meet at an edge, then they are colored differently.

**proper vertex-coloring:** a coloring such that the endpoints of each edge are assigned two different colors.

**total chromatic incidence of an edge-coloring on a graph:** the sum of the chromatic incidences at the vertices.

**Tutte's condition** on a graph  $G$ : the condition that for every subset  $S \subset V_G$ , the number of odd components of  $G - S$  does not exceed  $|S|$ .

**vertex  $k$ -colorable** graph: a graph that has a proper vertex  $k$ -coloring.

**vertex-coloring:** a function from the vertex-set of a graph to a set whose members are called *colors*.

**vertex- $k$ -coloring:** a vertex-coloring that uses exactly  $k$  different colors.

**(vertex)  $k$ -chromatic** graph: a graph  $G$  with  $\chi(G) = k$ .

**(vertex) chromatic number** of a graph  $G$ , denoted  $\chi(G)$ : the minimum number of different colors required for a proper vertex-coloring of  $G$ .

# Chapter 10

---

## MEASUREMENT AND MAPPINGS

- 10.1 Distance in Graphs**
  - 10.2 Domination in Graphs**
  - 10.3 Bandwidth**
  - 10.4 Intersection Graphs**
  - 10.5 Linear Graph Mappings**
  - 10.6 Modeling Network Emulation**
- 

### INTRODUCTION

This chapter provides the fundamental concepts and some basic results in graph-theoretic topics motivated by measurement considerations foreshadowed in earlier chapters. Distance-related invariants, including radius, diameter, and girth are treated systematically in the first section, domination in the second, and *bandwidth*, from the perspective of vertex labelings, in the third. The fourth section focuses on intersection graphs, which were introduced in §1.2.

The final two sections discuss *linear* and *semilinear* graph mappings, both of which are generalizations of isomorphism. In §10.6, we see how these graph mappings provide a model, of independent mathematical interest, for a computer software process called *emulation*, under which distributed algorithms for one parallel architecture are ported to another architecture. Emulation involves additional forms of measurement, called *load*, *congestion*, and *dilation*.

## 10.1 DISTANCE IN GRAPHS

REVIEW FROM §1.4: The **distance**  $d(s, t)$  from a vertex  $s$  to a vertex  $t$  in a graph is the length of a shortest  $s$ - $t$  path if one exists; otherwise,  $d(s, t) = \infty$ .

NOTATION: When there is more than one graph under consideration, we sometimes use a subscript (as in  $d_G(s, t)$ ) to indicate the graph in which the distance is taken.

**Proposition 10.1.1.** *Let  $G$  be a connected graph. The distance function  $d$  is a **metric** on  $V_G$ . That is, it satisfies the following four conditions:*

- $d(x, y) \geq 0$  for all  $x, y \in V_G$ .
- $d(x, y) = 0$  if and only if  $x = y$ .
- $d(x, y) = d(y, x)$  for all  $x, y \in V_G$ . (**symmetry property**)
- $d(x, y) + d(y, z) \geq d(x, z)$  for all  $x, y, z \in V_G$ . (**triangle inequality**)

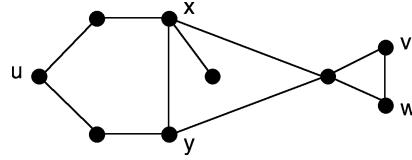
**Proof:** The first three conditions follow immediately from the definition of distance. The triangle inequality holds because the concatenation of a shortest  $x$ - $y$  path with a shortest  $y$ - $z$  path forms an  $x$ - $z$  walk whose length is certainly no smaller than a shortest  $x$ - $z$  walk.  $\diamond$

### Eccentricity, Diameter, and Radius

REVIEW FROM §1.4, §2.3, AND §2.4:

- 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,  $ecc(v) = \max_{x \in V_G} \{d(v, x)\}$ .
- 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,  $diam(G) = \max_{x \in V_G} \{ecc(x)\} = \max_{x, y \in V_G} \{d(x, y)\}$ .
- The **radius** of a graph  $G$ , denoted  $rad(G)$ , is the minimum of the vertex eccentricities. That is,  $rad(G) = \min_{x \in V_G} \{ecc(x)\}$ .
- A **central vertex**  $v$  of a graph  $G$  is a vertex with minimum eccentricity, i.e., such that  $ecc(v) = rad(G)$ .
- The **center of a graph**  $G$ , denoted  $Z(G)$ , is the subgraph induced on the set of central vertices of  $G$ .
- For every graph  $G$ , there is a connected graph  $H$  such that  $G = Z(H)$ .
- A **block** of a graph  $G$  is a maximal connected subgraph that contains no cut-vertices of  $G$ .

**Example 10.1.1:** The graph  $G$  in Figure 10.1.1 has diameter  $diam(G) = 4$  and radius  $rad(G) = 2$ . Its center  $Z(G)$  is isomorphic to  $K_2$  (induced on the vertex subset  $\{x, y\}$ ).



**Figure 10.1.1** A graph  $G$  with  $\text{diam}(G) = 4$ ,  $\text{rad}(G) = 2$ , and  $Z(G) \cong K_2$ .

**Proposition 10.1.2.** Let  $G$  be a connected graph. Then

$$\text{rad}(G) \leq \text{diam}(G) \leq 2 \cdot \text{rad}(G)$$

**Proof:** The first inequality is an immediate consequence of the definitions of radius and diameter. The second follows from the triangle inequality.  $\diamond$

**Remark:** Both inequalities in Proposition 10.1.2 are *tight* (or *best possible*). That is, there are graphs for which the inequality is strict, and there are graphs for which equality holds. (See Exercises.)

**Proposition 10.1.3([HaNo53]).** The center of every connected graph  $G$  lies within a single block.

**Proof:** Suppose, to the contrary, that the center  $Z(G)$  has a pair of vertices  $u$  and  $w$  that do not lie in the same block of  $G$ . Then  $G$  has a cut-vertex  $v$ , distinct from  $u$  and  $w$ , such that  $u$  and  $w$  lie in different components of  $G - v$ , say  $C_u$  and  $C_w$ , respectively. Let  $x$  be a vertex of  $G$  that is farthest from cut-vertex  $v$  (i.e.,  $d(x, v) = \text{ecc}(v)$ ). If  $x$  lies in component  $C_u$ , then  $v$  is on every  $xw$ -path (else  $x$  would lie in  $C_w$ ). It follows that  $d(x, w) > d(x, v) = \text{ecc}(v)$ , which contradicts the fact that  $w$  is a central vertex. A similar contradiction arises if  $x$  lies in component  $C_w$ .  $\diamond$

### Periphery

The *periphery* of a graph is the “opposite” of the center.

**DEFINITION:** A **peripheral vertex**  $v$  of a graph  $G$  is a vertex with maximum eccentricity, i.e.,  $\text{ecc}(v) = \text{diam}(G)$ .

**DEFINITION:** The **periphery of a graph**  $G$ , denoted  $\text{per}(G)$ , is the subgraph induced on the set of peripheral vertices of  $G$ .

**Example 10.1.2:** For the graph  $G$  in Figure 10.1.1, the peripheral vertices are  $u$ ,  $v$ , and  $w$ , and  $\text{per}(G) \cong K_1 \cup K_2$ .

In §2.4, we established that every graph is the center of some graph. The following result characterizes when a graph is the periphery of some graph.

**Proposition 10.1.4([BySy83]).** A non-trivial graph  $G$  is the periphery of some connected graph if and only if  $G$  is complete or no vertex of  $G$  has eccentricity 1.

**Proof:** ( $\Leftarrow$ ) If  $G$  is complete, then  $\text{per}(G) = G$ , i.e.,  $G$  is the periphery of itself. Alternatively, if no vertex of  $G$  has eccentricity 1, then let  $G^* = G + v$  be the join (see §2.4) of  $G$  with a new vertex  $v$ . Since  $\text{ecc}_G(x) > 1$  for every  $x \in V_G$ , it follows that  $\text{ecc}_{G^*}(x) = 2$ . Moreover,  $\text{ecc}_{G^*}(v) = 1$ , and hence,  $\text{per}(G^*) = G$ .

$(\Rightarrow)$  Suppose that  $G = \text{per}(G^*)$ . If  $\text{diam}(G^*) = 1$ , then every vertex in  $G$  has eccentricity 1. Suppose that  $\text{diam}(G^*) \geq 2$ , and let  $x$  be any vertex in  $G$ . Since  $x$  is in  $\text{per}(G^*)$ , there exists  $y$  in  $G^*$  such that  $d_{G^*}(x, y) = \text{diam}(G^*) \geq 2$ . Then  $y$  is also a vertex in  $\text{per}(G^*) = G$ , and, hence,  $d_G(x, y) \geq 2$ . Thus,  $\text{ecc}_G(x) \geq 2$ .  $\diamond$

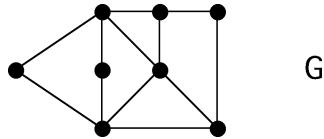
### Girth and Circumference

*Girth* and *circumference* are opposite measures.

REVIEW FROM §1.5: The **girth** of a graph  $G$  with at least one cycle is the length of a shortest cycle in  $G$  and is denoted  $\text{girth}(G)$ . The girth of  $G$  is undefined if  $G$  is acyclic.

DEFINITION: The **circumference** of a graph  $G$  with at least one cycle is the length of a longest cycle in  $G$  and is denoted  $\text{circum}(G)$ . The circumference of  $G$  is undefined if  $G$  is acyclic.

**Example 10.1.3:** The graph in Figure 10.1.2 has girth 3 and circumference 7.



**Figure 10.1.2** A graph  $G$  with  $\text{girth}(G) = 3$  and  $\text{circum}(G) = 7$ .

### Convexity

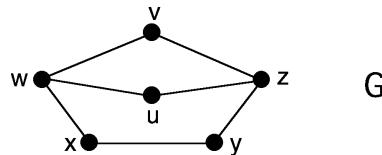
DEFINITION: A **geodesic** between vertices  $u$  and  $v$  is a  $u$ - $v$  path of minimum length.

DEFINITION: For any two vertices  $u, v$  of a connected graph  $G$ , the **closed interval**  $I[u, v]$  is the union of the sets of vertices of  $G$  that lie on  $u$ - $v$  geodesics.

DEFINITION: For any vertex subset  $S$  of a connected graph, the **closed interval**  $I[S]$  is the union

$$\bigcup_{u, v \in S} I[u, v]$$

**Example 10.1.4:** For the graph  $G$  shown in Figure 10.1.3,  $I[u, v] = \{u, v, w, z\}$  and  $I(\{v, x, y\}) = \{v, x, y, w, z\}$ .



**Figure 10.1.3**

DEFINITION: A subset  $S$  of vertices of a graph is **convex** if  $I[S] = S$ .

DEFINITION: The **convex hull** of a subset  $S$  of vertices of a graph is the smallest convex set that contains  $S$ .

**Example 10.1.4, continued:** For the graph  $G$  in Figure 10.1.3, the convex hull of  $\{u, v\}$  is  $\{u, v, w, z\}$ , and the convex hull of  $\{v, x, y\}$  is  $V_G$  (since  $\{v, x, y, w, z\}$  is not convex).

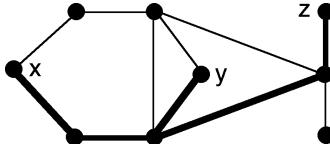
### Steiner Distance

The distance between two vertices in a graph is the number of edges in a shortest path between them. *Steiner distance* generalizes this concept to subsets of two or more vertices in a graph.

**DEFINITION:** Let  $U$  be a subset of vertices in a connected graph  $G$ .

- A **Steiner tree** for  $U$  is a smallest tree subgraph of  $G$  that contains all the vertices of  $U$ .
- The **Steiner distance** of  $U$ , denoted  $sd(U)$ , is the number of edges in a Steiner tree for  $U$ .

**Example 10.1.5:** In the graph of Figure 10.1.4, a Steiner tree for the vertex subset  $U = \{x, y, z\}$  is shown with edges in bold. Thus,  $sd(U) = 5$ .



**Figure 10.1.4** A Steiner tree for  $\{x, y, z\}$ .

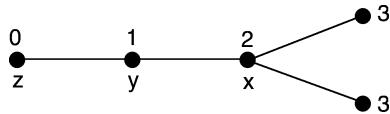
**Remark:** In §4.3, the *Steiner-tree problem* was introduced for weighted graphs. A tree that solves the Steiner-tree problem in a graph whose edges all have equal weight coincides with the definition of Steiner tree given above, for unweighted graphs.

### Total Distance and Medians

**DEFINITION:** The **total distance**  $td(v)$  of a vertex  $v$  in a connected graph  $G$  is given by

$$td(v) = \sum_{w \in V_G} d(v, w)$$

**Example 10.1.6:** In Figure 10.1.5, each vertex in the graph is labeled with its distance from the vertex  $z$ . Summing these distances yields  $td(z) = 9$ .



**Figure 10.1.5** Total distance  $td(z) = 9$ .

**Proposition 10.1.5.** Let  $v$  be any vertex of a connected graph with  $n$  vertices and  $m$  edges. Then the inequality

$$n - 1 \leq td(v) \leq \frac{(n - 1)(n + 2)}{2} - m$$

provides tight upper and lower bounds on total distance.

**Proof:** The lower bound is easily achieved by any graph having a vertex that is adjacent to every other vertex.

To establish the upper bound, we use induction on the number of edges,  $m$ . Observe that  $m \geq n - 1$ , since  $G$  is connected. If  $m = n - 1$ , then  $G$  is a tree. For any vertex  $v$ , let  $d_i$  be the number of vertices at distance  $i$  from  $v$ . Then

$$td(v) = \sum_i id_i \quad \text{and} \quad \sum_i d_i = n - 1$$

Clearly, if  $d_i = 0$ , then  $d_{i+1} = 0$ . It follows that the sum  $\sum_i id_i$  is maximum when  $d_i = 1$  for all nonzero  $d_i$  (i.e., for  $1 \leq i \leq n - 1$ ). Thus,

$$td(v) = \sum_{i=1}^{n-1} id_i = \sum_{i=1}^{n-1} i = \frac{n(n - 1)}{2} = \frac{(n - 1)(n + 2)}{2} - (n - 1)$$

which establishes the base of the induction.

Next assume for some  $m \geq n - 1$  that the upper bound holds for any  $n$ -vertex connected graph with  $m$  edges, and let  $v$  be a vertex in an  $n$ -vertex graph  $G$  with  $m + 1$  edges. Since  $G$  contains at least  $n$  edges, it must have at least one cycle. Among all cycle vertices in  $G$ , let  $x$  be one that is closest to  $v$ , and let  $e = xy$  be a cycle edge. The two possibilities, depending on whether  $v$  is a cycle vertex, are shown in Figure 10.1.6.

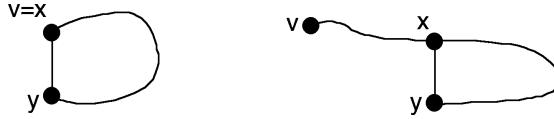


Figure 10.1.6

In either case, the choice of  $x$  implies that  $d_{G-e}(v, y) \geq d_G(v, y) + 1$ , and, hence,  $td_G(v) \leq td_{G-e}(v) - 1$ . Then, applying the induction hypothesis to the graph  $G - e$ , we have

$$td_G(v) \leq td_{G-e}(v) - 1 \leq \frac{(n - 1)(n + 2)}{2} - m - 1 = \frac{(n - 1)(n + 2)}{2} - (m + 1)$$

which establishes the upper bound.

To show that the upper bound can be achieved, first observe that the vertex at either end of the  $n$ -vertex path graph  $P_n$  has total distance  $1 + 2 + \dots + n - 1 = \frac{(n-1)n}{2} = \frac{(n-1)(n+2)}{2} - (n - 1)$ , and, thus, achieves the upper bound when  $m = n - 1$ .

Next, consider a graph  $G$  formed by joining one end of a path of length  $(n - c)$  to one of the vertices of the complete graph  $K_c$ . Then  $G$  has  $n$  vertices and  $m = \binom{c}{2} + n - c = \frac{c(c-3)+2n}{2}$  edges, and it is straightforward to verify that its univalent vertex  $v$  has total distance

$$td(v) = 1 + 2 + \dots + (n - c) + (c - 1)(n - c + 1) = \frac{(n - 1)(n + 2)}{2} - m$$

Thus, the upper bound is achieved by such a graph whenever there is a positive integer  $c$  that satisfies  $m = \frac{c(c-3)+2n}{2}$ . When there is no such  $c$ , the upper bound is achieved by a graph formed by joining two or more paths of the same length to different vertices of some complete graph. For the details, see, for example, [BuHa90].  $\diamond$

**DEFINITION:** A **median vertex** in a connected graph is a vertex whose total distance is minimum.

**DEFINITION:** The **median subgraph** of a connected graph  $G$  is the subgraph  $M(G)$  induced on the median vertices of  $G$ .

**Example 10.1.6, continued:** Vertex  $x$  of the graph in Figure 10.1.5 has total distance 5 and is the only median vertex. Observe that  $y$  is a central vertex but is not a median vertex, since  $td(y) = 6$ .

**Proposition 10.1.6.** *Every graph is the median subgraph of some graph.*

**Proof:** Let  $G$  be a graph with  $V_G = \{v_1, v_2, \dots, v_n\}$ . Form a supergraph  $H$  as follows: for each  $i = 1, 2, \dots, n$ , join a new vertex  $w_i$  to  $v_i$  and to each vertex of  $G$  not adjacent to  $v_i$ . It is straightforward to show that  $M(H) = G$  (see Exercises).  $\diamond$

### EXERCISES for Section 10.1

**NOTE:** Additional exercises on distance, radius, and girth appear in §1.4 and §1.5.

*In Exercises 10.1.1 through 10.1.8, determine the circumference of the given graph.*

- 10.1.1<sup>s</sup> Complete bipartite graph  $K_{3,7}$ .
- 10.1.2 Complete bipartite graph  $K_{m,n}$ ,  $m \geq n \geq 3$ .
- 10.1.3 Complete graph  $K_n$ .
- 10.1.4 Hypercube graph  $Q_3$ . Can you generalize to  $Q_n$ ?
- 10.1.5 Circular ladder graph  $CL_6$ . Can you generalize to  $CL_n$ ?
- 10.1.6 The Petersen graph.
- 10.1.7<sup>s</sup> Circulant graph  $circ(n : 2)$  (see §1.2).
- 10.1.8 Circulant graph  $circ(n : m)$ .

*In Exercises 10.1.9 and 10.1.10, find the periphery of the given graph.*

- 10.1.9<sup>s</sup> The graph in Figure 10.1.2.
- 10.1.10 The Petersen graph.
- 10.1.11 Prove or disprove: If a graph  $G$  is vertex-transitive, then  $per(G) = G$ .

*In Exercises 10.1.12 through 10.1.18, determine the median subgraph and the minimum total distance for the given graph.*

- 10.1.12<sup>s</sup> Path graph  $P_n$ ,  $n \geq 3$ .
- 10.1.13 Cycle graph  $C_n$ ,  $n \geq 4$ .

- 10.1.14 Complete graph  $K_n$ ,  $n \geq 3$ .
- 10.1.15 Complete bipartite graph  $K_{n,n}$ ,  $n \geq 3$ .
- 10.1.16 Petersen graph.
- 10.1.17 Hypercube graph  $Q_3$ .
- 10.1.18 Circular ladder graph  $CL_n$ ,  $n \geq 4$  (§1.2).
- 10.1.19 Both inequalities in Proposition 10.1.2 are tight. Each of the following parts confirms a different aspect of this assertion. Find a family of graphs for which
- the first inequality is always strict.
  - the first inequality is always an equality.
  - the second inequality is always strict.
  - the second inequality is always an equality.
- 10.1.20 Write out a complete proof of Proposition 10.1.2.
- 10.1.21<sup>s</sup> Construct a graph whose diameter, girth, and circumference are all equal.
- 10.1.22 Let  $G$  be a graph with at least two components. Prove that the edge-complement graph  $\overline{G}$  has diameter  $diam(\overline{G}) \leq 2$ .
- 10.1.23 Let  $T$  be a tree. Suppose, starting at any vertex  $v$  of  $T$ , you execute a breadth-first search, and the last vertex discovered is  $w$ . Then, starting at  $w$ , you execute a breadth-first search, and the last vertex discovered is  $z$ . Prove that  $diam(T) = d(w, z)$ .
- 10.1.24<sup>s</sup> Let  $x$  and  $y$  be any two adjacent vertices in a connected graph. Prove that their eccentricities differ by at most 1.
- 10.1.25 Let  $G$  be a connected graph with  $rad(G) \geq 3$ . Prove  $rad(\overline{G}) \leq 2$ .
- 10.1.26 Let  $r$  and  $d$  be positive integers such that  $r \leq d \leq 2r$ . Construct a graph with radius  $r$  and diameter  $d$ . (Hint: Try a graph with exactly one cycle.)
- 10.1.27 Let vertices  $u$  and  $v$  be two different neighbors of a vertex  $w$  in a tree. Prove that  $2ecc(w) \leq ecc(u) + ecc(v)$ .
- 10.1.28 Complete the proof of Proposition 10.1.6, by showing that  $G$  is the median subgraph of  $H$ .

## 10.2 DOMINATION IN GRAPHS

**TERMINOLOGY:** A vertex is said to **dominate** itself and each of its neighbors.

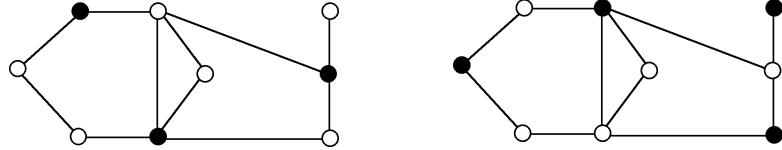
The applicability of *domination theory* to such fields as network design and analysis, linear algebra, and optimization has attracted researchers from a broad range of disciplines. The origins are ascribed to C. Berge [Be62] and O. Ore [Or62]. For a quick survey of domination, see [HaHe04]. For a comprehensive treatment, see [HaHeSl98].

**DEFINITION:** Let  $G$  be a graph and let  $D \subseteq V_G$ . A vertex subset  $D$  **dominates** a graph  $G$  (or is a **dominating set**) if every vertex of  $G$  is in  $D$  or is adjacent to at least one vertex in  $D$ .

**DEFINITION:** A **minimal dominating set** of a graph  $G$  is a dominating set such that every proper subset is non-dominating.

**DEFINITION:** The **domination number** of a graph  $G$ , denoted  $\text{dom}(G)$  (elsewhere, often  $\gamma(G)$ ), is the cardinality of a minimum dominating set of  $G$ .

**Example 10.2.1:** Figure 10.2.1 shows a graph with minimal dominating sets (the solid vertices) of two different cardinalities. It is straightforward to show that the dominating set on the left is a minimum one (i.e., there are no 2-vertex dominating sets). Thus, the dominating number of the graph is 3.



**Figure 10.2.1** Two minimal dominating sets for a graph.

**Example 10.2.2:** For the 5-vertex path graph  $P_5$ , we have  $\text{dom}(P_5) = 2$ . The second and fourth vertices on the path form the unique minimum dominating set.

### Three Applications of Domination

Here are three applications in which the problem of finding a minimum dominating set occurs naturally. It also has independent mathematical interest for many researchers.

**Application 10.2.1** [Be73] *Surveillance:* The minimum number of radar stations needed to keep a set of strategic locations under surveillance is the domination number of the associated graph.

**Application 10.2.2** *Emergency First Aid Stations:* Suppose that a natural disaster has struck some region consisting of many small villages. The vertices of a graph represent the villages in the region. An edge joining two vertices indicates that an emergency first-aid station set up in one of the corresponding villages can also serve the other one. Then a minimum dominating set of the graph would prescribe a way of serving the entire region with a minimum number of first aid stations.

**Application 10.2.3** *Chess:* Consider the problem of placing queens on a chessboard so that every square is either occupied by a queen or can be reached in one move by a queen. Determining the minimum number of queens is equivalent to finding the domination number of a 64-vertex graph, where two vertices are adjacent if and only if their corresponding squares lie on the same diagonal, same row, or same column. (See Exercises.)

### Private Neighbors

Both dominating sets in Figure 10.2.1 are minimal, because each vertex in each set dominates at least one vertex that no other vertex in that set dominates.

REVIEW FROM §1.1:

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

**DEFINITION:** Let  $D$  be a dominating set of a graph, and let  $v \in D$ . A vertex  $w$  is a **private neighbor** of  $v$  relative to  $D$  if  $v$  is the only vertex in  $D$  that dominates  $w$ . That is,  $N[w] \cap D = \{v\}$ .

**CAUTION REGARDING TERMINOLOGY:** Because a vertex dominates itself, it is possible for a vertex to be a private neighbor of itself, even if that vertex is not adjacent to itself. In such a case, a private neighbor might not be a neighbor!

The following facts are immediate consequences of the definition of private neighbor.

**Proposition 10.2.1.** *Let  $D$  be a dominating set of a graph.*

- (i) *No vertex of  $D$  is a private neighbor of any other vertex of  $D$ .*
- (ii) *Vertex  $v \in D$  is a private neighbor of itself if and only if  $v$  is not adjacent to any other vertex in  $D$ .* ◇

Two of the earliest results on domination, both due to Ore [Or62], establish basic properties of a minimal dominating set.

**Proposition 10.2.2.** [Or62] *Let  $D$  be a dominating set of a graph  $G$ . Then  $D$  is a minimal dominating set of  $G$  if and only if every vertex in  $D$  has at least one private neighbor.*

**Proof:** ( $\Rightarrow$ ) By way of contrapositive, suppose that some vertex  $v \in D$  does not have a private neighbor. Then every vertex dominated by  $v$  is also dominated by at least one other vertex in  $S$ . Thus,  $D - \{v\}$  dominates  $G$ , which implies that  $D$  is not a minimal dominating set.

( $\Leftarrow$ ) Suppose that every vertex in  $D$  has a private neighbor. Let  $v$  be any vertex in  $D$ , and let  $w$  be a private neighbor of  $v$ . Then  $D - \{v\}$  does not dominate  $w$ . Therefore,  $D$  is a minimal dominating set. ◇

**Proposition 10.2.3.** [Or62] *Let  $G$  be a graph with no isolated vertices, and let  $D$  be a minimal dominating set of  $G$ . Then the complementary vertex set  $V_G - D$  is also a dominating set of  $G$ .*

**Proof:** We show that  $V_G - D$  dominates an arbitrary vertex  $v \in V_G$ . If  $v \in V_G - D$ , then we are done. Alternatively, if  $v \in D$ , then it follows from Proposition 10.2.2 that vertex  $v$  has a private neighbor  $w$ . By Proposition 10.2.1(i), either  $w = v$  or  $w \in V_G - D$ . If  $w \in V_G - D$ , then we are done. If  $w = v$ , then  $v$  is not adjacent to any vertex in  $D$  (by Proposition 10.2.1(ii)), and since  $v$  is not an isolated vertex, it must be adjacent to some vertex in  $V_G - D$ . ◇

**Corollary 10.2.4.** *Let  $G$  be an  $n$ -vertex graph with no isolated vertices. Then we have the upper bound  $\text{dom}(G) \leq n/2$ .* ◇ (Exercises)

### Bounds on the Domination Number

**Theorem 10.2.5.** [WaAcSa79] Let  $G$  be an  $n$ -vertex graph. Then

$$\left\lceil \frac{n}{1 + \delta_{\max}(G)} \right\rceil \leq \text{dom}(G) \quad \diamondsuit (\text{Exercises})$$

**Theorem 10.2.6.** [Be73] Let  $G$  be an  $n$ -vertex graph. Then

$$\text{dom}(G) \leq n - \delta_{\max}(G) \quad \diamondsuit (\text{Exercises})$$

**Corollary 10.2.7.** [WaAcSa79] Let  $G$  be an  $n$ -vertex graph. Then

$$\text{dom}(G) \leq n - \kappa(G)$$

where  $\kappa(G)$  is the vertex-connectivity of  $G$  (§5.1).

**Proof:**  $\kappa(G) \leq \delta_{\max}(G)$ .  $\diamond$

### Independent Domination

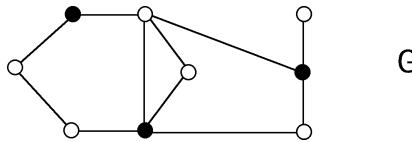
REVIEW FROM §2.3:

- A subset  $S$  of  $V_G$  is said to be an **independent set** if no two vertices in  $S$  are adjacent.
- The **independence number** of a graph  $G$ , denoted  $\alpha(G)$ , is the cardinality of a largest independent set of  $G$ .

DEFINITION: An **independent dominating set** of a graph  $G$  is an independent set of vertices that is also a dominating set of  $G$ .

DEFINITION: The **independent domination number** of a graph  $G$ , denoted  $i\text{-dom}(G)$ , is the cardinality of a minimum independent dominating set of  $G$ .

**Example 10.2.3:** The solid vertices of the graph  $G$  in Figure 10.2.2 form a minimum independent dominating set. Thus,  $i\text{-dom}(G) = 3$ .



**Figure 10.2.2** A minimum independent dominating set.

**Proposition 10.2.8.** For every graph  $G$ ,  $\text{dom}(G) \leq i\text{-dom}(G)$ .  $\diamond$

**Proposition 10.2.9.** Every independent dominating set is a minimal dominating set.

**Proof:** Let  $D$  be an independent dominating set of a graph. Then each vertex in  $D$  is its own private neighbor, and, hence,  $D$  is minimal by Proposition 10.2.2.  $\diamond$

**Proposition 10.2.10.** A set of vertices in a graph is a maximal independent set if and only if it is an independent dominating set.  $\diamondsuit$  (Exercises)

**Corollary 10.2.11.** For every graph  $G$ ,  $i\text{-dom}(G) \leq \alpha(G)$ .

**Proof:** Let  $S$  be a maximum independent set of  $G$  (i.e.,  $|S| = \alpha(G)$ ). Then  $S$  is an independent dominating set, by Proposition 10.2.10, and, hence,  $i\text{-dom}(G) \leq |S|$ .  $\diamondsuit$

**Corollary 10.2.12.** For every graph  $G$ ,  $\text{dom}(G) \leq \alpha(G)$ .  $\diamondsuit$

**Corollary 10.2.13.** Every maximal independent set is a minimal dominating set.

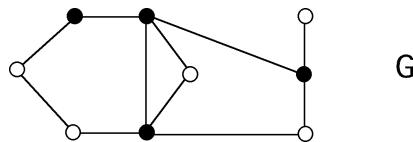
**Proof:** This is an immediate consequence of Propositions 10.2.9 and 10.2.10.  $\diamondsuit$

### Connected Domination

**DEFINITION:** A **connected dominating set** of a connected graph  $G$  is a dominating set  $D$  such that the subgraph induced on  $D$  is connected.

**DEFINITION:** The **connected domination number** of a connected graph  $G$ , denoted  $c\text{-dom}(G)$ , is the cardinality of a minimum connected dominating set of  $G$ .

**Example 10.2.4:** The solid vertices in the graph  $G$  of Figure 10.2.3 form a minimum connected dominating set. Thus,  $c\text{-dom}(G) = 4$ .



**Figure 10.2.3** A minimum connected dominating set.

### Distance- $k$ Domination

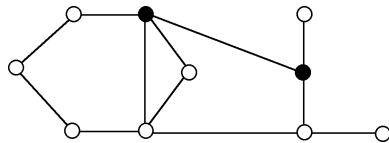
Whereas independent domination and connected domination are special cases of ordinary domination, **distance- $k$  domination** is a generalization.

**DEFINITION:** Given any integer  $k \geq 1$ , vertex subset  $D$  is a **distance- $k$  dominating set** of a graph  $G$  if for all  $v \in V_G - D$ , there exists  $x \in D$  such that  $d(v, x) \leq k$ .

**DEFINITION:** The **distance- $k$  domination number** of a graph  $G$ , denoted  $d_k\text{-dom}(G)$ , is the cardinality of a minimum distance- $k$  dominating set of  $G$ .

Observe that a  $d_1$ -dominating set is an ordinary dominating set, and observe also that  $d_1\text{-dom}(G) = \text{dom}(G)$ . Moreover,  $d_k\text{-dom}(G) \leq \text{dom}(G)$ .

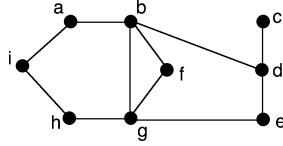
**Example 10.2.5:** The solid vertices in the graph  $G$  of Figure 10.2.4 form a minimum distance-2 dominating set. Thus,  $d_2\text{-dom}(G) = 2$ . Notice that  $\text{dom}(G) = 4$ .



**Figure 10.2.4** A minimum distance-2 dominating set.

**EXERCISES for Section 10.2**

**10.2.1** Find all minimum dominating sets of the following graph and argue why there are no smaller ones.



*In Exercises 10.2.2 through 10.2.6, determine the domination number  $\text{dom}(G)$  of the given graph, and justify your answer.*

**10.2.2<sup>s</sup>** The  $n$ -vertex path graph  $P_n$ .

**10.2.3** The  $n$ -vertex cycle graph  $C_n$ .

**10.2.4** Circular ladder graph  $CL_6$  (§1.2). Can you generalize to  $CL_n$ ?

**10.2.5** The 3-dimensional hypercube  $Q_3$ . Can you generalize to  $Q_n$ ?

**10.2.6<sup>s</sup>** The Petersen graph.

**10.2.7** Find the domination number of each of the following circulant graphs (§1.2), and justify your answer.

- a.  $\text{circ}(5 : 1, 2)$ ;
- b.  $\text{circ}(6 : 1, 2)$ ;
- c.  $\text{circ}(8 : 1, 2)$ .

**10.2.8** Determine the minimum number of queens that can be placed on a chessboard so that each square is either occupied by a queen or can be reached in one move by a queen.

**10.2.9** Determine the minimum number of knights that can be placed on a chessboard so that each square is either occupied by a knight or can be reached in one move by a knight.

*In Exercises 10.2.10 through 10.2.15, determine the independent domination number and the connected domination number of the given graph, and justify your answer.*

**10.2.10<sup>s</sup>** The  $n$ -vertex path graph  $P_n$ .

**10.2.11** The  $n$ -vertex cycle graph  $C_n$ .

**10.2.12** The Petersen graph.

**10.2.13** Circular ladder graph  $CL_6$ . Can you generalize to  $CL_n$ ?

**10.2.14** The 3-dimensional hypercube  $Q_3$ .

**10.2.15** The 4-dimensional hypercube  $Q_4$ .

**10.2.16** Determine the minimum number of non-attacking queens that can be placed on a chessboard so that each square is either occupied by a queen or can be reached in one move by a queen.

**10.2.17<sup>s</sup>** Find the independent domination number and the connected domination number of each of the following circulant graphs, and justify your answer.

- a.  $\text{circ}(5 : 1, 2)$ ;
- b.  $\text{circ}(6 : 1, 2)$ ;
- c.  $\text{circ}(8 : 1, 2)$ .

**10.2.18** In Figure 10.2.4, find all of the minimum distance-2 dominating sets of the graph.

In Exercises 10.2.19 through 10.2.23, determine the distance-2 domination number of the given graph, and justify your answer.

- 10.2.19<sup>s</sup> The  $n$ -vertex path graph  $P_n$ .
  - 10.2.20 The  $n$ -vertex cycle graph  $C_n$ .
  - 10.2.21 Petersen graph.
  - 10.2.22 Circular ladder graph  $CL_6$ . Can you generalize to  $CL_n$ ?
  - 10.2.23 The 3-dimensional hypercube  $Q_3$ .
  - 10.2.24 Draw a graph  $G$  such that  $d_k\text{-dom}(G) = 4 - k$ ,  $k = 1, 2, 3$ .
  - 10.2.25 For any integer  $m \geq 3$ , describe how to construct a graph  $G$  such that  $d_k\text{-dom}(G) = m + 1 - k$ ,  $k = 1, 2, \dots, m$ .
  - 10.2.26 Let  $G$  be a graph with  $\text{dom}(G) \geq 3$ . Prove that the diameter of the edge-complement graph  $\overline{G}$  is at most 2.
  - 10.2.27 Prove Corollary 10.2.4.
  - 10.2.28 Prove Theorem 10.2.5.
  - 10.2.29 Prove Theorem 10.2.6.
  - 10.2.30 Prove Proposition 10.2.10.
- 

## 10.3 BANDWIDTH

In this section, *bandwidth* is described in terms of vertex labelings and adjacency matrices. In §10.6, bandwidth is reinterpreted from the perspective of graph mappings.

**TERMINOLOGY:** A **standard (1-based) vertex-labeling** (or simply, **numbering**) of an  $n$ -vertex graph  $G$  is a bijection  $f : V_G \rightarrow \{1, 2, \dots, n\}$ .

**DEFINITION:** The **bandwidth of a numbering**  $f$  of a graph  $G$  is given by

$$bw_f(G) = \max \{ |f(u) - f(v)| \mid uv \in E_G \}$$

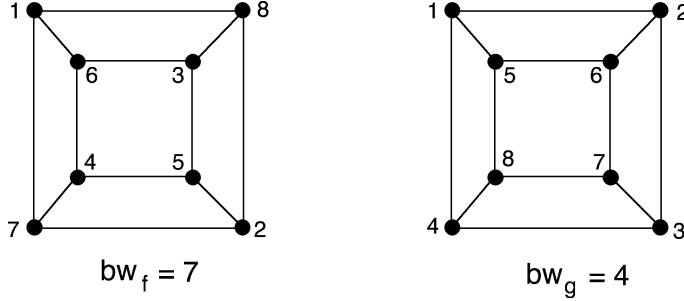
**DEFINITION:** The **bandwidth** of a graph  $G$  is given by

$$bw(G) = \min \{ bw_f(G) \mid f \text{ is a numbering of } G \}$$

The notations  $ban(G)$  and  $B(G)$  are also used for the bandwidth of  $G$ .

**DEFINITION:** A **bandwidth numbering** of a graph  $G$  is a numbering  $f$  that achieves  $bw(G)$ , i.e.,  $bw_f(G) = bw(G)$ .

**Example 10.3.1:** Two numberings,  $f$  and  $g$ , of the 3-dimensional hypercube  $Q_3$  are shown in Figure 10.3.1. One can show that  $g$  is a bandwidth numbering of  $G$ , that is,  $bw(G) = 4$  (see Exercises).



**Figure 10.3.1** Two different numberings of the hypercube  $Q_3$ .

### Interpreting Bandwidth in Terms of Adjacency Matrices

The term *bandwidth* originates from its description in terms of adjacency matrices.

NOTATION: For given a numbering  $f$  of a graph  $G$ , the adjacency matrix of  $G$  whose rows and columns are ordered according to  $f$  is denoted  $M_f$ .

**Example 10.3.1, continued:** The adjacency matrices  $M_f$  and  $M_g$  for the numberings  $f$  and  $g$  from Figure 10.3.1 are shown below. Notice that the 1's in  $M_f$  are spread over the seven diagonals immediately above and the seven immediately below the main diagonal. In contrast, the 1's in  $M_g$  are concentrated in the four diagonals immediately above and the four immediately below the main diagonal.

$$M_f = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad M_g = \begin{pmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \end{pmatrix}$$

**Proposition 10.3.1.** Let  $f$  be a bandwidth numbering of a graph  $G$ . Then every 1 in the corresponding adjacency matrix  $M_f$  lies in the **band** containing the  $\mathbf{bw}_f(G)$  diagonals above and the  $\mathbf{bw}_f(G)$  diagonals below that are closest to the main diagonal.

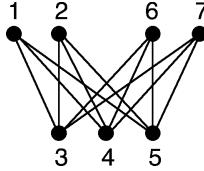
**Proof:** The result follows immediately from the observation that the  $(i, j)$ -entry in any matrix is in the  $|i - j|^{th}$  diagonal above the main diagonal if  $i < j$  or the  $|i - j|^{th}$  below if  $i > j$ .  $\diamond$

### Some Basic Results and Examples

**Proposition 10.3.2.** For any positive integer  $n$ ,

- (i)  $\mathbf{bw}(P_n) = 1$ .
- (ii)  $\mathbf{bw}(K_n) = n - 1$ .
- (iii)  $\mathbf{bw}(C_n) = 2$ .  $\diamond$  (Exercises)

**Example 10.3.2:** We claim that  $bw(K_{4,3}) = 4$ . The numbering shown in Figure 10.3.2 has bandwidth 4, so it suffices to show that  $bw_f(K_{4,3}) \geq 4$  for any numbering  $f$  of  $K_{4,3}$ . If the numbers 1, 6, and 7 are not all assigned to the same side of the bipartition, then  $bw_f(K_{4,3}) \geq 5$ . Moreover, if 2 is not assigned to the same side as 7, then  $bw_f(K_{4,3}) \geq 5$ . Thus, every numbering different from the one shown in Figure 10.3.2 has bandwidth greater than 4.



**Figure 10.3.2** A bandwidth numbering of  $K_{4,3}$ .

**Proposition 10.3.3.** Let  $H$  be a subgraph of  $G$ . Then  $bw(H) \leq bw(G)$ .

**Proof:** A bandwidth numbering  $f$  of  $G$  induces a numbering of subgraph  $H$  (the restriction of  $f$  to  $H$ ), which is at least as large as a bandwidth numbering of  $H$ .  $\diamond$

### Characterizing Bandwidth via Powers of the Path Graph

**DEFINITION:** The  $m^{th}$  **power of graph**  $G$ , denoted  $G^m$ , is the graph having vertex-set  $V_{G^m} = V_G$  and edge-set  $E_{G^m} = \{uv \mid d_G(u, v) \leq m\}$ .

**OBSERVATION** Let  $\langle v_1, v_2, \dots, v_n \rangle$  be the vertex sequence of the path graph  $P_n$ . Then for  $1 \leq m \leq n - 1$ , the adjacency matrix  $M_f$  of  $P_n^m$ , where  $f$  is the natural numbering  $f(v_i) = i$ , has all 1's on its first  $m$  diagonals and 0's everywhere else.

**Proposition 10.3.4.** For the path graph  $P_n$ ,  $bw(P_n^m) = m$  for  $1 \leq m \leq n - 1$ .

**Proof:** Proposition 10.3.1 and the observation above imply that the natural numbering corresponding to the vertex sequence of  $P_n$  is a bandwidth numbering of  $P_n$ .  $\diamond$

**Proposition 10.3.5.** Let  $G$  be an  $n$ -vertex graph. Then  $bw(G) \leq m$  if and only if  $G$  is a subgraph of  $P_n^m$ .

**Proof:** Follows immediately from Propositions 10.3.1, 10.3.3, and 10.3.4.  $\diamond$

**Corollary 10.3.6.** Let  $G$  be an  $n$ -vertex graph. Then  $bw(G) = m$  if and only if  $m$  is the smallest integer such that  $G$  is a subgraph of  $P_n^m$ .  $\diamond$

### Some Bounds on the Bandwidth

We close the section with four results, due to K. Dewdney [ChDeGiKo75], that establish bounds on the bandwidth, in terms of maximum degree, diameter, chromatic number, and vertex-connectivity.

**Proposition 10.3.7.** Let  $G$  be a graph with maximum degree  $\delta_{\max}(G) = \Delta$ . Then

$$bw(G) \geq \left\lceil \frac{\Delta}{2} \right\rceil$$

**Proof:** Let  $f$  be a bandwidth numbering of  $G$ , and let  $v$  be a vertex with neighborhood  $N(v) = \{w_1, w_2, \dots, w_\Delta\}$ . We may assume that  $f(w_1) < f(w_2) < \dots < f(w_\Delta)$ . Then

$$bw_f(G) = \max \left\{ |f(v) - f(w_1)|, |f(v) - f(w_\Delta)| \right\} \geq \left\lceil \frac{\Delta}{2} \right\rceil$$

whether  $f(w_1) < f(v) < f(w_\Delta)$  or not.  $\diamond$

**Proposition 10.3.8.** Let  $G$  be a connected  $n$ -vertex graph. Then

$$bw(G) \leq n - diam(G)$$

**Proof:** Let  $s, t \in V_G$  such that  $d(s, t) = diam(G) = l$ , and let  $P = \langle u_1, u_2, \dots, u_{l+1} \rangle$ , where  $s = u_1$  and  $u_{l+1} = t$ , be the vertex sequence of a shortest  $s-t$  path in  $G$ . First suppose  $l$  is even, and let  $r = l/2$ . We now describe how to construct a numbering  $f$  of  $G$  with  $bw_f(G) = n - 2r = n - diam(G)$ .

Assign the numbers  $1, 2, \dots, r+1$  to the vertices  $u_1, u_2, \dots, u_{r+1}$ , respectively, and assign the numbers  $n-r+1, n-r+2, \dots, n$  to the vertices  $u_{l-r+2}, u_{l-r+3}, \dots, u_{l+1}$ , respectively (see Figure 10.3.3). For convenience of notation, we will refer to the vertices of  $P$  by the numbers they were assigned. Let  $A = \{1, 2, \dots, r+1\}$  and  $B = \{n-r+1, n-r+2, \dots, n\}$  be the two vertex-subsets partitioning  $V_P$ .

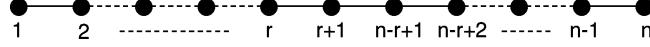


Figure 10.3.3

Observe that since  $P$  is a shortest  $s-t$  path, no vertex in set  $A$  is adjacent to a vertex in set  $B$ , except for the vertices  $r+1$  and  $n-r+1$ . Moreover, for every vertex  $v$  not yet assigned a number (i.e.,  $v \in V_G - (A \cup B)$ ),  $v$  cannot be adjacent to vertices in both sets. The remaining numbers,  $r+2, r+3, \dots, n-r$ , are assigned by the following iterative scheme:

```

While there remain vertices not yet assigned a number,
  Pick such a vertex  $v$ .
  If  $v$  is adjacent to a vertex in set  $A$ 
    Assign to  $v$  the smallest available number.
  Else
    Assign to  $v$  the largest available number.

```

It follows from the observation above that the largest possible difference between numbers assigned to adjacent vertices equals  $n - 2r = n - l = n - diam(G)$ , i.e.,

$$bw_f(G) \leq n - diam(G)$$

which completes the case where  $l$  is even.

A similar construction can be used for the case where  $l$  is odd (see Exercises).  $\diamond$

The next bound involves the *chromatic number* of a graph.

REVIEW FROM §9.1:

- The (**vertex**) **chromatic number** of graph  $G$ , denoted  $\chi(G)$ , is the smallest number  $k$  such that there is a function  $g : V(G) \rightarrow \{1, 2, \dots, k\}$  with the property that, if  $uv \in E_G$ , then  $g(u) \neq g(v)$ .
- If  $H$  is a subgraph of graph  $G$ , then  $\chi(H) \leq \chi(G)$ .

**Proposition 10.3.9.** *For the path graph  $P_n = \langle v_1, v_2, \dots, v_n \rangle$ ,  $\chi(P_n^m) = m + 1$  for  $1 \leq m \leq n - 1$ .*

**Proof:** We have  $\chi(P_n^m) \geq m + 1$ , since the complete graph  $K_{m+1}$  is a subgraph of  $P_n^m$ . To establish the reverse inequality, observe that in  $P_n^m$ , vertex  $v_1$  is not adjacent to  $v_{m+2}$ ,  $v_2$  is not adjacent to  $v_{m+3}$ , and so on. Thus, the function  $g$  defined by  $g(v_i) = i \pmod{m+1}$  is a proper  $(m+1)$ -coloring of  $P_n^m$ .  $\diamond$

**Corollary 10.3.10.** *Let  $G$  be an  $n$ -vertex graph. Then*

$$bw(G) \geq \chi(G) - 1$$

**Proof:** Let  $bw(G) = m$ . Then  $G$  is a subgraph of  $P_n^m$ , by Proposition 10.3.6. It follows that  $\chi(G) \leq \chi(P_n^m) = m + 1$ .  $\diamond$

**Remark:** Brigham and Dutton [BrDu85] showed that  $bw(G) \geq \chi'(G)/2$ , where  $\chi'(G)$  is the *edge-chromatic number* of  $G$  (§9.3).

Our last result shows that the *vertex-connectivity* is a lower bound on the bandwidth.

REVIEW FROM §5.1:

- A vertex subset  $S$  is a **vertex-cut** of a connected graph  $G$  if  $G - S$  is non-connected.
- The **vertex-connectivity** of a connected graph  $G$ , denoted  $\kappa_v(G)$ , is the minimum number of vertices whose removal can either disconnect  $G$  or reduce it to a 1-vertex graph.

**Lemma 10.3.11.** *For the path graph  $P_n = \langle v_1, v_2, \dots, v_n \rangle$ , the bound  $\kappa_v(P_n^m) \leq m$  holds for  $1 \leq m \leq n - 1$ .*

**Proof:** Since  $v_1$  is not adjacent to  $v_{m+2}$  in  $P_n^m$ , the vertex subset  $\{v_2, v_3, \dots, v_{m+1}\}$  is a vertex-cut in  $P_n^m$ .  $\diamond$

**Proposition 10.3.12.** *For any graph  $G$ ,  $bw(G) \geq \kappa_v(G)$ .  $\diamond$  (Exercises)*

**Remark:** For comprehensive surveys on bandwidth, each with extensive bibliographies, see [Br04a], [ChChDeGi82], and [LaWi99]. Further results can be found in [Ch88] and [Mi91].

### EXERCISES for Section 10.3

In Exercises 10.3.1 through 10.3.3, calculate the bandwidth of the given graph.

10.3.1<sup>s</sup> Circular ladder  $CL_3$ .

10.3.2  $K_n \times K_2$ .

10.3.3  $K_{m,n}$ , for  $m, n \geq 2$ .

10.3.4 Prove Proposition 10.3.2.

10.3.5<sup>s</sup> Prove that the bandwidth of  $K_{1,n}$  equals  $\lceil \frac{n}{2} \rceil$ .

10.3.6 Prove that the bandwidth of  $K_{2,4}$  equals 3.

10.3.7 Show that  $bw(Q_3) = 4$ . (Hint: make use of the symmetry in arguing that there is no numbering  $f$  with  $bw_f(Q_3) < 4$ .)

10.3.8<sup>s</sup> Prove that the  $2n$ -vertex wheel  $W_{2n-1}$  has bandwidth equal to  $n$ .

10.3.9 Complete the proof of Proposition 10.3.8, by constructing an appropriate numbering  $f$  for the case when  $l$  is odd, argue why  $bw_f \leq n - l$ .

10.3.10 Prove Proposition 10.3.12. (Hint: Use Lemma 10.3.11.)

## 10.4 INTERSECTION GRAPHS

Suppose that each vertex of a graph is associated with a subset of some set. We can define two vertices to be adjacent if the cardinality of their intersection exceeds some predetermined *tolerance* threshold. The threshold value of 1 corresponds to the family of *intersection graphs*, introduced in §1.2.

**CONVENTION FOR THIS SECTION:** All graphs in this section are assumed to be simple graphs, even when the modifier *simple* does not appear in the assertions.

### Intersection Graphs

**DEFINITION:** Let  $\mathcal{F} = \{S_1, S_2, \dots, S_n\}$  be a family of subsets of a set. The **intersection graph of  $\mathcal{F}$** , denoted  $\Omega(\mathcal{F})$ , is the graph whose vertex- and edge-sets are given by

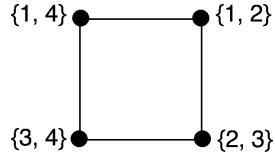
$$V_{\Omega(\mathcal{F})} = \{S_1, S_2, \dots, S_n\}$$

$$E_{\Omega(\mathcal{F})} = \{S_i S_j \mid i \neq j \text{ and } S_i \cap S_j \neq \emptyset\}$$

**DEFINITION:** Let  $G$  be a graph with  $V_G = \{v_1, v_2, \dots, v_n\}$ . Then  $G$  is an **intersection graph** if there exists a family of sets  $\mathcal{F} = \{S_1, S_2, \dots, S_n\}$  such that  $G \cong \Omega(\mathcal{F})$  with the natural correspondence  $f(v_i) = S_i$ ,  $i = 1, 2, \dots, n$ . The family  $\mathcal{F}$  is called a **set representation** of graph  $G$ .

**NOTATION:** If  $G$  is an intersection graph with set representation  $\mathcal{F} = \{S_1, S_2, \dots, S_n\}$ , then we typically refer to the vertex-set as  $\{S_1, S_2, \dots, S_n\}$ .

**Example 10.4.1:** Figure 10.4.1 depicts the cycle graph  $C_4$  as an intersection graph with a set representation given by the family  $\mathcal{F} = \{\{1, 4\}, \{1, 2\}, \{2, 3\}, \{3, 4\}\}$ .



**Figure 10.4.1** The cycle graph  $C_4$  depicted as an intersection graph.

**Proposition 10.4.1.** [Ma45] Every simple graph  $G$  is an intersection graph.

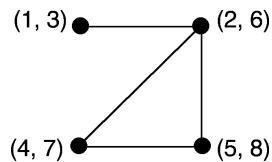
**Proof:** For each vertex  $v \in V_G$ , let  $E_v$  be the subset of edges incident on  $v$ . Then the family of subsets  $\{E_v\}_{v \in V_G}$  is a set representation of  $G$ .  $\diamond$

### Some Special Kinds of Intersection Graphs

REVIEW FROM §1.2:

- A graph  $G$  is an **interval graph** if it is an intersection graph corresponding to a family of intervals on the real line.
- 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  $G$ .

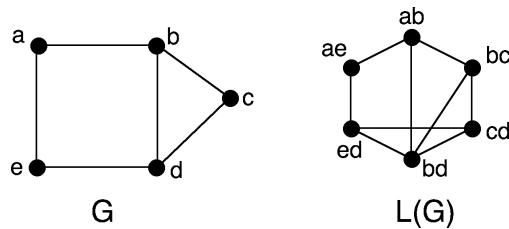
**Example 10.4.2:** The assignment of intervals to the vertices of the graph in Figure 10.4.2 shows that it is an interval graph.



**Figure 10.4.2** An interval graph.

**Remark:** The class of interval graphs is a *proper* subclass of intersection graphs. For instance, one can show that the cycle graph  $C_n$  for all  $n \geq 4$  is not an interval graph (see Exercises).

**Example 10.4.3:** Figure 10.4.3 shows a graph  $G$  and its line graph  $L(G)$ .



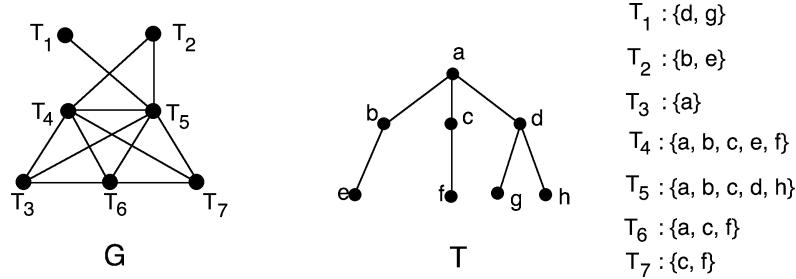
**Figure 10.4.3** A graph and its line graph.

### Subtree Graphs

**DEFINITION:** A graph  $G$  is a **subtree graph** if  $G$  is the intersection graph of a family  $\mathcal{F} = \{T_1, T_2, \dots, T_n\}$  of subtrees of a tree  $T$ , i.e., the vertex-sets of the  $T_i$ 's form a set representation of  $G$ . The tree  $T$  and the family  $\mathcal{F}$  are called a **tree representation** of  $G$ .

If  $T$  is a path (and the  $T_i$ 's are subpaths), then the tree representation is called a **path representation**, and  $G$  is said to be an intersection graph of a family of subpaths of a path.

**Example 10.4.4:** A subtree graph  $G$  and one its tree representations are shown in Figure 10.4.4. The vertex-sets of the subtrees  $T_1, T_2, \dots, T_7$  of tree  $T$  are listed at the right.



**Figure 10.4.4** A subtree graph  $G$  and one of its tree representations.

**Proposition 10.4.2.** A graph is an interval graph if and only if it is an intersection graph of a family of subpaths of a path.  $\diamondsuit$  (Exercises)

**DEFINITION:** A graph is a **chordal graph** if, for all  $n \geq 4$ , it does not contain an  $n$ -vertex cycle graph  $C_n$  as an induced subgraph.

**Example 10.4.4, continued:** The graph  $G$  in Figure 10.4.4 is a chordal graph.

**Proposition 10.4.3.** [Bu74, Ga74, Wa78] A graph is a chordal graph if and only if it is a subtree graph.  $\diamondsuit$  (See, e.g., [McMc99], Theorem 2.4.)

### Competition Graphs

**DEFINITION:** Let  $D$  be a digraph with vertex-set  $V_D$  and arc-set  $A_D$ . The **out-set** of a vertex  $v \in V_D$ , denoted  $Out(v)$ , and the **in-set**, denoted  $In(v)$ , are the vertex subsets given by

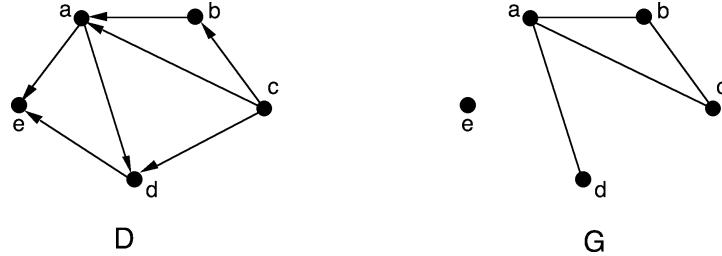
$$\begin{aligned} Out(v) &= \{w \in V_D \mid (v, w) \in A_D\} \\ In(v) &= \{w \in V_D \mid (w, v) \in A_D\} \end{aligned}$$

where the *ordered pair*  $(x, y)$  denotes the arc directed from vertex  $x$  to vertex  $y$ .

**DEFINITION:** The **competition graph of a digraph**  $D$  is the intersection graph of the family of out-sets of the vertices of  $D$ .

**DEFINITION:** A graph  $G$  is a **competition graph** if there is a digraph  $D$  such that  $G$  is (isomorphic to) the competition graph of  $D$ .

**Example 10.4.5:** A digraph  $D$  and its competition graph  $G$  are shown in Figure 10.4.5.



**Figure 10.4.5** A digraph  $D$  and its competition graph  $G$ .

**Application 10.4.1 Food Webs** [Co78]: Suppose that  $a_1, a_2, \dots, a_n$  are  $n$  animals in an ecosystem. A **food web** is a digraph model  $D$  of the predator-prey relationship among the  $a_i$ 's, where the vertex- and arc-sets of  $D$  are given by

$$\begin{aligned} V_D &= \{a_1, a_2, \dots, a_n\} \\ A_D &= \{(a_i, a_j) \mid a_i \text{ preys on } a_j\} \end{aligned}$$

Then two animals  $a_i$  and  $a_j$  compete for the same prey if and only  $a_i$  and  $a_j$  are adjacent in the competition graph of  $D$ . (See Application 1.3.9.)

### Characterizing Competition Graphs of Acyclic Digraphs

Initial investigations of competition graphs focused on acyclic digraphs, because food webs are typically acyclic.

**Lemma 10.4.4.** Every acyclic digraph contains a vertex with indegree 0.

**Proof:** If every vertex had positive indegree, then a directed cycle could be constructed by starting at any vertex, moving backward along an incoming arc to its tail vertex, and repeating the process until a vertex is repeated. ◇

**Lemma 10.4.5.** An  $n$ -vertex digraph  $D$  is acyclic if and only if its vertex-set  $V_D$  can be labeled  $\{v_1, v_2, \dots, v_n\}$  so that  $(v_i, v_j) \in A_D$  implies  $i < j$ .

**Proof:** ( $\Rightarrow$ ) To prove the necessity of the labeling condition, we use induction on  $n$ . The labeling condition holds trivially if  $n = 1$ . Assume for some  $n \geq 1$  that all  $n$ -vertex acyclic digraphs satisfy the labeling condition, and let  $D$  be an acyclic digraph on  $n+1$  vertices. By Lemma 10.4.4,  $D$  has a vertex  $w$  with indegree 0. By the induction hypothesis, the  $n$  vertices of the digraph  $\hat{D} = D - w$  can be labeled  $v_1, v_2, \dots, v_n$  so that  $(v_i, v_j) \in A_{\hat{D}}$  implies  $i < j$ . Then by adding 1 to each subscript of the  $v_i$ 's and labeling vertex  $w$  as  $v_1$ , we obtain a labeling of the vertices of  $D$  that satisfies the condition.

( $\Leftarrow$ ) To prove the sufficiency of the labeling condition, suppose that the vertices of digraph  $D$  are  $v_1, v_2, \dots, v_n$  and that  $(v_i, v_j) \in A_{\hat{D}}$  implies  $i < j$ . If there were a vertex sequence  $\langle v_{i_1}, v_{i_2}, \dots, v_{i_s}, v_{i_1} \rangle$  that represented a directed cycle, then the condition would imply that  $i_1 < i_2 < \dots < i_s < i_1$ , a contradiction. Thus,  $D$  must be acyclic. ◇

**Theorem 10.4.6.** [DuBr83, LuMa83] An  $n$ -vertex graph  $G$  is a competition graph of an acyclic digraph if and only if  $V_G$  can be labeled  $\{v_1, v_2, \dots, v_n\}$  and  $G$  has an edge clique cover  $\mathcal{K} = \{Q_1, Q_2, \dots, Q_n\}$  such that  $v_i \in Q_j$  implies  $i < j$ .

**Proof:** ( $\Rightarrow$ ) Suppose that  $G$  is the competition graph of an acyclic digraph  $D$ . By Lemma 10.4.5, we may assume that  $V_G = V_D = \{v_1, v_2, \dots, v_n\}$ , where  $(v_i, v_j) \in A_D$  implies  $i < j$ . Let  $\mathcal{K} = \{Q_1, Q_2, \dots, Q_n\}$  be the family of in-sets in  $D$ , i.e.,  $Q_i = \text{In}(v_i)$ ,  $i = 1, 2, \dots, n$ . Then  $\mathcal{K}$  is an edge clique cover of  $G$  such that  $v_i \in Q_j$  implies  $i < j$  (see Exercises).

( $\Leftarrow$ ) Suppose that  $V_G = \{v_1, v_2, \dots, v_n\}$  and  $\mathcal{K} = \{Q_1, Q_2, \dots, Q_n\}$  is an edge clique cover of  $G$  such that  $v_i \in Q_j$  implies  $i < j$ . Define a digraph  $D$  with  $V_D = V_G$  and  $A_D = \{(v_i, v_j) \mid v_i \in Q_j\}$ . By Lemma 10.4.5,  $D$  is acyclic. Moreover,  $G$  is the competition graph of  $D$  (see Exercises).  $\diamond$

**Remark:** Competition graphs of arbitrary digraphs and of loopless digraphs are characterized in [DuBr83].

### Edge Clique Covers and Intersection Graphs

*Edge clique covers* and set representations for intersection graphs are closely related.

**DEFINITION:** An **edge clique cover** of a graph  $G$  is a family  $\mathcal{K} = \{Q_1, Q_2, \dots, Q_t\}$  of complete subgraphs of  $G$  such that every edge of  $G$  is an edge of at least one of the  $Q_i$ 's, that is,

$$E_G = E_{Q_1} \cup E_{Q_2} \cup \dots \cup E_{Q_t}$$

**TERMINOLOGY NOTE:** Elsewhere in this book, a *clique* is a *maximal* complete subgraph, but to be consistent with the terminology used in intersection graph theory, we do *not* require that the complete subgraphs in an edge clique cover be maximal.

**Proposition 10.4.7.** Let  $\mathcal{F} = \{S_1, S_2, \dots, S_n\}$  be a set representation for an intersection graph  $G$ , and let  $S = \bigcup_{i=1}^n S_i$ . For each  $x \in S$ , let  $Q_x$  be the subgraph induced on the vertex subset  $V_x = \{S_i \mid x \in S_i\}$ . Then the family  $\mathcal{K}^\mathcal{F} = \{Q_x\}_{x \in S}$  is an edge clique cover of graph  $G$ .

**Proof:** To show that each  $Q_x$  is a complete subgraph of  $G$ , suppose that vertices  $S_i, S_j \in V_x$  with  $i \neq j$ . Then  $x \in S_i \cap S_j$ , and, hence,  $S_i$  and  $S_j$  are adjacent in  $G$ . Next, let  $e = S_k S_l$  be any edge of  $G$ . Then, since  $S_k$  and  $S_l$  are adjacent, there is at least one  $x \in S_k \cap S_l$ , i.e.,  $S_k, S_l \in V_x$ . Thus,  $e \in E_{Q_x}$ , which shows that  $\mathcal{K}^\mathcal{F}$  is an edge clique cover of  $G$ .  $\diamond$

**Proposition 10.4.8.** Let  $\mathcal{K} = \{Q_1, Q_2, \dots, Q_t\}$  be an edge clique cover of a graph  $G$ , and for each  $v \in V_G$ , let  $S_v = \{i \mid v \in V_{Q_i}\}$ . Then the collection  $\mathcal{F}^\mathcal{K} = \{S_v\}_{v \in V_G}$  is a set representation of graph  $G$ , that is,  $G \cong \Omega(\mathcal{F}^\mathcal{K})$ .

**Proof:** We must show that the vertex bijection  $f : V_G \rightarrow V_{\Omega(\mathcal{F}^\mathcal{K})}$  given by  $f(v) = S_v$  preserves adjacency and non-adjacency (i.e., is structure-preserving). Suppose that vertices  $v$  and  $w$  are adjacent in  $G$ . Then edge  $vw \in E_{Q_i}$  for some  $Q_i$ , and, hence,

$i \in S_v \cap S_w$ . Thus,  $S_v$  and  $S_w$  are adjacent vertices in the intersection graph  $\Omega(\mathcal{F}^{\mathcal{K}})$ . Conversely, suppose vertices  $S_v$  and  $S_w$  are adjacent in the graph  $\Omega(\mathcal{F}^{\mathcal{K}})$ . Then  $S_v \cap S_w \neq \emptyset$ , i.e.,  $i \in S_v \cap S_w$  for some  $i$ . It follows that  $v, w \in V_{Q_i}$ , and, since  $Q_i$  is a complete graph,  $v$  is adjacent to  $w$  in graph  $G$ .  $\diamond$

### Duality Between Edge Clique Covers and Set Representations

**DEFINITION:** The edge clique cover  $\mathcal{K}^{\mathcal{F}} = \{Q_x\}_{x \in S}$ , defined in Proposition 10.4.7, corresponding to a set representation  $\mathcal{F}$  of the intersection graph  $G$  is called the **dual edge clique cover of  $\mathcal{F}$** .

**DEFINITION:** The set representation  $\mathcal{F}^{\mathcal{K}} = \{S_v\}_{v \in V_G}$ , defined in Proposition 10.4.8, corresponding to an edge clique cover  $\mathcal{K}$  of a graph  $G$  is called the **dual set representation of  $\mathcal{K}$** .

**NOTATION:** Let  $G$  be an intersection graph with set representation  $\mathcal{F}$  and edge clique cover  $\mathcal{K}$ . Then  $\mathcal{F}^{(\mathcal{K}^{\mathcal{F}})}$  denotes the dual set representation of the dual edge clique cover of  $\mathcal{F}$ , and  $\mathcal{K}^{(\mathcal{F}^{\mathcal{K}})}$  denotes the dual edge clique cover of the dual set representation of  $\mathcal{K}$ .

**Proposition 10.4.9.** *Let  $G$  be an intersection graph with set representation  $\mathcal{F}$  and edge clique cover  $\mathcal{K}$ . Then both of the following are true.*

- (i) *There is a one-to-one correspondence between the sets in  $\mathcal{F}$  and the sets in  $\mathcal{F}^{(\mathcal{K}^{\mathcal{F}})}$ .*
- (ii) *There is a one-to-one correspondence between the subgraphs in  $\mathcal{K}$  and the subgraphs in  $\mathcal{K}^{(\mathcal{F}^{\mathcal{K}})}$ .*  $\diamond$  (Exercises)

### Intersection Number of a Graph

**DEFINITION:** The **intersection number** of an  $n$ -vertex graph  $G$ , denoted  $\text{int}(G)$ , is the minimum cardinality of a set  $S$  such that  $G$  is the intersection graph of a family  $\mathcal{F} = \{S_1, S_2, \dots, S_n\}$  of subsets of  $S$ .

**DEFINITION:** The **edge-clique-cover number** of a graph  $G$ , denoted  $\theta_e(G)$ , is the smallest  $t$  for which there exists an edge clique cover  $\mathcal{K} = \{Q_1, Q_2, \dots, Q_t\}$ .

**Theorem 10.4.10.** [ErGoPo66] *For every graph  $G$ ,  $\text{int}(G) = \theta_e(G)$ .*

**Proof:** Let  $S$  be a smallest set such that  $S = \bigcup_{i=1}^n S_i$  and  $\mathcal{F} = \{S_1, S_2, \dots, S_n\}$  is a set representation of  $G$  (i.e.,  $|S| = \text{int}(G)$ ). Then by Proposition 10.4.7,  $\mathcal{K}^{\mathcal{F}}$  is an edge clique cover of  $G$  of size  $|S|$ , which shows that  $\text{int}(G) \geq \theta_e(G)$ . The reverse inequality follows similarly from Proposition 10.4.8.  $\diamond$

### p-Intersection Graphs

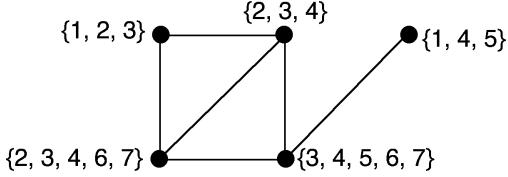
**DEFINITION:** The  **$p$ -intersection graph** of a family of subsets  $\mathcal{F} = \{S_1, S_2, \dots, S_n\}$  of a finite set  $S$ , denoted  $\Omega_p(\mathcal{F})$ , is the graph whose vertex- and edge-sets are given by

$$\begin{aligned} V_{\Omega_p(\mathcal{F})} &= \{S_1, S_2, \dots, S_n\} \\ E_{\Omega_p(\mathcal{F})} &= \{S_i S_j \mid i \neq j \text{ and } |S_i \cap S_j| \geq p\} \end{aligned}$$

**DEFINITION:** A graph  $G$  is a  **$p$ -intersection graph** if there exists a family of sets  $\mathcal{F} = \{S_1, S_2, \dots, S_n\}$  such that  $G \cong \Omega_p(\mathcal{F})$ .

Thus, the 1-intersection graphs are the ordinary intersection graphs.

**Example 10.4.6:** The subsets of  $\{1, 2, \dots, 7\}$  that label the vertices of the graph in Figure 10.4.6 show that it is a 2-intersection graph.



**Figure 10.4.6** A 2-intersection graph.

**Proposition 10.4.11.** Every graph is a  $p$ -intersection graph for all integers  $p \geq 1$ .

**Proof:** Proposition 10.4.1 establishes the base of an inductive proof on  $p$ . Assume that a graph  $G$  is a  $p$ -intersection graph for some  $p \geq 1$ , and let  $\mathcal{F} = \{S_1, S_2, \dots, S_n\}$  be a family of sets such that  $G \cong \Omega_p(\mathcal{F})$ . Let  $x \notin \bigcup_{i=1}^n S_i$ . Then

$$\hat{\mathcal{F}} = \{S_1 \cup \{x\}, S_2 \cup \{x\}, \dots, S_n \cup \{x\}\}$$

is a family of sets such that  $G \cong \Omega_{p+1}(\hat{\mathcal{F}})$ . ◊

### Tolerance Graphs

Intersection graphs and their variations discussed in this section have been generalized to *tolerance graphs*. The most general version of the definition was introduced in [JaMcMu91] and [JaMcSc91].

**DEFINITION:** Let  $\mathcal{F} = \{S_1, S_2, \dots, S_n\}$  be a family of subsets of a finite set  $S$ ; let  $\phi : R^+ \times R^+ \rightarrow R^{\geq 0}$  be a symmetric function taking pairs of positive real numbers to nonnegative reals; let  $\mu : \mathcal{P}(S) \rightarrow R^{\geq 0}$  be a function taking subsets of  $S$  to nonnegative reals; and let  $t_i$  be a positive real number, called a **tolerance**, assigned to subset  $S_i$ ,  $i = 1, 2, \dots, n$ . The  **$\phi$ -tolerance intersection graph** (or simply **tolerance graph**)  $G$  of the family  $\mathcal{F}$  with respect to  $\phi$ ,  $\mu$ , and the tolerances  $t_i$  has vertex- and edge-sets given by

$$\begin{aligned} V_G &= \{S_1, S_2, \dots, S_n\} \\ E_G &= \{S_i S_j \mid i \neq j \text{ and } \mu(S_i \cap S_j) \geq \phi(t_i, t_j)\} \end{aligned}$$

**DEFINITION:** An  $n$ -vertex graph  $G$  is a  **$\phi$ -tolerance intersection graph** if there exists a family  $\mathcal{F}$ , functions  $\phi$  and  $\mu$ , and tolerances  $t_i$ ,  $i = 1, 2, \dots, n$ , as defined above, such that  $G$  is isomorphic to the  $\phi$ -tolerance intersection graph of  $\mathcal{F}$  with respect to  $\phi$ ,  $\mu$ , and the  $t_i$ 's.

### Some Special Classes of Tolerance Graphs

For many of the classes of tolerance graphs under recent investigation, the function  $\mu$  is taken to be the cardinality of a set or the length of an interval, and sometimes, the tolerances  $t_i$  are constant or are set equal to  $\mu(S_i)$ . Some of the choices for  $\phi$  include the constant, minimum, maximum, sum, and absolute difference functions.

**Example 10.4.7:** A  $p$ -intersection graph of a family  $\mathcal{F}$  is a  $\phi$ -tolerance intersection graph, where  $\mu(S') = |S'|$  for each  $S' \subseteq S$ , tolerance  $t_i = \mu(S_i)$  for  $i = 1, 2, \dots, n$ , and  $\phi(t_i, t_j) = p$  for all pairs of tolerances.

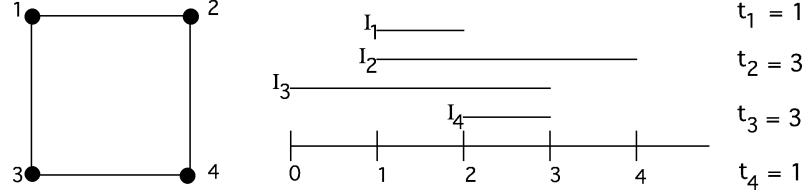
**DEFINITION:** A **min-tolerance interval graph** of a family  $\mathcal{F} = \{I_1, I_2, \dots, I_n\}$  of intervals on the real line is a  $\phi$ -tolerance intersection graph, where  $t_i$  is a positive real number assigned to interval  $I_i$ ,  $i = 1, 2, \dots, n$ ,  $\phi(t_i, t_j) = \min(t_i, t_j)$ , and for any interval  $J$ ,  $\mu(J)$  is its length.

**Proposition 10.4.12.** Every interval graph is a min-tolerance interval graph.  $\diamond$   
(Exercises)

**Remark:** The **max-tolerance** and **sum-tolerance interval graphs** are analogously defined.

**TERMINOLOGY NOTE:** The term *tolerance* was introduced in [GoMo82, GoMoTr84] in the context of min-tolerance interval graphs, and several authors refer to these simply as *tolerance graphs*.

**Example 10.4.8:** Figure 10.4.7 below depicts the cycle graph  $C_4$  as a min-tolerance interval graph, which shows that the interval graphs are a proper subclass of the min-tolerance interval graphs.



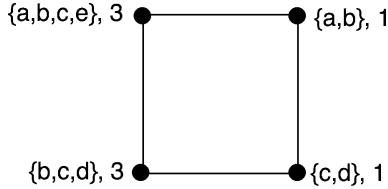
**Figure 10.4.7** The cycle graph  $C_4$  as a min-tolerance interval graph.

**DEFINITION:** Let  $\mathcal{F} = \{S_1, S_2, \dots, S_n\}$  be a family of subsets of a finite set  $S$ . An **abdiff-tolerance intersection graph** of  $\mathcal{F}$  is a  $\phi$ -tolerance intersection graph, where  $\mu(S') = |S'|$  for each  $S' \subseteq S$ ,  $t_i$  is a positive real number assigned to subset  $S_i$ ,  $i = 1, 2, \dots, m$ , and  $\phi(t_i, t_j) = |t_i - t_j|$ .

**Example 10.4.9:** Figure 10.4.8 depicts the cycle graph  $C_4$  as an abdiff-tolerance graph. Next to each vertex is its corresponding subset and tolerance.

**DEFINITION:** Let  $\phi$  be a symmetric function mapping pairs of nonnegative integers to nonnegative integers. A graph is a  **$\phi$ -tolerance competition graph** if it is the  $\phi$ -tolerance intersection graph of the family of out-sets of the vertices of some digraph.

**Remark:** For examples and results on  $\phi$ -tolerance competition graphs, see, for example, [AnLaLuMcMe94], [BrMcVi95], [BrMcVi96], and [BrCaVi00]. Some of the functions  $\phi$  considered in those papers include  $\phi(t_i, t_j) = \min(t_i, t_j)$ ,  $\max(t_i, t_j)$ ,  $t_i + t_j$ , and  $|t_i - t_j|$ .



**Figure 10.4.8** The cycle graph  $C_4$  as an abdiff-tolerance intersection graph.

**Remark:** For several other results, examples, and applications of intersection graphs and tolerance graphs, see the monograph by McKee and McMorris [McMc99]. Also, Golumbic and Trenk have written the first book [GoTr03] devoted entirely to tolerance graphs.

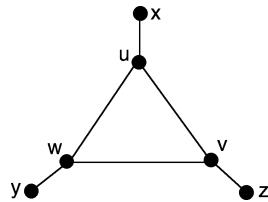
#### EXERCISES for Section 10.4

NOTE: Some of the exercises that appear in §1.2 are repeated here for convenience.

- 10.4.1 Draw the interval graph for the intervals  $(0, 2), (3, 8), (1, 4), (3, 4), (2, 5), (7, 9)$ .
- 10.4.2<sup>s</sup> Show that the complete graph  $K_n$  is an interval graph for all  $n \geq 1$ .
- 10.4.3 Show that for all  $n \geq 4$ , the cycle graph  $C_n$  is not an interval graph.

*In Exercises 10.4.4 through 10.4.8, determine the intersection number of the given graph, and justify your answer.*

- 10.4.4 The complete graph  $K_n$ .
- 10.4.5<sup>s</sup> The 3-vertex path graph  $P_3$ .
- 10.4.6 The n-fold self union  $nK_2$  (§2.4).
- 10.4.7 The complete graph  $K_{2,3}$ .
- 10.4.8<sup>s</sup>



- 10.4.9 Prove Proposition 10.4.2.
- 10.4.10 Prove Proposition 10.4.9.
- 10.4.11 (a) Show that the family  $\mathcal{K}$  of in-sets defined in the first part of the proof of Theorem 10.4.6 is an edge clique cover that satisfies the labeling condition.  
(b) Show that  $G$  is the competition graph of the digraph  $D$  defined in the second part of the proof of Theorem 10.4.6.
- 10.4.12<sup>s</sup> Show that every interval graph is a min-tolerance interval graph.

## 10.5 LINEAR GRAPH MAPPINGS

Recall from §2.1 that an isomorphism between two simple graphs consists of a vertex bijection that preserves adjacency and non-adjacency and an edge bijection that are consistent. Here, we introduce the more general *linear graph mapping* by requiring only that the vertex mapping preserve adjacency and dropping the requirement that the two mappings be bijective.<sup>†</sup>

REVIEW FROM §2.1:

- A vertex bijection  $f_V : V_G \rightarrow V_H$  between the vertex-sets of simple graphs  $G$  and  $H$  is **structure-preserving** if it preserves adjacency and non-adjacency, that is, if for every pair of vertices in  $G$ ,
- $$u \text{ and } v \text{ are adjacent in } G \iff f_V(u) \text{ and } f_V(v) \text{ are adjacent in } H$$
- Every structure-preserving vertex bijection  $f_V : V_G \rightarrow V_H$  of simple graphs induces an edge bijection  $f_E : E_G \rightarrow E_H$ , given by the rule  $f_E(uv) = f_V(u)f_V(v)$ .
  - A **graph isomorphism**  $f : G \rightarrow H$  of simple graphs is completely specified by a structure-preserving vertex bijection  $f_V : V_G \rightarrow V_H$ . Formally, it consists of the vertex bijection  $f_V$  paired with the induced edge bijection  $f_E$ .

**DEFINITION:** A **linear graph mapping** (or **graph homomorphism**)  $f : G \rightarrow H$  between two simple graphs  $G$  and  $H$  is a pair of functions,  $f_V : V_G \rightarrow V_H$  and  $f_E : E_G \rightarrow E_H$ , such that  $f_V$  is an adjacency-preserving vertex mapping and  $f_E$  is the induced edge mapping. That is,

$$u \text{ and } v \text{ are adjacent in } G \implies f_V(u) \text{ and } f_V(v) \text{ are adjacent in } H$$

and

$$f_E(uv) \equiv f_V(u)f_V(v)$$

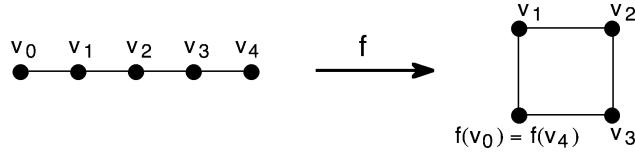
A linear graph mapping  $f$  between two simple graphs is completely specified by its adjacency-preserving vertex mapping  $f_V$ .

**NOTATION:** The subscripts  $V$  and  $E$  on  $f$  are sometimes omitted when no ambiguity can result.

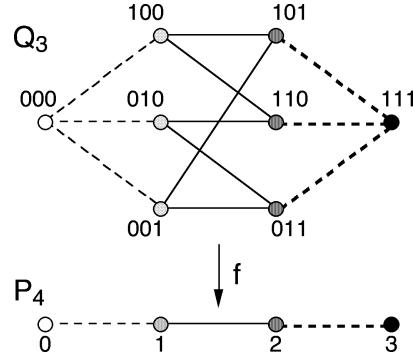
**Example 10.5.1:** Let  $v_0, v_1, \dots, v_n$  be the successive vertices along a path graph  $P_{n+1}$  of length  $n$ . The vertex mapping  $f_V : V_{P_{n+1}} \rightarrow V_{C_n}$  given by  $f(v_0) = f(v_n) = v_0$  and  $f(v_i) = v_i$  for  $i = 1, 2, \dots, n - 1$  is adjacency-preserving. Thus, if  $f_E : E_{P_{n+1}} \rightarrow E_{C_n}$  is its induced edge mapping, then  $f = (f_V, f_E)$  is a linear graph mapping from the path graph of length  $n$  to the cycle graph of length  $n$ . Figure 10.5.1 illustrates the mapping for  $n = 4$ .

---

<sup>†</sup> Readers familiar with group theory will notice the similarity between group homomorphism and linear graph mapping in that both are structure-preserving correspondences that are not necessarily bijective.

**Figure 10.5.1** A linear graph mapping from  $P_5$  to  $C_4$ .

**Example 10.5.2:** A linear mapping  $f : Q_3 \rightarrow P_4$  is represented in Figure 10.5.2 by the vertical projection of  $Q_3$  downward onto  $P_4$ . The shading of the vertices and the features of the edges further reinforce the imagery.

**Figure 10.5.2** Mapping the cube graph  $Q_3$  to a path of length 3.

**Proposition 10.5.1.** Let  $f = (f_V, f_E)$  be a linear graph mapping between two simple graphs. If the vertex function  $f_V$  and the edge function  $f_E$  are both bijections, then  $f$  is a graph isomorphism.

**Proof:** If the functions  $f_V$  and  $f_E$  are bijections, then the adjacency-preserving property of  $f_V$  implies that it must also preserve non-adjacency.  $\diamond$

**DEFINITION:** The **image of a graph**  $G$  under a linear mapping  $f : G \rightarrow H$  is the subgraph of  $H$  whose vertex-set is  $f(V_G)$  and whose edge-set is  $f(E_G)$ .

**Corollary 10.5.2.** If both the vertex function and the edge function of a linear graph mapping  $G \rightarrow H$  are one-to-one (but not necessarily onto), then the domain graph  $G$  is isomorphic to its image in the codomain graph  $H$ .  $\diamond$

### Linear Graph Mappings Between General Graphs

We saw in §2.1 how the concept of isomorphism can be easily extended to general graphs. Analogously, we define linear graph mapping between general graphs.

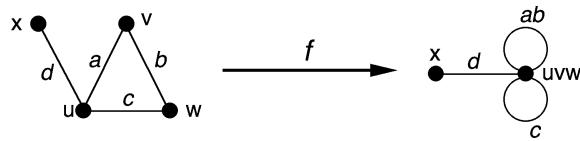
**DEFINITION:** A **linear graph mapping**  $f : G \rightarrow H$  between two (possibly non-simple) graphs  $G$  and  $H$  is a pair of functions

$$f_V : V_G \rightarrow V_H \text{ and } f_E : E_G \rightarrow E_H$$

such that the *incidence-preservation property* holds: for every edge  $e \in E_G$ , the endpoint set of  $e$  is mapped by  $f_V$  to the endpoint set of the edge  $f_E(e)$ .

**Remark:** Observe that if  $G$  and  $H$  are simple graphs, then the incidence-preservation property reduces to adjacency-preserving. Also, it is straightforward to verify that Proposition 10.5.1 and its corollary hold for this more general linear graph mapping.

**Example 10.5.3:** Figure 10.5.3 shows a linear graph mapping  $f$  from a simple graph to a non-simple graph. The vertex and edge labels on the codomain graph indicate the preimages of the functions  $f_V$  and  $f_E$ .



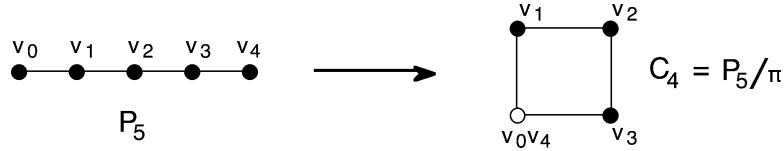
**Figure 10.5.3** Mapping a simple graph to a non-simple graph.

### Linear Graph Mappings Via Amalgamation

In light of Corollary 10.5.2, a linear graph mapping becomes a bona fide generalization of isomorphism when one or both of the vertex and edge mappings are not one-to-one. When two or more vertices are mapped to the same vertex in the codomain graph, they may be viewed as being merged into that one vertex. Similarly, edges that get mapped to the same edge in the codomain are merged into that one edge. These observations motivate the following definitions of two kinds of amalgamation that provide a different perspective for linear graph mappings.

**DEFINITION:** Let  $G$  be any graph, and let  $\pi = \{V_1, V_2, \dots, V_t\}$  be a partition of its vertex-set  $V_G$ . The **vertex-partition amalgamation** of graph  $G$  corresponding to partition  $\pi$  is the transformation of  $G$  into a graph  $G/\pi$  that results from merging (amalgamating) all of the vertices in each cell of the partition.<sup>†</sup> That is, for each  $i = 1, 2, \dots, t$ , the vertices in subset  $V_i$  are merged into a single vertex, generically denoted  $V_i$ . Thus, the vertex-set  $V_{G/\pi} = \{V_1, V_2, \dots, V_t\}$ , and the edge-set  $E_{G/\pi} = E_G$ , except that any edge in  $G$  that had  $u \in V_i$  as an endpoint now has, in  $G/\pi$ , the amalgamated vertex  $V_i$  as an endpoint instead.

**Example 10.5.4:** Figure 10.5.4 shows how the cycle graph  $C_4$  can be obtained by a vertex-partition amalgamation of the path graph  $P_5 = \langle v_0, v_1, v_2, v_3, v_4 \rangle$  corresponding to the partition  $\pi = \{\{v_0, v_4\}, \{v_1\}, \{v_2\}, \{v_3\}\}$ . The amalgamated vertex is labeled by juxtaposing the names of the vertices that were merged.



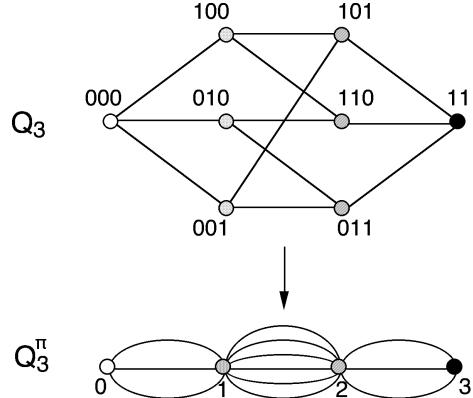
**Figure 10.5.4** Cycle graph  $C_4$  obtained from a vertex-amalgamation of  $P_5$ .

Notice that this transformation is identical to the linear graph mapping of Example 10.5.1.

---

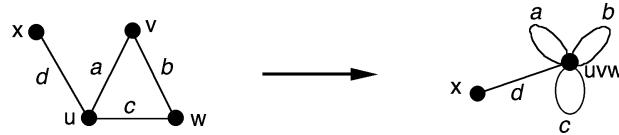
<sup>†</sup> This notation is analogous to the one used for quotient groups in group theory.

**Example 10.5.5:** Figure 10.5.5 shows the vertex-partition amalgamation of the cube graph  $Q_3$  corresponding to the partition of the vertices according to the number of 1's in their label.



**Figure 10.5.5** A vertex-partition amalgamation of the cube graph  $Q_3$ .

**Example 10.5.6:** Figure 10.5.6 shows the vertex-partition amalgamation of the domain graph from Example 10.5.3, where the corresponding partition of the vertex-set is  $\{\{x\}, \{u, v, w\}\}$ .

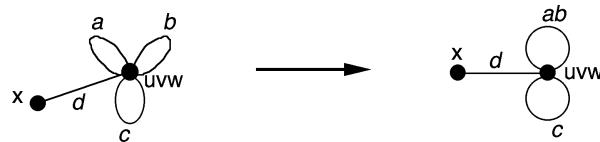


**Figure 10.5.6** A vertex-partition amalgamation based on  $\{\{x\}, \{u, v, w\}\}$ .

### Multi-Edge Amalgamation

**DEFINITION:** Let  $G$  be any graph, and let  $A = \{e_1, e_2, \dots, e_s\} \subseteq E_G$  be any subset of edges having the same endpoints. The **multi-edge amalgamation** corresponding to  $A$  is the graph  $G_A$  that results from merging (amalgamating) all of the edges in  $A$  into a single edge generically denoted  $e_1e_2 \cdots e_s$ . Thus,  $V_{G_A} = V_G$  and  $E_G = (E_G - A) \cup \{e_1e_2 \cdots e_s\}$ .

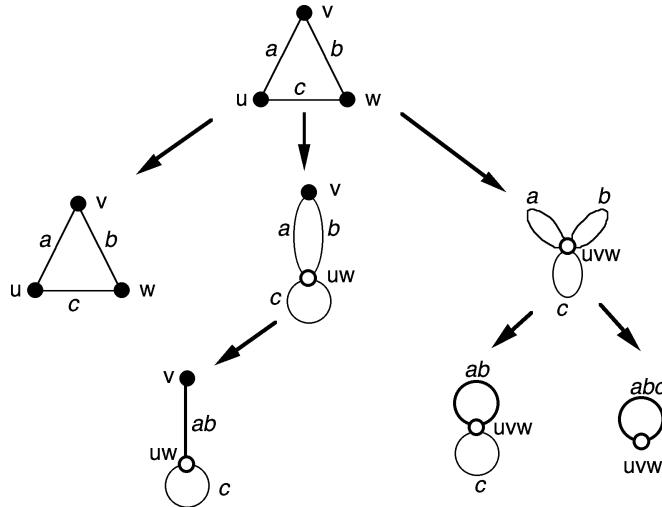
**Example 10.5.6, continued:** Figure 10.5.7 shows the graph that results from merging the edges in  $A = \{a, b\}$ . Notice that the resulting graph is the image of the linear graph mapping defined in Example 10.5.3.



**Figure 10.5.7** A multi-edge amalgamation of  $A = \{a, b\}$ .

**Example 10.5.7:** The six isomorphism types of graphs derivable from the cycle graph  $C_3$  by vertex-partition amalgamation followed by multi-edge amalgamation are shown in

Figure 10.5.8. The graphs at depth 1 are the isomorphism types of all possible vertex-partition amalgamations. The graphs at depth 2 are all possible isomorphism types that result from subsequent multi-edge amalgamations. It is easy to verify that these six types are the isomorphism types of all possible images of a linear graph mapping of  $C_3$ . Observe that the vertex and edge labels can be viewed from two different perspectives, either as preimages of linear graph mappings (the leftmost one being the identity isomorphism), or as the vertices and edges that are merged in the vertex-partition and multi-edge amalgamations, respectively.

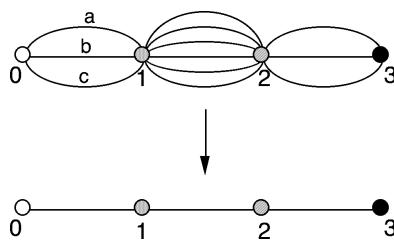


**Figure 10.5.8** Possible results of iterative amalgamation of the graph  $C_3$ .

### Complete Multi-Edge Amalgamation

**DEFINITION:** The **complete multi-edge amalgamation** of any graph  $G$  is the graph that results from taking each multi-edge and merging its edges into a single edge.

**Example 10.5.8:** The complete multi-edge amalgamation of the graph  $Q_3/\pi$  (from Example 10.5.5) results in the path graph  $P_4$ , as shown in Figure 10.5.9. Recall that  $P_4$  was the image of the linear graph mapping given in Example 10.5.2.



**Figure 10.5.9** The complete multi-edge amalgamation of the graph  $Q_3/\pi$ .

**Remark:** The last three examples showed how a linear graph mapping can be realized by vertex-partition amalgamation followed by iterative (or complete) multi-edge amalgamation. In fact, *every* linear graph mapping can be realized using these two kinds of amalgamations, and vice-versa. We state and prove the result for simple graphs.

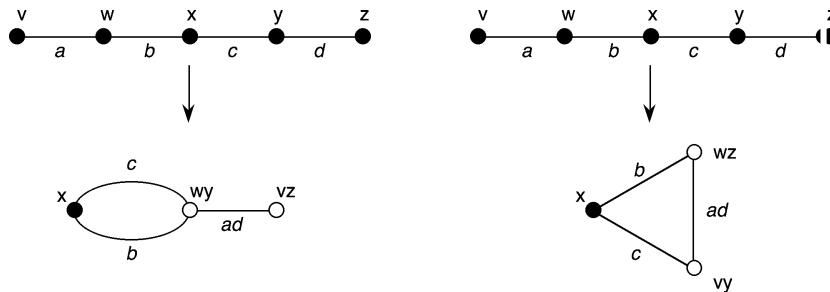
**Proposition 10.5.3.** Let  $f : G \rightarrow H$  be a linear graph mapping from a simple graph  $G$  onto a simple graph  $H$ . Then  $H$  can be obtained from  $G$  by a vertex-partition amalgamation  $G/\pi$  of  $G$  followed by a complete multi-edge amalgamation of  $G/\pi$ . Conversely, any vertex-partition amalgamation of a simple graph  $G$  followed by a complete multi-edge amalgamation that results in a graph  $H$  corresponds to a linear graph mapping from  $G$  onto  $H$ , where  $H$  has no multi-edges.

**Proof:** Let  $f^{-1}(w)$  be the preimage of vertex  $w$  for each  $w \in V_H$ , and let  $f^{-1}(e)$  be the preimage of edge  $e$  for each  $e \in E_H$ . Then  $\pi = \{f^{-1}(w) \mid w \in V_H\}$  is a partition of  $V_G$ , and each  $f^{-1}(e)$  corresponds to a multi-edge in the vertex-partition amalgamation  $G/\pi$  (since  $f$  preserves adjacency). It follows that  $H$  is obtained from the complete multi-edge amalgamation of  $G/\pi$ . The converse assertion is an immediate consequence of the definitions of the vertex-partition and complete multi-edge amalgamations.  $\diamond$

### Mapping by Merging Two Ordinary Edges

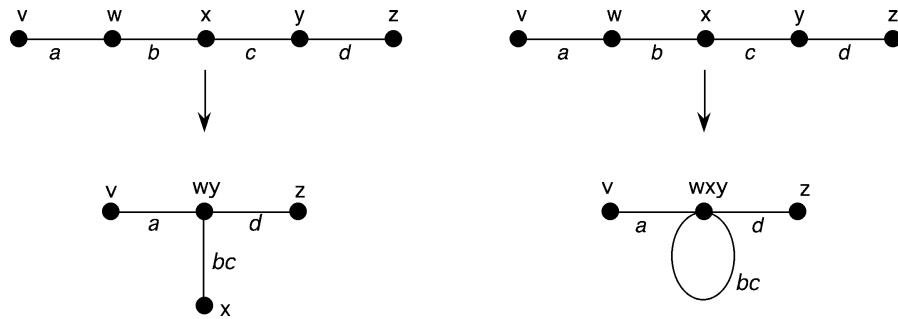
Merging any two edges of a graph  $G$  requires amalgamating their endpoint sets. If each of the other vertices and edges of  $G$  is mapped to itself, then the resulting transformation is a linear graph mapping from  $G$  onto the transformed graph. The next two examples illustrate four instances of this kind of transformation.

**Example 10.5.9:** The first and last edge of the path graph  $P_5$  could be merged in two different ways (without merging the other two edges), according to how their endpoints are merged. This is determined by the vertex-partition amalgamation that is used. The transformation shown on the left in Figure 10.5.10 results from the vertex-partition amalgamation corresponding to the partition  $\{\{v, z\}, \{w, y\}, \{x\}\}$ . The one on the right uses the partition  $\{\{v, y\}, \{w, z\}, \{x\}\}$ .



**Figure 10.5.10** Two ways to merge the first and last edges of  $P_5$ .

**Example 10.5.10:** The adjacent edges of the path graph  $P_5$  could be merged in two different ways (without merging the other two edges). The transformation shown on the left in Figure 10.5.11 results from the vertex-partition amalgamation corresponding to the partition  $\{\{w, y\}, \{v\}, \{x\}, \{z\}\}$ . The one on the right uses the partition  $\{\{v\}, \{z\}, \{w, x, y\}\}$ .

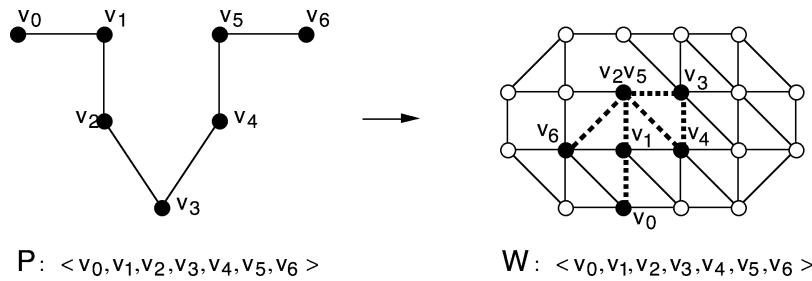


**Figure 10.5.11** Two ways to merge the adjacent edges of  $P_5$ .

### Linear Graph Mappings of Paths

A walk of length  $n$  in a graph can be regarded as the image of a path of length  $n$  under a linear graph mapping.

**Example 10.5.11:** Figure 10.5.12 shows a path  $P$  of length 6 being mapped to a walk  $W$  of length 6. Each vertex label along walk  $W$  indicates the preimage or preimages of the vertex.



**Figure 10.5.12** Mapping a path of length 6 to a walk of length 6.

**Proposition 10.5.4.** Every connected graph is the image of a path.

**Proof:** In a connected graph, there is a walk that traverses every edge. According to the remark immediately above, there is a path whose image is that walk.  $\diamond$

**Proposition 10.5.5.** Let  $f : G \rightarrow H$  be a linear mapping of graphs. Then  $G$  is bipartite whenever  $f(G)$  is bipartite.

**Proof:** Suppose that  $f(G)$  is bipartite, and that  $(U, V)$  is the corresponding bipartition of  $V_{f(G)}$ . If there were an edge  $e$  joining two vertices of  $f^{-1}(U)$ , then its endpoints would both be in  $U$ , which would imply (by the definition of a linear mapping) that its image  $f(e)$  would join two vertices of  $U$ . An edge joining two vertices of  $f^{-1}(V)$  would lead to the same contradiction.  $\diamond$

**Corollary 10.5.6.** A graph can be linearly mapped onto a path if and only if it is bipartite.

**Proof:** It follows from Proposition 10.5.5 that if a graph  $G$  can be linearly mapped onto a path, which is a bipartite graph, then  $G$  itself must be bipartite.

Conversely, consider a bipartite graph  $G$  on vertex parts  $X$  and  $Y$ . Graph  $G$  can be linearly mapped onto the path  $P_2$  of length 1, by mapping all the vertices of part  $X$  to one endpoint of  $P_2$  and all the vertices of part  $Y$  to the other. All the edges are mapped to the only edge of  $P_2$ .  $\diamond$

### Mapping Graphs Onto Pseudopaths

**DEFINITION:** A **pseudopath of length  $n$**  consists of a spanning path of length  $n$  with one or more self-loops at one or more of its vertices.

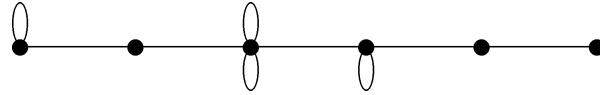


Figure 10.5.13 A pseudopath of length 5.

**Proposition 10.5.7.** Every graph can be linearly mapped onto a pseudopath.

**Proof:** In each component of  $G$ , choose a root and construct a breadth-first spanning tree. Let  $m$  be the maximum depth of any of these bfs-trees. Let  $P$  be the pseudopath of length  $m$  with consecutive vertices  $v_0, \dots, v_m$ , with edges  $e_j$  joining  $v_{j-1}$  and  $v_j$ , and with a self-loop  $\ell_j$  at each vertex  $v_j$ , as in Figure 10.5.14.

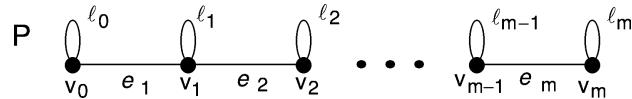


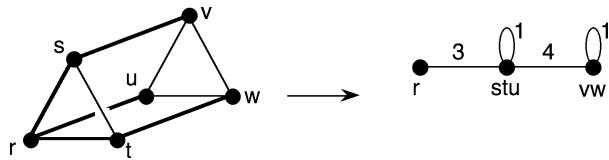
Figure 10.5.14 A pseudopath with a self-loop at every vertex.

For each component  $H$  of graph  $G$ , construct a bfs-tree rooted at an arbitrary vertex. The mapping is specified by the following sequence of steps.

1. Map the root of that bfs-tree to the vertex  $v_0$  of the pseudopath.
2. Map the children of the root to  $v_1$ .
3. Map the grandchildren to  $v_2$ , and so on. That is, for  $j = 0, \dots, m$ , map all the vertices at level  $j$  of the bfs-tree for  $H$  to the vertex  $v_j$ .
4. Map every (non-tree) edge joining two vertices within level  $j$  to the self-loop  $\ell_j$ .
5. Map every (tree or non-tree) edge joining a vertex at level  $j$  with a vertex at level  $j+1$  to the edge  $e_j$ .

From §4.2, it follows that the endpoints of every edge of a graph are always within one level with respect to any bfs-tree. Thus, this description of the edge mapping is complete.  $\diamond$

**Example 10.5.12:** The mapping from  $K_3 \times K_2$  to a pseudopath of length 2 in Figure 10.5.15 illustrates the construction of Proposition 10.5.7. The bold edges in  $K_3 \times K_2$  are in a bfs-tree. Each edge of the pseudopath is labeled by the number of edges mapped onto it.

**Figure 10.5.15** Mapping  $K_3 \times K_2$  to a pseudopath.**EXERCISES for Section 10.5**

For Exercises 10.5.1 through 10.5.8, draw all the different isomorphism types of graphs that can be obtained by amalgamating exactly two vertices of the given graph.

- |        |               |                     |             |
|--------|---------------|---------------------|-------------|
| 10.5.1 | $P_3$ .       | 10.5.2 <sup>s</sup> | $C_5$ .     |
| 10.5.3 | $W_5$ .       | 10.5.4              | $Q_3$ .     |
| 10.5.5 | $K_4 - K_2$ . | 10.5.6              | $W_6$ .     |
| 10.5.7 | $C_6$ .       | 10.5.8              | $K_{3,3}$ . |

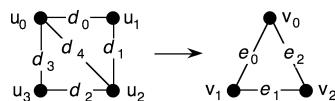
For Exercises 10.5.9 through 10.5.16, draw all the different isomorphism types of graphs that can be obtained by amalgamating exactly two edges of the given graph.

- |         |               |                      |             |
|---------|---------------|----------------------|-------------|
| 10.5.9  | $P_3$ .       | 10.5.10 <sup>s</sup> | $C_5$ .     |
| 10.5.11 | $W_5$ .       | 10.5.12              | $Q_3$ .     |
| 10.5.13 | $K_4 - K_2$ . | 10.5.14              | $W_6$ .     |
| 10.5.15 | $C_6$ .       | 10.5.16              | $K_{3,3}$ . |

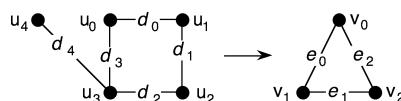
For Exercises 10.5.17 through 10.5.20, draw all the different isomorphism types of graphs that can be obtained as images of the given graph under a linear mapping.

- |         |             |         |          |
|---------|-------------|---------|----------|
| 10.5.17 | $P_3$ .     | 10.5.18 | $P_4$ .  |
| 10.5.19 | $K_{1,3}$ . | 10.5.20 | $2K_2$ . |

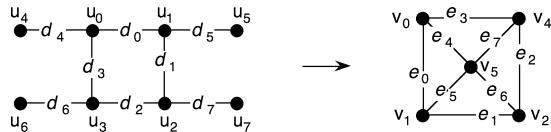
For Exercises 10.5.21 through 10.5.24, the graph at the tail of the arrow is to be linearly mapped ONTO the graph at the head. Construct a table with all the vertices and edges of the domain graph in the first row. In the second row, below each vertex or edge, write its image in the codomain graph.

10.5.21<sup>s</sup>

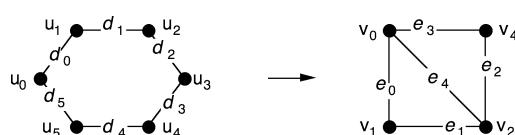
10.5.22



10.5.23



10.5.24



For Exercises 10.5.25 through 10.5.29, describe a linear mapping from the designated domain onto the designated codomain.

- 10.5.25 From the cube graph  $Q_3$  onto the complete graph  $K_4$ .
- 10.5.26<sup>s</sup> From the cycle graph  $C_{10}$  onto the complete graph  $K_5$ .
- 10.5.27 From the cycle graph  $C_{2n}$  onto the path graph  $P_{n+1}$ , for  $n \geq 1$ .
- 10.5.28 From the path graph  $P_n$  onto an arbitrary path graph of shorter length.
- 10.5.29 From the cycle graph  $C_{kn}$  onto the cycle graph  $C_n$ , for  $k, n \geq 1$ .

For Exercises 10.5.30 through 10.5.36, prove that there is no linear mapping from the designated domain onto the designated codomain.

- 10.5.30<sup>s</sup> From the cycle graph  $C_3$  onto the path graph  $P_4$ .
- 10.5.31 From the cycle graph  $C_3$  onto the path graph  $P_3$ .
- 10.5.32 From  $K_4 - K_2$  onto the path graph  $P_6$ .
- 10.5.33 From  $K_4 - K_2$  onto the path graph  $P_5$ .
- 10.5.34 From  $K_4 - K_2$  onto the cycle graph  $C_4$ .
- 10.5.35 From the cycle graph  $C_{2n+1}$  onto the cycle graph  $C_{2n}$ , for  $n \geq 1$ .
- 10.5.36 From the cycle graph  $C_{2n}$  onto the cycle graph  $C_{2n-1}$ , for  $n \geq 3$ .
- 10.5.37<sup>s</sup> Give an example of a linear mapping of a connected graph onto a connected graph such that the cycle rank (see §4.5) of the codomain is larger than the cycle rank of the domain.
- 10.5.38 Give an example of a linear mapping of connected graphs such that the cycle rank of the codomain is smaller than the cycle rank of the domain.

---

## 10.6 MODELING NETWORK EMULATION

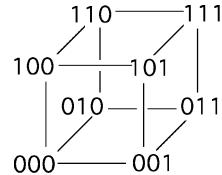
Sometimes a computational algorithm for some fixed process, such as matrix inversion or sorting, has been written for one parallel architecture, and there is a subsequent need to run that process on a computer with a different parallel architecture. Instead of designing, coding, and testing a new program for that different parallel architecture, it may be simpler to *emulate* the algorithm written for the first architecture on the second architecture, by means of a linear mapping. We briefly discuss graph models for the parallel architecture of a computer and how parallel algorithms are executed.

### Graph Models for Parallel Computers

**Application 10.6.1 Parallel Computer Architectures** In a parallel architecture for a computer, a large number of identical processors are the nodes of a network, and some miniaturized wires connect each processor to a few others. For purposes of coordinating all the processors in a distributed algorithm, interconnection network models are highly symmetrical. Input is fed to some of the processors at the start of the execution. During the execution, there is a two-phase alternating pattern of computation and communication. In the computational phase, each of the identical processors uses its

input to calculate some output. In the communication phase, some bits may be passed from a processor to some of its neighbors and may also be received from some neighbors. Ultimately, the correct answer is collected as output of some of the processors.

**Example 10.6.1:** Finding the minimum among  $2^n$  unsorted items requires  $O(2^n)$  steps, since each item must be considered as a candidate. Just as a tennis tournament for  $2^n$  players can find a winner in  $n$  elimination rounds, a parallel computer can find a minimum among  $2^n$  unsorted items in  $n$  computational steps. A parallel computer for finding the minimum of  $2^n$  items can be modeled as a hypercube of processors labeled by binary numerals, as shown in Figure 10.6.1.

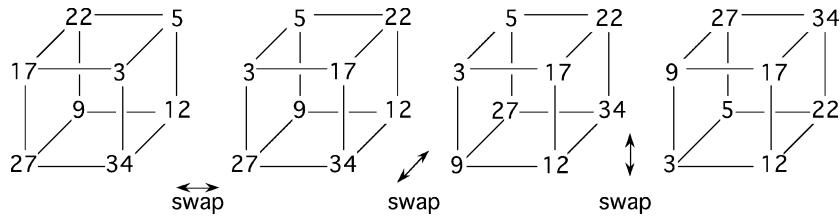


**Figure 10.6.1** Small-scale model of a hypercube parallel computer.

On the first step, the item in each processor is compared with the item in the processor whose bitstring label differs only in the first bit (reading right to left), and they are swapped, if necessary, so that the smaller item moves to the processor indexed by the smaller numeral. Thus, in Figure 10.6.1, all the winners of the first round would be in the left plane of the hypercube.

On the second step, the item in each processor is compared to the item in the processor whose bitstring label differs only in the second bit, and they are swapped, if necessary, so that, once again, the smaller item moves to the processor indexed by the smaller numeral. Thus, in Figure 10.6.1, all the winners of the second round would be in the front plane of the hypercube, and so on.

In general, within  $n$  steps, the minimum item is in processor  $00 \cdots 0$ . The execution time of this parallel algorithm is only  $O(n)$ , a vast improvement over  $O(2^n)$  strictly sequential comparison steps. Figure 10.6.2 illustrates the application of this algorithm to a list of 8 items.



**Figure 10.6.2** Finding the minimum on a hypercube.

### A Model for Porting Parallel Algorithms

**Application 10.6.2 Emulating an Interconnection Network:** It is sometimes necessary to execute a distributed algorithm on a parallel computer whose processors and network configuration differ from those on which the distributed algorithm is based, via a process called **emulation**. In an emulation, the network model for which the

distributed algorithm was written is called the *guest*, and the network model on which it is to be executed is called the *host*. The process of *porting* the algorithm from the guest parallel computer to the host parallel computer is modeled by a graph mapping from the guest network to the host network. This terminology has been adopted also by the graph-theoretic model.

**TERMINOLOGY:** For a graph mapping  $f : G \rightarrow H$  in the context of network emulation, the domain graph  $G$  is called the **guest**, and the codomain graph  $H$  is called the **host**.

Sometimes the network emulation model requires a more general graph mapping, called a *semilinear* graph mapping, which we define later in this section. The terminology introduced in the next subsection applies to this more general graph mapping as well.

### Load and Congestion of a Graph Mapping

Although a host processor can use its memory to do the job of more than one guest processor, it emulates just one of them at a time. The number of guest processors a host processor must emulate is called its *load*. Load delays a computation, relative to what it would have taken on the guest network itself.

**DEFINITION:** The **load at a vertex**  $v$  of the host  $H$  of a graph mapping  $f : G \rightarrow H$  is the number of vertices in the preimage  $f_V^{-1}(v)$ .

**DEFINITION:** The **load of a graph mapping**  $f : G \rightarrow H$  is its maximum load at any vertex, taken over all vertices of the host  $H$ .

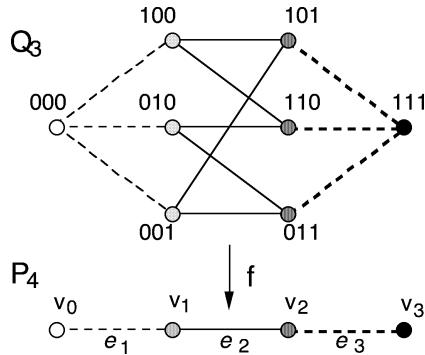
Moreover, if two neighboring host processors are emulating more than one guest processor apiece, then only one message at a time can traverse the wire between those processors, lest two messages interfere. While emptying the message queues for all the competing messages, more delay occurs. The number of guest wires that must be emulated by a given host wire is called its *congestion*.

**DEFINITION:** The **congestion on an edge**  $e$  of the host  $H$  of a graph mapping  $f : G \rightarrow H$  is the number of edges in the preimage  $f_E^{-1}(e)$ .

**DEFINITION:** The **congestion of a graph mapping**  $f : G \rightarrow H$  is its maximum congestion at any edge, taken over all edges of the host  $H$ .

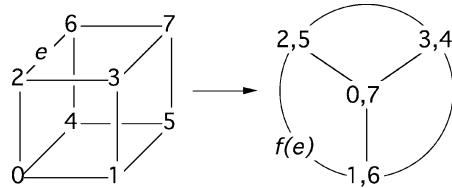
**Remark:** The load and the congestion of an isomorphism are both equal to 1.

**Example 10.6.2:** The linear graph mapping  $f : Q_3 \rightarrow P_4$  introduced in the previous section (Example 10.5.2) is shown in Figure 10.6.3. This graph mapping has load 1 at vertices  $v_0$  and  $v_3$  and load 3 at vertices  $v_1$  and  $v_2$ . Thus, the load of  $f$  is equal to 3. There is congestion 3 on edges  $e_1$  and  $e_3$  and congestion 6 on edge  $e_2$ . Thus, the mapping  $f$  has congestion 6.



**Figure 10.6.3** A linear graph mapping with load 3 and congestion 6.

**Example 10.6.3:** There are many possible mappings of the cube graph  $Q_3$  to the complete graph  $K_4$ . The vertex labels on  $K_4$  in Figure 10.6.4 specify the vertex images of one such mapping. Each edge  $e$  of  $Q_3$  is mapped to the edge of  $K_4$  whose endpoints are the images of the endpoints of  $e$ .



**Figure 10.6.4** A linear graph mapping from  $Q_3$  onto  $K_4$ .

Thus, the load of this linear graph mapping from  $Q_3$  to  $K_4$  equals 2, since two vertices of the guest  $Q_3$  are mapped to every vertex of the host  $K_4$ . Also, it has congestion equal to 2, since two edges of the guest are mapped to every edge of the host.

**Proposition 10.6.1.** Let  $f : G \rightarrow H$  be any linear graph mapping from a guest graph  $G$  to a host graph  $H$ . Then the load of  $f$  is at least  $\lceil |V_G|/|V_H| \rceil$ , and the congestion of  $f$  is at least  $\lceil |E_G|/|E_H| \rceil$ .

**Proof:** In the terminology of the generalized pigeonhole principle, the vertex function  $f_V$  has  $|V_G|$  pigeons and  $|V_H|$  pigeonholes, and the edge function  $f_E$  has  $|E_G|$  pigeons and  $|E_H|$  pigeonholes.  $\diamond$

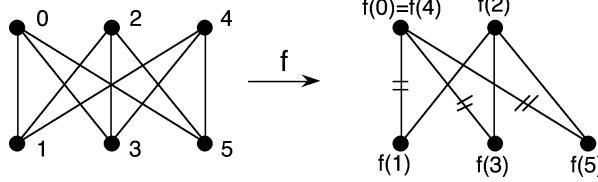
**Proposition 10.6.2.** Let  $f : G \rightarrow H$  be a linear graph mapping, and let  $e$  be a proper edge of guest  $G$  such that both endpoints of  $e$  are mapped to the same vertex of the host  $H$ . Then  $f(e)$  is a self-loop at that host vertex.

**Proof:** This is an immediate consequence of the incidence-preservation property of a linear mapping.  $\diamond$

**Corollary 10.6.3.** Let  $f : K_{m,n} \rightarrow H$  be a linear graph mapping to a graph with no self-loops, and let  $u$  and  $v$  be vertices in different parts of the bipartition of  $K_{m,n}$ . Then  $f(u) \neq f(v)$ .

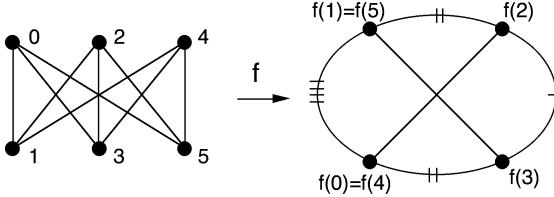
**Proof:** This follows from the contrapositive of Proposition 10.6.2.

**Example 10.6.4:** *Minimizing Load and Congestion:* Consider the bipartite graph  $K_{3,3}$ , with 6 vertices and 9 edges, as guest and the bipartite graph  $K_{2,3}$ , with 5 vertices and 6 edges, as host. Then the load of any linear graph mapping is at least  $\lceil 6/5 \rceil = 2$ , and the congestion is at least  $\lceil 9/6 \rceil = 2$ , by Proposition 10.6.1. Figure 10.6.5 depicts a linear graph mapping that achieves these lower bounds. The number of strikes on an edge of the host indicates its congestion.



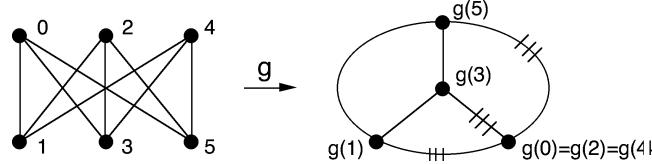
**Figure 10.6.5** A linear graph mapping  $f : K_{3,3} \rightarrow K_{2,3}$  with load 2 and congestion 2.

**Example 10.6.5:** *Minimizing Load and Congestion:* Consider the bipartite graph  $K_{3,3}$ , with 6 vertices and 9 edges, as guest and the complete graph  $K_4$ , with 4 vertices and 6 edges, as host. Then the load of any linear graph mapping is at least  $\lceil 6/4 \rceil = 2$ , and the congestion is at least  $\lceil 9/6 \rceil = 2$ . The lower bound on load is realized by the mapping  $f$  in Figure 10.6.6.



**Figure 10.6.6** A linear graph mapping  $f : K_{3,3} \rightarrow K_4$  with load 2 and congestion 4.

Figure 10.6.7 shows that congestion 3 is achievable by a linear graph mapping  $g$  with load 3.



**Figure 10.6.7** A linear graph mapping  $g : K_{3,3} \rightarrow K_4$  with load 3 and congestion 3.

The following proposition shows that 3 is the minimum possible congestion for a linear graph mapping from  $K_{3,3}$  to  $K_4$ .

**Proposition 10.6.4.** *The congestion of a linear graph mapping  $f : K_{3,3} \rightarrow K_4$  is at least 3.*

**Proof:** Suppose that all three vertices in one part of the bipartition of  $K_{3,3}$  are mapped to the same vertex  $v$  of  $K_4$ . Then, by incidence preservation, all nine edges of  $K_{3,3}$  are mapped to the three edges of  $K_4$  terminating at  $v$ . Thus, at least one of these three edges of  $K_4$  carries congestion at least 3.

If neither part of the bipartition of  $K_{3,3}$  is mapped to a single vertex of  $K_4$ , then Corollary 10.6.3 implies that one vertex  $b$  of  $K_4$  is the image of two vertices  $u, v$  from one part of  $K_{3,3}$ , and another vertex  $c$  is the image of two vertices  $x, y$  from the other part. This implies that the edge between vertices  $b$  and  $c$  has congestion 4, since all four edges between the pairs  $u, v$  and  $x, y$  must be mapped to the edge between  $a$  and  $b$ .  $\diamond$

### Semilinear Graph Mappings

In an interconnection network, a processor at a 2-valent vertex can be regarded as a relay station, since whatever other computation it may have performed could just as well have been accomplished by the next processor. This insight motivated the introduction of a more general form of graph mapping, under which, in effect, an edge of the guest (domain) graph can be mapped to a path in the host (codomain). In a network emulation, the intermediate vertices along that path simply relay the information to the next node. This *dilating* of an edge into a path introduces an additional source of delay in the emulation, but it sometimes compensates for this cost by facilitating the construction of mappings that would otherwise be impossible.

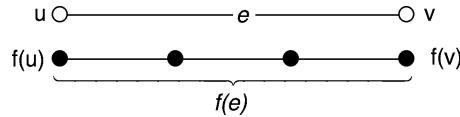
NOTATION:  $\mathcal{P}_H$  denotes the set of all paths in a graph  $H$ .

DEFINITION: A **semilinear graph mapping**  $f : G \rightarrow H$  is a pair of functions

$$f_V : V_G \rightarrow V_H \text{ and } f_E : E_G \rightarrow \mathcal{P}_H$$

such that for every edge  $e \in E_G$ , the endpoints of  $e$  are mapped by  $f_V$  to the first and last vertices of the path  $f_E(e)$ . The subscripts  $V$  and  $E$  on  $f$  are omitted when no ambiguity can result.

**Example 10.6.6:** Figure 10.6.8 illustrates a semilinear mapping from  $P_2$  to  $P_4$ , in which the only edge of  $P_2$  is mapped to a path that spans  $P_4$ . The two vertices of  $P_2$  are mapped to the first and last vertices of  $P_4$ . One may think of the single edge of the domain graph  $P_2$  as being stretched into a path of length 3 in the codomain  $P_4$ .



**Figure 10.6.8** A length-1 path semilinearly mapped to a length-3 path.

DEFINITION: The **dilation of an edge** in the guest graph  $G$  under a semilinear mapping  $f : G \rightarrow H$  is the length of the path onto which it is mapped.

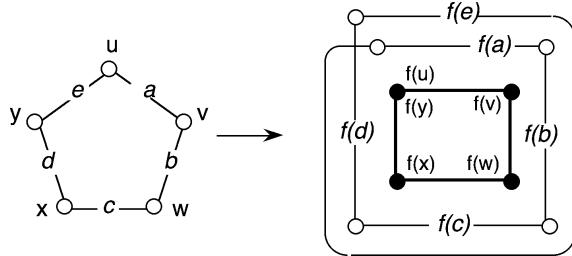
DEFINITION: The **dilation of a semilinear mapping**  $f : G \rightarrow H$  is the maximum dilation of any edge in the guest graph  $G$ .

**Proposition 10.6.5.** A linear mapping is a semilinear mapping with dilation 1.

**Proof:** This is an immediate consequence of the definitions.  $\diamond$

The following example demonstrates that dilation of edges permits the existence of semilinear mappings between graphs where no linear mappings are possible.

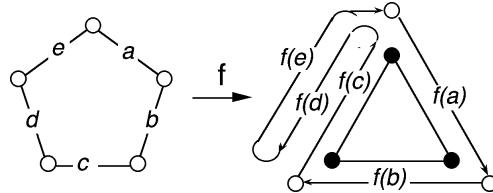
**Example 10.6.7:** There is no linear mapping  $C_5 \rightarrow C_4$ , since the host  $C_4$  is bipartite but the guest  $C_5$  is not. Having a semilinear mapping, as illustrated in Figure 10.6.9, is regarded as much better than having no mapping. Each of the edges  $a, b, c$ , and  $d$  is mapped to one edge of the cycle. Edge  $e$  is stretched so that it wraps around the entire 4-cycle, exactly as shown. Thus, edge  $e$  has dilation 4, and the semilinear mapping  $f$  has dilation 4.



**Figure 10.6.9** A 5-cycle (with one dilated edge) doubly wrapped on a 4-cycle.

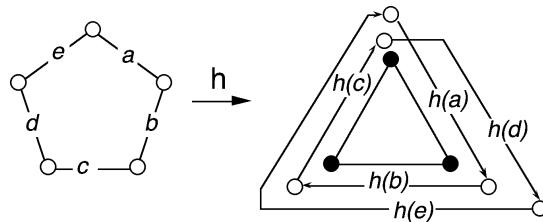
Sometimes, dilating edges permits the congestion of the resulting semilinear mapping to be lower than the least possible congestion of a linear mapping.

**Example 10.6.8:** In any linear mapping  $f : C_5 \rightarrow C_3$ , there is a 3-path in the domain  $C_5$  whose three edges are mapped to different edges of the codomain  $C_3$ . Moreover, the other two edges of  $C_5$  are mapped to the same edge of  $C_3$ , as illustrated in Figure 10.6.10. (Observe that the more natural wraparound — with  $f(d) = f(a)$  and  $f(e) = f(b)$  — is not linear.) Thus, the smallest possible congestion for a linear mapping  $C_5 \rightarrow C_3$  is 3.



**Figure 10.6.10** A linear graph mapping  $f : C_5 \rightarrow C_3$  with congestion 3.

However, this congestion can be reduced to 2 with a semilinear mapping  $h : C_5 \rightarrow C_3$ , illustrated in Figure 10.6.11, in which one of the edges is dilated so that  $C_5$  can be wrapped twofold around  $C_3$ .



**Figure 10.6.11** A semilinear graph mapping  $h : C_5 \rightarrow C_3$  with congestion 2.

**Proposition 10.6.6.** Given any graph  $G$  and any connected graph  $H$ , there is a semilinear mapping  $G \rightarrow H$ .

**Proof:** Start with any vertex function  $f : V_G \rightarrow V_H$ . Given an edge  $e \in E_G$  with endpoints  $u$  and  $v$ , define  $f(e)$  to be any path between  $f(u)$  and  $f(v)$  in  $H$ . At least one such path exists, because  $H$  is connected.  $\diamond$

### Bandwidth From the Perspective of Graph Mappings

In §10.3, we defined bandwidth in terms of vertex labelings and their corresponding adjacency matrices. We now reintroduce the concept from the perspective of graph mappings.

**DEFINITION:** The **bandwidth of a graph  $G$**  is the minimum dilation of any vertex-bijective semilinear mapping of  $G$  to a path graph.

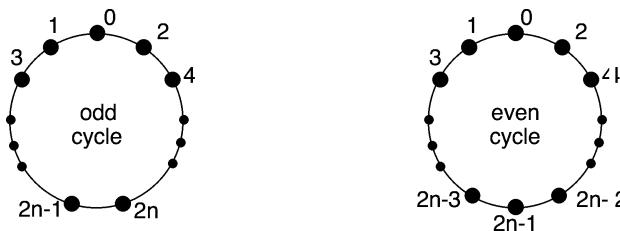
**Remark:** Any vertex-bijective semilinear mapping  $f : G \rightarrow P_n$  of an  $n$ -vertex graph  $G$  to the  $n$ -vertex path  $P_n$  can be specified by a standard 0-based integer labeling of  $V_G$ . The dilation of the mapping  $f : G \rightarrow P_n$  will then be equal to the maximum difference between the labels on the endpoints of an edge, taken over all edges. Thus, the graph-mapping and the numbering-of-a-graph perspectives of bandwidth do indeed coincide.

This suggests the following approach to calculating the bandwidth of a graph with a high level of symmetry. For each of the essentially different (up to symmetry) standard vertex labelings of the given graph, calculate the dilation, as described above. The minimum of these dilations is the bandwidth of the given graph.

**Example 10.6.9:** *Bandwidth of the complete graph  $K_n$*  No matter how the labels  $0, \dots, n - 1$  are assigned to the vertices of  $K_n$ , the numbers 0 and  $n - 1$  will be adjacent. Thus, the bandwidth is  $n - 1$ .

**Example 10.6.10:** *Bandwidth of the path  $P_n$*  Assigning the labels  $0, \dots, n$  in sequence to the vertices of  $P_n$  corresponds to a mapping of dilation 1. Thus, the bandwidth of  $P_n$  is equal to 1.

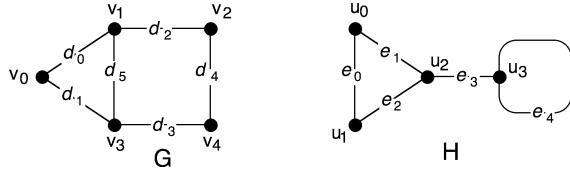
**Example 10.6.11:** *Bandwidth of the cycle  $C_n$*  Since the cycle  $C_n$  has  $n$  edges, and since there are only  $n - 1$  consecutive pairs among the numbers  $0, \dots, n - 1$ , it follows that some adjacent pair of vertices of  $C_n$  must be labeled with non-consecutive numbers. Hence, the bandwidth of  $C_n$  is at least 2. This lower bound is realizable, as illustrated in Figure 10.6.12, by first assigning ascending even integers to adjacent vertices, starting from 0 anywhere on  $C_n$ , and then finishing with descending odd numbers.



**Figure 10.6.12** A vertex-bijective labeling of  $C_n$  with dilation 2.

**EXERCISES for Section 10.6**

For Exercises 10.6.1 through 10.6.4, specify a linear mapping from the graph  $G$  to the graph  $H$  that has the prescribed load and congestion.



- 10.6.1<sup>s</sup> Load 2 and congestion 2.  
10.6.3 Load 2 and congestion 4.

- 10.6.2 Load 2 and congestion 3.  
10.6.4 Load 4 and congestion 4.

For Exercises 10.6.5 through 10.6.8, specify a linear mapping from the given domain graph to the given codomain graph with the prescribed load and congestion.

- 10.6.5<sup>s</sup> From  $Q_3$  to  $C_4$  with load equal to 2 and congestion equal to 3.  
10.6.6 From  $Q_3$  to the dipole  $D_4$  with load equal to 4 and congestion equal to 3.  
10.6.7 From  $Q_3$  to  $K_4$  with load equal to 2 and congestion equal to 2.  
10.6.8 From  $CL_6$  to  $K_{3,3}$  with load equal to 2 and congestion equal to 2.

For Exercises 10.6.9 through 10.6.12, specify a linear mapping from the given domain graph to the given codomain graph with the minimum possible load.

- 10.6.9<sup>s</sup>  $K_{8,4} \rightarrow D_4$ .  
10.6.11  $C_6 \rightarrow K_4 - K_2$ .

- 10.6.10  $K_{m,n} \rightarrow C_4$ .  
10.6.12  $CL_4 \rightarrow C_4$ .

For Exercises 10.6.13 through 10.6.16, specify a linear mapping from the given domain graph to the given codomain graph with the minimum possible congestion.

- 10.6.13<sup>s</sup>  $K_{8,4} \rightarrow D_4$ .  
10.6.15  $C_6 \rightarrow K_4 - K_2$ .

- 10.6.14  $K_{m,n} \rightarrow C_4$ .  
10.6.16  $CL_4 \rightarrow C_4$ .

- 10.6.17 The **dumbbell graph** has two vertices and one proper edge joining them, plus a self-loop at each vertex, as shown. Consider the linear mappings from the complete graph  $K_4$ .



What are the lower bounds, according to Proposition 10.6.1, for load and for congestion? Prove that the lower bound for congestion cannot be achieved. (Hint: The vertex mapping completely determines the edge mapping.)

- 10.6.18 Do the previous exercise for the general case in which the guest is  $K_n$ , with  $n \geq 3$ .

NOTE: Some of the bandwidth exercises from §10.3 are repeated below for convenience.

- 10.6.19<sup>s</sup> Prove that the bandwidth of  $K_{1,n}$  equals  $\lceil \frac{n}{2} \rceil$ .

- 10.6.20 Prove that the bandwidth of  $K_{2,4}$  equals 3.

- 10.6.21 To prove that the cube  $Q_3$  has bandwidth equal to 4, first show that 4 is a lower bound, and then exhibit a labeling that realizes that bound.

- 10.6.22 To prove that the wheel  $W_{2n-1}$  has bandwidth equal to  $n$ , first show that  $n$  is a lower bound, and then exhibit a labeling that realizes that bound. (See Exercise 10.3.8.)

For Exercises 10.6.23 through 10.6.26, calculate the bandwidth of the given graph.

- 10.6.23<sup>s</sup> Circular ladder  $CL_3$ .  
 10.6.25  $K_{m,n}$ , for  $m, n \geq 2$ .

- 10.6.24  $K_n \times K_2$ .  
 10.6.26  $K_n \times K_m$ .

- 10.6.27 Calculate the bandwidth of the graph obtained by amalgamating two 3-cycles at a vertex.

## 10.7 SUPPLEMENTARY EXERCISES

10.7.1 Construct a graph whose center and median subgraph have no vertices in common, or argue why no such graph exists.

10.7.2 Construct a graph whose periphery and median subgraph have a vertex in common, or argue why no such graph exists.

10.7.3 Let  $G$  be a connected graph and  $s$  an integer such that  $\text{rad}(G) \leq s \leq \text{diam}(G)$ . Prove that there is a vertex  $v$  with eccentricity  $s$ .

10.7.4 Let  $s$  and  $n$  be any two integers such that  $1 \leq s \leq \frac{n}{2}$ . Show that there exists an  $n$ -vertex connected graph  $G$  with  $\text{dom}(G) = s$ .

10.7.5 Let  $G$  be a graph with at least two components. Prove that the edge-complement graph  $\overline{G}$  has  $\text{dom}(\overline{G}) \leq 2$ .

10.7.6 Find a largest minimal dominating set of the 10-vertex path graph  $P_{10}$ .

10.7.7 Prove or disprove: For every tree, the domination number equals the cardinality of a minimum vertex cover.

10.7.8 Determine the bandwidth of the tripartite graph  $K_{2,3,5}$ .

10.7.9 Let  $G = G_1 \cup G_2 \cup \dots \cup G_t$  be the disjoint union of  $t$  graphs. Express the bandwidth  $bw(G)$  in terms of the bandwidths  $bw(G_1), bw(G_2), \dots, bw(G_t)$ .

10.7.10 Show that for any positive integer  $m$ , there exists a graph whose intersection number equals  $m$ .

10.7.11 Draw a 6-vertex, 12-edge simple graph with bandwidth = 3, or prove that none exists.

10.7.12 Draw a 3-regular, 8-vertex simple graph with bandwidth = 3.

10.7.13 Calculate the bandwidth of the Petersen graph.

10.7.14 Prove that 5 is the bandwidth of the wheel  $W_9$ .

10.7.15 Calculate  $bw(K_{4,8})$ . (Hint: consider diameter.)

For Exercises 10.7.16 through 10.7.27, calculate the bandwidth of the given graph.

- |                     |                       |
|---------------------|-----------------------|
| 10.7.16 $CL_4$ .    | 10.7.17 $CL_5$ .      |
| 10.7.18 $CL_{2n}$ . | 10.7.19 $CL_{2n+1}$ . |
| 10.7.20 $ML_4$ .    | 10.7.21 $ML_5$ .      |
| 10.7.22 $ML_{2n}$ . | 10.7.23 $ML_{2n+1}$ . |
| 10.7.24 $W_4$ .     | 10.7.25 $W_5$ .       |
| 10.7.26 $W_{2n}$ .  | 10.7.27 $W_{2n+1}$ .  |

- 10.7.28 Prove that any graph mapping from  $K_{3,3}$  to  $P_3$  has congestion at least 5.

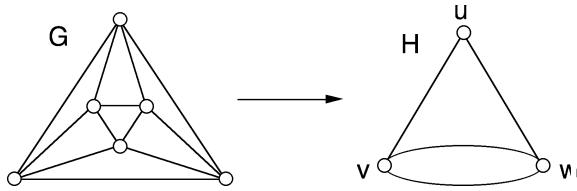
**10.7.29** Let  $f : G \rightarrow H$  be a linear graph mapping onto graph  $H$ , and let  $u, v \in V_G$ . Prove that the distance between  $f(u)$  and  $f(v)$  is less than or equal to the distance between  $u$  and  $v$ .

**10.7.30** Let  $f : G \rightarrow H$  be a linear graph mapping onto graph  $H$ . Prove that the diameter of  $H$  is less than or equal to the diameter of  $G$ .

**10.7.31** Label the vertices of  $K_{4,8}$  with numbers 0 to 7 so as to indicate a linear mapping to  $C_8$ . This provides an upper bound on congestion of such a mapping. Calculate the congestion of your mapping.

**10.7.32** Calculate a lower bound on the congestion of a linear mapping from  $K_{4,8}$  to  $C_8$ . (Hint: consider diameter.)

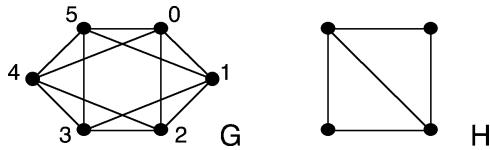
**10.7.33** Label each vertex of the graph  $G$  in Figure 10.7.1 with the name of the vertex of the graph  $H$  to which it is linearly mapped, so that the resulting mapping has load = 2. Prove that all linear graph mappings from  $G$  to  $H$  have load = 2.



**Figure 10.7.1**

**10.7.34** Calculate the minimum possible load for a linear mapping from graph  $G$  to graph  $H$  in Figure 10.7.2.

**10.7.35** Calculate the minimum possible congestion for a linear mapping from graph  $G$  to graph  $H$  in Figure 10.7.2.



**Figure 10.7.2**

## GLOSSARY

**adjacency-preserving** vertex mapping: see *linear graph mapping* between two *simple* graphs.

**bandwidth of a numbering**  $f$  of a graph  $G$ , denoted  $bw_f(G)$ : the quantity given by  $bw_f(G) = \max\{|f(u) - f(v)| : uv \in E_G\}$ .

**bandwidth numbering of a graph**  $G$ : a numbering  $f$  that achieves  $bw(G)$ , i.e.,  $bw_f(G) = bw(G)$ .

**bandwidth of a graph**  $G$ , denoted  $bw(G)$ : the quantity prescribed by the rule  $bw(G) = \min\{bw_f(G) : f \text{ is a numbering of } G\}$ ; the notations  $ban(G)$  and  $B(G)$  are also used for the bandwidth of  $G$ .

**bandwidth of a graph** from the perspective of graph mappings: the minimum dilation of any vertex-bijective semilinear mapping of that graph to a path graph; equivalent to the number-of-a-graph perspective given above (see §10.6).

**block of a graph  $G$** : a maximal connected subgraph that contains no cut-vertices of  $G$ .

**center of a graph  $G$** , denoted  $Z(G)$ : the subgraph induced on the set of central vertices of  $G$ .

**central vertex  $v$**  of a graph  $G$ : a vertex with minimum eccentricity, i.e., such that  $\text{ecc}(v) = \text{rad}(G)$ .

**chordal graph**: a graph  $G$  such that for all  $n \geq 4$ ,  $G$  does not contain an  $n$ -vertex cycle graph  $C_n$  as an induced subgraph.

**chromatic number of graph  $G$** , denoted  $\chi(G)$ : the smallest number  $k$  such that there is a function  $g : V(G) \rightarrow \{1, 2, \dots, k\}$  with the property that, if  $uv \in E_G$ , then  $g(u) \neq g(v)$ .

**circumference of a graph  $G$**  with at least one cycle: the length of a longest cycle in  $G$ ; denoted  $\text{circum}(G)$ . The circumference of an acyclic graph is undefined.

**closed interval for two vertices  $u, v$**  of a connected graph  $G$ : denoted  $I[u, v]$ ; the union of the sets of vertices of  $G$  that lie on  $u$ - $v$  geodesics.

**closed interval for a vertex subset  $S$**  of a connected graph  $G$ : denoted  $I[S]$ ; the union  $\bigcup_{u, v \in S} I[u, v]$ .

**competition graph of a digraph  $D$** : the intersection graph of the family of out-sets of the vertices of  $D$ .

**competition graph**: a graph  $G$  that is (isomorphic to) the competition graph of some digraph  $D$ .

—,  $\phi$ -tolerance: a special kind of *tolerance graph* (see §10.4).

**congestion on an edge** of the codomain of a graph mapping: the number of edges mapped to that edge.

**congestion of a graph mapping**: the maximum congestion on any edge of the codomain.

**convex hull of a vertex subset  $S$**  in a graph  $G$ : the smallest convex set that contains the subset  $S$ .

**convex vertex subset  $S$**  in a graph  $G$ : a vertex subset  $S$  such that  $I[S] = S$ .

**diameter of a graph  $G$** , denoted  $\text{diam}(G)$ : the maximum of the vertex eccentricities in  $G$  or, equivalently, the maximum distance between two vertices in  $G$ , i.e.,  $\text{diam}(G) = \max_{x \in V_G} \{\text{ecc}(x)\} = \max_{x, y \in V_G} \{d(x, y)\}$ .

**dilation of an edge** in the domain of a semilinear graph mapping: the length of the path onto which that edge is mapped.

**dilation of a semilinear mapping**: the maximum dilation of any edge in the domain graph.

**distance  $d(s, t)$**  from a vertex  $s$  to a vertex  $t$  in a graph  $G$ : the length of a shortest  $s$ - $t$  path if one exists; otherwise,  $d(s, t) = \infty$ .

**dominates**: what a vertex does to itself and to each of its neighbors.

**dominating set** in a graph  $G$ : a vertex subset  $D \subseteq V_G$  such that every vertex of  $G$  is in  $D$  or is adjacent to at least one vertex in  $D$ .

—, **connected**: a dominating set  $D$  such that the subgraph induced on  $D$  is connected.

—, **distance- $k$**  for an integer  $k \geq 1$ : vertex subset  $D$  such that for all  $v \in V_G - D$ , there exists  $x \in D$  such that  $d(v, x) \leq k$ .

—, **independent**: an independent set of vertices that is also a dominating set of  $G$ .  
 —, **minimal**: a dominating set such that every proper subset is non-dominating.

**domination number** of a graph  $G$ : the cardinality of a minimum dominating set of  $G$ ; denoted  $\text{dom}(G)$  (elsewhere, often  $\gamma(G)$ ).

—, **connected**: denoted  $c\text{-dom}(G)$ ; the cardinality of a minimum connected dominating set of  $G$ .

—, **distance- $k$** : denoted  $d_k\text{-dom}(G)$ ; the cardinality of a minimum distance- $k$  dominating set of  $G$ .

—, **independent**: denoted  $i\text{-dom}(G)$ ; the cardinality of a minimum independent dominating set of  $G$ .

**dual edge clique cover** of a set representation  $\mathcal{F}$  for an intersection graph  $G$ : the edge clique cover  $\mathcal{K}^{\mathcal{F}}$  defined in Proposition 10.4.7.

**dual set representation** of an edge clique cover  $\mathcal{K}$  of a graph  $G$ : the set representation  $\mathcal{F}^{\mathcal{K}}$  defined in Proposition 10.4.8.

**dumbbell graph**: the graph that has two vertices and one proper edge joining them, plus a self-loop at each vertex.

**eccentricity of a vertex**  $v$  in a graph  $G$ , denoted  $\text{ecc}(v)$ : the distance from  $v$  to a vertex farthest from  $v$ , i.e.,  $\text{ecc}(v) = \max_{x \in V_G} \{d(v, x)\}$ .

**edge clique cover** of a graph  $G$ : a family  $\mathcal{K} = \{Q_1, Q_2, \dots, Q_t\}$  of complete subgraphs of  $G$  such that every edge of  $G$  is an edge of at least one of the  $Q_i$ 's, i.e.,  $E_G = E_{Q_1} \cup E_{Q_2} \cup \dots \cup E_{Q_t}$ .

**food web**: a digraph model  $D$  of the predator-prey relationship among the  $n$  animals  $\{a_1, a_2, \dots, a_n\}$  in an ecosystem, where the vertex- and arc-sets of  $D$  are given by

$$V_D = \{a_1, a_2, \dots, a_n\}$$

$$A_D = \{(a_i, a_j) \mid a_i \text{ preys on } a_j\}$$

**geodesic** between vertices  $u$  and  $v$ : a  $u$ - $v$  path of minimum length.

**girth of a graph**  $G$  with at least one cycle: the length of a shortest cycle in  $G$ ; denoted  $\text{girth}(G)$ . The girth of an acyclic graph is undefined.

**graph homomorphism**: synonym for *linear graph mapping*.

**guest**, in the context of network emulation: the domain graph of the graph mapping that models the emulation.

**host**, in the context of network emulation: the codomain graph of the graph mapping that models the emulation.

**image of a graph**  $G$  under a linear mapping  $f : G \rightarrow H$ : the subgraph of  $H$  whose vertex-set is  $f(V_G)$  and whose edge-set is  $f(E_G)$ .

**incidence-preservation property**: see *linear graph mapping*.

**independence number** of a graph  $G$ , denoted  $\alpha(G)$ : the cardinality of a largest independent set of  $G$ .

**independent (vertex) set** in a graph  $G$ : a vertex subset  $S$  such that no two vertices in  $S$  are adjacent.

**in-set  $In(v)$  of a vertex  $v$**  in a digraph  $D$  with vertex-set  $V_D$  and arc-set  $A_D$ : the vertex subset  $In(v) = \{w \in V_D \mid (w, v) \in A_D\}$ .

**intersection graph of a family of subsets  $\mathcal{F}$**  =  $\{S_1, S_2, \dots, S_n\}$ , denoted  $\Omega(\mathcal{F})$ : the graph whose vertex- and edge-sets are given by

$$V_{\Omega(\mathcal{F})} = \{S_1, S_2, \dots, S_n\}$$

$$E_{\Omega(\mathcal{F})} = \{S_i S_j \mid i \neq j \text{ and } S_i \cap S_j \neq \emptyset\}$$

—, **p-** of a family of subsets  $\mathcal{F} = \{S_1, S_2, \dots, S_n\}$  of a *finite* set  $S$ : denoted  $\Omega_p(\mathcal{F})$ ; the graph whose vertex- and edge-sets are given by

$$V_{\Omega_p(\mathcal{F})} = \{S_1, S_2, \dots, S_n\}$$

$$E_{\Omega_p(\mathcal{F})} = \{S_i S_j \mid i \neq j \text{ and } |S_i \cap S_j| \geq p\}$$

**intersection graph:** a graph  $G$  with vertex-set  $V_G = \{v_1, v_2, \dots, v_n\}$  such that there exists a family of sets  $\mathcal{F} = \{S_1, S_2, \dots, S_n\}$  such that  $G \cong \Omega(\mathcal{F})$  with the natural correspondence  $f(v_i) = S_i$ ,  $i = 1, 2, \dots, n$ .

—, **abdiff-tolerance**: a special kind of *tolerance graph* (see §10.4).

—, **min-tolerance**: a special kind of *tolerance graph* (see §10.4).

—, **p-**: a graph  $G$  such that there exists a family of sets  $\mathcal{F} = \{S_1, S_2, \dots, S_n\}$  such that  $G \cong \Omega_p(\mathcal{F})$ .

**intersection number of a graph  $G$** , denoted  $\text{int}(G)$ : the minimum cardinality of a set  $S$  such that  $G$  is the intersection graph of a family of subsets of  $S$ .

**interval graph**: a graph  $G$  that is an intersection graph of a family of intervals on the real line.

—, **min-tolerance**: a generalization of an interval graph (see §10.4).

**isomorphism** between two **simple** graphs  $G$  and  $H$ : a structure-preserving vertex bijection  $f : V_G \rightarrow V_H$ .

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

**linear graph mapping**  $f : G \rightarrow H$  between two (possibly **non-simple**) graphs  $G$  and  $H$  is a pair of functions  $f_V : V_G \rightarrow V_H$  and  $f_E : E_G \rightarrow E_H$  such that the *incidence-preservation property* holds: for every edge  $e \in E_G$ , the endpoint set of  $e$  is mapped by  $f_V$  to the endpoint set of the edge  $f_E(e)$ .

**linear graph mapping**  $f : G \rightarrow H$  between two **simple** graphs  $G$  and  $H$ : a pair of functions,  $f_V : V_G \rightarrow V_H$  and  $f_E : E_G \rightarrow E_H$ , such that  $f_V$  is an adjacency-preserving vertex mapping and  $f_E$  is the induced edge mapping. That is,  $u$  and  $v$  are adjacent in  $G \implies f_V(u)$  and  $f_V(v)$  are adjacent in  $H$  and  $f_E(uv) \equiv f_V(u)f_V(v)$ .

**load of a graph mapping**: the maximum load on any vertex of the codomain.

**load on a vertex** of the codomain of a graph mapping: the number of vertices mapped to that vertex.

**median vertex** in a connected graph: a vertex whose total distance is minimum.

**median subgraph**  $M(G)$  of a connected graph  $G$ : the subgraph induced on the median vertices of  $G$ .

**metric** on the vertex-set  $V_G$  of a graph  $G$ : a mapping  $d : V_G \times V_G \rightarrow \mathbb{R}$  that satisfies the following four conditions:

- $d(x, y) \geq 0$  for all  $x, y \in V_G$ .
- $d(x, y) = 0$  if and only if  $x = y$ .
- $d(x, y) = d(y, x)$  for all  $x, y \in V_G$ . (*symmetry property*)
- $d(x, y) + d(y, z) \geq d(x, z)$  for all  $x, y, z \in V_G$ . (*triangle inequality*)

**multi-edge amalgamation** corresponding to a subset  $A = \{e_1, e_2, \dots, e_s\} \subseteq E_G$  of edges having the same endpoints: the graph  $G_A$  that results from merging (amalgamating) all of the edges in  $A$  into a single edge generically denoted  $e_1e_2 \cdots e_s$ . (See §10.5.)

—, **complete**: the graph that results from taking each multi-edge and merging its edges into a single edge.

**neighborhood of a vertex**  $v$  in a graph  $G$ :

—, **open**: denoted  $N(v)$ ; the set of all the neighbors of vertex  $v$ .

—, **closed**: denoted  $N[v]$ ; the set containing vertex  $v$  and all its neighbors, i.e.,  $N[v] = N(v) \cup \{v\}$ .

**numbering**: see *vertex-labeling*.

**out-set**  $Out(v)$  of a vertex  $v$  in a digraph  $D$  with vertex-set  $V_D$  and arc-set  $A_D$ : the vertex subset  $Out(v) = \{w \in V_D \mid (v, w) \in A_D\}$ .

**path representation of a graph**  $G$ : a family  $\mathcal{F} = \{P_1, P_2, \dots, P_n\}$  of subpaths of a path  $P$  such that  $G$  is the intersection graph of  $\mathcal{F}$ .

**peripheral vertex**  $v$  of a graph  $G$ : a vertex with maximum eccentricity, i.e.,  $ecc(v) = diam(G)$ .

**periphery of a graph**  $G$ , denoted  $per(G)$ : the subgraph induced on the set of peripheral vertices of  $G$ .

**$m^{th}$  power of graph**  $G$ : the graph  $G^m$  having vertex-set  $V_{G^m} = V_G$  and edge-set  $E_{G^m} = \{uv \mid d_G(u, v) \leq m\}$ .

**private neighbor of a vertex**  $v$  relative to a dominating set  $D$  of a graph  $G$ : a vertex  $w \in V_G$  such that  $v$  is the only vertex in  $D$  that dominates  $w$ . That is,  $N[w] \cap D = \{v\}$ .

**pseudopath of length**  $n$ : a spanning path of length  $n$  with one or more self-loops at one or more of its vertices.

**radius of a graph**  $G$ , denoted  $rad(G)$ : the minimum of the vertex eccentricities, i.e.,  $rad(G) = \min_{x \in V_G} \{ecc(x)\}$ .

**set representation of a graph**  $G$ : A family of subsets  $\mathcal{F}$  such that  $G$  is the intersection graph of  $\mathcal{F}$ .

**Steiner distance of a vertex subset**  $U$  in a connected graph  $G$ : the number of edges in a Steiner tree for  $U$ ; denoted  $sd(U)$ .

**Steiner tree** for a vertex subset  $U$  in a connected graph  $G$ : a smallest tree subgraph of  $G$  that contains all the vertices of  $U$ .

**structure-preserving vertex bijection** between two **simple** graphs  $G$  and  $H$ : a vertex bijection  $f : V_G \rightarrow V_H$  that preserves adjacency and non-adjacency, i.e., for every pair of vertices in  $G$ ,

$$u \text{ and } v \text{ are adjacent in } G \iff f(u) \text{ and } f(v) \text{ are adjacent in } H$$

**subtree graph**: a graph  $G$  that is the intersection graph of a family  $\mathcal{F}$  of subtrees  $\{T_1, T_2, \dots, T_n\}$  of a tree  $T$ , i.e., the vertex-sets of the  $T_i$ 's form a *set representation* of  $G$ . The tree  $T$  and the family  $\mathcal{F}$  are called a **tree representation** of  $G$ .

**symmetry property of a metric** on the vertex-set of a graph: see *metric*.

**total distance of a vertex**  $v$  in a connected graph  $G$ , denoted  $td(v)$ : the quantity  $td(v) = \sum_{w \in V_G} d(v, w)$ .

**tolerance graph**: a generalization of intersection graphs (see §10.4).

**tree representation of a graph**  $G$ : a family  $\mathcal{F} = \{T_1, T_2, \dots, T_n\}$  of subtrees of a tree  $T$  such that  $G$  is the intersection graph of  $\mathcal{F}$ .

**triangle property of a metric** on the vertex-set of a graph: see *metric*.

**vertex-connectivity of a connected graph**  $G$ , denoted  $\kappa_v(G)$ : the minimum number of vertices whose removal can either disconnect  $G$  or reduce it to a 1-vertex graph.

**vertex-cut of a connected graph**  $G$ : a vertex subset  $S$  such that  $G - S$  is non-connected.

**vertex-labeling, (1-based) standard**, of an  $n$ -vertex graph  $G$ : a bijection  $f : V_G \rightarrow \{1, 2, \dots, n\}$ .

**vertex-partition amalgamation** of a graph  $G$  corresponding to a partition  $\pi = \{V_1, V_2, \dots, V_t\}$  of its vertex-set  $V_G$ : the transformation of  $G$  into a graph  $G/\pi$  that results from merging (amalgamating) all of the vertices in each cell of the partition (see §10.5).

# Chapter 11

---

## ANALYTIC GRAPH THEORY

**11.1 Ramsey Graph Theory**

**11.2 Extremal Graph Theory**

**11.3 Random Graphs**

---

### INTRODUCTION

Several areas of graph theory are concerned with the likelihood or certainty of the presence in a graph of various subgraphs or, more generally, of graph properties that emerge as the number of vertices and/or the number of edges increases. Collectively they are grouped as *analytic graph theory*.

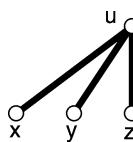
The foundational results in analytic graph theory were obtained decades ago, and there has been extensive subsequent development. Even though many such results can be stated easily, they can be quite difficult to prove. As an introduction to this deep and well-developed area, this chapter, like the others in this book, concentrates on the basic results whose proofs are the most readily accessible and illuminating.

## 11.1 RAMSEY THEORY

A theorem of the British logician Frank Ramsey in 1930 launched what is now called Ramsey theory. The classical Ramsey puzzle, a simple case of that theorem, is to prove that in any group of six persons, there are either three mutual acquaintances or three mutual non-acquaintances (see Exercise 2.4.38).

**Theorem 11.1.1.** *In any simple graph  $G$  with six vertices, either there are three mutually adjacent vertices or there are three mutually non-adjacent vertices. Equivalently, either the graph  $G$  or its edge-complement  $\overline{G}$  contains a  $K_3$ -subgraph.*

**Proof:** Let  $u \in V_G$ . Then either  $\deg_G(u) \geq 3$  or  $\deg_{\overline{G}}(u) \geq 3$ , since  $\deg_G(u) + \deg_{\overline{G}}(u) = 5$ . We may assume without loss of generality that  $\deg_G(u) \geq 3$ , because of the symmetry of the theorem statement with respect to  $G$  and  $\overline{G}$ ). Let  $x, y$ , and  $z$  be three of the neighbors of  $u$ , as in Figure 11.1.1.



**Figure 11.1.1** Three of the neighbors of vertex  $u$ .

If any two of vertices  $x, y$ , and  $z$  are adjacent, then those two, together with vertex  $u$ , form a  $K_3$ -subgraph of graph  $G$ . Alternatively, if  $x, y$ , and  $z$  are mutually non-adjacent, then they form a  $K_3$ -subgraph in graph  $\overline{G}$ .  $\diamond$

### Ramsey Numbers

The Ramsey puzzle is readily generalized to a vertex-extremal problem of calculating a minimum number of vertices in a simple graph that guarantees the occurrence of a property.

**DEFINITION:** Let  $s$  and  $t$  be positive integers. The **Ramsey number**  $r(s, t)$  is the minimum positive number  $n$  such that for every simple  $n$ -vertex graph  $G$ , either  $G$  contains a  $K_s$ -subgraph or its edge-complement graph  $\overline{G}$  contains a  $K_t$ -subgraph.

**Remark:** Thus, Theorem 11.1.1 establishes that  $r(3, 3) = 6$ .

**ALTERNATIVE DEFINITION:** The **Ramsey number**  $r(s, t)$  is the minimum number  $n$  such that for every edge-coloring of  $K_n$  with colors red and blue, there exists either a red  $K_s$ -subgraph or a blue  $K_t$ -subgraph.

This version of the definition generalizes to arbitrarily many edge colors and to arbitrary subgraphs in each color.

### Fundamental Properties of Ramsey Numbers

**Proposition 11.1.2.**  $r(s, t) = r(t, s)$ , for all  $s, t \geq 1$ .

**Proof:** The alternative definition is clearly symmetric in the roles of  $s$  and  $t$ .  $\diamond$

**Proposition 11.1.3.**  $r(s, 1) = 1$ , for all  $s \geq 1$ .

**Proof:** The edge-complement of a non-trivial graph always contains  $K_1$ .  $\diamond$

**Proposition 11.1.4.**  $r(s, 2) = s$ , for all  $s \geq 1$ .

**Proof:** If an  $s$ -vertex simple graph  $G$  is not complete, then its edge-complement  $\overline{G}$  contains  $K_2$ . Thus,  $r(s, 2) \leq s$ . The reverse inequality also holds, since  $K_{s-1}$  obviously does not contain a  $K_s$ -subgraph, and  $\overline{K}_{s-1}$  contains no edges.  $\diamond$

**Theorem 11.1.5 (Erdős and Szekeres, 1935).** (a) For all  $s, t \geq 3$ ,

$$r(s, t) \leq r(s-1, t) + r(s, t-1)$$

(b) Moreover, if  $r(s-1, t)$  and  $r(s, t-1)$  are both even, then

$$r(s, t) < r(s-1, t) + r(s, t-1)$$

**Proof:** This proof of part (a) generalizes the proof given for Theorem 11.1.1.

(a) Let  $G$  be a simple graph on  $r(s-1, t) + r(s, t-1)$  vertices, and let  $u \in V_G$ . Then either  $\deg_G(u) \geq r(s-1, t)$  or  $\deg_{\overline{G}}(u) \geq r(s, t-1) = r(t-1, s)$ , since  $\deg_G(u) + \deg_{\overline{G}}(u) = r(s-1, t) + r(s, t-1) - 1$ . If  $\deg_G(u) \geq r(s-1, t)$ , then the subgraph  $H$  of  $G$  induced on the open neighborhood of  $u$  has at least  $r(s-1, t)$  vertices. Thus, either subgraph  $H$  contains a  $K_{s-1}$ -subgraph, which together with vertex  $u$  forms a  $K_s$ -subgraph of  $G$ , or  $\overline{H}$  (and hence  $\overline{G}$ ) contains a  $K_t$ -subgraph. If  $\deg_{\overline{G}}(u) \geq r(t-1, s)$ , then the result follows by applying the same argument to  $\overline{G}$  with the roles of  $s$  and  $t$  reversed.

(b) Now suppose that  $r(s-1, t)$  and  $r(s, t-1)$  are both even, and that graph  $G$  has  $r(s-1, t) + r(s, t-1) - 1$  vertices. Then  $\deg_G(u) + \deg_{\overline{G}}(u) = r(s-1, t) + r(s, t-1) - 2$ . Since  $|V_G|$  is an odd number, there is some vertex  $u$  of  $G$  with even degree. If  $\deg_G(u) \geq r(s-1, t)$ , then we are done, as in part (a) above. If  $\deg_G(u) < r(s-1, t)$ , then  $\deg_G(u) \leq r(s-1, t) - 2$ , and, hence,  $\deg_{\overline{G}}(u) \geq r(t-1, s)$ . The result again follows, as in part (a).  $\diamond$

**Remark:** The next result shows that Ramsey numbers exist for all pairs of positive integers.

**Corollary 11.1.6 (Erdős and Szekeres, 1935).** For all positive integers  $s$  and  $t$ ,

$$r(s, t) \leq \binom{s+t-2}{s-1}$$

**Proof:** We use induction on the sum  $s+t$ . The assertion is trivially true for  $s+t=2$ , by Proposition 11.1.3. Assume for some  $k \geq 3$  that the inequality is true for all positive integers  $s$  and  $t$  whose sum is smaller than  $k$ , and suppose  $r+s=k$ . Then

$$\begin{aligned} r(s, t) &\leq r(s-1, t) + r(s, t-1) \quad (\text{by Theorem 11.1.5(a)}) \\ &\leq \binom{(s-1)+t-2}{(s-1)-1} + \binom{s+(t-1)-2}{s-1} \quad (\text{by the induction hypothesis}) \\ &= \binom{s+t-3}{s-2} + \binom{s+t-3}{s-1} \\ &= \binom{s+t-2}{s-1} \quad (\text{by Pascal's recursion}) \quad \diamond \end{aligned}$$

### Ramsey Number Calculations

Determining values of the Ramsey numbers  $r(s, t)$  for  $3 \leq s \leq t$  has proved to be quite difficult, except for the first few. In fact, the Ramsey numbers for only nine such pairs are known; they are shown in Table 1. Theorem 11.1.1 established  $r(3, 3) = 6$ . The next two results calculate  $r(3, 4)$  and  $r(3, 5)$ , and a third result produces an upper bound for  $r(4, 4)$ .

**Table 1.** The known Ramsey numbers  $r(s, t)$  for  $3 \leq s \leq t$ .

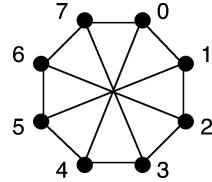
	$t = 3$	4	5	6	7	8	9
$s = 3$	6	9	14	18	23	28	36
4		18	25				

**Proposition 11.1.7.**  $r(3, 4) = 9$ .

**Proof:** Since the Ramsey numbers  $r(2, 4) = 4$  and  $r(3, 3) = 6$  are both even, Theorem 11.1.5(b) implies the upper bound

$$r(3, 4) \leq r(2, 4) + r(3, 3) - 1 = 9$$

For the reverse inequality, observe that the 8-vertex circulant graph  $G = \text{circ}(8; 1, 4)$ , shown in Figure 11.1.2, has neither a  $K_3$ -subgraph nor an independent set of four vertices (i.e.,  $\overline{G}$  has no  $K_4$ -subgraph).  $\diamond$



**Figure 11.1.2** The circulant graph  $\text{circ}(8; 1, 4)$ .

**Proposition 11.1.8.**  $r(3, 5) = 14$ .

**Proof:** The upper bound for  $r(3, 5)$  follows from Theorem 11.1.5(a) and Propositions 11.1.4 and 11.1.7. In particular,

$$r(3, 5) \leq r(2, 5) + r(3, 4) = 5 + 9 = 14$$

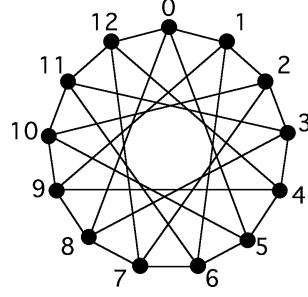
For the reverse inequality, consider the 13-vertex circulant graph  $G = \text{circ}(13; 1, 5)$ , shown in Figure 11.1.3 below.  $G$  clearly has no  $K_3$ -subgraph. If it had a set  $S$  of five mutually non-adjacent vertices, then two of them would have to be within distance two of each other. By symmetry, we may take these two vertices to be 0 and 2. This excludes vertices 1, 3, 5, 7, 8, 10, and 12, leaving only 4, 6, 9, and 11 as possible members of  $S$ . But at most one of the vertices 4 and 9 can be in  $S$ , because they are adjacent. Likewise, at most one of 6 and 11 can be in  $S$ .  $\diamond$

**Proposition 11.1.9.**  $r(4, 4) \leq 18$ .

**Proof:** By Theorem 11.1.5(a) and Propositions 11.1.2 and 11.1.7, we have

$$r(4, 4) \leq r(3, 4) + r(4, 3) = 9 + 9 = 18$$

$\diamond$



**Figure 11.1.3** The circulant graph  $\text{circ}(13; 1, 5)$ .

### Ramsey Numbers for Arbitrary Subgraphs

**DEFINITION:** For arbitrary simple graphs  $H_1$  and  $H_2$ , the **Ramsey number**  $r(H_1, H_2)$  is the minimum number  $n$  such that for every  $n$ -vertex simple graph  $G$ , either  $H_1$  is a subgraph of  $G$  or  $H_2$  is a subgraph of  $\overline{G}$ .

**Proposition 11.1.10.** *The Ramsey number  $r(H_1, H_2)$  exists for any two simple graphs  $H_1$  and  $H_2$ .*

**Proof:** Let  $H_1$  and  $H_2$  be simple graphs with  $m$  and  $n$  vertices, respectively. Since  $H_1$  is a subgraph of  $K_m$  and  $H_2$  is a subgraph of  $K_n$ , we have  $r(H_1, H_2) \leq r(m, n)$ .  $\diamond$

**Proposition 11.1.11.**

$$r(K_{1,m}, K_{1,n}) = \begin{cases} m+n & \text{if } m, n \text{ not both even} \\ m+n-1 & \text{if } m, n \text{ both even} \end{cases}$$

**Proof:** Case 1.  $m$  and  $n$  are not both even. We may assume without loss of generality that  $m$  is odd. Let  $G$  be the  $(m+n-1)$ -vertex,  $(m-1)$ -regular circulant graph  $G = \text{circ}(m+n-1; 1, 2, \dots, \frac{m-1}{2})$ . Then  $G$  does not contain  $K_{1,m}$ , and its  $(n-1)$ -regular edge-complement  $\overline{G}$  does not contain  $K_{1,n}$ . Therefore,  $r(K_{1,m}, K_{1,n}) \geq m+n$ . For the reverse inequality, suppose that  $G$  is an  $(m+n)$ -vertex graph that does not contain  $K_{1,m}$ . Then its maximum degree  $\delta_{\max}(G) \leq m-1$ , and, hence, its edge-complement  $\overline{G}$  has maximum degree  $\delta_{\max}(\overline{G}) \geq n$ . Thus,  $K_{1,n}$  is a subgraph of  $\overline{G}$ , which shows that  $r(K_{1,m}, K_{1,n}) \leq m+n$ .

Case 2.  $m$  and  $n$  are both even. The  $(m+n-2)$ -vertex,  $(m-1)$ -regular circulant graph  $G = \text{circ}(m+n-2; 1, 2, \dots, \frac{m-2}{2}, \frac{m+n-2}{2})$  does not contain  $K_{1,m}$ , and  $\overline{G}$  does not contain  $K_{1,n}$ . Thus,  $r(K_{1,m}, K_{1,n}) \geq m+n-1$ . Now suppose that  $G$  is an  $(m+n-1)$ -vertex graph that does not contain  $K_{1,m}$ . Then, as in Case 1,  $\delta_{\max}(G) \leq m-1$ . But  $G$  cannot be  $(m-1)$ -regular since  $m+n-1$  and  $m-1$  are both odd and the degree sum must be even. Therefore, there is at least one vertex  $v$  with  $\deg_G(v) < m-1$ . Then  $\deg_{\overline{G}}(v) \geq n$ , and, hence,  $K_{1,n}$  is a subgraph of  $\overline{G}$ . Thus,  $r(K_{1,m}, K_{1,n}) \leq m+n-1$ , which completes the proof.  $\diamond$

**Theorem 11.1.12 (Chvátal, 1977).** *Let  $T$  be a tree with  $t$  vertices. Then*

$$r(K_s, T) = (s-1)(t-1) + 1$$

**Proof:** The assertion is trivially true for  $s=1$  and  $s=2$ . Suppose that  $s \geq 3$ . We observe that the complete  $(s-1)$ -partite graph  $G = K_{t-1, \dots, t-1}$  does not contain  $K_s$ ,

and that each component of  $\overline{G} = (s-1)K_{t-1}$  has only  $t-1$  vertices, so it does not contain  $T$ . Thus, we have the lower bound

$$r(K_s, T) \geq (s-1)(t-1) + 1$$

To establish the upper bound, let  $G$  be any graph with  $(s-1)(t-1) + 1$  vertices, such that  $G$  does not contain  $K_s$ . Then its edge-complement  $\overline{G}$  does not contain a subset of  $s$  mutually non-adjacent vertices, or, equivalently, the independence number  $\alpha(\overline{G}) \leq s-1$ . By Proposition 9.1.4, we have

$$\chi(\overline{G}) \geq \left\lceil \frac{(s-1)(t-1) + 1}{\alpha(\overline{G})} \right\rceil \geq \left\lceil \frac{(s-1)(t-1) + 1}{s-1} \right\rceil = t$$

Thus,  $\overline{G}$  is not  $(t-1)$ -colorable. It follows that  $\overline{G}$  must contain a critically  $t$ -chromatic subgraph  $H$  (see §9.1). By Theorem 9.1.18, we have  $\delta_{\min}(H) \geq t-1$ , and hence, by Theorem 3.1.15, it follows that  $H$  contains tree  $T$ . Therefore,  $\overline{G}$  contains  $T$ , which establishes the upper bound.  $\diamond$

### Generalization to Arbitrarily Many Edge-Partition Classes

**DEFINITION:** The (**generalized**) **Ramsey number**  $r(G_1, G_2, \dots, G_k)$  for any collection of simple graphs  $G_1, G_2, \dots, G_k$  is the minimum number  $n$  such that in any edge  $k$ -coloring of a complete graph  $K_n$ , at least one of the subgraphs  $G_j$  has all of its edges in color class  $j$ .

In 1955, Greenwood and Gleason established the following result, which is still, as of this writing, the only non-trivial generalized Ramsey number known for the case in which all the graphs  $G_1, \dots, G_k$  are complete. A proof may be found in [GrGl55].

**Theorem 11.1.13 (Greenwood and Gleason, 1955).**  $r(3, 3, 3) = 17$ .  $\diamond$

**Remark:** For other results, conjectures, variations, and references in Ramsey theory, see [Fa04] and [GrRoSp90].

### EXERCISES for Section 11.1

11.1.1<sup>s</sup> Use Theorems 11.1.5 and 11.1.6 to calculate upper bounds for  $r(3, 6)$ . Which gives the tighter bound?

11.1.2 Prove that  $r(3, 6) \geq 17$  by considering  $\text{circ}(16; 1, 5, 8)$ .

11.1.3 Use Exercise 11.1.1 to calculate an upper bound for  $r(3, 7)$ .

11.1.4 Prove that  $r(3, 7) \geq 21$  by considering  $\text{circ}(21; 1, 5, 9)$ .

11.1.5 Use Theorems 11.1.5 and 11.1.6 to calculate upper bounds for  $r(4, 5)$ . Which gives the tighter bound?

11.1.6<sup>s</sup> Prove that  $r(3, t) \leq \frac{t^2 + t}{2}$ , by using Theorem 11.1.6.

11.1.7 Prove that  $r(3, t) \leq \frac{t^2 + 4}{2}$ , for  $t \geq 3$ , by using Theorem 11.1.5 inductively.

11.1.8 Prove that  $r(3, t) \leq \frac{t^2 + 3}{2}$ , for  $t \geq 3$  and  $t$  odd, by using Exercise 11.1.7.

11.1.9 Prove that  $r(3, t) \leq \frac{t^2 + 3}{2}$ , for  $t \geq 3$  and  $t$  odd, by using Theorem 11.1.5(b) and Exercise 11.1.8.

11.1.10 Prove that  $r(3, 3, 3) \leq 17$ .

## 11.2 EXTREMAL GRAPH THEORY

Extremal graph theory is primarily concerned with determining the maximum number of edges an  $n$ -vertex simple graph may contain without having a given property. This is an area of graph theory in which many of the proofs are long and computationally involved. We are focusing here on easily proved results.

**DEFINITION:** Within a given domain of graphs, a property  $\mathcal{P}$  is said to hold for every  $n$ -vertex simple graph with **sufficiently many edges** if there is a number  $M$  such that every graph in that domain with at least  $M$  edges has property  $\mathcal{P}$ .

**DEFINITION:** Within a given domain of graphs, a graph property  $\mathcal{P}$  that holds for every  $n$ -vertex simple graph with sufficiently many edges, for every sufficiently large number  $n$  of vertices, is said to hold for every **sufficiently large graph** in the domain.

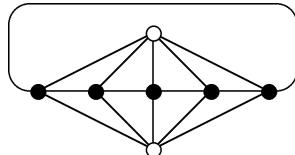
### Extremal Graphs and Extremal Functions

**DEFINITION:** Within a given domain of simple graphs, an **extremal graph** for a property  $\mathcal{P}$  is a largest  $n$ -vertex graph that does not have property  $\mathcal{P}$ , where “largest” means having the maximum number of edges.

**DEFINITION:** Within a given domain of simple graphs, an **extremal function** for a property  $\mathcal{P}$  is the function  $ex(n, \mathcal{P})$  whose value is the number of edges in an extremal graph for that property.

**Example 11.2.1:** Consider the property  $\mathcal{P}$  of having at least one cycle. An  $n$ -vertex graph with  $n - 1$  edges has no cycles if and only if it is a tree. Moreover, every  $n$ -vertex graph with at least  $n$  edges has a cycle. Thus, the extremal graphs for having a cycle are the trees, and the extremal function is  $ex(n, \mathcal{P}) = n - 1$ .

**Example 11.2.2:** Consider the property  $\mathcal{P}$  of nonplanarity. An  $n$ -vertex graph with at least  $3n - 5$  edges is non-planar (see §7.5). To construct a planar  $n$ -vertex graph with  $3n - 6$  edges, join an  $(n - 2)$ -cycle to two new vertices, as shown in Figure 11.2.1. Thus, the extremal function is  $ex(n, \mathcal{P}) = 3n - 6$ .



**Figure 11.2.1** The graph  $2K_1 + C_5$  is extremal for nonplanarity.

**Example 11.2.3:** The property of eulerian traversability (§6.2) does not hold for sufficiently large graphs. In particular, it does not hold for a complete graph  $K_{2n}$  on evenly many vertices.

### Turán's Theorem

The prototypical problem of extremal graph theory and its solution are due to P. Turán [Tu41]: what is the maximum number of edges that an  $n$ -vertex simple graph could have and still contain no subgraph isomorphic to  $K_3$ ?

**TERMINOLOGY:** A graph is said to be  **$F$ -free** if it contains no subgraph isomorphic to a given graph  $F$ . In particular, Turán's problem is about largest  $K_3$ -free graphs. More generally, if  $\mathcal{F}$  is a family of graphs, then we may describe a graph as  $\mathcal{F}$ -free, if it is free of every member of  $\mathcal{F}$ .

**NOTATION:** For a given family  $\mathcal{F}$  of graphs,  $ex(n, \mathcal{F})$  denotes the extremal number for the property of being  $\mathcal{F}$ -free. If  $\mathcal{F}$  contains only one graph  $F$ , then we commonly write  $ex(n, F)$ .

Turán showed that  $ex(n, K_3) = \left\lfloor \frac{n^2}{4} \right\rfloor$ . Theorems 11.2.1 and 11.2.2 establish the upper and lower bounds, respectively.

**Theorem 11.2.1 (Turán, 1941).** *Let  $G$  be a simple graph with  $n$  vertices and at least  $\left\lfloor \frac{n^2}{4} \right\rfloor + 1$  edges. Then  $G$  contains a subgraph isomorphic to  $K_3$ .*

**Proof:** Using  $n = 3$  and  $n = 4$  as base cases for an induction, we have  $\left\lfloor \frac{n^2}{4} \right\rfloor + 1 = 3$  and 5, respectively. Thus, graph  $G$  is isomorphic to  $K_3$  or to  $K_4 - K_2$ , so in both cases,  $G$  contains  $K_3$ . Assume that the assertion is true for all graphs with  $k$  vertices, with  $3 \leq k < n$  and  $n \geq 5$ , and let graph  $G$  have  $n$  vertices and at least  $\left\lfloor \frac{n^2}{4} \right\rfloor + 1$  edges.

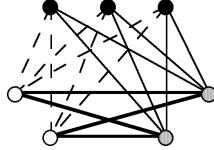
Let  $u$  and  $v$  be adjacent vertices in  $G$ . Then the subgraph  $H = G - \{u, v\}$  has  $n - 2$  vertices. If there is some vertex  $w$  of  $G - \{u, v\}$  that is adjacent to both  $u$  and  $v$ , then the subgraph of  $G$  induced on  $\{u, v, w\}$  is isomorphic to  $K_3$ . If there is no such vertex  $w$ , then at most  $n - 2$  edges join  $u$  and  $v$  to  $H$ , and hence, the number of edges in  $H$  is at least

$$\left\lfloor \frac{n^2}{4} \right\rfloor + 1 - (n - 1) = \left\lfloor \frac{n^2 - 4n + 4}{4} \right\rfloor + 1 = \left\lfloor \frac{(n - 2)^2}{4} \right\rfloor + 1$$

By the induction hypothesis,  $H$  contains a  $K_3$  subgraph. ◊

**REVIEW FROM §9.1:** An  **$r$ -partite graph** is a loopless graph whose vertices can be partitioned into  $k$  independent sets, which are sometimes called the **partite sets** of the partition.

**DEFINITION:** The **Turán graph**  $T_{n,r}$  is the complete  $r$ -partite graph on  $n$  vertices whose partite sets are as nearly equal in cardinality as possible.

**Figure 11.2.2** The Turán graph  $T_{7,3}$ .

**Theorem 11.2.2 (Turán, 1941).** The bipartite graph  $T_{n,2}$  is an extremal graph for  $\text{ex}(n, K_3)$ .

**Proof:** Since  $T_{n,2}$  is bipartite, it is  $K_3$ -free. If  $n$  is even, then  $T_{n,2} \cong K_{\frac{n}{2}, \frac{n}{2}}$  has  $\frac{n^2}{4}$  edges. If  $n$  is odd, then  $T_{n,2} \cong K_{\frac{n+1}{2}, \frac{n-1}{2}}$  has  $\frac{n^2-1}{4}$  edges. In either case, the graph  $T_{n,2}$  has  $\lfloor \frac{n^2}{4} \rfloor$  edges.  $\diamond$

**Remark:** The methods above for deriving  $\text{ex}(n, K_3)$  generalize readily to calculating  $\text{ex}(n, K_r)$  for arbitrary  $r \geq 3$ . In general, the Turán graph  $T_{n,r}$  is an extremal graph for this problem.

### Vertex Degree

A graph is extremal for the property of having a vertex of degree  $d$  if and only if it is extremal for being  $K_{1,d}$ -free.

**Theorem 11.2.3.**  $\text{ex}(n, K_{1,d}) = \left\lfloor \frac{n(d-1)}{2} \right\rfloor$ .

**Proof:** Let  $G$  be a graph with  $\lfloor \frac{n(d-1)}{2} \rfloor + 1$  edges. Then

$$2|E_G| = 2 \left( \left\lfloor \frac{n(d-1)}{2} \right\rfloor + 1 \right) > 2 \frac{n(d-1)}{2} = n(d-1)$$

Therefore,

$$\delta_{\text{avg}}(G) = \frac{2|E_G|}{n} > d-1$$

It follows that some vertex has degree at least  $d$ , which shows that

$$\text{ex}(n, K_{1,d}) \leq \left\lfloor \frac{n(d-1)}{2} \right\rfloor$$

If  $d$  is odd or  $n$  is even, then a  $(d-1)$ -regular graph on  $n$  vertices is  $K_{1,d}$ -free and has  $\lfloor \frac{n(d-1)}{2} \rfloor$  edges. Otherwise, add a maximum matching to a  $(d-2)$ -regular  $n$ -vertex graph to obtain a  $K_{1,d}$ -free graph whose number of edges is

$$\frac{n(d-2)}{2} + \frac{n-1}{2} = \frac{nd-n-1}{2} = \left\lfloor \frac{nd-n}{2} \right\rfloor$$

which establishes the lower bound and completes the proof.  $\diamond$

### Connectedness

**Proposition 11.2.4.** *In the domain of simple graphs, the graph  $K_{n-1} \cup K_1$  is an extremal graph for the property of connectedness.*

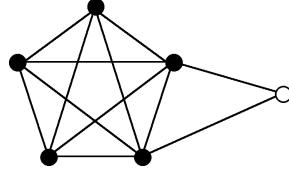
**Proof:** Let  $G$  be an  $n$ -vertex extremal graph for connectedness. Then  $G$  must have at least two components, or it would be connected. There cannot be more than two components, since otherwise, two components could be joined by an edge without yielding a connected graph. Hence, there are exactly two components. Both components must be complete, since otherwise an edge could be added without yielding a connected graph. Thus,  $G \cong K_r \cup K_{n-r}$ . It follows that

$$|E_G| = \binom{r}{2} + \binom{n-r}{2} = \frac{r^2 - r}{2} + \frac{(n-r)^2 - (n-r)}{2} = \frac{n^2 - 2nr + 2r^2 - n}{2}$$

which has a maximum on the interval  $[1, n-1]$  when  $r=1$  and  $r=n-1$ .  $\diamond$

**Proposition 11.2.5.** *In the domain of simple graphs, the graph  $G$  that results from joining one new vertex to  $k-1$  vertices of  $K_{n-1}$  is an extremal graph for the property of  $k$ -connectedness.*

**Proof:** Clearly  $G$  is not  $k$ -connected. Thus, it suffices to prove that any  $n$ -vertex simple graph  $H$  with one more edge, i.e., with  $\binom{n-1}{2} + k$  edges, is  $k$ -connected. The basis step for an induction is provided by Proposition 11.2.4.



**Figure 11.2.3** A 6-vertex extremal graph for 3-connectedness.

Let  $v$  be any vertex of graph  $H$ . Then

$$\begin{aligned} |E_{H-v}| &= |E_H| - \deg_H(v) \geq \binom{n-1}{2} + k - (n-1) \\ &= \frac{n^2 - 3n + 2}{2} + k - \frac{2n - 2}{2} \\ &= \frac{n^2 - 5n + 4}{2} + k = \frac{n^2 - 5n + 6}{2} + k - 1 \\ &= \binom{n-2}{2} + (k-1) \end{aligned}$$

This implies, by the induction hypothesis, that the  $(n-1)$ -vertex graph  $H-v$  is  $(k-1)$ -connected. Therefore, graph  $H$  must be  $k$ -connected.  $\diamond$

### Traversability

**Proposition 11.2.6.** A simple  $n$ -vertex graph  $G$  with at least  $\binom{n-1}{2} + 2$  edges is hamiltonian.

**Proof:** Let  $u$  and  $v$  be two non-adjacent vertices of  $G$ . Let  $H$  be the bipartite spanning subgraph of  $G$  whose partite sets are  $\{u, v\}$  and  $V_G - \{u, v\}$ . Clearly,

$$\begin{aligned}\deg(u) + \deg(v) &= 2|E_G| - 2|E_{G-\{u,v\}}| - |E_H| \\ &= |E_G| - 2|E_{G-\{u,v\}}| - (\deg(u) + \deg(v))\end{aligned}$$

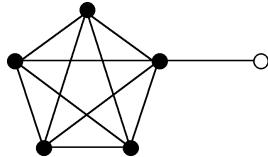
since  $|E_H| = \deg(u) + \deg(v)$ . Accordingly,

$$\begin{aligned}\deg(u) + \deg(v) &= \frac{1}{2} [2|E_G| - 2|E_{G-\{u,v\}}|] \\ &\geq \frac{1}{2} \left[ 2 \left( \binom{n-1}{2} + 2 \right) - 2 \binom{n-2}{2} \right] \\ &= \frac{1}{2} [(n^2 - 3n + 6) - (n^2 - 5n + 6)] = n\end{aligned}$$

By Ore's Theorem (§6.3), graph  $H$  (and hence  $G$ ) is hamiltonian.  $\diamond$

**Corollary 11.2.7.** The graph  $G$  obtained by joining a new vertex to one vertex of a complete graph  $K_{n-1}$  is extremal for hamiltonian traversability.

**Proof:** The graph  $G$  is non-hamiltonian, and it has  $\binom{n-1}{2} + 1$  edges. By Proposition 11.2.6, the result of adding one more edge to  $G$  would be a hamiltonian graph.  $\diamond$



**Figure 11.2.4** A 6-vertex extremal graph for hamiltonian traversability.

**Corollary 11.2.8.** A simple  $n$ -vertex graph with at least  $\binom{n-1}{2} + 1$  edges contains an open hamiltonian path.

**Proof:** Choose any two non-adjacent vertices  $u$  and  $v$ . By Proposition 11.2.6, the result of adding an edge between vertices  $u$  and  $v$  would be a hamiltonian graph. Thus, there must be a hamiltonian path from  $u$  to  $v$ .  $\diamond$

**Corollary 11.2.9.**  $ex(n, P_n) = \binom{n-1}{2}$ .

**Proof:** In view of Corollary 11.2.8, the graph  $K_1 \cup K_{n-1}$  is extremal.  $\diamond$

### Chromatic Number

It is trivial to calculate extremal numbers for some extreme values of chromaticity. It is easy to calculate them for nearly extreme values.

**Proposition 11.2.10.** *The extremal number of edges for chromatic number 2 is 0. The extremal graph is the edgeless graph.*  $\diamond$

**Proposition 11.2.11.** *For  $n$ -vertex simple graphs, the extremal number of edges for chromatic number  $n$  is  $\binom{n}{2}$ .*  $\diamond$

**Proposition 11.2.12.** *For  $n$ -vertex graphs, the extremal number of edges for chromatic number 3 is*

$$\binom{\lfloor n/2 \rfloor}{2} \binom{\lceil n/2 \rceil}{2}$$

**Proof:** The symmetric complete bipartite graph  $K_{n,n}$  is an extremal graph for chromatic number 3. If the partite sets have respective cardinalities  $r$  and  $n - r$ , then the number of edges must be

$$\binom{r}{2} \binom{n-r}{2}$$

which takes a maximum when  $r = \lfloor n/2 \rfloor$  and  $n - r = \lceil n/2 \rceil$ .  $\diamond$

### EXERCISES for Section 11.2

- 11.2.1<sup>s</sup> List all 5-vertex graphs that are extremal for nonplanarity.
- 11.2.2 List all 6-vertex graphs that are extremal for nonplanarity.
- 11.2.3 Prove that your list in Exercise 11.2.2 is complete.
- 11.2.4 Calculate  $ex(n, K_4)$  and  $ex(n, K_5)$ , by using the fact that the Turán graphs are extremal.
- 11.2.5<sup>s</sup> Calculate  $ex(4, 2K_2)$ , and draw the extremal graphs.
- 11.2.6 Calculate  $ex(5, 2K_2)$ , and draw the extremal graphs.
- 11.2.7 Calculate  $ex(6, 2K_2)$ , and draw the extremal graphs.
- 11.2.8 Calculate  $ex(6, 3K_2)$ , and draw the extremal graphs.
- 11.2.9 Calculate the extremal number for the property of having radius 1.
- 11.2.10 Calculate the extremal number for the property of having diameter 2.

## 11.3 RANDOM GRAPHS

A *random graph* is a graph in which the number of vertices is specified and the adjacencies between vertices are determined in some random way. The vertices are usually considered to have the *standard labeling*  $1, 2, \dots, n$ . The predominant concern is large graphs, with many vertices.

The *theory of random graphs* uses probabilistic methods to establish the existence of certain kinds of graphs and to determine some properties of “almost all” graphs in various families. Whereas we were concerned in §11.2 with *extremal numbers* for various properties, here we focus mainly on what happens *on average*, i.e., on what the *expected* properties are.

It is assumed throughout this section that the reader is reasonably familiar with the basic concepts of elementary probability, including the uniform and binomial distributions, expectation, and conditional probability.

NOTATION: We use  $pr(E)$  to denote the probability of an event  $E$ .

### Labeled Graph Isomorphism

DEFINITION: An isomorphism  $f : G \rightarrow H$  of two labeled graphs is a **label-preserving isomorphism** if for every vertex  $v \in V_G$ , the vertex  $f(v) \in V_H$  has the same label as  $v$ .

In the theory of random graphs we usually distinguish between two standard labeled graphs, unless there is a label-preserving isomorphism between them.

### The Probabilistic Method of Proof

As an illustration of the probabilistic method, we consider the problem of calculating a lower bound on **diagonal Ramsey numbers**, that is, Ramsey numbers of the form  $r(s, s)$ . We recall from §11.1 that  $r(3, 3) = 6$  and that  $r(4, 4) = 18$ .

**Theorem 11.3.1.** *For every integer  $s \geq 3$ , we have  $r(s, s) > \lfloor 2^{s/2} \rfloor$ .*

**Proof:** Let  $n = \lfloor 2^{s/2} \rfloor$ . The complete graph  $K_n$  has  $\binom{n}{2}$  edges. Thus, the space of all red-blue edge-colorings of a standard-labeled  $K_n$  contains  $2^{\binom{n}{2}}$  different red-blue graphs. Our objective is to show that at least one of them contains no monochromatic  $K_s$ , that is, neither a red  $K_s$  nor a blue  $K_s$ . It is sufficient to show that the probability that a random red-blue graph in the space has no monochromatic  $K_s$  is less than 1.

Toward that objective, suppose that each edge  $ij$  is red with probability  $1/2$ , and that we take the edge colors to be mutually independent. Then each of the red-blue graphs occurs with probability  $2^{-\binom{n}{2}}$ .

For any fixed subset  $L \subset \{1, \dots, n\}$  of cardinality  $s$ , let  $E_L$  be the event that the complete subgraph induced by the vertex set  $L$  is either a red  $K_s$  or a blue  $K_s$ . Then

$$pr(E_L) = \left(\frac{1}{2}\right)^{\binom{s}{2}} + \left(\frac{1}{2}\right)^{\binom{s}{2}} = 2 \left(\frac{1}{2}\right)^{\binom{s}{2}} = 2^{1-\binom{s}{2}}$$

Let  $E_s$  denote the union of all such events  $E_L$  where  $L$  ranges over all subsets of cardinality  $s$ . Since there are exactly  $\binom{n}{s}$  such subsets, it follows that

$$\begin{aligned} pr(E_s) &\leq \sum pr(E_L) = \binom{n}{s} \cdot 2^{1-\binom{s}{2}} \\ &= \frac{n!}{(n-s)! \cdot s!} \cdot 2^{1-\frac{s^2}{2}+\frac{s}{2}} \\ &< \frac{n^s}{s!} \cdot 2^{1-\frac{s^2}{2}+\frac{s}{2}} \\ &\leq \frac{2^{\frac{s^2}{2}}}{s!} \cdot 2^{1-\frac{s^2}{2}+\frac{s}{2}} \quad (\text{since } n \leq 2^{\frac{s}{2}}) \\ &= \frac{1}{s!} \cdot 2^{1+\frac{s}{2}} < 1 \quad (\text{since } s \geq 3) \end{aligned}$$

◇

As a second illustration of a probabilistic proof, we consider a problem on directed graphs.

**DEFINITION:** A **tournament** is a complete graph in which each edge is assigned a direction. (Tournaments are discussed in §12.3.)

**Theorem 11.3.2.** [Szele, 1943] For every positive integer  $n$ , there is an  $n$ -vertex tournament with at least  $n!2^{n-1}$  (directed) hamiltonian paths.

**Proof:** Let  $D$  be an  $n$ -vertex tournament whose edge-directions are randomly assigned, with probability  $1/2$  in each direction. For each ordering  $\tau$  of the vertices  $1, \dots, n$ , let

$$h_\tau(D) = \begin{cases} 1 & \text{if } \tau \text{ represents a hamiltonian path in } D \\ 0 & \text{otherwise} \end{cases}$$

Thus, the sum over all orderings  $\tau$ ,

$$h(D) = \sum_{\tau} h_\tau(D)$$

is the number of hamiltonian paths in tournament  $D$ .

The probability that a given ordering  $\tau$  corresponds to a hamiltonian path in  $D$  equals  $2^{-(n-1)}$ , and, hence, the expected value  $E(h_\tau(D))$  equals  $2^{-(n-1)}$ . It follows that the expected number of hamiltonian paths in a random tournament  $D$  is

$$E(h(D)) = E\left(\sum_{\tau} h_\tau(D)\right) = \sum_{\tau} E(h_\tau(D)) = n!2^{-(n-1)}$$

Since  $n!2^{-(n-1)}$  is the average number of hamiltonian paths in an  $n$ -vertex tournament, there must be at least one tournament whose number of hamiltonian paths is at least  $n!2^{-(n-1)}$ .  $\diamond$

## Models for Random Graphs

**DEFINITION:** A **model for random graphs** is a probability space whose elements are graphs.

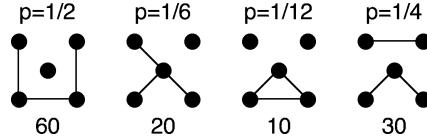
**TERMINOLOGY:** The phrase *random graph* can refer to the model for random graphs itself or to one of the elements in that probability space.

The two most common models — the only ones we consider here — are the *uniform random graph*, in which the number  $m$  of edges is specified and all labeled graphs with  $m$  edges are equally likely, and the *Bernoulli random graph*, in which the number of edges is not fixed and each edge is equally likely to occur.

## The Erdős-Rényi Random Graph

**DEFINITION:** For  $0 \leq m \leq \binom{n}{2}$ , the **uniform random graph**  $\mathcal{G}(n, m)$ , also called the **Erdős-Rényi random graph**, is the equiprobable space of all  $n$ -vertex,  $m$ -edge, labeled simple graphs, each having probability  $\left(\frac{\binom{n}{2}}{m}\right)^{-1}$ .

**Example 11.3.1:** There are  $\binom{5}{2} = 120$  random graphs in  $\mathcal{G}(5, 3)$ . Figure 11.3.1 shows the four possible isomorphism types.



**Figure 11.3.1** Probabilities of isomorphism types of graphs in  $\mathcal{G}(5, 3)$ .

Below each type is the number of labeled graphs corresponding to that type. Above each is the probability that a random  $\mathcal{G}(5, 3)$  graph will be of that type. We see that although each graph is equally likely, some isomorphism types are more likely than others. Notice also that the probabilities for a  $\mathcal{G}(5, 3)$  graph having 0, 1, and 2 isolated vertices are

$$\frac{30}{120} = \frac{1}{4}, \quad \frac{80}{120} = \frac{2}{3}, \quad \text{and} \quad \frac{10}{120} = \frac{1}{12}$$

respectively. Thus, the expected number of isolated vertices is

$$0 \cdot \frac{1}{4} + 1 \cdot \frac{2}{3} + 2 \cdot \frac{1}{12} = \frac{5}{6}$$

### Expected Number of Copies of $K_s$ in $\mathcal{G}(n, m)$

Calculating the expected number of copies of the complete subgraph  $K_3$  in a random graph  $\mathcal{G}(n, m)$  in the proof of Theorem 11.3.3 below involves all of the steps needed for the general complete subgraph  $K_s$ . The following definition simplifies the representation of summations over arbitrary sets and is used in that calculation.

**DEFINITION:** The **Iverson truth function** is

$$\text{true}(A) = \begin{cases} 1 & \text{if assertion A is true,} \\ 0 & \text{if assertion A is false} \end{cases}$$

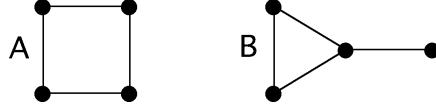
**Example 11.3.2:** Since there are  $\binom{4}{2} = 15$  random graphs in  $\mathcal{G}(4, 4)$ , each has probability  $\frac{1}{15}$ . The number of copies of  $K_3$  in a given graph  $G$  is

$$\sum_{u, v, w \in V_G} \text{true}(uv, uw, vw \in E_G)$$

Thus, the expected number of copies of  $K_3$  in a  $\mathcal{G}(4, 4)$  graph equals

$$\frac{1}{15} \sum_{G \in \mathcal{G}(4, 4)} \sum_{u, v, w \in V(G)} \text{true}(uv, uw, vw \in E_G)$$

Figure 11.3.2 shows the two isomorphism types of 4-vertex, 4-edge graphs. There are 3  $\mathcal{G}(4, 4)$  graphs of type A and 12 of type B. Graphs of type A have no  $K_3$  and graphs of type B each have one. Thus, the value of the double sum above is 12, and, hence, the expected number of copies of  $K_3$  in a  $\mathcal{G}(4, 4)$  graph is  $\frac{12}{15} = \frac{4}{5}$ .



**Figure 11.3.2** The two isomorphism types of 4-vertex, 4-edge graphs.

**Remark:** Example 11.3.2 underscores the distinction between labeled graphs, which are the elements of the probability space  $\mathcal{G}(4, 4)$ , and the isomorphism types of unlabeled graphs. Each of the 15 elements of  $\mathcal{G}(4, 4)$  is equally likely, but a graph of isomorphism type B is 4 times as likely as the one of type A. If both were equally likely, then the expected number of copies of  $K_3$  would have been  $\frac{1}{2}$ .

The next result establishes a general formula for the expected number of copies of  $K_3$  in the random graph  $\mathcal{G}(n, m)$ . The proof is based on a simple interchange of the order of summation.

**Theorem 11.3.3.** *The expected number of copies of the complete graph  $K_3$  in a random graph in  $\mathcal{G}(n, m)$  is  $\binom{n}{3} \cdot \binom{m}{3} \cdot \left(\binom{n}{2}/3\right)^{-1}$ .*

**Proof:** Arguing as in Example 11.3.2, we calculate that the expected number of copies of  $K_3$  in a random graph in  $\mathcal{G}(n, m)$  equals

$$\left(\binom{n}{2}/3\right)^{-1} \sum_{G \in \mathcal{G}(n, m)} \sum_{u, v, w \in V(G)} \text{true}(uv, uw, vw \in E_G)$$

Interchanging the order of summation, we obtain

$$\left(\binom{n}{2}/3\right)^{-1} \sum_{u, v, w \in \{1, \dots, n\}} \sum_{G \in \mathcal{G}(n, m)} \text{true}(uv, uw, vw \in E_G)$$

For each of the  $\binom{n}{3}$  triples  $u, v, w \in \{1, \dots, n\}$ , the inner sum has constant value

$$\binom{\binom{n}{2} - 3}{m - 3}$$

It follows that the expected number of copies is

$$\begin{aligned} \binom{n}{3} \cdot \binom{\binom{n}{2} - 3}{m - 3} \cdot \left(\binom{n}{2}/3\right)^{-1} &= \binom{n}{3} \cdot \frac{[(\binom{n}{2} - 3)!]}{(m - 3)! [(\binom{n}{2} - m)!]} \cdot \frac{m! [(\binom{n}{2} - m)!]}{(\binom{n}{2})!} \\ &= \binom{n}{3} \cdot \frac{m!}{(m - 3)!} \cdot \frac{[(\binom{n}{2} - 3)!]}{(\binom{n}{2})!} = \binom{n}{3} \cdot \binom{m}{3} \cdot \left(\binom{n}{2}/3\right)^{-1} \quad \diamond \end{aligned}$$

**Remark:** If we evaluate the formula of Theorem 11.3.3 for  $m \geq n + 1$ , we see that the expected number of copies of  $K_3$  is greater than 1.

The proof of Theorem 11.3.3 has an immediate generalization.

**Theorem 11.3.4.** *The expected number of copies of an arbitrary complete graph  $K_s$  in a random graph  $G \in \mathcal{G}(n, m)$  is  $\binom{n}{s} \cdot \binom{m}{\binom{s}{2}} \cdot \left(\binom{n}{2}\right)^{-1}$*   $\diamondsuit$  (Exercises)

### Expected Number of $k$ -cycles

Similar calculations enable us to determine the expected number of copies of any fixed graph  $H$  in a random graph  $G \in \mathcal{G}(n, m)$ . As an illustration, we consider the case in which  $H$  is a  $k$ -cycle. We count *unoriented*  $k$ -cycles, since a vertex sequence and its reverse represent the same  $k$ -cycle *subgraph*. The proof uses the elementary combinatorial result that the number of circular permutations of a  $k$ -element set is  $(k - 1)!$ .

**Theorem 11.3.5.** *The expected number of  $k$ -cycles in a random graph  $G \in \mathcal{G}(n, m)$  is*

$$\frac{(k - 1)!}{2} \cdot \binom{n}{k} \cdot \binom{m}{k} \cdot \left(\binom{n}{2}\right)^{-1}$$

**Proof:** Any given labeled  $k$ -cycle  $C$  uses  $k$  edges. Thus, the number of graphs in  $\mathcal{G}(n, m)$  that contain  $C$  is

$$\binom{\binom{n}{2} - k}{m - k}$$

The number of different labeled  $k$ -cycles in  $K_n$  is

$$\frac{1}{2} \binom{n}{k} (k - 1)! = \frac{(k - 1)!}{2} \binom{n}{k}$$

where the division by 2 results from counting unoriented cycles. It follows that the total number of  $k$ -cycles occurring among the graphs in  $\mathcal{G}(n, m)$  is

$$\frac{(k - 1)!}{2} \binom{n}{k} \cdot \binom{\binom{n}{2} - k}{m - k}$$

Thus, the expected number of  $k$ -cycles in a random graph in  $\mathcal{G}(n, m)$  is

$$\begin{aligned} & \frac{(k - 1)!}{2} \binom{n}{k} \cdot \binom{\binom{n}{2} - k}{m - k} \cdot \left(\binom{n}{2}\right)^{-1} \\ &= \frac{(k - 1)!}{2} \binom{n}{k} \cdot \frac{\left(\binom{n}{2} - k\right)!}{(m - k)! \left(\binom{n}{2} - m\right)!} \cdot \frac{m! \left(\binom{n}{2} - m\right)!}{\binom{n}{2}!} \\ &= \frac{(k - 1)!}{2} \cdot \binom{n}{k} \cdot \frac{m!}{(m - k)!} \cdot \frac{\left(\binom{n}{2} - k\right)!}{\binom{n}{2}!} = \frac{(k - 1)!}{2} \cdot \binom{n}{k} \cdot \binom{m}{k} \cdot \left(\binom{n}{2}\right)^{-1} \quad \diamondsuit \end{aligned}$$

### The Bernoulli Random Graph

**DEFINITION:** The **Bernoulli random graph**  $\mathcal{G}(n, p)$  is the probability space of all labeled simple graphs on  $n$  vertices, where each of the  $\binom{n}{2}$  possible edges occurs with probability  $p$ . Thus, the probability of each  $m$ -edge graph is  $p^m(1 - p)^{\binom{n}{2} - m}$ .

**Example 11.3.3:** Since there are  $\binom{\binom{5}{2}}{3}$  possible 5-vertex, 3-edge graphs, the probability that a random graph in  $\mathcal{G}(5, p)$  has 3 edges is

$$\binom{10}{3} p^3 (1-p)^7$$

For  $p = \frac{1}{2}$ , this probability equals

$$\frac{120}{1024} = \frac{15}{128}$$

### Bernoulli Space as a Union of Erdős-Rényi Spaces

The Bernoulli space  $\mathcal{G}(n, p)$  of  $2^{\binom{n}{2}}$  labeled simple  $n$ -vertex graphs is partitioned into  $\binom{n}{2} + 1$  different Erdős-Rényi spaces

$$\left\{ \mathcal{G}(n, m) \mid m = 0, \dots, \binom{n}{2} \right\}$$

where each  $\mathcal{G}(n, m)$  has  $\binom{\binom{n}{2}}{m}$  labeled simple graphs. Here, the probability distribution in  $\mathcal{G}(n, m)$  is the conditional distribution of  $\mathcal{G}(n, \frac{1}{2})$  restricted to the space of  $n$ -vertex,  $m$ -edge graphs.

**Example 11.3.4:** The probability that a random graph is a complete graph in  $\mathcal{G}(n, p)$  is  $p^{\binom{n}{2}}$ . Of course, in the Erdős-Rényi space  $\mathcal{G}(n, \binom{n}{2})$ , the probability is 1 (since the complete graph is the only element).

**Example 11.3.5:** To calculate the probability in  $\mathcal{G}(4, p)$  that a random graph is a 3-edge path graph  $P_4$ , we observe that there are  $\frac{4!}{2} = 12$   $P_4$ 's in  $K_4$ . Since it has 3 edges, each possible  $P_4$  occurs with probability  $p^3(1-p)^3$ . Thus, the net probability is  $12p^3(1-p)^3$ . In the Erdős-Rényi space  $\mathcal{G}(4, 3)$ , the  $\binom{6}{3} = 20$  graphs fall into three isomorphism types. There are 4 cases of  $K_3 \cup K_1$ , 4 cases of  $K_{1,3}$ , and 12 cases of  $P_4$ . Thus, the probability of  $P_4$  is  $12/20 = 3/5$ .

### Some Properties of Almost Every Graph

**DEFINITION:** A graph property  $\mathcal{P}$  is said to hold for **almost every graph** if the probability that a random graph  $G \in \mathcal{G}(n, p)$  has property  $\mathcal{P}$  has the limiting value of 1 as  $n \rightarrow \infty$ , for  $p > 0$ .

**Theorem 11.3.6.** *Almost every graph contains  $K_3$ .*

**Proof:** In a random graph  $G \in \mathcal{G}(n, p)$ , select  $\lfloor \frac{n}{3} \rfloor$  disjoint triples (i.e., size-3 subsets) of  $V_G$ . For any particular one of these triples  $\{u, v, w\}$ , the probability that all of the edges  $uv, uw, vw$  are present is  $p^3$ . Thus, the probability that not all of them are present is  $1 - p^3$ . Since the presence of edges in  $G$  is probabilistically independent, it follows that the probability that none of these triples is spanned by a  $K_3$  is

$$(1 - p^3)^{\lfloor n/3 \rfloor}$$

whose limit is 0 as  $n \rightarrow \infty$  for  $p > 0$ , because  $1 - p^3 < 1$ . Consequently, the probability that at least one of the triples is spanned by  $K_3$  has limiting value 1 as  $n \rightarrow \infty$  for  $p > 0$ .  $\diamond$

The proof of Theorem 11.3.6 has an immediate generalization.

**Theorem 11.3.7.** *For any integer  $s \geq 1$ , almost every graph contains  $K_s$ .*

◊ (Exercises)

**Corollary 11.3.8.** *Let  $H$  be any fixed simple graph. Then almost every graph contains a copy of  $H$ .*

**Proof:** This follows immediately from Theorem 11.3.7, because the graph  $H$  is a subgraph of some complete graph  $K_s$ . ◊

**Proposition 11.3.9.** *In almost every graph, every pair of vertices is joined by a path of length 2.*

**Proof:** Let  $G \in \mathcal{G}(n, p)$ , and let  $u, v \in G$ . For each vertex  $w \in V_G - \{u, v\}$ , the probability that not both the edges  $uw$  and  $vw$  are present is  $1-p^2$ . Thus, the probability that there is no path of length 2 joining  $u$  and  $v$  is

$$(1-p^2)^{n-2}$$

whose limit is 0 as  $n \rightarrow \infty$ , for  $p > 0$ . ◊

**Theorem 11.3.10.** *Almost every graph is connected.*

**Proof:** This is an immediate consequence of Proposition 11.3.9. ◊

The following generalization of Proposition 11.3.9 enables us to strengthen Theorem 11.3.10 to higher orders of connectedness.

**Proposition 11.3.11.** *Let  $r, s$ , and  $t$  be nonnegative integers. In almost every graph, for any  $r$ -subset  $U = \{u_1, \dots, u_r\}$  of vertices and any  $s$ -subset  $V = \{v_1, \dots, v_s\}$  disjoint from  $U$ , there is a  $t$ -subset  $W = \{w_1, \dots, w_t\}$ , disjoint from both  $U$  and  $V$ , such that every vertex  $w_j \in W$  is adjacent to every vertex of  $U$  and nonadjacent to every vertex of  $V$ .*

**Proof:** The probability that an arbitrary subset of  $t$  vertices disjoint from both  $U$  and  $V$  has this adjacency-nonadjacency property is  $(p^r(1-p)^s)^t$ . The  $n-r-s$  vertices in  $\overline{U \cup V}$  can be partitioned into  $\lfloor \frac{n-r-s}{t} \rfloor$   $t$ -subsets of vertices that are pairwise disjoint and disjoint from both  $U$  and  $V$ , with perhaps a few vertices left over. The probability that none of these  $t$ -subsets has the adjacency-nonadjacency property is

$$\left(1 - (p^r(1-p)^s)^t\right)^{\lfloor \frac{n-r-s}{t} \rfloor}$$

whose limit is 0 as  $n \rightarrow \infty$  for  $p > 0$ . ◊

**Theorem 11.3.12.** *For  $k > 0$ , almost every graph is  $k$ -connected.*

**Proof:** This follows immediately from Proposition 11.3.11, with  $r = 2$ ,  $s = 0$ , and  $t = k$ . ◊

### Some Statistical Properties of Random Graphs in $\mathcal{G}(n, p)$

**Theorem 11.3.13.** For a random graph  $G \in \mathcal{G}(n, p)$ , the number of edges has mean  $\mu$  and standard deviation  $\sigma$  given by

$$\mu(|E_G|) = p \cdot \binom{n}{2} \quad \text{and} \quad \sigma(|E_G|) = \sqrt{p(1-p) \cdot \binom{n}{2}}$$

**Proof:** By definition of the Bernoulli space  $\mathcal{G}(n, p)$ , the random variable  $|E_G|$  has the binomial distribution with  $\binom{n}{2}$  trials and success rate  $p$ , and, hence, has the stated mean and standard deviation.  $\diamond$

**Corollary 11.3.14.** For a random graph  $G \in \mathcal{G}(n, p)$ ,  $\sigma(|E_G|) \leq \frac{n}{2\sqrt{2}}$ .

**Proof:** For  $p$  such that  $0 \leq p \leq 1$ , we have  $p(1-p) \leq \frac{1}{4}$ . Thus, by Theorem 11.3.13,

$$\sigma(|E_G|) = \sqrt{p(1-p) \cdot \binom{n}{2}} \leq \sqrt{\frac{n^2}{8}} = \frac{n}{2\sqrt{2}} \quad \diamond$$

**Theorem 11.3.15.** For a random graph  $G \in \mathcal{G}(n, p)$ , the expected distance between two vertices is  $2 - p$ .

**Proof:** Let  $u, v \in V_G$ . Then

$$\Pr(d(u, v) = 1) = \Pr(uv \in E_G) = p$$

and by Proposition 11.3.9,

$$\Pr(d(u, v) = 2) = \Pr(uv \notin E_G) = 1 - p$$

Thus,

$$E(d(u, v)) = 1 \cdot p + 2 \cdot (1 - p) = 2 - p \quad \diamond$$

### EXERCISES for Section 11.3

11.3.1 Draw all six isomorphism types of 5-vertex, 4-edge simple graph. Calculate the probability of each type in  $\mathcal{G}(5, 4)$ .

11.3.2 Using Exercise 11.3.1, calculate the probability that a random graph in  $\mathcal{G}(5, 4)$  contains a  $K_3$ .

11.3.3 Use Theorem 11.3.3 to calculate the expected number of copies of  $K_3$  in a random graph in  $\mathcal{G}(5, 4)$ . (Compare to Exercise 11.3.2.)

11.3.4 Calculate the expected number of copies of the bipartite graph  $K_{3,3}$  in a random graph  $G \in \mathcal{G}(n, m)$ .

11.3.5<sup>s</sup> Calculate the expected number of vertices of degree 3 or more in a random graph in  $\mathcal{G}(5, 4)$ . (Hint: This is the same as the expected number of copies of  $K_{1,3}$ .)

11.3.6 Calculate the probability that a random graph in  $\mathcal{G}(5, 5)$  is the 5-cycle  $C_5$ .

11.3.7 Calculate the probability that a random graph in  $\mathcal{G}(5, \frac{1}{2})$  is the 5-cycle  $C_5$ .

- 11.3.8 Prove the remark immediately after Theorem 11.3.3.
- 11.3.9<sup>s</sup> Prove that almost every graph is nonplanar.
- 11.3.10 Prove that for every  $k$ , the chromatic number of almost every graph is larger than  $k$ .
- 11.3.11 Prove Theorem 11.3.4, that the expected number of copies of the complete graph  $K_s$  in a random graph  $G \in \mathcal{G}(n, m)$  is

$$\binom{n}{s} \cdot \binom{m}{\binom{s}{2}} \cdot \left( \frac{\binom{n}{2}}{\binom{s}{2}} \right)^{-1}$$

- 11.3.12 Prove Theorem 11.3.7, that almost every graph contains  $K_s$ .
- 11.3.13 Use Proposition 11.3.11 to prove that for any  $d > 0$ , almost every graph has minimum degree at least  $d$ .

## 11.4 SUPPLEMENTARY EXERCISES

- 11.4.1 Prove that  $r(K_3, K_{1,3}) > 6$ .
- 11.4.2 Prove that  $r(K_{1,3}, C_4) = 6$ .
- 11.4.3 Calculate  $r(C_4, C_4)$ .
- 11.4.4 Use Theorems 11.2.1 and 11.2.3 to prove that  $r(K_3, K_{1,7}) \leq 7$ .
- 11.4.5 Prove that  $r(K_{1,n}, K_{1,m}) \leq m + n$ .
- 11.4.6 Prove that if  $m$  and  $n$  are both even, then  $r(K_{1,n}, K_{1,m}) \leq m + n - 1$ .
- 11.4.7 Draw a 6-vertex graph that does not contain  $K_3$  and whose complement does not contain the 3-edge path  $P_3$ .
- 11.4.8 Calculate as tight an upper bound as you can for the Ramsey number  $r(5, 5)$ .
- 11.4.9 Prove that  $ex(5, C_4) = 7$ .
- 11.4.10 Prove that  $ex(5, C_5) = 8$ .
- 11.4.11 Let  $G$  be a 6-vertex, 8-edge simple graph. Prove that  $G$  contains a 4-edge path  $P_4$ , for these cases: (a)  $\delta_{\max} = 5$ ; (b)  $\delta_{\max} = 4$ ; (c)  $\delta_{\max} = 3$ . Conclude that  $ex(6, P_4) = 7$ . (Hint: Let vertex  $u$  have maximum degree in  $G$ . If any two of its neighbors are joined, then at each neighbor, there starts a 3-edge-path within the closed neighborhood  $N(u)$ .)
- 11.4.12 For  $n \geq 3$ , let  $t(n)$  be the smallest number such that there exists a  $K_3$ -free, simple,  $n$ -vertex,  $t(n)$ -edge graph  $G(n)$  such that joining any two non-adjacent vertices of  $G(n)$  creates a  $K_3$ . Establish a formula for the function  $t(n)$ .
- 11.4.13 Calculate the expected number of copies of  $K_{2,3}$  in a random graph in  $\mathcal{G}(n, m)$ .
- 11.4.14 Calculate the expected number of copies of  $CL_3$  in a random graph in  $\mathcal{G}(n, m)$ .
- 11.4.15 Calculate the expected number copies of the circulant graph  $circ(8 : 1, 4)$  in a random graph in  $\mathcal{G}(n, m)$ .

**11.4.16** A simple graph is constructed with 12 vertices and 36 randomly selected edges. Calculate the expected number of triangles.

**11.4.17** A simple graph is constructed with 8 vertices and 7 randomly selected edges. Calculate the probability that it is connected.

---

## GLOSSARY

**almost every graph, property that holds for:** a property  $\mathcal{P}$  such that the probability that a random graph  $G \in \mathcal{G}(n, p)$  has property  $\mathcal{P}$  has the limiting value of 1 as  $n \rightarrow \infty$ , for  $p > 0$ .

**Bernoulli random graph  $\mathcal{G}(n, p)$ :** the probability space of all labeled simple graphs on  $n$  vertices, where each of the  $\binom{n}{2}$  possible edges occurs with probability  $p$ ; thus, the probability of each  $m$ -edge graph in  $\mathcal{G}(n, p)$  is  $p^m(1-p)^{\binom{n}{2}-m}$ .

**Erdős-Rényi random graph  $\mathcal{G}(n, m)$ :** the equiprobable space of all  $n$ -vertex,  $m$ -edge, labeled simple graphs, each having probability  $\binom{\binom{n}{2}}{m}^{-1}$ ; also called the *uniform random graph*.

**extremal function  $ex(n, \mathcal{P})$  for a property  $\mathcal{P}$ :** the function whose value is the number of edges in an  $n$ -vertex extremal graph for that property.

**extremal  $n$ -vertex graph for a property  $\mathcal{P}$ :** a largest  $n$ -vertex simple graph that does not have property  $\mathcal{P}$ , where “largest” means having the maximum number of edges.

**Iverson truth function:** the function

$$\text{true}(A) = \begin{cases} 1 & \text{if assertion A is true,} \\ 0 & \text{if assertion A is false} \end{cases}$$

**label-preserving isomorphism:** an isomorphism  $f : G \rightarrow H$  of two labeled graphs such that for every vertex  $v \in V_G$ , the vertex  $f(v) \in V_H$  has the same label as  $v$ .

**model for random graphs:** a probability space whose elements are graphs.

**multipartite graph:** a loopless graph whose vertices can be partitioned into  $k$  independent sets, which are sometimes called the *partite sets*, is said to be  $k$ -partite.

**partite sets:** see *multipartite graph*.

**Ramsey number  $r(s, t)$  for positive integers  $s$  and  $t$ :** the minimum positive number  $n$  such that for every simple  $n$ -vertex graph  $G$ , either  $G$  contains a  $K_s$ -subgraph or its edge-complement graph  $\overline{G}$  contains a  $K_t$ -subgraph.

—, **diagonal:** a Ramsey number  $r(s, s)$ .

—, **for arbitrary subgraphs  $H_1$  and  $H_2$ , denoted  $r(H_1, H_2)$ :** the minimum number  $n$  such that for every  $n$ -vertex simple graph  $G$ , either  $H_1$  is a subgraph of  $G$  or  $H_2$  is a subgraph of  $\overline{G}$ .

—, **generalized, denoted  $r(G_1, G_2, \dots, G_k)$ :** for any collection of simple graphs  $G_1, G_2, \dots, G_k$ , the minimum number  $n$  such that in any edge  $k$ -coloring of a complete graph  $K_n$ , at least one of the subgraphs  $G_j$  has all of its edges in color class  $j$ .

**random graph:** either the *model for random graphs* itself or one of the elements in that probability space.

**sufficiently large graph, with respect to a graph property  $\mathcal{P}$ :** a graph whose number of edges, relative to its number of vertices, is large enough that every graph with as many edges must have the specified property.

**tournament:** a complete graph in which each edge is assigned a direction. (See §12.3.)

**Turán graph  $T_{n,r}$ :** the complete  $r$ -partite graph on  $n$  vertices whose partite sets are as nearly equal in cardinality as possible.

**uniform random graph  $\mathcal{G}(n,m)$ :** the equiprobable space of all  $n$ -vertex,  $m$ -edge, labeled simple graphs, each having probability  $\binom{\binom{n}{2}}{m}^{-1}$ ; also called the *Erdős-Rényi random graph*.



# Chapter 12

---

## SPECIAL DIGRAPH MODELS

**12.1 Directed Paths and Mutual Reachability**

**12.2 Digraphs as Models for Relations**

**12.3 Tournaments**

**12.4 Project Scheduling and Critical Paths**

**12.5 Finding the Strong Components of a Digraph**

---

### INTRODUCTION

Many of the methods and algorithms for digraphs are just like their counterparts for undirected graphs in Chapters 1 through 6, with little or no modification. For instance, the algorithm given in §6.1 for constructing an eulerian tour in an undirected graph works equally well on a digraph. On the other hand, the non-reversibility of directed paths creates complications in a digraph-connectivity algorithm.

The primary focus of this chapter is on some uniquely digraphic models that arise in selected problems and applications. Some of these applications require acyclic digraphs (e.g., *task scheduling* and *tournaments*), whereas others use a general digraph model (e.g., *Markov chains* and *transitive closure*).

Although depth-first search of a digraph and of a graph work the same way, the use of depth-first search in finding the strong components of a digraph is considerably more intricate than in the analogous problem of finding the components of an undirected graph.

## 12.1 DIRECTED PATHS AND MUTUAL REACHABILITY

Clearly, assigning directions to edges can affect the notions of reachability and distance in a graph, but we shall see throughout the chapter how assigning directions adapts a graph to modeling processes, relationships, task organization, and operations analysis.

The term *arc* is used throughout this chapter instead of its synonym *directed edge*. Some definitions from earlier chapters are now repeated for immediate reference.

REVIEW FROM §1.4:

- A **connected** digraph is a digraph whose underlying graph is connected. Some authors use the term *weakly connected* to describe such digraphs.
- Let  $u$  and  $v$  be vertices in a digraph  $D$ . Then  $u$  and  $v$  are said to be **mutually reachable** in  $D$  if  $D$  contains both a directed  $u$ - $v$  walk and a directed  $v$ - $u$  walk. Every vertex is regarded as reachable from itself (by the trivial walk).
- A digraph  $D$  is **strongly connected** if every two vertices are mutually reachable in  $D$ .

REVIEW FROM §3.2:

- A **directed tree** is a digraph whose underlying graph is a tree. Sometimes, when the context is clear, the word “directed” is dropped.
- A **rooted tree** is a directed tree having a distinguished vertex  $r$ , called the **root**, such that for every other vertex  $v$ , there is a directed  $r$ - $v$  path. (Since the underlying graph of a rooted tree is acyclic, the directed  $r$ - $v$  path is unique.)

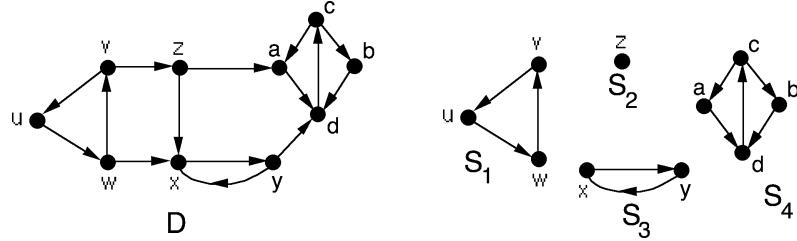
**Remark:** Designating a root in a directed tree does *not* necessarily make it a rooted tree.

### Strong Components

**DEFINITION:** A **strong component** of a digraph  $D$  is a maximal strongly connected subdigraph of  $D$ . Equivalently, a strong component is a subdigraph induced on a maximal set of mutually reachable vertices.

**Example 12.1.1:** Figure 12.1.1 below shows a digraph  $D$  and its four strong components. Notice that the vertex-sets of the strong components of  $D$  partition the vertex-set of  $D$  and that the edge-sets of the strong components do *not* include all the edges of  $D$ . This is in sharp contrast to the situation for an undirected graph  $G$ , in which the edge-sets of the components of  $G$  partition  $E_G$ . The effect that one-way edges have on the study of digraph connectivity is especially apparent when the relatively complicated strong component-finding in §12.5 is compared to the very simple tree-growing algorithm in §4.1, which finds the components of an undirected graph.

**Remark:** The analogy between a component of an undirected graph and a strong component of a digraph carries over to viewing each as an induced subgraph on an equivalence class of vertices. In particular, the components of an undirected graph  $G$  are the subgraphs induced on the equivalence classes of the reachability relation on  $V_G$  (§2.3), and the strong components of a digraph  $D$  are the subdigraphs induced on the equivalence classes of the *mutual-reachability* relation on  $V_D$ .



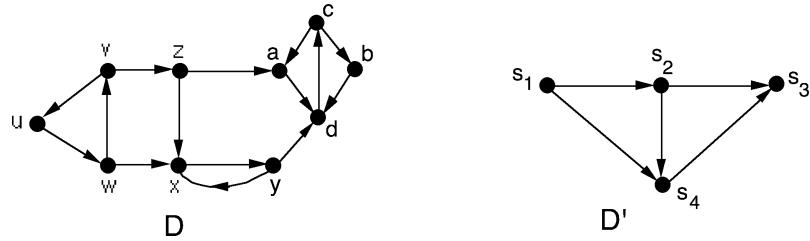
**Figure 12.1.1** A digraph and its four strong components.

**Proposition 12.1.1.** Let  $D$  be a digraph. Then the mutual-reachability relation is an equivalence relation on  $V_D$ , and the strong components of digraph  $D$  are the subdigraphs induced on the equivalence classes of this relation.  $\diamond$  (Exercises)

Corresponding to any digraph  $D$ , there is a new digraph whose definition is based on the strong components of  $D$ .

**DEFINITION:** Let  $S_1, S_2, \dots, S_r$  be the strong components of a digraph  $D$ . The **condensation** of  $D$  is the simple digraph  $D'$  with vertex set  $V_{D'} = \{s_1, s_2, \dots, s_r\}$ , such that there is an arc in digraph  $D'$  from vertex  $s_i$  to vertex  $s_j$  if and only if there is an arc in digraph  $D$  from a vertex in component  $S_i$  to a vertex in component  $S_j$ .

**Example 12.1.2:** Figure 12.1.2 shows the digraph  $D$  from the previous example and its condensation  $D'$ . Notice that the condensation  $D'$  is an acyclic digraph (i.e., it has no *directed cycles*).



**Figure 12.1.2** A digraph  $D$  and its condensation  $D'$ .

**Proposition 12.1.2.** The condensation of any digraph is an acyclic digraph.

**Proof:** A directed cycle in the condensation would contradict the maximality of any strong component that corresponds to a vertex on that cycle. The details are left as an exercise.  $\diamond$  (Exercises)

**TERMINOLOGY:** An acyclic digraph is often called a **dag** (as an acronym for *directed acyclic graph*).

### Tree-Growing Revisited

Algorithm 12.1.1, shown below, is simply the basic tree-growing algorithm of §4.1, recast for digraphs. Its output is a rooted tree whose vertices are reachable from the starting vertex. But because the paths to these vertices are directed (i.e., one-way), the vertices in this *output tree* need not be mutually reachable from one another.

If the digraph  $D$  is strongly connected, then the output tree is a spanning tree of  $D$ , regardless of the starting vertex. But for a general digraph  $D$ , the vertex-set of the output tree depends on the choice of a starting vertex, as the example following Algorithm 12.1.1 illustrates.

REVIEW FROM §4.1:

- A **frontier arc** for a rooted tree  $T$  in a digraph is an arc whose tail is in  $T$  and whose head is not in  $T$ .

**Algorithm 12.1.1: Basic Tree-Growing in a Digraph**

*Input:* a digraph  $D$  and a starting vertex  $v \in V_D$ .

*Output:* a rooted tree  $T$  with root  $v$  and a standard vertex-labeling of  $T$ .

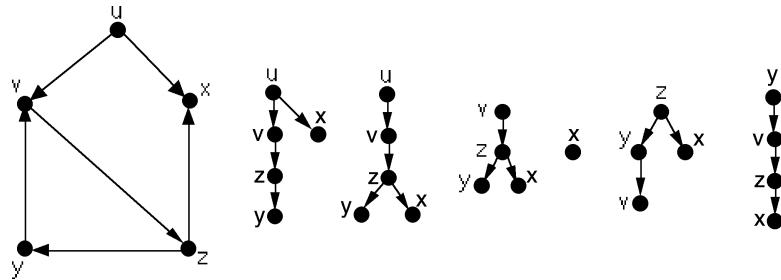
```

Initialize tree  $T$  as vertex  $v$ .
Write label 0 on vertex  $v$ .
Initialize label counter  $i := 1$ 
While there is at least one frontier arc for tree  $T$ 
    Choose a frontier arc  $e$  for tree  $T$ .
    Let  $w$  be  $head(e)$  (which lies outside of  $T$ ).
    Add arc  $e$  and vertex  $w$  to tree  $T$ .
    Write label  $i$  on vertex  $w$ .
     $i := i + 1$ 
Return tree  $T$  and vertex-labeling of  $T$ .

```

COMPUTATIONAL NOTE: We assume that there is some implicit *default priority* for choosing vertices or edges, which is invoked whenever there is more than one frontier arc from which to choose.

**Example 12.1.3:** Figure 12.1.3 shows a digraph and all possible output trees that could result for each of the different starting vertices and each possible default priority. Two opposite extremes for possible output trees are represented here. When the algorithm starts at vertex  $u$ , the output tree spans the digraph. The other extreme occurs when the algorithm starts at vertex  $x$ , because  $x$  has outdegree 0.



**Figure 12.1.3** A digraph and all possible output trees.

Notice that any two output trees in Figure 12.1.3 with the same vertex-set have roots that are mutually reachable. This property holds in general, as the following proposition asserts.

**Proposition 12.1.3.** Let  $u$  and  $v$  be two vertices of a digraph  $D$ . Then  $u$  and  $v$  are in the same strong component of  $D$  if and only if the output trees that result from starting Algorithm 12.1.2 at vertex  $u$  and at vertex  $v$  have the same vertex-set.  $\diamond$  (Exercises)

It is easy enough to modify Dijkstra's algorithm to find least-cost (or shortest) paths in directed graphs. The following application illustrates how finding least-cost directed paths might be useful in economic planning.

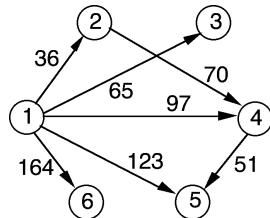
**Application 12.1.1 Equipment-Replacement Policy:** Suppose that today's price for a new car is \$16,000, and that the price will increase by \$500 for each of the next four years. The projected annual operating cost and resale value of this kind of car are shown in the table below. To simplify the setting, assume that these data do not change for the next five years. Starting with a new car today, determine a replacement policy that minimizes the net cost of owning and operating a car for the next five years.

Annual Operating Cost	Resale Value
\$600 (for 1st year of car)	\$13,000 (for a 1-year-old car)
\$900 (for 2nd year of car)	\$11,000 (for a 2-year-old car)
\$1200 (for 3rd year of car)	\$9,000 (for a 3-year-old car)
\$1600 (for 4th year of car)	\$8,000 (for a 4-year-old car)
\$2100 (for 5th year of car)	\$6,000 (for a 5-year-old car)

The digraph model has six vertices, labeled 1 through 6, representing the *beginning* of years 1 through 6. The beginning of year 6 signifies the end of the planning period. For each  $i$  and  $j$  with  $i < j$ , an arc is drawn from vertex  $i$  to vertex  $j$  and is assigned a weight  $c_{ij}$ , where  $c_{ij}$  is the total net cost of purchasing a new car at the beginning of year  $i$  and keeping it until the beginning of year  $j$ . Thus,

$$\begin{aligned} c_{ij} = & \text{ price of new car at beginning of year } i \\ & + \text{ sum of operating costs for years } i, i+1, \dots, j-1 \\ & - \text{ resale value at beginning of year } j \end{aligned}$$

For example, the net cost of buying a car at the beginning of year 2 and selling it at the beginning of year 4 is  $c_{24} = 16,500 + 600 + 900 - 11,000 = \$7,000$ . Figure 12.1.4 shows the resulting digraph with seven of its 15 arcs drawn. The arc-weights are in units of \$100.



**Figure 12.1.4** Part of the digraph model for a car-replacement problem.

The problem of determining the optimal replacement policy is reduced to finding the shortest (least-cost) path from vertex 1 to vertex 6. This is a simple task for Dijkstra's algorithm, even for much larger instances of this kind of problem.

### Characterization of Strongly Orientable Graphs

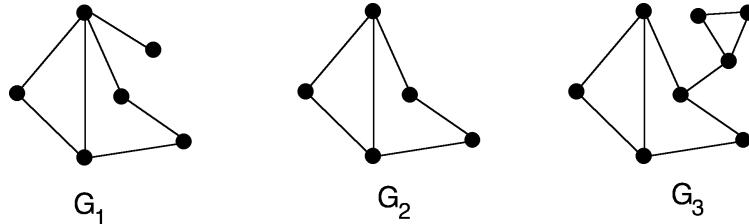
A classical theorem of Herbert Robbins arose as an application of digraph models to a whimsical traffic problem. Seeing how the one-way streets of New York sometimes necessitated roundabout driving patterns, Robbins determined the class of road-networks in which it was possible to make every street one-way, without eliminating mutual reachability between any two locations.

**REVIEW FROM §1.4:** A graph  $G$  is **strongly orientable** if there exists an assignment of directions to the edge-set of  $G$  such that the resulting digraph is strongly connected.

**REVIEW FROM §5.1 AND §5.2:**

- A graph  $G$  is **2-edge-connected** if  $G$  is connected and has no cut-edges.
- A **path addition** to a graph  $G$  is the addition to  $G$  of a path between two existing vertices of  $G$ , such that the edges and internal vertices of the path are not in  $G$ .
- A **cycle addition** is the addition to  $G$  of a cycle that has exactly one vertex in common with  $G$ .
- A **Whitney-Robbins synthesis** of a graph  $G$  from a graph  $H$  is a sequence of graphs,  $G_0, G_1, \dots, G_l$ , where  $G_0 = H$ ,  $G_l = G$ , and  $G_i$  is the result of a path or cycle addition to  $G_{i-1}$ , for  $i = 1, \dots, l$ .
- **Theorem:** A graph  $G$  is 2-edge-connected if and only if  $G$  is a cycle or a Whitney-Robbins synthesis from a cycle.

**Example 12.1.4:** Of the three graphs shown in Figure 12.1.5, only graph  $G_2$  is strongly orientable.



**Figure 12.1.5** Only one of these graphs is strongly orientable.

Notice that  $G_2$  is the only graph in the example that does not have a cut-edge. In fact, as Robbins proved, the absence of cut-edges is a necessary and sufficient condition for a graph to be strongly orientable.

**Theorem 12.1.4 [Robbins, 1939].** A connected graph  $G$  is strongly orientable if and only if  $G$  has no cut-edges.

**Proof:** ( $\Rightarrow$ ) Arguing by contrapositive, suppose that graph  $G$  has a cut-edge  $e$  joining vertices  $u$  and  $v$ . Then the only  $u-v$  path in  $G$  and the only  $v-u$  path is edge  $e$  itself (see Figure 12.1.6). Thus, for any assignment of directions to the edges of  $G$ ,  $\text{tail}(e)$  will not be reachable from  $\text{head}(e)$ .

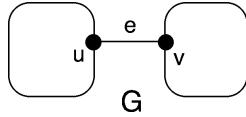


Figure 12.1.6

( $\Leftarrow$ ) Suppose that  $G$  is a connected graph with no cut-edges. By the theorem from §5.2 cited above,  $G$  is a cycle or a Whitney-Robbins synthesis from a cycle. Since a cycle is obviously strongly orientable, and strong orientability is clearly preserved under path or cycle addition, it follows that  $G$  is strongly orientable.  $\diamond$

### Markov Chains

The concept of a *Markov process* was mentioned briefly in §1.3 and is reintroduced here. Several details are omitted in order to get to the presentation of the digraph model as quickly as possible. In such a digraph, there is at most one arc from any vertex to any other vertex, but some vertices may have self-arcs.

The topic of Markov processes is part of a more general area known as *stochastic processes*, a branch of mathematics and operations research with far-ranging applications and theoretical challenges. The reader may consult any of the standard texts in this subject for a formal presentation of these concepts (e.g., [Ci75]).

Suppose that some characteristic of a phenomenon is being observed at discrete instants, labeled  $t = 0, 1, 2, \dots$ . Let  $X_t$  be the value (or **state**) of the characteristic at each such  $t$ , and suppose that  $S = \{s_1, s_2, \dots, s_n\}$  is the set of possible values (called the *state space*).

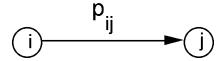
**DEFINITION:** A (*discrete-time*) **Markov chain** is a phenomenon whose behavior can be modeled by a sequence  $\{X_t\}$ ,  $t = 0, 1, 2, \dots$ , such that the (*one-step*) **transition probability**  $p_{ij}$  that  $X_{t+1} = s_j$ , given that  $X_t = s_i$ , does *not* depend on any earlier terms in the sequence  $\{X_t\}$  and does not depend on  $t$ . Thus,  $p_{ij}$  is the *conditional probability* given by

$$p_{ij} = \text{prob}(X_{t+1} = s_j | X_t = s_i), \quad \text{for } t = 1, 2, \dots$$

**TERMINOLOGY NOTE:** We are calling a *Markov chain* what experts in this area are likely to consider a special kind of Markov chain, called a *finite discrete-time stationary Markov chain*.

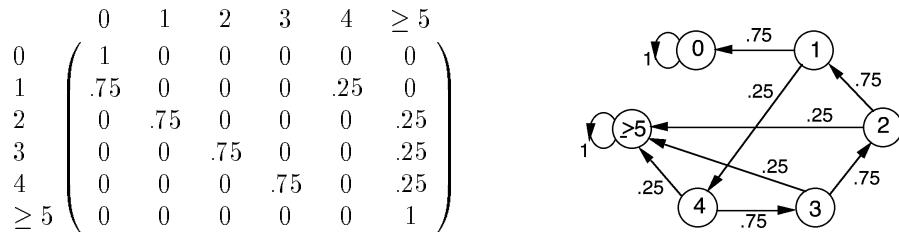
**DEFINITION:** The **transition matrix** of a Markov chain is the matrix whose  $ij$ th entry is the transition probability  $p_{ij}$ .

**DEFINITION:** The **Markov diagram** (or **Markov digraph**) of a given Markov chain is a digraph such that each vertex  $i$  corresponds to state  $s_i$ , and an arc directed from vertex  $i$  to vertex  $j$  corresponds to the transition from state  $s_i$  to state  $s_j$  and is assigned the weight  $p_{ij}$ .



Thus, the sum of the weights on the out-directed arcs from any vertex of a Markov digraph must be equal to 1.

**Application 12.1.2 A Gambler's Problem:** Suppose that a gambler starts with \$3 and agrees to play the following game. Two coins are tossed. If both come up heads, then he wins \$3; otherwise, he loses \$1. He agrees to play until either he loses all his money or he reaches a total of at least \$5. Let  $X_t$  be the amount of money he has after  $t$  plays, with  $X_0 = 3$ . The state space is  $S = \{0, 1, 2, 3, 4, 5\}$ , and the sequence  $\{X_t\}$  is a discrete-time stochastic process. Moreover, the value of  $X_{t+1}$  depends only on the amount of money the gambler has after the  $t$ th play and not on any other part of the past history. Thus, the sequence  $\{X_t\}$  is a Markov chain. The transition matrix and Markov diagram for this Markov chain are shown in Figure 12.1.7.



**Figure 12.1.7** Gambler's transition matrix and Markov diagram.

### k-Step Transition Probability

**DEFINITION:** Let  $\{X_t\}$  be a Markov chain with state space  $S = \{s_1, s_2, \dots, s_n\}$ . The conditional probability that  $X_{t+k} = s_j$ , given that  $X_t = s_i$  is called the  **$k$ -step transition probability** and is denoted  $p_{ij}^{[k]}$ . The matrix whose  $ij$ th entry is  $p_{ij}^{[k]}$  is called the  **$k$ -step transition matrix** and is denoted  $P^{[k]}$ .

The  $k$ -step transition probability has a simple interpretation in terms of the Markov digraph, and it is easily calculated from the transition matrix of the Markov chain.

**Example 12.1.5:** A company has three copier machines. During any given day, each machine that is working at the beginning of that day has a 0.1 chance of breaking down. If a machine breaks down during the day, it is sent to a repair center and will be working at the beginning of the second day after the day it broke down. Let  $X_t$  be the number of machines that are in working order at the start of day  $t$ . If at the start of a particular day, there is exactly one copier working, what is the probability that exactly two copiers are working three days later?

It is easy to verify that the transition matrix and Markov digraph are as shown in Figure 12.1.8 (see Exercises). For instance, if two are working, then the probability that exactly one of them breaks down is  $.1 \times .9 + .9 \times .1 = .18$ .



**Figure 12.1.8** Copier-machine transition matrix and Markov diagram.

The problem is to calculate  $p_{1,2}^{[3]}$ , the 3-step transition probability of going from state 1 to state 2. Each directed 1-2 walk of length 3 in the digraph represents one such 3-step probability, and the product of the arc-steps on that walk is the probability of that particular 3-step sequence of transitions. Thus, the 3-step transition probability is the sum of the products of the arc-steps of all possible walks of length 3 from vertex 1 to vertex 2. There are six different 1-2 paths of length 3, and their arc probabilities lead to the following calculation.

$$\begin{aligned} p_{1,2}^{[3]} &= (.9 \times .729 \times .243) + (.9 \times .009 \times .1) + (.9 \times .243 \times .18) \\ &\quad + (.1 \times .18 \times .18) + (.1 \times .81 \times .243) + (.1 \times .01 \times .1) \\ &= .223 \end{aligned}$$

In general, an entry of the transition matrix of a Markov chain is nonzero if and only if the corresponding entry in the adjacency matrix of the Markov diagram is nonzero. Because of this relationship, the  $k$ -step transition probabilities can be obtained by simple matrix multiplication, in much the same way that matrix multiplication can be used to determine the number of  $v_i$ - $v_j$  walks of length  $k$  in an undirected graph (§2.6).

**Proposition 12.1.5.** *Let  $P$  be the transition matrix of a Markov chain. Then the  $k$ -step transition matrix is the  $k$ th power of matrix  $P$ , i.e.,  $P^{[k]} = P^k$ .*

**Proof:** By definition of the transition matrix  $P$ , the assertion is clearly true for  $k = 1$ , which establishes the base of an induction on the number of steps  $k$ . Next, each  $i$ - $j$  path of length  $k + 1$  in the Markov diagram is the concatenation of an  $i$ - $l$  path of length  $k$  with an arc from vertex  $l$  to vertex  $j$ , for some vertex  $l$ . Thus,  $p_{ij}^{[k+1]} = \sum_l p_{il}^{[k]} \times p_{lj}$ , from which the induction step follows, by the definition of matrix multiplication. ◇

### Classification of States in a Markov Chain

The digraph model is helpful in classifying the different types of states that can occur in a Markov chain. To illustrate, we introduce the following standard terminology for Markov chains.

**DEFINITION:** A state  $s_j$  is **reachable** from state  $s_i$  in a Markov chain if there is a sequence of transitions, each having nonzero probability, from state  $s_i$  to state  $s_j$ .

Thus, the term *reachable* for Markov chains coincides with the digraph term applied to the corresponding Markov diagram.

**DEFINITION:** Two states are said to **communicate** if they are mutually reachable in the corresponding Markov diagram.

**DEFINITION:** A **communication class** in a Markov chain is a maximal subset of states that communicate with one another.

Thus, a communication class is the vertex-set of one of the strong components of the Markov diagram.

**DEFINITION:** A Markov chain is **irreducible** if every two states communicate — that is, if the Markov diagram is strongly connected.

**DEFINITION:** A state  $s_i$  is an **absorbing state** if  $p_{ii} = 1$ .

**DEFINITION:** A state  $s_i$  is a **transient state** if there exists a state  $s_j$  that is reachable from state  $s_i$ , but state  $s_i$  is not reachable from state  $s_j$ .

Thus, if the current state is  $s_i$ , there is a nonzero probability of leaving  $s_i$  and never returning.

**DEFINITION:** A state  $s_i$  is a **recurrent state** if it is not a transient state.

Observe that an absorbing state is trivially recurrent.

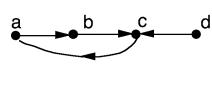
**Example 12.1.6:** The copiers Markov chain of Example 12.1.5 is irreducible, and every state is recurrent.

**Example 12.1.7:** In the Gambler's Markov chain of Application 12.1.2, the 0 state and the  $\geq 5$  state are absorbing states, and all other states are transient.

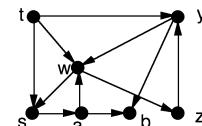
### EXERCISES for Section 12.1

For Exercises 12.1.1 through 12.1.6, identify the strong components and draw the condensation of the given digraph.

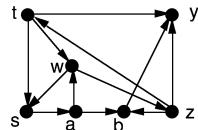
12.1.1<sup>s</sup>



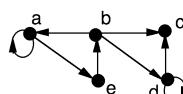
12.1.2



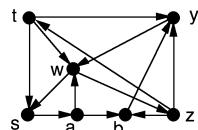
12.1.3



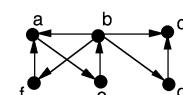
12.1.4



12.1.5



12.1.6<sup>s</sup>



For Exercises 12.1.7 through 12.1.12, apply Basic Tree-Growing Algorithm 12.1.1 to the specified digraph, and resolve ties (if any) using alphabetical order of vertices. Keep restarting the algorithm until all vertices are labeled.

12.1.7 The graph of Exercise 12.1.1.

12.1.8 The graph of Exercise 12.1.2.

12.1.9 The graph of Exercise 12.1.3.

12.1.10 The graph of Exercise 12.1.4.

12.1.11<sup>s</sup> The graph of Exercise 12.1.5.

12.1.12 The graph of Exercise 12.1.6.

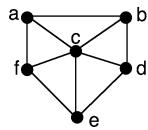
12.1.13<sup>s</sup> Complete the digraph model for the equipment-replacement problem of Application 12.1.1, and apply Dijkstra's algorithm to determine the optimal replacement policy.

**12.1.14** Referring to Application 12.1.1, suppose that the new-car price increases by \$1,000 each year. Also, assume that the resale value of a car in a given year is \$500 more than a car of the same age the previous year and that that car's operating cost is \$100 more than a comparably aged car the previous year. Modify the arc-weights accordingly and solve this version of the problem.

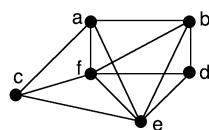
**12.1.15** Suppose that a new fax machine has just been purchased for \$200, and assume that the price stays at \$200 for the next six years. Also assume that the machine must be replaced by a new machine after five years, but that it can be replaced sooner if desired. The estimated maintenance cost for each year of operation is \$80 for the first year, \$120 for the second, \$160 for the third, \$240 for the fourth, and \$280 for the fifth year. Determine a replacement policy to minimize the total cost of purchasing and operating a fax machine for the next six years.

*For Exercises 12.1.16 through 12.1.21, find a strong orientation of the given graph.*

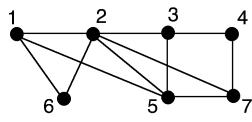
12.1.16



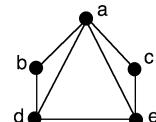
12.1.17



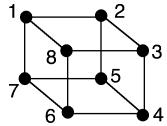
12.1.18<sup>s</sup>



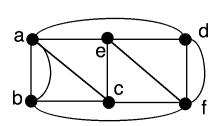
12.1.19



12.1.20



12.1.21



**12.1.22<sup>s</sup>** Find an orientation of the complete graph  $K_5$  that is a dag.

**12.1.23** Complete the proof of Proposition 12.1.2.

**12.1.24<sup>s</sup>** Characterize those digraphs that are the same as their condensation.

**12.1.25** Prove Proposition 12.1.1.

**12.1.26** Prove Proposition 12.1.3.

**12.1.27** Prove that two vertices are in the same strong component of a given digraph if and only if every output tree that contains one of them also contains the other.

**12.1.28<sup>s</sup>** Prove or disprove: Let  $G$  be a graph with  $k$  cut-edges, and let  $D$  be a digraph that results from assigning a direction to each edge of  $G$ . Then digraph  $D$  has at least  $k + 1$  strong components.

**12.1.29** Verify that the transition matrix and Markov diagram for the Gambler's problem of Application 12.1.2 are correct.

**12.1.30<sup>s</sup>** Determine the 2-step transition matrix for the Markov chain of Application 12.1.2, by working directly with the Markov diagram. Check your result by applying Proposition 12.1.5.

**12.1.31** Verify that the transition matrix and Markov diagram for the copier-machine Markov chain of Example 12.1.5 are correct.

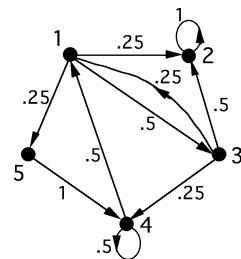
12.1.32 Referring to Example 12.1.5, calculate  $p_{2,1}^{[3]}$  by working directly with the Markov diagram.

12.1.33 Determine the 2-step transition matrix for the Markov chain of Example 12.1.5 by working directly with the Markov diagram. Check your result by applying Proposition 12.1.5.

12.1.34<sup>s</sup> Consider the Markov chain whose Markov diagram is shown.

a. Classify each of the states.

b. Calculate  $p_{34}^{[4]}$ .



12.1.35 Find the 2-step transition matrix for the Markov diagram in the previous exercise, by working directly with the digraph. Check your results by applying Proposition 12.1.5.

12.1.36 **Application 12.1.3 Random Walks:** Suppose that a person who has trouble making up his mind is standing on a straight road between two locations,  $A$  and  $B$ , 400 yards apart. After each minute he chooses from one of three options of what to do during the next minute. With probability  $\frac{1}{2}$ , he walks 100 yards toward  $A$ ; with probability  $\frac{1}{3}$ , he walks 100 yards toward  $B$ ; and with probability  $\frac{1}{6}$ , he stays where he is and thinks about it for the next minute. His position after each minute is one of five possible locations, as shown in the figure. Assume that if he arrives at either  $A$  or  $B$ , he remains there forever.



- a. Draw the Markov diagram.
- b. Determine the probability that he is at position 3 after two minutes, given that he starts there.
- c. Find the 2-step transition matrix by working directly with the digraph, and then check it by applying Proposition 12.1.5.
- d. Classify each of the states.

12.1.37<sup>s</sup> Prove that every finite Markov chain has at least one recurrent state.

12.1.38 [Computer Project] Implement Basic Tree-Growing Algorithm 12.1.1, and run the program on each of the digraphs in Exercises 12.1.1 through 12.1.6.

12.1.39 [Computer Project] Write a program whose input is a graph  $G$  with no cut-edges and whose output is a strong orientation of  $G$ . (Hint: see the second part of the proof of Theorem 12.1.4.)

## 12.2 DIGRAPHS AS MODELS FOR RELATIONS

**DEFINITION:** The **digraph representation of a relation**  $R$  on a finite set  $S$  is the digraph whose vertices correspond to the elements of  $S$ , and whose arcs correspond to the ordered pairs in the relation. That is, an arc is drawn from vertex  $x$  to vertex  $y$  if  $(x, y) \in R$ .<sup>†</sup>

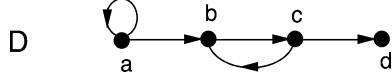
Conversely, a digraph  $D$  induces a relation  $R$  on  $V_D$  in a natural way, namely,  $(x, y) \in R$  if and only if there is an arc in digraph  $D$  from vertex  $x$  to vertex  $y$ .

**Remark:** Because the digraph representation of a relation has no multi-arcs, an arc can be specified by the ordered pair it represents. We adopt this convention for this subsection.

**Example 12.2.1:** Suppose a relation  $R$  on the set  $S = \{a, b, c, d\}$  is given by

$$\{(a, a), (a, b), (b, c), (c, b), (c, d)\}$$

Then the digraph  $D$  representing the relation  $R$  is as shown in Figure 12.2.1.



**Figure 12.2.1** The digraph  $D$  representing the relation  $R$ .

### The Transitive Closure of a Digraph

**DEFINITION:** A **transitive digraph** is a digraph whose corresponding relation is transitive. That is, if there is an arc from vertex  $x$  to vertex  $y$  and an arc from  $y$  to  $z$ , then there is an arc from  $x$  to  $z$ .

**DEFINITION:** The **transitive closure**  $R^*$  of a binary relation  $R$  is the relation  $R^*$  defined by  $(x, y) \in R^*$  if and only if there exists a sequence  $x = v_0, v_1, v_2, \dots, v_k = y$  such that  $k \geq 1$  and  $(v_i, v_{i+1}) \in R$ , for  $i = 0, 1, \dots, k - 1$ .

Equivalently, the transitive closure  $R^*$  of the relation  $R$  is the smallest transitive relation that contains  $R$ .

**DEFINITION:** Let  $D$  be the digraph representing a relation  $R$ . Then the digraph  $D^*$  representing the transitive closure  $R^*$  of  $R$  is called the **transitive closure of the digraph  $D$** .

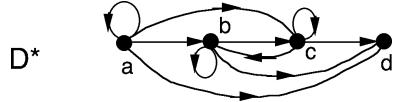
Thus, an arc  $(x, y)$ ,  $x \neq y$  is in the transitive closure  $D^*$  if and only if there is a directed  $x$ - $y$  path in  $D$ . Similarly, there is a self-loop in digraph  $D^*$  at vertex  $x$  if and only if there is a directed cycle in digraph  $D$  that contains  $x$ .

**Remark:** In many applications, the transitive closure of a digraph is used without making any explicit reference to the underlying relation.

---

<sup>†</sup> Relations and related terminology appear in Appendix A.2.

**Example 12.2.2:** The transitive closure of the digraph in Figure 12.2.1 is shown in Figure 12.2.2.



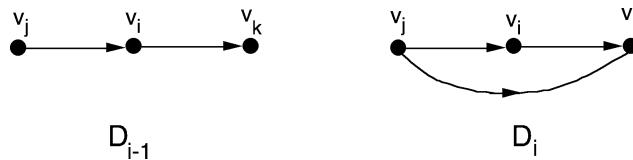
**Figure 12.2.2** The transitive closure  $D^*$  of digraph  $D$ .

**Application 12.2.1 Transitive Closure in a Paging Network:** Suppose that the arcs of an  $n$ -vertex digraph  $D$  represent the one-way direct links between specified pairs of nodes in an  $n$ -node paging network. Thus, an arc from vertex  $i$  to vertex  $j$  indicates that a page call can be transmitted from person  $i$  to person  $j$ .

To send an alert from person  $i$  to person  $j$ , it is not necessary to have a direct link from  $i$  to  $j$ . There need only be a directed  $i$ - $j$  path. The transitive closure  $D^*$  of digraph  $D$  specifies all pairs  $i, j$  of vertices for which there exists a directed  $i$ - $j$  path in  $D$ .

### Constructing the Transitive Closure of a Digraph: Marshall's Algorithm

Let  $D$  be an  $n$ -vertex digraph with vertices  $v_1, v_2, \dots, v_n$ . A computationally efficient algorithm, due to Marshall, constructs a sequence of digraphs,  $D_0, D_1, \dots, D_n$ , such that  $D_0 = D$ ,  $D_{i-1}$  is a subgraph of  $D_i$ ,  $i = 1, \dots, n$ , and such that  $D_n$  is the transitive closure of  $D$ . Digraph  $D_i$  is obtained from digraph  $D_{i-1}$  by adding to  $D_{i-1}$  an arc  $(v_j, v_k)$  (if it is not already in  $D_{i-1}$ ) if there is a directed path of length 2 in  $D_{i-1}$  from  $v_j$  to  $v_k$ , having  $v_i$  as the internal vertex.



**Figure 12.2.3** The arc  $(v_j, v_k)$  is added to digraph  $D_{i-1}$ .

#### Algorithm 12.2.1: Marshall's Transitive Closure

*Input:* an  $n$ -vertex digraph  $D$  with vertices  $v_1, v_2, \dots, v_n$ .  
*Output:* the transitive closure of digraph  $D$ .

```

Initialize digraph  $D_0$  to be digraph  $D$ .
For  $i = 1$  to  $n$ 
  For  $j = 1$  to  $n$ 
    If  $(v_j, v_i)$  is an arc in digraph  $D_{i-1}$ 
      For  $k = 1$  to  $n$ 
        If  $(v_i, v_k)$  is an arc in digraph  $D_{i-1}$ 
          Add arc  $(v_j, v_k)$  to  $D_{i-1}$  (if it is not already there).
Return digraph  $D_n$ .

```

**COMPUTATIONAL NOTE:** An efficient implementation of the algorithm uses the *adjacency matrix*  $A_D$  of the digraph  $D$  (§2.6), where the  $ij$ th element of  $A_D$  is given by

$$a_{jk} = \begin{cases} 1, & \text{if } (v_j, v_k) \text{ is an arc in } D \\ 0, & \text{otherwise} \end{cases}$$

If the values 0 and 1 are interpreted as *false* and *true*, respectively, then the innermost loop body can be replaced by  $a_{jk} := a_{jk} \vee (a_{ji} \wedge a_{ik})$ .

The correctness of Algorithm 12.2.1 is a consequence of the following proposition.

**Proposition 12.2.1.** Let  $D$  be an  $n$ -vertex digraph with vertices  $v_1, v_2, \dots, v_n$ , and let  $D_i$  be the digraph that results from the  $i$ th iteration of Algorithm 12.2.1. Let  $x$  and  $y$  be any two vertices in  $D$  for which there exists a directed  $x$ - $y$  path in  $D$  whose internal vertices are from the set  $\{v_1, v_2, \dots, v_i\}$ . Then digraph  $D_i$  contains the arc  $(x, y)$ .

**Proof:** The assertion is true for  $i = 1$ , because in the first iteration of the algorithm, an arc is added to digraph  $D_0 (=D)$  if  $D_0$  contains the arcs  $(x, v_1)$  and  $(v_1, y)$ . By way of induction, assume that the assertion is true for  $D_k$ , for all  $k < i$ .

Next, let  $x$  and  $y$  be vertices in  $D$  such that there exists a directed  $x$ - $y$  path  $P$  whose internal vertices are from the set  $\{v_1, v_2, \dots, v_i\}$ . If vertex  $v_i$  is not an internal vertex of path  $P$ , then it follows from the induction hypothesis that digraph  $D_{i-1}$  contains arc  $(x, y)$ , and, hence, so does digraph  $D_i$ . If vertex  $v_i$  is an internal vertex of path  $P$ , then by the induction hypothesis, digraph  $D_{i-1}$  contains arcs  $(x, v_i)$  and  $(v_i, y)$ . Hence, arc  $(x, y)$  is added during the  $i$ th iteration.  $\diamond$

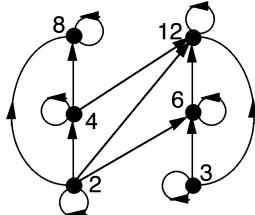
### Partial Orders, Hasse Diagrams, and Linear Extensions

**NOTATION:** Throughout the rest of this subsection, the symbol  $\prec$  is used to denote the relation under discussion.

**FROM APPENDIX A.2:** A relation  $\prec$  on a set  $S$  is called a **partial order** if  $\prec$  is reflexive, antisymmetric, and transitive.

**TERMINOLOGY:** The pair  $(S, \prec)$  is called a **partially ordered set** or **poset**.

**Example 12.2.3:** Let  $S = \{2, 3, 4, 6, 8, 12\}$ , and let  $\prec$  be the relation on set  $S$  given by  $x \prec y \iff x \text{ divides } y$  (i.e.,  $\frac{y}{x}$  is an integer). Then  $\prec$  is a partial order on set  $S$ , and its digraph representation is shown in Figure 12.2.4.



**Figure 12.2.4** Representing the divides relation.

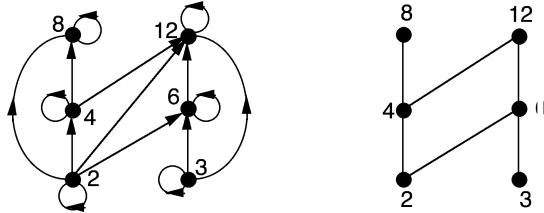
A partial order on a finite set is often represented by a simplified graph drawing called a *Hasse diagram*.

**DEFINITION:** Let  $\prec$  be a partial order on a finite set  $S$ . A **Hasse diagram** representing the poset  $(S, \prec)$  is a graph drawing defined as follows:

- Each element of set  $S$  is represented by a vertex.
- If  $x \prec y$  for distinct elements  $x$  and  $y$ , then the vertex for  $y$  is positioned higher than the vertex for  $x$ ; and if there is no  $w$  different from both  $x$  and  $y$  such that  $x \prec w$  and  $w \prec y$ , then an edge is drawn from vertex  $x$  upward to vertex  $y$ .

The effect of the first condition is to allow arcs to be represented by undirected edges, since all arcs are directed upward. The last condition avoids redundant lines that are implied by transitivity. Notice also that the self-loops of the poset are absent from the Hasse diagram.

**Example 12.2.4:** The digraph representation and its Hasse diagram for the *divides* poset of the previous example are shown in Figure 12.2.5.



**Figure 12.2.5** Digraph and Hasse diagram for the *divides* relation.

Notice that if  $\vec{H}$  denotes the digraph obtained by replacing each edge of the Hasse diagram for the example poset by an arc directed upward, then the transitive closure of  $\vec{H}$  recaptures the digraph representation of the poset, except for the self-loops. This property holds for all posets, by the definition of Hasse diagram.

**DEFINITION:** Two elements  $x$  and  $y$  of a poset  $(S, \prec)$  are said to be **comparable** if either  $x \prec y$  or  $y \prec x$ . Otherwise,  $x$  and  $y$  are **incomparable**.

**DEFINITION:** A **total order** on a set  $S$  is a partial order on  $S$  such that every two elements are comparable.

**Example 12.2.5:** *Lexicographic order* is a total order on the set of words in a dictionary.

**Example 12.2.6:** The *divides* partial order shown in Figure 12.2.5 is *not* a total order, because there are several incomparable elements (e.g., 3 and 8 are not comparable).

**Remark:** Because the Hasse diagram of a finite total order can be drawn as a path graph, it should not be surprising that the term *linear order* is a synonym for total order.

**DEFINITION:** A partial order  $\prec'$  on a set  $S$  is an **extension of a partial order**  $\prec$  on  $S$  (or is **compatible with** that partial order) if  $x \prec y \Rightarrow x \prec' y$ .

**DEFINITION:** A **linear extension of a poset**  $(S, \prec)$  is a total order  $\prec^t$  on  $S$  that is compatible with  $\prec$ .

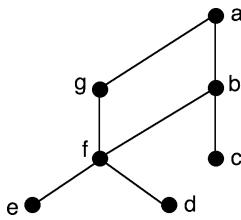
**DEFINITION:** A **topological sort** is a process that produces a linear extension of a poset. It also refers informally to the resulting linear extension.

**Application 12.2.2 Job Sequencing on a Single Machine:** Suppose a procedure, consisting of several operations, must be performed on a single machine. There is a natural partial order defined on the set of operations. For any two operations  $x$  and  $y$ ,

$$x \prec y \iff \text{operation } x \text{ cannot occur after operation } y$$

A linear extension of this poset would solve the problem of sequencing the operations on the machine.

**Illustration:** Suppose the Hasse diagram shown in Figure 12.2.6 represents the precedence relations among the set  $T = \{a, b, c, d, e, f, g\}$  of operations required to perform a particular procedure.



**Figure 12.2.6** Hasse diagram for a set of operations.

There is a simple algorithm to construct a linear extension of a poset. The idea is always to choose a **minimal element** at each step.

**DEFINITION:** A **minimal element** of a poset  $(S, \prec)$  is an element  $m \in S$  such that for all other  $x \in S$ ,  $x \not\prec m$ .

In terms of a Hasse diagram, a minimal element is a vertex that has no vertex positioned below it and joined to it by an edge.

**Algorithm 12.2.2: Linear Extension of a Poset**

*Input:* an  $n$ -element poset  $(S, \prec)$ .

*Output:* a permutation  $\langle s_1, s_2, \dots, s_n \rangle$  of the elements of  $S$  that represents a linear extension of poset  $(S, \prec)$ .

For  $i = 1$  to  $n$

    Let  $s_i$  be a minimal element of poset  $(S, \prec)$ .

$S := S - \{s_i\}$

    Return  $\langle s_1, s_2, \dots, s_n \rangle$ .

**COMPUTATIONAL NOTE:** At any iteration of Algorithm 12.2.2, there may be more than one minimal element from which to choose. Assume that there is some default ordering of the elements of  $S$  to resolve ties.

**Example 12.2.7:** The sequence of Hasse diagrams shown below illustrates Algorithm 12.2.2 for the poset whose Hasse diagram is shown in Figure 12.2.6. Below each Hasse diagram is the total order constructed so far, and the final iteration produces the total order  $\langle c, d, e, f, b, g, a \rangle$ . Ties are resolved alphabetically.

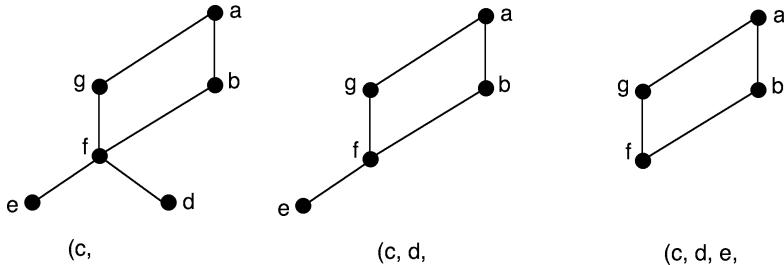


Figure 12.2.7 Iterations 1 through 3.

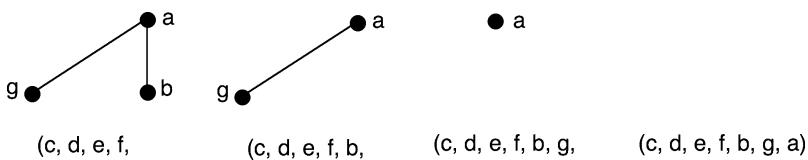
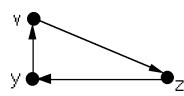


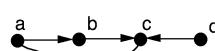
Figure 12.2.8 Iterations 4 through 7.

## EXERCISES for Section 12.2

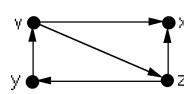
For Exercises 12.2.1 through 12.2.6, draw the transitive closure of the given digraph.

12.2.1<sup>s</sup>

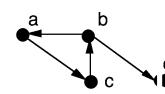
12.2.2



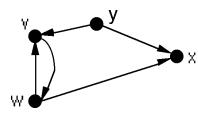
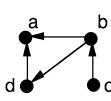
12.2.3



12.2.4



12.2.5

12.2.6<sup>s</sup>

For Exercises 12.2.7 through 12.2.12, apply Warshall's Algorithm 12.2.1 to the digraph specified.

12.2.7 The graph of Exercise 12.2.1.

12.2.8 The graph of Exercise 12.2.2.

12.2.9 The graph of Exercise 12.2.3.

12.2.10<sup>s</sup> The graph of Exercise 12.2.4.

12.2.11 The graph of Exercise 12.2.5.

12.2.12 The graph of Exercise 12.2.6.

For Exercises 12.2.13 through 12.2.15, draw the Hasse diagram for the given non-reflexive poset and identify all the minimal elements.

12.2.13  $(S, \prec)$ , where  $S = \{1, 2, 3, 4, 5, 6\}$  and  $x \prec y \iff x > y$ .12.2.14<sup>s</sup>  $(S, \prec)$ , where  $S = \{2, 3, 4, 5, 6, 8, 9, 72\}$  and  $x \prec y \iff x$  properly divides  $y$ .12.2.15  $(S, \prec)$ , where  $S$  is the set of all subsets of  $A = \{1, 2, 3\}$  and  $x \prec y \iff x$  is a proper subset of  $y$ .

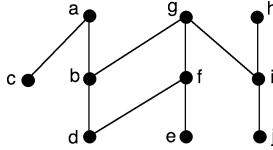
For Exercises 12.2.16 through 12.2.18, construct a linear extension for the specified poset.

12.2.16 The poset of Exercise 12.2.13.

12.2.17<sup>s</sup> The poset of Exercise 12.2.14.

12.2.18 The poset of Exercise 12.2.15.

12.2.19 Construct a linear extension for the poset whose Hasse diagram is shown below. (Resolve ties alphabetically.)



12.2.20 [Computer Project] Implement Warshall's Algorithm 12.2.1 and test the program on each of the digraphs in Exercises 12.2.1 through 12.2.6.

## 12.3 TOURNAMENTS

The applications discussed in this section use models with no multi-arcs or self-arcs, and it will sometimes be convenient for discussion purposes to refer to an arc by its ordered pair of endpoints. However, the more flexible computer representations of digraphs are unlikely to represent arcs in this way.

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

Thus, a tournament is obtained by assigning a direction to each edge of a complete graph.

### Transitive Tournaments

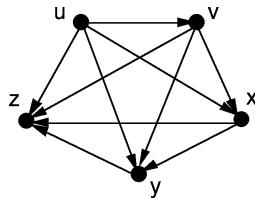
Much of the interest in tournament models to is determine the winner of a competition. This pursuit sometimes leads to paradoxes.

**DEFINITION:** A **transitive tournament** is a tournament that is transitive as a digraph.

**Example 12.3.1:** Each participant in a *round-robin* tennis competition plays exactly one match with every other player. If each player  $v$  is represented by a vertex  $v$ , and if an arc from  $u$  to  $v$  means that player  $u$  beat player  $v$ , then the outcome of the complete round-robin competition is represented by a tournament. Figure 12.3.1 below shows the outcome of a 5-person round-robin for which the representing digraph is a transitive tournament.

**Proposition 12.3.1.** A tournament is transitive if and only if it is acyclic.

**Proof:** Suppose that a tournament  $T$  is transitive, and by way of contradiction, assume that  $T$  contains a directed cycle  $C = \langle v_1, v_2, \dots, v_k, v_1 \rangle$ . By transitivity, there is an arc from vertex  $v_1$  to vertex  $v_k$ , which contradicts the existence of the arc from  $v_k$  to  $v_1$ .



**Figure 12.3.1** A 5-vertex transitive tournament.

Conversely, let  $T$  be an acyclic tournament, and suppose that there is an arc from vertex  $x$  to vertex  $y$  and an arc from  $y$  to  $z$ . If the arc between vertex  $x$  and vertex  $z$  were directed from  $z$  to  $x$ , then there would be a directed cycle in  $T$ . Hence, that arc must be directed from  $x$  to  $z$ .  $\diamond$

**Example 12.3.2:** In a transitive tennis tournament, there are no ambiguities regarding the relative strengths of the players. That is, if player  $x$  beat  $y$  and player  $y$  beat  $z$ , then player  $x$  will have beaten  $z$ . The winner beat everyone, and the second place finisher beat everyone except the winner, etc.

**TERMINOLOGY:** In a tournament, if there is an arc directed from  $x$  to  $y$ , we say  $x$  **dominates** (or **beats**)  $y$ .

Notice that the number of matches that a player  $x$  wins is equal to  $\text{outdegree}(x)$ . The next proposition shows that the player rankings from a transitive tournament are clearcut and indisputable.

### Score Sequence

**DEFINITION:** The **score sequence** of a tournament is the outdegree sequence of the digraph.

**Proposition 12.3.2.** An  $n$ -vertex tournament  $T$  is transitive if and only if the score sequence of  $T$  is  $\langle 0, 1, \dots, n - 1 \rangle$ .

**Proof:** ( $\Rightarrow$ ) Suppose that tournament  $T$  is transitive. Since  $T$  is a simple digraph, it suffices to show that all of its  $n$  vertices have different outdegrees. Let  $x$  and  $y$  be any two vertices in  $T$ , and assume, without loss of generality, that  $(x, y)$  is an arc in  $T$ . Next, let  $U$  be the set of all vertices in  $T$  that are adjacent from vertex  $y$ . That is,  $U = \{u \in V_T | (y, u) \text{ is an arc in } T\}$ , from which it follows that  $\text{outdegree}(y) = |U|$ . For each  $u \in U$ , the transitivity of tournament  $T$  implies that  $(x, u)$  is an arc in  $T$ . Thus,  $\text{outdegree}(x) \geq |U| + 1 > \text{outdegree}(y)$ .

( $\Leftarrow$ ) Suppose that the score sequence of tournament  $T$  is  $\langle 0, 1, \dots, n - 1 \rangle$ . Relabel the vertices of  $T$  so that  $\text{outdegree}(v_i) = i$  for  $i = 0, 1, \dots, n - 1$ . Then there is an arc from vertex  $v_{n-1}$  to each of the vertices  $v_0, v_1, \dots, v_{n-2}$ , there is an arc from vertex  $v_{n-2}$  to each of the vertices  $v_0, v_1, \dots, v_{n-3}$ , and so on. In other words, there is an arc from vertex  $v_i$  to vertex  $v_j$  if and only if  $i > j$ , from which transitivity follows.  $\diamond$

**Corollary 12.3.3.** Every transitive tournament contains exactly one hamiltonian path.

**Proof:** The relabeling of the vertices given in the proof of Proposition 12.3.2 produces the vertex sequence of a hamiltonian path. Its uniqueness is left as an exercise (see Exercises).  $\diamond$

**Remark:** This last result reaffirms the notion that a transitive tournament has an indisputable ranking. Although there is no clearcut ranking for non-transitive tournaments, one can use additional criteria to choose the “best” of the hamiltonian paths. The next result guarantees that there is at least one from which to choose.

**Proposition 12.3.4 [Rédei, 1934].** *Every tournament contains a hamiltonian path.*

**Proof:** Let  $T$  be an  $n$ -vertex tournament, and let  $P = \langle v_1, v_2, \dots, v_l \rangle$  be a path in  $T$  of maximum length. By way of contradiction, suppose that  $l < n$ . Then there is some vertex  $w$  not on path  $P$ . By the maximality of  $P$ , the arc between vertices  $w$  and  $v_1$  must be directed from  $v_1$  to  $w$ , and, similarly,  $(w, v_l)$  must be an arc in  $T$ . If  $(w, v_2)$  were an arc in  $T$ , then the path  $\langle v_1, w, v_2, \dots, v_l \rangle$  would contradict the maximality of  $P$ . It follows that  $(v_2, w)$  is an arc, and similarly,  $(v_j, w)$  is an arc in  $T$ , for all  $j = 1, \dots, l - 1$ . In particular, the vertex sequence  $\langle v_1, w, v_2, \dots, v_{l-1}, w, v_l \rangle$  forms a path that contradicts the maximality of  $P$ .  $\diamond$

**Remark:** *Tournament Rankings.* Given the results of a round-robin competition, one would like to rank the teams or at least pick a clearcut winner. Ranking by the order of a hamiltonian path will not work unless the path is unique. Ranking by score sequence usually results in ties, and a team that beats only a few teams, with those few teams having lots of wins, might deserve a better ranking. This suggests considering the *second-order score sequence*, where each team’s score is the sum of the outdegrees of the teams it beats. One can continue by defining the  $k$ th-order score sequences recursively. See [Mo68] for more details and references.

The following result shows that the vertex-sets of the strong components can be ranked.

**Proposition 12.3.5.** *The condensation of a tournament is transitive.*

**Proof:** This is an immediate consequence of Propositions 12.1.2 and 12.3.1.  $\diamond$

**Application 12.3.1 Partitioning a Sports League into Divisions:** Suppose that an adult soccer league is to consist of twelve teams. To avoid having a few teams dominate the rest of the league, the organizers would like to group the teams into divisions of comparable strength. One way to achieve this is to have a round-robin competition involving the twelve teams and then use the strong components of the outcome to form the divisions. For an overview of round-robin sports competition scheduling and its relationship to edge-coloring, see [BuWeKi04].

### Regular Tournaments and Kelly’s Conjecture

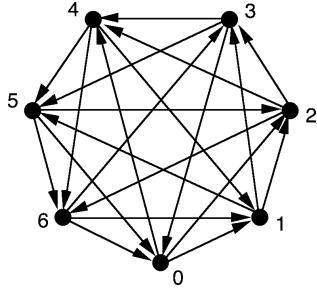
**DEFINITION:** A **regular tournament** is a tournament  $T$  in which all scores are the same, i.e., there is an integer  $s$  such that  $\text{outdegree}(v) = s$  for all vertices  $v \in V_T$ .

**Example 12.3.3:** A regular 7-vertex tournament is shown in Figure 12.3.2 below. Notice that the arc-set of this regular tournament can be partitioned into three hamiltonian cycles:

$$0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 0$$

$$0 \rightarrow 2 \rightarrow 4 \rightarrow 6 \rightarrow 1 \rightarrow 3 \rightarrow 5 \rightarrow 0$$

$$0 \rightarrow 4 \rightarrow 1 \rightarrow 5 \rightarrow 2 \rightarrow 6 \rightarrow 3 \rightarrow 0$$



**Figure 12.3.2** A regular 7-vertex tournament.

**Kelly's Conjecture** (see [Mo68]). The arc-set of a regular  $n$ -vertex tournament can be partitioned into  $(n - 1)/2$  subsets, each of which induces a hamiltonian cycle.

**Remark:** B. Alspach has shown that the conjecture is true for  $n \geq 9$ . For this result and other evidence for the conjecture, see [BeTh81], [Zh80], [Th82], and [Hä93]. Partitioning the arc-set is a special case of *graph factorization*, which is discussed in §9.4.

### Tournament Kings

H. G. Landau [La53] developed the idea of a *king* in attempting to determine the “strongest” individuals in certain animal societies in which there exists a pairwise *pecking* relationship.

**DEFINITION:** A **king** in a tournament  $T$  is a vertex  $x$  such that every vertex in  $T$  is reachable from  $x$  by a directed path of length 2 or less.

**Proposition 12.3.6.** *Every tournament has a king.*

**Proof:** We use induction on the number of vertices. The lone vertex in a 1-vertex tournament is trivially a king. Assume for some  $n \geq 1$  that every  $n$ -vertex tournament has a king. Let  $T$  be an  $(n + 1)$ -vertex tournament, and let  $v$  be any vertex in  $T$ . By the induction hypothesis, the tournament  $T - v$  has a king, say  $x$ . Let  $D$  be the set containing vertex  $x$  and all vertices that  $x$  dominates. If any vertex in  $D$  dominates vertex  $v$ , then  $x$  is a king in tournament  $T$ . Otherwise,  $v$  dominates every vertex in  $D$ , in which case,  $v$  is a king in  $T$ . ◇

**Remark:** The article by S. Maurer [Ma80] generated early interest in the topic of kings. For a discussion of kings and generalizations in other digraphs, see [Re96].

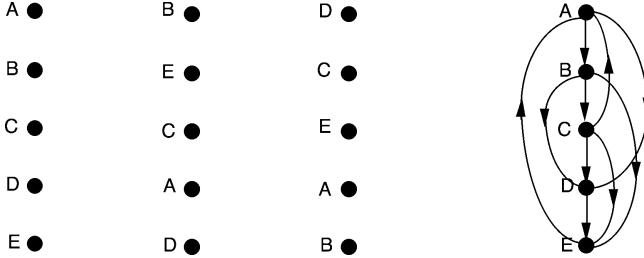
### Voting and the Condorcet Paradox

An  $n$ -vertex tournament is a natural way to model a voter’s pairwise preferences among  $n$  candidates (or alternatives). Each of the  $n$  candidates is represented by a vertex, and an arc is directed from  $x$  to  $y$  if the voter prefers candidate  $x$  to candidate  $y$ .

**DEFINITION:** Let  $\mathcal{T}$  be a set of tournaments, each having the same vertex-set  $V$ . The **majority digraph**  $D$  of set  $\mathcal{T}$  has vertex-set  $V$ , and there is an arc directed from vertex  $x$  to vertex  $y$  in  $D$  if and only if  $x$  dominates  $y$  in a majority of the tournaments.

**Remark:** If each of the tournaments represents one voter's preferences, then the resulting majority digraph represents the voters' preferences under majority voting. If there is an odd number of voters or there are no ties, then the majority digraph is a tournament.

**Example 12.3.4:** Figure 12.3.3 shows the majority digraph of a set of three transitive 5-vertex tournaments, representing three voters' pairwise preferences for five candidates. Each vertical line of vertices represents a tournament. Each vertex dominates exactly the vertices below it. For example, in the first tournament on the left, vertex  $C$  dominates exactly vertices  $D$  and  $E$ , and vertex  $D$  is dominated by exactly vertices  $A$ ,  $B$ , and  $C$ .



**Figure 12.3.3** The majority digraph of three tournaments.

**DEFINITION:** The **Condorcet winner** is a candidate  $x$  such that for every other candidate  $y$ ,  $x$  is preferred over  $y$  by a majority of the voters.

Thus, a Condorcet winner corresponds to a vertex in the majority digraph that has an arc directed to every other vertex.

**DEFINITION:** The **Condorcet paradox** occurs when the majority digraph of a set of tournaments contains a directed cycle.

**Example 12.3.4, continued:** The existence of the directed cycle  $\langle A, B, C \rangle$  in the majority digraph of Figure 12.3.3 illustrates the Condorcet paradox. We see that a majority of voters prefer  $A$  to  $B$ , a majority prefer  $B$  to  $C$ , and yet, a majority prefer  $C$  to  $A$ . Thus, if candidate  $A$  wins, then a majority of the voters would have been happier if  $C$  had won, if candidate  $C$  wins, then a majority of the voters would have been happier if  $B$  had won, and if candidate  $B$  wins, then a majority of the voters would have been happier if  $A$  had won.

**Remark:** There is a rich source of literature in voting theory appearing in periodicals such as *Public Choice*, *Social Choice and Welfare*, and *Mathematical Social Sciences*.

### EXERCISES for Section 12.3

- 12.3.1<sup>s</sup> Group the 3-vertex tournaments into isomorphism classes.
- 12.3.2 Construct a 5-vertex tournament with at least two vertices of maximum out-degree.
- 12.3.3<sup>s</sup> Determine the number of non-isomorphic 4-vertex tournaments.
- 12.3.4 Show that every tournament has at most one vertex of outdegree 0.
- 12.3.5 Prove that every vertex of maximum outdegree in a tournament is a king.

12.3.6 Let  $T$  be a tournament having no vertex with indegree 0, and let  $x$  be a king. Prove that there is some vertex that dominates  $x$  and is also a king.

12.3.7<sup>s</sup> Prove that a transitive tournament with at least two vertices cannot be strongly connected.

12.3.8 Complete the proof of Corollary 12.3.3.

12.3.9 Prove that every tournament is either strongly connected or can be transformed into a strongly connected tournament by reversing the direction on one arc.

12.3.10 Prove that a non-transitive tournament contains at least three hamiltonian paths.

## 12.4 PROJECT SCHEDULING

A digraph model can be used as an aid in the scheduling of large complex projects that consist of several interrelated activities. Typically, some of the activities can occur simultaneously, but some activities cannot begin until certain others are completed. For instance, in building a house, the electrical work cannot start until the framing is complete. Two desirable objectives are to schedule the activities so that the project completion time is minimized and to identify those activities whose delay will delay the overall completion time.

### The Critical-Path Method for Project Scheduling

If the duration of each activity in the project is known in advance, then the **critical-path method (CPM)** can be used to solve this problem. CPM can also be used to determine how long each activity in the project can be delayed without delaying the completion of the project.

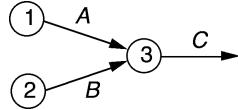
One way to represent the precedences among the various activities is to use a digraph called an **AOA (activity-on-arc) network**. Each arc in the digraph represents an activity (or task), with its *head* indicating the direction of progress in the project. The weight of the arc is the duration time of that activity. Each vertex in the AOA network represents an **event** that signifies the completion of one or more activities and the beginning of new ones.

**TERMINOLOGY:** Activity  $A$  is said to be a **predecessor** of activity  $B$  if  $B$  cannot begin until  $A$  is completed.

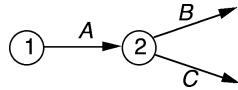
**Example 12.4.1:** The next few figures illustrate how various precedences and events may be depicted in an AOA network. In each figure, the activities are represented by arcs  $A$ ,  $B$ , and  $C$ , and the events are represented by vertices 1, 2, and 3.



**Figure 12.4.1** Event 2 marks the completion of activity  $A$  and the start of  $B$ .

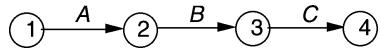


**Figure 12.4.2** Activities  $A$  and  $B$  are predecessors of activity  $C$ .



**Figure 12.4.3** Activity  $A$  is the predecessor of both  $B$  and  $C$ .

When listing the predecessors of an activity, only the *immediate* predecessors are specified. For instance, in the figure below, activity  $A$  is an immediate predecessor of activity  $B$ , and activity  $B$  is an immediate predecessor of activity  $C$ . The predecessor list for  $C$  does not include activity  $A$ , since its precedence is implied.



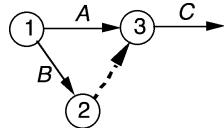
**Figure 12.4.4** Activity  $A$  is not listed as the predecessor of  $C$ .

**Remark:** Notice that if there were a directed cycle in an AOA network, then none of the activities corresponding to the arcs on that cycle could ever begin. Thus, if an AOA network represents a feasible project, it must be an *acyclic* digraph.

Given a list of activities and predecessors, an AOA network representation of a project can be constructed according to the following rules:

1. Vertex 1 represents the event marking the start of the project, and vertex  $n$  marks the end of the project.
2. Each activity that has no predecessors is represented by an arc incident from the start vertex.
3. The vertices in the network are numbered so that the vertex representing the completion of an activity (i.e., the *head* of that arc) has a larger number than the vertex that represents the beginning of that activity (the *tail* of that arc).
4. Each activity corresponds to exactly one arc in the network.
5. Two vertices can be joined by at most one arc (i.e., no multi-arcs).

Adhering to these rules sometimes requires using an artificial arc representing a *dummy activity* with zero duration time. For instance, if activities  $A$  and  $B$  can be performed concurrently, but they are both predecessors of activity  $C$ , then their relationship can be represented by introducing a dummy activity as shown in the figure. Notice that according to the figure, activity  $C$  cannot begin until both  $A$  and  $B$  have been completed.

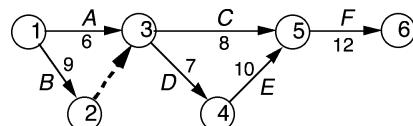


**Figure 12.4.5** Using a dummy arc so that  $A$  and  $B$  precede  $C$ .

**Application 12.4.1 Scheduling the Construction of a House:** Suppose that the tasks (activities) remaining to complete the construction of a house are as shown in the table.

Activity	Predecessors	Duration (days)
$A = \text{Wiring}$	—	6
$B = \text{Plumbing}$	—	9
$C = \text{Walls \& Ceilings}$	A, B	8
$D = \text{Floors}$	A, B	7
$E = \text{Carpeting}$	D	10
$F = \text{Interior decorating}$	C, E	12

In the AOA network shown in Figure 12.4.6, the activity and its duration are listed alongside each arc.



**Figure 12.4.6** AOA network for completing the house construction.

The critical-path method requires the calculation of the earliest and latest times at which each event can occur.

NOTATION: Let  $wt(i, j)$  denote the weight of arc  $(i, j)$ , that is, the duration of the corresponding activity.

### Calculating the Earliest Event Times

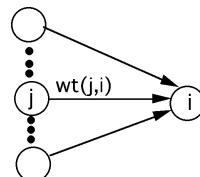
NOTATION: Let  $ET(i)$  denote the earliest time at which the event corresponding to vertex  $i$  can occur.

TERMINOLOGY: A vertex  $j$  is an **immediate predecessor** of a vertex  $i$  if there is an arc from  $j$  to  $i$ .

NOTATION: Let  $\text{pred}(i)$  be the set of immediate predecessors of vertex  $i$ .

The calculation of the earliest times for the vertices begins by setting event time  $ET(1) = 0$ . The value of  $ET(i)$  for each of the remaining vertices depends on  $ET(j)$  for all immediate predecessors  $j$  of vertex  $i$ .

In particular, suppose that vertex  $j$  is any immediate predecessor of vertex  $i$ . Then the event corresponding to vertex  $i$  cannot occur before  $ET(j) + wt(j, i)$ . Since this is true for every immediate predecessor of vertex  $i$ ,  $ET(i)$  is the maximum of these sums taken over all the immediate predecessors of  $i$ , as illustrated in Figure 12.4.7.



**Figure 12.4.7**  $ET(i) = \max_{j \in \text{pred}(i)} \{ET(j) + wt(j, i)\}$ .

**Example 12.4.2:** In the house-construction example,

$$\begin{aligned} ET(2) &= ET(1) + 9 = 9 \\ ET(3) &= \max\{ET(1) + 6 = 6, ET(2) + 0 = 9\} = 9 \\ ET(4) &= ET(3) + 7 = 16 \\ ET(5) &= \max\{ET(3) + 8 = 17, ET(4) + 10 = 26\} = 26 \\ ET(6) &= ET(5) + 12 = 38 \end{aligned}$$

Thus, the earliest completion time for the construction is 38 days.

**Proposition 12.4.1.** Let  $D$  be an AOA network with vertices  $1, 2, \dots, n$ . Then the earliest time  $ET(i)$  for the event  $i$  to occur is the length of the longest directed path in the network from vertex 1 to vertex  $i$ .

**Proof:** Since the digraph is acyclic, the longest path from vertex 1 to itself is 0, which establishes the base of an induction proof on  $i$ . The induction step follows from the way  $ET(i)$  is computed from the  $ET$  values of its immediate predecessors. The details are left as an exercise (see Exercises).  $\diamond$

**Proposition 12.4.2.** Let  $D$  be an acyclic digraph. Then there is a vertex whose indegree is 0, and there is a vertex whose outdegree is 0.  $\diamond$  (Exercises)

The following algorithm, which computes the  $ET$  values, is based on the iterative application of Proposition 12.4.2. In each iteration, a vertex  $v$  with indegree 0 is selected, and the  $ET$  values of its successor vertices are updated accordingly. Vertex  $v$  is then deleted from the digraph, and the next iteration begins. An induction argument can be used to show that the algorithm does indeed compute the longest paths from the start vertex to each other vertex (see Exercises).

**Algorithm 12.4.1: Computation of Earliest Event Time**

*Input:* an  $n$ -vertex AOA network  $N$  with vertices  $1, 2, \dots, n$ .  
*Output:* the earliest event times  $ET(1), ET(2), \dots, ET(n)$ .

```
[Initialization]
Initialize a working copy  $D := N$ .
For  $i = 1$  to  $n$ 
     $ET(i) := 0$ 
[Begin computation]
For  $j = 1$  to  $n$ 
    Let  $v$  be a vertex with indegree 0.
    For each arc  $(v, w)$ 
        Set  $ET(w) := \max\{ET(w), ET(v) + wt(v, w)\}$ .
     $D := D - \{v\}$ 
Return  $ET(1), ET(2), \dots, ET(n)$ .
```

### Calculating the Latest Event Times

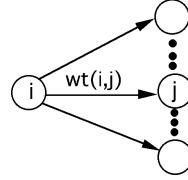
NOTATION: Let  $LT(i)$  denote the latest time at which the event corresponding to vertex  $i$  can occur without delaying the completion of the project.

**TERMINOLOGY:** A vertex  $j$  is an **immediate successor** of a vertex  $i$  if there is an arc from  $i$  to  $j$ .

**NOTATION:** Let  $\text{succ}(i)$  be the set of immediate successors of vertex  $i$ .

The calculation of the latest time  $LT(i)$  is similar to the  $ET$  calculation but works backward from the end-of-project vertex  $n$ . It begins by setting  $LT(n) = ET(n)$ .

Next, suppose that vertex  $j$  is an immediate successor of vertex  $i$ . If the event corresponding to vertex  $i$  occurs after  $LT(j) - wt(i, j)$ , then event  $j$  will occur after  $LT(j)$ , thereby delaying the completion of the project. Since this is true for any immediate successor of  $i$ ,  $LT(i)$  is the minimum of these differences taken over all immediate successors of  $i$ , as illustrated in Figure 12.4.8.



**Figure 12.4.8**  $LT(i) = \min_{j \in \text{succ}(i)} \{LT(j) - wt(i, j)\}$ .

**Example 12.4.3:** Continuing with the house-construction example,

$$LT(6) = 38$$

$$LT(5) = LT(6) - 12 = 26$$

$$LT(4) = LT(5) - 10 = 16$$

$$LT(3) = \min\{LT(4) - 7 = 9, LT(5) - 8 = 18\} = 9$$

$$LT(2) = LT(3) - 0 = 9$$

$$LT(1) = \min\{LT(2) - 9 = 0, LT(3) - 6 = 3\} = 0$$

Algorithm 12.4.2, which computes the  $LT$  values, is analogous to Algorithm 12.4.1.

**Algorithm 12.4.2: Computation of Latest Event Time**

*Input:* an  $n$ -vertex AOA network  $N$  with vertices  $1, 2, \dots, n$  and with earliest project completion time  $K$ .

*Output:* the latest event times  $LT(1), LT(2), \dots, LT(n)$ .

[Initialization]

Initialize a working copy  $D := N$ .

For  $i = 1$  to  $n$

$LT(i) := K$

[Begin computation]

For  $j = 1$  to  $n$

Let  $v$  be a vertex with outdegree 0.

For each arc  $(w, v)$

Set  $LT(w) := \min\{LT(w), LT(w) - wt(w, v)\}$ .

$D := D - \{v\}$

Return  $LT(1), LT(2), \dots, LT(n)$ .

### Determining the Critical Activities

**DEFINITION:** The **total float** of an activity corresponding to arc  $(i, j)$ , denoted  $TF(i, j)$ , is the amount by which the duration  $wt(i, j)$  of activity  $(i, j)$  can be increased without delaying the completion of the project.

Equivalently, the total float is the amount by which the starting time of activity  $(i, j)$  can be delayed without delaying the completion of the overall project.

The total float is easily calculated once the earliest and latest event times have been determined.

**Proposition 12.4.3.** *The total float for activity  $(i, j)$  is given by  $TF(i, j) = LT(j) - ET(i) - wt(i, j)$ .*

**Proof:** Suppose that the occurrence of event  $i$  or the duration of activity  $(i, j)$  is delayed by  $k$  time units. Then activity  $(i, j)$  will be completed at time  $ET(i) + k + wt(i, j)$ . Thus, the largest value for  $k$  without incurring a delay in the project is the value that satisfies  $ET(i) + k + wt(i, j) = LT(j)$ .  $\diamond$

**Example 12.4.4:** The earliest and latest event times for the house-construction example are summarized in the table shown.

Vertex	$ET(i)$	$LT(i)$
1	0	0
2	9	9
3	9	9
4	16	16
5	26	26
6	38	38

By Proposition 12.4.3, the total float for each activity is calculated as follows.

$$TF(A) = TF(1, 3) = LT(3) - ET(1) - 6 = 3$$

$$TF(B) = TF(1, 2) = LT(2) - ET(1) - 9 = 0$$

$$TF(C) = TF(3, 5) = LT(5) - ET(3) - 8 = 9$$

$$TF(D) = TF(3, 4) = LT(4) - ET(3) - 7 = 0$$

$$TF(E) = TF(4, 5) = LT(5) - ET(4) - 10 = 0$$

$$TF(F) = TF(5, 6) = LT(6) - ET(5) - 12 = 0$$

Notice that there is some leeway for activities  $A$  and  $C$  but none for the other activities. Identifying the activities that have this kind of leeway would enable a contractor to shift workers from such an activity to one that is threatening the delay of the overall project completion.

**DEFINITION:** A **critical activity** is an activity whose total float equals zero.

**DEFINITION:** A **critical path** is a directed path from the start vertex to the finish vertex such that each arc on the path corresponds to a critical activity.

**Example 12.4.5:** For the house-construction project, the path

$1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6$  is a critical path. This is the only critical path in this example, but in general, the critical path is *not unique*.

**Proposition 12.4.4.** Let  $D$  be an AOA network. A directed path from the start vertex to the finish vertex is a critical path if and only if it is a longest path from start to finish.  
 $\diamond$  (Exercises)

### EXERCISES for Section 12.4

In Exercises 12.4.1 through 12.4.3, a table listing the tasks of a project, along with their duration times and immediate predecessors is given. Apply CPM to determine the earliest starting time of each task, the earliest possible completion time of the entire project, and the critical tasks.

12.4.1<sup>s</sup>

Task	a	b	c	d	e	f	g
Duration	10	7	5	3	2	1	14
Predecessors	—	—	a	c	d	b,e	e,f

12.4.2

Task	a	b	c	d	e	f	g	h
Duration	10	5	3	4	5	6	5	5
Predecessors	—	—	b	a,c	a,c	d	e	f,g

12.4.3

Task	a	b	c	d	e	f	g	h	i	j
Duration	7	3	2	8	4	6	1	10	5	9
Predecessors	—	a	a	c	b,d	e	d	g	f,g	h,i

12.4.4 A student must complete 10 courses before he or she can graduate in applied mathematics. The courses and their prerequisites are listed in the following table. Apply CPM to determine the minimum number of semesters needed to graduate.

Course	Prerequisites
C1 = Calculus 1	None
C2 = Calculus 2	C1
DM = Discrete Math	C1
C3 = Calculus 3	C2
A = Algorithms	C1,DM
GT = Graph Theory	DM
DE = Differential Equations	C2
S = Statistics	C1
P = Probability	C2,DM
LA = Linear Algebra	DM,C3

12.4.5 Complete the proof of Proposition 12.4.1.

12.4.6<sup>s</sup> Prove Proposition 12.4.2.

12.4.7 Prove that the Earliest-Time Algorithm 12.4.1 is correct.

12.4.8 Prove that the Latest-Time Algorithm 12.4.2 is correct.

12.4.9<sup>s</sup> Prove Proposition 12.4.4.

12.4.10 [Computer Project] Write a computer program that takes as input an  $n$ -vertex AOA network and that produces as output the earliest and latest event times and the total float of each activity.

---

## 12.5 FINDING THE STRONG COMPONENTS OF A DIGRAPH

The backbone of the algorithm for finding the strong components of a digraph is *depth-first search*, just as it was for the block-finding algorithm of §5.4. The strategy for the one-pass algorithm given here can also be used to convert the block-finding algorithm into a one-pass algorithm (see Exercises).

### Depth-First Search Revisited

**TERMINOLOGY:** An output tree produced by executing a depth-first search on a digraph is sometimes referred to as a *dfs tree*.

**TERMINOLOGY:** A **non-tree arc** of a dfs tree  $T$  is an arc that is not part of tree  $T$  but whose endpoints are.

**Remark:** Although the depth-first search algorithm in §4.2 works equally well for digraphs, the version given below is better suited for finding strong components. The vertex stack used in Algorithm 12.5.1 makes it easier to examine and process the non-tree arcs, which is essential for Algorithm 12.5.2.

**TERMINOLOGY:** A tree vertex  $v$  is said to be **completely processed** if all arcs incident from  $v$  and all arcs incident from descendants of  $v$  have been examined.

**TERMINOLOGY:** If an arc incident from a vertex  $v$  is currently being examined, then the vertex  $v$  is said to be **active**.

**NOTATION:** Recall from §4.2 that the integer-label assigned to a vertex  $w$  during depth-first search is denoted  $dfnumber(w)$ .

During the execution of Algorithm 12.5.1, there are two possibilities that can occur when an arc is examined. If the arc is a frontier arc (i.e., its head is not yet in the dfs tree), then (i) its head vertex is assigned the next  $dfnumber$ , (ii) the arc and vertex are added to the growing dfs tree, and (iii) the vertex becomes the active vertex. The second possibility is that the head vertex of the arc is already in the tree, in which case the arc does not get added to the tree (i.e., it is a non-tree arc), and the current vertex stays active.

To distinguish tree vertices from non-tree vertices, Algorithm 12.5.1 initializes  $dfnumber(w)$  at  $-1$ . The function *examined* keeps track of which edges have been examined by initializing  $examined(e)$  at  $F(\text{false})$ , for each edge  $e$ , and setting it to  $T(\text{true})$  when  $e$  is examined.

### Classifying the Non-Tree Arcs of a DFS Tree

**REVIEW FROM §4.1:** For tree-growing in digraphs, the non-tree arcs fall into three categories according to how their endpoints are related in the dfs tree.

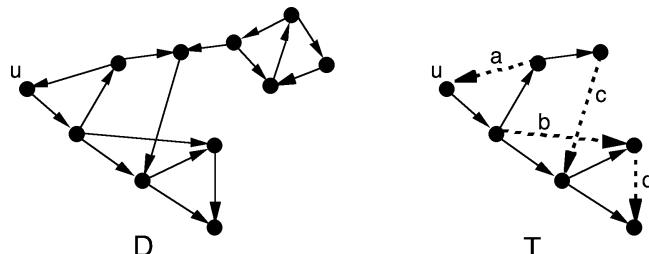
- A **back-arc** is directed from a vertex to one of its ancestors.
- A **forward-arc** is directed from a vertex to one of its descendants.
- A **cross-arc** is directed from a vertex to another vertex that is neither an ancestor nor a descendant.

**Algorithm 12.5.1: Depth-First Search of a Digraph**

*Input:* a digraph  $D$  and a starting vertex  $v$ .  
*Output:* a rooted tree  $T$  with root  $v$  and a  $dfnumber$  for each  $v \in v_T$ .

```
[Initialization]
For each vertex  $w \in V_D$ 
     $dfnumber(w) := -1$ 
For each arc  $e \in E_D$ 
     $examined(e) := F$ 
Initialize  $vertexstack$  to contain vertex  $v$ 
Initialize tree  $T$  as vertex  $v$ .
 $dfnumber(v) := 0$ 
Initialize  $dfnumber$  counter  $dfnum := 1$ 
[Begin processing]
While  $vertexstack$  is not empty
    Let vertex  $t$  be  $top(vertexstack)$ .
    While there are unexamined arcs incident from vertex  $t$ 
        Let  $e$  be an unexamined arc incident from vertex  $t$ .
         $examined(e) := T$ 
        Let vertex  $w$  be  $head(e)$ .
        If  $dfnumber(w) = -1$  [vertex w is not yet in tree T]
             $dfnumber(w) := dfnum$ 
             $dfnum := dfnum + 1$ 
            Add edge  $e$  (and vertex  $w$ ) to tree  $T$ .
            Push vertex  $w$  onto  $vertexstack$ .
            [begin processing of vertex  $w$ ]
             $t := w$ 
        [processing of vertex  $t$  is complete; remove it from  $vertexstack$ ]
    Return tree  $T$  with its  $dfnumbers$ .
```

**Example 12.5.1:** Figure 12.5.1 shows one possible result of executing Algorithm 12.5.1 on the digraph  $D$  shown on the left, starting at vertex  $u$ . The non-tree arcs of the dfs tree  $T$  on the right are drawn as dashed lines. Arc  $a$  is a back-arc,  $b$  is a forward-arc, and  $c$  and  $d$  are both cross-arcs.



**Figure 12.5.1** A digraph  $D$  and the non-tree arcs of an output tree  $T$ .

Each type of non-tree arc of a dfs tree can be described in terms of the  $dfnumbers$  of the vertices. It follows directly from the definitions that a forward-arc is directed from a smaller  $dfnumber$  to a larger one, and a back-arc is directed from a larger  $dfnumber$  to a smaller one. The following proposition shows that a cross-arc is always directed from a larger  $dfnumber$  to a smaller one.

**Proposition 12.5.1.** Let  $e$  be a cross-arc of a depth-first search tree from vertex  $x$  to vertex  $y$ . Then  $dfnumber(x) > dfnumber(y)$ .

**Proof:** By way of contradiction, suppose that  $dfnumber(x) \leq dfnumber(y)$ . This means that vertex  $x$  was added to the search tree before vertex  $y$ . Immediately after  $x$  was added to the tree, arc  $e$  became a frontier arc. Since  $e$  is a non-tree arc, vertex  $y$  must have been labeled via some other frontier arc. But in each iteration after vertex  $x$  was labeled, until and including the iteration in which vertex  $y$  was labeled, the chosen frontier arc was directed either from vertex  $x$  or from a descendant of  $x$ . In either case, vertex  $y$  is a descendant of vertex  $x$ , which contradicts the definition of cross-arc.  $\diamond$

### Additional Preliminaries for the Strong-Component-Finding Algorithm

The strategy for finding mutually reachable vertices is to keep track of how the non-tree arcs connect the tree vertices. In §4.4 and in §5.4, a function  $low$  was used in determining the cut-vertices and the blocks, respectively, of an undirected graph, and it plays a similar role for Algorithm 12.5.2. The presentation given here is similar to those given in [ThSw92] and [Ba83], and additional details, including proofs of correctness, may be found in either reference. A somewhat different approach may be found in [St93].

**TERMINOLOGY:** Let  $T$  be a dfs tree for a digraph, and let  $v \in V_T$ . Then  $low(v)$  is the smallest  $dfnumber$  among all vertices in  $V_T$  that are known to be in the same strong component as vertex  $v$ .

The examination of an arc directed from the active vertex  $t$  results either in assigning a  $dfnumber$  to a new vertex  $w$  and making  $w$  active or recognizing that the arc is a non-tree arc and updating  $low(t)$  appropriately.

### Additional Variables and Initialization for Algorithm 12.5.2

$placed(w)$ : is T if vertex  $w$  has been placed in a strong component, and is F otherwise.

$parent(v)$ : the parent of vertex  $v$  in the dfs tree.

$holdstack$ : a stack that holds the vertices that have been processed but not yet placed in a strong component.

#### Initialization for Algorithm 12.5.2

```

For each vertex  $w \in V_D$ 
   $dfnumber(w) := -1$ 
   $low(w) := -1$ 
   $placed(w) := F$ 
For each arc  $e \in E_D$ 
   $examined(e) := F$ 
Initialize  $vertexstack$  to contain vertex  $v$ 
Initialize tree  $T$  as vertex  $v$ .
 $dfnumber(v) := 0$ 
Initialize  $dfnumber$  counter  $dnum := 1$ 
Initialize strong component counter  $k := 0$ 
Initialize  $holdstack$  as empty.

```

### Computation of $low$

When the algorithm assigns a  $dfnumber$  to a vertex  $w$ , it also initializes  $low(w)$  to be that  $dfnumber$ . The  $low$  values change as the algorithm proceeds. After vertex  $v$  is completely processed,  $low(v) = dfnumber(v)$  if and only if vertex  $v$  is the root of a subtree whose vertices form a strong component. A description of how the value of  $low$  is updated appears after the algorithm.

#### Algorithm 12.5.2: Strong-Component Finding

*Input:* a digraph  $D$  and a starting vertex  $v$ .

*Output:* a dfs tree  $T$  with root  $v$ ;

vertex-sets  $S_1, S_2, \dots, S_k$  of the strong components of tree  $T$

Initialize variables as prescribed above.

[*Begin processing*]

While  $vertexstack$  is not empty

    Let vertex  $t$  be  $top(vertexstack)$ .

    While there are unexamined arcs incident from vertex  $t$

        Let  $e$  be an unexamined arc incident from vertex  $t$ .

*examined*( $e$ ) := T

        Let vertex  $w$  be  $head(e)$ .

        If  $dfnumber(w) = -1$  [vertex  $w$  is not yet in tree  $T$ ]

$dfnumber(w) := dfnum$

$dfnum := dfnum + 1$

$low(w) := dfnumber(w)$

            Push vertex  $w$  onto  $vertexstack$ .

$t := w$

        Else If  $placed(w) = F$

            [vertex  $w$  has not yet been placed in a strong component]

$low(t) := \min\{low(t), dfnumber(w)\}$

            [processing of vertex  $t$  has been completed]

        If  $low(t) = dfnumber(t)$

            [vertex  $t$  is the root of a subtree of  $T$  whose vertices

                form a strong component]

$k := k + 1$

$placed(t) := T$

            Initialize set  $S_k := \{t\}$ . [place vertex  $t$  in strong component  $S_k$ ]

            While  $holdstack \neq \emptyset$  AND  $low(top(holdstack)) \geq dfnumber(t)$

$z := top(holdstack)$

$placed(z) := T$  [ $z$  and  $t$  are mutually reachable]

$S_k := S_k \cup \{z\}$

                Pop  $holdstack$ .

            Pop vertex  $t$  from  $vertexstack$ .

        Else

            Push vertex  $t$  onto  $holdstack$

            Pop  $vertexstack$ .

            If  $parent(t)$  exists (i.e.,  $t$  is not the root of  $T$ )

$low(parent(t)) := \min\{low(parent(t)), low(t)\}$

                [depth-first search backs up to  $parent(t)$ ]

Return tree  $T$  vertex-sets  $S_1, S_2, \dots, S_k$ .

### Updating the Value of $low$

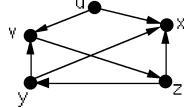
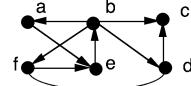
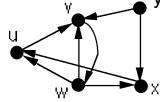
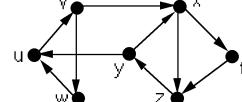
The value of  $low(v)$  is updated according to the following rules. Their justification follows from the properties of the non-tree arcs, including the one established in Proposition 12.5.1.

1. If the processing of vertex  $v$  is complete and if  $low(v) < dfnumber(v)$ , then the value of  $low(v)$  is carried back up to the parent of  $v$ . That is,  
 $low(parent(v)) := \min\{low(parent(v)), low(v)\}$ .
2. If vertex  $v$  is active and the arc being examined is a back-arc from  $v$  to  $w$ , then  
 $low(v) := \min\{low(v), dfnumber(w)\}$ .
3. If vertex  $v$  is active and the arc being examined is a cross-arc from  $v$  to  $w$ , then set  
 $low(v) := \min\{low(v), dfnumber(w)\}$   
if and only if  $w$  has not already been assigned to a strong component.

**Remark:** For a digraph  $D$ , the vertex-set of a dfs tree  $T$  is not necessarily the vertex-set of one of the strong components of  $D$ , but it is always the union of the vertex-sets of one or more of the strong components of  $D$  (see Exercises). Algorithm 12.5.2 above produces each of these vertex-sets. If the dfs tree does not include all the vertices of digraph  $D$ , then Algorithm 12.5.2 will have found the vertex-sets of only some of the strong components. However, the algorithm can be restarted with the deletion subdigraph  $D' = D - V_T$  to find additional strong components of  $D$ . The process can continue until all the components of digraph  $D$  are found.

### EXERCISES for Section 12.5

For Exercises 12.5.1 through 12.5.4, draw the dfs trees that result when Depth-First Search Algorithm 12.5.1 is applied to the given digraph using each of the specified vertices as the starting vertex. Whenever there is more than one frontier arc from which to choose, choose the one that is directed to the lexicographically smallest vertex. Also, classify all the non-tree arcs for each dfs tree.

12.5.1  $u, v, x$ 12.5.2<sup>s</sup>  $a, b, d$ 12.5.3  $w, x, y$ 12.5.4  $v, x, y$ 

For Exercises 12.5.5 through 12.5.8, verify the assertion of Proposition 12.5.1 for each of the cross-arcs in each of the dfs trees from the specified exercise.

12.5.5 The graph of Exercise 12.5.1.

12.5.6<sup>s</sup> The graph of Exercise 12.5.2.

12.5.7 The graph of Exercise 12.5.3.

12.5.8<sup>s</sup> The graph of Exercise 12.5.4.

For Exercises 12.5.9 through 12.5.12, apply Strong-Component-Finding Algorithm 12.5.2 to the specified digraph. If there remain unlabeled vertices, reapply the algorithm on the subdigraph induced on these unlabeled vertices. Continue until all strong components have been found.

12.5.9 The graph of Exercise 12.5.1.

12.5.10<sup>s</sup> The graph of Exercise 12.5.2.

12.5.11 The graph of Exercise 12.5.3.

12.5.12 The graph of Exercise 12.5.4.

12.5.13 Apply Strong-Component-Finding Algorithm 12.5.2 to the digraph in Figure 12.1.0. If there remain unlabeled vertices, reapply the algorithm on the subdigraph induced on these unlabeled vertices. Continue until all strong components have been found.

12.5.14<sup>s</sup> Characterize those digraphs for which one application of Strong-Component-Finding Algorithm 12.5.2 is guaranteed to find *all* of the strong components, regardless of where the algorithm starts.

12.5.15 Prove that the vertex-set of a dfs tree for a digraph  $D$  is partitioned by the vertex-sets of one or more of the strong components of  $D$ .

12.5.16 Adapt the strategy in Strong-Component-Finding Algorithm 12.5.2 to convert the block-finding algorithm of §5.4 to a one-pass algorithm.

12.5.17 [Computer Project] Implement a modified version of Strong-Component-Finding Algorithm 12.5.2 that finds *all* of the strong components. Test the program on each of the digraphs in Exercises 12.5.1 through 12.5.4.

## 12.6 SUPPLEMENTARY EXERCISES

12.6.1 Draw all the isomorphism types of simple digraphs with 4 vertices and 3 arcs.

12.6.2 Draw all the isomorphism types of acyclic digraphs with 4 vertices and 4 arcs.

12.6.3 Draw all the isomorphism types of acyclic digraphs with 5 vertices and 3 arcs.

12.6.4 Draw all the isomorphism types of eulerian digraphs with 4 vertices.

DEF: A **self-converse digraph** is a digraph  $D$  such that the result of reversing every arc direction is isomorphic to  $D$ .

12.6.5 Draw all the isomorphism types of self-converse simple digraphs on 3 vertices.

12.6.6 Draw all the isomorphism types of self-converse simple digraphs on 4 vertices and 4 arcs.

DEF: The **arc-complement of a digraph**  $D$  is the digraph  $\overline{D}$  with the same vertex set  $V_D$ , such that  $uv \in E_{\overline{D}}$  if and only if  $uv \notin E_D$ .

DEF: A **self-complementary digraph** is a digraph  $D$  such that  $\overline{D} \cong D$ .

12.6.7 Draw all the isomorphism types of self-complementary simple digraphs on 3 vertices.

**12.6.8** Draw all the isomorphism types of self-complementary simple digraphs on 4 vertices.

**12.6.9** Let  $G$  be an undirected graph  $G$  without self-loops. Prove that there is an orientation of  $G$  that is acyclic.

**12.6.10** Prove that a digraph is acyclic if and only if it is isomorphic to its condensation.

**12.6.11** Prove that the condensation of any digraph has at least one vertex with indegree 0 and at least one with outdegree 0.

**12.6.12** Draw a Hasse diagram for a 5-element poset that has exactly four linear extensions.

**12.6.13** Draw all the isomorphism types of tournaments with 4 vertices.

**12.6.14** Draw all the isomorphism types of tournaments with 5 vertices.

**12.6.15** Do there exist  $n$ -vertex tournaments, for  $n \geq 4$ , that have a constant score sequence? If they do not exist, then explain why not, and if they do exist, then say as much as you can about such a tournament.

**12.6.16** Draw a 5-vertex tournament so that every vertex is a king.

**12.6.17** Prove or disprove: There exists an  $n$ -vertex tournament with  $\text{indegree}(v) = \text{outdegree}(v)$  at each vertex  $v$  if and only if  $n$  is odd.

**12.6.18** Let  $T$  be an  $n$ -vertex tournament. Prove that the following statements are equivalent.

- (1)  $T$  is transitive.
  - (2)  $T$  is acyclic.
  - (3)  $T$  contains a unique hamiltonian path.
- 

## GLOSSARY

**absorbing state** in a Markov chain: a state  $s_i$  with  $p_{ii} = 1$ .

**active vertex** during a depth-first search: the tail of the arc that is currently being examined.

**AOA (activity-on-arc) network:** a digraph model of a project such that each arc in the digraph represents an activity (or task). The weight of the arc is the duration time of that activity. Each vertex represents an *event* that signifies the completion of one or more activities and the beginning of new ones.

**arc-complement of a digraph  $D$ :** the digraph  $\overline{D}$  with the same vertex set  $V_D$ , such that  $uv \in E_{\overline{D}}$  if and only if  $uv \notin E_D$ .

**back-arc** in a dfs tree  $T$ : an arc whose tail is a descendant of its head.

**communicating states** in Markov chain: two states that are mutually reachable in the corresponding Markov diagram.

**communication class** in a Markov chain: a maximal subset of states that communicate with one another.

**comparable elements** in a poset  $(S, \prec)$ : two elements  $x$  and  $y$  such that either  $x \prec y$  or  $y \prec x$ .

**compatible** total order with respect to a given partial order  $\prec$  on  $S$ : a total order  $\prec^t$  on set  $S$  such that  $x \prec y \Rightarrow x \prec^t y$ .

**completely processed** tree vertex in a depth-first search: a vertex  $v$  such that all arcs incident from  $v$  and all arcs incident from descendants of  $v$  have been examined.

**condensation** of a digraph  $D$ : the simple digraph  $D'$  with vertex-set

$V_{D'} = \{s_1, s_2, \dots, s_r\}$ , such that there is an arc in digraph  $D'$  from vertex  $s_i$  to vertex  $s_j$  if and only if there is an arc in digraph  $D$  from a vertex in strong component  $S_i$  to a vertex in strong component  $S_j$ .

**Condorcet winner**: a candidate  $x$  such that for every other candidate  $y$ ,  $x$  is preferred over  $y$  by a majority of the voters. Thus, a Condorcet winner corresponds to a vertex  $x$  in the majority digraph that has an arc directed to every other vertex.

**Condorcet paradox**: an occurrence of a directed cycle in the majority digraph of a set of tournaments (see §12.3).

**connected digraph**: a digraph whose underlying graph is connected. Some authors use the term *weakly connected* to describe such digraphs.

**critical activity** in an AOA network: an activity whose total float equals zero; any delay will delay the overall project completion.

**critical path** in an AOA network: a directed path from the start vertex to the finish vertex such that each arc on the path corresponds to a critical activity.

**cross-arc** in a dfs tree  $T$ : an arc whose tail is neither an ancestor nor a descendant of its head.

**cycle addition to a graph  $G$** : the addition of a cycle that has exactly one vertex in common with  $G$ .

**dag**: a mnemonic for *directed acyclic digraph*.

**dfs tree**: an output tree resulting from a depth-first search.

**digraph representation of a relation  $R$**  on a finite set  $S$ : the digraph whose vertices correspond to the elements of  $S$  and whose arcs correspond to the ordered pairs in the relation. That is, an arc is drawn from vertex  $x$  to vertex  $y$  if  $(x, y) \in R$ .

**directed tree**: a digraph whose underlying graph is a tree.

**dominates** (or **beats**) a vertex  $y$  in a tournament: said of any vertex  $x$  that has an outgoing arc directed to  $y$ .

**2-edge-connected graph**: a connected graph that has no cut-edges.

**forward-arc** in a dfs tree  $T$ : an arc whose tail is an ancestor of its head.

**frontier arc** for a rooted tree  $T$ : an arc whose tail is in  $T$  and whose head is not in  $T$ .

**Hasse diagram** representing the poset  $(S, \prec)$ : a graph drawing defined as follows:

- (1) Each element of set  $S$  is represented by a vertex.
- (2) If  $x \prec y$ , then the vertex for  $y$  is positioned higher than the vertex for  $x$ ; and if there is no  $w$  such that  $x \prec w$  and  $w \prec y$ , then an edge is drawn from vertex  $x$  upward to vertex  $y$ .

**immediate predecessor** of a vertex  $i$  in an AOA network: a vertex  $j$  such that there is an arc from  $j$  to  $i$ .

**immediate successor** of a vertex  $i$  in an AOA network: a vertex  $j$  such that there is an arc from  $i$  to  $j$ .

**king in a tournament  $T$ :** a vertex  $x$  such that every vertex in  $T$  is reachable from  $x$  by a directed path of length 2 or less.

**linear extension of a poset  $(S, \prec)$ :** a total order  $\prec^t$  on  $S$  that is compatible with  $\prec$ .

**majority digraph** of a set  $\mathcal{T}$  of tournaments, each having the same vertex-set  $V$ : a digraph  $D$  with vertex-set  $V$ , and such that there is an arc directed from vertex  $x$  to vertex  $y$  in  $D$  if and only if  $x$  dominates  $y$  in a majority of the tournaments.

**Markov chain, discrete-time:** a phenomenon whose behavior can be modeled by a sequence  $\{X_t\}$ ,  $t = 0, 1, 2, \dots$ , such that the (**one-step**) **transition probability**  $p_{ij}$  is the *conditional probability* given by

$$p_{ij} = \text{prob}(X_{t+1} = s_j | X_t = s_i), \quad \text{for } t = 1, 2, \dots$$

—, **irreducible**: a Markov chain whose Markov diagram is strongly connected.

—, **stationary**: a Markov chain in which the transition probabilities are independent of time  $t$ .

**Markov diagram** of a given Markov chain: the digraph whose vertices represent the states and whose arcs represent the transitions from one state to the next. The arc from vertex  $i$  to vertex  $j$  is assigned the weight  $p_{ij}$ .

**minimal element** of a poset  $(S, \prec)$ : an element  $m \in S$  such that for all other  $x \in S$ ,  $x \not\prec m$ .

**mutually reachable** vertices  $u$  and  $v$  in a digraph  $D$ : there are both a directed  $u$ - $v$  path and a directed  $v$ - $u$  path in digraph  $D$ . A single vertex is regarded as reachable from itself.

**non-tree arc** of a dfs tree  $T$ : an arc that is not part of tree  $T$  but whose endpoints are.

**partial order**: a relation that is reflexive, antisymmetric, and transitive.

**partially ordered set**: a pair  $(S, R)$ , where  $R$  is a partial order on set  $S$ .

**path addition to a graph  $G$** : the addition to  $G$  of a path between two existing vertices of  $G$ , such that the edges and internal vertices of the path are new.

**poset**: synonym for partially ordered set.

**predecessor** of an activity  $B$  in a project: an activity  $A$  that must be completed before  $B$  can begin.

—, **immediate** of a vertex  $i$  in an AOA network: a vertex  $j$  such that there is an arc from  $j$  to  $i$ .

**reachability among states** in a Markov chain: coincides with the digraph term applied to the corresponding Markov diagram (digraph).

**recurrent state** in a Markov chain: a state that is not a transient state.

**root** of a rooted tree: see *rooted tree*.

**rooted tree**: a directed tree having a distinguished vertex  $r$ , called the **root**, such that for every other vertex  $v$ , there is a (unique) directed  $r$ - $v$  path.

**score sequence** of a tournament: the outdegree sequence of the tournament.

**self-complementary digraph**: a digraph  $D$  such that  $\overline{D} \cong D$ .

**self-converse digraph** a digraph  $D$  such that the result of reversing every arc direction is isomorphic to  $D$ .

**strong component** of a digraph  $D$ : a maximal strongly connected subdigraph of  $D$ ; a subdigraph induced on a maximal set of mutually reachable vertices.

**strongly connected** digraph  $D$ : every two vertices are mutually reachable in  $D$ .

**strongly orientable** graph: a graph for whose edge-set there exists an assignment of directions such that the resulting digraph is strongly connected.

**topological sort**: a process that produces a linear extension of a poset, or (informally) the resulting linear extension.

**total float**  $TF(i, j)$  of an activity corresponding to arc  $(i, j)$  in an AOA network: the amount by which the duration  $wt(i, j)$  of activity  $(i, j)$  can be increased without delaying the completion of the project.

**total order** on a set  $S$ : a partial order on  $S$  such that every two elements are comparable.

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

—, **regular**: a tournament  $T$  in which there is an integer  $s$  such that  $\text{outdegree}(v) = s$  for all vertices  $v \in V_T$ .

—, **transitive**: a tournament that is transitive as a digraph.

**transient state** in a Markov chain: a state  $s_i$  such that there exists a state  $s_j$  that is reachable from state  $s_i$ , but  $s_i$  is not reachable from  $s_j$ .

**transition matrix** for a Markov chain: a matrix whose  $ij$ th entry is the transition probability  $p_{ij}$ .

—,  **$k$ -step**  $P^{[k]}$ : the matrix whose  $ij$ th entry is the  $k$ -step transition probability  $p_{ij}^{[k]}$ .

**transition probability** in a Markov chain: the conditional probability  $p_{ij}$  that  $X_{t+1} = s_j$ , given that  $X_t = s_i$ .

—,  **$k$ -step**  $p_{ij}^{[k]}$ : the conditional probability that  $X_{t+k} = s_j$ , given that  $X_t = s_i$ .

**transitive closure of a digraph**  $D$ : the smallest transitive superdigraph of  $D$ .

**transitive digraph**: a digraph whose corresponding relation is transitive. That is, if there is an arc from vertex  $x$  to vertex  $y$  and an arc from  $y$  to  $z$ , then there is an arc from  $x$  to  $z$ .

**Whitney-Robbins synthesis** of a graph  $G$  from a graph  $H$ : a sequence of graphs,  $G_0, G_1, \dots, G_l$ , where  $G_0 = H$ ,  $G_l = G$ , and  $G_i$  is the result of a path or cycle addition to  $G_{i-1}$ , for  $i = 1, \dots, l$ .

# Chapter 13

---

## NETWORK FLOWS AND APPLICATIONS

**13.1 Flows and Cuts in Networks**

**13.2 Solving the Maximum-Flow Problem**

**13.3 Flows and Connectivity**

**13.4 Matchings, Transversals, and Vertex Covers**

---

### INTRODUCTION

*Flow in a network* can mean literally the flow of oil or water through a system of pipelines. More often in scientific writing, it refers to the flow of electricity, phone calls, email messages, commodities being transported across truck routes, or other such kinds of flow. Indeed, the richness of the network-flow model extends well beyond these applications.

The classical theory of network flows bridges several diverse and seemingly unrelated areas of combinatorial optimization. The equivalences among the celebrated *Max-Flow Min-Cut Theorem* of Ford and Fulkerson, the connectivity theorems of Menger, and the *Marriage Theorem* of Phillip Hall have led to the development of efficient algorithms for a number of practical problems. These include calculating the edge- and vertex-connectivity of a graph and finding *matchings*, which are used to solve various scheduling and assignment problems and which have applications in other areas of operations research, computer science, and engineering.

## 13.1 FLOWS AND CUTS IN NETWORKS

A pipeline network for transporting oil from a single source to a single sink is one prototype of a network model. Each arc represents a section of pipeline, and the endpoints of an arc correspond to the junctures at the ends of that section. The arc capacity is the maximum amount of oil that can flow through the corresponding section per unit of time. A network could just as naturally represent a system of truck routes for transporting commodities from one point to another, or it could represent a network of phone lines from one distribution center to another.

### Single Source–Single Sink Capacitated Networks

**DEFINITION:** A **single source–single sink network** is a connected digraph that has a distinguished vertex called the **source**, with nonzero outdegree, and a distinguished vertex called the **sink**, with nonzero indegree.

**TERMINOLOGY:** A single source–single sink network with source  $s$  and sink (or *target*)  $t$  is often referred to as an  **$s$ - $t$  network**.

**DEFINITION:** A **capacitated network** is a connected digraph such that each arc  $e$  is assigned a nonnegative weight  $\text{cap}(e)$ , called the **capacity** of arc  $e$ .

**Remark:** Later in this chapter, several applications with no apparent connection to networks are approached by transforming them into network problems, thereby demonstrating the robustness of the network model.

**TERMINOLOGY:** All of the networks discussed in this chapter are assumed to be capacitated  $s$ - $t$  networks, even when one or both of the modifiers are dropped.

**DEFINITION:** Let  $v$  be a vertex in a digraph  $N$ .

- The **out-set** of  $v$ , denoted  $\text{Out}(v)$ , is the set of all arcs that are directed *from* vertex  $v$ . That is,

$$\text{Out}(v) = \{e \in E_N \mid \text{tail}(e) = v\}$$

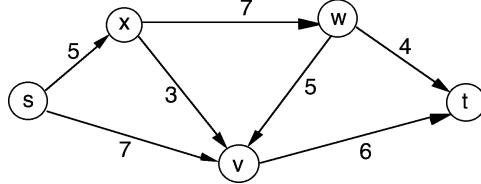
- The **in-set** of  $v$ , denoted  $\text{In}(v)$ , is the set of all arcs that are directed *to* vertex  $v$ . That is,

$$\text{In}(v) = \{e \in E_N \mid \text{head}(e) = v\}$$

**NOTATION:** For any two vertex subsets  $X$  and  $Y$  of a digraph  $N$ , let  $\langle X, Y \rangle$  denote the set of all arcs in  $N$  that are directed *from* a vertex in  $X$  to a vertex in  $Y$ . That is,

$$\langle X, Y \rangle = \{e \in E_N \mid \text{tail}(e) \in X \text{ and } \text{head}(e) \in Y\}$$

**Example 13.1.1:** A 5-vertex capacitated  $s$ - $t$  network is shown in Figure 13.1.1. If  $X = \{x, v\}$  and  $Y = \{w, t\}$ , then the elements of arc set  $\langle X, Y \rangle$  are the arc directed from vertex  $x$  to vertex  $w$  and the arc directed from vertex  $v$  to sink  $t$ . The only element in arc set  $\langle Y, X \rangle$  is the arc directed from vertex  $w$  to vertex  $v$ .



**Figure 13.1.1** A 5-vertex capacitated network with source  $s$  and sink  $t$ .

**Remark:** The examples and applications throughout this chapter involve networks with integer capacities, which simplifies the exposition. There is no additional complication if capacities are non-integer rational numbers. Such a network can be transformed into an “equivalent” one whose capacities are integers by multiplying each capacity by the least common multiple of the denominators of the capacities (see Exercises).

### Feasible Flows

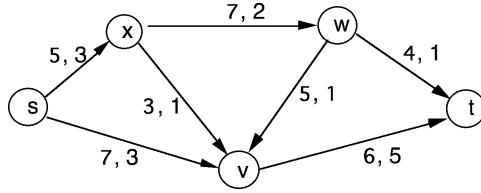
**DEFINITION:** Let  $N$  be a capacitated  $s$ - $t$  network. A (**feasible**) **flow**  $f$  in  $N$  is a function  $f : E_N \rightarrow R^+$  that assigns a nonnegative real number  $f(e)$  to each arc  $e$  such that:

1. (*capacity constraints*)  $f(e) \leq \text{cap}(e)$ , for every arc  $e$  in network  $N$ .
2. (*conservation constraints*)  $\sum_{e \in \text{In}(v)} f(e) = \sum_{e \in \text{Out}(v)} f(e)$ , for every vertex  $v$  in network  $N$ , other than source  $s$  and sink  $t$ .

**TERMINOLOGY:** Property 2 in the flow definition is called the **conservation-of-flow** condition. For an oil pipeline, it says that the total flow of oil going into any juncture (vertex) in the pipeline must equal the total flow leaving that juncture.

**NOTATION:** To distinguish visually between the flow and the capacity of an arc, we adopt the convention in drawings that when both numbers appear, the capacity will always be in bold and to the left of the flow.

**Example 13.1.2:** Figure 13.1.2 shows a feasible flow for the capacitated network of Example 13.1.1. Notice that the total amount of flow leaving source  $s$  equals 6, which is also the net flow entering sink  $t$ . The conservation of flow at every internal vertex in the network is intuitively consistent with this phenomenon. Later in this section, we establish the general result that outflow from the source equals inflow to the sink.



**Figure 13.1.2** A flow for the example network.

**DEFINITION:** The **value of flow**  $f$  in a capacitated network, denoted  $\text{val}(f)$ , is the net flow leaving the source  $s$ , that is,

$$\text{val}(f) = \sum_{e \in \text{Out}(s)} f(e) - \sum_{e \in \text{In}(s)} f(e)$$

**DEFINITION:** A **maximum flow**  $f^*$  in a capacitated network  $N$  is a flow in  $N$  having the maximum value, i.e.,  $\text{val}(f) \leq \text{val}(f^*)$ , for every flow  $f$  in  $N$ .

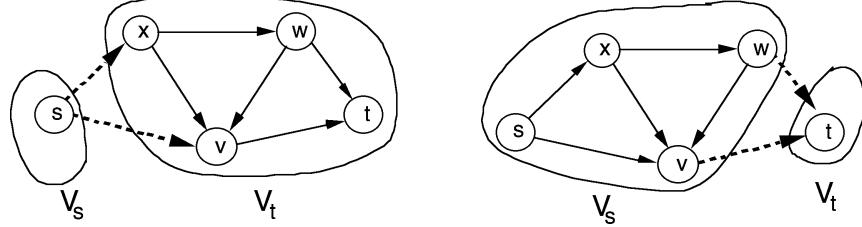
### Cuts in $s$ - $t$ Networks

By definition, any nonzero flow must use at least one of the arcs in  $\text{Out}(s)$ . In other words, if all of the arcs in  $\text{Out}(s)$  were deleted from network  $N$ , then no flow could get from source  $s$  to sink  $t$ . This is a special case of the following definition, which combines the concepts of *partition-cut* (from §4.5) and *s*- $t$  *separating set* (from §5.3).

**DEFINITION:** Let  $N$  be an  $s$ - $t$  network, and let  $V_s$  and  $V_t$  form a partition of  $V_N$  such that source  $s \in V_s$  and sink  $t \in V_t$ . Then the set of all arcs that are directed from a vertex in set  $V_s$  to a vertex in set  $V_t$  is called an  $s$ - $t$  **cut** of network  $N$  and is denoted  $\langle V_s, V_t \rangle$ .

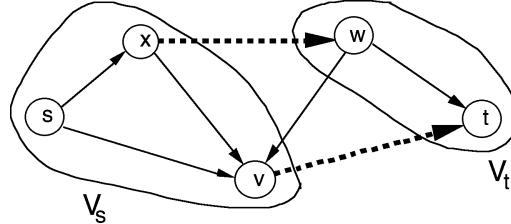
**Remark:** Notice that the arc sets  $\text{Out}(s)$  and  $\text{In}(t)$  for an  $s$ - $t$  network  $N$  are the  $s$ - $t$  cuts  $\langle \{s\}, V_N - \{s\} \rangle$  and  $\langle V_N - \{t\}, \{t\} \rangle$ , respectively.

**Example 13.1.3:** Figure 13.1.3 portrays the arc sets  $\text{Out}(s)$  and  $\text{In}(t)$  as  $s$ - $t$  cuts, where  $\text{Out}(s) = \langle \{s\}, \{x, v, w, t\} \rangle$  and  $\text{In}(t) = \langle \{s, x, v, w\}, \{t\} \rangle$ .



**Figure 13.1.3** Arc sets  $\text{Out}(s)$  and  $\text{In}(t)$  shown as  $s$ - $t$  cuts.

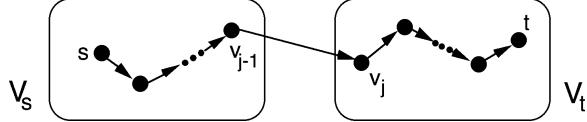
**Example 13.1.4:** A more general  $s$ - $t$  cut  $\langle V_s, V_t \rangle$  is shown in Figure 13.1.4, where  $V_s = \{s, x, v\}$  and  $V_t = \{w, t\}$ .



**Figure 13.1.4** The  $s$ - $t$  cut  $\langle \{s, x, v\}, \{w, t\} \rangle$ .

**Proposition 13.1.1.** Let  $\langle V_s, V_t \rangle$  be an  $s$ - $t$  cut of a network  $N$ . Then every directed  $s$ - $t$  path in  $N$  contains at least one arc in  $\langle V_s, V_t \rangle$ .

**Proof:** Let  $P = \langle s = v_0, v_1, v_2, \dots, v_l = t \rangle$  be the vertex sequence of a directed  $s$ - $t$  path in network  $N$ . Since  $s \in V_s$  and  $t \in V_t$ , there must be a first vertex  $v_j$  on this path that is in set  $V_t$  (see Figure 13.1.5). Then the arc from vertex  $v_{j-1}$  to  $v_j$  is in  $\langle V_s, V_t \rangle$ .  $\diamond$



**Figure 13.1.5** A directed  $s$ - $t$  path with an arc in  $\langle V_s, V_t \rangle$ .

### Relationship Between Flows and Cuts

Similar to viewing the set  $Out(s)$  of arcs directed from source  $s$  as the  $s$ - $t$  cut  $\langle \{s\}, V_N - \{s\} \rangle$ , the set  $In(s)$  may be regarded as the set of “backward” arcs relative to this cut, namely, the arc set  $\langle V_N - \{s\}, \{s\} \rangle$ . From this perspective, the definition of  $val(f)$  may be rewritten as

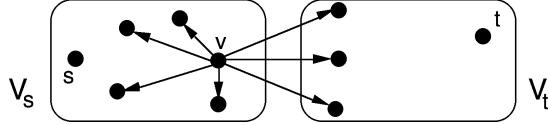
$$val(f) = \sum_{e \in \langle \{s\}, V_N - \{s\} \rangle} f(e) - \sum_{e \in \langle V_N - \{s\}, \{s\} \rangle} f(e)$$

In other words, the value of any flow equals the total flow across the arcs of the cut  $\langle \{s\}, V_N - \{s\} \rangle$  minus the flow across the arcs of  $\langle V_N - \{s\}, \{s\} \rangle$ . The next proposition generalizes this property to all  $s$ - $t$  cuts. Its proof uses the following simple consequence of the definitions.

**Lemma 13.1.2.** Let  $\langle V_s, V_t \rangle$  be any  $s$ - $t$  cut of an  $s$ - $t$  network  $N$ . Then

$$\bigcup_{v \in V_s} Out(v) = \langle V_s, V_s \rangle \cup \langle V_s, V_t \rangle \quad \text{and} \quad \bigcup_{v \in V_t} In(v) = \langle V_s, V_s \rangle \cup \langle V_t, V_s \rangle$$

**Proof:** For any vertex  $v \in V_s$ , each arc directed from  $v$  is either in  $\langle V_s, V_s \rangle$  or in  $\langle V_s, V_t \rangle$ . (Figure 13.1.6 illustrates for a vertex  $v$ , the partition of  $Out(v)$  into a 4-element subset of  $\langle V_s, V_s \rangle$  and a 3-element subset of  $\langle V_t, V_s \rangle$ .) Similarly, each arc directed to vertex  $v$  is either in  $\langle V_s, V_s \rangle$  or in  $\langle V_t, V_s \rangle$ .  $\diamond$



**Figure 13.1.6**  $\bigcup_{v \in V_s} Out(v) = \langle V_s, V_s \rangle \cup \langle V_s, V_t \rangle$

**Proposition 13.1.3.** Let  $f$  be a flow in an  $s$ - $t$  network  $N$ , and let  $\langle V_s, V_t \rangle$  be any  $s$ - $t$  cut of  $N$ . Then

$$val(f) = \sum_{e \in \langle V_s, V_t \rangle} f(e) - \sum_{e \in \langle V_t, V_s \rangle} f(e)$$

**Proof:** By definition,  $val(f) = \sum_{e \in Out(s)} f(e) - \sum_{e \in In(s)} f(e)$ , and by the conservation of flow,  $\sum_{e \in Out(v)} f(e) - \sum_{e \in In(v)} f(e) = 0$  for every  $v \in V_s$  other than  $s$ . Thus,

$$val(f) = \sum_{v \in V_s} \left( \sum_{e \in Out(v)} f(e) - \sum_{e \in In(v)} f(e) \right) = \sum_{v \in V_s} \sum_{e \in Out(v)} f(e) - \sum_{v \in V_s} \sum_{e \in In(v)} f(e)$$

By Lemma 13.1.2,

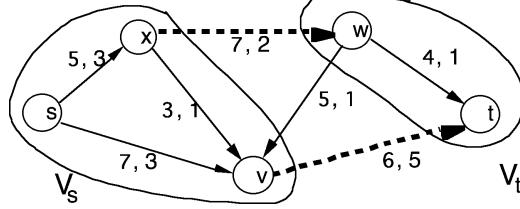
$$\sum_{v \in V_s} \sum_{e \in \text{Out}(v)} f(e) = \sum_{e \in \langle V_s, V_s \rangle} f(e) + \sum_{e \in \langle V_s, V_t \rangle} f(e)$$

and

$$\sum_{v \in V_s} \sum_{e \in \text{In}(v)} f(e) = \sum_{e \in \langle V_s, V_s \rangle} f(e) + \sum_{e \in \langle V_t, V_s \rangle} f(e)$$

The result follows by substitution.  $\diamond$

**Example 13.1.5:** The flow  $f$  and cut  $\{s, x, v\}, \{w, t\}$ , shown in Figure 13.1.7, illustrate Proposition 13.1.3.



**Figure 13.1.7**  $6 = \text{val}(f) = \sum_{e \in \langle \{s, x, v\}, \{w, t\} \rangle} f(e) - \sum_{e \in \langle \{w, t\}, \{s, x, v\} \rangle} f(e) = 7 - 1$

The next result confirms what was apparent earlier from intuition, namely, that the net flow out of source  $s$  equals the net flow into sink  $t$ .

**Corollary 13.1.4.** Let  $f$  be a flow in an  $s$ - $t$  network. Then

$$\text{val}(f) = \sum_{e \in \text{In}(t)} f(e) - \sum_{e \in \text{Out}(t)} f(e)$$

**Proof:** Apply Proposition 13.1.3 to the  $s$ - $t$  cut  $\text{In}(t) = \langle V_N - \{t\}, \{t\} \rangle$ .  $\diamond$

**DEFINITION:** The **capacity of a cut**  $\langle V_s, V_t \rangle$ , denoted  $\text{cap}\langle V_s, V_t \rangle$ , is the sum of the capacities of the arcs in cut  $\langle V_s, V_t \rangle$ . That is,

$$\text{cap}\langle V_s, V_t \rangle = \sum_{e \in \langle V_s, V_t \rangle} \text{cap}(e)$$

**DEFINITION:** A **minimum cut** of a network  $N$  is a cut with the minimum capacity.

**Example 13.1.6:** The capacity of the cut shown in Figure 13.1.7 is 13, and the cut  $\langle \{s, x, v, w\}, \{t\} \rangle$ , with capacity 10, is the only minimum cut.

### The Maximum-Flow Problem and the Minimum-Cut Problem

The next few results demonstrate that the problems of finding the maximum flow in a capacitated network  $N$  and finding a minimum cut in  $N$  are closely related. In fact, these two optimization problems form a *max-min pair*, much like the max-min pair that appears in §5.3.

The first result provides an upper bound for the maximum-flow problem.

**Proposition 13.1.5.** Let  $f$  be any flow in an  $s$ - $t$  network  $N$ , and let  $\langle V_s, V_t \rangle$  be any  $s$ - $t$  cut. Then

$$\text{val}(f) \leq \text{cap}\langle V_s, V_t \rangle$$

**Proof:** The following chain of inequalities, which begins with the assertion of Proposition 13.1.3, establishes the result.

$$\begin{aligned} \text{val}(f) &= \sum_{e \in \langle V_s, V_t \rangle} f(e) - \sum_{e \in \langle V_t, V_s \rangle} f(e) && (\text{by Proposition 13.1.3}) \\ &\leq \sum_{e \in \langle V_s, V_t \rangle} \text{cap}(e) - \sum_{e \in \langle V_t, V_s \rangle} f(e) && (\text{by capacity constraints}) \\ &= \text{cap}\langle V_s, V_t \rangle - \sum_{e \in \langle V_t, V_s \rangle} f(e) && (\text{by definition of } \text{cap}\langle V_s, V_t \rangle) \\ &\leq \text{cap}\langle V_s, V_t \rangle && (\text{since each } f(e) \text{ is nonnegative}) \quad \diamond \end{aligned}$$

The following corollary resembles the weak-duality result in §5.3.

**Corollary 13.1.6 [Weak Duality for Flows].** Let  $f^*$  be a maximum flow in an  $s$ - $t$  network  $N$ , and let  $K^*$  be a minimum  $s$ - $t$  cut in  $N$ . Then

$$\text{val}(f^*) \leq \text{cap}(K^*)$$

**Proof:** This is an immediate consequence of Proposition 13.1.5.  $\diamond$

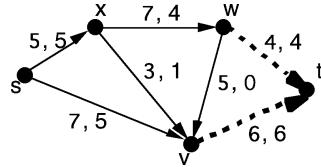
**Corollary 13.1.7 [Certificate of Optimality for Flows].** Let  $f$  be a flow in an  $s$ - $t$  network  $N$  and  $K$  an  $s$ - $t$  cut, and suppose that  $\text{val}(f) = \text{cap}(K)$ . Then flow  $f$  is a maximum flow in network  $N$ , and cut  $K$  is a minimum cut.

**Proof:** Let  $\hat{f}$  be any feasible flow in network  $N$ . The maximality of flow  $f$  is implied by Proposition 13.1.5 and the premise, as follows.

$$\text{val}(\hat{f}) \leq \text{cap}(K) = \text{val}(f)$$

The minimality of cut  $K$  can be established with a similar argument (see Exercises).  $\diamond$

**Example 13.1.7:** The flow for the example network shown in Figure 13.1.8 has value 10, which also is the capacity of the  $s$ - $t$  cut  $\langle \{s, x, v, w\}, \{t\} \rangle$ . By Corollary 13.1.7, both the flow and the cut are optimal for their respective problems.



**Figure 13.1.8** A maximum flow and minimum cut.

The final result of this section is used in the next section to establish the correctness of a classical algorithm for finding the maximum flow in a network.

**Corollary 13.1.8.** Let  $\langle V_s, V_t \rangle$  be an  $s$ - $t$  cut in a network  $N$ , and suppose that  $f$  is a flow such that

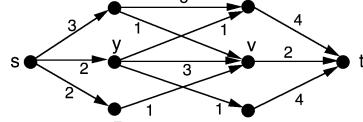
$$f(e) = \begin{cases} \text{cap}(e) & \text{if } e \in \langle V_s, V_t \rangle \\ 0 & \text{if } e \in \langle V_t, V_s \rangle \end{cases}$$

Then  $f$  is a maximum flow in  $N$ , and  $\langle V_s, V_t \rangle$  is a minimum cut. ◊ (Exercises)

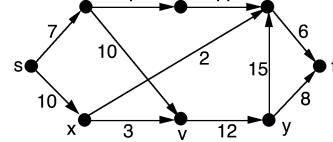
### EXERCISES for Section 13.1

For Exercises 13.1.1 through 13.1.4, use trial and error and Certificate-of-Optimality Corollary 13.1.7 to find a maximum flow and a minimum cut in the given network.

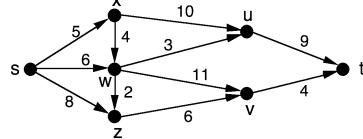
13.1.1<sup>s</sup>



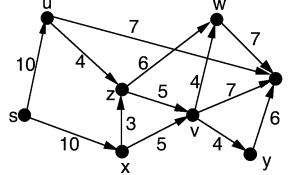
13.1.2



13.1.3

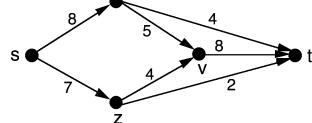


13.1.4

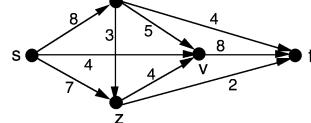


For Exercises 13.1.5 and 13.1.6, find all  $s$ - $t$  cuts and identify all the minimum ones for the capacitated  $s$ - $t$  network shown.

13.1.5<sup>s</sup>



13.1.6

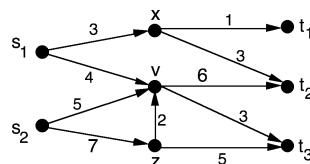


13.1.7 Show that if there exists no directed  $s$ - $t$  path in an  $s$ - $t$  network, then there does not exist a nonzero flow.

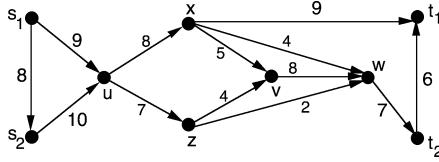
13.1.8 Some authors define an  $s$ - $t$  network such that the source  $s$  must have indegree 0 and the sink  $t$  must have outdegree 0. Describe how a network whose source and sink do not satisfy these additional requirements can be transformed by graphical operations into one whose source and sink do meet the requirements.

13.1.9<sup>s</sup> a. Describe how a multi-source, multi-sink network can be transformed into a network with a single source and single sink, so that the methods in this section can be used to analyze it.

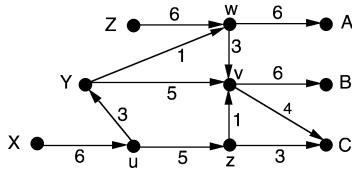
b. Use your transformation to solve the problem of finding the maximum total flow from the two sources to the three sinks in the network shown. Use Certificate-of-Optimality Corollary 13.1.7 to confirm your result.



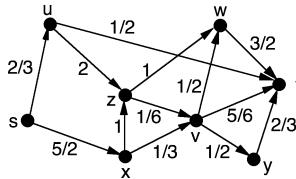
- 13.1.10 Use the transformation given in Exercise 13.1.9 to find the maximum total flow from the two sources to the two sinks in the network shown. Use Certificate-of-Optimality Corollary 13.1.7 to confirm your result.



- 13.1.11 Use the transformation given in Exercise 13.1.9 to find the maximum total flow from the three sources  $X$ ,  $Y$ , and  $Z$  to the three sinks  $A$ ,  $B$ , and  $C$  in the network shown. Use Certificate-of-Optimality Corollary 13.1.7 to confirm your result.

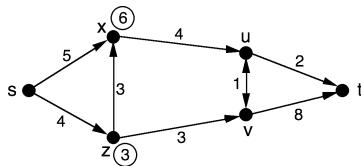


- 13.1.12 Find a maximum flow and a minimum cut for the network shown, by first transforming the network to one whose capacities are all integers. Use Certificate-of-Optimality Corollary 13.1.7 to confirm your result.



- 13.1.13<sup>s</sup> a. Suppose that some of the vertices in an  $s$ - $t$  network have capacities on the amount of flow that can pass through them. Describe how to transform such a network to the standard one used in this section.

- b. Apply the transformation in part (a) to find the maximum flow in the network shown. The circled numbers represent vertex capacities. Apply Certificate-of-Optimality Corollary 13.1.7 to the transformed network to confirm your result.



- 13.1.14 Let  $A$  be a subset of arcs in an  $s$ - $t$  network  $N$ , and suppose that every directed  $s$ - $t$  path contains at least one arc of set  $A$ .

- a. Show that there exists an  $s$ - $t$  cut  $\langle V_s, V_t \rangle$  in network  $N$  such that  $\langle V_s, V_t \rangle \subseteq A$ .  
b. Give an example where the  $s$ - $t$  cut is a proper subset of  $A$ .

- 13.1.15 Complete the proof of Certificate-of-Optimality Corollary 13.1.7.

- 13.1.16<sup>s</sup> Prove Corollary 13.1.8.

## 13.2 SOLVING THE MAXIMUM-FLOW PROBLEM

The basic idea of the algorithm presented in this section, originated by Ford and Fulkerson [FoFu62], is to increase the flow in a network iteratively until it cannot be increased any further. Each increase is based on a suitably chosen alternating sequence of vertices and arcs, called an *augmenting flow path*. Before introducing the most general such path, we consider the simplest and most obvious one.

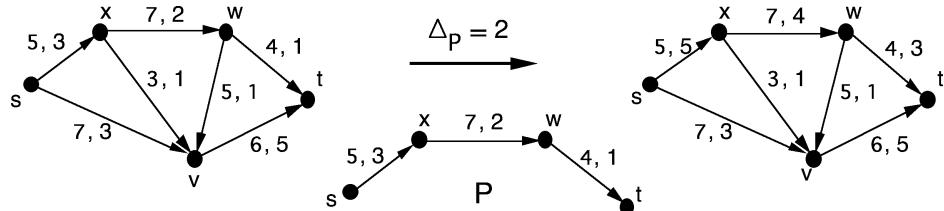
### Using Directed Paths to Augment Flow

Suppose that  $f$  is a flow in a capacitated  $s$ - $t$  network  $N$ , and suppose that there exists a directed  $s$ - $t$  path

$$P = \langle s, e_1, v_1, e_2, \dots, e_k, t \rangle$$

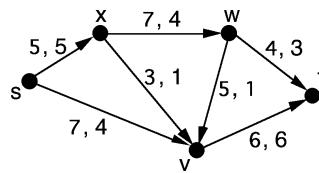
in  $N$ , such that  $f(e_i) < \text{cap}(e_i)$  for  $i = 1, \dots, k$ . Then considering arc capacities only, the flow on each arc  $e_i$  can be increased by as much as  $\text{cap}(e_i) - f(e_i)$ . But to maintain the conservation-of-flow property at each of the vertices  $v_i$ , the increases on all of the arcs of path  $P$  must be equal. Thus, if  $\Delta_P$  denotes this increase, then the largest possible value for  $\Delta_P$  is  $\min\{\text{cap}(e_i) - f(e_i)\}$ .

**Example 13.2.1:** In the example network shown on the left in Figure 13.2.1, the value of the current flow is 6. Consider the directed  $s$ - $t$  path  $P = \langle s, x, w, t \rangle$ . The flow on each arc of path  $P$  can increase by  $\Delta_P = 2$ , and the resulting flow, which has value 8, is shown on the right.



**Figure 13.2.1** Using directed path  $P$  to increase the flow from 6 to 8.

Using the directed path  $\langle s, v, t \rangle$ , the flow can then be increased to 9. The resulting flow is shown in Figure 13.2.2. At this point, the flow cannot be increased any further along directed  $s$ - $t$  paths, because each such path must use either the arc directed from source  $s$  to vertex  $x$  or the one from vertex  $v$  to sink  $t$ , and both these arcs have flow at capacity.



**Figure 13.2.2** Current flow cannot be increased along a directed path.

However, the flow can be increased further. One way to accomplish this is to increase the flow on the arc from source  $s$  to vertex  $v$  by one unit, *decrease* the flow on the arc from  $w$  to  $v$  by one unit, and increase the flow on the arc from  $w$  to  $t$  by one

unit. The decrease in the flow on the arc from  $w$  to  $v$  has the effect of redirecting one unit of the flow from vertex  $w$ , so that instead of going to vertex  $v$ , it goes directly to sink  $t$ . This makes room for the extra unit of flow on the arc from source  $s$  to vertex  $v$ .

**Remark:** Notice that in increasing the flow from 9 to 10 in the example network, the arcs whose flow have changed form an  $s$ - $t$  path *if directions are ignored*. A generalization of a directed path helps in replacing the ad hoc strategy used above with a systematic one.

### $f$ -Augmenting Paths

**DEFINITION:** An  $s$ - $t$  **quasi-path** in a network  $N$  is an alternating sequence

$$\langle s = v_0, e_1, v_1, \dots, v_{k-1}, e_k, v_k = t \rangle$$

of vertices and arcs that forms an  $s$ - $t$  path in the underlying graph of  $N$ .

**TERMINOLOGY:** For a given  $s$ - $t$  quasi-path

$$Q = \langle s = v_0, e_1, v_1, \dots, v_{k-1}, e_k, v_k = t \rangle$$

arc  $e_i$  is called a **forward arc** if it is directed from vertex  $v_{i-1}$  to vertex  $v_i$ , and arc  $e_i$  is called a **backward arc** if it is directed from  $v_i$  to  $v_{i-1}$ .

**Remark:** Clearly, a directed  $s$ - $t$  path is a quasi-path whose arcs are all forward.

**TERMINOLOGY NOTE:** Some authors use the term *semipath* instead of *quasi-path*.

**Example 13.2.2:** On the  $s$ - $t$  quasi-path shown in Figure 13.2.3, arcs  $a$  and  $b$  are backward, and the three other arcs are forward.



**Figure 13.2.3** A quasi-path with two backward arcs.

**DEFINITION:** Let  $f$  be a flow in an  $s$ - $t$  network  $N$ . An  **$f$ -augmenting path**  $Q$  is an  $s$ - $t$  quasi-path in  $N$  such that the flow on each forward arc can be increased, and the flow on each backward arc can be decreased.

Thus, for each arc  $e$  on an  $f$ -augmenting path  $Q$ ,

$$\begin{aligned} f(e) &< \text{cap}(e), & \text{if } e \text{ is a forward arc} \\ f(e) &> 0, & \text{if } e \text{ is a backward arc} \end{aligned}$$

**NOTATION:** For each arc  $e$  on a given  $f$ -augmenting path, let  $\Delta_e$  be the quantity given by

$$\Delta_e = \begin{cases} \text{cap}(e) - f(e), & \text{if } e \text{ is a forward arc} \\ f(e), & \text{if } e \text{ is a backward arc} \end{cases}$$

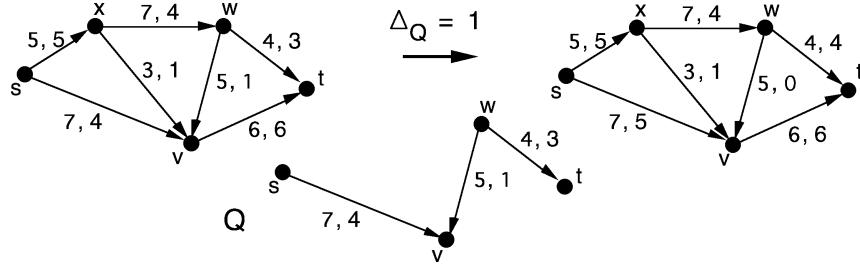
**TERMINOLOGY:** The quantity  $\Delta_e$  is called the **slack on arc**  $e$ . Its value on a forward arc is the largest possible increase in the flow, and on a backward arc, the largest possible decrease in the flow, *disregarding conservation of flow*.

**Remark:** Conservation of flow requires that the change in the flow on the arcs of an augmenting flow path be of equal magnitude. Thus, the maximum allowable change in the flow on an arc of quasipath  $Q$  is  $\Delta_Q$ , where

$$\Delta_Q = \min_{e \in Q} \{\Delta_e\}$$

Notice that this definition of  $\Delta_Q$  coincides with that of  $\Delta_P$ , defined earlier, whenever the quasi-path  $Q$  is a directed path.

**Example 13.2.3:** For the example network in Figure 13.2.4, the current flow  $f$  has value 9, and the quasi-path  $Q = \langle s, v, w, t \rangle$  is an  $f$ -augmenting path, with  $\Delta_Q = 1$ .



**Figure 13.2.4** Using an  $f$ -augmenting path  $Q$  to increase the flow by  $\Delta_Q = 1$ .

**TERMINOLOGY NOTE:** To simplify the terminology, the prefix *quasi-* was not used in the definition of  *$f$ -augmenting path*. But to underscore that an  $f$ -augmenting path is not necessarily a directed path, the letter “Q” is often used for the name of these quasi-paths.

The following proposition summarizes how an  $f$ -augmenting path is used to increase the flow  $f$  in a network.

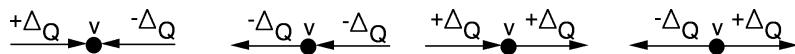
**Proposition 13.2.1 [Flow Augmentation].** Let  $f$  be a flow in a network  $N$ , and let  $Q$  be an  $f$ -augmenting path with minimum slack  $\Delta_Q$  on its arcs. Then the augmented flow  $\hat{f}$  given by

$$\hat{f}(e) = \begin{cases} f(e) + \Delta_Q, & \text{if } e \text{ is a forward arc of } Q \\ f(e) - \Delta_Q, & \text{if } e \text{ is a backward arc of } Q \\ f(e), & \text{otherwise} \end{cases}$$

is a feasible flow in network  $N$ , and  $\text{val}(\hat{f}) = \text{val}(f) + \Delta_Q$ .

**Proof:** Clearly,  $0 \leq \hat{f}(e) \leq \text{cap}(e)$ , by the definition of  $\Delta_Q$ . The only vertices through which the net flow may have changed are those vertices on the augmenting path  $Q$ . Thus, to verify that  $\hat{f}$  satisfies conservation of flow, only the internal vertices of  $Q$  need to be checked.

For a given vertex  $v$  on augmenting path  $Q$ , the two arcs of  $Q$  that are incident on  $v$  are configured in one of four ways, as illustrated in Figure 13.2.5. In each case, the net flow into or out of vertex  $v$  does not change, thereby preserving the conservation-of-flow property.



**Figure 13.2.5** The four possibilities for an internal vertex  $v$  of a quasi-path.

It remains to be shown that the flow has increased by  $\Delta_Q$ . The only arc incident on source  $s$  whose flow has changed is the first arc  $e_1$  of augmenting path  $Q$ . If  $e_1$  is a forward arc, then  $\hat{f}(e_1) = f(e_1) + \Delta_Q$ , and if  $e_1$  is a backward arc, then  $\hat{f}(e_1) = f(e_1) - \Delta_Q$ . In either case,

$$\text{val}(\hat{f}) = \sum_{e \in \text{Out}(s)} \hat{f}(e) - \sum_{e \in \text{In}(s)} \hat{f}(e) = \Delta_Q + \text{val}(f)$$

◊

**Corollary 13.2.2.** *Let  $f$  be an integer-valued flow in a network whose arc capacities are integers. Then the flow that results from each successive flow augmentation will be integer-valued.* ◊

### Max-Flow Min-Cut Theorem

**Theorem 13.2.3 [Characterization of Maximum Flow].** *Let  $f$  be a flow in a network  $N$ . Then  $f$  is a maximum flow in network  $N$  if and only if there does not exist an  $f$ -augmenting path in  $N$ .*

**Proof:** *Necessity ( $\Rightarrow$ )* Suppose that  $f$  is a maximum flow in network  $N$ . Then by Proposition 13.2.1, there is no  $f$ -augmenting path.

*Sufficiency ( $\Leftarrow$ )* Suppose that there does not exist an  $f$ -augmenting path in network  $N$ . Consider the collection of all quasi-paths that start at vertex  $s$  and have the following property: each forward arc on the quasi-path has positive slack (i.e., it can be increased), and each backward arc on the quasi-path has positive flow (i.e., it can be decreased). Let  $V_s$  be the union of the vertex-sets of these quasi-paths.

Since there is no  $f$ -augmenting path, it follows that sink  $t \notin V_s$ . Let  $V_t = V_N - V_s$ . Then  $\langle V_s, V_t \rangle$  is an  $s$ - $t$  cut of network  $N$ . Moreover, by definition of the sets  $V_s$  and  $V_t$ ,

$$f(e) = \begin{cases} \text{cap}(e) & \text{if } e \in \langle V_s, V_t \rangle \\ 0 & \text{if } e \in \langle V_t, V_s \rangle \end{cases}$$

Hence,  $f$  is a maximum flow, by Corollary 13.1.8. ◊

**Theorem 13.2.4 [Max-Flow Min-Cut].** *For a given network, the value of a maximum flow is equal to the capacity of a minimum cut.*

**Proof:** The  $s$ - $t$  cut constructed in the proof of Theorem 13.2.3 has capacity equal to value of the maximum flow. ◊

The outline of an algorithm (shown below) for maximizing the flow in a network emerges from Proposition 13.2.1 and Theorem 13.2.3.

### Finding an $f$ -Augmenting Path

The discussion of  $f$ -augmenting paths culminating in the flow-augmentation Proposition 13.2.1 provides the basis of a vertex-labeling strategy, due to Ford and Fulkerson, that finds an  $f$ -augmenting path, when one exists. Their labeling scheme is essentially basic *tree-growing* (§4.1). Here, the idea is to grow a tree of quasi-paths, each starting at source  $s$ . If the flow on each arc of these quasi-paths can be increased or decreased,

**Algorithm 13.2.1: Outline for Maximum Flow**

*Input:* an  $s$ - $t$  network  $N$ .

*Output:* a maximum flow  $f^*$  in network  $N$ .

[Initialization]

For each arc  $e$  in network  $N$

$$f^*(e) := 0$$

[Flow Augmentation]

While there exists an  $f^*$ -augmenting path in network  $N$

Find an  $f^*$ -augmenting path  $Q$ .

$$\text{Let } \Delta_Q = \min_{e \in Q} \{\Delta_e\}.$$

For each arc  $e$  of augmenting path  $Q$

If  $e$  is a forward arc

$$f^*(e) := f^*(e) + \Delta_Q$$

Else [ $e$  is a backward arc]

$$f^*(e) := f^*(e) - \Delta_Q$$

Return flow  $f^*$ .

according to whether that arc is *forward* or *backward*, then an  $f$ -augmenting path is obtained as soon as the sink  $t$  is labeled.

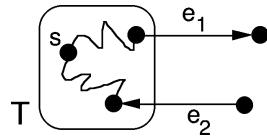
REVIEW FROM §4.1: A **frontier arc** is an arc  $e$  directed from a labeled endpoint  $v$  to an unlabeled endpoint  $w$ .

For constructing an  $f$ -augmenting path, frontier arc  $e$  is allowed to be backward (directed from vertex  $w$  to vertex  $v$ ), and it can be added to the tree as long as it has slack  $\Delta_e > 0$ .

TERMINOLOGY: At any stage during tree-growing for constructing an  $f$ -augmenting path, let  $e$  be a frontier arc of tree  $T$ , with endpoints  $v$  and  $w$ . Then arc  $e$  is said to be **usable** if, for the current flow  $f$ , either

$e$  is directed from vertex  $v$  to vertex  $w$  and  $f(e) < \text{cap}(e)$ , or

$e$  is directed from vertex  $w$  to vertex  $v$  and  $f(e) > 0$ .



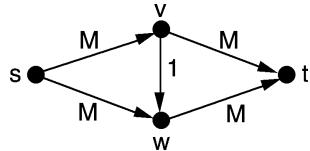
**Figure 13.2.6** Frontier arcs  $e_1$  and  $e_2$  are *usable* if  $f(e_1) < \text{cap}(e_1)$  and  $f(e_2) > 0$ .

**Remark:** From this vertex-labeling scheme, any of the existing  $f$ -augmenting paths could result. But the efficiency of Algorithm 13.2.0 hinges on being able to find “good”  $f$ -augmenting paths. In fact, Ford and Fulkerson showed that if the arc capacities are irrational numbers, then an algorithm using their labeling scheme may not terminate (strictly speaking, with irrational labels, it is not an algorithm).

But even when flows and capacities are restricted to be integers, problems concerning efficiency still exist. For instance, if each flow augmentation were to increase the flow by only one unit, then the number of augmentations required for maximization would

equal the capacity of a minimum cut. Such an algorithm would depend on the size of the arc capacities instead of on the size of the network.

**Example 13.2.4:** For the network shown in Figure 13.2.7, the arc from vertex  $v$  to vertex  $w$  has flow capacity 1, and the other arcs have capacity  $M$ , which could be made arbitrarily large. If the choice of augmenting flow path at each iteration were to alternate between the directed path  $\langle s, v, w, t \rangle$  and the quasi-path  $\langle s, w, v, t \rangle$ , then the flow would increase by only one unit at each iteration. Thus, it could take as many as  $2M$  iterations to obtain the maximum flow.



**Figure 13.2.7** This network could require as many as  $2M$  augmentations.

Edmonds and Karp avoid these problems with the following algorithm, which is a slight refinement of the Ford-Fulkerson labeling scheme [EdKa72]. It uses *breadth-first search* to find an  $f$ -augmenting path with the least number of arcs. Algorithm 13.2.2 either returns an  $f$ -augmenting path or returns a minimum cut that indicates the current flow  $f$  is a maximum flow.

**Algorithm 13.2.2: Finding an Augmenting Path**

*Input:* a flow  $f$  in an  $s$ - $t$  network  $N$ .

*Output:* an  $f$ -augmenting path  $Q$  or a minimum  $s$ - $t$  cut with capacity  $\text{val}(f)$ .

```

Initialize vertex set  $V_s := \{s\}$ .
Write label 0 on vertex  $s$ .
Initialize label counter  $i := 1$ 
While vertex set  $V_s$  does not contain sink  $t$ 
  If there are usable arcs
    Let  $e$  be a usable arc whose labeled endpoint  $v$ 
      has the smallest possible label.
    Let  $w$  be the unlabeled endpoint of arc  $e$ .
    Set  $\text{backpoint}(w) := v$ .
    Write label  $i$  on vertex  $w$ .
     $V_s := V_s \cup \{w\}$ 
     $i := i + 1$ 
  Else
    Return  $s$ - $t$  cut  $\langle V_s, V_N - V_s \rangle$ .
Reconstruct the  $f$ -augmenting path  $Q$  by following backpointers,
  starting from sink  $t$ .
Return  $f$ -augmenting path  $Q$ .
  
```

Algorithm 13.2.3 combines Algorithms 13.2.0 and 13.2.2. The “FFEK” in the algorithm name refers to Ford, Fulkerson, Edmonds, and Karp.

**Algorithm 13.2.3: FFEK - Maximum Flow**

*Input:* an  $s$ - $t$  network  $N$ .

*Output:* a maximum flow  $f^*$  in network  $N$ .

[Initialization]

For each arc  $e$  in network  $N$

$$f^*(e) := 0$$

[Flow Augmentation]

Repeat

    Apply Algorithm 13.2.2 to find an  $f^*$ -augmenting path  $Q$ .

$$\text{Let } \Delta_Q = \min_{e \in Q} \{\Delta_e\}.$$

    For each arc  $e$  of augmenting path  $Q$

        If  $e$  is a forward arc

$$f^*(e) := f^*(e) + \Delta_Q$$

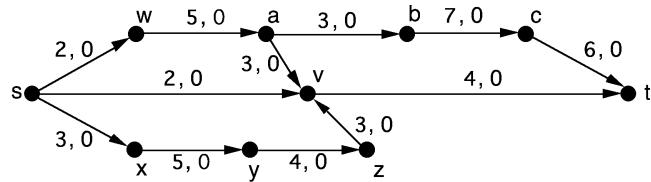
        Else [ $e$  is a backward arc]

$$f^*(e) := f^*(e) - \Delta_Q$$

Until an  $f^*$ -augmenting path cannot be found in network  $N$ .

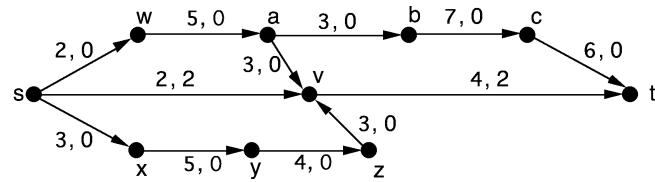
Return flow  $f^*$ .

**Example 13.2.5:** The following sequence of figures illustrates Algorithm 13.2.3 for the example network shown, starting with zero flow. Each of the first three figures shows the current flow in the network and a shortest  $f$ -augmenting path (in number of arcs) for that flow, which is then used to get the flow shown in the subsequent figure. The final figure in this example shows an  $s$ - $t$  cut with capacity equal to the value of the current flow, thereby establishing optimality.



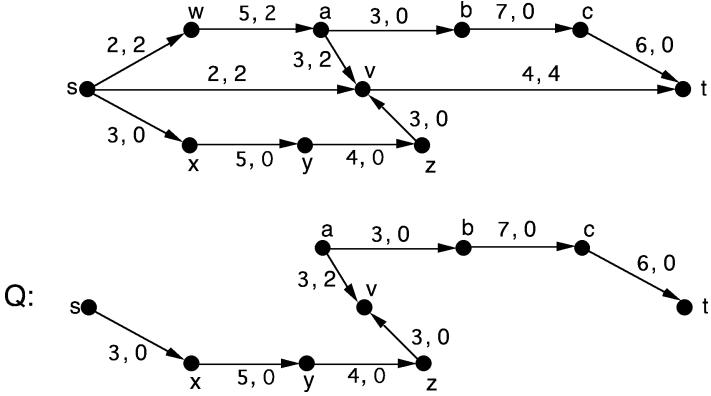
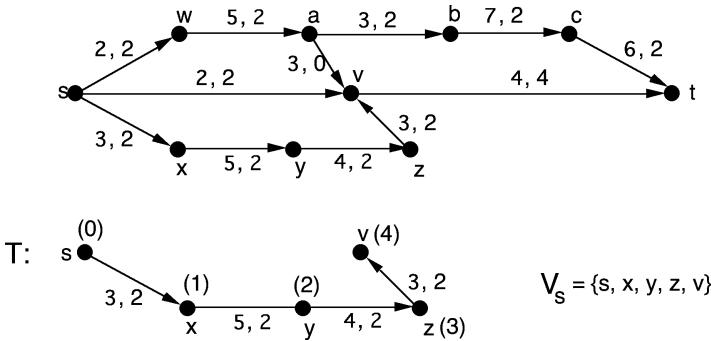
Q:  $s \rightarrow v \rightarrow t$

**Figure 13.2.8** Iteration 0:  $\text{val}(f) = 0$ ; augmenting path  $Q$  has  $\Delta_Q = 2$ .



Q:  $s \rightarrow w \rightarrow a \rightarrow v \rightarrow t$

**Figure 13.2.9** Iteration 1:  $\text{val}(f) = 2$ ; augmenting path  $Q$  has  $\Delta_Q = 2$ .

**Figure 13.2.10** Iteration 2:  $\text{val}(f) = 4$ ; augmenting path  $Q$  has  $\Delta_Q = 2$ .**Figure 13.2.11** Final iteration:  $\text{val}(f) = 6 = \text{cap}(\{s, x, y, z, v\}, \{w, a, b, c, t\})$ .

Notice that at the end of the final iteration, the arc directed from source  $s$  to vertex  $w$  and the arc directed from vertex  $v$  to sink  $t$  are the only frontier arcs of tree  $T$ , but neither is usable. These two arcs form the minimum cut  $(\{s, x, y, z, v\}, \{w, a, b, c, t\})$ . This illustrates the  $s$ - $t$  cut that was constructed in the proof of Theorem 13.2.3.

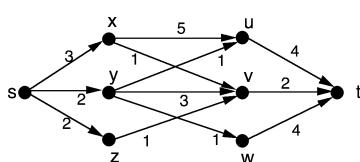
**COMPUTATIONAL NOTE:** Edmonds and Karp [EdKa72] show that Algorithm 13.2.3 finds a maximum flow in no more than  $\frac{|E_N|(|V_N|+2)}{2}$  augmentations. Since the computation for breadth-first search is  $O(|E_N|)$ , it follows that the overall computation of Algorithm 13.2.3 is  $O(|E_N|^2|V_N|)$ . There are more efficient algorithms that are considerably more complicated and outside the scope of this book (see [Mi78] or [ThSw92]).

### EXERCISES for Section 13.2

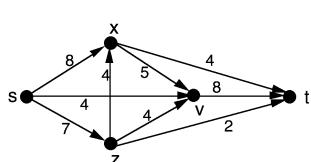
13.2.1<sup>s</sup> Starting with the flow of value 9 given in Example 13.2.3, obtain a different maximum flow by using a different  $f$ -augmenting path.

For Exercises 13.2.2 through 13.2.6, apply Maximum-Flow Algorithm 13.2.3 to find a maximum flow and minimum cut for the given network.

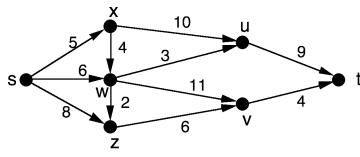
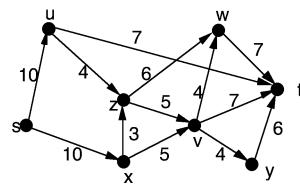
#### 13.2.2



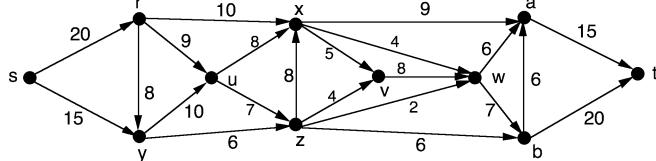
#### 13.2.3



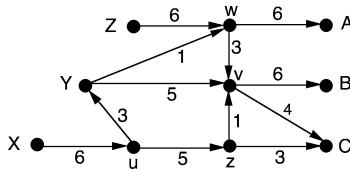
13.2.4

13.2.5<sup>s</sup>

13.2.6



13.2.7 A company maintains three warehouses,  $X$ ,  $Y$ , and  $Z$ , and three stores,  $A$ ,  $B$ , and  $C$ . The warehouses have, respectively, 500, 500, and 900 lawn mowers in stock. There is an immediate demand from the stores for 700, 600, and 600 mowers, respectively. In the graph model shown, the arc capacities represent upper bounds on the number of mowers that can be shipped in a single day on that truck route segment. The intermediate vertices may be regarded as *transshipment points*, where trucks are checked, refueled, maintained, etc. Can all the demand be met? If not, how close can the company come to satisfying the demand?



13.2.8<sup>s</sup> The Johnson, Pate, Shargaa-Sears, and Ward families are going to the Winter Park Sidewalk Art Festival. Four cars are available to transport the families to the show. The cars can carry the following numbers of people: car 1, four; car 2, three; car 3, three; and car 4, four. There are four people in each family, and no car can carry more than two people from any one family. Formulate the problem of transporting the maximum possible number of people to the festival as a maximum-flow problem.

13.2.9 Seven types of chemicals are to be shipped in five trucks. There are three containers storing each type of chemical, and the capacities of the five trucks are 6, 4, 5, 4, and 3 containers, respectively. For security reasons, no truck can carry more than one container of the same chemical. Determine whether it is possible to ship all 21 containers in the five trucks.

13.2.10 Let  $N$  be a slightly more general  $s-t$  network, in which a nonnegative lower bound is specified for the flow on each arc. Prove that there exists a feasible flow in network  $N$  if and only if every arc  $e$  satisfies one of the following conditions:

- a. arc  $e$  is on a directed circuit of  $N$ ;
- b. arc  $e$  is on a directed path from source  $s$  to sink  $t$ ;
- c. arc  $e$  is on a directed path from sink  $t$  to source  $s$ .

13.2.11 Modify Maximum-Flow Algorithm 13.2.3 so that it finds a maximum flow in a network that has multiple sources and multiple sinks (see Exercise 13.1.9).

13.2.12 Modify Maximum-Flow Algorithm 13.2.3 so that it finds a maximum flow in a network in which a nonnegative lower bound is specified for the flow on each arc.

**13.2.13** Design an algorithm whose input is an  $s$ - $t$  network  $N$  and whose output is an arc with the property that increasing its capacity will increase the maximum flow in network  $N$ .

**13.2.14 [Computer Project]** Implement Maximum-Flow Algorithm 13.2.3 and test the program on the networks in Exercises 13.2.2 through 13.2.6.

## 13.3 FLOWS AND CONNECTIVITY

The theory of network flows is used in this section to give constructive proofs of Menger's Theorems, which were introduced in §5.3. These proofs lead directly to algorithms for determining the edge-connectivity and vertex-connectivity of a graph.

The strategy behind using network flows to prove Menger's Theorems is based on properties of networks whose arcs all have unit capacity. These *0-1 networks* are constructed from the original graph. The next few results are used to establish certain properties of 0-1 networks needed later.

**Lemma 13.3.1.** *Let  $N$  be an  $s$ - $t$  network such that  $\text{outdegree}(s) > \text{indegree}(s)$ ,  $\text{indegree}(t) > \text{outdegree}(t)$ , and  $\text{outdegree}(v) = \text{indegree}(v)$  for all other vertices  $v$ . Then there exists a directed  $s$ - $t$  path in network  $N$ .*

**Proof:** Let  $W$  be a longest directed trail in network  $N$  that starts at source  $s$ , and let  $z$  be its terminal vertex. If vertex  $z$  were not the sink, then there would be an arc not in trail  $W$  that is directed from  $z$  (since  $\text{indegree}(z) = \text{outdegree}(z)$ ). But this would contradict the maximality of trail  $W$ . Thus,  $W$  is a directed trail from source  $s$  to sink  $t$ . If  $W$  has a repeated vertex, then part of  $W$  determines a directed cycle, which can be deleted from  $W$  to obtain a shorter directed  $s$ - $t$  trail. This deletion step can be repeated until no repeated vertices remain, at which point, the resulting directed trail is an  $s$ - $t$  path. ◇

**Proposition 13.3.2.** *Let  $N$  be an  $s$ - $t$  network such that*

$$\begin{aligned} \text{outdegree}(s) - \text{indegree}(s) &= m = \text{indegree}(t) - \text{outdegree}(t), \\ \text{and} \quad \text{outdegree}(v) &= \text{indegree}(v), \quad \text{for all vertices } v \neq s, t. \end{aligned}$$

*Then there exist  $m$  arc-disjoint directed  $s$ - $t$  paths in network  $N$ .*

**Proof:** If  $m = 1$ , then there exists an open eulerian directed trail  $T$  from source  $s$  to sink  $t$ , by Theorem 6.1.3. By Theorem 1.5.2, trail  $T$  is either an  $s$ - $t$  directed path or can be reduced to an  $s$ - $t$  path.

By way of induction, assume that the assertion is true for  $m = k$ , for some  $k \geq 1$ , and consider a network  $N$  for which the condition holds for  $m = k + 1$ . There exists a directed  $s$ - $t$  path  $P$  by Lemma 13.3.1. If the arcs of path  $P$  are deleted from network  $N$ , then the resulting network  $\hat{N}$  satisfies the condition of the proposition for  $m = k$ . By the induction hypothesis, there exist  $k$  arc-disjoint directed  $s$ - $t$  paths in network  $\hat{N}$ . These  $k$  paths together with path  $P$  form a collection of  $k + 1$  arc-disjoint directed  $s$ - $t$  paths in network  $N$ . ◇

## Two Basic Properties of 0-1 Networks

**DEFINITION:** A **0-1 network** is a capacitated network whose arc capacities are either 0 or 1.

**Proposition 13.3.3.** Let  $N$  be an  $s-t$  network such that  $\text{cap}(e) = 1$  for every arc  $e$ . Then the value of a maximum flow in network  $N$  equals the maximum number of arc-disjoint directed  $s-t$  paths in  $N$ .

**Proof:** Let  $f^*$  be a maximum flow in network  $N$ , and let  $r$  be the maximum number of arc-disjoint directed  $s-t$  paths in  $N$ . Consider the network  $N^*$ , obtained by deleting from  $N$  all arcs  $e$  such that  $f^*(e) = 0$ . Then  $f^*(e) = 1$  for all arcs  $e$  in network  $N^*$ . It follows from the definitions that for every vertex  $v$  in network  $N^*$ ,

$$\sum_{e \in \text{Out}(v)} f^*(e) = |\text{Out}(v)| = \text{outdegree}(v) \quad \text{and} \quad \sum_{e \in \text{In}(v)} f^*(e) = |\text{In}(v)| = \text{indegree}(v)$$

Thus, by the definition of  $\text{val}(f^*)$  and by the conservation-of-flow property,

$$\begin{aligned} \text{outdegree}(s) - \text{indegree}(s) &= \text{val}(f^*) = \text{indegree}(t) - \text{outdegree}(t) \\ \text{and} \quad \text{outdegree}(v) &= \text{indegree}(v), \quad \text{for all vertices } v \neq s, t. \end{aligned}$$

By Proposition 13.3.2, there are  $\text{val}(f^*)$  arc-disjoint directed  $s-t$  paths in network  $N^*$  and, hence, also in  $N$ , which implies that  $\text{val}(f^*) \leq r$ .

To obtain the reverse inequality, let  $\{P_1, P_2, \dots, P_r\}$  be a largest collection of arc-disjoint directed  $s-t$  paths in  $N$ , and consider the function  $f : E_N \rightarrow R^+$  defined by

$$f(e) = \begin{cases} 1, & \text{if some path } P_i \text{ uses arc } e \\ 0, & \text{otherwise.} \end{cases}$$

Then  $f$  is a feasible flow in network  $N$ , with  $\text{val}(f) = r$  (see Exercises). It follows that  $\text{val}(f^*) \leq r$ .  $\diamond$

## Separating Sets and Cuts

The next proposition is stated in terms of a digraph analogue of an  $s-t$  **separating edge set** (introduced in Chapter 5).

**REVIEW FROM §5.3:** Let  $s$  and  $t$  be distinct vertices in a graph  $G$ . An  $s-t$  **separating edge set** in  $G$  is a set of edges whose removal destroys all  $s-t$  paths in  $G$ , i.e., an edge subset that contains at least one edge of every  $s-t$  path in  $G$ .

**DEFINITION:** Let  $s$  and  $t$  be distinct vertices in a digraph  $D$ . An  $s-t$  **separating arc set** in  $D$  is a set of arcs whose removal destroys all directed  $s-t$  paths in  $D$ .

Thus, an  $s-t$  separating arc set in  $D$  is an arc subset of  $E_D$  that contains at least one arc of every directed  $s-t$  path in digraph  $D$ .

**Remark:** For the degenerate case in which the original graph or digraph has no  $s-t$  paths, the empty set is regarded as an  $s-t$  separating set.

**Proposition 13.3.4.** Let  $N$  be an  $s$ - $t$  network such that  $\text{cap}(e) = 1$  for every arc  $e$ . Then the capacity of a minimum  $s$ - $t$  cut in network  $N$  equals the minimum number of arcs in an  $s$ - $t$  separating arc set in  $N$ .

**Proof:** Let  $K^* = \langle V_s, V_t \rangle$  be a minimum  $s$ - $t$  cut in network  $N$ , and let  $q$  be the minimum number of arcs in an  $s$ - $t$  separating arc set in  $N$ . Since  $K^*$  is an  $s$ - $t$  cut, it is also an  $s$ - $t$  separating arc set. Thus,  $\text{cap}(K^*) \geq q$ .

To obtain the reverse inequality, let  $S$  be an  $s$ - $t$  separating arc set in network  $N$  containing  $q$  arcs, and let  $R$  be the set of all vertices in  $N$  that are reachable from source  $s$  by a directed path that contains no arc from set  $S$ . Then, by the definitions of arc set  $S$  and vertex set  $R$ ,  $t \notin R$ , which means that  $\langle R, V_N - R \rangle$  is an  $s$ - $t$  cut. Moreover,  $\langle R, V_N - R \rangle \subseteq S$ . Therefore,

$$\begin{aligned} \text{cap}(K^*) &\leq \text{cap}(\langle R, V_N - R \rangle) && (\text{since } K^* \text{ is a minimum } s\text{-}t \text{ cut}) \\ &= |\langle R, V_N - R \rangle| && (\text{since all capacities are 1}) \\ &\leq |S| && (\text{since } \langle R, V_N - R \rangle \subseteq S) \\ &= q \end{aligned}$$

which completes the proof.  $\diamond$

### Arc and Edge Versions of Menger's Theorem Revisited

**Theorem 13.3.5 [Arc Form of Menger's Theorem].** Let  $s$  and  $t$  be distinct vertices in a digraph  $D$ . Then the maximum number of arc-disjoint directed  $s$ - $t$  paths in  $D$  is equal to the minimum number of arcs in an  $s$ - $t$  separating arc set of  $D$ .

**Proof:** Let  $N$  be the  $s$ - $t$  network obtained by assigning a unit capacity to each arc of digraph  $D$ . Then the result follows from Propositions 13.3.3 and 13.3.4, together with the Max-Flow Min-Cut Theorem (Theorem 13.2.4).  $\diamond$

**Remark:** Conversely, the arc form of Menger's Theorem together with Propositions 13.3.3 and 13.3.4 can be used to prove the Max-Flow Min-Cut Theorem (see Exercises).

The edge form of Menger's Theorem for undirected graphs follows directly from the next two assertions concerning the relationship between a graph  $G$  and the digraph  $\overset{\leftrightarrow}{G}$  obtained by replacing each edge  $e$  of graph  $G$  with a pair of oppositely directed arcs having the same endpoints as edge  $e$ . Each of these assertions follows directly from the definitions.

**Assertion 13.3.6.** Let  $s$  and  $t$  be distinct vertices of a graph  $G$ , and let  $\overset{\leftrightarrow}{G}$  be the digraph obtained by replacing each edge  $e$  of  $G$  with a pair of oppositely directed arcs having the same endpoints as edge  $e$ . Then there is a one-to-one correspondence between the  $s$ - $t$  paths in graph  $G$  and the directed  $s$ - $t$  paths in digraph  $\overset{\leftrightarrow}{G}$ . Moreover, if two directed  $s$ - $t$  paths in  $\overset{\leftrightarrow}{G}$  are arc-disjoint, then their corresponding  $s$ - $t$  paths in  $G$  are edge-disjoint.  $\diamond$  (Exercises)

**Lemma 13.3.7.** Let  $S$  be a minimal  $s$ - $t$  separating arc set of a digraph  $D$ . Then for every pair of vertices  $x, y$  in  $D$ , at most one of the arcs  $a = xy$  and  $b = yx$  is in  $S$ .

**Proof:** Suppose not. Let  $P_a = \langle s, v_1, v_2, \dots, v_j, x, y, v_{j+1}, v_{j+2}, \dots, t \rangle$  be (the vertex sequence of) an  $s$ - $t$  directed path in  $D$  whose only arc in  $S$  is  $a = xy$ , and let  $P_b =$

$\langle s, w_1, w_2, \dots, w_k, y, x, w_{k+1}, w_{k+2}, \dots, t \rangle$  be an  $s$ - $t$  directed path in  $D$  whose only arc in  $S$  is  $b = yx$ . (The minimality of  $S$  guarantees the existence of two such paths.) Then  $P = \langle s, v_1, v_2, \dots, v_j, x, w_{k+1}, w_{k+2}, \dots, t \rangle$  is an  $s$ - $t$  directed path in  $D$  that contains neither arc  $a$  nor arc  $b$ . Thus,  $P$  (and hence  $P_a$  and  $P_b$ ) must contain some other arc in  $S$ , which contradicts the definitions of paths  $P_a$  and  $P_b$ .  $\diamond$

**Assertion 13.3.8.** Let  $s$  and  $t$  be distinct vertices of a graph  $G$ , and let  $\overset{\leftrightarrow}{G}$  be the digraph obtained by replacing each edge  $e$  of  $G$  with a pair of oppositely directed arcs having the same endpoints as edge  $e$ . Then the minimum number of edges in an  $s$ - $t$  separating edge set of graph  $G$  is equal to the minimum number of arcs in an  $s$ - $t$  separating arc set of digraph  $\overset{\leftrightarrow}{G}$ .

**Proof:** Let  $m$  be the size of a minimum  $s$ - $t$  separating edge set  $S$  of  $G$ , and let  $\overset{\leftrightarrow}{m}$  be the size of a minimum  $s$ - $t$  separating arc set of  $\overset{\leftrightarrow}{G}$ . Let  $\overset{\leftrightarrow}{S}$  be the arc set of  $\overset{\leftrightarrow}{G}$  that results when each edge in  $S$  is replaced by its two oppositely directed arcs. Then  $\overset{\leftrightarrow}{S}$  is an  $s$ - $t$  separating arc set of  $\overset{\leftrightarrow}{G}$  of size  $2m$ . Let  $\overset{\leftrightarrow}{S}_*$  be a minimal  $s$ - $t$  separating arc set of  $\overset{\leftrightarrow}{G}$  contained in  $\overset{\leftrightarrow}{S}$ . Lemma 13.3.7 implies that  $|\overset{\leftrightarrow}{S}_*| \leq |S|$ , and hence,  $\overset{\leftrightarrow}{m} \leq m$ . Now let  $\overset{\leftrightarrow}{T}_*$  be a minimum  $s$ - $t$  separating arc set of  $\overset{\leftrightarrow}{G}$ . Then the corresponding edge set  $T$  that results from ignoring directions is an  $s$ - $t$  separating edge set of  $G$ , from which it follows that  $m \leq \overset{\leftrightarrow}{m}$ .  $\diamond$

**Theorem 13.3.9 [Edge Form of Menger's Theorem].** Let  $s$  and  $t$  be distinct vertices in a graph  $G$ . Then the maximum number of edge-disjoint  $s$ - $t$  paths in  $G$  equals the minimum number of edges in an  $s$ - $t$  separating edge set of graph  $G$ .

**Proof:** Let  $m$  and  $M$  be the sizes of a minimum  $s$ - $t$  separating edge set and a maximum set of edge-disjoint  $s$ - $t$  paths, respectively, in graph  $G$ , and let  $\overset{\leftrightarrow}{m}$  and  $\overset{\leftrightarrow}{M}$  be the sizes of a minimum  $s$ - $t$  separating arc set and a maximum set of arc-disjoint  $s$ - $t$  directed paths, respectively, in digraph  $\overset{\leftrightarrow}{G}$ .

If  $\overset{\leftrightarrow}{F}_*$  is a maximum set of arc-disjoint  $s$ - $t$  directed paths in  $\overset{\leftrightarrow}{G}$ , then by Assertion 13.3.6, the corresponding set  $F$  of  $s$ - $t$  paths in  $G$  is edge-disjoint, which implies that

$$\overset{\leftrightarrow}{M} = |\overset{\leftrightarrow}{F}_*| = |F| \leq M$$

By Assertion 13.3.8 and Theorem 13.3.5, we have

$$\overset{\leftrightarrow}{m} = \overset{\leftrightarrow}{M} \leq M \leq m = \overset{\leftrightarrow}{m}$$

where the second inequality follows because it takes at least  $m$  edges to destroy  $m$  edge-disjoint paths.  $\diamond$

### Determining Edge-Connectivity Using Network Flows

**REVIEW FROM §5.1:** The **edge-connectivity**  $\kappa_e(G)$  is the size of a smallest edge-cut in graph  $G$ .

**DEFINITION:** The **local edge-connectivity**  $\kappa_e(s, t)$  between distinct vertices  $s$  and  $t$  in a graph  $G$  is the minimum number of edges in an  $s$ - $t$  separating edge set in  $G$ .

The following proposition shows that the edge-connectivity of a graph can be expressed in terms of the local edge-connectivity between all pairs of vertices. The assertion and its proof are analogous to Lemma 5.3.5 and its proof in §5.3.

**Proposition 13.3.10.** *The edge-connectivity of a graph  $G$  is equal to the minimum of the local edge-connectivities, taken over all pairs of vertices  $s$  and  $t$ . That is,*

$$\kappa_e(G) = \min_{s,t \in V_G} \{\kappa_e(s,t)\} \quad \diamondsuit \text{ (Exercises)}$$

Proposition 13.3.10 and the results used in the proof of Theorem 13.3.9 suggest an algorithm for determining the edge-connectivity  $\kappa_e(G)$  of an arbitrary graph  $G$ . The algorithm calculates the local edge-connectivity between each pair of vertices in  $G$ , by solving an appropriate maximum-flow problem in the network  $\overset{\leftrightarrow}{G}$  (referred to in Assertions 13.3.6 and 13.3.8). In fact, it is not necessary to calculate the local edge-connectivity between every pair of vertices, as the next two results show. Both results make use of the relationship between local edge-connectivity and *partition-cuts*.

REVIEW FROM §4.5: Let  $G$  be a graph, and let  $V_1$  and  $V_2$  form a partition of  $V_G$ . The set of all edges of  $G$  having one endpoint in vertex subset  $V_1$  and the other endpoint in vertex subset  $V_2$  is called a **partition-cut** of  $G$  and is denoted  $\langle V_1, V_2 \rangle$ .

**Proposition 13.3.11.** *Let  $\langle V_1, V_2 \rangle$  be a partition-cut of minimum cardinality in a graph  $G$ , and let  $v_1$  and  $v_2$  be any vertices in  $V_1$  and  $V_2$ , respectively. Then the edge-connectivity  $\kappa_e(G)$  equals the local edge-connectivity  $\kappa_e(v_1, v_2)$ .*

**Proof:** Suppose that the minimum local edge-connectivity is achieved between vertices  $x$  and  $y$ . Then  $\kappa_e(G) = \kappa_e(x, y)$  (by Proposition 13.3.10). It suffices to show that  $\kappa_e(v_1, v_2) \leq \kappa_e(x, y)$ .

Let  $\overset{\leftrightarrow}{G}$  be the digraph obtained by replacing each edge of graph  $G$  with two oppositely directed arcs. Then digraph  $\overset{\leftrightarrow}{G}$  can be regarded as a  $v_1$ - $v_2$  capacitated network  $\overset{\leftrightarrow}{G}_{v_1 v_2}$  and as an  $x$ - $y$  capacitated network  $\overset{\leftrightarrow}{G}_{xy}$ , where each arc is assigned unit capacity. Let  $K^*$  be a minimum  $v_1$ - $v_2$  cut in network  $\overset{\leftrightarrow}{G}_{v_1 v_2}$ . It follows that  $\text{cap}(K^*) \leq \text{cap}(\langle V_1, V_2 \rangle)$ , since the partition-cut  $\langle V_1, V_2 \rangle$  corresponds to a  $v_1$ - $v_2$  cut in network  $\overset{\leftrightarrow}{G}_{v_1 v_2}$ .

Next, let  $f^*$  be a maximum flow and  $\langle V_x, V_y \rangle$  a minimum  $x$ - $y$  cut in  $x$ - $y$  network  $\overset{\leftrightarrow}{G}_{xy}$ , so that  $\text{cap}(\langle V_x, V_y \rangle) = \text{val}(f^*)$ . Then the following chain of inequalities establishes the desired inequality.

$$\begin{aligned} \kappa_e(v_1, v_2) &= \text{cap}(K^*) && (\text{Proposition 13.3.4 and Assertion 13.3.8}) \\ &\leq \text{cap}(\langle V_1, V_2 \rangle) && (\langle V_1, V_2 \rangle \text{ corresponds to a } v_1\text{-}v_2 \text{ cut in } \overset{\leftrightarrow}{G}_{v_1 v_2}) \\ &= |\langle V_1, V_2 \rangle| && (\text{all arcs have unit capacity}) \\ &\leq |\langle V_x, V_y \rangle| && (\langle V_x, V_y \rangle \text{ corresponds to a partition-cut in } G) \\ &= \text{cap}(\langle V_x, V_y \rangle) && (\text{all arcs have unit capacity}) \\ &= \text{val}(f^*) && (\text{Max-Flow Min-Cut Theorem 13.2.4}) \\ &= \kappa_e(x, y) && (\text{Proposition 13.3.3 and Theorem 13.3.5}) \end{aligned}$$

◊

**Corollary 13.3.12.** Let  $s$  be any vertex in a graph  $G$ . Then

$$\kappa_e(G) = \min_{t \in V_G - \{s\}} \{\kappa_e(s, t)\}$$

**Proof:** Let  $\langle V_1, V_2 \rangle$  be a partition-cut of minimum cardinality, and suppose, without loss of generality, that vertex  $s \in V_1$ . There must be some vertex  $t \in V_2$  (otherwise,  $E_G = \emptyset$ , and the assertion would be trivially true). By Proposition 13.3.11, it follows that  $\kappa_e(G) = \kappa_e(s, t)$ .  $\diamond$

The variable  $\kappa_e$ , used in the next algorithm, represents the edge-connectivity of graph  $G$  and is initialized with the sufficiently large positive integer  $|E_G|$ .

**Algorithm 13.3.1: Edge-Connectivity Calculation**

*Input:* a graph  $G$ .

*Output:* the edge-connectivity  $\kappa_e$  of graph  $G$ .

Construct digraph  $\overset{\leftrightarrow}{G}$ .

Let  $s$  be an arbitrary vertex of graph  $G$ .

Initialize edge-connectivity  $\kappa_e := |E_G|$ .

For each vertex  $t \in V_G - \{s\}$

    Apply Algorithm 13.2.3 to  $s-t$  network  $\overset{\leftrightarrow}{G}$  to obtain maximum flow  $f^*$ .

    If  $\text{val}(f^*) < \kappa_e$

$\kappa_e := \text{val}(f^*)$

Return  $\kappa_e$ .

**COMPUTATIONAL NOTE:** Algorithm 13.3.1 requires  $O(n)$  iterations, and since Algorithm 13.2.3 requires  $O(n|E|^2)$  computations, the overall computation of Algorithm 13.3.1 is  $O(n^2|E|^2)$ . This can be reduced if one of the more efficient maximum-flow algorithms cited earlier is used instead of Algorithm 13.2.3.

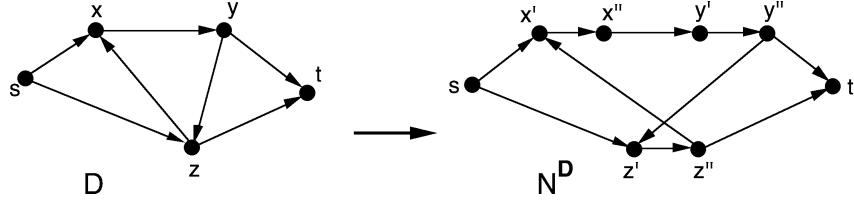
### Using Network Flows to Prove the Vertex Forms of Menger's Theorem

The vertex form of Menger's Theorem for digraphs and graphs can be proved using the arc and edge forms (Theorems 13.3.5 and 13.3.9). The proofs are based on the following construction.

#### Construction of Digraph $N^D$ From a Digraph $D$ :

Let  $s$  and  $t$  be any pair of non-adjacent vertices in a digraph  $D$ . The digraph  $N^D$  is obtained from digraph  $D$  as follows:

- Each vertex  $x \in V_D - \{s, t\}$  corresponds to two vertices  $x'$  and  $x''$  in digraph  $N^D$  and an arc directed from  $x'$  to  $x''$ .
- Each arc in digraph  $D$  that is directed from vertex  $s$  to a vertex  $x \in V_D - \{s, t\}$  corresponds to an arc in digraph  $N^D$  directed from  $s$  to  $x'$ .
- Each arc in  $D$  that is directed from a vertex  $x \in V_D - \{s, t\}$  to vertex  $t$  corresponds to an arc in  $N^D$  directed from  $x''$  to  $t$ .
- Each arc in  $D$  that is directed from a vertex  $x \in V_D - \{s, t\}$  to a vertex  $y \in V_D - \{s, t\}$  corresponds to an arc in  $N^D$  directed from  $x''$  to  $y'$ .



**Figure 13.3.1** The digraph  $N^D$  corresponding to a digraph  $D$ .

REVIEW FROM §5.3: Let  $s$  and  $t$  be a pair of non-adjacent vertices in a graph  $G$  (or digraph  $D$ ). An  $s$ - $t$  **separating vertex set** in  $G$  (or  $D$ ) is a set of vertices whose removal destroys all  $s$ - $t$  paths in  $G$  (or all directed  $s$ - $t$  paths in  $D$ ).

Thus, an  $s$ - $t$  separating vertex set is a set of vertices that contains at least one internal vertex of every (directed)  $s$ - $t$  path.

DEFINITION: Two (directed)  $s$ - $t$  paths in a digraph are **internally disjoint** if they have no internal vertices in common.

### Relationships Between Digraphs $D$ and $N^D$

The vertex forms of Menger's Theorem are easily established from the following four relationships between a digraph  $D$  and its corresponding digraph  $N^D$ . Each of these follows from the definitions.

**Assertion 13.3.13.** There is a one-to-one correspondence between directed  $s$ - $t$  paths in digraph  $D$  and directed  $s$ - $t$  paths in digraph  $N^D$ .  $\diamondsuit$  (Exercises)

**Assertion 13.3.14.** Two directed  $s$ - $t$  paths in  $D$  are internally disjoint if and only if their corresponding  $s$ - $t$  directed paths in  $N^D$  are arc-disjoint.  $\diamondsuit$  (Exercises)

**Assertion 13.3.15.** The maximum number of internally disjoint directed  $s$ - $t$  paths in  $D$  is equal to the maximum number of arc-disjoint directed  $s$ - $t$  paths in  $N^D$ .  $\diamondsuit$  (Exercises)

**Assertion 13.3.16.** The minimum number of vertices in an  $s$ - $t$  separating vertex set in digraph  $D$  is equal to the minimum number of arcs in an  $s$ - $t$  separating arc set in digraph  $N^D$ .  $\diamondsuit$  (Exercises)

**Theorem 13.3.17 [Vertex Form of Menger for Digraphs].** Let  $s$  and  $t$  be a pair of non-adjacent vertices in a digraph  $D$ . Then the maximum number of internally disjoint directed  $s$ - $t$  paths in  $D$  is equal to the minimum number of vertices in an  $s$ - $t$  separating vertex set in  $D$ .

**Proof:** This follows from Assertions 13.3.13 through 13.3.16, together with the arc form of Menger's Theorem (Theorem 13.3.5).  $\diamondsuit$

**Theorem 13.3.18 [Vertex Form of Menger for Undirected Graphs].** Let  $s$  and  $t$  be a pair of non-adjacent vertices in a graph  $G$ . Then the maximum number of internally disjoint  $s$ - $t$  paths in  $G$  is equal to the minimum number of vertices in an  $s$ - $t$  separating vertex set in  $G$ .

**Proof:** Let  $\overset{\leftrightarrow}{G}$  be the digraph obtained by replacing each edge  $e$  of  $G$  with a pair of oppositely directed arcs having the same endpoints as edge  $e$ . The result follows by Theorem 13.3.17 and Assertions 13.3.6 and 13.3.8.  $\diamond$

### Determining Vertex-Connectivity Using Network Flows

The proof of the vertex form of Menger's Theorem based on the theory of network flows leads to an algorithm for determining the vertex-connectivity of a graph, in much the same way as the edge form of Menger's Theorem leads to an algorithm for edge-connectivity.

REVIEW FROM §5.3:

- Let  $s$  and  $t$  be non-adjacent vertices of a connected graph. Then the **local vertex-connectivity** between  $s$  and  $t$ , denoted  $\kappa_v(s, t)$ , is the minimum number of vertices in an  $s$ - $t$  separating vertex set.
- Lemma 5.3.5: Let  $G$  be a connected graph containing at least one pair of non-adjacent vertices. Then the vertex-connectivity  $\kappa_v(G)$  is the minimum of the local vertex-connectivity  $\kappa_v(s, t)$ , taken over all pairs of non-adjacent vertices  $s$  and  $t$ .

**Remark:** The following algorithm, which has  $O(n|E_G|^3)$  time-complexity, computes the vertex-connectivity of an  $n$ -vertex graph by calculating the local vertex-connectivity between various pairs of non-adjacent vertices.

#### Algorithm 13.3.2: Vertex-Connectivity Calculation

*Input:* a graph  $G$  with  $V_G = \{v_1, v_2, \dots, v_n\}$ .  
*Output:* the vertex-connectivity  $\kappa_v$  of graph  $G$ .

```

Construct digraph  $\overset{\leftrightarrow}{G}$ .
Initialize vertex-connectivity  $\kappa_v := |V_G|$ .
Initialize index  $k := 0$ .
While  $k \leq \kappa_v$ 
     $k := k + 1$ 
    For  $j = k + 1$  to  $n$ 
        If vertices  $v_k$  and  $v_j$  are not adjacent
            Construct digraph  $N^G$ .
            Assign unit capacity to each arc in digraph  $N^G$ .
            Apply Algorithm 13.2.3 to network  $N^G$  with source  $v_k$ 
                and sink  $v_j$  to obtain maximum flow  $f^*$ .
            If  $val(f^*) < \kappa_v$ 
                 $\kappa_v := val(f^*)$ 
Return  $\kappa_v$ .

```

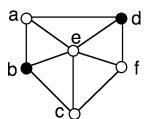
As in Algorithm 13.3.1, it is not necessary to calculate the local vertex-connectivity between every pair. Verification of its correctness and its time-complexity are omitted but may be found in [Ev79].

The variable  $\kappa_v$ , appearing in Algorithm 13.3.3, represents the vertex-connectivity of graph  $G$  and is initialized with the sufficiently large positive integer  $|V_G|$ .

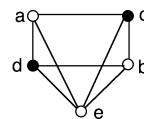
### EXERCISES for Section 13.3

For Exercises 13.3.1 through 13.3.4, find the local edge-connectivity between the pair of solid vertices for the graph shown, by finding the maximum flow in an appropriate network.

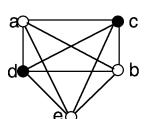
13.3.1<sup>s</sup>



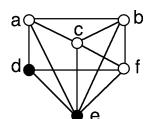
13.3.2



13.3.3



13.3.4



For Exercises 13.3.5 through 13.3.8, apply Algorithm 13.3.1 to find the edge-connectivity of the given graph.

13.3.5<sup>s</sup> The graph of Exercise 13.3.1.

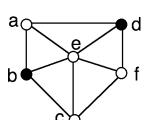
13.3.6 The graph of Exercise 13.3.2.

13.3.7 The graph of Exercise 13.3.3.

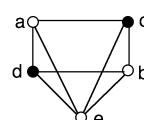
13.3.8 The graph of Exercise 13.3.4.

For Exercises 13.3.9 through 13.3.12, find the local vertex-connectivity between the pair of solid vertices for the graph shown, by finding the maximum flow in an appropriate network.

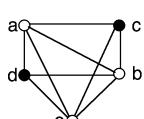
13.3.9<sup>s</sup>



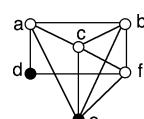
13.3.10



13.3.11



13.3.12



For Exercises 13.3.13 through 13.3.16, apply Algorithm 13.3.2 to find the vertex-connectivity of the given graph.

13.3.13<sup>s</sup> The graph of Exercise 13.3.9.

13.3.14 The graph of Exercise 13.3.10.

13.3.15 The graph of Exercise 13.3.11.

13.3.16 The graph of Exercise 13.3.12.

**13.3.17** Supply the details that were omitted in the proof of Lemma 13.3.1.

**13.3.18<sup>s</sup>** Suppose that  $\{P_1, P_2, \dots, P_r\}$  is a collection of  $r$  arc-disjoint directed  $s-t$  paths in a network  $N$  whose arc capacities are all  $\geq 1$ . Show that the function  $f : E_N \rightarrow R^+$  defined by

$$f(e) = \begin{cases} 1, & \text{if some path } P_i \text{ uses arc } e \\ 0, & \text{otherwise} \end{cases}$$

is a flow in network  $N$ , with  $\text{val}(f) = r$ .

**13.3.19** Prove Assertion 13.3.6.

**13.3.20** Prove Proposition 13.3.10.

**13.3.21<sup>s</sup>** Prove Assertion 13.3.13.

**13.3.22** Prove Assertion 13.3.14.

**13.3.23** Prove Assertion 13.3.15.

**13.3.24** Prove Assertion 13.3.16.

**13.3.25<sup>s</sup>** Prove Max-Flow-Min-Cut Theorem 13.2.4 using Arc-Form-of-Menger's Theorem 13.3.5 together with Propositions 13.3.3 and 13.3.4.

## 13.4 MATCHINGS, TRANSVERSALS, AND VERTEX COVERS

The link between Menger's theorems and the theory of network flows, which was established in the last section, extends to several other combinatorial problems discussed here.

### Matchings in a Graph

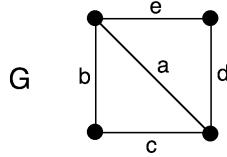
**REVIEW FROM §9.3:** A set  $M$  of edges in a graph  $G$  is called a **matching** in  $G$  if no two edges in set  $M$  have an endpoint in common.

**TERMINOLOGY:** If  $e$ , with endpoints  $x$  and  $y$ , is an edge in a matching  $M$ , then  $M$  **matches vertex  $x$  with vertex  $y$** , or  $x$  is **matched with  $y$  by  $M$** .

**DEFINITION:** A **maximum (-cardinality) matching** is a matching with the greatest number of edges.

**Remark:** A *maximal* matching in a graph  $G$  is a matching that is not a proper subset of any other matching in  $G$ . Clearly, every maximum matching is a maximal matching, but a maximal matching need not be a maximum one.

**Example 13.4.1:** Consider the graph  $G$  shown in Figure 13.4.1. The singleton edge sets  $\{b\}$ ,  $\{c\}$ ,  $\{d\}$  and  $\{e\}$  are the nonempty matchings in graph  $G$  that are *not* maximal. The edge set  $\{a\}$  is a maximal matching in  $G$  that is not maximum, and the edge sets  $\{b, d\}$ ,  $\{c, e\}$  are the maximum matchings in  $G$ .



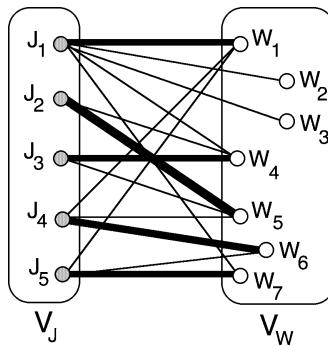
**Figure 13.4.1** Two of the three maximal matchings in  $G$  are maximum.

### Maximum Matching in a Bipartite Graph

REVIEW FROM §1.2: A graph  $G$  is a **bipartite graph** if there exists a vertex bipartition  $\{X, Y\}$  of  $V_G$  such that every edge of  $G$  has one endpoint in vertex set  $X$  and one endpoint in vertex set  $Y$ .

**Application 13.4.1** *The Personnel-Assignment Problem:* Suppose that a company requires a number of different types of jobs, and suppose each worker 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?

The bipartite graph of Figure 13.4.2 has vertex bipartition  $\{V_J, V_W\}$ , where the solid vertices represent the jobs, and the hollow vertices represent the workers. Each edge corresponds to a suitable job assignment. The maximum matching whose edges appear in bold shows that all five jobs can be assigned to workers.



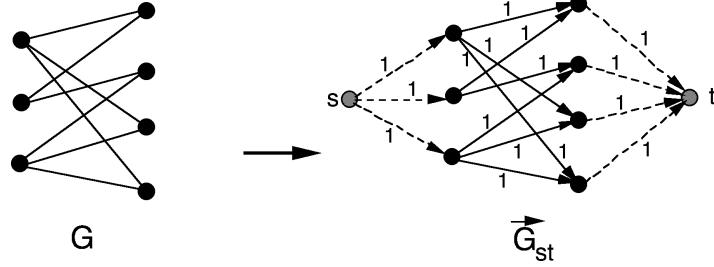
**Figure 13.4.2** A maximum matching that solves the assignment problem.

### Converting a Maximum-Matching Problem into a Maximum-Flow Problem

Let  $G$  be a bipartite graph with vertex bipartition  $\{X, Y\}$ . Then the problem of finding a maximum matching in graph  $G$  can be transformed into a maximum-flow problem in an  $s$ - $t$  network  $\vec{G}_{st}$ , constructed from graph  $G$  as follows:

- (1) The vertex-set of network  $\vec{G}_{st}$  is  $V_G \cup \{s, t\}$ , where  $s$  and  $t$  are two new vertices that are the source and sink, respectively, of  $\vec{G}_{st}$ .
- (2) Each edge in graph  $G$  between a vertex  $x \in X$  and a vertex  $y \in Y$  corresponds to an arc in network  $\vec{G}_{st}$  directed from vertex  $x$  to vertex  $y$ .
- (3) For each vertex  $x \in X$ , an arc in network  $\vec{G}_{st}$  is drawn from source  $s$  to vertex  $x$ .
- (4) For each vertex  $y \in Y$ , an arc in network  $\vec{G}_{st}$  is drawn from vertex  $y$  to sink  $t$ .
- (5) Each arc  $e$  in network  $\vec{G}_{st}$  is assigned a capacity  $cap(e) = 1$ .

**Example 13.4.2:** Figure 13.4.3 shows the bipartite graph from Application 13.4.1 and the corresponding  $s$ - $t$  network  $\vec{G}_{st}$ .



**Figure 13.4.3** A bipartite graph  $G$  and its corresponding network  $\vec{G}_{st}$ .

### Relationship Between Matchings in $G$ and Flows in $\vec{G}_{st}$

The following proposition and its corollary show that a maximum-matching problem can be solved as a maximum-flow problem.

**Proposition 13.4.1.** Let  $G$  be a bipartite graph and  $\vec{G}_{st}$  the  $s$ - $t$  network constructed from  $G$ . Then there is a one-to-one correspondence between the integral flows of network  $\vec{G}_{st}$  and the matchings in graph  $G$ .

**Proof:** Let  $f$  be an integer-valued flow in the network  $\vec{G}_{st}$ . Then the unit capacities imply that  $f(e) = 0$  or  $1$ , for each arc  $e$  in network  $\vec{G}_{st}$ . Let  $M$  be the set of edges in graph  $G$  whose corresponding arcs in network  $\vec{G}_{st}$  have unit flow. It follows from the structure of  $\vec{G}_{st}$  and the conservation-of-flow property that edge set  $M$  is a matching in graph  $G$  (see Exercises).

Conversely, let  $M$  be a matching in graph  $G$ , and for any edge  $e$  in graph  $G$ , let  $\vec{e}$  denote the corresponding arc in network  $\vec{G}_{st}$ . Then for every arc  $a \in \vec{G}_{st}$ , the function defined by

$$f(a) = \begin{cases} 1, & \text{if } a = \vec{e}, \text{ for some edge } e \in M \\ 1, & \text{if arc } a \text{ is adjacent to } \vec{e}, \text{ for some edge } e \in M \\ 0, & \text{otherwise} \end{cases}$$

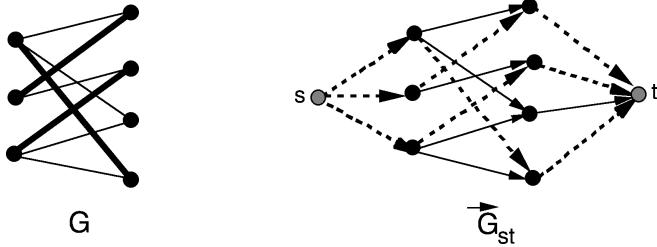
is a flow in network  $\vec{G}_{st}$  (see Exercises).  $\diamond$

**Corollary 13.4.2.** Let  $M$  be a matching in a bipartite graph  $G$  and  $f$  the corresponding flow in network  $\vec{G}_{st}$ . Then  $\text{val}(f) = |M|$ , and  $f$  is a maximum flow if and only if  $M$  is a maximum matching.

**Proof:** This follows from the correspondence established in Proposition 13.4.1.  $\diamond$

**Example 13.4.3:** On the left in Figure 13.4.4 below is a maximum matching of size 3 in a bipartite graph  $G$ . Its corresponding maximum flow in network  $\vec{G}_{st}$  has unit flow on each arc represented as a dashed line.

Algorithm 13.4.1 finds a maximum matching in a bipartite graph  $G$  by finding a maximum flow in the corresponding network  $\vec{G}_{st}$ , as prescribed by Proposition 13.4.1.



**Figure 13.4.4** A maximum matching and its corresponding maximum flow.

**Algorithm 13.4.1: Maximum Bipartite Matching**

*Input:* a bipartite graph  $G$  with vertex bipartition  $\{X, Y\}$ .

*Output:* a maximum matching  $M$  of graph  $G$ .

```

Initialize edge set  $M := \emptyset$ .
Construct the  $s-t$  network  $\vec{G}_{st}$  that corresponds to bipartite graph  $G$ .
Apply Algorithm 13.2.3 to network  $\vec{G}_{st}$  to obtain maximum flow  $f^*$ .
For each arc  $\vec{e}$  in network  $\vec{G}_{st}$ 
    If  $f^*(\vec{e}) = 1$ 
         $M := M \cup \{e\}$ 
Return edge set  $M$ .

```

As in the proposition,  $\vec{e}$  denotes the arc in network  $\vec{G}_{st}$  that corresponds to edge  $e$  in graph  $G$ .

**Remark:** One can view the algorithm above as iteratively augmenting a matching  $M$  by finding an  $M$ -augmenting path, analogous to the flow-augmenting paths of the maximum-flow algorithm. Edmonds [Ed65b] extended this notion by designing an algorithm for finding a maximum matching in a general graph. The Edmonds algorithm is more complicated than a bipartite-matching algorithm, but it and its more recent improvements have running time not much beyond that needed for bipartite matching. The following application illustrates the need for a more general algorithm.

**Application 13.4.2 Pairing Volunteers for a Rescue Mission:** Suppose that several first-aid workers from around the world have volunteered for a rescue mission in some country that has been struck by a disaster. The volunteers are to be divided into two-person teams, but the members of each team must speak the same language. What is the maximum number of pairs that can be sent out on a rescue mission?

A graph model for this problem has a vertex corresponding to each volunteer, and an edge exists between a pair of vertices if the corresponding volunteers speak the same language. Then a maximum matching in the graph corresponds to a maximum set of two-person teams.

### Transversals for Families of Subsets

**DEFINITION:** Let  $A$  be a nonempty finite set, and let  $\mathcal{F} = \{S_1, S_2, \dots, S_r\}$  be a family of (not necessarily distinct) nonempty subsets of set  $A$ . Then a **transversal** (or **system of distinct representatives**) of family  $\mathcal{F}$  is a sequence  $T = \langle a_1, a_2, \dots, a_r \rangle$  of  $r$  distinct elements of set  $A$ , such that  $a_i \in S_i$ , for each  $i = 1, \dots, r$ .

**Example 13.4.4:** Let  $A = \{a, b, c, d, e\}$ , and suppose that  $\mathcal{F} = \{S_1, S_2, \dots, S_5\}$ , where

$$S_1 = \{a, b\}; \quad S_2 = \{b, c, d\}; \quad S_3 = \{c, d, e\}; \quad S_4 = \{d, e\}; \quad \text{and } S_5 = \{e, a, b\}$$

Then  $T = \langle b, c, e, d, a \rangle$  is a transversal for family  $\mathcal{F}$ .

**Application 13.4.3 Pairing Interns with Hospitals:** Suppose that  $r$  medical school graduates have applied for internships at various hospitals. For the  $i$ th medical school graduate, let  $S_i$  be the set of all hospitals that find applicant  $i$  acceptable. Then a transversal for the family  $\mathcal{F} = \{S_1, S_2, \dots, S_r\}$  would assign each potential intern to a hospital that is willing to take him or her, such that no hospital gets assigned more than one intern.

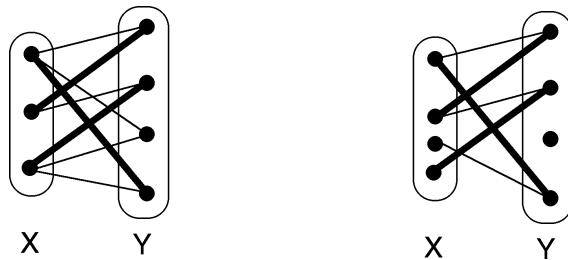
### Relationship Between Bipartite Matchings and Transversals

The problem of finding a transversal for a given family of subsets can be formulated as a matching problem in an associated bipartite graph. The statement of the matching problem uses the following definition.

**DEFINITION:** Let  $G$  be a bipartite graph with vertex bipartition  $\{X, Y\}$ , and let edge set  $M$  be a matching in  $G$ . Then the matching  $M$  is said to be  **$X$ -saturating** if each vertex in set  $X$  is an endpoint of an edge in  $M$ , and to be  **$Y$ -saturating** if each vertex in  $Y$  is an endpoint of an edge in  $M$ .

Notice that every  $X$ -saturating matching and every  $Y$ -saturating matching must be a maximum matching.

**Example 13.4.5:** The maximum matching shown on the left in Figure 13.4.5 is  $X$ -saturating, but the one on the right is not.



**Figure 13.4.5** The matching on the left is  $X$ -saturating.

**TERMINOLOGY NOTE:** Because a matching *matches* vertices in the  $X$ -set to vertices in the  $Y$ -set, some authors refer to an  $X$ -saturating matching as a *complete matching from  $X$  to  $Y$* .

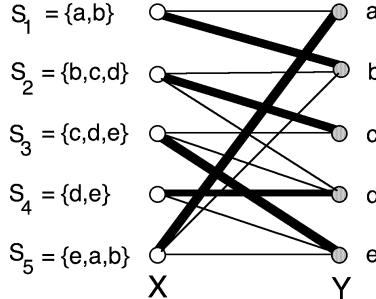
### Finding a Transversal by Finding an X-Saturating Matching

Let  $A = \{a_1, a_2, \dots, a_n\}$  be a set of  $n$  elements and  $\mathcal{F} = \{S_1, S_2, \dots, S_r\}$  be a family of (not necessarily distinct) subsets of set  $A$ . Consider the following bipartite graph  $G$  constructed from  $\mathcal{F}$ . The vertex bipartition  $\{X, Y\}$  of graph  $G$  consists of two vertex sets

$$X = \{x_1, x_2, \dots, x_r\} \text{ and } Y = \{y_1, y_2, \dots, y_n\}$$

where each vertex  $x_i \in X$  corresponds to the subset  $S_i$  in family  $\mathcal{F}$ , and each vertex  $y_j \in Y$  corresponds to the element  $a_j \in A$ . An edge exists between vertex  $x_i$  and vertex  $y_j$  if  $a_j \in S_i$ . Then a transversal for family  $\mathcal{F}$  corresponds to an  $X$ -saturating matching of graph  $G$ .

**Example 13.4.6:** Figure 13.4.6 shows the  $X$ -saturating matching that corresponds to the transversal  $T = \langle b, c, e, d, a \rangle$  from Example 13.4.4.



**Figure 13.4.6** Transversal  $T = \langle b, c, e, d, a \rangle$  and corresponding matching.

### Hall's Theorem

The next theorem, known as *Hall's Theorem*, gives a necessary and sufficient condition for a bipartite graph to have an  $X$ -saturating matching. Its corollary, which is simply a restatement in terms of a family of subsets, characterizes those families of subsets that have a transversal. The graph version uses the following notation.

**NOTATION:** Let  $G$  be a bipartite graph with vertex bipartition  $\{X, Y\}$ , and let  $W$  be a subset of vertex set  $X$ . Then  $N(W)$  denotes the subset of  $Y$  consisting of all vertices in  $Y$  that are adjacent to at least one vertex in set  $W$ . In other words,  $N(W)$  is the set of *neighbors* of set  $W$ .

**DEFINITION:** A bipartite graph with vertex bipartition  $\{X, Y\}$  is said to satisfy **Hall's Condition** if  $|W| \leq |N(W)|$  for every subset  $W$  of  $X$ .

**Theorem 13.4.3 [Hall's Theorem for Bipartite Graphs, 1935].** *Let  $G$  be a bipartite graph with vertex bipartition  $\{X, Y\}$ . Then graph  $G$  has an  $X$ -saturating matching if and only if for each subset  $W$  of  $X$ ,  $|W| \leq |N(W)|$ .*

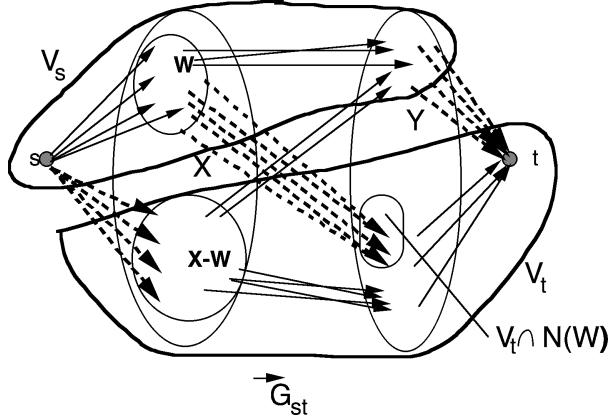
**Proof:** *Necessity ( $\Rightarrow$ )* Suppose that there exists an  $X$ -saturating matching  $M$  in graph  $G$ , and let  $W$  be any subset of  $X$ . Then every vertex  $w \in W$  is matched to a vertex  $y \in Y$  by an edge in matching  $M$ . Each such  $y$  is in the neighbor set  $N(W)$  of  $W$ , and, thus,  $|W| \leq |N(W)|$ .

*Sufficiency ( $\Leftarrow$ )* Suppose that Hall's Condition holds, that is,  $|W| \leq |N(W)|$ , for every subset  $W$  of  $X$ . Consider the  $s$ - $t$  network  $\vec{G}_{st}$  corresponding to the bipartite graph  $G$  (as in Proposition 13.4.1). An  $X$ -saturating matching in graph  $G$  would be a maximum matching in  $G$  and would correspond to a maximum flow in network  $\vec{G}_{st}$  with value  $|X|$ . Thus, by the Max-Flow Min-Cut Theorem (Theorem 13.2.4), it suffices to show that a minimum  $s$ - $t$  cut in network  $\vec{G}_{st}$  has capacity  $|X|$ . The  $s$ - $t$  cut  $\langle \{s\}, X \cup Y \cup \{t\} \rangle$  has capacity  $|X|$ , so it remains to show that every other  $s$ - $t$  cut  $K$  has capacity  $cap(K) \geq |X|$ .

Let  $\langle V_s, V_t \rangle$  be any  $s-t$  cut in network  $\vec{G}_{st}$ , and let  $W = V_s \cap X$ . Then the cut  $\langle V_s, V_t \rangle$  can be expressed as the disjoint union of three arc sets (depicted as dashed lines in Figure 13.4.7), that is,

$$\langle V_s, V_t \rangle = \langle \{s\}, V_t \cap X \rangle \cup \langle W, V_t \cap Y \rangle \cup \langle V_s \cap Y, \{t\} \rangle$$

where  $\langle A, B \rangle$  denotes the set of arcs directed from a vertex in  $A$  to a vertex in  $B$ .



**Figure 13.4.7**  $\langle V_s, V_t \rangle = \langle \{s\}, X - W \rangle \cup \langle W, V_t \cap Y \rangle \cup \langle V_s \cap Y, \{t\} \rangle$

The following chain of inequalities completes the proof.

$$\begin{aligned} \text{cap}\langle V_s, V_t \rangle &= |\langle \{s\}, X - W \rangle| + |\langle W, V_t \cap Y \rangle| + |\langle V_s \cap Y, \{t\} \rangle| \quad (\text{all capacities} = 1) \\ &= |X - W| + |\langle W, V_t \cap Y \rangle| + |V_s \cap Y| \quad (\text{by the construction of } \vec{G}_{st}) \\ &\geq |X - W| + |V_t \cap N(W)| + |V_s \cap Y| \quad (\text{by definition of } N(W)) \\ &= |X - W| + |N(W)| - |V_s \cap N(W)| + |V_s \cap Y| \quad (\text{see Figure 13.4.7}) \\ &\geq |X - W| + |N(W)| - |V_s \cap Y| + |V_s \cap Y| \quad (\text{since } N(W) \subseteq Y) \\ &= |X - W| + |N(W)| \\ &\geq |X - W| + |W| \quad (\text{by Hall's Condition}) \\ &= |X| \end{aligned}$$

◊

**Corollary 13.4.4 [Hall's Theorem for Transversals].** Let  $A$  be a nonempty finite set, and let  $\mathcal{F} = \{S_1, S_2, \dots, S_r\}$  be a family of nonempty subsets of set  $A$ . Then family  $\mathcal{F}$  has a transversal if and only if the union of any  $k$  of the subsets  $S_i$  contains at least  $k$  elements of set  $A$  ( $1 \leq k \leq r$ ).

**Proof:** This is Hall's Theorem for Bipartite Graphs, restated in terms of transversals.

◊

**Remark:** One of the earliest incarnations of Hall's Theorem appeared as a solution to the following problem, known as the **Marriage Problem**: Given a set of women, each of whom knows a subset of men, under what conditions can each of the women marry a man whom she knows? (See Exercises.)

## Two Graph Factorization Theorems

REVIEW FROM §9.4:

- A **factor** of a graph is a spanning subgraph.
- A **factorization** of a graph  $G$  is a set of factors whose edge-sets form a partition of the edge-set  $E_G$ .
- A  **$k$ -factor** of a graph  $G$  is a  $k$ -regular factor of  $G$ .
- A  **$k$ -factorization** of a graph is a factorization into  $k$ -factors.

**Theorem 13.4.5 [König's 1-Factorization Theorem].** [Kö16] Every  $r$ -regular bipartite graph  $G$  with  $r > 0$  is 1-factorable.

**Proof:** Suppose that  $X$  and  $Y$  are the two partite sets, and let  $W \subseteq X$ . The number of edges from  $W$  to  $Y$  is  $r|W|$ . Since each vertex of  $Y$  has degree  $r$ , at most  $r$  of these edges are incident on any one vertex of  $Y$ , from which it follows (by the generalized pigeonhole principle) that

$$N(W) \geq \left\lceil \frac{r|W|}{r} \right\rceil = |W|$$

By Hall's Theorem for Bipartite Graphs, the graph  $G$  has an  $X$ -saturating matching. Since  $|X| = |Y|$ , such a matching is a 1-factor in  $G$ . The graph obtained from  $G$  by deleting the edges of this 1-factor is an  $(r - 1)$ -regular bipartite graph, and the result follows by induction.  $\diamond$

REVIEW FROM §1.5 AND §4.5:

- An **eulerian trail** in a graph is a trail that contains every edge of that graph.
- An **eulerian tour** is a closed eulerian trail.
- An **eulerian graph** is a graph that has an eulerian tour.
- **Eulerian-Graph Characterization:** A connected graph  $G$  is eulerian if and only if the degree of every vertex in  $G$  is even.

**Theorem 13.4.6 [Petersen's 2-Factorization Theorem].** [Pe1891] Every regular graph  $G$  of even degree is 2-factorable.

**Proof:** We may assume that  $G$  is connected since a graph is factorable if and only if each of its components is factorable. Let  $G$  be a  $2r$ -regular graph with vertices  $v_1, \dots, v_n$ , and let  $C$  be a closed eulerian trail in  $G$ , whose existence is guaranteed by the Eulerian-Graph Characterization, cited above. We define a new bipartite graph  $H$  with partite sets

$$U = \{u_1, \dots, u_n\} \quad \text{and} \quad W = \{w_1, \dots, w_n\}$$

such that  $u_i$  and  $w_j$  are adjacent if vertex  $v_j$  immediately follows vertex  $v_i$  on the eulerian trail  $C$ . Graph  $H$  is  $r$ -regular, because the trail  $C$  enters and leaves each vertex of  $G$  exactly  $r$  times.

Since  $H$  is a regular bipartite graph, it follows from Theorem 13.4.5 that  $H$  has a 1-factor  $F$ . By the construction of  $H$ , the edge  $e \in F$  with endpoints  $u_i$  and  $w_j$  corresponds to edge  $e'$  of  $G$  with endpoints  $v_i$  and  $v_j$ . Moreover, for each subscript

$k = 1, \dots, n$ , the vertices  $u_k$  and  $w_k$  both occur exactly once in the 1-factor  $F$ , and hence, the edge set

$$F' = \{e' \mid e \in F\}$$

contains exactly two edges of  $G$  that are incident on the vertex  $v_k$ . It follows that  $F'$  is a 2-factor of  $G$ . The graph obtained from  $G$  by deleting the edges of this 2-factor is a  $(2r - 2)$ -regular graph, and the result follows by induction.  $\diamond$

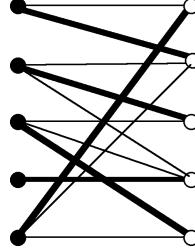
### Maximum Matchings and Minimum Vertex Covers

The theme of *max-min* pairs of optimization problems, seen earlier in this chapter and in Chapter 5, appears once again in the context of *vertex covers*.

**DEFINITION:** Let  $G$  be a graph, and let  $C$  be a subset of the vertices of  $G$ . Then set  $C$  is a **vertex cover** of graph  $G$  if every edge of  $G$  is incident on at least one vertex in  $C$ .

**DEFINITION:** A **minimum vertex cover** is a vertex cover with the least number of vertices.

**Example 13.4.7:** For the bipartite graph shown in Figure 13.4.8, a maximum matching (the bold edges) and minimum vertex cover (the solid vertices) both have cardinality 5.



**Figure 13.4.8** A maximum matching and minimum vertex cover.

**Proposition 13.4.7 [Weak Duality for Matchings].** Let  $M$  be a matching in a graph  $G$ , and let  $C$  be a vertex cover of  $G$ . Then  $|M| \leq |C|$ .  $\diamond$  (Exercises)

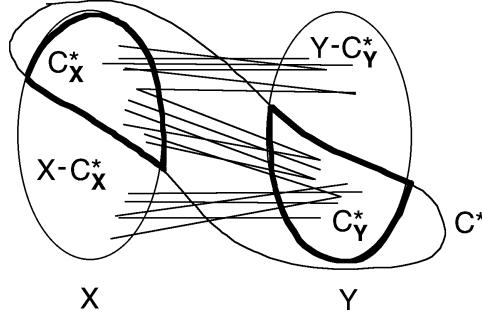
**Corollary 13.4.8 [Certificate of Optimality for Matchings].** Let  $M$  be a matching in a graph  $G$ , and let  $C$  be a vertex cover of  $G$  such that  $|M| = |C|$ . Then  $M$  is a maximum matching and  $C$  is a minimum vertex cover.  $\diamond$  (Exercises)

**Remark:** The converse of Corollary 13.4.8 does not hold in general (see Exercises); however, it does hold for bipartite graphs.

**NOTATION:** The neighbor set of a given subset  $W$  of vertices in a graph  $G$  is denoted  $N_G(W)$  (instead of the usual  $N(W)$ ) when there is more than one graph involved.

**Theorem 13.4.9 [König, 1931].** Let  $G$  be a bipartite graph. Then the number of edges in a maximum matching in  $G$  is equal to the number of vertices in a minimum vertex cover of  $G$ .

**Proof:** Let  $\{X, Y\}$  be the vertex bipartition of bipartite graph  $G$ , and let  $C^*$  be a minimum vertex cover of  $G$ . Then  $C^*$  is the disjoint union of its set of  $X$ -vertices,  $C_X^* = C^* \cap X$ , and its set of  $Y$ -vertices,  $C_Y^* = C^* \cap Y$ , as illustrated in Figure 13.4.9.



**Figure 13.4.9** Minimum vertex cover  $C^* = C_X^* \cup C_Y^*$ .

Consider the bipartite subgraph  $G_1$  of graph  $G$  induced on the vertex bipartition  $\{C_X^*, Y - C_Y^*\}$ . Let  $W$  be any subset of  $C_X^*$ . If  $|W| > |N_{G_1}(W)|$ , then there would exist  $w \in W$  such that  $N_{G_1}(W - \{w\}) = N_{G_1}(W)$ . But this would imply that  $(C_X^* - \{w\}) \cup C_Y^* = C^*$  is a vertex cover of graph  $G$ , contradicting the minimality of  $C^*$ . Thus, the bipartite graph  $G_1$  satisfies Hall's Condition, and by Hall's Theorem (Theorem 13.4.3),  $G_1$  has a  $C_X^*$ -saturating matching  $M_1^*$ , with  $|M_1^*| = |C_X^*|$ .

Next, let  $G_2$  be the bipartite subgraph induced on the vertex bipartition  $\{X - C_X^*, C_Y^*\}$ . Then a similar argument applied to graph  $G_2$  shows that it has a  $C_Y^*$ -saturating matching  $M_2^*$ , with  $|M_2^*| = |C_Y^*|$ . The edge set  $M = M_1^* \cup M_2^*$  is clearly a matching in graph  $G$ , and  $|M| = |C_X^*| + |C_Y^*| = |C^*|$ . Thus, by Corollary 12.4.6,  $M$  is a maximum matching.  $\diamond$

### 0-1 Matrices and the König-Egerváry Theorem

An interesting interpretation of this last theorem involves *0-1 matrices*, which are matrices each of whose entries is 0 or 1.

**Theorem 13.4.10 [König-Egerváry, 1931].** *Let  $A$  be a 0-1 matrix. Then the maximum number of 1's in matrix  $A$ , no two of which lie in the same row or column, is equal to the minimum number of rows and columns that together contain all the 1's in  $A$ .*

**Proof:** Let  $G$  be a bipartite graph with vertex bipartition  $\{X, Y\}$ , such that  $A$  is an adjacency matrix of graph  $G$ , where  $X$  is the set of vertices corresponding to the rows of matrix  $A$ , and  $Y$  is the vertex set corresponding to the columns. The result follows by applying Theorem 13.4.9 (see Exercises).  $\diamond$

**Application 13.4.4 The Bottleneck Problem:** Suppose that a manufacturing process consists of five operations that are performed simultaneously on five machines. The time in minutes that each operation takes when executed on each machine is given in the table below. Determine whether it is possible to assign the operations so that the process is completed within 4 minutes.

	M1	M2	M3	M4	M5
Op1	4	5	3	6	4
Op2	5	6	2	3	5
Op3	3	4	5	2	4
Op4	4	8	3	2	7
Op5	2	6	6	4	5

Let  $M$  be the  $5 \times 5$  matrix whose  $ij^{\text{th}}$  entry  $a_{ij}$  equals 1 if operation  $i$  takes no more than 4 minutes when performed on machine  $j$ , and equals 0 otherwise. Thus,

$$M = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

It is easy to check that this matrix contains five 1's no two of which are in the same row or column, which implies that it is possible to complete the process within 4 minutes.

### Summary of Equivalences Among Theorems in Chapter 13

The occurrence of various max-min pairs of problems and the similarities among many of the proofs in this chapter suggest strong connections among the theorems involved, some of which have already been established. In fact, any one of the following theorems can be used to prove the others.

- Menger's Theorem (Theorem 13.3.18)
- Max-Flow Min-Cut Theorem (Theorem 13.2.4)
- König's Theorem (Theorem 13.4.9)
- Hall's Theorem (Theorem 13.4.3)
- König-Egerváry Theorem (Theorem 13.4.10)

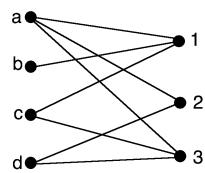
Below is an outline of some of these connections.

Max-Flow Min-Cut  $\implies$  Menger (Theorems 13.3.5, 13.3.9, 13.3.17, 13.3.18)  
 Max-Flow Min-Cut  $\implies$  Hall (Theorem 13.4.3)  
 Hall  $\implies$  König (Theorem 13.4.9)  
 König  $\implies$  König-Egerváry (Theorem 13.4.10)  
 König-Egerváry  $\implies$  Hall (Exercise 13.4.32)  
 Menger  $\implies$  Hall (Exercise 13.4.31)  
 Menger  $\implies$  Max-Flow Min-Cut (Exercise 13.3.25)

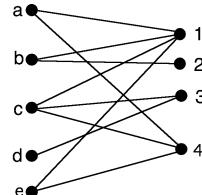
### EXERCISES for Section 13.4

For Exercises 13.4.1 through 13.4.4, use Maximum-Bipartite-Matching Algorithm 13.4.1 to find a maximum matching for the given bipartite graph.

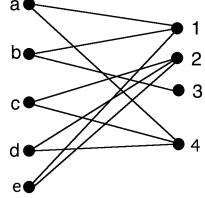
13.4.1



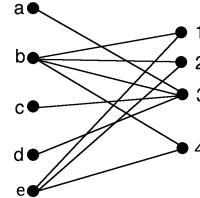
13.4.2<sup>s</sup>



13.4.3



13.4.4



13.4.5 Five men and five women are attending a dance. Katie will dance only with Gary or Harry, Barbara will dance only with Fred or Ignatz, Carol will dance only with Harry or Jack, Donna will dance only with Fred or Gary, and Emma will dance only with Gary or Ignatz. Is it possible for all ten people to dance the last dance so that each woman dances with someone she finds acceptable?

13.4.6<sup>s</sup> The Art History Department would like to offer six courses during the fall semester. There are seven professors in the department, each of whom is willing to teach certain courses, as shown in the table. Is there an assignment of professors to courses so that no professor teaches more than one course?

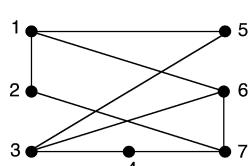
Course	Professor
Greek & Roman	Shargaa, Ward, Johnson, Pate
Renaissance	Maupin, Shargaa, Margeson
Baroque	Maupin, Shargaa
Impressionism	Maupin, Margeson
Early Modern	Vigorito, Johnson, Margeson
Contemporary	Shargaa, Margeson

For Exercises 13.4.7 through 13.4.10, determine whether Hall's Condition for the existence of a transversal is met by the given family of subsets. If the condition is met, then find a transversal; otherwise, show how the condition is violated.

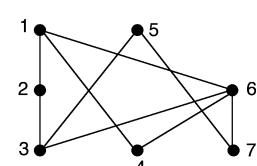
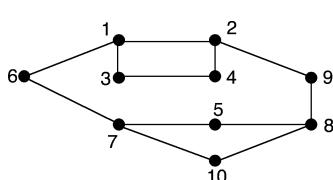
13.4.7  $\{1, 2, 4\}, \{2, 4\}, \{2, 3\}, \{1, 2, 3\}$ .13.4.8  $\{1, 2, 5\}, \{1, 5\}, \{1, 2\}, \{2, 5\}$ .13.4.9  $\{1, 2, 3\}, \{1, 2, 4\}, \{1, 3, 4\}, \{1, 2, 3, 4\}, \{2, 3, 4\}$ .13.4.10<sup>s</sup>  $\{1, 2\}, \{2, 3\}, \{5\}, \{1, 3\}, \{4, 5\}, \{4, 5\}$ .

For Exercises 13.4.11 through 13.4.14, find a maximum matching and a minimum vertex cover for the given graph.

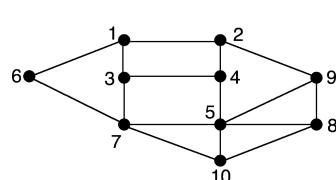
13.4.11



13.4.12

13.4.13<sup>s</sup>

13.4.14



**13.4.15** Let  $G$  be a bipartite graph and  $\vec{G}_{st}$  be its corresponding  $s-t$  network (referred to in Proposition 13.4.1). Prove that if  $f$  is an integral flow in network  $\vec{G}_{st}$ , then the set  $M$  of edges of graph  $G$  whose corresponding arcs in  $\vec{G}_{st}$  have unit flow is a matching in  $G$ .

**13.4.16** Let  $M$  be a matching in a bipartite graph  $G$ . For any edge  $e$  in  $G$ , let  $\vec{e}$  denote the corresponding arc in network  $\vec{G}_{st}$  (described in the subsection preceding Proposition 13.4.1). Consider the function  $f$  defined by

$$f(a) = \begin{cases} 1, & \text{if } a = \vec{e}, \text{ for some edge } e \in M \\ 1, & \text{if arc } a \text{ is adjacent to } \vec{e}, \text{ for some edge } e \in M \\ 0, & \text{otherwise} \end{cases}$$

for every arc  $a$  in network  $\vec{G}_{st}$ . Show that  $f$  is a feasible flow in  $\vec{G}_{st}$ .

**13.4.17<sup>s</sup>** Let  $G$  be a bipartite graph with vertex bipartition  $\{X, Y\}$ , and let  $\delta_X$  be the minimum degree of the vertices in  $X$ , and  $\Delta_Y$  be the maximum degree of the vertices in  $Y$ . Prove that if  $\delta_X \geq \Delta_Y$ , then there exists an  $X$ -saturating matching in graph  $G$ .

**13.4.18** Suppose that there are  $n$  workers and  $n$  jobs to be performed. Each worker is qualified to perform exactly  $k$  jobs,  $k \geq 1$ , and each job can be performed by exactly  $k$  workers. Prove that each job can be assigned to a different worker who is qualified for that job.

**13.4.19<sup>s</sup>** Let  $G$  be a bipartite graph with vertex bipartition  $\{X, Y\}$  such that  $|X| = |Y| = m$  and every vertex has degree  $k$ ,  $k \geq 1$ . Prove that there exists an  $X$ -saturating matching in graph  $G$ .

**13.4.20** Let  $\mathcal{F} = \{S_1, S_2, \dots, S_r\}$  be a family of subsets of the set  $\{1, 2, \dots, r\}$ , each of size  $m$ ,  $m \geq 1$ . Prove that there exists a transversal of  $\mathcal{F}$ .

**13.4.21** Let  $G$  be a graph, and let  $W$  be a subset of  $V_G$ . Prove that  $W$  is an *independent set* of graph  $G$  (§2.3) if and only if the set  $V_G - W$  is a vertex cover of  $G$ .

For Exercises 13.4.22 through 13.4.24, verify König-Egerváry Theorem 13.4.10 for the given 0-1 matrix.

**13.4.22** 
$$\begin{pmatrix} 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{pmatrix}$$

**13.4.23** 
$$\begin{pmatrix} 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

**13.4.24** 
$$\begin{pmatrix} 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{pmatrix}$$

**13.4.25<sup>s</sup>** Suppose that there are four jobs to be performed and five workers who are qualified for every job. The time in hours each worker needs to do each job is given in the table below. Is it possible to assign each job to a different worker so that each worker starts at the same time, and all four jobs are completed within four hours?

	Job1	Job2	Job3	Job4
W1	3	7	5	8
W2	6	3	2	3
W3	3	5	8	6
W4	5	8	6	4
W5	6	5	7	3

**13.4.26** Prove the weak duality result (Proposition 13.4.7) for matchings and vertex covers.

**13.4.27<sup>s</sup>** Prove the certificate of optimality (Corollary 13.4.8) for matchings and vertex covers.

**13.4.28** Show that Theorem 13.4.9 of König does not necessarily hold for a non-bipartite graph.

**13.4.29<sup>s</sup>** Express Hall's Theorem as a solution to the *Marriage Problem* (see the remark following Corollary 13.4.8).

**13.4.30** Fill in the details of the proof of König-Egerváry Theorem 13.4.10 for 0-1 matrices.

**13.4.31** Derive Hall's Theorem 13.4.3 from Menger's Theorem 13.4.18.

**13.4.32** Derive Hall's Theorem 13.4.3 from König-Egerváry Theorem 13.4.10.

## 13.5 SUPPLEMENTARY EXERCISES

**13.5.1** Let  $G$  be a bipartite graph with vertex bipartition  $\{X, Y\}$  such that  $|X| = |Y| = m$  and every vertex has degree  $k$ ,  $k \geq 1$ . Prove that there exists a partition  $\{M_1, M_2, \dots, M_k\}$  of  $E_G$  such that each edge set  $M_i$  is a matching in graph  $G$ . (Hint: see Exercise 13.4.19.)

**13.5.2** At a certain job fair, there are 10 prospective employers and 10 job seekers. Each employer would like to interview exactly three job seekers, and each job seeker would like to be interviewed by exactly three employers. By prior arrangement, each of the interviews has been mutually agreed upon. In other words, an employer wants to interview a job seeker if and only if that job seeker also wants to be interviewed by that employer. Prove that it is possible to arrange all 30 interviews over the course of three days, such that all 20 people interview exactly once each day.

**13.5.3** Let  $M$  be an  $n \times n$  matrix of 0's and 1's such that every row and every column contain exactly  $s$  1's,  $s \geq 1$ . Prove that there are  $n$  1's such that no two lie in the same row or column.

**13.5.4** Prove or disprove: A graph  $G$  is non-bipartite if and only if the number of edges in a maximum matching is unequal to the number of vertices in a minimum vertex cover.

**13.5.5** Let  $G$  be a simple  $n$ -vertex graph. Let  $i^*$  be the number of vertices in a largest independent set of  $G$ , and let  $c^*$  be the number of vertices in a minimum vertex cover of  $G$ . Prove that  $i^* + c^* = n$ . (Hint: see Exercise 13.4.21.)

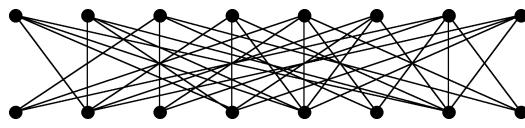
**DEF:** An  $m \times n$  **latin rectangle** is an  $m \times n$  matrix  $L = (m_{ij})$  whose entries are integers between 1 and  $n$  such that no two entries in any row or in any column are equal.

**DEF:** A **latin square** is a latin rectangle with an equal number of rows and columns.

### 13.5.6 Application 13.5.1 Extending Latin Rectangles to Latin Squares:

Use Hall's Theorem to prove that any  $m \times n$  latin rectangle  $L$  with  $m < n$  can be extended to a latin square by the addition of  $n - m$  new rows. (Hint: Let  $A = \{1, 2, \dots, n\}$  and  $\mathcal{F} = \{S_1, S_2, \dots, S_n\}$ , where  $S_i$  is the subset of  $A$  of elements that do not occur in the  $i$ th column of rectangle  $L$ . Then show that  $L$  can be extended to an  $(m+1) \times n$  latin rectangle by showing that family  $\mathcal{F}$  satisfies Hall's Condition.)

**13.5.7** Use Hall's Theorem to show there is no perfect matching in this graph. (Hint: think strategically; don't fight the  $2^8$  subsets.) Also, find a maximum matching.



**Figure 13.5.1**

**13.5.8** Determine whether the following collection of subsets of  $\{1, 2, 3, 4, 5, 6, 7, 8\}$  has a system of distinct representatives?  $B = \{5, 6, 7\}$ ,  $C = \{4, 5\}$ ,  $J = \{2, 3, 6, 7, 8\}$ ,  $K = \{1, 4\}$ ,  $L = \{1, 3, 4\}$ ,  $Q = \{2, 5, 6, 8\}$ ,  $T = \{1, 5\}$ ,  $U = \{3, 5\}$ . Hint: if your approach would require exponential time, think of a polynomial-time method.

## GLOSSARY

**$f$ -augmenting path**  $Q$  for a given flow  $f$  in a network  $N$ : an  $s-t$  quasi-path in  $N$  such that the flow on each forward arc can be increased, and the flow on each backward arc can be decreased.

**backward arc** on an  $s-t$  quasi-path  $Q = \langle s = v_0, e_1, v_1, \dots, v_{k-1}, e_k, v_k = t \rangle$ : an arc  $e_i$  that is directed from vertex  $v_i$  to vertex  $v_{i-1}$ .

**bipartite graph**: a graph for which there exists a vertex bipartition  $\{X, Y\}$  of  $V_G$  such that every edge of  $G$  has one endpoint in vertex set  $X$  and one endpoint in vertex set  $Y$ .

**capacitated network**: a connected digraph such that each arc  $e$  is assigned a non-negative weight  $cap(e)$ , called the *capacity* of arc  $e$ .

**capacity of an arc**: see *capacitated network*.

**capacity of an  $s-t$  cut**  $\langle V_s, V_t \rangle$ : the sum of the capacities of the arcs in cut  $\langle V_s, V_t \rangle$ . That is,  $cap(V_s, V_t) = \sum_{e \in (V_s, V_t)} cap(e)$ .

**conservation-of-flow** property for a flow  $f$  in a network  $N$ : one of two conditions that a feasible flow must satisfy;  $\sum_{e \in In(v)} f(e) = \sum_{e \in Out(v)} f(e)$  for every vertex  $v$  in network  $N$ , excluding source  $s$  and sink  $t$ .

**s-t cut**  $\langle V_s, V_t \rangle$  in an s-t network  $N$ : the set of all arcs from a vertex subset  $V_s$  to a vertex subset  $V_t$ , where  $V_s, V_t$  is a partition of  $V_N$  with source  $s \in V_s$  and sink  $t \in V_t$ .

**edge-connectivity**  $\kappa_e(G)$  of a graph  $G$ : the size of a minimum edge-cut in  $G$ .

—, **local**  $\kappa_e(s, t)$  between distinct vertices  $s$  and  $t$ : the minimum number of edges in an s-t separating edge set in  $G$ .

**eulerian graph**: a graph that has an eulerian tour.

**eulerian tour in a graph**  $G$ : a closed eulerian trail in  $G$ .

**eulerian trail in a graph**  $G$ : a trail that contains every edge of  $G$ .

**factor of a graph**  $G$ : a spanning subgraph of  $G$ .

—,  $k$ -: a  $k$ -regular factor of  $G$ .

**factorization of a graph**  $G$ : a set of factors whose edge-sets form a partition of the edge-set  $E_G$ .

—,  $k$ -: a factorization of  $G$  into  $k$ -factors.

**(feasible) flow**  $f$  in a capacitated network  $N$ : a function  $f : E_N \rightarrow R^+$  that assigns a nonnegative real number  $f(e)$  to each arc  $e$  such that:

1. (*capacity constraints*)  $f(e) \leq \text{cap}(e)$ , for every arc  $e$  in network  $N$ .

2. (*conservation constraints*)  $\sum_{e \in \text{In}(v)} f(e) = \sum_{e \in \text{Out}(v)} f(e)$ , for every vertex  $v$  in network  $N$ , other than source  $s$  and sink  $t$ .

—, **maximum**: a flow in network  $N$  having the maximum value.

**flow augmenting path**: see (*f*-)augmenting path.

**forward arc** on an s-t quasi-path  $Q = \langle s = v_0, e_1, v_1, \dots, v_{k-1}, e_k, v_k = t \rangle$ : an arc  $e_i$  that is directed from vertex  $v_{i-1}$  to vertex  $v_i$ .

**frontier arc**: an arc  $e$  directed from a tree vertex  $v$  to a non-tree vertex  $w$ .

**Hall's Condition** for a bipartite graph  $G$  with vertex bipartition  $\{X, Y\}$ : the condition  $|W| \leq |N(W)|$  for every subset  $W$  of  $X$ .

**in-set for a vertex**  $v$  in a digraph  $N$ , denoted  $\text{In}(v)$ : the set of all arcs  $e$  that are directed to  $v$ , i.e.,  $\text{In}(v) = \{e \in E_N \mid \text{head}(e) = v\}$ .

**independent set** of vertices of  $G$ : a subset of  $V_G$  such that no two vertices in  $W$  are adjacent in graph  $G$ .

**internally disjoint** (directed) paths in a (di)graph: (directed) paths that have no internal vertices in common.

**$m \times n$  latin rectangle**: an  $m \times n$  matrix  $L = (m_{ij})$  whose entries are integers between 1 and  $n$  such that no two entries in any row or in any column are equal.

**latin square**: a latin rectangle with an equal number of rows and columns.

**local edge-connectivity**  $\kappa_e(s, t)$  between vertices  $s$  and  $t$  in a graph  $G$ : the minimum number of edges in an s-t separating edge set in  $G$ .

**local vertex-connectivity**  $\kappa_v(s, t)$  between non-adjacent vertices  $s$  and  $t$  in a graph  $G$ : the minimum number of vertices in an s-t separating vertex set in  $G$ .

**matched with a vertex**  $y$  by a matching  $M$ : said of a vertex  $x$  that is joined to  $y$  by an edge in  $M$ .

**matching** in a graph  $G$ : a set of edges of  $G$  such that no two of them have an endpoint in common.

—, **maximum**: a matching with the greatest number of edges.

**maximum flow**  $f^*$  in a network  $N$ : a flow in network  $N$  having the maximum value.

**maximum matching** in a graph: see *matching, maximum*.

**minimum cut** of a network  $N$ : a cut with the minimum capacity.

**minimum vertex cover**: a vertex cover with the least number of vertices.

**network**: shortened name used in this chapter for *single source – single sink, capacitated* network.

**$s$ - $t$  network**: a single source – single sink network with source  $s$  and sink  $t$ .

**0-1 network**: a network such that each arc has capacity 0 or 1.

**out-set for a vertex  $v$**  in a digraph  $N$ , denoted  $Out(v)$ : the set of all arcs  $e$  that are directed from  $v$ , i.e.,  $Out(v) = \{e \in E_N | tail(e) = v\}$ .

**partition-cut**  $\langle V_1, V_2 \rangle$  in a graph  $G$ : the set of all edges of  $G$  having one endpoint in vertex subset  $V_1$  and the other endpoint in vertex subset  $V_2$ , where  $\{V_1, V_2\}$  form a partition of  $V_G$ .

**$s$ - $t$  quasi-path** in a network  $N$ : an alternating sequence  $\langle s = v_0, e_1, v_1, \dots, v_{k-1}, e_k, v_k = t \rangle$  of vertices and arcs that forms an  $s$ - $t$  path in the underlying graph of  $N$ .

**$X$ -saturating matching**  $M$  in a bipartite graph  $G$  with vertex bipartition  $\{X, Y\}$ : a matching  $M$  such that each vertex in set  $X$  is an endpoint of an edge in  $M$ .

**$s$ - $t$  separating arc set** in a digraph  $D$ : a set of arcs whose removal destroys all directed  $s$ - $t$  paths in  $D$ ; an arc subset of  $E_D$  that contains at least one arc of every directed  $s$ - $t$  path in digraph  $D$ .

**$s$ - $t$  separating edge set** in a graph  $G$ : a set of edges whose removal destroys all  $s$ - $t$  paths in  $G$ ; an edge subset of  $E_G$  that contains at least one edge of every  $s$ - $t$  path in  $G$ .

**$s$ - $t$  separating vertex set** in a graph  $G$  (or digraph  $D$ ): a set of vertices whose removal destroys all  $s$ - $t$  paths in  $G$  (or all directed  $s$ - $t$  paths in  $D$ ); a set of vertices that contains at least one internal vertex of every (directed)  $s$ - $t$  path.

**single source – single sink network**: a connected digraph that has a distinguished vertex called the **source** with nonzero outdegree, and a distinguished vertex called the **sink** with nonzero indegree.

**sink**: see *single source – single sink network*.

**slack  $\Delta_e$  for arc  $e$**  on a given  $f$ -augmenting path  $Q$ : the quantity  $cap(e) - f(e)$  if  $e$  is a forward arc, and the quantity  $f(e)$  if  $e$  is a backward arc.

**source**: see *single source – single sink network*.

**system of distinct representatives**: synonym for *transversal*.

**transversal** of a family  $\mathcal{F} = \{S_1, S_2, \dots, S_r\}$  of (not necessarily distinct) nonempty subsets of a nonempty finite set  $A$ : a sequence  $T = \langle a_1, a_2, \dots, a_r \rangle$  of  $r$  distinct elements of set  $A$ , such that  $a_i \in S_i$ , for each  $i = 1, \dots, r$ .

**value of flow  $f$**  in a network  $N$ , denoted  $val(f)$ : the net flow leaving the source  $s$ ;  

$$val(f) = \sum_{e \in Out(s)} f(e) - \sum_{e \in In(s)} f(e)$$

**vertex cover** of a graph  $G$ : a subset  $C$  of the vertices of  $G$  such that every edge of  $G$  is incident on at least one vertex in  $C$ .

**—, minimum**: a vertex cover with the least number of vertices.

# Chapter 14

---

## GRAPHICAL ENUMERATION

- 14.1 Automorphisms of Simple Graphs**
  - 14.2 Graph Colorings and Symmetry**
  - 14.3 Burnside's Lemma**
  - 14.4 Cycle-Index Polynomial of a Permutation Group**
  - 14.5 More Counting, Including Simple Graphs**
  - 14.6 Pólya-Burnside Enumeration**
- 

### INTRODUCTION

Enumerating the different graphs with some given property is a problem of frequent interest. For instance, Arthur Cayley wanted to determine for the saturated hydrocarbon formula  $C_nH_{2n+2}$  the number of different molecular structures that could realize that formula, for each value of  $n$ , and he modeled such hydrocarbons with trees. Knowing the number of graphs with a particular property may enable a computer scientist to estimate the length of a database calculation. This chapter develops the basic mathematical concepts and tools for counting graphs. The main themes are using symmetries to form equivalence classes and using algebra to count the equivalence classes.

Although we restrict our attention mainly to simple graphs, the methods studied in this chapter are applicable to general graphs.

## 14.1 AUTOMORPHISMS OF SIMPLE GRAPHS

The concept of graph automorphism was introduced in §2.2. For convenience, the definitions from Chapter 2 that are needed for this chapter are reviewed below.

REVIEW FROM §2.1 AND §2.2:

- A vertex bijection  $f_V : V_G \rightarrow V_H$  between the vertex-sets of simple graphs  $G$  and  $H$  is **structure-preserving** if it preserves adjacency and non-adjacency, that is, if for every pair of vertices in  $G$ ,
- $$u \text{ and } v \text{ are adjacent in } G \iff f_V(u) \text{ and } f_V(v) \text{ are adjacent in } H.$$
- Every structure-preserving vertex bijection  $f_V : V_G \rightarrow V_H$  of simple graphs induces an edge bijection  $f_E : E_G \rightarrow E_H$ , given by the rule  $f_E(uv) = f_V(u)f_V(v)$ .
  - A **graph isomorphism**  $f : G \rightarrow H$  of simple graphs is completely specified by a structure-preserving vertex bijection  $f_V : V_G \rightarrow V_H$ . Formally, it consists of the vertex bijection  $f_V$  paired with the induced edge bijection  $f_E$ .
  - An **automorphism**  $\pi$  of a graph  $G$  is an isomorphism from  $G$  to itself. Thus, the vertex bijection  $\pi_V$  is a permutation on  $V_G$ , and the edge bijection  $\pi_E$  is a permutation on  $E_G$ .
  - The action of the automorphism group  $\mathcal{A}ut(G)$  on a graph  $G$  partitions  $V_G$  into **vertex-orbits**. That is, the vertices  $u$  and  $v$  are in the same orbit if there exists an automorphism  $\pi$  such that  $\pi(u) = v$ . Similarly,  $\mathcal{A}ut(G)$  partitions  $E_G$  into **edge-orbits**.
  - A graph  $G$  is **vertex-transitive** if all the vertices are in a single orbit. Similarly,  $G$  is **edge-transitive** if all the edges are in a single orbit.

NOTATION: The set of automorphisms of a graph  $G$  is denoted  $\mathcal{A}ut(G)$ , and the corresponding sets of vertex-permutations and of edge-permutations are denoted  $\mathcal{A}ut_V(G)$  and  $\mathcal{A}ut_E(G)$ , respectively.

### The Sets $\mathcal{A}ut_V(G)$ and $\mathcal{A}ut_E(G)$ are Permutation Groups

DEFINITION: A collection  $P$  of permutations on the same set of objects is **closed under composition** if for every pair  $\pi_1, \pi_2 \in P$ , the composition  $\pi_1\pi_2$  is in  $P$ .

DEFINITION: Let  $P$  be a nonempty collection of permutations of the finite set of objects  $Y$  such that  $P$  is closed under composition. Then the set  $P$  together with set  $Y$  is called a **permutation group** and is denoted  $[P : Y]$ .

TERMINOLOGY: The permutation group is said to **act on the set  $Y$** .

DEFINITION: Let  $[P : Y]$  be a permutation group. The **orbit** of an object  $y \in Y$  is the set of all objects  $z$  such that there exists a permutation  $\pi \in P$  such that  $\pi(y) = z$ .

DEFINITION: The **full symmetric group**  $\Sigma_Y$  on a set  $Y$  is the collection of *all* permutations on  $Y$ .

**Theorem 14.1.1.** *The set  $\text{Aut}(G)$  of all automorphisms of a simple graph  $G$  acts as a permutation group on  $V_G$  and on  $E_G$ .*

**Proof:** It is straightforward to verify that the composition of two structure-preserving vertex-permutations is a structure-preserving vertex-permutation (see Exercises). Thus,  $\text{Aut}_V(G)$  and  $\text{Aut}_E(G)$  are permutation groups.  $\diamond$

**Remark:** Throughout this chapter, permutations are represented in *disjoint-cycle form* (introduced in §2.2). Permutation groups and abstract groups are discussed in Appendix A.4.

**Example 14.1.1:** The graph  $K_{1,3}$ , shown below, has six automorphisms, and the vertex- and edge-permutation groups corresponding to  $\text{Aut}(K_{1,3})$  are given by

$$\begin{aligned}\text{Aut}_V(K_{1,3}) = & \{(u)(v)(w)(x), (x)(u\ v\ w), (x)(u\ w\ v), (x)(u)(v\ w), \\ & (x)(v)(u\ w), (x)(w)(u\ v)\}\end{aligned}$$

$$\text{Aut}_E(K_{1,3}) = \{(a)(b)(c), (a\ b\ c), (a\ c\ b), (a)(b\ c), (b)(a\ c), (c)(a\ b)\}$$

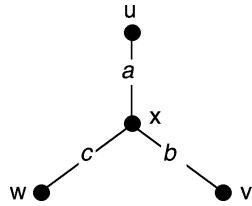


Figure 14.1.1 The graph  $K_{1,3}$ .

The vertex-orbits are  $\{u, v, w\}$  and  $\{x\}$ . The only edge-orbit is  $\{a, b, c\}$ .

**REVIEW FROM §2.2:** Although it is sometimes possible to design a plane drawing of a graph whose geometric symmetry completely captures all the combinatorial symmetry of the graph, we saw in §2.2 some examples where some combinatorial symmetries are not geometrically represented.

### Automorphism Groups of Some Other Simple Graphs

The methods used above to determine  $\text{Aut}(K_{1,3})$  are now applied to calculating the automorphism groups of some other graphs.

**Example 14.1.2:**  $\text{Aut}(K_n)$  Each of the  $n!$  permutations on the vertex-set of  $K_n$  is structure-preserving, because every pair of vertices is joined by an edge. Thus, every vertex-permutation specifies a different automorphism of  $K_n$ .

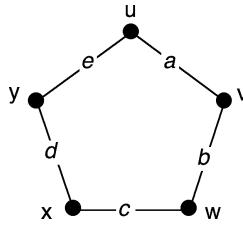


Figure 14.1.2 Automorphism action on an edge and its endpoints.

The complete graph  $K_4$  can be represented as the 1-skeleton of a regular tetrahedron in Euclidean 3-space. All 24 automorphisms of  $K_4$  can be realized by rotations and

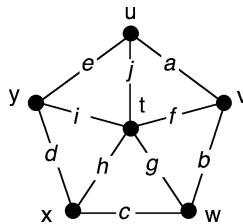
reflections of a regular tetrahedron. To generalize this geometric viewpoint to larger values of  $n$ , represent the complete graph  $K_n$  as the 1-skeleton of a regular  $n$ -simplex in Euclidean  $n$ -space. Then all  $n!$  automorphisms can be realized by rotations and reflections of a regular  $n$ -simplex.

**Example 14.1.3:**  $\mathcal{A}ut(C_n)$  The cycle graph  $C_n$  can be represented as the 1-skeleton of a regular  $n$ -gon in the plane, as illustrated in Figure 14.1.3 for  $n = 5$ . All of its automorphisms can be realized as rotations and reflections of a regular  $n$ -gon. For instance, rotation  $72^\circ$  clockwise corresponds to the automorphism of  $C_5$  whose vertex-permutation is  $(u \ v \ w \ x \ y)$  and whose edge-permutation is  $(a \ b \ c \ d \ e)$ . Also, reflection through the vertical axis of symmetry corresponds to the automorphism whose vertex-permutation is  $(u)(v \ y)(w \ x)$  and whose edge-permutation is  $(a \ e)(b \ d)(c)$ . In general, a regular  $n$ -gon has  $n$  rotations and  $n$  reflections, which gives the graph  $C_n$  a total of  $2n$  automorphisms.



**Figure 14.1.3** The cycle graph  $C_5$  has 5 rotations and 5 reflections.

**Example 14.1.4:**  $\mathcal{A}ut(W_n)$  A geometric representation of the  $n$ -spoked wheel graph  $W_n$  can be constructed by placing a vertex at the center of a regular  $n$ -gon in the plane, and joining it to every vertex of the  $n$ -gon, as illustrated for  $n = 5$  in Figure 14.1.4. All of its automorphisms can be realized as rotations and reflections of the  $n$ -gon. For instance, rotation  $72^\circ$  clockwise corresponds to the automorphism of  $W_5$  whose vertex-permutation is  $(t)(u \ v \ w \ x \ y)$  and whose edge-permutation is  $(a \ b \ c \ d \ e)(f \ g \ h \ i \ j)$ . Also, reflection through the vertical axis of symmetry corresponds to the automorphism whose vertex-permutation is  $(t)(u)(v \ y)(w \ x)$  and whose edge-permutation is  $(a \ e)(b \ d)(c \ f \ i)(g \ h \ j)$ . Thus, the wheel graph  $W_n$  has  $n$  rotations and  $n$  reflections, for a total of  $2n$  automorphisms, except when  $n = 3$ . In the special case  $W_3 \cong K_4$ , in which the hub vertex has the same degree as the rim vertices, there are 24 automorphisms.



**Figure 14.1.4** The wheel graph  $W_5$  has 5 rotations and 5 reflections.

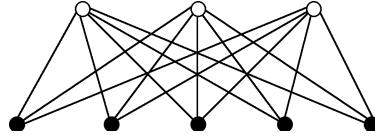
**Example 14.1.5:**  $\mathcal{A}ut(K_{m,n})$  Let  $K_{m,n}$  have vertex bipartition  $R \cup S$ , with  $m = |R|$  and  $n = |S|$ , and let  $\tau \in \mathcal{A}ut(K_{m,n})$ . Then, since  $\tau$  preserves non-adjacency, two vertices on one side of the bipartition must map to two vertices on that same side or to two vertices on the other side. It follows that either  $\tau(R) = R$  and  $\tau(S) = S$ , or else the bijection  $\tau$  swaps  $R$  and  $S$  (i.e.,  $\tau(R) = S$  and  $\tau(S) = R$ ).

If  $m \neq n$ , then  $\tau(R) = R$  and  $\tau(S) = S$ . Thus,  $\tau$  permutes the elements of set  $R$  and permutes the elements of set  $S$ . That is,  $\tau = \rho \oplus \sigma$ , where  $\rho \in \Sigma_R$  and  $\sigma \in \Sigma_S$ , and

$$(\rho \oplus \sigma)(u) = \begin{cases} \rho(u) & \text{if } u \in R \\ \sigma(u) & \text{if } u \in S \end{cases}$$

Conversely, each pair of permutations  $\rho \in \Sigma_R$  and  $\sigma \in \Sigma_S$  forms the structure-preserving vertex-permutation  $\rho \oplus \sigma$ . It follows that if  $m \neq n$ , then  $\mathcal{A}ut_V(K_{m,n}) = \Sigma_R \times \Sigma_S$ , and, hence,  $|\mathcal{A}ut_V(K_{m,n})| = m!n!$

If  $m = n$ , then  $\tau \in \mathcal{A}ut(K_{n,n})$  is either a permutation in the set  $\Sigma_R \times \Sigma_S$  or is a vertex bijection that swaps  $R$  and  $S$ . Moreover, each of the  $(n!)^2$  vertex bijections that swap  $R$  and  $S$  is structure-preserving, since every vertex in  $R$  is adjacent to every vertex in  $S$ . Therefore,  $|\mathcal{A}ut(K_{n,n})| = 2(n!)^2$ .

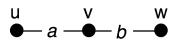


**Figure 14.1.5** The bipartite graph  $K_{3,5}$  has  $3!5! = 720$  automorphisms.

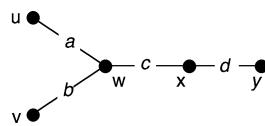
### EXERCISES for Section 14.1

In Exercises 14.1.1 through 14.1.8, write the vertex-permutation and the edge-permutation for every automorphism of the graph shown. Also list the vertex-orbits and the edge-orbits.

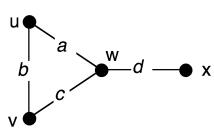
14.1.1



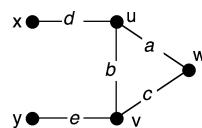
14.1.2



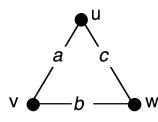
14.1.3<sup>s</sup>



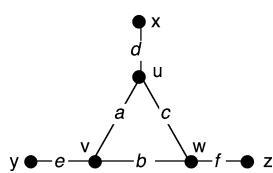
14.1.4



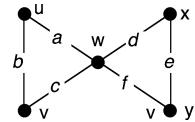
14.1.5<sup>s</sup>



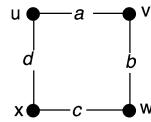
14.1.6



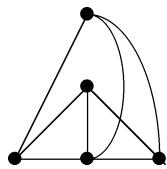
14.1.7



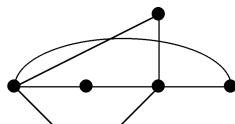
14.1.8



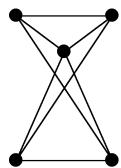
In Exercises 14.1.9 through 14.1.12, redraw the graph so that as many automorphisms as possible are represented by symmetries of the drawing.

14.1.9<sup>s</sup>

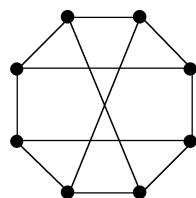
14.1.10



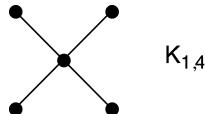
14.1.11



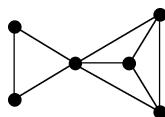
14.1.12



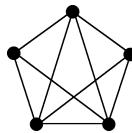
14.1.13<sup>s</sup> How many automorphisms are in the group  $\text{Aut}(K_{1,4})$ ?



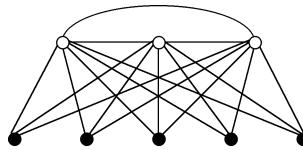
14.1.14 Suppose that  $K_3$  and  $K_4$  are amalgamated at a vertex. How many automorphisms does the resulting graph have? How many vertex-orbits and edge-orbits?



14.1.15<sup>s</sup> Suppose that an edge is deleted from  $K_5$ . How many automorphisms does the resulting graph have? How many vertex-orbits and edge-orbits?



14.1.16 Suppose that the complete graph  $K_3$  is joined to the edgeless graph on 5 vertices. How many automorphisms does the resulting graph have? How many vertex-orbits and edge-orbits?



14.1.17<sup>s</sup> How many automorphisms are in the group  $\text{Aut}(K_{2,3,4})$ , where  $K_{2,3,4}$  is the *complete tripartite graph* with tripartition consisting of a 2-vertex, 3-vertex, and 4-vertex subset? How many vertex-orbits and edge-orbits?

14.1.18 How many automorphisms are in the group  $\text{Aut}(CL_n)$ , where  $CL_n$  is the  $n$ -vertex circular ladder graph (§1.2)? How many vertex-orbits and edge-orbits?

## 14.2 GRAPH COLORINGS AND SYMMETRY

The symmetry of a graph  $G$  has a substantial effect on how we count its vertex-colorings and edge-colorings. Indeed, what we actually count is the number of equivalence classes on its vertex- and edge-colorings, induced by the automorphism group  $\text{Aut}(G)$ . This section describes these equivalence classes and lays the groundwork for further enumerative results in §14.3 through §14.5.

### Coloring a Set Subject to the Action of a Permutation Group

Counting equivalence classes of graph colorings lies within the general context of counting equivalent colorings of a set acted upon by a permutation group.

**DEFINITION:** A  *$k$ -coloring* of a set  $Y$  is a mapping  $f$  from  $Y$  onto the set  $\{1, 2, \dots, k\}$ , in which the value  $f(y)$  is called the *color* of  $y$ . Any  $k$ -coloring of  $Y$  is also called a *coloring*.

**DEFINITION:** A  *$(\leq k)$ -coloring* of a set  $Y$  is a coloring that uses  $k$  or fewer colors, formally a mapping  $f$  from  $Y$  onto any set  $\{1, 2, \dots, t\}$  with  $t \leq k$ .

**NOTATION:** The set of all  $(\leq k)$ -colorings of the elements of a set  $Y$  is denoted  $\text{Col}_k(Y)$ .

**Proposition 14.2.1.** Let  $Y$  be a set. Then  $|\text{Col}_k(Y)| = k^{|Y|}$ .

**Proof:** This is a direct application of the Rule of Product, a familiar counting principle in elementary discrete mathematics (see Appendix A.3).  $\diamond$

**DEFINITION:** Let  $\mathcal{P} = [P : Y]$  be a permutation group acting on a set  $Y$ , and let  $f$  and  $g$  be  $(\leq k)$ -colorings of the objects in  $Y$ . Then the coloring  $f$  is  $\mathcal{P}$ -**equivalent** to the coloring  $g$  if there is a permutation  $\pi \in P$  such that  $g = f\pi$ , that is, if for every object  $y \in Y$ , the color  $g(y)$  is the same as the color  $f(\pi(y))$ .

**TERMINOLOGY:** The  $\mathcal{P}$ -equivalence classes are also called *coloring classes*, or  *$\mathcal{P}$ -orbits*. That is, any two  $\mathcal{P}$ -equivalent colorings are in the same  $\mathcal{P}$ -orbit.

**NOTATION:** The set of  $\mathcal{P}$ -orbits of  $\text{Col}_k(Y)$  is denoted  $\{\text{Col}_k(Y)\}_{\mathcal{P}}$ . Thus,  $|\{\text{Col}_k(Y)\}_{\mathcal{P}}|$  equals the number of non-equivalent  $(\leq k)$ -colorings.

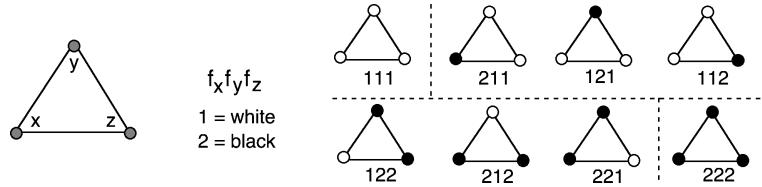
**NOTATION:** The identity of any group is often denoted  $\epsilon$ .

**Example 14.2.1:** Let  $Y = \{x, y, z\}$  be the vertex-set  $V_{C_3}$  of the 3-cycle graph  $C_3$ , shown on the left in Figure 14.2.1, and let  $P = \{\epsilon, (x\ y\ z), (x\ z\ y)\}$  be the subgroup<sup>†</sup> of vertex-permutations corresponding, respectively, to the  $0^\circ$ ,  $120^\circ$ , and  $240^\circ$  clockwise rotations of  $C_3$ . The set  $\text{Col}_2(V_{C_3})$ , which consists of eight  $(\leq 2)$ -colorings, is shown on the right in Figure 14.2.1. Each  $(\leq 2)$ -coloring  $f : V_{C_3} \rightarrow \{1, 2\}$  is represented graphically using white (color 1) and black (color 2) vertices, and also as a string  $f_x f_y f_z$  of length 3, where  $f_x$ ,  $f_y$ , and  $f_z$  are the colors assigned to vertices  $x$ ,  $y$ , and  $z$ , respectively.

These eight vertex-colorings are partitioned into four  $\mathcal{P}$ -orbits of  $\text{Col}_2(V_{C_3})$ , given by  $[\text{Col}_2(V_{C_3})]_{\mathcal{P}} = \{\{111\}, \{211, 121, 112\}, \{122, 212, 221\}, \{222\}\}$ .

---

<sup>†</sup> A subgroup is a subset of a group that is itself a group under the same operation.



**Figure 14.2.1** The four coloring classes of  $V_{C_3}$  under rotational equivalence.

**Remark:** Intuitively, the reason why the colorings in one class, e.g.  $\{211, 121, 112\}$ , are equivalent is because rotating a drawing of the graph superimposes one of these colorings automorphically onto another with correct color matching. This intuitive notion is what is formally represented by the condition  $g = f\pi$ .

### Induced Permutation Actions

In Example 14.2.1, we saw how the permutations acting on the three vertices of  $C_3$  induced another action on the set of eight vertex- $(\leq 2)$ -colorings. More generally, a permutation group acting on a set of objects also acts on the set of colorings of those objects, as we now show.

**Proposition 14.2.2.** Let  $\mathcal{P} = [P : Y]$  be a permutation group acting on a set  $Y$ . Let  $f \in Col_k(Y)$  be a  $(\leq k)$ -coloring of  $Y$ , and let  $\pi \in P$  be a permutation in  $P$ . Then the composition  $f\pi$  of permutation  $\pi$  followed by coloring  $f$  is a coloring in  $Col_k(Y)$ .

**Proof:** The composition  $f\pi$  is a coloring of  $Y$  because it assigns to each object  $y \in Y$  whatever color  $f$  assigns to the object  $\pi(y)$ .  $\diamond$

**Corollary 14.2.3.** Let  $\mathcal{P} = [P : Y]$  be a permutation group acting on a set  $Y$ , and let  $\pi \in P$ . Then the mapping  $\pi_{YC} : Col_k(Y) \rightarrow Col_k(Y)$  defined by

$$\pi_{YC}(f) = f\pi, \quad \text{for every coloring } f \in Col_k(Y)$$

is a permutation on the set  $Col_k(Y)$ .

**Proof:** The mapping  $\pi_{YC}$  is a bijection, because it has an inverse, namely, the rule  $g \mapsto g\pi^{-1}$ .  $\diamond$

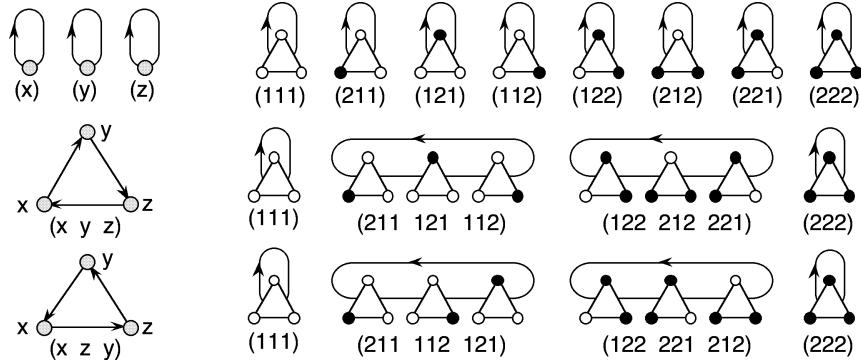
**DEFINITION:** The mapping  $\pi_{YC} : Col_k(Y) \rightarrow Col_k(Y)$  defined (in Corollary 14.2.3) by the rule  $f \mapsto f\pi$  is called the **induced permutation action** of  $\pi$  on  $Col_k(Y)$ .

**NOTATION:** To distinguish between the action of a permutation  $\pi$  on a set  $Y$  and its induced action on the set  $Col_k(Y)$  of colorings of  $Y$ , we let  $\pi_Y$  denote its action on  $Y$  and  $\pi_{YC}$  its action on  $Col_k(Y)$ . When there is no risk of confusion, the subscripts  $Y$  and  $YC$  may both be omitted.

**DEFINITION:** Let  $\mathcal{P} = [P : Y]$  be a permutation group acting on a set  $Y$ . The collection  $\mathcal{P}_C = [P : Col_k(Y)]$  of induced permutations on  $Col_k(Y)$  is called the **induced permutation group**.

**NOTATION:** When it is necessary to distinguish between the group that acts on the set  $Y$  and the group that acts on the set  $Col_k(Y)$ , the respective notations  $P_Y$  and  $P_{YC}$  are used.

**Example 14.2.1, continued:** Figure 14.2.2 below depicts the disjoint-cycle form of each permutation  $\pi_Y$  and also of the corresponding induced permutation  $\pi_{Y_C}$  acting on the set  $Col_2(Y) = \{111, 112, 121, 122, 211, 212, 221, 222\}$ .



**Figure 14.2.2** Correspondence of permutations and induced permutations.

Notice that two colorings are  $\mathcal{P}$ -equivalent (i.e., in the same  $\mathcal{P}$ -orbit) if the two colorings appear in a cycle of at least one induced permutation.

### Equivalent Colorings of a Graph $G$ under $\text{Aut}(G)$

If the symmetries of a graph are ignored, then counting vertex- or edge-colorings is trivial; using  $k$  or fewer colors, there are  $k^{|V_G|}$  vertex-colorings and  $k^{|E_G|}$  edge-colorings (by Proposition 14.2.1). However, taking symmetry into account, by counting orbits of colorings, is more complicated.

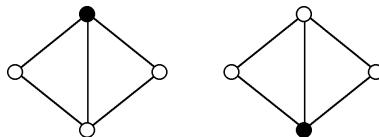
**TERMINOLOGY:**  $Col_k(V_G)$  is sometimes called the **full set of vertex- $(\leq k)$ -colorings** of a graph  $G$ , and  $Col_k(E_G)$  the **full set of edge- $(\leq k)$ -colorings**.

**DEFINITION:** Let  $G$  be a graph with automorphism group  $\text{Aut}(G)$ . **Equivalent vertex-colorings** are vertex-colorings that are  $\text{Aut}_V(G)$ -equivalent, and **equivalent edge-colorings** are edge-colorings that are  $\text{Aut}_E(G)$ -equivalent.

Thus, equivalent vertex-colorings are in the same  $\text{Aut}_V(G)$ -orbit, and equivalent edge-colorings are in the same  $\text{Aut}_E(G)$ -orbit.

**NOTATION:** The set of all  $\text{Aut}_V(G)$ -orbits (coloring classes) of  $Col_k(V_G)$  is denoted  $\{Col_k(V_G)\}_{\text{Aut}_V(G)}$ . Similarly,  $\{Col_k(E_G)\}_{\text{Aut}_E(G)}$  denotes the set of all  $\text{Aut}_E(G)$ -orbits of  $Col_k(E_G)$ .

**Example 14.2.2:** The two vertex-colorings in Figure 14.2.3 are equivalent, because a  $180^\circ$ -rotation or a reflection of one graph drawing through its horizontal axis corresponds to a graph automorphism.



**Figure 14.2.3** Two equivalent vertex-colorings of the graph  $K_4 - K_2$ .

**Example 14.2.3:** The two edge-colorings in Figure 14.2.4 are equivalent. Reflection of the graph through its vertical or horizontal axis corresponds to an automorphism that maps one coloring onto the other.

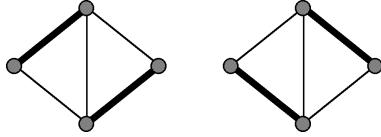


Figure 14.2.4 Two equivalent edge-colorings of the graph  $K_4 - K_2$ .

### Counting Vertex- and Edge-Coloring Orbits One by One

For a small graph and a small number of colors, it is possible to count the orbits of vertex- and edge-colorings by drawing a list of representatives of those classes. Using graph automorphism invariants like vertex degree simplifies this kind of counting.

**Example 14.2.4:** An automorphism on the graph  $K_4 - K_2$  must either fix the two 2-valent vertices or swap them, and it must either fix the two 3-valent vertices or swap them. Thus,  $\text{Aut}(K_4 - K_2)$  has the following representations as a group of vertex-permutations and as a group of edge-permutations.<sup>†</sup>

Symmetry	$\pi \in \text{Aut}_V(G)$	$\pi \in \text{Aut}_E(G)$
identity	$(u)(v)(w)(x)$	$(a)(b)(c)(d)(e)$
refl. thru vert. axis	$(u)(w)(v\ x)$	$(e)(a\ b)(c\ d)$
refl. thru horiz. axis	$(v)(x)(u\ w)$	$(e)(a\ c)(b\ d)$
180° rotation	$(u\ w)(v\ x)$	$(e)(a\ d)(b\ c)$

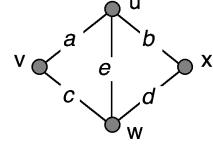


Figure 14.2.5 shows how the (full) set  $\text{Col}_2(V_{K_4 - K_2})$  of 16 vertex-( $\leq 2$ )-colorings is partitioned into nine  $\text{Aut}_V(K_4 - K_2)$ -orbits.

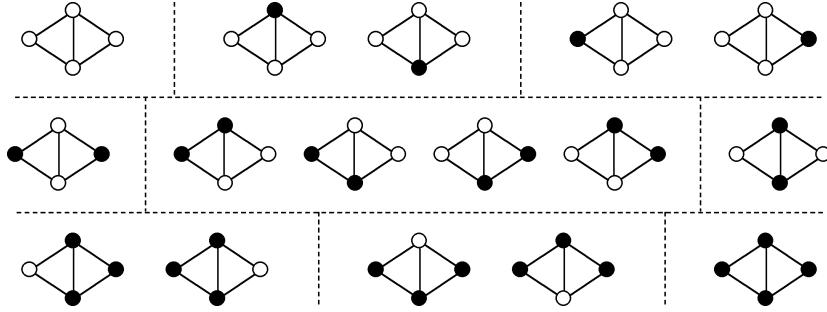
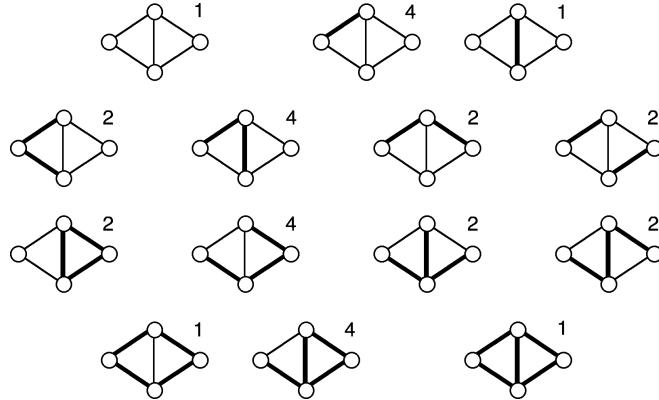


Figure 14.2.5 The  $\text{Aut}_V(K_4 - K_2)$ -orbits of  $\text{Col}_2(V_{K_4 - K_2})$ .

**Example 14.2.4, continued:** Since the graph  $K_4 - K_2$  has five edges, there are, ignoring equivalences,  $2^5 = 32$  edge-colorings that use two or fewer colors. Figure 14.2.6 below shows a representative of each of the 14  $\text{Aut}_E(K_4 - K_2)$ -orbits of  $\text{Col}_2(E_{K_4 - K_2})$ . Next to each representative edge-( $\leq 2$ )-coloring of  $K_4 - K_2$  in the figure is the number of colorings in its orbit.

<sup>†</sup> The reader familiar with group theory will notice that the group  $\text{Aut}(K_4 - K_2)$  is abstractly isomorphic to  $Z_2 \oplus Z_2$ .

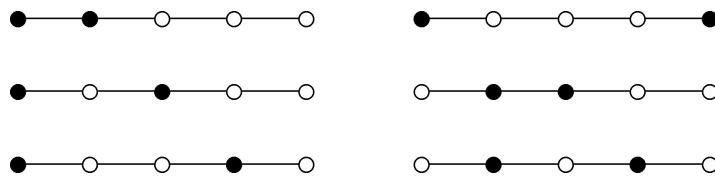


**Figure 14.2.6** Representatives and sizes of the 14 orbits of  $Col_2(E_{K_4-K_2})$ .

### Elementary Application of Symmetries in Itemizing Colorings

As the size of the graph and the number of colors increase, it becomes progressively less practical to count orbits by ordinary itemization. However, systematic exploitation of graph symmetries often reduces the work.

**Example 14.2.5:** A systematic approach to counting the  $\text{Aut}_V(P_5)$ -orbits (coloring classes) of vertex- $(\leq 2)$ -colorings of the path graph  $P_5$  may begin with the observation that there is only one vertex- $(\leq 2)$ -coloring with all vertices white. There are three coloring classes with four white vertices and one black, depending on whether the black vertex is at an end, next to an end, or in the middle. There are six coloring classes with three white and two black, as shown in Figure 14.2.7.



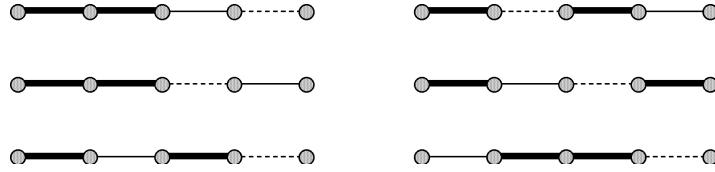
**Figure 14.2.7** The six vertex-colorings of  $P_5$  with 3 white and 2 black.

By symmetry between the colors white and black, there are also six coloring classes with three black vertices and two white, three coloring classes with four black and one white, and one class with all five vertices black, for a total of 20  $\text{Aut}_V(P_5)$ -orbits of colorings. In other words, there are 20 non-equivalent vertex- $(\leq 2)$ -colorings.

**Example 14.2.6:** The number of coloring classes of edge- $(\leq 2)$ -colorings of  $P_5$  is 10, according to the following inventory:

$$10 \text{ total} = \begin{cases} 1 & \text{with all four edges light} \\ 1 & \text{with all four edges dark} \\ 2 & \text{with three edges light and one edge dark} \\ 2 & \text{with one edge light and three edges dark} \\ 4 & \text{with two edges light and two edges dark} \end{cases}$$

**Example 14.2.7:** The number of (non-equivalent) edge- $(\leq 3)$ -colorings of  $P_5$  can be derived from the number of edge- $(\leq 2)$ -colorings. We begin by calculating the number of edge-3-colorings. The three different edge colors are depicted by *bold*, *plain*, and *dashed* edges. Figure 14.2.8 shows the six kinds of edge-3-colorings of  $P_5$  with the color *bold* used twice.



**Figure 14.2.8** Six of the non-equivalent edge-3-colorings of  $P_5$ .

Since there are three choices for the color that is used twice, there are 18 edge-3-colorings. Since  $P_5$  has eight edge-2-colorings with exactly two colors, according to the inventory in Example 14.2.6, and there are three choices for which two of the three colors are used, it follows that there are 24 edge- $(\leq 3)$ -colorings that use exactly two of the three available colors. Finally, there are three ways to color all the edges with exactly one of the three available colors. The following inventory summarizes the number of non-equivalent edge- $(\leq 3)$ -colorings of  $P_5$ .

$$45 \text{ total} = \begin{cases} 18 & \text{using all three colors} \\ 24 & \text{using exactly two of the three colors} \\ 3 & \text{using only one color} \end{cases}$$

If equivalences were ignored, the total would be  $81 = 3^4$ .

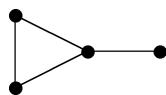
### EXERCISES for Section 14.2

In Exercises 14.2.1 through 14.2.7, group the full set of vertex- $(\leq 2)$ -colorings of the given graph  $G$  into  $\text{Aut}_V(G)$ -orbits, as in Figure 14.2.5.

14.2.1  $K_3$ .

14.2.2<sup>s</sup>  $P_4$ .

14.2.3



14.2.4  $K_{1,3}$ .

14.2.5  $K_{2,3}$ .

14.2.6  $K_5 - K_2$ .

14.2.7  $CL_3$ .

In Exercises 14.2.8 through 14.2.14, determine the number of  $\text{Aut}_V(G)$ -orbits of the full set of vertex- $(\leq 3)$ -colorings of the given graph  $G$ . Itemize by inventory as in Example 14.2.7.

14.2.8  $K_3$ .

14.2.9<sup>s</sup>  $P_4$ .

14.2.10 The graph of Exercise 14.2.3.

14.2.11  $K_{1,3}$ .

14.2.12  $K_{2,3}$ .

14.2.13  $K_5 - K_2$ .

14.2.14  $CL_3$ .

*In Exercises 14.2.15 through 14.2.21, determine the number of  $\text{Aut}_E(G)$ -orbits of the full set of edge- $(\leq 2)$ -colorings of the given graph. Itemize by inventory as in Example 14.2.6.*

14.2.15  $K_3$ .

14.2.16<sup>s</sup>  $P_4$ .

14.2.17 The graph of Exercise 14.2.3.

14.2.18  $K_{1,3}$ .

14.2.19  $K_{2,3}$ .

14.2.20  $K_5 - K_2$ .

14.2.21  $CL_3$ .

*In Exercises 14.2.22 through 14.2.28, determine the number of  $\text{Aut}_E(G)$ -orbits of the full set of edge- $(\leq 3)$ -colorings of the given graph. Itemize by inventory as in Example 14.2.7.*

14.2.22  $K_3$ .

14.2.23<sup>s</sup>  $P_4$ .

14.2.24 The graph of Exercise 14.2.3.

14.2.25  $K_{1,3}$ .

14.2.26  $K_{2,3}$ .

14.2.27  $K_5 - K_2$ .

14.2.28  $CL_3$ .

## 14.3 BURNSIDE'S LEMMA

There is a mathematical principle whose application permits quick calculation of the number of orbits under the action of a permutation group. It is commonly called Burnside's lemma, although it was first published by Frobenius. To state and prove Burnside's lemma, some additional terminology is helpful.

We illustrate the definitions and results in this section using the basic actions of  $\text{Aut}(G)$  on  $V_G$  and  $E_G$ . However, we will apply Burnside's lemma (in §14.5) to the *induced* actions of  $\text{Aut}(G)$  on the sets  $\text{Col}_k(V_G)$  and  $\text{Col}_k(E_G)$ .

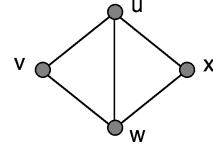
### Stabilizer of an Object

**DEFINITION:** Let  $\mathcal{P} = [P : Y]$  be a permutation group, and let  $y \in Y$ . The **stabilizer** of  $y$  is the subgroup  $\mathcal{S}\text{tab}(y) = \{\pi \in P \mid \pi(y) = y\}$ .

Thus, the stabilizer of an object  $y$  is simply the subgroup comprising all the permutations whose disjoint-cycle form contains the 1-cycle  $(y)$ .

**Example 14.3.1:** The analysis of  $K_4 - K_2$  and  $\text{Aut}(K_4 - K_2)$  continues.

Symmetry	$\pi \in \text{Aut}_V(G)$
identity	$\epsilon = (u)(v)(w)(x)$
refl. thru vert. axis	$\pi_1 = (u)(w)(v\ x)$
refl. thru horiz. axis	$\pi_2 = (v)(x)(u\ w)$
$180^\circ$ rotation	$\pi_3 = (u\ w)(v\ x)$

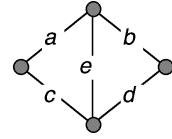


In  $\text{Aut}_V(K_4 - K_2)$ , the stabilizers are as follows:

$$\begin{aligned} \text{Stab}(u) &= \{\epsilon, \pi_1\} \\ \text{Stab}(v) &= \{\epsilon, \pi_2\} \\ \text{Stab}(w) &= \{\epsilon, \pi_1\} \\ \text{Stab}(x) &= \{\epsilon, \pi_2\} \end{aligned}$$

**Example 14.3.2:** That stabilizers of different objects need not have the same number of permutations is illustrated by  $\text{Aut}_E(K_4 - K_2)$ .

Symmetry	$\pi \in \text{Aut}_E(G)$
identity	$\epsilon = (a)(b)(c)(d)(e)$
refl. thru vert. axis	$\pi_1 = (e)(a\ b)(c\ d)$
refl. thru horiz. axis	$\pi_2 = (e)(a\ c)(b\ d)$
$180^\circ$ rotation	$\pi_3 = (e)(a\ d)(b\ c)$



In  $\text{Aut}_E(K_4 - K_2)$ , the stabilizers are as follows:

$$\begin{aligned} \text{Stab}(a) &= \text{Stab}(b) = \text{Stab}(c) = \text{Stab}(d) = \{\epsilon\} \\ \text{Stab}(e) &= \{\epsilon, \pi_1, \pi_2, \pi_3\} \end{aligned}$$

### Fixed-Point Set of a Permutation

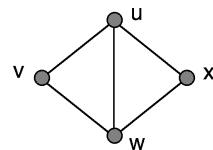
**DEFINITION:** Let  $\mathcal{P} = [P : Y]$  be a permutation group, and let  $\pi \in P$ . The **fixed-point set** of the permutation  $\pi$  is the subset  $\text{Fix}(\pi) = \{y \in Y \mid \pi(y) = y\}$ .

Thus,  $\text{Fix}(\pi)$  consists of the objects of  $Y$  appearing as 1-cycles in the disjoint-cycle form of  $\pi$ .

**TERMINOLOGY:** For a given automorphism  $\pi$  on a graph, the **fixed-vertex set** and **fixed-edge set** are the fixed-point sets of  $\pi_V$  and  $\pi_E$ , respectively.

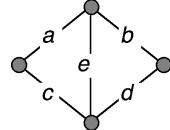
**Example 14.3.3:** The vertex-permutations in  $\text{Aut}_V(K_4 - K_2)$  have the following fixed-point sets

$$\begin{aligned} \text{Fix}((u)(v)(w)(x)) &= \{u, v, w, x\} \\ \text{Fix}((u)(w)(v\ x)) &= \{u, w\} \\ \text{Fix}((v)(x)(u\ w)) &= \{v, x\} \\ \text{Fix}((u\ w)(v\ x)) &= \emptyset \end{aligned}$$



**Example 14.3.4:** The edge-permutations in  $\text{Aut}_E(K_4 - K_2)$  have the following fixed-point sets

$$\begin{aligned}
 Fix((a)(b)(c)(d)(e)) &= \{a, b, c, d, e\} \\
 Fix((e)(a\ b)(c\ d)) &= \{e\} \\
 Fix((e)(a\ c)(b\ d)) &= \{e\} \\
 Fix((e)(a\ d)(b\ c)) &= \{e\}
 \end{aligned}$$



The next three lemmas are needed for the proof of Burnside's lemma.

### Relationship Between Stabilizers and Fixed-Point Sets

**Lemma 14.3.1.** Let  $\mathcal{P} = [P : Y]$  be a permutation group. Then

$$\sum_{y \in Y} |\mathcal{S}tab(y)| = \sum_{\pi \in P} |Fix(\pi)|$$

**Proof:** Consider a matrix whose rows are indexed by the objects of the set  $Y$  and whose columns are indexed by the permutations in  $P$ , and whose entry in row  $y$  and column  $\pi$  is 1 if  $\pi(y) = y$ , but 0 otherwise. Then for each  $y$ , the summand on the left side of the equation is the sum of row  $y$  (i.e., the number of 1's), and the summand on the right side is the sum of column  $\pi$ . The equation simply asserts that the sum of the row sums of the matrix equals the sum of the column sums.  $\diamond$

**Example 14.3.5:** In  $\mathcal{A}ut_V(K_4 - K_2)$ , the sum of the stabilizer sizes (from Example 14.3.1) is

$$\sum_{y \in \mathcal{A}ut_V(K_4 - K_2)} |\mathcal{S}tab(y)| = 2 + 2 + 2 + 2 = 8$$

and the sum of the sizes of the fixed-vertex sets (from Example 14.3.3) is

$$\sum_{\pi \in P} |Fix(\pi)| = 4 + 2 + 2 + 0 = 8$$

**Example 14.3.6:** In  $\mathcal{A}ut_E(K_4 - K_2)$ , the sum of the stabilizer sizes (from Example 14.3.2) is

$$\sum_{y \in \mathcal{A}ut_E(K_4 - K_2)} |\mathcal{S}tab(y)| = 1 + 1 + 1 + 1 + 4 = 8;$$

and the sum of the sizes of the fixed-edge sets (from Example 14.3.4) is

$$\sum_{\pi \in P} |Fix(\pi)| = 5 + 1 + 1 + 1 = 8$$

### Relationship Between Stabilizers and Orbits

**Lemma 14.3.2.** Let  $\mathcal{P} = [P : Y]$  be a permutation group and  $y \in Y$ . Then

$$|\mathcal{S}tab(y)| = \frac{|P|}{|\text{orbit}(y)|}$$

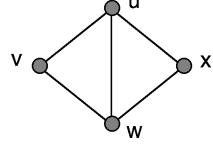
**Proof:** Suppose that  $\text{orbit}(y) = \{y=y_1, y_2, \dots, y_n\}$  and that, for  $j = 1, \dots, n$ ,  $P_j$  is the subset of permutations of  $P$  that maps object  $y$  to object  $y_j$ . Then the subsets  $P_1, P_2, \dots, P_n$  partition the permutation group  $P$ , and  $P_1 = \mathcal{S}tab(y)$ .

For  $j = 1, \dots, n$ , let  $\pi_j$  be any permutation such that  $\pi_j(y) = y_j$ . Then the rule  $\pi \mapsto \pi \circ \pi_j$  (composition with  $\pi_j$ ) is a bijection from  $P_1$  to  $P_j$ , which implies that  $|P_j| = |P_1| = |\text{Stab}(y)|$ , for  $j = 1, \dots, n$ .

Since each of the  $n$  partition cells  $P_1, P_2, \dots, P_n$  of group  $P$  has cardinality  $|\text{Stab}(y)|$ , it follows that  $n \cdot |\text{Stab}(y)| = |P|$ . But  $n = |\text{orbit}(y)|$ , which completes the proof.  $\diamond$

**Example 14.3.7:** Analysis of  $\text{Aut}_V(K_4 - K_2)$  continues.

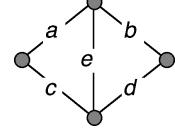
Symmetry	$\pi \in \text{Aut}_V(G)$
identity	$\epsilon = (u)(v)(w)(x)$
refl. thru vert. axis	$\pi_1 = (u)(w)(v\ x)$
refl. thru horiz. axis	$\pi_2 = (v)(x)(u\ w)$
180° rotation	$\pi_3 = (u\ w)(v\ x)$



The orbits of  $\text{Aut}_V(K_4 - K_2)$  are  $\{u, w\}$  and  $\{v, x\}$ , both of cardinality 2. All the stabilizers are of cardinality 2, as determined in Example 14.3.1. The cardinality of the group  $\text{Aut}_V(K_4 - K_2)$  is 4. Thus, for each vertex, the equation  $|\text{Stab}(y)| = \frac{|P|}{|\text{orbit}(y)|}$  takes the form  $2 = \frac{4}{2}$ .

**Example 14.3.8:** Analysis of  $\text{Aut}_E(K_4 - K_2)$  continues.

Symmetry	$\pi \in \text{Aut}_E(G)$
identity	$\epsilon = (a)(b)(c)(d)(e)$
refl. thru vert. axis	$\pi_1 = (e)(a\ b)(c\ d)$
refl. thru horiz. axis	$\pi_2 = (e)(a\ c)(b\ d)$
180° rotation	$\pi_3 = (e)(a\ d)(b\ c)$



The orbits of  $\text{Aut}_E(K_4 - K_2)$  are  $\{e\}$  and  $\{a, b, c, d\}$ . For each of the edges  $a, b, c$ , and  $d$ , the equation  $|\text{Stab}(y)| = \frac{|P|}{|\text{orbit}(y)|}$  becomes  $1 = \frac{4}{4}$ . For edge  $e$ , it becomes  $4 = \frac{4}{1}$ .

**Remark:** Although the next lemma is expressed in terms of the orbits of permutation groups, it is really a fact about set partitions.

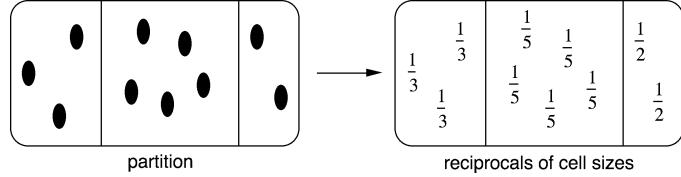
**Lemma 14.3.3.** Let  $\mathcal{P} = [P : Y]$  be a permutation group with  $n$  orbits. Then

$$\sum_{y \in Y} \frac{1}{|\text{orbit}(y)|} = n$$

**Proof:** Suppose that  $Y_1, \dots, Y_n$  are the orbits. Then  $Y = Y_1 \cup \dots \cup Y_n$ . It follows that

$$\begin{aligned} \sum_{y \in Y} \frac{1}{|\text{orbit}(y)|} &= \sum_{j=1}^n \sum_{y \in Y_j} \frac{1}{|\text{orbit}(y)|} \\ &= \sum_{j=1}^n \sum_{y \in Y_j} \frac{1}{|Y_j|} \\ &= \sum_{j=1}^n \frac{1}{|Y_j|} \sum_{y \in Y_j} 1 = \sum_{j=1}^n \frac{1}{|Y_j|} |Y_j| \\ &= \sum_{j=1}^n 1 = n \end{aligned} \quad \diamond$$

**Example 14.3.9:** The left side of the equation in Lemma 14.3.3 is the sum of the reciprocals of the sizes of the cells in a partition. The partition depicted at the left of Figure 14.3.1 has three cells. When the objects in the cells are converted to the reciprocals of the cell sizes, as on the right, it becomes apparent that the sum of the fractions within each cell must be equal to 1. Thus, the sum of all the reciprocals must equal the number of cells, as on the right side of the equation in Lemma 14.3.3.



**Figure 14.3.1** Reciprocals of cell sizes of a partition.

### Proof of Burnside's Lemma

**Theorem 14.3.4. (Burnside's lemma)** Let  $\mathcal{P} = [P : Y]$  be a permutation group with  $n$  orbits. Then

$$n = \frac{1}{|P|} \sum_{\pi \in P} |\text{Fix}(\pi)|$$

**Proof:** Lemmas 14.3.1, 14.3.2, and 14.3.3 establish the following chain of equalities, which proves Burnside's lemma.

$$\begin{aligned} \frac{1}{|P|} \sum_{\pi \in P} |\text{Fix}(\pi)| &= \frac{1}{|P|} \sum_{y \in Y} |\text{Stab}(y)| && \text{(by Lemma 14.3.1)} \\ &= \frac{1}{|P|} \sum_{y \in Y} \frac{|P|}{|\text{orbit}(y)|} && \text{(by Lemma 14.3.2)} \\ &= \frac{1}{|P|} |P| \sum_{y \in Y} \frac{1}{|\text{orbit}(y)|} \\ &= \sum_{y \in Y} \frac{1}{|\text{orbit}(y)|} = n && \text{(by Lemma 14.3.3)} \quad \diamond \end{aligned}$$

### Direct Application of Burnside's Lemma

The most powerful applications of Burnside's lemma are not to counting vertex orbits or edge orbits, but rather, to counting induced equivalence classes, for which purpose they require the use of auxiliary results and techniques, which are the focus of the rest of the chapter. However, a direct orbit-counting application of Burnside's lemma to a permutation group  $\mathcal{P} = [P : Y]$  would proceed as follows:

1. the values of  $|\text{Fix}(\pi)|$  are added over all  $\pi \in P$ ;
2. the resulting sum is divided by  $|P|$ .

The following two examples apply the direct method to counting vertex-orbits and edge-orbits.

**Example 14.3.10:**  $\text{Aut}_V(K_4 - K_2)$  has four automorphisms. The sum of the sizes of their fixed-point sets, previously calculated in Example 14.3.5, is

$$\sum_{\pi \in P} |Fix(\pi)| = 4 + 2 + 2 + 0 = 8$$

The orbits are  $\{u, w\}$  and  $\{v, x\}$ . Since  $|Aut_V(K_4 - K_2)| = 4$ , Burnside's lemma implies correctly that the number of orbits is  $2 = 8/4$ .

**Example 14.3.11:**  $Aut_E(K_4 - K_2)$  has four automorphisms. The sum of the sizes of their fixed-point sets, previously calculated in Example 14.3.6, is

$$\sum_{\pi \in P} |Fix(\pi)| = 5 + 1 + 1 + 1 = 8$$

The orbits are  $\{a, b, c, d\}$  and  $\{e\}$ . Since  $|Aut_V(K_4 - K_2)| = 4$ , Burnside's lemma implies correctly that the number of orbits is  $2 = 8/4$ .

### EXERCISES for Section 14.3

*In Exercises 14.3.1 through 14.3.16, do each of the following for the vertex-permutation group of the indicated graph.*

- a. Determine the stabilizers of all objects.
- b. Determine the fixed points of all permutations.
- c. Determine the number of orbits.
- d. Confirm Lemma 14.3.1.
- e. Confirm Lemma 14.3.2.
- f. Confirm Lemma 14.3.3.
- g. Confirm Burnside's lemma.

- 14.3.1 The graph of Exercise 14.1.1.
- 14.3.2 The graph of Exercise 14.1.2.
- 14.3.3<sup>s</sup> The graph of Exercise 14.1.3.
- 14.3.4 The graph of Exercise 14.1.4.
- 14.3.5<sup>s</sup> The graph of Exercise 14.1.5.
- 14.3.6 The graph of Exercise 14.1.6.
- 14.3.7 The graph of Exercise 14.1.7.
- 14.3.8 The graph of Exercise 14.1.8.
- 14.3.9 The graph of Exercise 14.1.9.
- 14.3.10 The graph of Exercise 14.1.10.
- 14.3.11 The graph of Exercise 14.1.11.
- 14.3.12  $P_4$ .
- 14.3.13  $P_4 + K_1$ .
- 14.3.14 The graph of Exercise 14.1.14.
- 14.3.15 The graph of Exercise 14.1.15.
- 14.3.16  $P_4 + K_2$ .

In Exercises 14.3.17 through 14.3.32, do each of the following for the edge-permutation group of the indicated graph.

- a. Determine the stabilizers of all objects.
- b. Determine the fixed points of all permutations.
- c. Determine the number of orbits.
- d. Confirm Lemma 14.3.1.
- e. Confirm Lemma 14.3.2.
- f. Confirm Lemma 14.3.3.
- g. Confirm Burnside's lemma.

14.3.17 The graph of Exercise 14.1.1.

14.3.18 The graph of Exercise 14.1.2.

14.3.19<sup>s</sup> The graph of Exercise 14.1.3.

14.3.20 The graph of Exercise 14.1.4.

14.3.21<sup>s</sup> The graph of Exercise 14.1.5.

14.3.22 The graph of Exercise 14.1.6.

14.3.23 The graph of Exercise 14.1.7.

14.3.24 The graph of Exercise 14.1.8.

14.3.25 The graph of Exercise 14.1.9.

14.3.26 The graph of Exercise 14.1.10.

14.3.27 The graph of Exercise 14.1.11.

14.3.28  $P_4$ .

14.3.29  $P_4 + K_1$ .

14.3.30 The graph of Exercise 14.1.14.

14.3.31 The graph of Exercise 14.1.15.

14.3.32  $P_4 + K_2$ .

---

## 14.4 CYCLE-INDEX POLYNOMIAL OF A PERMUTATION GROUP

The *cycle-index polynomial* is a polynomial that displays the cycle structure of a permutation group. What is commonly considered to be the most important principle in enumerative graph theory is that substituting the value  $k$  into the cycle-index polynomial yields the number of equivalence classes of  $(\leq k)$ -colorings. This section examines some examples of applications of this substitution principle and then proves its correctness.

### Cycle-Structure Monomial of a Permutation

TERMINOLOGY: The **cycle structure of a permutation** is the number of cycles of each length in its disjoint-cycle form.

**TERMINOLOGY:** A **monomial** is a polynomial with only one term.

**DEFINITION:** Let  $\mathcal{P} = [P : Y]$  be a permutation group on a set  $Y$  of  $n$  objects, and let  $\pi \in P$ . The **cycle-structure monomial** of  $\pi$  is the  $n$ -variable monomial

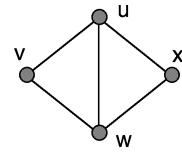
$$\zeta(\pi) = \prod_{k=1}^n z_k^{r_k} = z_1^{r_1} z_2^{r_2} \cdots z_n^{r_n}$$

where  $z_k$  is a formal variable and  $r_k$  is the number of  $k$ -cycles in the disjoint-cycle form of  $\pi$ .

**Example 14.4.1:** The permutation  $\pi = (1\ 7\ 9\ 3)(2\ 4\ 8\ 6)(5)$  has cycle-structure monomial  $\zeta(\pi) = z_1 z_4^2$ , because  $\pi$  has one 1-cycle and two 4-cycles.

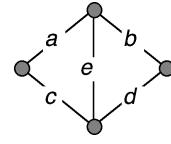
**Example 14.4.2:** The cycle structure of all the vertex-permutations in  $\text{Aut}_V(K_4 - K_2)$  is given in the following table.

Symmetry	$\pi \in \text{Aut}_V(G)$	Cycle structure
identity	$(u)(v)(w)(x)$	$z_1^4$
refl. thru vert. axis	$(u)(w)(v\ x)$	$z_1^2 z_2$
refl. thru horiz. axis	$(v)(x)(u\ w)$	$z_1^2 z_2$
180° rotation	$(u\ w)(v\ x)$	$z_2^2$



**Example 14.4.3:** The cycle structure of all the edge-permutations in  $\text{Aut}_E(K_4 - K_2)$  is given in the following table.

Symmetry	$\pi \in \text{Aut}_E(G)$	Cycle structure
identity	$(a)(b)(c)(d)(e)$	$z_1^5$
refl. thru vert. axis	$(e)(a\ b)(c\ d)$	$z_1 z_2^2$
refl. thru horiz. axis	$(e)(a\ c)(b\ d)$	$z_1 z_2^2$
180° rotation	$(e)(a\ d)(b\ c)$	$z_1 z_2^2$



### Cycle-Index Polynomial

The last two examples used the actions of  $\text{Aut}(G)$  on  $V_G$  and  $E_G$ , but the remaining examples use the *induced* actions of  $\text{Aut}(G)$  on the sets  $\text{Col}_k(V_G)$  and  $\text{Col}_k(E_G)$ .

**DEFINITION:** Let  $\mathcal{P} = [P : Y]$  be a permutation group on a set of  $n$  objects. Then the **cycle-index polynomial** of  $\mathcal{P}$  is the polynomial

$$\mathcal{Z}_{\mathcal{P}}(z_1, \dots, z_n) = \frac{1}{|P|} \sum_{\pi \in P} \zeta(\pi)$$

where  $\zeta(\pi)$  is the cycle-structure monomial of the permutation  $\pi$ .

**Example 14.4.4:** From Example 14.4.2, it follows that the cycle-index polynomial of the permutation group  $\text{Aut}_V(K_4 - K_2)$  is

$$\mathcal{Z}_{\text{Aut}_V(K_4 - K_2)}(z_1, z_2) = \frac{1}{4}(z_1^4 + 2z_1^2 z_2 + z_2^2)$$

Substituting 2, the number of colors, for both of the variables  $z_1$  and  $z_2$  yields the number

$$\mathcal{Z}_{\text{Aut}_V(K_4-K_2)}(2, 2) = \frac{1}{4}(2^4 + 2 \cdot 2^2 \cdot 2 + 2^2) = 9$$

which was shown in Figure 14.2.5 to be the number of  $\text{Aut}_V(K_4-K_2)$ -orbits of vertex- $(\leq 2)$ -colorings of the graph  $K_4-K_2$ .

**Example 14.4.5:** From Example 14.4.3, it follows that the cycle-index polynomial of the permutation group  $\text{Aut}_E(K_4-K_2)$  is

$$\mathcal{Z}_{\text{Aut}_E(K_4-K_2)}(z_1, z_2) = \frac{1}{4}(z_1^5 + 3z_1z_2^2)$$

Again, substituting 2 for both variables, we obtain

$$\mathcal{Z}_{\text{Aut}_E(K_4-K_2)}(2, 2) = \frac{1}{4}(2^5 + 3 \cdot 2 \cdot 2^2) = 14$$

which was shown in Figure 14.2.6 to be the number of  $\text{Aut}_E(K_4-K_2)$ -orbits of edge- $(\leq 2)$ -colorings.

That the substitutions in the last two examples yielded the number of coloring classes is not a coincidence.

### Correctness of Substituting Into the Cycle-Index Polynomial

We now show as a consequence of Burnside's lemma (Theorem 14.3.4) that substituting the value  $k$  into the cycle-index polynomial *always* yields the number of coloring classes (orbits) of the  $(\leq k)$ -colorings.

REVIEW FROM §14.2: Let  $\mathcal{P} = [P : Y]$  be a permutation group acting on a set  $Y$ , and let  $\pi \in P$ .

- The mapping  $\pi_{YC} : Col_k(Y) \rightarrow Col_k(Y)$  defined by  $\pi_{YC}(f) = f\pi$ , for every coloring  $f \in Col_k(Y)$  is a permutation on the set  $Col_k(Y)$ , called the **induced permutation action** of  $\pi$  on  $Col_k(Y)$ .
- To distinguish between the action of a permutation  $\pi$  on a set  $Y$  and its induced action on the set  $Col_k(Y)$  of colorings of  $Y$ , we let  $\pi_Y$  denote its action on  $Y$  and  $\pi_{YC}$  its action on  $Col_k(Y)$ . When there is no risk of confusion, the subscripts  $Y$  and  $YC$  may both be omitted.
- The collection  $\mathcal{P}_C = [P : Col_k(Y)]$  of induced permutations on  $Col_k(Y)$  is called the **induced permutation group**.
- The set of orbits (coloring classes) of the induced permutation group on  $Col_k(Y)$  is denoted  $\{Col_k(Y)\}_{\mathcal{P}}$ .
- When it is necessary to distinguish between the group that acts on the set  $Y$  and the group that acts on the set  $Col_k(Y)$ , the respective notations  $P_Y$  and  $P_{YC}$  are used.

NOTATION: If  $p(x_1, \dots, x_n)$  is a multivariate polynomial, then  $p(k, \dots, k)$  denotes the result of substituting the value  $k$  for every variable  $x_j$ .

**Lemma 14.4.1.** Let  $\mathcal{P} = [P : Y]$  be a permutation group, and let  $\pi \in P$ , with induced action  $\pi_{YC}$  on  $Col_k(Y)$ . Then the number of  $(\leq k)$ -colorings of  $Y$  that are fixed by  $\pi_{YC}$  is given by

$$|Fix(\pi_{YC})| = \zeta(\pi_Y)(k, \dots, k)$$

**Proof:** A  $(\leq k)$ -coloring  $c$  is fixed by  $\pi_{YC}$  if and only if within each cycle of  $\pi_Y$ , all the objects are assigned the same color by  $c$ . Thus, there are  $k$  independent choices possible for each cycle of  $\pi_Y$ . Therefore,  $|Fix(\pi_{YC})| = k^n$ , where  $n$  is the number of cycles in  $\pi_Y$ . But  $k^n$  is precisely the value of  $\zeta(\pi_Y)(k, \dots, k)$ .  $\diamond$

**Example 14.4.6:** Recall from Figure 14.2.2 the induced permutations from the action of  $Aut(C_3)$  on  $Col_2(C_3)$ . The following table illustrates the application of Lemma 14.4.1.

$\pi_Y \in P_Y$	$\pi_{YC} \in P_{YC}$	$\zeta(\pi_Y)$	$ Fix(\pi_{YC}) $
$\epsilon_Y$	$\epsilon_{YC}$	$z_1^3$	8
$(x \ y \ z)$	$(111)(211 \ 121 \ 112)(122 \ 212 \ 221)(222)$	$z_3$	2
$(x \ z \ y)$	$(111)(211 \ 112 \ 121)(122 \ 221 \ 212)(222)$	$z_3$	2

**Theorem 14.4.2.** Let  $\mathcal{P} = [P : Y]$  be a permutation group. Then

$$|\{Col_k(Y)\}_{\mathcal{P}}| = \mathcal{Z}_{\mathcal{P}}(k, \dots, k)$$

**Proof:** Applying Burnside's lemma to the induced permutation group  $\mathcal{P}_C$  yields the equation

$$\begin{aligned} |\{Col_k(Y)\}_{\mathcal{P}}| &= \frac{1}{|P_C|} \sum_{\pi_{YC} \in P_C} |Fix(\pi_{YC})| \\ &= \frac{1}{|P_C|} \sum_{\pi_{YC} \in P_C} \zeta(\pi_Y)(k, \dots, k) \quad (\text{by Lemma 14.4.1}) \\ &= \frac{1}{|P_Y|} \sum_{\pi_Y \in P_Y} \zeta(\pi_Y)(k, \dots, k) \\ &= \mathcal{Z}_{\mathcal{P}}(k, \dots, k) \end{aligned} \quad \diamond$$

**Example 14.4.7:**  $Aut(P_5)$  has two automorphisms, the identity and the reflection, and the cycle-index polynomial for  $Aut_V(P_5)$  is given by

$$\mathcal{Z}_{Aut_V(P_5)}(z_1, z_2) = \frac{1}{2}(z_1^5 + z_1 z_2^2)$$

Substituting 2 for both the variables yields

$$\mathcal{Z}_{Aut_V(P_5)}(2, 2) = \frac{1}{2}(2^5 + 2 \cdot 2^2) = 20$$

which is the number of vertex- $(\leq 2)$ -colorings calculated in Example 14.2.5.

Similarly, the cycle-index polynomial for  $\text{Aut}_E(P_5)$  is

$$\mathcal{Z}_{\text{Aut}_E(P_5)}(z_1, z_2) = \frac{1}{2}(z_1^4 + z_2^2)$$

and, hence,

$$\mathcal{Z}_{\text{Aut}_E(P_5)}(2, 2) = \frac{1}{2}(2^4 + 2^2) = 10$$

which is the number of edge- $(\leq 2)$ -colorings calculated in Example 14.2.6.

Moreover, substituting 3 for both the variables yields

$$\mathcal{Z}_{\text{Aut}_E(P_5)}(3, 3) = \frac{1}{2}(3^4 + 3^2) = 45$$

which was calculated in Example 14.2.7 to be the number of edge- $(\leq 3)$ -colorings of  $P_5$ .

### EXERCISES for Section 14.4

*In Exercises 14.4.1 through 14.4.16, for the given graph,*

- a. Calculate the cycle-index polynomial of  $\text{Aut}_V(G)$ , and evaluate the result of substituting the number 2 for all the variables.
- b. Calculate the cycle-index polynomial of  $\text{Aut}_E(G)$ , and evaluate the result of substituting the number 2 for all the variables.
- c. Use drawings and/or description to account for all the non-equivalent vertex- $(\leq 2)$ -colorings, and make sure that their number agrees with your answer to (a).
- d. Use drawings and/or description to account for all the non-equivalent edge- $(\leq 2)$ -colorings, and make sure that their number agrees with your answer to (b).

14.4.1 The graph of Exercise 14.1.1.

14.4.2 The graph of Exercise 14.1.2.

14.4.3<sup>s</sup> The graph of Exercise 14.1.3.

14.4.4 The graph of Exercise 14.1.4.

14.4.5 The graph of Exercise 14.1.5.

14.4.6 The graph of Exercise 14.1.6.

14.4.7 The graph of Exercise 14.1.7.

14.4.8 The graph of Exercise 14.1.8.

14.4.9 The graph of Exercise 14.1.9.

14.4.10 The graph of Exercise 14.1.10.

14.4.11 The graph of Exercise 14.1.11.

14.4.12  $P_4$ .

14.4.13<sup>s</sup>  $P_4+K_1$ .

14.4.14 The graph of Exercise 14.1.14.

14.4.15 The graph of Exercise 14.1.15.

14.4.16  $P_4+K_2$ .

*Exercises 14.4.17 through 14.4.21 are concerned with making a table, as in the continuation of Example 14.2.1, for the automorphism group  $\text{Aut}(G)$  of a given graph. In the first column, list all the automorphisms (as vertex-permutations), and in the second column, list the corresponding induced permutations on  $\text{Col}_2(V_G)$ . In the strings that code the colorings, the left-to-right order of the symbols should correspond to lexicographic order of the vertex-names.*

- 14.4.17 The graph of Exercise 14.1.1.
- 14.4.18 The graph of Exercise 14.1.2.
- 14.4.19 The graph of Exercise 14.1.3.
- 14.4.20 The graph of Exercise 14.1.4.
- 14.4.21 The graph of Exercise 14.1.5.

*Exercises 14.4.22 through 14.4.26 are concerned with making a table, as in the continuation of Example 14.2.1, for the automorphism group  $\text{Aut}(G)$  of a given graph. In the first column, list all the automorphisms (as edge-permutations), and in the second column, list the corresponding induced permutations on  $\text{Col}_2(E_G)$ . In the strings that code the colorings, the left-to-right order of the symbols should correspond to lexicographic order of the edge-names.*

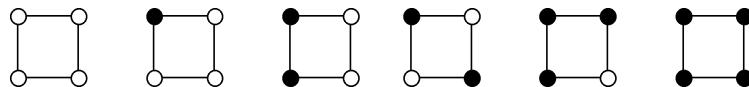
- 14.4.22 The graph of Exercise 14.1.1.
- 14.4.23 The graph of Exercise 14.1.2.
- 14.4.24 The graph of Exercise 14.1.3.
- 14.4.25 The graph of Exercise 14.1.4.
- 14.4.26 The graph of Exercise 14.1.5.

## 14.5 MORE COUNTING, INCLUDING SIMPLE GRAPHS

In this section, we illustrate further applications of Burnside's lemma and the substitution principle (Theorem 14.4.2). The section ends by showing how the problem of counting the number of isomorphism classes of simple graphs can be reduced to the problem of counting the non-equivalent edge- $(\leq 2)$ -colorings of the complete graph.

### Counting Necklaces

**Example 14.5.1:** Consider the set of 4-beaded necklaces that can be constructed using only black beads and white beads. This set can be modeled as the coloring classes of vertex- $(\leq 2)$ -colorings of the cycle graph  $C_4$  (i.e., the  $\text{Aut}_V(C_4)$ -orbits of  $\text{Col}_2(V_{C_4})$ ). Figure 14.5.1 shows all the possible kinds of colorings.



**Figure 14.5.1** The six non-equivalent vertex- $(\leq 2)$ -colorings of  $C_4$ .

It is straightforward to verify that the cycle-index polynomial for  $\text{Aut}_V(C_4)$  is given by

$$\mathcal{Z}_{\text{Aut}_V(C_4)}(z_1, z_2, z_3, z_4) = \frac{1}{8}(z_1^4 + 2z_1^2z_2 + 3z_2^2 + 2z_4)$$

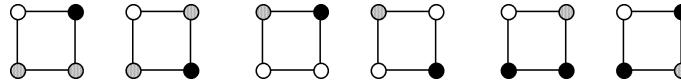
Applying Theorem 14.4.2, we have

$$\mathcal{Z}_{\text{Aut}_V(C_4)}(2, 2, 2, 2) = \frac{1}{8}(2^4 + 2 \cdot 2^2 \cdot 2 + 3 \cdot 2^2 + 2 \cdot 2) = 6$$

**Example 14.5.2:** Now consider vertex- $(\leq 3)$ -colorings of  $C_4$ . By Theorem 14.4.2, the number of non-equivalent vertex- $(\leq 3)$ -colorings can be calculated as follows:

$$\mathcal{Z}_{\text{Aut}_V(C_4)}(3, 3, 3, 3) = \frac{1}{8}(3^4 + 2 \cdot 3^2 \cdot 3 + 3 \cdot 3^2 + 2 \cdot 3) = 21$$

Direct counting gives three colorings that use only one of the three colors. Since there are four ways to vertex-color  $C_4$  with exactly two colors and three ways to choose two colors from a set of three, there are  $3 \cdot 4 = 12$  colorings of  $C_4$  that use exactly two of the three colors. Figure 14.5.2 shows that when exactly three colors are used, there are six colorings.

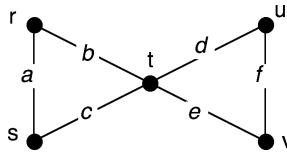


**Figure 14.5.2** The six vertex-colorings of  $C_4$  with three colors.

Thus, in all, there are  $3 + 12 + 6 = 21$  vertex- $(\leq 3)$ -colorings of  $C_4$ , confirming the value obtained by Theorem 14.4.2.

### Counting Vertex-Colorings of a Bowtie

**Example 14.5.3:** Figure 14.5.3 shows the *bowtie*  $BT$ , the graph that results from the vertex-amalgamation of two copies of  $K_3$ .



**Figure 14.5.3** The bowtie graph  $BT$ .

The bowtie  $BT$  has eight automorphisms. The corresponding vertex- and edge-permutations, along with their cycle structures, are given in the table below.

$\pi \in \text{Aut}_V(BT)$	Cycle structure	$\pi \in \text{Aut}_E(BT)$	Cycle structure
$(r)(s)(t)(u)(v)$	$z_1^5$	$(a)(b)(c)(d)(e)(f)$	$z_1^6$
$(r\ s)(t)(u)(v)$	$z_1^3 z_2$	$(a)(b\ c)(d)(e)(f)$	$z_1^4 z_2$
$(r)(s)(t)(u\ v)$	$z_1^3 z_2$	$(a)(b)(c)(d\ e)(f)$	$z_1^4 z_2$
$(r\ s)(t)(u\ v)$	$z_1 z_2^2$	$(a)(b\ c)(d\ e)(f)$	$z_1^2 z_2^2$
$(r\ u)(t)(s\ v)$	$z_1 z_2^2$	$(a\ f)(b\ d)(c\ e)$	$z_2^3$
$(r\ v\ s\ u)(t)$	$z_1 z_4$	$(a\ f)(b\ e\ c\ d)$	$z_2 z_4$
$(r\ u\ s\ v)(t)$	$z_1 z_4$	$(a\ f)(b\ d\ e\ c)$	$z_2 z_4$
$(r\ v)(t)(s\ u)$	$z_1 z_2^2$	$(a\ f)(b\ e)(c\ d)$	$z_2^3$

Thus, the cycle-index polynomial for  $\mathcal{A}ut_V(BT)$  is

$$\mathcal{Z}_{\mathcal{A}ut_V(BT)}(z_1, z_2, z_3, z_4) = \frac{1}{8}(z_1^5 + 2z_1^3z_2 + 3z_1z_2^2 + 2z_1z_4)$$

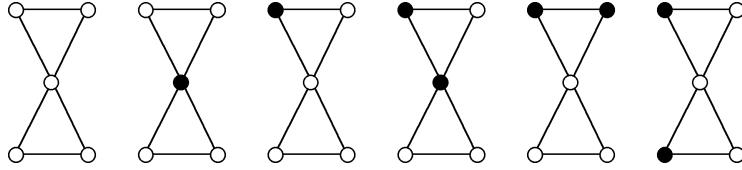
and the cycle-index polynomial for  $\mathcal{A}ut_E(BT)$  is

$$\mathcal{Z}_{\mathcal{A}ut_E(BT)}(z_1, z_2, z_3, z_4) = \frac{1}{8}(z_1^6 + 2z_1^4z_2 + z_1^2z_2^2 + 2z_2^3 + 2z_2z_4)$$

By Theorem 14.4.2, the number of non-equivalent vertex- $(\leq 2)$ -colorings of  $BT$  is

$$\mathcal{Z}_{\mathcal{A}ut_V(BT)}(2, 2, 2, 2) = \frac{1}{8}(2^5 + 2 \cdot 2^3 \cdot 2 + 3 \cdot 2 \cdot 2^2 + 2 \cdot 2 \cdot 2) = 12$$

Figure 14.5.4 confirms this calculation. Since there are six white-black vertex-colorings with at most two black, there must also be another six with at most two white.



**Figure 14.5.4** White-black vertex-colorings of  $BT$  with at most two black.

**Example 14.5.3, continued:** Similarly, by Theorem 14.4.2, the number of edge- $(\leq 2)$ -colorings of  $BT$  is obtained as follows:

$$\mathcal{Z}_{\mathcal{A}ut_E(BT)}(2, 2, 2, 2) = \frac{1}{8}(2^6 + 2 \cdot 2^4 \cdot 2 + 2^2 \cdot 2^2 + 2 \cdot 2^3 + 2 \cdot 2 \cdot 2) = 21$$

Confirming this calculation by drawings is left as an exercise.

### Counting Simple Graphs

Theorem 14.4.2 can be used to count the isomorphism types (classes) of  $n$ -vertex simple graphs by modeling the types as the coloring classes ( $\mathcal{A}ut_E(K_n)$ -orbits) of the edge- $(\leq 2)$ -colorings of the complete graph  $K_n$ . The following proposition formalizes this connection.

**Proposition 14.5.1.** *The number of isomorphism types of the  $n$ -vertex simple graphs equals the number of coloring classes of the edge- $(\leq 2)$ -colorings of  $K_n$ , that is,  $|\{\text{Col}_2(E_{K_n})\}_{\mathcal{A}ut_E(K_n)}|$ .*

**Proof:** If the two colors used for coloring the edges of  $K_n$  are regarded as *present* and *absent*, then the full set of  $2^n$  edge- $(\leq 2)$ -colorings is in one-to-one correspondence with the set of  $2^n$   $n$ -vertex simple graphs, and hence, the coloring classes correspond to the isomorphism types.  $\diamond$

Proposition 14.5.1 suggests the following strategy for counting the isomorphism types of the  $n$ -vertex simple graphs. The strategy is illustrated by a series of propositions for counting the isomorphism types of simple graphs with 4 and 5 vertices.

### General Strategy for Counting n-Vertex Simple Graphs

**Step 1:** Calculate the cycle-index polynomial of  $\text{Aut}_V(K_n)$ .

Since knowing the cycle-index polynomial is sufficient for algebraic counting, writing out all the permutations in a large permutation group can be avoided.

**Step 2:** Calculate the cycle-index polynomial of  $\text{Aut}_E(K_n)$ .

The cycle-index polynomial of  $\text{Aut}_E(K_n)$  is obtained by considering each cycle size and each pair of cycle sizes in the cycle-index polynomial of  $\text{Aut}_V(K_n)$ .

**Step 3:** Apply Theorem 14.4.2.

This final step in counting the isomorphism types of graphs with  $n$  vertices is simply to substitute 2 for every variable in the cycle-index polynomial  $\mathcal{Z}_{\text{Aut}_E(K_n)}$ .

### Counting Simple Graphs on 4 Vertices

**Proposition 14.5.2.** The cycle-index polynomial of  $\text{Aut}_V(K_4)$  is

$$\mathcal{Z}_{\text{Aut}_V(K_4)}(z_1, z_2, z_3, z_4) = \frac{1}{24}(z_1^4 + 6z_1^2z_2 + 8z_1z_3 + 3z_2^2 + 6z_4)$$

**Proof:** The 24 vertex-permutations in  $\text{Aut}_V(K_4)$  are naturally partitioned according to the five possible cycle structures:  $z_1^4$ ,  $z_1^2z_2$ ,  $z_1z_3$ ,  $z_2^2$ , and  $z_4$ . Each cell in this partition is to be counted.

$z_1^4$ : 1 automorphism.

Only the identity permutation has this cycle structure.

$z_1^2z_2$ : 6 automorphisms.

There are  $\binom{4}{2} = 6$  ways to choose two vertices for the 2-cycle.

$z_1z_3$ : 8 automorphisms.

There are  $\binom{4}{3} = 4$  ways to choose three vertices for the 3-cycle and  $(3-1)! = 2$  ways to arrange them in a cycle.

$z_2^2$ : 3 automorphisms.

There are three ways to group four objects into two cycles, when it does not matter which cycle is written first.

$z_4$ : 6 automorphisms.

They correspond to the  $(4-1)! = 6$  ways that four objects can be arranged in a cycle.  $\diamond$

**Proposition 14.5.3.** The cycle-index polynomial of  $\text{Aut}_E(K_4)$  is

$$\mathcal{Z}_{\text{Aut}_E(K_4)}(z_1, z_2, z_3, z_4) = \frac{1}{24}(z_1^6 + 9z_1^2z_2^2 + 8z_3^2 + 6z_2z_4)$$

**Proof:** The size of the cycle to which an edge belongs is determined by the cycles to which its endpoints belong. Thus, for every automorphism  $\pi \in \text{Aut}_E(K_n)$ , the cycle structure  $\zeta(\pi_E)$  of the edge-permutation is determined by the cycle structure  $\zeta(\pi_V)$  of the vertex-permutation.

*Case 1.* If  $\zeta(\pi_V) = z_1^4$ , then  $\zeta(\pi_E) = z_1^6$ .

*Justification:* If both endpoints of an edge  $e$  are in a 1-cycle of the vertex-permutation, then they are both fixed points. In a simple graph, the corresponding edge-permutation must map that edge to itself.

*Case 2.* If  $\zeta(\pi_V) = z_1^2 z_2$ , then  $\zeta(\pi_E) = z_1^2 z_2^2$ .

*Justification:* An edge of  $K_4$  is mapped to itself if both its endpoints are in a 2-cycle or if each endpoint is in a 1-cycle. Thus, two edges of  $K_4$  are fixed by  $\pi_V$ . Each of the other four edges has one endpoint in a 1-cycle, which is fixed by  $\pi_V$ , and the other in a 2-cycle of  $\pi_V$ , which is mapped by  $\pi_V$  to the other vertex in that 2-cycle. It follows that such an edge lies in a 2-cycle of  $\pi_E$ .

*Case 3.* If  $\zeta(\pi_V) = z_1 z_3$ , then  $\zeta(\pi_E) = z_3^2$ .

*Justification:* The three edges of  $K_4$  that have both their ends in the 3-cycle of  $\pi_V$  lie in a 3-cycle of  $\pi_E$ . The three edges of  $K_4$  that have one endpoint in a 1-cycle of  $\pi_V$  and the other endpoint in a 3-cycle all lie in another 3-cycle of  $\pi_E$ .

*Case 4.* If  $\zeta(\pi_V) = z_2^2$ , then  $\zeta(\pi_E) = z_1^2 z_2^2$ .

*Justification:* The two edges that have both endpoints in the same 2-cycle of  $\pi_V$  are both fixed by  $\pi_E$ . If an edge has one endpoint in one 2-cycle of  $\pi_V$  and the other endpoint in another 2-cycle of  $\pi_V$ , then that edge lies in a 2-cycle of  $\pi_E$  with the edge whose respective endpoints are the other vertices of those 2-cycles of  $\pi_V$ .

*Case 5.* If  $\zeta(\pi_V) = z_4$ , then  $\zeta(\pi_E) = z_2 z_4$ .

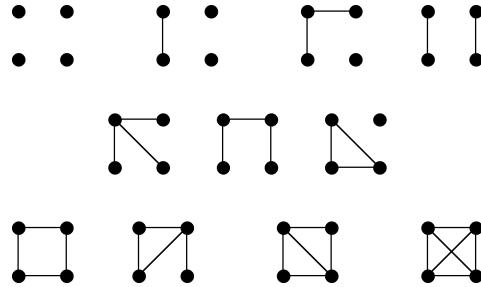
*Justification:* The four edges whose endpoints are consecutive vertices in the 4-cycle of  $\pi_V$  form a cycle of  $\pi_E$ . The two edges whose endpoints are spaced 2 apart in the 4-cycle of  $\pi_V$  form a 2-cycle of  $\pi_E$ .  $\diamond$

**Corollary 14.5.4.** *There are exactly 11 isomorphism types of simple graph with 4 vertices.*

**Proof:** Using Proposition 14.5.3, we have

$$\mathcal{Z}_{\text{Aut}_E(K_4)}(2, 2, 2, 2) = \frac{1}{24}(2^6 + 9 \cdot 2^2 \cdot 2^2 + 8 \cdot 2^2 + 6 \cdot 2 \cdot 2) = 11$$

The 11 graphs promised by this calculation are shown in Figure 14.5.5.  $\diamond$



**Figure 14.5.5** The 11 simple 4-vertex graphs.

**Remark:** The full set of  $(\leq 2)$ -colorings of  $E_{K_4}$  has cardinality 64 (each of the six edges can be colored *present* or *absent*), and each of the 11 graphs in Figure 14.5.5 represents an isomorphism class. For instance, the second graph in the top row represents a class with six graphs, since there are six ways to choose the two endpoints of the *present* edge. The sizes of these 11 isomorphism classes must, of course, sum to 64 (see Exercises).

### Simple Graphs with 5 Vertices

**Proposition 14.5.5.** *There are exactly 34 isomorphism types of simple graph with 5 vertices.*

**Proof:** By using the same approach as in the calculation of the number of 4-vertex simple graphs, it can be shown that

$$\mathcal{Z}_{\text{Aut}_V(K_5)}(z_1, \dots, z_5) = \frac{1}{120}(z_1^5 + 10z_1^3z_2 + 15z_1z_2^2 + 20z_1^2z_3 + 30z_1z_4 + 20z_2z_3 + 24z_5)$$

Therefore,

$$\mathcal{Z}_{\text{Aut}_E(K_5)}(z_1, \dots, z_6) = \frac{1}{120}(z_1^{10} + 10z_1^4z_2^3 + 15z_1^2z_2^4 + 20z_1z_3^3 + 30z_2z_4^2 + 20z_1z_3z_6 + 24z_5^2)$$

and, accordingly,

$$\begin{aligned} \mathcal{Z}_{\text{Aut}_E(K_5)}(2, \dots, 2) &= \\ \frac{1}{120}(2^{10} + 10 \cdot 2^7 + 15 \cdot 2^6 + 20 \cdot 2^4 + 30 \cdot 2^3 + 20 \cdot 2^3 + 24 \cdot 2^2) &= 34 \quad \diamond \end{aligned}$$

**Remark:** The same process can be repeated for any number of vertices.

### EXERCISES for Section 14.5

*In Exercises 14.5.1 through 14.5.7, do the following:*

- a. Use Theorem 14.4.2 to count the number of ways to vertex-color the graph with a set of two colors, and confirm this by drawings and elementary counting methods;
- b. Use Theorem 14.4.2 to count the number of ways to vertex-color the graph with a set of three colors, and confirm this by drawings and elementary counting methods.

14.5.1 The path graph  $P_4$ .

14.5.2 The path graph  $P_5$ .

14.5.3<sup>s</sup> The cycle graph  $C_5$ .

14.5.4 The cycle graph  $C_6$ .

14.5.5 The complete bipartite graph  $K_{2,3}$ .

14.5.6 The wheel graph  $W_5$ .

14.5.7 The graph  $K_5 - K_2$ .

*In Exercises 14.5.8 through 14.5.14, do the following:*

- a. Use Theorem 14.4.2 to count the number of ways to edge-color the graph with a set of two colors, and confirm this by drawings and elementary counting methods;
- b. Use Theorem 14.4.2 to count the number of ways to edge-color the graph with a set of three colors, and confirm this by drawings and elementary counting methods.

14.5.8 The path graph  $P_4$ .

14.5.9 The path graph  $P_5$ .

14.5.10<sup>s</sup> The cycle graph  $C_5$ .

- 14.5.11 The cycle graph  $C_6$ .
- 14.5.12 The complete bipartite graph  $K_{2,3}$ .
- 14.5.13 The wheel graph  $W_5$ .
- 14.5.14 The graph  $K_5 - K_2$ .
- 14.5.15 In the bowtie graph  $BT$ , do the following.
- Draw the white-black edge-colorings, using at most two black edges.
  - Draw the white-black edge-colorings, using exactly three black edges.
  - Add twice the number from part (a) to the number from part (b). Compare your answer with the one obtained in Example 14.5.3.
- 14.5.16 For each graph in Figure 14.5.5, state how many graphs are in its isomorphism class in the *present-absent* coloring of  $K_4$ .
- 14.5.17 Apply the algebraic enumeration techniques of this section to counting the number of isomorphism types of simple graphs with three vertices.
- 14.5.18 Draw all the isomorphism types of simple graphs with three vertices.
- 14.5.19 Draw all the isomorphism types of 3-vertex graphs with no self-loops and with at most two edges between any two vertices.
- 14.5.20<sup>s</sup> Substitute the number 3 into  $\mathcal{Z}_{Aut_E(K_3)}$  and evaluate. Why should this agree with the result from Exercise 14.5.19?
- 14.5.21 Determine the number of isomorphism types of simple graphs with six vertices.

## 14.6 PÓLYA-BURNSIDE ENUMERATION

This section describes Pólya's ingenious extension of the enumeration strategy presented in §14.3, §14.4, and §14.5. With this extension, we are not only able to count the number of coloring classes, but we can also categorize those classes according to the number of objects that are assigned each color. For the application to counting the non-isomorphism types of the  $n$ -vertex simple graphs, we can determine the number of non-isomorphism types having each possible number of edges.

### Inventory of the Coloring Classes

**DEFINITION:** Let  $\mathcal{P} = [P : Y]$  be a permutation group acting on a set  $Y$  of  $n$  objects. The **inventory** of the coloring classes,  $col_k(Y)_{\mathcal{P}}$ , indicates for each possible combination of how many times each color is used, the number of coloring classes that use that combination of colors.

**Example 14.6.1:** According to Figure 14.2.5, the inventory of coloring classes of vertex- $(\leq 2)$ -colorings of the graph  $K_4 - K_2$  is given by

1	4 white	0 black
2	3 white	1 black
3	2 white	2 black
2	1 white	3 black
1	0 white	4 black

**Example 14.6.2:** According to Figure 14.2.6, the inventory of coloring classes of edge- $(\leq 2)$ -colorings of the graph  $K_4 - K_2$  is

1	5	light	0	dark
2	4	light	1	dark
4	3	light	2	dark
4	2	light	3	dark
2	1	light	4	dark
1	0	light	5	dark

### Pólya Substitution

George Pólya ([Po37]) sharpened the application of Theorem 14.4.2 to enumeration by devising a special way of substituting a  $k$ -variate polynomial (instead of the number  $k$ ) into the cycle-index polynomial.

**DEFINITION:** Let  $\mathcal{Z}_{\mathcal{P}}(z_1, \dots, z_n)$  be the cycle-index polynomial for a permutation group  $\mathcal{P} = [P : Y]$  on a set  $Y$  of  $n$  objects. The **Pólya substitute of order  $k$**  for the cycle-index variable  $z_j$  is the  $k$ -variate polynomial

$$x_1^j + x_2^j + \cdots + x_k^j$$

**DEFINITION:** Let  $\mathcal{Z}_{\mathcal{P}}(z_1, \dots, z_n)$  be the cycle-index polynomial for a permutation group  $\mathcal{P} = [P : Y]$  on a set  $Y$  of  $n$  objects. The **Pólya-inventory polynomial of order  $k$**  is the polynomial obtained by replacing each cycle-index variable  $z_j$  by its Pólya substitute of order  $k$ . It is denoted  $\mathcal{Z}_{\mathcal{P}}(x_1 + \cdots + x_k)$ .

### Pólya's Enumeration Theorem

The culminating result of this chapter, the *Pólya Enumeration Theorem*, provides an elegant method for determining the number of non-equivalent colorings for a specified number of occurrences of each color. For a proof, see, for example, [Bo00] or [Br04b].

**Theorem 14.6.1 [Pólya Enumeration Theorem].** Let  $\mathcal{P} = [P : Y]$  be a permutation group on a set  $Y$  of  $n$  objects, and let  $\mathcal{Z}_{\mathcal{P}}(x_1 + \cdots + x_k)$  be the Pólya-inventory polynomial of order  $k$  for the coloring classes of  $\text{Col}_k(\mathcal{P})$ . Then the coefficient of the term  $x^{j_1}x^{j_2}\cdots x^{j_k}$  in the expansion of  $\mathcal{Z}_{\mathcal{P}}(x_1 + \cdots + x_k)$  is the number of coloring classes that use color  $i$  exactly  $j_i$  times,  $i = 1, 2, \dots, k$ .  $\diamond$

**Example 14.6.3:** Returning to Example 14.4.4, the cycle-index polynomial for  $\text{Aut}_V(K_4 - K_2)$  is

$$\mathcal{Z}_{\text{Aut}_V(K_4 - K_2)}(z_1, z_2) = \frac{1}{4}(z_1^4 + 2z_1^2z_2 + z_2^2)$$

Thus, the Pólya-inventory polynomial of order 2 is

$$\begin{aligned} \mathcal{Z}_{\text{Aut}_V(K_4 - K_2)}(x_1 + x_2) &= \frac{1}{4}((x_1 + x_2)^4 + 2(x_1 + x_2)^2(x_1^2 + x_2^2) + (x_1^2 + x_2^2)^2) \\ &= \frac{1}{4}(x_1^4 + 4x_1^3x_2 + 6x_1^2x_2^2 + 4x_1x_2^3 + x_2^4 \\ &\quad + 2x_1^4 + 4x_1^3x_2 + 4x_1^2x_2^2 + 4x_1x_2^3 + 2x_2^4 \\ &\quad + x_1^4 + 2x_1^2x_2^2 + x_2^4) \\ &= x_1^4 + 2x_1^3x_2 + 3x_1^2x_2^2 + 2x_1x_2^3 + x_2^4 \end{aligned}$$

In this Pólya-inventory polynomial, the variable  $x_1$  stands for the color white and  $x_2$  for the color black. Thus, its coefficients agree exactly with the inventory given in Example 14.6.1.

**Example 14.6.4:** Returning to Example 14.4.5, the cycle-index polynomial for  $\text{Aut}_E(K_4 - K_2)$  is

$$\mathcal{Z}_{\text{Aut}_E(K_4 - K_2)}(z_1, z_2) = \frac{1}{4}(z_1^5 + 3z_1z_2^2)$$

Thus, the Pólya-inventory polynomial of order 2 is

$$\begin{aligned} \mathcal{Z}_{\text{Aut}_E(K_4 - K_2)}(x_1 + x_2) &= \frac{1}{4}((x_1 + x_2)^5 + 3(x_1 + x_2)(x_1^2 + x_2^2)^2) \\ &= \frac{1}{4}(x_1^5 + 5x_1^4x_2 + 10x_1^3x_2^2 + 10x_1^2x_2^3 + 5x_1x_2^4 + x_2^5 \\ &\quad + 3x_1^5 + 3x_1^4x_2 + 6x_1^3x_2^2 + 6x_1^2x_2^3 + 3x_1x_2^4 + 3x_2^5) \\ &= x_1^5 + 2x_1^4x_2 + 4x_1^3x_2^2 + 4x_1^2x_2^3 + 2x_1x_2^4 + x_2^5 \end{aligned}$$

This represents the detailed inventory given in Example 14.6.2.

### Inventory of n-Vertex Simple Graphs

The Pólya-inventory polynomial of order 2 for  $\text{Aut}_E(K_n)$  gives an inventory of the  $n$ -vertex simple graphs according to their number of edges. A term of the form  $c_k x_1^j x_2^k$  means that there are  $c_k$  configurations with  $k$  edges present and  $j$  edges absent.

**Example 14.6.5:** According to Proposition 14.5.3, the cycle-index polynomial for  $\text{Aut}_E(K_4)$  is

$$\mathcal{Z}_{\text{Aut}_E(K_4)} = \frac{1}{24}(z_1^6 + 9z_1^2z_2^2 + 8z_3^2 + 6z_2z_4)$$

This leads to the following calculation of  $\mathcal{Z}_{\text{Aut}_E(K_4)}(x_1 + x_2)$ .

$$\begin{aligned} &\frac{1}{24}((x_1 + x_2)^6 + 9(x_1 + x_2)^2(x_1^2 + x_2^2)^2 + 8(x_1^3 + x_2^3)^2 + 6(x_1^2 + x_2^2)(x_1^4 + x_2^4)) \\ &= \frac{1}{24}(x_1^6 + 6x_1^5x_2 + 15x_1^4x_2^2 + 20x_1^3x_2^3 + 15x_1^2x_2^4 + 6x_1x_2^5 + x_2^6 \\ &\quad + 9x_1^6 + 18x_1^5x_2 + 27x_1^4x_2^2 + 36x_1^3x_2^3 + 27x_1^2x_2^4 + 18x_1x_2^5 + 9x_2^6 \\ &\quad + 8x_1^6 + 16x_1^5x_2^3 + 8x_2^6 \\ &\quad + 6x_1^6 + 6x_1^4x_2^2 + 6x_1^2x_2^4 + 6x_2^6) \\ &= x_1^6 + x_1^5x_2 + 2x_1^4x_2^2 + 3x_1^3x_2^3 + 2x_1^2x_2^4 + x_1x_2^5 + x_2^6 \end{aligned}$$

In complete agreement with Figure 14.5.5, this means that the inventory of 4-vertex simple graphs is as follows:

- 1 graph with 0 edges
- 1 graph with 1 edges
- 2 graphs with 2 edges
- 3 graphs with 3 edges
- 2 graphs with 4 edges
- 1 graph with 5 edges
- 1 graph with 6 edges

**EXERCISES for Section 14.6**

- 14.6.1 Construct the Pólya-inventory polynomial for 3-vertex simple graphs.
- 14.6.2 Draw all the different isomorphism types of 5-vertex simple graphs with three or fewer edges.
- 14.6.3 Draw all the different isomorphism types of 5-vertex simple graphs with four edges.
- 14.6.4<sup>s</sup> Draw all the different isomorphism types of 5-vertex simple graphs with five edges.
- 14.6.5 Add the number of graphs in Exercise 14.6.4 to twice the number of graphs in Exercise 14.6.3 and twice the number of graphs in Exercise 14.6.2. Why should this number equal 34?
- 14.6.6<sup>s</sup> Construct the Pólya-inventory polynomial for 5-vertex simple graphs.
- 14.6.7 Construct the Pólya-inventory polynomial for 6-vertex simple graphs.
- 14.6.8<sup>s</sup> Substitute the Pólya-inventory polynomial of order 3 into  $\mathcal{Z}_{\text{Aut}_E(K_3)}$ , and compare the result with your answer for Exercise 14.5.20.

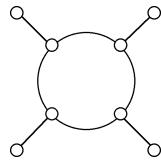
**14.7 SUPPLEMENTARY EXERCISES**

*In Exercises 14.7.1 through 14.7.10, do the following for the given graph:*

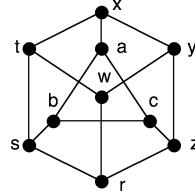
- a. List all the vertex automorphisms.
- b. Write the cycle index of the vertex automorphism group.
- c. Use Burnside enumeration to count the essentially different ways to color the vertices with at most two colors.
- d. Calculate a Pólya inventory of the colorings from part (c).
- e. Draw all the different ways from part (c).
- f. List all the edge automorphisms.
- g. Write the cycle index of the edge automorphism group.
- h. Use Burnside enumeration to count the essentially different ways to color the edge with at most two colors.
- i. Calculate a Pólya inventory of the colorings from part (c).
- j. Draw all the different ways from part (h).

- 14.7.1 The cartesian product  $C_3 \times P_3$ .
- 14.7.2 The graph obtained by joining a 5-cycle to  $K_2$ .
- 14.7.3 The result of a vertex amalgamation of two copies of  $K_4$ .
- 14.7.4 The result of an edge-amalgamation of two copies of  $K_4$ .
- 14.7.5 The Möbius ladder  $ML_4$ .
- 14.7.6 The cartesian product  $C_3 \times C_3$ .

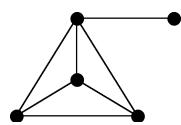
14.7.7



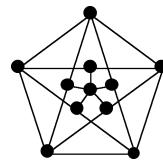
14.7.8



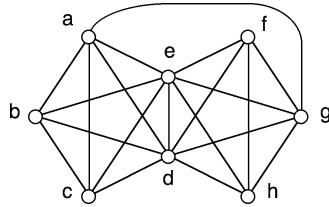
14.7.9



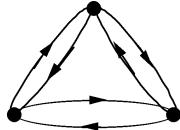
14.7.10



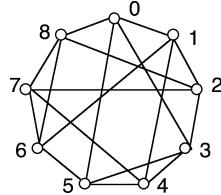
14.7.11 Write the vertex orbits and edge orbits of the graph below. (Use the notation  $uv$  for an edge between vertices  $u$  and  $v$ .) Calculate the cycle index of the vertex automorphism group.



14.7.12 List all six automorphisms that preserve directions on the arcs of the digraph below, and write the cycle index for this automorphism group. Use it to count the possible isomorphism types of simple digraphs on three vertices. Verify your answer with drawings.



14.7.13 a. In the graph below, find the only vertex that does not lie on a 3-cycle, and thus, lies in an orbit by itself. b. Partition the vertices according to distance from this special vertex. Suggestion: redraw the graph. c. List the vertex orbits.



## GLOSSARY

**automorphism  $\pi$  of a graph  $G$ :** an isomorphism from the graph  $G$  to itself.

**automorphism group  $\text{Aut}(G)$  of a graph  $G$ :** the permutation group of all automorphisms of the graph  $G$ .

**closed collection of permutations:** a collection  $P$  of permutations on the same set of objects, such that for every pair  $\pi_1, \pi_2 \in P$ , the composition  $\pi_1\pi_2$  is in  $P$ .

**coloring of a set  $Y$ :** a mapping  $f$  from  $Y$  onto a set  $\{1, 2, \dots, k\}$  of integers, in which the number  $f(y)$  is called the *color* of  $y$ .

**$(\leq k)$ -coloring of a set  $Y$ :** a coloring that uses  $k$  or fewer colors. The set of all  $(\leq k)$ -colorings of  $Y$  is denoted  $Col_k(Y)$ .

**coloring class:** for a set with a coloring, a subset of all objects of like color.

**cycle-index (polynomial) of a permutation group  $\mathcal{P}$ :** the polynomial

$$\mathcal{Z}_{\mathcal{P}}(z_1, \dots, z_n) = \frac{1}{|P|} \sum_{\pi \in P} \zeta(\pi)$$

**cycle structure of a permutation:** the number of cycles of each length in its disjoint-cycle form.

**cycle-structure monomial of a permutation  $\pi$ :** the multivariate monomial  $\zeta(\pi) = \prod_{k=1}^n z_j^{r_j}$  where  $z_k$  is a formal variable and where  $r_k$  is the number of  $k$ -cycles in the disjoint cycle form of  $\pi$ .

**edge-automorphism group  $Aut_E(G)$  of a graph  $G$ :** the permutation group whose object set is  $E_G$  and whose permutations are the edge functions of the automorphisms of the graph  $G$ .

**$\mathcal{P}$ -equivalent colorings of a set  $Y$  under a permutation group  $\mathcal{P} = [P : Y]$ :** colorings  $f$  and  $g$  for which there is a permutation  $\pi \in P$  such that  $g = f\pi$ , that is, for every object  $y \in Y$ , the color  $g(y)$  is the same as the color  $f(\pi(y))$ . The set of  $\mathcal{P}$ -equivalence classes of  $Col_k(Y)$  is denoted  $\{Col_k(Y)\}_P$ .

**equivalent edge-colorings on a graph  $G$ :** edge-colorings  $c_1$  and  $c_2$  on  $G$  that are  $Aut_E(G)$ -equivalent.

**equivalent vertex-colorings on a graph  $G$ :** vertex-colorings  $c_1$  and  $c_2$  on  $G$  that are  $Aut_V(G)$ -equivalent.

**fixed-point set of a permutation  $\pi : Y \rightarrow Y$ :** the subset

$$Fix(\pi) = \{y \in Y \mid \pi(y) = y\}$$

**full symmetric group  $\Sigma_Y$ :** the collection of *all* permutations on a set  $Y$ .

**induced permutation  $\pi_C$ :** given a permutation  $\pi \in \mathcal{P} = [P : Y]$ , the rule  $c \mapsto c\pi$  that permutes the set  $Col_k(Y)$  of colorings of  $Y$ .

**induced permutation group  $\mathcal{P}_k$ :** given a permutation group  $\mathcal{P} = [P : Y]$ , the group of induced permutations on  $Col_k(Y)$ .

**inventory of the coloring classes:** for a permutation group acting on a set  $Y$  of  $n$  objects, specification for each possible combination of how many times each color is used, the number of coloring classes that use that combination of colors.

**involution:** a permutation  $\pi$  such that  $\pi = \pi^{-1}$ .

**monomial:** a polynomial with only one term.

**orbit of an object  $y \in Y$  under a permutation group  $P$ :** the set  $\{\pi(y) \mid \pi \in P\}$ .

**permutation group  $[P : Y]$ :** a mathematical structure such that  $P$  is a nonempty closed collection of permutations on the same finite set of objects  $Y$ .

**Pólya-counting polynomial:** the polynomial that gives detailed inventories, obtained by substituting the Pólya substitutes into the cycle-index polynomial.

**Pólya substitute of order  $k$**  for the cycle-index variable  $z_j$ : the  $k$ -variate polynomial  $x_1^j + x_2^j + \cdots + x_k^j$  that replaces the cycle-index variable  $z_j$  in the cycle-index polynomial  $\mathcal{Z}_{\mathcal{P}}(z_1, \dots, z_n)$ .

**stabilizer of an object  $y$** : in a permutation group  $\mathcal{P} = [P : Y]$ , the subgroup  $Stab(y) = \{\pi \in P \mid \pi(y) = y\}$ .

**symmetric group  $\Sigma_Y$  on a set  $Y$** : the collection of *all* permutations on  $Y$ .

**vertex-automorphism group  $Aut_V(G)$  of a graph  $G$** : the permutation group whose object set is  $V_G$  and whose permutations are the vertex functions of the automorphisms of the graph  $G$ .

# Chapter 15

---

## ALGEBRAIC SPECIFICATION OF GRAPHS

- 15.1 Cyclic Voltages**
  - 15.2 Cayley Graphs and Regular Voltages**
  - 15.3 Permutation Voltages**
  - 15.4 Symmetric Graphs and Parallel Architectures**
  - 15.5 Interconnection-Network Performance**
- 

### INTRODUCTION

Exploiting the algebraic symmetries of a large network can be crucial to realizing its most efficient computational and communications capabilities. In a specification by incidence table of a graph with, say, a million vertices, algebraic symmetries and other global features can be obscured by the sheer mass of details of the specification. Using a *voltage graph* to specify a network puts the symmetries into focus. A voltage graph is a vertex-labeled digraph whose edges are labeled by algebraic elements. If  $G$  is a large graph with  $n$ -fold symmetry, then the vertex-set and edge-set of the voltage graph used to specify  $G$  are smaller than those of  $G$  by a factor of  $n$ . Many interesting large graphs are specifiable by a voltage graph with only one vertex. The general idea is that incipient patterns designed into the voltage graph are developed and replicated throughout the larger graph it specifies. Voltage graphs were first developed for application to the construction of surface imbeddings of graphs with geometric symmetry. Application of voltage graphs to algebraic specification of parallel-processor networks is more recent.

Some algebraic results used in this chapter are summarized in Appendix A.4.

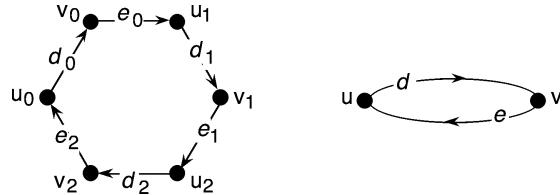
## 15.1 CYCLIC VOLTAGES

Extremely useful ideas commonly evolve from prototype solutions, in mathematics as well as in engineering. The prototype form of voltage graph described in this section is adapted to cyclic symmetry. In the two subsequent sections, more general forms of voltage graphs are developed.

### Symmetry and Specification Reduction

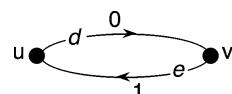
To understand how voltage graphs might help with a large problem, it helps to begin with a small example that illustrates how a graph can be specified by a smaller graph. Definitions of the terminology and precise descriptions of the notational conventions follow this example. The underlying idea of a voltage graph is that it is used to replicate patterns in the graph that it specifies.

**Example 15.1.1:** The vertices and edges of the directed 6-cycle graph on the left in Figure 15.1.1 have subscripted labels, whose subscripts are integers modulo 3. One pattern in this labeled 6-cycle is that the  $u$ -labels and the  $v$ -labels alternate. Another pattern is that the  $d$ -labels and the  $e$ -labels on the arcs alternate. These two alternation patterns are represented by the directed 2-cycle on the right side of Figure 15.1.1.



**Figure 15.1.1** A directed 6-cycle and a directed 2-cycle.

In the 2-cycle of Figure 15.1.1, the subscript patterns of the 6-cycle remain unrepresented. Observe that the subscript on the head vertex of each  $d$ -arc of the 6-cycle is the same as the subscript on the tail vertex, and that the subscript on the head of each  $e$ -arc increments the subscript on the tail vertex by 1 (mod 3). Labeling the 2-cycle arcs  $d$  and  $e$  by the *voltages* 0 (mod 3) and 1 (mod 3), respectively, as shown in Figure 15.1.2, is used to specify these subscript increments.



**Figure 15.1.2** A directed 2-cycle with voltages mod 3 on its arcs.

Using a smaller digraph to specify the directed 6-cycle in Example 15.1.1 is facilitated by these three features of the labeling of the directed 6-cycle:

- (1) All the subscripts on vertex- and edge-names in the 6-cycle are selected from the same algebraic structure; in particular, they are all integers modulo 3.
- (2) The number of  $u$ -vertices, the number of  $v$ -vertices, the number of  $d$ -edges, and the number of  $e$ -edges are all exactly the same; in particular, there are three of each.
- (3) Every  $d$ -arc in the 6-cycle has a  $u$ -tail and a  $v$ -head, and the increment from the subscript of the tail vertex to the subscript of the head vertex is the same for all  $d$ -arcs

in the 6-cycle. The  $e$ -arcs in the 6-cycle also follow such patterns of tail-commonality, head-commonality, and subscript-increment commonality.

### Voltage Assignments and Their Meaning

When applying *voltage graphs* to the problem of writing a convenient-sized specification for a large graph or digraph, one starts by imagining a system of vertex labels and edge labels (for the large graph object) that has the three kinds of features described immediately above for Example 15.1.1, and one then tries to design the appropriate voltage graph. In contrast, the definitions start with a *voltage graph* and use it to give details of the large graph that it specifies.

**DEFINITION:** A  $\mathbb{Z}_n$ -voltage on an arc of a digraph  $G$  is a label by a number  $j(\bmod n)$ .

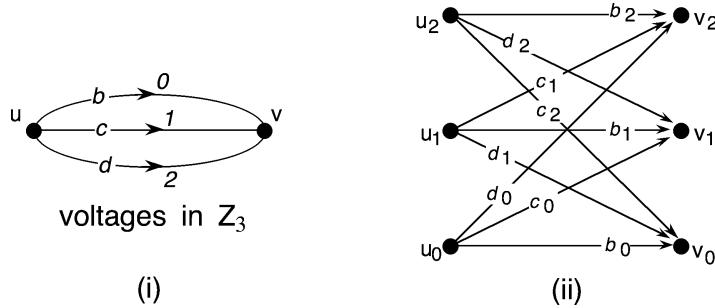
**DEFINITION:** A  $\mathbb{Z}_n$ -voltage assignment on a digraph  $G$  is a function  $\alpha$  that labels every arc  $e \in E_G$  with a  $\mathbb{Z}_n$ -voltage.

**DEFINITION:** A  $\mathbb{Z}_n$ -voltage graph is a pair  $\langle G, \alpha : E_G \rightarrow \mathbb{Z}_n \rangle$  such that  $G$  is a digraph and  $\alpha$  is a  $\mathbb{Z}_n$ -voltage assignment on  $G$ . A  $\mathbb{Z}_n$ -voltage graph is also called a **cyclic-voltage graph**.

**DEFINITION:** The **voltage group** of a voltage graph  $\langle G, \alpha : E_G \rightarrow \mathbb{Z}_n \rangle$  is the group  $\mathbb{Z}_n$  from which the voltages are assigned.

**Example 15.1.2:** The labeled 2-cycle in Figure 15.1.2 is a  $\mathbb{Z}_3$ -voltage graph. The 6-cycle in Figure 15.1.1 is the digraph it specifies.

**Example 15.1.3:** Another  $\mathbb{Z}_3$ -voltage graph is illustrated in Figure 15.1.3(i). Observe that if the drawing did not say that the voltages are in  $\mathbb{Z}_3$ , there would be no way of knowing that they were not from some other group. When a voltage graph is specified by a drawing, the voltage group is indicated in the drawing itself, as shown. Part (ii) of the figure depicts the digraph specified by part (i).



**Figure 15.1.3** (i) A  $\mathbb{Z}_3$ -voltage graph; (ii) the digraph it specifies.

**DEFINITION:** The **covering digraph** of the voltage graph  $\langle G, \alpha : E_G \rightarrow \mathbb{Z}_n \rangle$  is the graph  $G^\alpha = (V^\alpha, E^\alpha)$ , with

$$V^\alpha = \{v_j \mid v \in V_G \text{ and } j \in \mathbb{Z}_n\}$$

$$E^\alpha = \{e_j \mid e \in E_G \text{ and } j \in \mathbb{Z}_n\}$$

such that whenever an arc  $e$  in a voltage graph is directed from vertex  $u$  to vertex  $v$ , the arc  $e_j$  is directed from vertex  $u_j$  to vertex  $v_{j+\alpha(e)(\bmod n)}$ .

**DEFINITION:** The **covering graph** of the voltage graph  $\langle G, \alpha : E_G \rightarrow \mathbb{Z}_n \rangle$  is the *underlying graph* of the covering digraph  $G^\alpha$ . It is also denoted  $G^\alpha$ .

**NOTATION:** Formally,  $V^\alpha$  can be regarded as the cartesian product  $V \times \mathbb{Z}_n$ , and  $E^\alpha$  as the cartesian product  $E \times \mathbb{Z}_n$ . Using pairs instead of subscripts tends to look cluttered on a drawing, but is occasionally convenient, for instance, when the vertices and edges of the voltage graph itself are already subscripted.

**Example 15.1.4:** For the voltage graph of Example 15.1.3, the covering graph vertex-set and edge-set are

$$V^\alpha = \{u_0, u_1, u_2, v_0, v_1, v_2\} \quad \text{and} \quad E^\alpha = \{b_0, b_1, b_2, c_0, c_1, c_2, d_0, d_1, d_2\}$$

Reading the incidence table (§1.1) of the formal specification of a graph is not as helpful in gaining insight as examining Figure 15.1.3, which reveals that the covering graph is the complete bipartite graph  $K_{3,3}$ .

### Vertex Fibers and Edge Fibers

The vertex-set of the covering graph has a natural partition, in which the vertices that arise from a single vertex of the voltage graph are grouped into a cell. Likewise, the edge-set of the covering graph has such a partition. Convenient names and notation for these cells are borrowed from algebraic topology.

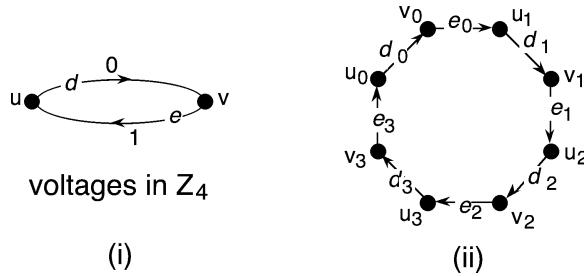
**DEFINITION:** Let  $\langle G, \alpha : E_G \rightarrow \mathbb{Z}_n \rangle$  be a voltage graph. For each vertex  $v \in V_G$ , the vertex set  $\{v_j \mid j \in \mathbb{Z}_n\}$  in the covering graph  $G^\alpha$  is called the **(vertex) fiber** over  $v$  and is denoted  $\tilde{v}$ .

**DEFINITION:** Let  $\langle G, \alpha : E_G \rightarrow \mathbb{Z}_n \rangle$  be a voltage graph. For each edge  $e \in E_G$ , the edge set  $\{e_j \mid j \in \mathbb{Z}_n\}$  in the covering graph  $G^\alpha$  is called the **(edge) fiber** over  $e$  and is denoted  $\tilde{e}$ .

**Example 15.1.5:** Suppose that the voltages in the voltage graph of Figure 15.1.2 are regarded as integers modulo 4, rather than modulo 3. Then the covering vertex-set and covering edge-set are

$$V^\alpha = \{u_0, u_1, u_2, u_3, v_0, v_1, v_2, v_3\} \quad \text{and} \quad E^\alpha = \{d_0, d_1, d_2, d_3, e_0, e_1, e_2, e_3\}$$

and the covering digraph is an 8-cycle, as shown in Figure 15.1.4.



**Figure 15.1.4** (i) A 2-cycle with  $\mathbb{Z}_4$ -voltages; (ii) the 8-cycle it specifies.

Thus, the vertex fibers are

$$\tilde{u} = \{u_0, u_1, u_2, u_3\} \quad \text{and} \quad \tilde{v} = \{v_0, v_1, v_2, v_3\}$$

and the edge fibers are

$$\tilde{d} = \{d_0, d_1, d_2, d_3\} \quad \text{and} \quad \tilde{e} = \{e_0, e_1, e_2, e_3\}$$

If the voltages on the 2-cycle were regarded as integers modulo 5, then the covering digraph would be a 10-cycle. Moreover, each vertex fiber and each edge fiber would have five elements.

**NOTATION:** When it is clear what voltage group is intended, the voltage graph may be denoted  $\langle G, \alpha \rangle$ .

**Proposition 15.1.1.** *Let  $\langle G, \alpha \rangle$  be a cyclic-voltage graph with voltages mod  $n$ . Then there are  $n$  times as many vertices and edges in the covering digraph  $G^\alpha$  as in  $G$ .*

**Proof:** There are  $n$  vertices  $v_j$  and  $n$  edges  $e_j$ , respectively, in each vertex fiber  $\tilde{v}$  and edge fiber  $\tilde{e}$  of  $G^\alpha$  for each vertex  $v \in V_G$  and for each edge  $e \in E_G$ , respectively.  $\diamond$

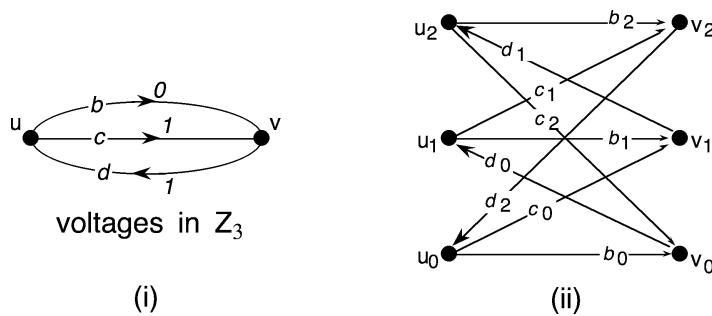
**Proposition 15.1.2.** *Let  $\langle G, \alpha \rangle$  be a cyclic-voltage graph, and let  $v$  be any vertex of  $G$ . Then the outdegree and indegree of each vertex  $v_j$  in the fiber  $\tilde{v}$  are equal to the outdegree and indegree, respectively, of vertex  $v$ .*

**Proof:** For each arc  $e$  directed from  $v$  in voltage graph  $G$ , there is an arc  $e_j$  directed from  $v_j$  in the covering digraph, and no other arcs originate at  $v_j$ . For each arc  $d$  terminating at  $v$ , there is an arc  $d_{j-\alpha(d)}$  terminating at  $v_j$ , and no other arcs terminate at  $v_j$ .  $\diamond$

### Artificiality of the Arc Directions

A voltage graph is defined formally to be a digraph, and the mathematical object derived from a voltage graph is also defined formally to be a digraph. However, the arc directions in this context are usually regarded simply as an ingredient of the construction (and not as “onewayness”). The result of reversing an arc direction in a voltage graph and replacing its voltage by the inverse voltage is a new voltage graph that still specifies the same undirected graph, as confirmed by the next example and the proposition that follows it.

**Example 15.1.6:** Figure 15.1.5(i) shows a  $\mathbb{Z}_3$ -voltage graph obtained from the voltage graph of Figure 15.1.3(i) by reversing the direction on edge  $d$  and replacing the voltage 2 (mod 3) by its additive inverse, namely, the voltage 1 (mod 3). Figure 15.1.5(ii) shows the digraph that the modified voltage graph specifies. Observe that its underlying graph is isomorphic to the underlying graph of Figure 15.1.3(ii), even though the names and directions of the arcs in the  $d$ -fiber have changed.



**Figure 15.1.5** (i) A  $\mathbb{Z}_3$ -voltage graph; (ii) the digraph it specifies.

**Proposition 15.1.3.** Let  $\langle G, \alpha \rangle$  be a  $\mathbb{Z}_n$ -voltage graph, and let  $d$  be an arc of  $G$  from  $u$  to  $v$ . Let  $(H, \beta)$  be the voltage graph obtained from  $(G, \alpha)$  by reversing the direction of arc  $d$  and changing its voltage to  $\beta(d) = n - \alpha(d)$ . Then the covering graph  $H^\beta$  is isomorphic to the covering graph  $G^\alpha$ .

**Proof:** This alteration in the voltage graph has no effect on the covering graph except for the  $d$ -fiber. Its effect on the edges of the  $d$ -fiber is to change the names and directions, without changing the specification of pairs to be joined. That is, for  $i = 0, \dots, n-1 \pmod{n}$ , an edge  $d_i$  from  $u_i$  to  $v_{i+\alpha(d)}$  is transformed into an edge  $d_{i+\alpha(d)}$  from  $v_{i+\alpha(d)}$  to  $u_{i+\alpha(d)+\beta(d)} = u_i$  that still joins the same two vertices.  $\diamond$

The following proposition and its proof further illustrate the convention of de-emphasizing edge directions in voltage graphs.

**Proposition 15.1.4.** Let  $\langle G, \alpha \rangle$  be a cyclic-voltage graph such that  $G$  is bipartite. Then the covering graph  $G^\alpha$  is bipartite.

**Proof:** Let  $(U, W)$  be the bipartition of  $G$ . Then every edge in the covering graph joins some vertex  $u_i$  with  $u \in U$  to a vertex  $w_j$  with  $w \in W$ . Thus, the vertices of the covering graph are bipartitioned, so that one part is the union of the vertex fibers over vertices in  $U$  and the other part is the union of the vertex fibers over vertices in  $W$ .  $\diamond$

**Remark:** Even when there is no need to mention arc directions in a discussion, it may still be helpful to draw them in a figure. This enables someone to verify the correctness of the drawing. In short, the figure drawn might be the covering digraph, while the discussion might call no attention to the presence of the directions.

### Voltage Graphs with Self-Loops

The same specification rules apply when there is a self-loop in the voltage graph. If  $e$  is a self-loop with endpoint  $v$  in a voltage graph, and if the voltage assigned to edge  $e$  is  $\alpha(e)$ , then the edge  $e_j$  in the covering digraph has tail  $v_j$  and head  $v_{j+\alpha(e)}$ .

**Example 15.1.7:** Figure 15.1.6 shows that adding a self-loop with voltage 1 (mod 3) at both vertices of the voltage graph of Figure 15.1.3 transforms it into a specification of the complete graph  $K_6$ .

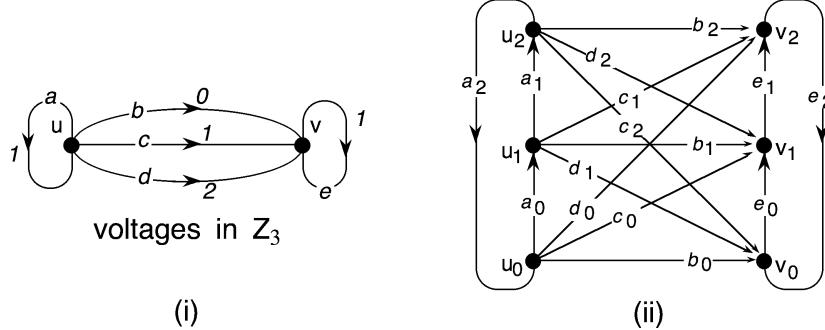
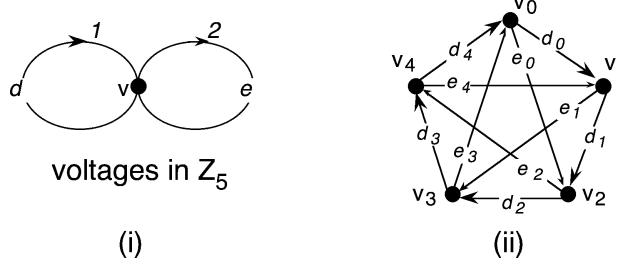


Figure 15.1.6 (i) A voltage graph for  $K_6$ ; (ii) the covering graph  $K_6$ .

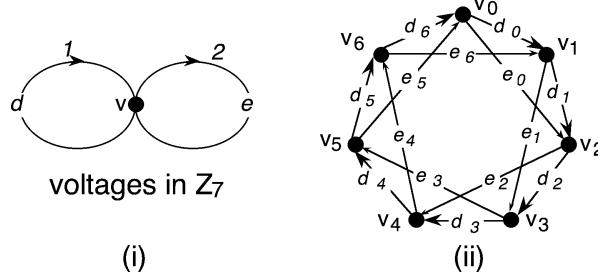
Sometimes a voltage graph has only one vertex, so that all the arcs are self-loops. When it is possible to specify a graph or digraph with a one-vertex voltage graph, this is optimal, in the sense that it is the smallest possible specification.

**Example 15.1.8:** Figure 15.1.7 shows how to specify the complete graph  $K_5 \cong \text{circ}(5 : 1, 2)$  by assigning  $\mathbb{Z}_5$ -voltages to the bouquet  $B_2$ . Observe that self-loop  $d$  has voltage 1 (mod 5), and that in traversing the 5-cycle  $d_0, d_1, d_2, d_3, d_4$  arising from edge  $d$ , the subscript increases by 1 (mod 5) at a time. Similarly, observe that self-loop  $e$  has voltage 2 (mod 5), and that in traversing the 5-cycle  $e_0, e_2, e_4, e_1, e_3$  arising from edge  $e$ , the subscript increases by 2 (mod 5) at a time.



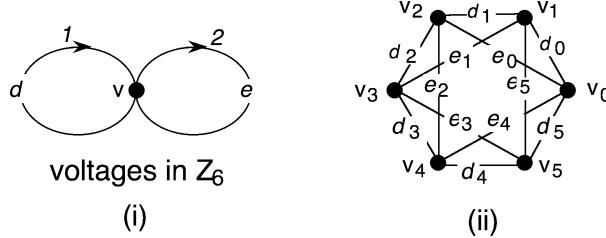
**Figure 15.1.7** (i)  $\mathbb{Z}_5$ -voltages on  $B_2$ ; (ii) covering by  $K_5 \cong \text{circ}(5 : 1, 2)$ .

**Example 15.1.9:** If the voltages of the previous example are in  $\mathbb{Z}_7$ , then the covering graph is  $K_7 - C_7 \cong \text{circ}(7 : 1, 2)$ . This time, a 7-cycle arises from edge  $d$ , and in traversing it, the subscript increases by 1 (mod 7) at a time. Similarly, a 7-cycle arises from edge  $e$ , and in its traversal, the subscript increases by 2 (mod 7) at a time.



**Figure 15.1.8** (i)  $\mathbb{Z}_7$ -voltages on  $B_2$ ; (ii) covering by  $K_7 - C_7 \cong \text{circ}(7 : 1, 2)$ .

**Example 15.1.10:** When the voltages of Example 15.1.9 are in  $\mathbb{Z}_6$ , then the covering graph is  $K_6 - 3K_2 \cong \text{circ}(6 : 1, 2)$ . This time, a 6-cycle arises from edge  $d$ , and in traversing it, the subscript increases by 1 (mod 6) at a time. However, two 3-cycles arise from edge  $e$ , and in the traversal of either of them, the subscript increases by 2 (mod 6) at a time.



**Figure 15.1.9** (i)  $\mathbb{Z}_6$ -voltages on  $B_2$ ; (ii) covering by  $K_6 - 3K_2 \cong \text{circ}(6 : 1, 2)$ .

**Proposition 15.1.5.** In a  $\mathbb{Z}_n$ -voltage graph  $\langle G, \alpha \rangle$ , let  $e$  be a self-loop at vertex  $v$  with voltage  $k$ . Then the subgraph induced on the edges  $e_0, e_1, \dots, e_{n-1}$  in the covering graph  $G^\alpha$  is the disjoint union of  $\gcd(n, k)$   $\frac{n}{\gcd(n, k)}$ -cycles.

**Proof:** This proof is a generalization of the phenomenon that occurs in Example 15.1.10. Its formal details are left as an exercise.  $\diamond$

### Circulant Graphs as Coverings of Voltage Graphs

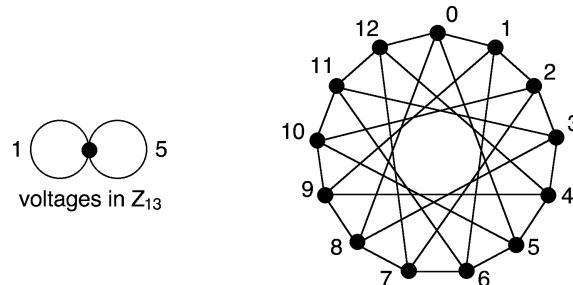
In three preceding examples, we have obtained circulant graphs as the covering graphs of bouquets with cyclic voltages. There is an easy generalization to all circulant graphs.

**TERMINOLOGY:** In discussing a subscripted variable, everyone knows that the little character string at the lower right is called a *subscript*. Unfortunately, whatever it is that has the subscript to its lower right seems to be one of those objects that has no familiar name. In order to discuss that hitherto nameless object here, we have coined the word **mainscript**.

**CONVENTION:** If a voltage graph has only one vertex, then the name of that one vertex is the mainscript of every vertex in the covering graph, so all the identifying information is in the subscript. Accordingly, we sometimes simplify our voltage graph diagrams by *suppressing the mainscripts*, which means omitting them altogether.

**Theorem 15.1.6.** Let the distinct positive integers  $x_1, \dots, x_r \bmod n$ , each less than  $\frac{n}{2}$ , be assigned as  $\mathbb{Z}_n$ -voltages to the self-loops of the bouquet  $B_r$ . Then the covering graph is isomorphic to the circulant graph  $\text{circ}(n : x_1, \dots, x_r)$ .

**Proof:** This is an immediate consequence of the definitions of circulant graph and cyclic voltage graph. It is illustrated by Figure 15.1.10, in which the mainscripts are suppressed.  $\diamond$



**Figure 15.1.10**  $\text{circ}(13 : 1, 5)$  as a covering of  $B_2$  with  $\mathbb{Z}_{13}$ -voltages.

The voltage  $n$  in the cyclic voltage group  $\mathbb{Z}_{2n}$  would cause doubled edges to appear in the covering graph. Collapsing such doubled edges, by discretion, to a single edge enhances the usefulness of voltage graphs in algebraic specification.

**DEFINITION: Bidirected-arc convention:** The pair of arcs that would otherwise join vertex  $j$  to vertex  $j + n$  and vertex  $j + n$  to vertex  $j$  in a covering digraph for a cyclic voltage graph is replaced by a single bidirected arc between  $j$  and  $j + n$ . The **underlying graph of a digraph with bidirected arcs** has a single edge for each bidirected arc.

**Example 15.1.11:** With the bidirected-arc convention in effect, the complete graph  $K_{2n} \cong \text{circ}(2n : 1, 2, \dots, n)$  is specified by a bouquet  $B_n$  with  $\mathbb{Z}_{2n}$ -voltages  $1, 2, \dots, n$  on its self-loops. Figure 15.1.11 illustrates this for  $K_6$ .

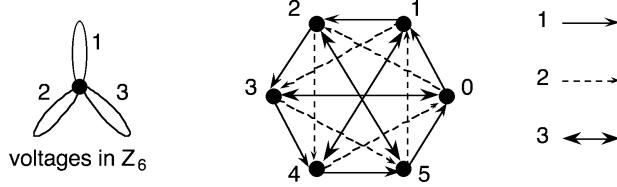


Figure 15.1.11 A  $K_6$ -digraph with bidirectional arcs.

**Remark:** A graph specified via the bidirectional arc convention is not a covering graph (in the standard sense used by algebraic topologists) of the underlying graph of the voltage graph.

### Designing Specifications of Given Graphs

In learning to use voltage graphs, the initial step is to understand what digraph is specified by an arbitrary assignment of voltages to a digraph. Ultimately, however, it is the reverse process that makes voltage graphs valuable. That is, given a graph  $G$ , one tries to design a voltage graph that specifies  $G$ .

A starting point is to calculate common divisors of the cardinalities of the vertex- and edge-sets of the graph to be specified. In accordance with Proposition 15.1.1, the larger the common divisor, the smaller the voltage graph needed.

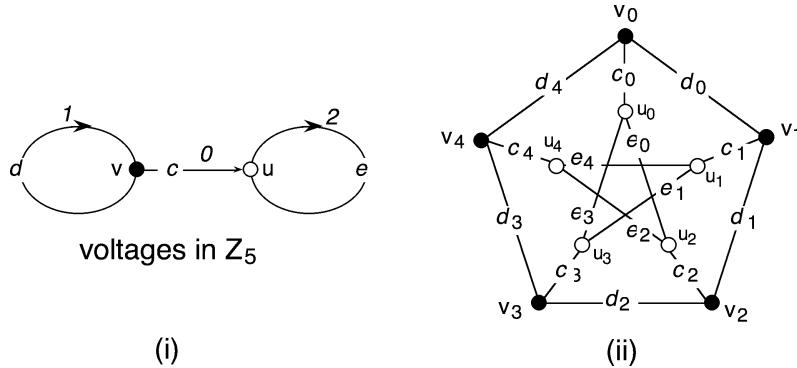


Figure 15.1.12 (i) A  $\mathbb{Z}_5$ -voltage graph; (ii) the Petersen graph, as specified.

**Example 15.1.12:** The Petersen graph, shown above in Figure 15.1.12(ii), has 10 vertices and 15 edges. We calculate  $\gcd(10, 15) = 5$ . By Proposition 15.1.1, it makes sense to try a  $\mathbb{Z}_5$ -voltage graph with two vertices and three arcs. By Proposition 15.1.2, the degree at each vertex of the voltage graph must be three. Since the dipole  $D_3$  is bipartite, Proposition 15.1.4 implies that it cannot be used to specify the Petersen graph, which is not bipartite. The only other connected 3-regular 2-vertex graph is the dumbbell, which is shown in Figure 15.1.12(i), along with a plausible voltage assignment.

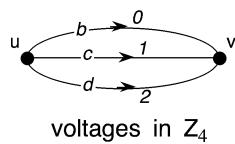
**Example 15.1.13:** The complete graph  $K_9$  has 9 vertices and 36 edges. It makes sense to try a voltage graph with one vertex and four edges, that is, the bouquet  $B_4$ . Assigning cyclic voltages 1, 2, 3, and 4 (mod 9) to the four respective self-loops yields a specification of  $K_9$ . This assertion can be verified by drawing the covering digraph.

For an alternative verification, let  $v$  be the single vertex of the voltage graph. Then each vertex  $v_i$  of the covering graph is adjacent to each of the vertices  $v_{i+j(\text{mod } 9)}$  for  $j = \pm 1, \pm 2, \pm 3, \pm 4$ , that is, to every other vertex.

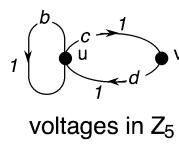
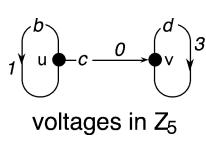
**Example 15.1.14:** The odd complete graph  $K_{2n+1}$  has  $2n+1$  vertices and  $n(2n+1)$  edges. The bouquet  $B_n$  with voltages  $1, 2, \dots, n \pmod{2n+1}$  specifies  $K_{2n+1}$ . This is a generalization of Example 15.1.13.

### EXERCISES for Section 15.1

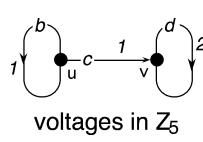
In Exercises 15.1.1 through 15.1.12, draw the covering digraph for the given voltage graph.

15.1.1<sup>s</sup>voltages in  $Z_4$ 

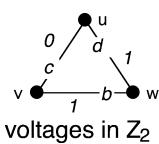
15.1.2

voltages in  $Z_5$ 15.1.3<sup>s</sup>voltages in  $Z_5$ 

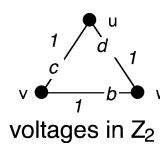
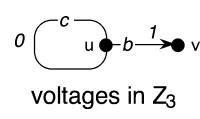
15.1.4

voltages in  $Z_5$ 

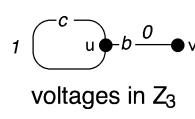
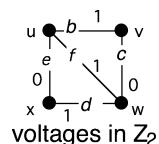
15.1.5

voltages in  $Z_2$ 

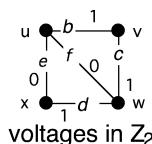
15.1.6

voltages in  $Z_2$ 15.1.7<sup>s</sup>voltages in  $Z_3$ 

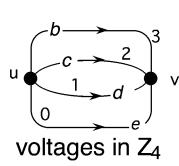
15.1.8

voltages in  $Z_3$ 15.1.9<sup>s</sup>voltages in  $Z_2$ 

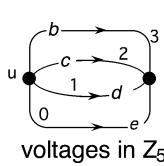
15.1.10

voltages in  $Z_2$ 

15.1.11

voltages in  $Z_4$ 

15.1.12

voltages in  $Z_5$ 

15.1.13 For the voltage graph drawing of Exercise 15.1.1, consider the voltages to be in the group  $\mathbb{Z}_5$ , and draw the covering digraph.

15.1.14 For the voltage graph drawing of Exercise 15.1.2, consider the voltages to be in the group  $\mathbb{Z}_6$ , and draw the covering digraph.

15.1.15 For the voltage graph drawing of Exercise 15.1.3, consider the voltages to be in the group  $\mathbb{Z}_6$ , and draw the covering digraph.

15.1.16 For the voltage graph drawing of Exercise 15.1.4, consider the voltages to be in the group  $\mathbb{Z}_6$ , and draw the covering digraph.

15.1.17<sup>s</sup> Design a cyclic-voltage graph to specify the circular ladder  $CL_n$ .

15.1.18 Give details of the proof of Proposition 15.1.5.

15.1.19 Prove that the covering graph for a cyclic-voltage graph has no self-loops, unless the voltage graph has a self-loop with voltage 0.

15.1.20<sup>s</sup> How can the voltage graph specifying  $K_{2n-1}$  be modified so that it specifies the octahedral graph  $\mathcal{O}_n$ ?

## 15.2 CAYLEY GRAPHS AND REGULAR VOLTAGES

Complete digraphs were used by Arthur Cayley (1878) to construct pictorial representations of finite groups. This elementary idea was refined and generalized by Max Dehn (1911) into a class of highly symmetric graphs, now called *Cayley graphs*, that represent groups and that have acquired additional applications, including the design of parallel computer architectures. A form of voltage graph, called a *regular voltage graph*, in which the voltages may be from any group, not just from a cyclic group, has Cayley graph coverings as a special case. <sup>†</sup>

### Cayley Graphs

The Cayley graph construction can be viewed as a generalization of the circulant graph construction.

**FROM APPENDIX A.4:** A **group**  $\mathcal{G} = \langle G, \cdot \rangle$  is a nonempty set  $G$  and an associative binary operation such that  $\mathcal{G}$  has an identity element and that each  $g \in G$  has an inverse.

Generically, the group operation is regarded as a product, and it is denoted by juxtaposition. However, for groups such as  $\mathbb{Z}_n$ , whose operation is commonly regarded as additive, the plus operator is used.

**FROM APPENDIX A.4:** A subset  $X$  of elements of a group  $\mathcal{B} = \langle B, \cdot \rangle$  is a **generating set** if every element of  $B$  is obtainable as the product (or sum) of elements of  $X$ .

**Example 15.2.1:** For the group  $\mathbb{Z}_n$ , a nonempty set of integers mod  $n$  is a generating set if and only if its greatest common divisor is 1. This observation employs the standard fact that the greatest common divisor of a set of numbers equals the smallest positive number that can be formed by taking sums and differences of numbers in the set. For instance, the set  $\{4, 7\}$  generates  $\mathbb{Z}_{24}$ , but  $\{6, 9\}$  does not.

<sup>†</sup> Some familiarity with groups would probably be helpful for this section.

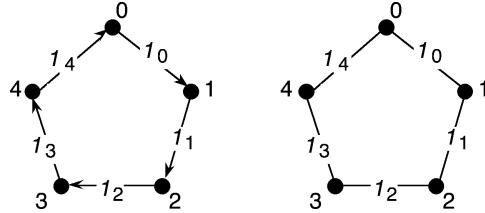
**DEFINITION:** Let  $\mathcal{B} = \langle B, \cdot \rangle$  be a group with generating set  $X$ . Then the **Cayley digraph**  $\vec{C}(\mathcal{B}, X)$  has as its vertex-set and arc-set

$$V_{\vec{C}(\mathcal{B}, X)} = B \quad \text{and} \quad E_{\vec{C}(\mathcal{B}, X)} = \{x_b \mid x \in X, b \in B\}$$

respectively. Arc  $x_b$  joins vertex  $b$  to vertex  $bx$ . (Bidirected arcs are sometimes used for generators of order 2, as described in §15.1.)

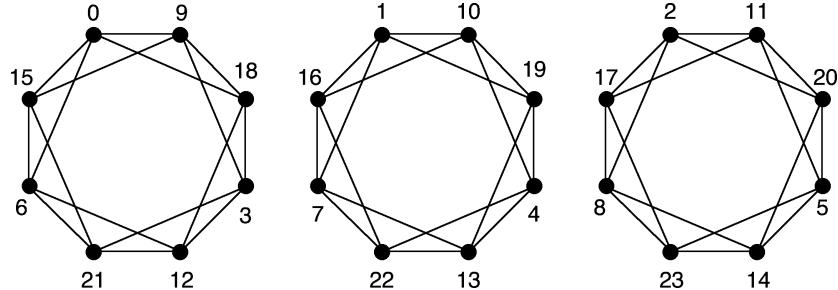
**DEFINITION:** Let  $\mathcal{B} = \langle B, \cdot \rangle$  be a group with generating set  $X$ . The **Cayley graph**  $C(\mathcal{B}, X)$  is the *underlying graph* of the **Cayley digraph**  $\vec{C}(\mathcal{B}, X)$ .

**Example 15.2.2:** Figure 15.2.1 shows the Cayley digraph for the cyclic group  $\mathbb{Z}_5$  with generating set  $\{1\}$  and the corresponding Cayley graph, which is clearly isomorphic to the circulant graph  $\text{circ}(5 : 1)$ .



**Figure 15.2.1** The Cayley digraph  $\vec{C}(\mathbb{Z}_5, \{1\})$  and Cayley graph  $C(\mathbb{Z}_5, \{1\})$ .

**Example 15.2.1, continued:** We observe in Figure 15.2.2 that the circulant graph  $\text{circ}(24 : 6, 9)$  has three components, which are mutually isomorphic. Every Cayley graph is connected, because the edges are defined by a generating set. A circulant graph is a Cayley graph if and only if it is connected.

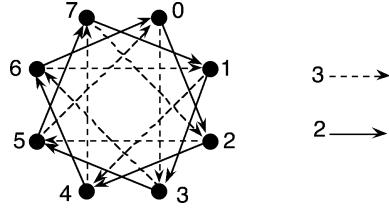


**Figure 15.2.2** The circulant graph  $\text{circ}(24 : 6, 9)$  has three components.

**DEFINITION:** Let  $\mathcal{B} = \langle B, \cdot \rangle$  be a group with generating set  $X$ . The **fiber** over a generator  $x \in X$  in the Cayley digraph  $\vec{C}(\mathcal{B}, X)$  or Cayley graph  $C(\mathcal{B}, X)$  is the set  $\tilde{x} = \{x_b \mid b \in B\}$ .

**TERMINOLOGY NOTE:** The traditional way to draw a Cayley digraph  $\vec{C}(\mathcal{B}, X)$  labels the vertices by group elements. Edges were not given distinct names. Instead, a different color or graphic feature was used for each edge fiber  $\tilde{x}$ , which led to the terminology *Cayley color graph*.

**Example 15.2.3:** Figure 15.2.3 shows the Cayley digraph for the cyclic group  $\mathbb{Z}_8$  with generating set  $\{2, 3\}$ . The legend at the right identifies the edge fibers, according to their graphic features.

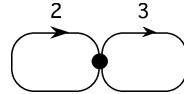


**Figure 15.2.3** A traditional drawing of the Cayley digraph  $\vec{C}(\mathbb{Z}_8, \{2, 3\})$ .

**DEFINITION:** An arbitrary graph  $G$  is said to be a **Cayley graph** if there exists a group  $\mathcal{B}$  and a generating set  $X$  such that  $G$  is isomorphic to the Cayley graph for  $\mathcal{B}$  and  $X$ .

**Remark:** Figure 15.2.3 illustrates that a non-minimal generating set for a group can be used in a Cayley-graph specification of a graph. A minimum generating set for  $\mathbb{Z}_8$  would have only one generator, and the corresponding graph would specify an 8-cycle, not the graph shown above.

**Example 15.2.4:** The Cayley digraph  $\vec{C}(\mathbb{Z}_8, \{2, 3\})$  can also be specified by the  $\mathbb{Z}_8$ -voltage graph in Figure 15.2.4.

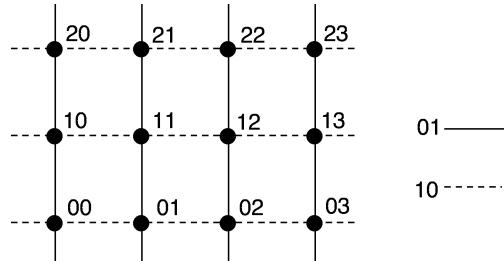


**Figure 15.2.4** A  $\mathbb{Z}_8$ -voltage graph that specifies  $\vec{C}(\mathbb{Z}_8, \{2, 3\})$ .

**Remark:** Later in this section, we establish that every Cayley graph can be specified by assigning voltages to a bouquet.

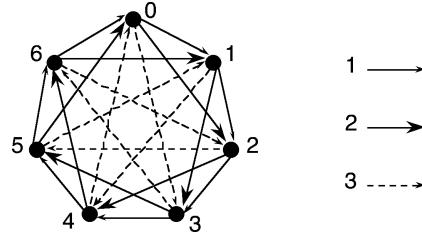
**NOTATION:** The notation used here for an element of a direct sum of  $k$  small cyclic groups is a string of  $k$  digits, rather than a  $k$ -tuple. For instance, 20 stands for the element  $(2, 0)$ . This convention avoids cluttering the drawings with parentheses and commas.

**Example 15.2.5:** Figure 15.2.5 illustrates that the  $3 \times 4$  wraparound mesh is a Cayley graph for the group  $\mathbb{Z}_3 \times \mathbb{Z}_4$ .



**Figure 15.2.5** The Cayley graph  $\vec{C}(\mathbb{Z}_3 \times \mathbb{Z}_4, \{01, 10\})$ .

**Example 15.2.6:** The complete graph  $K_{2n+1}$  is a Cayley graph for the group  $\mathbb{Z}_{2n+1}$ , with generating set  $\{1, 2, \dots, n\}$ . Figure 15.2.6 illustrates this for  $K_7$ .



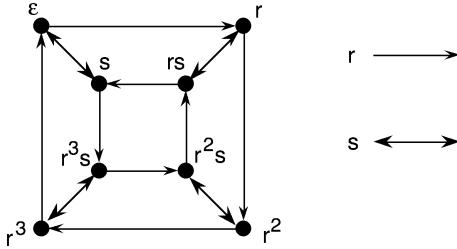
**Figure 15.2.6** The Cayley digraph  $\vec{C}(\mathbb{Z}_7, \{1, 2, 3\})$ .

An element of a generating set  $X$  of order 2 would cause doubled edges to appear in the Cayley graph. Collapsing such doubled edges to a single edge enhances the usefulness of Cayley graphs in algebraic specification.

**DEFINITION: Bidirected-arc convention:** Let  $y$  be a generator of order 2 in a group  $\mathcal{B}$ . Then the pair of arcs that would otherwise join vertex  $b$  to vertex  $by$  and vertex  $by$  to vertex  $b$  in a Cayley digraph for  $\mathcal{B}$  is replaced by a single bidirected arc between  $b$  and  $by$ . The **underlying graph of a digraph with bidirected arcs** has a single edge for each bidirected arc.

**NOTATION:** The notation  $\varepsilon$  refers to the identity element of a group.

**Example 15.2.7:** The dihedral group  $D_4$  is describable as the group of rigid-body motions on the unit square. Let  $r$  denote a  $90^\circ$  clockwise rotation and let  $s$  denote a reflection through a vertical axis. Then the elements of  $D_4$  are  $\{\varepsilon, r, r^2, r^3, s, rs, r^2s, r^3s\}$ . Figure 15.2.7 shows a Cayley digraph for  $D_4$  with generating set  $\{r, s\}$ .



**Figure 15.2.7** A Cayley digraph for the dihedral group  $D_4$ .

### Regular-Voltage Graphs

The power of voltage graphs is increased when the algebraic structure from which the voltages are selected is permitted to be an arbitrary, possibly non-abelian, group. The corresponding generalized voltage-graph construction has the specification of Cayley graphs as a special case.

**DEFINITION:** Let  $G = (V, E)$  be a digraph, and let  $\mathcal{B}$  be a group. A **regular-voltage assignment** on  $G$  in the group  $\mathcal{B}$  is a function  $\alpha$  that assigns to every arc  $e \in E$  an element  $\alpha(e) \in \mathcal{B}$ . The element  $\alpha(e) \in \mathcal{B}$  is called the **voltage** on  $e$ .

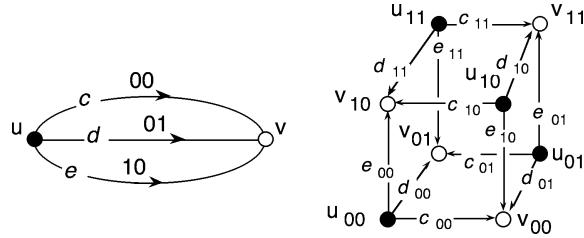
**DEFINITION:** A **regular-voltage graph** is a pair  $\langle G, \alpha : E_G \rightarrow \mathcal{B} \rangle$  such that  $G = (V, E)$  is a digraph,  $\mathcal{B}$  is a group, and  $\alpha$  is a  $\mathcal{B}$ -voltage assignment on  $G$ .

**DEFINITION:** The **voltage group** of a voltage graph  $\langle G, \alpha : E_G \rightarrow \mathcal{B} \rangle$  is the group  $\mathcal{B}$  in which the voltages are assigned.

**TERMINOLOGY NOTE:** A *regular* voltage assignment leads to what algebraic topologists call a *regular covering space* of the voltage graph. Moreover, it corresponds to what algebraists call a *regular action* of the group on the covering graph. Originally, [GrTu77, GrTu87] used the terminology “ordinary voltages”, in order to distinguish them from *permutation voltages*, which are described in the next section.

As in the previous section, it is helpful to examine an example of a covering digraph before considering the formal definition of its construction. The generalization to voltages in arbitrary, possibly non-abelian, groups is only slightly more complicated than cyclic voltages.

**Example 15.2.8:** The cube graph  $Q_3$  can be specified by assigning voltages in  $\mathbb{Z}_2 \times \mathbb{Z}_2$  to the dipole  $D_3$  as shown in Figure 15.2.8.



**Figure 15.2.8** A  $(\mathbb{Z}_2 \times \mathbb{Z}_2)$ -voltage graph and the cube graph that it specifies.

**Remark:** The cube graph  $Q_3$  can also be specified by assigning  $\mathbb{Z}_4$ -voltages 0, 1, and 2, respectively, to edges  $c$ ,  $d$ , and  $e$  of the dipole in Figure 15.2.8. Moreover, Example 15.2.7 specifies  $Q_3$  as a Cayley graph for the group  $\mathcal{D}_4$ . Finding a polynomial-time algorithm to decide whether a given regular graph is a Cayley graph, or proving that this decision problem is NP-complete, is a research problem. Accordingly, finding all the different ways in which a given regular graph can be specified as a Cayley graph is often regarded as a difficult problem.

**DEFINITION:** The **covering digraph** of the regular-voltage graph  $\langle G, \alpha : E_G \rightarrow \mathcal{B} \rangle$  is the graph  $G^\alpha = (V^\alpha, E^\alpha)$ , with

$$\begin{aligned} V^\alpha &= \{v_b \mid v \in V_G \text{ and } b \in \mathcal{B}\} \\ E^\alpha &= \{e_b \mid e \in E_G \text{ and } b \in \mathcal{B}\} \end{aligned}$$

such that if an arc  $e$  in a voltage graph joins vertex  $u$  to vertex  $v$ , then the arc  $e_b$  joins vertex  $u_b$  to vertex  $v_{b\alpha(e)}$ . The **covering graph** is the underlying graph of the covering digraph  $G^\alpha$ . It is also denoted  $G^\alpha$ .

**TERMINOLOGY:** The terminology **vertex fiber** and **edge fiber** is used for regular-voltage graphs, as for cyclic-voltage graphs.

CONVENTIONS: Regular voltage graphs may also suppress mainscripts and selectively use the bidirected arc convention.

**Example 15.2.9:** The dumbbell graph on the left in Figure 15.2.9 has voltages in the group  $\mathbb{Z}_2 \times \mathbb{Z}_2$ . Observe that the covering digraph can also be drawn as an 8-cycle in which every other edge is doubled.

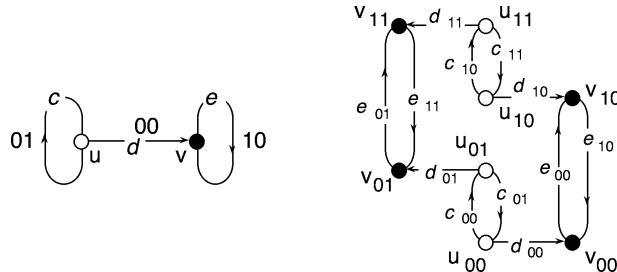


Figure 15.2.9 A  $(\mathbb{Z}_2 \times \mathbb{Z}_2)$ -voltage graph and the digraph that it specifies.

### Graphic Conventions for Clarifying Voltage-Graph Drawings

Although it is a great convenience to be able to specify a large graph by drawing a small voltage graph, the proliferation of subscripted labels on vertices and edges throughout a drawing of a covering graph is somewhat of an inconvenience. To relieve the cluttered appearance, there are two helpful graphic conventions that may be used separately or in combination.

DEFINITION: The **clarifying-vertex-graphic convention** for voltage graphs is that each vertex is drawn in a unique graphic, and in the covering graph, all the vertices in its fiber are drawn in that same graphic. Moreover, in the covering graph, each vertex is labeled by its subscript only, with its mainscript omitted.

DEFINITION: The **clarifying-edge-graphic convention** for voltage graphs is that each edge is drawn in a unique graphic, and in the covering graph, all the edges in its fiber are drawn in that same graphic. In the covering digraph and graph, edge labels are altogether omitted. Mainscripts can be inferred from the edge graphic, and subscripts from the endpoints.

**Example 15.2.10:** Figure 15.2.10 shows how these clarifying-graphic conventions are applied to the voltage graph of Figure 15.2.9. The rightmost graph also uses the bidirected-arc convention.

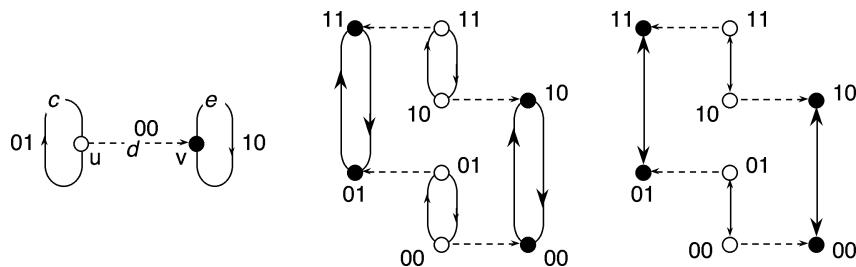


Figure 15.2.10 Simplified drawings of the covering graph of Figure 15.2.9.

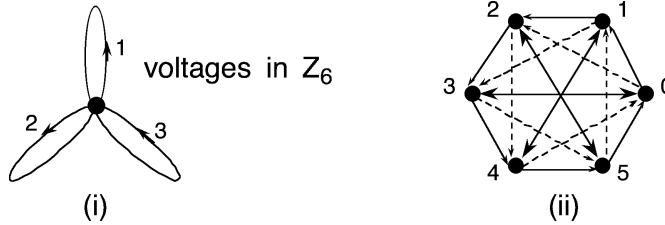
### Voltages on a Bouquet to Specify a Cayley Graph

The following proposition declares how every Cayley graph can be obtained by assigning voltages to a bouquet.

**Proposition 15.2.1.** *Let  $\mathcal{A}$  be group with an  $n$ -element generating set  $X$ , and let  $\langle B_n, \alpha \rangle$  be a voltage graph, where  $\alpha$  bijectively assigns the elements of  $X$  to the self-loops of bouquet  $B_n$  as voltages. Then the Cayley digraph  $\vec{C}(\mathcal{A}, X)$  is isomorphic to the covering digraph  $B_n^\alpha$ .*

**Proof:** The vertex function of the isomorphism maps each vertex  $a$  of the Cayley digraph  $\vec{C}(\mathcal{A}, X)$  to the vertex labeled  $a$  in the covering digraph  $B_n^\alpha$ . For each  $x \in X$ , the edge function maps edge  $x_a$  of  $\vec{C}(\mathcal{A}, X)$  to the edge  $e_a$  of  $B_n^\alpha$ , where self-loop  $e$  carries voltage  $x$ .  $\diamond$

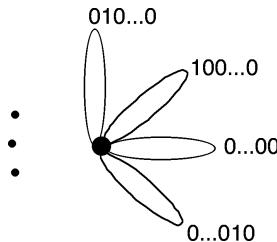
**Example 15.2.11:** In accordance with Proposition 15.2.1, Figure 15.2.11 shows (i) a  $\mathbb{Z}_6$ -voltage graph and (ii) the complete graph  $K_6$ , labeled as the Cayley digraph for that voltage graph.



**Figure 15.2.11** (i) A  $\mathbb{Z}_6$ -voltage graph; (ii) its Cayley digraph.

More generally, assigning voltages  $1, 2, \dots, [\frac{n}{2}] (\text{mod } n)$  to the self-loops of the bouquet  $B_{[\frac{n}{2}]}$  yields a Cayley graph isomorphic to the complete graph  $K_n$ , assuming that the bidirected-arc convention is operational in the case of even  $n$ .

**Example 15.2.12:** Suppose that the objective is to specify an arbitrary  $d$ -dimensional  $m_1 \times m_2 \times \dots \times m_d$  wraparound mesh. This is a generalization of Example 15.2.11. Voltages from the group  $\mathbb{Z}_{m_1} \times \mathbb{Z}_{m_2} \times \dots \times \mathbb{Z}_{m_d}$  should be assigned to the bouquet  $B_d$ , as shown in Figure 15.2.12. Observe that no directions are drawn on the self-loops. Such a lapse is acceptable in this instance, because no matter which way the self-loops are directed, the same graph is specified.



**Figure 15.2.12** A voltage graph for a  $d$ -dimensional wraparound mesh.

### Multiple Components

It might seem more natural in Example 15.2.8 if the vertices of the cube graph were labeled by bitstrings of length 3, that is, by voltages in  $\mathbb{Z}_2 \times \mathbb{Z}_2 \times \mathbb{Z}_2$ . Yet from Proposition 15.1.1, it appears as if the cube graph  $Q_3$ , with 8 vertices and 12 edges, cannot be specified with the voltage group  $\mathbb{Z}_2 \times \mathbb{Z}_2 \times \mathbb{Z}_2$ , which has eight elements. This apparent problem is overcome by slightly relaxing the concept of specification.

**Example 15.2.13:** In Figure 15.2.13, the covering graph has two components, each isomorphic to  $Q_3$ .

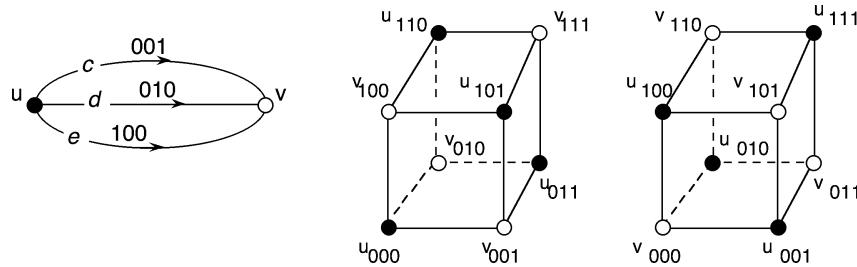


Figure 15.2.13 Another way to obtain a cube graph.

**Example 15.2.14:** The previous example is easily generalized. Every hypercube graph  $Q_n$  can be specified either by  $\mathbb{Z}_2^{n-1}$ -voltages or by  $\mathbb{Z}_2^n$ -voltages on the dipole  $D_n$ . Thus, an advantage of allowing multiple components here is that the subscripts of the vertex names of the covering graph are natural labels for a hypercube.

One need not be concerned about choosing the correct component of the covering graph, since all the components are isomorphic. Indeed, each element of the voltage group gives rise to an automorphism on the covering graph, as the final results of this section show.

### The Natural Transformation and Vertex-Transitivity

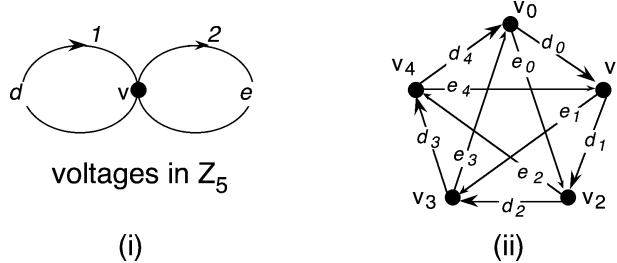
**DEFINITION:** Let  $\langle G, \alpha : E_G \rightarrow \mathcal{B} \rangle$  be a regular-voltage graph, and let  $a \in \mathcal{B}$ . The **natural transformation**  $\varphi_a$  on the covering graph  $G^\alpha$  is given by the rules that  $v_b \mapsto v_{ab}$  and  $e_b \mapsto e_{ab}$ , for all  $b \in B$ .

**Remark:** When the voltage group is non-abelian, it is important to observe that the automorphism  $\varphi_a$  acts by multiplying from the left. When the voltage group is abelian, as in most of our examples, the natural transformation is given by addition to the subscripts.

**Proposition 15.2.2.** Let  $\langle G, \alpha : E_G \rightarrow \mathcal{B} \rangle$  be a regular-voltage graph, and let  $a \in \mathcal{B}$ . Then the natural transformation  $\varphi_a : G^\alpha \rightarrow G^\alpha$  is an automorphism.

**Proof:** Let arc  $e \in E_G$  have tail  $u$  and head  $v$ . Then the endpoints of the edge  $e_b$  in  $G^\alpha$  are  $u_b$  and  $v_{b\alpha(e)}$ . The definition of the natural transformation asserts that  $\varphi_a(e_b) = e_{ab}$ , and the definition of the voltage-graph construction prescribed that the endpoints of the edge  $e_{ab}$  are  $u_{ab}$  and  $v_{ab\alpha(e)}$ . The definition of the natural transformation asserts that  $\varphi_a(u_b) = u_{ab}$  and  $\varphi_a(v_{b\alpha(e)}) = v_{ab\alpha(e)}$ . Thus, the mapping pair  $\varphi_a$  preserves incidence and is a graph mapping. Since  $\varphi_a$  has an inverse, namely  $\varphi_{a^{-1}}$ , it follows that  $\varphi_a$  is an automorphism.  $\diamond$

**Example 15.2.15:** The natural transformation  $\varphi_1$  on the specified copy of  $K_5$  in Figure 15.2.14 (which is identical to Figure 15.1.7) corresponds to a rotation of  $\frac{2\pi}{5}$  radians clockwise. More generally, the natural transformation  $\varphi_m$  corresponds to a rotation of  $\frac{2m\pi}{5}$  radians clockwise.



**Figure 15.2.14** (i)  $\mathbb{Z}_5$ -voltages on  $B_2$ ; (ii) the specified copy of  $K_5$ .

**Corollary 15.2.3.** Let  $\langle G, \alpha : E_G \rightarrow \mathcal{B} \rangle$  be a regular-voltage graph, and let  $v \in V$ . Then any two components of the covering graph  $G^\alpha$  that contain a vertex from the vertex fiber  $\tilde{v}$  are isomorphic.

**Proof:** Let  $v_a, v_b \in \tilde{v}$ . Then  $\varphi_{ba^{-1}}$  is an isomorphism from the component of  $v_a$  to the component of  $v_b$ .  $\diamond$

**Example 15.2.16:** The natural transformation  $\varphi_{001}$  swaps the two cubes in the covering graph of Example 15.2.13.

REVIEW FROM §2.2: A graph  $G$  is **vertex-transitive** if for all vertex pairs  $u, v \in V_G$ , there is an automorphism of  $G$  that maps  $u$  to  $v$ .

**Proposition 15.2.4.** For any group  $\mathcal{B}$ , the Cayley graph  $\vec{C}(\mathcal{B}, X)$  is vertex-transitive.

**Proof:** Let  $b, c \in \mathcal{B}$ . Then the natural transformation  $\varphi_{cb^{-1}}$  maps vertex  $b$  to vertex  $c$  in  $\vec{C}(\mathcal{B}, X)$ .  $\diamond$

### EXERCISES for Section 15.2

In Exercises 15.2.1 through 15.2.4, draw the Cayley graph for the given group and generating set.

- 15.2.1<sup>s</sup> The cyclic group  $\mathbb{Z}_9$  with generating set  $\{1, 3\}$ .
- 15.2.2 The cyclic group  $\mathbb{Z}_{10}$  with generating set  $\{2, 5\}$ .
- 15.2.3 The group  $\mathbb{Z}_2 \times \mathbb{Z}_3$  with generating set  $\{10, 01\}$ .
- 15.2.4 The group  $\mathbb{Z}_2 \times \mathbb{Z}_3 \times \mathbb{Z}_4$  with generating set  $\{110, 101, 011\}$ .

In Exercises 15.2.5 through 15.2.8, draw the Cayley graph for the given group and generating set, and draw its voltage-graph specification.

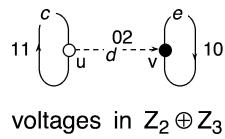
- 15.2.5<sup>s</sup> The group  $\mathbb{Z}_2 \times \mathbb{Z}_3$  with generating set  $\{11, 12\}$ .
- 15.2.6 The dihedral group  $D_5$  of rigid-body motions on a regular pentagon, using a  $72^\circ$  rotation  $r$  and a reflection  $s$  through the vertical axis as generators.

**15.2.7** (algebraic) The symmetric group  $\Sigma_4$  of permutations on the set  $\{1, 2, 3, 4\}$ , with generating set  $\{(1 \ 2 \ 3 \ 4), (1 \ 2)\}$ .

**15.2.8** (algebraic) The *alternating group*  $A_4$  of permutations on the set  $\{1, 2, 3, 4\}$ , with generating set  $\{(1 \ 2 \ 3), (1 \ 2 \ 4)\}$ .

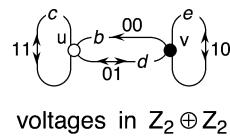
*In Exercises 15.2.9 through 15.2.12, draw the covering graph for the given regular-voltage graph.*

**15.2.9<sup>s</sup>**



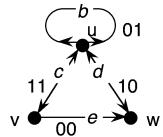
voltages in  $Z_2 \oplus Z_3$

**15.2.10**



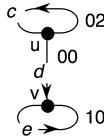
voltages in  $Z_2 \oplus Z_2$

**15.2.11**



voltages in  $Z_2 \oplus Z_2$

**15.2.12**



voltages in  $Z_3 \oplus Z_3$

*In Exercises 15.2.13 through 15.2.15, give a geometric description of the effect produced by the given natural transformation on the covering graph of the given voltage graph.*

**15.2.13<sup>s</sup>** Voltage graph of Exercise 15.2.1 and transformation  $\varphi_2$ .

**15.2.14** Voltage graph of Exercise 15.2.2 and transformation  $\varphi_3$ .

**15.2.15** Voltage graph of Exercise 15.2.6 and transformation  $\varphi_s$ .

*In Exercises 15.2.16 through 15.2.18, for the covering graph of the given voltage graph, write the vertex-permutation and the edge-permutation of the given natural transformation.*

**15.2.16** Voltage graph of 15.2.9 and transformation  $\varphi_{11}$ .

**15.2.17<sup>s</sup>** Voltage graph of Exercise 15.2.10 and transformation  $\varphi_{10}$ .

**15.2.18** Voltage graph of Exercise 15.2.11 and transformation  $\varphi_{01}$ .

**15.2.19** Let  $\langle G, \alpha : E_G \rightarrow \mathcal{B} \rangle$  be a regular-voltage graph. Prove that the composition  $\varphi_a \circ \varphi_b$  is the natural automorphism  $\varphi_{ab}$ .

## 15.3 PERMUTATION VOLTAGES

Modifying the covering graph so that subscripts may be the objects of any permutation group is a powerful way to expand the class of graphs that can be specified by voltage graphs.

### Permutation-Voltage Specifications

The terminology for each detail of a permutation-voltage graph is similar to the terminology for the corresponding detail of a regular-voltage graph.

**DEFINITION:** Let  $\mathcal{P} = [P : Y]$  be a permutation group. A  **$\mathcal{P}$ -voltage** on an edge of a digraph is a label on that edge by a permutation  $\pi \in P$ . Any such voltage is called a **permutation voltage**.

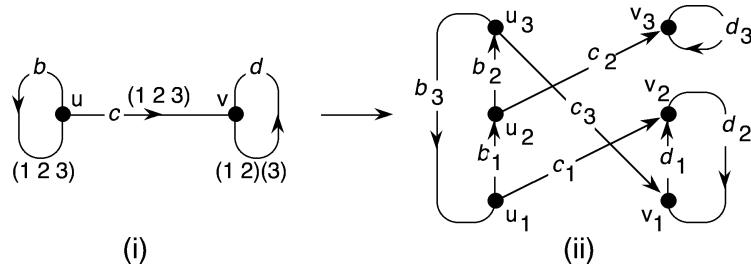
**DEFINITION:** A **permutation-voltage assignment** on a digraph  $G = (V, E)$  is a function  $\alpha$  that labels every arc  $e \in E$  with a permutation voltage from some permutation group  $\mathcal{P} = [P : Y]$ . A  **$\mathcal{P}$ -voltage graph** is a permutation-voltage graph with voltages in  $\mathcal{P} = [P : Y]$ .

**DEFINITION:** A **permutation-voltage graph** is a pair  $\langle G, \alpha \rangle$  such that  $G$  is a digraph and  $\alpha$  is a permutation-voltage assignment on  $G$ .

**DEFINITION:** The **voltage group** of a permutation-voltage graph  $\langle G, \alpha \rangle$  is the permutation group  $\mathcal{P} = [P : Y]$  in which the voltages are assigned.

Most often, the group for a permutation-voltage assignment is the symmetric group  $\Sigma_n$  acting on the objects  $\{1, 2, \dots, n\}$ . Usually, permutation voltages are written in disjoint cycle form. It may help to examine some properties of an example before reading the definitions.

**Example 15.3.1:** Figure 15.3.1 shows (i) a permutation-voltage graph and (ii) the digraph it specifies.



**Figure 15.3.1** (i) A  $\Sigma_3$ -voltage graph; (ii) the covering digraph.

Observe in the voltage graph that arc  $c$  joins vertex  $u$  to vertex  $v$ , and that the voltage on arc  $c$  is the permutation  $(1 2 3)$ . Observe in the covering digraph that arc  $c_1$  joins vertex  $u_1$  to vertex  $v_2$ , since the permutation  $(1 2 3)$  maps the object 1 to the object 2. Similarly, arc  $c_2$  joins vertex  $u_2$  to vertex  $v_3$ , since  $(1 2 3)$  maps the object 2 to the object 3, and arc  $c_3$  joins vertex  $u_3$  to vertex  $v_1$ , since  $(1 2 3)$  maps the object 3 to the object 1.

**DEFINITION:** The **covering digraph** of the permutation-voltage graph  $\langle G, \alpha : E_G \rightarrow \mathcal{P} \rangle$  with voltages in  $\mathcal{P} = [P : Y]$  is the graph  $G^\alpha = (V^\alpha, E^\alpha)$ , with

$$V^\alpha = \{v_j \mid v \in V_G \text{ and } j \in Y\}$$

$$E^\alpha = \{e_j \mid e \in E_G \text{ and } j \in Y\}$$

such that if an arc  $e$  in a voltage graph joins vertex  $u$  to vertex  $v$ , then the arc  $e_j$  joins vertex  $u_j$  to vertex  $v_{\alpha(e)(j)}$ . The **covering graph** is the underlying graph of the covering digraph  $G^\alpha$ . It is also denoted  $G^\alpha$ .

**TERMINOLOGY:** The terminology **vertex fiber** and **edge fiber** is used for permutation-voltage graphs, as for cyclic-voltage graphs.

**CONVENTIONS:** Permutation-voltage graphs may also suppress mainscripts and selectively use the bidirected-arc convention.

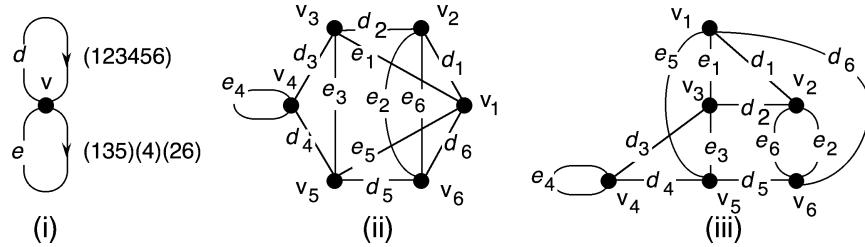
**Remark:** The preceding definition specifies that the subscript of the tail vertex  $u_j$  always agrees with the subscript of its arc  $e_j$ . It also specifies that the subscript of the head vertex  $v_{\alpha(e)(j)}$  is the object  $\alpha(e)(j)$  to which object  $j$  is permuted by the voltage  $\alpha(e)$  assigned to edge  $e$ , as in Example 15.3.1.

**Remark:** Every regular  $\mathcal{B}$ -voltage graph is equivalent to a permutation-voltage graph under the regular representation of the group  $\mathcal{B}$  as a permutation group on itself. In this sense, permutation-voltage graphs generalize regular-voltage graphs.

### Drawing Graphs Specified by Permutation Voltages

There is no fixed rule to determine the best way to position the vertices in drawing the covering graph from a voltage graph. If the voltage graph has only one vertex, one way is to arrange all the vertices in a circle, and another is to group the covering vertices according to the partition imposed by the disjoint-cycle structure of the permutation voltage on one of the self-loops of the voltage graph.

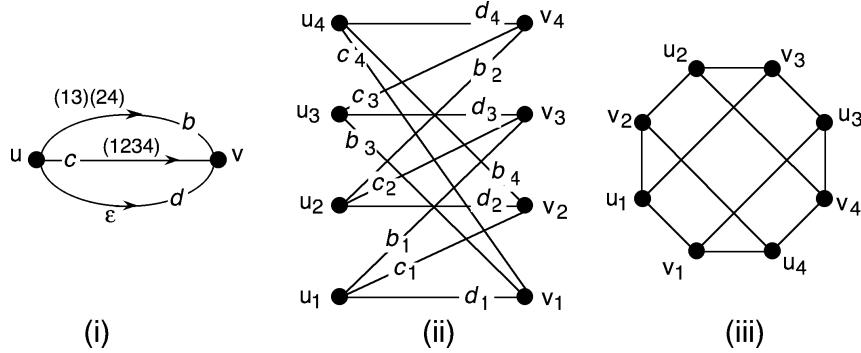
**Example 15.3.2:** Figure 15.3.2(i) shows  $\Sigma_6$ -voltages on the bouquet  $B_2$ . Figure 15.3.2(ii) shows the covering graph with its vertices arranged around the regular hexagon generated by self-loop  $d$ . Figure 15.3.2(iii) shows the covering graph with the vertices grouped into vertical columns according to the partition imposed by the disjoint-cycle structure of the permutation voltage on self-loop  $e$ .



**Figure 15.3.2** A  $\Sigma_6$ -voltage graph and two views of its covering graph.

When the voltage graph has more than one vertex, sometimes the covering graph is drawn so that its vertices are organized into columns according to their vertex fibers.

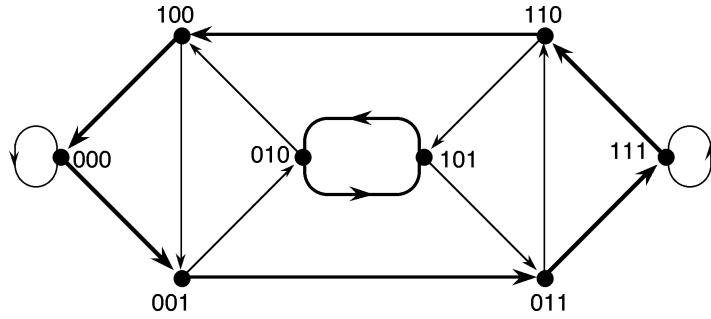
**Example 15.3.3:** Figure 15.3.3(i) shows a  $\Sigma_4$ -voltage graph. Figure 15.3.3(ii) shows the covering graph with its vertices in fiber columns. Figure 15.3.3(iii) shows the covering graph with its vertices arranged around a regular octagon. This covering graph is isomorphic to the cube graph  $Q_3$  (see Exercises).



**Figure 15.3.3** A  $\Sigma_4$ -voltage graph and two views of its covering graph.

### Specifying the deBruijn Graphs

In §6.2, the  $(2, n)$ -deBruijn digraph  $D_{2,n}$  is defined to have  $2^{n-1}$  vertices, labeled by the bitstrings of length  $n - 1$ , and  $2^n$  arcs. Figure 15.3.4 shows  $D_{2,4}$ .



**Figure 15.3.4** The  $(2, 4)$ -deBruijn digraph  $D_{2,4}$ .

The deBruijn digraph  $D_{2,n}$  is not vertex-transitive (see Exercises). Thus, by Proposition 15.2.4, it cannot be specified by regular voltages on a bouquet. Of course, since a bouquet voltage graph would provide the smallest possible specification (in the sense of the fewest vertices), it would be desirable to use voltages on a bouquet. By using permutation voltages, this becomes possible. In anticipation of using the clarifying vertex graphic convention, the object set is

$$Y = \{000, 001, 010, 011, 100, 101, 110, 111\}$$

**DEFINITION:** The (**left**) **cycle-shift** is a permutation that transfers the leftmost bit of a bitstring to the right end. It is denoted  $\zeta$ . Its restriction to bitstrings of length  $k$  is denoted  $\zeta_k$ .

**DEFINITION:** The **deBruijn permutation** is a permutation that transfers the leftmost bit of a bitstring to the right end, thus shifting each of the other  $n - 1$  bits one position to the left, and then adds a 1-bit to the rightmost bit (i.e., *complements* the rightmost bit). It is denoted  $\delta$ . Its restriction to bitstrings of length  $k$  is denoted  $\delta_k$ .

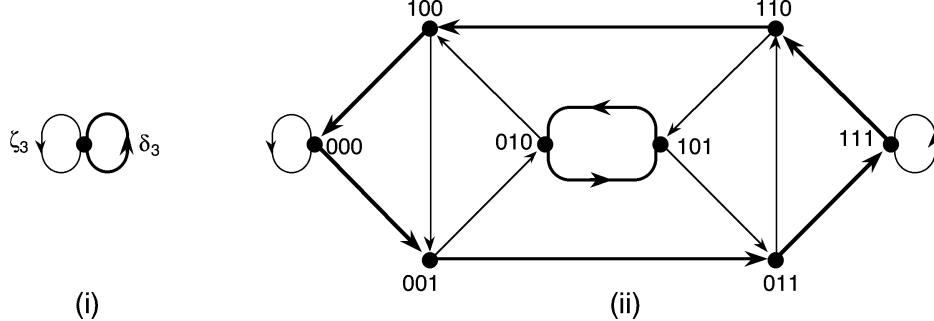
The disjoint-cycle representation of  $\zeta_3$  is

$$\zeta_3 = (000) (001 \ 010 \ 100) (011 \ 110 \ 101) (111)$$

and the disjoint-cycle representation of  $\delta_3$  is

$$\delta_3 = (000 \ 001 \ 011 \ 111 \ 110 \ 100) (101 \ 010)$$

Let  $\mathcal{P} = [P : Y]$  be the permutation group generated by  $\zeta_3$  and  $\delta_3$ . (The group  $\mathcal{P}$  has 24 elements, as explained below, but it is not necessary to know this.) Figure 15.3.5(i) shows a permutation-voltage graph, with voltages in  $\mathcal{P}$ , that specifies the deBruijn graph  $D_{2,4}$ . Figure 15.3.5(ii) shows the covering graph, using the clarifying-graphic conventions. This construction generalizes to all deBruijn graphs.



**Figure 15.3.5** Specifying the deBruijn digraph  $D_{2,4}$  with permutation voltages.

### Wreath Product $\mathbb{Z}_n \otimes_{wr} \mathbb{Z}_2$ <sup>†</sup>

The permutation group generated by  $\zeta_n$  and  $\delta_n$  is known as the *wreath product*  $\mathbb{Z}_n \otimes_{wr} \mathbb{Z}_2$ . Each of its elements can be denoted by a pair  $\langle k, b_0 b_1 \dots b_{n-1} \rangle$  containing an integer  $k \in \{0, 1, \dots, n-1\}$  and a bitstring  $b_0 b_1 \dots b_{n-1}$  of length  $n$ . It permutes a bitstring of length  $n$  by first cycle-shifting its bits  $k$  positions leftward and then adding the bitstring  $b_0 b_1 \dots b_{n-1}$  to the result. In this notation,

$$\zeta_3 = \langle 1, 000 \rangle \quad \text{and} \quad \delta_3 = \langle 1, 001 \rangle$$

For instance,

$$\langle 1, 001 \rangle(011) = 110 + 001 = 111$$

and

$$\langle 1, 000 \rangle(011) = 110 + 000 = 110$$

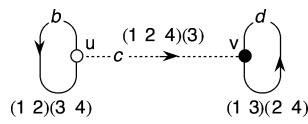
**Remark:** The graph  $G^\alpha$  specified by any voltage-graph construction is like a set of layers of  $G$  in which some edges cross-connect layers, instead of staying in their own layer. Voltage graphs are used extensively in the construction of surface imbeddings, as described in Chapter 16. The paper [GrTu77] in which Gross and Tucker introduce permutation-voltage graphs shows that they are sufficiently powerful to construct every possible instance of what algebraic topologists call a *covering space* of a graph.

---

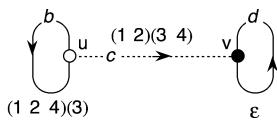
<sup>†</sup> Prior acquaintance with wreath products is probably necessary to understand this subsection.

**EXERCISES for Section 15.3**

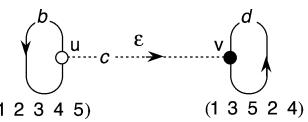
In Exercises 15.3.1 through 15.3.8, draw the covering digraph for the given permutation-voltage graph.

15.3.1<sup>s</sup>

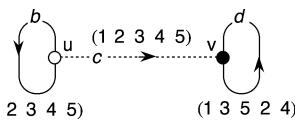
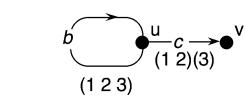
15.3.2



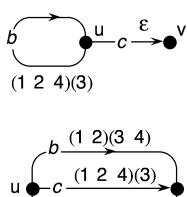
15.3.3



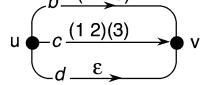
15.3.4

15.3.5<sup>s</sup>

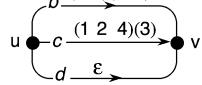
15.3.6



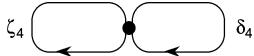
15.3.7



15.3.8



15.3.9 Draw the deBruijn digraph  $D_{2,4}$  as specified by the following permutation-voltage graph.



15.3.10 Prove that the covering graph of Figure 15.3.3 is isomorphic to the cube graph  $Q_3$ .

15.3.11 Prove that the deBruijn graph  $D_{2,3}$  is not vertex-transitive.

15.3.12<sup>s</sup> Write the disjoint-cycle decomposition of the product  $\delta_3\delta_3$ .

15.3.13 Write the product  $\delta_3\delta_3$  as a pair  $\langle k, b_0b_1b_3 \rangle$ .

In Exercises 15.3.14 through 15.3.16, calculate the image of the given permutation from the wreath product  $\mathbb{Z}_3 \otimes_{wr} \mathbb{Z}_2$  on the given bitstring.

15.3.14<sup>s</sup>  $\langle 1, 101 \rangle(010)$ .15.3.15  $\langle 2, 010 \rangle(100)$ .15.3.16  $\langle 2, 101 \rangle(101)$ .

## 15.4 SYMMETRIC GRAPHS AND PARALLEL ARCHITECTURES

When there is a need to write a brief specification of a given fixed graph  $\tilde{G}$ , it is natural to turn to voltage-graph methods to try to design such a specification. Historically, voltage graphs were developed to specify network layouts on surfaces. Use of voltage graphs to specify networks of processors for parallel-computation architectures began more recently.

### Theoretical Criteria for Designing Base Graphs

**DEFINITION:** When voltages are assigned to the edges of a digraph  $G$ , that digraph is called the **base digraph** of the construction.

**DEFINITION:** The **base graph** of a voltage-graph construction is the underlying graph of the base digraph.

**DEFINITION:** The **natural projection** of a voltage-graph construction is the graph mapping that carries every vertex fiber  $\tilde{v}$  to its base vertex  $v$  and every edge fiber  $\tilde{e}$  to its base edge  $e$ . (In effect, the natural projection “erases” the subscripts.)

Theorem 15.4.1 and Theorem 15.4.2 are fundamental to every application of voltage graphs, since they prescribe properties of the base graph that narrow the candidates to a tractably small number of possibilities. The importance of Theorem 15.4.3 is its implication that the base graph can be developed first and the directions assigned afterward. The proofs of Theorems 15.4.1 through 15.4.4 parallel those of Propositions 15.1.1 through 15.1.4.

**Theorem 15.4.1.** *Let  $\langle G, \alpha : E_G \rightarrow \mathcal{A} \rangle$  be either a regular-voltage graph with  $|\mathcal{A}| = n$  or a permutation-voltage graph with  $n$  objects in the permuted set. Then there are  $n$  times as many vertices and edges in the covering digraph  $G^\alpha$  as in the base graph  $G$ .  $\diamond$*

**Remark:** Another way to express Theorem 15.4.1 is to say that the natural projection is  $n$ -to-1.

**Theorem 15.4.2.** *Let  $\langle G, \alpha \rangle$  be any kind of voltage graph, and let  $v$  be any vertex of  $G$ . Then the outdegree and indegree of each vertex  $v_j$  in the fiber  $\tilde{v}$  are equal to the outdegree and indegree, respectively, of vertex  $v$ .  $\diamond$*

**Remark:** Another way to express Theorem 15.4.2 is to say that the natural projection preserves degree.

**Theorem 15.4.3.** *Let  $\langle G, \alpha \rangle$  be any kind of voltage graph, and let  $d$  be an arc of  $G$  from  $u$  to  $v$ . Let  $\langle H, \beta \rangle$  be the voltage graph obtained from  $\langle G, \alpha \rangle$  by reversing the direction of arc  $d$  and changing its voltage to the inverse of  $\alpha(d)$  in the voltage group. Then the covering graph  $H^\beta$  is isomorphic to the covering graph  $G^\alpha$ .  $\diamond$*

**Theorem 15.4.4.** *Let  $\langle G, \alpha \rangle$  be any kind of voltage graph, such that the base graph  $G$  is bipartite. Then the covering graph  $G^\alpha$  is bipartite.  $\diamond$*

### Specification Problems with Elementary Solutions

Given a fixed graph  $\tilde{G} = (\tilde{V}, \tilde{E})$ , a solution to the fundamental specification problem has three steps:

- (1) Design a plausible base digraph  $G = (V, E)$ .
- (2) Select a plausible voltage group.
- (3) Design a suitable voltage assignment.

Quite often, the objective is to specify every member of a graph family, not just a single graph.

**DEFINITION:** A **unified specification** for an entire family of graphs (usually) consists of a single base graph and a single voltage assignment formula that specifies every graph in the family.

In accordance with Theorem 15.4.1, the first step in designing a specification for a given graph  $\tilde{G}$  (or family of graphs) begins by calculating  $\gcd(|\tilde{V}|, |\tilde{E}|)$  (or a formula for the whole family). Ideally, there will be a base digraph  $G = (V, E)$  that satisfies the conditions

$$|V| = \frac{|\tilde{V}|}{\gcd(|\tilde{V}|, |\tilde{E}|)} \quad \text{and} \quad |E| = \frac{|\tilde{E}|}{\gcd(|\tilde{V}|, |\tilde{E}|)}$$

With the aid of Theorem 15.4.2, it is often easy to restrict the supply of plausible candidates for the base digraph to a reasonable number.

**Example 15.4.1:** Consider the problem of specifying the circular ladder  $CL_n$ . Since

$$|\tilde{V}| = |V(CL_n)| = 2n \quad \text{and} \quad |\tilde{E}| = |E(CL_n)| = 3n$$

it follows that

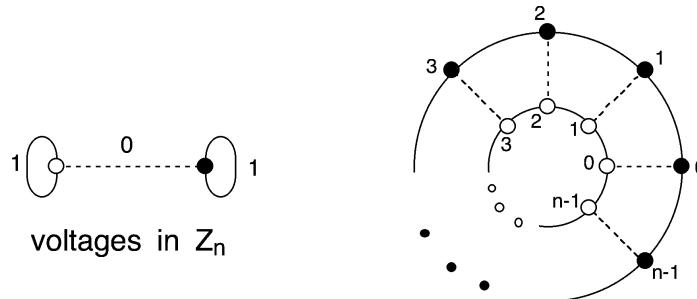
$$\gcd(|\tilde{V}|, |\tilde{E}|) = \gcd(2n, 3n) = n$$

Thus, the first base graphs  $G = (V, E)$  to consider have

$$|V| = \frac{|\tilde{V}|}{n} = 2 \quad \text{and} \quad |E| = \frac{|\tilde{E}|}{n} = 3$$

Since every vertex of  $CL_n$  has degree 3, it follows from Theorem 15.4.2 that the vertices of the base digraph must have degree 3. The only two 3-regular graphs with  $|V| = 2$  and  $|E| = 3$  are the dipole  $D_3$  and the dumbbell graph. For odd  $n$ , the circular ladder  $CL_n$  is not bipartite, so, by Theorem 15.4.4, the dipole, which is bipartite, cannot be the base graph. Thus, the dumbbell is the only plausible candidate for a unified specification.

The cyclic group  $\mathbb{Z}_n$  is a plausible candidate for the voltage group, since it has  $n$  elements. Figure 15.4.1 shows a  $\mathbb{Z}_n$ -voltage assignment on the dumbbell that specifies  $CL_n$ .



**Figure 15.4.1** Using the dumbbell to specify the circular ladders  $CL_n$ .

**Example 15.4.2:** The complete bipartite graph  $K_{n,n}$  has

$$|\tilde{V}| = |V(K_{n,n})| = 2n \quad \text{and} \quad |\tilde{E}| = |E(K_{n,n})| = n^2$$

so that

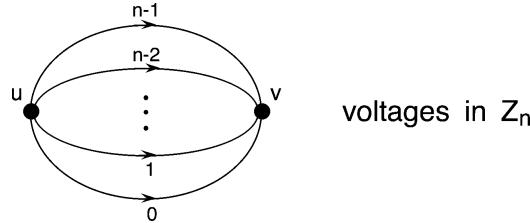
$$\gcd(|\tilde{V}|, |\tilde{E}|) = \gcd(2n, n^2) = \begin{cases} n & \text{if } n \text{ is odd} \\ 2n & \text{if } n \text{ is even} \end{cases}$$

Thus, for  $n$  odd, the first base graphs  $G = (V, E)$  to consider have

$$|V| = \frac{|\tilde{V}|}{n} = 2 \quad \text{and} \quad |E| = \frac{|\tilde{E}|}{n} = n$$

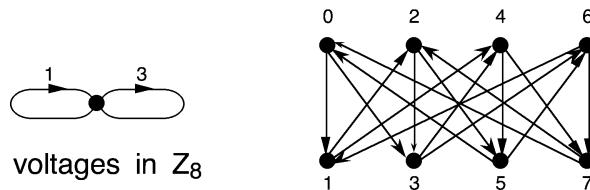
Since every vertex of  $K_{n,n}$  has degree  $n$ , it follows from Theorem 15.4.2 that the vertices of the base digraph must have degree  $n$ . Thus, the plausible candidates for the base digraph have two vertices,  $2k+1$  arcs joining them, and  $\frac{n-2k-1}{2}$  self-loops at each vertex. The most attractive candidate among them (it is likely to be the easiest to work with) is the dipole  $D_n$ . By Proposition 15.4.3, all the arcs can go in the same direction from one vertex to the other. By Theorem 15.4.4, with this base graph, it is certain in advance that every voltage assignment specifies a bipartite graph.

A logical choice for the voltage group is the cyclic group  $\mathbb{Z}_n$ . In this case, the only plausible voltage assignment is to use a different voltage on every edge, as shown in Figure 15.4.2, since otherwise, there would be multi-edges. We observe that this solution is a generalization of Example 15.1.3.



**Figure 15.4.2** Specifying the complete bipartite graph  $K_{n,n}$ .

It is easy to see that this solution for odd  $n$  also works for even  $n$ . However, the theoretical criteria suggest that for even  $n$ , there might be a base graph with  $|V| = 1$  and  $|E| = \frac{n}{2}$ , that is, the base graph  $G = B_{n/2}$ . Assigning the  $\frac{n}{2}$  odd voltages in  $\mathbb{Z}_{2n}$  to the  $n$  self-loops forces the endpoints of every arc from a vertex in the specified digraph to have opposite parity. The net effect is that the specified graph is  $K_{n,n}$ , as illustrated in Figure 15.4.3 for the case  $n = 4$ .

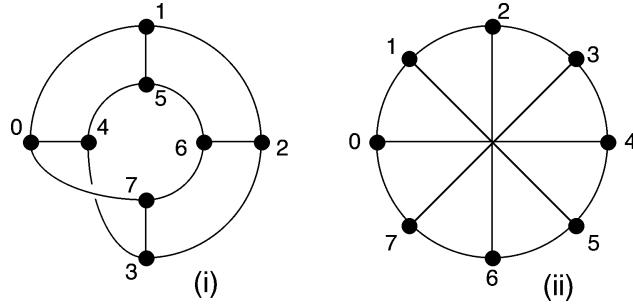


**Figure 15.4.3** Using  $B_2$  to specify the complete bipartite graph  $K_{4,4}$ .

**Example 15.4.3:** The Möbius ladder  $ML_n$  (depicted for  $n = 4$  in Figure 15.4.4) has

$$|\tilde{V}| = |V(ML_n)| = 2n \quad \text{and} \quad |\tilde{E}| = |E(ML_n)| = 3n$$

so that, just like the circular ladder  $CL_n$ , we have  $\gcd(|\tilde{V}|, |\tilde{E}|) = \gcd(2n, 3n) = n$ .



**Figure 15.4.4** Two drawings of the Möbius ladder  $ML_4$ .

Once again, the first base graphs  $G = (V, E)$  to consider have

$$|V| = \frac{|\tilde{V}|}{n} = 2 \quad \text{and} \quad |E| = \frac{|\tilde{E}|}{n} = 3$$

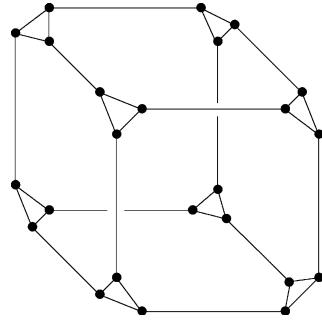
and as for  $CL_n$ , every vertex of  $ML_n$  has degree 3. Once again, the only two 3-regular graphs with  $|V| = 2$  and  $|E| = 3$  are the dipole  $D_3$  and the dumbbell graph. Unlike some circular ladders, every Möbius ladder is bipartite, so the dipole is a plausible candidate for a unified specification.

When  $ML_n$  is drawn so that it looks ladder-like, as in Figure 15.4.4(i), some of the symmetry remains hidden. However, Figure 15.4.4(ii) shows another way to visualize  $ML_4$  (and, through generalization, other Möbius ladders) so that some additional symmetry becomes apparent.

From the symmetric drawing, it is clear that  $ML_n$  can be specified as the Cayley graph  $C(\mathbb{Z}_{2n}, \{1, n\})$ , while adopting the bidirected-arc convention.

### Specification Problems with Complicated Solutions

For various computational problems, it would be desirable to perform a distributed computation on an  $n$ -dimensional hypercube network. However, there is a physical limit to the number of communications links that can converge on a microscopic-size processor. To overcome this problem, the processor at each corner of the hypercube is replaced by an  $n$ -cycle of processors, with one edge of the hypercube linking each processor in the  $n$ -cycle to another corner. Figure 15.4.5 shows the resulting network for dimension 3.



**Figure 15.4.5** The cube-connected-cycle graph  $CCC_3$ .

**DEFINITION:** The **cube-connected-cycle graph**  $CCC_n$  is the 1-skeleton of the polyhedron obtained by replacing each vertex of the  $n$ -cube by an  $n$ -cycle.

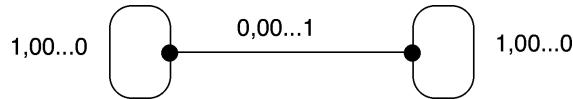
**Example 15.4.4:** The  $n$ -dimensional hypercube has  $2^n$  corners and  $n2^{n-1}$  edges. The cube-connected-cycle graph  $CCC_n$  replaces each vertex of the hypercube by an  $n$ -cycle. Thus,

$$|\tilde{V}| = |V(CCC_n)| = n2^n \quad \text{and} \quad |\tilde{E}| = |E(CCC_n)| = n2^{n-1} + n2^n = 3n2^{n-1}$$

so that

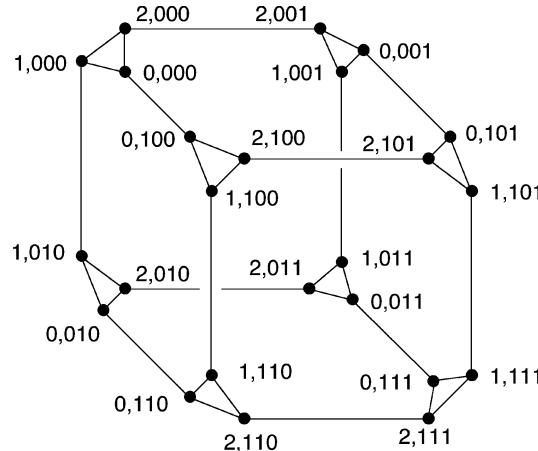
$$\gcd(|\tilde{V}|, |\tilde{E}|) = \gcd(n2^n, 3n2^{n-1}) = n2^{n-1}$$

which suggests that  $|V| = 2$  and  $|E| = 3$ . Since  $CCC_n$  is 3-regular, the dumbbell and the dipole  $D_3$  seem to be the two best candidates for a base graph. However, since  $CCC_n$  is not bipartite when  $n$  is odd, the best hope for a unified solution is to use the dumbbell as a base graph. Although not obvious, it turns out that a solution is the regular- $(\mathbb{Z}_n \otimes_{wr} \mathbb{Z}_2)$ -voltage graph in Figure 15.4.6, whose covering graph is two isomorphic copies of  $CCC_n$ .



**Figure 15.4.6** A  $(\mathbb{Z}_n \otimes_{wr} \mathbb{Z}_2)$ -voltage graph specifying  $CCC_n$ .

The specified graph for  $n = 3$  is shown in Figure 15.4.7.

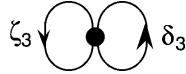


**Figure 15.4.7** The graph  $CCC_3$ , as specified by Figure 15.4.6.

The next two examples show that when the voltages on a graph are from a permutation group, it is necessary to declare explicitly whether they are to be regarded as regular voltages or as permutation voltages.

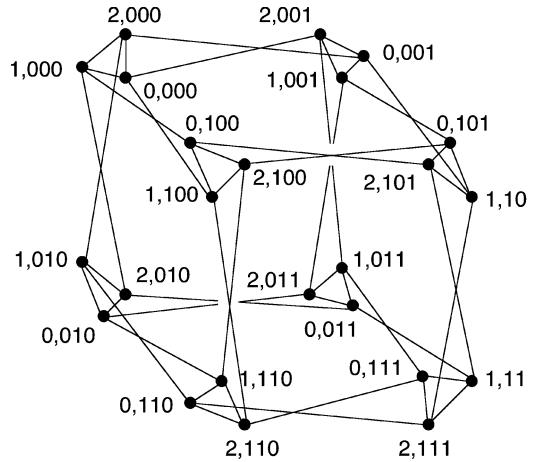
**Example 15.4.5:** If Figure 15.4.6 were interpreted as a permutation-voltage assignment, then the resulting specified graph would have only  $2^n$  vertices, rather than  $n2^n$ , as is clear from the definitions of the two different types of voltage-graph constructions of specified graphs. (See Exercises.)

**Example 15.4.6:** The permutation-voltage graph of Figure 15.3.5, which specified a deBruijn graph, is reproduced here as Figure 15.4.8.



**Figure 15.4.8** A regular-voltage graph specifying  $WBF_3$ .

Suppose it is now reinterpreted as a regular- $(\mathbb{Z}_3 \otimes_{wr} \mathbb{Z}_2)$ -voltage graph, instead of a permutation- $(\mathbb{Z}_3 \otimes_{wr} \mathbb{Z}_2)$ -voltage graph. Then the result, shown in Figure 15.4.9, is called the **3-dimensional wrapped-butterfly graph**  $WBF_3$ . Higher dimensional wrapped-butterfly graphs are used in parallel computation of fast Fourier transforms.



**Figure 15.4.9** The wrapped-butterfly graph  $WBF_3$ .

### EXERCISES for Section 15.4

In Exercises 15.4.1 through 15.4.8, design a voltage graph to specify the given graph or family of graphs.

- 15.4.1<sup>s</sup> The octahedral graph  $\mathcal{O}_3$ .
- 15.4.2 The family  $\mathcal{O}_n$  of octahedral graphs.
- 15.4.3 The graph  $Q_3 \times C_3$ .
- 15.4.4 The family  $Q_3 \times C_n$ .
- 15.4.5 The graph  $K_5 \times C_4$ .
- 15.4.6<sup>s</sup> The family  $K_5 \times C_n$ .
- 15.4.7 The graph  $K_5 \times K_5$ .
- 15.4.8<sup>s</sup> The graph  $K_5 + C_5$ .

**DEFINITION:** The **shuffle-exchange digraph**  $\tilde{SE}_n$  has as its vertex-set the length- $n$  bitstrings. There is an arc from each bitstring  $v$  to the bitstring  $\zeta_n(v)$  obtainable by cycle-shifting leftward one position. There is also a bidirected arc between each bitstring and the bitstring that can be obtained by changing its rightmost bit.

- 15.4.9 Construct a permutation-voltage graph that specifies the shuffle-exchange graph.
- 15.4.10 Give a counterexample to the converse of Theorem 15.4.4.
- 

## 15.5 INTERCONNECTION-NETWORK PERFORMANCE

One of the factors in a parallel architecture that affects the performance of an interconnection network is the distances between nodes. This parameter bears directly on the communication time during execution of a distributed algorithm.

### Diameter and Mean Pair-Distance

Both worst-case and average-case performance measures are of practical importance. Evaluation of these measures often leads to summation formulas that apply to an infinite family of Cayley graphs.

**REVIEW FROM §1.4:** The **diameter** of a graph  $G$  is the maximum of the distances  $d(u, v)$ , taken over all pairs of vertices of that graph. It is denoted  $\text{diam}(G)$ .

**DEFINITION:** The **mean pair-distance** of a graph  $G$  is the average of the distances  $d(u, v)$ , taken over all pairs of distinct vertices. It is denoted  $d_{\text{avg}}(G)$ .

**Proposition 15.5.1.** *Let  $G$  be a Cayley graph. Then the diameter  $\text{diam}(G)$  equals the maximum of the distances from the identity vertex to the other vertices.*

**Proof:** Since Cayley graphs are vertex-transitive, it follows that the maximum distance from any vertex is the same as the maximum distance from any other. Thus, the maximum distance from any vertex is the diameter of  $G$ .  $\diamond$

**Proposition 15.5.2.** *Let  $G$  be a Cayley graph. Then  $d_{\text{avg}}(G)$  equals the average of the distances from the identity vertex to the other vertices.*

**Proof:** Suppose that  $|V_G| = n$ . Then

$$d_{\text{avg}}(G) = \frac{1}{n(n-1)} \sum_{v \in V_G} \sum_{w \in V_G - \{v\}} d(v, w)$$

Since  $G$  is vertex-transitive, the value of the double sum is

$$n \sum_{w \neq 0} d(0, w)$$

The conclusion follows easily.  $\diamond$

**Example 15.5.1:** The cycle graph  $C_n$  can be represented as the Cayley graph of  $\mathbb{Z}_n$  with generating set  $\{1\}$ . Since  $d(0, j) = \min\{j, n-j\}$ , the maximum distance from vertex 0 to another vertex is  $\lfloor \frac{n}{2} \rfloor$ . Therefore,

$$\text{diam}(C_n) = \left\lfloor \frac{n}{2} \right\rfloor.$$

When  $n$  is odd, the sum of the distances from vertex 0 equals

$$\begin{aligned}\sum_{j=1}^{n-1} d(0, j) &= 2 \left( 1 + 2 + \cdots + \left( \frac{n-1}{2} \right) \right) \\ &= 2 \left( \frac{\frac{n+1}{2}}{2} \right) = \frac{n+1}{2} \cdot \frac{n-1}{2} = \frac{n^2 - 1}{4}\end{aligned}$$

When  $n$  is even, the sum of the distances from vertex 0 equals

$$\begin{aligned}\sum_{j=1}^{n-1} d(0, j) &= 2 \left( 1 + 2 + \cdots + \left( \frac{n-1}{2} - 1 \right) \right) + \frac{n}{2} \\ &= 2 \left( \frac{\frac{n}{2}}{2} \right) + \frac{n}{2} = \frac{n}{2} \cdot \frac{n-2}{2} + \frac{n}{2} \\ &= \frac{n^2 - 2n}{4} + \frac{n}{2} = \frac{n^2}{4}\end{aligned}$$

Using Proposition 15.5.2, it follows that

$$d_{\text{avg}}(C_n) = \begin{cases} \frac{n+1}{4} & \text{if } n \text{ odd} \\ \frac{n^2}{4(n-1)} & \text{if } n \text{ even} \end{cases}$$

**Example 15.5.2:** The distance in the hypercube graph  $Q_n$  from the bitstring  $00 \dots 0$  to any other bitstring equals the number of 1-bits in the other bitstring. It follows by Proposition 15.5.1 that

$$\text{diam}(Q_n) = n$$

and that there are  $\binom{n}{j}$  vertices at distance  $j$ , for  $j = 1, \dots, n$ . Thus, the sum of the distances equals

$$\sum_{j=1}^n j \cdot \binom{n}{j} = n \cdot \sum_{j=1}^{n-1} \binom{n-1}{j} = n \cdot 2^{n-1}$$

By Proposition 15.5.2, it follows that

$$d_{\text{avg}}(Q_n) = \frac{n \cdot 2^{n-1}}{2^n - 1}$$

### EXERCISES for Section 15.5

In Exercises 15.5.1 through 15.5.15, calculate the diameter of the given graph or family of graphs.

- 15.5.1<sup>s</sup> The circular ladder  $CL_6$ .
- 15.5.2 The circular ladder  $CL_7$ .
- 15.5.3 The circular ladder  $CL_{2n}$ .
- 15.5.4 The circular ladder  $CL_{2n+1}$ .
- 15.5.5<sup>s</sup> The Möbius ladder  $ML_6$ .

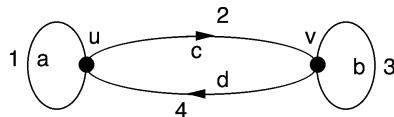
- 15.5.6 The Möbius ladder  $ML_7$ .
- 15.5.7 The Möbius ladder  $ML_{2n}$ .
- 15.5.8 The Möbius ladder  $ML_{2n+1}$ .
- 15.5.9 The wraparound mesh  $C_4 \times C_4$ .
- 15.5.10<sup>s</sup> The wraparound mesh  $C_5 \times C_5$ .
- 15.5.11 The wraparound mesh  $C_{2n} \times C_{2n}$ .
- 15.5.12 The wraparound mesh  $C_{2n+1} \times C_{2n+1}$ .
- 15.5.13 The join  $C_5 + C_5$ .
- 15.5.14<sup>s</sup> The join  $K_5 + C_5$ .
- 15.5.15 The join  $K_5 + K_5$ .

*In Exercises 15.5.16 through 15.5.30, calculate the mean pair-distances of the given graph or family of graphs.*

- 15.5.16<sup>s</sup> The circular ladder  $CL_6$ .
- 15.5.17 The circular ladder  $CL_7$ .
- 15.5.18 The circular ladder  $CL_{2n}$ .
- 15.5.19 The circular ladder  $CL_{2n+1}$ .
- 15.5.20<sup>s</sup> The Möbius ladder  $ML_6$ .
- 15.5.21 The Möbius ladder  $ML_7$ .
- 15.5.22 The Möbius ladder  $ML_{2n}$ .
- 15.5.23 The Möbius ladder  $ML_{2n+1}$ .
- 15.5.24<sup>s</sup> The wraparound mesh  $C_4 \times C_4$ .
- 15.5.25 The wraparound mesh  $C_5 \times C_5$ .
- 15.5.26 The wraparound mesh  $C_{2n} \times C_{2n}$ .
- 15.5.27 The wraparound mesh  $C_{2n+1} \times C_{2n+1}$ .
- 15.5.28 The join  $C_5 + C_5$ .
- 15.5.29<sup>s</sup> The join  $K_5 + C_5$ .
- 15.5.30 The join  $K_5 + K_5$ .

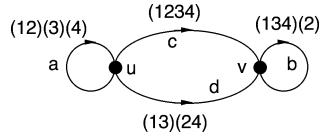
## 15.6 SUPPLEMENTARY EXERCISES

- 15.6.1 Draw the covering graph specified by this voltage graph with cyclic voltages in the group  $Z_5$ . Suppose the voltages were from an arbitrary cyclic group  $Z_n$ . Why is the covering graph 2-factorable (see §9.4)?



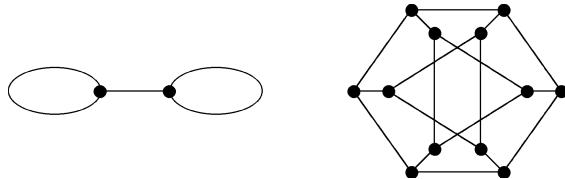
15.6.2 Give a regular voltage assignment on  $K_{3,3}$  and a labeling of the vertices of  $CL_6$  as the covering graph specified by that voltage graph.

15.6.3 Construct the covering graph for the following permutation voltage graph.

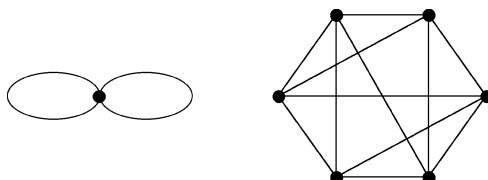


15.6.4 Give a permutation voltage assignment on  $D_3$  that specifies  $CL_6$  as the covering graph.

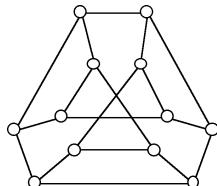
15.6.5 Label the dumbbell with cyclic voltages in  $Z_6$  so that the covering graph is isomorphic to the graph at the right.



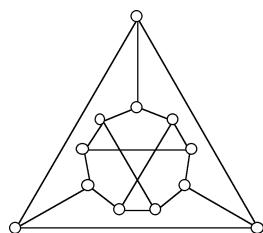
15.6.6 Label the bouquet with cyclic voltages in  $Z_6$  so that the covering graph is isomorphic to the graph at the right.



15.6.7 The graph below is called **Franklin's graph**. Draw a 4-vertex voltage graph for it with vertex labels  $w, x, y$ , and  $z$ , and label the vertices of Franklin's graph as the specified covering graph.



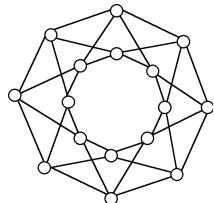
15.6.8 The graph below is called the **Tietze graph**. Draw a 4-vertex cyclic voltage graph for it, and label the vertices  $w, x, y$ , and  $z$ . Label the vertices of the Tietze graph as the specified covering graph.



**15.6.9** Draw three mutually non-isomorphic 3-regular, 10-vertex simple graphs, each vertex transitive. (Optional hint: use cyclic voltage graphs.) Sketch proofs that your three graphs are mutually non-isomorphic.

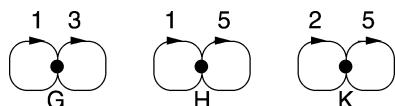
**15.6.10** Draw a 2-vertex  $Z_8$ -voltage graph whose covering graph is isomorphic to the graph of Figure 15.6.1.

**15.6.11** Draw a 1-vertex  $Z_2 \times Z_8$ -voltage graph whose covering graph is isomorphic to the graph of Figure 15.6.1.

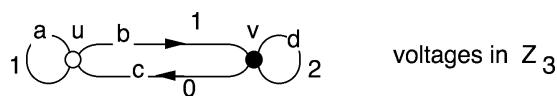


**Figure 15.6.1**

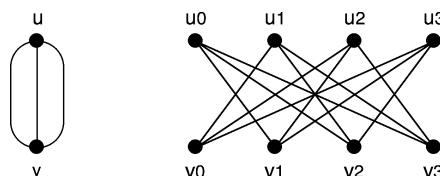
**15.6.12** Consider the covering graphs for the three voltage graphs below with voltages in  $Z_{11}$ . Decide whether some pair of them is isomorphic or whether they are mutually non-isomorphic.



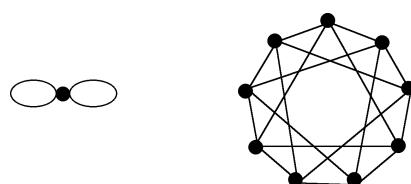
**15.6.13** a. Draw the covering graph for this voltage graph. b. Then draw a 1-vertex voltage graph whose covering graph is isomorphic to that covering graph of part a.



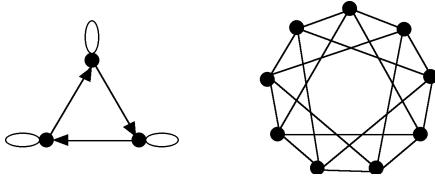
**15.6.14** Assign cyclic voltages in  $Z_4$  to the graph at the left so that the graph at the right is the covering graph.



**15.6.15** Assign cyclic voltages in  $Z_9$  to the graph at the left so that the graph at the right is the covering graph.



**15.6.16** Assign cyclic voltages in  $Z_3$  to the graph at the left so that the graph at the right is the covering graph.



## GLOSSARY

**base digraph of a voltage graph  $\langle G, \alpha \rangle$ :** the digraph  $G$ .

**base graph of a voltage graph  $\langle G, \alpha \rangle$ :** the underlying graph of the *base digraph*.

**bidirected-arc convention:** in a *Cayley digraph* or *Cayley graph*, if  $y$  is a generator of order 2, then the pair of arcs that would join vertex  $b$  to vertex  $by$  and vertex  $by$  to vertex  $b$  may be replaced by a single bidirected arc between  $b$  and  $by$ .

**Cayley digraph  $\vec{C}(\mathcal{B}, X)$ :** for a group  $\mathcal{B} = \langle B, \cdot \rangle$  with generating set  $X$ , the digraph with vertex-set  $V_{\vec{C}(\mathcal{B}, X)} = B$  and arc-set  $E_{\vec{C}(\mathcal{B}, X)} = \{x_b \mid x \in X, b \in B\}$ , such that arc  $x_b$  joins vertex  $b$  to vertex  $bx$ . (*Bidirected arcs* are sometimes used for generators of order 2.)

**Cayley graph  $C(\mathcal{B}, X)$ :** for a group  $\mathcal{B} = \langle B, \cdot \rangle$  with generating set  $X$ , the underlying graph of the *Cayley digraph*  $\vec{C}(\mathcal{B}, X)$ .

**Cayley graph:** any graph  $G$  such that there exists a group  $\mathcal{B}$  and a generating set  $X$  such that  $G$  is isomorphic to the *Cayley graph*  $C(\mathcal{B}, X)$ .

**clarifying-edge-graphic convention for voltage graphs:** each edge of the *base graph* is drawn in a unique graphic, and all the edges in its *fiber* are drawn in that same graphic; moreover, in the *covering digraph and graph*, edge labels are commonly omitted altogether.

**clarifying-vertex-graphic convention for voltage graphs:** each vertex of the *base graph* is drawn in a unique graphic, and all the vertices in its *fiber* are drawn in that same graphic; moreover, each vertex in the *fiber* is labeled by its subscript only, with its *mainscript* omitted.

**covering digraph for a voltage graph  $\langle G, \alpha \rangle$ :** the graph  $G^\alpha = (V^\alpha, E^\alpha)$ .

**covering graph for a voltage graph  $\langle G, \alpha \rangle$ :** the underlying graph of the *covering digraph*  $G^\alpha$ .

**cube-connected-cycle graph  $CCC_n$ :** the vertex-set is  $\{(k, b) \mid k \in Z_n, b \in Z_2^n\}$ , and two vertices  $(k, b)$  and  $(m, c)$  are adjacent if and only if  $k = m$  while  $b$  and  $c$  differ in only one bit or  $b = c$  and  $k = m \pm 1$ ; isomorphic to the 1-skeleton of the polyhedron obtained by chopping the corners off an  $n$ -dimensional hypercube.

**(left) cycle-shift:** a permutation that transfers the leftmost bit of a bitstring to the right end.

**cyclic voltage (or  $Z_n$ -voltage):** a label on an arc by a number  $j(\text{mod } n)$  (i.e., by an element of the cyclic group  $Z_n$ ).

**deBruijn permutation:** a permutation on the set of bitstrings that transfers the leftmost bit of a bitstring to the right end, thus shifting each of the other  $n - 1$  bits one position to the left, and then adds a 1-bit to the rightmost bit.

**diameter  $\text{diam}(G)$  of a graph  $G$ :** the maximum of the distances  $d(u, v)$ , taken over all pairs of vertices of  $G$ .

**fiber  $\tilde{e}$  over an edge  $e$  of a voltage graph:** (1) for *regular voltages* in a group  $\mathcal{B}$ , the edge set  $\{e_b \mid b \in \mathcal{B}\}$  in the covering graph  $G^\alpha$ ; (2) for *permutation voltages* in a permutation group  $\mathcal{P} = [P : Y]$ , the edge set  $\{e_y \mid y \in Y\}$  in  $G^\alpha$ .

**fiber  $\tilde{v}$  over a vertex  $v$  of a voltage graph:** (1) for *regular voltages* in a group  $\mathcal{B}$ , the vertex set  $\{v_b \mid b \in \mathcal{B}\}$  in the covering graph  $G^\alpha$ ; (2) for *permutation voltages* in a permutation group  $\mathcal{P} = [P : Y]$ , the vertex set  $\{v_y \mid y \in Y\}$  in  $G^\alpha$ .

**mainscript of a subscripted variable  $X_j$ :** the variable name  $X$ , i.e., the part without the subscript.

**mean pair-distance  $d_{\text{avg}}(G)$  of a graph  $G$ :** the average of the distances  $d(u, v)$ , taken over all pairs of distinct vertices in  $G$ .

**natural projection of a voltage-graph construction:** the “subscript-erasing” graph mapping that carries every vertex fiber  $\tilde{v}$  to its base vertex  $v$  and every edge fiber  $\tilde{e}$  to its base edge  $e$ .

**natural transformation  $\varphi_a$  on a voltage-specified covering graph  $G^\alpha$ :** the graph automorphism with vertex function  $v_b \mapsto v_{ab}$  and edge function  $e_b \mapsto e_{ab}$ , for all  $b \in B$ .

**permutation voltage:** a label on an arc by an element of a permutation group  $\mathcal{P} = [P : Y]$ .

**regular voltage:** a label on an arc by an element of a group  $\mathcal{B}$ .

**shuffle-exchange digraph  $\vec{SE}_n$ :** the vertex-set is the length- $n$  bitstrings; there is an arc from each bitstring  $v$  to the bitstring  $\zeta_n(v)$  obtainable by cycle-shifting leftward one position. There is also a bidirected arc between each bitstring and the bitstring that can be obtained by changing its rightmost bit.

**unified specification for a family of graphs or digraphs:** usually, a single *base graph* and a single *voltage-assignment* formula that specifies every graph or digraph in the family.

**vertex-transitive graph:** a graph such that for all vertex pairs  $u, v$ , there is a graph automorphism that maps  $u$  to  $v$ .

**voltage:** on an arc of a digraph, a label by an element of a group  $\mathcal{B}$  or of a permutation group  $\mathcal{P} = [P : Y]$ .

**voltage assignment:** on a digraph, a function  $\alpha$  that labels every arc with a voltage from a group  $\mathcal{B}$  or from a permutation group  $\mathcal{P} = [P : Y]$ .

**voltage graph:** a pair  $\langle G, \alpha \rangle$  such that  $G$  is a digraph and  $\alpha$  is a *voltage assignment* on  $G$ .

**voltage group:** the group  $\mathcal{B}$  in which the *voltages* are assigned.

# Chapter 16

---

## NONPLANAR LAYOUTS

- 16.1 Representing Imbeddings by Rotations**
  - 16.2 Genus Distribution of a Graph**
  - 16.3 Voltage-Graph Specification of Graph Layouts**
  - 16.4 Non-KVL Imbedded Voltage Graphs**
  - 16.5 The Heawood Map-Coloring Problem**
- 

### INTRODUCTION

The most important quantification of nonplanarity of a graph is the number of handles or crosscaps one must add to the plane to eliminate all the crossings. There are two elementary ways to represent an arbitrary graph drawing on an arbitrary surface. When specifying imbeddings devoid of adequate symmetry, one resorts to these elementary representations. One is to draw the graph on a flat polygon representation (§8.5) of the surface. Another way, completely combinatorial, is to write the cycle of edge-ends incident on each vertex, as described in §16.1. The list of such cycles, called a *rotation system*, completely specifies an imbedding. The set of all such rotation systems for a graph corresponds to the set of all imbeddings, thereby yielding information about the distribution of the imbeddings according to the genus of the imbedding surface.

When a graph is sufficiently rich in symmetry that it can be specified by a small voltage graph, there is an extension of the voltage-graph construction. The idea is that a voltage graph drawn on a surface can specify a symmetric imbedding of the covering graph. Voltage graphs have been used (often in a dual guise) in problem solutions. They are used frequently to find an imbedding of a graph into a surface with the smallest possible number of handles. Solving this imbedding-minimization problem for complete graphs was the main part of the solution by Ringel and Youngs of the Heawood problem of finding the chromatic number of all the higher-order closed surfaces.

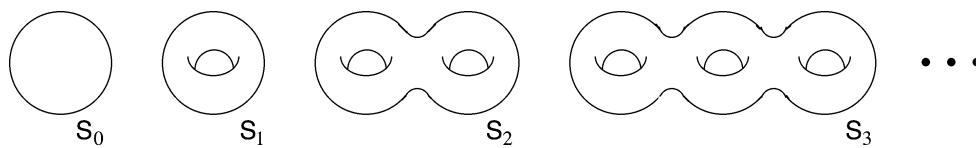
## 16.1 REPRESENTING IMBEDDINGS BY ROTATIONS

We have seen that a graph imbedding can be represented by a drawing on a flat polygon representation of a surface. It is helpful also to have a precise combinatorial form of representation of a graph imbedding that does not depend on drawing pictures.

### Review of Closed Surfaces and Flat Polygon Drawings

REVIEW FROM §8.2 OF THE CLASSIFICATION OF CLOSED SURFACES:

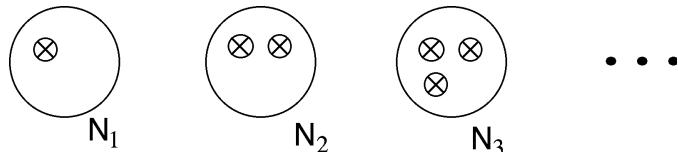
**Theorem 8.2.2.** *Classification of Closed Orientable Surfaces.* Every closed orientable surface is topologically equivalent to exactly one of the surfaces in the infinite sequence  $S_0, S_1, S_2, \dots$ .



**Figure 16.1.1** The sequence of closed orientable surfaces.

REVIEW FROM §8.2: A **crosscap** is a portion of a surface that is topologically equivalent to a Möbius band. In a drawing of a nonorientable surface, a crosscap is commonly represented by a circle with an inscribed crossmark.

**Theorem 8.2.3.** *Classification of Closed Non-Orientable Surfaces.* Every closed non-orientable surface is topologically equivalent to exactly one of the surfaces in the infinite sequence  $N_1, N_2, N_3, \dots$ .



**Figure 16.1.2** The sequence of closed non-orientable surfaces.

REVIEW FROM §8.5: A **flat polygon representation** of a surface  $S$  is a drawing of a polygon with markings to match its sides in pairs, such that when the sides are pasted together as the markings indicate, the resulting surface obtained is topologically equivalent to  $S$ .

### Parametrization

It was explained in §8.1 how the parametrization of each edge of a graph by the unit interval  $[0, 1]$  creates a distinction between its *0-end* and its *1-end*. Such a distinction facilitates the combinatorial description of a graph imbedding.

**NOTATION:** Let  $e$  be an edge of a graph. Then  $e^+$  denotes the 0-end of edge  $e$ , and  $e^-$  denotes the 1-end. The mnemonic for this notation is that starting at  $e^+$  means going in the positive direction from 0 to 1. The usual rules of sign composition apply, so that

$$e^{-\sigma} = \begin{cases} e^+ & \text{if } \sigma = - \\ e^- & \text{if } \sigma = + \end{cases}$$

The set of edge-ends of a graph  $G$  is denoted  $E_G^\pm$ , or sometimes, simply  $E^\pm$ .

**NOTATION:** In a *drawing* of an undirected parametrized graph, the *parametrization arrow* points to the 1-end.

**Remark:** A directional arrow on an edge, if it exists, is independent of the parametrization arrow; that is, both arrows may point the same way or opposite ways. The distinction in meaning is that a parametrization arrow gives different names to the two possible senses in which an edge can be traversed, whereas a direction arrow forbids passage in one of those two senses. It is not necessary here to display both arrows simultaneously. However, in circumstances where it is necessary to display both arrows simultaneously, it is helpful to use graphically different types of arrowhead.

## Rotations and Rotation Systems

**DEFINITION:** A **rotation at a vertex  $v$**  of a graph  $G$  is a cyclic permutation of the edge-ends incident on  $v$ .

**DEFINITION:** A **rotation system** for a graph  $G$  is an assignment of a rotation to every vertex.

**TERMINOLOGY:** A rotation system is also regarded as a permutation on the set  $E_G^\pm$  of edge-ends. This permutation is also called a *rotation system*.

**Remark:** It is assumed here that the orientable surfaces we are discussing are imbedded in Euclidean 3-space. The intent of this assumption is to impose a fixed distinction between *clockwise* and *countrerclockwise* on the surface.

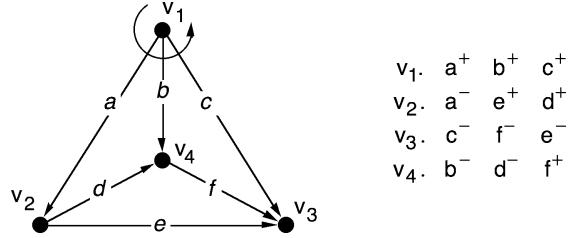
**TERMINOLOGY:** An orientable surface is **oriented** if one of the orientations *clockwise* or *countrerclockwise* is designated as *preferred*.

**DEFINITION:** Let  $h : G \rightarrow S_g$  be a graph imbedding in an oriented surface. The **induced rotation at  $v$**  is the cyclic permutation  $\rho_h(v)$  of edge-ends incident on  $v$  in the order in which they are encountered in a traversal around  $v$  in the preferred orientation.

**DEFINITION:** Let  $h : G \rightarrow S_g$  be a graph imbedding in an oriented surface. The **induced rotation system  $\rho_h$**  is the function that assigns to each vertex  $v \in V_G$  the rotation  $\rho_h(v)$ .

**DEFINITION:** Let  $\rho$  be a rotation system for a graph  $G$ . The **rotation table for  $\rho$** , denoted  $T_\rho$ , has one row for each vertex of  $G$ . The content of each row of the table is the name of a vertex, followed by a complete list of the edge-ends incident on that vertex, in an order consistent with  $\rho(v)$ . (It is a common custom to write each row in lexicographic order.)

**Example 16.1.1:** Figure 16.1.3 shows an imbedding of  $K_4$  in the sphere and a table representing the rotation system it induces. The direction arc around vertex  $v_1$  indicates that the preferred orientation is counterclockwise (globally).

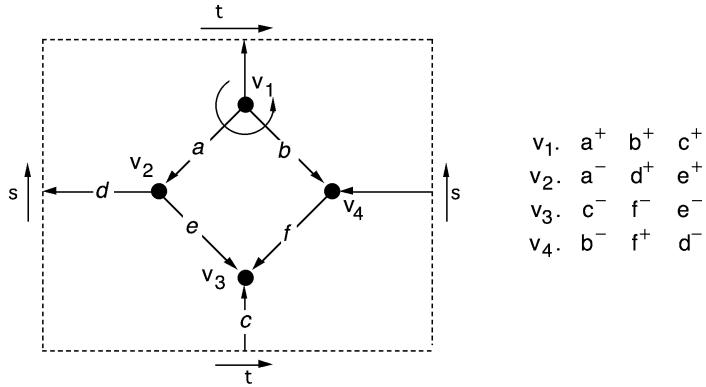


**Figure 16.1.3** An imbedding  $g : K_4 \rightarrow S_0$  and its induced rotation system  $\rho_g$ .

**Proposition 16.1.1.** Let  $h : G \rightarrow S$  be an oriented graph imbedding, and let  $\rho_h$  be the induced rotation. Then  $\prod_{v \in V_G} \rho_h(v)$  is a permutation on the edge-ends of  $G$ .

**Proof:** Every edge-end occurs at exactly one vertex. Thus, the product of the rotations over all vertices is the disjoint-cycle form of a permutation.  $\diamond$

**Example 16.1.2:** Figure 16.1.4 shows an imbedding of  $K_4$  in the torus and its rotation system. The only difference from the rotation system induced by the imbedding  $g : K_4 \rightarrow S_0$  is that the cyclic orders of the  $v_2$ -row and of the  $v_4$ -row are reversed; yet, despite this seemingly small difference, one imbedding is in the sphere and the other in the torus.



**Figure 16.1.4** An imbedding  $h : K_4 \rightarrow S_1$  and its induced rotation system  $\rho_h$ .

### Dual Rotations

**DEFINITION:** Let  $\rho : E_G^\pm \rightarrow E_G^\pm$  be a rotation system on a graph  $G$ . The **dual rotation** (or **circulation**)  $\rho^*$  is the permutation  $\rho^* : E_G^\pm \rightarrow E_G^\pm$  given by the rule  $e^\sigma \mapsto \rho(e^{-\sigma})$ .

Invoking the following algorithm is the practical way to use a rotation table  $T_\rho$  to determine the edge-end  $\rho^*(e^\sigma)$ .

**Algorithm 16.1.1: Circulation**

*Input:* a rotation table  $T_\rho$ , an edge-end  $e^\sigma$   
*Output:* the edge-end  $\rho^*(e^\sigma)$

Find the (only) row  $v$  in table  $T_\rho$  that contains the edge-end  $e^{-\sigma}$ .  
 Return (whatever edge-end follows  $e^{-\sigma}$  in row  $v$ ).

**Example 16.1.3:** In the imbedding  $g : K_4 \rightarrow S_0$  of Example 16.1.1,

$$\rho^* = (a^+ e^+ c^-) (a^- b^+ d^-) (b^- c^+ f^-) (d^+ f^+ e^-)$$

**Example 16.1.4:** In the imbedding  $h : K_4 \rightarrow S_1$  of Example 16.1.2,

$$\rho^* = (a^- b^+ f^+ e^-) (a^+ d^+ b^- c^+ f^- d^- e^+ c^-)$$

Observe that the four cycles of  $\rho^*$  in Example 16.1.3 correspond to the boundary walks of the four faces in Figure 16.1.0 and that the two cycles of  $\rho^*$  in Example 16.1.4 correspond to the boundary walks of the two faces in Figure 16.1.4. The Heffter-Edmonds theorem below asserts that this phenomenon always occurs.

**TERMINOLOGY:** A sequence of edge-ends is said to *coincide* with a walk in a graph if the order of the edge-ends is the order in which they occur on a traversal of the walk.

**Theorem 16.1.2 (Heffter-Edmonds Theorem) [He1891], [Ed60].** *In a cellular imbedding  $h : G \rightarrow S_g$ , each cycle of the induced dual rotation  $\rho_h^*$  coincides with a face-boundary walk of the imbedding  $h$ .*

**Proof:** The edge  $\rho_h^*(e)$  is defined to be  $\rho(e^{-\sigma})$ , which is precisely the next edge-end after  $e^{-\sigma}$  in the boundary walk that traverses edge  $e$  from  $e^\sigma$  to  $e^{-\sigma}$ .  $\diamond$

**TERMINOLOGY:** In view of Theorem 16.1.2, the construction from a rotation system  $\rho$  of the cycles of its induced circulation  $\rho^*$  is commonly called **face-tracing**.

**Algorithm 16.1.2: Face-Tracing**

*Input:* edge-end list  $E^\pm$ , rotation table  $T_\rho$   
*Output:* list of all cycles of the circulation  $\rho^*(e^\sigma)$

```

{Initialize} mark all edge-ends unused
While any unused edge-ends remain
  Choose next (lex order) unused edge-end  $e_1^\sigma$  from  $E^\pm$ 
  Start new cycle by writing left parenthesis "("
   $e := e_1^\sigma$ 
  Repeat
    Write  $e$  next in current cycle
     $e = \rho^*(e)$  {call Circulation Algorithm}
    Until  $e = e_1^\sigma$ 
  Close current cycle by writing right parenthesis ")"
Return
  
```

### The Induced Surface of a Rotation System

NOTATION: Let  $z$  be a cyclic permutation. Then  $\text{len}(z)$  denotes its length.

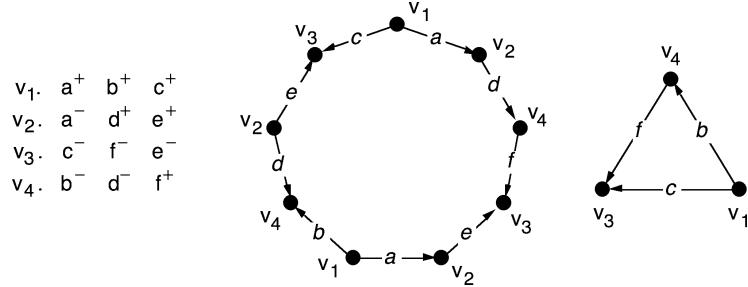
DEFINITION: Let  $\rho$  be a rotation system for a graph  $G$ . For each cycle  $z$  of the dual rotation  $\rho^*$ , let  $p_z$  be a polygon with  $\text{len}(z)$  sides, labeled consecutively by the edges of the cycle in graph  $G$  that corresponds to the permutation cycle  $z$ . The set of all such polygons is called the ***polygon set*** for rotation  $\rho$ .

**Proposition 16.1.3.** *Let  $\rho$  be a rotation system for a graph  $G$ , and let  $\{p_z\}$  be the polygon set for  $\rho$ . Then the list of all boundary walks of all the polygons in  $\{p_z\}$  mentions each edge of graph  $G$  exactly twice.*

**Proof:** This follows immediately from the fact that rotation  $\rho$  is a permutation of the set of edge-ends.  $\diamond$

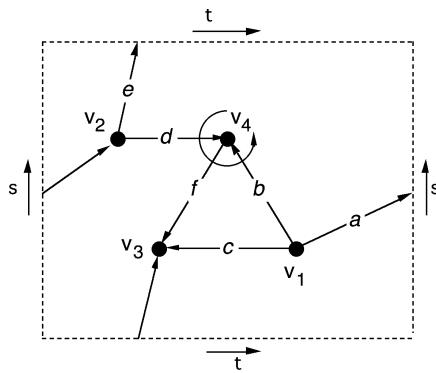
DEFINITION: Let  $\rho$  be a rotation system for a graph  $G$ . The ***surface induced by  $\rho$***  is the topological space  $S(\rho)$  formed from the graph  $G$  and the polygon set for  $\rho$  by fitting the boundary walk of each polygon to the corresponding cycle of  $G$ .

**Example 16.1.5:** Figure 16.1.5 shows a rotation table for  $K_4$  and the set of labeled polygons that arise when the induced surface is constructed from that table. Observe that this rotation table differs from the table in Example 16.1.2 only in the bottom row.



**Figure 16.1.5** A rotation system for  $K_4$  and its labeled polygons.

Whereas the imbedding of  $K_4$  in Example 16.1.2 has as its faces a 4-gon and an 8-gon, the faces of this imbedding are a 3-gon and a 9-gon. Figure 16.1.6 below shows the resulting imbedding in the torus.



**Figure 16.1.6** An imbedding  $h : K_4 \rightarrow S_1$  and its induced rotation system  $\rho_h$ .

**DEFINITION:** Let  $\rho$  be a rotation system for a graph  $G$ . The **imbedding induced by  $\rho$**  is the oriented imbedding of  $G$  into the induced surface  $S(\rho)$  whose face-boundary walks coincide with the cycles of the circulation  $\rho^*$ . It is denoted  $\iota_\rho$ .

### Counting the Imbeddings of a Graph

**TERMINOLOGY:** An imbedding on an oriented surface is called an *oriented imbedding*.

**DEFINITION:** Let  $h : G \rightarrow S$  and  $h' : G \rightarrow S'$  be two oriented imbeddings of a graph  $G$ . They are **equivalent imbeddings** if they have exactly the same set of face-boundary walks.

**Proposition 16.1.4.** *Let  $h : G \rightarrow S$  and  $h' : G \rightarrow S'$  be two oriented imbeddings such that  $\rho_h = \rho_{h'}$ . Then  $h$  and  $h'$  are equivalent imbeddings.*

**Proof:** It follows from Theorem 16.1.3 that oriented imbeddings with the same induced rotation system have the same set of face-boundary walks.  $\diamond$

**NOTATION:**  $\gamma_g(G)$  is the number of rotation systems  $\rho$  such that the induced surface has genus  $g$ .

**Theorem 16.1.5.** *Let  $G$  be a connected graph. The correspondence between equivalence classes of oriented imbeddings of  $G$  and rotation systems of  $G$  is a bijection.*

**Proof:** Proposition 16.1.4 establishes that the correspondence is one-to-one. Since every rotation system  $\rho$  on a connected graph  $G$  induces a surface  $S(\rho)$  and an imbedding  $\iota_\rho : G \rightarrow S(\rho)$  whose induced rotation system is  $\rho$ , the correspondence is onto.  $\diamond$

**Remark:** Given a rotation system  $\rho$ , the genus of the induced imbedding surface  $S(\rho)$  can be deduced from the formula  $|V| - |E| + |F| = 2 - 2g$ , by substituting the number of cycles of the circulation  $\rho^*$  for  $|F|$ . Thus, it is not necessary to construct the surface  $S(\rho)$  to determine its genus.

**Theorem 16.1.6.** *Let  $G$  be a connected graph. Then*

$$\sum_{g=0}^{\infty} \gamma_g(G) = \prod_{v \in V_G} [\deg(v) - 1]!$$

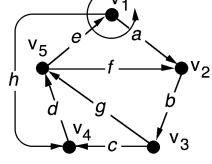
**Proof:** The left side is the total number of rotation systems of graph  $G$ . The rotations at each vertex  $v$  are in one-to-one correspondence with the ways of arranging the edge-ends at  $v$  into a cycle, so there are  $[\deg(v) - 1]!$  rotations at  $v$ . Thus, the number of rotation systems of  $G$  equals the value of the product on the right side.  $\diamond$

**DEFINITION:** The **genus distribution** of a graph  $G$  is the function that assigns to each nonnegative integer  $n$  the number  $\gamma_n(G)$ .

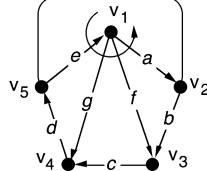
**Example 16.1.6:** By Theorem 16.1.6, it follows that  $K_4$  has 16 inequivalent imbeddings. All of them can be obtained by systematically listing the rotation systems for  $K_4$ . Two of them are imbeddings in  $S_0$  with four 3-sided faces, six of them are imbeddings in  $S_1$  with a 4-sided face and an 8-sided face, and eight of them are imbeddings in  $S_1$  with a 3-sided face and a 9-sided face. Rather than exhaustively applying the Face-Tracing Algorithm to all 16 rotation systems, it is possible to use the symmetry of the graph and Examples 16.1.1, 16.1.2, and 16.1.5 to complete this calculation.

**EXERCISES for Section 16.1**

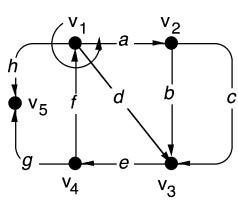
In Exercises 16.1.1 through 16.1.4, write the rotation system for the given imbeddings in the sphere  $S_0$ .

16.1.1<sup>s</sup>

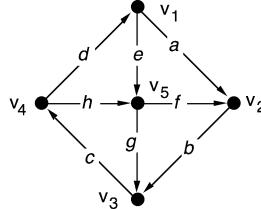
16.1.2



16.1.3

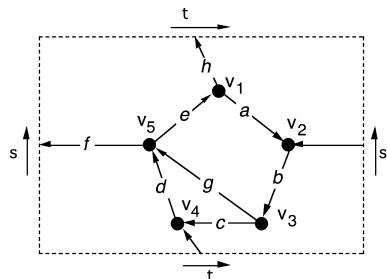


16.1.4

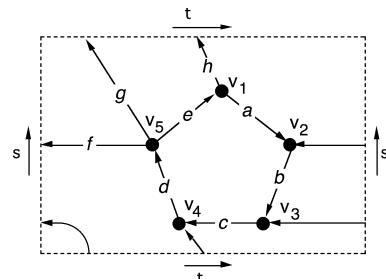


In Exercises 16.1.5 and 16.1.6, write the rotation system for the given imbeddings in the torus  $S_1$ .

16.1.5



16.1.6



In Exercises 16.1.7 through 16.1.12, for the given rotation system, (a) draw the graph, and (b) write the circulation.

16.1.7    u.  $a^+$   $b^+$   $d^+$   
 v.  $b^-$   $c^-$   $e^+$   
 w.  $a^-$   $f^+$   $c^+$   
 x.  $d^-$   $g^-$   $i^+$   
 y.  $e^-$   $h^+$   $g^+$   
 z.  $f^-$   $i^-$   $h^-$

16.1.8    u.  $a^+$   $b^+$   $d^+$   
 v.  $b^-$   $e^+$   $c^-$   
 w.  $a^-$   $f^+$   $c^+$   
 x.  $d^-$   $g^-$   $i^+$   
 y.  $e^-$   $h^+$   $g^+$   
 z.  $f^-$   $i^-$   $h^-$

16.1.9    u.  $a^+$   $b^+$   $d^+$   
 v.  $b^-$   $e^+$   $c^-$   
 w.  $a^-$   $f^+$   $c^+$   
 x.  $d^-$   $g^-$   $i^+$   
 y.  $e^-$   $g^+$   $h^+$   
 z.  $f^-$   $i^-$   $h^-$

16.1.10

v.  $a^+$   $c^-$   $b^+$   
 w.  $a^-$   $d^+$   $e^+$   
 x.  $e^-$   $g^+$   $h^+$   
 y.  $c^+$   $h^-$   $f^+$   
 z.  $b^-$   $f^-$   $g^-$   $d^-$

16.1.11<sup>s</sup>

v.  $a^+$   $c^-$   $b^+$   
 w.  $a^-$   $d^+$   $e^+$   
 x.  $e^-$   $g^+$   $h^+$   
 y.  $c^+$   $h^-$   $f^+$   
 z.  $b^-$   $f^-$   $d^-$   $g^-$

16.1.12

v.  $a^+$   $c^-$   $b^+$   
 w.  $a^-$   $d^+$   $e^+$   
 x.  $e^-$   $h^+$   $g^+$   
 y.  $c^+$   $h^-$   $f^+$   
 z.  $b^-$   $f^-$   $d^-$   $g^-$

In Exercises 16.1.13 through 16.1.18, for the given rotation system, (a) draw the labeled polygons, (b) draw the imbedding.

16.1.13 The rotation system of Exercise 16.1.7.

16.1.14 The rotation system of Exercise 16.1.8.

16.1.15 The rotation system of Exercise 16.1.9.

16.1.16 The rotation system of Exercise 16.1.10.

16.1.17<sup>s</sup> The rotation system of Exercise 16.1.11.

16.1.18 The rotation system of Exercise 16.1.12.

*In Exercises 16.1.19 through 16.1.28, determine the number of inequivalent imbeddings for the given graph.*

16.1.19  $K_4$ .

16.1.20  $K_5 - K_2$ .

16.1.21  $K_6 - 3K_2$ .

16.1.22  $W_n$ .

16.1.23  $Q_n$ .

16.1.24  $K_n$ .

16.1.25<sup>s</sup>  $Q_3 + K_1$ .

16.1.26  $K_{5,5}$ .

16.1.27  $K_2 \times K_4$ .

16.1.28  $P_4 + P_4$ .

## 16.2 GENUS DISTRIBUTION OF A GRAPH

The difficult problem of minimizing the number of handles needed to eliminate all the edge-crossings from a drawing of a complete graph arose in the 19th century. Gerhard Ringel worked from 1950 to 1968 to solve this problem, which he completed in 1968, with the aid of J.W.T. Youngs. The present perspective incorporates a much wider range of problems, involving algebraic enumeration and algorithmics.

### Minimum Genus

DEFINITION: The **minimum genus** of a connected graph  $G$  is the number

$$\gamma_{\min}(G) = \min \{g \mid \gamma_g(G) > 0\}$$

that is, the minimum genus of any orientable surface on which  $G$  can be imbedded.

**Example 16.2.1:** If a graph is planar, then its minimum genus is zero.

**Example 16.2.2:** If a nonplanar graph  $G$  has an edge  $e$  such that  $G - e$  is planar, then  $\gamma_{\min}(G) = 1$ . Starting with a planar drawing of  $G - e$ , using surgery to add a handle that joins the regions incident on the endpoints of  $e$ , and then drawing edge  $e$  on that handle yields an imbedding  $G \rightarrow S_1$ .

TERMINOLOGY NOTE: Before 1970, the phrase *genus of a graph* was used universally for what we now call its *minimum genus*. When there is no danger of ambiguity, we sometimes use the old phrase.

DEFINITION: A **minimum (orientable) surface** for a graph  $G$  is a surface whose genus is  $\gamma_{\min}(G)$ .

DEFINITION: Any imbedding of a graph  $G$  on a minimum (orientable) surface is called a **minimum (orientable) imbedding**.

REVIEW FROM §1.5: The **girth of a graph**  $G$  is the length of a smallest cycle in  $G$ .

**Theorem 16.2.1.** Let  $G$  be a connected graph. Then

$$\gamma_{\min}(G) \geq \left\lceil \frac{|E|(girth(G) - 2)}{2 \cdot girth(G)} - \frac{|V|}{2} + 1 \right\rceil$$

**Proof:** Given a minimum imbedding  $G \rightarrow S$ , the Euler polyhedral equation is

$$|V| - |E| + |F| = 2 - 2\gamma_{\min}(G)$$

The Edge-Face Inequality (ses §7.5 and §8.5) implies that

$$|F| \leq \frac{2|E|}{girth(G)}$$

which implies that

$$|V| - \frac{|E|(girth(G) - 2)}{girth(G)} \geq 2 - 2\gamma_{\min}(G)$$

and, in turn, that

$$\gamma_{\min}(G) \geq \frac{|E|(girth(G) - 2)}{2 \cdot girth(G)} - \frac{|V|}{2} + 1$$

Since  $\gamma_{\min}(G)$  is integer-valued, the conclusion follows.  $\diamond$

**Corollary 16.2.2.** Let  $G$  be a simple connected graph. Then

$$\gamma_{\min}(G) \geq \left\lceil \frac{|E|}{6} - \frac{|V|}{2} + 1 \right\rceil$$

**Proof:** Since  $G$  is simple, it follows that  $girth(G) \geq 3$ . Substitution of this inequality into the inequality of Theorem 16.2.1 yields the conclusion.  $\diamond$

**Example 16.2.3:** Since  $girth(K_8) = 3$ , it follows from Corollary 16.2.2 that

$$\begin{aligned} \gamma_{\min}(K_8) &\geq \left\lceil \frac{|E|}{6} - \frac{|V|}{2} + 1 \right\rceil \\ &= \left\lceil \frac{28}{6} - \frac{8}{2} + 1 \right\rceil = \left\lceil \frac{28 - 24 + 6}{6} \right\rceil = \left\lceil \frac{5}{3} \right\rceil = 2 \end{aligned}$$

### Maximum Genus

The study of the surface of largest genus on which a graph can be cellularly imbedded began with [NoStWh71], followed by [NoRiStWh72].

**DEFINITION:** The **maximum genus** of a connected graph  $G$  is the number

$$\gamma_{\max}(G) = \max \{g \mid \gamma_g(G) > 0\}$$

that is, the maximum genus of any orientable surface on which  $G$  can be cellularly imbedded.

**DEFINITION:** A **maximum (orientable) surface** for a graph  $G$  is a surface whose genus is  $\gamma_{\max}(G)$ .

**DEFINITION:** Any imbedding of a graph on a maximum (orientable) surface is called a **maximum (orientable) imbedding**.

REVIEW FROM §4.5: **Cycle rank**  $\beta(G) = |E| - |V| + 1$ .

**Theorem 16.2.3.** Let  $G$  be a connected graph. Then

$$\gamma_{\max}(G) \leq \left\lfloor \frac{\beta(G)}{2} \right\rfloor$$

**Proof:** Given a maximum imbedding  $G \rightarrow S$ , the Euler polyhedral equation is

$$|V| - |E| + |F| = 2 - 2\gamma_{\max}(G)$$

Rearranging the Euler equation yields

$$2\gamma_{\max}(G) + |F| = |E| - |V| + 2 = \beta(G) + 1$$

which implies (since  $|F| \geq 1$ ) that

$$\gamma_{\max}(G) \leq \frac{\beta(G)}{2}$$

Since  $\gamma_{\max}$  is integer-valued, the conclusion follows.  $\diamond$

**Example 16.2.4:** The maximum genus of a tree  $T$  is 0, since  $\beta(T) = 0$ .

**Example 16.2.5:** Example 16.1.2 contains an imbedding of  $K_4$  in  $S_1$ . Therefore,  $\gamma_{\max}(K_4) \geq 1$ . Moreover, by Theorem 16.2.3,

$$\gamma_{\max}(K_4) \leq \left\lfloor \frac{\beta(K_4)}{2} \right\rfloor = \left\lfloor \frac{3}{2} \right\rfloor = 1$$

Combining these two bounds yields the value  $\gamma_{\max}(K_4) = 1$ .

### Genus Range

Our immediate objective is to prove that a graph can be cellularly imbedded on any surface whose genus lies between its minimum genus and its maximum genus.

**DEFINITION:** The **genus range** of a graph  $G$  is the set

$$\{g \mid G \text{ has a rotation system } \rho \text{ such that } g = \gamma(S(\rho))\}$$

**DEFINITION:** Two rotation systems are **adjacent** if one can be obtained from the other by the operation of transposing two consecutive edge-ends in the rotation at one vertex.

**Theorem 16.2.4.** Let  $\rho$  and  $\lambda$  be adjacent rotation systems for a graph  $G$ . Then the genus of the induced surface  $S(\rho)$  differs from the genus of the induced surface  $S(\lambda)$  by at most 1.

**Proof:** Suppose that transposing edge-ends  $d^\sigma$  and  $e^\tau$  at vertex  $v$  transforms rotation system  $\rho$  into rotation system  $\lambda$ . Then deleting edge  $e$  from the respective induced

imbeddings  $\iota_\rho$  and  $\iota_\lambda$  (which eliminates  $e^\tau$  and  $e^{-\tau}$  from their rotation systems) transforms both  $\rho$  and  $\lambda$  into the same rotation system  $\eta$ . From an alternative viewpoint, each of the imbeddings

$$\iota_\rho : G \rightarrow S(\rho) \quad \text{and} \quad \iota_\lambda : G \rightarrow S(\lambda)$$

can be obtained by inserting edge  $e$  into the same imbedding  $\iota_\eta : G - e \rightarrow S(\eta)$ . Inserting one edge into an imbedding cannot decrease the genus of the surface, and it increases the genus by at most 1, since it can be drawn, if necessary, on a single new handle. This yields the inequalities

$$\gamma(S(\eta)) \leq \gamma(S(\rho)) \leq \gamma(S(\eta)) + 1 \quad \text{and} \quad \gamma(S(\eta)) \leq \gamma(S(\lambda)) \leq \gamma(S(\eta)) + 1$$

Thus,  $|\gamma(S(\rho)) - \gamma(S(\lambda))| \leq 1$ . ◊

**Corollary 16.2.5 (Interpolation Theorem [Du66]).** *Let  $G$  be a connected graph, and let  $\gamma_{\min}(G) \leq g \leq \gamma_{\max}(G)$ . Then  $\gamma_g(G) \geq 1$ .*

**Proof:** Any rotation system can be obtained from any other by a sequence of transitions of edge-ends. Thus, there is a sequence of adjacent rotation systems from some minimum-genus imbedding to a maximum-genus imbedding. By Theorem 16.2.4, the genus of the imbedding surface varies by at most 1 as one progresses through this sequence. Thus, some intermediate imbedding surface has genus  $g$ . ◊

**Remark:** Thus, Corollary 16.2.5 asserts that the genus range of a graph  $G$  is the integer interval  $[\gamma_{\min}(G), \gamma_{\max}(G)]$ .

**Example 16.2.6:** The complete graph  $K_8$  has  $(7!)^8$  different imbeddings, by Theorem 16.1.6. By Theorem 16.2.1 and Theorem 16.2.3, they all lie in the genus range  $[2, 10]$ .

**COMPUTATIONAL NOTE:** Since there are super-exponentially many rotation systems to consider, trying to determine the minimum genus or the maximum genus of a given graph, by an exhaustive algorithm, is computationally infeasible. The upper bound of Theorem 16.2.3 is not necessarily the maximum genus. There is a polynomial-time algorithm [FuGrMc88] to calculate the maximum genus. The lower bound of Theorem 16.2.1 is not necessarily the minimum genus. Deciding whether a given integer is the minimum genus of a graph is an NP-complete problem [Th89].

### Non-Orientable Imbeddings

The study of non-orientable imbeddings has the significant complication that a graph may be imbedded so that traversing some particular edges, in effect, reverses the rotation order at the destination vertex. For this reason, the coverage here is limited to two basic definitions and two basic theorems.

**DEFINITION:** The **minimum crosscap number** of a connected graph  $G$  is the smallest crosscap number of any surface  $N_k$  on which  $G$  can be imbedded. It is denoted  $\overline{\gamma}_{\min}(G)$ . If  $G$  is planar, then  $\overline{\gamma}_{\min}(G) = 0$ .

**DEFINITION:** The **maximum crosscap number** of a connected graph  $G$  is the largest crosscap number of any surface  $N_k$  on which  $G$  can be cellularly imbedded. It is denoted  $\overline{\gamma}_{\max}(G)$ . If  $G$  is planar, then  $\overline{\gamma}_{\min}(G) = 0$ .

**Theorem 16.2.6.** Let  $G$  be a connected simple graph. Then

$$\overline{\gamma}_{\min}(G) \geq \left\lceil \frac{|E|}{3} - |V| + 2 \right\rceil$$

**Proof:** This proof is completely analogous to the proof of Corollary 16.2.2. Details are left as an exercise.  $\diamond$

**Theorem 16.2.7.** Let  $G$  be a connected graph. Then

$$\overline{\gamma}_{\max}(G) = \beta(G)$$

where  $\beta(G) = |E| - |V| + 1$ .

**Proof:** That  $\beta(G)$  is an upper bound for  $\overline{\gamma}_{\max}(G)$  is provable by reasoning parallel to that for Theorem 16.2.3. Proof of equality was first published by [Ed65c].  $\diamond$

## EXERCISES for Section 16.2

In Exercises 16.2.1 through 16.2.12, use Corollary 16.2.2 to calculate a lower bound for the minimum genus of the given graph.

- |                      |   |         |                   |
|----------------------|---|---------|-------------------|
| 16.2.1               | $K_9$ .   | 16.2.2  | $K_9 - C_9$ .     |
| 16.2.3 <sup>s</sup>  | $K_{10} - C_3$ .  | 16.2.4  | $K_{10} - P_5$ .  |
| 16.2.5               | $K_8$ .   | 16.2.6  | $K_8 - K_{2,3}$ . |
| 16.2.7               | $C_3 \times C_3 \times C_3$ .                               | 16.2.8  | $C_7 + C_8$ .     |
| 16.2.9               | $K_{6,6}$ .   | 16.2.10 | $Q_6$ .           |
| 16.2.11              | The result of amalgamating two copies of $K_5$ at a vertex. |         |                   |
| 16.2.12 <sup>s</sup> | The result of amalgamating two copies of $K_5$ on an edge.  |         |                   |

16.2.13 Apply Theorem 16.2.1 to derive a lower bound for minimum genus that is sharper than the bound of Corollary 16.2.2, using the fact that  $\text{girth}(K_{6,6}) = 4$ .

16.2.14 Apply the result of Exercise 16.2.13 to the graphs of Exercises 16.2.11 and 16.2.12.

In Exercises 16.2.15 through 16.2.24, use Theorem 16.2.3 to calculate an upper bound for the maximum genus of the given graph.

- |         |                               |                      |                   |
|---------|-------------------------------|----------------------|-------------------|
| 16.2.15 | $K_9$ .                       | 16.2.16 <sup>s</sup> | $K_9 - C_9$ .     |
| 16.2.17 | $K_{10} - C_3$ .              | 16.2.18              | $K_{10} - P_5$ .  |
| 16.2.19 | $K_8$ .                       | 16.2.20              | $K_8 - K_{2,3}$ . |
| 16.2.21 | $C_3 \times C_3 \times C_3$ . | 16.2.22              | $C_7 + C_8$ .     |
| 16.2.23 | $W_6$ .                       | 16.2.24              | $CL_3$ .          |
| 16.2.25 | Prove Theorem 16.2.6.         |                      |                   |

## 16.3 VOLTAGE-GRAF SPECIFICATION OF GRAPH LAYOUTS

The application that inspired the invention of voltage graphs originally was the specification of graph imbeddings. This section explains how a cellular drawing of a voltage graph on a surface generates an imbedding for the covering graph. In this section, it is assumed, for the sake of simplicity, that the voltages are regular and that the group is additive.

### Signed Walks

The concept of signed walks illustrates emphatically that in voltage graphs the edge-directions are *not* to be regarded as traversal restrictions.

**DEFINITION:** A **signed walk** in a digraph is an alternating sequence of vertices and edge-ends

$$v_0, e_1^{\sigma_1}, v_1, e_2^{\sigma_2}, v_2, \dots, v_{n-1}, e_n^{\sigma_n}, v_n$$

such that the result of deleting signs and ignoring directions is a walk

$$v_0, e_1, v_1, e_2, v_2, \dots, v_{n-1}, e_n, v_n$$

(i.e., in the underlying graph), and such that reversing the direction on every minus-signed edge and then deleting the signs would yield a directed walk. The fundamental idea is not difficult: in a signed walk, a minus-signed edge-end represents an edge traversed against its designated direction.

**DEFINITION:** Let

$$W = \langle v_0, e_1^{\sigma_1}, v_1, e_2^{\sigma_2}, v_2, \dots, v_{n-1}, e_n^{\sigma_n}, v_n \rangle$$

be a signed walk in a voltage graph  $\langle G, \alpha : E_G \rightarrow \mathcal{B} \rangle$ . The **net voltage** on the walk  $W$  is the sum

$$\alpha(W) = \sum_{j=1}^n \sigma_j \alpha(e_j)$$

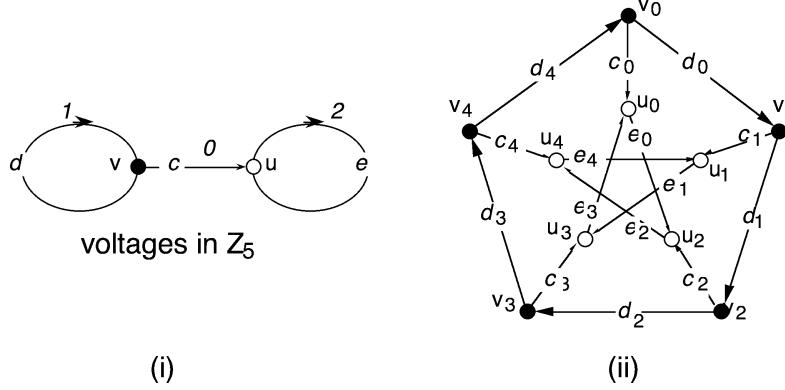
That is, the net voltage is the sum of the voltages on the plus-signed edges minus the sum of the voltages on the minus-signed edges.

**Example 16.3.1:** Figure 16.3.1 shows a specification of the Petersen graph. The signed walk

$$W = \langle v, d^+, v, c^+, u, e^-, u \rangle$$

starts at vertex  $v$ , traverses self-loop  $d$  in forward direction back to vertex  $v$ , next traverses edge  $c$  to vertex  $u$ , and then traverses self-loop  $e$  in reverse direction back to vertex  $u$ . Thus, its net voltage is

$$\alpha(W) = \alpha(d) + \alpha(c) - \alpha(e) = 1 + 0 - 2 = -1 \pmod{5}$$



**Figure 16.3.1** A voltage graph specifying the Petersen graph.

**Theorem 16.3.1 (Unique Walk Lifting).** Let  $\langle G, \alpha : E_G \rightarrow \mathcal{A} \rangle$  be a voltage graph in which there is a signed walk

$$W = v_0, e_1^{\sigma_1}, v_1, e_2^{\sigma_2}, v_2, \dots, v_{n-1}, e_n^{\sigma_n}, v_n$$

and let  $a \in \mathcal{A}$ . Then there is a unique alternating sequence

$$W_a = v_{0,a}, e_{1,a}^{\sigma_1}, v_{1,k_1}, e_{2,k_1}^{\sigma_2}, v_{2,a}, \dots, v_{n-1,k_{n-1}}, d_{n,k_{n-1}}^{\sigma_n}, v_{n,k_n}$$

that forms a signed walk in the covering digraph  $G^\alpha$ .

**Proof:** For  $j = 1, \dots, n$ , choose

$$k_j = a + \sum_{\ell=1}^{j-1} \alpha(e_\ell)$$

This formula supports an inductive argument that each successive edge choice  $e_{j,k_j}$  is the only choice of an edge in the fiber of  $e_j$  that extends the signed walk.  $\diamond$

**DEFINITION:** Let  $W$  be a signed walk in a voltage graph with initial vertex  $v$ . The unique signed walk  $W_a$  in the covering graph that starts at vertex  $v_a$  and projects onto walk  $W$  (as in Theorem 16.3.1) is called a *lift* of walk  $W$ .

**Example 16.3.2:** The walk  $W = \langle v, d^+, v, c^+, u, e^-, u \rangle$  in Example 16.3.1 has these five lifts:

$$\begin{aligned} W_0 &= \langle v_0, d_0^+, v_1, c_1^+, u_1, e_4^-, u_4 \rangle \\ W_1 &= \langle v_1, d_1^+, v_2, c_2^+, u_2, e_0^-, u_0 \rangle \\ W_2 &= \langle v_2, d_2^+, v_3, c_3^+, u_3, e_1^-, u_1 \rangle \\ W_3 &= \langle v_3, d_3^+, v_4, c_4^+, u_4, e_2^-, u_2 \rangle \\ W_4 &= \langle v_4, d_4^+, v_0, c_0^+, u_0, e_3^-, u_3 \rangle \end{aligned}$$

**Proposition 16.3.2.** Let  $\langle G, \alpha : E_G \rightarrow \mathcal{A} \rangle$  be a voltage graph, and let  $W$  be a signed walk in  $G$  from  $u$  to  $v$ . Then the lift  $W_b$  is a signed walk in the covering graph  $G^\alpha$  from the vertex  $u_b$  to the vertex  $v_{b+\alpha(W)}$ .

**Proof:** This follows by induction on the length of the walk.  $\diamond$

**Example 16.3.2, continued:** The net voltage on walk  $W$  from vertex  $v$  to vertex  $u$  is  $4 \bmod 5$ . Observe that each walk  $W_j$  begins at  $v_j$  and ends at  $u_{j+4}$ .

### Kirchoff Voltage Law

DEFINITION: A closed walk of an imbedded voltage graph satisfies the **Kirchoff voltage law (abbr. KVL)** if its net voltage is the identity element of the voltage group.

**Theorem 16.3.3.** Let  $W$  be a signed closed walk in a voltage graph that satisfies KVL. Then every lift  $W_b$  is a closed walk in the covering graph  $G^\alpha$ .

**Proof:** This is an immediate consequence of Proposition 16.3.2.  $\diamond$

**Theorem 16.3.4.** Let  $\langle G, \alpha : E_G \rightarrow \mathcal{A} \rangle$  be a voltage graph, and let  $W$  be a signed closed walk in  $G$ , such that  $\alpha(W) = b$  is of order  $m$  in the voltage group  $\mathcal{A}$ . Then the concatenation

$$W_a W_{a+b} \cdots W_{a+(m-1)b}$$

is a closed walk in  $G^\alpha$ .

**Proof:** Suppose that walk  $W$  starts at vertex  $v$ . By Proposition 16.3.2, the lift  $W_{a+\ell b}$  is a walk from  $v_{a+\ell b}$  to  $v_{a+(\ell+1)b}$ , for  $\ell = 0, \dots, m-1$ . It follows that the iterated concatenation  $W_a W_{a+b} \cdots W_{a+(m-1)b}$  is a walk from  $v_a$  to  $v_{a+mb} = v_a$ , that is, a closed walk.  $\diamond$

**Remark:** A cyclic permutation of the constituents of the concatenation

$$W_a W_{a+b} \cdots W_{a+(m-1)b}$$

may start and stop at some vertex  $v_{a+\ell b}$  different from  $v_a$ , but it traverses exactly the same cyclic sequence of vertices and edges. In the context of imbedded voltage graphs, it is called a *cyclically equivalent concatenation*.

### KVL Imbedded Voltage Graphs

DEFINITION: An **imbedded voltage graph** is a pair  $\langle \iota : G \rightarrow S, \alpha : E_G \rightarrow \mathcal{A} \rangle$ . The first component is a cellular imbedding of a graph  $G$  in a surface  $S$ , called the **base imbedding**. The second component is a voltage assignment on graph  $G$ .

DEFINITION: A face of an imbedded voltage graph is said to satisfy the **Kirchoff voltage law (abbr. KVL)** if the net voltage on every face boundary walk is 0.

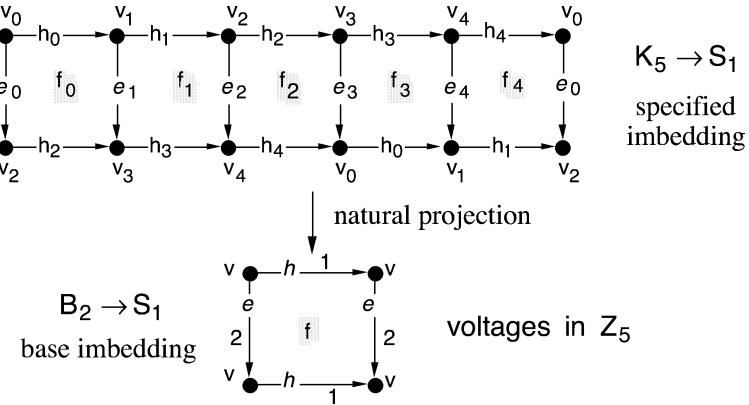
DEFINITION: A **KVL imbedded voltage graph** is an imbedded voltage graph in which every face satisfies KVL.

**Remark:** In the preceding definition, notice the difference between the classical Kirchoff voltage law of physics and its topological namesake. Whereas the physical law is that a net voltage gain of 0 occurs in the traversal of every closed walk, the topological law imposes this requirement only on face-boundary walks. (See Exercises.)

DEFINITION: In a KVL imbedded voltage graph with voltage group  $\mathcal{A}$ , let  $f$  be a  $k$ -sided face with boundary walk  $W$ . Then the **fiber over  $f$** , denoted  $\tilde{f}$ , is a set  $\{f_a \mid a \in \mathcal{A}\}$  of  $k$ -sided polygons, called the **covering faces**. The boundary walk of the covering face  $f_a$  is labeled by the lifted walk  $W_a$  of the covering graph.

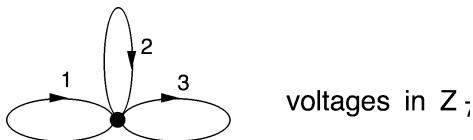
**DEFINITION:** Let  $\langle \iota : G \rightarrow S, \alpha : E_G \rightarrow A \rangle$  be a KVL imbedded voltage graph. Fitting each covering face  $f_a$ , for  $f \in F_\iota$  and  $a \in A$ , in accordance with its labeling to the corresponding lift of a boundary walk of  $f$  forms the **covering surface**  $S^\alpha$  and the **covering imbedding**  $\iota^\alpha : G^\alpha \rightarrow S^\alpha$ .

**Example 16.3.3:** Figure 16.3.2 illustrates a one-face KVL imbedding  $B_2 \rightarrow S_1$  with voltages in  $\mathbb{Z}_5$  and the covering imbedding  $K_5 \rightarrow S_1$  that it specifies.



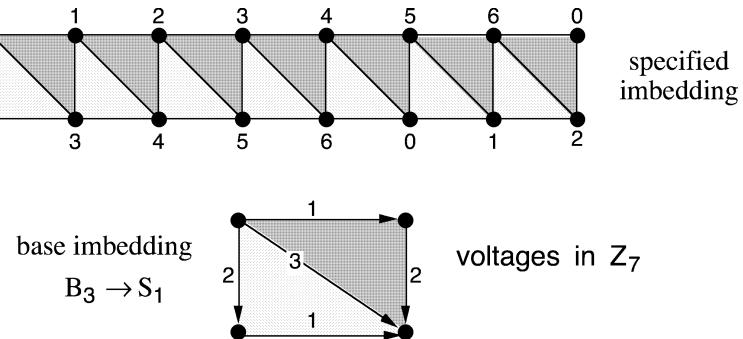
**Figure 16.3.2** A KVL voltage graph and its covering imbedding.

**Example 16.3.4:** Often, it helps to construct the covering graph alone before constructing the covering imbedding. For instance, Figure 16.3.3 shows how to assign voltages from  $\mathbb{Z}_7$  to the bouquet  $B_3$  so that the covering graph is the complete graph  $K_7$ , with 7 vertices and 21 edges.



**Figure 16.3.3** Specifying  $K_7$  by  $\mathbb{Z}_7$ -voltages on  $B_3$ .

Figure 16.3.4 illustrates a two-face KVL imbedding of that voltage graph on  $S_1$  and the covering imbedding. Since the covering imbedding has 14 faces, the covering surface must be the surface  $S_g$  whose Euler characteristic is  $7 - 21 + 14 = 0 = 2 - 2g$ , that is, the surface  $S_1$ . Shading is used to group the faces of the covering imbedding into fibers.



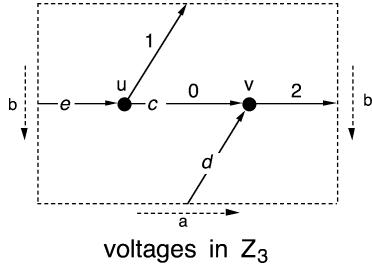
**Figure 16.3.4** Specification of an imbedding  $K_7 \rightarrow S_1$ .

**TERMINOLOGY:** The imbedded-voltage-graph construction extends the *natural projection*  $p : G^\alpha \rightarrow G$ , so that it maps each face  $f_a$  in the fiber  $\tilde{f}$  to the face  $f$ , and thereby becomes a mapping  $p : S^\alpha \rightarrow S$  of surfaces.

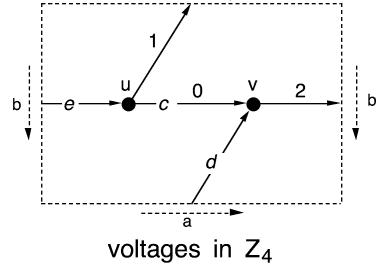
### EXERCISES for Section 16.3

In Exercises 16.3.1 and 16.3.2, draw the covering imbedding generated by the given KVL voltage graph in  $S_1$ .

16.3.1<sup>s</sup>



16.3.2



16.3.3<sup>s</sup> Prove that a KVL imbedded voltage graph in  $S_1$  specifies an imbedding such that every component of the imbedding surface is a torus. (Hint: Prove that the value of the Euler polyhedral formula  $|V| - |E| + |F|$  for the covering imbedding is 0.)

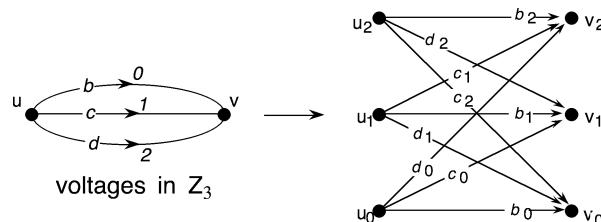
16.3.4 Prove that a KVL imbedded voltage graph in  $S_0$  with nontrivial voltage group specifies a non-connected graph and a non-connected surface. (Hint: Prove that the value of the formula  $|V| - |E| + |F|$  for the covering imbedding is larger than 2.)

16.3.5 Suppose the voltages in Figure 16.3.2 are interpreted as elements of  $\mathbb{Z}_6$ . Draw the covering imbedding and identify the covering surface. Describe the covering graph.

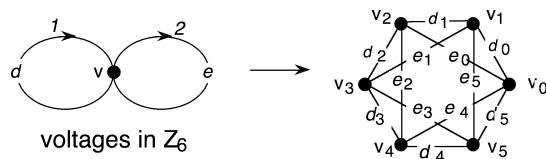
16.3.6 Suppose the voltages in Figure 16.3.4 are interpreted as elements of  $\mathbb{Z}_8$ . Draw the covering imbedding and identify the covering surface. Describe the covering graph.

In Exercises 16.3.7 through 16.3.11, construct the designated lift of a signed walk.

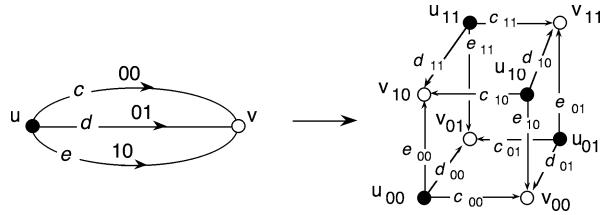
16.3.7 Signed walk  $c^+, d^-, c^+, b^-$  in this  $\mathbb{Z}_3$ -voltage graph, starting at vertex  $u_2$ .



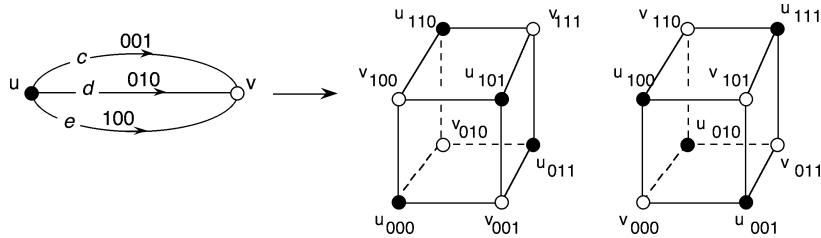
16.3.8 Signed walk  $e^-, e^-, d^+$  in this  $\mathbb{Z}_6$ -voltage graph, starting at vertex  $v_4$ .



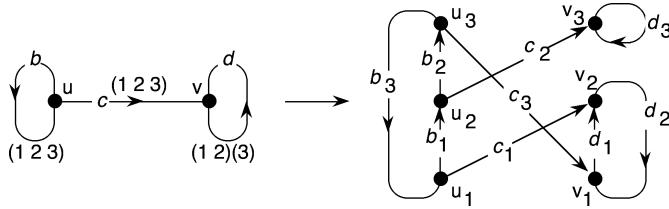
16.3.9<sup>s</sup> Signed walk  $c^+, d^-, e^+$  in this  $\mathbb{Z}_2 \times \mathbb{Z}_2$ -voltage graph, starting at vertex  $u_{01}$ .



16.3.10 Signed walk  $c^+, d^-, e^+$  in this  $\mathbb{Z}_2 \times \mathbb{Z}_2 \times \mathbb{Z}_2$ -voltage graph, starting at vertex  $u_{101}$ .



16.3.11 Signed walk  $b^+, c^+, d^-$  in this permutation- $\Sigma_3$ -voltage graph, starting at vertex  $u_3$ .



**DEFINITION:** Let  $\langle G, \alpha : E_G \rightarrow \mathcal{B} \rangle$  be a regular-voltage graph, and let  $v \in V_G$ . The **local group** at  $v$  is the subgroup of  $\mathcal{B}$  comprising the net voltages on all closed walks based at  $v$ . (This concept concerns components of the covering graph.)

16.3.12 Let  $\langle G, \alpha : E_G \rightarrow \mathcal{B} \rangle$  be a voltage graph, and let  $u, v \in V_G$ . Prove that vertex  $v_j$  is in the same component of the covering graph as vertex  $u_i$  if and only if there is a signed walk from  $u$  to  $v$  in  $G$  whose net voltage equals  $j - i$ .

16.3.13 Let  $\langle G, \alpha : E_G \rightarrow \mathcal{B} \rangle$  be a voltage graph. Use Exercise 16.3.12 to prove that the components of the covering graph are in one-to-one correspondence with the cosets of the local group in the voltage group  $\mathcal{B}$  [AlGr76].

16.3.14 Let  $G$  be a bipartite graph with voltage  $1 \in \mathbb{Z}_2$  assigned to every edge. Prove that the Kirchoff voltage law holds for every closed walk. Prove also that the covering graph consists of two disjoint copies of  $G$ . (This illustrates what happens when KVL holds on every closed walk, rather than only on face-boundary walks.)

## 16.4 NON-KVL IMBEDDED VOLTAGE GRAPHS

When an imbedded voltage graph  $\langle \iota : G \rightarrow S, \alpha : E_G \rightarrow \mathcal{A} \rangle$  does not satisfy KVL, constructing the covering faces is somewhat more complicated, because a single lift of a non-KVL face-boundary walk  $W$  in the voltage graph is not a closed walk in the covering graph. However, it is possible to concatenate several lifts of  $W$  together into a closed walk and to fit a polygon to that closed walk.

**NOTATION:** In a non-KVL imbedded voltage graph with voltage group  $\mathcal{A}$ , let  $f$  be a  $k$ -sided face with boundary walk  $W$  having net voltage  $b$  of order  $m$ . Then the set containing the walk

$$W_a W_{a+b} \cdots W_{a+(m-1)b}$$

and all cyclically equivalent walks is denoted  $W_{a+<b>}$ .

### Non-KVL Covering Faces

**DEFINITION:** In a non-KVL imbedded voltage graph with voltage group  $\mathcal{A}$ , let  $f$  be a  $k$ -sided face with boundary walk  $W$  having net voltage  $b$  of order  $m$ . Then the **fiber over**  $f$ , denoted  $\tilde{f}$ , is a set of  $mk$ -sided polygons  $f_{a+<b>}$ , called the **covering faces**, one for each equivalence class  $W_{a+<b>}$  of closed walks. The boundary walk of the covering face  $f_{a+<b>}$  is labeled by any closed walk in  $W_{a+<b>}$ .

**DEFINITION:** Let  $\langle \iota : G \rightarrow S, \alpha : E_G \rightarrow \mathcal{A} \rangle$  be a non-KVL imbedded voltage graph. Fitting each covering face  $f_{a+<b>}$ , for  $f \in F_\iota$  and  $a \in \mathcal{A}$ , in accordance with its labeling to the corresponding closed lift of a boundary walk of  $f$  forms the **covering surface**  $S^\alpha$  and the **covering imbedding**  $\iota^\alpha : G^\alpha \rightarrow S^\alpha$ .

**Example 16.4.1:** The imbedded  $(Z_2 \times Z_2)$ -voltage graph in Figure 16.4.1 has three faces, all 2-sided. The net voltage on each face boundary has order 2. Thus, the 12 boundary walk lifts combine to form 6 face boundaries in the covering graph. They fit together, as shown, to form the surface of a cube. In particular, the upper digon of the imbedded voltage graph specifies the top and bottom faces of the cube; the lower digon specifies the left and right faces; and the exterior digon specifies the front and back faces.

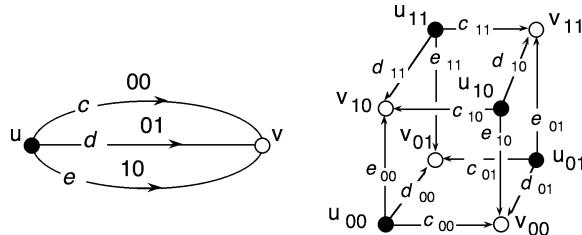


Figure 16.4.1 Specification of an imbedding  $Q_3 \rightarrow S_0$ .

### Minimum-Genus Formula for Hypercube Graphs

A standard approach to deriving a minimum-genus formula for a family of graphs is to use algebraic methods to establish a lower-bound formula and a voltage-graph construction of imbeddings that realize that lower bound. The family of hypercubes

$Q_n$  illustrates the doubly fortunate circumstance in which no surface surgery is needed, and one simple pattern of imbedded-voltage-graph drawing is enough to specify all the imbeddings.

**Proposition 16.4.1.**  $\gamma_{\min}(Q_n) \geq (n-4) \cdot 2^{n-3} + 1$ , for  $n \geq 2$ .

**Proof:** For  $n = 2$  or  $n = 3$ , the right side is 0. For  $n \geq 4$ , Theorem 16.2.1 gives the generic lower bound

$$\gamma_{\min}(G) \geq \left\lceil \frac{|E|(girth(G) - 2)}{2girth(G)} - \frac{|V|}{2} + 1 \right\rceil$$

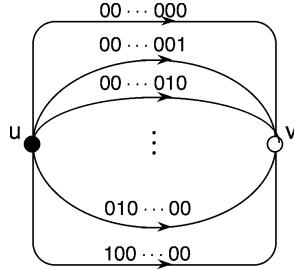
Since  $|V(Q_n)| = 2^n$ ,  $|E(Q_n)| = n \cdot 2^{n-1}$ , and  $girth(Q_n) = 4$ , this particularizes to

$$\begin{aligned} \gamma_{\min}(Q_n) &\geq \left\lceil \frac{n \cdot 2^{n-1}(4-2)}{2 \cdot 4} - \frac{2^n}{2} + 1 \right\rceil \\ &= \frac{n \cdot 2^{n-1}(4-2)}{2 \cdot 4} - \frac{2^n}{2} + 1 \\ &= n \cdot 2^{n-3} - 2^{n-1} + 1 \\ &= (n-4) \cdot 2^{n-3} + 1 \end{aligned}$$

◊

**Proposition 16.4.2.**  $\gamma_{\min}(Q_n) \leq (n-4) \cdot 2^{n-3} + 1$ , for  $n \geq 2$ .

**Proof:** It suffices to specify an imbedding of  $Q_n$  on the surface of genus  $(n-4) \cdot 2^{n-3} + 1$ . The voltage graph of Figure 16.4.1 for  $Q_3$  generalizes to dimension  $n$ , as shown in Figure 16.4.2.



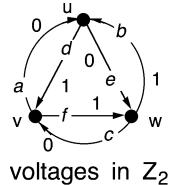
**Figure 16.4.2** Voltage graph specifying  $Q_n$ .

Since  $girth(Q_n) = 4$ , every face of an imbedding of  $Q_n$  must have at least four sides. The formula provided by Theorem 16.2.1 depends on having as many faces as possible, which implies that the number of sides of almost every face of the imbedding must equal the girth.

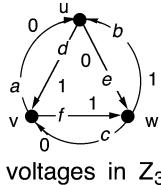
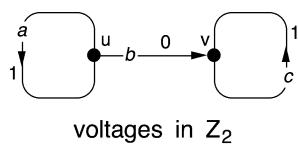
With the voltage graph of Figure 16.4.2, it is easy to construct an imbedding in which all the faces are 2-sided and have net voltage of order 2 on their boundary walks. In fact, if that drawing is interpreted as an imbedding in  $S_0$ , then each face is a digon with net voltage of order 2 on its boundary walk. ◊

**EXERCISES for Section 16.4**

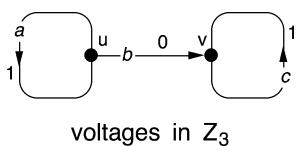
In Exercises 16.4.1 through 16.4.6, draw the derived imbedding for the given voltage graph imbedded in  $S_0$ .

16.4.1<sup>s</sup>voltages in  $Z_2$ 

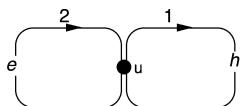
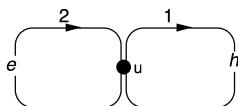
16.4.2

voltages in  $Z_3$ 16.4.3<sup>s</sup>voltages in  $Z_2$ 

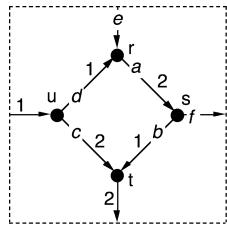
16.4.4

voltages in  $Z_3$ 

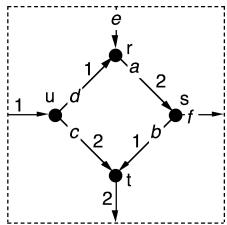
16.4.5

voltages in  $Z_5$ 16.4.6<sup>s</sup>voltages in  $Z_6$ 

In Exercises 16.4.7 and 16.4.8, for the given voltage graph imbedded in  $S_1$ , draw the derived faces and determine the genus of the imbedding surface.

16.4.7<sup>s</sup>voltages in  $Z_4$ 

16.4.8

voltages in  $Z_5$ 

## 16.5 HEWOOD MAP-COLORING PROBLEM

The question of sufficiency of 4 colors (see §9.2) needed for a proper coloring of an arbitrary sphere map was generalized by Percy Heawood. Heawood derived an upper bound for the number of colors needed for an arbitrary map on any closed surface, given as a formula in its Euler characteristic. Heawood's upper bound was ultimately proved to be the exact maximum for all surfaces except the Klein bottle  $N_2$ .

**DEFINITION:** The **chromatic number of a surface**  $S$  is the maximum of the chromatic numbers of the simple graphs that can be imbedded on  $S$ .

**REVIEW FROM §8.5:** The **Euler characteristic** of the surface  $S_g$  with  $g$  handles is the number  $2 - 2g$ .

NOTATION: As explained in §8.5, when colorings arise in topological graph theory, we need to avoid confusion between conflicting uses of the Greek letter  $\chi$ . Here we denote the chromatic number of a graph or of a surface by  $chr(G)$  or  $chr(S)$ , respectively. The Euler characteristic of a surface is denoted by  $ec(S)$ .

### Heawood Number of a Surface

The following lemmas provide basic upper bounds for the chromatic number.

**Lemma 16.5.1.** *Let  $G$  be a  $chr(S)$ -colorable chromatically critical graph imbedded on a closed surface  $S$  of Euler characteristic  $ec(S)$ . Then  $chr(S) \leq \left\lfloor 7 - \frac{6 \cdot ec(S)}{|V_G|} \right\rfloor$ .*

**Proof:** The notations  $\delta_{avg}$  and  $\delta_{min}$  indicate average degree and minimum degree.

$$\begin{aligned}
 (1) \quad & \delta_{avg}(G) \leq 6 - \frac{6 \cdot ec(S)}{|V_G|} && (\text{by Theorem 8.5.6}) \\
 (2) \quad & \delta_{min}(G) \leq \delta_{avg}(G) \\
 (3) \quad & chr(G) - 1 \leq \delta_{min}(G) && (\text{by Theorem 9.1.18}) \\
 (4) \quad & chr(G) - 1 \leq 6 - \frac{6 \cdot ec(S)}{|V_G|} && (\text{combine (1), (2), (3)}) \\
 (5) \quad & chr(S) \leq 7 - \frac{6 \cdot ec(S)}{|V_G|} && (\text{from (4)}) \\
 (6) \quad & chr(S) \leq \left\lfloor 7 - \frac{6 \cdot ec(S)}{|V_G|} \right\rfloor && (\text{from (5), since } chr(S) \text{ is an integer})
 \end{aligned}$$

◊

**Lemma 16.5.2.** *Let  $S$  be a closed surface of Euler characteristic  $ec(S) \leq 0$ . Then*

$$chr(S) \leq 7 - \frac{6 \cdot ec(S)}{chr(S)}$$

**Proof:** Let  $G$  be a  $chr(S)$ -colorable chromatically critical graph imbedded on a closed surface  $S$  of Euler characteristic  $ec(S)$ . Then

$$\begin{aligned}
 (1) \quad & chr(S) \leq 7 - \frac{6 \cdot ec(S)}{|V_G|} && (\text{by Lemma 16.5.1}) \\
 (2) \quad & chr(S) \leq |V_G| && (\text{since } chr(S) = chr(G) \leq |V_G|) \\
 (3) \quad & \frac{6 \cdot ec(S)}{|V_G|} \geq \frac{6 \cdot ec(S)}{chr(S)} && (\text{by (2), since } -6c \geq 0) \\
 (4) \quad & chr(S) \leq 7 - \frac{6 \cdot ec(S)}{chr(S)} && (\text{from (1) and (3)})
 \end{aligned}$$

◊

**DEFINITION:** The **Heawood number** of a closed surface of Euler characteristic  $ec(S)$  is the number

$$H(ec(S)) = \left\lfloor \frac{7 + \sqrt{49 - 24 \cdot ec(S)}}{2} \right\rfloor$$

**Theorem 16.5.3 (Heawood, 1890).** *Let  $S$  be a closed surface, orientable or non-orientable, with Euler characteristic  $ec(S) \leq 1$ . Then  $chr(S) \leq H(ec(S))$ .*

**Proof:** By Poincaré duality, it suffices to show that an arbitrary graph  $G$  imbeddable on a surface  $S$  has chromatic number less than or equal to the Heawood number of the surface. Moreover, it suffices to assume that  $G$  is chromatically critical.

For  $ec(S) = 1$ , Lemma 16.5.1 yields  $chr(N_1) \leq 6 = H(1)$ . For  $ec(S) \leq 0$ , Lemma 16.5.2 implies that

$$\text{chr}(S)^2 - 7\text{chr}(S) + 6 \cdot \text{ec}(S) \leq 0$$

Factoring the quadratic polynomial on the left side yields the inequality

$$\left( \text{chr}(S) - \frac{7 - \sqrt{49 - 24 \cdot \text{ec}(S)}}{2} \right) \left( \text{chr}(S) - \frac{7 + \sqrt{49 - 24 \cdot \text{ec}(S)}}{2} \right) \leq 0$$

Since  $\text{ec}(S) \leq 0$ , the value of the radical is larger than 7, from which it follows that the first factor is positive. This implies that the second factor is nonpositive. Since  $\text{chr}(S)$  is an integer, the conclusion follows.  $\diamond$

**Remark:** Of course, the sphere has Euler characteristic  $\text{ec}(S) = 2$ . Although  $H(2) = 4$ , the proof of Theorem 16.5.3 does not cover this case.

### Heawood Conjecture and Minimum Genus

Heawood omitted proof that, for each surface, there is a map that realizes the Heawood number. When the gap was noticed, Heawood's assertion was reformulated as a conjecture.

**DEFINITION:** The **Heawood conjecture** is that a surface  $S$  of Euler characteristic  $\text{ec}(S)$  has chromatic number  $H(\text{ec}(S))$ .

**Proposition 16.5.4.** If  $\gamma_{\min}(K_n) \leq \left\lceil \frac{(n-3)(n-4)}{12} \right\rceil$  for all  $n \geq 4$ , then the Heawood conjecture holds for all orientable surfaces.

**Proof:** The surface  $S_g$  has Heawood number  $H(2-2g) = \left\lceil \frac{7 + \sqrt{1+48g}}{2} \right\rceil$ . If

$$\gamma_{\min}(K_n) \leq \left\lceil \frac{(n-3)(n-4)}{12} \right\rceil \text{ for all } n \geq 4$$

and if

$$(♣) \quad \left\lceil \frac{(H(2-2g)-3)(H(2-2g)-4)}{12} \right\rceil \leq g$$

then  $\gamma_{\min}(K_{H(2-2g)}) \leq g$ . Proving inequality (♣) requires routine computation and is left as an exercise.  $\diamond$

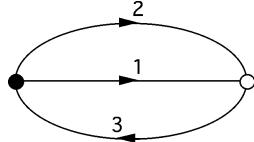
Philip Franklin proved in 1934 that  $\text{chr}(N_2) = 6$ . Since the Euler characteristic of the Klein bottle (the surface  $N_2$ ) is 0, and since  $H(0) = 7$ , Franklin's result is a counterexample to the Heawood conjecture. However, the Klein bottle is the only surface whose chromatic number is less than its Heawood number.

**Proposition 16.5.5.** If  $\tilde{\gamma}_{\min}(K_n) \leq \left\lceil \frac{(n-3)(n-4)}{6} \right\rceil$  for all  $n \geq 8$ , then the Heawood conjecture holds for all non-orientable surfaces except the Klein bottle.

**Proof:** The proof follows the same lines as for the orientable case.  $\diamond$

In the 1950s and early 1960, Gerhard Ringel proved the Heawood conjecture completely for the non-orientable surfaces (except  $N_2$ , of course), based on Proposition 16.5.5. The denominator of 12 in Proposition 16.5.4 led to the partition of the complete graphs  $K_n$  into 12 cases, one for each residue class of  $n$  modulo 12. By the early 1960s, Ringel had also constructed minimum imbeddings of  $K_n$  for four of the 12 cases.

Ringel described his minimum imbeddings by rotation systems. Each of his rotation systems was developed algebraically from one or more generating rows. Great combinatorial skill was involved in the construction of generating rows. Gustin [Gu63] introduced a graphical *nomogram* called a **current graph** as an aid in constructing generating rows of rotation systems. Figure 16.5.1 shows the Gustin nomogram used to construct the generating row for  $K_7$ .



**Figure 16.5.1** The Gustin current graph for a toroidal imbedding of  $K_7$ .

Gustin called the algebraic elements in a nomogram **currents** and he took these currents to be in the group  $\mathbb{Z}_7$ . The generating row was taken to be the sequence of currents one encountered in a traversal associated with the nomogram, in which a solid vertex ( $\bullet$ ) indicates clockwise rotation and a hollow vertex ( $\circ$ ) indicates counterclockwise rotation.

For instance, if one traverses current 1 to ( $\circ$ ), and goes counterclockwise, one next traverses current 3 to ( $\bullet$ ), where one goes clockwise. One then traverses current 2 to ( $\circ$ ), and goes counterclockwise to the edge with current 1, but against the current, so we regard this as current 6, the additive inverse of 1 mod 7. Current 6 terminates at ( $\bullet$ ), where clockwise rotation leads to current 4, which leads at ( $\circ$ ) to current 5, which cycles back to current 1, signifying termination of the tour. We summarize the tour.

$$0. \quad 1 \quad 3 \quad 2 \quad 6 \quad 4 \quad 5$$

This tour is taken as the generating row of a rotation system for  $K_7$ , in which each row is generated from the top row by adding the row number modulo 7 to the entries of the top row.

$$\begin{array}{ccccccc} 0. & 1 & 3 & 2 & 6 & 4 & 5 \\ 1. & 2 & 4 & 3 & 0 & 5 & 6 \\ 2. & 3 & 5 & 4 & 1 & 6 & 0 \\ 3. & 4 & 6 & 5 & 2 & 0 & 1 \\ 4. & 5 & 0 & 6 & 3 & 1 & 2 \\ 5. & 6 & 1 & 0 & 4 & 2 & 3 \\ 6. & 0 & 2 & 1 & 5 & 3 & 4 \end{array}$$

Gustin proved a theorem that if the current graph is 3-regular and if the sum of the currents at each vertex is 0 (a condition called the **Kirchoff current law**), then the resulting rotation system specifies a triangulation.

Ringel and Youngs augmented Gustin's method into an extensive family of nomograms, each with its own set of computational rules, and completed their proof of the Heawood conjecture for orientable surfaces in 1968, one of the outstanding mathematical accomplishments of the twentieth century. Their complete proof occupies over 300 journal pages.

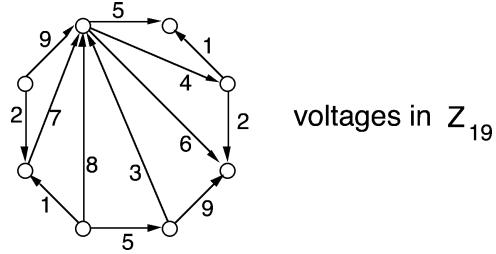
Gross and Alpert [GrAl73, GrAl74] introduced a topological interpretation of current graphs, which enabled them to generalize the construction greatly and to replace all the nomograms, with their separate rule systems, by a single form of imbedding specification. Topologists recognize the underlying mechanism as a *branched covering*, which generalizes the topological relationship of a Riemann surface to the complex plane. This topological interpretation reduced the length of the proof by about half.

Gross [Gr74] introduced voltage graphs, whose topologically dual perspective simplifies the situation by permitting the specified graph to be considered separately from the specified imbedding. Gross and Tucker [GrTu77] developed the generalization to permutation-voltage graphs (§15.3).

### Minimum Genus of Complete Graphs, Case 7

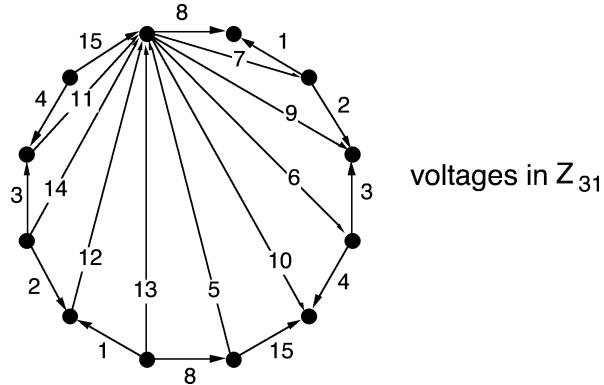
The construction of a minimum imbedding for the complete graph  $K_n$  has many details. The least complicated case is for  $n = 7 \bmod 12$ . We recall that Example 16.3.4 gives an imbedded voltage graph that specifies a minimum imbedding of  $K_7$ . This smallest instance starts a pattern that continues with minimum imbeddings for  $K_{19}$  and for  $K_{31}$ .

**Example 16.5.1:** The imbedded voltage graph in Figure 16.5.2 specifies a minimum imbedding of  $K_{19}$ .



**Figure 16.5.2** KVL voltage graph for imbedding  $K_{19} \rightarrow S_{20}$ .

**Example 16.5.2:** The imbedded voltage graph in Figure 16.5.3 specifies a minimum imbedding of  $K_{31}$ .



**Figure 16.5.3** KVL voltage graph for imbedding of  $K_{31} \rightarrow S_{63}$ .

Notice that KVL holds globally. Assigning voltages so that KVL holds globally is the difficult part.

**EXERCISES for Section 16.5**

In Exercises 16.5.1 through 16.5.6, draw an imbedded voltage graph that specifies a minimum imbedding of the given graph.

16.5.1<sup>s</sup>  $K_{43}$ .

16.5.2<sup>s</sup>  $K_{20} - 10K_2$ .

16.5.3  $K_{21} - C_{21}$ .

16.5.4  $K_{44} - 22K_2$ .

16.5.5<sup>s</sup>  $K_{4,4} - 4K_2$ .

16.5.6<sup>s</sup>  $K_{5,5} - C_{10}$ .

16.5.7 Verify the inequality (♣) in the proof of Proposition 16.5.4.

## 16.6 SUPPLEMENTARY EXERCISES

16.6.1 Calculate the face-boundary walks induced by the following rotation system. What is the genus of the induced surface?

$$v_1. \quad a+ \quad b+ \quad c+ \quad d+$$

$$v_2. \quad d- \quad e- \quad h+$$

$$v_3. \quad c- \quad g- \quad h-$$

$$v_4. \quad b- \quad f- \quad g+$$

$$v_5. \quad a- \quad f+ \quad e+$$

16.6.2 Calculate lower bounds for the minimum genus of the following graphs.

a.  $K_{13}$ .      b.  $K_{13} - K_3$ .      c.  $K_{8,8}$ .

16.6.3 Calculate upper bounds for the maximum genus of the following graphs.

a.  $K_{13}$ .      b.  $K_{13} - K_3$ .      c.  $K_{8,8}$ .

16.6.4 Prove that minimum genus of  $C_4 \times C_4 \times C_4$  is at least 17.

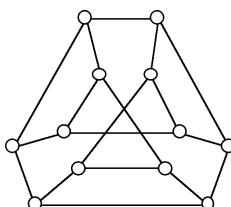
16.6.5 Prove that the minimum genus of a simple, connected graph with 8 vertices and 25 edges is at least two.

16.6.6 Prove that the minimum genus of  $K_9 - C_9$  is 1.

16.6.7 Explain how to construct 6-regular  $n$ -vertex simple graphs of girth at least 4 for all  $n \geq 12$ . (Hint: start with the even case.)

16.6.8 Calculate a lower bound for the minimum genus of the join  $C_7 + Q_3$ .

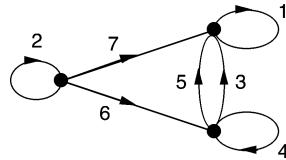
16.6.9 The graph below is **Franklin's graph**.  
 a. Count the number of 4-cycles, and use this number to prove that an imbedding of Franklin's graph has at most 7 faces.  
 b. Find a Kuratowski subgraph.



16.6.10 Prove that for every  $n \geq 7$  there is a 6-regular,  $n$ -vertex simple graph of genus 1. (Hint: use voltage graphs.)

**16.6.11** a. Draw a cyclic voltage graph for  $K_{10} - CL_5$ , with the minimum possible number of vertices. b. To cellularly imbed  $K_{10} - CL_5$  into the torus, how many faces would there be? c. To cellularly imbed the voltage graph of 6a in the torus, how many faces would there be? d. Draw an imbedded voltage graph for an imbedding of  $K_{10} - CL_5$  in the torus.

**16.6.12** The graph shown below is imbedded in the sphere, with voltages in the cyclic group  $Z_8$ . a. How many vertices and edges does the covering graph have? b. How many faces are generated by the digon, the triangle, and the exterior face? How many sides do they have? c. What is the genus of the covering surface?



**16.6.13** a. For the imbedded voltage graph on the torus of Figure 16.6.1, calculate the genus of the derived surface when the voltages are in  $Z_7$ . b. Calculate the genus of the derived surface when the voltages are in  $Z_8$ .

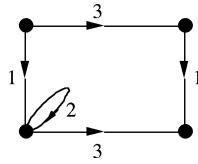


Figure 16.6.1

**16.6.14** Draw an imbedding of the following graph on the torus.

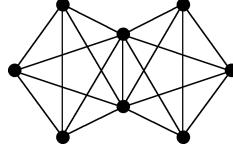


Figure 16.6.2

## GLOSSARY

**base imbedding:** the imbedding of the voltage graph in the base surface of the covering construction.

**chromatic number of a surface  $S$ :** the maximum of the chromatic numbers of the simple graphs that can be imbedded on  $S$ .

**circulation for a rotation system  $\rho^*$  (induced):** the permutation of edge-ends given by the rule  $e^\sigma \mapsto \rho(e^{-\sigma})$ .

**covering face:** a face of the graph imbedding specified by an imbedded voltage graph; a polygon whose boundary walk is matched to a lifted boundary walk or to concatenation of lifted boundary walks of the specified graph.

**covering imbedding:** the imbedding of the covering graph specified by an imbedded voltage graph.

**covering surface:** the surface formed by matching the boundaries of the covering faces to the corresponding closed walks in the covering graph.

**crosscap on a surface:** a subspace that is topologically equivalent to a Möbius band.

**current graph:** the dual of an imbedded voltage graph.

—, KCL: a current graph in which every vertex satisfies the Kirchoff current law.

**cycle rank of a graph  $G$ :** the number  $\beta(G) = |E| - |V| + 1$ .

**dual rotation:** synonym for circulation.

**equivalent imbeddings of a graph:** imbeddings that have exactly the same set of face-boundary walks.

**Euler characteristic  $\chi(S)$  of a surface:** the value of the Euler polyhedral formula  $|V| - |E| + |F|$  for any cellular imbedding on that surface:  $2 - 2g$  for the orientable surface  $S_g$  and  $2 - k$  for the non-orientable surface  $N_k$ .

**fiber over a face  $f$  of a voltage graph:** the set  $\tilde{f}$  of covering faces  $f_j$  (with  $f$  as the mainscript).

**flat polygon representation of a surface  $S$ :** a polygon with its sides marked as pairs, such that when every pair is pasted together, the resulting surface obtained is topologically equivalent to  $S$ .

**genus distribution of a graph  $G$ :** the function that assigns to each nonnegative integer  $g$  the number  $\gamma_g(G)$  of different cellular imbeddings in the surface  $S_g$

**genus range of a graph  $G$ :** the integer interval of values of  $g$  such that graph  $G$  has a cellular imbedding in surface  $S_g$ .

**girth of a graph  $G$ :** the length of a smallest cycle in  $G$ .

**Heawood conjecture:** the conjecture that a surface of Euler characteristic  $c$  has chromatic number  $H(c)$ , for  $c = 1, 0, -1, \dots$

**Heawood number of a closed surface  $S$ :** if  $ec(S)$  is the Euler characteristic, the number  $H(ec(S)) = \left\lceil \frac{7 + \sqrt{49 - 24 \cdot ec(S)}}{2} \right\rceil$ .

**imbedding induced by rotation system  $\rho$ :** the imbedding  $\iota_\rho$  whose face-boundary walks coincide with the cycles of the circulation  $\rho^*$ .

**induced rotation at a vertex  $v$  of a graph imbedding  $h$ :** the cyclic permutation  $\rho_h(v)$  of edge-ends incident at  $v$  in the order in which they are encountered in an orientation-consistent traversal around  $v$ .

**induced rotation system of an oriented graph imbedding  $h$ :** the function that assigns to each vertex its induced rotation.

**KCL:** stands for *Kirchoff current law*.

**Kirchoff current law (KCL) at a vertex:** in a current graph, the property that the net current into that vertex is the identity element of the voltage group.

**Kirchoff voltage law (KVL) on a face:** in an imbedded voltage graph, the property that the net voltage on the boundary walk of that face is the identity element of the voltage group.

**Kirchoff voltage law (KVL) on a closed walk:** in a voltage graph, the property that the net voltage on the walk is the identity element of the voltage group.

**KVL:** stands for *Kirchoff voltage law*.

**lift of a walk  $W$  in a voltage graph:** a walk  $W_a$  in the specified graph whose vertices are mapped in sequence onto walk  $W$  by the natural projection mapping.

**local group at a vertex  $v$  in a voltage graph:** the subgroup comprising the net voltages on all closed walks based at  $v$ .

**maximum crosscap number  $\overline{\gamma}_{\max}(G)$ :** for a connected graph  $G$ , the largest crosscap number of any surface  $N_k$  on which  $G$  can be cellularly imbedded. If  $G$  is planar, then  $\overline{\gamma}_{\min}(G) = 0$ .

**maximum genus of a connected graph  $G$ :** the maximum number  $g$  such that  $G$  has a cellular imbedding on  $S_g$ .

**maximum (orientable) imbedding:** any imbedding of a graph  $G$  on a *maximum (orientable) surface*.

**maximum (orientable) surface:** for a graph  $G$ , a surface whose genus is  $\gamma_{\max}(G)$ .

**minimum crosscap number  $\overline{\gamma}_{\min}(G)$ :** for a connected graph  $G$ , the smallest crosscap number of any surface  $N_k$  on which  $G$  can be imbedded. If  $G$  is planar, then  $\overline{\gamma}_{\min}(G) = 0$ .

**minimum genus  $\gamma_{\min}(G)$  of a connected graph  $G$ :** the smallest number  $g$  such that  $\gamma_g(G) > 0$ .

**minimum imbedding:** any imbedding of a graph into a *minimum surface*.

**minimum (orientable) surface:** for a graph  $G$ , a surface whose genus is  $\gamma_{\min}(G)$ .

**net voltage on a signed walk:** the sum of the voltages on the plus-signed edges minus the sum of the voltages on the minus-signed edges.

**orientation on a graph imbedding:** a choice of either *counterclockwise* or *clockwise* as a preferred traversal direction around the vertices.

**oriented graph imbedding:** a graph imbedding with a designated orientation.

**rotation at a vertex  $v$ :** a cyclic permutation of the edge-ends incident on  $v$ .

**rotation system for a graph:** an assignment of a rotation to every vertex.

—, **adjacent:** two rotation systems are adjacent if one can be obtained from the other by the operation of transposing two consecutive edge-ends in the rotation at one vertex.

**rotation table:** representation of a rotation system  $\rho$  by a list with one row for each vertex. In each row, the name of a vertex is followed by a complete list of the edge-ends incident on that vertex, in an order consistent with  $\rho(v)$ . (It is a common custom to write each row in lexicographic order.)

**signed walk in a digraph:** an undirected walk in which a minus-signed edge-end represents an edge traversed against its designated direction.

**surface induced by rotation system  $\rho$ :** the surface  $S(\rho)$  that is codomain of the circulation.

**voltage graph:** a pair  $\langle G, \alpha \rangle$  such that  $G$  is a digraph and  $\alpha$  is a *voltage assignment* on  $G$ .

—, **imbedded:** a cellular imbedding of a graph  $G$  with a voltage assignment on  $G$ .

—, **KVL:** an imbedded voltage graph in which every face satisfies KVL.

---

# APPENDIX

- A.1 Logic Fundamentals**
  - A.2 Relations and Functions**
  - A.3 Some Basic Combinatorics**
  - A.4 Algebraic Structures**
  - A.5 Algorithmic Complexity**
  - A.6 Supplementary Reading**
- 

## A.1 LOGIC FUNDAMENTALS

### Propositional Logic

DEFINITION: Let  $p$  and  $q$  be propositions. The **conditional proposition**  $p \rightarrow q$  is a proposition that is false when  $p$  is true and  $q$  is false and is true otherwise. It is read “if  $p$  then  $q$ ”.

TERMINOLOGY: The following are equivalent statements.

- $p \rightarrow q$
- if  $p$  then  $q$
- $p$  is a **sufficient** condition for  $q$
- $q$  is a **necessary** condition for  $p$
- $p$  only if  $q$
- $q$  if  $p$

DEFINITION: The **converse** of  $p \rightarrow q$  is the conditional proposition  $q \rightarrow p$ .

DEFINITION: The **contrapositive** of  $p \rightarrow q$  is the conditional proposition  $\neg q \rightarrow \neg p$ .

**Proposition A.1.1.** *The proposition  $p \rightarrow q$  and its contrapositive  $\neg q \rightarrow \neg p$  are logically equivalent.*

DEFINITION: The statement  $p$  **if and only if**  $q$  means that both conditionals  $p \rightarrow q$  and  $q \rightarrow p$  are true.

### Types of Proof

**TERMINOLOGY:** When an assertion is in the form  $p \rightarrow q$ , the proposition  $p$  is often referred to as the **antecedent** and  $q$  the **consequent**.

**DEFINITION:** A **direct proof** of the conditional  $p \rightarrow q$  consists of showing that statement  $q$  is true under the assumption that statement  $p$  is true.

**DEFINITION:** A **proof by contrapositive** of the conditional  $p \rightarrow q$  is a direct proof of the contrapositive  $\neg q \rightarrow \neg p$ .

**Example A.1.1:** A proof by contrapositive of the assertion “If  $n^2$  is odd, then  $n$  is odd.” takes the form: Assume  $n$  is even. Then  $n = 2k$ , for some integer  $k$ . Hence,  $n^2 = (2k)^2 = 4k^2$ , which is even.

**DEFINITION:** A **proof by contradiction** of an assertion  $S$  consists of assuming that  $S$  is false and deriving some contradiction.

**Example A.1.2:** Prove: There is no largest prime number.

Proof by contradiction: Assume that  $P$  is the largest prime number. Then the product  $r = 2 \times 3 \times \dots \times P + 1$  is not divisible by any prime and, hence, is itself prime, which contradicts the maximality of  $P$ .

### Mathematical Induction

**DEFINITION: Axiom of Mathematical Induction — First Form:**

Statement  $P(n)$  is true for all integers  $n \geq b$  if both of the following are true.

**Base:**  $P(b)$

**Inductive step:** For all  $k \geq b$ ,  $P(k) \rightarrow P(k+1)$

**Remark:** Typically (but not always), the inductive step is demonstrated by a direct proof of the statement  $P(k) \rightarrow P(k+1)$ .

**Example A.1.3:** Prove: For all  $n \geq 1$ ,  $\sum_{i=1}^n i = \frac{n}{2}(n+1)$ .

**Proof:** The equation holds for  $n = 1$  since its left and right sides are both equal to 1 when  $n = 1$ . Assume that the equation holds for some  $k \geq 1$ . Then

$$\sum_{i=1}^{k+1} i = \left( \sum_{i=1}^k i \right) + (k+1) = \left( \frac{k}{2}(k+1) \right) + (k+1) = \frac{k+1}{2}(k+2)$$

**DEFINITION: Axiom of Mathematical Induction — Second Form:**

Statement  $P(n)$  is true for all integers  $n \geq b$  if both of the following are true.

**Base:**  $P(b)$

**Inductive step:** For all  $k \geq b$ ,  $(P(b) \wedge P(b+1) \wedge \dots \wedge P(k)) \rightarrow P(k+1)$

Thus, a direct proof of this inductive step is to show that  $P(k+1)$  is true under the assumption that the statements  $P(b), P(b+1), \dots, P(k)$  are true.

## A.2 RELATIONS AND FUNCTIONS

### Relations

DEFINITION: The **cartesian product** of two sets  $A$  and  $B$  is the set

$$A \times B = \{(a, b) | a \in A \text{ and } b \in B\}$$

DEFINITION: A **binary relation**  $R$  from a set  $S$  to a set  $B$  is a subset of  $A \times B$ . Alternatively, a binary relation may be regarded as a function from  $A \times B$  to the Boolean set  $\{\text{true}, \text{false}\}$ .

TERMINOLOGY: Let  $R$  be a relation from set  $A$  to set  $B$ . If  $(x, y) \in R$ , then  $x$  is said to be **related to**  $y$  (by  $R$ ). This is often denoted  $xRy$ .

DEFINITION: A **(binary) relation on** a set  $A$  is a relation from  $A$  to  $A$ .

DEFINITION: A relation  $R$  on a set  $A$  is **reflexive** if for all  $x \in A$ ,

$$xRx \in R$$

DEFINITION: A relation  $R$  on a set  $A$  is **symmetric** if for all  $x, y \in A$ ,

$$xRy \Rightarrow yRx$$

DEFINITION: A relation  $R$  on a set  $A$  is **transitive** if for all  $x, y, z \in A$ ,

$$(xRy \wedge yRz) \Rightarrow xRz$$

DEFINITION: A relation  $R$  on a set  $A$  is an **equivalence relation** if  $R$  is reflexive, symmetric, and transitive.

**Example A.2.1:** Let  $R$  be the relation on the set  $\mathbb{Z}$  of all integers, given by

$$xRy \iff x - y = 3k, \text{ for some } k \in \mathbb{Z}$$

Then  $R$  is an equivalence relation.

DEFINITION: Let  $R$  be an equivalence relation on a set  $A$ , and let  $x \in A$ . The **equivalence class of  $a$** , denoted  $[a]$ , is given by

$$[x] = \{a \in A | aRx\}$$

**Example A.2.2:** For the equivalence relation defined in Example A.2.1,

$$\begin{aligned} [0] &= \{3k | k \in \mathbb{Z}\} \\ [1] &= \{3k + 1 | k \in \mathbb{Z}\} \\ [2] &= \{3k + 2 | k \in \mathbb{Z}\} \end{aligned}$$

Observe that the equivalence class of any other element is one of these three classes.

DEFINITION: A collection of distinct non-empty subsets  $\{S_1, S_2, \dots, S_l\}$  of a set  $A$  is a **partition** of  $A$  if both of the following conditions are satisfied.

- $S_i \cap S_j = \emptyset$ , for all  $1 \leq i < j \leq l$
- $\bigcup_{i=1}^l S_i = A$

**Proposition A.2.1.** Let  $R$  be an equivalence relation on a set  $A$ , and let  $x, y \in A$ . Then the following three statements are logically equivalent.

1.  $xRy$
2.  $[x] = [y]$
3.  $[x] \cap [y] \neq \emptyset$

**Corollary A.2.2.** Let  $R$  be an equivalence relation on a set  $A$ . Then the distinct equivalence classes of  $R$  partition  $A$ .

**Example A.2.3:** The three equivalence classes  $[0]$ ,  $[1]$ , and  $[2]$  of Example A.2.2 partition the set  $\mathbb{Z}$  of integers.

### Partial Orderings and Posets

DEFINITION: A relation  $R$  on a set  $A$  is **antisymmetric** if for all  $x, y \in A$ ,

$$xRy \wedge yRx \Rightarrow x = y$$

DEFINITION: A relation  $R$  on a set  $A$  is a **partial ordering** (or **partial order**) if  $R$  is reflexive, antisymmetric, and transitive.

**Example A.2.4:** Let  $R$  be the *divisibility relation* on the set  $\mathbb{Z}^+$  of positive integers, given by  $xRy \iff x$  divides  $y$ . Then  $R$  is a partial ordering.

NOTATION: The symbol  $\prec$  is often used to denote a partial ordering.

DEFINITION: Let  $\prec$  be a partial ordering on a set  $A$ . The pair  $\{A, \prec\}$  is called a **partially ordered set** (or **poset**).

### Functions

DEFINITION: Let  $A$  and  $B$  be two sets. A **function**  $f$  from  $A$  to  $B$ , denoted  $f : A \rightarrow B$ , is an assignment of exactly one element of  $B$  to each element of  $A$ . We also say that  $f$  **maps**  $A$  to  $B$ .

ALTERNATIVE DEFINITION: A **function**  $f$  from  $A$  to  $B$ , denoted  $f : A \rightarrow B$ , is a relation from  $A$  to  $B$ , such that each element of set  $A$  appears as a first component in exactly one of the ordered pairs of  $f$ .

TERMINOLOGY: Let  $f : A \rightarrow B$  be a function from set  $A$  to set  $B$ . Then  $A$  is called the **domain of  $f$** , and  $B$  is called the **codomain of  $f$** .

TERMINOLOGY: Let  $f : A \rightarrow B$  be a function from set  $A$  to set  $B$ , and let  $b = f(a)$ . The element  $b$  is called the **image of  $a$  under  $f$** , and element  $a$  is called a **preimage of  $b$  under  $f$** .

DEFINITION: Let  $f : A \rightarrow B$  be a function from set  $A$  to set  $B$ , and let  $S \subseteq A$ . Then the set  $f(S) = \{f(s) \mid s \in S\}$  is the **image of  $S$  under  $f$** .

DEFINITION: Let  $f : A \rightarrow B$  be a function from set  $A$  to set  $B$ , and let  $S \subseteq B$ . Then the set  $f^{-1}(S) = \{x \in A \mid f(x) \in S\}$  is the **inverse image of  $S$  under  $f$** .

**DEFINITION:** A function  $f : A \rightarrow B$  is **one-to-one** (or **injective**) if for all  $x, y \in A$ ,  $x \neq y \Rightarrow f(x) \neq f(y)$ .

Thus, a function  $f : A \rightarrow B$  is one-to-one if and only if each element of set  $B$  has at most one preimage under  $f$ .

**DEFINITION:** A function  $f : A \rightarrow B$  is **onto** (or **surjective**) if for all  $b \in B$ , there exists at least one  $a \in A$  such that  $f(a) = b$ .

Thus, a function  $f : A \rightarrow B$  is onto if and only if each element of set  $B$  has at least one preimage under  $f$ .

**DEFINITION:** A function  $f : A \rightarrow B$  is a **bijection** if  $f$  is both injective and surjective.

**Example A.2.5:** In a standard (0-based) vertex-labeling of an  $n$ -vertex graph  $G$ , each vertex of  $G$  is assigned an integer label from the set  $\{0, 1, \dots, n - 1\}$ , such that no two vertices get the same label. Thus, the labeling is a bijection

$$f : \{0, 1, \dots, n - 1\} \rightarrow V_G$$

It is common to say that the integers from 0 to  $n - 1$  **have been bijectively assigned** to the vertices of  $G$ .

**Theorem A.2.3 [Pigeonhole Principle].** Let  $f : A \rightarrow B$  be a function from a finite set  $A$  to a finite set  $B$ . Let any two of the following three statements be true.

1.  $f$  is one-to-one.
2.  $f$  is onto.
3.  $|A| = |B|$ .

Then the third statement is also true.

The following corollary uses familiar pigeonhole jargon.

**Corollary A.2.4.** If each member of a flock of pigeons flies into a pigeonhole, and if there are more pigeons than pigeonholes, then there must be at least one pigeonhole that contains more than one pigeon.

**DEFINITION:** Let  $f : A \rightarrow B$  be a function from a set  $A$  to a set  $B$ , and let  $g : B' \rightarrow C$  be a function from a set  $B'$  to a set  $C$ , where  $B \subseteq B'$ . Then the **composition of  $f$  and  $g$**  is the function  $g \circ f : A \rightarrow C$  given by  $g \circ f(a) = g(f(a))$ .

**DEFINITION:** Let  $f : A \rightarrow B$  be a function from a set  $A$  onto a set  $B$ , and let  $g : B \rightarrow A$  be a function from a set  $B$  to a set  $A$ . Then  $g$  is the **inverse function** of  $f$  if for all  $a \in A$ ,  $g \circ f(a) = a$ .

**Proposition A.2.5.** Let  $f : A \rightarrow B$  be a function from a set  $A$  onto a set  $B$ , and let  $g : B \rightarrow A$  be a function from a set  $B$  onto a set  $A$ . Then  $g$  is the inverse function  $f$  if and only if  $f$  is the inverse function of  $g$ .

**Proposition A.2.6.** A function  $f : A \rightarrow B$  has an inverse if and only if  $f$  is a bijection.

---

## A.3 SOME BASIC COMBINATORICS

**Proposition A.3.1 [Rule of Product].** Let  $A$  and  $B$  be two finite sets. Then

$$|A \times B| = |A| \cdot |B|$$

DEFINITION: A **permutation** of an  $n$ -element set  $S$  is a bijection  $\pi : S \rightarrow S$ .

DEFINITION: A  **$k$ -permutation** of an  $n$ -element set  $S$  is an *ordered subset* of  $S$ .

Thus, a  $k$ -permutation of  $S$  is a permutation of a subset of  $S$ .

An immediate consequence of the Rule of Product is the following simple formula for the number of  $k$ -permutations.

**Proposition A.3.2.** The number of  $k$ -permutations of an  $n$ -element set is

$$\frac{n!}{(n-k)!}$$

DEFINITION: A  **$k$ -combination** of a set  $S$  is a  $k$ -element subset of  $S$ .

NOTATION: The number of  $k$ -combinations of an  $n$ -element set is denoted  $\binom{n}{k}$  or, alternatively,  $C(n, k)$ .

**Proposition A.3.3.**  $\binom{n}{k} = \frac{n!}{k!(n-k)!}$

**Theorem A.3.4 [Binomial Theorem].** For every nonnegative integer  $n$  and any numbers  $x$  and  $y$ ,

$$(x+y)^n = \sum_{k=0}^n \binom{n}{k} x^{n-k} y^k$$

DEFINITION: Let  $\langle a_1, a_2, \dots, a_k, \dots \rangle$  be a sequence of real numbers. The **generating function** for the sequence  $\{a_n\}$  is the power series

$$g(x) = \sum_{k=0}^{\infty} a_k x^k = a_0 + a_1 x + a_2 x^2 + \dots + a_k x^k + \dots$$

**Example A.3.1:** The generating function for the constant sequence  $\langle 1, 1, \dots \rangle$  is

$$\sum_{k=0}^{\infty} x^k = \frac{1}{1-x}$$

**Example A.3.2:** The generating function for the sequence  $\langle 1, 2, 3, \dots \rangle$  is

$$\sum_{k=0}^{\infty} k x^{k-1} = \sum_{k=0}^{\infty} \frac{d}{dx} x^k = \frac{d}{dx} \left[ \sum_{k=0}^{\infty} x^k \right] = \frac{d}{dx} \left( \frac{1}{1-x} \right) = \frac{1}{(1-x)^2}$$

## A.4 ALGEBRAIC STRUCTURES

### Groups

**DEFINITION:** A **binary operation**  $*$  on a nonempty set  $A$  is a function  $f : A \times A \rightarrow A$ , given by  $f((a, b)) = a * b$ .

**NOTATION:** The set  $A$  together with a binary operation  $*$  is denoted  $(A, *)$ .

**Example A.4.1:** Let  $\mathbb{Z}_{10} = \{0, 1, 2, \dots, 9\}$ , and let  $\oplus$  and  $\odot$  be the operations of addition and multiplication mod 10. For instance,  $6 \oplus 6 = 12 = 2 \pmod{10}$  and  $6 \odot 6 = 36 = 6 \pmod{10}$ . Then  $\oplus$  and  $\odot$  are binary operations on the set  $\mathbb{Z}_{10}$ .

**DEFINITION:** The binary operation  $*$  on set  $A$  is **associative** if for all  $a, b, c \in A$ ,  $(a * b) * c = a * (b * c)$ .

**DEFINITION:** The binary operation  $*$  on set  $A$  is **commutative** if for all  $a, b \in A$ ,  $a * b = b * a$ .

**DEFINITION:** Let  $*$  be a binary operation on set  $A$ . An element  $e \in A$  is an **identity element** of set  $A$  under  $*$  if for all  $a \in A$ ,

$$a * e = e * a = a$$

**DEFINITION:** Let  $*$  be a binary operation on set  $A$ , and let  $e$  be an identity element of  $(A, *)$ . For  $a \in A$ , an element  $a' \in A$  is an **inverse** of  $a$  in  $(A, *)$  if

$$a * a' = a' * a = e$$

**DEFINITION:** A **group**  $\mathcal{G} = (G, *)$  is a nonempty set  $G$  and a binary operation  $*$  that satisfy the following conditions.

1. The operation  $*$  is associative.
2.  $\mathcal{G}$  has an identity element.
3. Each  $g \in G$  has an inverse in  $(G, *)$ .

**NOTATION:** Each element  $g$  in a group  $\mathcal{G}$  has a *unique* inverse, which is denoted  $g^{-1}$ .

**DEFINITION:** If  $\mathcal{G} = (G, *)$  is a group, the set  $G$  is the **domain** (set) of the group  $\mathcal{G}$ .

**NOTATION:** When convenient, and when there is no ambiguity, we sometimes write “ $g \in \mathcal{G}$ ” to mean that  $g$  is an element of the domain set  $G$ .

**DEFINITION:** An **abelian** group is a group whose operation is commutative.

**NOTATION:** In an abelian group, the operation is commonly denoted “ $+$ ” and called “sum”.

**TERMINOLOGY:** For an abstract group  $(G, *)$ , the term “product” is used generically to refer to the element  $x * y$ , and its use does not imply anything about the type of group operation.

**Example A.4.2:** The group  $\mathbb{Z}_n = (\mathbb{Z}_n, +)$ , which generalizes  $(\mathbb{Z}_{10}, \oplus)$  in Example A.4.1, is abelian and is denoted  $\mathbb{Z}_n$ .

**Example A.4.3:** Observe that  $(\mathbb{Z}_n, \odot)$  is *not* a group, since the element 0 has no inverse under  $\odot$ .

**Example A.4.4:** Let  $S$  be an  $n$ -element set, and let  $P$  be the set of all permutations of set  $S$ . Then  $(P, \circ)$ , where  $\circ$  is the *composition* operation, is a group.

**DEFINITION:** Let  $\mathcal{G} = (G, *)$  be a group. A subset  $X$  of  $G$  is a **generating set** for group  $\mathcal{G}$  if every element of  $\mathcal{G}$  can be expressed as a product of elements of set  $X$ .

**Example A.4.5:** The set  $\{2, 5\}$  is a generating set for  $\mathbb{Z}_{10}$ .

**DEFINITION:** A **cyclic group** is a group that has a 1-element generating set.

**NOTATION:** If  $\{a\}$  is a generating set for a cyclic group  $\mathcal{G}$ , then we may write  $\mathcal{G} = \langle a \rangle$ .

**Example A.4.6:**  $\mathbb{Z}_n = (\mathbb{Z}_n, +) = \langle 1 \rangle$ .

**Example A.4.7:**  $(\mathbb{Z}_7 - \{0\}, \odot) = \langle 3 \rangle = \langle 5 \rangle$ .

**DEFINITION:** Let  $\mathcal{G}_1 = (G_1, *_1)$  and  $\mathcal{G}_2 = (G_2, *_2)$  be two groups. The **direct sum** (or **direct product**)  $\mathcal{G}_1 \times \mathcal{G}_2$  is the group  $(G_1 \times G_2, *)$  whose domain set is the cartesian product of the domain sets  $G_1$  and  $G_2$  and whose operation is defined by

$$(x_1, x_2) * (y_1, y_2) = (x_1 *_1 y_1, x_2 *_2 y_2)$$

**DEFINITION:** The  $n$ -fold direct sum of groups  $\mathcal{G}_1 = (G_1, *_1)$ ,  $\mathcal{G}_2 = (G_2, *_2)$ , ...,  $\mathcal{G}_n = (G_n, *_n)$  is defined recursively as

$$\mathcal{G}_1 \times \mathcal{G}_2 \times \cdots \times \mathcal{G}_n = (\mathcal{G}_1 \times \mathcal{G}_2 \times \cdots \times \mathcal{G}_{n-1}) \times \mathcal{G}_n$$

**Theorem A.4.1 [Fundamental Theorem of Finite Abelian Groups].** Let  $\mathcal{G}$  be a finite abelian group. Then  $\mathcal{G}$  can be expressed as the direct sum of cyclic groups.

**Remark:** In fact, the cyclic groups in the direct-sum decomposition of an abelian group have prime-power order, and the direct sum is *essentially* unique, but the above assertion is all that we need in this book.

### Order of an Element in a Group

**NOTATION:** Let  $x$  be an element of a group  $\mathcal{G} = (G, *)$ . The element  $x * x$  may be denoted  $x^2$ , and the product of  $r$  factors of element  $x$  may be denoted  $x^r$ . In an abelian group, the element  $x + x$  may be denoted  $2x$ , and the sum of  $r$  occurrences of element  $x$  may be denoted  $r \cdot x$  or sometimes  $rx$ .

**DEFINITION:** Let  $x$  be an element of a group with identity  $e$ . The **order** of  $x$  is the smallest positive integer  $m$  such that  $x^m = e$ .

**DEFINITION:** The **order** of a group  $\mathcal{G} = (G, *)$  is the cardinality  $|G|$  of its domain set.

**Proposition A.4.2.** In a finite group, every element has finite order.

**Example A.4.8:** For  $2 \in \mathbb{Z}_{10}$ , we have  $5 \cdot 2 = 2 + 2 + 2 + 2 + 2 = 0 \pmod{10}$ . Thus, the order of the element  $2 \in \mathbb{Z}_{10}$  is 5.

## Permutation Groups

**DEFINITION:** The group of all permutations of a set  $S$  with  $n$  elements is called the **full symmetric group** on  $S$  and is denoted  $\Sigma_S$  or  $\Sigma_n$ .

**TERMINOLOGY:** The elements of the set on which a permutation group acts are called **objects**.

**NOTATION:** One way of representing a permutation  $\pi$  of the set  $\{1, 2, \dots, n\}$  is to use a  $2 \times n$  matrix, where the first row contains the  $n$  integers in order, and the second row contains the images, so that  $\pi(j)$  appears below  $j$ .

**Example A.4.9:** The group  $\Sigma_3$  consists of the six permutations of  $S = \{1, 2, 3\}$ . They are as follows:

$$\begin{aligned}\pi_1 &= \begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}; & \pi_2 &= \begin{pmatrix} 1 & 2 & 3 \\ 1 & 3 & 2 \end{pmatrix}; & \pi_3 &= \begin{pmatrix} 1 & 2 & 3 \\ 2 & 1 & 3 \end{pmatrix}; \\ \pi_4 &= \begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{pmatrix}; & \pi_5 &= \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \end{pmatrix}; & \pi_6 &= \begin{pmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \end{pmatrix}\end{aligned}$$

The permutation  $\pi_1$  is the identity element of  $\Sigma_3$ . Remembering that the composition  $f \circ g$  is applied *right-to-left*, we have the following sample calculations:

$$\pi_2 \circ \pi_2 = \pi_3 \circ \pi_3 = \pi_4 \circ \pi_4 = \pi_1; \quad \pi_5 \circ \pi_5 = \pi_6; \quad \pi_4 \circ \pi_5 = \pi_3; \quad \pi_5 \circ \pi_4 = \pi_2$$

The last two calculations show that the full symmetric group is *non-abelian*. Additional calculation shows that the set  $\{\pi_4, \pi_5\}$  is a generating set for  $\Sigma_3$ . Moreover,  $\pi_5^3 = \pi_6 \circ \pi_5 = \pi_1$ . Thus, the order of the permutation  $\pi_5$  is 3.

**DEFINITION:** A permutation  $\pi$  is an **involution** if  $\pi = \pi^{-1}$ .

## Cycle Notation for a Permutation

Each object of  $\{1, 2, 3\}$  in Example A.4.9 *cycles* back to itself after three applications of the permutation  $\pi_5 = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \end{pmatrix}$ . That is,

$$1 \xrightarrow{\pi_5} 2 \xrightarrow{\pi_5} 3 \xrightarrow{\pi_5} 1 \quad 2 \xrightarrow{\pi_5} 3 \xrightarrow{\pi_5} 1 \xrightarrow{\pi_5} 2 \quad 3 \xrightarrow{\pi_5} 1 \xrightarrow{\pi_5} 2 \xrightarrow{\pi_5} 3$$

The *cycling* of the images under successive applications of a given permutation is compactly represented by enclosing the chain of images in parentheses so that the image of each element appears to the right of that element, with *wraparound* for the rightmost element.

**NOTATION:** If  $S$  is a set and  $a_1, a_2, \dots, a_k \in S$ , then  $(a_1 \ a_2 \ \cdots a_k)$  denotes the permutation  $\pi$  of  $S$  for which

$$\pi(a_i) = \begin{cases} a_{i+1} & \text{for } i = 1, \dots, k-1 \\ a_1 & \text{for } i = k \end{cases}$$

and

$$\pi(x) = x \quad \text{for all other } x \in S$$

**Example A.4.10:** The cycle representation of permutation  $\pi_5$  from Example A.4.9 can be written in any one of three equivalent ways:  $(1 \ 2 \ 3)$ ,  $(2 \ 3 \ 1)$ , or  $(3 \ 1 \ 2)$ .

**NOTATION:** This cycle notation can be extended to any permutation by writing that permutation as a **product of disjoint cycles**. The elements that appear in each cycle form the chain of images under the action of that permutation.

**Example A.4.11:** Consider the permutation  $\alpha \in \Sigma_9$  given by  $\alpha = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 7 & 2 & 6 & 1 & 8 & 3 & 4 & 9 & 5 \end{pmatrix}$ . Then the disjoint cycle representation of  $\alpha$  is

$$\alpha = (1\ 7\ 4)(2)(3\ 6)(5\ 8\ 9) \text{ or } (1\ 7\ 4)(3\ 6)(5\ 8\ 9)$$

## Fields

**DEFINITION:** A **field**  $\mathcal{F} = (F, +, \cdot)$  is a set  $F$  together with two operations,  $+$  and  $\cdot$ , (called generically *addition* and *multiplication*), such that each of the following conditions is satisfied:

1.  $(F, +)$  is an abelian group
2.  $(F - \{0\}, \cdot)$  is an abelian group, where  $0$  is the additive identity
3.  $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$  and  $(a + b) \cdot c = (a \cdot c) + (b \cdot c)$

### Finite Field $GF(2)$

**DEFINITION:** The **finite field**  $GF(2)$  consists of the set  $\mathbb{Z}_2 = \{0, 1\}$  together with the mod 2 operations  $\oplus$  and  $\odot$ . Thus,

$$0 \oplus 0 = 1 \oplus 1 = 0; \quad 0 \oplus 1 = 1 \oplus 0 = 1; \quad 0 \odot 0 = 1 \odot 0 = 0 \odot 1 = 0; \quad 1 \odot 1 = 1$$

## Vector Spaces

**DEFINITION:** A **vector space** over a field (of **scalars**)  $\mathcal{F}$  is a set  $V$  (of **vectors**) together with an operation  $+$  on  $V$  and a mapping, called **scalar multiplication**, from the cartesian product  $\mathcal{F} \times V$  to  $V$  ( $(a, v) \rightarrow av$ ), such that the following conditions are satisfied for all scalars  $a, b \in \mathcal{F}$  and all vectors  $v, w \in V$ .

1.  $(V, +)$  is an abelian group (the symbol “+” is being used to denote two different operations, addition in the field  $\mathcal{F}$  and addition in the set  $V$ )
2.  $(a \cdot b)v = a(bv)$
3.  $(a + b)v = av + bv$
4.  $a(v + w) = av + aw$
5.  $ev = v$ , where  $e$  is the multiplicative identity of field  $\mathcal{F}$

**Example A.4.12:** Let  $W = \{(\alpha_1, \alpha_2, \alpha_3) \mid \alpha_i \in GF(2)\}$ , and consider componentwise addition  $\oplus$  (mod 2) on the elements of  $W$ . For each scalar  $\alpha \in GF(2)$  and element  $w = (\alpha_1, \alpha_2, \alpha_3)$ , let scalar multiplication be defined by

$$\alpha w = (\alpha \odot \alpha_1, \alpha \odot \alpha_2, \alpha \odot \alpha_3)$$

Then it is easy to verify that  $W$  with these operations forms a vector space over  $GF(2)$ .

**Remark:** The vector space defined in Example A.4.12 is a special case of the vector space  $\mathcal{F}^n$  of all  $n$ -tuples whose components are elements in a field  $\mathcal{F}$ .

### Independence, Subspaces, and Dimension

The remaining definitions in this section all assume that  $V$  is a vector space over a field  $\mathcal{F}$ . The examples refer to the vector space  $W$  defined in Example A.4.12.

**DEFINITION:** A vector  $v$  is a **linear combination** of vectors  $v_1, v_2, \dots, v_m$  if  $v = \alpha_1 v_1 + \alpha_2 v_2 + \dots + \alpha_m v_m$  for scalars  $\alpha_i \in \mathcal{F}, i = 1, \dots, m$ .

**Example A.4.13:** In vector space  $W$ , the vector  $(1, 0, 1)$  is a linear combination of vectors  $(1, 1, 0)$  and  $(0, 1, 1)$ , given by  $(1, 0, 1) = 1(1, 1, 0) + 1(0, 1, 1)$ . This last equation also implies that  $(1, 0, 1)$  is a linear combination of any superset of  $\{(1, 1, 0), (0, 1, 1)\}$ , where the scalar multipliers for the other vectors in the superset are 0.

**DEFINITION:** The vectors  $v_1, v_2, \dots, v_m$  are **linearly independent** if none of them is expressible as a linear combination of the remaining ones.

**Example A.4.14:** It is easy to check that  $\{(1, 1, 0), (0, 1, 0), (0, 1, 1)\}$  is a set of linearly independent vectors in vector space  $W$ .

**DEFINITION:** A set  $S$  of vectors **spans**  $V$  if every vector in  $V$  is expressible as a linear combination of vectors in  $S$ .

**Example A.4.15:** The three vectors in Example A.4.14 span  $W$ . A more obvious spanning set is  $\{(1, 0, 0), (0, 1, 0), (0, 0, 1)\}$ .

**DEFINITION:** A set  $B$  is a **basis** for  $V$  if the vectors in  $B$  spans  $V$  and are linearly independent.

**Proposition A.4.3.** All the bases of a given vector space have the same number of vectors.

**DEFINITION:** The **dimension** of a vector space  $V$ , denoted  $\dim(V)$ , is the number of vectors in a basis of  $V$ .

**DEFINITION:** A subset  $U$  of vector space  $V$  is a **subspace** of  $V$  if  $U$  is itself a vector space with respect to the addition and scalar multiplication of  $V$ .

**Proposition A.4.4.** Let  $V$  be a vector space over a field  $\mathcal{F}$ , and let  $U$  be a subset of  $V$ . Then  $U$  is a subspace of  $V$  if and only if

1.  $v + w \in U$  for all  $v, w \in U$
2.  $\alpha v \in U$  for all  $\alpha \in \mathcal{F}, v \in U$

**Example A.4.16:** Either of the 3-element sets given in Example A.4.15 shows that  $W$  is a 3-dimensional vector space. The set  $U = \{(0, 0, 0), (1, 1, 0), (0, 1, 1), (1, 0, 1)\}$  is a 2-dimensional subspace of  $W$ .

**Proposition A.4.5.** Let  $U_1$  and  $U_2$  be two subspaces of a vector space  $V$ . Then the intersection  $U_1 \cap U_2$  is a subspace of  $V$ .

**DEFINITION:** The **direct sum**  $U_1 \oplus U_2$  of two subspaces  $U_1$  and  $U_2$  of  $V$  is the set of all vectors of the form  $v_1 + v_2$ , where  $v_1 \in V_1$  and  $v_2 \in V_2$ .

**Proposition A.4.6.** Let  $U_1$  and  $U_2$  be two subspaces of a vector space  $V$ . Then the direct sum  $U_1 \oplus U_2$  is a subspace of  $V$ , and  $\dim(U_1 \oplus U_2) = \dim(U_1) + \dim(U_2) - \dim(U_1 \cap U_2)$ .

---

## A.5 ALGORITHMIC COMPLEXITY

One approach to the analysis and comparison of algorithms is to consider the *worst-case* performance, ignoring constant factors that are often outside a given programmer's control. These factors include the computer on which the algorithm is run, the operating system, and the machine code translation of the programming language in which the algorithm is written. The objective of this approach is to determine the functional dependence of the running time (or of some other measure) on the number  $n$  of inputs (or on some other such variable). A first step toward this goal is to make the notion of "proportional to" mathematically precise.

### Big- $O$ Notation

**DEFINITION:** Let  $f : Z^+ \rightarrow R$  and  $g : Z^+ \rightarrow R$  be functions from the set of positive integers to the set of real numbers. The function  $f(n)$  is in the family  $O(g(n))$  (read "big- $O$ " of  $g(n)$ ) if there are constants  $c$  and  $N$  such that

$$|f(n)| \leq c|g(n)| \quad \text{for all } n > N$$

**TERMINOLOGY:** If  $f(n)$  is in  $O(g(n))$ , then  $f(n)$  is said to be **asymptotically dominated** by  $g(n)$ . In other words, for sufficiently large  $n$  and a sufficiently large constant  $c$ ,  $cg(n)$  is larger than  $f(n)$ .

**TERMINOLOGY NOTE:** The usage " $f$  is  $O(g(n))$ " (omitting the preposition "in") is quite common.

**Remark:** It follows from the definition that  $f(n)$  is in  $O(g(n))$  if and only if there is a constant  $c$  such that  $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = c$ .

**Example A.5.1:** The function  $f(n) = \sum_{i=1}^n i$  is in  $O(n^2)$ . To see this, observe that for all  $n > 0$

$$f(n) = \frac{n^2 + n}{2} < n^2 + n \leq n^2 + n^2 = 2n^2$$

Thus, the condition is satisfied for  $c = 2$  and  $N = 0$ . It is also true that  $f(n)$  is in  $O(n^r)$  for any  $r \geq 2$ , but  $O(n^2)$  is the *sharpest* among these. In fact, letting  $c = 2$  and  $N = 0$ , the following chain of inequalities shows that  $n^2$  is in  $O(f(n))$ .

$$n^2 < n^2 + n = 2f(n)$$

The next example illustrates the effect of information-structure representation of a graph on algorithmic computation.

**Example A.5.2:** Given a simple graph  $G$ , find the degree of each vertex  $v \in V_G$ . For each vertex, we must count the number of edges incident on that vertex. The number of steps required depends on how the input graph is represented in the computer. We consider two of the possibilities. Let  $n$  be the number of vertices and  $m$  the number of edges.

*Case 1:* representation by adjacency matrix

Because the adjacency matrix is symmetric, only the upper (or lower) triangular submatrix (excluding the main diagonal) needs to be examined. There are  $\sum_{i=1}^{n-1} i$  cells in this triangular submatrix, and each cell must be checked to see if it is nonzero. Hence, the number of checks is  $O(n^2)$ .

*Case 2:* representation by adjacency lists of edges

For each vertex, the list of edges incident with that vertex is traversed. Thus, the number of add operations is  $O(m)$ .

**DEFINITION:** Functions  $f(n)$  and  $g(n)$  are **big-O equivalent** if  $f(n)$  is in  $O(g(n))$  and  $g(n)$  is in  $O(f(n))$ .

**Proposition A.5.1.** Functions  $f(n)$  and  $g(n)$  are big-O equivalent if and only if  $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = c$  for some nonzero constant  $c$ .

**Proposition A.5.2.** Two polynomial functions are big-O equivalent if and only if they have the same degree.

**DEFINITION:** A **polynomial-time** algorithm is an algorithm whose running time (as a function on the input size  $n$ ) is in  $O(p(n))$  for some polynomial  $p(n)$ .

**TERMINOLOGY:** A *linear-time* algorithm is an algorithm whose running time is in  $O(n)$ . A *quadratic-time* algorithm is an algorithm whose running time is in  $O(n^2)$ .

## Decision Problems

**DEFINITION:** A **decision problem** is a problem that requires only a *yes* or *no* answer regarding whether some element of its domain has a particular property.

**DEFINITION:** A decision problem belongs to the **class P** if there is a polynomial-time algorithm to solve the problem.

**DEFINITION:** A decision problem belongs to the **class NP** if there is a way to provide evidence of the correctness of a *yes* answer so that it can be confirmed by a polynomial-time algorithm.

**Example A.5.3:** The decision problem “Is this graph bipartite?” is in class  $P$ .

**Example A.5.4:** It is not known whether the decision problem “Can the vertices of this graph be colored with three colors such that adjacent vertices get different colors?” is in class  $P$ . However, a *yes* answer could be supported by supplying a proposed 3-coloring of the vertices. Checking the correctness of this 3-coloring would require counting the number of different colors used on the vertices, which can be accomplished in linear time, and checking for each pair of vertices, whether the two vertices have received the same color and whether they are adjacent, which can be performed in quadratic time. Hence, this decision problem is in class  $NP$ .

**Remark:** The letters  $NP$  stand for *nondeterministic polynomial*. This terminology derives from the fact that problems in class  $NP$  would be answered in polynomial time if there was a nondeterministic (limitlessly parallel) machine that could check all possible evidence sets simultaneously. This philosophically creative notion provides a meaningful theoretical construct for categorizing problems.

It might appear that the class of problems that could be solved by a nondeterministic computer should “obviously” be larger than those that can be solved by strictly sequential polynomial-time algorithms. However, no one has been able to find a problem in class  $NP$  that is provably not in class  $P$ . Determining whether such a decision problem exists is among the foremost unsolved problems in theoretical computer science.

### The Problem Classes $NP$ -Complete and $NP$ -Hard

There is a large collection of problems (called *NP-complete*), including the vertex-coloring problem and the traveling salesman problem, for which no polynomial-time algorithms have been found, despite considerable effort over the last several decades. It remains open whether there exist such algorithms for these problems. It turns out that if one can find a polynomial-time algorithm for an  $NP$ -complete problem, then polynomial-time algorithms can be found for all  $NP$ -complete problems. Our goal here is to provide a brief and informal introduction to this topic. A precise, detailed description of this class of problems may be found, for instance, in [Ev79], [GaJo79], or [Wi86].

**DEFINITION:** A decision problem  $R$  is **polynomially reducible** to  $Q$  if there is a polynomial-time transformation of each instance  $I_R$  of problem  $R$  to an instance  $I_Q$  of problem  $Q$ , such that instances  $I_R$  and  $I_Q$  have the same answer (*yes* or *no*).

**DEFINITION:** A decision problem is  **$NP$ -hard** if every problem in class  $NP$  is polynomially reducible to it.

**DEFINITION:** An  $NP$ -hard problem  $R$  is  **$NP$ -complete** if  $R$  is in class  $NP$ .

**DEFINITION:** Let  $X = \{x_1, x_2, \dots, x_n\}$  be a set of  $n$  boolean variables. The **satisfiability problem** is to decide for a propositional form  $S(x_1, \dots, x_n)$  in the variables  $x_i \in X$ , whether there is a set of truth values  $\{a_1, \dots, a_n\}$  such that  $S(a_1, \dots, a_n)$  is true.

Stephen Cook ([Co71]) showed that the class  $NP$ -complete is nonempty by showing that the satisfiability problem is  $NP$ -complete. Since that time, the list has grown rapidly.

## A.6 SUPPLEMENTARY READING

For discrete mathematics: [PoSt90], [Ro84], [Ro03], [St93], [Tu95].

For algebra: [Ga90], [MaBi67].

For algorithmics: [AhHoUl83], [Ba88], [Se88], [Wi86].

---

# BIBLIOGRAPHY

## **B.1 General Reading**

## **B.2 References**

---

## **B.1 GENERAL READING**

The first part of this bibliography is a list of books of general interest on graph theory and on background topics for graph theory, at a level consistent with our present text, grouped according to topic.

### **ALGEBRA and ENUMERATION**

- [Ga90] J. A. Gallian, *Contemporary Abstract Algebra*, Second Edition, D. C. Heath, 1990.
- [GrKnPa94] R. L. Graham, D. E. Knuth, and O. Patashnik, *Concrete Mathematics: A Foundation for Computer Science*, Second Edition, Addison-Wesley, 1994.
- [MaBi67] S. MacLane and G. Birkhoff, *Algebra*, Macmillan, 1967.

### **ALGORITHMS and COMPUTATION**

- [AhHoUl83] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *Data Structures and Algorithms*, Addison-Wesley, 1983.
- [Ba83] S. Baase, *Computer Algorithms: Introduction to Design and Analysis*, Addison-Wesley, 1983.
- [Ev79] S. Even, *Graph Algorithms*, Computer Science Press, 1979.
- [GaJo79] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, 1979.
- [Gr97] J. Gruska, *Foundations of Computing*, Thomson, 1997.

- [La76] E. L. Lawler, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston, 1976.
- [Se88] R. Sedgewick, *Algorithms*, Addison-Wesley, 1988.
- [Wi86] H. S. Wilf, *Algorithms and Complexity*, Prentice-Hall, 1986.

## COMBINATORIAL MATHEMATICS

- [Bo00] K. P. Bogart, *Introductory Combinatorics*, Third Edition, Academic Press, 2000.
- [Br04b] R. A. Brualdi, *Introductory Combinatorics*, Fourth Edition, Prentice Hall, 2004.
- [MiRo91] J. G. Michaels and K. H. Rosen (Eds.), *Applications of Discrete Mathematics*, McGraw-Hill, 1991.
- [PoSt90] A. Polimeni and H. J. Straight, *Foundations of Discrete Mathematics*, Brooks/Cole, 1990.
- [Ro84] F. S. Roberts, *Applied Combinatorics*, Prentice-Hall, 1984.
- [Ro03] K. H. Rosen, *Discrete Mathematics and Its Applications*, Fifth Edition, McGraw-Hill, 2003.
- [St93] H. J. Straight, *Combinatorics: An Invitation*, Brooks/Cole 1993.
- [Tu01] A. Tucker, *Applied Combinatorics*, Fourth Edition, John Wiley & Sons, 2001.

## GRAPH THEORY

- [Be85] C. Berge, *Graphs*, North-Holland, 1985.
- [BiLiWi86] N. L. Biggs, E. K. Lloyd, and R. J. Wilson, *Graph Theory 1736-1936*, Oxford, 1986.
- [Bo98] B. Bollobás, *Modern Graph Theory*, Springer, 1998.
- [BoMu76] J. A. Bondy and U. S. R. Murty, *Graph Theory With Applications*, American Elsevier, 1976.
- [ChLe04] G. Chartrand and L. Lesniak, *Graphs & Digraphs*, Fourth Edition, CRC Press, 2004.
- [ChZh05] G. Chartrand and P. Zhang, *Introduction to Graph Theory*, McGraw Hill, 2005.
- [Di00] R. Diestel, *Graph Theory*, Springer-Verlag, Second Edition, 2000.
- [Gi85] A. Gibbons, *Algorithmic Graph Theory*, Cambridge University Press, 1985.

- [Go80] M. C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, 1980.
- [Go88] R. Gould, *Graph Theory*, Benjamin/Cummings, 1988.
- [GrYe04] J. L. Gross and J. Yellen, eds, *Handbook of Graph Theory*, CRC Press, 2004.
- [Ha69] F. Harary, *Graph Theory*, Addison-Wesley, 1969.
- [ThSw92] K. Thulasiraman and M. N. S. Swamy, *Graphs: Theory and Algorithms*, John Wiley & Sons, 1992.
- [We01] D. B. West, *Introduction to Graph Theory*, Second Edition, Prentice-Hall, 2001.
- [Wi96] R. J. Wilson, *Introduction to Graph Theory*, Addison Wesley Longman, 1996.
- [WiWa90] R. J. Wilson and J. J. Watkins, *Graphs: An Introductory Approach*, John Wiley & Sons, 1990.

## SURFACES and TOPOLOGICAL GRAPH THEORY

- [GrTu87] J. L. Gross and T. W. Tucker, *Topological Graph Theory*, Dover, 2001. (First Edition, Wiley-Interscience, 1987.)
- [Ma67] W. S. Massey, *Algebraic Topology: An Introduction*, Harbrace, 1967.
- [Ri74] G. Ringel, *Map Color Theorem*, Springer, 1974.
- [Wh84] A. T. White, *Graphs, Groups and Surfaces*, Revised Edition, North-Holland, 1984.

## B.2 REFERENCES

In addition to the references cited in the text and listed below, the reader may wish to consult the *Handbook of Graph Theory*, which contains many contributed sections with summaries of the most important research results and extensive bibliographies.

### Chapter 1: Introduction to Graph Models

- [Ro84] F. S. Roberts, *Applied Combinatorics*, Prentice-Hall, 1984.
- [Wi04] R. J. Wilson, *History of Graph Theory*, §1.3 in *Handbook of Graph Theory*, eds., J. L. Gross and J. Yellen, CRC Press, 2004.
- [WiWa90] R. J. Wilson and J. J. Watkins, *Graphs: An Introductory Approach*, John Wiley & Sons, 1990.

### **Chapter 2: Structure and Representation**

[Mc77] B. D. McKay, Computer reconstruction of small graphs, *J. Graph Theory* 1 (1977), 281–283.

[Ni77] A. Nijenhuis, Note on the unique determination of graphs by proper subgraphs, *Notices Amer. Math. Soc.* 24 (1977), A-290.

### **Chapter 3: Trees**

[AhHoUl83] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *Data Structures and Algorithms*, Addison-Wesley, 1983.

[Hu52], D. A. Huffman, A method for the construction of minimum-redundancy codes, *Proc. IRE* 40 (1952), 1098–1101.

### **Chapter 4: Spanning Trees**

[AhHoUl83] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *Data Structures and Algorithms*, Addison-Wesley, 1983.

[Ba83] S. Baase, *Computer Algorithms: Introduction to Design and Analysis*, Addison-Wesley, 1983.

[Di59] E. W. Dijkstra, A note on two problems in connexion with graphs, *Numerische Mathematik* 1 (1959), 269–271.

[Fl62] R. W. Floyd, Algorithm 97: shortest path, *Comm. ACM* 5:6 (1962), 345.

[Kr56] J. B. Kruskal Jr., On the shortest spanning subtree of a graph and the traveling salesman problem, *Proceedings AMS* 7, 1 (1956).

[La76] E. L. Lawler, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston, 1976.

[Ox92] J. G. Oxley, *Matroid Theory*, Oxford, 1992.

[Pr57] R. C. Prim, Shortest connection networks and some generalizations, *Bell System Technical Journal* 36 (1957).

[Se88] R. Sedgewick, *Algorithms*, Addison-Wesley, 1988.

[ThSw92] K. Thulasiraman and M. N. S. Swamy, *Graphs: Theory and Algorithms*, John Wiley & Sons, 1992.

[We76] D. J. A. Welsh, *Matroid Theory*, Academic Press, 1976.

[Wi96] R. J. Wilson, *Introduction to Graph Theory*, Addison Wesley Longman, 1996.

## Chapter 5: Connectivity

- [AhHoUl83] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *Data Structures and Algorithms*, Addison-Wesley, 1983.
- [Ba83] S. Baase, *Computer Algorithms: Introduction to Design and Analysis*, Addison-Wesley, 1983.
- [ChHa68] *Theory of Graphs* [P. Erdős and G. Katona, eds.] Akadémiai Kiadó, Budapest, 1968, pp.61–63.
- [FoFu56] L. R. Ford and D. R. Fulkerson, Maximal flow through a network, *Canad. J. Math.* 8 (1956), 399–404.
- [Ha62] F. Harary, The maximum connectivity of a graph, *Proc. Natl. Acad. Sci., U.S.* 48 (1962), 1142–1146.
- [ThSw92] K. Thulasiraman and M. N. S. Swamy, *Graphs: Theory and Algorithms*, John Wiley & Sons, 1992.
- [Tu61] W. T. Tutte, A theory of 3-connected graphs, *Indag. Math.* 23 (1961), 441–455.

## Chapter 6: Optimal Graph Traversals

- [CrPa80] H. Crowder and M. W. Padberg, Solving large-scale symmetric traveling salesman problem to optimality, *Management Science* 26 (1980), 495–509.
- [DaFuJo54] G. B. Dantzig, D. R. Fulkerson, and S. M. Johnson, Solution of a large-scale traveling salesman problem, *Oper. Res.* 2 (1954), 393–410.
- [DeEh51] N. G. deBruijn and T. Ehrenfest, Circuits and trees in Oriental graphs, *Simon Stevin* 28 (1951), 203–217.
- [Di52] G. A. Dirac, Some theorems on abstract graphs, *Proc. London Math. Soc.* 2 (1952), 69–81.
- [Ed65a] J. Edmonds, The Chinese postman problem, *Oper. Res.* 13-1 (1965), 373.
- [EdJo73] J. Edmonds and E. L. Johnson, Matching Euler tours and the Chinese postman, *Math. Programming* 5 (1973), 88–124.
- [Fl90] H. Fleischner, *Eulerian Graphs and Related Topics, Part 1, Vol. 1*, Ann. Discrete Math. **45** North-Holland, Amsterdam, 1990.
- [Fl91] H. Fleischner, *Eulerian Graphs and Related Topics, Part 1, Vol. 2*, Ann. Discrete Math. **50** North-Holland, Amsterdam, 1991.
- [Fl04] H. Fleischner, *Eulerian Graphs*, §4.2 in *Handbook of Graph Theory*, editors, J. L. Gross and J. Yellen, CRC Press, Boca Raton, Fl, 2004.

- [Fr79] G. N. Frederickson, Approximation algorithms for some postman problems, *JACM* 26 (1979), 538–554.
- [Gu62] M. Guan, Graphic programming using odd and even points, *Chinese Math.* 1 (1962), 273–277.
- [Hu69] G. Hutchinson, Evaluation of polymer sequence fragment data using graph theory, *Bull. Math. Biophys.* 31 (1969), 541–562.
- [HuWi75] J. P. Hutchinson and H. S. Wilf, On Eulerian circuits and words with prescribed adjacency patterns, *J. Comb. Theory* 18 (1975), 80–87.
- [LaLeKaSh85] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. Shmoys, *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, John Wiley, 1985.
- [Or60] O. Ore, Note on Hamilton circuits, *Amer. Math. Monthly* 67 (1960), 55.
- [Pa76] C. H. Papadimitriou, On the complexity of edge traversing, *JACM* 23 (1976), 544–554.
- [PaSt82] C. H. Papadimitriou and K. Stieglitz, *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, 1982.
- [ReTa81] E. M. Reingold and R. E. Tarjan, On a greedy heuristic for complete matching, *SIAM J. Computing* 10 (1981), 676–681.
- [Ro99] H. Robinson, Graph Theory Techniques in Model-Based Testing, 1999 International Conference on Testing Computer Software.
- [RoStLe77] D. J. Rosenkrantz, R. E. Stearns, and P. M. Lewis, An analysis of several heuristics for the travelling salesman problem, *SIAM J. Comput.* 6 (1977), 563–581.
- [SaGo76] S. Sahni and T. Gonzalez, P-complete approximation problems, *JACM* 23 (1976), 555–565.
- [SmTu41] C. A. B. Smith and W. T. Tutte, On unicursal paths in a network of degree 4, *Amer. Math. Monthly* 48 (1941), 233–237.
- [TuBo83] A. C. Tucker and L. Bodin, A model for municipal street-sweeping operations, in W. F. Lucas, F. S. Roberts, and R. M. Thrall (Eds.), *Discrete and System Models* 3, of *Modules in Applied Mathematics*, Springer-Verlag (1983), 76–111.
- [Wo72] D. R. Woodall, Sufficient conditions for circuits in graphs, *Proc. London Math. Soc.* 24 (1972), 739–755.

## Chapter 7: Planarity and Kuratowski's Theorem

- [BoMu76] J. A. Bondy and U. S. R. Murty, *Graph Theory With Applications*, American Elsevier, 1976.

- [DeMaPe64] G. Demoucron, Y. Malgrange, and R. Pertuiset, Graphes planaires: reconnaissance et construction de représentations planaires topologiques, *Rev. Francaise Recherche Operationelle* 8 (1964), 33–47.
- [Fa48] I. Fary, On the straight-line representations of planar graphs, *Acta Sci. Math.* 11 (1948), 229–233.
- [GrRo79] J. L. Gross and R. H. Rosen, A linear-time planarity algorithm for 2-complexes, *J. Assoc. Comput. Mach.* 20 (1979), 611–617.
- [GrRo81] J. L. Gross and R. H. Rosen, A combinatorial characterization of planar 2-complexes, *Colloq. Math.* 44 (1981), 241–247.
- [Ha69] F. Harary, *Graph Theory*, Addison-Wesley, 1969.
- [HoTa74] J. Hopcroft and R. E. Tarjan, Efficient Planarity Testing, *JACM* 21 (1974), 549–568.
- [Ne54] M. H. A. Newman, *Elements of the Topology of Plane Sets of Points*, Cambridge University Press, 1954.
- [Th80] C. Thomassen, Planarity and duality of finite and infinite graphs, *J. Combin. Theory Ser. B* 29 (1980), 244–271.
- [Th81] C. Thomassen, Kuratowski’s theorem, *J. Graph Theory* 5 (1981), 225–241.
- [Wa36] K. Wagner, Bemerkungen zum Vierfarbenproblem, *Jber. Deutch. Math. Verien.* 46 (1936), 21–22.
- [We01] D. B. West, *Introduction to Graph Theory*, Second Edition, Prentice-Hall, 2001.
- [Wi04] R. J. Wilson, *History of Graph Theory*, §1.3 in *Handbook of Graph Theory*, eds., J. L. Gross and J. Yellen, CRC Press, 2004.

## Chapter 8: Drawing Graphs and Maps

- [AgWe88] A. Aggarwal and J. Wein, Computational Geometry, *MIT LCS Research Seminar Series* 3, 1988.
- [Ch00] J. Chen, Algorithms and Complexity in Computational Geometry, §13.5 of *Handbook of Discrete and Combinatorial Mathematics* (ed. K. H. Rosen), CRC Press, 2000.
- [CoMo72] H. S. M. Coxeter and W. O. J. Moser, *Generators and Relators for Discrete Groups*, Third Edition, Springer-Verlag, 1972.
- [EaWh96] P. Eades and S. Whitesides, The realization problem for Euclidean minimum spanning trees is NP-hard, *Algorithmica* 16 (1996), 60–82. (Special issue on Graph Drawing, ed. by G. Di Battista and R. Tamassia.)

- [GrTu87] J. L. Gross and T. W. Tucker, *Topological Graph Theory*, Dover, 2001. (First Edition, Wiley-Interscience, 1987.)
- [LiDi95] G. Liotta and G. Di Battista, Computing proximity drawings of trees in the 3-dimensional space, pp 239–250 in Proc. 4th Workshop Algorithms Data Struct., *Lecture Notes Comput. Sci.* vol. 955, Springer-Verlag, 1995.
- [LiTa04] G. Liotta and R. Tamassia, *Drawings of Graphs*, §10.3 in *Handbook of Graph Theory*, eds., J. L. Gross and J. Yellen, CRC Press, 2004.
- [Ma67] W. S. Massey, *Algebraic Topology: An Introduction*, Springer-Verlag, 1989. (First Edition: Harcourt Brace & World, 1967.)
- [MoSu92] C. Monma and S. Suri, Transitions in geometric minimum spanning trees, *Discrete Comput. Geom.* 8 (1992), 265–293.

### Chapter 9: Graph Colorings

- [ApHa76] K. I. Appel and W. Haken, Every planar map is four-colorable, *Bull. Amer. Math. Soc.* 82 (1976), 711–712.
- [Bä38] F. Bäbler, Über die Zerlegung regulärer Streckenkomplexe ungerader Ordnung, *Comment. Math. Helv.* 10 (1938), 275–287.
- [Be68] L. W. Beineke, Derived graphs and digraphs, in *Beiträge zur Graphentheorie* Tübner, 1968.
- [Br79] D. Brelaz, New methods to color the vertices of a graph, *Commun. ACM* 22 (1979), 251–256.
- [Ca86] M. W. Carter, A survey of practical applications of examination timetabling algorithms, *Oper. Res.* 34 (1986), 193–202.
- [GaJo79] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*, W. H. Freeman, 1979.
- [Ho81] I. Holyer, The NP-completeness of edge-coloring, *SIAM J. Computing* 10 (1981), 718–720.
- [KiYe92] L. Kiaer and J. Yellen, Weighted graphs and university course timetabling, *Computers and Oper. Res.* 19 (1992), 59–67.
- [Kö16] D. König, Über Graphen und ihre Anwendung auf Determinantentheorie und Mengenlehre, *Math. Ann.* 77 (1916), 453–465.
- [Lo75] L. Lovász, Three short proofs in graph theory, *J. Combin. Theory Ser. B* 19 (1975), 269–271.
- [Ma81] B. Manvel, Coloring large graphs, *Proceedings of the 1981 Southeastern Conference on Graph Theory, Combinatorics and Computer Science*, 1981.

- [Pe1891] J. Petersen, Die Theorie der regulären Graphen, *Acta Math.* 15 (1891), 193–220.
- [Pl04] M. Plummer, *Factors and Factorization*, §5.4 in *Handbook of Graph Theory*, eds., J. L. Gross and J. Yellen, CRC Press, 2004.
- [RoSaSeTh97] N. Robertson, D. P. Sanders, P. Seymour, and R. Thomas, The four-colour theorem, *J. Combin. Theory Ser. B* 70 (1997), 166–183.

## Chapter 10: Measurement and Mappings

- [AnLaLuMcMe94] C. A. Anderson, L. Langley, J. R. Lundgren, P. A. McKenna and S. K. Merz, New classes of  $p$ -competition graphs and  $\phi$ -tolerance competition graphs, *Congr. Numer.* 100 (1994), 97–107.
- [Be62] C. Berge, *Theory of Graphs and its Applications*, Methuen, 1962.
- [Be73] C. Berge, *Graphs and Hypergraphs*, North-Holland, 1973.
- [BiSy83] H. Bielak and M.M. Syslo, Peripheral vertices in graphs, *Studia Sci. Math. Hungar.* 18 (1983), 269–275.
- [Br04a] R. C. Brigham, *Bandwidth*, §9.4 in *Handbook of Graph Theory*, eds., J. L. Gross and J. Yellen, CRC Press, 2004.
- [BrCaVi00] R. C. Brigham, J. R. Carrington and R. P. Vitray, Bipartite graphs and absolute difference tolerances, *Ars Combin.* 54 (2000), 3–27.
- [BrDu85] R. C. Brigham and R. D. Dutton, A compilation of relations between graph invariants, *Networks* 15 (1985), 73–107.
- [BrMcVi95] R. C. Brigham, F. R. McMorris and R. P. Vitray, Tolerance competition graphs, *Linear Algebra Appl.* 217 (1995), 41–52.
- [BrMcVi96] R. C. Brigham, F. R. McMorris and R. P. Vitray, Two- $\phi$ -tolerance competition graphs, *Discrete Appl. Math* 66 (1996), 101–108.
- [Bu74] P. Buneman, A characterisation of rigid circuit graphs, *Discrete Math.* 9 (1974), 205–212.
- [BuHa90] F. Buckley and F. Harary, *Distance in Graphs*, Addison-Wesley, 1990.
- [Ch88] F. R. K. Chung, Labelings of graphs, *Selected Topics in Graph Theory 3*, Academic Press Limited, San Diego, CA (1988), 151–168.
- [ChChDeGi82] P. Z. Chinn, J. Chvátalová, A. K. Dewdney, and N. E. Gibbs, The bandwidth problem for graphs and matrices—a survey, *Journal of Graph Theory* 6 (1982), 223–254.

- [ChDeGiKo75] J. Chvátalová, A. K. Dewdney, N. E. Gibbs, and R. R. Korfhage, The bandwidth problem for graphs: a collection of recent results, Research Report 24, Department of Computer Science, University of Western Ontario, (1975).
- [ChOeTiZo89] G. Chartrand, O. R. Oellermann, S. Tian, and H. B. Zou, Steiner distance in graphs, *Časopis Pro Pest. Mat.* 114 (1989), 399–410.
- [ChZh04] G. Chartrand and P. Zhang, *Distance in Graphs*, §9.1 in *Handbook of Graph Theory*, eds., J. L. Gross and J. Yellen, CRC Press, 2004.
- [Co78] J. E. Cohen, *Food Webs and Niche Space*, Princeton University Press, 1978.
- [DuBr83] R. D. Dutton and R. C. Brigham, A characterization of competition graphs, *Discrete Appl. Math.* 6 (1983), 315–317.
- [ErGoPo66] P. Erdős, A. W. Goodman and L. Pósa, The representation of a graph by set intersections, *Canad. J. Math.* 18 (1966), 106–112.
- [Ga74] F. Gavril, The intersection graphs of subtrees in trees are exactly the chordal graphs, *Journal of Comb. Theory, Ser. B* 16 (1974), 47–56.
- [GoMo82] M. C. Golumbic and C. L. Monma, A generalization of interval graphs with tolerances, *Congr. Numer.* 35 (1982), 321–331.
- [GoMoTr84] M. C. Golumbic, C. L. Monma and W. T. Trotter, Tolerance graphs, *Discrete Appl. Math.* 9 (1984), 157–170.
- [GoTr03] M. C. Golumbic and A. N. Trenk, *Tolerance Graphs*, Cambridge University Press, 2003.
- [Ha64] L. H. Harper, Optimal assignment of numbers to vertices, *Journal of SIAM* 12 (1964), 131–135.
- [HaHe04] T. W. Haynes and M. A. Henning, Domination in Graphs, §9.2 in *Handbook of Graph Theory*, eds., J. L. Gross and J. Yellen, CRC Press, 2004.
- [HaHeSl98] T. W. Haynes, S. T. Hedetniemi, and P. J. Slater, *Fundamentals of Domination in Graphs*, Marcel Dekker, 1998.
- [HaNo53] F. Harary and R. Z. Norman, The dissimilarity characteristic of Husimi trees, *Ann. of Math.* 58 (1953), 134–141.
- [HeOeSw90] M. A. Henning, O. R. Oellermann, and H. C. Swart, On Steiner radius and Steiner diameter of a graph, *Ars Combin.* 29 (1990), 13–19.
- [HeOeSw91] M. A. Henning, O. R. Oellermann, and H. C. Swart, On vertices with maximum Steiner eccentricity in graphs, *Graph Theory, Combinatorics, Algorithms and Applications* (eds. Y. Alavi, F. R. K. Chung, R. L. Graham, and D. F. Hsu), SIAM Publications (1991), 393–403.

- [JaMcMu91] M. S. Jacobson, F. R. McMorris and H. M. Mulder, An introduction to tolerance intersection graphs, In *Graph Theory, Combinatorics and Applications* (Y. Alavi, et al., Eds.) Wiley Interscience, vol. 2 (1991), pp. 705–723.
- [JaMcSc91] M. S. Jacobson, F. R. McMorris and E. R. Scheinerman, General results on tolerance intersection graphs, *J. Graph Theory* 15 (1991), 573–577.
- [LaWi99] Y. Lai and K. Williams, A survey of solved problems and applications on bandwidth, edgesum, and profile of graphs, *Journal of Graph Theory* 31 (1999), 75–94.
- [LuMa83] J. R. Lundgren and J. S. Maybee, A characterization of graphs of competition number  $m$ , *Discrete Appl. Math.* 6 (1983), 319–322.
- [Ma45] E. Szpilrajn-Marczewski, Sur deux propriétés des classes d'ensembles, *Fund. Math.* 33 (1945), 303–307.
- [McMc99] T. A. McKee and F. R. McMorris, *Topics in Intersection Graph Theory*, SIAM monograph, 1999.
- [Mi91] Z. Miller, Graph layouts, *Applications of Discrete Mathematics*, J. G. Michaels and K. H. Rosen (Editors), McGraw-Hill, New York (1991), 365–393.
- [Or62] O. Ore, *Theory of graphs*, *Amer. Math. Soc. Transl.* 38 (1962), 206–212.
- [Wa78] J. R. Walter, Representations of chordal graphs as subtrees of a tree, *Journal of Graph Theory* 2 (1978), 265–267.
- [WaAcSa79] H. B. Walikar, B. D. Acharya, and E. Sampathkumar, Recent developments in the theory of domination in graphs, Mehta Research Institute, Allahabad, MRI *Lecture Notes in Math.*, 1 (1979).

## Chapter 11: Analytic Graph Theory

- [BoNi04] B. Bollobás and V. Nikiforov, *Extremal Graph Theory*, §8.1 in *Handbook of Graph Theory*, eds., J. L. Gross and J. Yellen, CRC Press, 2004.
- [Fa04] R. Faudree, *Ramsey Graph Theory*, §8.3 in *Handbook of Graph Theory*, eds., J. L. Gross and J. Yellen, CRC Press, 2004.
- [GaRoSp90] R. L. Graham, B. L. Rothschild, and J. H. Spencer, *Ramsey Theory*, John Wiley & Sons, 1990.
- [GrGl55] R. E. Greenwood and A. M. Gleason, Combinatorial relations and chromatic graphs, *Canad. J. Math.* 7 (1955), 1–7.
- [Tu41] P. Turán, On an extremal problem in graph theory (Hungarian), *Mat. és Fiz. Lapok* 48 (1941), 436–452.

[Wo04] N. Wormald, *Random Graphs*, §8.2 in *Handbook of Graph Theory*, eds., J. L. Gross and J. Yellen, CRC Press, 2004.

### Chapter 12: Special Digraph Models

[Ba83] S. Baase, *Computer Algorithms: Introduction to Design and Analysis*, Addison-Wesley, 1983.

[BeTh81] J.-C. Bermond and C. Thomassen, Cycles in digraphs: a survey, *J. Graph Theory* 5 (1981) 1–43.

[BuWeKi04] E. Burke, D. de Werra, and J. Kingston, *Applications to Timetabling*, §5.6 in *Handbook of Graph Theory*, eds., J. L. Gross and J. Yellen, CRC Press, 2004.

[Ci75] E. Cinlar, *An Introduction to Stochastic Processes*, Prentice-Hall, 1975.

[Hä93] R. Häggkvist, Hamiltonian cycles in oriented graphs, *Combin. Probab. Comput.* 2 (1993), 25–32.

[La53] H. G. Landau, On dominance relations and the structure of animal societies. III. The condition for a score structure, *Bull. Math. Biophys.* 15 (1953), 143–148.

[Ma04] S.B. Maurer, *Directed Acyclic Graphs*, §3.2 in *Handbook of Graph Theory*, eds., J. L. Gross and J. Yellen, CRC Press, 2004.

[Ma80] S. Maurer, The king chicken theorems, *Math. Mag.* 53 (1980), 67–80.

[Mo68] J. W. Moon, *Topics on Tournaments*, Holt, Rinehart, and Winston, 1968.

[Re04] K. B. Reid, *Tournaments*, §3.3 in *Handbook of Graph Theory*, eds., J. L. Gross and J. Yellen, CRC Press, 2004.

[Re96] K. B. Reid, Tournaments: scores, kings, generalizations and special topics, *Congressus Numer.* 115 (1996), 171–211.

[St93] H. J. Straight, *Combinatorics: An Invitation*, Brooks/Cole, 1993.

[Th82] C. Thomassen, Edge-disjoint Hamiltonian paths and cycles in tournaments, *Proc. London Math. Soc.* 45 (1982), 151–168.

[ThSw92] K. Thulasiraman and M. N. S. Swamy, *Graphs: Theory and Algorithms*, John Wiley & Sons, 1992.

[Zh80] C. Q. Zhang, Every regular tournament has two arc-disjoint Hamiltonian cycles, *J. Qufu Normal College, Special Issue Oper. Res.* (1980), 70–81.

### Chapter 13: Network Flows and Related Problems

[Ed65b] J. Edmonds, Paths, trees, and flowers, *Canad. J. Math.* 17 (1965), 449–467.

- [EdKa72] J. Edmonds and R. M. Karp, Theoretical improvements in algorithmic efficiency for network flow problems, *JACM* 19 (1972), 248–264.
- [Ev79] S. Even, *Graph Algorithms*, Computer Science Press, 1979.
- [FoFu62] L. R. Ford and D. R. Fulkerson, *Flows in Networks*, Princeton University Press, 1962.
- [Kö16] D. König, Über Graphen und ihre Anwendung auf Determinantentheorie und Mengenlehre, *Math. Ann.* 77 (1916), 453–465.
- [Mi78] E. Minieka, *Optimisation Algorithms for Networks and Graphs*, Marcel Dekker, 1978.
- [Pe1891] J. Petersen, Die Theorie der regulären Graphen, *Acta Math.* 15 (1891), 193–220.
- [ThSw92] K. Thulasiraman and M. N. S. Swamy, *Graphs: Theory and Algorithms*, John Wiley & Sons, 1992.

### **Chapter 14: Graphical Enumeration**

- [Bo00] K. P. Bogart, *Introductory Combinatorics*, Third Edition, Academic Press, 2000.
- [Br04b] R. A. Brualdi, *Introductory Combinatorics*, Fourth Edition, Prentice Hall, 2004.
- [HaPa73] F. Harary and E. M. Palmer, *Graphical Enumeration*, Academic Press, 1973.
- [Po37] G. Polya, Kombinatorische Anzahlbestimmungen für Gruppen, Graphen und Chemische Verbindungen, *Acta Mathematica* 68 (1937), 145–254.

### **Chapter 15: Algebraic Specification of Graphs**

- [GrCh96] J. L. Gross and J. Chen, Algebraic specification of interconnection networks by permutation voltage graph morphisms, *Math. Systems Theory* 29 (1996), 451–470.
- [GrTu77] J. L. Gross and T. W. Tucker, Generating all graph coverings by permutation voltage assignments, *Discrete Math.* 18 (1977), 273–283.
- [Le92] F. T. Leighton, *Introduction to Parallel Algorithms and Architectures*, Morgan Kaufmann, 1992.

### **Chapter 16: Non-Planar Layouts**

- [AlGr76] S. R. Alpert and J. L. Gross, Components of branched coverings of current graphs, *J. Combin. Theory Ser. B* 20 (1976), 283–303.
- [Du66] R. A. Duke, The genus, regional number, and Betti number of a graph, *Canad. J. Math.* 18 (1966), 817–822.

- [Ed60] J. Edmonds, A combinatorial representation for polyhedral surfaces, *Notices Amer. Math. Soc.* 7 (1960), A-646.
- [Ed65c] J. R. Edmonds, On the surface duality of linear graphs, *J. Res. Nat. Bur. Standards Sect. B* (1965), 121–123.
- [FuGrMc88] M. Furst, J. L. Gross, and L. McGeoch, Finding a maximum-genus graph imbedding, *JACM* 35 (1988), 523–534.
- [Gr74] J. L. Gross, Voltage graphs, *Discrete Math.* 9 (1974), 239–246.
- [GrAl73] J. L. Gross and S. R. Alpert, Branched coverings of graph imbeddings, *Bull. Amer. Math. Soc.* 79 (1973), 942–945.
- [GrAl74] J. L. Gross and S. R. Alpert, The topological theory of current graphs, *J. Combin. Theory Ser. B* 17 (1974), 218–233.
- [GrTu77] J. L. Gross and T. W. Tucker, Generating all graph coverings by permutation voltage assignments, *Discrete Math.* 18 (1977), 273–283.
- [GrTu87] J. L. Gross and T. W. Tucker, *Topological Graph Theory*, Dover, 2001. (First Edition, Wiley-Interscience, 1987.)
- [Gu63] W. Gustin, Orientable embedding of Cayley graphs, *Bull. Amer. Math. Soc.* 69 (1963), 272–275.
- [He1891] L. Heffter, Über das Problem der Nachbargebiete, *Math. Annalen* 38 (1891), 477–580.
- [NoStWh71] E. A. Nordhaus, B. M. Stewart, and A. T. White, On the maximum genus of a graph, *J. Combin. Theory Ser. B* 11 (1971), 258–267.
- [NoRiStWh72] E. A. Nordhaus, R. D. Ringeisen, B. M. Stewart, and A. T. White, A Kuratowski-type theorem for the maximum genus of a graph, *J. Combin. Theory Ser. B* 12 (1972), 260–267.
- [Th89] C. Thomassen, The graph genus problem is NP-complete, *J. of Algorithms* 10 (1989), 568–576.

## Appendix

- [Co71] S. A. Cook, The complexity of theorem proving procedures, *Proc. 3rd Annual ACM Symposium on Theory of Computing (STOC 71)* (1971), 151–158.
- [Ev79] S. Even, *Graph Algorithms*, Computer Science Press, 1979.
- [GaJo79] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, 1979.
- [Wi86] H. S. Wilf, *Algorithms and Complexity*, Prentice-Hall, 1986.

---

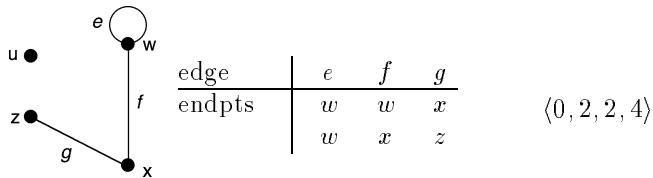
# SOLUTIONS AND HINTS

---

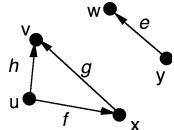
## Chapter 1 Introduction to Graph Models

### 1.1 Graphs and Digraphs

1.1.1:



1.1.4:



1.1.7:

edge	a	b	c	d
endpts	$y^h$	$y$	$z$	$x$
	$x$	$x^h$	$z^h$	$z^h$

1.1.10:

edge	a	b	c	d
endpts	$y$	$y$	$z$	$x$
	$x$	$x$	$z$	$z$

1.1.15: a. Does not exist. b. Does not exist.

1.1.22: No.

1.1.24: The *endpts* function must be one-to-one, and none of its images is a one-element set.

1.1.32: No, the conditions imply an even number of vertices.

1.1.36: The minimum number is 4.

1.1.39: The maximum number is 6.

1.1.42: The maximum number is 5.

1.1.45: A minimum dominating set has four vertices.

## 1.2 Common Families of Graphs

**1.2.1:** a.  $\binom{n}{2} = \frac{n^2-n}{2}$  b.  $mn$

**1.2.4:** a.  $n \leq 2$  b. All even  $n$ . c. All  $n$ .

**1.2.6:**  $U = \{x, v\}$  and  $W = \{u, z\}$

**1.2.10:** Use Euler's Degree-Sum Theorem.

**1.2.13:**  $2^{\frac{n^2-n}{2}}$

**1.2.23:** Let  $I_k = (0, k)$  for  $k = 1, \dots, n$ .

**1.2.25:** With a minus sign denoting BC, the following collection of intervals assigned to  $a_1, \dots, a_5$ , respectively, shows that the given graph is an interval graph.

$$(-100, 50) \quad (250, 400) \quad (100, 300) \quad (0, 200) \quad (150, 300)$$

## 1.3 Graph Modeling Applications

**1.3.1:** A 4-coloring is easy to construct, and a 3-coloring is impossible because vertices  $G, C, D, E$  are mutually adjacent. Hence four is the minimum number of frequencies.

**1.3.3:** .048

**1.3.5:** All binary strings that have an odd number of ones.

**1.3.6:** Vertices correspond to the volunteers, and two vertices are adjacent if the corresponding volunteers have a language in common. Then a maximum-size collection of vertex-disjoint edges is desired.

**1.3.13:** d. No two vertices have a pair of oppositely directed arcs between them.

## 1.4 Walks and Distance

**1.4.1:** a. yes b. yes c. no d. yes

**1.4.4:** Only the first two represent connected graphs.

**1.4.6:** Not strongly connected. For example, vertex  $u$  is not reachable from vertex  $x$ .

**1.4.10:** Any digraph that has a directed closed walk.

**1.4.12:** The distance equals 4.

**1.4.15:** Any connected graph that contains no closed walks.

**1.4.18:**  $diam(G) = 3$ ,  $rad(G) = 2$ , and the central vertices are  $v$  and  $z$ .

**1.4.32:** For the second inequality, let  $x$  and  $y$  be two vertices such that  $d(x, y) = diam(G)$ , and let  $w$  be a central vertex. Then apply the triangle inequality.

**1.4.34:** Rigid

**1.4.38:** Choose any vertex  $x_1 \in V_G - \{u, v\}$ , and let  $W_1$  be a  $u-x_1$  walk. Choose  $x_2 \in V_G - \{u, v, x_1\}$ , and let  $W_2$  be an  $x_1-x_2$  walk. Continue until the last vertex,  $x_m$ , is chosen, and let  $W_{m+1}$  be an  $x_m-v$  walk. Then form the concatenation of walks  $W_1$  through  $W_{m+1}$ .

**1.4.41:** There are two walks of length 2 and seven walks of length 3.

**1.4.46:** Let  $x$  and  $y$  be any two non-adjacent vertices of graph  $G$ . Since  $G$  is a simple graph, the condition implies that the set of vertices adjacent to vertex  $x$  and the set of vertices adjacent to  $y$  have at least one vertex in common.

## 1.5 Paths, Cycles, and Trees

**1.5.1:** a. It is a walk of length 4, but it is not a path.

**1.5.3:** b.  $\langle r, s, n \rangle$

**1.5.5:** The three  $w-v$  paths can be concatenated with the three  $v-r$  paths. Hence there are nine different  $w-r$  paths.

**1.5.8:** Let  $e$  be any edge with endpoints  $x$  and  $y$ , and consider the walk  $\langle x, e, y, e, x \rangle$ .

**1.5.11:** None of length 2 or 4, and four paths of length 3.

**1.5.15:** The girth equals 4.

**1.5.30:** Consider a nontrivial closed subtrail of minimal length that contains vertex  $v$ .

**1.5.32:** Repeated vertices would imply the existence of a directed cycle, which could then be deleted, leading to a shorter directed walk.

**1.5.40:** The root represents the empty board. Each child represents the configuration that results from placing an x in one of the positions, etc.

**1.5.43:** Start at any vertex. Since that vertex has nonzero outdegree, there is an arc directed from it to some vertex. Now use the finiteness of the vertex set to argue that eventually a vertex must be repeated.

## 1.6 Vertex and Edge Attributes: More Applications

**1.6.2:**  $(n - 1)!$

**1.6.3:** The  $n$  vertices of a graph represent the exams. When it is undesirable to schedule two exams in overlapping timeslots, the two corresponding vertices are joined by an edge, and that edge is assigned a weight indicating how undesirable a conflict would be. The objective would then be to assign one of  $k$  possible *colors* to each of the  $n$  vertices so that the sum of the weights of all edges whose endpoints have the same color is minimized. Vertex coloring is discussed in §9.1.

**1.6.5:** Modify the model used in Application 1.3.1 by assigning a weight to each edge. The objective is to find a maximum-weight matching (see §13.4).

**1.6.7:** Some variation of a network-flow model (see §13.1).

**1.6.8:** This is a type of vehicle-routing problem that is notoriously difficult computationally. One approach is to first partition the bus stops into  $m$  subsets and then solve  $m$  separate traveling salesman problems to assess the cost (mileage) of that partition.

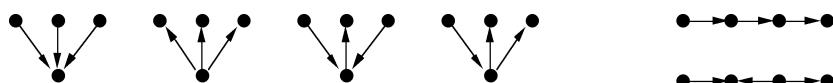
## Chapter 2 Structure and Representation

### 2.1 Graph Isomorphism

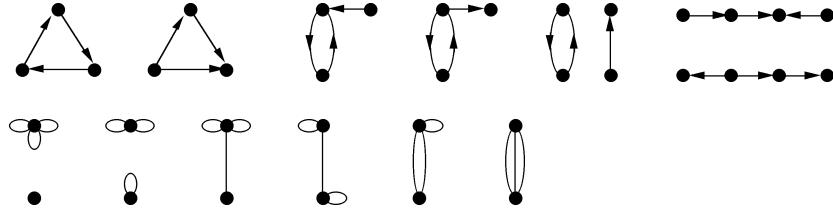
**2.1.1:**



2.1.7:



2.1.11:

2.1.15:  $v \rightarrow r; w \rightarrow s; u \rightarrow t; x \rightarrow z$  or  $v \rightarrow r; w \rightarrow s; u \rightarrow z; x \rightarrow t$ 

## 2.2 Automorphisms and Symmetry

2.2.1: identity automorphism and  $(u)(x)(v\ w)$ 2.2.5: identity automorphism and  $\{f_V = (u)(x)(v)(w), f_E = (a)(b)(c\ d)\}$ 

2.2.9: For any two vertices of  $K_n$ , the vertex-bijection that swaps them and fixes all the others is adjacency-preserving. It is its own inverse, so the inverse bijection is also adjacency-preserving.

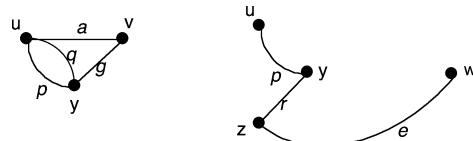
2.2.14:  $\text{circ}(9 : 1, m)$  is edge-transitive unless  $m = 3$  or  $6$  (in which case a 3-edge or a 6-edge would lie on a 3-cycle, but a 1-edge does not).

2.2.18: vertex orbits  $\{u\}, \{x\}, \{v, w\}$ ; edge orbits  $\{ux\}, \{vw\}, \{vx, wx\}$ 

## 2.3 Subgraphs

2.3.1: a. Yes:  $\text{endpoints}(q) = \{u, g\} \subset W$ ;  $\text{endpoints}(g) = \{v, y\} \subset W$ .

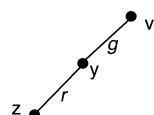
2.3.3:



2.3.7:



2.3.13:



2.3.19: (a) There are 6 cliques; each has 3 vertices. (b)  $\omega = 3$ . (c)  $uyz, uw, xv, xw, xz, rw, rz$  are maximal independent sets. (d)  $\alpha = 3$ . (e)  $x$  and  $z$  are non-central. The center is the subgraph induced by the other five vertices.

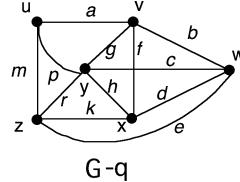
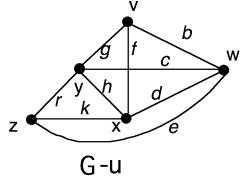
2.3.23: Using the solution above to Exercise 19, we see that the largest cardinality among such sets is 2. For instance,  $wz$  is such a set.

2.3.27: There are four spanning trees that contain edge  $p$  and four that do not.

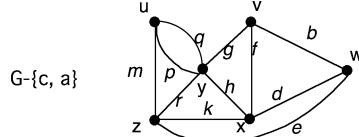
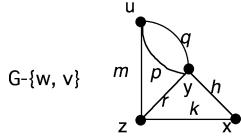
2.3.37: If one tree is  $K_1$ , the other tree is one of the three trees with 5 vertices. If one tree is  $K_2$ , the other tree is  $K_{1,3}$  or  $P_4$ . Or both trees are  $P_3$ .

## 2.4 Some Graph Operations

**2.4.1:**



**2.4.4:**



**2.4.8:**  $u, v, z$  are the cutpoints.

**2.4.12:** 2;  $\{b, d\}$ .

**2.4.16:** 4;  $\{da, db, du, dc\}$ .

**2.4.20:**



**2.4.22:** There are 4 vertices and 4 edges. The degree sequence is 3211. There is only one such graph.



**2.4.35:** False. If  $G$  is not connected, then vertex  $v$  is a cut-vertex of graph  $G + v$ . If  $G$  is connected, then  $G + v$  has no cut-vertices.

**2.4.40:** Assume that  $G$  is not connected, and let  $x$  and  $y$  be two vertices in  $\overline{G}$ . If  $x$  and  $y$  are in different components of  $G$ , then they are adjacent in  $\overline{G}$ . If  $x$  and  $y$  are in the same component of  $G$ , then there is at least one vertex  $w$  in some other component of  $G$ , and  $w$  is adjacent to both  $x$  and  $y$  in  $\overline{G}$ .

## 2.5 Tests for Non-Isomorphism

**2.5.1:** a. The given graph is isomorphic to graph  $B$ .

**2.5.2:** One graph has two 3-cycles, but the other graph has none.

**2.5.8:** Non-isomorphic because they have a different degree sequence.

**2.5.13:** Start with a digraph version of Theorem 2.1.3.

**2.5.20:** Isomorphic.

## 2.6 Matrix Representations

**2.6.1:**

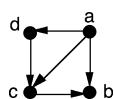
$$A_G = \begin{pmatrix} u & v & w & x & y \\ u & 0 & 2 & 1 & 0 & 0 \\ v & 2 & 1 & 1 & 1 & 0 \\ w & 1 & 1 & 0 & 0 & 0 \\ x & 0 & 1 & 0 & 0 & 0 \\ y & 0 & 0 & 0 & 0 & 0 \end{pmatrix}; \quad I_G = \begin{pmatrix} a & b & c & d & e & f \\ u & 1 & 0 & 1 & 0 & 0 & 1 \\ v & 1 & 2 & 1 & 1 & 1 & 0 \\ w & 0 & 0 & 0 & 0 & 1 & 1 \\ x & 0 & 0 & 0 & 1 & 0 & 0 \\ y & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

2.6.4:

$$A_D = w \begin{pmatrix} u & v & w & x & y \\ u & 0 & 1 & 0 & 0 & 0 \\ v & 1 & 1 & 1 & 1 & 0 \\ w & 1 & 0 & 0 & 0 & 0 \\ x & 0 & 0 & 0 & 0 & 0 \\ y & 0 & 0 & 0 & 0 & 0 \end{pmatrix}; \quad I_D = w \begin{pmatrix} a & b & c & d & e & f \\ u & -1 & 0 & 1 & 0 & 0 & 1 \\ v & 1 & 2 & -1 & -1 & -1 & 0 \\ w & 0 & 0 & 0 & 0 & 1 & -1 \\ x & 0 & 0 & 0 & 1 & 0 & 0 \\ y & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

2.6.9: If the  $m$  vertices of one side of the bipartition are listed first, then the adjacency matrix will consist of an  $m \times m$  matrix of 0's in the upper left corner, an  $m \times n$  matrix of 1's in the upper right corner, an  $n \times m$  matrix of 1's in the lower left corner, and an  $m \times m$  matrix of 0's in the lower right corner.

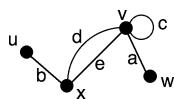
2.6.14:



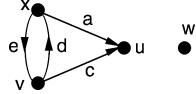
2.6.16:  $A_G^2[a, a] = (0, 1, 1, 1) \cdot (0, 1, 1, 1) = 3$ .

2.6.18:  $A_D^2[d, a] = (0, 0, 1, 0) \cdot (0, 0, 0, 0) = 0$ .

2.6.20:

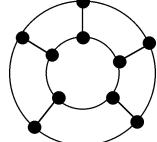


2.6.22:

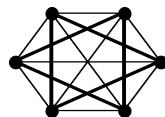


## 2.7 More Graph Operations

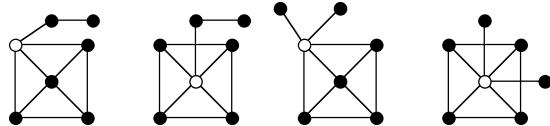
2.7.1:



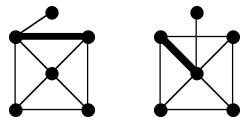
2.7.7:



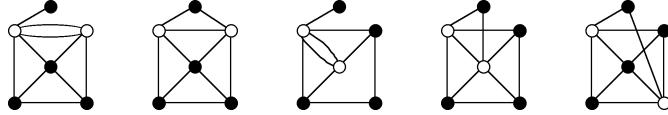
2.7.12:



2.7.16:



2.7.20:



## Chapter 3 Trees

### 3.1 Characterizations and Properties of Trees

**3.1.3:** The only such graph is  $K_5$  plus two isolated vertices.

**3.1.9:** There is no such graph, because a tree would have eight edges, and each additional edge would create at least one cycle.

**3.1.13:** The smallest average degree occurs when the connected graph is a tree, so by Euler's Degree-Sum Theorem and Proposition 3.1.3,  $\frac{\sum v}{n} = \frac{2|E|}{n} = 2 - \frac{2}{n}$ .

**3.1.17:** Use Part 6 of Theorem 3.1.8 to show it is true.

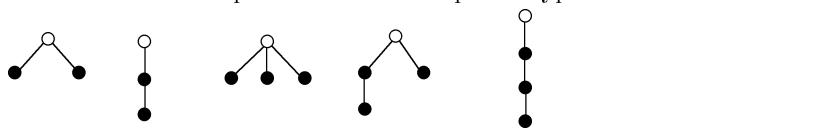
**3.1.20:** If  $H$  were not connected, then there would be an edge  $e \in E_G - E_H$  whose endpoints are in different components of  $H$ . But then  $H + e$  would contradict the maximality of  $H$ . Use Part 6 of Theorem 3.1.8 to prove the reverse implication.

**3.1.24:** Let  $x$  and  $y$  be any two non-adjacent vertices of graph  $G$ . Since  $G$  is a simple graph, the condition implies that the set of vertices adjacent to vertex  $x$  and the set of vertices adjacent to  $y$  have at least one vertex in common.

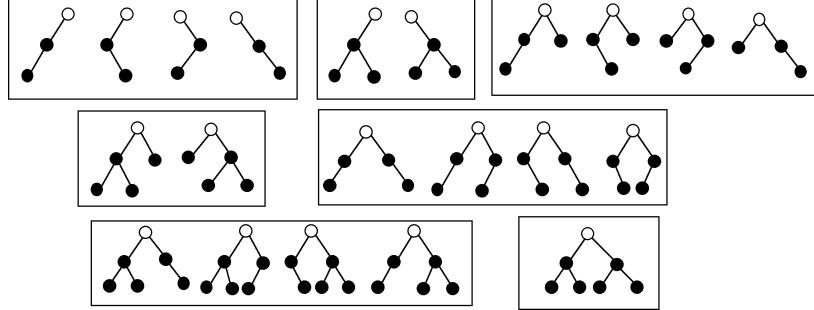
**3.1.27:** All acyclic graphs.

### 3.2 Rooted Trees, Ordered Trees, and Binary Trees

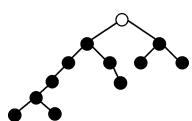
**3.2.1:** The two 3-vertex rooted trees shown below are isomorphic as graphs, and the three 4-vertex rooted trees represent two isomorphism types.



3.2.3:

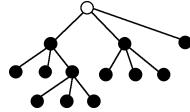


3.2.5:

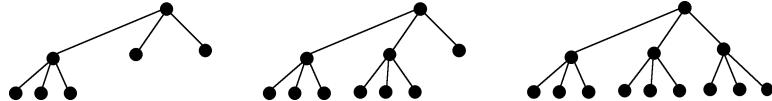


**3.2.8:** There are several such trees. For instance, start with a path of length 2, and attach 11 children to the last vertex.

**3.2.13:**



**3.2.17:** Start by considering the three ternary trees of height 2 whose internal vertices have exactly three children, as shown below.



Then determine the number of different ways three children can be attached to leaves at depth 2. For instance, there is only one way (up to rooted-tree isomorphism) to attach three children to each of two of the leaves of the tree on the left, but there are two different ways of attaching three children to each of two of the leaves of the middle tree.

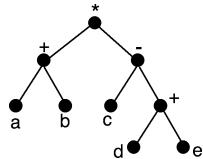
**3.2.21:**  $-0, 0, 1, 1, 1, 5, 5$

**3.2.23:** The recurrence relation is  $f(h) = 2f(h-1) + 1$ . By the induction hypothesis,  $f(h) = 2(2^h - 1) + 1 = 2^{h+1} - 1$ .

### 3.3 Binary-Tree Traversals

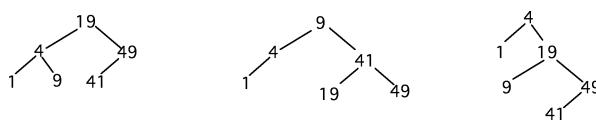
**3.3.1:** in-order: 1,2,3,4,5,6,7,8,9,10; pre-order: 1,2,4,7,8,5,3,6,9,10;  
post-order: 7,8,4,5,2,9,10,6,3,1

**3.3.5:** prefix: \*+ab-c+de      postfix: ab+cd+-\*

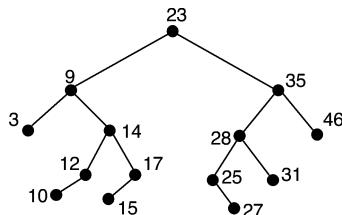


### 3.4 Binary-Search Trees

**3.4.1:**



**3.4.5:** The following tree results after the two insertions and one deletion.



**3.4.11:** 13, 3, 55, 2, 5, 34, 144, 1, 8, 21, 89

**3.4.13:** When the keys are in either descending order or in ascending order.

### 3.5 Huffman Trees and Optimal Prefix Codes

3.5.1: facade

3.5.4:

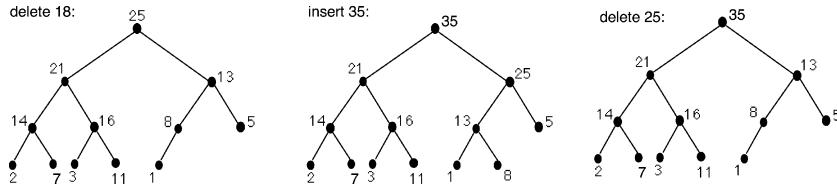
letter	a	b	c	d	e	f	g	h	av. wt. length
codeword	00	1000	010	011	110	101	111	1001	

defaced = 011110101000101110011; baggage = 10000011111100111110

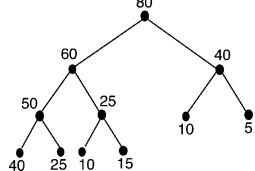
### 3.6 Priority Trees

3.6.2:  $2^h$

3.6.6:



3.6.9:



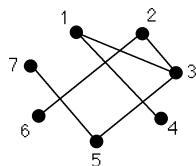
3.6.11: The heap does not represent a priority tree because it puts 22 as a child of 20.

### 3.7 Counting Labeled Trees: Prüfer Encoding

3.7.1:  $\langle 4, 2, 2, 4 \rangle$

3.7.6:  $\langle 1, 1, 3, 1, 3 \rangle$

3.7.10:



3.7.14: Exactly one of the  $m$  vertices on the  $m$ -side of the bipartition has degree 2, and each of the other  $m - 1$  vertices on that side is joined to exactly one of the two vertices on the other side of the bipartition. Hence, the number of spanning trees is  $m \cdot 2^{m-1}$ .

### 3.8 Counting Binary Trees: Catalan Recursion

3.8.1:  $b_4 = b_0 b_3 + b_1 b_2 + b_2 b_1 + b_3 b_0 = 14 = \frac{1}{5} \binom{8}{4}$

**3.8.4:** Each vertex of a binary tree on  $n - 1$  vertices represents a multiplication, and the lower the depth of the vertex, the earlier that multiplication must be performed. Thus, the number of different ways of multiplying  $n$  numbers equals the number of different binary trees on  $n - 1$  vertices.

**3.8.7:** Use the fact that

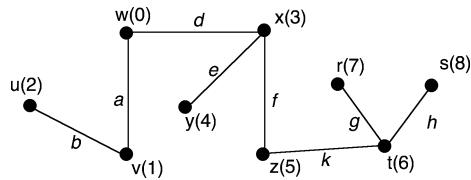
$$(2n)! = (2n)(2n-1)(2n-2) \dots (3)(2)(1) = 2^n n!(2n-1)(2n-3) \dots (3)(1)$$

## Chapter 4 Spanning Trees

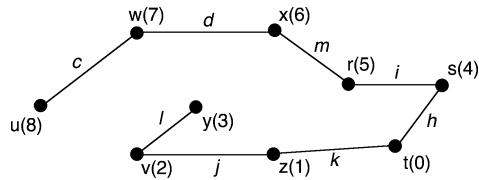
### 4.1 Tree Growing

**4.1.1:**  $\{f, g, i, j\}$

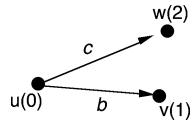
**4.1.4a:**



**4.1.8b:**



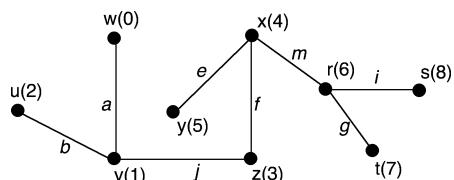
**4.1.10:**



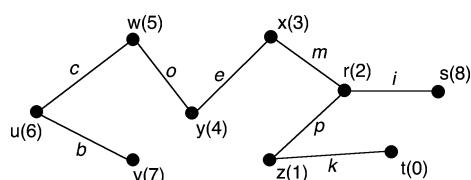
**4.1.16:** The graph would have to be a path graph, and the starting vertex would have to be at one of the ends of the path.

### 4.2 Depth-First and Breadth-First Search

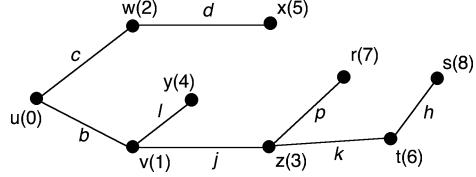
**4.2.1a:**



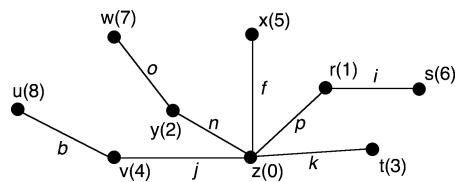
**4.2.5b:**



4.2.9a:



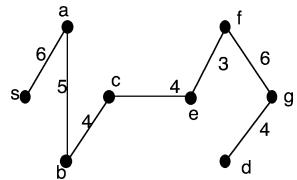
4.2.13b:



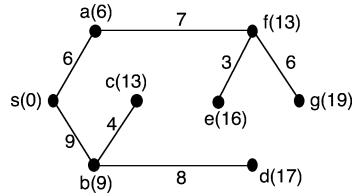
4.2.16: Trees.

### 4.3 Minimum Spanning Trees and Shortest Paths

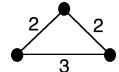
4.3.1: total weight = 32



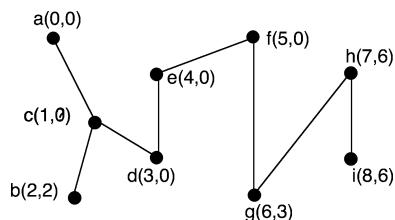
4.3.5:



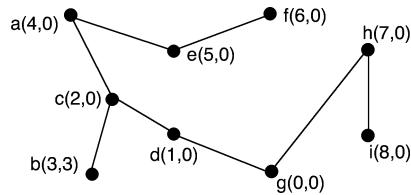
4.3.10:



### 4.4 Applications of Depth-First Search

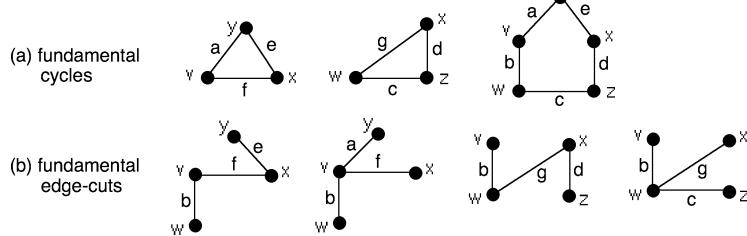
4.4.1: According to the depth-first tree shown below,  $dfnumber(c) \leq low(b)$  and  $dfnumber(g) \leq low(h)$ , verifying the assertion of Corollary 4.4.12.

**4.4.5:** Vertex  $c$  is a cut-vertex and is the only non-root in the tree shown below that meets the condition for a cut-vertex given in Corollary 4.4.12.



## 4.5 Cycles, Edge-Cuts, and Spanning Trees

**4.5.1:**



**4.5.6a:** For instance, the first fundamental cut in the solution to 4.5.1(a) above has two edges in common with the spanning tree whose edge-set is  $\{a, b, d, f\}$ , it has one edge in common with  $\{a, e, g, d\}$ , and it has two edges in common with  $\{e, f, c, d\}$  (there are several other spanning trees to be considered).

**4.5.6b:** For instance, the first fundamental cycle in the solution to 4.5.1(a) above has edge  $g$  in common with the relative complement of the spanning tree whose edge-set is  $\{a, b, d, f\}$ , has edge  $f$  in common with the relative complement of the tree whose edge-set is  $\{a, e, g, d\}$ , and has edges  $e$  and  $f$  in common with the relative complement of the tree whose edge-set is  $\{a, b, g, c\}$ .

**4.5.6c:** The 5-cycle has edges  $b$  and  $e$  in common with the first fundamental edge-cut listed in the solution to 4.5.1(b) above, it has edges  $a$  and  $b$  in common with the second edge-cut in the list, it has edges  $b$  and  $d$  in common with the third one, and it has edges  $b$  and  $c$  in common with the last one in the list.

**4.5.10a:** Consider the spanning tree whose edge-set is  $\{a, e, c, d\}$ , and consider the fundamental cycle with respect to edge  $f$  (which is listed first in the solution to 4.5.1a above). The other two edges of this cycle are  $a$  and  $e$ . The two fundamental edge-cuts with respect to these two edges are the first two listed in the solution to 4.5.1(b), and these are the only two that contain the edge  $f$ .

**4.5.10b:** Consider the spanning tree whose edge-set is  $\{a, e, c, d\}$ , and consider the fundamental edge-cut with respect to edge  $e$  (which is listed first in the solution to 4.5.1(b)). The other two edges of this edge-cut are  $f$  and  $b$ . The fundamental cycle with respect to  $f$  and the fundamental cycle with respect to  $b$  are the first and third ones listed in the solution to 4.5.1(a), and these two fundamental cycles are the only ones that contain edge  $e$ .

**4.5.16:** Let  $v$  be a vertex adjacent to vertices  $x$  and  $y$ . If there is an  $x-y$  path  $P$  that avoids  $v$ , then there is a cycle involving  $P$ ,  $v$ , and the two edges joining  $v$  to  $x$  and  $y$ .

**4.5.22:** Consider the fundamental systems of cycles and edge-cuts in Figure 4.5.2. The edge-cut  $\{b, c, e, f\}$  has three edges in common with the 5-cycle of the graph.

## 4.6 Graphs and Vector Spaces

**4.6.1a:**  $\{b, c, f\}, \{a, c, e\}, \{b, d, e\}, \{a, d, f\}, \{b, c, f\}, \{a, b, c, d\}, \{a, b, e, f\}, \{c, d, e, f\}$

**4.6.1b:**  $\{a, c, f\}, \{a, d, e\}, \{b, c, e\}, \{b, d, f\}, \{a, b, e, f\}, \{a, b, c, d\}, \{c, d, e, f\}$

**4.6.4a:** The fundamental system of cycles consists of  $\{a, c, e\}$ ,  $\{b, d, e\}$ , and  $\{a, d, f\}$ . Each of the other five non-null elements of  $W_C(G)$  (listed in the solution of 4.6.1(a)) is a mod 2 sum of some or all of these three edge-sets. For instance,  $\{a, b, c, d\} = \{a, c, e\} + \{b, d, e\}$ .

**4.6.4b:** The fundamental system of edge-cuts consists of  $\{b, c, e\}$ ,  $\{b, d, f\}$ , and  $\{a, c, f\}$ . Each of the other four non-null elements of  $W_C(G)$  (listed in the solution of 4.6.1(b)) is a mod 2 sum of some or all of these three edge-sets. For instance,  $\{a, b, c, d\} = \{a, c, f\} + \{b, d, f\}$ .

**4.6.10:** For the vertex order  $u, w, x, y, z$ , the columns of  $C_D$  for the edge set  $D = \{a, b, c, d\}$  are

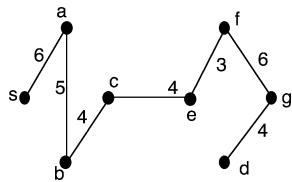
$$\begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

and their mod 2 sum is not the zero vector. This is consistent with the assertion that edge-set  $D$  is neither a cycle nor a disjoint union of cycles. Also, the sum of the first two column vectors equals the third one, which shows that  $C_D$  does not form a basis for the column space. This is consistent with the assertion that the edge-set  $D$  does not induce a spanning tree of the graph.

**4.6.17:** The non-null elements of  $W_C(G)$  are the three cycles of graph  $G$ , and none of these is an element of  $W_S(G)$ . Thus, by Theorem 4.6.11, the two subspaces are orthogonal complements.

## 4.7 Matroids and the Greedy Algorithm

**4.7.1:** Since the edges were not labeled, ties were broken according to the alphabetical order of the edges' endpoints.



**4.7.5:** It is an hereditary subset system but is not a matroid. For instance, in the star graph  $K_{1,n}$ , the vertex of degree  $n$  and any two other vertices do not satisfy the augmentation property.

**4.7.9:** It is not an hereditary subset system because  $\emptyset \notin \mathcal{I}$ .

**4.7.13:** Show that  $(C_1 \cup C_2) - x$  is a dependent set.

**4.7.16:** Suppose that the independent set  $B$  obtained by the greedy algorithm is not of minimum weight. Then there is some maximal independent set  $A$  such that  $w(A) < w(B)$ . Since  $A$  and  $B$  are both maximal independent sets, the augmentation property implies that they have the same number of elements. Let  $A = \{a_1, a_2, \dots, a_r\}$ , where  $w(a_1) \leq w(a_2) \leq \dots \leq w(a_r)$ , and let  $B = \{b_1, b_2, \dots, b_r\}$ , where  $b_i$  is the element chosen in the  $i$ th iteration of the greedy algorithm. Let  $j$  be the smallest subscript such that  $w(a_j) < w(b_j)$  ( $j$  exists since  $w(A) < w(B)$ ). By the augmentation property applied to the independent sets  $A_j = \{a_1, \dots, a_j\}$  and  $B_{j-1} = \{b_1, \dots, b_{j-1}\}$ , there is some  $a \in A_j - B_{j-1}$  such that  $B_{j-1} \cup \{a\}$  is an independent set. But  $w(a) \leq w(a_j) < w(b_j)$ , which contradicts the choice of  $b_j$  at the  $j$ th iteration.

## Chapter 5 Connectivity

### 5.1 Vertex- and Edge-Connectivity

**5.1.1:**  $\{y, v\}$  and  $\{v, w\}$

**5.1.5:** Corollary 5.1.5 implies no such graph exists.

**5.1.9:**  $\kappa_v(G) = \kappa_e(G) = 4$

**5.1.13:** Consider three specific pairs of adjacent vertices and three pairs of non-adjacent ones for the cases in which both vertices are on the inner cycle, both are on the outer cycle, and one is on the inner cycle and the other is on the outer cycle, and show none of these pairs can be a vertex-cut. Then conclude, by Corollary 5.1.6, that  $\kappa_v(G) = \kappa_e(G) = 3$ .

**5.1.17:** The graph in Figure 5.1.1.

**5.1.21:** Generalize the graph in Figure 5.1.1. (This construction is due to Chartrand and Harary [ChHa68].)

**5.1.23:** Since  $2 = \kappa_v(G) \leq \kappa_e(G)$ , every edge must be on that unique cycle. Hence, the graph must be a cycle graph.

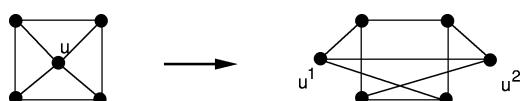
**5.1.26:** See Menger's Theorem 5.3.4.

### 5.2 Constructing Reliable Networks

**5.2.1:**

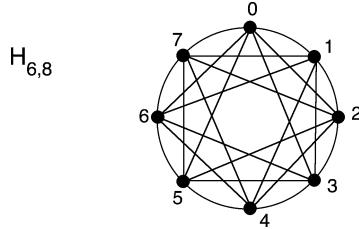


**5.2.5:**



**5.2.7:** Use induction. Observe that the Harary graph  $H_{3,4}$  is the 4-vertex wheel graph and that  $H_{3,n}$  is an  $n$ -cycle plus some diameters or quasi-diameters, depending on the parity of  $n$ . Then show that if  $n$  is odd,  $H_{3,n+1}$  can be obtained from  $H_{3,n}$  by a single type 2 operation, and if  $n$  is even, by a type 1 operation followed by a type 2.

5.2.11:



5.2.14: Use Corollary 5.1.6.

### 5.3 Max-Min Duality and Menger's Theorems

**5.3.1:** There are several collections of two internally disjoint paths, for instance,  $\mathcal{P} = \{\langle u, t, y, v \rangle, \langle u, s, w, z, v \rangle\}$ . The  $u-v$  separating vertex set  $\{s, t\}$  shows that  $\mathcal{P}$  is a maximum-size collection.

**5.3.5:** The  $u-v$  separating edge set consisting of the two edges incident on vertex  $u$  shows that the two paths given in the solution to 5.3.1 form a maximum-size collection of edge-disjoint  $u-v$  paths.

**5.3.10:** The collection  $\mathcal{P} = \{\langle u, t, y, v \rangle, \langle u, s, a, b, v \rangle, \langle u, a, w, x, z, v \rangle\}$  is a maximum-size collection of arc-disjoint directed  $u-v$  paths, because the three arcs directed from vertex  $u$  form a  $u-v$  separating arc set.

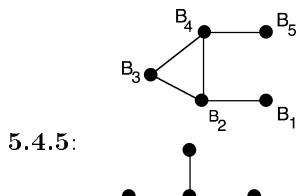
**5.3.15:** Clearly, between each pair of vertices  $u, v$  in a wheel graph, there are exactly three internally disjoint  $u-v$  paths. For the induction step, a type 1 operation in a Tutte synthesis adds an edge and hence, cannot reduce the number of paths. Let  $u$  and  $v$  be any two vertices in the graph that results from a type 2 operation, and suppose vertex  $w$  is split into vertices  $w'$  and  $w''$ .

Case 1:  $u \neq w'$  and  $v \neq w''$ . By the induction hypothesis, there are at least  $k$  internally disjoint  $u-v$  paths in the graph before the operation is performed. If none of these paths uses vertex  $w$ , then they are unaffected by the type 2 operation and hence, are still internally disjoint. Suppose that one of these  $k$  paths uses vertex  $w$ , say  $P = \langle u, x_1, x_2, \dots, x_j, w, x_{j+2}, \dots, x_l, v \rangle$ . After the type 2 operation, vertices  $x_j$  and  $x_{j+2}$  are either both adjacent to  $w'$ , both adjacent to  $w''$ , or one is adjacent to  $w'$  and the other is adjacent to  $w''$ . In each of these cases, path  $P$  is transformed into a  $u-v$  path whose only internal vertices are  $x_1, \dots, x_j, x_{j+2}, x_l$  and one or both of  $w'$  and  $w''$ , and thus, is still internally disjoint from the other  $k-1$  paths.

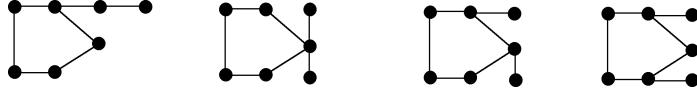
The two cases  $u = w'$  and  $v = w''$  and  $u = w'$  and  $v \neq w''$  are handled in a similar way.

### 5.4 Block Decompositions

**5.4.1:** The vertex-sets of the five blocks are  $B_1 = \{a, b\}$ ,  $B_2 = \{a, v\}$ ,  $B_3 = \{v, t\}$ ,  $B_4 = \{u, v, w, x, y, z\}$ , and  $B_5 = \{y, s\}$ . The block graph is shown below.



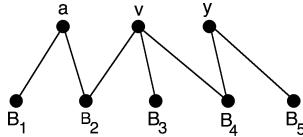
**5.4.8:** By Theorem 3.1.8, the graph must contain a unique cycle. Therefore, the three blocks must be a 5-cycle, and two  $K_2$ 's. Either the two  $K_2$ 's form a path that is joined to one vertex on the cycle, or they are adjacent to the same cycle-vertex, to two adjacent cycle-vertices, or to two non-adjacent cycle-vertices. Thus, there are four non-isomorphic graphs, as shown in the figure below.



**5.4.14:** The assertion is false even if the graphs are restricted to be simple and connected. For instance, the following two graphs have the same block graph.



**5.4.18:** For the block-cutpoint graph shown below, the labels on the block vertices correspond to the labels used in the solution to 5.4.1.



**5.4.22:** If the block-cutpoint graph had a cycle, then the union of the blocks corresponding to the block vertices on that cycle would be 2-connected, contradicting the maximality of each of those blocks.

## Chapter 6 Optimal Graph Traversals

### 6.1 Eulerian Trails and Tours

**6.1.1:**  $K_n$  is eulerian for all odd  $n \geq 3$ .

**6.1.5:** The octahedral graph is the only platonic graph that is eulerian.

**6.1.7:** Starting at the bottom left vertex and using alphabetical order for the default priority, we obtain the following sequence of iterations.

iteration 0: closed trail  $T = \langle b, d, e, f \rangle$

iteration 1: closed trail  $D = \langle g, a, j, c, k, h, m, i \rangle$ ;

enlarged trail  $T = \langle b, d, g, a, j, c, k, h, m, i, e, f \rangle$

**6.1.10:** see Algorithm 6.1.1a below.

**6.1.13:** Edges are chosen using an alphabetical default priority.

iteration 0: Let  $x$  be the odd-valent endpoint of edge  $e$ .

Let  $y$  be the odd-valent endpoint of edge  $n$ .

Let  $e^*$  be the new edge between  $x$  and  $y$ , and add  $e^*$  to graph  $G$ .

closed trail  $T = \langle e^*, c, b, p, n, l, j, k, g, d, m, i, q \rangle$

iteration 1: closed trail  $D = \langle e, a, r, o \rangle$ ;

enlarged trail  $T = \langle e, a, r, o, e^*, c, b, p, n, l, j, k, g, d, m, i, q \rangle$

circular shift:  $T = \langle e^*, c, b, p, n, l, j, k, g, d, m, i, q, e, a, r, o \rangle$

delete  $e^*$  from  $T$  and return the resulting open eulerian trail:

$T = \langle c, b, p, n, l, j, k, g, d, m, i, q, e, a, r, o \rangle$

**Algorithm 6.1.1a: Constructing an Open Eulerian Trail**

*Input:* a connected graph  $G$  that has exactly two vertices of odd degree.

*Output:* an open eulerian trail  $T$ .

Let  $x$  and  $y$  be the two vertices of odd degree.

Let  $e^*$  be a new edge between vertices  $x$  and  $y$ .

$G := G + e^*$

Start at vertex  $x$ , and construct a closed trail  $T$  in  $G$ , starting with edge  $e^*$ .

While there are edges of  $G$  not already in trail  $T$

    Choose any vertex  $w$  in  $T$  that is incident with an unused edge.

    Starting at vertex  $w$ , construct a closed trail  $D$  of unused edges.

    Enlarge trail  $T$  by splicing trail  $D$  into  $T$  at vertex  $w$ .

    Circularly shift the terms of the edge sequence of trail  $T$  to the right until edge  $e^*$  appears first.

    Delete edge  $e^*$  from the edge sequence of  $T$ .

Return  $T$ .

**6.1.15:****Algorithm 6.1.1b: Constructing a Directed Eulerian Tour**

*Input:* an eulerian digraph  $G$ .

*Output:* an eulerian tour  $T$ .

Start at any vertex  $v$ , and construct a closed directed trail  $T$  in  $G$ .

While there are arcs of  $G$  not already in trail  $T$

    Choose any vertex  $w$  in  $T$  that is the tail of an unused arc.

    Starting at vertex  $w$ , construct a closed directed trail  $D$  of unused arcs.

    Enlarge trail  $T$  by splicing trail  $D$  into  $T$  at vertex  $w$ .

Return  $T$ .

**6.1.18:** This digraph meets the conditions of Theorem 6.1.3. Let  $x$  be the vertex incident with arcs  $e$ ,  $m$ , and  $s$ , and let  $y$  be the vertex incident with arcs  $a$ ,  $b$ , and  $f$ . Adding a new arc directed from vertex  $y$  to vertex  $x$  makes the digraph eulerian. Assign the label  $e^*$  to this new arc. Shown below is the result of starting at vertex  $y$  and arc  $e^*$  and applying Algorithm 6.1.1b (from the solution to Exercise 6.1.15), using alphabetical order for the default priority.

iteration 0: closed trail  $T = \langle e^*, m, f \rangle$

iteration 1: closed trail  $D = \langle a, b \rangle$ ; enlarged trail  $T = \langle a, b, e^*, m, f \rangle$

iteration 2: closed trail  $D = \langle d, e, s \rangle$ ; enlarged trail  $T = \langle a, d, e, s, b, e^*, m, f \rangle$

iteration 3: closed trail  $D = \langle h, g, i, j, k \rangle$ ;

enlarged trail  $T = \langle a, d, e, s, b, e^*, m, h, g, i, j, k, f \rangle$

Next, analogous to Algorithm 6.1.1a (from the solution to Exercise 6.1.10), shift circularly the arc sequence of trail  $T$  so that  $e^*$  appears first:

$$T = \langle e^*, m, h, g, i, j, k, f, a, d, e, s, b \rangle$$

Then delete arc  $e^*$  from trail  $T$  to obtain the following open directed eulerian trail from  $x$  to  $y$ .

$$T = \langle m, h, g, i, j, k, f, a, d, e, s, b \rangle$$

**6.1.21:** If there were no such edge, then the input graph would not be connected.

**6.1.25:** In graph  $G$ , each endpoint of each edge is also an endpoint of an odd number of other edges.

**6.1.27:** False. For instance, the line graph of the non-eulerian star graph  $K_{1,3}$  is the complete graph  $K_3$ , which is eulerian.

## 6.2 DeBruijn Sequences and Postman Problems

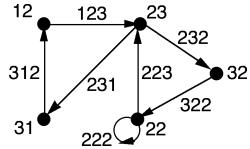
**6.2.1:** A different  $(2, 4)$ -deBruijn sequence is obtained from the eulerian tour specified by the following sequence of vertices and arcs.

$$\begin{aligned} 000 &\xrightarrow{0} 001 \xrightarrow{0} 010 \xrightarrow{0} 101 \xrightarrow{1} 010 \xrightarrow{0} 100 \xrightarrow{1} 001 \xrightarrow{0} 011 \xrightarrow{0} 111 \\ &\qquad\qquad\qquad \xrightarrow{1} 111 \xrightarrow{1} 110 \xrightarrow{1} 101 \xrightarrow{1} 011 \xrightarrow{0} 110 \xrightarrow{1} 100 \xrightarrow{1} 000 \xrightarrow{0} 000 \end{aligned}$$

The sequence of arc labels is the desired  $(2, 4)$ -deBruijn sequence: 0001010011110110.

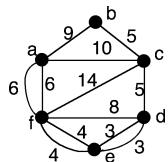
**6.2.5:** To obtain the line graph  $L(G)$  of a digraph  $G$ , each arc in  $G$  is a vertex in  $L(G)$ , and there is an arc in  $L(G)$  directed from vertex  $a$  to vertex  $b$  if, in digraph  $G$ ,  $head(a) = tail(b)$ . Each vertex of the line graph  $L(D_{2,n})$  is labeled with the full label on the corresponding arc in deBruijn digraph  $D_{2,n}$ . In digraph  $D_{2,n}$ , the arc directed from vertex  $b_1 b_2 \cdots b_{n-1}$  to vertex  $b_2 \cdots b_{n-1} b_n$  is labeled  $b_1 b_2 \cdots b_n$ , and the arc directed from vertex  $b_2 \cdots b_{n-1} b_n$  to some vertex  $b_3 \cdots b_{n-1} b_n c$  is labeled  $b_2 \cdots b_{n-1} b_n c$ . Thus, in the line graph  $L(D_{2,n})$ , there is an arc directed from vertex  $b_1 b_2 \cdots b_n$  to vertex  $b_2 \cdots b_{n-1} b_n c$ . Show that if this arc is labeled  $b_1 b_2 \cdots b_{n-1} b_n c$ , then the resulting digraph is  $D_{2,n+1}$ .

**6.2.8:** Each of the nine vertices of  $D_{3,3}$  is labeled with a different sequence of length 2 of characters drawn from  $\{0, 1, 2\}$ . The arc that is directed from vertex  $b_1 b_2$  to vertex  $b_2 b_3$  is labeled  $b_1 b_2 b_3$ . The figure shown below shows some of the vertices and arcs of  $D_{3,3}$ :



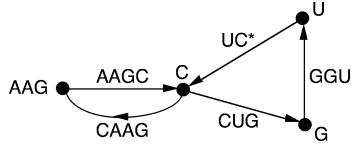
**6.2.14:** The set  $S$  of odd-degree vertices is  $S = \{a, d\}$ . The shortest  $a$ - $d$  path is  $P = \langle a, f, e, d \rangle$ , which has length 13.

The complete graph  $K$  on the odd-degree vertices is simply the vertices  $a$  and  $d$  and the edge between them has weight 13. This edge is itself the minimum-weight perfect matching, so the edges on the corresponding path  $P$  are duplicated, and the following graph is the eulerian graph shown below. Apply Algorithm 6.1.1 to obtain an eulerian tour of this graph, which will correspond to an optimal postman tour of length 77 for the original graph.

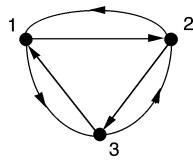


**6.2.16:** See Corollary 1.1.3.

**6.2.22:** 1. The only abnormal fragment is UC, and hence, the chain must end with it. 2. The only irreducible fragment that does not appear elsewhere as an internal subfragment is C, and hence, it must start the chain. 3. The normal fragments with at least two subfragments are: CUG; CAAG; AAGC; GGU. Thus, the associated digraph is as shown below. Only one eulerian tour in this digraph ends with the arc labeled UC. The corresponding RNA chain is CAAGCUGGGUC.



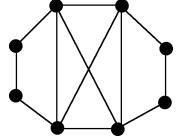
**6.2.27:** The corresponding digraph (without the self-loops), shown below, has several different eulerian tours. Observe that the sequence can begin with any of the three numbers, but it must end with the same number. Each of the eulerian tours corresponds to a different sequence (except for the  $f_{ii}$ 's). For instance, two of the possible sequences are 1232131 and 1312321. The  $f_{ii}$ 's are now handled by inserting the appropriate number of consecutive occurrences of the same number. Thus, some of the different ways to fill out the first sequence above are 1122233332131, 1122333221331, 1232221133331, and 1233222133311.



### 6.3 Hamiltonian Paths and Cycles

**6.3.2:** For all  $m$  and  $n$  such that  $m \geq 2$ ,  $n \geq 2$ , and  $m + n$  is even.

**6.3.6:**



**6.3.11:**  $\langle a, j, i, k, f, c, d, h, e, b, g, a \rangle$

**6.3.13:** The graph is not hamiltonian, because vertices  $c$  and  $w$  have degree 2, which forces the 4-cycle  $\langle c, f, w, b, c \rangle$ .

**6.3.18:**  $\langle a, e, f, j, i, m, n, o, p, l, k, g, h, d, c, b, a \rangle$

**6.3.21:** a. Hamiltonian graphs.    b. Cycle graphs.

**6.3.24:** The cycle graph on five vertices is a counterexample.

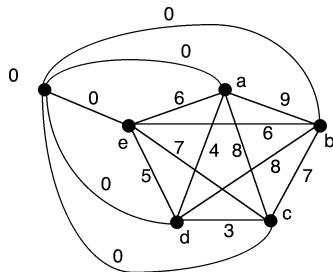
### 6.4 Gray Codes and Traveling Salesman Problems

**6.4.1:** First delete the last 000 term of the Gray code given in Example 6.4.1. Then put the reverse of this truncated sequence to the right, and append a 0 to the right of each of the first eight terms and a 1 to the right of each of the second eight terms. The resulting sequence, shown below, is a Gray code of order 4.

$$\begin{aligned} &\langle 0000, 1000, 1100, 0100, 0110, 1110, 1010, 0010, \\ &\quad 0011, 1011, 1111, 0111, 0101, 1101, 1001, 0001 \rangle \end{aligned}$$

- 6.4.3:** nearest neighbor output:  $\langle a, d, c, b, e, a \rangle$  with weight 26  
 double the tree output:  $\langle a, d, c, e, b, a \rangle$  with weight 29

**6.4.7:** Transformation 1 can be applied to undirected graphs. The graph  $G^*$  that results is shown below. The nearest neighbor algorithm with ties resolved alphabetically results in the cycle  $\langle a, 0, b, e, d, c, a \rangle$ . The open hamiltonian path that results from deleting vertex 0 is  $\langle a, b, e, d, c \rangle$ , with weight 23. An optimal hamiltonian path that starts at  $a$  is  $\langle a, d, c, b, e \rangle$  with weight 20.



- 6.4.12:** In order to apply Transformation 2, the given graph must be transformed into a digraph by replacing each edge with a pair of oppositely directed arcs that have weight equal to the weight of that edge.

**6.4.16:** The modified version of the nearest neighbor (Algorithm 6.4.1) would be to take the shortest path to an unvisited vertex, but the internal vertices of this path are allowed to have been visited. For the given graph, the iterations for the modified version are identical to those obtained in Exercise 6.4.3. Thus, the minimum-weight walk is no better than the minimum-weight hamiltonian cycle.

However, if both edge-weights that are 7 are changed to 9, then the modified version of nearest neighbor will have a different iteration 3. In particular,

- iteration 1:  $\langle a, d \rangle$
- iteration 2:  $\langle d, c \rangle$
- iteration 3:  $\langle c, d, e \rangle$
- iteration 4:  $\langle e, b \rangle$
- iteration 5:  $\langle b, a \rangle$

This would result in the closed walk  $\langle a, d, c, d, e, b, a \rangle$ .

## Chapter 7 Planarity and Kuratowski's Theorem

### 7.1 Planar Drawings and Some Basic Surfaces

- 7.1.1:** Since  $[(\frac{1}{2}, \frac{1}{2}, 0) - (0, 1, 0)] = (\frac{1}{2}, \frac{-1}{2}, 0)$ , the locus of the line through the points  $(\frac{1}{2}, \frac{1}{2}, 0)$  and  $(0, 1, 0)$  is
- $$\{(0, 1, 0) + c(\frac{1}{2}, \frac{-1}{2}, 0) \mid -\infty \leq c \leq \infty\}$$

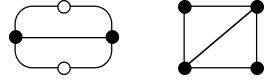
This line meets the plane  $y = 0$  when  $c = 2$ ; thus, the point of intersection is  $(1, 0, 0)$ .

- 7.1.5:** The locus of the line through the points  $(\frac{\sqrt{3}}{3}, 0, 0)$  and  $(0, 1, 0)$  is
- $$\{(0, 1, 0) + c(\frac{\sqrt{3}}{3}, -1, 0) \mid -\infty \leq c \leq \infty\}$$

This line meets the sphere  $x^2 + (y - \frac{1}{2})^2 + z^2 = \frac{1}{4}$  when  $c = \frac{3}{4}$ ; thus, the point of intersection is  $(\frac{\sqrt{3}}{4}, \frac{1}{4}, 0)$ .

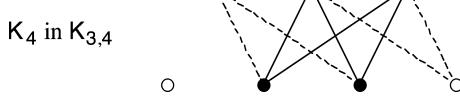
## 7.2 Subdivision and Homeomorphism

**7.2.1:** Subdividing two of the edges of  $D_3$  as shown yields a graph that is isomorphic to  $K_4 - K_2$ .



**7.2.5:** Place 2 subdivision vertices on each of the  $n$  self-loops, for a total of  $2n$  subdivision vertices.

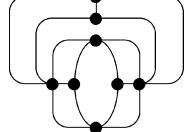
**7.2.7:**



**7.2.11:** No matter how the four vertices of a homeomorphic copy of  $K_4$  could be chosen in  $K_{2,3}$ , at least two vertex pairs would be non-adjacent. It would be necessary to join each such pair with a path in the homeomorphic copy of  $K_4$ , and such paths would have to be mutually internally disjoint. Thus, two or more internally disjoint paths in  $K_{2,3}$  would have to go through the one remaining vertex, an impossibility.

## 7.3 Extending Planar Drawings

**7.3.1:**



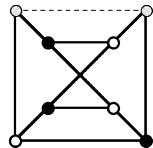
**7.3.5:** Nonplanar. After pasting, the hub of  $W_6$  is adjacent to all three of the “pasted” vertices. Also, the two nonpasted vertices from  $W_4$  are adjacent to all three pasted vertices. Thus, the amalgamated graph contains  $K_{3,3}$ .

**7.3.9:** One appendage has vertex-set  $\{u, v, w\}$  and edge-set  $\{a, b\}$ . The other has vertex-set  $\{u, x, w\}$  and edge-set  $\{d, e\}$ .

**7.3.13:** The contact points of the appendage with vertex-set  $\{u, v, w\}$  and edge-set  $\{a, b\}$  are  $u$  and  $w$ . The contact points of the appendage with vertex-set  $\{u, x, w\}$  and edge-set  $\{d, e\}$  are also  $u$  and  $w$ . These two appendages do not overlap.

## 7.4 Kuratowski's Theorem

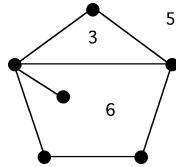
**7.4.2:** In the figure below, the partite sets of a  $K_{3,3}$  are represented by three white vertices and three black vertices. The gray vertices are interior vertices on paths joining a pair.



**7.4.7:**  $K_9 - K_{4,5}$  is isomorphic to  $K_4 \cup K_5$ .

## 7.5 Algebraic Tests for Planarity

7.5.2:

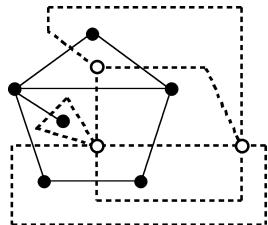


(c) Sum of face-sizes =  $3 + 6 + 5 = 14 = 2 \cdot 7 = 2|E|$ .

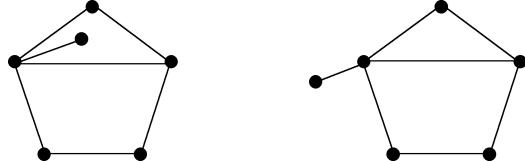
7.5.6: (a)  $\text{girth} = 3$ . (b)  $|F| = 3$ .  $\text{girth} \cdot |F| = 3 \cdot 3 \leq 2 \cdot 7 = 2 \cdot |E|$ .

7.5.10: (a)  $|V| = 6$ .  $|E| = 7$ .  $|F| = 3$ . (b)  $|V| - |E| + |F| = 6 - 7 + 3 = 2$ .

7.5.14:



7.5.18:



Since the face-size combinations of these two new imbeddings and the original imbedding are mutually distinct, it follows that the dual graphs are mutually distinct.

7.5.22: The graph  $K_4 - K_2$  has 4 vertices. By the Euler polyhedral formula, a planar imbedding of it has 3 faces. Thus, the dual graph has 3 vertices. It follows that the dual graph cannot be isomorphic to  $K_4 - K_2$ .

7.5.29: Each of the four vertices lying on one of the two disjoint (deleted) copies of  $C_4$  is joined to each of the four vertices in the other copy, so the graph contains  $K_{4,4}$ .

7.5.33: Violates the inequality  $|E| \leq 3|V| - 6$ .

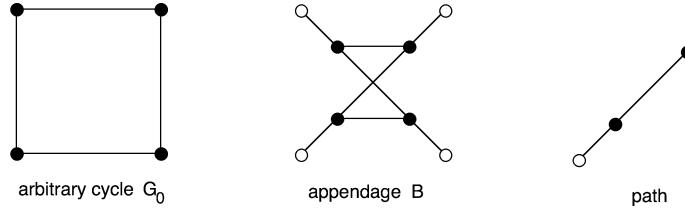
7.5.42: Nonplanar because it contains  $K_{3,3}$ . However, since  $|E| = 15$  and  $|V| = 7$ , it satisfies the inequality  $|E| \leq 3|V| - 6$ .

## 7.6 Planarity Algorithm

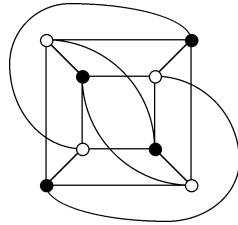
7.6.2: The blocked appendage in Figure Sol7.6.2 on the facing page shows that the input graph is nonplanar. It is blocked since no region contains all four of its contact points.

## 7.7 Crossing Numbers and Thickness

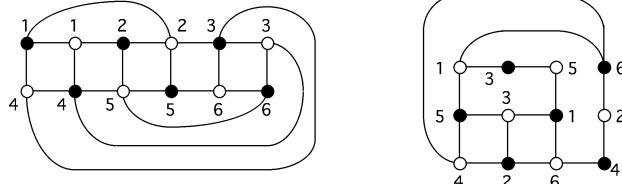
7.7.1: Apply Theorem 7.7.3.

**Figure Sol7.6.2**

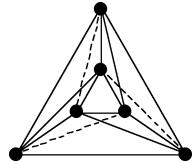
**7.7.5:** By Theorem 7.7.5,  $\nu(K_{4,4}) \geq 16 - 2 \cdot 8 + 4 = 4$ . This figure establishes that  $\nu(K_{4,4}) \leq 4$ .



**7.7.10:** By Theorem 7.7.9,  $\theta(K_{6,6}) \geq 2$ . This figure establishes that  $\theta(K_{6,6}) \leq 2$ .



**7.7.12:**



## Chapter 8 Drawing Graphs and Maps

### 8.1 The Topology of Low Dimensions

**8.1.1:**  $f(x) = \frac{2x-1}{1-|2x-1|}$

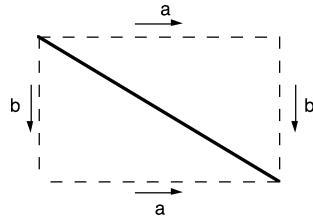
**8.1.3:** Only one.

## 8.2 Higher-Order Surfaces

**8.2.1:** Although it is finite and boundaryless, it does not contain the endpoints of the open arc  $\{(x, 0) \mid 0 < x < 1\}$ .

**8.2.6:** It is a boundaryless surface.

**8.2.12:** The bold closed circuit in the figure traverses the torus once in the meridian direction (*b*) and once in the longitude direction (*a*). It does not separate the torus, because any point in the “upper” triangle can reach any point in the “lower” triangle by a path through meridian (*b*) or by a path through longitude (*a*).



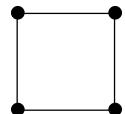
## 8.3 Mathematical Model for Drawing Graphs

**8.3.1:**



## 8.4 Regular Maps on a Sphere

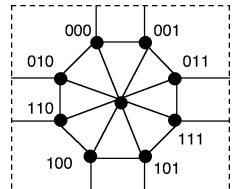
**8.4.2:**



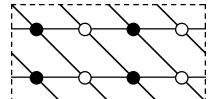
**8.4.5:** The dipoles  $\{D_n \mid n \geq 4\}$ .

## 8.5 Imbeddings on Higher-Order Surfaces

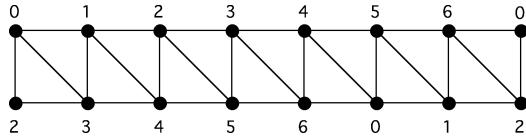
**8.5.1:**



**8.5.4:**

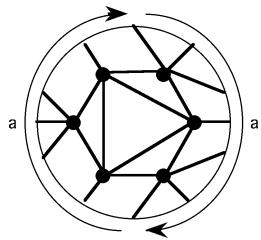


**8.5.5:** Paste left side directly to the right. Then paste the resulting top cycle to the bottom cycle with a  $2/7$  twist.

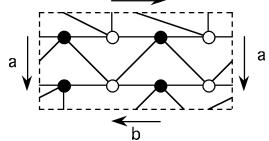


**8.5.6:** Extend the solution to Exercise 8.5.5 by one more square-with-diagonal, so that it has eight squares instead of seven.

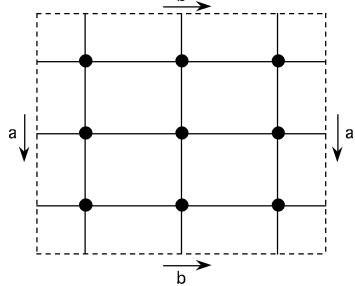
**8.5.11:** The flat polygon representation of  $N_1$  has 2 sides, both labeled  $a$  in the figure, and pasted as shown. This implies that a partial edge drawn through a side of the circular boundary of the flat digon joins the partial edge at the antipodal location  $180^\circ$  around the circle.



**8.5.16:**



**8.5.21:**

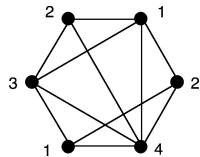


**8.5.27:** Apply Theorem 8.5.6.

## Chapter 9 Graph Colorings

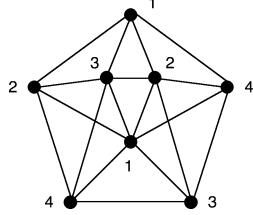
### 9.1 Vertex-Colorings

**9.1.3:**



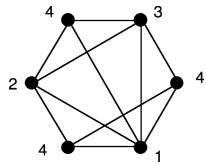
Fewer than 4 colors is impossible because of the  $K_4$ .

9.1.7:



Fewer than 4 colors is impossible because of the wheel  $W_6$ .

9.1.10:



9.1.14: The coloring given above for Exercise 9.1.7 is obtainable by application of the largest-degree-first heuristic.

9.1.19: In each of the two parts of the bipartition, all vertices have the same degree. In the part with larger degree, all vertices are assigned color 1. In the other, all are assigned color 2.

9.1.22: Vertices 1 and 2 both get color 1. This ultimately forces the use of a third vertex color.

9.1.27: No matter what edge is deleted, there remains a 3-cycle.

9.1.31: The independence number is 2. The four vertices of the  $K_4$  are mutually adjacent, and the two other vertices are adjacent.

9.1.35: Finding two non-adjacent vertices is easy, which establishes a lower bound of 2 for the independence number. Since the minimum degree is 4, and since the graph has only eight vertices, it follows that any subset of two non-adjacent vertices has an adjacency to every vertex not in the subset.

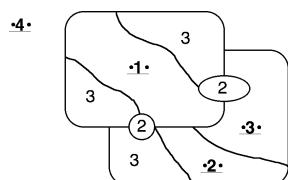
9.1.37: The domination number is 1, since the 5-valent vertex is adjacent to all the others.

9.1.41: Any pair of two non-adjacent vertices qualifies as a dominating set.

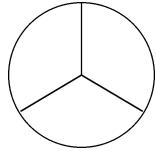
9.1.45: A subgraph of a bipartite graph is 1-chromatic if it contains no edges, in which case its clique number is 1. Otherwise, it is bipartite and 2-chromatic, and its clique number is 2.

## 9.2 Map-Colorings

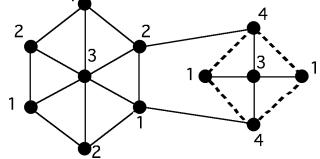
9.2.2: The four regions whose colors are underlined are mutually adjacent.



**9.2.5:** Yes. Here is a minimum example.

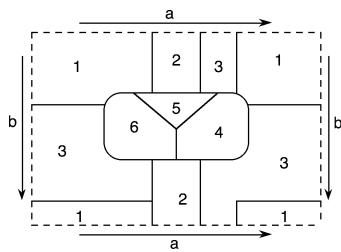


**9.2.8:**



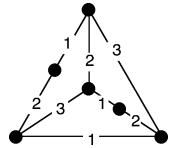
**9.2.14:** The chromatic number of Africa is 4. Since Africa is planar, 4 is an upper bound. There are several even-order wheels — for instance, with Niger or Mali as the hub — which implies that 4 is also a lower bound.

**9.2.16:** Six. There are six mutually adjacent regions.

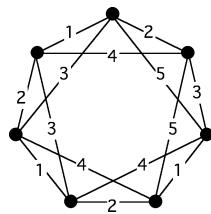


### 9.3 Edge-Colorings

**9.3.2:** Since the maximum degree is 3, this 3-edge-coloring must be minimum.



**9.3.11:** There are 14 edges. Any collection of more than three edges has a common vertex. Thus, no single color class may have more than three edges. Thus, the edge-chromatic number must be 5.



**9.3.16:** Alternate the colors.

**9.3.21:** Give the new edge a new color.

**9.3.25:** Use any edge in an even cycle  $C_{2n}$ .

**9.3.28:** Deleting the cut-edge  $e$  from graph  $G$  must yield two components each with an odd number of vertices (since smoothing the two resulting 2-valent vertices would yield a 3-regular graph with two components, each necessarily with an even number of vertices). Suppose now, that  $G$  were in class one. Then deleting the color class of the cut-edge would yield a 2-colorable 2-regular graph, i.e., the disjoint union of some even cycles. However, one sub-union of these even cycles must span the one components of  $G - e$  and the complementary sub-union must span the other component, which contradicts the fact that these components each have an odd number of vertices.

## 9.4 Factorization

**9.4.3:** The union of a  $k$ -factor of  $G$  and a  $k$ -factor of  $H$  is a  $k$ -factor of  $G + H$ .

**9.4.8:** The edge-complement of the graph of Exercise 9.4.1.

**9.4.11:**  $K_2 \times K_2$ .

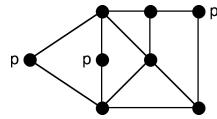
# Chapter 10 Measurement and Mappings

## 10.1 Distance in Graphs

**10.1.1:**  $\text{circum}(K_{3,7}) = 6$

**10.1.7:** For  $n$  odd, the circumference equals  $n$ , and for  $n$  even, it equals  $n/2$ .

**10.1.9:** The three labeled vertices form the periphery.



**10.1.12:** For  $n$  odd, the minimum total distance equals  $\frac{n^2-1}{4}$ , and the median subgraph is the middle vertex of the path. For  $n$  even, the median subgraph is the middle edge, and the minimum total distance (achieved by each endpoint of the middle edge) equals  $\frac{n^2}{4}$ .

**10.1.21:** Start with a cycle and attach a path of appropriate length to one of the cycle vertices.

**10.1.24:** Suppose that  $\text{ecc}(x) \leq \text{ecc}(y)$ , and let  $z$  be a vertex such that  $\text{ecc}(y) = d(y, z)$ . Use the triangle inequality to show that  $\text{ecc}(y) \leq \text{ecc}(x) + 1$ .

## 10.2 Domination in Graphs

**10.2.2:**  $\text{dom}(P_n) = \lceil \frac{n}{3} \rceil$ , since each vertex can dominate at most three vertices.

**10.2.6:** The Petersen graph has domination number 3.

**10.2.10:**  $i\text{-dom}(P_n) = \lceil \frac{n}{3} \rceil$ ;  $c\text{-dom}(P_n) = n - 2$ .

**10.2.17:** a.  $i\text{-dom}(\text{circ}(5 : 1, 2)) = c\text{-dom}(\text{circ}(5 : 1, 2)) = 1$ ;  
b.  $i\text{-dom}(\text{circ}(6 : 1, 2)) = c\text{-dom}(\text{circ}(6 : 1, 2)) = 2$ ;  
c.  $i\text{-dom}(\text{circ}(8 : 1, 2)) = 2$  and  $c\text{-dom}(\text{circ}(8 : 1, 2)) = 3$ .

**10.2.19:**  $d_2\text{-dom}(P_n) = \lceil \frac{n}{5} \rceil$ .

### 10.3 Bandwidth

**10.3.1:**  $bw(CL_3) = 3$

**10.3.5:** Since 1 and  $n$  must be assigned to univalent vertices (otherwise, the numbering would have bandwidth  $n - 1$ ), the smallest bandwidth numbering occurs when the hub is assigned a value midway between 1 and  $n$ .

**10.3.8:** Since neither 1 nor  $2n$  can be assigned to the hub (otherwise, the numbering would have bandwidth  $2n - 1$ ), the bandwidth is at least  $n$ . Now find a numbering that has bandwidth  $n$ .

### 10.4 Intersection Graphs

**10.4.2:** Let each of the  $n$  intervals be the same interval.

**10.4.5:**  $int(P_3) = 2$

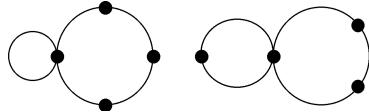
**10.4.8:** The edge-clique-cover number equals 4, and hence, by Theorem 10.4.10, the intersection number equals 4.

**10.4.12:** Let  $\mathcal{F} = \{I_1, I_2, \dots, I_n\}$  be a family of intervals corresponding to an interval graph  $G$ . Let  $\mu(S) = \text{length}(S)$  for any real interval  $S$ , and let  $\phi$  be the constant function  $\epsilon$ , where  $\epsilon = \min\{\mu(I_j \cap I_k) \mid 1 \leq j < k \leq n \text{ and } I_j \cap I_k \neq \emptyset\}$ .

### 10.5 Linear Graph Mappings

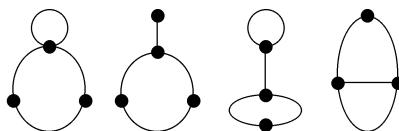
**10.5.2:**

self - amalgamations  
of  $C_5$  on two vertices

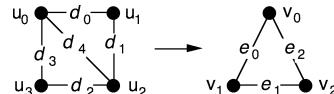


**10.5.10:**

self - amalgamations  
of  $C_5$  on two edges



**10.5.21:**



$u_0 \quad u_1 \quad u_2 \quad u_3 \quad d_0 \quad d_1 \quad d_2 \quad d_3 \quad d_4$

$v_0 \quad v_1 \quad v_2 \quad v_1 \quad e_0 \quad e_1 \quad e_0 \quad e_2$

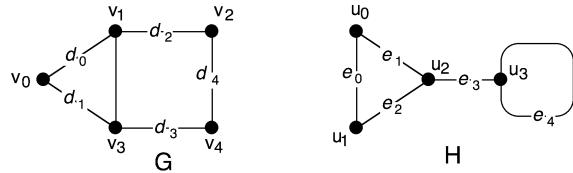
**10.5.26:** Wrap a 5-path around the pentagon and the complementary 5-path around the pentagram.

**10.5.30:** The vertex set of the domain graph  $C_3$  has fewer vertices than the vertex graph of the codomain graph  $P_4$ .

**10.5.37:** See Exercise 10.5.2.

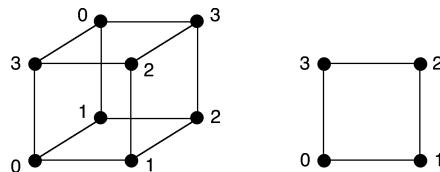
## 10.6 Modeling Network Emulation

**10.6.1:** Load 2 and congestion 2.



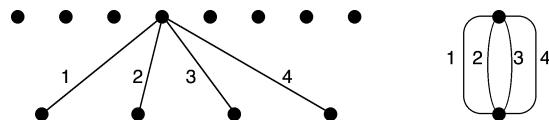
$v_0$	$v_1$	$v_2$	$v_3$	$v_4$	$d_0$	$d_1$	$d_2$	$d_3$	$d_4$	$d_5$
$u_0$	$u_2$	$u_3$	$u_1$	$u_2$	$e_1$	$e_0$	$e_1$	$e_3$	$e_3$	$e_2$

**10.6.5:** The vertex labels in the following figure indicate a mapping with load 2 and congestion 3.



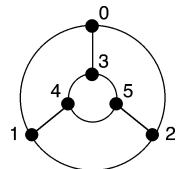
**10.6.9:** Obtain load 8 by mapping 4 domain vertices to one codomain vertex and mapping 8 domain vertices to the other codomain vertex.

**10.6.13:** This partial figure gives the idea for a linear mapping with congestion 8.



**10.6.19:** Minimum dilation is achieved by mapping the  $n$ -valent vertex to the “middle” of the  $(n+1)$ -path.

**10.6.23:** Since some vertex of  $CL_3$  must be mapped to an extreme vertex of the path, and since that vertex is 3-valent, there is a lower bound of 3 for the dilation of a vertex-bijective semilinear mapping. The labeling in the following figure realizes this lower bound.



## Chapter 11 Analytic Graph Theory

### 11.1 Ramsey Graph Theory

**11.1.1:** By Theorem 11.1.5b,  $r(3, 6) \leq r(2, 6) + r(3, 5) - 1 = 6 + 14 - 1 = 19$ . By Corollary 11.1.6,

$$r(3, 6) \leq \binom{3+6-2}{2}$$

**11.1.6:**

$$r(3, t) \leq \binom{1+t}{2} = \frac{t^2 + t}{2}$$

## 11.2 Extremal Graph Theory

**11.2.1:**  $K_5 - K_2$ .

**11.2.5:**  $ex(4, 2K_2) = 3$ . The extremal graphs are  $K_{1,3}$  and  $K_3 \cup K_1$ .

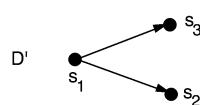
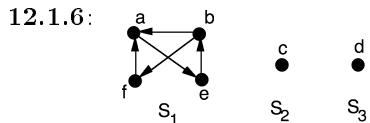
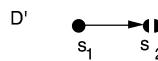
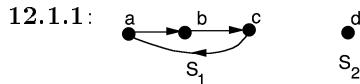
## 11.3 Random Graphs

**11.3.5:** There are  $\binom{\binom{5}{2}}{4} = 210$  graphs in  $\mathcal{G}(5, 4)$ . There are  $5 \cdot \binom{4}{3} = 20$  choices of  $K_{1,3}$  in a labeled  $K_5$  and 7 possible choice of a fourth edge, which implies  $20 \cdot 7 = 140$  total instances of a  $K_{1,3}$  among all the graphs of  $\mathcal{G}(5, 4)$ . Thus, the expected number in a random graph is  $\frac{140}{210} = \frac{2}{3}$ .

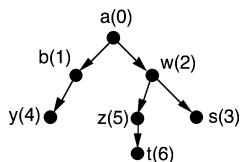
**11.3.9:** Almost every graph contains  $K_5$ . (See Exercise 11.3.12.)

# Chapter 12 Special Digraph Models

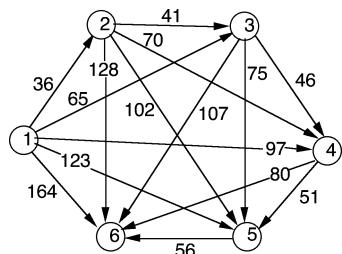
## 12.1 Directed Paths and Mutual Reachability

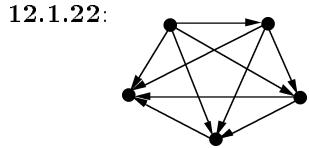
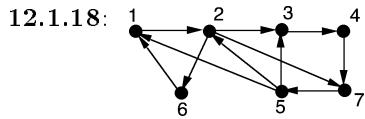


**12.1.11:** This digraph is strongly connected. The output tree is shown below.



**12.1.13:** The full model for Application 12.1.1 is shown below. There are two optimal policies for this set of data. Either sell the first car after one year and keep the second for the next four years, or keep the first car for all five years.





12.1.24: Dags.

12.1.28: The stronger statement that the resulting digraph has at least  $k + 1$  strong components can be proved by induction on  $k$ . The base for the induction is Theorem 12.1.4.

12.1.30: The 2-step transition matrix is shown below. As an illustration of how these 2-step transition probabilities can be obtained directly from the Markov diagram in Figure 12.1.7, consider  $p_{45}^{[2]}$ . There are two different paths of length 2 from vertex 4 to vertex 5, namely  $4 \rightarrow 3 \rightarrow 5$  and  $4 \rightarrow 5 \rightarrow 5$ . Multiplying the arc probabilities for each of these two paths and then taking the sum, we obtain  $\frac{3}{4} \cdot \frac{1}{4} + \frac{1}{4} \cdot 1 = \frac{7}{16}$ , which agrees with the 4,5 entry of the 2-step transition matrix.

$$\begin{array}{cccccc} & 0 & 1 & 2 & 3 & 4 & \geq 5 \\ 0 & \left( \begin{array}{cccccc} 1 & 0 & 0 & 0 & 0 & 0 \\ .75 & 0 & 0 & .1875 & 0 & .0625 \\ .5625 & 0 & 0 & 0 & .1875 & .0625 \\ 0 & .5625 & 0 & 0 & 0 & .4375 \\ 0 & 0 & .5625 & 0 & 0 & .4375 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{array} \right) \\ 1 & & & & & & \\ 2 & & & & & & \\ 3 & & & & & & \\ 4 & & & & & & \\ \geq 5 & & & & & & \end{array}$$

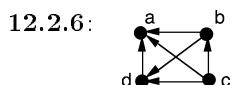
12.1.34: a. All of the states are transient except state 2, which is absorbing.

b. There are four different paths of length 4 from vertex 3 to vertex 4, namely,  $\langle 3, 1, 3, 4, 4 \rangle$ ;  $\langle 3, 1, 5, 4, 4 \rangle$ ;  $\langle 3, 4, 1, 5, 4 \rangle$ ; and  $\langle 3, 4, 4, 4, 4 \rangle$ . By multiplying the arc probabilities along each of these four paths and then taking the sum, we obtain the result  $p_{34}^{[4]} = \frac{1}{64} + \frac{1}{32} + \frac{1}{32} + \frac{1}{32} = \frac{7}{64}$ .

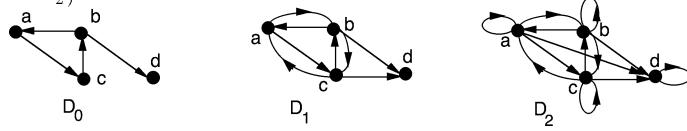
12.1.37: If all of the states were transient, then for each vertex in the Markov digraph, there would exist a vertex that is reachable from it. Show that this would imply the existence of a directed cycle in the digraph, which would then contradict the assumption that all states were transient.

## 12.2 Digraphs as Models for Relations

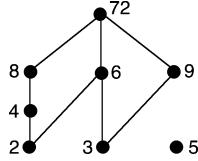
12.2.1: Since the given digraph is strongly connected, the transitive closure has both arcs between each pair of vertices and has a self-loop at each vertex.



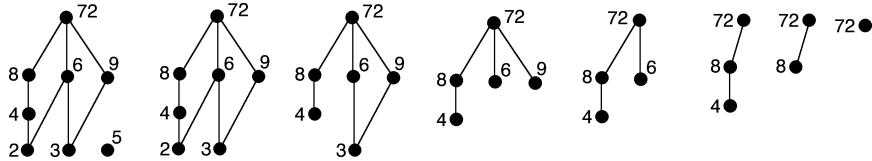
- 12.2.10:** The transitive closure is obtained at the end of the second iteration (i.e.,  $D_4 = D_3 = D_2$ ).



- 12.2.14:** The minimal elements in the Hasse diagram shown below are 2, 3, and 5.

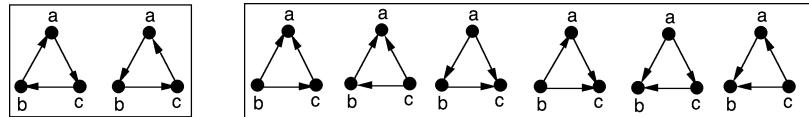


- 12.2.17:** The sequence  $\langle 5, 2, 3, 9, 6, 4, 8, 72 \rangle$  represents a compatible total order of the given poset and is the result of the following sequence of iterations of the topological sort.

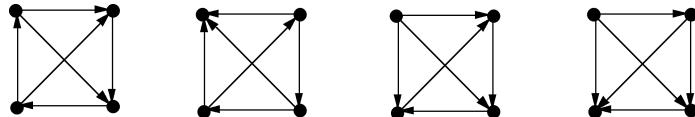


### 12.3 Tournaments

- 12.3.1:** Since each of the three edges of  $K_3$  can be oriented in one of two ways, there are eight different 3-vertex tournaments. They are shown below, grouped into isomorphism classes.



- 12.3.3:** Each 4-vertex tournament is isomorphic to one of the four tournaments shown in the figure below. The first contains a 4-cycle, the second and third each contain exactly one 3-cycle, and the fourth is acyclic.



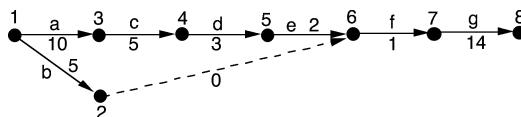
To see that there is only one tournament, up to isomorphism, that contains a 4-cycle, first arrange the vertices so that the 4-cycle forms a square, as in the figure. Then observe that no matter how the diagonal edges are directed, a rotation of  $90^\circ$ ,  $180^\circ$ , or  $270^\circ$  of the tournament will match the one in the figure. For a tournament that contains exactly one cycle, it is not hard to show that the non-cycle vertex must have either 0 indegree or 0 outdegree. Similarly, one can argue that if a 4-vertex tournament has more than one cycle, then it must contain a 4-cycle, putting it in category 1. Finally, to see that there is only one isomorphism type for the acyclic 4-vertex tournaments, start with the unique hamiltonian path (whose existence is guaranteed by Corollary 12.3.3) and argue that there is only one way to direct the three arcs that are not on this path without creating a cycle.

**12.3.7:** If the tournament were strongly connected, then any two vertices would be mutually reachable, which would imply the existence of a directed cycle.

## 12.4 Project Scheduling and Critical Paths

**12.4.1:** In the AOA network for the problem shown below at the right the ET and LT values for each event are as follows:

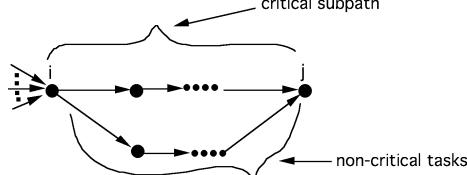
Event	$ET(i)$	$LT(i)$
1	0	0
2	5	20
3	10	10
4	15	15
5	18	18
6	20	20
7	21	21
8	35	35



The earliest possible completion time for the project is 35 days, and task b is the only non-critical one. Its total float is  $LT(2) - ET(1) - wt(b) = 20 - 0 - 5 = 15$  days.

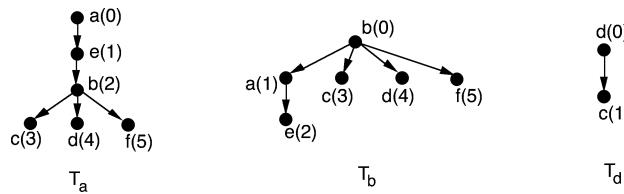
**12.4.6:** Suppose that every vertex has nonzero indegree, and consider any vertex  $v_1$ . There is an arc directed from some vertex  $v_2$  to vertex  $v_1$ . Similarly, there is an arc directed from some vertex  $v_3$  to vertex  $v_2$ . Since there are finitely many vertices, this process will eventually produce some  $v_k = v_l$  for  $k \neq l$ , which would contradict the acyclic property. A similar argument can be used to show that there must be a vertex whose outdegree is 0.

**12.4.9:** Suppose that the critical path were not the longest path. Then there would be a subpath  $P$  of the critical path from event  $i$  to event  $j$  such that there is a longer  $i-j$  path consisting of non-critical tasks only, as illustrated in the figure below. Work backwards from event  $j$  and use the definitions of  $TF$ ,  $ET$ , and  $LT$  to reach a contradiction.



## 12.5 Finding the Strong Components of a Digraph

**12.5.2:** The unlabeled vertex in the given digraph has been assigned the label  $f$ . The three dfs trees are shown in the figure below. In tree  $T_a$ ,  $ba$  and  $fe$  are back arcs, and  $dc$  is a cross arc. In tree  $T_b$ ,  $fe$  and  $dc$  are cross arcs, and  $eb$  is a back arc.



**12.5.6:** Referring to the dfs trees from the solution to Exercise 12.5.2, the cross arc  $dc$  in tree  $T_a$  is directed from *dfnumber* 4 to *dfnumber* 3. In tree  $T_b$ , cross arc  $fe$  is directed from *dfnumber* 5 to *dfnumber* 2, and cross arc  $dc$  is directed from *dfnumber* 4 to *dfnumber* 3.

**12.5.8:** There are no cross arcs.

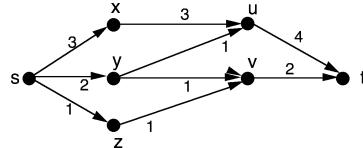
**12.5.10:** The unlabeled vertex in the given digraph has been assigned the label  $f$ . Starting at vertex  $a$ , the dfs tree produced by Algorithm 12.5.2 contains all of the vertices. The vertex-sets of the strong components are:  $\{a, b, e, f\}$ ;  $\{c\}$ ; and  $\{d\}$ .

**12.5.14:** Strongly connected digraphs.

## Chapter 13 Network Flows and Applications

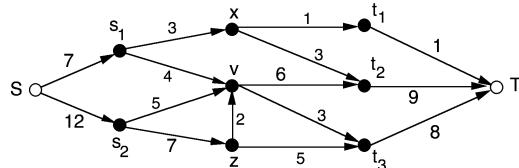
### 13.1 Flows and Cuts in Networks

**13.1.1:** The arc set  $\langle \{s, z\}, \{x, y, u, v, w, t\} \rangle$  is an  $s-t$  cut with capacity 6, and the flow shown in the figure below has value 6. Hence, by Corollary 13.1.7, both are optimal. The number on each arc in the figure indicates the flow across that arc.



**13.1.5:** There are eight different  $s-t$  cuts depending on the subset of  $\{x, z, v\}$  into which source  $s$  is placed. The two minimum ones, with capacity 14, are  $\langle \{s, x, z, v\}, \{t\} \rangle$  and  $\langle \{s, z\}, \{x, v, t\} \rangle$ .

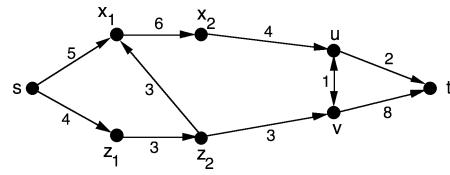
**13.1.9:** Add a “super source”  $S$  directed to each of the original sources and add a “super sink”  $T$  directed from each of the original sinks to obtain the network shown below. For each original sink  $s_i$ , assign to the arc from  $S$  to  $s_i$  a capacity equal to the total capacity of  $Out(s_i)$ . Similarly, for each original sink  $t_j$ , assign to the arc from  $t_j$  to  $T$  a capacity equal to the total capacity of  $In(t_j)$ .



A minimum  $S-T$  cut for this network is  $\langle \{S, s_1, s_2, v, z\}, \{x, t_1, t_2, t_3\} \rangle$  and has capacity 17. A flow  $f$  achieving this value is given by:  $f(Ss_1) = 7$ ;  $f(Ss_2) = 10$ ;  $f(s_1x) = 3$ ;  $f(s_1v) = 4$ ;  $f(s_2v) = 3$ ;  $f(s_2z) = 7$ ;  $f(zv) = 2$ ;  $f(zt_3) = 5$ ;  $f(vt_3) = 3$ ;  $f(vt_2) = 6$ ;  $f(xt_2) = 3$ ;  $f(xt_1) = f(t_1T) = 0$ ;  $f(t_2T) = 9$ ; and  $f(t_3T) = 8$ . Finally, dropping the arcs from  $S$  and the ones to  $T$  results in a maximum flow for the original network.

**13.1.13:** Split each such vertex  $v$  into two vertices  $v_1$  and  $v_2$ , draw an arc directed from  $v_1$  to  $v_2$ , and assign to it a capacity equal to the capacity of vertex  $v$ . Then replace each arc in the original network that was directed from a vertex  $x$  to vertex  $v$  with an arc directed from  $x$  to  $v_1$ , and replace each arc that was directed from vertex  $v$  to a

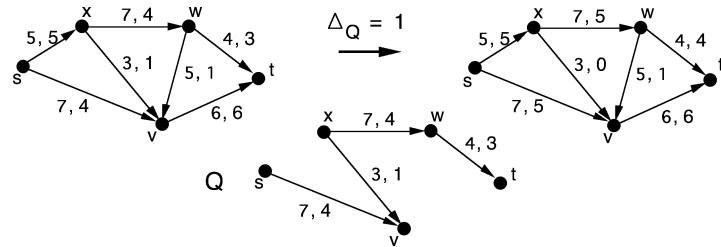
vertex  $y$  with an arc from  $v_2$  to  $y$ . The network that results when this transformation is applied to the given digraph is as follows.



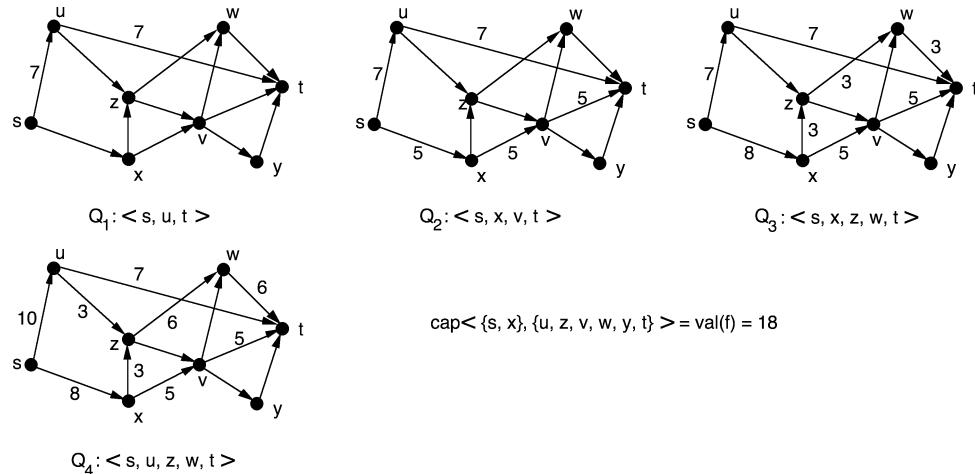
**13.1.16:** Use Proposition 13.1.3 and Corollary 13.1.7.

## 13.2 Solving the Maximum-Flow Problem

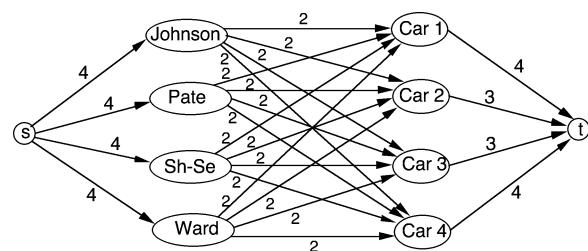
**13.2.1:**



**13.2.5:** A sequence of quasi-paths and flows is shown below.

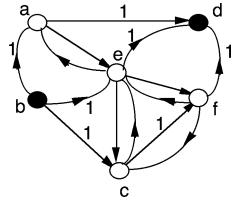


**13.2.8:** Solve the following maximum-flow problem.



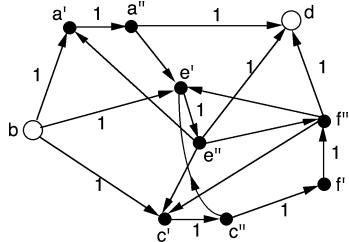
### 13.3 Flows and Connectivity

**13.3.1:** Each arc of the digraph  $\overset{\leftrightarrow}{G}$  below is assumed to have capacity 1. The arcs directed to source  $b$  and the arcs directed from sink  $d$  are not drawn, since they cannot be part of a directed path from  $b$  to  $d$ . A maximum flow from  $b$  to  $d$  of value 3 is shown in the figure and is achieved by assigning a flow of 1 to each arc of the three  $b$ - $d$  paths:  $\langle b, a, d \rangle$ ,  $\langle b, e, d \rangle$ ,  $\langle b, c, f, d \rangle$ , which are arc-disjoint. Thus, the local edge-connectivity between vertices  $b$  and  $d$  equals 3.



**13.3.5:** From the solution to Exercise 13.3.1, we know that  $\kappa_e(b, d) = 3$ . Similarly,  $\kappa_e(b, a) = 3$ , and by symmetry,  $\kappa_e(b, c) = \kappa_e(b, e) = 3$ . Thus, by Corollary 13.3.12,  $\kappa_e(G) = 3$ .

**13.3.9:** Each arc of the digraph  $N^{\overset{\leftrightarrow}{G}}$ , shown below, is assumed to have capacity 1. The arcs directed to source  $b$  and the arcs directed from sink  $d$  are not drawn since they cannot be part of a directed path from  $b$  to  $d$ . A maximum flow of value 3 is obtained by assigning a flow of 1 to each arc of the three arc-disjoint  $b$ - $d$  paths  $\langle b, a', a'', d \rangle$ ,  $\langle b, e', e'', d \rangle$ ,  $\langle b, c', c'', f', f'', d \rangle$ . Thus,  $\kappa_v(b, d) = 3$ . The three paths correspond to the paths  $\langle b, a, d \rangle$ ,  $\langle b, e, d \rangle$ ,  $\langle b, c, f, d \rangle$  in the original graph, which are internally disjoint.



**13.3.13:** It follows from the solution to Exercise 13.3.9 and symmetry that  $\kappa_v(b, f) = 3$ , which, by Lemma 5.3.5, implies that  $\kappa_v(G) = 3$ .

**13.3.18:** Since the initial arc of each of the  $r$  paths has flow 1, we have  $val(f) = \sum_{e \in O_{out}(s)} f(e) = r$ . Since  $f(e) \leq cap(e)$ , it remains to show that  $f$  satisfies conservation of flow. Let  $v$  be any vertex in network  $N$  other than  $s$  or  $t$ . On each of the paths that contain  $v$ , the arc directed to  $v$  and the arc from  $v$  both have flow 1, and hence, the conservation of flow property is satisfied.

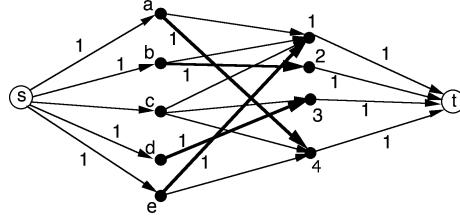
**13.3.21:** In digraph  $N^D$ , there is only one arc directed from vertex  $v'$  to  $v''$ . This implies that any  $s$ - $t$  directed path in  $N^D$  that uses vertex  $v'$  must have vertex  $v''$  as the next vertex.

**13.3.25:** If all the arc capacities are integer, then convert the given network  $N$  into a 0-1 network  $\hat{N}$  by replacing each arc  $e$  in  $N$  by  $k$  arcs of unit capacity, where  $k = cap(e)$ . Then use the digraph version of Theorem 5.3.10. If all the arc capacities are rational numbers, then multiply each arc by the least common multiple of the denominators (see

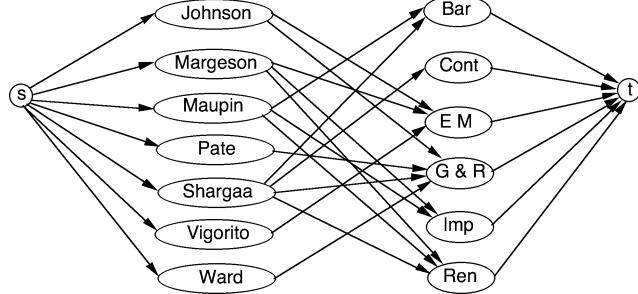
Exercise 13.1.12). If some of the capacities are irrational numbers, then the maximum flow can be approximated to any desired accuracy by replacing each irrational by a suitably near rational so that the difference between the value of the flow and the capacity of the cut is sufficiently close to zero.

### 13.4 Matchings, Transversals, and Vertex Covers

**13.4.2:** The maximum flow in the network  $\vec{G}_{st}$  is shown below. The set of edges in the given graph whose corresponding arcs have unit flow in  $\vec{G}_{st}$  is the maximum matching  $\{a4, b2, d3, e1\}$ .



**13.4.6:** Assume all arcs in the following network have unit capacity and determine whether the maximum flow is 6.



**13.4.10:** One of several violations of Hall's condition is the set of three subsets  $\{\{5\}, \{4, 5\}, \{4, 5\}\}$ , since the union contains only two elements.

**13.4.13:** The edge set given by  $\{(6, 1), (3, 4), (2, 9), (8, 10), (5, 7)\}$  is a maximum matching, and the vertex set  $\{1, 3, 2, 8, 5\}$  is a minimum vertex cover.

**13.4.17:** Let  $W$  be a subset of  $X$ . Since  $G$  is bipartite, the degree sum  $\sum_{w \in W} \deg(w)$  equals the number of edges having one endpoint in  $W$  and the other endpoint in  $N(W)$ . But this number is less than or equal to the total number of edges having an endpoint in  $N(W)$ . Thus,

$$|W| \cdot \delta_X \leq \sum_{w \in W} \deg(w) \leq \sum_{w \in N(W)} \deg(w) \leq N(W) \cdot \Delta_Y \leq N(W) \cdot \delta_X$$

which shows that Hall's condition is satisfied.

**13.4.19:** This is a corollary of the assertion in Exercise 13.4.17, but it can be proved directly with a similar argument. In particular, if  $W$  is a subset of  $X$ , then since  $G$  is bipartite,

$$|W| \cdot k = \sum_{w \in W} \deg(w) \leq \sum_{w \in N(W)} \deg(w) = N(W) \cdot k$$

**13.4.25:** Either construct a 0-1 matrix and apply König-Egerváry, or construct a bipartite graph and apply Theorem 13.4.3 or Theorem 13.4.9.

**13.4.27:** Let  $\hat{M}$  be any matching in graph  $G$ . Then, using weak duality (Proposition 13.4.7),  $|\hat{M}| \leq |C| = |M|$ , which shows that  $M$  is a maximum matching. A similar argument shows that  $C$  is a minimum vertex cover.

**13.4.29:** If each subset of women collectively knows at least as many men as there are women in that subset, then each woman can marry a man whom she knows.

## Chapter 14 Graphical Enumeration

### 14.1 Automorphisms of Simple Graphs

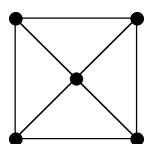
**14.1.3:**

Symmetry	Vertex permutation	Edge permutation
identity	$(u)(v)(w)(x)$	$(a)(b)(c)(d)$
horiz.refl.	$(x)(w)(u\ v)$	$(b)(d)(a\ c)$

**14.1.5:**

Vertex permutation	Edge permutation
$(u)(v)(w)$	$(a)(b)(c)$
$(u\ v\ w)$	$(a\ b\ c)$
$(u\ w\ v)$	$(a\ c\ b)$
$(u)(v\ w)$	$(b)(a\ c)$
$(v)(u\ w)$	$(c)(a\ b)$
$(w)(u\ v)$	$(a)(b\ c)$

**14.1.9:**



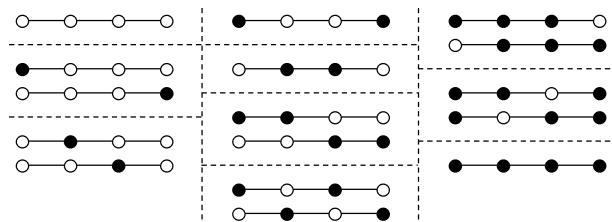
**14.1.13:** 4! automorphisms

**14.1.15:** 2!3! automorphisms

**14.1.17:** 2!3!4! automorphisms

### 14.2 Graph Colorings and Symmetry

**14.2.2:**



**14.2.9:**  $39 \text{ total} = \begin{cases} 12 & \text{using all three colors} \\ 24 & \text{using exactly two of the three colors} \\ 3 & \text{using only one color} \end{cases}$

**14.2.16:**  $6 \text{ total} = \begin{cases} 1 & \text{with all three edges light} \\ 1 & \text{with all three edges dark} \\ 2 & \text{with two edges light and one edge dark} \\ 2 & \text{with one edge light and two edges dark} \end{cases}$

**14.2.23:**  $18 \text{ total} = \begin{cases} 3 & \text{using all three colors} \\ 12 & \text{using exactly two of the three colors} \\ 3 & \text{using only one color} \end{cases}$

### 14.3 Burnside's Lemma

**14.3.3a:**  $\mathcal{S}tab(u) = \{\epsilon\}$   
 $\mathcal{S}tab(v) = \{\epsilon\}$   
 $\mathcal{S}tab(w) = \{\epsilon, (u v)(x)(w)\}$   
 $\mathcal{S}tab(x) = \{\epsilon, (u v)(x)(w)\}$

**14.3.3b:**  $Fix((u)(v)(w)(x)) = \{u, v, w, x\}$   
 $Fix((u v)(w)(x)) = \{w, x\}$

**14.3.3c:** 3 orbits:  $\{u, v\}$ ,  $\{w\}$ ,  $\{x\}$ .

**14.3.3d:**  $1+1+2+2 = 4+2$

**14.3.3e:** For  $u$  or  $v$ ,  $1 = \frac{2}{2}$ . For  $x$  or  $w$ ,  $2 = \frac{2}{1}$ .

**14.3.3f:**  $\frac{1}{1} + \frac{1}{1} + \frac{1}{2} + \frac{1}{2} = 3$ .

**14.3.3g:**  $3 = \frac{6}{2}$ .

**14.3.5a:**  $\mathcal{S}tab(u) = \{\epsilon, (u)(v w)\}$   
 $\mathcal{S}tab(v) = \{\epsilon, (v)(u w)\}$   
 $\mathcal{S}tab(w) = \{\epsilon, (w)(u v)\}$

**14.3.5b:**  $Fix((u)(v)(w)) = \{u, v, w\}$   
 $Fix((u v)(w)) = \{w\}$   
 $Fix((u w)(v)) = \{v\}$   
 $Fix((v w)(u)) = \{u\}$   
 $Fix((u v w)) = \emptyset$   
 $Fix((u w v)) = \emptyset$

**14.3.5c:** 1 orbits:  $\{u, v, w\}$ .

**14.3.5d:**  $2+2+2 = 3+1+1+1+0+0$ .

**14.3.5e:** For any vertex,  $2 = \frac{6}{3}$ .

**14.3.5f:**  $\frac{1}{3} + \frac{1}{3} + \frac{1}{3} = 1$ .

**14.3.5g:**  $1 = \frac{6}{6}$ .

**14.3.19:** These answers are essentially the same as for Exercise 14.3.3.

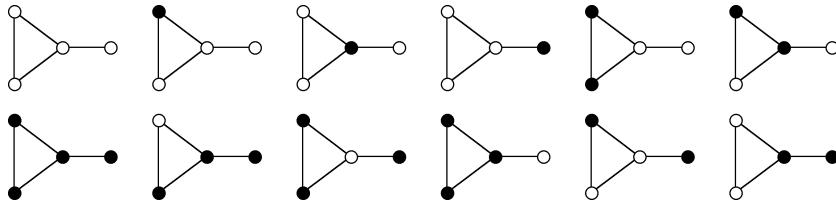
**14.3.21:** These answers are essentially the same as for Exercise 14.3.5.

#### 14.4 Cycle-Index Polynomial of a Permutation Group

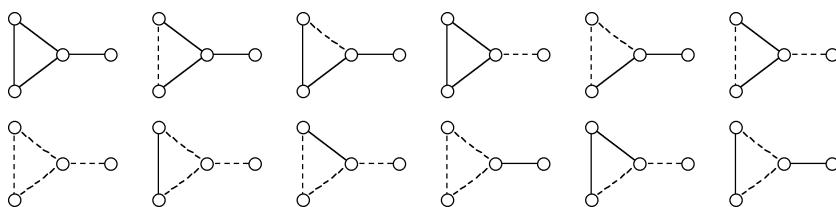
**14.4.3a:**  $\frac{1}{2}(z_1^4 + z_1^2 z_2)$  Substituting 2 for  $z_1$  and  $z_2$  yields 12.

**14.4.3b:**  $\frac{1}{2}(z_1^4 + z_1^2 z_2)$  Substituting 2 for  $z_1$  and  $z_2$  yields 12.

**14.4.3c:**



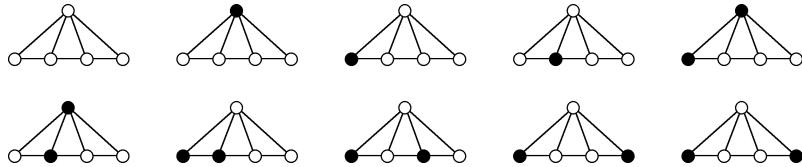
**14.4.3d:**



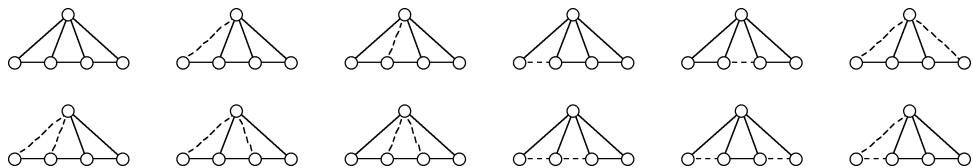
**14.4.13a:**  $\frac{1}{2}(z_1^5 + z_1 z_2^2)$  Substituting 2 for  $z_1$  and  $z_2$  yields 20.

**14.4.13b:**  $\frac{1}{2}(z_1^7 + z_1 z_2^3)$  Substituting 2 for  $z_1$  and  $z_2$  yields 72.

**14.4.13c:** These 10 colorings and their black-white complements.



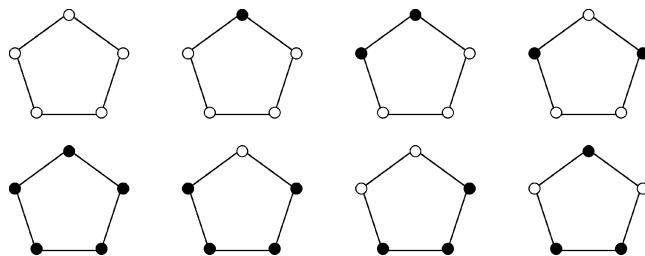
**14.4.13d:** Here are 12 of the 72 edge- $(\leq 2)$ -colorings.



### 14.5 More Counting, Including Simple Graphs

**14.5.3a:**

$$\begin{aligned}\mathcal{Z}_{\text{Aut}_V(C_5)} &= \frac{1}{10}(z_1^5 + 5z_1z_2^2 + 4z_5) \\ \mathcal{Z}_{\text{Aut}_V(C_5)}(2, \dots, 2) &= \frac{1}{10}(2^5 + 5 \cdot 2 \cdot 2^2 + 4 \cdot 2) = \frac{1}{10}(32 + 40 + 8) = 8\end{aligned}$$



**14.5.3b:**

$$\begin{aligned}\mathcal{Z}_{\text{Aut}_V(C_5)}(3, \dots, 3) &= \frac{1}{10}(3^5 + 5 \cdot 3 \cdot 3^2 + 4 \cdot 3) \\ &= \frac{1}{10}(243 + 135 + 12) = 39\end{aligned}$$

Confirm as follows, using drawings:

1 way with exactly 1 color,  $\binom{3}{1}$  ways to choose, yields 3 colorings.

6 ways with exactly 2 colors,  $\binom{3}{2}$  ways to choose, yields 18 colorings.

All 3 colors, with three vertices getting a single color, yields 6 colorings.

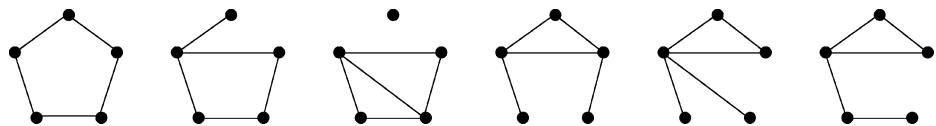
All 3 colors, with no three vertices getting a single color, yields 12 colorings.

**14.5.10:** This computation and answer are exactly like Exercise 14.5.3.

**14.5.20:** There are 10 in all, as per solution to Exercise 14.6.8.

### 14.6 Pólya-Burnside Enumeration

**14.6.4:** The six simple graphs with 5 vertices and 5 edges.



**14.6.6:**

$$\begin{aligned}x_1^{10} + x_1^9x_2 + 2x_1^8x_2^2 + 4x_1^7x_2^3 + 6x_1^6x_2^4 + 6x_1^5x_2^5 \\ + 6x_1^4x_2^6 + 4x_1^3x_2^7 + 2x_1^2x_2^8 + x_1x_2^9 + x_2^{10}\end{aligned}$$

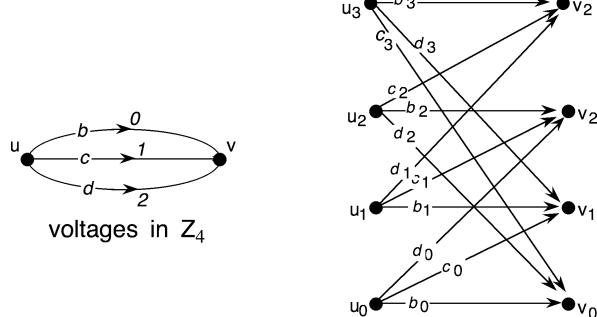
**14.6.8:**

$$\begin{aligned}\mathcal{Z}_{\text{Aut}_E(K_5)}(3, \dots, 3) &= \frac{1}{6}(3^3 + 3 \cdot 3 \cdot 3 + 2 \cdot 3) \\ &= \frac{1}{6}(27 + 27 + 6) = 10\end{aligned}$$

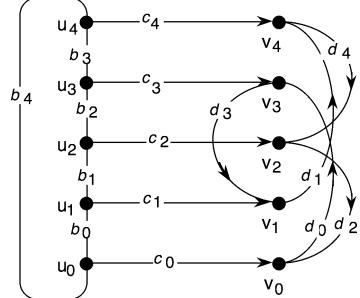
## Chapter 15 Algebraic Specification of Graphs

### 15.1 Cyclic Voltages

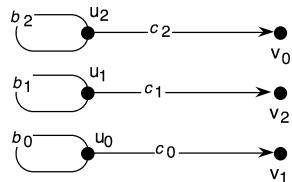
15.1.1:



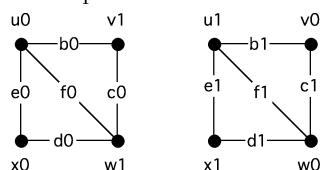
15.1.3:



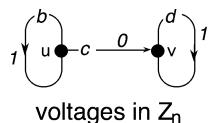
15.1.7:



15.1.9: As long as the notation is clear, there is no reason to insist that subscripts look like subscripts.



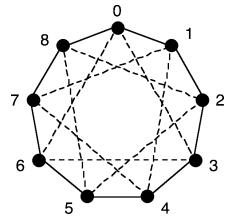
15.1.17: This voltage graph generates the circular ladder  $CL_n$ .



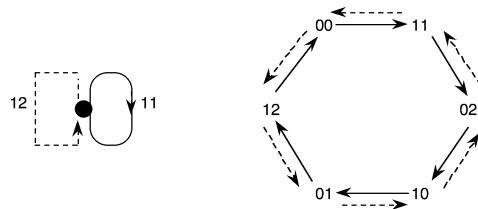
15.1.20: The bouquet  $B_{n-1}$  with voltages  $1, 2, \dots, n-1 \pmod{2n}$  yields the octahedral graph  $\mathcal{O}_n$ .

## 15.2 Cayley Graphs and Regular Voltages

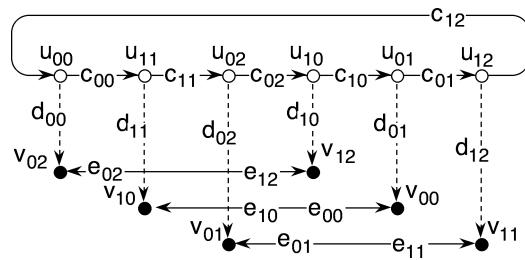
**15.2.1:** Solid edges have generator 1 and dashed edges have generator 3.



**15.2.5:** These two generators happen to be inverses of each other.



**15.2.9:** This Cayley graph could also be drawn to look like a hexagon with three “diameters” that cross at the center.



**15.2.13:** Rotation 2/9 clockwise.

**15.2.17:** Use the vertex transformation

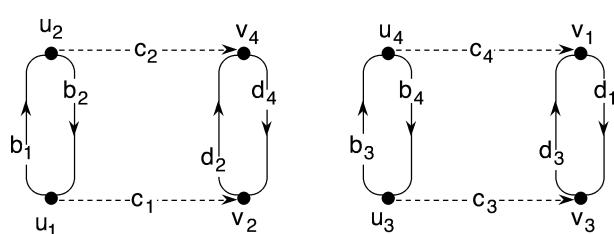
$$(u_{00} \ u_{10})(u_{01} \ u_{11})(v_{00} \ v_{10})(v_{01} \ v_{11})$$

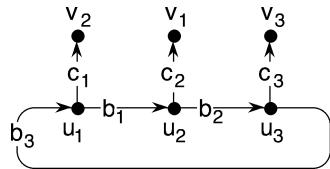
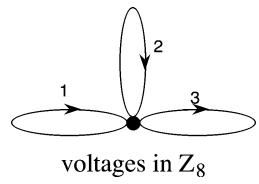
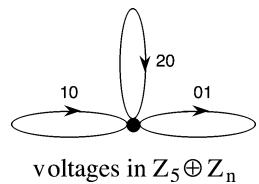
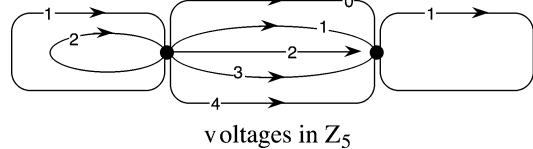
and the edge transformation

$$(b_{00} \ b_{10})(b_{01} \ b_{11})(c_{00} \ c_{10})(c_{01} \ c_{11})(d_{00} \ d_{10})(d_{01} \ d_{11})(e_{00} \ e_{10})(e_{01} \ e_{11})$$

## 15.3 Permutation Voltages

**15.3.1:**



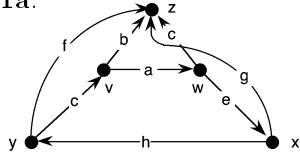
**15.3.5:****15.3.12:**  $(000 \ 011 \ 110)(001 \ 111 \ 100)(101)(010)$ **15.3.14:**  $(001)$ **15.4 Symmetric Graphs and Parallel Architectures****15.4.1:****15.4.6:****15.4.8:****15.5 Interconnection-Network Performance****15.5.1:**  $diam(CL_6) = 4$ .**15.5.5:**  $diam(ML_6) = 3$ .**15.5.10:**  $diam(C_5 \times C_5) = 4$ .**15.5.14:**  $diam(K_5 + C_5) = 2$ .**15.5.16:**  $d_{avg}(CL_6) = \frac{24}{11}$ .**15.5.20:**  $d_{avg}(ML_6) = \frac{23}{11}$ .**15.5.24:**  $d_{avg}(C_4 \times C_4) = \frac{32}{15}$ .**15.5.29:**  $d_{avg}(K_5 + C_5) = \frac{12}{11}$ .

## Chapter 16 Nonplanar Layouts

## 16.1 Representing Imbedding by Rotations

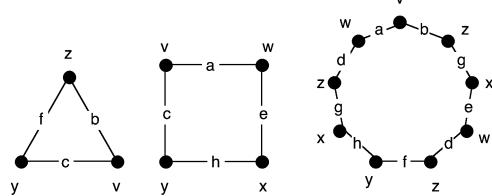
16.1.1:	$v_1.$	$a^+$	$h^+$	$e^-$
	$v_2.$	$a^-$	$f^-$	$b^+$
	$v_3.$	$b^-$	$g^+$	$c^+$
	$v_4.$	$c^-$	$d^+$	$h^-$
	$v_5.$	$d^-$	$g^-$	$f^+ \quad e^+$

### 16.1.11a:

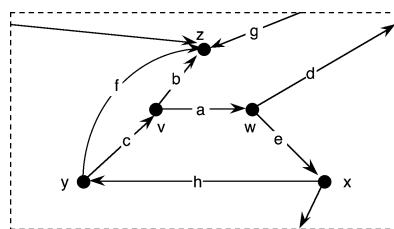


$$16.1.11b: \quad (a^+ d^+ g^- h^+ f^+ d^- e^+ g^+ b^-)(a^- c^- h^- e^-)(b^+ f^- c^+).$$

### 16.1.17a:



### 16.1.17b:



$$16.1.25: \quad (3!)^8 \cdot 7!$$

## 16.2 Genus Distribution of a Graph

### 16.2.3:

$$\left\lceil \frac{42}{6} - \frac{10}{2} + 1 \right\rceil = 3$$

### 16.2.12:

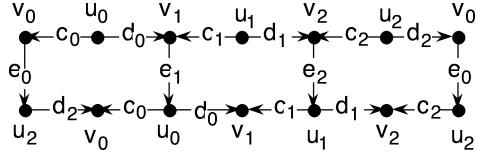
$$\left[ \frac{19}{6} - \frac{8}{2} + 1 \right] = 1$$

## 16.2.16:

$$\left\lfloor \frac{27 - 9 + 1}{2} \right\rfloor = 9$$

### 16.3 Voltage-Graph Specification of Graph Layouts

16.3.1: Paste left to right, then top to bottom with a one-sixth twist.

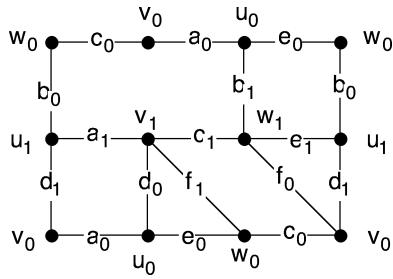


16.3.3: Suppose the voltage group has order  $n$ . Then  $|V^\alpha| = n|V|$ ,  $|E^\alpha| = n|E|$ , and (by KVL)  $|F^\alpha| = n|F|$ .

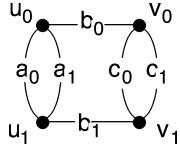
16.3.9:  $\langle u_{01}, c_{01}^+, v_{01}, d_{00}^-, u_{00}, e_{00}^+, v_{10} \rangle$ .

### 16.4 Non-KVL Imbedded Voltage Graphs

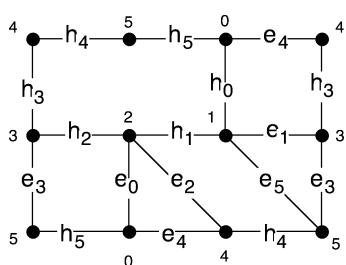
16.4.1: Paste left to right, then top to bottom with a one-third twist.



16.4.3: This imbedding is in the sphere.



16.4.6:



16.4.7: There are four faces: two 8-gons and two 16-gons.

### 16.5 The Heawood Map-Coloring Problem

16.5.1: Use a 16-gon that looks like Figures 16.5.2 and 16.5.3.

16.5.2: Use voltages in  $Z_{20}$  on the imbedded voltage graph of Figure 16.5.2.

**16.5.5:** Use  $Z_4$ -voltages on the dipole  $D_3$ .

**16.5.6:** Use  $Z_5$ -voltages on the dipole  $D_3$ .



# Graph Theory

and ITS APPLICATIONS

Second Edition

Interest in graphs and their applications continues to grow rapidly. Already an international bestseller, with the release of this second edition, **Graph Theory and Its Applications** is now an even better choice for formal course work, for self-study, and for reference.

The superior explanations, broad coverage, and abundance of illustrations and exercises that positioned this as the premier graph theory text remain, but are now augmented by a broad range of improvements. Nearly 200 pages have been added for this edition, including nine new sections and hundreds of new exercises, mostly non-routine.

#### What else is new?

- New chapters on measurement and analytic graph theory
- Supplementary exercises in each chapter – ideal for reinforcing, reviewing, and testing
- Solutions and hints, often illustrated with figures, to selected exercises — nearly 50 pages
- Reorganization and extensive revision in more than half of the existing chapters for smoother flow of exposition
- Foreshadowing — the first three chapters now preview a number of concepts, mostly via the exercises, to pique the interest of the reader

Gross and Yellen take a comprehensive approach to graph theory that integrates careful exposition of classical developments with emerging methods, models, and practical needs. Whether your interests lean toward mathematics, computer science, operations research, or engineering and the physical sciences, this unparalleled treatment builds the foundation and provides the tools needed for real-world applications.

#### Features

- Offers a comprehensive but accessible, applications-driven treatment of graph theory suitable for a variety of graduate and advanced undergraduate courses
- Provides better coverage of algorithms and algebraic and topological graph theory than any other text
- Supplies hundreds of drawings that promote spatial intuition
- Incorporates levels of carefully designed exercises that promote retention and develop and sharpen problem-solving

C505X

ISBN 1-58488-505-X

9 0000



9 781584 885054