Undergraduate Topics in Computer Science

Md. Saidur Rahman

Basic Graph Theory

Springer

Undergraduate Topics in Computer Science

Undergraduate Topics in Computer Science (UTiCS) delivers high-quality instructional content for undergraduates studying in all areas of computing and information science. From core foundational and theoretical material to final-year topics and applications, UTiCS books take a fresh, concise, and modern approach and are ideal for self-study or for a one- or two-semester course. The texts are all authored by established experts in their fields, reviewed by an international advisory board, and contain numerous examples and problems. Many include fully worked solutions.

More information about this series at http://www.springer.com/series/7592

Md. Saidur Rahman

Basic Graph Theory

Springer

Md. Saidur Rahman
Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology (BUET)
Dhaka
Bangladesh

ISSN 1863-7310 ISSN 2197-1781 (electronic)
Undergraduate Topics in Computer Science
ISBN 978-3-319-49474-6 ISBN 978-3-319-49475-3 (eBook)
DOI 10.1007/978-3-319-49475-3

Library of Congress Control Number: 2016961329

Printed on acid-free paper

This Springer imprint is published by Springer Nature
The registered company is Springer International Publishing AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Preface

This book is written based on my class notes developed while teaching the undergraduate graph theory course "Basic Graph Theory" at the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology (BUET). Due to numerous applications in modeling problems in almost every branch of science and technology, graph theory has appeared as a vital component of mathematics and computer science curricula of universities all over the world.

There are several excellent books on graph theory. Harary's book (Graph Theory, Addison-Wesley, Reading, Mass, 1969) is legendary while Wilson's book (Introduction to Graph Theory, 4th edn, Longman, 1996) is an excellent introductory textbook of graph theory. The book by West (Introduction to Graph Theory, 2nd edn, Prentice-Hall, 2001) is the most comprehensive book, which covers both introductory and advanced topics of graph theory. Agnarsson and Greenlaw in their book (Graph Theory: Modeling, Applications and Algorithms, Pearson Education, Inc., 2007) presented graph theory in a rigorous way using practical, intuitive, and algorithmic approaches. This list also includes many other nice books such as the book of Deo (Graph Theory with Applications to Engineering and Computer Science, 1974) and the book of Pirzada (An Introduction to Graph Theory, University Press, India, 2009). Since I have followed those books in my classes, most of the contents of this book are taken from those books. However, in this book all good features of those books are tied together with the following features:

- terminologies are presented in simple language with illustrative examples,
- proofs are presented with every details and illustrations for easy understanding,
- constructive proofs are preferred to existential proofs so that students can easily develop algorithms.

This book is primarily intended for use as a textbook at the undergraduate level. Topics are organized sequentially in such a way that an instructor can follow as it is. The organization of the book is as follows. In Chapter 1 historical background, motivation, and applications of graph theory are presented. Chapter 2 provides

basic graph theoretic terminologies. Chapter 3 deals with paths, cycles, and connectivity. Eulerian graphs and Hamiltonian cycles are also presented in this chapter. Chapter 4 deals with trees whereas Chapter 5 focuses on matchings and coverings. Planar graphs are treated in Chapter 6. Basic and fundamental results on graph coloring are presented in Chapter 7. Chapter 8 deals with digraphs. Chapters 9 and 10 exhibit the unique feature of this book; Chapter 9 presents some special classes of graphs, and some research topics are introduced in Chapter 10. While teaching the graph theory course to undergraduate students of computer science and engineering, I have found many students who started their research career by doing research on graph theory and graph algorithms. Some special classes of graphs (on which many hard graph problems are efficiently solvable) together with some research topics can give direction to such students for selecting their first research topics.

While revising research articles of my students I often face difficulties, since they are not familiar with formal mathematical writing. Thus I have used formal mathematical styles in writing this book so that the students can learn these styles while reading this book.

I would like to thank my undergraduate students of the Department of Computer Science and Engineering, BUET, who took notes on my class lectures and handed those to me. My undergraduate student Muhammad Jawaherul Alam started to compile those lectures. I continued it and prepared a complete manuscript during my sabbatical leave at Military Institute of Science and Technology (MIST), Dhaka. I have used the manuscript in undergraduate courses at BUET and MIST for students' feedback. I thank the students of Basic Graph Theory Course at BUET and MIST for pointing out several typos and inconsistencies. My heartfelt thanks go to Shin-ichi Nakano of Gunma University who read the manuscript thoroughly, pointed out several mistakes and suggested for improving the presentation of the book. I must appreciate the useful feedback provided by my former Ph.D. student Md. Rezaul Karim who used the manuscript in a course in Computer Science and Engineering Department of University of Dhaka. Some parts of this book are taken from our joint papers listed in bibliography; I thank all coauthors of those joint papers. I would like to thank my students Afia, Aftab, Debajyoti, Iqbal, Jawaherul, Manzurul, Moon, Rahnuma, Rubaiyat, and Shaheena for their useful feedback. I would particularly like to express my gratitude to Mohammad Kaykobad for his continuous encouragement. I thank Md. Afzal Hossain of MIST for providing me a wonderful environment in MIST for completing this book. I sincerely appreciate the editorial team of Springer for their nice work.

I am very much indebted to my Ph.D. supervisor Takao Nishizeki for his enormous contribution in developing my academic and research career. I thank my parents for their blessings and good wishes. Of course, no word can express the support given by my family; my wife Anisa, son Atiq, and daughter Shuprova.

Dhaka, Bangladesh
2017

Md. Saidur Rahman

Contents

1 Graphs and Their Applications 1
1.1 Introduction 1
1.2 Applications of Graphs 2
1.2.1 Map Coloring 2
1.2.2 Frequency Assignment 3
1.2.3 Supply Gas to a Locality 4
1.2.4 Floorplanning 6
1.2.5 Web Communities 7
1.2.6 Bioinformatics 7
1.2.7 Software Engineering 8
Exercises 8
References 9

2 Basic Graph Terminologies 11
2.1 Graphs and Multigraphs 11
2.2 Adjacency, Incidence, and Degree 13
2.2.1 Maximum and Minimum Degree 13
2.2.2 Regular Graphs 14
2.3 Subgraphs 15
2.4 Some Important Trivial Classes of Graphs 16
2.4.1 Null Graphs 17
2.4.2 Complete Graphs 17
2.4.3 Independent Set and Bipartite Graphs 17
2.4.4 Path Graphs 18
2.4.5 Cycle Graphs 19
2.4.6 Wheel Graphs 19
2.5 Operations on Graphs 19
2.5.1 Union and Intersection of Graphs 19
2.5.2 Complement of a Graph 20
2.5.3 Subdivisions 21

2.5.4 Contraction of an Edge . . . 22
2.6 Graph Isomorphism . . . 22
2.7 Degree Sequence . . . 24
2.8 Data Structures and Graph Representation . . . 26
2.8.1 Adjacency Matrix . . . 26
2.8.2 Incidence Matrix . . . 27
2.8.3 Adjacency List . . . 28
Exercises . . . 28
References . . . 29

3 Paths, Cycles, and Connectivity . . . 31
3.1 Walks, Trails, Paths, and Cycles . . . 31
3.2 Eulerian Graphs . . . 34
3.3 Hamiltonian Graphs . . . 36
3.4 Connectivity . . . 39
3.4.1 Connected Separable Graphs . . . 41
3.4.2 Block-Cutvertex Tree . . . 42
3.4.3 2-Connected Graphs . . . 43
3.4.4 Ear Decomposition . . . 44
Exercises . . . 46
References . . . 46

4 Trees . . . 47
4.1 Introduction . . . 47
4.2 Properties of a Tree . . . 47
4.3 Rooted Trees . . . 50
4.4 Spanning Trees of a Graph . . . 51
4.5 Counting of Trees . . . 54
4.6 Distances in Trees and Graphs . . . 58
4.7 Graceful Labeling . . . 59
Exercises . . . 60
References . . . 62

5 Matching and Covering . . . 63
5.1 Matching . . . 63
5.1.1 Perfect Matching . . . 63
5.1.2 Maximum Matching . . . 63
5.1.3 Hall's Matching Condition . . . 66
5.2 Independent Set . . . 68
5.3 Covers . . . 68
5.4 Dominating Set . . . 69
5.5 Factor of a Graph . . . 72
Exercises . . . 73
References . . . 74

6 Planar Graphs . . . 77
6.1 Introduction . . . 77
6.2 Characterization of Planar Graphs . . . 77
6.3 Plane Graphs . . . 78
6.3.1 Euler's Formula . . . 82
6.3.2 Dual Graph . . . 84
6.4 Thickness of Graphs . . . 85
6.5 Straight-Line Drawings of Planar Graphs . . . 85
Exercises . . . 87
References . . . 89

7 Graph Coloring . . . 91
7.1 Introduction . . . 91
7.2 Vertex Coloring . . . 91
7.3 Edge Coloring . . . 94
7.4 Face Coloring (Map Coloring) . . . 97
7.5 Chromatic Polynomials . . . 97
7.6 Acyclic Coloring . . . 98
Exercises . . . 101
References . . . 102

8 Digraphs . . . 103
8.1 Introduction . . . 103
8.2 Digraph Terminologies . . . 103
8.3 Eulerian Digraphs . . . 105
8.4 Hamiltonian Digraphs . . . 106
8.5 Digraphs and Tournaments . . . 106
8.6 Flow Networks . . . 107
Exercises . . . 109
References . . . 109

9 Special Classes of Graphs . . . 111
9.1 Introduction . . . 111
9.2 Outerplanar Graphs . . . 111
9.3 Triangulated Plane Graphs . . . 114
9.3.1 Canonical Ordering . . . 114
9.3.2 Separating Triangles . . . 116
9.3.3 Plane 3-Trees . . . 119
9.4 Chordal Graphs . . . 121
9.5 Interval Graphs . . . 124
9.6 Series-Parallel Graphs . . . 125
9.7 Treewidth and Pathwidth . . . 130
Exercises . . . 131
References . . . 132

10 Some Research Topics 135
10.1 Introduction 135
10.2 Graph Representation 135
10.3 Graph Drawing 137
10.3.1 Drawings of Planar Graphs 138
10.3.2 Simultaneous Embedding 145
10.3.3 Drawings of Nonplanar Graphs 146
10.4 Graph Labeling 148
10.5 Graph Partitioning 150
10.6 Graphs in Bioinformatics 152
10.6.1 Hamiltonian Path for DNA Sequencing 152
10.6.2 Cliques for Protein Structure Analysis 153
10.6.3 Pairwise Compitability Graphs 154
10.7 Graphs in Wireless Sensor Networks 156
10.7.1 Topology Control 157
10.7.2 Fault Tolerance 158
10.7.3 Clustering 158
References 159

Index 165

Chapter 1
Graphs and Their Applications

1.1 Introduction

A graph consists of a set of vertices and set of edges, each joining two vertices. Usually an object is represented by a vertex and a relationship between two objects is represented by an edge. Thus a graph may be used to represent any information that can be modeled as objects and relationships between those objects. Graph theory deals with study of graphs. The foundation stone of graph theory was laid by Euler in 1736 by solving a puzzle called Königsberg seven-bridge problem [1]. Königsberg is an old city in Eastern Prussia lies on the Pregel River. The Pregel River surrounds an island called Kneiphof and separates into two branches as shown in Fig. 1.1(a) where four land areas are created: the island a, two river banks b and c, and the land d between two branches. Seven bridges connect the four land areas of the city. It is said that the people of Königsberg used to entertain themselves by trying to devise a walk around the city which would cross each of the seven bridges just once. Since their attempts had always failed, many of them believed that the task was impossible, but there was no proof until 1736. In that year, one of the leading mathematicians of that time, Leonhard Euler published a solution to the problem that no such walk is possible. He not only dealt with this particular problem, but also gave a general method for other problems of the same type. Euler constructed a mathematical model for the problem in which each of the four lands a, b, c and d is represented by a point and each of the seven bridges is represented by a curve or a line segment as illustrated in Fig. 1.2(b). The problem can now be stated as follows: Beginning at one of the points a, b, c and d, is it possible to trace the figure without traversing the same edge twice? The mathematical model constructed for the problem is known as a graph model of the problem. The points a, b, c and d are called vertices, the line segments are called edges, and the whole diagram is called a graph.

Before presenting some applications of graphs, we need to know some terminologies. A *graph* G is a tuple (V, E) which consists of a finite set V of *vertices* and a finite set E of *edges*; each edge is an unordered pair of vertices. The two vertices associated with an edge e are called the *end-vertices* of e. We often

M.S. Rahman, *Basic Graph Theory*, Undergraduate Topics in Computer Science,
DOI 10.1007/978-3-319-49475-3_1

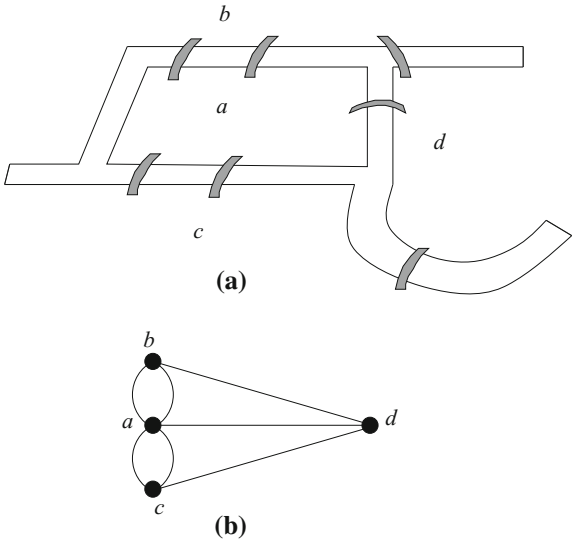

Fig. 1.1 The graph model for Königsberg bridges

denote by (u, v), an edge between two vertices u and v. We also denote the set of vertices of a graph G by $V(G)$ and the set of edges of G by $E(G)$. Let $e = (u, v)$ be an edge of a graph G. Then the two vertices u and v are said to be *adjacent* in G and the edge e is said to be *incident* to the vertices u and v. The vertex u is also called a *neighbor* of v in G and vice versa. The graph in Fig. 1.2(b) has six vertices a, b, c, d, e and f, and ten edges. Vertices a and b are the end vertices of edge (a, b). So a and b are adjacent. Vertices b, c and f are the neighbors of the vertex a.

1.2 Applications of Graphs

Graphs have applications in almost all branches of science and engineering [2, 3]. In this section we will see applications of graphs in modeling some real-world problems.

1.2.1 Map Coloring

Given a map containing several countries, where each country is a contiguous region in the map, we are asked to color the countries using different colors so that no two countries with a common boundary share the same color. Of course, our objective is to use a minimum number of colors. Such a problem can easily be modeled by a graph as follows. We represent each country by a vertex and add an edge between two vertices

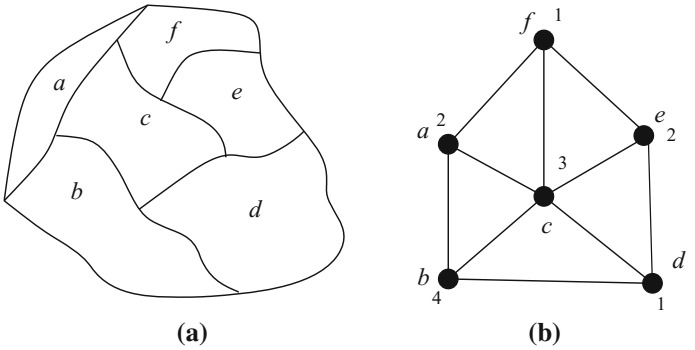

Fig. 1.2 **(a)** A map and **(b)** a graph model for map coloring

if the two countries corresponding to the vertices share a boundary, as illustrated in Fig. 1.2 where Fig. 1.2(b) illustrates the graph model for the map in Fig. 1.2(a). Now the problem becomes a graph problem which asks to color the vertices of the graph using a minimum number of colors so that two adjacent vertices get different colors. The vertices of the graph in Fig. 1.2(b) are colored with four colors and hence the regions of the map in Fig. 1.2(a) can be colored with four colors. Interestingly, the regions of every map can be colored with four colors [4].

1.2.2 Frequency Assignment

A communication engineer is going to assign frequencies to several transmitters. Due to the physical locations of the transmitters, some of the transmitter pairs are in the range of interference. Assume that there are n transmitters and that n frequencies $\{1, 2, \ldots, n\}$ will be assigned to the transmitters such that no two transmitters are assigned the same frequency, and if two transmitters are in interference range then the difference between their assigned frequencies should be as large as possible. We can easily develop a graph model of this problem as follows. We construct a graph G by representing each transmitter by a vertex of G and add an edge between two vertices u and v of G if the transmitters corresponding to vertices u and v are in interference range. Now the frequency assignment problem becomes the problem of labeling the vertices of G by $1, 2, \ldots, n$ such that one label is used for exactly one vertex by keeping the difference of the labels of two end vertices of an edge is as large as possible [5]. Five transmitters a, b, c, d, e with their transmission ranges indicated by circles are shown in Fig. 1.3(a) and the graph model is shown in Fig. 1.3(b). In the graph model, there is an edge between a and b since their transmission ranges overlap, i.e., they are in the range of interference. Other edges have also been added in the graph following the same rule. Observe that the frequencies are assigned to the vertices in such a way that the minimum difference between two adjacent labels is 2.

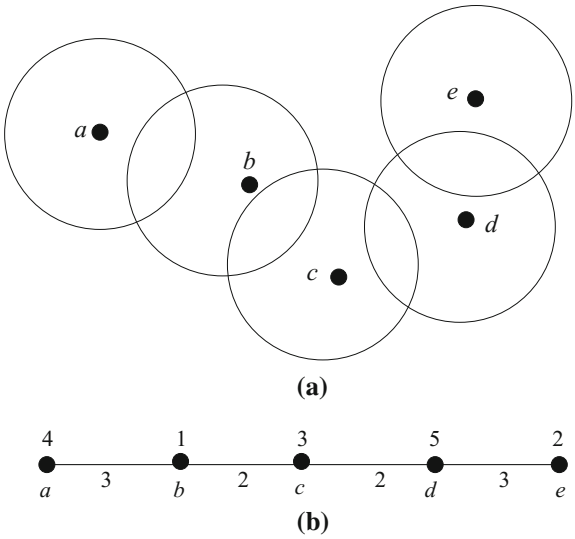

Fig. 1.3 **(a)** Transmitters with their transmission ranges, **(b)** a graph model with frequency assignment

1.2.3 Supply Gas to a Locality

A gas company wants to supply gas to a locality from a single gas source. They are allowed to pass the underground gas lines along the road network only, because no one allows to pass gas lines through the bottom of one's building. The road network divides the locality into many regions as illustrated in Fig. 1.4(a), where each road is represented by a line segment and a point at which two or more roads meet is represented by a small black circle. A point at which two or more roads meet is called an intersection point. Each region is bounded by some line segments and intersection points. These regions need to be supplied gas. If a gas line reaches an intersection point on the boundary of a region, then the region may receive gas from the line at that intersection point. Thus, the gas lines should reach the boundaries of all the regions of the locality. Gas will be supplied from a gas field which is located outside of the locality, and a single pipe line will be used to supply gas from the gas field to an intersection point on the outer boundary of the locality.

The gas company wants to minimize the establishment cost of gas lines by selecting the roads for laying gas lines such that the total length of the selected roads is minimal. Since gas will be supplied from the gas field using a single line to the locality, the selected road network should be connected and contain an intersection point on the outer boundary of the locality. Thus, the gas company needs to find a set of roads that induces a connected road network, supply gas in all the regions of the locality and the length of the induced road network is minimum. Such a set of roads is illustrated by thick lines in Fig. 1.4(b).

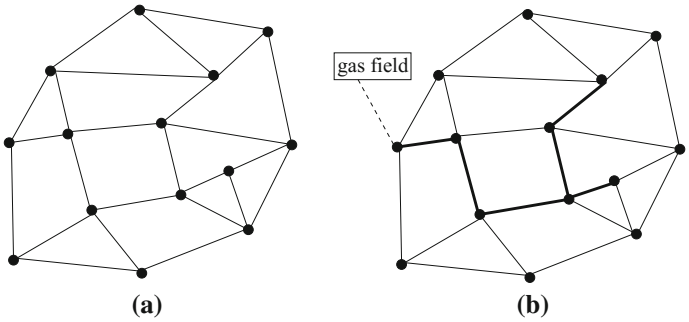

Fig. 1.4 **(a)** A locality and **(b)** a gas network

The problem mentioned above can be modeled using a "plane graph." A graph is *planar* if it can be embedded in the plane without edge crossings. A *plane graph* is a planar graph with a fixed planar embedding in the plane. A plane graph divides the plane into connected regions called *faces*. Let $G = (V, E)$ be an edge-weighted connected plane graph, where V and E are the sets of vertices and edges, respectively. Let F be the set of faces of graph G. For each edge $e \in E$, $w(e) \geq 0$ is the weight of the edge e of G. A *face-spanning subgraph* of G is a connected subgraph H induced by a set of edges $S \subseteq E$ such that the vertex set of H contains at least one vertex from the boundary of each face $f \in F$ of G [6]. Figure 1.5 shows two face-spanning subgraphs drawn by thick lines where the cost of the face-spanning subgraph in Fig. 1.5(a) is 22 and the cost of the face-spanning subgraph in Fig. 1.5(b) is 16. Thus, a plane graph may have many face-spanning subgraphs whose costs are different. A *minimum face-spanning subgraph* H of G is a face-spanning subgraph of G, where $\sum_{e \in S} w(e)$ is minimum, and a *minimum face-spanning subgraph problem*

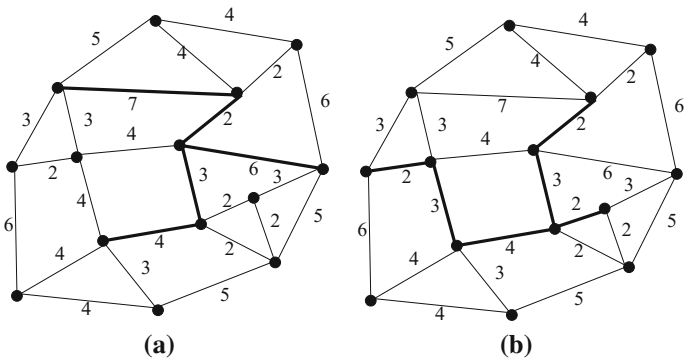

Fig. 1.5 A simple graph with **(a)** a face-spanning subgraph of cost 22 and **(b)** another face-spanning subgraph of cost 16

asks to find a minimum face-spanning subgraph of a plane graph. If we represent each road of the road network by an edge of G, each intersection point by a vertex of G, each region by a face of G, and assign the length of a road to the weight of the corresponding edge, then the problem of finding a minimum face-spanning subgraph of G is the same as the problem of the gas company mentioned above [6]. A minimum face-spanning subgraph problem often arises in applications like establishing power transmission lines in a city, power wires layout in a complex circuit, planning irrigation canal networks for irrigation systems, etc.

1.2.4 Floorplanning

Graph modeling has applications in VLSI floorplanning as well as architectural floorplaning [7]. In a VLSI floorplanning problem, an input is a plane graph F as illustrated in Fig. 1.6(a); F represents the functional entities of a chip, called *modules*, and interconnections among the modules; each vertex of F represents a module, and an edge between two vertices of F represents the interconnections between the two corresponding modules. An output of the problem for the input graph F is a partition of a rectangular chip area into smaller rectangles as illustrated in Fig. 1.6(d); each module is assigned to a smaller rectangle, and furthermore, if two modules have interconnections, then their corresponding rectangles must be adjacent, i.e., they must have a common boundary. A similar problem may arise in architectural floorplanning also. When building a house, the owner may have some preference; for example, a bedroom should be adjacent to a reading room. The owner's choice of room adjacencies can be easily modeled by a plane graph F, as illustrated in Fig. 1.6(a); each vertex represents a room and an edge between two vertices represents the desired adjacency between the corresponding rooms.

A "rectangular drawing" of a plane graph may provide a suitable solution to the floorplanning problem described above. (In a rectangular drawing of a plane graph each vertex is drawn as a point, each edge is drawn as either a horizontal line segment or a vertical line segment and each face including the outer face is drawn as a rectangle.) First, obtain a plane graph F' by triangulating all inner faces of F as illustrated in Fig. 1.6(b), where dotted lines indicate new edges added to F. Then

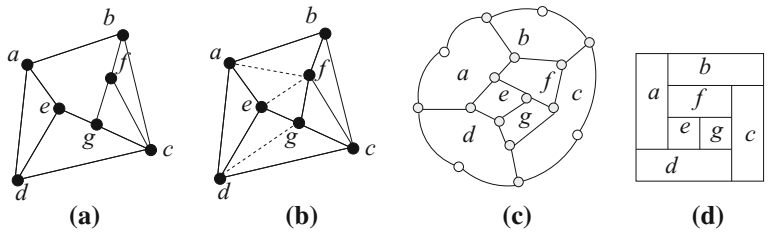

Fig. 1.6 **(a)** Graph F, **(b)** triangulated graph F', **(c)** dual-like graph G, and **(d)** rectangular drawing of G

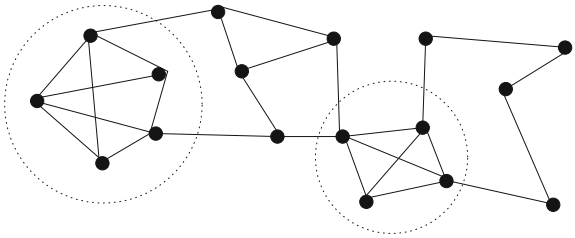

Fig. 1.7 Communities of common interests

obtain a "dual-like" graph G of F' as illustrated in Fig. 1.6(c), where the four vertices of degree 2 drawn by white circles correspond to the four corners of the rectangular area. Finally, by finding a rectangular drawing of the plane graph G, obtain a possible floorplan for F as illustrated in Fig. 1.6(d).

1.2.5 Web Communities

The World Wide Web can be modeled as a graph, where the web pages are represented by vertices and the hyperlinks between them are represented by edges. By examining web graphs it is possible to discover interesting information. For example, by extracting dense subgraphs in a web graph we can find a community of particular interest [8, 9]. In a graph representation of a web graph shown in Fig. 1.7, two possible communities of particular interests are indicated by dotted circles.

1.2.6 Bioinformatics

Graph theoretical modeling is used in multiple areas of bioinformatics including the description of taxonomic trees, phylogenetic analysis, genome ontology, and proteomics, to name a few. The graph in Fig. 1.8 illustrates an RNA secondary structure [10].

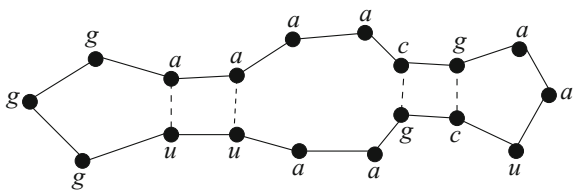

Fig. 1.8 RNA secondary structure

1.2.7 Software Engineering

There are enormous applications of graphs in various areas software engineering such as project planning, data flow analysis, software testing, etc. A control flow graph used in software testing describes how the control flows through the program. A *control flow graph* is a directed graph where each vertex corresponds to a unique program statement and each directed edge represents a control transfer from one statement to the other. By analyzing a control flow graph one can understand the complexity of the program and can design a suitable test suite for software testing [11].

Exercises

1. Study your campus map and model the road network inside your campus by a graph.
2. Construct a graph to represent the adjacency relationship of rooms in a floor of your university building.
3. Consider a party where there are exactly two alternate options of foods for each category of foods as follows. Rice: plane/yellow, Curry: fish/chicken, Naan: plain/butter, Kebab: chicken/mutton, Fruit: banana/mango, Drink: tea/coffee. Registered participants of the party gave their options as in Table 1.1. Two participants conflict in their options if they give different options in the same category. Represent the conflicts of the participants using a conflict graph where each participant is represented by a vertex and there is an edge between two vertices if the corresponding participants conflict. By observing the conflict graph, find out the minimum number of persons whose absence divides the participants into two conflict free groups.

Table 1.1 Food option

Participants	Rice	Curry	Naan	Kebab	Fruit	Drink
Abir	Plain	Fish	Plain	Chicken	Mango	Tea
Bony	Plain	Chicken	Butter	Mutton	Banana	Coffee
Dristy	Plain	Fish	Plain	Chicken	Mango	Tea
Elis	Plain	Chicken	Butter	Mutton	Banana	Coffee
Faria	Plain	Fish	Plain	Chicken	Mango	Tea
Snigdha	Yellow	Fish	Butter	Chicken	Mango	Coffee
Subir	Yellow	Chicken	Butter	Mutton	Banana	Tea
Jony	Plain	Fish	Plain	Chicken	Mango	Tea

4. There are five jobs $\{J_1, J_2, J_3, J_4, J_5\}$ in a company for which there are five workers A, B, C, D and E to do those jobs. However, everybody does not have expertise to do every job. Their expertise is as follows: $A = \{J_1, J_2, J_3\}$, $B = \{J_2, J_4\}$, $C = \{J_1, J_3, J_5\}$, $D = \{J_3, J_5\}$, $E = \{J_1, J_5\}$. Develop a graph model to represent the job expertise of the persons and find an assignment of jobs to the workers such that every worker can do a job.
5. An industry has 600 square meter rectangular area on a floor of a building where it needs to establish four processing units A, B, C, and D. Processing units A and D require 100 square meter area each whereas B and C require 200 square meter each. Furthermore, the following adjacency requirements must be satisfied: B, C, and D should be adjacent to A; A and D should be adjacent to B; A and D should be adjacent to C; and A, B, and C should be adjacent to D. Can you construct a floor layout where the space for each processing unit will be a rectangle? Propose a suitable layout in your justification.

References

1. Biggs, N.L., Lloyd, E.K., Wilson, R.J.: Graph Theory: 1736–1936. Oxford University Press, Oxford (1976)
2. Bondy, J.A., Murty, U.S.R.: Graph Theory with Applications. Elsevier Science Ltd/North-Holland, New York (1976)
3. Gross, J.L., Yellen, J.: Graph Theory and Its Applications, 2nd edn. Chapman and Hall, London (2005)
4. Appel, K., Haken, W.: Every map is four colourable. Bull. Am. Math. Soc. **82**, 711–712 (1976)
5. Calamoneri, T., Massini, A., Török, L., Vrt'o, I.: Antibandwidth of complete k-ary trees. Discret. Math. **309**(22), 6408–6414 (2009)
6. Patwary, M.M.A., Rahman, M.S.: Minimum face-spanning subgraphs of plane graphs. AKCE J. Graphs. Combin. **7**(2), 133–150 (2010)
7. Nishizeki, T., Rahman, M.S.: Planar Graph Drawing. World Scientific, Singapore (2004)
8. Porter, M.A., Onnela, J.-P., Mucha, P.J.: Communities in networks. Not. Amer. Math. Soc. **56**, 1082–1097, 1164–1166 (2009)
9. Chen, J., Saad, Y.: Dense subgraph extraction with application to community detection, in IEEE Trans. Knowl. Data Eng. **24**(7), 1216–1230 (2012). doi:10.1109/TKDE.2010.271
10. Sung, W.: Algorithms in Bioinformatics. CRC Press, London (2010)
11. Mall, R.: Fundamentals of Software Engineering, 4th edn. Prentice-Hall of India Pvt. Ltd. (2014)

Chapter 2
Basic Graph Terminologies

In this chapter, we learn some definitions of basic graph theoretic terminologies and know some preliminary results of graph theory.

2.1 Graphs and Multigraphs

A *graph* G is a tuple consisting of a finite set V of vertices and a finite set E of edges where each edge is an unordered pair of vertices. The two vertices associated with an edge e are called the *end-vertices* of e. We often denote by (u, v), an edge between two vertices u and v. We also denote the set of vertices of a graph G by $V(G)$ and the set of edges of G by $E(G)$. A vertex of a graph is also called as a *node* of a graph.

We generally draw a graph G by representing each vertex of G by a point or a small circle and each edge of G by a line segment or a curve between its two end-vertices. For example, Fig. 2.1 represents a graph G where $V(G) = \{v_1, v_2, \ldots, v_{11}\}$ and $E(G) = \{e_1, e_2, \ldots, e_{17}\}$. We often denote the number of vertices of a graph G by n and the number of edges of G by m; that is, $n = |V(G)|$ and $m = |E(G)|$. We will use these two notations n and m to denote the number of vertices and the number of edges of a graph unless any confusion arises. Thus $n = 11$ and $m = 17$ for the graph in Fig. 2.1.

A *loop* is an edge whose end-vertices are the same. *Multiple edges* are edges with the same pair of end-vertices. If a graph G does not have any loop or multiple edge, then G is called a *simple graph*; otherwise, it is called a *multigraph*. The graph in Fig. 2.1 is a simple graph since it has no loop or multiple edge. On the other hand, the graph in Fig. 2.2 contains a loop e_5 and two sets of multiple edges $\{e_2, e_3, e_4\}$ and $\{e_6, e_7\}$. Hence the graph is a multigraph. In the remainder of the book, when we say a graph, we shall mean a simple graph unless there is any possibility of confusion.

We call a graph a *directed graph* or *digraph* if each edge is associated with a direction, as illustrated in Fig. 2.3(a). One can consider a directed edge as a one-way street. We thus can think an undirected graph as a graph where each edge is directed

M.S. Rahman, *Basic Graph Theory*, Undergraduate Topics in Computer Science,
DOI 10.1007/978-3-319-49475-3_2

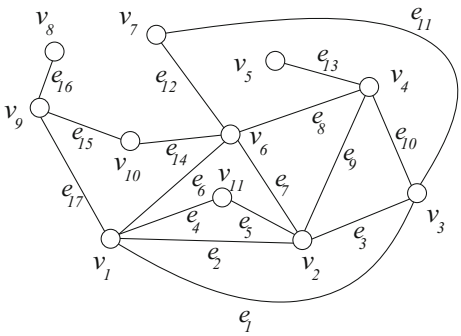

Fig. 2.1 A simple graph with 11 vertices and 17 edges

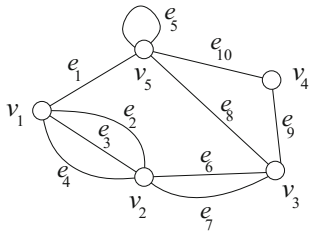

Fig. 2.2 A multigraph

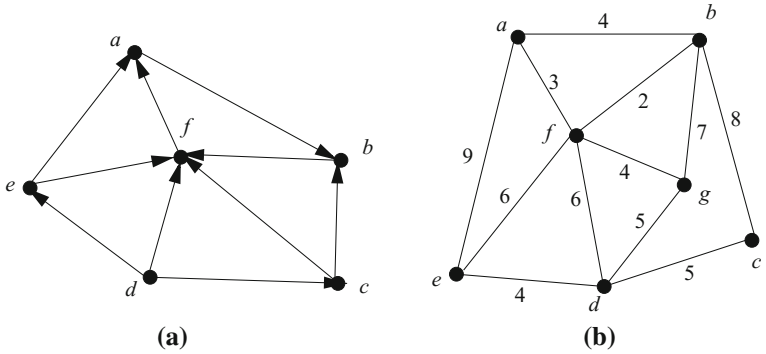

Fig. 2.3 **(a)** A directed graph and **(b)** an edge-weighted graph

in both directions. We deal with digraphs in Chapter 8. We call a graph a *weighted graph* if a weight is assigned to each vertex or to each edge. Figure 2.3(b) illustrates an edge-weighted graph where a weight is assigned to each edge.

2.2 Adjacency, Incidence, and Degree

Let $e = (u, v)$ be an edge of a graph G. Then the two vertices u and v are said to be *adjacent* in G, and the edge e is said to be *incident* to the vertices u and v. The vertex u is also called a *neighbor* of v in G and vice versa. In the graph in Fig. 2.1, the vertices v_1 and v_3 are adjacent; the edge e_1 is incident to the vertices v_1 and v_3. The neighbors of the vertex v_1 in G are v_2, v_3 v_6, v_9, and v_{11}.

The *degree* of a vertex v in a graph G, denoted by $deg(v)$ or $d(v)$, is the number of edges incident to v in G, with each loop at v counted twice. The degree of the vertex v_1 in the graph of Fig. 2.1 is 5. Similarly, the degree of the vertex v_5 in the graph of Fig. 2.2 is also 5.

Since the degree of a vertex counts its incident edges, it is obvious that the summation of the degrees of all the vertices in a graph is related to the total number of edges in the graph. In fact the following lemma, popularly known as the "Degree-sum Formula," indicates that summing up the degrees of each vertex of a graph counts each edge of the graph exactly twice.

Lemma 2.2.1 (Degree-sum Formula) *Let $G = (V, E)$ be a graph with m edges. Then $\sum_{v \in V} deg(v) = 2m$.*

Proof Every nonloop edge is incident to exactly two distinct vertices of G. On the other hand, every loop edge is counted twice in the degree of its incident vertex in G. Thus, every edge, whether it is loop or not, contributes a two to the summation of the degrees of the vertices of G. □

The above lemma, due to Euler (1736), is an essential tool of graph theory and is sometimes refer to as the "First Theorem of Graph Theory" or the "Handshaking Lemma." It implies that if some people shake hands, then the total number of hands shaken must be even since each handshake involves exactly two hands. The following corollary is immediate from the degree-sum formula.

Lemma 2.2.2 *The number of odd degree vertices in a graph is an even number.*

Proof Let G be a graph with m edges. Let x be the sum of the degrees of even degree vertices and y be the sum of the degrees of odd degree vertices. By Lemma 2.2.1 $x + y = 2m$. Since x is the sum of even integers, x is even, and hence $y = 2m - x$ is also an even integer. Since y is the sum of odd integers, the number of addends in the sum must be even. Thus, the number of odd degree vertices must be even. □

2.2.1 Maximum and Minimum Degree

The *maximum degree of a graph* G, denoted by $\Delta(G)$, is the maximum value among the degrees of all the vertices of G, i.e., $\Delta(G) = \max_{v \in V(G)} deg(v)$. Similarly, we define

the *minimum degree of a graph G* and denote it by $\delta(G)$, i.e., $\delta(G) = \min_{v \in V(G)} deg(v)$. The maximum and the minimum degree of the graph in Fig. 2.1 are 5 and 1, respectively.

Let G be a graph with n vertices and m edges. Then by the degree-sum formula, the average degree of a vertex in G is $\frac{2m}{n}$ and hence $\delta(G) \leq \frac{2m}{n} \leq \Delta(G)$.

2.2.2 Regular Graphs

If all the vertices of a graph G have equal degrees, then we call G a *regular graph*. We call it a *k-regular graph* if the common degree is k. Figure 2.4 represents some regular graphs.

Only the graphs with empty edge sets (having nonempty vertex sets) are 0-regular. These are called *null graphs*. Similarly, a 1-regular graph consists of a set of edges such that no two edges are incident to a common vertex. A graph which consists of a list of vertices such that each pair of consecutive vertices are adjacent with each other and the first vertex is also adjacent with the last vertex, as illustrated in Fig. 2.4(a), is a 2-regular graph. Such a graph is called a *cycle* or a *simple cycle*. A graph which is a collection of simple cycles is also a 2-regular graph.

A 3-regular graph is also called a *cubic graph*. The graph in Fig. 2.4(b) is a cubic graph. This particular graph is also known as the "Petersen graph" after the name of Julius Petersen, who constructed it in 1898. This graph has 10 vertices and 15 edges. The vertices of Petersen graph can be labeled by two element subsets of the set $\{1, 2, 3, 4, 5\}$ such that the labels of two adjacent vertices are disjoint, as illustrated in Fig. 2.5. Petersen graph shows some interesting properties and also serves as a minimum-sized example and counterexample for many problems in graph theory. Doughnut graphs [1] are examples of 5-regular graphs. A p-doughnut graph has exactly $4p$ vertices. Figure 2.4(d) illustrates a p-doughnut graph for $p = 4$. Another important example of a regular graph is a "d-dimensional hypercube" or simply "hypercube." A d-dimensional hypercube has 2^d vertices and each of its vertices has degree d. Figure 2.4(c) illustrates a d-dimensional hypercube for $d = 4$.

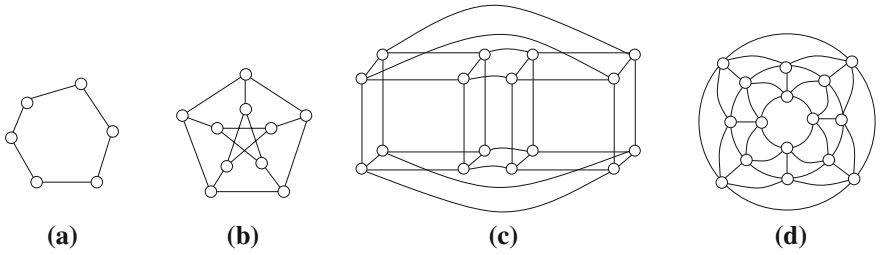

Fig. 2.4 **(a)** A 2-regular graph (a simple cycle), **(b)** a 3-regular graph (Petersen graph), **(c)** a 4-regular graph (4-dimensional hypercube), and **(d)** a 5-regular graph (a doughnut graph)

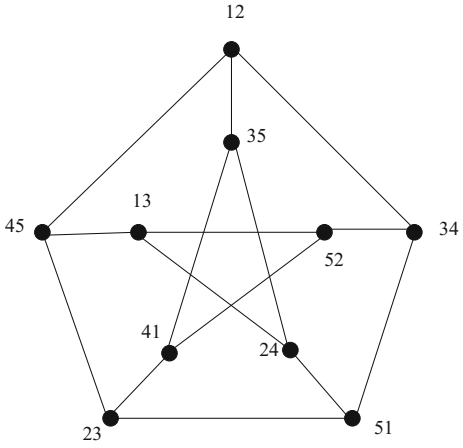

Fig. 2.5 A labeled Petersen graph

The degree-sum formula implies the following two corollaries for regular graphs.

Corollary 2.2.3 *Every regular graph with an odd degree has an even number of vertices.*

Corollary 2.2.4 *A k-regular graph with n vertices has $nk/2$ edges.*

2.3 Subgraphs

A subgraph of a graph $G = (V, E)$ is a graph $G' = (V', E')$ such that $V' \subseteq V$ and $E' \subseteq E$. For instance, the graphs in Figs. 2.6(b)–(e) are subgraphs of the graph in Fig. 2.6(a).

We can obtain subgraphs of a graph G by deleting some vertices and edges of G. Let e be an edge of G. We denote by $G - e$ the graph obtained by deleting the edge e from G. More generally, if F is a set of edges of G, we denote by $G - F$ the graph obtained by deleting all the edges in F from G. Figure 2.6(a) illustrates a graph G and Fig. 2.6(b) illustrates the graph $G - e_6$ obtained by deleting the edge e_6 from G.

Similarly, we can define the deletion of a vertex from a graph. However, deleting a vertex v from a graph G also requires that we also delete the edges incident to v in G. Let v be a vertex of a graph G. We denote by $G - v$ the graph obtained by deleting the vertex v and all its incident edges from G. More generally, if W is a set of vertices of G, we denote by $G - W$ the graph obtained by deleting the vertices in W (and all the incident edges) from G. Figure 2.6(a) illustrates a graph G and Fig. 2.6(c) illustrates the graph $G - v_7$ obtained by deleting the vertex v_7 from G.

Let $G = (V, E)$ be a graph and let W be a set of vertices of G. A subgraph $G' = (V', E')$ of G is called *a subgraph of G induced by W* if $V' = W$ and E'

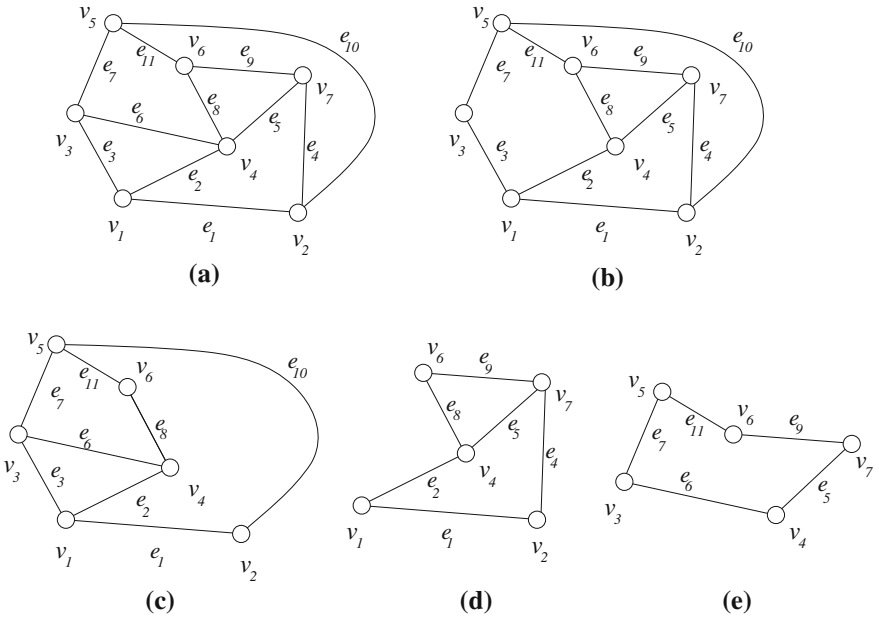

Fig. 2.6 **(a)** A Graph G, **(b)–(e)** subgraphs of G

consists of all those edges e of G such that both the end-vertices of e are in W. The graph in Fig. 2.6(d) is a subgraph of the graph of Fig. 2.6(a) induced by the set of vertices $\{v_1, v_2, v_4, v_6, v_7\}$.

Let $G = (V, E)$ be a graph and let F be a set of edges of G. A subgraph $G' = (V', E')$ of G is called *a subgraph of G induced by F* if $E' = F$ and V' consists of all those vertices of G each of which is an end-vertex of some edge in F. The graph in Fig. 2.6(e) is a subgraph of the graph of Fig. 2.6(a) induced by the set of edges $\{e_5, e_6, e_7, e_9, e_{11}\}$.

2.4 Some Important Trivial Classes of Graphs

In this section, we see some special classes of graphs, which will often appear in our discussion in the subsequent chapters.

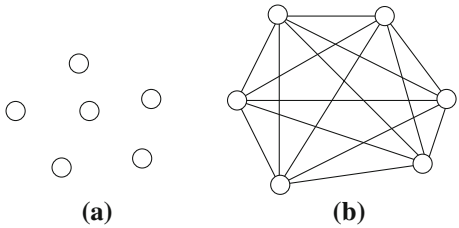

Fig. 2.7 **(a)** A null graph N_6 with six vertices, **(b)** a complete graph K_6 with six vertices

2.4.1 Null Graphs

A graph with an empty edge set is called a *null graph*. A null graph with n vertices is denoted by N_n. Figure 2.7(a) illustrates the null graph N_6 with six vertices. A null graph is a subgraph of any graph with the same number of vertices.

2.4.2 Complete Graphs

A graph in which each pair of distinct vertices are adjacent is called a *complete graph*. A complete graph with n vertices is denoted by K_n. It is trivial to see that K_n contains $n(n-1)/2$ edges. Figure 2.7(b) illustrates a complete graph K_6 with six vertices. Any graph is a subgraph of the complete graph with the same number of vertices and thus the number of edges in a graph with n vertices is at most $n(n-1)/2$.

2.4.3 Independent Set and Bipartite Graphs

Let $G = (V, E)$ be a graph. A subset of vertices $V' \subseteq V$ is called an *independent set* in G if for every pair of vertices $u, v \in V'$, there is no edge in G joining the two vertices u and v.

A graph G is called a *bipartite graph* if the vertex set V of G can be partitioned into two disjoint nonempty sets V_1 and V_2, both of which are independent. The two sets V_1 and V_2 are often called the *partite sets* of G. Each edge of a bipartite graph G thus joins exactly one vertex of V_1 to exactly one vertex of V_2. Figure 2.8 shows two bipartite graphs where the independent partitions are shaded in both the graphs. Given a graph G, one can test whether G is a bipartite graph in a naive approach by considering each possible bipartition of the vertices of G and checking whether the two partitions are independent or not. However since there are $2^n - 2$ possible bipartition of a graph with n vertices, this approach takes exponential time. Fortunately, there is a linear-time algorithm to test whether a graph is bipartite or not. The idea is simple. Using a breadth first search (BFS) on the graph G, color

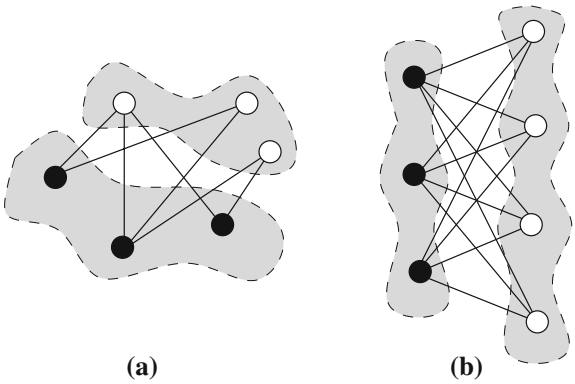

Fig. 2.8 Two bipartite graphs: the two independent partite sets are highlighted for each of them, one containing the *black*-colored vertices and the other containing the *white*-colored vertices

the vertices of G with two colors so that no two adjacent vertices receive the same color. We say colors of two vertices *conflict* if the vertices are adjacent and receive the same color. If a conflict-free coloring can be done by BFS, then G is bipartite, otherwise not.

Let G be a bipartite graph with the two independent sets V_1 and V_2. We call G a *complete bipartite graph* if for each vertex $u \in V_1$ and each vertex $v \in V_2$, there is an edge (u, v) in G. Figure 2.8(b) illustrates a complete bipartite graph where the two partite sets contains 3 and 4 vertices, respectively. This graph is denoted by $K_{3,4}$. In general, a complete bipartite graph is denoted by $K_{m,n}$ if its two partite sets contain m and n vertices, respectively. One can easily see that $K_{m,n}$ contains $m \times n$ edges.

2.4.4 Path Graphs

A *path graph* is a graph G that contains a list of vertices $v_1, v_2, \ldots, v_p$ of G such that for $1 \leq i \leq p-1$, there is an edge (v_i, v_{i+1}) in G and these are the only edges in G. The two vertices v_1 and v_p are called the *end-vertices* of G. Figure 2.9(a) illustrates a path graph with six vertices. A path graph with n vertices is denoted by P_n. Note that the degree of each vertex of a path graph is two except for the two end-vertices, both of which have degree one.

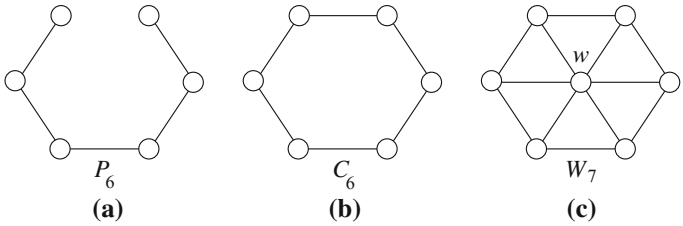

Fig. 2.9 **(a)** P_6, **(b)** C_6, and **(c)** W_7

2.4.5 Cycle Graphs

A *cycle graph* is one that is obtained by joining the two end-vertices of a path graph. Thus, the degree of each vertex of a cycle graph is two. Figure 2.9(b) illustrates a cycle graph with six vertices. A cycle graph with n vertices is often denoted by C_n.

2.4.6 Wheel Graphs

A *wheel graph* with n vertices, denoted by W_n, is obtained from a cycle graph C_{n-1} with $n-1$ vertices by adding a new vertex w and joining an edge from w to each vertex of C_{n-1}. Figure 2.9(c) illustrates a wheel graph with seven vertices.

2.5 Operations on Graphs

We are already familiar with two operations on graphs, namely deletion of vertices and deletion of edges. In this section, we see some other operations on graphs. Since a graph $G = (V, E)$ is defined as a tuple of two sets; the vertex set and the edge set, some operations on sets can naturally be extended to graphs. We first show some examples of such set operations on graphs. Later in this section, we also define some other operations on graphs.

2.5.1 Union and Intersection of Graphs

Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two graphs. The *union* of G_1 and G_2, denoted by $G_1 \cup G_2$, is another graph $G_3 = (V_3, E_3)$, whose vertex set $V_3 = V_1 \cup V_2$ and edge set $E_3 = E_1 \cup E_2$.

Similarly, the *intersection* of G_1 and G_2, denoted by $G_1 \cap G_2$, is another graph $G_4 = (V_4, E_4)$, whose vertex set $V_4 = V_1 \cap V_2$ and edge set $E_4 = E_1 \cap E_2$.

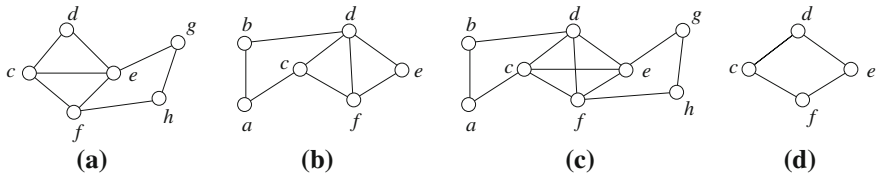

Fig. 2.10 **(a)** G_1, **(b)** G_2, **(c)** $G_1 \cup G_2$, and **(d)** $G_1 \cap G_2$

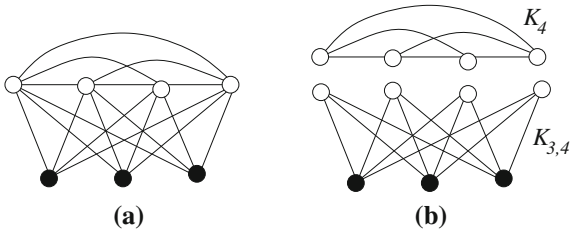

Fig. 2.11 **(a)** The graph representing handshakes, **(b)** K_4 and $K_{3,4}$

Figures 2.10(a) and (b) show two graphs G_1 and G_2, and Figs. 2.10(c) and (d) illustrate their union and intersection, respectively.

Clearly, we can define the union and intersection of more than two graphs in a similar way. These operations on graphs can be used to solve many problems very easily. We now present such an application of these operations on two graphs [2].

Suppose there are $h + g$ people in a party; h of them are hosts and g of them are guests. Each person shakes hands with each other except that no host shakes hands with any other host. The problem is to find the total number of handshakes. As usual, we transform the scenario into a graph problem as follows. We form a graph with $h + g$ vertices; h of them are black vertices, representing the hosts and the other g vertices are white, representing the guests. The edges of the graph represent the handshakes. Thus, there is an edge between every pair of vertices except for that there is no edge between any pair of black vertices. Thus, the problem now is to count the number of edges in the graph thus formed. The graph is illustrated for $h = 3$ and $g = 4$ in Fig. 2.11(a).

To solve the problem, we note that the graph can be thought of as a union of two graphs: a complete graph K_g and a complete bipartite graph $K_{h,g}$ as illustrated in Fig. 2.11(b). Since there is no common edge between the two graphs, their intersection contains no edges. Thus, the total number of edges in the graph (i.e., the total number of handshakes in the party) is $n(n-1)/2 + m \times n$.

2.5.2 Complement of a Graph

The *complement* of a graph $G = (V, E)$ is another graph $\overline{G} = (V, \overline{E})$ with the same vertex set such that for any pair of distinct vertices $u, v \in V$, $(u, v) \in \overline{E}$

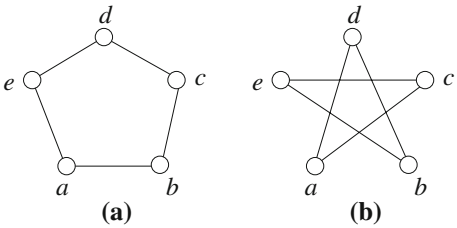

Fig. 2.12 **(a)** A graph G, and **(b)** the complement $\overline{G}$ of G

if and only if $(u, v) \notin E$. We often denote the complement of a graph G by $\overline{G}$. Figure 2.12(b) illustrates the complement of the graph in Fig. 2.12(a). A null graph is the complement of the complete graph with the same number of vertices and vice versa. The following lemma is an interesting observation in terms of the complement of a graph.

Lemma 2.5.1 *For any graph of six vertices, G or $\overline{G}$ contains a triangle.*

Proof Let G be a graph of six vertices, and let v be a vertex of G. Since the total number of neighbors of v in G and $\overline{G}$ is five, v has at least three neighbors either in G or in $\overline{G}$ by the pigeonhole principle. Without loss of generality we can assume that v has three neighbors x, y and z in G. If any two of x, y, and z are adjacent to each other, then G contains a triangle. If no two of x, y, and z are adjacent, then x, y, and z will form a triangle in $\overline{G}$. □

2.5.3 Subdivisions

Subdividing an edge (u, v) of a graph G is the operation of deleting the edge (u, v) and adding the path u, w, v through a new vertex w of degree two. A graph G' is said to a *subdivision of a graph* G if G' can be obtained from G by successively subdividing some of the edges of G. Figure 2.13(b) illustrates a subdivision of the graph in Fig. 2.13(a).

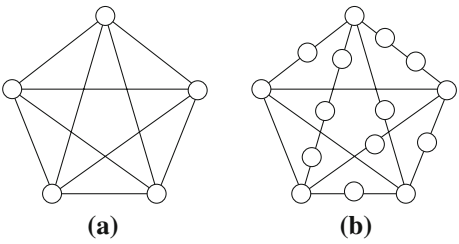

Fig. 2.13 **(a)** A graph G, and **(b)** a subdivision G' of G

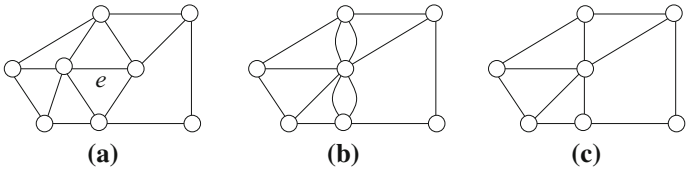

Fig. 2.14 **(a)** A graph G, **(b)** the graph obtained by contracting the edge e in G, and **(c)** the simple graph obtained by contracting the edge e in G

2.5.4 Contraction of an Edge

The *contraction* of an edge (u, v) of a graph G is the operation of deleting the edge (u, v) and identifying the two vertices u and v. Thus to contract the edge (u, v), we delete the two vertices u, v and add a new vertex w where all the edges incident to u and v in G other than the edge (u, v) are made incident to w. Figure 2.14(a) illustrates a graph G and Fig. 2.14(b) shows the new graph obtained by contracting the edge e in G. We denote by $G\backslash e$ the graph obtained from G by contracting an edge e.

After contracting an edge (u, v) of a graph, the new graph may contain multi-edges. For example, the graph in Fig. 2.14(b) contains some multi-edges. When we require only a simple graph for our consideration, then we often take a simple graph by replacing each set of multi-edges by a single edge from the multigraph. For example Fig. 2.14(c) illustrates a simple graph obtained by contracting the edge e of the graph in Fig. 2.14(a).

2.6 Graph Isomorphism

An *isomorphism* between two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ is a one-to-one correspondence between the vertices in V_1 and V_2 such that the number of edges between any two vertices in V_1 is equal to the number of edges between the corresponding two vertices in V_2. Thus, an *isomorphism* between two simple graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ is defined to be a bijection $f : V_1 \longrightarrow V_2$ such that for any two vertices u and v of G_1, $(u, v) \in E_1$ if and only if $(f(u), f(v)) \in E_2$. If there is an isomorphism between two graphs G_1 and G_2, then we say that G_1 is *isomorphic* to G_2 and write $G_1 \cong G_2$.

Figure 2.15(a) shows two graphs G_1 and G_2 that are isomorphic to each other. This isomorphism can be noticed by mapping the vertices a, b, c, d, and e of G_1 to the vertices u, w, y, v, and x of G_2, respectively. Similarly, the two graphs illustrated in Fig. 2.15(b) are also isomorphic. This isomorphic class of graphs is known as Petersen Graph. Observe the two graphs in Fig. 2.15(a). If we map the vertices a, b, c, d, and e of G_1 to the vertices u, v, w, x, and y of G_2, respectively, then the two graphs become complement to each other. A graph which is isomorphic to

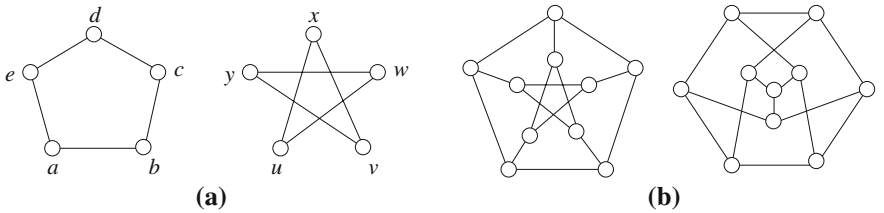

Fig. 2.15 Two pairs of isomorphic graphs

its complement is called a *self-complimentary*. The graphs in Fig. 2.15(a) (C_5) are self-complimentary. Similarly, P_4 is also self-complimentary.

In order to decide whether two graphs G_1 and G_2 are isomorphic or not, a trivial approach would take all the possible permutations of the vertices of G_2 to check whether any of these permutations induces an isomorphism. Clearly, this approach takes exponential time on the number of vertices. Unfortunately, there is no known polynomial-time algorithm to examine whether two graphs are isomorphic or not; neither there is any proof of the claim that there cannot be any such polynomial-time algorithm. Thus, it is an interesting open problem to prove whether the existence or inexistence of a polynomial-time algorithm to test two graphs to be isomorphic.

We end this section with the following lemma, which shows that isomorphism between graphs gives equivalence classes under the isomorphism relation [3].

Lemma 2.6.1 *The isomorphism relation is an equivalence relation on the set of graphs.*

Proof The reflexivity property of the isomorphism is trivial since for any graph $G = (V, E)$, the bijection $f : V \longrightarrow V$ that maps every vertex $v \in V$ to itself gives an isomorphism. Again if $f : V_1 \longrightarrow V_2$ is an isomorphism from a graph $G_1 = (V_1, E_1)$ to another graph $G_2 = (V_2, E_2)$, then f^{-1} defines an isomorphism from G_2 to G_1, because $(u, v) \in E_1$ if and only if $(f(u), f(v)) \in E_2$ implies that $(x, y) \in E_2$ if an only if $(f^{-1}(x), f^{-1}(y)) \in E_1$. Thus, isomorphism relation is also symmetric on the set of graphs. We now prove the transitivity property of isomorphism. Suppose $f : V_1 \longrightarrow V_2$ and $g : V_2 \longrightarrow V_3$ define isomorphisms from G_1 to G_2 and from G_2 to G_3, where $G_1 = (V_1, E_1)$, $G_2 = (V_2, E_2)$ and $G_3 = (V_3, E_3)$ are three graphs. Hence, $(u, v) \in E_1$ if and only if $(f(u), f(v)) \in E_2$ and $(x, y) \in E_2$ if and only if $(g(x), g(y)) \in E_3$. Therefore, $(u, v) \in E_1$ if and only if $(g(f(u)), g(f(v))) \in E_3$. Thus $g \circ f$ defines an isomorphism from G_1 to G_3. □

For many problems, we often require only structural properties of a graph. For such cases, the "labels" of the vertices of the graph are often unnecessary, and we informally use the term "unlabeled graphs" to denote an isomorphic class of graphs.

2.7 Degree Sequence

The *degree sequence* of a graph is the list of vertex degrees. The degree sequence of a graph is usually written in nonincreasing order as $d_1 \geq \cdots \geq d_n$. The set of distinct nonnegative integers occurring in a degree sequence of a graph is called its *degree set.* For the graph in Fig. 2.16(a) the degree sequence is 5, 4, 3, 3, 3, 2 and the degree set is {2, 3, 4, 5}. Two graphs with the same degree sequence are said to be *degree equivalent.* The two graphs in Figs. 2.16(a) and (b) are degree equivalent.

Every graph has a degree sequence. But does there exist a graph for a given sequence of nonincreasing nonnegative integers? To answer this question, a trivial necessary condition comes from degree-sum formula. That is, the sum of the integers in the sequence must be even. It is interesting that this trivial necessary condition is also sufficient, as can be seen from the following construction. Assume that the sum of the integers in the sequence is even. Since the sum is even, the number of odd values in the sequence is even. First form an arbitrary pairing of the vertices with odd degrees, and add an edge between the two vertices of each pair. Then the remaining degree needed to each vertex is even and nonnegative. Construct loops in each vertex to fulfill these degree requirements of each vertex. The construction of a graph for the sequence 5, 4, 3, 2, 1, 1 is illustrated in Fig. 2.17.

Allowing loops makes the realization of a degree sequence easy. If we do not allow loops and multiple edges, some sequences are not realizable even if their sum is even. For example, it is not possible to realize the sequence 2, 2, 0 if we do not allow loops and multiple edges. We call a degree sequence a *graphic sequence* if it realizes a simple graph. Thus, the degree sequence 2, 2, 0 is not a graphic sequence. Figure 2.18(a) shows a simple graph G with the graphic sequence 3, 3, 3, 2, 1 with $\Delta = 3$. We can construct a simple graph G' from G by adding a new vertex x and adding edges x to each of the $\Delta + 1$ vertices of largest degrees in G, as illustrated in Fig. 2.18(b), where the degree sequence of G' is 4, 4, 4, 4, 3, 1. Now consider the

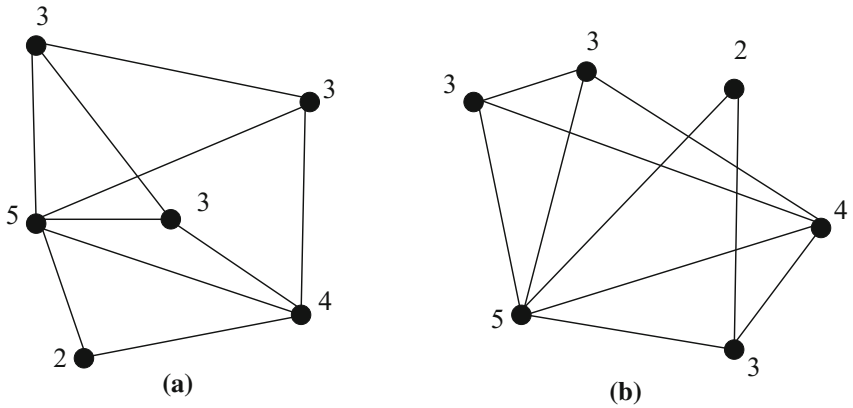

Fig. 2.16 Illustration for degree sequence

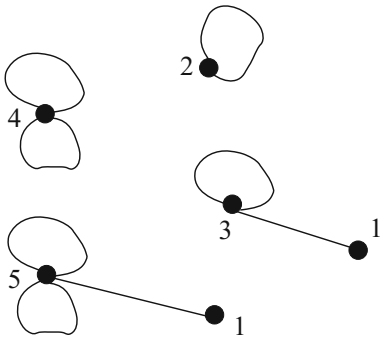

Fig. 2.17 Illustration of a construction for the sequence 5, 4, 3, 2, 1, 1

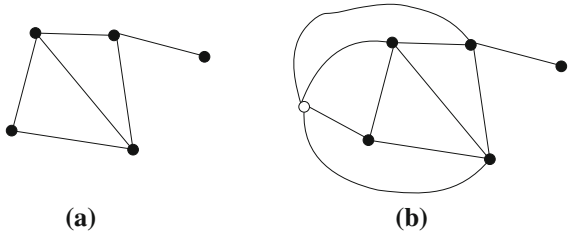

Fig. 2.18 **(a)** A graph G with the degree sequence 3, 3, 3, 2, 1 and **(b)** a graph G' with the degree sequence 4, 4, 4, 4, 3, 1

following scenario. Assume that we have a degree sequence $d = d_1, d_2, \ldots, d_n$. We obtain a degree sequence d' from d by deleting d_1 and subtracting 1 from each of $d_2, \ldots, d_{d_1+1}$. Then clearly d is graphic if d' is graphic, since we can construct a simple graph realizing d from a simple graph realizing d' by adding a new vertex and d_1 edges. The reverse of the implication above is also hold as Havel [4] and Hakimi [5] showed that d is graphic if and only if d' is graphic. Therefore, based on the construction above, we can easily develop a recursive algorithm to check whether a degree sequence is graphic or not. Note that $d_1 = 0$ is the only 1-element graphic sequence. Steps of an recursive algorithm for testing a graphic sequence are illustrated in Fig. 2.19. Note that when we obtain d' from d, we rearrange d' in nonincreasing

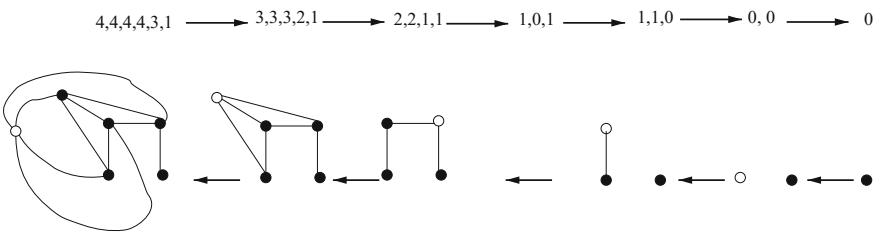

Fig. 2.19 Illustration for testing the realization of a graphic sequence

order if d' is not in nonincreasing order. For example, observe $1, 0, 1 \rightarrow 1, 1, 0$ in Fig. 2.19.

2.8 Data Structures and Graph Representation

We are already accustomed with one way of representing a graph, i.e., by a diagram where each vertex is represented by a point or a small circle and each edge is represented by a straight-line between the end-vertices. This graphical representation is convenient for the visualization of a graph, yet it is unsuitable if we want to store a graph in a computer. However, there are other ways of representing a graph. In this section, we first give a very brief account of some basic data structures and then we show three methods for representing a graph in a computer.

A vector or a set of variables is usually stored as a (one-dimensional) array and a matrix is stored as a two-dimensional array. The main feature of an array is that the location of an entry can be uniquely determined by its index in the array and the entry can be accessed in constant time. A list is a data structure which consists of homogeneous records, linked together in a linear fashion. Each record contains one or more items of data and one or more pointers. In a *singly linked lists*, each record has a single forwarding pointer indicating the address of the memory cell of the next record. In a *doubly linked list*, each record has forward and backward pointers indicating the address of the memory cells of the next and the previous records, respectively.

We now present three different representations of graphs.

2.8.1 Adjacency Matrix

Let G be a graph with the vertex set $V = \{v_1, v_2, \ldots, v_n\}$ and the edge set $E = \{e_1, e_2, \ldots, e_m\}$. The adjacency matrix $A(G)$ of G is an $n \times n$ matrix $A(G) = [a_{ij}]$ in which a_{ij} is the number of edges between the two vertices v_i and v_j. Figure 2.20(a) shows a graph and Fig. 2.20(b) illustrates its adjacency matrix. Note that an adjacency matrix of a graph is always symmetric.

If G is a simple graph, then each entry of $A(G)$ is either a zero or one and the main diagonal of the matrix contains only zeros. Figure 2.21(b) illustrates the adjacency matrix of the simple graph in Fig. 2.21(a).

An adjacency matrix uses $O(n^2)$ space to represent a graph of n vertices. It is not economical when the number of edges in the graph is much less than the maximum possible $n(n-1)/2$. If $A(G)$ is stored as a two-dimensional array, then checking whether $(v_i, v_j) \in E$ for a pair of vertices v_i and v_j or deleting an edge (v_i, v_j) from G requires only constant time. However, scanning all the neighbors of a vertex v requires n steps even if $deg(v)$ is much less than n.

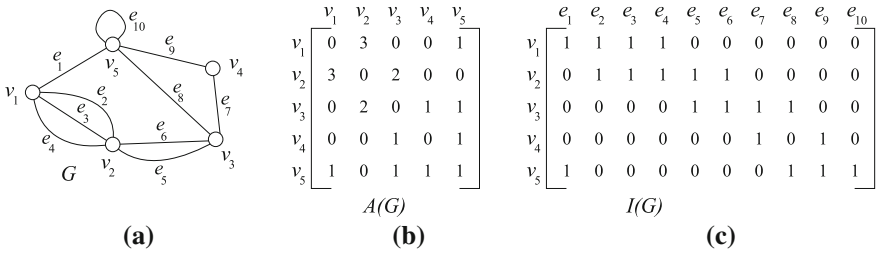

Fig. 2.20 **(a)** A graph G, **(b)** adjacency matrix representation of G, and **(c)** incidence matrix representation of G

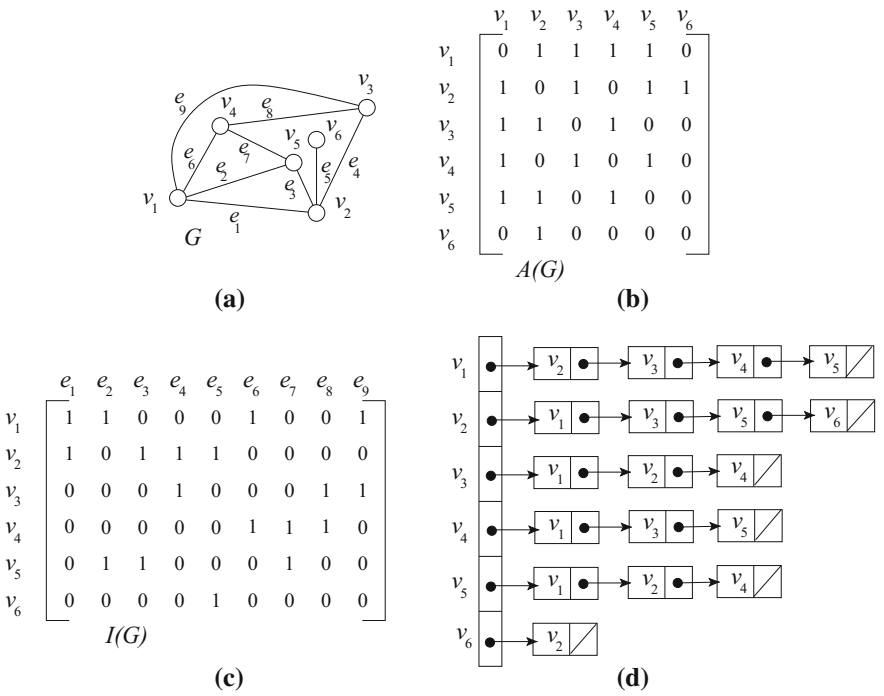

Fig. 2.21 **(a)** A simple graph G, **(b)** adjacency matrix representation of G, **(c)** incidence matrix representation of G, and **(d)** adjacency list representation G

2.8.2 *Incidence Matrix*

Let G be a graph with the vertex set $V = \{v_1, v_2, \ldots, v_n\}$ and the edge set $E = \{e_1, e_2, \ldots, e_m\}$. The incidence matrix $I(G)$ of G is an $n \times m$ matrix $M(G) = [m_{ij}]$ in which m_{ij} is 1 if the vertex v_i is adjacent to the edge e_j, and 0 otherwise. Figure 2.20(c) illustrates the incidence matrix of the graph in Fig. 2.20(a). Similarly, Fig. 2.21(c) illustrates the incidence matrix of the graph in Fig. 2.21(a).

The space requirement for the incidence matrix is $n \times m$. For a graph which contains much more number of edges compared to n, this requirement is much higher than the adjacency matrix. Scanning all the neighbors of the graph also takes n steps even if $deg(v)$ is much less than n. However, this representation is helpful for the query to know whether a vertex is incident to an edge or not and this query can be responded in constant time from this representation.

If a graph G is a simple graph, then there is another representation of G, called the adjacency lists.

2.8.3 Adjacency List

Let G be a graph with the vertex set $V = \{v_1, v_2, \ldots, v_n\}$ and the edge set $E = \{e_1, e_2, \ldots, e_m\}$. The adjacency lists $Adj(G)$ of G is an array of n lists, where for each vertex v of G, there is a list corresponding to v, which contains a record for each neighbor of v. Figure 2.21(d) illustrates the adjacency lists of the graph in Fig. 2.21(a).

The space requirement for the adjacency lists is $\sum_{v \in V}(1 + deg(v)) = O(n + m)$. Thus, this representation is much more economical than the adjacency matrix and the incidence matrix, particularly if the number of edge in the graph is much less than $n(n-1)/2$. Scanning the neighbors of a vertex v takes $O(deg(v))$ steps only but checking whether $(v_i, v_j) \in E$ requires $O(deg(v_i))$ steps for a pair of vertices v_i and v_j of G.

Bibliographic Notes

The books [3, 6–9] were used in preparing this chapter.

Exercises

1. Show that every regular graph with an odd degree has an even number of vertices.
2. Construct the complement of $K_{3,3}$, W_5, and C_5.
3. Can you construct a disconnected graph G of two or more vertices such that $\overline{G}$ is also disconnected. Give a proof supporting your answer.
4. Give two examples of self-complementary graphs.
5. What is the necessary and sufficient condition for $K_{m,n}$ to be a regular graph?
6. Is there a simple graph of n vertices such that the vertices all have distinct degrees? Give a proof supporting your answer.
7. Draw the graph $G = (V, E)$ with vertex set $V = \{a, b, c, d, e, f, g, h\}$ and edge set $\{(a, b), (a, e), (b, c), (b, d), (c, d), (c, g), (d, e)(e, f), (f, g), (f, h), (g, h)\}$. Draw $G - (d, e)$. Draw the subgraph of G induced by $\{c, d, e, f\}$. Contract the edge (d, e) from G.
8. Show that two graphs are isomorphic if and only if their complements are isomorphic.

References

1. Karim, M.R., Rahman, M.S.: On a class of planar graphs with straight-line grid drawings on linear area. J. Graph Algorithms Appl. **13**(2), 153–177 (2009)
2. Koshy, T.: Discrete Mathematics with Applications. Elsevier, Amsterdam (2004)
3. West, D.B.: Introduction to Graph Theory, 2nd edn. Prentice Hall, New Jersey (2001)
4. Havel, V.: A remark on the existence of finite graphs [Czec] Casopis. Pest. Mat. **80**, 477–480 (1955)
5. Hakimi, S.L.: On realizability of a set of integers as degrees of vertices of a linear graph. SIAM J. Appl. Math. **10**, 496–506 (1962)
6. Nishizeki, T., Rahman, M.S.: Planar graph drawing. World Scientific, Singapore (2004)
7. Wilson, R.J.: Introduction to Graph Theory, 4th edn. Longman, London (1996)
8. Clark, J., Holton, D.A.: A First Look at Graph Theory. World Scientific, Singapore (1991)
9. Pirzada, S.: An Introduction to Graph Theory. University Press, India (2009)

Chapter 3
Paths, Cycles, and Connectivity

In this chapter, we study some important fundamental concepts of graph theory. In Section 3.1 we start with the definitions of walks, trails, paths, and cycles. The well-known Eulerian graphs and Hamiltonian graphs are studied in Sections 3.2 and 3.3, respectively. In Section 3.4, we study the concepts of connectivity and connectivity-driven graph decompositions.

3.1 Walks, Trails, Paths, and Cycles

Let G be a graph. A *walk* in G is a nonempty list $W = v_0, e_1, v_1, \ldots, v_{f-1}, e_f, v_f$, whose elements are alternately vertices and edges of G where for $1 \leq i \leq f$, the edge e_i has end vertices v_{i-1} and v_i. The vertices v_0 and v_f are called the *end vertices* of W. If the end vertices of a walk W of a graph G are u and v respectively, W is also called an u, v-*walk* in G. For example in the graph G of Fig. 3.1, $W = a, (a, i), i, (i, h), h, (h, c), c, (c, b), b$ is a walk. W is also an a, b-walk in G since the end vertices of W are a and b. Often for convenience, a walk in a simple graph is represented by the sequence of its vertices only. Since the graph G of Fig. 3.1 is simple, the walk W can also be represented by $W = a, i, h, c, b$.

A *trail* of a graph G is a walk in G with no repeated edges. That is, in a trail an edge cannot appear more than once. In Fig. 3.1, the walk $W_1 = a, (a, i), i, (i, h), h, (h, c), c, (c, b), b$ is a trail but the walk $W_2 = a, (a, i), i, (i, h), h, (h, c), c, (c, b), b, (b, a), a, (a, i)$ is not a trail since the edge (a, i) has appeared in W_2 more than once. A trail is called a *circuit* when the two end vertices of the trail are the same.

A *path* is a walk with no repeated vertex (except end vertices). When the end vertices repeat, then it is called a closed path. In Fig. 3.1, $a, (a, i), i, (i, h), h, (h, c), c, (c, b), b, (b, a), a$ is a closed path. A closed path is called a *cycle.* A u, v-path is a path whose end vertices are u and v. A vertex on a path P which is not an end vertex of P is called an *internal vertex* of P. The *length* of a walk, a trail, a path, or a cycle

M.S. Rahman, *Basic Graph Theory*, Undergraduate Topics in Computer Science,
DOI 10.1007/978-3-319-49475-3_3

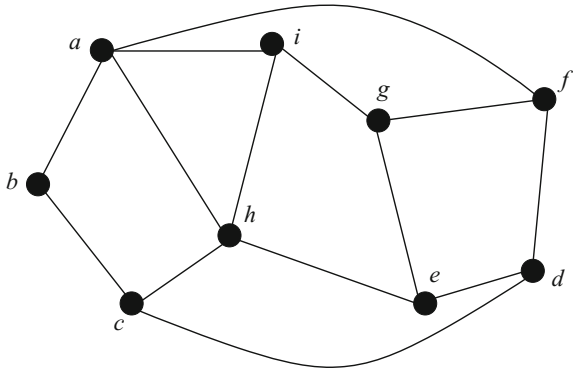

Fig. 3.1 Walks, trails, and paths

is its number of edges. Thus a path of n vertices has length $n - 1$, and a cycle of n vertices has length n.

We now have the following lemma.

Lemma 3.1.1 *Every u, v-walk contains a u, v-path.*

Proof We prove the claim by induction on length l of a u, v-walk W. If $l = 0$, then W contains single vertex. It is also a path of single vertex. Thus the basis is true.

Assume that $l \geq 1$ and the claim is true for walks of length less than l.

If W has no repeated vertex, then its vertices and edges form a u, v-path. We thus assume that W contains a repeated vertex w. We obtain a walk W' by deleting from W the vertices and edges between appearances of w and one copy of w. Clearly the walk W' is contained in W and has length less than l. By induction hypothesis W' contains a u, v-path P. Clearly this path P is contained in W. □

A graph G is *connected* if there is a path between each pair of vertices in G. Otherwise, G is called a *disconnected graph.* A *maximal connected subgraph* of G is a subgraph that is connected and is not contained in any other connected subgraph of G. A maximal connected subgraph of G is called a *connected component* of G. The graph in Fig. 3.2(a) is connected since there is a path between every pair of vertices. The graph in Fig. 3.2(b) is disconnected since there is no path between the vertices a and h. The graph in Fig. 3.2(a) has one connected component whereas the graph in Fig. 3.2(b) has two connected components. Sometimes we call a connected component simply a *component* if there is no confusion. In the next two lemmas we see relationships among the number of vertices, the number of edges, and the number of connected components of a graph.

Lemma 3.1.2 *Every graph with n vertices and m edges has at least $n - m$ connected components.*

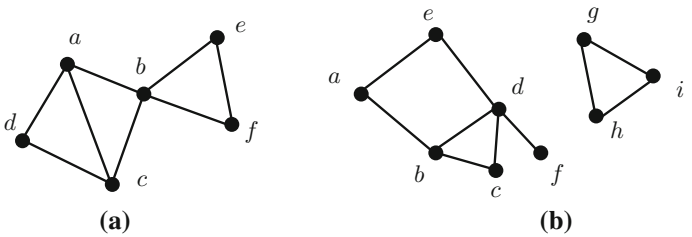

Fig. 3.2 **(a)** A connected graph and **(b)** a disconnected graph

Proof An n-vertex graph with no edges has n components. Adding an edge decreases the number of components by 0 or 1. Thus after adding m edges the number of components is still at least $n - m$. □

Lemma 3.1.3 *Let G be a simple graph of n vertices. If G has exactly k components, then the number m of edges of G satisfies*

$$n - k \leq m \leq (n - k)(n - k + 1)/2.$$

Proof We first prove the lower bound $m \geq n - k$ by induction on the number of edges of G. Let m_0 be the fewest possible edges of G with k components and G has exactly m_0 edges. If $m_0 = 0$ then G is a null graph with $n = k$ and the bound trivially holds. We now assume that $m_0 \geq 1$ and the claim holds for any graph with less than m_0 edges. If we delete any edge from G, the number of components must increase. Let G' be the graph obtained from G by deleting an edge from G. Then G' has n vertices, $k + 1$ components and $m_0 - 1$ edges. By induction hypothesis $m_0 - 1 \geq n - (k + 1)$. This implies $m_0 \geq n - k$.

We now prove the upper bound. We can assume that each component of G has the maximum number of edges, that is, each component is a complete graph. To observe an interesting scenario, assume that G has two components H_i and H_j with n_i and n_j vertices, respectively, such that $n_i \geq n_j > 1$. If we replace H_i and H_j by complete graphs of $n_i + 1$ and $n_j - 1$ vertices, then the total number of vertices remains unchanged, and the number of edges is increased by

$$\{(n_i + 1)n_i - n_i(n_i - 1)\}/2 - \{n_j(n_j - 1) - (n_j - 1)(n_j - 2)\}/2 = n_i - n_j + 1.$$

It follows that, in order to attain the maximum number of edges, G must consist of a complete graph of $n - k + 1$ vertices and $k - 1$ isolated vertices. Thus $m \leq (n - k)(n - k + 1)/2$. □

A *cut-edge* of a graph is an edge whose deletion increases the number of components. The edge (d, f) is a cut-edge in the graph in Fig. 3.2(b).

Lemma 3.1.4 *An edge is a cut-edge if and only if it belongs to no cycle.*

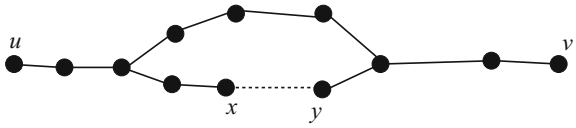

Fig. 3.3 Illustration for the proof of Lemma 3.1.4

Proof Let (x, y) be an edge in a graph G and let H be the component containing (x, y). Since deletion of (x, y) affects no other component, it is sufficient to prove that $H - (x, y)$ is connected if and only if (x, y) belongs to a cycle.

Assume that $H - (x, y)$ is connected. Then $H - (x, y)$ contains an x, y-path P, and hence $P + (x, y)$ is a cycle in H.

We now assume that (x, y) lies on a cycle C. Let u and v be any two vertices in H. Since H is connected, H has a u, v-path P. If P does not contain (x, y), then P exists in $H - (x, y)$. We thus assume that P contains (x, y). Without loss of generality, we assume that x is between u and y on P, as illustrated in Fig. 3.3. Since $H - (x, y)$ contains a u, x-path along P, and x, y-path along C, and a y, v path along P, $H - (x, y)$ contains a u, v-walk. Hence by Lemma 3.1.1 $H - (x, y)$ contains a u, v-path. We can prove this for all $u, v \in V$, and hence $H - (x, y)$ is connected. □

A cycle is *odd* if its length is odd, and a cycle is *even* if its length is even. Bipartite graphs can be characterized in terms of odd cycles as follows.

Theorem 3.1.5 *A graph G is bipartite if and only if G does not contain any odd cycle.*

Proof Necessity Assume that G is bipartite with partite sets V_1 and V_2. Let $x_1, x_2, \ldots, x_l, x_1$ be a cycle in G of length l. Without loss of generality, assume that $x_1 \in V_1$. Then $x_2 \in V_2$ and also $x_l \in V_2$. One can observe that $x_i \in V_1$ if i is odd, and $x_i \in V_2$ if i is even. Since $x_l \in V_2$, l is even.

Sufficiency Assume that G does not contain an odd cycle. One can easily observe that a graph G is bipartite if and only if each connected component of G is bipartite. We thus assume that G is connected. Let x be an arbitrary vertex of G. We construct a set V_1 of vertices such that $V_1 = \{y | d(x, y) \text{ is odd}\}$. We take $V_2 = V - V_1$. Then both V_1 and V_2 are independent sets, otherwise G would have an odd cycle, a contradiction. Hence G is bipartite. □

3.2 Eulerian Graphs

Remember the Königsberg seven-bridge problem introduced at the very beginning of the introduction chapter. In this section, we formally deal with the problem. A trail in a connected graph is an *Eulerian trail* if it contains every edge exactly once.

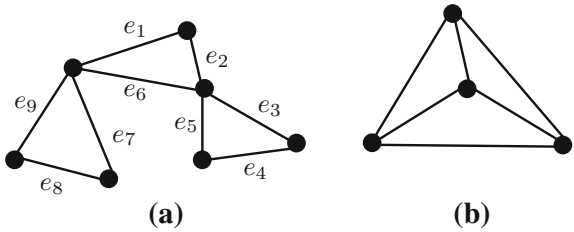

Fig. 3.4 **(a)** An Eulerian graph and **(b)** a graph which is not Eulerian

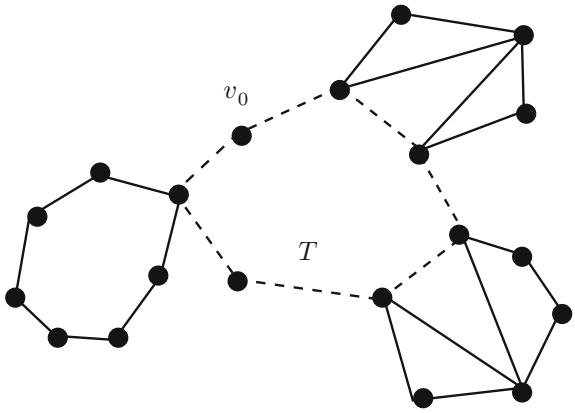

Fig. 3.5 Illustration for the proof of sufficiency of Theorem 3.2.1

A circuit in a connected graph is an *Eulerian circuit* if it contains every edge of the graph. A connected graph with an Eulerian circuit is an *Eulerian graph.* The graph in Fig. 3.4(a) is Eulerian since it has a Eulerian circuit $e_1, e_2, e_3, \ldots, e_9$, whereas the graph in Fig. 3.4(b) is not Eulerian since it is not possible to find an Eulerian circuit in it. Note that Euler showed the impossibility of a desired walk through Königsberg bridges by showing that there is no Eulerian circuit in the obtained graph in Fig. 1.1. In fact, Euler in 1736 proved the impossibility by proving a general claim as in the following theorem.[1]

Theorem 3.2.1 *A connected graph G is Eulerian if and only if every vertex of G has even degree.*

Proof Necessity Suppose G is Eulerian. We will show that every vertex of G has even degree. Since G is Eulerian, G contains an Eulerian circuit, say, $v_0, e_1, v_1, e_2, \ldots, e_n, v_0$. Both edges e_1 and e_n contribute 1 to the degree of v_0; so degree of v_0 is at least 2. Each time the circuit passes through a vertex (including v_0), the degree of the vertex is increased by two. Therefore, the degree of every vertex including v_0 is an even integer.

[1] A complete proof for the sufficiency of Theorem 3.2.1 was missing in Euler's paper, and the sufficiency was established by Hierholzer in 1873 [1].

Sufficiency Assume that every vertex of G has even degree. We will show that G is Eulerian, that is, G contains an Eulerian circuit. We give a constructive proof.

Let v_0 be an arbitrary vertex. Starting from v_0, we find a maximal trail T. Since every vertex of G has even degree, T will be a closed trail. If T contains all the edges of G, then G is Eulerian. Otherwise, let G' be the graph obtained from G by deleting all the edges of T (see Fig. 3.5). Since T has even degree at every vertex, every vertex of G' has even degree. Since G is connected, some edge e of G' is incident to a vertex v of T. We find a maximal trail T' in G' starting from v along e. Combining T and T' we get a longer circuit. Let $T = T \cup T'$ and we repeat the argument. Since $E(G)$ is finite, this process must end, and it can only end by constructing an Eulerian circuit. □

We say a graph has a *cycle decomposition* if the graph can be expressed as a union of edge-disjoint cycles. From the proof of sufficiency of Theorem 3.2.1, it is evident that an Eulerian graph has cycle decomposition. In fact this condition is also sufficient, and hence one can easily prove the following lemma.

Lemma 3.2.2 *A connected graph G is Eulerian if and only if G has a cycle decomposition.*

We can transform a non-Eulerian graph into an Eulerian graph by adding edges. It is not difficult to count the number of edges which is required to add to make a non-Eulerian graph an Eulerian graph.

3.3 Hamiltonian Graphs

An Eulerian circuit visits each edge exactly once, but may visit some vertices more than once. In this section, we consider a round trip through a given graph G such that every vertex is visited exactly once. The original question was posed by a well-known Irish mathematician, Sir William Rowan Hamilton.

Let G be a graph. A path in G that includes every vertex of G is called a *Hamiltonian path* of G. A cycle in G that includes every vertex in G is called a *Hamiltonian cycle of G*. If G contains a Hamiltonian cycle, then G is called a *Hamiltonian Graph*.

Every Hamiltonian cycle of a Hamiltonian graph of n vertices has exactly n vertices and n edges. If the graph is not a cycle, some edges of G are not included in a Hamiltonian Cycle.

Not all graphs are Hamiltonian. For example, the graph in Fig. 3.6(a) is Hamiltonian, since a, b, c, d, a is a Hamiltonian cycle. On the other hand, the graph in Fig. 3.6(b) is not Hamiltonian, since there is no Hamiltonian cycle in this graph. Note that the path a, b, c, d, e is a Hamiltonian path in the graph in Fig. 3.6(b). Thus a natural question is: What is the necessary and sufficient condition for a graph to be Hamiltonian? Clearly a Hamiltonian graph must be connected and cannot be acyclic, but these are not sufficient. The graph in Fig. 3.6(b) is connected and not acyclic but it is not Hamiltonian. The following lemma gives a necessary condition which is not also sufficient.

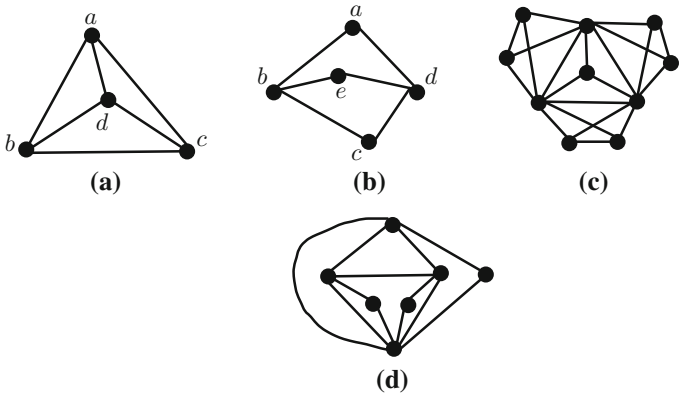

Fig. 3.6 **(a)** A Hamiltonian graph, and **(b)–(d)** non-Hamiltonian graphs

Lemma 3.3.1 *Let G be a simple Hamiltonian graph. Then for each subset S of $V(G)$, the number of connected components in $G - S$ is at most $|S|$.*

Proof G has a Hamiltonian cycle since G is Hamiltonian. Let C be a Hamiltonian cycle in G. Since C is a cycle, the number of connected components of $C - S$ is at most $|S|$. Since C is a subgraph of G which contains all vertices of G and may not contain all edges of G, $G - S$ contains at most the number of connected components of $C - S$. Hence $G - S$ has at most $|S|$ connected components. □

If we delete vertices b and d from the graph in Fig. 3.6(b) we get three connected components, and hence the graph does not satisfy the condition in Lemma 3.3.1 and not Hamiltonian. One can observe that the graph in Fig. 3.6(c) is not Hamiltonian by Lemma 3.3.1. On the other hand, the graph in Fig. 3.6(d) is not Hamiltonian although it satisfies the condition in Lemma 3.3.1. Although no complete characterization is known for a graph to be a Hamiltonian graph, some sufficient conditions are known as in the following lemmas [2].

Lemma 3.3.2 *Let G be a simple graph of n vertices. Let u and v be two vertices in G such that $(u, v) \notin E(G)$ and $d_G(u) + d_G(v) \geq n$. Then G is Hamiltonian if and only if $G + (u, v)$ is Hamiltonian.*

Proof The necessity of the claim is trivially true since addition of an edge does not destroy the Hamiltonicity of a graph. We now prove the sufficiency.

Assume that $G + (u, v)$ is Hamiltonian and let C be a Hamiltonian cycle in $G + (u, v)$. If C does not contain the edge (u, v) then C is also a Hamiltonian cycle in G, and hence G is Hamiltonian. We thus assume that C contains the edge (u, v). Let $C = (u_0 = u), (u_1 = v), u_2, \ldots, u_n = u$ (see Fig. 3.7.) We now define two sets as follows.

$$U = \{i : (u, u_i) \in E(G), 2 \leq i \leq n - 2\}$$

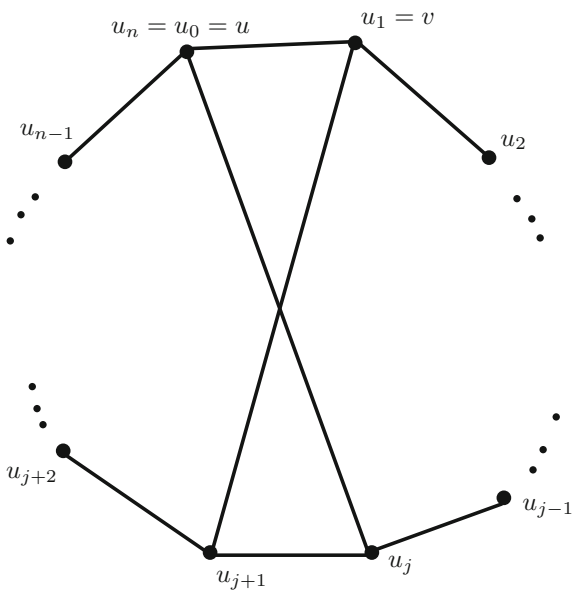

Fig. 3.7 Illustration for the proof of Lemma 3.3.2

$$V = \{i : (v, u_{i+1}) \in E(G), 2 \leq i \leq n-2\}$$

One can observe that each edge incident to u except (u, u_{n-1}) contributes an element in U and each edge incident to v except (v, u_2) contributes an element in V. Again, $(u, v) \notin E(G)$. Therefore $|U| = d_G(u) - 1$ and $|V| = d_G(v) - 1$ holds. Then

$$|U| + |V| = d_G(u) + d_G(v) - 2 \geq n - 2. \tag{3.1}$$

Since $1, n-1$ and n are not included in any of U and V, $|U \cup V| \leq n - 3$. Then

$$|U| + |V| = |U \cup V| + |U \cap V| \leq (n-3) + |U \cap V|. \tag{3.2}$$

From Eqs. 3.1 and 3.2 we get $|U \cap V| \geq 1$. Let $j \in U \cap V$. Then G has the edges (u, u_j) and (v, u_{j+1}), and we can construct a Hamiltonian cycle $C' = (u_0 = u), u_j, u_{j-1}, \ldots, (u_1 = v), u_{j+1}, u_{j+2}, \ldots, u_{n-1}, u_n = u_0$ which does not contain the edge (u, v), as illustrated in Fig. 3.7. Hence G is Hamiltonian. □

Using Lemma 3.3.2, we can prove the following result which is due to Ore [2].

Theorem 3.3.3 *Let G be a simple graph of $n \geq 3$ vertices. Then G is Hamiltonian if $d_G(u) + d_G(v) \geq n$ for every pair of nonadjacent vertices u and v in G.*

Proof Let $G = G_0$. Let u and v be a pair of nonadjacent vertices and let $G_1 = G_0 + (u, v)$. By Lemma 3.3.2, G_0 is Hamiltonian if and only if G_1 is Hamiltonian. If G_1 is a complete graph, then it is clearly Hamiltonian. Therefore, $G_0 = G$ is Hamiltonian. Otherwise, we continue to add edges in this fashion and form a sequence

$$G = G_0 \subseteq G_1 \subseteq G_2 \cdots \subseteq G_k = K_n.$$

We eventually reach the complete graph K_n. By Lemma 3.3.2 G_i is Hamiltonian if and only if G_{i+1} is Hamiltonian. Since K_n is Hamiltonian, eventually $G_0 = G$ is Hamiltonian. □

3.4 Connectivity

The *connectivity* $\kappa(G)$ of a connected graph G is the minimum number of vertices whose removal results in a disconnected graph or a single vertex graph K_1. A graph G is *k-connected* if $\kappa(G) \geq k$. A *separating set* or a *vertex cut* of a connected graph G is a set $S \subset V(G)$ such that $G - S$ has more than one component. If a vertex cut contains exactly one vertex, then we call the vertex cut a *cut vertex*. If a vertex cut in a 2-connected graph contains exactly two vertices, then we call the two vertices a *separation-pair*.

The *edge connectivity* $\kappa'(G)$ of a connected graph G is the minimum number of edges whose removal results in a disconnected graph. A graph is *k-edge-connected* if $\kappa'(G) \geq k$. A *disconnecting set* of edges in a connected graph is a set $F \subseteq E(G)$ such that $G - F$ has more than one component. If a disconnecting set contains exactly one edge, it is called a *bridge*.

For two disjoint subsets S and T of $V(G)$, we denote $[S, T]$ the set of edges which have one endpoint in S and the other in T. An *edge cut* is an edge set of the form $[S, \bar{S}]$, where S is a nonempty proper subset of $V(G)$ and $\bar{S}$ denotes $V(G) - S$.

We now explore the relationship among the connectivity $\kappa(G)$, the edge connectivity $\kappa'(G)$, and the minimum degree $\delta(G)$ of a connected simple graph G. In a cycle of three or more vertices $\kappa(G) = \kappa'(G) = \delta(G) = 2$. For complete graphs of $n \geq 1$ vertices $\kappa(G) = \kappa'(G) = \delta(G) = n - 1$. For the graph G in Fig. 3.8, $\kappa(G) = 1$,

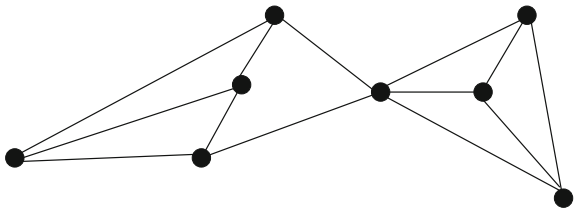

Fig. 3.8 A graph G with $\kappa(G) = 1$, $\kappa'(G) = 2$ and $\delta(G) = 3$

$\kappa'(G) = 2$ and $\delta(G) = 3$. Whitney in 1932 showed that the following relationship holds [3].

Lemma 3.4.1 *Let G be a connected simple graph. Then $\kappa(G) \leq \kappa'(G) \leq \delta(G)$.*

Proof Since the edges incident to a vertex of minimum degree form an edge cut, $\kappa'(G) \leq \delta(G)$. It is thus remained to show that $\kappa(G) \leq \kappa'(G)$. One can observe that $\kappa(G) = n(G) - 1$ if G is a complete graph; otherwise, $\kappa(G) \leq n(G) - 2$. Let $[S, \bar{S}]$ be a smallest edge cut. We now have the following two cases to consider:

Case 1: Every vertex of S is adjacent to every vertex of $\bar{S}$.

In this case the number of edges in $[S, \bar{S}]$ is $|S||\bar{S}|$. Clearly $|S||\bar{S}| \geq n(G) - 1 \geq \kappa(G)$. Hence the desired inequality $\kappa(G) \leq \kappa'(G)$ holds.

Case 2: Otherwise.

In this case, there exist a vertex $x \in S$ and a vertex $y \in \bar{S}$ such that x is not adjacent to y. Let P be the set of vertices which consists of the neighbors of x in $\bar{S}$ and Q be the set of all vertices in $S - x$ having neighbors in $\bar{S}$, as illustrated in Fig. 3.9. Then every xy-path passes through a vertex in $P \cup Q$, and hence $P \cup Q$ is a separating set of G. Let Z be the set of edges which consists of the edges from x to P and one edge from each vertex of Q to $\bar{S}$. The edges in Z are drawn by thick lines in Fig. 3.9. Then Z has exactly $|P \cup Q|$ edges. Clearly $|Z| = |P \cup Q| \leq |[S, \bar{S}]|$. Therefore $\kappa(G) \leq |P \cup Q| \leq |[S, \bar{S}]| = \kappa'(G)$. □

For connected 2-regular graphs, i.e., for cycles, connectivity, and edge connectivity are equal. This is also true for 3-regular graphs as shown in the following lemma.

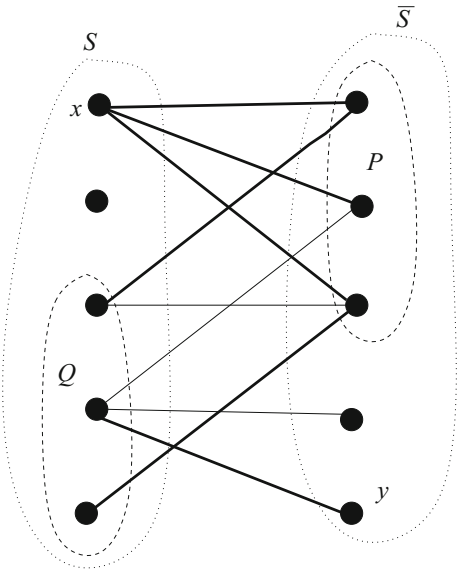

Fig. 3.9 Illustration for the proof of Lemma 3.4.1

Lemma 3.4.2 *Let G be a connected cubic graph. Then $\kappa(G) = \kappa'(G)$.*

Proof Let S be a minimum vertex cut of G, that is, $|S| = \kappa(G)$. By Lemma 3.4.1 $\kappa(G) \leq \kappa'(G)$. To prove the claim, it is thus sufficient to find an edge cut of size $|S|$ in G.

Let H_1 and H_2 be two components of $G - S$, as illustrated in Fig. 3.10. Since S is a minimum vertex cut, each $v \in S$ has a neighbor in H_1 and a neighbor in H_2. Since G is a cubic graph, v cannot have two neighbors in H_1 and two neighbors in H_2. Thus a vertex v in S is one of the three types: (i) v has exactly one neighbor in H_1 and exactly two neighbors in H_2, (ii) v has exactly two neighbors in H_1 and exactly one neighbor in H_2, and (iii) v has exactly one neighbor in H_1, exactly one neighbor in H_2. If v is a vertex of type (iii), then there is another vertex u in S of type (iii) such that there is an u, v-path not containing the vertices of H_1 and H_2. We call u the *pair* of v. We now construct an edge cut by choosing an edge for each vertex v in $|S|$ as follows. If v is of type (i), we choose the edge from v to H_1. If v is of type (ii), we choose the edge from v to H_2. If v is type (iii) then let u be the pair of v. In this case, we choose the edge from v to H_1 and the edge from u to H_1. Thus we construct an edge cut of size $|S|$. □

One can easily prove the following lemma.

Lemma 3.4.3 *Let G be a k-connected graph, and let G' be obtained from G by adding a new vertex x with at least k neighbors of G. Then G' is also k-connected.*

3.4.1 Connected Separable Graphs

A connected graph is *separable* if G has at least one cut vertex, otherwise G is nonseparable. Thus a nonseparable graph is 2-connected. However, K_1 and K_2 are considered nonseparable. A maximal nonseparable connected subgraph of G is called

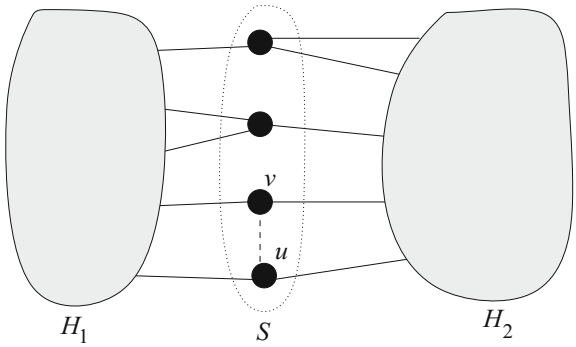

Fig. 3.10 Illustration for S, H_1 and H_2 in the proof of Lemma 3.4.2

a *block* of G. Thus a block of a connected graph G of $n \geq 2$ vertices is either a biconnected component or a bridge of G. We now have the following lemma.

Lemma 3.4.4 *Let G be a connected graph and let B_1 and B_2 be two distinct blocks of G. Then B_1 and B_2 can have at most one common vertex.*

The proof of Lemma 3.4.4 is left as an exercise.

3.4.2 Block-Cutvertex Tree

The blocks and cut vertices in G can be represented by a tree T, called the *BC-tree* of G. In T each block is represented by a B-node and each cut vertex of G is represented by a C-node. The graph in Fig. 3.11(a) has the blocks $B_1, B_2, \ldots, B_9$ depicted in Fig. 3.11(b). The BC-tree T of the plane graph G in Fig. 3.11(a) is depicted in Fig. 3.11(c), where each B-node is represented by a rectangle and each C-node is represented by a circle.

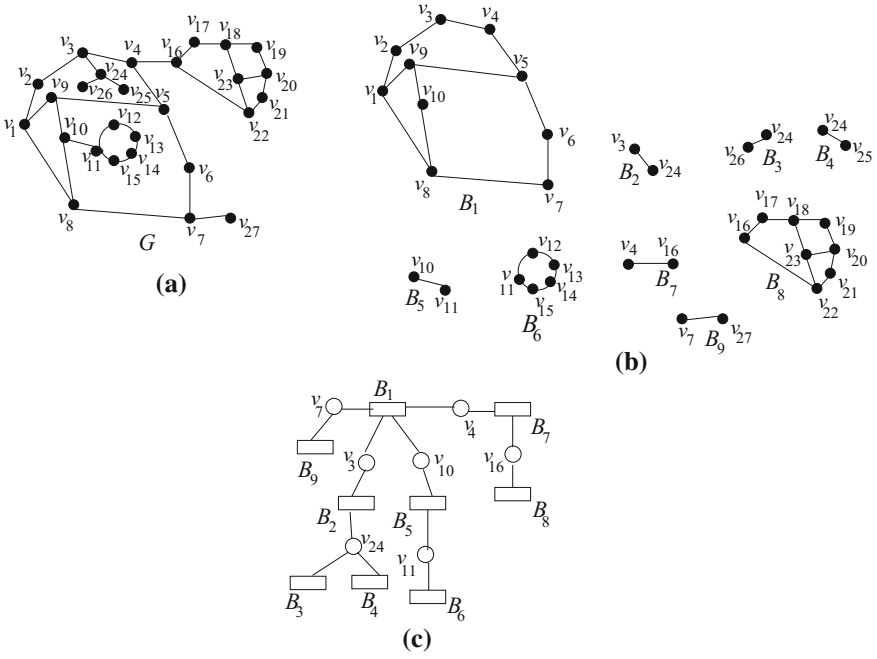

Fig. 3.11 **(a)** A connected graph G, **(b)** blocks of G, and **(c)** BC-tree T

3.4.3 2-Connected Graphs

The reliability of a computer network can be increased by providing alternative paths between workstations. In fact the connectivity of a graph is a measure of number of alternative paths. If the graph is 1-connected then there is a path between any two workstations. We will show that there are two alternative paths between any two workstations in a 2-connected network, as in Theorem 3.4.5 below due to Whitney [3]. Two paths P_1 and P_2 with the same end vertices are *internally disjoint* if P_1 and P_2 do not share any internal vertex.

Theorem 3.4.5 *A graph G of three or more vertices is 2-connected if and only if there are two internally disjoint paths between every pair of vertices in G.*

Proof Sufficiency Assume that there are two internally disjoint paths between every pair $u, v \in V(G)$. Then deletion of one vertex cannot separate u from v. Since this condition is valid for every pair of vertices in G, G is 2-connected.

Necessity Assume that G is 2-connected. We show that G has two internally disjoint paths between every pair $u, v \in V(G)$ by induction on the length l of a shortest path between u and v.

If $l = 1$, there is an edge (u, v) in G. In this case $G - (u, v)$ is connected since $\kappa'(G) \geq \kappa(G) \geq 2$. Thus an u, v-path in $G - (u, v)$ and (u, v) are two internally disjoint paths in G. Assume that the claim is true for $l < k$ and we will prove the claim for $l = k$.

Let w be the vertex before v on a shortest u, v-path. Then the length of the shortest path between u and w is $k - 1$. By induction hypothesis, G has two internally disjoint paths P and Q between u and w. We now have two cases to consider.

Case 1: v is on P or Q. In this case v is on the cycle $P \cup Q$, and hence there are two internally disjoint paths between u and v.

Case 2: v is neither on P nor on Q. Since G is 2-connected, $G - w$ is connected and hence there is a path R between u and v in G which does not contain w. If R is internally disjoint to $P \cup v$ or $Q \cup v$, then we have found two internally disjoint paths between u and v. Otherwise, R contains vertices on both P and Q. Let z be the last vertex R before v belonging to $P \cup Q$, as illustrated in Fig. 3.12. Without loss of generality, we assume that z is on P. Then we obtain a path R' by concatenating u, z-subpath of P to z, v-subpath of R. Clearly R' and $Q \cup (w, v)$ are two internally disjoint paths between u and v. □

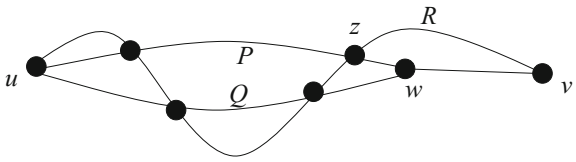

Fig. 3.12 Illustration for the proof of Theorem 3.4.5

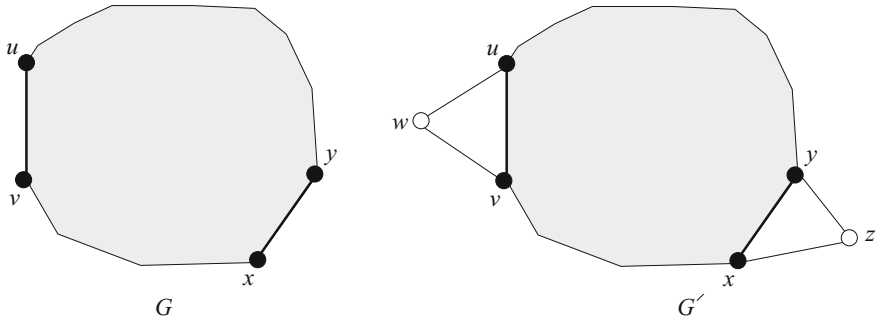

Fig. 3.13 Illustration for the proof of Lemma 3.4.7

The following corollary is very trivial from the lemma above.

Corollary 3.4.6 *For a graph G with at least three vertices, the following conditions are equivalent and characterize 2-connected graphs.*

(a) G is connected and has no cut vertex.
(b) For all $x, y \in V(G)$, there are internally disjoint x, y-paths.
(c) For all $x, y \in V(G)$, there is a cycle through x and y.

The following lemma also gives an useful characterization of a 2-connected graph.

Lemma 3.4.7 *Let G be a simple connected graph with three or more vertices. Then G is 2-connected if and only if every pair of edges in G lies on a common cycle.*

Proof We only prove the necessity, since the proof of sufficiency is trivial from Corollary 3.4.6.

Assume that G is 2-connected and let (u, v) and (x, y) be two edges of G (see Fig. 3.13(a)). We construct a graph G' by adding two vertices w and z of degree 2 to G such that u and v are the two neighbors of w and x and y are the two neighbors of z, as illustrated in Fig. 3.13(b). Clearly G' is 2-connected by Lemma 3.4.3, since G is 2-connected. Then by Corollary 3.4.6(c), there is a cycle C' in G' which contains both w and z. Since w and z are vertices of degree 2, edges (u, w), (w, v), (x, z) and (z, y) are on the cycle C'. If we replace from c' the path u, w, v by edge (u, v) and the path x, z, y by edge (x, y) we get a cycle C in G containing edges (u, v) and (x, z). □

3.4.4 Ear Decomposition

An *ear* of a graph G with $\delta(G) = 2$ is a maximal path with distinct end vertices whose internal vertices have degree 2 in G. Note that an edge $(u, v), u \neq v$ in G with $d_G(u) \geq 3$ and $d_G(v) \geq 3$ is also an ear of G. An *ear decomposition* of G is a

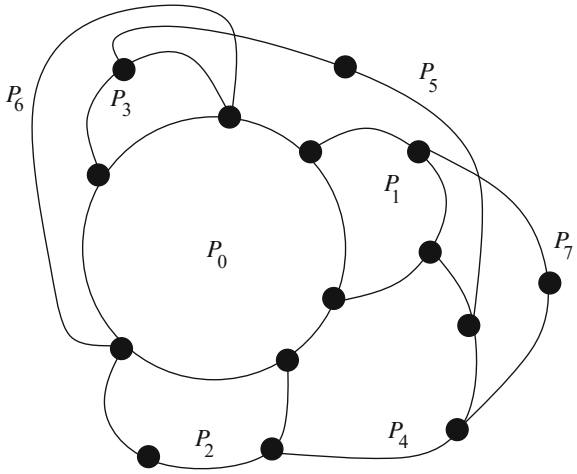

Fig. 3.14 Illustration for an ear decomposition a graph

decomposition $P_0, \ldots, P_k$ of edges such that P_0 is a cycle and P_i for $i \geq 1$ is an ear of the graph induced by $P_0 \cup \cdots P_i$. An ear decomposition $P_0, P_1, \ldots, P_7$ of a graph is illustrated in Fig. 3.14 where the ear P_6 is an edge. Whitney [3] in 1932 gave a characterization of a 2-connected graph in terms of ear decomposition as in the following theorem.

Theorem 3.4.8 *A graph G has an ear decomposition if and only if G is 2-connected.*

Proof Necessity Assume that G has an ear decomposition $P_0, P_1, \ldots, P_k$. By definition of ear decomposition P_0 is a cycle, which is 2-connected. Assume that $G_i = P_0 \cup \cdots \cup P_i$ is biconnected. We now show that $G_{i+1} = G_i \cup P_{i+1}$ is biconnected. G_i is biconnected and the two end vertices of P_{i+1} are on G_i. Then adding P_{i+1} to G_i will not introduce any cut vertex in G_{i+1}, and hence G_{i+1} is biconnected.

Sufficiency We give a constructive proof. Assume that G is 2-connected. Then G has a cycle. Let C be a cycle in G. We choose C as $P_0 = G_0$. If $G_0 \neq G$, we can choose an edge (u, v) of $G - E(P_0)$ and an edge $(x, y) \in E(P_0)$. Since G is 2-connected, (u, v) and (x, y) lie on a cycle C' by Lemma 3.4.7. Let P be a path in C' that contains (u, v) and exactly two vertices of G_0. We choose P as P_1. Clearly $G_1 = P_0 \cup P_1$ is biconnected and P_1 is an ear of G_1. Let G_i be the graph obtained by successively adding ears $P_1, P_2, \ldots, P_i$. We can find an ear P_{i+1} similarly as we found P_1. The process ends only by absorbing all edges of G. □

Since the end vertices of an ear defined above are distinct vertices, sometimes such an ear is called an *open ear*. A cycle C in a graph G is called a *closed ear* if all vertices of C except one have degree 2 in G.

Bibliographic Notes

For preparing this chapter several books [4–6] were used.

Exercises

1. Let G be a graph having exactly two vertices u and v of degree three and all other vertices have even degree. Then show that there is an u, v-path in G.
2. Write an algorithm to find a Eulerian circuit in an Eulerian graph based on the sufficiency proof of Lemma 3.2.1.
3. Give a proof of Lemma 3.2.2.
4. Count the minimum number of edges required to add for making a non-Eulerian graph an Eulerian graph.
5. Let K_n be a complete graph of n vertices where n is odd and $n \geq 3$. Show that K_n has $(n-1)/2$ edge-disjoint Hamiltonian cycles.
6. Show that every k-regular graph on $2k+1$ vertices is Hamiltonian [7].
7. Write an algorithm to find a pair of vertex disjoint paths between a pair of vertices in a 2-connected graph.
8. Write an algorithm to find an ear decomposition of a biconnected graph.
9. Let s be a designated vertex in a connected graph $G = (V, E)$. Design an $O(n+m)$ time algorithm to find a path between s and v with the minimum number of edges for all vertices $v \in V$.
10. Show that a 3-connected graph has at least six edges.

References

1. Biggs, N.L., Lloyd, E.K., Wilson, R.J.: Graph Theory: 1736–1936. Oxford University Press, Oxford (1976)
2. Ore, O.: Note on Hamiltonian circuits. Am. Mat. Mon. **67**, 55 (1960)
3. Whitney, H.: Congruent graphs and the connectivity of graphs. Am. J. Math. **54**, 150–168 (1932)
4. Agnarsson, G., Greenlaw, R.: Graph Theory: Modeling Applications and Algorithms. Pearson Education Inc., London (2007)
5. West, D.B.: Introduction to Graph Theory, 2nd edn. Prentice Hall, New Jersey (2001)
6. Wilson, R.J.: Introduction to Graph Theory, 4th edn. Longman, London (1996)
7. Nash-Williams, C.St.J.A.: Edge-disjoint Hamiltonian circuits in graphs with vertices of large valency. Studies in Pure Mathematics. Academic Press, London (1971)

Chapter 4
Trees

4.1 Introduction

A *tree* is a connected graph that contains no cycle. Figures 4.1(a), (b), and (c) illustrate trees with one, two, and three vertices, respectively. Figures 4.1(d) and (e) illustrate two different trees with four vertices. A path is always a tree. The trees in Figs. 4.1(a)–(d) are all paths. Recall that a path with n vertices has $n - 1$ edges. In fact, every tree has $n - 1$ edges as we shall see later in this chapter. For example, Fig. 4.1(f) illustrates a tree with 14 vertices and 13 edges.

A vertex with degree one in a tree T is called a *leaf* of T. All the vertices of T other than the leaves are called the *internal vertices* of T. The vertex d of the tree in Fig. 4.1(f) is a leaf and the vertex a is an internal vertex.

A collection of trees is called a *forest*. In other words, a forest is a graph with no cycle. Such a graph is also called an *acyclic graph*. Each component of a forest is a tree.

4.2 Properties of a Tree

There are many different sets of properties of trees; any of these sets of properties can be taken as a definition of a tree. In this section, we discuss these properties of a tree. We first observe the following two trivial properties of a tree which play crucial roles while dealing with trees.

Lemma 4.2.1 *Every tree with two or more vertices has at least two leaves.*

Proof Let T be a tree of two or more vertices and let P be a maximal path in T. Then the end vertices u and v of P have degree 1 (see Fig. 4.1(f)), otherwise P would not be a maximal path in T. □

M.S. Rahman, *Basic Graph Theory*, Undergraduate Topics in Computer Science,
DOI 10.1007/978-3-319-49475-3_4

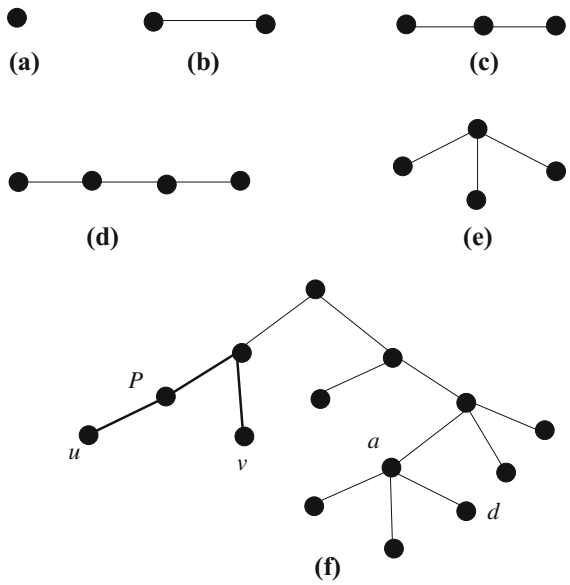

Fig. 4.1 Trees

Lemma 4.2.2 *Every edge in a tree is a cut edge.*

Proof Immediate from Lemma 3.1.4. □

The following lemma gives some characterization of trees.

Lemma 4.2.3 *Let G be a graph with n vertices. Then, any two of the following three statements imply the third (and characterize a tree of n vertices).*

(a) G is connected.
(b) G contains no cycle.
(c) G has $n - 1$ edges.

Proof (a) & (b) $\Rightarrow$ (c). We first prove that a connected and acyclic graph G with n vertices has $n - 1$ edges. The claim is obvious for $n = 1$ since a graph with a single vertex and no cycle has no edges. We thus assume that $n > 1$ and the claim is true for any connected and acyclic graph with less than n vertices. We now show that the graph G with n vertices has $n - 1$ edges. Since G contains no cycle, every edge e of G is a cut edge. Let H_1 and H_2 be the two connected components of $G - e$ with n_1 and n_2 vertices, respectively, where $n_1 + n_2 = n$. Since both H_1 and H_2 are acyclic and connected, they contain $n_1 - 1$ and $n_2 - 1$ edges respectively. Then the total number of edges in G is $n_1 - 1 + n_2 - 1 + 1 = n - 1$.

(a) & (c) $\Rightarrow$ (b). We now prove that a connected graph G with n vertices and $n - 1$ edges is acyclic. We delete edges from cycles of G one by one until the resulting graph G' is acyclic. Since no edge on a cycle is a cut edge by Lemma 3.1.4, G' is

connected. Then G' is connected and acyclic, and hence G' has $n-1$ edges. Since G has $n-1$, we have not deleted any edge from G to construct G'. Therefore G has no cycle.

(b) & (c) $\Rightarrow$ (a). We now prove that an acyclic graph G with n vertices and $m = n-1$ edges is connected. Assume for a contradiction that G is not connected. Then G consists of two or more connected components and each connected component is also acyclic. Let $G_1, \ldots, G_k$ be the connected components of G. Let n_i and m_i be the number of vertices and edges in G_i respectively. Clearly $\sum_i n_i = n$. Since each G_i is acyclic and connected $m_i = n_i - 1$. Then $m = \sum_i [n_i - 1] = n - k$. Since we have $m = n-1$, $k = 1$. Hence G is connected. □

The following corollaries are very trivial from the lemma above, and their proofs are left as exercises.

Corollary 4.2.4 *(1) A forest with n vertices and k components has $n-k$ edges.*
(2) A forest with n vertices and m edges contains $n-m$ components.
(3) A graph with n vertices, m edges and $n-m$ components is acyclic.

We also have the following lemma, which gives other characterizations of a tree.

Lemma 4.2.5 *A simple graph G is a tree if and only if for each pair of vertices u and v of G, there is a unique u, v-path in G*

Proof We first prove that for each pair of vertices u and v of a tree G, G contains a unique u, v-path. Since G is connected, there is at least one u, v-path between each pair of vertices. Assume for a contradiction that there are two distinct paths P_1 and P_2 between a pair of vertices u and v in G. Let x be a vertex on P_1 which is not on P_2. Then P_1 contains a subpath $P' = u', \ldots, x, v'$ such that only u' and v' of P' are on P_2. (Assume that u' is closer to u and v' is closer to v along P_1, and $u' = u$ or $v' = v$ may hold.) Then the parts of P_1 and P_2 from u' to v' forms a cycle in G as illustrated in Fig. 4.2, which is a contradiction.

We now prove that if for each pair of vertices u and v of a graph G, there is a unique u, v-path in G, then G is a tree. It is obvious from the assumption that G is connected. G is also acyclic since if G contains any cycle C, then C induces two distinct path in G between any pair of vertices on C. □

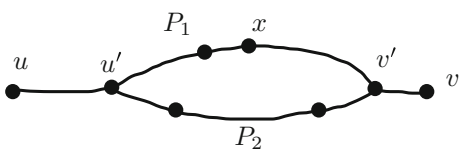

Fig. 4.2 An illustration for the proof of Lemma 4.2.5

4.3 Rooted Trees

A *rooted tree* is a tree in which one of the vertices is distinguished from the others. The distinguished vertex is called the *root* of the tree. The root of a tree is usually drawn at the top. In Fig. 4.3, the root is v_1. If a rooted tree is regarded as a directed graph in which each edge is directed from top to bottom, then every vertex u other than the root is connected by an edge from some other vertex p, called the *parent* of u. We also call u a *child* of vertex p. We draw the parent of a vertex above that vertex. For example, in Fig. 4.3, v_1 is the parent of v_2, v_3, and v_4, while v_2 is the parent of v_5 and v_6; v_2, v_3, and v_4 are the children of v_1, while v_5 and v_6 are the children of v_2. A *leaf* is a vertex of a tree that has no children. An *internal vertex* is a vertex that has one or more children. Thus every vertex of a tree is either a leaf or an internal vertex. In Fig. 4.3, the leaves are v_4, v_5, v_6, and v_7, and the vertices v_1, v_2, and v_3 are internal vertices.

The parent–child relationship can be extended naturally to ancestors and descendants. Suppose that $u_1, u_2, \ldots, u_l$ is a sequence of vertices in a tree such that u_1 is the parent of u_2, which is a parent of u_3, and so on. Then vertex u_1 is called an *ancestor* of u_l and vertex u_l a *descendant* of u_1. The root is an ancestor of every other vertex in a tree and every other vertex is a descendant of the root. In Fig. 4.3, v_1 is an ancestor of all other vertices, and all other vertices are descendants of the root v_1. Note that the definition of ancestor (descendant) does not allow a vertex to be an ancestor (descendant) of itself. However, there are some definitions of ancestor (descendant) which allow a vertex to be an ancestor (descendant) of itself.

A rooted tree is called a *binary tree* if each vertex has at most two children. In general, a rooted tree is called a *k-ary tree* if each vertex has at most k children. That is, the maximum number of children of a vertex in a k-ary tree is k.

An *ordered rooted tree* is a rooted tree in which the children of a vertex is somehow ordered. For example, in a ordered rooted binary tree the children of a vertex are ordered as the left child and the right child. The children of a vertex in a ordered rooted tree may also be ordered in a clockwise or in a counterclockwise order.

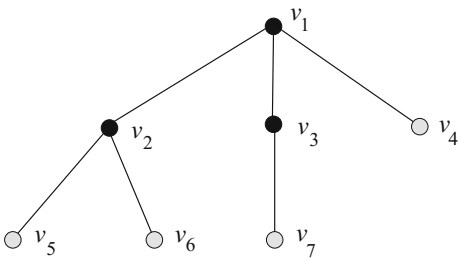

Fig. 4.3 A rooted tree

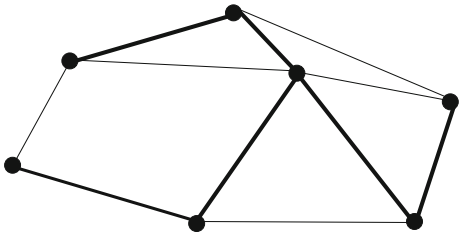

Fig. 4.4 Illustration for a spanning tree of a graph; *edges* of a spanning tree are drawn by *thick lines*

4.4 Spanning Trees of a Graph

A subgraph G' of a graph G is a *spanning subgraph* of G if G' contains all vertices of G. A spanning subgraph T of a graph G is a *spanning tree* of G if T is a tree. In Fig. 4.4 the edges of a spanning tree of a graph is drawn by thick lines. The following claim holds.

Lemma 4.4.1 *Every connected graph contains a spanning tree.*

Proof Let G be a connected graph. If G has no cycle, then G itself a spanning tree of G. We thus assume that G has cycles. We delete edges from cycles one by one until the resulting graph G' is acyclic. Since no edge on a cycle is a cut edge by Lemma 3.1.4, G' is connected. Hence, G' is a desired spanning tree T of G. □

Based on the proof of Lemma 4.4.1, one can develop an algorithm for finding a spanning tree of a connected graph as follows: Check whether the graph has a cycle or not. If the graph has a cycle, delete an edge from the cycle and repeat the process until the resulting graph is acyclic.

We now prove the following lemma.

Lemma 4.4.2 *Let $G = (V, E)$ be a connected graph and let $S \subseteq E$ such that $G - S$ is disconnected. Then every spanning tree of G has an edge in S.*

Proof Let T be a spanning tree of G. Assume for a contradiction that T does not contain any edge in S. Then $G - S$ would not be disconnected since T contains all vertices of G. □

A graph may have many spanning trees. Counting spanning trees is an essential step in many methods for computing, bounding, and approximating network reliability; in a network modeled by a graph, intercommunication between all nodes of the network implies that the graph must contain a spanning tree and thus, maximizing the number of spanning trees is a way of maximizing reliability [1]. We denote by $\tau(G)$ the number of spanning trees of a connected graph G. Recall that we denote by $G - e$ the graph obtained from a graph G by deleting its edge e, and by $G\backslash e$ the graph obtained from G by contracting its edge e. These two graph operations play crucial role in counting the number of a spanning trees of a graph, as shown in the following lemma.

Lemma 4.4.3 *Let G be a connected graph and let e be an edge in G. Then $\tau(G) = \tau(G - e) + \tau(G\backslash e)$.*

Proof Let $\mathcal{T}$ be the set of all spanning trees of G. Let $\mathcal{T}_1$ be the set of all spanning trees of G containing e and $\mathcal{T}_2$ be the set of all spanning trees not containing e. Clearly the sets $\mathcal{T}_1 \cap \mathcal{T}_2 = \phi$ and $\mathcal{T}_1 \cup \mathcal{T}_2 = \mathcal{T}$. Thus $\mathcal{T}_1 \cup \mathcal{T}_2$ is a partition of $\mathcal{T}$. One can easily observe that $|\mathcal{T}_1| = \tau(G\backslash e)$ and $|\mathcal{T}_2| = \tau(G - e)$. Therefore $\tau(G) = |\mathcal{T}| = |\mathcal{T}_1 \cup \mathcal{T}_2| = \tau(G - e) + \tau(G\backslash e)$. □

Note that $\tau(G - e) = 0$ if e is a cut edge of G. We can use Lemma 4.4.3 for counting number of spanning trees of a graph recursively, although this is not an efficient way of counting spanning trees. A more efficient method for counting spanning trees of graphs is to use Matrix-Tree theorem due to Kirchhoff (1847). Recently Nikolopoulos et al. used a modular decomposition technique to calculate the number of spanning trees in some special classes of graphs in linear time [1].

Let $G = (V, E)$ be an edge-weighted graph and let T be a spanning tree of G. We call the sum of the weights of the edges of T the *cost* of T. Figures 4.5(b) and (c) show two spanning trees of the graph in Fig. 4.5(a) with costs 24 and 15, respectively. A spanning tree T of G is called a *minimum spanning tree* of G if the cost of T is minimum among the costs of all spanning trees of G. The spanning tree in Fig. 4.5(c) is a minimum spanning tree of the graph in Fig. 4.5(a). Although a graph may have an exponential number of spanning trees, a minimum spanning tree of a graph can be found using greedy methods in polynomial time. Kruskal's algorithm and Prim's algorithm are two well-known greedy algorithms, which find a minimum spanning tree of a graph efficiently [2]

Kruskal's greedy algorithm for constructing a minimum spanning tree of a graph is very simple. Step by step illustration of Kruskal's algorithm is given in Fig. 4.6. Let $G = (V, E)$ be the input graph as illustrated in Fig. 4.6(a). The algorithm starts with a spanning subgraph $G_0 = (V, E_0)$ of G as illustrated in Fig. 4.6(b), where E_0 is an empty set of edges. Let e_1 be an edge of G with the minimum weight. Then $G_1 = (V, E_1)$ is constructed in Step 1 as illustrated in Fig. 4.6(c), where $E_1 = E_0 \cup \{e\}$. That is, G_1 is constructed by adding an edge of G with the minimum weight in G_0. In Step 2, the graph $G_2 = (V, E_2)$ with $E_2 = E_1 \cup e_2$ is constructed as

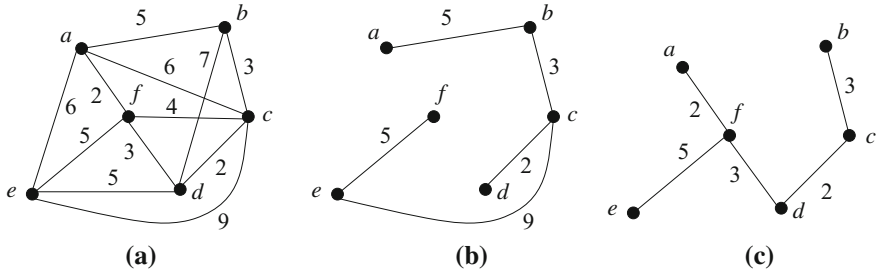

Fig. 4.5 **(a)** An edge-weighted graph G, **(b)** a spanning tree of G with cost 24, and **(c)** a minimum spanning tree of G with cost 15

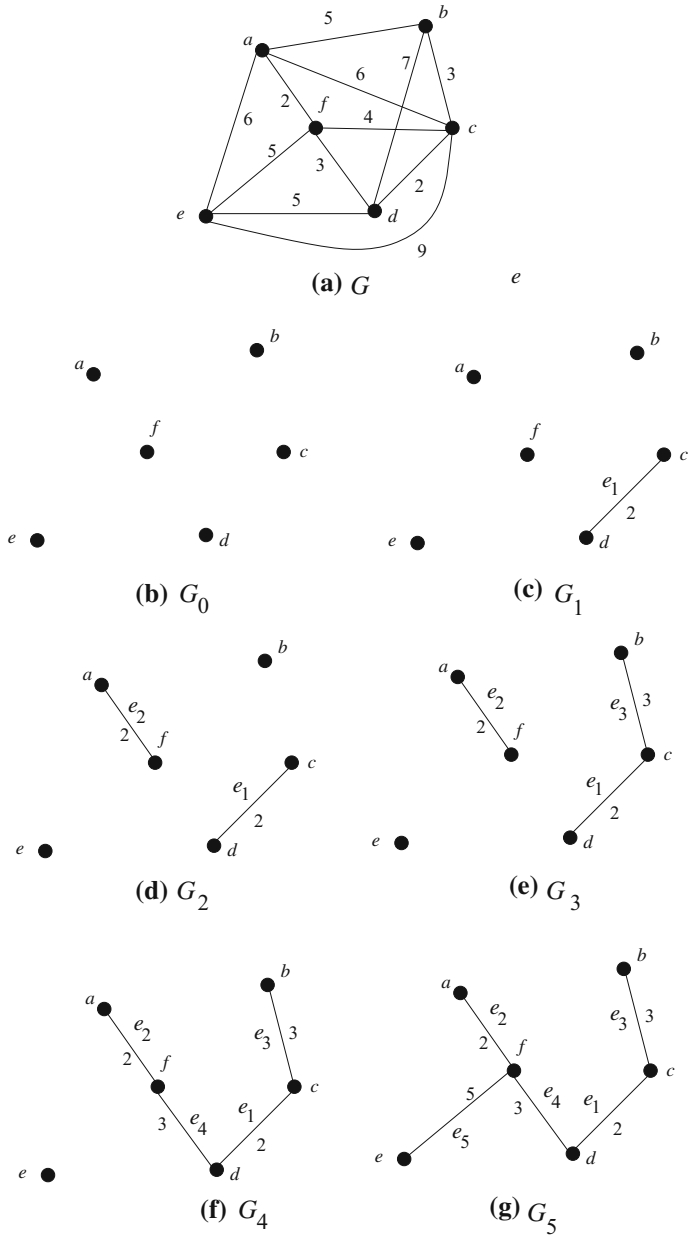

Fig. 4.6 Illustration for Kruskal's algorithm

illustrated in Fig. 4.6(d), where e_2 is an edge of $G - E_1$ with the minimum weight such that e_2 does not create a cycle in G_2. Then in each successive step i, $G_i = (V, E_i)$ with $E_i = E_{i-1} \cup \{e_i\}$ is constructed as illustrated in Figs. 4.6(e)–(g), where e_i is an edge in $G - E_{i-1}$ with the minimum weight such that e_i does not create a cycle in G_i. This process continues until construction of G_{n-1}, where n is the number of vertices in G. The constructed G_{n-1} is a desired minimum spanning tree T of G. The correctness proof of the algorithm above is not so difficult, which is left for an exercise.

4.5 Counting of Trees

To deal with counting of graphs we need to be familiar with the definitions of "labeled graphs" and "unlabeled graphs," which are informally introduced in Sections 2.2.2 and 2.5.5. A graph is called a *labeled graph* if each vertex of the graph is assigned an element from a set of symbols, so that the vertices can be distinguished one from another. A graph which has no such labeling is called an *unlabeled graph*. The two graphs in Figs. 4.7(a) and (b) are labeled graphs and the graph in Fig. 4.7(c) is an unlabeled graph. Note that the two graphs Figs. 4.7(a) and (b) are different labeled graphs although they are the same unlabeled graph as in Fig. 4.7(c).

There is exactly one tree with a single vertex. The number of trees with two vertices is also one. The number of labeled trees with three vertices is three although there is exactly one unlabeled tree of three vertices. With four vertices 16 labeled

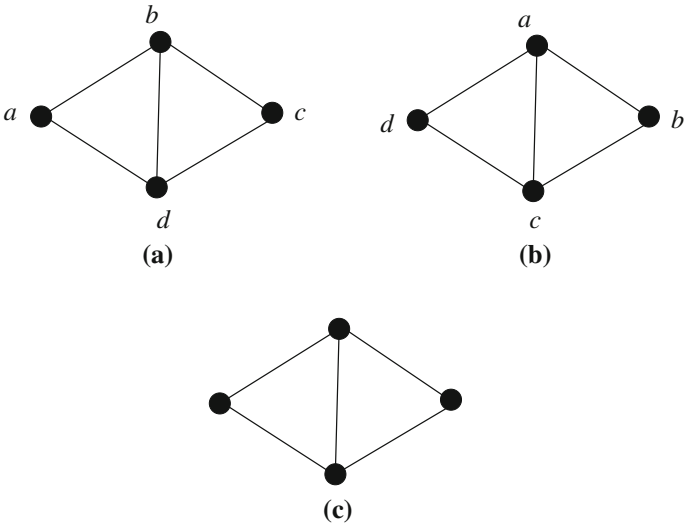

Fig. 4.7 **(a)–(b)** Labeled graphs and **(c)** an unlabeled graph

trees can be constructed. In this section, we present Cayley's theorem [3] on number of labeled trees.

Theorem 4.5.1 *There are n^{n-2} distinct labeled trees of n vertices.*

Proof The idea is to establish a one-to-one correspondence between the set of labeled trees of n vertices and the set of sequences $(a_1, a_2, \ldots, a_{n-2})$, where each a_i is an integer satisfying $1 \leq a_i \leq n$. Since there are precisely n^{n-2} such sequences, the result follows immediately.

We assume $n \geq 3$, since the result is trivial if $n = 1$ or 2.

Let T be a labeled tree of n vertices. We can obtain a unique sequence $(x_1, x_2, \ldots, x_{n-2})$ from tree T as follows. Let y_1 be the smallest label among the labels assigned to all leaf vertices, and let x_1 be the label of the vertex adjacent to y_1. We delete the vertex y_1. Clearly $T' = T - y_1$ is a tree of $n - 1$ vertices. Let y_2 be the leaf vertex in T' with the smallest label and let x_2 be the neighbor of y_2. We delete y_2. We continue the process until there are only two vertices left and we obtain the sequence $(x_1, x_2, \ldots, x_{n-2})$. An illustrative example for step by step construction of the sequence from a tree is given in Fig. 4.8, where the edge (y_i, x_i) is drawn by dotted line in each step.

We now show that from a sequence $(x_1, x_2, \ldots, x_{n-2})$ we can construct a unique tree of n vertices. Let $\{1, 2, \ldots, n\}$ be the set of n vertices. We will construct a tree with the vertex set $\{1, 2, \ldots, n\}$. Let y_1 be the smallest number that does not appear in $(x_1, x_2, \ldots, x_{n-2})$. We add the edge (x_1, y_1), remove x_1 from the sequence and remove y_1 from consideration and continue the process. Finally, we add the last edge between two vertices whose label have not appeared in $\{y_1, y_2, \ldots, y_{n-2}\}$. We now need to show that the process mentioned above constructs a tree. We started with n isolated vertices labeled by $\{1, 2, \ldots, n\}$. We consider each of the vertices unmarked. Then each component has an unmarked vertices. First we join edge (x_1, y_1), mark y_1, and x_1 remains unmarked. Hence each component has an unmarked vertices. One can observe that when we add an edge (x_i, y_i), both x_i and y_i are unmarked vertices and hence they are in two different components. After adding the edge (x_i, y_i), the two components are merged to one component and we mark y_1. Hence each component has exactly one unmarked vertex. Finally, we have exactly two components and each component has exactly one unmarked vertex. We join the two unmarked vertices and obtain a connected graph. Since we have added $n - 1$ edges the connected graph is a tree. An illustrative example for construction of a tree from a sequence 1, 3, 3, 7, 9, 9, 7 is given in Fig. 4.9. In Fig. 4.9(a), each vertex is a component and every vertex is unmarked. In Fig. 4.9(b), $x_1 = 1$ and $y_1 = 2$ since 2 is the smallest integer that does not appear in the sequence 1, 3, 3, 7, 9, 9, 7. After adding edge (x_1, y_1), y_1 is marked and each component has exactly one unmarked vertex. Incremental construction of the remaining of the tree is shown in Figs. 4.9(b)–(i). □

The proof of Theorem 4.5.1 is due to Prüffer [4] and Clarke [5] and the sequence used in the proof is known as Prüffer's code. Some other proofs can be found in [6].

Cayley's formula immediately gives the number of spanning trees of a complete graph. Since K_n has all the edges that can be used in forming trees of n vertices, the number of spanning trees of K_n is equal to the number of trees of n vertices, i.e., n^{n-2}.

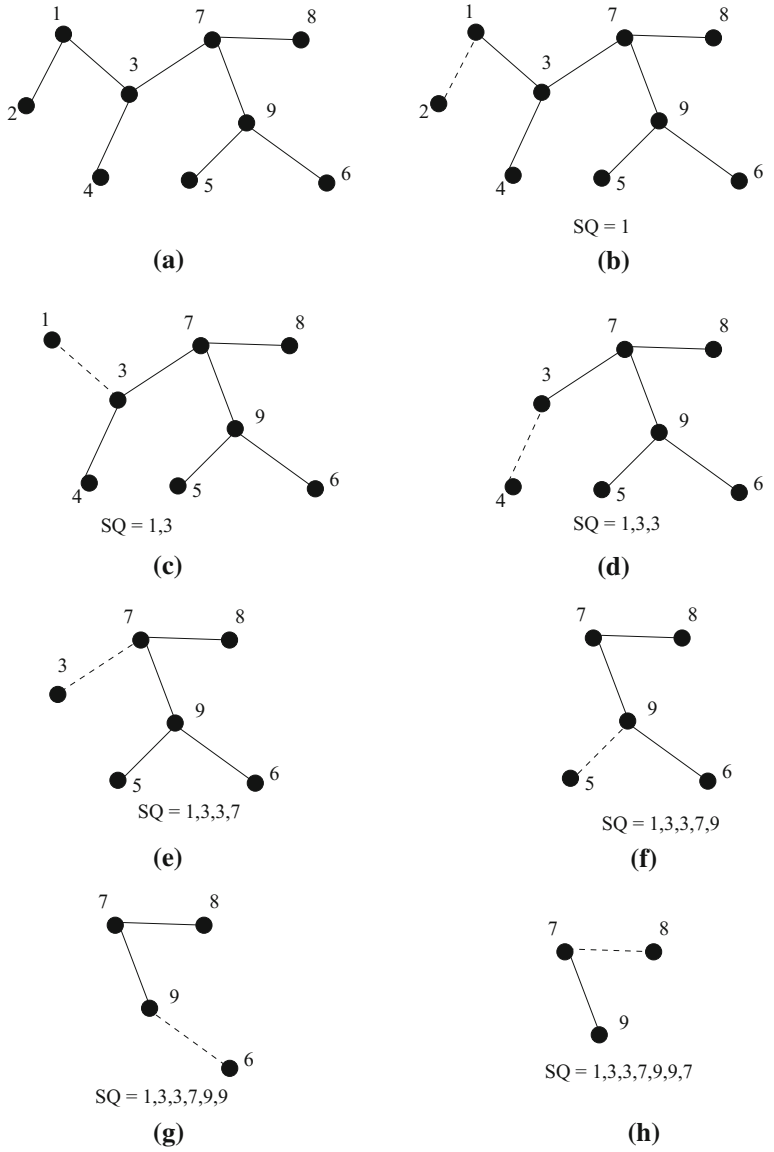

Fig. 4.8 Construction of the sequence from a tree

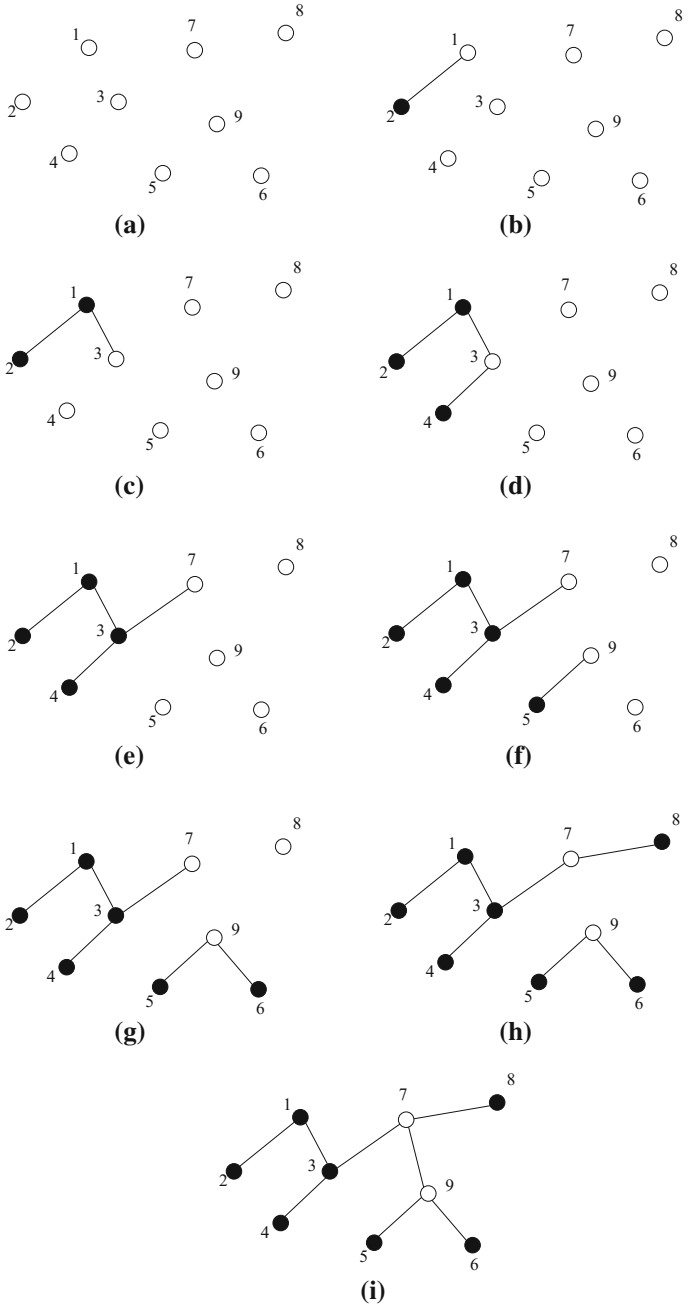

Fig. 4.9 Construction of the tree from a sequence

4.6 Distances in Trees and Graphs

In this section, we study some distance parameters of graphs and trees. If G has a u, v-path, then the *distance* from u to v is the length of a shortest u, v-path. The distance from u to v in G is denoted by $d_G(u, v)$ or simply by $d(u, v)$. For the graph in Fig. 4.10, $d(a, j) = 5$ although there is a path of length 8 between a and j. If G has no u, v-path then $d(u, v) = \infty$. The *diameter* of G is the longest distance among the distances of all pair of vertices in G. The graph in Fig. 4.10 has diameter 6. The *eccentricity* of a vertex u in G is $max_{v \in V(G)} d(u, v)$ and denoted by $\epsilon(u)$. Eccentricities of all vertices of the graph in Fig. 4.10 are shown in the figure. The *radius* of a graph is $min_{u \in V(G)} \epsilon(u)$. The *center* of a graph G is the subgraph of G induced by vertices of minimum ecentricity. The maximum of the vertex eccentricities is equal to the diameter. The radius and the diameter of the graph in Fig. 4.10 is 3 and 6 respectively. The vertex h is the center of the graph in Fig. 4.10. The diameter and the radius of a disconnected graph are infinite.

Since there is only one path between any two vertices in a tree, computing the distance parameters for trees is not difficult. We have the following lemma on the center of a tree.

Lemma 4.6.1 *The center of a tree is a vertex or an edge.*

Proof We use induction on the number of vertices in a tree T. If $n \leq 2$ then the entire tree is the center of the tree. Assume that $n \geq 3$ and the claim holds for any tree with less than n vertices. Let T' be the graph obtained from T by deleting all leaves of T. T' has at least one vertex, since T has a non-leaf vertex as $n \geq 3$. Clearly T' is connected and has no cycle, and hence T' be a tree with less than n vertices. By induction hypothesis the center of T' is a vertex or an edge. To complete the proof, we now show that T and T' have the same center. Let v be a vertex in T. Every vertex u in T which is at maximum distance from v is a leaf. Since all leaves of T have been deleted to obtain T' and any path between two non-leaf vertices in T does not contain a leaf, $\epsilon_{T'}(u) = \epsilon_T(u) - 1$ for every vertex u in T'. Furthermore, the ecentricity of a leaf is greater than the ecentricity of its neighbor in T. Hence the

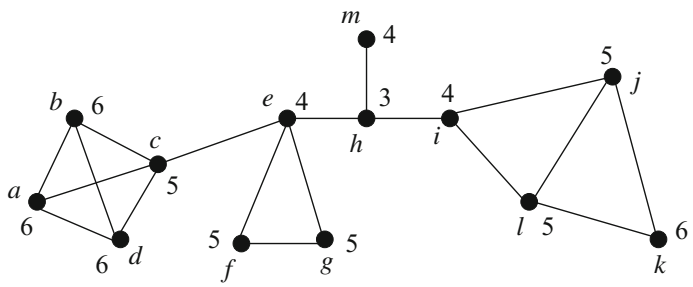

Fig. 4.10 Eccentricities of vertices

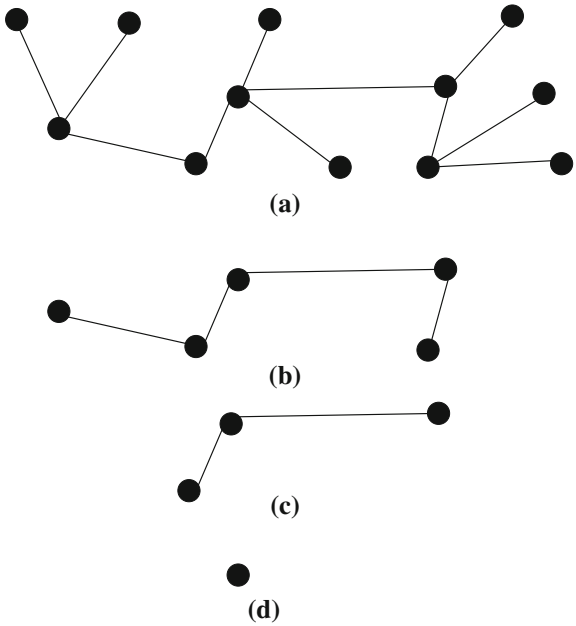

Fig. 4.11 Finding a center of a tree

vertices whose ecentricities are minimum in T and the vertices whose ecentricities are minimum in T' are the same. Therefore T and T' have the same center. □

Following the proof of Lemma 4.6.1, we can find the center of a tree if we continue to delete all the leaves of the tree recursively until we are left with a single vertex or a single edge as illustrated in Fig. 4.11.

4.7 Graceful Labeling

A *graceful labeling* of a simple graph G, with n vertices and m edges, is a one-to-one mapping f of the vertex set V into the set $\{0, 1, 2, \ldots, m\}$, such that distinct vertices receive distinct numbers and f satisfies $\{|f(u) - f(v)| : uv \in E(G)\} = \{1, 2, 3, \ldots, m\}$. The absolute difference $|f(u) - f(v)|$ is regarded as the *label of the edge* $e = (u, v)$ in the graceful labeling. The number received by a vertex in a graceful labeling is regarded as the *label of the vertex*. A graph G is called *graceful* if G admits a graceful labeling.

One can easily compute a graceful labeling of a path as follows. Start labeling of vertices (i.e., assigning labels to vertices) at either end. The first vertex is labeled by 0 and the next vertex on the path is labeled by $n - 1$, the next vertex is labeled by 1, the next vertex is labeled by $n - 2$, and so on. Figure 4.12 illustrates a graceful labeling of a path. It is not difficult to observe that edges get labels $n - 1, n - 2, \ldots, 1$.

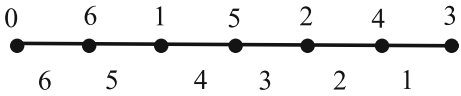

Fig. 4.12 Labels of vertices and edges in a graceful labeling of a path

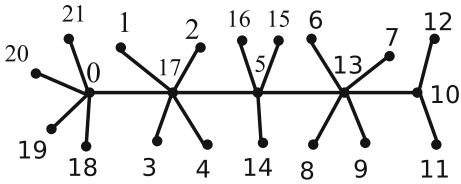

Fig. 4.13 Labels of vertices in a graceful labeling of a caterpillar

An interesting pattern of labeling exits for a graceful labeling of a "caterpillar." A *caterpillar* is a tree for which deletion of all leaves produces a path. Observe Fig. 4.13 for an interesting pattern of graceful labeling of a caterpillar.

All trees are graceful — is the famous Ringel–Kotzig conjecture which has been the focus of many papers. Graphs of different classes have been proven mathematically to be graceful or nongraceful. All trees with 27 vertices are graceful was shown by Aldred and McKay using a computer program in 1998. Aryabhatta et. al showed that a fairly large class of trees constructed from caterpillars are graceful [7]. A *lobster* is a tree for which deletion of all leaves produces a caterpillar. Morgan showed that a subclass of lobsters are graceful [8].

Bibliographic Notes

For preparing this chapter several books [9–12] were used. Interested readers can find more in those books.

Exercises

1. Show that a forest with n vertices and k components has $n - k$ edges.
2. Show that a forest with n vertices and m edges contains $n - m$ components.

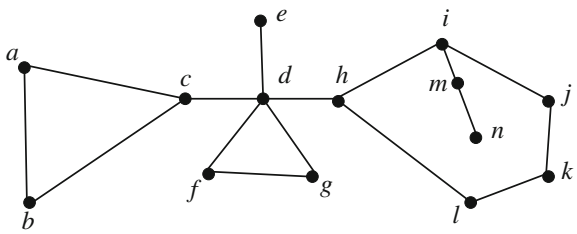

Fig. 4.14 A graph

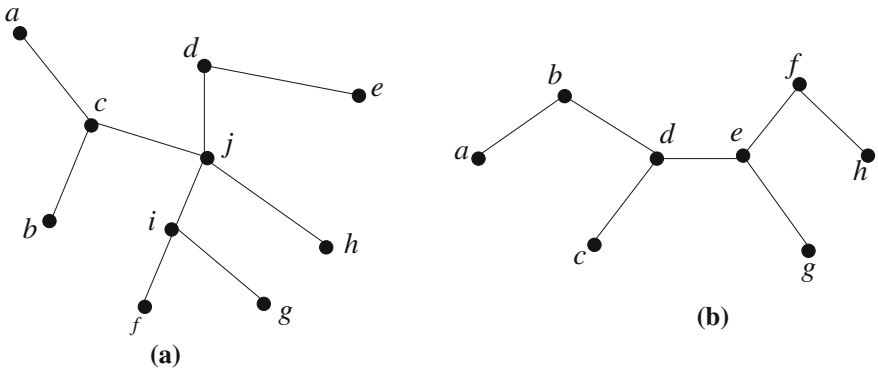

Fig. 4.15 Trees

3. Show that a graph with n vertices, m edges, and $n - m$ components is acyclic.
4. Prove the correctness of Kruskal's algorithm described in Section 4.4.
5. Draw all labeled trees with four vertices.
6. Construct all labeled spanning trees of K_4. How many of them are non-isomorphic?
7. Compute the ecentricities of the vertices in the graph in Fig. 4.14.
8. Find the centers of the trees in Figs. 4.15(a) and (b).
9. Show that the center of a tree is a single vertex if the diameter of the tree is even.
10. Compute the Prüffer's code for Figs. 4.15(a) and (b).
11. Construct the tree corresponding to Prüffer's code 1,2,2,7,6,6,5.
12. Find a graceful labeling of the tree in Fig. 4.15(b).

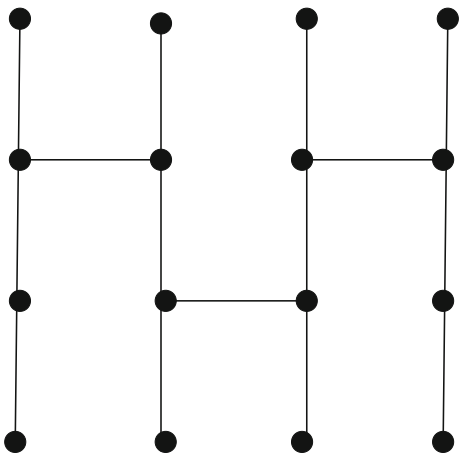

Fig. 4.16 A graceful tree

13. Develop an algorithm for computing a graceful labeling of a caterpillar.
14. Find a graceful labeling of the tree in Fig. 4.16. Can you develop an algorithm for finding graceful labelings of this type of trees?
15. Show that every tree containing a vertex of degree k contains at least k leaves.

References

1. Nikolopoulos, S.D., Palios, L., Papadopoulos, C.: Counting spanning trees in graphs using modular decomposition. In: Proceedings of WALCOM 2011. Lecture Notes in Computer Science, vol. 6552, pp. 202–213. Springer, Berlin (2011)
2. Goodrich, M.T., Tamassia, R.: Algorithm Design: Foundations, Analysis, and Internet Examples. Wiley, New York (2002)
3. Cayley, A.: A theorem on trees. Quart. J. Math. **23**, 376–378 (1889)
4. Prüfer, H.: Neuer beweis eines satzes über permutationen. Arch. Math. Phys. **27**, 142–144 (1918)
5. Clarke, L.E.: On Cayley's formula for counting trees. J. London Math. Soc. **33**, 471–474 (1958)
6. Moon, J.W.: Counting Labeled Trees. Canadian Mathematical Congress, Montreal (1970)
7. Arybhatta, S., Roy, T.G., Uddin, M., Rahman, M.S.: On graceful labeling of trees. In: Proceedings of WALCOM 2011. Lecture Notes in Computer Science, vol. 6552, pp. 214–220. Springer, Heidelberg (2011)
8. Morgan, D.: All lobsters with perfect matchings are graceful. Bull. Inst. Comb. Appl. **53**, 82–85 (2008)
9. Agnarsson, G., Greenlaw, R.: Graph Theory: Modeling, Applications and Algorithms. Pearson Education Inc., New Jersey (2007)
10. Nishizeki, T., Rahman, M.S.: Planar Graph Drawing. World Scientific, Singapore (2004)
11. West, D.B.: Introduction to Graph Theory, 2nd edn. Prentice Hall, New Jersey (2001)
12. Wilson, R.J.: Introduction to Graph Theory, 4th edn. Longman, London (1996)

Chapter 5
Matching and Covering

5.1 Matching

A *matching* in a graph is a set of non-loop edges with no common endpoints. The set $\{(a, e), (b, c), (d, f)\}$ of edges is a matching in graphs in Figs. 5.1(a) and (b). The vertices incident to the edges of a matching M are *saturated* by M; the others are *unsaturated*. In the graph in Fig. 5.1(a) vertices a, b, c, d, e, and f are saturated, whereas the vertex g is unsaturated.

5.1.1 Perfect Matching

A *perfect matching* in a graph is a matching that saturates every vertex. The matching $\{(a, e), (b, c), (d, f)\}$ is a perfect matching in the graph in Fig. 5.1(b), since all vertices are saturated. A complete graph of odd vertices does not have a perfect matching, but a complete graph of even vertices always has a perfect matching. A graph may have many perfect matchings. For example, $K_{n,n}$ has $n!$ perfect matchings. The bipartite graph in Fig. 5.2 has unique perfect matching.

5.1.2 Maximum Matching

A *maximal matching* in a graph is a matching that cannot be enlarged by adding an edge. A *maximum matching* is a matching of maximum size among all matchings in the graph. The matching $\{(b, c)\}$ in Fig. 5.3(a) is a maximal matching but not a maximum matching since there is a larger matching $\{(a, b), (c, d)\}$ of this graph as shown in Fig. 5.3(b).

Given a matching M in a graph G, an *M-alternating path* is a path that alternates between edges in M and edges not in M. An M-alternating path is an *M-augmenting*

M.S. Rahman, *Basic Graph Theory*, Undergraduate Topics in Computer Science,
DOI 10.1007/978-3-319-49475-3_5

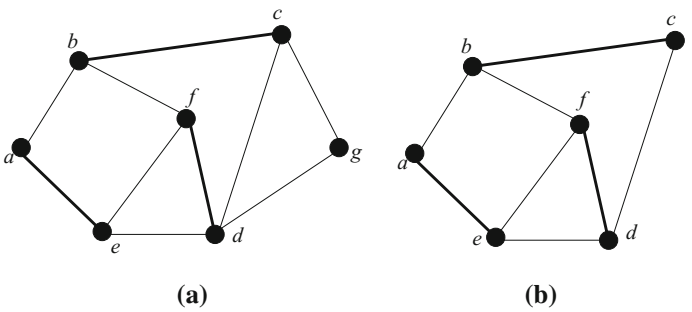

Fig. 5.1 Illustration of matchings: **(a)** a matching of a graph with an unsaturated vertex and **(b)** a perfect matching of a graph

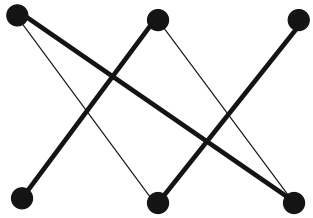

Fig. 5.2 A bipartite graph with unique perfect matching

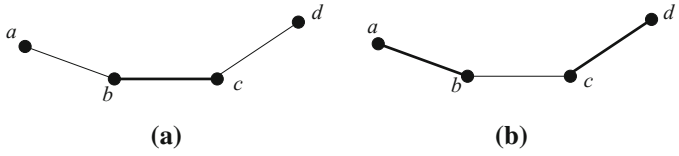

Fig. 5.3 Illustration of maximal and maximum matchings

path if the two endpoints of the path are unsaturated by M. If G has an M-augmenting path P, one can obtain a new matching M' with one more edge by replacing the edges of M in P with the other edges of P. Thus the following fact holds.

Fact 5.1.1 *M is not a maximum matching in G if G has an M-augmenting path.*

On the other hand, it can be proved that if M is not a maximum matching in G then G has an M-augmenting path.

Let M and M' be two matchings in $G = (V, E)$. The *symmetric difference of two matchings* M and M' is the graph with the vertex set V and the edge set consisting of all edges appearing exactly one of M and M'. The symmetric difference of two matchings M and M' is denoted by $M \triangle M'$. Let S be the set of edges which is contained in M but not in M' and let S' be the set of edges which is contained in M' but not in M. Then $M \triangle M' = S \cup S'$. In Fig. 5.4 $M = \{(a, b), (c, f), (d, e), (i, h)\}$, $M' = \{(a, e), (b, d), (c, f), (g, h)\}$ and

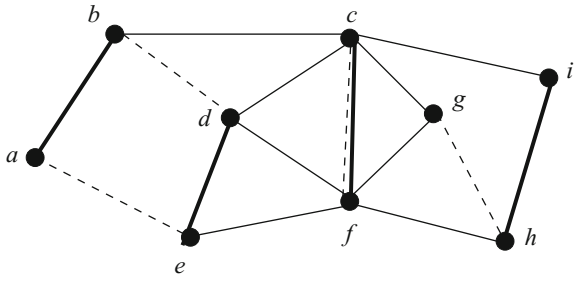

Fig. 5.4 Symmetric difference of two matchings

$M \triangle M' = \{(a, b), (b, d), (d, e), (a, e), (i, h), (g, h)\}$. Since the edge (c, f) is contained in both M and M', it does not appear in $M \triangle M'$. Observe that $M \triangle M'$ has two connected components: one is a cycle and the other is a path. In fact a connected component of the symmetric difference of two matchings is either a cycle or a path as in the following lemma.

Lemma 5.1.2 *Every connected component of the symmetric difference of two matchings is a path or an even cycle.*

Proof Let M and M' be two matchings in a graph G and let $H = M \triangle M'$.

We first claim that $\Delta(H) \leq 2$. At most one edge from a matching is incident to a vertex of G. Since M and M' are both matchings, at most two edges can be incident to a vertex of H. Hence $\Delta(H) \leq 2$.

Since $\Delta(H) \leq 2$, every connected component of H is either a path or a cycle. It is thus remained to show that if a connected component of H is a cycle C, then C is an even cycle. One can observe that the edges on C alternate between edges of M and M'. Then to close the cycle C, C must have equal number of edges from M and M', and hence C is an even cycle. □

We now ready to prove the following lemma.

Lemma 5.1.3 *A matching M in a graph G has an M-augmenting path if M is not a maximum matching in G.*

Proof Let M' be a matching in G larger than M. We will prove that G has an M-augmenting path. We give a constructive proof. Let $H = M \triangle M'$. By Lemma 5.1.2, every connected component of H is either a path or a cycle. If every component of H is a cycle then $|M| = |M'|$, a contradiction. Since $|M'| > |M|$, H has a connected component which is a path P containing more edges of M' than of M. Then P starts with an edge of M' and also ends with an edge of M'. Since P contains edges from M and M' alternately, P is an M-augmenting path. □

Fact 5.1.1 and Lemma 5.1.3 immediately prove the following theorem due to Berge [1].

Theorem 5.1.4 *A matching M in a graph G is a maximum matching in G if and only if G has no M-augmenting path.*

5.1.3 Hall's Matching Condition

A company has received applications for its job vacancies. Assume that the company has a set J of job vacancies and has received a set A of applicants. An applicant in A may have applied for several jobs, but each applicant will be given at most one job. Naturally in this era of job crisis, the number of applicants is larger than the number of jobs. But does it ensure that every vacancy will be filled out? The scenario can be modeled by a bipartite graph as illustrated in Fig. 5.5, where job vacancies of five jobs are represented by the vertices $j_1, j_2, \ldots, j_5$, seven applicants are represented by the vertices $a_1, a_2, \ldots, a_7$, and an edge between a job and an applicant represents that the corresponding applicant has applied for the corresponding job. In the example in Fig. 5.5, it is impossible to fill out every vacancy since the graph has no matching where all of vertices $j_1, j_2, \ldots, j_5$ are saturated. However, the graph in Fig. 5.6 has a matching drawn by thick edges which saturates all vertices $j_1, j_2, \ldots, j_5$, and hence each vacancy can be filled out. Thus it is interesting to know in which case such a matching exists.

The problem can be formulated as follows. Let G be a bipartite graph with bipartition $V(G) = X \cup Y$. Find necessary and sufficient conditions for the existence of a matching in G which saturates every vertex in X. This problem is well known as the *marriage problem.* For any set S of vertices in G, the *neighbor set* of S in G, denoted by $N(S)$, is defined to be the set of all vertices adjacent to vertices in S. A vertex in $N(S)$ is called a *neighbor of* S. If a matching M saturates X, then for every $S \subseteq X$ there must be at least $|S|$ vertices that have neighbors in S. Thus $|N(S)| \geq |S|$

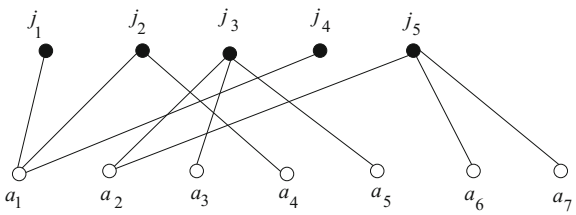

Fig. 5.5 Modeling of job vacancies and applicants

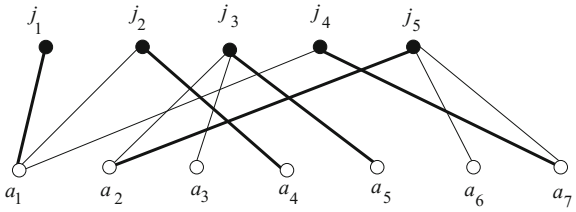

Fig. 5.6 A matching indicating a feasible solution for job vacancy fill out

for every $S \subseteq X$ is a necessary condition. Hall proved that this obvious necessary condition is also sufficient as in the following theorem [2].

Theorem 5.1.5 *Let G be a bipartite graph with bipartition $V(G) = X \cup Y$. Then G contains a matching that saturates every vertex in X if and only if $|N(S)| \geq |S|$ for every subset S of X.*

Proof Since the necessity is obvious, we only prove the sufficiency. Assume that $|N(S)| \geq |S|$ for every subset S of X. We show that G contains a matching M that saturates every vertex in X. Assume for a contradiction that G has a maximum matching M that does not saturate every vertex in X. Then there is a vertex $x \in X$ that is not saturated by M. Let $A \subseteq V(G)$ be the set of vertices consisting of x and all vertices that can be connected to x by M-alternating paths in G. Let $S = A \cap X$ and $T = A \cap Y$. (See Fig. 5.7.) Clearly $N(S) \subseteq Y$. Since M is a maximum matching, G has no M-augmenting path. Hence, each vertex in T is an end vertex of an edge in M, whose other end vertex is in S. Then $T \subseteq N(S)$. Again there is an M-alternating path P from x to each vertex in S and the last edge of P is from M. There is also an M-alternating path from x to any neighbor of S. Hence by the definition of T we have $N(S) \subseteq T$. (Note that T contains all vertices in Y which can be reached by an M-augmenting path from x.) Therefore $|N(S)| = |T|$ holds.

Since M is a maximum matching, G has no M-augmenting path, and hence each $a \in A - \{x\}$ is saturated by M. Therefore, M yields a one-to-one correspondence between $S - \{x\}$ and T. Thus $|T| = |S| - 1$. Since $|N(S)| = |T|$, $|N(S)| = |S| - 1$. This implies $|N(S)| < |S|$, a contradiction to our assumption that $|N(S)| \geq |S|$. □

In the graph in Fig. 5.5, the vertex set $S = \{j_1, j_4\}$ has only one neighbor, and hence it does not satisfy the condition in Theorem 5.1.5.

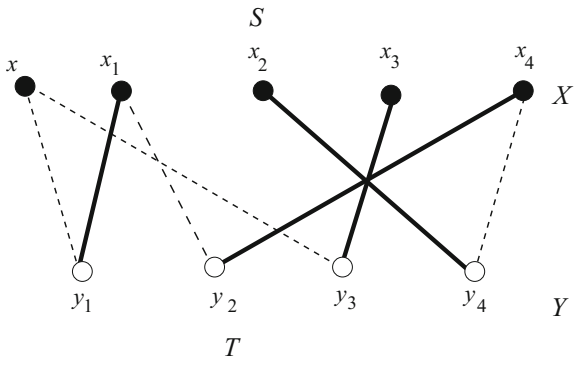

Fig. 5.7 Illustration for the proof of Theorem 5.1.5; $S = \{x, x_1, x_2, x_3, x_4\}$ and $T = \{y_1, y_2, y_3, y_4\}$

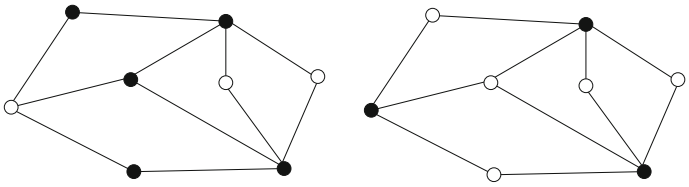

Fig. 5.8 Illustration for independent sets

5.2 Independent Set

A set S of vertices of a graph G is *independent* if no two of its vertices are adjacent in G. The subgraph of G induced by the vertices in S is a null graph. Figures 5.8(a) and (b) illustrate independent sets of 3 and 5 vertices, respectively, where vertices in the independent sets are drawn by white circles. A set containing a single vertex is trivially an independent set. Finding a larger independent set needs careful checking of the adjacency of vertices. An independent set S of G is *maximal* if S is not a proper subset of any other independent set of G. An independent set S of G is *maximum* if no other independent set of G has more vertices than S. That is, a maximum independent set of G contains the maximum number of vertices among all independent sets of G. The independent set shown in Fig. 5.8(a) is a maximal independent set, whereas the independent set shown in Fig. 5.8(b) is a maximum independent set. The *independence number* of G, denoted by $\alpha(G)$, is the number of vertices in a maximum independent set. For the graph in Fig. 5.8, $\alpha(G) = 5$.

5.3 Covers

A *vertex cover* of a graph $G = (V, E)$ is a set $Q \subseteq V$ that contains at least one endpoint of every edge.

The government plans to establish police check post with sophisticated equipment in road crossings of a city in such a way that every road has a check post. In a graph model of the city, where each vertex represents a road crossings and each edge represents a road, a vertex cover gives a feasible solution for the locations of police check posts. If the government wishes to minimize the number of police check posts for budget constraint, a vertex cover having the minimum number of vertices gives a feasible solution. A vertex cover of a graph G is a *minimum vertex cover* if it contains the minimum number of vertices among all vertex covers of G. The vertex set $\{b, d, e, f, h\}$ is a minimum vertex cover in the graph in Fig. 5.9.

An independent set of a graph has a complement relation with a vertex cover of the graph, as stated in the following lemma.

Lemma 5.3.1 *Let $G = (V, E)$ be a graph. Then a set $S \subseteq V$ is an independent set of G if and only if $V - S$ is a vertex cover of G.*

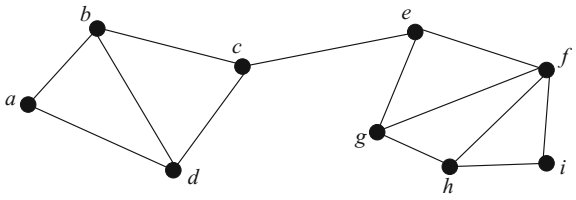

Fig. 5.9 Illustration for vertex covers and dominating sets

Proof Necessity. Let S be an independent set of G. We show that $V - S$ is a vertex cover. Let (u, v) be an arbitrary edge of G. Since S is an independent set, S cannot contain both u and v; one of u and v will be contained in $V - S$. This implies that every edge has at least one end vertex in $V - S$, and hence $V - S$ is a vertex cover.

Sufficiency. Assume that $V - S$ is a vertex cover. We show that S is an independent set. Assume for a contradiction that S is not an independent set. Then there is a pair of adjacent vertices u and v in S. That means there is an edge (u, v) such that none of its end vertices belongs to $V - S$, and hence $V - S$ is not a vertex cover, a contradiction. □

Due to the complement relation in Lemma 5.3.1, the problem of finding a maximum independent set and the problem of finding a minimum vertex cover of a graph are equally hard. In fact, both the problems are NP-hard [3].

An *edge cover* of G is a set L of edges such that every vertex of G is incident to some edge of L.

5.4 Dominating Set

For a graph $G = (V, E)$, a set $D \subseteq V(G)$ of vertices is a *dominating set* of G if every vertex in V is either in D or adjacent to a vertex of D. A dominating set D of G is *minimal* if D does not properly contain a dominating set of D. The vertex set $\{b, e, i\}$ is a minimal dominating set in the graph in Fig. 5.9. A dominating set D of G is *minimum* if no other dominating set has fewer vertices than D. The cardinality of a minimum dominating set of G is called the *domination number* of G and denoted by $\gamma(G)$. For the graph in Fig. 5.9 $\gamma(G) = 2$ and the vertex set $\{b, f\}$ is a minimum dominating set. We say a vertex in a dominating set *dominates* itself and all of its neighbors.

The government plans to establish fire stations in a new city in such a way that a locality or one of its neighbor localities will have a fire station. In a graph model of the city, where each vertex represents a locality and each edge represents the neighborhood of two localities, a dominating set gives a feasible solution for the locations of fire stations. If the government wishes to minimize the number of fire stations for budget constraint, a minimum dominating set gives a feasible solution.

Domination of vertices has been studied extensively due its practical applications in scenarios described above [4].

Domination number has a relation with diameter of a graph as we see in the following lemma.

Lemma 5.4.1 *Let G be a connected simple graph, $\gamma(G)$ be the domination number of G, and* $\text{diam}(G)$ *be the diameter of G. Then*

$$\gamma(G) \geq \left\lceil \frac{\text{diam}(G)+1}{3} \right\rceil.$$

Proof Let x and y be two vertices of G such that $d(x, y) = \text{diam}(G) = k$, and let $P = u_0(= x), u_1, \ldots, u_k(= y)$ be a path of length k in G from x to y. Let D be a domination set of G. We now prove that each vertex in D can dominate at most three vertices on P. Let u be a vertex in D. If u is on P, u can dominate at most three vertices on P: v itself and its (at most) two neighbors. If u is not on P, u can also dominate at most three vertices on P and those vertices must be consecutive on P; otherwise, there would exist a path between x and y shorter than P, a contradiction to the definition of $\text{diam}(G)$. Therefore, each vertex in D dominates at most three vertices on P.

Since the number of vertices on P is $k + 1 = \text{diam}(G) + 1$, $\gamma(G) \geq \lceil \frac{\text{diam}(G)+1}{3} \rceil$. □

We now describe an algorithm to compute domination number. If the graph has a few number of vertices we can compute domination number by simple observation. Thus the idea is to make the graph into smaller pieces recursively by deleting edges, compute domination numbers of smaller pieces, and compute the domination number of a larger graph from the domination number of its smaller pieces.

Let $e = (u, v)$ be an edge of a simple graph $G = (V, E)$. Let $E_v(u)$ be the set of edges incident to u not including the edge (u, v). Similarly $E_u(v)$ be the set of edges incident to v not including the edge (u, v). Then it is known [5] that the domination number $\gamma(G)$ satisfies

$$\gamma(G) = \min(\{\gamma(G - e), \gamma(G - E_v(u)), \gamma(G - E_u(v))\}). \tag{5.1}$$

Using Eq. 5.1 we can recursively compute $\gamma(G)$, as illustrated in Fig. 5.10. For computing domination number we also need the following two facts.

Fact 5.4.2 *Let G be a disconnected graph with connected components $H_1, H_2, \ldots, H_k$.Then $\gamma(G) = \sum_{i=1}^{k} \gamma(H_i)$.*

Fact 5.4.3 *Let G be a connected graph with blocks $B_1, B_2, \ldots, B_k$. Then $\gamma(G) \leq \sum_{i=1}^{k} \gamma(B_i)$.*

We now compute the domination number of the graph in Fig. 5.10. Note that G is decomposed into smaller subgraphs using Eq. 5.1. Each component of G_4 has domination number 1. Since it has three components domination number of G_4 is 3

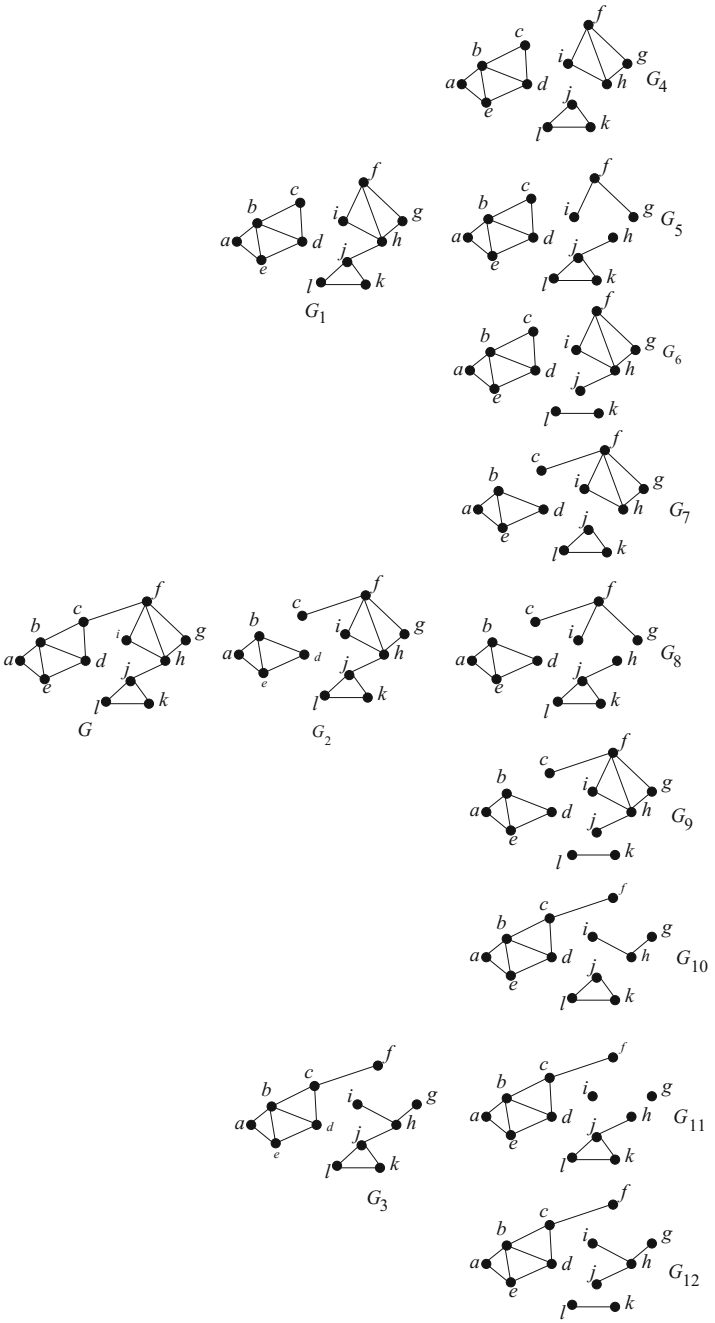

Fig. 5.10 Computation of domination number

by Fact 5.4.2. Similarly we can compute $\gamma(G_5) = 3$ and $\gamma(G_6) = 3$. Using Eq. 5.1 $\gamma(G_1) = min\{3, 3, 3\} = 3$. Similarly $\gamma(G_2) = min\{3, 3, 4\} = 3$ and $\gamma(G_3) = min\{4, 5, 4\} = 4$. Finally $\gamma(G) = min\{3, 3, 4\} = 3$.

Of course, the algorithm is not efficient. In fact the decision version of the problem is known to be NP-Complete [3] and hence it is unlikely to have a polynomial time algorithm to compute the domination number.

In recent years, a variation of a dominating set called a connected dominating set has been studied extensively due to its application in wireless sensor networks [6, 7]. A dominating set D of a graph G is a *connected dominating set* if the vertices in D induce a connected subgraph of G. If D is a connected dominating set, one can form a spanning tree of G in which D forms the set of non-leaf vertices of the tree; conversely, if T is any spanning tree in a graph with more than two vertices, the non-leaf vertices of T form a connected dominating set. Therefore, finding minimum connected dominating sets is equivalent to finding spanning trees with the maximum possible number of leaves.

5.5 Factor of a Graph

A *factor* of a graph G is a spanning subgraph of G. A k-factor is a spanning k-regular subgraph. Clearly, a 1-factor is a perfect matching and exists only for graphs with an even number of vertices. A 2-factor of G is a disjoint union of cycles of G if the 2-factor is not connected; a connected 2-factor is a Hamiltonian cycle.

We now know Tutte's condition [8] for 1-factor. A connected component H of a graph is an *odd component* if H has odd number of vertices. We denote by $oc(G)$ the number of odd components in a graph G. The following theorem is from Tutte [8].

Theorem 5.5.1 *A graph G has a 1-factor if and only if $oc(G - S) \leq |S|$ for every $S \subseteq V(G)$.*

If we delete the vertex x from the graph in Fig. 5.11(a), then we get two odd components. Taking $S = \{x\}$, the graph violates the condition in Theorem 5.5.1, and hence it does not have a 1-factor. However, the graph in Fig. 5.11(b) satisfies the condition in Theorem 5.5.1 and it has a 1-factor as shown by thick edges.

We now see an application of Theorem 5.5.1 in the proof of Theorem 5.5.2 (due to Chungphaisan [9]) which gives a necessary and sufficient condition for a tree to have a 1-factor. The proof presented here is due to Amahashi [10, 11].

Theorem 5.5.2 *A tree T of even order has a 1-factor if and only if $oc(T - v) = 1$ for every vertex v of T.*

Proof Assume that T has a 1-factor F. Then for every vertex v of T, let w be the vertex of T joined to v by an edge of F, as illustrated in Fig. 5.12(a). It follows that the component of $T - v$ containing w is odd, and all the other components of $T - v$ are even. Hence $oc(T - v) = 1$. Suppose that $oc(T - v) = 1$ for every $v \in V(T)$.

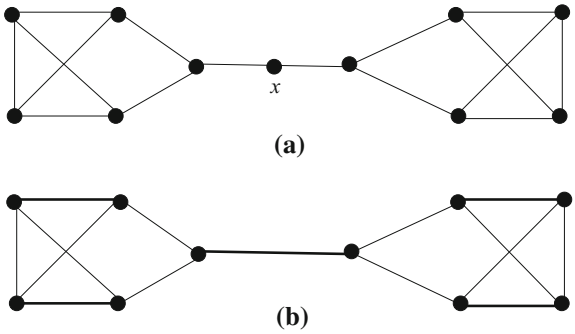

Fig. 5.11 **(a)** A graph violating the condition in Theorem 5.5.1 and **(b)** a graph satisfying the condition in Theorem 5.5.1

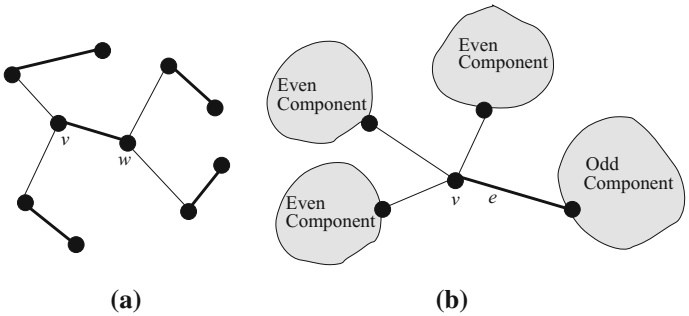

Fig. 5.12 Illustration for the proof of Theorem 5.5.2

It is obvious that for each edge e of T, $T - e$ has exactly two components, and both of them are simultaneously odd or even. Define a set F of edges of T as follows: $F = \{e \in E(T) : oc(T - e) = 2\}$. For every vertex v of T, there exists exactly one edge e that is incident with v and satisfies $oc(T - e) = 2$ since $T - v$ has exactly one odd component, where e is the edge joining v to this odd component. (See Fig. 5.12(b).) Therefore e is an edge of F, and thus F is a 1-factor of G. □

Bibliographic Notes

The books [5, 10, 12] were used in preparing this chapter. Interested readers can find the book [10] very useful for studying matchings, covers, and factors.

Exercises

1. Find a maximum matching in each of the graphs in Fig. 5.13 and identify whether it has a perfect matching or not.
2. Compute the number of perfect matchings in K_n.
3. Write a formal proof of Theorem 5.1.4.

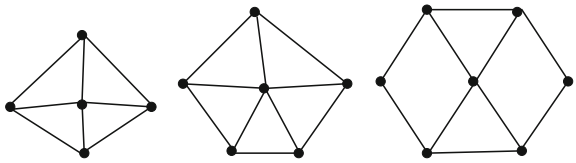

Fig. 5.13 Graphs

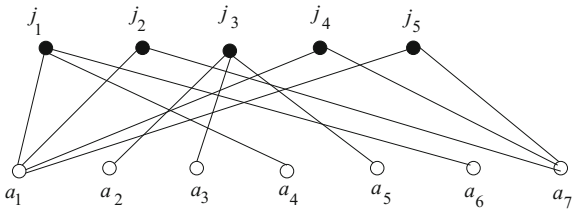

Fig. 5.14 A bipartite graph

4. Show that every k-regular bipartite graph has a perfect matching.
5. Using Theorem 5.1.5 show that the graph in Fig. 5.14 has no matching where all vertices $j_1, j_2, \ldots, j_5$ are saturated.
6. Find all maximum independent sets in the graph in Fig. 5.6.
7. Find a maximum independent set, a minimum vertex cover, and a minimum dominating set in the graph in Fig. 5.14.
8. How many maximum independent sets of vertices can a path of n vertices have?
9. Let T be a tree in which every leaf is of distance 2 from another leaf. Show that every maximum independent set of T must contain all the leaves of T.
10. Show that a tree in which every maximal path has an even length has a unique maximum independent set. Can you describe the vertices of this unique set?
11. Show that a graph with n vertices, m edges, and $n - m$ components is acyclic.
12. A dominating set D of a graph G is an *independent dominating set* if D is an independent set of G. Show that D is an independent dominating set if and only if D is a maximal independent set.
13. Prove or disprove: if a tree has a 1-factor, then the 1-factor is unique.

References

1. Berge, C.: Two theorems in graph theory. Proc. Natl. Acad. Sci. USA **43**, 842–844 (1957)
2. Hall, P.: On representatives of subsets. J. Lond. Math. Soc. **10**, 26–30 (1935)
3. Garey, M.R., Johnson, D.S.: Computers and Intractability. W.H. Freeman and Company, New York (1979)
4. Haynes, T.W., Hedetniemi, S., Slater, P.: Fundamentals of Domination in Graphs. Marcel Dekker Inc., New York (1998)

5. Agnarsson, G., Greenlaw, R.: Graph Theory: Modeling, Applications and Algorithms. Pearson Education Inc., New Jersey (2007)
6. Du, D., WAN, P.: Connected Dominating Set: Theory and Applications. Springer Optimization and Its Applications, vol. 77. Springer Science+Business Media, New York (2013)
7. Yu, J., Wang, N., Wang, G., Yu, D.: Connected dominating sets in wireless ad hoc and sensor networks - a comprehensive survey. Comput. Commun. **36**(2), 121–134 (2013)
8. Tutte, W.T.: The factorization of linear graphs. J. Lond. Math. Soc. **22**, 107–111 (1947)
9. Bondy, J.A., Murty, U.S.R.: Graph Theory with Applications. Elsevier Science Ltd/North-Holland, New York (1976)
10. Akiyama, J., Kano, M.: Factors and Factorizations of Graphs. Lecture Notes in Mathematics 2031. Springer, New York (2011)
11. Amahashi, A.: On factors with all degrees odd. Graphs Comb. **1**, 1–6 (1985)
12. West, D.B.: Introduction to Graph Theory, 2nd edn. Prentice-Hall, New Jersey (2001)

Chapter 6
Planar Graphs

6.1 Introduction

Consider a graph of six vertices and eight edges in Fig. 6.1(a) where some pairs of edges cross each other. Is it possible to redraw the graph in such a way that there is no edge crossing in the drawing? Yes, it is possible to draw the graph without any edge crossing as in Fig. 6.1(b). However, it is not possible to redraw the graph in Fig. 6.1(c) such that there is no edge crossing in the drawing. A graph is *planar* if it can be drawn or embedded in the plane so that no two edges intersect geometrically except at a vertex to which they are both incident. The graph in Fig. 6.1(a) is a planar graph since it has a planar embedding as in Fig. 6.1(b), whereas the graph in Fig. 6.1(c) is a nonplanar graph.

6.2 Characterization of Planar Graphs

Since not all graphs are planar, it is sometimes necessary to know whether a given graph is planar or not. Consider the graph K_5 in Fig. 6.2(a). Is it planar? We can proceed with simple observation as follows. Let p, q, r, s, and t be the five vertices of K_5. There is a cycle $C = p, q, r, s, t$ containing all the vertices of K_5. We draw the cycle as a pentagon as illustrated in Fig. 6.2(b). We can draw pr either inside the cycle C or outside C. We draw the edge (p, r) inside C, the other case is similar. Since (q, s) and (q, t) cross (p, r) if we draw them inside C, we must draw them outside C. Since (p, s) cannot cross (q, t), we must draw (p, s) inside C. Now we can draw (r, t) neither inside nor outside C without edge crossing. So K_5 cannot be drawn without edge crossings, and hence K_5 is not a planar graph. Using a similar argument we can observe that $K_{3,3}$ is also nonplanar. It is interesting that based on this two forbidden graphs, Kuratowski gave a complete characterization of planar graphs as in the following theorem.

M.S. Rahman, *Basic Graph Theory*, Undergraduate Topics in Computer Science,
DOI 10.1007/978-3-319-49475-3_6

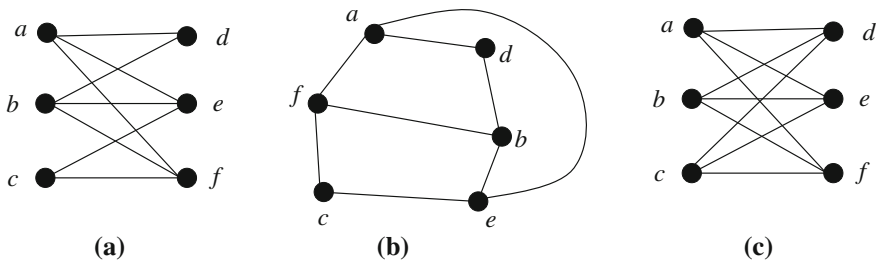

Fig. 6.1 Illustration for planar and nonplanar graphs

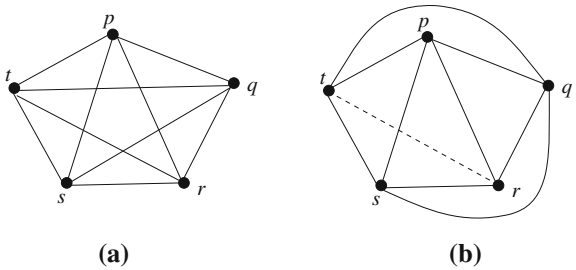

Fig. 6.2 Drawing of K_5

Theorem 6.2.1 (Kuratowski [1]) *A graph is planar if and only if it contains neither a subdivision of* K_5 *nor a subdivision of* $K_{3,3}$.

Kuratowski's characterization is one of the most beautiful theorems in graph theory. However, the characterization does not lead to an efficient algorithm.

6.3 Plane Graphs

Consider the four graphs in Figs. 6.3(a)–(d). If we observe carefully, we realize that these are the planar embeddings of the same planar graph. Thus, a planar graph can have more than one planar embeddings. In fact, a planar graph may have an exponential number of planar embeddings. One can differentiate two planar embeddings of a planar graph by observing the clockwise or counterclockwise ordering of the edges incident to each vertex. For example, clockwise ordering of the edges incident to vertex c is (c, b), (c, g), (c, f), (c, d), (c, e) in Fig. 6.3(a), whereas the clockwise ordering is (c, b), (c, d), (c, e), $(c, f)(c, g)$ in Fig. 6.3(b). Usually, by the term "an embedding of a planar graph" we mean a planar embedding of a planar graph.

A *plane graph* G is a planar graph with a fixed embedding in the plane. A plane graph divides the plane into connected regions called *faces*. The unbounded region is called the *outer face* of G. The boundary of a face of a connected plane graph G is a closed walk in general, and is a cycle if G is 2-connected and has at least three vertices. The boundary of the outer face of G is called the *outer boundary* of G and denoted by $C_o(G)$. If $C_o(G)$ is a cycle, then $C_o(G)$ is called the *outer cycle* of G.

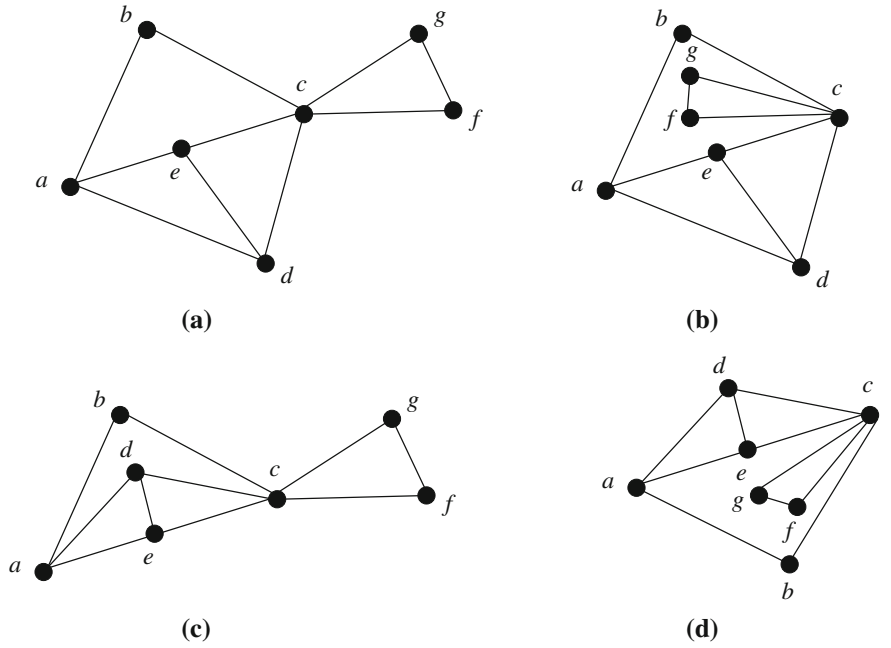

Fig. 6.3 Four planar embeddings of the same planar graph

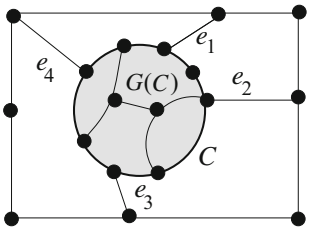

Fig. 6.4 Illustration of $G(C)$ and legs of a cycle C

We call a vertex v of G an *outer vertex of* G if v is on $C_o(G)$; otherwise v is an *inner vertex of* G. Similarly, we define an *outer edge* and an *inner edge of* G.

For a cycle C in a plane graph G, we denote by $G(C)$ the plane subgraph of G inside C (including C). The subgraph shaded in Fig. 6.4 is $G(C)$ for a cycle C drawn by a thick circle. An edge which is incident to exactly one vertex of C and located outside of C is called a *leg* of C, and the vertex of C to which the leg is incident is called a *leg-vertex* of C. A cycle C in G is called a *k-legged cycle* of G if C has exactly k legs. The cycle C in Fig. 6.4 is a 4-legged cycle with the legs e_1, e_2, e_3, and e_4.

We say that cycles C and C' in a plane graph G are *independent* if $G(C)$ and $G(C')$ have no common vertex. A set $\mathcal{S}$ of cycles is *independent* if any pair of cycles in $\mathcal{S}$ are independent.

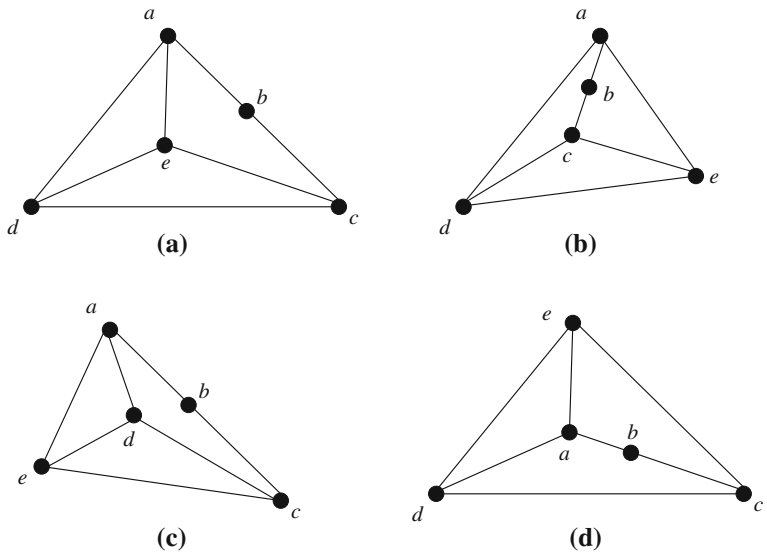

Fig. 6.5 Four equivalent planar embeddings of a planar graph

As mentioned earlier in this section, a planar graph may have an exponential number of embeddings in the plane. We shall now define an equivalence relation among these embeddings. Two embeddings of a planar graph are *equivalent* when the boundary of each face in one embedding corresponds to the boundary of a face in the other. Figure 6.5 illustrates four equivalent planar embeddings of a planar graph. Note that a facial cycle of a planar embedding has not become a nonfacial cycle in another plane embedding. (Note that the four embeddings in Fig. 6.5 are not same but equivalent.) On the other hand, the planar embeddings in Figs. 6.3(a) and (b) are not equivalent. We say that the plane embedding of a graph is *unique* when the embeddings are all equivalent [2]. (We may visualize this uniqueness on the surface of a sphere.) The embedding of the planar graph in Fig. 6.5 is unique since its all embeddings are equivalent. If G is a disconnected plane graph, one can obtain a new nonequivalent embedding simply by replacing a connected component within another face. Similarly, if G has a cut vertex v, one may obtain a new nonequivalent embedding by replacing a component of $G - v$ (together with the edges joining v and vertices in the component) in another face incident to v or changing the ordering of the components of $G - v$ around v. Thus, we shall assume that G is 2-connected if the embedding is unique. Whitney [3] proved that the embedding of a 3-connected planar graph is unique. Before proving the result, we need some definitions.

If G has a separation pair $\{x, y\}$, then we often split G into two graphs G_1 and G_2, called *split graphs* of G, as follows. Let $G'_1 = (V_1, E'_1)$ and $G'_2 = (V_2, E'_2)$ be two subgraphs satisfying the following conditions (a) and (b):

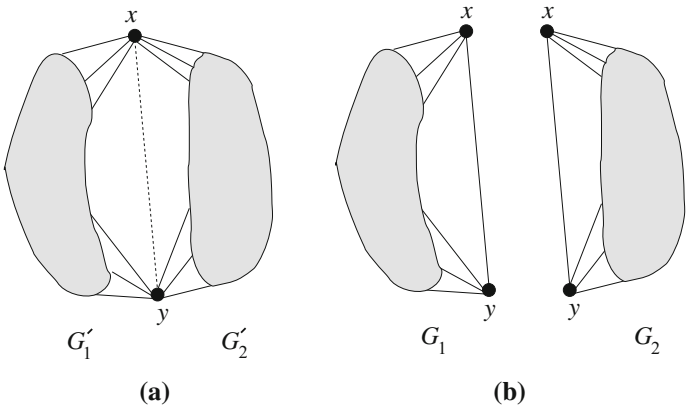

Fig. 6.6 **(a)** A graph G with a separation pair $\{x, y\}$ where edge (x, y) may exist, and **(b)** split graphs G_1 and G_2 of G

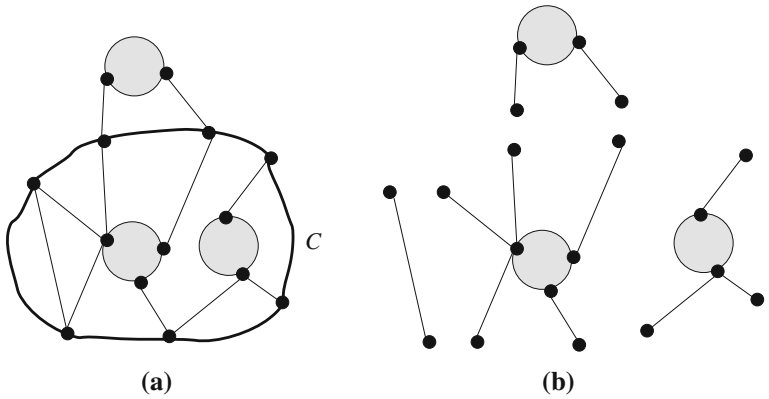

Fig. 6.7 Illustration for C-components

(a) $V = V_1 \cup V_2$, $V_1 \cap V_2 = \{x, y\}$;
(b) $E = E_1' \cup E_2'$, $E_1' \cap E_2' = \emptyset$, $|E_1'| \geq 2$, $|E_2'| \geq 2$.

Define a split graph G_1 to be the graph obtained from G_1' by adding a new edge (x, y) if it does not exist; similarly define a split graph G_2. (See Fig. 6.6.)

Let C be a cycle of a graph G. A graph of a single edge, not in C, joining two vertices in C is called a *C-component* of G. A graph which consists of a connected component of $G - V(C)$ and all edges joining vertices in that component and vertices in C is also called a *C-component*. The C-components for the cycle C drawn by thick line in Fig. 6.7(a) are shown in Fig. 6.7(b). We now present the result of Whitney [3] as in the following theorem whose proof is taken from [2, 4].

Theorem 6.3.1 *The embedding of a 2-connected planar graph G is unique if and only if G is a subdivision of a 3-connected graph.*

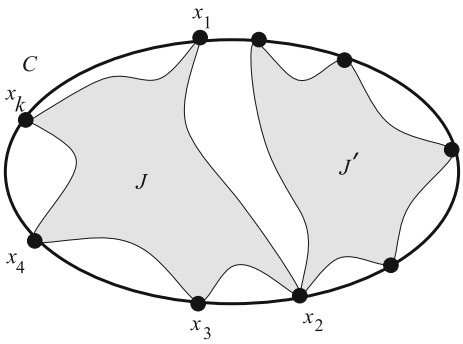

Fig. 6.8 Illustration for the proof of Theorem 6.3.1

Proof Necessity: Suppose that a 2-connected planar graph G is not a subdivision of a 3-connected graph. Then there is a separation pair $\{x, y\}$ having split graphs G_1 and G_2 such that both G'_1 and G'_2 are not paths. (See Fig. 6.6.) Consider a plane embedding of G in which both x and y are outer vertices. Then a new embedding of G is obtained by a reflection or twist of G'_1 or G'_2. The boundary of the outer face in the original embedding is no longer a face boundary in the new embedding. Thus the embedding of G is not unique.

Sufficiency: Suppose that the embedding of a 2-connected planar graph G is not unique. Thus, according to the definition, the original embedding $\Gamma(G)$ of G has a face F whose facial cycle C in $\Gamma(G)$ is no longer a facial cycle in another embedding $\Gamma'(G)$ of G. Clearly G has two C-components J and J'; one in the interior and the other in the exterior of C in $\Gamma'(G)$. One may assume that C is the boundary of the outer face of $\Gamma(G)$. Let $x_1, x_2, \ldots, x_k$ be the vertices of C contained in J occurring in cyclic order. One may assume that all the vertices of C contained in J' are in the subpath of C joining x_1 and x_2 and containing no other x_i, as illustrated in Fig. 6.8. Then $\{x_1, x_2\}$ is a separation pair of G, for which both G'_1 and G'_2 are not paths. Therefore, G is not a subdivision of a 3-connected graph. □

Theorem 6.3.1 immediately implies that every 3-connected planar graph has a unique plane embedding.

6.3.1 Euler's Formula

There is a simple formula relating the numbers of vertices, edges, and faces in a connected plane graph. It is known as *Euler's Formula* because Euler established it for those plane graphs defined by the vertices and edges of polyhedra. In this section, we discuss Euler's Formula and its immediate consequences.

Theorem 6.3.2 (Euler 1750) *Let G be a connected plane graph, and let n, m, and f denote, respectively, the numbers of vertices, edges, and faces of G. Then $n - m + f = 2$.*

Proof We employ an induction on m, the result being obvious for $m = 0$ or 1. Assume that $m \geq 2$ and the result is true for all connected plane graphs having fewer than m edges, and suppose that G has m edges. Consider first the case G is a tree. Then G has a vertex v of degree one. The connected plane graph $G - v$ has $n - 1$ vertices, $m - 1$ edges and $f(=1)$ faces, so by the inductive hypothesis, $(n - 1) - (m - 1) + f = 2$, which implies that $n - m + f = 2$. Consider next the case when G is not a tree. Then G has an edge e on a cycle. In this case, the connected plane graph $G - e$ has n vertices, $m - 1$ edges, and $f - 1$ faces, so that the desired formula immediately follows from the inductive hypothesis. □

A *maximal planar* graph is one to which no edge can be added without losing planarity. Thus in any embedding of a maximal planar graph G with $n \geq 3$, the boundary of every face of G is a triangle, and hence the embedding is often called a *triangulated plane graph*. Although a general graph may have up to $n(n - 1)/2$ edges, it is not true for planar graphs.

Corollary 6.3.3 *If G is a planar graph with $n(\geq 3)$ vertices and m edges, then $m \leq 3n - 6$. Moreover the equality holds if G is maximal planar.*

Proof We can assume without loss of generality that G is a maximal planar graph; otherwise add new edges without increasing n so that the resulting graph is maximal planar. Consider a plane embedding of G. Every face is bounded by exactly three edges, and each edge is on the boundaries of two faces. Therefore, counting up the edges around each face, we have $3f = 2m$. Applying Theorem 6.3.2, we obtain $m = 3n - 6$. □

Corollary 6.3.4 *Let G be a planar graph. Then G has a vertex of degree at most 5.*

Proof The claim is trivially true for planar graph of six or less vertices. We thus consider the case where G has more than six vertices. Assume that each vertex of G has degree at least 6. Then $6n \leq 2m$ holds for G. This implies $3n \leq m$. By Corollary 6.3.3, $m \leq 3n - 6$. Thus $3n \leq 3n - 6$ would hold, which is a contradiction. □

In fact, every planar graph of four or more vertices has at least four vertices of degree five or less as stated in the following lemma.

Lemma 6.3.5 *Every maximal planar graph of four or more vertices has at least four vertices of degree five or less.*

Proof By Euler's Formula, every maximal planar graph G has $3n - 6$ edges. Hence the degree-sum for G is $6n - 12$. Let v be a vertex of G. We define the *deficiency* for v by $6 - d(v)$. Then the total deficiency of the vertices of G is 12. Since G is maximal planar and any embedding of G is triangulated, G is triconnected, and hence the degree of a vertex of G is at least 3. Then the deficiency of a vertex is at most 3. Since the total deficiency is 12, there are at least four vertices of degree five or less. □

6.3.2 Dual Graph

For a plane graph G, we often construct another graph G^* called the (*geometric*) *dual* of G as follows. A vertex v_i^* is placed in each face F_i of G; these are the vertices of G^*. Corresponding to each edge e of G, we draw an edge e^* which crosses e (but no other edge of G) and joins the vertices v_i^* which lie in the faces F_i adjoining e; these are the edges of G^*. The edge e^* of G^* is called the *dual edge* of e of G. The construction is illustrated in Fig. 6.9; the vertices v_i^* are represented by small white circles, and the edges e^* of G^* by dotted lines. G^* is not necessarily a simple graph even if G is simple. Clearly, the dual G^* of a plane graph G is also a plane graph. One can easily observe the following lemma.

Lemma 6.3.6 *Let G be a connected plane graph with n vertices, m edges, and f faces, and let the dual G^* have n^* vertices, m^* edges, and f^* faces, then $n^* = f$, $m^* = m$, and $f^* = n$.*

Clearly, the dual of the dual of a connected plane graph G is the original graph G. However, a planar graph may give rise to two or more geometric duals since the plane embedding is not necessarily unique.

A connected plane graph G is called *self-dual* if it is isomorphic to its dual G^*. The graph G in Fig. 6.10 drawn with black vertices and solid edges is a self-dual graph where G^* is drawn with white vertices and dotted edges.

A *weak dual* of a plane graph G is the subgraph of the dual graph of G whose vertices correspond to the inner faces of G.

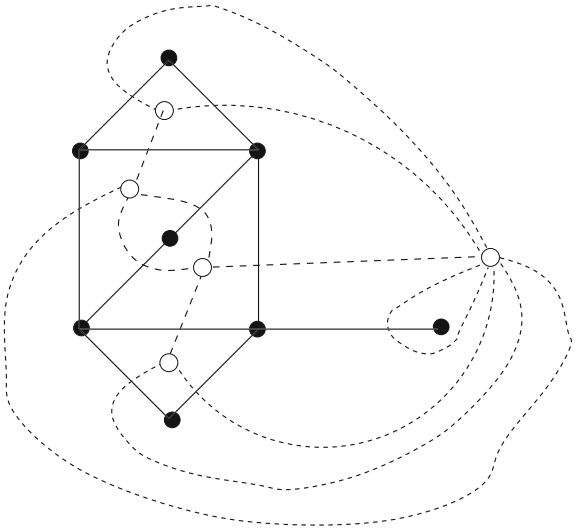

Fig. 6.9 A plane graph G drawn by *solid lines* and its dual graph G^* drawn by *dotted lines*

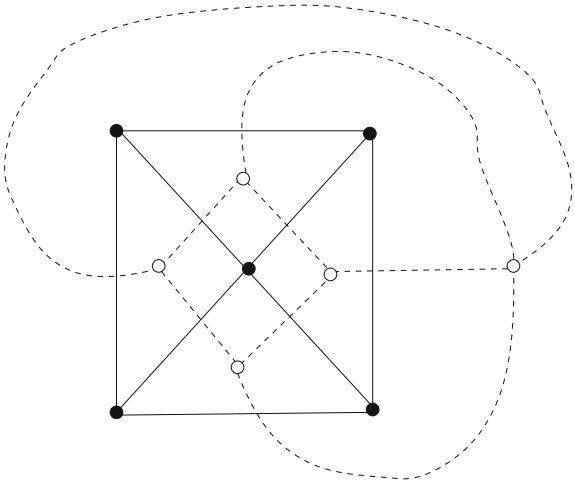

Fig. 6.10 A self-dual plane graph G and its dual graph G^*

6.4 Thickness of Graphs

In a Printed Circuit Board (PCB), we cannot print a circuit with wire crossings. Thus, a planar drawing of the circuit is required. Since not all graphs are planar, every circuit cannot be drawn on a single PCB. In such a case multiple PCBs are used, and it is desirable to use the PCBs for a circuit as few as possible. For a general circuit, we may need to know how many PCBs are needed to complete the entire circuit. This notion can be expressed by the term "thickness of a graph." The *thickness* $t(G)$ of a graph G is defined to be the smallest number of planar graphs that can be superimposed to form G. A lower bound for the thickness of a graph can be obtained from Corollary 6.3.3 as in the following theorem.

Theorem 6.4.1 *Let G be a simple graph with $n(\geq 3)$ vertices and let $t(G)$ be the thickness of G. Then $t(G) \geq \lceil \frac{m}{(3n-6)} \rceil$.*

6.5 Straight-Line Drawings of Planar Graphs

By definition every planar graph has a planar embedding. In a planar embedding, an edge is allowed to draw as a simple curve. We call a drawing of a planar graph a *straight-line drawing* if every edge is drawn as a straight-line segment. Now the question is: does every planar graph admit a straight-line drawing? The question was answered affirmatively independently by Klaus Wagner [5], Fáry [6], and S.K. Stein [7], as in the following theorem.

Theorem 6.5.1 *Every simple planar graph has a straight-line drawing.*

Proof We can assume that G is a maximal planar graph; otherwise, we add edges to make G maximal. We take a plane embedding of G. Since G is maximal planar, each face of G is a triangle. Let a, b, and c be the three vertices on the outer face of G. Using induction on the number of vertices, we show that G has a straight-line drawing where a, b, and c are drawn as the outer face. If $n = 3$, the claim is trivially true. We thus assume that $n \geq 4$, and the claim is true for less than n vertices. By Lemma 6.3.5 G has at least four vertices of degree five or less. Then G has an inner vertex v of degree five or less (see Fig. 6.11(a)). Let G' be the graph obtained by deleting v from G and triangulating the face formed by deleting v (see Figs. 6.11(a) and (b)). Then G' is maximal planar and have $n-1$ vertices. By induction hypothesis, G' has a straight-line drawing where a, b, and c are drawn on outer face, as illustrated in Fig. 6.11(d). We remove the drawing of the added edges. Then draw v inside the

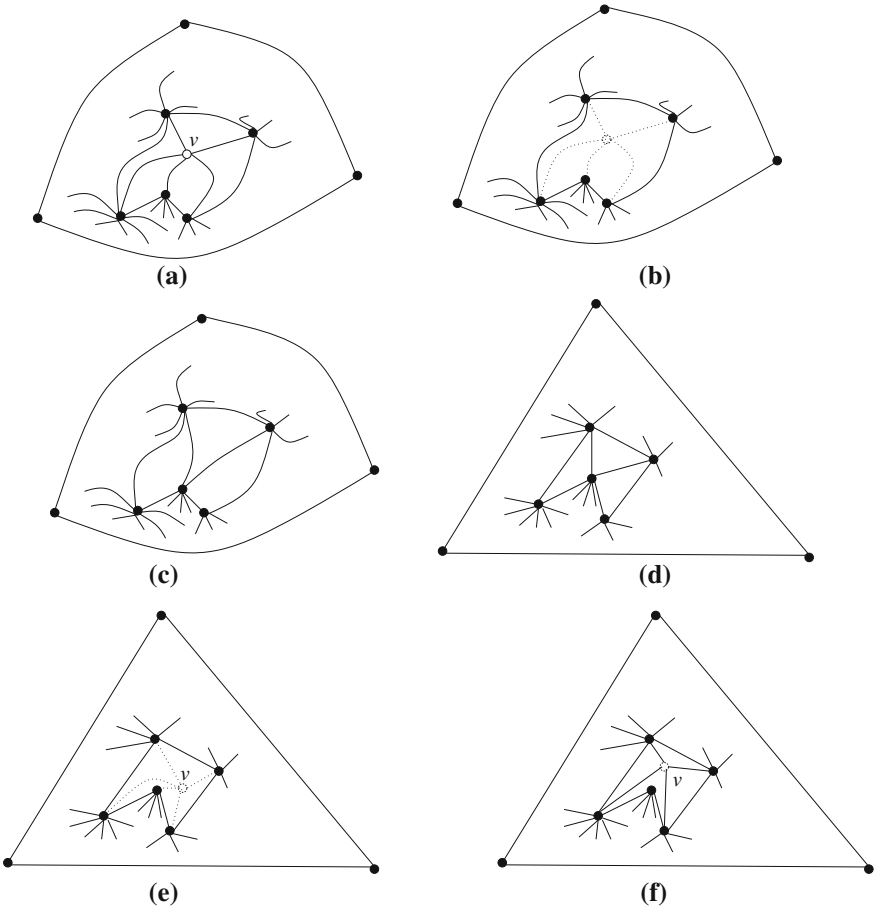

Fig. 6.11 Illustration for straight-line drawing

polygon P formed by the neighbors of v (see Figs. 6.11(e) and (f)) as follows. Since P has at most five vertices, by art gallery theorem[1] we can put v so that v is visible from all of its neighbors. We thus can draw the each edge from v to its neighbors using a straight line segment and obtain a straight-line drawing of G where a, b, and c are drawn as the outer face. □

A straight-line drawing of a planar graph is called a *straight-line grid drawing* if every vertex is drawn as a grid point of an integer grid. Finding a straight-line grid drawing of a planar graph on a grid of polynomial size was a challenging problem for a long time. In 1990, it was shown that every planar graph of n vertices has a straight-line grid drawing on a grid of size $O(n^2)$ [9, 10]. Interested readers can find more on planar graph drawing in [4].

Bibliographic Notes

The books [2, 4, 11, 12] were used in preparing this chapter.

Exercises

1. Show that Petersen graph is nonplanar. (Use Kuratowsk's Theorem.)
2. Using Kuratowsk's Theorem show that the two graphs in Fig. 6.12 are nonplanar.
3. Show that $m \leq 2n - 4$ for a planar bipartite graph of n vertices and m edges.

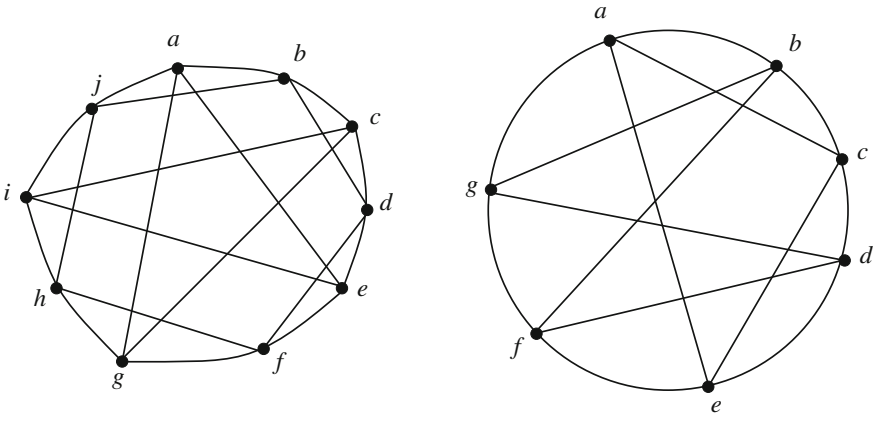

Fig. 6.12 Nonplanar graphs

[1] Chvátal's art gallery theorem is a classical result from combinatorial geometry which states that $\lfloor n/3 \rfloor$ guards are always sufficient and sometimes necessary to guard a simple polygon with n vertices [8].

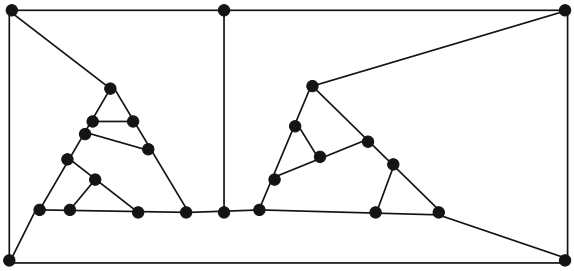

Fig. 6.13 3-legged cycles

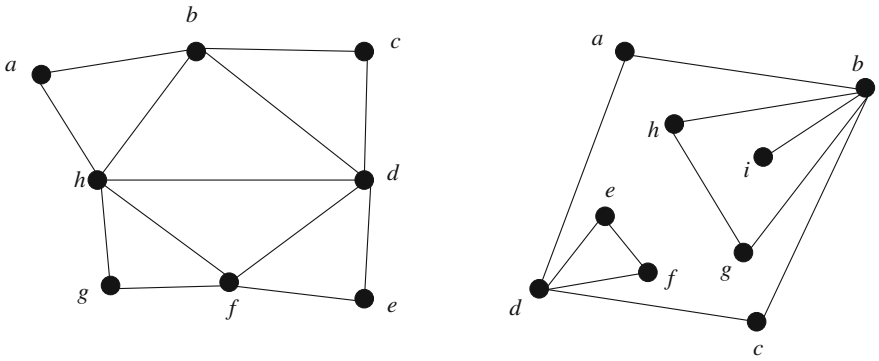

Fig. 6.14 Plane graphs

4. Identify all 3-legged cycles in the plane graph in Fig. 6.13. How many of them are minimal?
5. Find a set of three independent 3-legged cycles in Fig. 6.13 which are not minimal.
6. Construct all plane embeddings of each of the graphs in Fig. 6.14.
7. Show that dual graph of a maximal plane graph is a cubic graph.
8. Show that every graph with at most three cycles is planar.
9. Let G be a simple planar graph with no C_3. Show that G has a vertex of degree at most three.
10. Can you construct a plane graph whose dual graph is disconnected? Justify your answer.
11. Give an example of a self-dual graph of seven vertices.
12. Prove that a plane connected graph is bipartite if and only if its dual is an Eulerian graph.

References

1. Kuratowski, K.: Sur le probleme des courbes gauches en topologie. Fund. Math. **15**, 271–283 (1930)
2. Nishizeki, T., Chiba, N.: Planar Graphs: Theory and Algorithms. North-Holland, Amsterdam (1988)
3. Whitney, H.: 2-isomorphic graphs. Amer. J. Math. **55**, 245–254 (1933)
4. Nishizeki, T., Rahman, M.S.: Planar Graph Drawing. World Scientific, Singapore (2004)
5. Wagnar, K.: Bemerkungen zum vierfarbenproblem. Jahresbericht der Deutschen Mathematiker-Vereinigung **46**, 26–32 (1936)
6. Fary, I.: On straight-line representation of planar graphs. Acta. Sci. Math. **11**, 229–233 (1948)
7. Stein, S.K.: Convex maps. Proc. of American Math. Society **2**(3), 464–466 (1951)
8. Chvátal, V.: A combinatorial theorem in plane geometry. J. Combin, Theory Ser. **B 18**, 39–41 (1975)
9. de Fraysseix, H., Pach, J., Pollack, R.: How to draw a planar graph on a grid. Combinatorica **10**, 41–51 (1990)
10. Schnyder, W.: Embedding planar graphs on the grid. In: Proceedings of First ACM-SIAM Symposium on Discrete Algorithms, San Francisco, pp. 138–148 (1990)
11. Wilson, R.J.: Introduction to Graph Theory, 4th edn. Longman, London (1996)
12. Clark, J., Holton, D.A.: A First Look at Graph Theory. World Scientific, Singapore (1991)

Chapter 7
Graph Coloring

7.1 Introduction

Probably graph coloring concept naturally arose from its application in map coloring: given a map containing several countries, we wish to color the countries in the map in such a way that neighboring countries receive different colors to make the countries distinct. A map can be treated as a planar graph, and hence the problem was thought on graphs. In addition to map coloring, coloring graphs has found its applications in many other areas and has become an active area of research in graph theory for many years.

7.2 Vertex Coloring

A *vertex coloring* of a graph is an assignment of colors to the vertices so that adjacent vertices have distinct colors. A *k-coloring* of a graph uses at most k colors. A 5-coloring of a graph is illustrated in Fig. 7.1(a), where positive integer designate colors. The vertex coloring of graph in Fig. 7.1(a) is not optimal since there is another vertex coloring of the graph with three colors as illustrated in Fig. 7.1(b). The smallest integer k such that a graph G has a k-coloring is called the *chromatic number* of G and is denoted by $\chi(G)$. If $\chi(G) = k$, the graph G is said to be *k-chromatic*. Since the graph G in Fig. 7.1(a) has a triangle, one can observe that G cannot have a vertex coloring with less than three colors, and hence $\chi(G) = 3$ for the graph in Fig. 7.1(a). It is clear that $\chi(K_n) = n$. On the other hand, if G is a null graph then $\chi(G) = 1$. Every bipartite graph is 2-chromatic since to color the vertices in an independent set, we need only one color. Since a tree is a bipartite graph, every tree is 2-chromatic. The definition of k-coloring implies that the k-coloring of a graph $G = (V, E)$ partitions the vertex set V into k independent set $V_1, V_2, \ldots, V_k$ such that $V = V_1 \cup V_2 \cup \ldots V_k$. The independent sets $V_1, V_2, \ldots, V_k$ are called the *color*

M.S. Rahman, *Basic Graph Theory*, Undergraduate Topics in Computer Science,
DOI 10.1007/978-3-319-49475-3_7

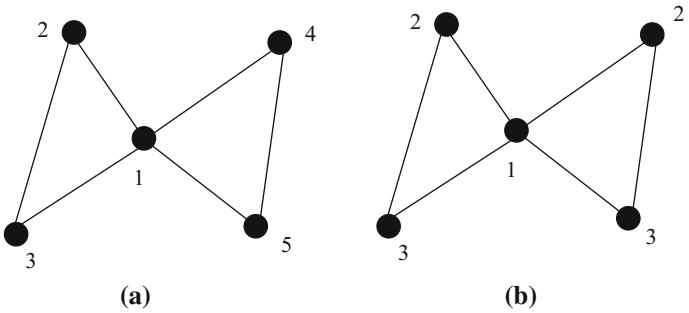

Fig. 7.1 **(a)** A 5-coloring and **(b)** a 3-coloring of a graph

classes. The *vertex coloring problem*, i.e., coloring vertices of a graph G with $\chi(G)$ colors, is a NP-hard problem [1].

We can easily prove the following claim.

Lemma 7.2.1 *Every simple graph of the maximum degree* Δ *has a* $(\Delta + 1)$*-coloring.*

Proof We prove the claim using induction on the number of vertices of G. Every simple graph of $n \leq 2$ vertices has maximum degree 1 and the graph is 2-colorable. We now assume that $n > 2$ and the claim is true for every graph of less than n vertices. Let G be a simple graph of n vertices, and let v be a vertex in G. Let G' be the graph obtained from G by deleting v. Then G' has $n - 1$ vertices and has maximum degree at most Δ. By induction hypothesis, G' is $(\Delta + 1)$ colorable, where the neighbors of v in G' used at most Δ colors. Then a $(\Delta + 1)$-coloring of G can be obtained from the $(\Delta + 1)$-coloring of G' by coloring v with a different color from the vertices adjacent to v. □

Based on the proof of Lemma 7.2.1, one can develop a simple recursive algorithm to find a vertex coloring of a graph with $(\Delta + 1)$ colors. There are some greedy heuristics which find vertex coloring of a graph with less number of colors (in the worst case they need $(\Delta + 1)$ colors). The basic idea of those greedy algorithms is to take an initial ordering of the vertices and assign colors to the vertices following the ordering [2, 3]. An unused color is assigned to a vertex only when no used color is available to color the vertex in that step.

The claim in Lemma 7.2.1 is made stronger by Brooks in 1941 using more careful treatment as in the following Theorem.

Theorem 7.2.2 *Every simple graph of the maximum degree* Δ *which is not a complete graph has a* Δ*-coloring.*

The proof of Theorem 7.2.2 is a bit involved, and we omit the proof here.

A planar graph may be colorable with a less number of colors, which does not depend on Δ, as we see in the following lemma.

Lemma 7.2.3 *Every simple planar graph is 6-colorable.*

Proof We prove the claim by induction on the number of vertices. The claim is trivially true for simple planar graphs of at most six vertices. Assume that G is a simple planar graph of $n > 6$ vertices, and every simple planar graph of $n - 1$ vertices is 6-colorable. By Corollary 6.3.4, G has a vertex v of degree at most 5. Let G' be the graph obtained from G by deleting v. Then G' is a planar graph with $n - 1$ vertices. By induction hypothesis, G' is 6-colorable, where the neighbors of v in G' used at most 5 colors. Then a 6-coloring of G can be obtained by from the 6-coloring of G' by coloring v with a different color from the vertices adjacent to v. □

Thus whatever the maximum degree of G is, G is 6-colorable. With the help of edge contraction operation, a stronger result as in Theorem 7.2.4 can be proved.

Theorem 7.2.4 *Every simple planar graph is 5-colorable.*

Proof We prove the theorem by induction on the number n of vertices. The claim is trivially true for simple planar graphs of at most five vertices. Thus suppose that G is a simple planar graph of $n, n > 5$, vertices, and that all simple planar graphs with less than n vertices are 5-colorable. By Corollary 6.3.4, G has a vertex v of degree at most 5. We have two cases to consider.

Case 1: $d(v) \leq 4$. The deletion of v leaves us with a graph $G - v$ having $n - 1$ vertices which is 5-colorable by induction hypothesis. Then v can be colored with any color not used by the (at most four) neighbors, completing the proof.

Case 2: $d(v) = 5$. Since G planar, Kuratowski's theorem implies that there are two vertices among five neighbors of v which are nonadjacent, as illustrated in Fig. 7.2(a). Let x and y be two neighbors of v which are not adjacent. We now contract the two edges vx and vy and obtain a simple graph, as illustrated in Fig. 7.2(b). The resulting graph is a planar graph with $n - 2$ vertices, and is thus 5-colorable. We now reinstate the two edges, giving both x and y the color originally assigned to v. A 5-coloring of G is then obtained by coloring v with a color different from the (at most four) colors assigned to the neighbors of v. □

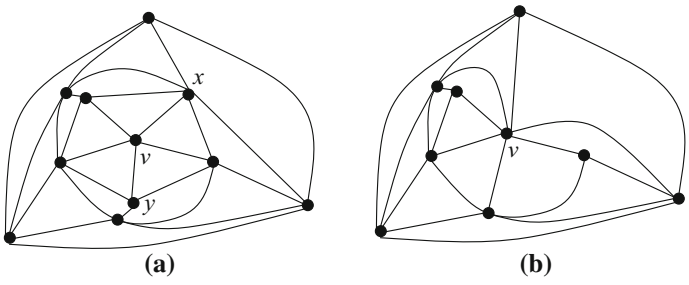

Fig. 7.2 Illustration for Case 2 in the proof of Theorem 7.2.4

Based on the proof of Theorem 7.2.4, one can easily design an $O(n^2)$ time recursive algorithm which finds a 5-coloring of a planar graph. Linear algorithms are also known for 5-coloring of planar graphs [4–6].

In 1878, Cayley presented a conjecture that every plane graph is 4-colorable. This conjecture became a famous open problem for about a century. Apple, Haken, and Koch finally proved the conjecture in 1976 using a lengthy proof and the conjectured turned into the 4-*color theorem* [7].

A graph G is called *k-critical* if $\chi(G) = k$ and $\chi(G - v) < k$ for each vertex v of G. Thus a graph is k-critical if it needs k colors but each of its vertex deleted subgraphs can be colored with less than k colors. Observe that K_1 is the only one 1-critical graph and K_2 is the only one 2-critical graph. Every odd cycle is a 3-critical graph. However, there is no easy characterization of k-critical graphs for $k \geq 4$. The following observation is due to Direc [8].

Lemma 7.2.5 *Let G be a k-critical graph. Then G is connected.*

Proof Assume for a contradiction that a k-critical graph G is not connected. Since $\chi(G) = k$, there is a connected component H of G such that $\chi(H) = k$. Let v be a vertex of G not in H. Then H is also a connected component in $G - v$ for which $\chi(G) = k$. Then $\chi(G - v) = \chi(H) = k$. This contradicts to the assumption that G is a k-critical graph. □

A *clique* in a graph G is a subgraph of G that is a complete graph. The number of vertices in a largest clique of G is called the *clique number* of G and denoted by $\omega(G)$. Let G' be a vertex-induced subgraph of G. Then one can easily observe that $\chi(G') \geq \omega(G')$. A simple graph G is called a *perfect graph* if $\chi(G') = \omega(G')$ for every vertex-induced subgraphs G' of G. Not all simple graphs are perfect graphs. For example, an odd cycle G of five or more vertices is not a perfect graph since $\chi(G) = 3$ and $\omega(G) = 2$. However, the class perfect graphs is a large class of graphs for which greedy coloring algorithms work well.

7.3 Edge Coloring

An *edge coloring* of a graph is an assignment of colors to the edges so that the edges incident to a vertex have distinct colors. A graph G is *k-edge colorable* if G has an edge coloring with k colors. The *chromatic index* of a graph G is k if G is k-edge colorable but not $(k - 1)$-edge colorable. The chromatic index of a graph is denoted by $\chi'(G)$. The definition of k-edge coloring implies that the k-edge coloring of a graph $G = (V, E)$ partitions the edge set E into k matchings $E_1, E_2, \ldots, E_k$ such that $E = E_1 \cup E_2 \cup \ldots E_k$. The sets $E_1, E_2, \ldots, E_k$ are called the *color classes*.

It is trivial observation that $\chi'(G) \geq \Delta$, where Δ is the maximum degree of graph G. The following theorem known as Vizing's theorem [9] gives very sharp bounds for the chromatic index of a simple graph G.

Theorem 7.3.1 *Let G be a simple graph with the maximum degree Δ. Then $\Delta \leq \chi'(G) \leq \Delta + 1$.*

Which graphs have chromatic index Δ and which have chromatic index $\Delta + 1$ are not known. However, chromatic index for bipartite graphs is given by the following theorem [10].

Theorem 7.3.2 *Let G be a bipartite graph with the maximum degree Δ. Then $\chi'(G) = \Delta$.*

Proof We prove the theorem by induction on m, the number of edges of G. The claim is trivially true for $m = 1$. We thus assume that $m > 1$ and the claim is true for every bipartite graph of less than m edges. Let G be a bipartite graph of m edges with the maximum degree Δ, and let (u, v) be an edge in G. Let G' be the graph obtained by deleting (u, v) from G. Then by induction hypothesis, G' has an edge coloring with a set S of Δ colors. Since G' is obtained from G by deleting (u, v), each of u and v has degree less than Δ in G'. Then there is a color in S which is not used to color the edges incident to u and there is a color in S which is not used to color the edges incident to v in G'. If there is some color, say b, which is not used to color the incident edges of both u and v in G', we can extend the coloring of G' to a coloring of G with Δ colors by coloring (u, v) with b. Otherwise, let b be a color in S which is not used to color the edges incident to u and let g be a color in S which is not used to color the edges incident to v in G'. Let C_{bg} be the subgraph of G' induced by the vertices reachable from u through the edges colored by b and g. C_{bg} is shown by thick edges in Fig. 7.3. Then the edge in C_{bg} incident to u is colored by g. Since G is bipartite and (u, v) is an edge in G, u and v are in different bipartite set. Let V_1 be the bipartite set which contains v. Then any vertex in $V_1 \cap V(C_{bg})$ must have an incident edge which is colored by g. Since none of the edges incident to v has color g, v is not contained in C_{bg}. We can thus swap the colors b and g for each of the edges in C_{bg} without affecting the coloring of the rest of the edges in G', and get an edge coloring of G' with the Δ colors, where the color g is not used to color the incident edges of both u and v. We then extend the coloring of G' to a coloring of G with Δ colors by coloring (u, v) with g. □

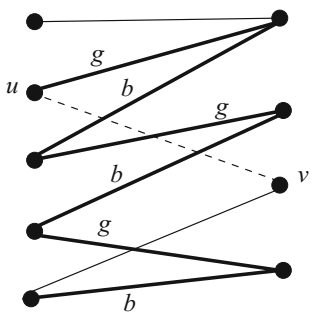

Fig. 7.3 Edge coloring of a bipartite graph

Edge coloring of a complete bipartite graph $K_{n,n}$ can be used to construct an $n \times n$ Latine square which has applications in experimental design for quality control [11].

The chromatic index for complete graphs can be computed from the following theorem.

Theorem 7.3.3 *For $n \geq 2$, $\chi'(K_n) = n$ if n is odd and $\chi'(K_n) = n - 1$ if n is even.*

Proof If $n = 2$, the complete graph K_2 has one edge and it needs one color. Hence the claim is trivially true for $n = 2$. We thus assume that $n \geq 3$.

We first assume that n is odd. We draw the vertices of K_n in the form of a regular n-gon. We color the edges along the boundary using a different color for each edge. Now each of the remaining internal edges of G is parallel to exactly one edge on the boundary. Each such edge is colored with the same color as the boundary edge. Thus, two edges have the same color if they are parallel and hence we have an edge coloring of K_n (See Fig. 7.4). Since we have used n colors, $\chi'(K_n) \leq n$. To prove $\chi'(K_n) = n$, it is remaining to show that K_n is not $(n-1)$-colorable. Assume for a contradiction that K_n has a $(n-1)$-coloring. From the definition of edge coloring,

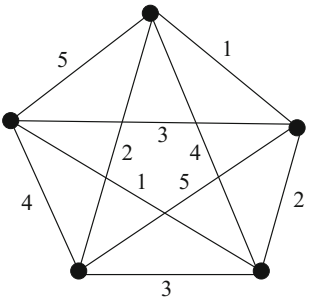

Fig. 7.4 Edge coloring of K_n for odd n

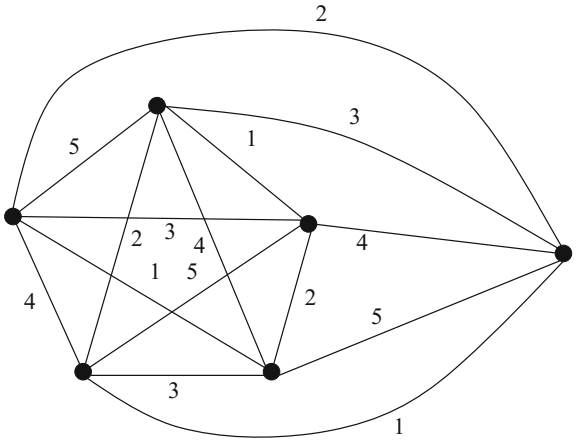

Fig. 7.5 Edge coloring of K_n for even n

the edges of one particular color form a matching in K_n. Since n is odd, such a matching can contain at most $(n-1)/2$ edges. Since K_n has a $(n-1)$-coloring, K_n can have at most $(n-1)(n-1)/2$ edges. This is a contradiction since K_n has exactly $n(n-1)/2$ edges.

We now assume that n is even. In this case, we obtain K_n by joining the complete graph K_{n-1} to a single vertex. We now color the edges of K_{n-1} as described above. Then there is one color missing in each vertex and these missing colors are all different. We complete the edge coloring of K_n by coloring the remaining edges with the missing colors (See Fig. 7.5). □

7.4 Face Coloring (Map Coloring)

A *face coloring* of a plane graph is a coloring of its faces such that no two adjacent faces get the same color. A *k-face coloring* of a plane graph is a face coloring of the graph using k colors. If a plane graph admits a k-face coloring, then it is *k-face colorable*.

The four-color problem arose historically in connection with coloring maps. Given a map containing several countries, we may ask how many colors are needed to color them so that no two countries with a boundary line in common share the same color. A map coloring can be modeled as a face coloring of a plane graph where the map is represented by a plane graph and we are asked to color the faces of the plane graph in such a way that two faces having a common edge receive different colors. Of course the problem is to minimize the number of colors to be used. The following is the formal statement of the 4-color theorem [7].

Theorem 7.4.1 *Every map can be colored in four or less colors.*

The following theorem relates the face coloring problem and the vertex coloring problem in a plane graph.

Theorem 7.4.2 *A plane graph G is k-vertex colorable if and only if the dual graph of G is k-face colorable.*

7.5 Chromatic Polynomials

The *chromatic polynomial* of a graph describes the number of different proper vertex colorings that the graph have using a fixed number of colors. Clearly, for any graph G with at least one edge, there is no way to vertex-color G properly using one color. If G is not bipartite, there is no way to vertex-color G properly using two colors. In general, if $k < \chi(G)$, there is no way to vertex color G properly using k colors. Therefore, if for all natural numbers k we know the number of k-vertex colorings of G, then we can determine the chromatic number $\chi(G)$. That is why we are interested in determining the number of k-colorings of a graph for all values of k.

Let G be a simple graph, and let $P_G(k)$ be the number of ways of coloring the vertices of G with k colors so that no two adjacent vertices have the same color. If G is a path of three vertices, $P_G(k) = k(k-1)^2$, since the middle vertex can be colored in k ways and each end vertex can be colored in any of $(k-1)$ ways. Observe that $P_G(k)$ for the path graph P_3 is a polynomial of k. In fact, $P_G(k)$ for any simple graph G can be expressed as a polynomial of the number k of the colors. For this reason, the polynomial of k for expressing $P_G(k)$ is known as the chromatic polynomial of G. The chromatic polynomial has some beautiful properties [12]. For example, the degree of the chromatic polynomial is equal to the number of vertices of G.

Like recursive counting of spanning trees of a graph, we can compute $P_G(k)$ recursively in the following theorem:

Theorem 7.5.1 *Let $G = (V, E)$ be a simple graph and $e \in E$. For any integer k, we have $P_G(k) = P_{G-e}(k) - P_{G\setminus e}(k)$.*

Proof Let $e = (u, v)$. Consider the vertex colorings of $G - e$ with k colors. These k-colorings are split into two sets of colorings. In one set of colorings, u and v receive the same color and in the other set of colorings, u and v receive different colors. The number of colorings in the first set is equal to the number of k-colorings of $G\setminus e$ and the number of k-colorings in the second set is equal to the number of k-colorings of G. Therefore $P_{G-e}(k) = P_G(k) + P_{G\setminus e}(k)$. This implies the claim. □

7.6 Acyclic Coloring

Graph coloring is a well-studied area of graph theory. Depending on the application area, many variants of graph coloring have been introduced. In this section, we study such a variant of graph coloring known as "acyclic coloring" which has applications in Hessian computation [13, 14] as well as in coding theory [15]. Dujmović et al. [16] have used acyclic coloring of planar graphs to obtain upper bounds on the volume of three-dimensional straight-line grid drawings of planar graphs.

An *acyclic coloring* of a graph G is a vertex coloring of G such that no cycle of G is bichromatic. That is, the vertices on a cycle in G cannot be colored with exactly two colors in an acyclic coloring of G. An acyclic k-coloring of G is an acyclic coloring of G using at most k colors. The smallest number of colors needed to acyclically color the vertices of a graph is called its *acyclic chromatic number*. Figure 7.6 illustrates an acyclic coloring of graph with five colors. Acyclic coloring was first studied by Grünbaum in 1973 [17]. He proved an upper bound of nine for the acyclic chromatic number of any planar graph G. He also conjectured that five colors are sufficient for acyclic coloring of any planar graph. Testing acyclic 3-colorability is NP-complete for planar bipartite graphs with the maximum degree 4, and testing acyclic 4-colorability is NP-complete for planar bipartite graphs with the maximum degree 8 [18].

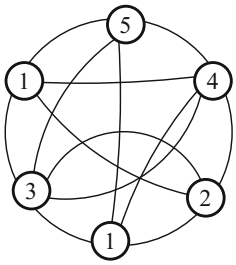

Fig. 7.6 An acyclic coloring of a graph using five colors

In the rest of this section, we will see some preliminary observation on acyclic 3-coloring from [19]. These observations are related to subdivisions, independent sets, and ear decompositions that we have learnt in earlier chapters.

Let $P = u_0, u_1, u_2, \ldots, u_{l+1}, l \geq 1$, be a path of G such that $d(u_0) \geq 3, d(u_1) = d(u_2) = \cdots = d(u_l) = 2$, and $d(u_{l+1}) \geq 3$. Then we call the subpath $P' = u_1, u_2, \ldots, u_l$ of P a *chain* of G. Let G' be a subdivision of G. A vertex v of G' is called an *original vertex* if v is a vertex of G; otherwise, v is called a *division* vertex.

The following lemma is on expanding acyclically 3-colorable graphs by adding chains.

Lemma 7.6.1 *Let G be a graph with two distinct vertices u and v and let G' be a graph obtained by adding a chain $w_1, \ldots, w_k$ between the vertices u and v of G. Let G be acyclically 3-colorable such that the colors of u and v are different. Then G' is acyclically 3-colorable.*

Proof In an acyclic coloring of G that colors vertices u and v differently, let the colors of vertices u and v be c_1 and c_2, respectively. For each w_i, $i = 1, 2, \ldots, k$, we assign color c_3 when i is odd and color c_1 when i is even as in Fig. 7.7. Clearly, no two adjacent vertices of G' have the same color. Therefore, the coloring of G' is a valid 3-coloring. Suppose for a contradiction that the coloring of G' is not acyclic.

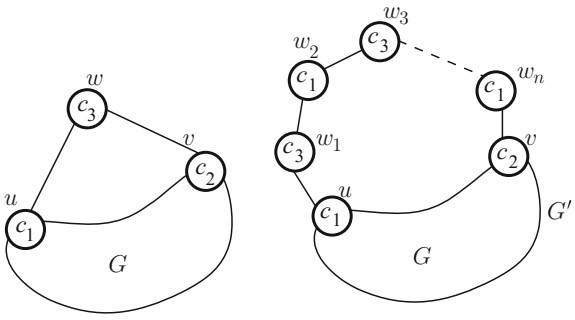

Fig. 7.7 Illustration for the proof of Lemma 7.6.1

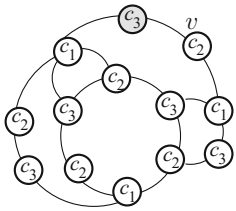

Fig. 7.8 Illustration for the proof of Lemma 7.6.2

Then G' must contain a bichromatic cycle C. The cycle C either contains the chain $u, w_1, w_2, \ldots, w_k, v$ or is a cycle in G. C cannot contain the chain since the three vertices u, v, and w_1 are assigned three different colors c_1, c_2, and c_3, respectively. Thus we can assume that C is a cycle in G. Since G does not contain any bichromatic cycle, C cannot be a bichromatic cycle, a contradiction. □

The following lemma gives a bound on the number of division vertices of a subdivision of a biconnected graph for being acyclically 3-colorable.

Lemma 7.6.2 *Let G be a biconnected graph with n vertices and let $P_1 \cup \ldots \cup P_k$ be an ear decomposition of G where each ear P_i, $2 \leq i \leq k$, contains at least one internal vertex. Then G has a subdivision G', with at most $k-1$ division vertices, that is acyclically 3-colorable.*

Proof We prove the claim by induction on k. The case $k = 1$ is trivial since P_1 is a cycle, which is acyclically 3-colorable. Therefore, we assume that $k > 1$ and that the claim is true for the graphs $P_1 \cup \ldots \cup P_i$, $1 \leq i \leq k-1$. By induction, $G - P_k$ has a subdivision G'' that is acyclically 3-colorable and that has at most $k-2$ division vertices. Let the end vertices of P_k in G be u and v. If u and v have different colors in G'', then we can prove in a similar way as in the proof of Lemma 7.6.1 that G has a subdivision G' that is acyclically 3-colorable and that has the same number of division vertices as G'', which is at most $k-2$. Otherwise, u and v have the same color in G''. Let the color of u and v be c_1 and let the two other colors in G'' be c_2 and c_3. If P_k contains more than one internal vertices, then we assign the colors c_2 and c_3 to the vertices alternately. If P_k contains only one internal vertex v, then we subdivide an edge of P_k once. We color v with c_2 and the division vertex with c_3 as shown in Fig. 7.8. In both cases, we can prove in a similar way as in the proof of Lemma 7.6.1 that G' has no bichromatic cycle. Moreover, the number of division vertices in G' is at most $(k-2)+1 = k-1$. □

The following lemma exhibits a relationship of cyclically 3-colorable graphs with independent sets:

Lemma 7.6.3 *Let S be an independent set of a graph G. If $G - S$ is acyclic then G is acyclically 3-colorable.*

Proof If $G - S$ is acyclic then $G - S$ is a tree or a forest and hence, it is 2-colorable. Color the vertices of $G - S$ with colors c_1 and c_2. Add the vertices of S to $G - S$

and assign the vertices color c_3. Since S is an independent set, a cycle in G contains at least one edge (u_1, u_2) from $G - S$ and at least one vertex u_3 from S. Since, by the coloring method given above, u_1, u_2, and u_3 have different colors, there is no bichromatic cycle in G. □

Bibliographic Notes

The books [11, 12, 20] were used in preparing this chapter.

Exercises

1. Obtain vertex coloring of the graphs in Fig. 7.9 with the minimum number of colors.
2. Determine the chromatic number of Petersen graph.
3. Show that dual graph of a maximal plane graph is a cubic graph.
4. Let G be a simple planar graph containing no triangle. Then show that $\chi(H) \leq 4$.
5. Show that for a tree of n vertices $P_G(k) = k(k-1)^{n-1}$.
6. Construct a 4-critical graph.
7. Let G be a k-critical graph. Then show that the degree of every vertex of G is at least $k - 1$.

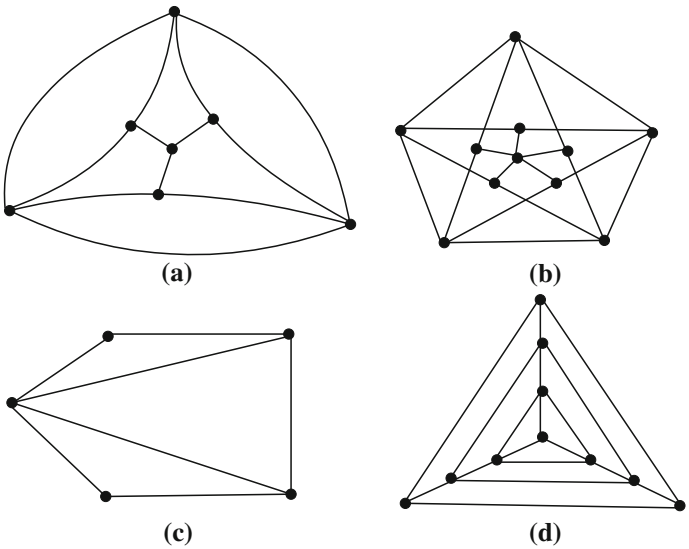

Fig. 7.9 Graphs

References

1. Garey, M.R., Johnson, D.S.: Computers and Intractability. W.H. Freeman and Company, New York (1979)
2. Matula, D.W., Marble, G., Isaacson, J.D.: Graph colouring algorithms. In: Read, R.C. (ed.) Graph Theory and Computing, pp. 109–122. Academic Press, New York (1972)
3. Welsh, D.J.A., Powell, M.B.: An upper bound for chromatic number of a graph and its application to timetabling problem. Comput. J. **10**, 85–86 (1967)
4. Chiba, N., Nishizeki, T., Saito, N.: A linear 5-coloring algorithm of planar graphs. J. Algorithms **2**(4), 317–327 (1981)
5. Matula, D.W., Schiloach, Y., Tarjan, R.E.: Two linear algorithms for five coloring of planar graphs. STAN-CS-80-830 (1980)
6. Frederickson, G.N.: On linear-time algorithms for five-coloring planar graphs. Inform. Process. Lett. **19**(5), 219–224 (1984)
7. Appel, K., Haken, W.: Every map is four colourable. Bull. Am. Math. Soc. **82**, 711–712 (1976)
8. Dirac, G.A.: A property of 4-chromatic graph and some remarks on critical graphs. J. Lond. Math. Soc. **27**, 85–92 (1952)
9. Vizing, V.G.: On an estimate of the chromatic class of a p-graph, Diskret. Analiz. **3**, 25–30 (1964)
10. Konig, D.: Uber graphen und ihre anwendung auf determinantentheorie und mengenlehre. Math. Ann. **77**(4), 453–465 (1916)
11. Clark, J., Holton, D.A.: A First Look at Graph Theory. World Scientific, Singapore (1991)
12. Agnarsson, G., Greenlaw, R.: Graph Theory: Modeling, Applications and Algorithms. Pearson Education Inc., New Jersey (2007)
13. Gebremedhin, A.H., Tarafdar, A., Manne, F., Pothen, A.: New acyclic and star coloring algorithms with application to computing hessians. SIAM J. Sci. Comput. **29**(3), 1042–1072 (2007)
14. Gebremedhin, A.H., Tarafdar, A., Pothen, A., Walther, A.: Efficient computation of sparse hessians using coloring and automatic differentiation. INFORMS J. Comput. **21**(2), 209–223 (2009)
15. Serra, O., Zémor, G.: Cycle codes of graphs and mds array codes. Electron. Notes Discret. Math. **34**, 95–99 (2009)
16. Dujmović, V., Morin, P., Wood, D.R.: Layout of graphs with bounded tree-width. SIAM J. Comput. **34**, 553–579 (2005)
17. Grünbaum, B.: Acyclic colorings of planar graphs. Isr. J. Math. **14**, 390–408 (1973)
18. Ochem, P.: Negative results on acyclic improper colorings. In: Proceedings of European Conference on Combinatorics (EuroComb 05), pp. 357–362 (2005)
19. Mondal, D., Nishat, R.I., Whitesides, S., Rahman, M.S.: Acyclic colorings of graph subdivisions revisited. J. Discret. Algorithms **16**, 90–103 (2012)
20. Wilson, R.J.: Introduction to Graph Theory, 4th edn. Longman, London (1996)

Chapter 8
Digraphs

8.1 Introduction

A graph is usually called a directed graph or a digraph if its edges have directions. The concept of directed graphs or digraphs has many applications in solving real-world problems. For example, flow networks as well as electrical networks are represented by digraphs.

8.2 Digraph Terminologies

A *directed graph* or *digraph* $D = (V, A)$ consists of a nonempty finite set $V(D)$ of vertices and a (multi) set $A(D)$ of ordered pairs of elements of $V(D)$ called *arcs* or *directed edges*. If $A(D)$ is a multiset, then D is called a *multidigraph*. Figure 8.1(a) illustrates a simple digraph whereas the graph in Fig. 8.1(b) is a multidigraph. For $u, v \in V$, an arc $a = (u, v) \in A$ is denoted by uv and implies that a is directed from u to v. For an arc uv, u is called the *tail* of uv and v is called the *head* of uv. For the arc cd in Fig. 8.1(a), c is the tail and d is the head. We say the edge uv is leaving u and terminating at v. A digraph is represented by a diagram which is similar to the representation of a graph where each arc contains an arrow directed from tail to head, as illustrated in Fig. 8.1.

The concept of the degree of a vertex for graphs also extends to digraphs. The *indegree* of a vertex v of a digraph is the number of directed edges terminating at v, and the *outdegree* of v is the number of directed edges leaving v. In the digraph in Fig. 8.1(b), the indegree of e is 2 and the outdegree of e is 3; the indegree of c is 2 and the outdegree of c is 3. The *degree* of v is the sum of its indegree and outdegree. A vertex with indegree 0 is a *source* and with outdegree 0 is a *sink.* In the digraph in Fig. 8.1(a), a is a source and f is a sink. The digraph in Fig. 8.1(b) has no sink. We now have the following lemma which is analogous to Lemma 2.2.1.

M.S. Rahman, *Basic Graph Theory*, Undergraduate Topics in Computer Science,
DOI 10.1007/978-3-319-49475-3_8

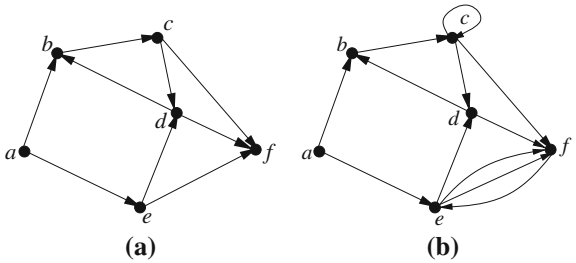

Fig. 8.1 **(a)** A simple digraph and **(b)** a multidigraph

Lemma 8.2.1 *Let $D = (V, A)$ be a digraph with n vertices and with m arcs. Then* $\sum_{v \in V} indeg(v) = \sum_{v \in V} outdeg(v) = m.$

Proof Every arc is counted exactly once while counting indegrees. Again every arc is counted once while counting outdegrees. Thus the claim holds. □

Let $D = (V, A)$ be a digraph. The graph $G = (V, E)$ is called the *underlying graph* of D if G is constructed from D such that $(u, v) \in E$ if and only if uv or vu or both are in A. The underlying graph of a digraph D is denoted by $G(D)$. The underlying graph of the digraph in Fig. 8.2(a) is shown in Fig. 8.2(b).

Let $G = (V, E)$ be an undirected graph. We call $D = (V, A)$ the *digraph corresponding to* G if D is constructed from G such that A contains both uv and vu if and only if $(u, v) \in E$. The digraph corresponding to G is denoted by $D(G)$. The digraph in Fig. 8.3(b) is corresponding to the graph in Fig. 8.3(a). An oriented graph obtained from an undirected graph $G = (V, E)$ by replacing each edge $(u, v) \in E$ by an arc uv or vu, but not both is called an *orientation of* G. An orientation of G is denoted by $O(G)$. Figures 8.3(c) and (d) illustrates two orientations of the graph in Fig. 8.3(a).

A digraph $D = (V, A)$ is said to be *complete* if A contains both uv and vu for every pair u, v of vertices in V. The *complement of a digraph* D is the simple digraph $\overline{D}$ where there is an arc from a vertex u to a vertex v if and only if there is no such arc in D. The *converse* of a digraph D, denoted by $\overleftrightarrow{D}$, is the digraph obtained from D by reversing the direction of each arc of D. Two digraphs are said to be *isomorphic*

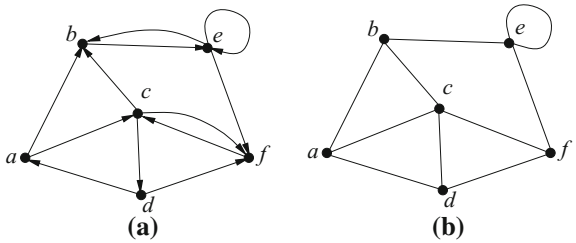

Fig. 8.2 **(a)** A digraph D and **(b)** the underlying graph $G(D)$

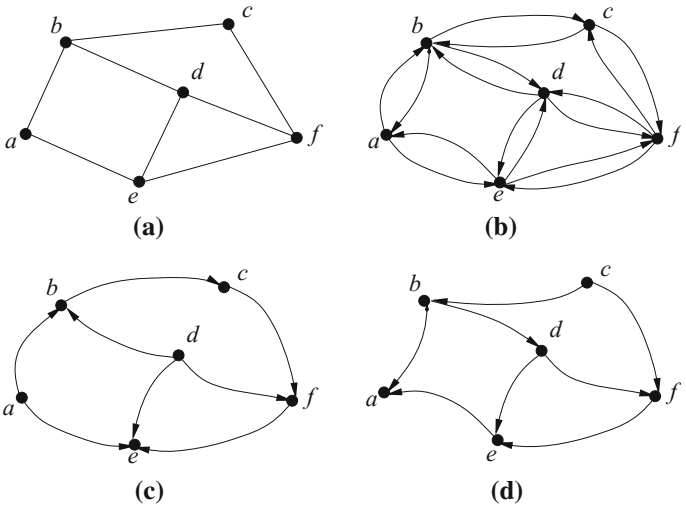

Fig. 8.3 **(a)** A graph G, **(b)** the corresponding digraph $D(G)$ and **(c)–(d)** orientations of G

if their underlying graphs are isomorphic and the direction of the corresponding arcs are same.

A *walk* in a digraph is a finite sequence of arcs of the form $v_0v_1, v_1v_2, \ldots, v_{k-1}v_k$. In an analogous way, we can define directed trails, directed paths, and directed cycles or simply, trails, paths, and cycles, if there is no possibility of confusion. A digraph D is *weakly connected* or *connected* if its underlying graph is connected. A vertex u is said to be *reachable* from a vertex v, if there is a path from v to u. The relation "is reachable from" is reflexive, transitive but not symmetric. A digraph is *strongly connected* if every pair of vertices is reachable from each other. Note that a digraph is connected if its underlying graph is connected. A *strongly connected component* H of a digraph D is a subdigraph of D which is strongly connected and is not a proper subdigraph of any other strongly connected subdigraph of D.

8.3 Eulerian Digraphs

A digraph D is said to be *Eulerian* if it contains a closed trail which traverses every arc of D. The following theorem characterizes Eulerian digraphs.

Theorem 8.3.1 *A connected digraph $D(V, A)$ is Eulerian if and only if $indeg(v) = outdeg(v)$ for every vertex $v \in V$.*

Proof We first assume that D is Eulerian. Then D contains a Eulerian circuit C. In traversing C every time a vertex v is encountered we pass along an arc incident towards v and then an arc incident away from v. This is true for all the vertices of D including the initial vertex. Initially, we start traversing by an arc incident away

from the start vertex u and end the traversing by an arc incident towards u. Thus for every vertex $indeg(v) = outdeg(v)$.

We now assume that $indeg(v) = outdeg(v)$ for every vertex of D. Then we can construct a Eulerian circuit in D using a method similar to one in the proof of the sufficiency of Theorem 3.2.1. □

8.4 Hamiltonian Digraphs

A digraph D is *Hamiltonian* if there is a (directed) cycle that includes every vertex of D. A (directed) path P in a digraph D is a *Hamiltonian path* if P contains every vertex of D. The following theorem due to Meyneil [1] gives a sufficient condition for a digraph to be a Hamiltonian.

Theorem 8.4.1 *Let D be a strongly connected digraph of n vertices. Then D is Hamiltonian if $d(u) + d(v) \geq 2n - 1$ for every pair of nonadjacent vertices u and v.*

8.5 Digraphs and Tournaments

Assume that every team in a round-robin tournament plays every other team and no ties are allowed. The result of such a tournament can be represented by a digraph where an arc uv indicates u beat v. Clearly, there is no loop in such a digraph and there is exactly one edge between any two distinct vertices. A complete antisymmetric digraph or a complete oriented graph is called a *tournament*. In fact a tournament is an orientation of K_n. The following result is due to Redei [2].

Theorem 8.5.1 *Every tournament T contains a Hamiltonian path.*

Proof We prove the claim using an induction on the number n of vertices in T. The claim is trivially true for $n = 1, 2$ or 3. Assuming that $n \geq 4$ the claim is true for all tournaments with fewer than n vertices. Let v be any vertex of T. Then $T - v$ is a tournament and has $n - 1$ vertices. By induction hypothesis $T - v$ contains a Hamiltonian path. Let $P = v_1, v_2, v_3, \ldots, v_{n-1}$ be the Hamiltonian path in $T - v$. If there is an arc (v, v_1) or (v_{n-1}, v) in T then we can obtain a Hamiltonian path in T

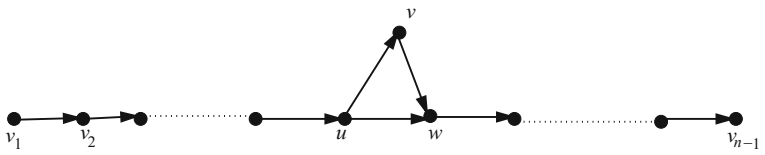

Fig. 8.4 Illustration for the proof of Theorem 8.5.1

by including v to P. We thus assume that T has neither (v, v_1) nor (v_{n-1}, v). Then there is a vertex $u \neq v_{n-1}$ such that T contains an arc (u, v). Assume that u is the last such vertex along the path P and w be the vertex next to u on P. Then there is an arc (u, v) and an arc (v, w) in T. Then we can find a Hamiltonian path in T from P by replacing arc (u, w) with the path u, v, w, as illustrated in Fig. 8.4. □

In a tournament T, the outdegree of a vertex represents the score of the corresponding team. Thus the *score sequence* of a tournament is the listing of outdegrees of vertices. It is listed in nondecreasing order.

8.6 Flow Networks

A *flow network* $\mathcal{N}$ is a weakly connected, simple directed graph in which every arc a of $\mathcal{N}$ has been assigned a nonnegative integer $\mu(a)$, called the *capacity* of a. A vertex s of a network $\mathcal{N}$ is called a *source* if it has indegree zero while a vertex t of $\mathcal{N}$ is called a *sink* if it has outdegree zero. Any other vertex of $\mathcal{N}$ is called an *intermediate vertex*. Figure 8.5 shows a flow network where s is the source, t is the sink and w, x, y, z are intermediate vertices. Capacity of each arc is written beside the arc in Fig. 8.5.

A *flow* ϕ in $\mathcal{N}$ associates a nonnegative integer $\phi(a)$ with each arc a; $\phi(a)$ is called a *flow of arc* a. The flow $\phi(a)$ of each arc a must satisfy $\phi(a) \leq \mu(a)$. This is called *capacity constraint*. Furthermore, ϕ must satisfy the so-called *conservation law* as follows. For each intermediate node u of $\mathcal{N}$, the sum of the flows of the outgoing arcs from u must be equal to the sum of the flows of the incoming arcs to u. Each source s has a *production* $\sigma(s) \geq 0$ of flow which is equal to the sum of the flows of outgoing arcs of s, and each sink t has a *consumption* $-\sigma(t) \geq 0$ of flow which is equal to the sum of the flows of the incoming arcs to t. The total amount of production of the sources is equal to the total amount of consumption of the sinks. An assignment of flow to each arc together with its capacity of the flow network

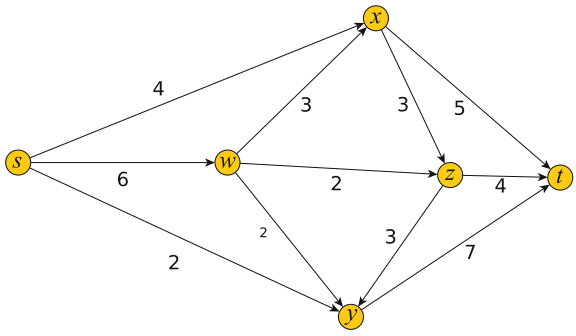

Fig. 8.5 A flow network

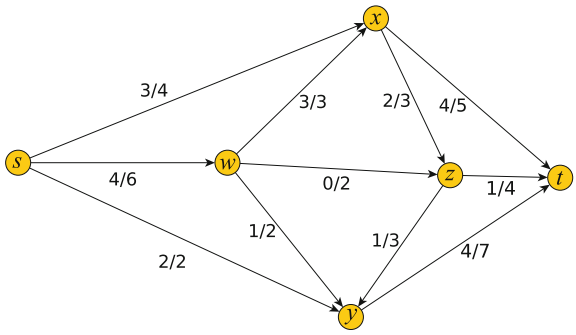

Fig. 8.6 An assignment of flow to a flow network

in Fig. 8.5 is shown in Fig. 8.6, where total amount of production in the source s is $3 + 4 + 2 = 9$ which is equal to the total amount of consumption in the sink t. One can observe that the capacity constraint is satisfied in each edge and the conservation law is also satisfied. The *value of a flow* ϕ which is denoted by $|\phi|$ is equal to the total amount of flow coming out from the sources. The value of a flow is equal to the total amount of consumption of the sinks. Thus the value of flow in the network in Fig. 8.6 is 9. The *maximum flow* for a flow network $\mathcal{N}$ is a flow with the maximum value over all flows for $\mathcal{N}$.

Assume that $\mathcal{N}$ is a network having exactly one source and exactly one sink. For two subsets X and Y of the vertex set V of $\mathcal{N}$, let $A(X, Y)$ denote the set of arcs from vertices in X to vertices in Y. For $X \subseteq V$, $\overline{X}$ denotes the vertex set $V - X$. Then a *cut* is a set of arcs $A(X, \overline{X})$ where the source s is in X and the sink t is in $\overline{X}$. Intuitively, a cut in $\mathcal{N}$ is a set of arcs whose deletion disconnects the sink from the source. The *value of a cut* in $\mathcal{N}$ is equal to the sum of the capacities of the arcs in $A(X, \overline{X})$. The *flow of a cut* is equal to the sum of the flows of the arcs in $A(X, \overline{X})$ minus the sum of the flow of the arcs in $A(\overline{X}, X)$. For $X = \{s, w, y\}$ in the graph in Fig. 8.6, $A(X, \overline{X}) = \{(s, x), (w, x), (w, z), (y, t)\}$. The arc $(z, y) \in A(\overline{X}, X)$. The value of the cut $A(X, \overline{X})$ is 16 and the flow in the cut is 10.

One can easily observe that there are 2^n cuts in $\mathcal{N}$, if $\mathcal{N}$ has n intermediate vertices. We call a cut a *minimum cut* or *min-cut* if its value is minimum over the values of all cuts in $\mathcal{N}$. The following theorem, which is known as max-flow min-cut theorem, was proven by Ford and Fulkerson in 1956 [3].

Theorem 8.6.1 *The value of a maximum flow in a flow network is equal to the value of a minimum cut of the network.*

Ford and Fulkerson gave a greedy iterative algorithm to compute a maximum flow in a network. The algorithm incrementally increases the value of a flow in steps, where in each step some amount of flow is pushed along an "augmenting path" from the source to the sink [4]. Initially, the flow of each edge is equal to zero. At each step, an augmenting path is found and an amount of flow is pushed along the path. The algorithm terminates when the flow does not admit an augmenting path. Finding

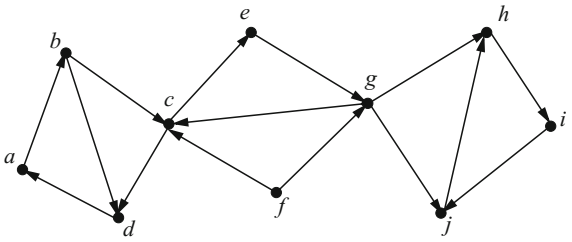

Fig. 8.7 A digraph

a maximum flow in a network has applications in finding disjoint paths, bipartite matching, airline scheduling, etc. [5].

Bibliographic Notes

The books [6–8] were used in preparing this chapter. Interested readers can find more in those books.

Exercises

1. Construct a complete digraph of five vertices. Construct three different orientations of K_5.
2. Find a directed path of the longest possible length in the digraph in Fig. 8.7.
3. Find all strongly connected components in the digraph in Fig. 8.7.
4. Prove that a connected acyclic digraph always has a source and a sink.
5. Let T be any tournament. Prove that the converse of T and the complement of T are isomorphic.

References

1. Meyniel, M.: Une condition sufficiante d'exittance d'un circuit Hamiltonian dans un graph oriente. J. Comb. Theory Ser. B **14**, 137–147 (1073)
2. Redei, L.: Ein Kombinatorischer Satz. Acta Litterania Szeged **7**, 39–43 (1934)
3. Ford, L.R., Fulkerson, D.R.: Maximum flow through a network. Can. J. Math. **8**, 399–404 (1956)
4. Goodrich, M.T., Tamassia, R.: Algorithm Design: Foundations, Analysis, and Internet Examples. Wiley, New York (2002)
5. Jon Kleinberg, M., Tardos, E.: Algorithm Design. Addison-Wesley, Boston (2006)
6. Clark, J., Holton, D.A.: A First Look at Graph Theory. World Scientific, Singapore (1991)
7. Wilson, R.J.: Introduction to Graph Theory, 4th edn. Longman, London (1996)
8. Pirzada, S.: An Introduction to Graph Theory. University Press, India (2009)

Chapter 9
Special Classes of Graphs

9.1 Introduction

In this chapter, we know about some special classes of graphs. Special classes of graphs play important roles in graph algorithmic studies. When we find a computationally hard problem for general graphs, we try to solve those problems for special classes of graphs.

9.2 Outerplanar Graphs

A graph is *outerplanar* if it has a planar embedding where all vertices are on the outer face. An outerplanar embedding of an outerplanar graph is called an *outerplane* graph. The graph in Fig. 9.1(a) is an outerplanar graph since it has an outerplanar embedding as shown in Fig. 9.1(b). An *outer edge* of an outerplane graph is an edge on the outer face of the outerplane graph and all other edges are *inner edges*.

Remember from Section 6.3.2 that the weak dual of a plane graph G is a graph, where each vertex corresponds to each inner face of G and each edge corresponds to a common edge between two inner faces of G. Figure 9.2 illustrates the weak dual of an outerplane graph where the vertices of the weak dual are drawn by white circles and the edges are drawn by dotted lines.

Lemma 9.2.1 *The weak dual of a biconnected outerplane graph is a tree.*

Proof Let G be a biconnected outerplane graph. Since G is biconnected, the outer boundary of G is a simple cycle C and all inner faces are inside C. Furthermore, two neighboring faces share an inner edge. Therefore, the weak dual T is connected. We now show that T does not contain a cycle. Assume for a contradiction that T contains a cycle. Then G would have an inner vertex v, a contradiction to the definition of an outerplane graph. □

M.S. Rahman, *Basic Graph Theory*, Undergraduate Topics in Computer Science,
DOI 10.1007/978-3-319-49475-3_9

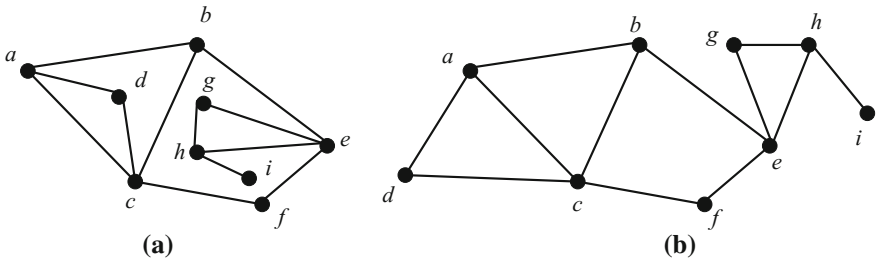

Fig. 9.1 An example of an outerplanar graph

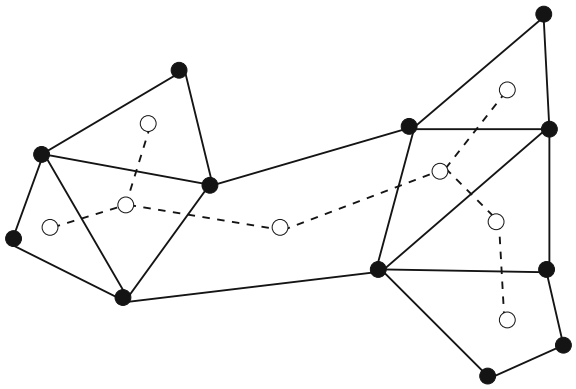

Fig. 9.2 Illustration for the weak dual of a biconnected outerplane graph

Observe that a leaf vertex of the weak dual tree T of an outerplane graph G corresponds to a face of G containing a vertex of degree 2. If T is a tree of a single vertex, then G is a triangle and G contains exactly three vertices of degree 2. If T has two or more vertices then T has at least two leaves, and hence the following fact holds.

Fact 9.2.2 *Every biconnected outerplane graph has at least two vertices of degree 2.*

An outerplane graph is a *maximal outerplane graph* if it has the maximum possible number of edges for the given number of vertices. Clearly, every maximal outerplane graph is a triangulation of a polygon. A triangle in a maximal outerplanar graph is an *internal triangle* if none of its edges is an outer edge. The following properties of a maximal outerplane graph are known [1].

Theorem 9.2.3 *Let G be a maximal outerplane graph with $n \geq 3$ vertices. Then the following claims hold:*

(a) G has $n - 2$ inner faces and $n - 3$ inner edges.
(b) G has $2n - 3$ edges.
(c) G has at least three vertices having degree three or less.

Proof (a) We prove the claim using an induction on the number of vertices. Obviously, the claim holds for $n = 3$. Assume that $n > 3$ and the claim holds for any maximal outerplane graph of less than n vertices. Let G be a maximal outerplane graph of n vertices. By Fact 9.2.2, G must have a vertex v of degree 2 on the outer face. We obtain a graph G' from G by deleting v from G. Clearly, G' is a maximal outerplane graph of $n - 1$ vertices. By induction hypothesis G' has $(n - 1) - 2$ inner faces and $(n - 1) - 3$ inner edges. In constructing G' by deleting v we reduced the number of inner faces by 1 and number of inner edges by 1. Then G has $(n - 1) - 2 + 1 = n - 2$ inner faces and $(n - 1) - 3 + 1 = n - 3$ inner edges.

(b) Outer edges form a cycle of n vertices. Thus there are n outer edges. From (a), G has $n - 3$ inner edges. Then total number of edges in G is $n + n - 3 = 2n - 3$.

(c) By (b) G has $2n - 3$ edges. Hence the degree sum for G is $4n - 6$. Let v be a vertex of G with degree less than 4. We define the *deficiency* for v by $4 - d(v)$. Then the total deficiency of the vertices of G is 6. Since G is maximal planar, G is biconnected and hence the degree of a vertex of G is at least 2. Then the deficiency of a vertex is at most 2. Since the total deficiency is 6, there are at least three vertices of degree three or less. □

The following property is known from [2] whose proof is left as an exercise.

Lemma 9.2.4 *Let n_2 be the number of vertices of degree 2 and t be the number of internal triangles in a maximal outerplane graph with n vertices. If $n \geq 4$, then $t = n_2 - 2$.*

Sometimes we are interested in graphs of restricted degrees. The following interesting properties hold for maximal outerplane graphs of maximum degree 4 [3].

Theorem 9.2.5 *Let G be a maximal outerplane graph of three or more vertices with the maximum degree 4. Assume that G is not an outerplane graph as shown in Fig. 9.3. Then the following (a)–(d) hold.*

(a) G has no internal triangle.
(b) The inner dual of G is a path.
(c) If G has four or more vertices, then G has exactly two pairs of adjacent vertices x and y such that degree of x is 2 and degree of y is 3, and G has exactly $n - 4$ vertices of degree 4.
(d) If G has more than four vertices, then every inner edge of G is incident to a vertex of degree 4 in G.

Proof (a) Let $v_1, v_2, \ldots, v_n$ be the consecutive vertices on the outer cycle of G in this order. Assume for a contradiction that G has an internal triangle f consisting v_x, v_y, and v_z and they are not consecutive on the outer cycle. In order to f be a triangle, each v_x, v_y, v_z needs to have at least two inner edges incident to it. Since two outer edges are incident to each of v_x, v_y, v_z, the degree of each of these three vertices is 4 in G. Let f_{xy}, f_{yz}, and f_{zx} be the three adjacent faces of f containing

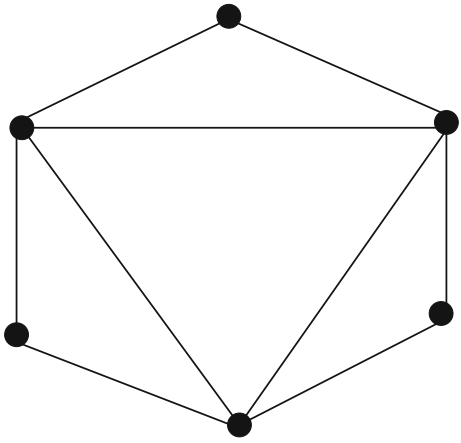

Fig. 9.3 Outerplanar octahedron

edges (v_x, v_y), (v_y, v_z), and (v_z, v_x), respectively. If any of these faces, i.e., f_{xy} is a triangle, then either G would be the graph in Fig. 9.3 or either v_x or v_y would have degree more than 4, a contradiction.

Proofs of (b)–(d) are left for exercises. □

A generalization of an outerplanar graph is found in literature as a k-outerplanar graph. A planar graph G is *k-outerplanar* if G is outerplanar for $k = 1$, and for $k > 1$, G has a planar embedding $\Gamma(G)$ such that a $(k-1)$-outerplanar graph can be obtained from G by deleting all vertices on the outer face of $\Gamma(G)$. Thus, a 2-outerplanar graph G has a planar embedding $\Gamma(G)$ such that deletion of all outer vertices of $\Gamma(G)$ results in an outerplanar graph.

9.3 Triangulated Plane Graphs

We have learned from Chapter 6 that a triangulated plane graph is a planar embedding of a maximal planar graph. Each face of a triangulated plane graph is a triangle. In this section, we study various aspects of triangulated plane graphs.

9.3.1 Canonical Ordering

In this section, we know about an elegant ordering of vertices of a triangulated plane graph called canonical ordering [4]. In 1990, de Fraysseix et al. [5] introduced canonical ordering for showing that every planar graph admits a straight-line

drawing on an $O(n^2)$ grid. After that, canonical ordering has appeared as an important graph algorithmic tool.

For a cycle C in a graph, an edge joining two nonconsecutive vertices in C is called a *chord* of C. For a 2-connected plane graph G, we denote by $C_o(G)$ the *outer cycle* of G, that is, the boundary of the outer face of G. A vertex on $C_o(G)$ is called an *outer vertex* and an edge on $C_o(G)$ is called an *outer edge*. A plane graph is *internally triangulated* if every inner face is a triangle.

Let $G = (V, E)$ be a triangulated plane graph of $n \geq 3$ vertices, as illustrated in Fig. 9.4. Since G is triangulated, there are exactly three vertices on $C_o(G)$. One may assume that these three vertices, denoted by v_1, v_2, and v_n, appear on $C_o(G)$ counterclockwise in this order. Let $\pi = (v_1, v_2, \ldots, v_n)$ be an ordering of all vertices in G. For each integer $k, 3 \leq k \leq n$, we denote by G_k the plane subgraph of G induced by the k vertices $v_1, v_2, \ldots, v_k$. Then $G_n = G$. We call π a *canonical ordering* of G if the following conditions (co1)–(co3) hold for each index $k, 3 \leq k \leq n$:

(co1) G_k is 2-connected and internally triangulated;
(co2) (v_1, v_2) is an outer edge of G_k; and
(co3) if $k + 1 \leq n$, then vertex v_{k+1} is located in the outer face of G_k, and all neighbors of v_{k+1} in G_k appear on $C_o(G_k)$ consecutively.

An example of a canonical ordering is illustrated for a triangulated plane graph of $n = 16$ vertices in Fig. 9.4.

We now have the following theorem due to de Fraysseix et al. [5] whose proof is taken from [4].

Theorem 9.3.1 *Every triangulated plane graph G has a canonical ordering.*

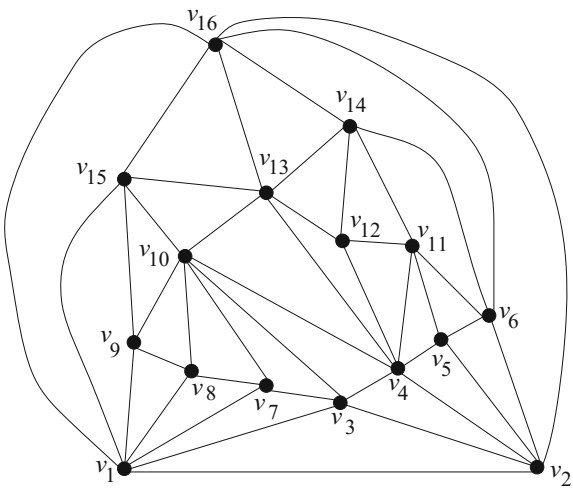

Fig. 9.4 A canonical ordering of a triangulated plane graph of $n = 16$ vertices [4]

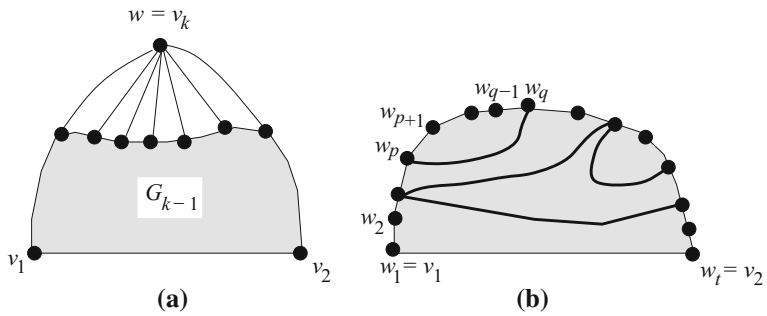

Fig. 9.5 **(a)** Graph G_k and **(b)** chords

Proof Obviously, G has a canonical ordering if $n = 3$. One may thus assume that $n \geq 4$. Since $G = G_n$, clearly (co1)–(co3) hold for $k = n$. We then choose the $n - 3$ inner vertices $v_{n-1}, v_{n-2}, \ldots, v_3$ in this order, and show that (co1)–(co3) hold for $k = n - 1, n - 2, \ldots, 3$.

Assume for inductive hypothesis that the vertices $v_n, v_{n-1}, \ldots, v_{k+1}$, $k + 1 \geq 4$, have been appropriately chosen, and that (co1)–(co3) hold for $n, n - 1, \ldots, k$. If one can choose as v_k a vertex $w \neq v_1, v_2$ on the cycle $C_o(G_k)$ which is not an end of a chord of $C_o(G_k)$, as illustrated in Fig. 9.5(a), then clearly (co1)–(co3) hold for $k - 1$ since $G_{k-1} = G_k - v_k$. Thus it suffices to show that there is such a vertex w.

Let $C_o(G_k) = w_1, w_2, \ldots, w_t$, where $w_1 = v_1$ and $w_t = v_2$. If $C_o(G_k)$ has no chord, then any of the vertices $w_2, w_3, \ldots, w_{t-1}$ is such a vertex w. One may thus assume that $C_o(G_k)$ has a chord. Then G_k has a "minimal" chord (w_p, w_q), $p + 2 \leq q$, such that none of the vertices $w_{p+1}, w_{p+2}, \ldots, w_{q-1}$ is an end of a chord, as illustrated in Fig. 9.5(b), where chords are drawn by thick lines. Then any of the vertices $w_{p+1}, w_{p+2}, \ldots, w_{q-1}$ is such a vertex w. □

Based on the proof of Theorem 9.3.1, a linear-time algorithm can be developed to find a canonical ordering of a triangulated plane graph [4].

9.3.2 Separating Triangles

A *separating triangle* of a triangulated plane graph G is a triangle in G whose interior and exterior contain at least one vertex each. The triangle bcd is a separating triangle in the triangulated plane graph in Fig. 9.6, since the vertex a is outside of the triangle and the vertex e is inside of the triangle. Observe that neither the outer face nor an inner face of a triangulated plane graph is a separating triangle. In solving many algorithmic problems in graph drawing area, separating triangles appear as problematic elements and it is often required to count the number of separating triangles. In this section, we establish a tight upper bound on the number of separating triangles in a triangulated plane graph. We show that the number of

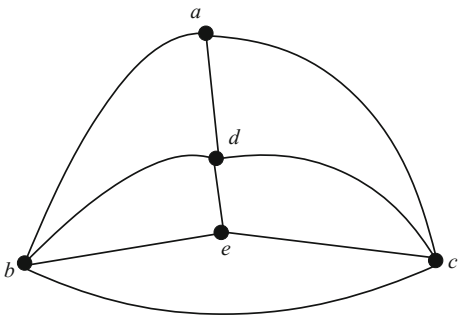

Fig. 9.6 Illustration for separating triangles

separating triangles in a triangulated plane graph with n vertices is at most $n-4$. We also give an example of a class of triangulated plane graphs with exactly $n-4$ separating triangles.

Let $\mathcal{T}$ be a separating triangle of a triangulated plane graph G. We denote by $G(\mathcal{T})$ the plane subgraph of G inside $\mathcal{T}$ including $\mathcal{T}$. A face of G is called an *internal face of* $\mathcal{T}$ if it is contained in $G(\mathcal{T})$; otherwise it is called an *external face of* $\mathcal{T}$. Let $\mathcal{T}_p$ and $\mathcal{T}_c$ be two separating triangles of G. We say that $\mathcal{T}_p$ is an *ancestor* of $\mathcal{T}_c$ and $\mathcal{T}_c$ is a *descendant* of $\mathcal{T}_p$ if $\mathcal{T}_c$ is properly contained in $G(\mathcal{T}_p)$. Then a separating triangle is neither an ancestor nor a descendant of itself. Additionally, if $\mathcal{T}_c$ is not a descendant of any descendant separating triangle of $\mathcal{T}_p$, then we also say that $\mathcal{T}_p$ is the *parent* of $\mathcal{T}_c$ and $\mathcal{T}_c$ is a *child* of $\mathcal{T}_p$. In the triangulated plane graph in Fig. 9.7(a), the separating triangle alb is the parent of the separating triangle alf. We now have the following lemma, whose proof is trivial.

Lemma 9.3.2 *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two separating triangles of a triangulated plane graph G. Then exactly one of the following three conditions holds:*

(1) $\mathcal{T}_1$ is a descendant of $\mathcal{T}_2$.
(2) $\mathcal{T}_2$ is a descendant of $\mathcal{T}_1$.
(3) $\mathcal{T}_1$ and $\mathcal{T}_2$ have no common internal face.

Lemma 9.3.2 implies that the containment relation among the separating triangles of G can be represented by a forest. The roots of the trees of this forest are the separating triangles without any parent separating triangles. We complete a tree from this forest by adding a root vertex representing the graph G and making it the parent of each of these separating triangles with no parent in the forest. We call this tree the *genealogical tree* of G. The genealogical tree of the triangulated plane graph in Fig. 9.7(a) is illustrated in Fig. 9.7(b). We now prove the following theorem [6].

Theorem 9.3.3 *The number of separating triangles in a triangulated plane graph G with n vertices is at most $n-4$.*

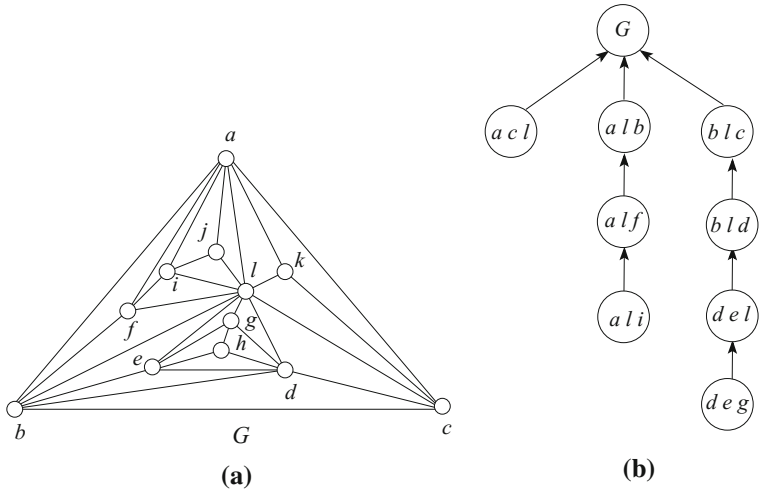

Fig. 9.7 **(a)** A plane graph G, **(b)** the genealogical tree of G

Proof We first show that we can assign two faces of G to each separating triangle in G without duplication; thus each face of G is assigned to at most one separating triangle in G. We perform the assignment in the top-down order on the genealogical tree of G as follows.

We assume that there is at least one separating triangle in G. For each separating triangle $\mathcal{T} = uvw$ of G, there are exactly three internal faces f_1, f_2, and f_3 of $\mathcal{T}$, containing the edges (u, v), (v, w), and (w, u), respectively. Furthermore by Lemma 9.3.2, these three internal faces of $\mathcal{T}$ are not internal faces of any other separating triangles except for the ancestors and the descendants of $\mathcal{T}$. We first consider the case where $\mathcal{T}$ is a child of the root of the genealogical tree of G. In this case, at most one of the faces f_1, f_2, f_3 contains an edge on the outer face of G. We assign the other two faces to $\mathcal{T}$. We next consider the case where $\mathcal{T}$ is not a child of the root. In this case, $\mathcal{T}$ shares at most one edge with its parent and thus at most one of the three faces f_1, f_2, f_3 has already been assigned to its parent. We assign the other two faces to $\mathcal{T}$. In this manner, each of the separating triangles of G can be assigned exactly two faces of G without duplication.

From Euler's formula, the number of faces in a triangulated plane graph G of n vertices is $2n - 4$ [4]. Furthermore, we can show that there are at least four faces of G that have not been assigned to any separating triangles as follows. Let abc be the outer face of the graph G and let F_1, F_2, and F_3 be the three inner faces of G containing the edges (a, b), (b, c), and (c, a), respectively. Since only the internal faces are assigned to separating triangles, the outer face abc has not been assigned to any separating triangle. Moreover, from the assignment of the faces to the separating triangles, it is clear that the three faces F_1, F_2, F_3 are not assigned to any separating triangle. Thus at most $2n - 8$ faces are assigned to all the separating triangles and since two faces of G are assigned to each separating triangle without duplication, the number of separating triangles in G is at most $n - 4$. □

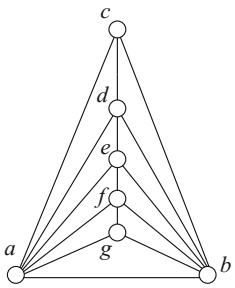

Fig. 9.8 A plane graph with $n-4$ separating triangles

Theorem 9.3.3 implies that the number of separating triangles in a triangulated plane graph of n vertices is at most $n-4$. This upper bound on the number of separating triangles in a triangulated plane graph is also tight since there are triangulated plane graphs of n vertices, as one illustrated in Fig. 9.8, which have exactly $n-4$ separating triangles.

9.3.3 Plane 3-Trees

In this section, we study some properties of a subclass of plane triangulations called plane 3-trees. A plane graph G with $n \geq 3$ vertices is called a *plane 3-tree* if the following (a) and (b) hold:

(a) G is a triangulated plane graph;
(b) if $n > 3$, then G has a vertex x whose deletion gives a plane 3-tree G' of $n-1$ vertices.

Note that vertex x may be an inner vertex or an outer vertex of G. We denote a plane 3-tree of n vertices by G_n. Examples of plane 3-trees are shown in Fig. 9.9; G_6 is obtained from G_7 by removing the inner vertex c of degree three. Then G_5 is obtained from G_6 by deleting the inner vertex b of degree three. G_4 is obtained from G_5 by deleting the outer vertex g of degree three and G_3 is obtained in a similar way.

The following results are known on plane 3-trees.

Lemma 9.3.4 ([7]) *Let G_n be a plane 3-tree with n vertices, where $n > 3$. Then the following (a) and (b) hold. (a) G_n has an inner vertex x of degree three such that the removal of x gives the plane 3-tree G_{n-1}. (b) G_n has exactly one inner vertex y such that y is the neighbor of all the three outer vertices of G_n.*

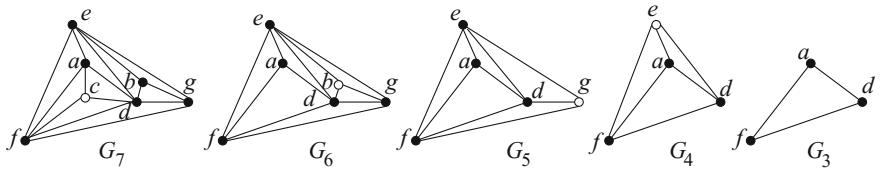

Fig. 9.9 Examples of plane 3-trees

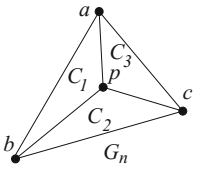

Fig. 9.10 Nested triangles around p

By Lemma 9.3.4b for any plane 3-tree $G_n, n > 3$, there is exactly one inner vertex y which is the common neighbor of all the outer vertices of G_n. We call vertex y the *representative vertex* of G_n.

Let G_n be a plane 3-tree and C be a triangle in G_n, Mondal et al. [8] showed that $G_n(C)$ is also a plane 3-tree as in the following lemma.

Lemma 9.3.5 *Let G_n be a plane 3-tree with $n > 3$ vertices and C be any triangle of G_n. Then the subgraph $G_n(C)$ is a plane 3-tree.*

Let p be the representative vertex and a, b, c be the outer vertices of G_n. The vertex p, along with the three outer vertices a, b, and c, form three triangles $\{a, b, p\}$, $\{b, c, p\}$, and $\{c, a, p\}$ as illustrated in Fig. 9.10. We call those three triangles the *nested triangles around p*.

We now define the *representative tree* of G_n as an ordered, rooted tree T_{n-3} satisfying the following two conditions (a) and (b):

(a) if $n = 3$, T_{n-3} consists of a single vertex.
(b) if $n > 3$, then the root p of T_{n-3} is the representative vertex of G_n and the subtrees rooted at the three counterclockwise ordered children q_1, q_2, and q_3 of p in T_{n-3} are the representative trees of $G_n(C_1)$, $G_n(C_2)$, and $G_n(C_3)$, respectively, where C_1, C_2, and C_3 are the three nested triangles around p in counterclockwise order.

Figure 9.11(b) illustrates the representative tree T_{n-3} of a plane 3-tree G_n in Fig. 9.11(a).

We now prove that T_{n-3} is unique for G_n in the following theorem [8].

Theorem 9.3.6 *Let G_n be any plane 3-tree with $n \geq 3$ vertices. Then G_n has a unique representative tree T_{n-3} with exactly $n - 3$ internal vertices and $2n - 5$ leaves.*

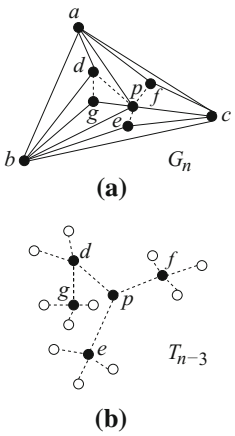

Fig. 9.11 **(a)** A plane 3-tree G_n and **(b)** the representative tree T_{n-3} of G_n

Proof The case $n = 3$ is trivial since the representative tree of G_3 is a single vertex. We may thus assume that G has four or more vertices. By Lemma 9.3.4b, G_n has exactly one representative vertex. Let p be that representative vertex of G_n and C_1, C_2, C_3 be the three nested triangles around p. By Lemma 9.3.5, $G_n(C_1)$, $G_n(C_2)$, and $G_n(C_3)$ are plane 3-trees. Let n_1, n_2, and n_3 be the number of vertices in $G_n(C_1)$, $G_n(C_2)$, and $G_n(C_3)$, respectively. Then by the induction hypothesis, T_{n_1-3}, T_{n_2-3}, and T_{n_3-3} are the unique representative trees of $G_n(C_1)$, $G_n(C_2)$, and $G_n(C_3)$, respectively. We now assign p as the parent of q_1, q_2, and q_3, where q_1, q_2, and q_3 are the roots of T_{n_1-3}, T_{n_2-3}, and T_{n_3-3}, respectively. Since p is the unique representative vertex of G_n, the choice for the root of T_{n-3} is unique. Since G_n has n vertices and any inner vertex of G_n except p belongs to exactly one of $G_n(C_1)$, $G_n(C_2)$, and $G_n(C_3)$, the total number of vertices in T_{n_1-3}, T_{n_2-3}, and T_{n_3-3} is $n_1 - 3 + n_2 - 3 + n_3 - 3 = n - 4$. Thus the new tree T_{n-3} with root p has $n - 4 + 1 = n - 3$ internal vertices. Since T_{n_1-3}, T_{n_2-3}, and T_{n_3-3} are ordered trees and q_1, q_2, and q_3 are ordered counterclockwise around p, T_{n-3} is also an ordered tree. Furthermore one can easily observe that the leaves represent only the inner faces of G_n. Since the number of inner faces of G_n is $2n - 5$ by Euler's Theorem, T_{n-3} has $2n - 5$ leaves. □

9.4 Chordal Graphs

In Section 9.3, we studied plane triangulations. In this section, we study a generalization of triangulations to nonplanar graphs.

A *chord* in a cycle is an edge which goes between two vertices which are not consecutive on the cycle. A graph G is *chordal* if there are no chordless cycles in G of length greater than three. Figure 9.12(a) illustrates a chordal graph of nine

vertices. Chordal graphs always contain a vertex v such that the neighborhood of v is a clique; such a vertex is called a *simplicial vertex*. This gives rise to a *perfect elimination scheme* $v_1, v_2, \ldots, v_n$ in which each v_i is simplicial in the graph induced by $v_i, v_{i+1}, \ldots v_n$. In the chordal graph in Fig. 9.12(a), the vertex sequence 1, 2, 3, 4, 5, 8, 6, 7, 9 is a perfect elimination scheme. Many problems on chordal graphs can be solved efficiently using the ordering of the vertices in a perfect elimination scheme. For example, we can find a maximum independent set in a chordal graph by selecting vertices from the perfect elimination scheme greedily if they have no previous neighbor in the independent set.

If the input graph is chordal, one can find a perfect elimination scheme using lexicographic breadth first search (LBFS) as follows [9]. Vertices are partitioned into sets; initially all vertices are in the same set. At each step, an arbitrary vertex x from the last set is chosen. We remove x from the graph and place it in the output list. Note that x comes before all previously selected vertices in the elimination scheme, and after all vertices which remain in the graph. Each set S is subdivided into neighbors and non-neighbors of x in S with neighbors of x placed immediately after the set of non-neighbors of x in S. The perfect elimination scheme is the reverse order of the output list. A step-by-step illustration of the algorithm is shown in Fig. 9.12(b). Initially, all the vertices are kept in one set (123456789). In Step 1, 9 is sent to the output list as x and the other vertices are divided as the non-neighbor set (123458) and the neighbor set (6, 7). In Step 2, 7 from the last set is sent to the output list and the sets are subdivided as neighbor and non-neighbor sets of 7. The process continues in this way as illustrated in Fig. 9.12(b).

We can always find an ordering of vertices above even if the input graph is not chordal. But if the graph is not chordal, the sequence will not be a perfect elimination scheme. If we check by eliminating vertices one by one following the ordering, in some step we will find a vertex which is not simplicial. Thus by producing a sequence and then verifying the sequence, we can recognize a chordal graph.

A clique in a graph G is a *maximal clique* if it is not a proper subgraph of any other clique in G. A chordal graph G corresponds exactly to an intersection graph of subtrees of a tree, where the nodes of the tree correspond to maximal cliques of G, each vertex v of G corresponds to a subtree of cliques which contain v. This model is called a *clique tree representation* of a chordal graph. A clique tree representation of the chordal graph in Fig. 9.12(a) is given in Fig. 9.12(c).

Given a graph $G = (V, E)$, a set of vertices $S \subset V$ is a *separator* if the subgraph of G induced by $V - S$ is disconnected. The set S is a uv-*separator* if u and v are in different connected components of $G - S$. A uv-separator S is *minimal* if no subset of S separates u and v. S is a *minimal separator* of G if there exist two vertices u and v in G such that S is a minimal uv-separator. We have the following theorem due to Dirac [10].

Theorem 9.4.1 *A graph G is chordal if and only if every minimal separator of G is a clique.*

Proof Necessity Let $G = (V, E)$ be chordal and let S be a minimal separator of G. Let x and y be any two vertices in S. We will show that (x, y) must be an edge of G.

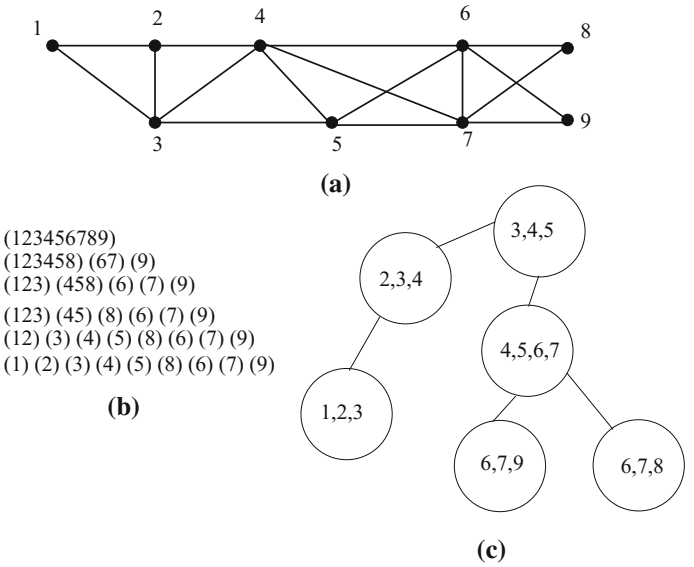

Fig. 9.12 **(a)** A chordal graph G, **(b)** step-by-step illustration for construction of a perfect elimination scheme of G and **(c)** a clique tree representation of G

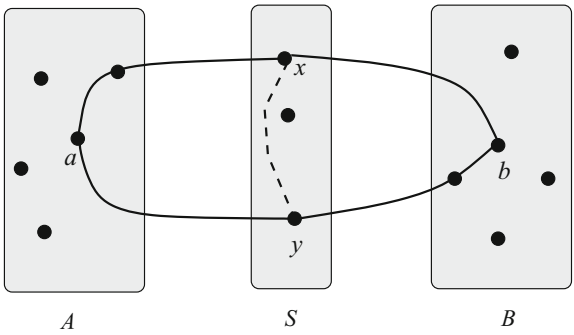

Fig. 9.13 Illustration for the proof of the necessity of Theorem 9.4.1

Let a and b be the vertices for which S is a minimal ab-separator, and let A and B be the connected components of $G - S$ containing, respectively, a and b. There must exist a path between x and y through vertices belonging to A. Let P_1 be a shortest such path. Let analogously P_2 be a shortest path between x and y through vertices of B. Merging of paths P_1 and P_2 makes a cycle of length at least 4 as illustrated in Fig. 9.13. Since G is chordal, this cycle must have a chord. Since no edges exist between vertices of A and vertices of B, the edge (x, y) must be present and a chord of the mentioned cycle, as indicated by a dotted line in Fig. 9.13.

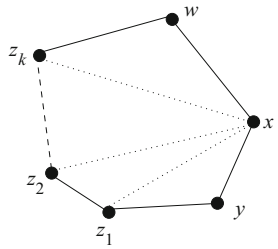

Fig. 9.14 Illustration for the proof of the sufficiency of Theorem 9.4.1

Sufficiency Let G be a graph where each minimal separator is a clique. Assume that G is not chordal, and let $w, x, y, z_1, \ldots, z_k, w$ be a chordless cycle of length at least 4 in G ($k \geq 1$). Any minimal wy-separator of G must contain x and at least one z_i for $1 \leq i \leq k$. Since all minimal separators are cliques, the edge (x, z_i) must belong to G contradicting that the mentioned cycle is chordless, as illustrated in Fig. 9.14. □

Any graph G can be turned into a chordal graph by adding edges, and the resulting chordal graph is called a triangulation of G. The above idea can be used to compute a triangulation of an input graph G as follows: Choose any vertex x to start with, and add the necessary edges so that the neighbors of x become a clique. Remove x from the modified graph, and continue this process until all vertices are processed. In the end, all the added edges of each step are added to the original graph G and this results in a filled graph, which is a triangulation of G.

The complete graph on k vertices is a *k-tree.* A k-tree G with $n + 1$ vertices can be constructed from a k-tree H with n vertices by adding a vertex adjacent to exactly k vertices that form a k-clique in H. k-trees are chordal graphs. A *partial k-tree* is a graph that contains all the vertices and a subset of the edges of a k-tree.

9.5 Interval Graphs

The class of interval graphs is an important subclass of chordal graphs. In an interval graph, vertices correspond to intervals on the real line, and two vertices are adjacent if and only if the corresponding intervals have a nonempty intersection. Figure 9.15 illustrates an interval graph with corresponding intervals on real line.

Since the class of interval graphs is a subclass of chordal graphs, an interval graph has a clique tree representation. However, an interval graph always has a clique tree representation which is a path as in the following theorem due to Gilmore and Hoffman [11].

Theorem 9.5.1 *A graph G is an interval graph if and only if G has a clique tree that is a simple path.*

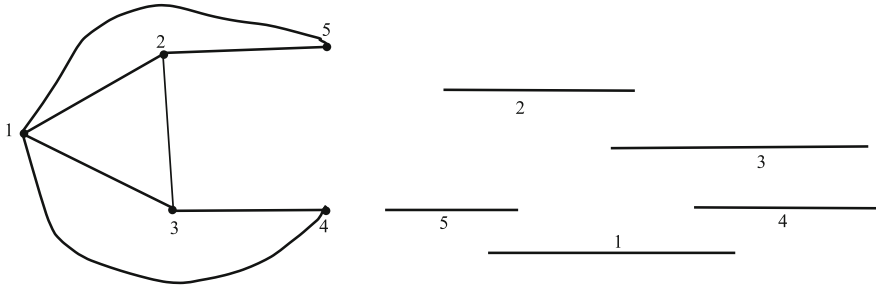

Fig. 9.15 An interval graph with corresponding intervals

The class of interval graphs is a very nice class of graphs which support many efficient algorithmic techniques [9]. Many problems which are difficult on general graphs can be solved on interval graphs using a simple greedy approach. For example, consider the independent set problem. It is easy to see that the interval which ends first can be included in a maximum independent set. To solve the independent set problem on interval graphs when we are given a representation, we can place this interval in the independent set, delete the vertex and its neighbors, and continue until all vertices are deleted.

Dynamic programming is also an effective technique for solving many problems on interval graphs. In dynamic programming, we build up a solution for a large problem from a table of solutions on subproblems. In interval graphs, the number of subproblems can be reasonably bounded which lead to efficient dynamic programming algorithms.

9.6 Series-Parallel Graphs

In this section, we study a popular special class of graphs called series-parallel graphs. Many hard problems on this class of graphs were solved using its SPQ-tree decomposition [12].

A graph $G = (V, E)$ is called a *series-parallel graph* (with source s and sink t) if either G consists of a pair of vertices connected by a single edge, as illustrated in Fig. 9.16(a), or there exist two series-parallel graphs $G_i = (V_i, E_i)$, $i = 1, 2$, with source s_i and sink t_i such that $V = V_1 \cup V_2$, $E = E_1 \cup E_2$, and either $s = s_1, t_1 = s_2$, and $t = t_2$ as illustrated in Fig. 9.16(b) or $s = s_1 = s_2$ and $t = t_1 = t_2$ as illustrated in Fig. 9.16(c). Figure 9.16(d) illustrates a series-parallel graph. By definition, a series-parallel graph G is a connected planar graph and G has exactly one source s and exactly one sink t. A biconnected component of a series-parallel graph is also a series-parallel graph.

Before describing an SPQ-tree decomposition of a series-parallel graph, we review some definitions. A pair $\{u, v\}$ of vertices of a connected graph G is a *split pair* if

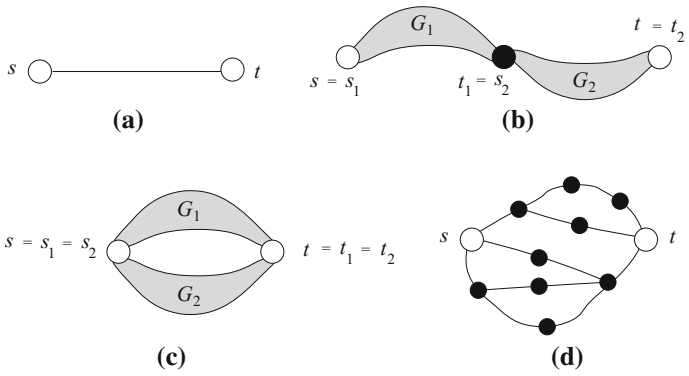

Fig. 9.16 **(a)** A basic series-parallel graph, **(b)** series connection, **(c)** parallel connection, and **(d)** a series-parallel graph

there exist two subgraphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ satisfying the following two conditions:

1. $V = V_1 \cup V_2$, $V_1 \cap V_2 = \{u, v\}$; and
2. $E = E_1 \cup E_2$, $E_1 \cap E_2 = \emptyset$, $|E_1| \geq 1$, $|E_2| \geq 1$.

Thus every pair of adjacent vertices is a split pair. For the graph in Fig. 9.17, the split pairs are $\{c, f\}$, $\{c, g\}$, $\{d, f\}$, $\{d, g\}$ and the pairs of adjacent vertices. A *split component* of a split pair $\{u, v\}$ is either an edge (u, v) or a maximal connected subgraph H of G such that $\{u, v\}$ is not a split pair of H. There are two split components of split pair $\{c, g\}$ for the graph G in Fig. 9.17: one is the subgraph of G induced by vertices a, b, c, g, h, and i; and the other by c, d, e, f, g, and j. A split pair $\{u, v\}$ of G is called a *maximal split pair* with respect to a *reference split pair* $\{s, t\}$ if, for any other split pair $\{u', v'\}$, vertices s, t, u, and v are in the same split component of $\{u', v'\}$. In Fig. 9.17, the maximal split pairs with respect to reference edge (a, b) are the pair $\{c, g\}$ and the pairs $\{a, i\}$, $\{a, g\}$, $\{b, i\}$, $\{b, c\}$, $\{c, h\}$, $\{h, i\}$, $\{g, h\}$ of adjacent vertices. The split pair $\{d, f\}$ is not a maximal split pair, because vertices a, b, d, and f are not in the same split component of $\{c, g\}$. Similarly, neither the split pair $\{c, f\}$ nor the split pair $\{d, g\}$ is a maximal split pair.

Let G be a biconnected series-parallel graph. Let (s, t) be an edge of G. The SPQ-tree $\mathcal{T}$ of G with respect to a *reference edge* $e = (s, t)$ describes a recursive decomposition of G induced by its split pairs [13]. Tree $\mathcal{T}$ is a rooted, ordered

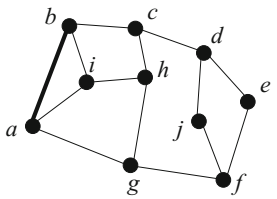

Fig. 9.17 A connected planar graph G

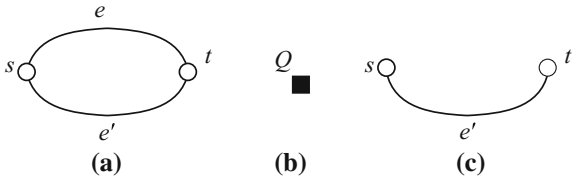

Fig. 9.18 **(a)** $G = skeleton(x)$, **(b)** $\mathcal{T}$, and **(c)** G_x for trivial case

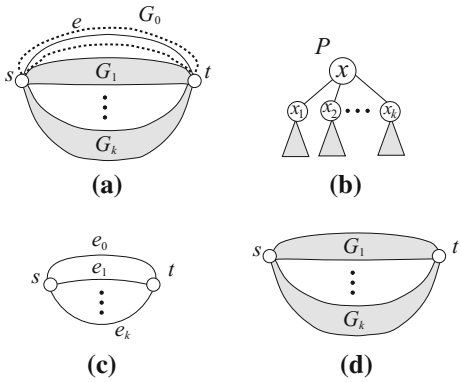

Fig. 9.19 **(a)** G, **(b)** $\mathcal{T}$, **(c)** $skeleton(x)$, and **(d)** G_x for parallel case

tree whose nodes are of three types: S, P, and Q. Each node x of $\mathcal{T}$ corresponds to a subgraph of G, called its *pertinent graph* G_x. Each node x of $\mathcal{T}$ has an associated biconnected multigraph, called the *skeleton* of x and denoted by $skeleton(x)$. Tree $\mathcal{T}$ is recursively defined as follows.

• *Trivial Case*: In this case, G consists of exactly two parallel edges e and e' joining s and t as illustrated in Fig. 9.18(a). $\mathcal{T}$ consists of a single Q-node x as illustrated in Fig. 9.18(b), where a Q-node is drawn by a black square. The skeleton of x is G itself, as illustrated in Fig. 9.18(a). The pertinent graph G_x consists of only the edge e' as illustrated in Fig. 9.18(c). Note that the edge e is the reference edge.

• *Parallel Case*: In this case, the split pair $\{s, t\}$ has three or more split components $G_0, G_1, \ldots, G_k, k \geq 2$, and G_0 consists of only a reference edge $e = (s, t)$, as illustrated in Fig. 9.19(a). The root of $\mathcal{T}$ is a P-node x, as illustrated in Fig. 9.19(b). The $skeleton(x)$ consists of $k + 1$ parallel edges $e_0, e_1, \ldots, e_k$ joining s and t, as illustrated in Fig. 9.19(c), where $e_0 = e = (s, t)$ and $e_i, 1 \leq i \leq k$, corresponds to G_i. The pertinent graph $G_x = G_1 \cup G_2 \cup \cdots \cup G_k$ is a union of $G_1, G_2, \ldots, G_k$ as illustrated in Fig. 9.19(d). (The *skeleton* of P-node p_2 in Fig. 9.21 consists of three parallel edges joining vertices e and g. Figure 9.21(e) depicts the pertinent graph of p_2.)

• *Series Case*: In this case the split pair $\{s, t\}$ has exactly two split components, and one of them consists of the reference edge e. One may assume that the other split component has cut vertices $c_1, c_2, \ldots, c_{k-1}, k \geq 2$, that partition the component into its blocks $G_1, G_2, \ldots, G_k$ in this order from s to t, as illustrated in Fig. 9.20(d). Then the root of $\mathcal{T}$ is an S-node x, as illustrated in Fig. 9.20(b). The skeleton of

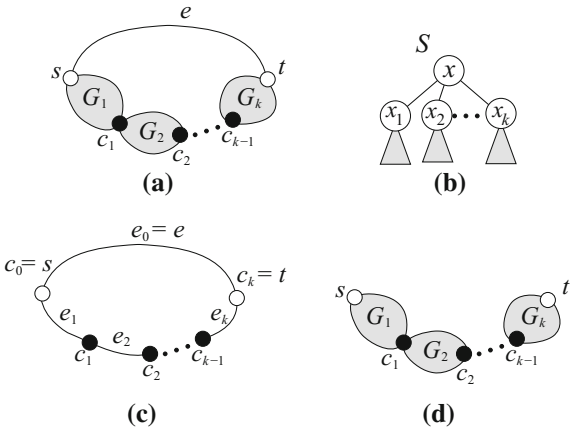

Fig. 9.20 **(a)** G, **(b)** $\mathcal{T}$, **(c)** *skeleton*(x), and **(d)** G_x for series case

x is a cycle $e_0, e_1, \ldots, e_k$, where $e_0 = e$, $c_0 = s$, $c_k = t$, and e_i joins c_{i-1} and c_i, $1 \leq i \leq k$, as illustrated in Fig. 9.20(c). The pertinent graph G_x of node x is a union of $G_1, G_2, \ldots, G_k$ as illustrated in Fig. 9.20(d). (The *skeleton* of S-node s_2 in Fig. 9.21 is the cycle c, d, e, g, h, a, c. Figure 9.21(d) depicts the pertinent graph G_{s_2} of s_2.)

In all cases above, we call the edge e the *reference edge* of node x. Except for the trivial case, node x of $\mathcal{T}$ has children $x_1, x_2, \ldots, x_k$ in this order; x_i is the root of the SPQ-tree of graph $G_i \cup e_i$ with respect to the reference edge e_i, $1 \leq i \leq k$. We call edge e_i *the reference edge of node* x_i, and call the endpoints of edge e_i the *poles* of node x_i. The tree obtained so far has a Q-node associated with each edge of G, except the reference edge e. We complete the SPQ-tree $\mathcal{T}$ by adding a Q-node, representing the reference edge e, and making it the parent of x so that it becomes the root of $\mathcal{T}$. An example of the SPQ-tree of a series-parallel graph in Fig. 9.21(a) is illustrated in Fig. 9.21(b), where the edge drawn by a thick line in each skeleton is the reference edge of the skeleton.

The SPQ-tree $\mathcal{T}$ defined above is a special case of an "SPQR-tree" [13–15] where there is no "R-node" and the root of the tree is a Q-node corresponding to the reference edge e. One can easily modify $\mathcal{T}$ in Fig. 9.21(b) to an SPQ-tree $\mathcal{T}'$ with an arbitrary P-node as the root as illustrated in Fig. 9.22(a). Note that with the change of the root, the pertinent graph of a node may change. For example, the pertinent graph of the S-node s_2 has been changed as illustrated in Fig. 9.22(b).

In recent years, SPQ-tree has been used in solving several problems on series-parallel graphs in the field of graph drawing [12, 16]. The basic idea of these works is to construct a drawing of a series-parallel graph with desired properties by merging the drawings of smaller subgraphs using a bottom-up computation on a SPQ-tree.

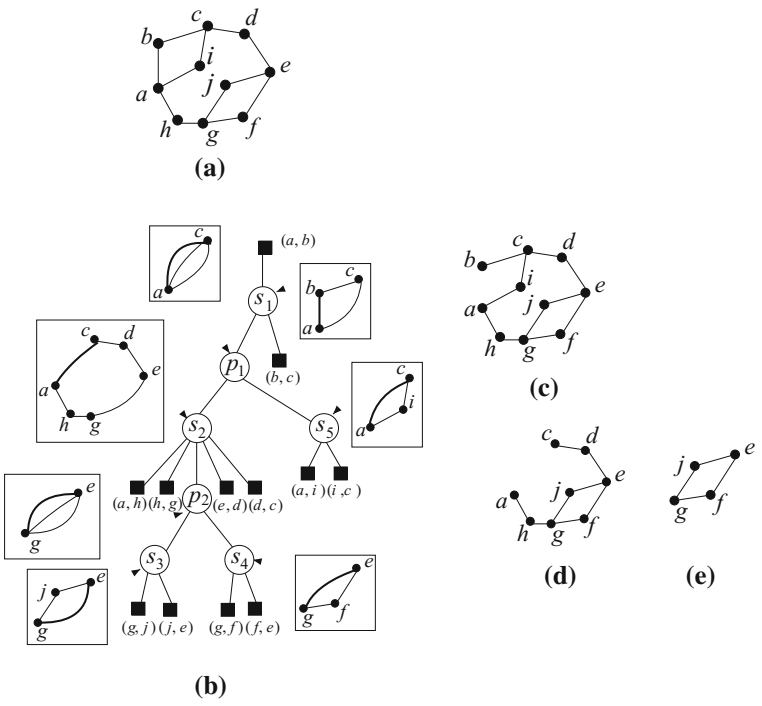

Fig. 9.21 **(a)** A series-parallel graph G, **(b)** SPQ-tree $\mathcal{T}$ of G with respect to reference edge (a, b), and skeletons of P- and S-nodes, **(c)** the pertinent graph G_{s_1} of S-node s_1, **(d)** the pertinent graph G_{s_2} of S-node s_2, **(e)** the pertinent graph G_{p_2} of P-node p_2

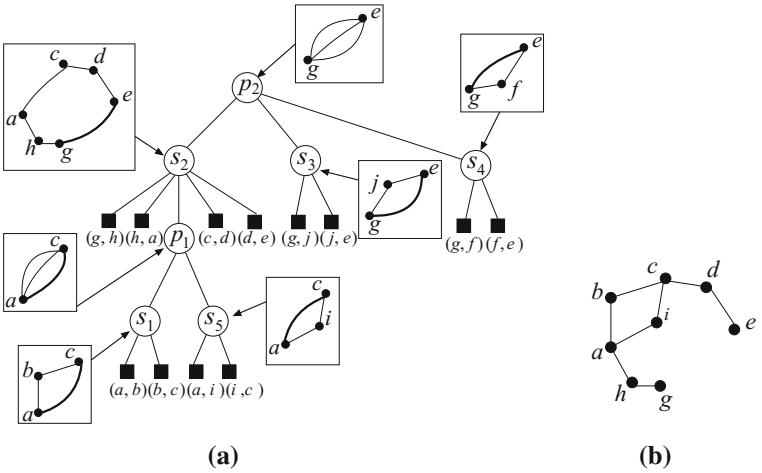

Fig. 9.22 (**a**) SPQ-tree $\mathcal{T}'$ of G in Fig. 9.21 a with P-node p_2 as the root, and (**b**) the pertinent graph of S-node s_2

9.7 Treewidth and Pathwidth

Treewidth is a parameter that gives a measure of how tree-like or close to being a tree a graph is. The smaller the treewidth, the more tree-like the graph is. As many NP-hard graph problems have simple and polynomial (even linear) time solutions on trees, it is often the case that these problems can also be solved in polynomial time for graphs that have constant treewidth, using dynamic programming techniques. The classes of graphs studied in this chapter can be unified by the parameter treewidth; all of them have bounded treewidth.

In order to define treewidth, we need first to define a tree decomposition of a graph. A *tree decomposition* of a graph $G = (V, E)$ is a tree T with nodes $X_1, X_2, \ldots, X_k$ such that

(a) each set X_i is a subset of V (called a bag) and the union of all sets X_i equals to V,
(b) for every edge (u, v) in G, there is a subset X_i that contains both u and v, and
(c) for all vertices $v \in V$, the nodes of T containing v induce a connected subtree of T.

Figures 9.23(b) and (c) illustrates two tree decompositions of the graph in Fig. 9.23(a). Observe that the graph in Fig. 9.23(a) is a chordal graph and the tree in Fig. 9.23(b) is a clique tree of the graph. If G is a chordal graph, then any clique tree of G is also a tree decomposition of G. However, the opposite is not necessarily true. The tree decomposition in Fig. 9.23(c) is not a clique tree.

The *width of a tree decomposition* is one less than the size of its largest set X_i. The *treewidth* $tw(G)$ of a graph G is the minimum width among all possible tree decompositions of G.

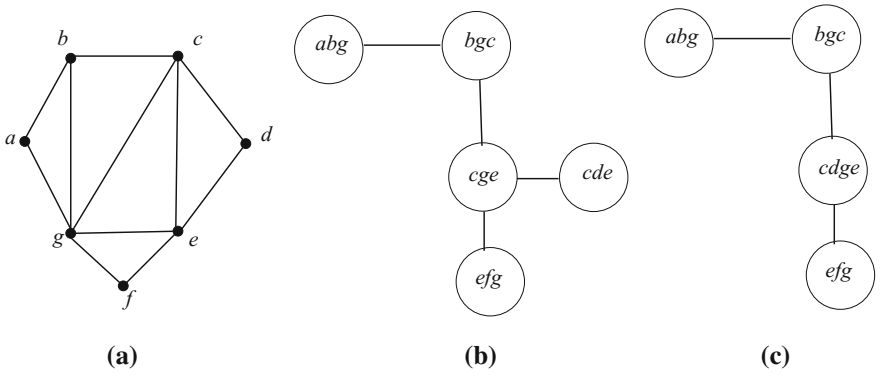

Fig. 9.23 Illustration for tree decomposition

A *path decomposition* of G is a tree decomposition T of G such that T is a path. Figure 9.23(c) illustrates a path decomposition of the graph in Fig. 9.23(a). The pathwidth of a graph G, $pw(G)$, is the minimum width over all path decompositions of G. Since every path decomposition is also a tree decomposition, pathwidth $\geq$ treewidth for all graphs.

A tree decomposition of width equal to the treewidth is called an *optimal tree decomposition.* A path decomposition of width equal to the pathwidth is called an *optimal path decomposition.* Unfortunately, computing treewidth and pathwidth are NP-hard problems [17, 18].

Exercises

1. Show that every maximal outerplanar graph contains at least two vertices of degree 2.
2. Prove Lemma 9.2.4.
3. Prove Theorem 9.2.5b–d.
4. Find canonical ordering of the graph in Fig. 9.24.
5. Identify all separating triangles in Fig. 9.24 and construct a genealogical tree of separating triangles.
6. Construct a clique tree for the graph in Fig. 9.25.
7. Find a perfect elimination scheme for the graph in Fig. 9.25.

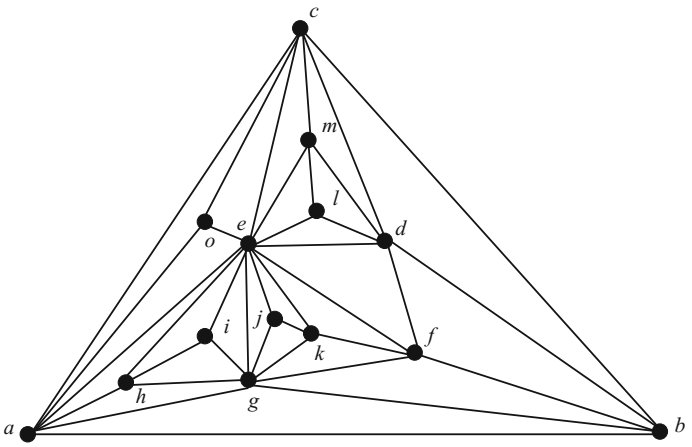

Fig. 9.24 A plane triangulation

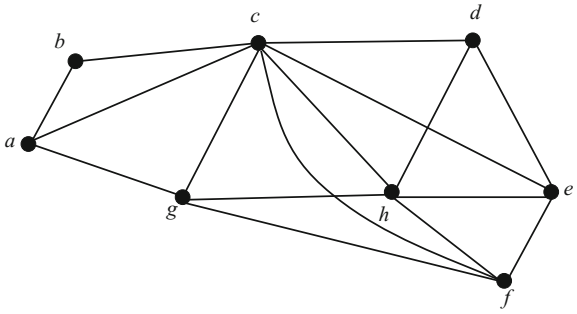

Fig. 9.25 A chordal graph

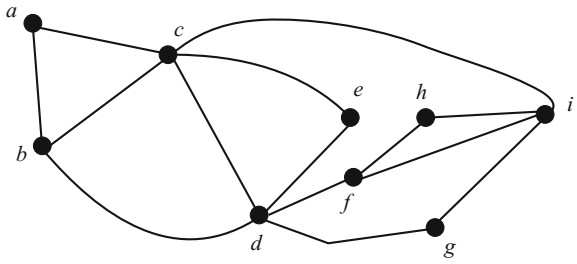

Fig. 9.26 A series-parallel graph

8. Show that all vertex-induced subgraphs of a chordal graph are also chordal graphs.
9. Construct a tree decomposition of the graph in Fig. 9.25 which is not a clique tree.
10. Construct a SPQ-tree decomposition of the graph in Fig. 9.26.

References

1. Harary, F.: Graph Theory. Addison-Wesley, Reading, Mass (1972)
2. Jao, K.F., West, D.B.: Vertex Degrees in Outerplanar Graphs, preprint (2010). www.math.uiuc.edu/west/pubs/outerpl.pdf
3. Khan, N., Karima, N., Rahman, M.S., Hossain, M.I.: Orthogonal grid pointset embeddings of maximal outerplanar graphs. In: Proceedings of International Conference on Electrical Engineering and Information and Communication Technology (ICEEICT) 2014. IEEE Explore Digital Library (2014)
4. Nishizeki, T., Rahman, M.S.: Planar Graph Drawing. World Scientific, Singapore (2004)
5. de Fraysseix, H., Pach, J., Pollack, R.: How to draw a planar graph on a grid. Combinatorica **10**, 41–51 (1990)
6. Khan, I.K., Rayana, S., Alam, M.J., Rahman, M.S.: On Topological Book Embedding with the Minimum Number of Spine Crossings, Manuscript (2009)

7. Biedl, T., Velasquez, L.E.R.: Drawing planar 3-trees with given face-areas. In: Proceedings of the 17th International Symposium on Graph Drawing (GD 2009), Lecture Notes in Computer Science, vol. 5849, pp. 316–322. Springer, Heidelberg (2010)
8. Mondal, D., Nishat, R.I., Rahman, M.S., Alam, M.J.: Minimum-area drawings of plane 3-trees. J. Graph Algorithms Appl. **15**(2), 177–204 (2011)
9. Spinrad, J.P.: Efficient Graph Representations. American Mathematical Society, Rhode Island (2003)
10. Dirac, G.A.: On rigid circuit graphs. Anh. Math. Sem. Univ. Hamburg **25**, 71–76 (1961)
11. Gilmore, P.C., Hoffman, A.J.: A characterization of comparability graphs and of interval graphs. Canadian J. Math. **16**, 539548 (1964)
12. Rahman, M.S., Egi, N., Nishizeki, T.: No-bend orthogonal drawings of series-parallel graphs. In: Proceedings of the 13th International Conference on Graph Drawing (GD'05), Lecture Notes in Computer Science, vol. 3843, pp. 409–420. Springer, Heidelberg (2006)
13. Garg, A., Liotta, G.: Almost bend-optimal planar orthogonal drawings of biconnected degree-3 planar graphs in quadratic time. In: Proceedings of Graph Drawing'99. Lecture Notes in Computer Science, vol. 1731, pp. 38–48. Springer, Heidelberg (1999)
14. Di Battista, G., Tamassia, R.: On-line maintenance of triconnected components with SPQR-trees. Algorithmica **15**(4), 302–318 (1996)
15. Di Battista, G., Tamassia, R., Vismara, L.: Output-sensetive reporting of disjoint paths. Algorithmica **23**, 302–340 (1999)
16. Hossain, M.I., Rahman, M.S.: Straight-line monotone grid drawings of series-parallel graphs. Discret. Math. Alg. Appl. **7**(2) (2015)
17. Arnborg, S., Corneil, D., Proskurowski, A.: Complexity of finding embeddings in a k-tree. SIAM J. Matrix Anal. Appl. **8**(2), 277–284 (1987)
18. Bodlaender, H.L.: A tourist guide through treewidth. In: Dassow, J., Kelemenová, A. (eds.) Developments in Theoretical Computer Science (Proc. 7th International Meeting of Young Computer Scientists, Smolenice, 16–20 November 1992), Topics in Computer Mathematics, vol. 6, pp. 1–20. Gordon and Breach (1994)

Chapter 10
Some Research Topics

10.1 Introduction

In this chapter, we introduce some research areas where the students learning graph theory may find interest. As we have seen in Chapter 1, graphs have applications in modeling many real-world problems. So a researcher in graph theory can work on fundamental properties of graphs as well as developing graph models for practical applications and on devising efficient algorithms.

10.2 Graph Representation

In Chapter 2, we have read adjacency matrix and adjacency list for graph representation and learned that an adjacency matrix needs $O(n^2)$ space and an adjacency list takes $O(n+m)$ space to represent a graph of n vertices and m edges. Which one is better? The answer is not straightforward; it depends on which graph we are going to represent. In this section, we focus on the space complexity of graph representations in terms of the requirements of bits for storing the graph and see that different classes of graphs admit different forms of efficient representations [1].

We first address the space complexity of the two representations mentioned above. Adjacency matrices use $\Theta(n^2)$ bits to store a graph and the same amount of space is required for every graph. Adjacency lists take a total of $2m$ entries for an undirected graph, since each entry appears on two lists. Each entry holds a number in the range $1 \ldots n$, and thus takes $\log n$ bits. Thus the total space complexity is $\Theta(m \log n)$ bits.

Let a set has $2^{\Theta(f(n))}$ elements. We call a representation of the members of the set by $O(f(n))$ bits an *optimal representation*. Using this definition, we call the adjacency matrix representation an optimal representation, since it uses $O(n^2)$ bits to represent a set of $2^{n(n-1)/2}$ graphs of n vertices. On the other hand, adjacency list is not space optimal since it used $\Theta(n^2 \log n)$ bits to represent a complete graph.

M.S. Rahman, *Basic Graph Theory*, Undergraduate Topics in Computer Science,
DOI 10.1007/978-3-319-49475-3_10

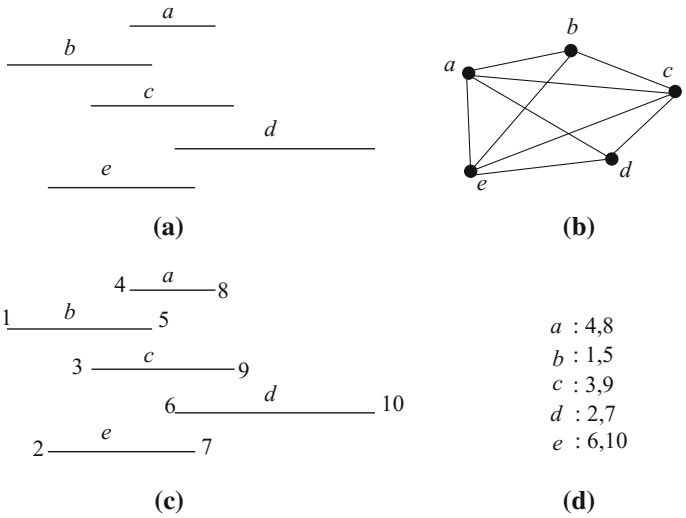

Fig. 10.1 Illustration for an efficient representation of interval graphs: **(a)** intervals, **(b)** the interval graph G corresponding to the intervals in (a), **(c)** labeling of the ends of the intervals by integers, and **(d)** an efficient representation of G using the labeling in (c)

As well as space complexity, there are many other aspects of graph representations. These are speed for adjacency testing, ease of finding the representation, help with solving optimization problems, etc. Adjacency matrix and adjacency lists representations may not work fine in all aspects for every classes of graphs.

We take the class of interval graphs as an example [1]. In an interval graph, vertices corresponding to intervals on the real line, and two vertices are adjacent if and only if the corresponding intervals have a nonempty intersection. Figure 10.1(b) shows an interval graph corresponding to the intervals in Fig. 10.1(a). We cannot capture this property in either adjacency matrix and adjacency list data structure. However, this interval property of interval graphs leads to an efficient representation as follows. Label the endpoints of intervals in increasing order of their appearance on the line, as illustrated in Fig. 10.1(c). This assigns to every vertex two integers in the range $1 \ldots 2n$. For each vertex v, we store the labels of the endpoints of v; this uses $O(\log n)$ bits per vertex. We can easily determine whether two vertices x and y are adjacent in constant time from the label stored for x and y, as illustrated in Fig. 10.1(d). Thus we can reconstruct the graph from $O(\log n)$ bits at each vertex, and interval graphs can be stored in $O(n \log n)$ bits space. Thus, this representation is a space-efficient representation with fast adjacency test capability.

Thus, efficient representation of a graph depends on the classes of graphs which we want to represent. Some special properties of a class make the class suitable for a particular representation. The following research areas are closely related to the study of efficient graph representations.

(a) Counting: The number of graphs in a particular class gives a lower bound on the space complexity of its representation. Thus, counting the number of graphs in a special class of graphs is an interesting research problem.
(b) Representation: Finding space optimal representation with fast adjacency testing for different classes of graphs is the main focus of the representation problem. Space optimal representation is motivated from the applications in smart devices in recent years. Determining whether a particular form of representation exists for a given class of graphs is an interesting problem.
(c) Constructing Representation: Some classes of graphs may have efficient representation, but they may not be constructed easily. Thus, efficient construction of a representation is also an interesting research problem.
(d) Recognition: The recognition problem for a class of graphs is the problem of determining whether a given graph is in some class. This problem is interesting not only from the representation point of view, but also for other graph algorithmic area. Say, some algorithm works for a particular class of graphs. If we can determine whether the given graph is in that class, we can use the algorithm.
(e) Characterization: The problem of deriving necessary and sufficient conditions for a particular class of graphs is known as characterization problem. By deriving such conditions for the graphs in a class, we can develop efficient algorithms for recognizing the graphs in the class.

Content of this section is based on the book of Spinrad [1], interested readers can find more on graph classes and their representations in the book.

10.3 Graph Drawing

A drawing of a graph can be thought of as a diagram consisting of a collection of objects corresponding to the vertices of the graph together with some line segments corresponding to the edges connecting the objects. People are using diagrams from ancient time to represent abstract things like ideas, concepts, etc., as well as concrete things like maps, structures of machines, etc. A graph may be used to represent any information which can be modeled as objects and relationship between those objects. A drawing of a graph is a sort of visualization of information represented by the graph. The graph in Fig. 10.2(a) represents eight components and their interconnections in an electronic circuit, and Fig. 10.2(b) depicts a drawing of the graph. Although the graph in Fig. 10.2(a) correctly represents the circuit, the representation is messy and hard to trace the circuit for understanding and troubleshooting. Furthermore, in this representation one cannot lay the circuit on a single-layered PCB (Printed Circuit Board) because of edge crossings. On the other hand, the drawing of the graph in Fig. 10.2(b) looks better and it is easily traceable. Furthermore one can use the drawing to lay the circuit on a single-layered PCB, since it has no edge crossing. Thus, the objective of graph drawing is to obtain a nice representation of a graph such

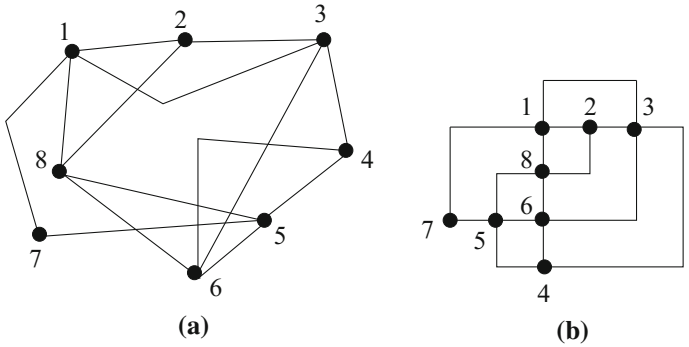

Fig. 10.2 An example of graph drawing in circuit schematics

that the structure of the graph is easily understandable, and moreover the drawing should satisfy some criteria that arises from the application point of view.

The field of graph drawing has been flourished very much in the past two decades. Recent progress in computational geometry, topological graph theory, and order theory has considerably affected the evolution of this field, and has widened the range of issues being investigated. Several books on graph drawing have been published [2–6]. In the following subsections, we will know current research trends in graph drawing.

10.3.1 Drawings of Planar Graphs

10.3.1.1 Straight-Line Drawing

In Section 6.5 we have seen that every planar graph has a straight-line drawing. A *straight-line grid drawing* of a planar graph G is a straight-line drawing of G on an integer grid such that each vertex is drawn as a grid point. Figure 10.3(b) illustrates a straight-line grid drawing of the planar graph in Fig. 10.3(a). In 1990, de Fraysseix et al. [7] and Schnyder [8] showed by two different methods that every planar graph of $n \geq 3$ vertices has a straight-line grid drawing on an integer grid of size $(2n-4) \times (n-2)$ and $(n-2) \times (n-2)$, respectively. A natural question arises: what is the minimum size of a grid required for a straight-line grid drawing? de Fraysseix et al. showed that, for each $n \geq 3$, there exists a plane graph of n vertices, for example nested triangles, which needs a grid size of at least $\lfloor 2(n-1)/3 \rfloor \times \lfloor 2(n-1)/3 \rfloor$ for any straight-line grid drawing [7, 9]. It has been conjectured that every plane graph of n vertices has a straight-line grid drawing on a $\lceil 2n/3 \rceil \times \lceil 2n/3 \rceil$ grid, but it is still an open problem. For some restricted classes of graphs, more compact straight-line grid drawings are known. For example, a 4-connected plane graph G having at least four vertices on the outer face has

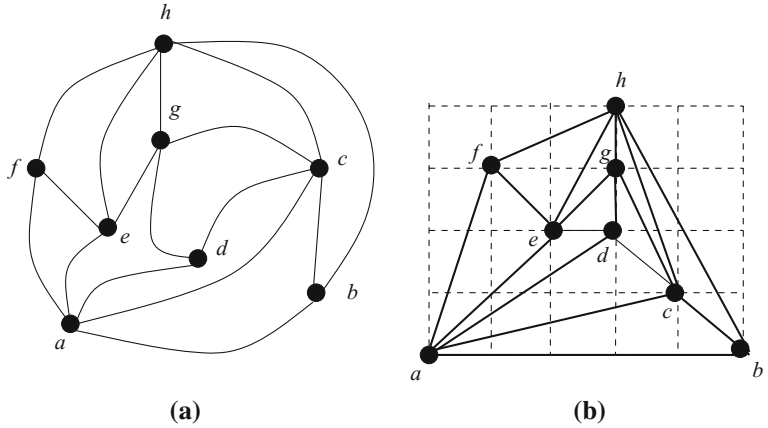

Fig. 10.3 **(a)** A planar graph G and **(b)** a straight-line drawing of G

a straight-line grid drawing with area $(\lceil n/2 \rceil - 1) \times (\lfloor n/2 \rfloor)$ as shown in Fig. 10.4 [10]. Garg and Rusu showed that an n-node binary tree has a planar straight-line grid drawing with area $O(n)$ [11, 12]. Although trees admit straight-line grid drawings with linear area, it is generally thought that triangulations may require a grid of quadratic size. Hence, finding nontrivial classes of planar graphs of n vertices richer than trees that admit straight-line grid drawings with area $o(n^2)$ is posted as an open problem in [13]. Garg and Rusu showed that an outerplanar graph with n vertices and the maximum degree d has a planar straight-line grid drawing with area $O(dn^{1.48})$ [14]. Di Battista and Frati showed that a "balanced" outerplanar graph of n vertices has a straight-line grid drawing with area $O(n)$ and a general outerplanar graph of n vertices has a straight-line grid drawing with area $O(n^{1.48})$ [15]. Karim and Rahman [16] introduced a new class of planar graphs which has a straight-line grid drawing on a grid of area $O(n)$. They also gave a linear-time algorithm to find such a drawing. The new class of planar graphs is a subclass of 5-connected planar graphs, and they call the class "doughnut graphs," since a graph in this class has a doughnut-like embedding. In an embedding of a doughnut graph of n vertices, there are two vertex-disjoint faces each having exactly $n/4$ vertices and each of all the other faces has exactly three vertices. Figure 10.5(b) shows a straight-line grid drawing of the doughnut graph in Fig. 10.5(a) on a grid of linear area. Finding other nontrivial classes of planar graphs that admit straight-line grid drawings on grids of linear area is an interesting open problem.

10.3.1.2 Convex Drawing

A convex drawing of a planar graph G is a planar drawing of G where every vertex is drawn as a point, every edge is drawn as a straight-line segment, and every face is drawn as a convex polygon. Not every planar graph has a convex drawing. The

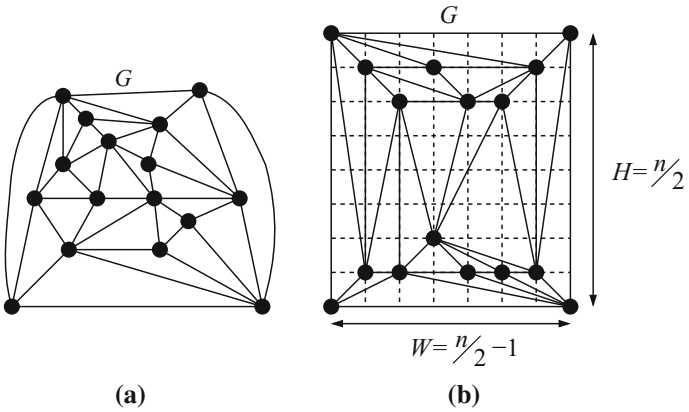

Fig. 10.4 **(a)** A planar graph G and **(b)** a straight-line drawing of G on a $(\lceil n/2 \rceil - 1) \times (\lfloor n/2 \rfloor)$ grid

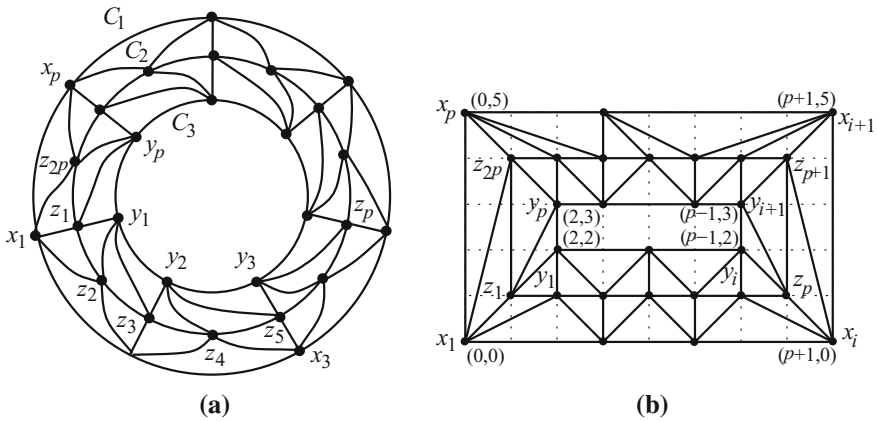

Fig. 10.5 **(a)** A doughnut graph G and **(b)** a straight-line drawing of G on a grid of linear area

planar graph in Fig. 10.6(a) has a convex drawing as shown in Fig. 10.6(b) whereas the planar graph in Fig. 10.6(c) has no convex drawing. Tutte [17] showed that every 3-connected planar graph has a convex drawing, and obtained a necessary and sufficient condition for a planar graph to have a convex drawing with a prescribed outer polygon. Furthermore, he gave a "barycentric mapping" method for finding a convex drawing, which requires solving a system of $O(n)$ linear equations [18], and leads to an $O(n^{1.5})$ time convex drawing algorithm for a planar graph with a fixed embedding. Chiba, Yamanouchi, and Nishizeki [19] gave linear algorithms for determining whether a planar graph (where the embedding is not fixed) has a convex drawing and finding such a drawing if it exits.

A convex drawing is called a *convex grid drawing*, if each vertex is drawn at a grid point of an integer grid. Using canonical ordering and shift method, Chrobak and Kant

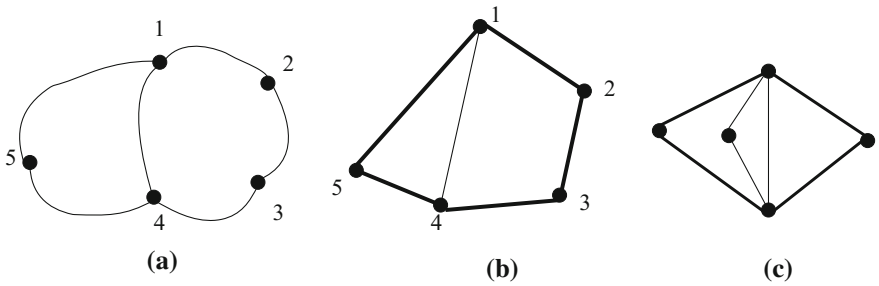

Fig. 10.6 **(a)** A planar graph G, **(b)** a convex drawing of G, and **(c)** a planar graph having no convex drawing

[20] showed that every 3-connected plane graph has a convex grid drawing on an $(n-2) \times (n-2)$ grid and such a grid drawing can be found in linear time. However, the question of whether every planar graph which has a convex drawing admits a convex grid drawing on a grid of polynomial size is remained as an open problem. Several research works are concentrated in this direction [21–23]. For example, Zhou and Nishizeki showed that every internally triconnected plane graph G whose "decomposition tree $T(G)$" has exactly four leaves has a convex grid drawing on a $2n \times 4n = O(n^2)$ grid, and presented a linear algorithm to find such a drawing [23].

10.3.1.3 Orthogonal Drawing

An *orthogonal drawing* of a plane graph G is a drawing of G with the given embedding in which each vertex is mapped to a point, each edge is drawn as a sequence of alternate horizontal and vertical line segments, and any two edges do not cross except at their common end. A *bend* is a point where an edge changes its direction in a drawing. Every plane graph of the maximum degree four has an orthogonal drawing, but may need bends. For the cubic plane graph in Fig. 10.7(a) each vertex of which has degree 3, two orthogonal drawings are shown in Figs. 10.7(b) and (c) with 6 and 5 bends respectively. Orthogonal drawings have applications in circuit schematics, and it is desired to find an orthogonal drawing with a small number of bends, because a bend corresponds to a "via" or "throughhole," and increases the fabrication cost of VLSI [24]. Minimization of the number of bends in an orthogonal drawing is a challenging problem. Several works have been done on this issue [25–27]. In particular, Garg and Tamassia [25] presented an algorithm to find an orthogonal drawing of a given plane graph G with the minimum number of bends in time $O(n^{7/4}\sqrt{\log n})$, where n is the number of vertices in G. Rahman et al. gave an algorithm to find an orthogonal drawing of a given triconnected cubic plane graph with the minimum number of bends in linear time [26]. Note that for a given planar graph G of n vertices, if one is allowed to choose its planar embedding, then finding

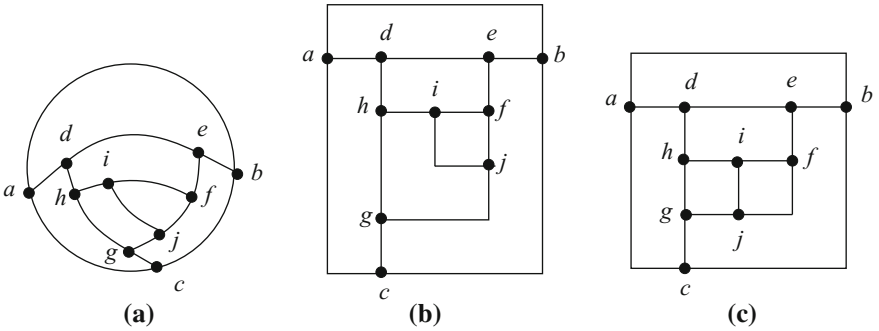

Fig. 10.7 **(a)** A plane graph G, **(b)** an orthogonal drawing of G with 6 bends, and **(c)** an orthogonal drawing of G with 5 bends

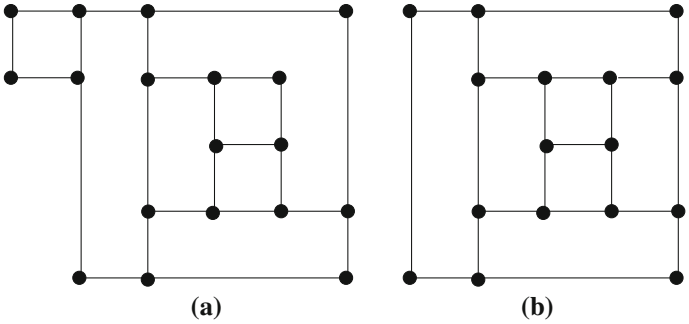

Fig. 10.8 **(a)** A no-bend orthogonal drawing and **(b)** a rectangular drawing

an orthogonal drawing of G with the minimum number of bends is NP-complete [27].

A plane graph G may have an orthogonal drawing without bends. An orthogonal drawing D of a plane graph G is called a *no-bend orthogonal drawing* of G if D has no bend, as illustrated in Fig. 10.8(a). However, not every plane graph has a no-bend orthogonal drawing. For example, the cubic plane graph in Fig. 10.7(a) has no no-bend orthogonal drawing, since any orthogonal drawing of an outer cycle have at least four convex corners which must be bends in a cubic graph. Rahman et al. gave a simple necessary and sufficient condition for a plane 2-connected graph with $\Delta \leq 3$ to have a no-bend orthogonal drawing [28]. A no-bend orthogonal drawing D of a plane graph G is called a *rectangular drawing* of G if every face including the outer face of G is drawn as a rectangle in D, as illustrated in Fig. 10.8(b). In Section 1.2.4, we have seen applications of rectangular drawing of a plane graph in VLSI floorplanning and architectural floorplanning. A necessary and sufficient condition for a plane graph G to have a rectangular drawing is known [29], and several linear-time algorithms to find a rectangular drawing of G are also known [30–32].

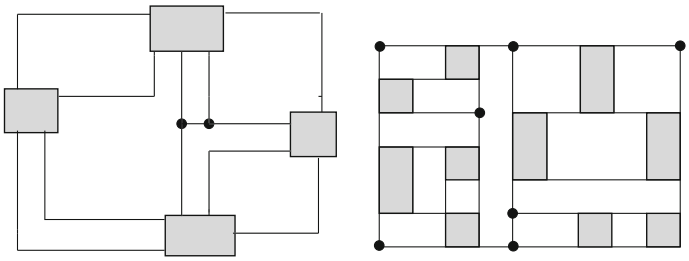

Fig. 10.9 **(a)** A box-orthogonal drawing and **(b)** a box-rectangular drawing

An orthogonal drawing is called an *orthogonal grid drawing* if all vertices and bends are located on integer grid points. Given an orthogonal drawing, one can transform it to an orthogonal grid drawing in linear time [5]. Let W be the *width* of a grid, that is, the number of vertical lines in the grid minus one, and let H be the *height* of a grid. Let n be the number of vertices, and let m be the number of edges in a given graph. It is known that any orthogonal drawing using b bends has a grid drawing on a grid such that $W + H \leq b + 2n - m - 2$ [33]. Finding orthogonal grid drawings of plane graphs on a grid of smaller area is desirable for VLSI design. The algorithm of Rahman et al. [26] produces a bend-minimum orthogonal grid drawing of a 3-connected cubic plane graph on a grid such that $W + H \leq b(G) + \frac{1}{2}n - 2$, $W \leq \frac{n}{2}$ and $H \leq \frac{n}{2}$. Finding orthogonal grid drawings of plane graphs on a grid of smaller area is an interesting research area.

A plane graph with a vertex of degree 5 or more has no orthogonal drawing. However in literature, a "box-orthogonal drawing" is found as a generalization of an orthogonal grid drawing of a plane graph with $\Delta \geq 5$ [5]. A *box-orthogonal drawing* of a plane graph G is a drawing of G on an integer grid such that each vertex is drawn as a rectangle, called a *box*, and each edge is drawn as a sequence of alternate horizontal and vertical line segments along grid lines, as illustrated in Fig. 10.9(a). Some of the boxes may be degenerated rectangles, i.e., points. Several results are known for box-orthogonal drawings [34, 35]. A "box-rectangular drawing" is a generalization of a rectangular grid drawing for a plane graph with $\Delta \geq 5$. A *box-rectangular drawing* of a plane graph G is a drawing of G on an integer grid such that each vertex is drawn as a (possibly degenerated) rectangle, called a *box*, and the contour of each face is drawn as a rectangle, as illustrated in Fig. 10.9(b). If G has multiple edges or a vertex of degree 5 or more, then G has no rectangular drawing but may have a box-rectangular drawing. However, not every plane graph has a box-rectangular drawing. Rahman et al. [36] gave a necessary and sufficient condition for a plane graph to have a box-rectangular drawing. Linear-time algorithms are also known for finding a box-rectangular drawing of a plane graph if it exists [36, 37].

In recent years research on orthogonal drawings have focused on finding orthogonal drawings of plane graphs with prescribed areas [38] and shapes on faces [39].

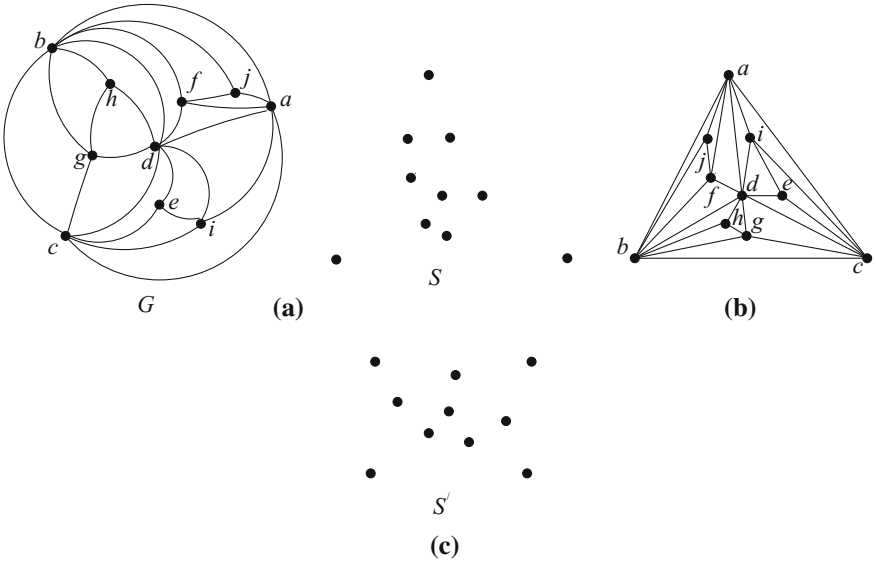

Fig. 10.10 **(a)** A plane graph G with ten vertices and a set S of ten points, **(b)** a point-set embedding of G on S and **(c)** a point-set of ten vertices on which G has no point-set embedding

Finding orthogonal drawings with various constraints, such as bends are only allowed on a prescribed subset of edges or each edge is allowed to have a prescribed number of bends, is an interesting research area.

10.3.1.4 Point-Set Embedding

Let G be a plane graph with n vertices and let S be a set of n points in the Euclidean plane. A *point-set embedding of G on S* is a straight-line drawing of G, where each vertex of G is mapped to a distinct point of S. We do not restrict the points of S to be in general position. In other words, three or more points in S may be collinear. Figure 10.10(a) depicts a plane graph G of ten vertices and a point-set S of ten vertices, and a point-set embedding of G on S is illustrated in Fig. 10.10(b). However, G in Fig. 10.10(a) does not have a point-set embedding on the point-set S' in Fig. 10.10(c), since the convex hull of S' contains four points whereas the outer face of G has three vertices.

A rich body of literature has been published on point-set embeddings when the input graph G is restricted to trees or outerplanar graphs. Given a tree T with n nodes and a set S of n points in general position, Ikebe et al. [40] proved that T always admits a point-set embedding on S. In 1991 Gritzmann et al. gave a quadratic-time algorithm to obtain an embedding of an outerplanar graph G with n vertices on a set of n points in general position. A significant improvement on this problem is given by Bose [41]; who presented an $O(n \log^3 n)$ time algorithm to compute a

straight-line embedding of an outerplanar graph G with n vertices on a set of n points. Cabello [42] proved that the problem is NP-complete for planar graphs in general and even when the input graph is 2-connected and 2-outerplanar. Garcia et al. gave a characterization of a set S of points such that there exists a 3-connected cubic plane graph that admits a point-set embedding on S [43]. Recently, Nishat et al. [44] gave an $O(n^2)$ time algorithm that decides whether a plane 3-tree G with n vertices admits a point-set embedding on a set S of n points or not; and computes a point-set embedding of G if such an embedding exists. They also showed that the problem is NP-complete for planar partial 3-trees. Durocher and Mondal [45] proved that the point-set embeddability problem remains NP-complete for 3-connected plane graphs.

Point-set embedding is an active area of research. Researchers are considering the problem from various point of views. A set P of points in $\mathcal{R}^2$ is n-universal, if every planar graph of n vertices admits a plane straight-line drawing on P. Finding an n-universal point-set of smaller size is an interesting area of research [46, 47]. Instead of point-set, this problem has also been studied on line-sets [48, 49]. Interesting research problems can be formulated by combining other drawing styles like convex drawing, orthogonal grid drawing with point-set embeddings [45, 50].

10.3.2 Simultaneous Embedding

Usually in a graph-drawing problem we deal with a drawing of a single graph. But in a simultaneous graph-drawing problem, we are concerned with the layout of a series of graphs that share all, or parts of the same vertex set. Figure 10.11(b) illustrates a simultaneous embedding of the two graphs in Fig. 10.11(a). In a drawing of such a series of graph readability of the individual graph as well as mental map representation are two important parameters [51]. Readability depends on some aesthetic criteria, whereas mental map preservation demands to make the position of the vertices in the drawing unchanged of the series of graphs. Simultaneous embedding finds applications in software engineering, database, in displaying phylogenetic trees etc. In evolutionary biology, phylogenetic trees are used to visualize the ancestral relationship among groups of species. Comparing two phylogenetic trees produced by different methods on the same set of species become easier if a simultaneous embedding of the two trees is constructed. Simultaneous embedding is an active area of research and there are many interesting open problems on simultaneous embeddings [51].

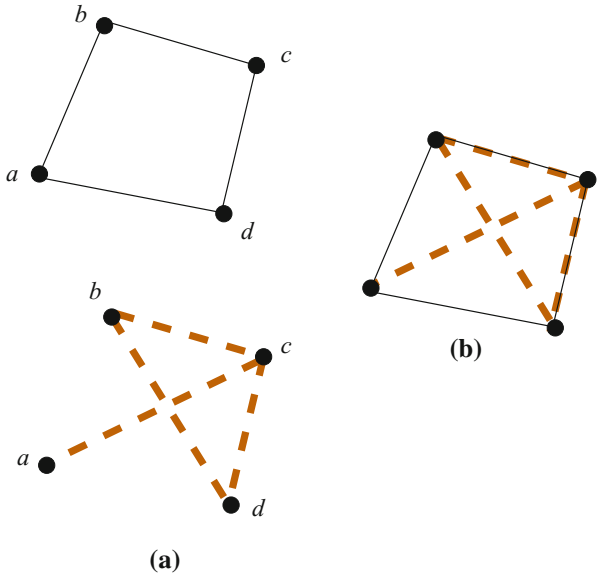

Fig. 10.11 **(a)** Two graphs, and **(b)** a simultaneous embedding of the two graphs in (**a**)

10.3.3 Drawings of Nonplanar Graphs

In Section 10.3.1 we considered drawings of planar graphs. Planar drawings are easily readable and aesthetically pleasant. Unfortunately not all graphs are planar. In recent years researchers have become interested in drawings of nonplanar graphs. In this section, we thus consider drawings of nonplanar graphs.

10.3.3.1 RAC Drawing

Usually in drawing nonplanar graphs, special attention is paid to edge crossings. It is desirable to draw a nonplanar graph with the minimum number of edge crossings. Unfortunately, the edge crossing minimization problem is NP-complete in general [52]. However, interesting eye-tracking experiments by Huang et al. [53, 54] indicate that the negative impact of an edge crossing is eliminated in the case where the crossing angle is greater than 70°. These results motivated the study of a new class of drawings, called right-angle drawings or RAC drawings for short. A *RAC drawing of a graph* is a polyline drawing in which every pair of crossing edges intersects at right angle. Figure 10.12 shows a RAC drawing of a nonplanar graph. Didimo, Eades, and Liotta [55] proved that it is always feasible to construct a RAC drawing of a given graph with at most three bends per edge. They also showed that any straight-line RAC drawing with n vertices has at most $4n - 10$ edges. Eades

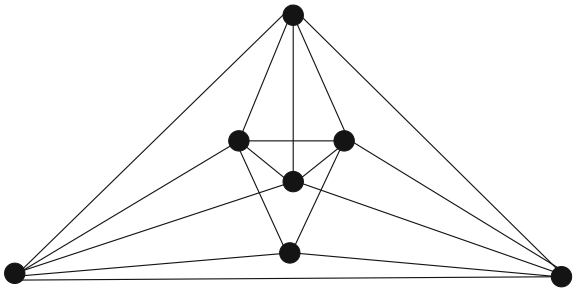

Fig. 10.12 A RAC drawing of a nonplanar graph

and Liotta [56] showed that every RAC graph with n vertices and $4n - 10$ edges (such graphs are called maximally dense) is 1-planar, i.e., it admits a drawing in which every edge is crossed by at most one other edge. Argyriou et al. [57] showed that the problem of determining whether an input graph admits a straight-line RAC drawing is NP-hard. Thus, reducing the number of bends in RAC drawings of some nontrivial nonplanar graphs is an interesting area of research.

10.3.3.2 Bar k-Visibility Drawing

A *bar visibility drawing* of a planar graph G is a drawing of G, where each vertex is drawn as a horizontal line segment and each edge is drawn as a vertical line segment such that the vertical line segment representing an edge must connect the horizontal line segments representing the end vertices. Otten and Wijk [58] showed that every planar graph admits a visibility drawing, and Tamassia and Tollis [59] gave $O(n)$ time algorithm for constructing a visibility drawing of a planar graph of n vertices. Dean et al. [60] introduced a generalization of visibility drawing for a nonplanar graph which is called a bar k-visibility drawing. In a *bar k-visibility drawing* of a graph, the vertical line segment corresponding to an edge intersects at most k bars which are not end points of the edge. Thus a visibility drawing is a bar k-visibility drawing for $k = 0$.

A *bar* 1*-visibility drawing* of a graph G is a drawing of G where each vertex is drawn as a horizontal line segment called a bar, each edge is drawn as a vertical line segment between its end vertices such that each edge crosses at most one bar. A graph G is *bar* 1*-visible* if G has a bar 1-visibility drawing. A bar 1-visible graph and a bar 1-visibility drawing of the same graph is shown in Figs. 10.13(a) and (b), respectively.

A 1*-planar drawing* of a graph G is a drawing of G on a two-dimensional plane where each edge can be crossed by at most one other edge. A graph G is 1*-planar* if G has a 1-planar drawing.

Sultana et al. [61] gave linear algorithms for finding bar 1-visibility drawings of some subclasses of 1-planar graphs and conjectured that every 1-planar graph has

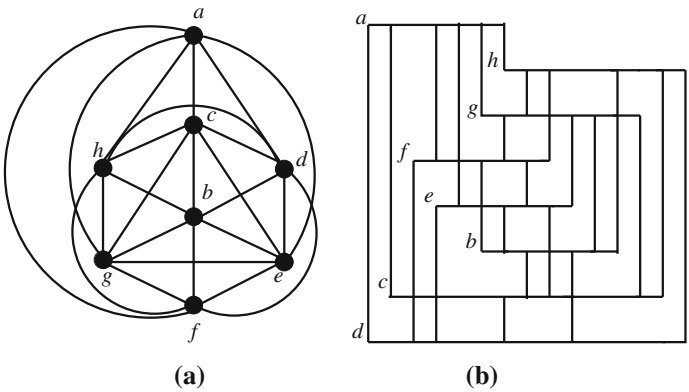

Fig. 10.13 **(a)** A 1-planar graph G, **(b)** a bar 1-visibility drawing of G

a bar 1-visibility drawing. The conjecture of Sultana et al. triggered the research in this area and several results [62] are published recently.

10.4 Graph Labeling

Graph labeling is an interesting research topic of graph theory which has many applications in computer science and engineering. It concerns the assignment of values, usually represented by integers, to the edges and/or vertices of a graph such that the assignments meet certain conditions within the graph. Many of the graph labeling methods were motivated by applications to coding theory, circuit design, communication network addressing, secret sharing schemes, etc. [63]. In Section 4.7 we came to know about graceful labeling. Graph colorings studied in Chapter 7 are also graph labelings. Canonical ordering described in Section 9.3.1 is a labeling of vertices of a triangulated planar graphs. In this section, we study a popular vertex labeling called antibandwidth labeling which has many practical applications such as the frequency assignment described in Section 1.2.2.

In antibandwidth problem [64, 65], we are asked to label the vertices in such way that the minimum difference between the labels of two adjacent vertices is maximized. More formally, let G be a graph on n vertices. Given a bijection $f : V(G) \rightarrow \{1, 2, 3, \ldots, n\}$, if $|f| = \min\{|f(u) - f(v)| : uv \in E(G)\}$ then the antibandwidth of G is the maximum $\{|f|\}$ over all such bijections f of G. In Fig. 10.14, an antibandwidth labeling of a full binary tree of 15 vertices is shown where the minimum difference between the two labelings of any two adjacent vertices is 7. In the literature, the antibandwidth problem is also known as dual bandwidth problem [66], separation number of graphs [67], and maximum differential graph coloring problem [68].

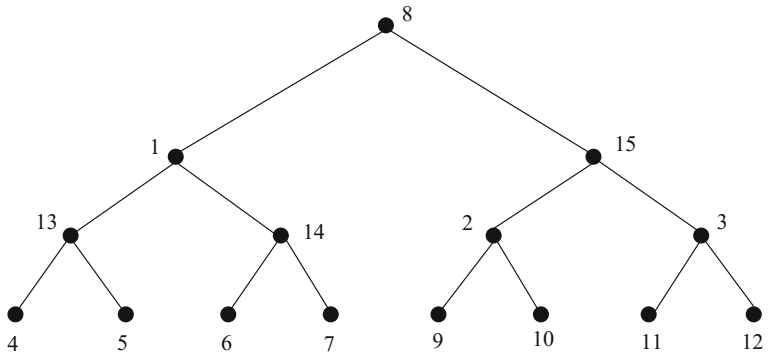

Fig. 10.14 Antibandwidth labeling of a full binary tree

The antibandwidth problem is NP-hard [67] for general graphs. It is known to be polynomially solvable for the complements of interval, "arborescent comparability" and "threshold" graphs [69]. Exact results and tight bounds are known for paths, cycles, grid, meshes, hypercubes [64, 66, 68]. For a complete binary tree with height h the antibandwidth is $2^h - 1$ [70] and for complete k-ary tree [71], the antibandwidth is $\frac{n+1-k}{2}$, when k is even. Millar et al. [72] introduced an antibandwidth labeling scheme for forests, which is known as Millar–Pritikin labeling scheme. According to Millar–Pritikin labeling scheme, the antibandwidth of a forest with the bipartition X and Y is at least $|X|$, where $|X| \leq |Y|$. They also showed that for a balanced bipartite graph ($|X| = |Y|$) Millar–Pritikin labeling scheme produces optimal antibandwidth value.

Finding the antibandwidth of a graph has several practical applications other than the frequency assignment [65]. For example, if the vertices of the graph G represent sensitive facilities or chemicals, then placing them too close together can be risky. Formally the problem is known as enemy facility location problem [71]. Antibandwidth labeling has also an application in map coloring problem where countries in the map are not all contiguous. That is, two regions of a country might be separated by other countries. This necessitates the use of a unique color for each country to avoid ambiguity. Furthermore, the colors selected for coloring two countries having shared nontrivial boundary should be clearly distinct. Given a map, we define the country graph G to be the undirected graph where countries are nodes and two countries are connected by an edge if they share a nontrivial boundary. We then consider the problem of assigning colors to nodes of G so that the color distance between nodes that share an edge is maximized [68].

There are several open problems on antibandwidth labeling [73].

- For general caterpillars, it is not known whether the antibandwidth problem can be solved in polynomial time or whether it is an NP-hard problem.
- Computational complexity of antibandwidth problem is not also known for general trees and outerplanar graphs although it is known to be NP-complete for planar graphs.
- Finding optimal labeling schemes and approximations for special classes of graphs like lobster, interval graphs, cubic graphs, and regular bipartite graphs.

10.5 Graph Partitioning

Given a graph $G = (V, E)$, k distinct vertices $u_1, u_2, \ldots, u_k \in V$ and k natural numbers $n_1, n_2, \ldots, n_k$ such that $\sum_{i=1}^{k} n_i = |V|$, we wish to find a partition $V_1, V_2, \ldots, V_k$ of the vertex set V such that $u_i \in V_i$, $|V_i| = n_i$, and V_i induces a connected subgraph of G for each i, $1 \leq i \leq k$. Such a partition is called a *k-partition* of G. The vertices $u_1, u_2, \ldots, u_k$ are called the *base vertices* for the k-partition. A 4-partition of a graph G is depicted in Fig. 10.15, where the edges of four connected subgraphs are drawn by solid lines and the remaining edges of G are drawn by dotted lines. The problem of finding a k-partition of a given graph often appears in the load distribution among different power plants and the fault-tolerant routing of communication networks [74, 75]. The problem is NP-hard in general [76], and hence it is very unlikely that there is a polynomial-time algorithm to solve the problem. Although not every graph has a k-partition, Györi and Lovász independently proved that every k-connected graph has a k-partition for any base vertices $u_1, u_2, \ldots, u_k$ and $n_1, n_2, \ldots, n_k$ [77, 78]. However, their proofs do not yield any polynomial-time algorithm for actually finding a k-partition of a k-connected graph. For the case $k \leq 5$, the following algorithms are known:

(i) a linear-time algorithm to find a bipartition of a biconnected graph [79, 80];
(ii) an algorithm to find a tripartition of a triconnected graph in $O(n^2)$ time, where n is the number of vertices of a graph [79];
(iii) a linear-time algorithm to find a tripartition of a triconnected planar graph [81];
(iv) a linear-time algorithm to find a 4-partition of a four-connected planar graph with base vertices located on the same face of the given graph [82];
(v) a linear-time algorithm for 5-partitioning of a 5-connected internally triangulated plane graph if the five base vertices are all on the boundary of one face [83]; and
(vi) a linear-time algorithm for finding a k-partition of a doughnut graph G with at most two base vertices [84].

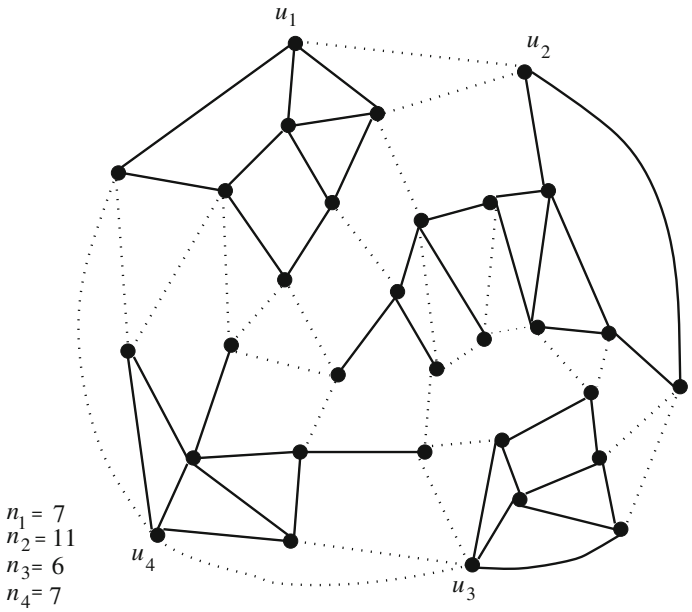

Fig. 10.15 A 4-partitioning of a 4-connected plane graph G

Finding a polynomial-time algorithm for k-partitioning k-connected graphs is a challenging open problem.

A variation of k-partitioning problem, known as "resource k-partitioning problem" is also found in the literature [75, 85, 86]. A *resource k-partitioning* is defined as a partition $V_1, V_2, \ldots, V_k$ of V with a set $V_r \subseteq V$ of r resource vertices, base vertices $u_1, u_2, \ldots, u_k \in V$, k natural numbers $r_1, r_2, \ldots, r_k$ such that $\sum_{j=1}^{k} r_j = r$, where $u_i \in V_i$, V_i contains r_i resource vertices and V_i induces a connected subgraph of G for each i, $1 \leq i \leq k$. Resource partitioning has significant applications in various areas. In a computer network printers, routers, scanners, etc., can be considered as resources. Resources require partitioning to balance loads on these resources and to prevent network traffic bottleneck. In a multimedia network, it is required to assign a server to a specific group of clients for balancing loads among the servers. Resource partitioning has its application in the fault-tolerant routing of communication networks [75] and in computational aspects, too. For example, in grid computing we are required to divide a complex task into several subtasks and then we wish to delegate each of these subtasks to a computing element in the grid such that no computing element is overwhelmed with tasks. This concept applies to telecommunication networks, fault-tolerant systems, various producer–consumer problems, and so on [86]. Graph partitioning has become very useful concepts for clustering in the areas of social network analysis [87] and wireless sensor networks [88].

10.6 Graphs in Bioinformatics

Graph theory has a glorious history with bioinformatics. Prior to Watson and Crick elucidation of the DNA double helix, it seemed a reasonable hypothesis that the DNA content of genes was branched or even looped rather than linear. But in 1950s, Seymour Benzer applied graph theory to show that genes are linear. Benzer's problem was equivalent to deciding whether the graph obtained from his bacteriophage experiment represented an interval graph [89].

To deal with the problems of bioinformatics, researchers of graph algorithms often develop a graph model for a problem of bioinformatics. Then they try to use graph algorithmic techniques to solve the problem on the graph model. In the next three subsections, we know some graph models of bioinformatics problems.

10.6.1 Hamiltonian Path for DNA Sequencing

In this section, we consider a very simplified version of DNA sequencing problem called sequencing by hybridization [89]. Given an unknown DNA sequence, an array provides information about all strings of length l that the sequence contains, but does not provide information about their position in the sequence. The problem is to predict the DNA sequence. For a string s of length n, there are $n - l + 1$ substrings of length l. These substrings are called *l-mers*, and the set of $n - l + 1$ l-mers is written as *Spectrum* (s, l). Now the problem can be formally defined as follows. Let S be a set of substrings representing all l-mers of a string s. We need to predict a string s such that $Spectrum(s, l) = S$. This problem can be modeled as a Hamiltonian path problem as follows. We say two l-mers p and q *overlap* if the last $l - 1$ letters of p coincide with the first $l - 1$ letters of q. Construct a directed graph H such that each vertex of H corresponds to an l-mer in S and there is a directed edge in H from p to q if p and q overlaps. Now a Hamiltonian path in H corresponds to a string s such that $Spectrum(s, l) = S$. Figure 10.16 illustrates the reconstruction of a sequence using Hamiltonian graph. Construction of a digraph H from $S = \{GCA, AGC, CAT, TTC, TCA, AGT, CAG, ATT\}$ is illustrated in Fig. 10.16(a) and reconstruction of a sequence AGCATTCAGT from a Hamiltonian path in H is illustrated in Fig. 10.16(b).

Although the problem of sequence reconstruction above is modeled as a Hamiltonian path problem, it does not lead to an efficient algorithm since Hamiltonian path problem is a hard problem. Fortunately this sequencing problem can be reduced to an Eulerian path problem, by representing every l-mer by an edge of a graph, which leads to a simple linear-time algorithm [89].

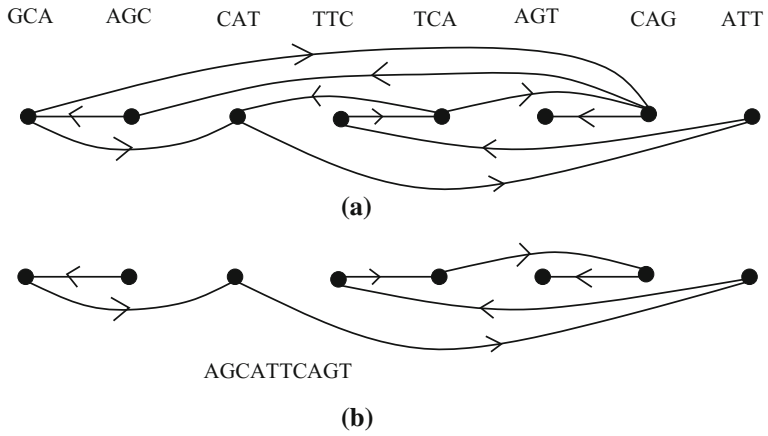

Fig. 10.16 Reconstruct a DNA sequence from its l-mers

10.6.2 Cliques for Protein Structure Analysis

In this subsection, we see how a problem in protein structure analysis can be modeled as a clique problem in a graph [90].

Comparison of protein structures is very important for understanding the functions of proteins because proteins with similar structures often have common functions. Pairwise comparison of proteins is usually done via protein structure alignment using some scoring scheme, where an alignment is a mapping of amino acids between two proteins. Because of its importance, many methods have been proposed for protein structure alignment. However, most existing methods are heuristic ones in which optimality of the solution is not guaranteed. Bahadur et al. developed a clique-based method for computing structure alignment under some local similarity measure [91]. Let $P = (p_1, p_2, \ldots, p_m)$ be a sequence of three-dimensional positions of amino acids (precisely, positions of C_α atoms) in a protein. Let $Q = (q_1, q_2, \ldots, q_n)$ be a sequence of positions of amino acids of another protein. For two points x and y, $|x - y|$ denotes the Euclidean distance between x and y. Let $f(x)$ be a function from the set of nonnegative reals to the set of reals no less than 1.0. We call a sequence of pairs $\mathrm{M} = ((p_{i_1}, q_{i_1}), \ldots, (p_{i_l}, q_{i_l}))$ an alignment under nonuniform distortion if the following conditions are satisfied:

(a) $i_k < i_h$ and $j_k < j_h$ hold for all $k < h$,
(b) $(\forall k)(\forall h \neq k)\left(\frac{1}{f(r)} < \frac{|q_{j_h} - q_{j_k}|}{|p_{j_h} - p_{j_k}|} < f(r)\right)$, where $r = min\{|q_{j_h} - q_{j_k}|, |p_{j_h} - p_{j_k}|\}$.

Then, a protein structure alignment is defined as the problem of finding a longest alignment (i.e., l is the maximum). It is known that protein structure alignment is NP-hard under this definition. This protein structure alignment problem can be reduced to the maximum clique problem in a simple way. We construct an undirected graph $G = (V, E)$ by

$$V = \{(p_i, q_j) | i = 1, \ldots, m, j = 1, \ldots, n\},$$

$$E = \{\{(p_i, q_j), (p_k, q_h)\} | i < k, j < h, \frac{1}{f(r)} < \frac{|q_h - q_j|}{|p_k - p_i|} < f(r)\}$$

Then, it is straightforward to see that a maximum clique corresponds to a longest alignment.

Some other protein structure analysis problem such as the protein side-chain packing problem, protein threading, etc., can also be modeled by cliques. Developing efficient heuristics for protein structure analysis based on graph models is an interesting research area.

10.6.3 Pairwise Compitability Graphs

The analysis of ancestral relationships among the species is a well-studied area in computational biology since such ancestral relationships provide evolutionary and functional insights into biological systems. The evolutionary history of a set of species is usually represented by a tree named *phylogenetic tree*, where each leaf represents an existing species and the internal nodes represent hypothetical ancestors. The objective of phylogenetic analysis is to analyze all branching relationship in a tree and their respective branch length. So the edges of the tree is usually weighted in order to represent the branch length which is a sort of evolutionary distance among species. Since it is difficult to define and determine the criteria of an exact phylogenetic tree, researchers usually reconstruct or predict several such trees and compare their suitability to explain the evolutionary relationship among the species. In the phylogenetic tree reconstruction problem, given a set of species and their genetic similarities, it is desired to find a phylogenetic tree that can explain the evolutionary relationship of the given species. Addressing the phylogenetic tree reconstruction problem, Kearney et al. [92] in 2003 introduced the concept of "pairwise compatibility graph" and showed how to use it to model evolutionary relationships among a set of species.

Let T be an edge-weighted tree and let d_{min}, d_{max} be two nonnegative real numbers. The *pairwise compatibility graph (PCG)* of T is a graph G such that each vertex of G corresponds to a distinct leaf of T and two vertices are adjacent in G if and only if the weighted distance between their corresponding leaves in T is in the interval $[d_{min}, d_{max}]$. Similarly, a given graph G is a PCG if there exist suitable T, d_{min}, d_{max}, such that G is a PCG of T. Figure 10.17(a) illustrates an edge-weighted tree T, and Fig. 10.17(b) shows the corresponding PCG G, where $d_{min} = 2$ and $d_{max} = 3.5$. Figure 10.17(c) shows another edge-weighted tree T' such that G is a PCG of T' when $d_{min} = 1.5$ and $d_{max} = 2$.

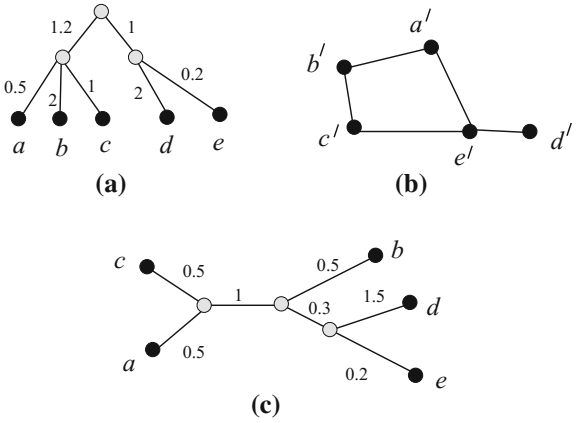

Fig. 10.17 **(a)** An edge-weighted tree T. **(b)** A PCG G of T, where $d_{min} = 2, d_{max} = 3.5$. **(c)** Another edge-weighted tree T' such that G is a PCG of T' when $d_{min} = 1.5, d_{max} = 2$

Kearney et al. hoped to show that every graph is a PCG, but later, Yanhaona et al. [93] constructed a 15-vertex graph that is not a PCG. Several researchers have attempted to characterize pairwise compatibility graphs. Yanhaona et al. [94] proved that graphs having cycles as its maximal biconnected components are PCGs. Calamoneri et al. [95] gave some sufficient conditions for a split matrogenic graph to be a PCG.

Finding a pairwise compatibility tree of a given graph appeared to be difficult, even for graphs with few vertices. In 2003, Kearney et al. [92] showed that every graph with at most five vertices is a PCG. Yanhaona et al. [93] constructed a 15-vertex graph which is not a PCG. This graph consists of a bipartite graph with partite sets A and B, where $|A| = 5$ and $|B| = 10$, and each subset of three vertices of A is adjacent to a distinct vertex of B. Calamoneri et al. [96] proved that every graph with at most seven vertices is a PCG. Recently Durocher et al. [97] have constructed a graph of eight vertices that is not a PCG. They also constructed a planar graph with 16 vertices that is not a PCG. This is the first planar graph known not to be a PCG. In the same paper, they showed that a variation of a generalized PCG recognition problem is NP-hard.

There are many open problems in the domain of PCGs, some are as follows:

- Find an example of an outer planar graph which is not PCG.
- It is not known whether wheels on at least eight nodes are PCGs or not.
- What is the complexity of PCG recognition problem?

Recently Calamoneri and Sinaimeri have published an extensive survey on PCG [98].

10.7 Graphs in Wireless Sensor Networks

In this section, we present some applications of graphs in a wireless sensor network, a recent communication network, from [88]. A wireless sensor network (WSN) consists of a number of sensor nodes spread over a geographic area. Each sensor node has wireless communication and signal processing capabilities. A conventional sensor node contains a controlling unit, a sensing unit, and a communication interface. The control unit is usually a microcontroller. The sensing unit senses a physical event and provides data representing the event to the controller. The communication interface transmits and receives data in the form of radio signals. Usually a wireless sensor network is an ad hoc since there is no fixed or static infrastructure. In a sensor network, some designated sensor nodes have more advanced communication capabilities which collects data from other sensors, possibly perform further processing of this data, and then transfer it to base station for advanced data processing. Such a designated sensor node is called a *sink* or a *fusion center* of the wireless sensor network.

In a conventional sensor network, data from simple sensors with short-range communication capabilities are transferred to the sink or fusion center for further processing as illustrated in Fig. 10.18, where sensor nodes send their sensed data to the nearest neighbor sensor node toward the sink over a spanning tree rooted at the sink. One sensor node may receive the same data from multiple neighbors. So a sensor node that receives data from neighbors needs to eliminate received redundant data before sending it to another neighbor. This operation is called *data aggregation.* For example, only the average temperature value received from a number of sensors in a particular region may be transmitted toward the root by an intermediate node to reduce the number of messages and to save energy. In doing such operations, WSN designers and implementers face the following challenges [88].

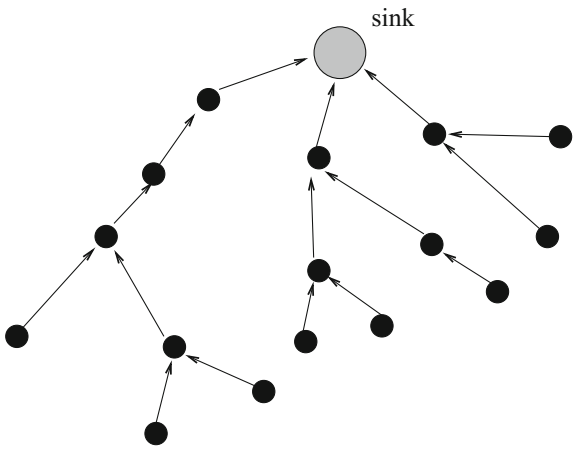

Fig. 10.18 A wireless sensor network

Scalability: A sensor network may consist of hundreds or thousands of sensor nodes. Protocols such as for routing and algorithms for these networks should be scalable to support operations over such a high number of sensor nodes.

Fault Tolerance: Failure of a single sensor node should not affect the overall operation of a sensor network. Hence connectivity of the network should be maintained using new links if required. New links should be selected in such a way that efficient routing can be established.

Node Deployment: Sensor nodes can be deployed randomly or regularly with the aim of providing maximum coverage of the area under sensing by the sensors. Furthermore, the deployment should provide a connected graph of the network for routing data to the sink.

Power Management: Sensor nodes of a sensor network are operated on batteries with limited lifetime. Replacing these batteries may be difficult or even impossible in many applications where sensors are deployed in harsh conditions. Since transmission and reception of messages consume much more energy than local computations, protocols and algorithms should be designed to work with minimum number of messages. Usually some of the sensor nodes are put into sleep mode to conserve energy when there are neither many activities to be sensed nor many messages under transmission. Then an important requirement is to provide full coverage of the area by the awake sensor nodes.

To deal with the challenges above, some graph models such as unit disk graph model, interference model, etc., have been developed. In the *unit disk graph model*, sensor nodes are represented by the vertices of the graph and there is an edge between two vertices u and v in the graph if the Euclidean distance between the corresponding sensor nodes is at within the transmission radius. In such a model identical transmission capabilities are assumed for each sensor node. However, in practice, transmission range and energy level may vary for the nodes in a network. In the *interference model*, edges are added considering the interferences of the sensor nodes while transmitting or receiving messages.

We will highlight some research problems on wireless sensor networks below which can be modeled as graph problems.

10.7.1 Topology Control

A sensor network may be densely deployed to provide redundancy for fault tolerance. Again, less energy consuming paths may be preferred instead of a direct and a more power consuming route to a destination. In such a case, a sparser and connected subgraph of the network graph may be used for communication. Such a subgraph is obtained by *topology control*. Topology control in wireless ad hoc networks restricts the allowed communication paths by obtaining a sparser subgraph of the network with less communication links and sometimes with less communicating nodes. Formally, given the network graph $G = (V, E)$, a topology control method obtains a subgraph

$G' = (V', E')$ such that V' and/or E' satisfy some desired criterion. For example, a chosen subset of the neighbors of a node are considered as connected neighbors, and the rest are discarded in G'. With the node failure for battery power outage, a subset of edges are activated dynamically to maintain a shortest spanning tree over the active sensor nodes. A minimal subset of active nodes that induces connected subgraph and capable of performing special tasks of forwarding data of other active nodes to a sink is called the *backbone* of the wireless sensor network. Usually the backbone of an ad hoc wireless sensor network is called a *virtual backbone*, since there is no rigid (wired) connection among the nodes on the backbone and the backbone changes dynamically with time. Backbone construction is an effective method for topology control that provides a significant reduction in the number of messages communicated in an ad hoc wireless network. When a backbone is constructed, a message from a source is first routed to the nearest neighbor backbone node, then along the backbone to the backbone node closest to the destination, and finally to the destination. Finding a path in a graph of a given length such that the shortest distance from each node to the path remains within a given bound is closely related to backbone construction problem. Finding minimum connecting domination sets (See Section 5.4) has also applications in construction of the backbone networks.

10.7.2 Fault Tolerance

Failure of a single node or up to a certain number of nodes should not affect the overall operation of a sensor network, and connectivity of the network should be maintained using new links among active nodes if required. Since activating more links is costly from power consumption point of view, it is desirable to increase connectivity by activating a smaller number of new nodes. This problem is related to connectivity augmenting problems of graphs which asks to increase connectivity by adding the minimum number of edges [99, 100]. There are other constraints also. For example, increase the connectivity by adding smaller number of edges so that the maximum degree of the graph remains small.

10.7.3 Clustering

Clustering is a method for partitioning nodes of a network into groups, which can provide a hierarchical architecture for efficient routing.

Given a graph $G = (V, E)$, clustering divides the vertex set V into a collection of subsets $\{V_1, V_2, \ldots, V_k\}$, where $V = \cup_{i=1}^{k} V_i$, and each $V_i \in V$ induces a connected subgraph G_i of G that may overlap. (Remember the Graph Partitioning problems in Section 10.5 which have applications in clustering.) An elected node in each cluster acts as the *cluster head* or the *leader of the cluster*. The cluster heads dominate other nodes within the same cluster and hence they are responsible for cluster management

functions and for providing hierarchical routing. The responsibility of the cluster head may be rotated among the members of the cluster to provide load balancing, fault tolerance and power management. A *gateway node* is a node that connects two clusters.

Graph partitioning as well as clustering is NP-hard in general. Hence various heuristics may be used to form clusters. Clustering algorithms determine the cluster heads using some static properties of the graph such as the degrees of the nodes, eccentricities of the nodes, etc. and build a cluster around them. The main advantages to be gained by clustering an ad hoc wireless network are as follows. A virtual backbone consisting of cluster heads and the gateway nodes provides an efficient method of hierarchical routing in ad hoc networks as it forms a simple topology to maintain. This method of hierarchical routing results in smaller routing tables, which are stored by the cluster heads and the gateway nodes only, and hence efficiency of routing is improved. Designing clustering (heuristics) algorithms for constructing clusters of desired properties for wireless sensor networks is a useful direction of research.

References

1. Spinrad, J.P.: Efficient Graph Representations. American Mathematical Society, Providence (2003)
2. Di Battista, G., Eades, P., Tamassia, R., Tollis, I.G.: Graph Drawing: Algorithms for the Visualization of Graphs. Prentice-Hall Inc., Upper Saddle River (1999)
3. Kaufmann, M., Wagner, D. (eds.): Drawing Graphs: Methods and Models. Lecture Notes in Computer Science, vol. 2025. Springer, Berlin (2001)
4. Sugiyama, K.: Graph Drawing and Applications for Software and Knowledge Engineers. World Scientific, Singapore (2002)
5. Nishizeki, T., Rahman, M.S.: Planar Graph Drawing. World Scientific, Hackensack (2004)
6. Tamassia, R. (Ed.): Handbook of Graph Drawing and Visualization, CRC Press (2014)
7. de Fraysseix, H., Pach, J., Pollack, R.: How to draw a planar graph on a grid. Combinatorica **10**, 41–51 (1990)
8. Schnyder, W.: Embedding planar graphs on the grid. In: Proceedings of First ACM-SIAM Symposium on Discrete Algorithms, San Francisco, pp. 138–148 (1990)
9. Chrobak, M., Nakano, S.: Minimum-width grid drawings of plane graphs. Comp. Geom. Theory Appl. **11**, 29–54 (1998)
10. Miura, K., Nakano, S., Nishizeki, T.: Grid drawings of 4-connected plane graphs. Discret. Comput. Geom. **26**(1), 73–87 (2001)
11. Garg, A., Rusu, A.: Straight-line drawings of binary trees with linear area and arbitrary aspect ration. Proceedings of Graph Drawing 2002. Lecture Notes in Computer Science, vol. 2528, pp. 320–331. Springer, Berlin (2002)
12. Garg, A., Rusu, A.: A more practical algorithm for drawing binary trees in linear area with arbitrary aspect ratio. In: Proceedings of Graph Drawing 2003, Lecture Notes in Computer Science, vol. 2912, pp. 159–165. Springer (2003)
13. Brandenburg, F., Eppstein, D., Goodrich, M.T., Kobourov, S., Liotta, G., Mutzel, P.: Selected open problems in graph drawing. In: Proceedings of Graph Drawing 2003, Lecture Notes in Computer Science, vol. 2912, pp. 515–539. Springer (2003)
14. Garg, A., Rusu, A.: Area-efficient drawings of outerplanar graphs. In: Proceedings of Graph Drawing 2003, Lecture Notes in Computer Science, vol. 2912, pp. 129–134. Springer (2003)

15. Di Battista, G., Frati, F.: Small area drawings of outerplanar graphs. In: Proceedings of Graph Drawing 2005, Lecture Notes in Computer Science, vol. 3843, pp. 89–100. Springer (2005)
16. Karim, M.R., Rahman, M.S.: On a class of planar graphs with straight-line grid drawings on linear area. J. Graph Algorithms Appl. **13**(2), 153–177 (2009)
17. Tutte, W.T.: Convex representations of graphs. Proc. London Math. Soc. **10**, 304–320 (1960)
18. Tutte, W.T.: How to draw a graph. Proc. of London Math. Soc. **13**, 743–768 (1963)
19. Chiba, N., Yamanouchi, T., Nishizeki, T.: Linear algorithms for convex drawings of planar graphs. In: Bondy, J.A., Murty, U.S.R. (eds.) Progress in Graph Theory, pp. 153–173. Academic Press, Canada (1984)
20. Chrobak, M., Kant, G.: Convex grid drawings of 3-connected planar graphs. Inter. J. Comput. Geom. Appl. **7**(3), 211–223 (1997)
21. Miura, K., Azuma, M., Nishizeki, T.: Convex drawings of plane graphs of minimum outer apices. Int. J. Found. Comput. Sci. **17**, 1115–1127 (2006)
22. Miura, K., Nakano, S., Nishizeki, T.: Convex grid drawings of four connected plane graphs. Int. J. Found. Comput. Sci. **17**, 1032–1060 (2006)
23. Zhou, X., Nishizeki, T.: Convex drawings of internally triconnected plane graphs on $o(n^2)$ grids. Discret. Math. Algorithms Appl. **2**, 347–362 (2010)
24. Lengauer, T.: Combinatorial Algorithms for Integrated Circuit Layout. Wiley, Chichester (1990)
25. Garg, A., Tamassia, R.: A new minimum cost flow algorithm with applications to graph drawing. In: Proceedings of Graph Drawing '96, Lecture Notes in Computer Science, vol. 1190, pp. 201–216. Springer (1997)
26. Rahman, M.S., Nakano, S., Nishizeki, T.: A linear algorithm for bend-optimal orthogonal drawings of triconnected cubic plane graphs. J. Graph Alg. Appl. **3**(4), 31–62 (1999). http://jgaa.info
27. Tamassia, R.: On embedding a graph in the grid with the minimum number of bends. SIAM J. Comput. **16**(3), 421–444 (1987)
28. Rahman, M.S., Nishizeki, T., Naznin, M.: Orthogonal drawings of plane graphs without bends. J. Graph Alg. Appl. **7**(4), 335–362 (2003). http://jgaa.info
29. Thomassen, C.: Plane Representations of Graphs. In: Bondy, J.A., Murty, U.S.R. (eds.) Progress in Graph Theory, pp. 43–69. Academic Press, Canada (1984)
30. Bhasker, J., Sahni, S.: A linear algorithm to find a rectangular dual of a planar triangulated graph. Algorithmica **3**, 247–278 (1988)
31. Kant, G., He, X.: Regular edge labeling of 4-connected plane graphs and its applications in graph drawing problems. Theory Comput. Sci. **172**, 175–193 (1997)
32. Rahman, M.S., Nakano, S., Nishizeki, T.: Rectangular grid drawings of plane graphs. Comp. Geom. Theory Appl. **10**(3), 203–220 (1998)
33. Biedl, T.C.: Optimal orthogonal drawings of triconnected plane graphs. In: Proceedings of SWAT'96, Lecture Notes in Computer Science, vol. 1097, pp. 333–344 (1996)
34. Biedl, T.C., Kaufmann, M.: Area-efficient static and incremental graph drawings. In: Proceedings of 5th European Symposium on Algorithms, Lecture Notes in Computer Science, vol. 1284, pp. 37–52. Springer (1997)
35. Papakostas, A., Tollis, I.G.: Efficient orthogonal drawings of high degree graphs. Algorithmica **26**, 100–125 (2000)
36. Rahman, M.S., Nakano, S., Nishizeki, T.: Box-rectangular drawings of plane graphs. J. Algorithms **37**, 363–398 (2000)
37. He, X.: A simple linear time algorithm for proper box rectangular drawings of plane graphs. J. Algorithms **40**(1), 82–101 (2001)
38. Rahman, M.S., Miura, K., Nishizeki, T.: Octagonal drawings of plane graphs with prescribed face areas. Comput. Geom. **42**(3), 214–230 (2009)
39. Alam, M.J., Kobourov, S.G., Mondal, D.: Orthogonal layout with optimal face complexity. In: Proceedings of SOFSEM 2016, Lecture Notes in Computer Science, vol. 9587, pp. 121–133 (2016)

40. Ikebe, Y., Perles, M.A., Tamura, A., Tokunaga, S.: The rooted tree embedding problem into points in the plane. Discret. Comput. Geom. **11**, 51–63 (1994)
41. Bose, P.: On embedding an outer-planar graph in a point set. Comput. Geom. Theory Appl. **23**(3), 303–312 (2002)
42. Cabello, S.: Planar embeddability of the vertices of a graph using a fixed point set is NP-hard. J. Graph Algorithms Appl. **10**(2), 353–363 (2006)
43. Garcia, A., Hurtado, F., Huemer, C., Tejel, J., Valtr, P.: On embedding triconnected cubic graphs on point sets. Electron. Notes Discret. Math. **29**, 531–538 (2007)
44. Nishat, R.I., Mondal, D., Rahman, M.S.: Point-set embed dings of plane 3-trees. Comput. Geom. Theory Appl. **45**(3), 88–98 (2012)
45. Durocher, S., Mondal, D.: On the hardness of point-set embeddability. In: Proceedings of WALCOM 2012, Lecture Notes in Computer Science, vol. 7157, pp. 148–159. Springer (2012)
46. Cardinal, J., Hoffmann, M., Kusters, V.: On universal point sets for planar graphs. In: Proceedings of Thailand-Japan Joint Conference on Computational Geometry and Graphs, LNCS **8296**, 30–41 (2013)
47. Dujmovic, V., Evans, W.S., Lazard, S., Lenhart, W., Liotta, G., Rappaport, D., Wismath, S.K.: On point-sets that support planar graphs. Comput. Geom. Theory Appl. **46**(1), 29–50 (2013)
48. Dujmovic, V., Evans, W.S., Kobourov, S.G., Liotta, G., Weibel, C., Wismath, S.K.: On graphs supported by line sets In: Proceedings of Graph Drawing 2010, LNCS 6502, pp. 177–182 (2011)
49. Hossain, M.I., Mondal, D., Rahman, M.S., Salma, S.A.: Universal line-sets for drawing planar 3-trees. J. Graph Algorithms Appl. **17**(2), 59–79 (2013)
50. Di Giacomo, E., Frati, F., Fulek, R., Grilli, L., Krug, M.: Orthogeodesic point-set embedding of trees. In: Proceedings of Graph Drawing 2011, LNCS 7034, pp. 52–63 (2012)
51. Bläsius, T., Kobourov, S.G., Rutter, I.: Simultaneous embedding of planar graphs. In: Tamassia, R. (ed.) Handbook of Graph Drawing and Visualization, pp. 349–381. CRC press (2014)
52. Garey, M., Johnson, D.: Crossing number is NP-complete. SIAM J. Algebr. Discret. Methods **4**(3), 312–316 (1983)
53. Huang, W., Hong, S.-H., Eades, P.: Using eye tracking to investigate graph layout effects. In: Proceedings of 6th Asia-Pacific Symposium on Visualisation 2007 (APVIS2007), pp. 97–100. IEEE (2007)
54. Huang, W., Hong, S.-H., Eades, P.: Effects of crossing angles. In: Proceedings of IEEE VGTC Pacific Visualization Symposium 2008 (PacificVis 2008), pp. 41–46. IEEE (2008)
55. Didimo, W., Eades, P., Liotta, G.: Drawing graphs with right angle crossings. Theor. Comput. Sci. **412**(39), 5156–5166 (2011)
56. Eades, P., Liotta, G.: Right angle crossing graphs and 1-planarity. Discret. Appl. Math. **161**(7–8), 961–969 (2013)
57. Argyriou, E.N., Bekos, M.A., Symvonis, A.: The straight-line RAC drawing problem is NP-Hard. J. Graph Algorithms Appl. **16**(2), 569–597 (2012)
58. Otten, J., Wijk, J.G.V.: Graph representation in interactive layout design. In: Proceedings of IEEE International Symposium On Circuits and Systems, pp. 914–918 (1978)
59. Tamassia, R., Tollis, I.G.: A unified approach to visibility representations of planar graphs. Discret. Comput. Geom. **1**, 321–341 (1986)
60. Dean, A.M., Evans, W., Gethner, E., Laison, J.D., Safari, M.: Bar k-visibility graphs. J. Graph Algorithms Appl. **11**(1), 45–59 (2007)
61. Sultana, S., Rahman, M.S., Roy, A., Tairin, S.: Bar 1-visibility drawings of 1-planar graphs. In: Proceedings of ICAA 2014, LNCS 8321, pp. 62–76. Springer (2014)
62. Brandenburg, F.J.: 1-visibility representations of 1-planar graphs. J. Graph Algorithms Appl. **18**(3), 421–438 (2014)
63. Gallian, J.A.: A dynamic survey of graph labeling. Electron. J. Comb. DS6 (2015)
64. Raspaud, A., Schroder, H., Sÿkora, O., Török, L.: Vrťo, I.: Antibandwidth and cyclic antibandwidth of meshes and hypercubes. Discret. Math. **309**(11), 3541–3552 (2009)

65. Rahaman, M.S., Eshan, T.A., Al Abdullah, S., Rahman, M.S.: Antibandwidth problem for itchy caterpillars. In: Proceeding of ICIEV 2014, IEEE Computer Society, pp. 1–6 (2014). doi:10.1109/ICIEV.2014.6850837
66. Yixun, L., Jinjiang, Y.: The dual bandwidth problem for graphs. J. Zhengzhou Univ. (Natural Science Edition) **35**, 1–5 (2003)
67. Leung, J.Y.-T., Vornberger, O., Witthoff, J.: On some variants of the bandwidth minimization problem. SIAM J. Comput. **13**(3), 650–667 (1984)
68. Hun, Y., Kobourov, S., Veeramon, S.: On maximum differential graph coloring. In: GD'10 Proceedings of the 18th International Conference on Graph Drawing, pp. 274–286 (2010)
69. Donnelly, S., Isaak, G.: Hamiltonian powers in threshold and arborescent comparability graphs. Discret. Math. **202**(1–3), 33–44 (1999)
70. Weili, Y., Ju, Z., Xiaoxu, L.: Dual bandwidth of some special trees. J. Zhengzhou Univ. (Natural Science Edition) **35**, 16–19 (2003)
71. Calamoneri, T., Massini, A., Török, L.: Vrťo, I.: Antibandwidth of complete k-ary trees. Discret. Math. **309**(22), 6408–6414 (2009)
72. Miller, Z., Pritikin, D.: On separation number of a graph. Networks **19**(6), 651–666 (1989)
73. Bekos, M.A., Kaufmann, M., Kobourov, S., Veeramoni, S.: A note on maximum differential coloring of planar graphs. J. Discret. Algorithms (2014). Published online
74. Wada, K., Kawaguchi, K.: Efficient algorithms for triconnected graphs and 3-edge-connected graphs. In: Proceedings of the 19th International Workshop on Graph Theoretic Concepts in Computer Science (WG'93), Lecture Notes in Computer Science, vol. 790, pp. 132–143. Springer (1994)
75. Wada, K., Takaki, A., Kawaguchi, K.: Efficient algorithms for a mixed k-partition problem of graphs without specifying bases. In: Proceedings of the 20th International Workshop on Graph Theoretic Concepts in Computer Science (WG'94), Lecture Notes in Computer Science, vol. 903, pp. 319–330. Springer (1995)
76. Dyer, M.E., Frieze, A.M.: On the complexity of partitioning graphs into connected subgraphs. Discret. Appl. Math. **10**, 139–153 (1985)
77. Györi, E.: On division of connected subgraphs. In: Proceedings of 5th Hungarian Combinational Coll., pp. 485–494 (1978)
78. Lovász, L.: A homology theory for spanning trees of a graph. Acta Math. Acad. Sci. Hunger **30**, 241–251 (1977)
79. Suzuki, H., Takahashi, N., Nishizeki, T., Miyano, H., Ueno, S.: An algorithm for tripartitioning 3-connected graphs. J. Inf. Process. Soc. Jpn. **31**(5), 584–592 (1990)
80. Suzuki, H., Takahashi, N., Nishizeki, T.: A linear algorithm for bipartition of biconnected graphs. Inform. Process. Lett. **33**(5), 227–232 (1990)
81. Jou, L., Suzuki, H., Nishizeki, T.: A linear algorithm for finding a nonseparating ear decomposition of triconnected planar graphs, Technical Report of Information Processing Society of Japan, AL40-3 (1994)
82. Nakano, S., Rahman, M.S., Nishizeki, T.: A linear time algorithm for four-partitioning four-connected planar graphs. Inform. Process. Lett. **62**, 315–322 (1997)
83. Nagai, S., Nakano, S.: A Linear-time algorithm for five-partitioning five-connected internally triangulated plane graphs. IEICE Trans. Fundam. **E84-A**(9), 2330–2337 (2001)
84. Karim, M.R., Nahiduzzaman, K.M., Rahman, M.S.: A linear-time algorithm for k-Partitioning Doughnut Graphs. Infocomp J. Comput. Sci. **8**(1), 8–13 (2009)
85. Awal, T., Rahman, M.S.: A linear algorithm for resource tripartitioning triconnected planar graphs. Infocomp J. Comput. Sci. **9**(2), 39–48 (2010)
86. Awal, T., Rahman, M.S.: A linear algorithm for resource four-partitioning four-connected planar graphs. AKCE Int. J. Graphs Comb. **9**(1), 11–20 (2012)
87. Furht, B. (ed.): Handbook of Social Network Technologies and Applications. Springer, Berlin (2010)
88. Erciyes, K.: Distributed Graph Algorithms for Computer Networks. Springer, London (2013)
89. Jones, N.C., Pevzner, P.: An Introduction to Bioinformatics Algorithms. The MIT Press (2004)

90. Tomita, E., Akutsu, T., Matsunaga, T.: Efficient algorithms for finding maximum and maximal cliques: effective tools for bioinformatics. In: Laskovski, A.N. (ed.) Biomedical Engineering, pp. 625–638. Communications and Software, Trends in Electronics (2011)
91. Bahadur, D.K.C., Akutsu, T., Tomita, E., Seki, T., Fujiyama, A.: Point matching under non-uniform distortions and protein side chain packing based on an efficient maximum clique algorithm. Genome Informatics **13**, 143–152 (2002)
92. Kearney, P.E., Munro, J.I., Phillips, D.: Efficient generation of uniform samples from phylogenetic trees. In: Proceedings of the 3rd International Workshop on Algorithms in Bioinformatics (WABI), LNCS 2812, pp. 177–189. Springer (2003)
93. Yanhaona, M.N., Bayzid, M.S., Rahman, M.S.: Discovering pairwise compatibility graphs. Discret. Math. Algorithms Appl. **2**(4), 607–624 (2010)
94. Yanhaona, M.N., Hossain, K.S.M.T., Rahman, M.S.: Pairwise compatibility graphs. J. Appl. Math. Comput. **30**, 479–503 (2009)
95. Calamoneri, T., Petreschi, R., Sinaimeri, B.: On relaxing the constraints in pairwise compatibility graphs. In: Proceedings of the 6th International Workshop on Algorithms and Computation (WALCOM), LNCS 7157, pp. 124–135. Springer (2012)
96. Calamoneri, T., Frascaria, D., Sinaimeri, B.: All graphs with at most seven vertices are pairwise compatibility graphs. Comput. J. (to appear) (2012). http://arxiv.org/abs/1202.4631
97. Durocher, S., Mondal, D., Rahman, M.S.: On graphs that are not PCGs, In: Proceedings of the 7th International Workshop on Algorithms and Computation (WALCOM), Lecture Notes in Computer Science, vol. 7748, pp. 310–321. Springer (2013)
98. Calamoneri, T., Sinaimeri, B.: Pairwise compatibility graphs: a survey. SIAM Rev. **58**(3), 445–460 (2016)
99. Kant, G.: Augmenting outerplanar graphs. J. Algorithms **21**, 1–25 (1996)
100. Abellanas, M., García, A., Hurtado, F., Tejel, J., Urrutia, J.: Augmenting the connectivity of geometric graphs. Comput. Geom. Theory Appl. **40**(3), 220–230 (2008)

Index

Symbols
1-planar drawing, 147
1-planar graph, 147
C-component, 81
C_n, 19
$C_o(G)$, 78
$E(G)$, 11
$G(C)$, 79
$G - F$, 15
$G - W$, 15
$G - e$, 15
$G - v$, 15
$G \backslash e$, 22
$G_1 \cap G_2$, 19
$G_1 \cup G_2$, 19
K_n, 17
$K_{m,n}$, 18
$M \triangle M'$, 64
N_n, 17
$P_G(k)$, 98
P_n, 18
$V(G)$, 11
W_n, 19
$\Delta(G)$, 13
$\Gamma(G)$, 82
$\alpha(G)$, 68
$\chi(G)$, 91
$\chi'(G)$, 94
$\delta(G)$, 14
$\gamma(G)$, 70
$\kappa(G)$, 39
$\kappa'(G)$, 39
$\overline{G}$, 21
ϕ, 107
$\tau(G)$, 51
f, 83
k-ary tree, 50
k-chromatic, 91
k-factor, 72
k-outerplanar, 114
k-tree, 124
m, 11, 83
n, 11, 83
u, v-walk, 31
2-outerplanar, 114

A
Acyclic coloring, 98
Adjacency list, 28, 135
Adjacency matrix, 26, 135
Adjacent, 2, 13
Alternating path, 63
Ancestor, 50, 117
Antibandwidth labeling, 148
Augmenting path, 64, 108

B
Backbone construction, 158
Bacteriophage experiment, 152
Bar 1-visibility drawing, 147
Bar k-visibility drawing, 147
Bar visibility drawing, 147
Base vertices, 150
Binary tree, 50
Bipartite graph, 17, 34
  complete, 18
Block, 42
Block-cutvertex tree, 42
Box-orthogonal drawing, 143
Box-rectangular drawing, 143
Bridge, 39

M.S. Rahman, *Basic Graph Theory*, Undergraduate Topics in Computer Science,
DOI 10.1007/978-3-319-49475-3

C
Canonical ordering, 115, 148
Capacity constraint, 107
Caterpillar, 60, 150
Cayley's theorem, 55
Center, 58
Chain, 99
Chord, 115, 121
Chordal graph, 121
Chromatic index, 94
Chromatic number, 91
Chromatic polynomial, 97
Clique, 94
Clique number, 94
Clique tree representation, 122
Clustering, 158
Color class, 92, 94
Complete graph, 17
Connected, 32
Connected component, 32
Connected graph, 51
Connectivity, 39
Conservation law, 107
Control flow graph, 8
Convex drawing, 139
Convex grid drawing, 140
Cost, 52
Critical graph, 94
Cubic graph, 14
Cut edge, 33, 48
Cut vertex, 39
Cycle, 14
  chord, 115
  independent, 79
  k-legged cycle, 79
  leg, 79
Cycle decomposition, 36
Cycle graph, 19

D
Data aggregation, 156
Degree, 13, 103
Degree sequence, 24
Degree set, 24
Degree-sum formula, 13
Descendant, 50, 117
Diameter, 58
Differential graph coloring, 148
Digraph, 11, 103
  connected, 105
  degree, 103
  Hamiltonian, 106
  indegree, 103
  isomorphic, 104
  outdegree, 103
  score sequence, 107
  strongly connected, 105
  tournament, 106
  walk, 105
Disconnecting set, 39
Division vertex, 99
DNA doublehelix, 152
DNA sequencing, 152
Dominating set, 69
Domination number, 69
Doughnut graph, 14, 139
Dual bandwidth, 148
Dual edge, 84
Dual graph, 84
  self-dual, 84
  weak dual, 84

E
Ear decomposition, 44
Ecentricity, 58
Edge
  contraction, 22
  inner, 111
  outer, 111
Edge coloring, 94
Edge connectivity, 39
Edge cut, 39
End-vertex
  of an edge, 1, 11
  of a path graph, 18
  of a walk, 31
Equivalent embedding, 80
Eulerian circuit, 35
Eulerian graph, 35
Eulerian trail, 34
Euler's formula, 82
Even cycle, 34
External face, 117

F
Face, 5
Face coloring, 97
Face-spanning subgraph, 5
Fault tolerance, 158
Floorplanning, 6
Flow network, 107
  capacity, 107
  consumption, 107

production, 107
Forest, 47
Frequency assignment, 3

G
Genealogical tree, 117
Graceful labeling, 59, 148
Graph, 1, 11
acyclic, 47
center, 58
complement, 20
connected, 32
connected component, 32
connectivity, 39
diameter, 58
ecentricity, 58
edge, 1
factor, 72
intersection, 19
isomorphism, 22
k-regular, 14
multigraph, 11
planar, 77
radius, 58
regular, 14
simple, 11
thickness, 85
union, 19
vertex, 1
Graph coloring, 91, 148
Graph drawing, 128, 137
Graph labeling, 148
Graph representation, 135
Graphic sequence, 24
Grid drawing, 87, 138

H
Hall's matching condition, 66
Hamiltonian cycle, 36
Hamiltonian graph, 36
Hamiltonian path, 36, 152
Handshaking lemma, 13
Hybridization, 152
Hypercube, 14

I
Incidence matrix, 26
Incident, 2, 13
Indegree, 103
Independent set, 17, 68, 122
Inner edge, 79, 111
Inner vertex, 79
Interference model, 157
Internal face, 117
Internal triangle, 112
Internal vertex, 47
Internally triangulated, 115
Interval graph, 124, 136, 150
Isomorphic, 104
Isomorphism, 22

K
Königsberg bridge, 1, 35
Kruskal's algorithm, 52
Kuratowski's theorem, 78

L
Labeled graph, 54
Leaf, 47
Leg, 79
Leg-vertex, 79
Lobster, 60, 150
Loop, 11

M
Map coloring, 2, 97
Marriage problem, 66
Matching, 63
alternating path, 63
augmenting path, 64
maximal, 63
maximum, 63
perfect, 63
symmetric difference, 64
Maximal clique, 122
Maximal independent set, 68
Maximal matching, 63
Maximal outerplanar, 112
Maximal planar graph, 83
Maximum degree, 13
Maximum flow, 108
Maximum independent set, 68
Maximum matching, 63
Millar-Pritikin labeling scheme, 149
Min-cut, 108
Minimum degree, 14
Minimum spanning tree, 52
Minimum vertex cover, 68
Modules, 6
Multidigraph, 103
Multiple edge, 11

N
Neighbor, 2, 13
Nested triangle, 120
Node, 11
Nonplanar graphs, 146
Null graph, 14, 17

O
Odd cycle, 34
Optimal representation, 135
Ordered rooted tree, 50
Orientation, 104
Orthogonal drawing, 141
Orthogonal grid drawing, 143
Outdegree, 103
Outer boundary, 78
Outer cycle, 115
Outer edge, 79, 111, 115
Outer face, 78
Outer vertex, 79, 115
Outerplanar, 111
Outerplane, 111

P
Pairwise compitability graph (PCG), 154
Partial k-tree, 124
Partite set, 17
Path decomposition, 131
Path graph, 18
Pathwidth, 131
Perfect elimination scheme, 122
Perfect graph, 94
Perfect matching, 63
Pertinent graph, 127
Petersen graph, 14
Phylogenetic tree, 154
Planar graph, 5, 77
  straight-line drawing, 85
Plane 3-tree, 119
Plane graph, 5, 78
Plane triangulation, 119
Point-set embedding, 144
Prüffer's code, 55
Prim's algorithm, 52
Protein structure analysis, 153

R
RAC drawing, 146
Radius, 58
Rectangular drawing, 6, 142
Representative tree, 120
Representative vertex, 120
Ringel-Kotzig conjecture, 60
RNA secondary structure, 7

S
Saturated vertex, 63
Score sequence, 107
Self-dual, 84
Separable, 41
Separating set, 39
Separating triangle, 116
Separation number, 148
Separation-pair, 39
Separator, 122
Series-parallel graphs, 125
Simultaneous embedding, 145
Sink, 103
Skeleton, 127
Software engineering, 8
Software testing, 8
Source, 103
Spanning subgraph, 51
Spanning tree, 51
Split component, 126
Split pair, 125
SPQ-tree, 125
Straight-line drawing, 85, 138
Strongly connected, 105
Subdivision
  of an edge, 21
Subgraph, 15
  edge-induced, 16
  vertex-induced, 15
Symmetric difference, 64

T
Taxonomic tree, 7
Thickness, 85
Topology control, 157
Tournament, 106
Trail, 31
Tree, 47
  ancestor, 50
  center, 58
  child, 50
  descendant, 50
  internal vertex, 50
  leaf, 47, 50
  parent, 50
  rooted, 50
Tree decomposition, 130

Treewidth, 130
Triangulated plane graph, 83

U
Underlying graph, 104
Unique embedding, 80
Unit disk graph model, 157
Unlabeled graph, 54
Unsaturated, 63

V
Vertex coloring, 91
Vertex cover, 68
Vertex cut, 39
VLSI floorplanning, 6

W
Walk, 31, 105
Weak dual, 84, 111
Web community, 7
Weighted graph, 12
Wheel graph, 19
Wireless sensor networks (WSN), 156