Preface

This book is written based on my classnotes 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 of 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 [Har72] is legendary while Wilson's book [Wil96] is an excellent introductory textbook of graph theory. The book by Douglas B. West [Wes96] is the most comprehensive book which covers both introductory and advanced topics of graph theory. Geir Agnarsson and Raymond Greenlaw in their book "Graph Theory: Modeling, Applications and Algorithms" [AG07] presented graph theory in a rigorous way using practical, intuitive and algorithmic approaches. This list also includes many other nice books [Deo74, Pir09]. Since I have followed those books in my classes, most of the contents of this book is 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 prefered 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 connectivities. 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. Chapter 9 and Chapter 10 exhibit the unique feature of this book; Chapter 9 presents some special classes of graphs, and some reasearch 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.

In revising research articles of my students I often face difficultes since they are not familiar with formal mathematical writing. I thus 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 Muham-

mad Jawaherul Alam started to compile those lectures. I continued it and prepared a complete manuscript during my sabbatical leave period 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 typoes and inconsistencies. My heartfelt thanks goes 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 Dhaka University.

I have many people to thank for helping and encouraging me to write this book. I would particularly like to express my gratitude to Mohammad Kaykobad for his continuous encouragement. I would like to thank my students Rahnuma, Debajyoti, Aftab, Rubaiyat and Iqbal. I thank Afzal Hussain of MIST for providing me a wonderful environment in MIST for completing this book.

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 Eva, son Anonno and daughter Shuprova.

Md. Saidur Rahman
Dhaka, 2016.

Contents

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 can be 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. 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 brances 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 beleived that the task was impossible, but there was no proof until 1736. In that year, one of the leading mathematician 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 four points and each of the seven bridges is represented by a curve or a line segment as illustrated in Fig. 1.1(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 matematical 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.

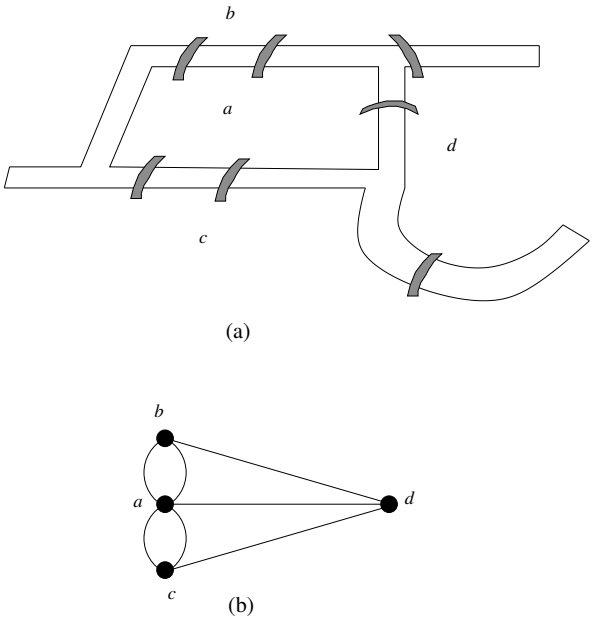


Fig. 1.1 The graph model for Königsberg bridges.

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 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 Figure 1.2(b) has six vertices a, b, c, d, e and f, and ten edges. Vertices a and b are 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. 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, 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 will be to use 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 if the two countries corresponding to the vertices share a boundary, as illustrated in Figure 1.2 where the Figure 1.2(b) illustrates the graph model for the map in Figure 1.2(b). Now the problem becomes a graph problem which asks to color the vertices of the graphs using the minimum number of colors so that two adjecent vertices get different colors. The vertices of the graph in Figure 1.2(b) are colored with four colors and hence the regions of the map in Figure 1.2(b) can be colored with four colors.

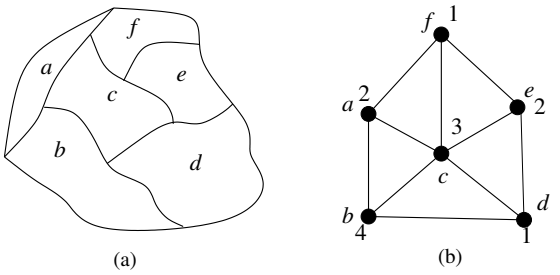


Fig. 1.2 (a) A map and (b) a graph model for map coloring.

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 $\{1, 2, \cdots, n\}$ 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 $1, 2, \cdots, n$ 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. Five transmitters a, b, c, d, e with their transmission range indicated by circles are shown in Figure 1.3(a) and the graph model is shown in Figure 1.3(b). In the graph model, there is an edge between a and b since their transmission range overlap, i.e., they are in the range of interference. Similarly other edges are added in the graph. Obseve that the frequencies are assigned to the vertices in such a way that

the minimum difference of two adjacent labels is 2.

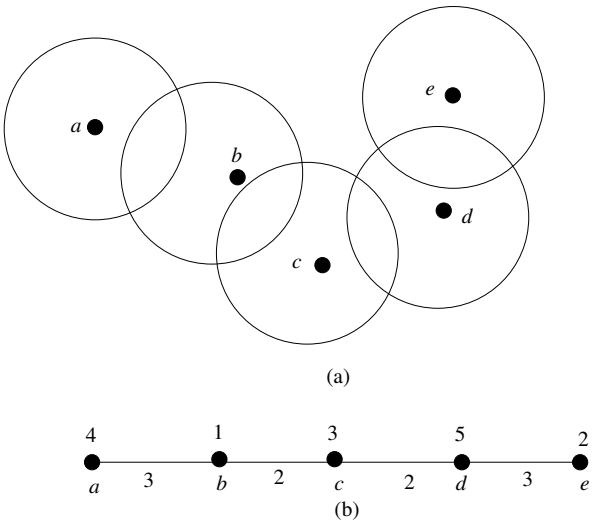


Fig. 1.3 (a) Transmitters with their transmission range, (b) a graph model with frequency assignment.

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 his building. The road network divides the locality into many regions as illustrated in Figure 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 gasfield which is located outside of the locality and a single pipe line will be used to supply gas from the gasfield to an intersection point on the outer boundary of the locality.

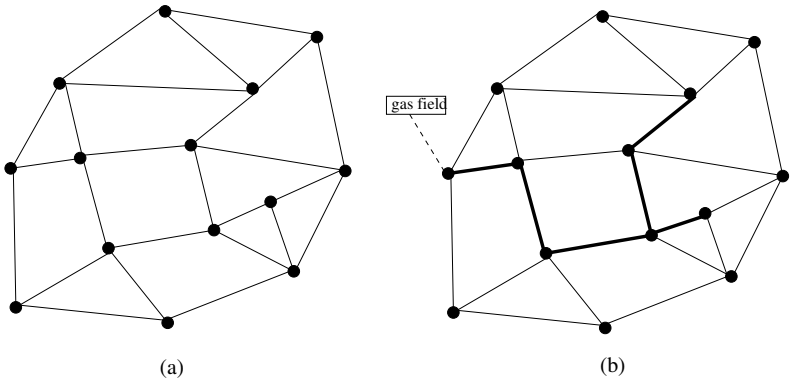


Fig. 1.4 (a) A locality and (b) a gas network.

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 minimum. Since gas will be supplied from the gasfield using a single line to the locality, the selected road network should be connected and contains 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 Figure 1.4(b).

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 set 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 [PR10]. Figure 1.5 shows two face-spanning subgraphs drawn by thick lines where the cost of the face-spanning subgraph in Figure 1.5(a) is 22 and the cost of the face-spanning subgraph in Figure 1.5(b) is 17. Thus a plane graph may

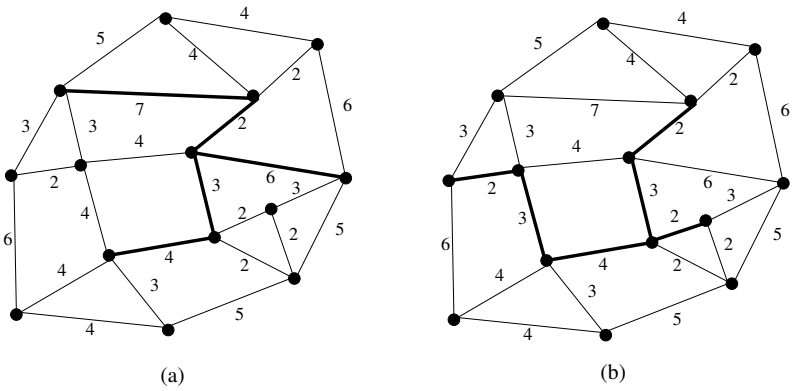


Fig. 1.5 A simple graph with (a) a face-spanning subgraph of cost 22 and (b) another face-spanning subgraph of cost 16.

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 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. 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 have applications in VLSI floorplanning as well as architectural floorplaning [NR04]. 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, that is, 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 bed room 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 of 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 obtain a "dual-like"

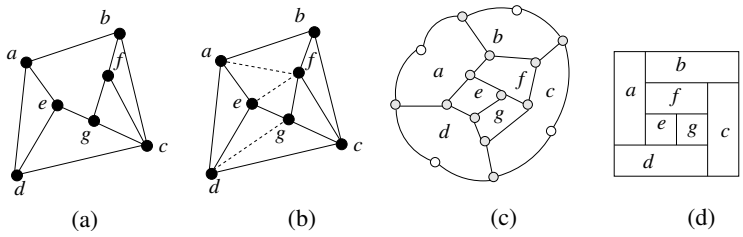


Fig. 1.6 (a) Graph F, (b) triangulated graph F', (c) dual-like graph G, and (d) rectangular drawing of G.

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. Examining web graphs it is possible to discover interesting information. For example, extracting dense subgraphs in a web graphs we can find a community of particular interest. In a graph representation of a web graph shown in Figure 1.7, two possible communites of particular interests are indicated by dotted circles.

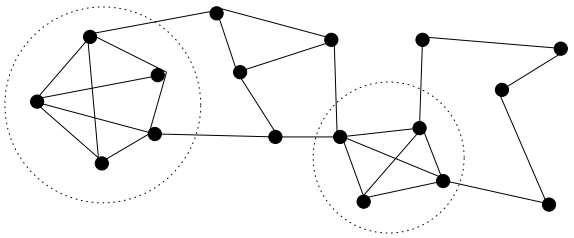


Fig. 1.7 Communities of common interest.

1.2.6 *Bioinformatics*

Graph theoretical modelings are used in multiple areas of bioinformatics including the description of taxonomic trees, phylogenetic analysis, the genome ontology and proteomics to name a few examples. The graph in Figure 1.8 illustrates a RNA secondary structure [Sun10].

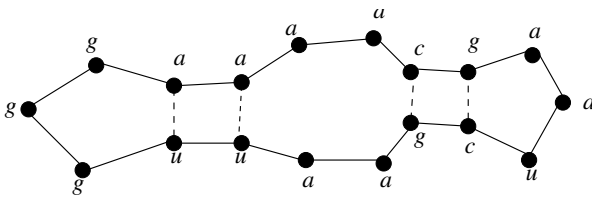


Fig. 1.8 RNA secondary structure.

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. Analyzing a control flow graph one can understand the complexity of the program and can design suitable test suit for software testing.

Exercise

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 alernate options of foods for each category of foods as follows. Rice: plane / yellow, Curry: fish / chicken, Nan: plain / butter, Kabab: 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 option 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. 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	Nun	Kabab	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 & 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 & D. Pro-

cessing units A and D requires 100 squre meter area each whereas B and C requires 200 squre meter each. Furthermore the following adjacency requirements must be satisfied: B, C & D should be adjacent to A; A & D should be adjacent to B; A & D should be adjacent to C; and A, B & 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.

Chapter 2

Basic Graph Terminologies

In this chapter, we give some definitions of basic graph theoretic terminologies.

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 $E(G)$.

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

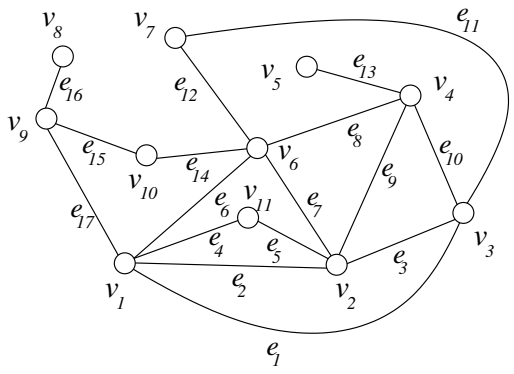


Fig. 2.1 A simple graph with eleven vertices and seventeen edges.

loops or multiple edges, 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 loops or multiple edges. 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.

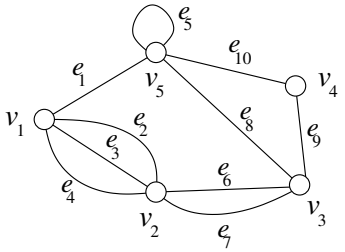


Fig. 2.2 A multigraph.

We call a graph a *directed graph* or *a digraph* if edge is associated with a direction, as illustrated in Figure 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 in both direction. We deal with digraphs in

Chapter 8. We call a graph an *weighted graph* if an weight is assigned to each vertex or each edge. Figure 2.3(b) illustrates an edge-weighted graph.

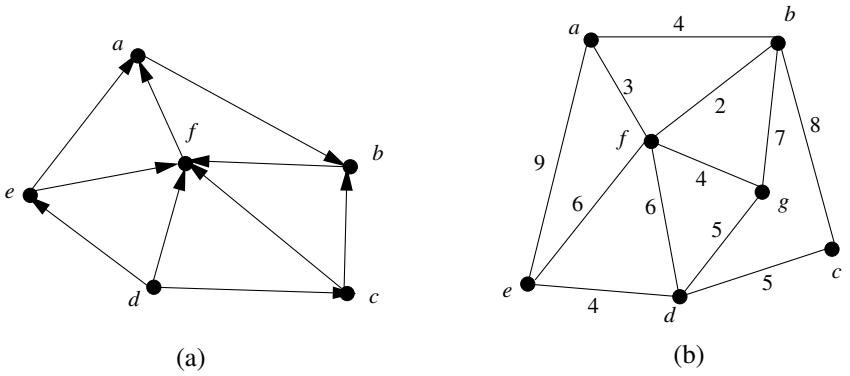


Fig. 2.3 (a) A directed graph and (b) an edge-weighted graph.

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 non-loop 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 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. Fig. 2.4 represents some regular graph.

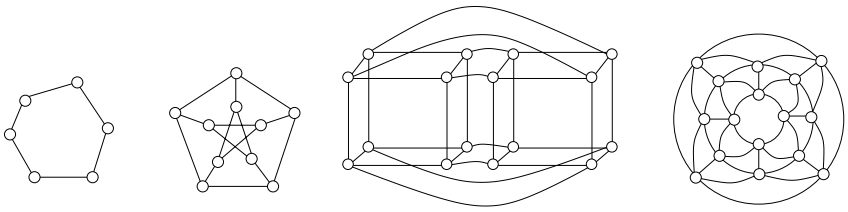


Fig. 2.4 (a) A 2-regular graph (a simple cyle), (b) a 3-regular graph (Petersen graph), (c) a 4-regular graph (4-dimensional hypercube), and (d) a 5-regular graph (a doughnut graph).

Only the graphs with empty edge sets (having non-empty 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 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 Figure 2.5. Petersen graph shows some interesting properties and also serves as a minimum-sized example and counter-example for many problems in graph theory. Doughnut graphs [KR09] 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 $d = 4$.

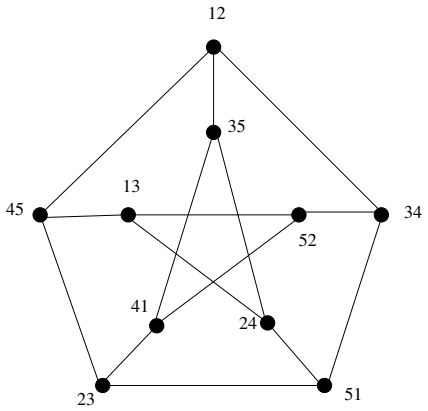


Fig. 2.5 (a) A labeled Petersen graph.

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 Fig. 2.6(b)–(e) are subgraphs of the graph in Fig. 2.6(a).

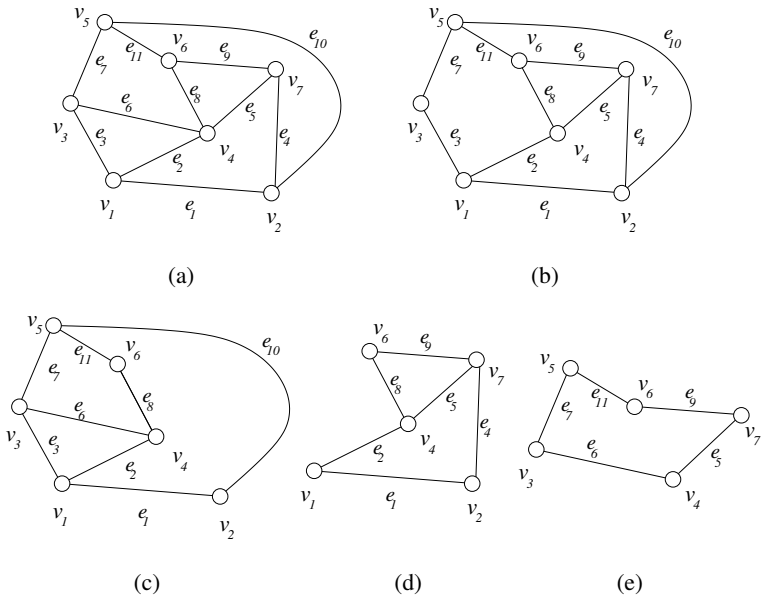


Fig. 2.6 (a) A Graph G, (b)–(e) subgraphs of G.

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' 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.

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.

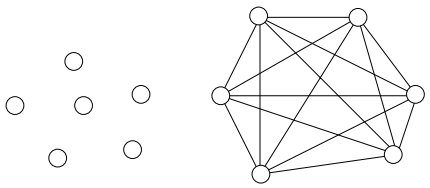


Fig. 2.7 (a) A null graph N_6 with six vertices, (b) a complete graph K_6 with six vertices.

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 non-empty 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 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.

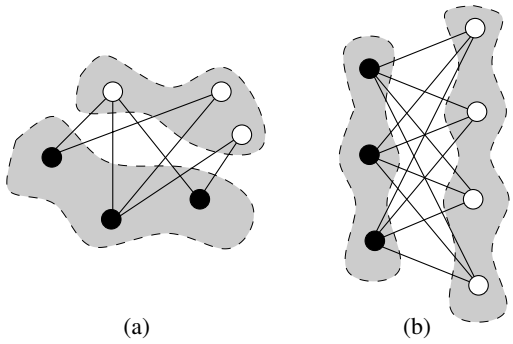


Fig. 2.8 Two bipartite graphs: the two independent partite sets are highlighted for each of them, one containing the black colored vertices and the other containing the white colored vertices.

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 graphs 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 contains 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, ..., 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.

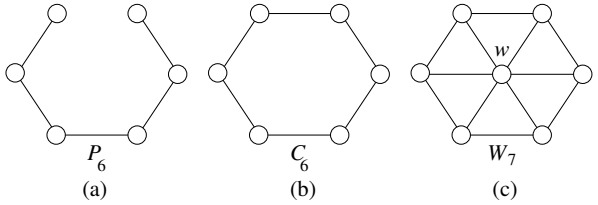


Fig. 2.9 (a) P_6, (b) C_6, and (c) W_7.

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 $E_4 = E_1 \cap E_2$.

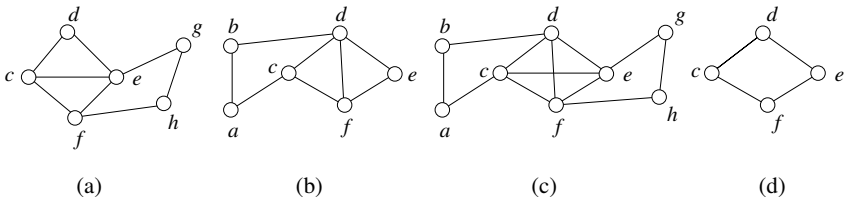


Fig. 2.10 (a) G_1 (b) G_2, (c) $G_1 \cup G_2$, and (d) $G_1 \cap G_2$.

Figure 2.10(a) and (b) shows two graphs G_1 and G_2, and Fig. 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.

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 hand-shakes. As usual, we transform the scenerio 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 hand-shakes. 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).

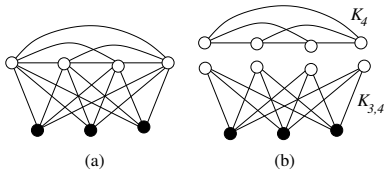


Fig. 2.11 (a) The graph representing hand-shakes, (b) K_4 and $K_{3,4}$.

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 hand-shakes 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, $v \in V$, $(u, v) \in \overline{E}$ 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.

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

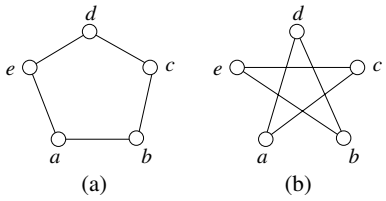


Fig. 2.12 (a) A graph G, and (b) the complement $\overline{G}$ of G.

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).

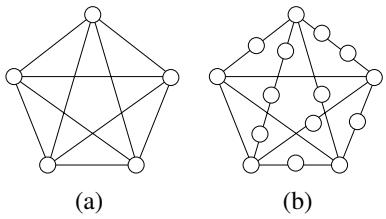


Fig. 2.13 (a) A graph G, and (b) a subdivision G' of G.

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.

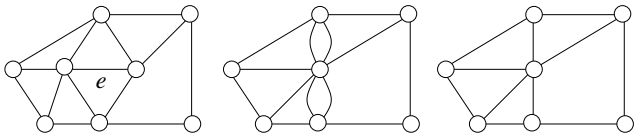


Fig. 2.14 (a) A graph G, (b) the graph obtained by contrating the edge e in G, and (c) the simple graph obtained by contracting the edge e in G.

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).

2.5.5 *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, repectively, then the two graphs become complement to each other. A graph which is isomorphic to 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.

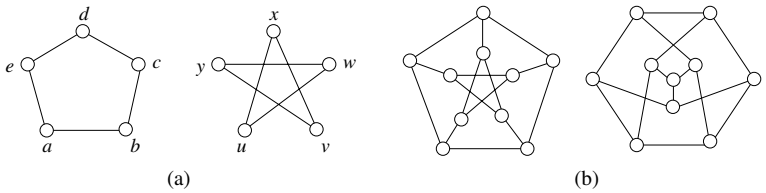


Fig. 2.15 Two pairs of isomorphic graphs.

In order to decide whether two graphs G_1 and G_2 are isomorphic or not, a tirivial 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 examin 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 iso-

morphism between graphs gives equivalence classes under the isomorphism relation.

Lemma 2.5.2 *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 $g : V_2 \longrightarrow V_3$ defines 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 $g \circ f$ defins 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 and we informally use the term "unlabeled graphs" to denote an isomorphic class of graphs.

2.6 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 non-negative integers occuring in a degree sequence of a graph is called its *degree set.* For the graph in Figure 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 Figures 2.16(a) and (b) are degree equivalent.

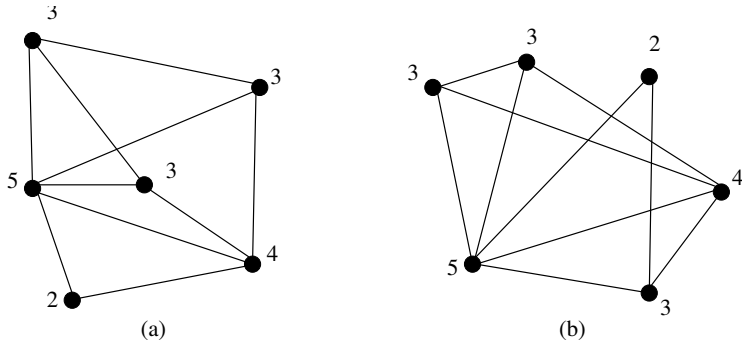


Fig. 2.16 Illustration for degree sequence.

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 fulfil these degree requirements of each vertex. The construction of a graph for the sequence $5, 4, 3, 2, 1, 1$ is illustrated in Figure 2.17.

Allowing loops makes the realization of a degree sequence easy. If we do not allow loops and muliple 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

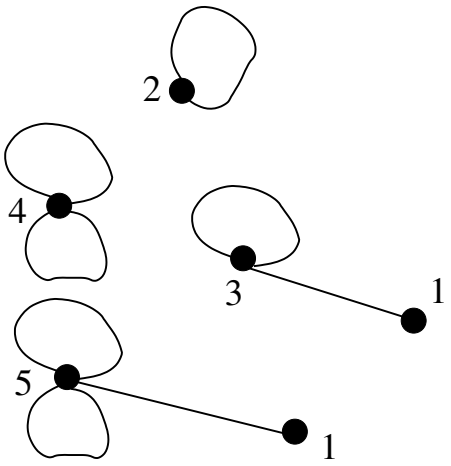


Fig. 2.17 Illustration of a construction for the sequence $5, 4, 3, 2, 1, 1$.

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 Figure 2.18(b), where the degree sequence of G' is $4, 4, 4, 4, 3, 1$. Now consider the following scenario. Assume that we have a degree sequence $d = d_1, d_2, \cdots, d_n$. We obtain a degree sequence d' from d by deleting d_1 and subtracting 1 from each of $d_2, \cdots, d_{d_1+1}$. 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 and Hakimi 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 Figure 2.19. Note that when we obtain d' from d, we rearrange d' in nonincreasing order if d' is not in nonincreasing order. For

example, observe $1, 0, 1 \to 1, 1, 0$ in Figure 2.19.

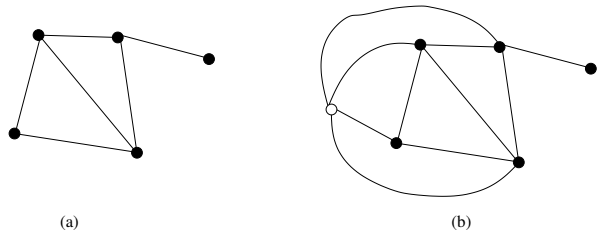


Fig. 2.18 (a) A graph G with the degree sequence $3, 3, 3, 2, 1$ and (b) a graph G' with the degree sequence $4, 4, 4, 4, 3, 1$.

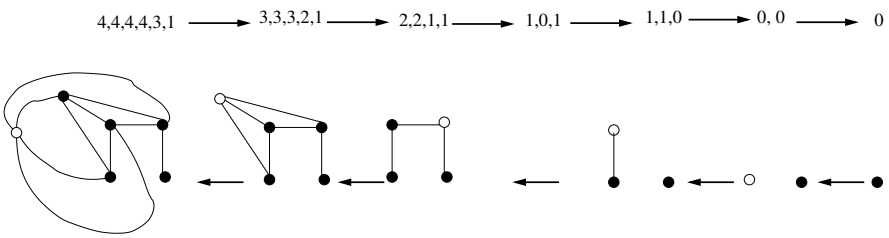


Fig. 2.19 Illustration for testing realizing a graphic sequence.

2.7 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.

2.7.1 *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.

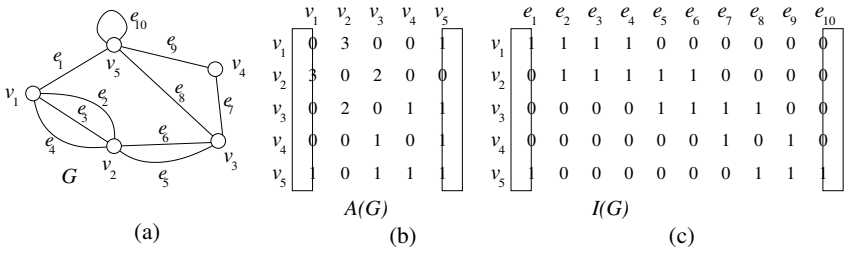


Fig. 2.20 (a) A graph G, (b) adjacency matrix representation of G, and (c) incidence matrix representation of G.

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. Fig. 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.

2.7.2 *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 responsed 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.

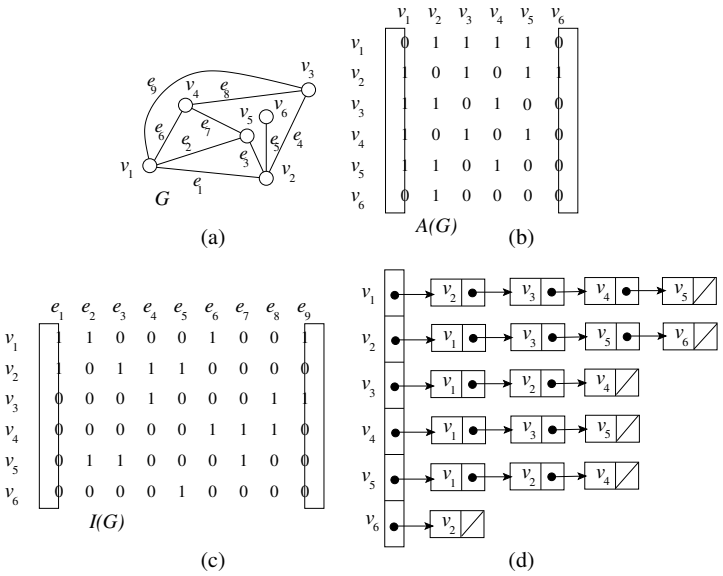


Fig. 2.21 (a) A simple graph G, (b) adjacency matrix representation of G, (c) incedence matrix representation of G, and (d) adjacency list representation G.

2.7.3 *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. Fig. 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.

Exercise

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.
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 isomprphic.

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 non-empty 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 can not appear more than once. In Figure 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 $W_2 =$

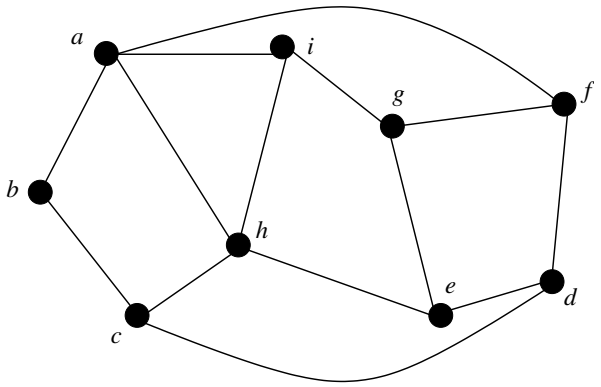


Fig. 3.1 Walks, trails and paths.

$a, (a, i), i, (i, h), h, (h, c), c, (c, b), b, (b, a), a, (a, i)$ 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 Figure 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 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 Figure 3.2(a) is connected since there is a path between every pair of vertices. The graph in Figure 3.2(b) is disconnected since there is no path between the vertices a and h. The graph in Figure 3.2(a) has one connected component whereas the graph in Figure 3.2(b) has two connected components. Sometimes we call a connected component simply a *component* if there is no confusion.

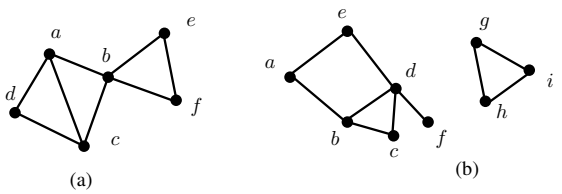


Fig. 3.2 (a) A connected graph and (b) a disconnected graph.

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

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 C_i and C_j with n_i and n_j vertices, respectively, such that $n_i \geq n_j > 1$. If we replace C_i and C_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 consists 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 Figure 3.2(b).

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

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 in 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 Figure 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. □

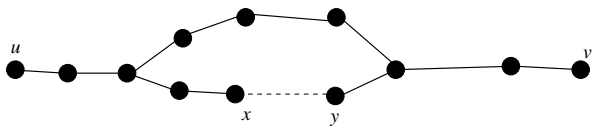


Fig. 3.3 Illustration for the proof of Lemma 3.1.4.

A cycle is *odd* if its length is odd. 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 $x_1, x_2, \cdots, x_l, x_1$ 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. 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 Figure 3.4(a) is Eulerian since it has a Eulerian circuit $e_1, e_2, e_3, \cdots, e_9$, whereas the graph in Figure 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 Figure 1.1. In fact Euler proved the impossibility by proving a general claim as in the following theorem.

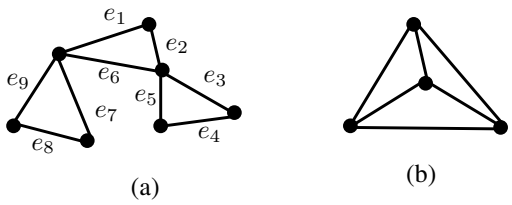


Fig. 3.4 (a) An Eulerian graph and (b) a graph which is not Eulerian.

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, $v_0, e_1, v_1, e_2, \cdots, e_n, v_0$. Both edges e_1 and e_n contribute 1 to the degree of v_0; so degree of v_0 is at least two. 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.

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

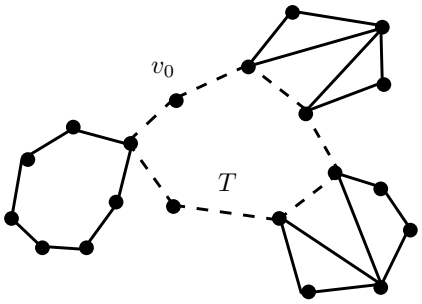


Fig. 3.5 Illustration for the proof of sufficiency of of Theorem 3.2.1.

the graph obtained from G by deleting all the edges of T. 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 Figure 3.6(a) is Hamiltonian, since a, b, c, d, a is a Hamiltonian cycle. On the other hand the graph in Figure 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 Figure 3.6(b). Thus a natural question is: What is the necessary and sufficient condition for a graph to be a Hamiltonian? Clearly a Hamiltonian graph must be connected and can not be acyclic, but these are not sufficient. The graph in Figure 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.

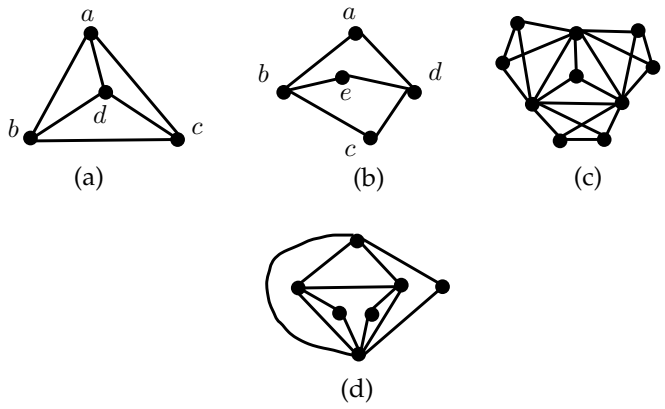


Fig. 3.6 (a) A Hamiltonian graph, and (b)-(d) non-Hamiltonian graphs.

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 Figure 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 Figure 3.6(c) is not Hamiltonian by Lemma 3.3.1. On the otherhand, the graph in Figure 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 [Ore60].

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 $C = (u_0 = u), (u_1 = v), u_2, \cdots, u_n = u$. We now define two sets as follows.

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

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

One can observe that each edge incident to u except (u, u_{n-1}) contribute an element in U and each edge incident to v except (v, u_2) contribute 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 $C' = (u_0 = u), u_j, u_{j-1}, \cdots, (u_1 = v), u_{j+1}, u_{j+2}, \cdots, u_{n-1}, u_n = u_0$ which does not contain the edge (u, v). Hence G is Hamiltonian. □

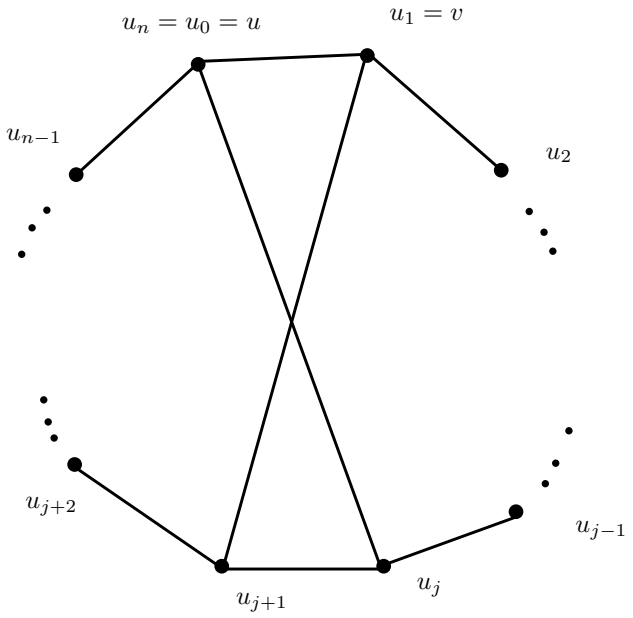


Fig. 3.7 Illustration for the proof of Lemma 3.3.2.

Using Lemma 3.3.2, we can prove the following result which is due to Ore [Ore60].

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 non-adjacent vertices u*

and v in G.

Proof. Let $G = G_0$. Let u and v be a pair of non-adjacent 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 simple connected 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 Figure 3.8, $\kappa(G) = 1$, $\kappa'(G) = 2$ and $\delta(G) = 3$. Whitney in 1932 showed that the following relationship holds [Whi32].

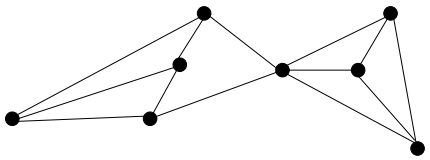


Fig. 3.8 A graph G with $\kappa(G) = 1$, $\kappa'(G) = 2$ and $\delta(G) = 3$.

Lemma 3.4.1 *Let G be a simple connected 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 $\kappa(G) \leq |P \cup Q| \leq |[S, \bar{S}]| = \kappa'(G)$. □

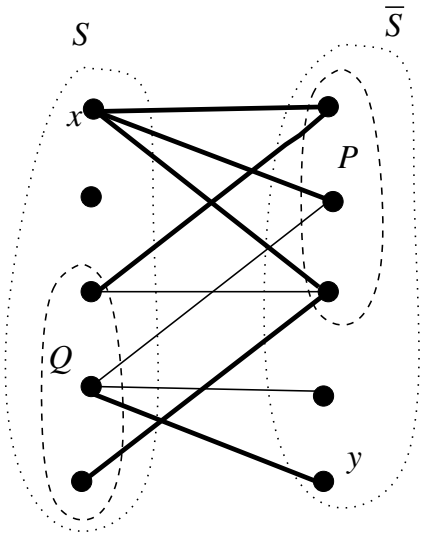


Fig. 3.9 Illustration for the proof of Lemma 3.4.1.

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.

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 Figure 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 neigbors 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 delete the edge from v to H_1 and the edge from u to H_1. Thus we construct an edge-cut of size $|S|$. □

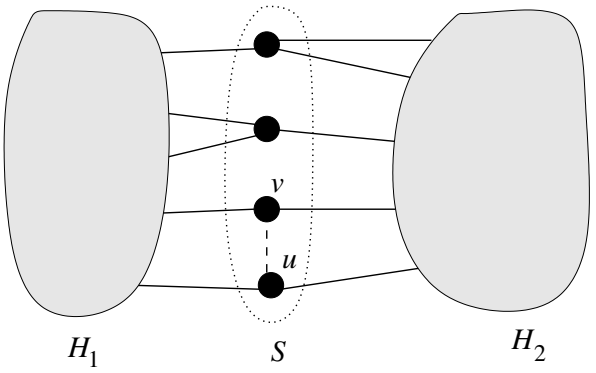


Fig. 3.10 Illustration for S, H_1 and H_2 in the proof of Lemma 3.4.2.

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 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 $B_1, B_2, \cdots, B_9$ 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.

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 workstation in a 2-connected networks, as in Theorem 3.4.5 below due to Whitney. 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.

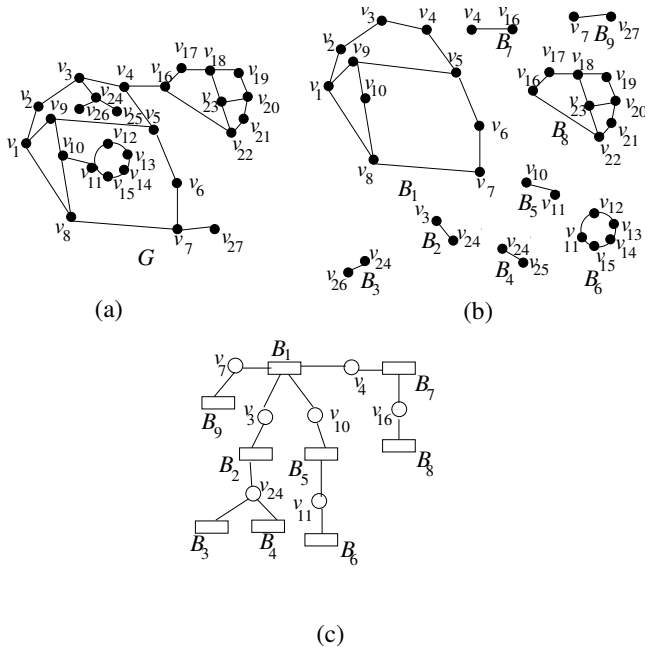


Fig. 3.11 (a)A connected graph G, (b) blocks of G, and (c) BC-tree T.

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. □

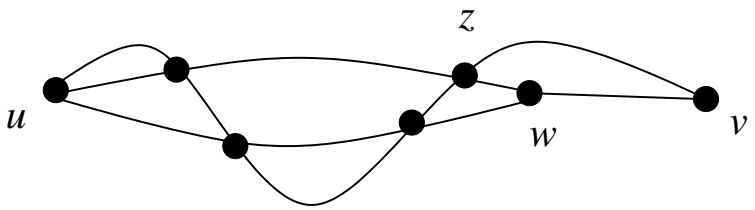


Fig. 3.12 Illustration for the proof of Theorem 3.4.5.

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

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. 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. Clearly G' is 2-connected, 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 decomposition $P_0, \cdots, P_k$ 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 $P_0, P_1, \cdots, P_7$ of a graph is illustrated in Fig. 3.13 where the ear P_6 is an edge. Whitney [Whi32] in 1932 gave a characterization

of a 2-connected graph in terms of ear decomposition as in the following theorem.

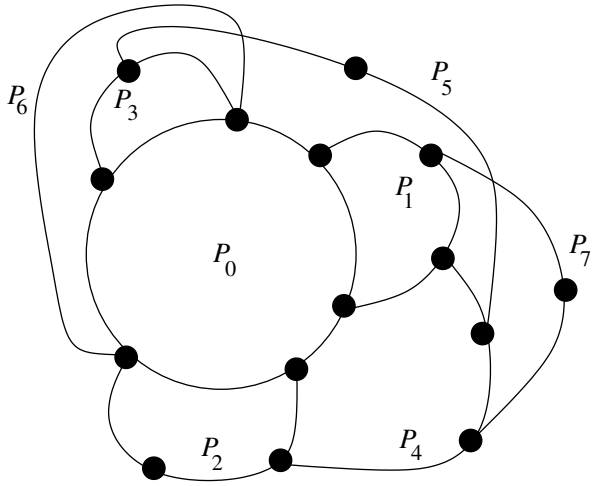


Fig. 3.13 Illustration for an ear decomposition a graph.

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 $P_0, P_1, \cdots P_k$. 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'. 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 $P_1, P_2, \cdots, P_i$. 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 [AG07, Wes96, Wil96] were used.

Exercise

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 [Nas71].
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.

Chapter 4

Trees

4.1 Introduction

A *tree* is a connected graph that contains no cycle. Figure 4.1(a)–(c) illustrates trees with one, two and three vertices respectively. Figure 4.1(d)–(e) illustrates two different trees with four vertices. A path is always a tree. The trees in Fig. 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 the definition of a tree. In this section, we discuss these properties of a tree. We first observe the following two trivial

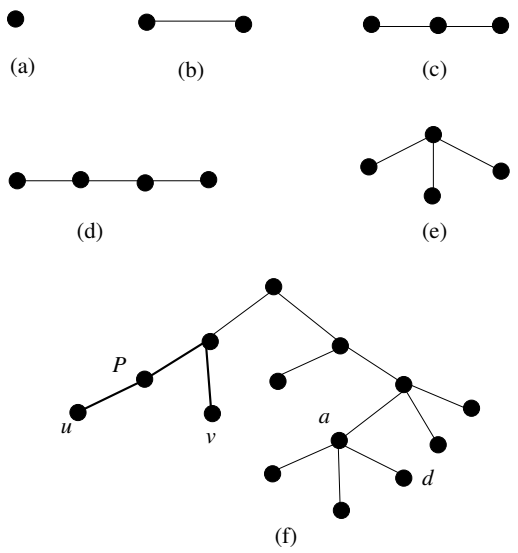


Fig. 4.1 Trees

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 Figure 4.1(f)), otherwise P would not be a maximal path in T. □

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)⇒ (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 contains 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)⇒ (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)⇒ (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 $G_1,\cdots,G_k$ 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 another 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 $P' = u', \cdots, x, v'$ 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.

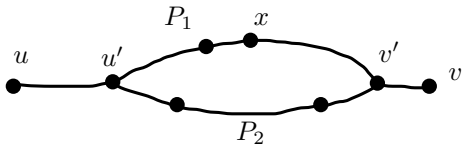


Fig. 4.2 An illustration for the proof of Lemma 4.2.5

.

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. □

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.

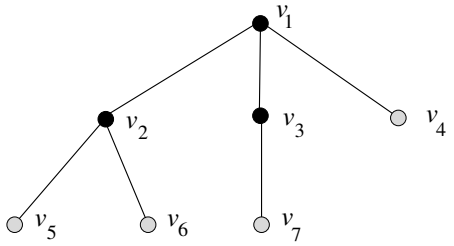


Fig. 4.3 A tree.

The parent-child relationship can be extended naturally to ancestors and descendants. Suppose that $u_1, u_2, \cdots, u_l$ 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.

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 Figure 4.4 the edges of a spanning tree of a graph is drawn by thick lines. The following claim holds.

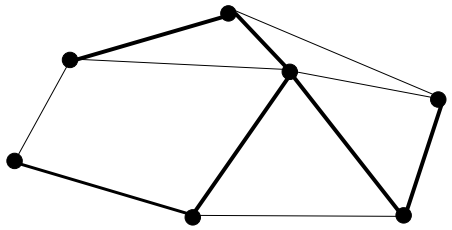


Fig. 4.4 Spanning tree of a graph; edges of a spanning tree are drawn by thick lines.

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 [NPP11]. 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 [NPP11].

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 *labeled graph* is a graph whose vertices are each assigned an element from a set of symbols by which the vertices can be distinguished one from another. A graph which has no such labeling is called an *unlabeled graph*. The two graphs in Figures 4.5(a) and (b) are labeled graphs and the graph in Figures 4.5(c) is an unlabeled graph. Note that the two graphs Figures 4.5(a) and (b) are different labeled graphs although they are the same unlabeled graph as in Figures 4.5(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 trees can be constructed. In this section we present Cayley's theorem 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 $(a_1, a_2, \cdots, a_{n-2})$, 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 $(x_1, x_2, \cdots, x_{n-2})$ from tree T as follows. Let y_1 be the smallest label

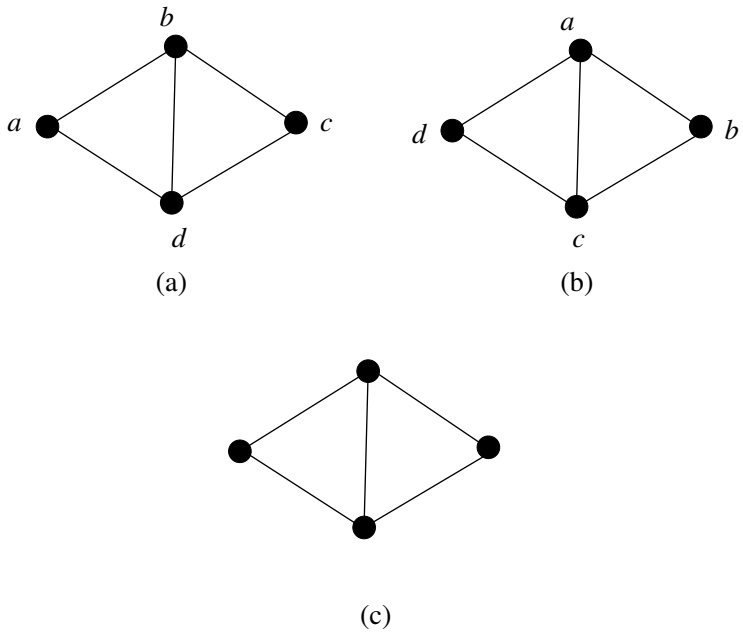


Fig. 4.5 (a)-(b) Labeled graphs and (c) an unlabeled graph.

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 $(x_1, x_2, \cdots, x_{n-2})$. An illustrative example for step by step construction of the sequence from a tree is given in Figure 4.6, where the edge (y_i, x_i) is drawn by dotted line in each step.

We now show that from a sequence $(x_1, x_2, \cdots, x_{n-2})$ we can construct a unique tree of n vertices. Let $\{1, 2, \cdots n\}$ be the set of n vertices. We will construct a tree with the vertex set $\{1, 2, \cdots n\}$. Let y_1 be the smallest number that does not appear in $(x_1, x_2, \cdots, x_{n-2})$. We add the edge (x_1, y_1), remove x_1 from the sequence and remove y_1 from consideration

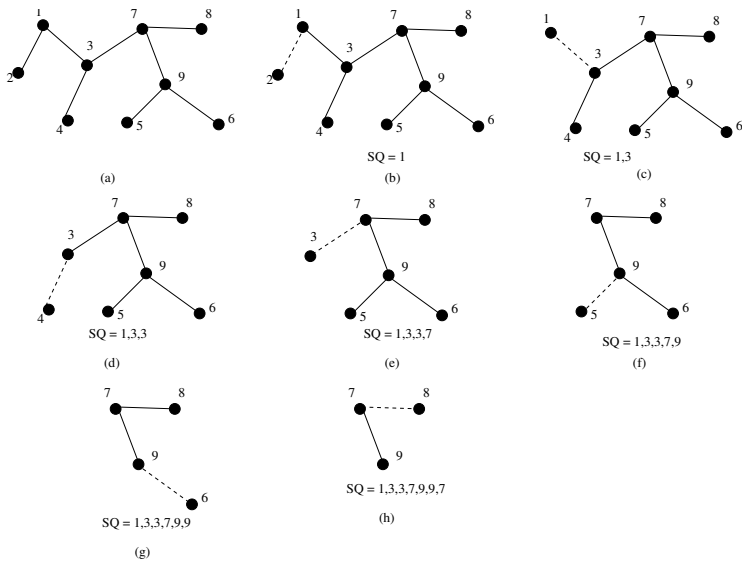


Fig. 4.6 Construction of the sequence from a tree.

and continue the process. Finally we add the last edge between two vertices whose label have not appeared in $\{y_1, y_2, \cdots, y_{n-2}\}$. We now need to show that the process mentioned above constructs a tree. We started with n isolated vertices labeled by $\{1, 2, \cdots n\}$. 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 observed 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 obtained a connected graph. Since we have added $n-1$ edges the connected graph is a tree. An illustrative example for construction of tree from a sequence $1, 3, 3, 7, 9, 9, 7$ is give in Figure 4.7. In

Figure 4.7(a) each vertex is a component and every vertex is unmarked. In Figure 4.7(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 Figure 4.7(b)-(i). □

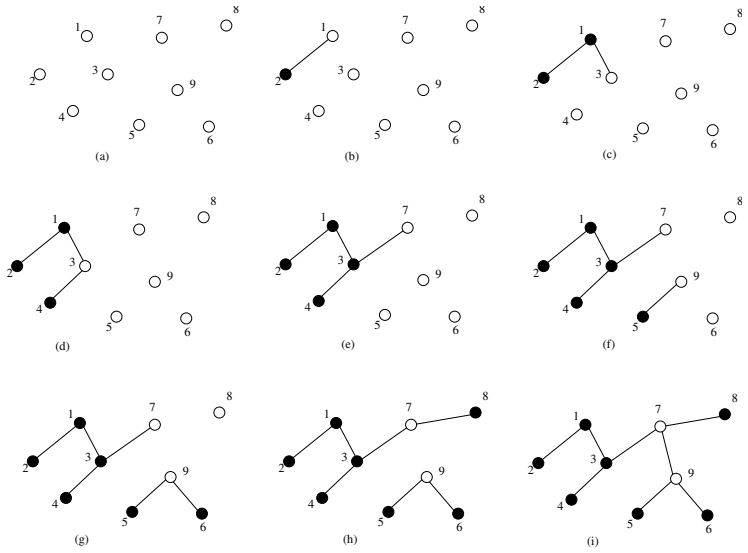


Fig. 4.7 Construction of the tree from a sequence.

The proof of Theorem 4.5.1 is due to Prüffer and Clarke and the sequence used in the proof is known as Prüffer's code. Some other proofs can be found in [Moo70].

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}.

4.6 Distances in Trees and Graphs

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 Figure 4.8 $d(a, j) = 5$ 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 Figure 4.8 has diameter 6. The *ecentricity* of a vertex u in G is $max_{v \in V(G)} d(u, v)$ and denoted by $\epsilon(u)$. Ecencrities of all vertices of the graph in Figure 4.8 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 ecentricities is equal to the diameter. The diameter and the radius of a disconnected graph are infinite.

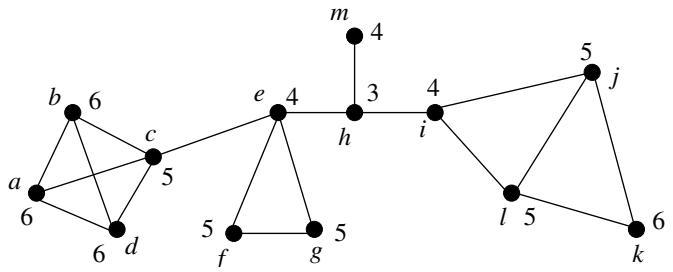


Fig. 4.8 Ecentricities of vertices.

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 neighbors in T. Hence the 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 Figure 4.9.

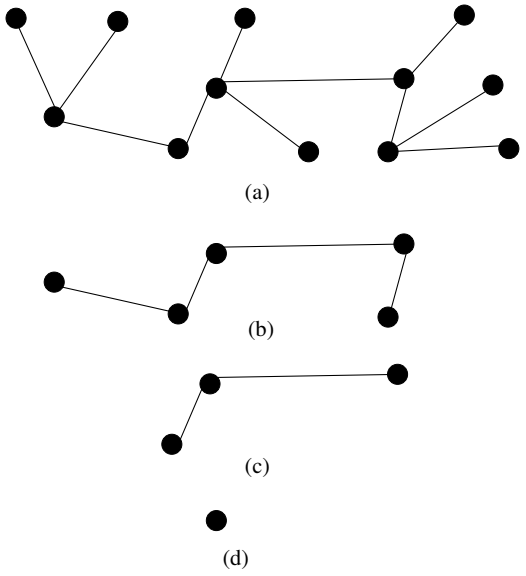


Fig. 4.9 Finding a center of a tree.

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 $\{0, 1, 2, \cdots, m\}$, so that distinct vertices receive distinct numbers and it satisfies $\{|f(u)-f(v)| : uv \in E(G)\} = \{1, 2, 3, \cdots, m\}$. The absolute difference of the labels of two end vertices of an edge e in a graceful labeling is regarded as the label of e in the graceful labeling.

One can easily compute a graceful labeling of a path as follows. Start labeling 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 4.10 illustrates a graceful labeling of a path. It is not difficult to observe that edges get labels $n-1, n-2, \cdots, 1$.

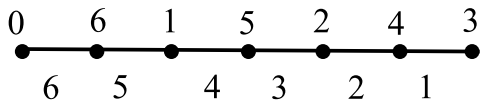


Fig. 4.10 Labels of vertices and edges in a graceful labeling of a path.

An interesting pattern of labeling exits for a graceful labeling of a caterpillars. Observe Figure 4.11 for the interesting pattern.

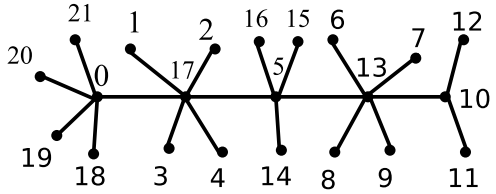


Fig. 4.11 Labels of vertices in a graceful labeling of a caterpillar.

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 of 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 [ARUR11].

Bibliographic Notes

For preparing this chapter several books [AG07, NR04, Wes96, Wil96] were used. Interested readers can find more in those books.

Exercise

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.
3. Show that a graph with n vertices, m edges and $n - m$ components is acyclic.
4. Draw all labeled trees with four vertices.
5. Construct all labeled spanning trees of K_4. How many of them are non-isomorphic?
6. Compute the ecentricities of the vertices in the graph in Figure 4.12.

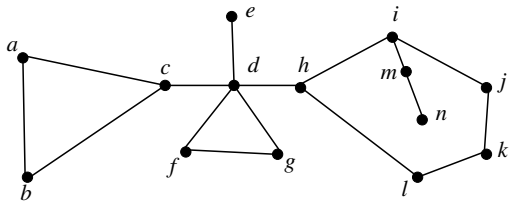


Fig. 4.12 A graph.

7. Find the centers of the trees in Figures 4.13(a) and (b).

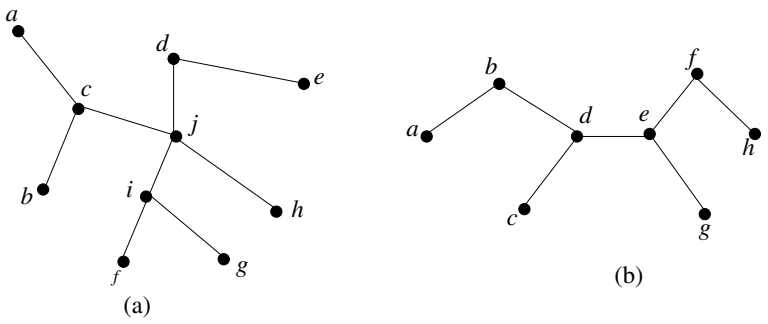


Fig. 4.13 Trees.

8. Show that the center of a tree is a single vertex if the diameter of the tree is even.
9. Compute the Prüffer's code for Figures 4.13(a) and (b).
10. Construct the tree corresponding to Prüffer's code 1,2,2,7,6,6,5.
11. Find a graceful labeling of the tree in Figure 4.13(b).
12. Develop an algorithm for computing a graceful labeling of a caterpillar.
13. Find a graceful labeling of the tree in Figure 4.14. Can you develop an algorithm for finding graceful labelings of this type of trees?

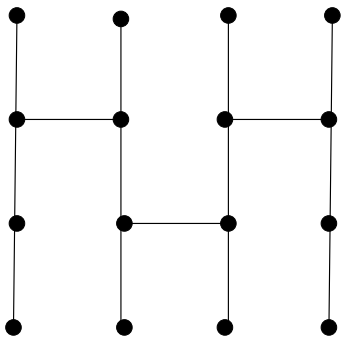


Fig. 4.14 A graceful tree.

14. Show that every tree containing a vertex of degree k contains at least k leaves.

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.

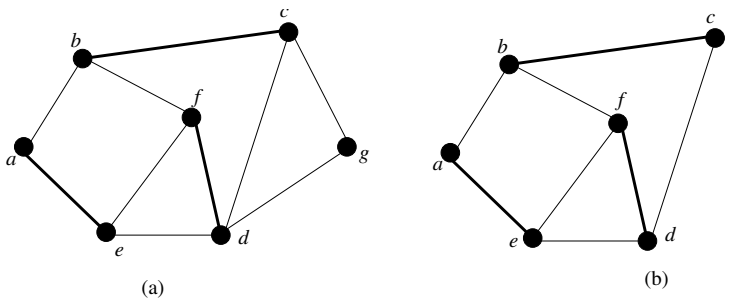


Fig. 5.1 Illustration of matchings: (a) a matching of a graph with an unsaturated vertex and (b) a perfect matching of a graph.

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.

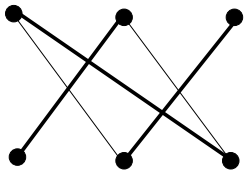


Fig. 5.2 A bipartite graph with unique perfect matching.

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 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).

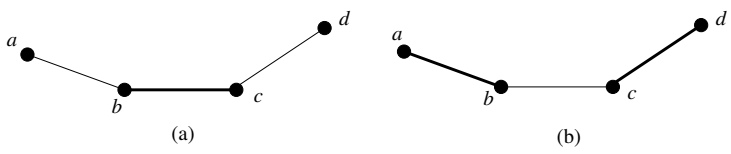


Fig. 5.3 Illustration of maximal and maximum matchings.

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 path* if the two endpoints of the path is 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 $M \triangle M' = \{(a, b), (b, d), (d, e), (a, e), (i, h), (g, h)\}$. 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. □

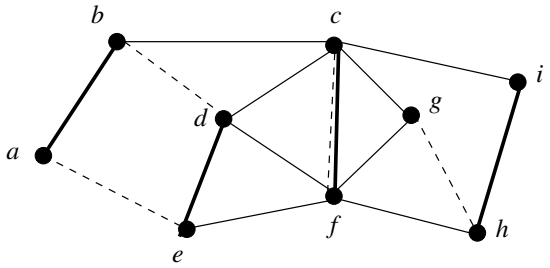


Fig. 5.4 Symmetric difference of two matchings.

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 [Ber57].

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 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 Figure 5.5, where job vacancies of five jobs are represented by the vertices $j_1, j_2, \cdots, j_5$, seven applicants are represented by the vertices $a_1, a_2, \cdots, a_7$ and an edge between a job and an applicant represents that the corresponding applicant has applied for the corresponding job. In the example in Figure 5.5, it is impossible to fill out every vacancy since the graph has no matching where all of vertices $j_1, j_2, \cdots, j_5$ are saturated. However, the graph in Figure 5.6 has a matching drawn by thick edges which saturates all vertices $j_1, j_2, \cdots, j_5$, and hence each vacancies can be filled out. Thus it is interesting to know in which case such a matching exists.

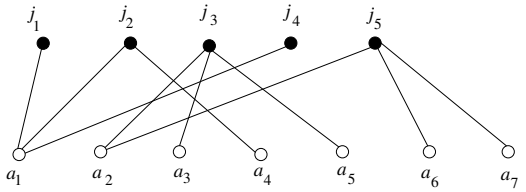


Fig. 5.5 Modeling of job vacancies and applicants.

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

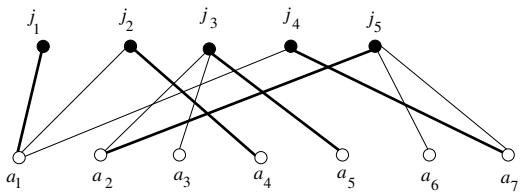


Fig. 5.6 A matching indicating a feasible solution for job vacancy fill out.

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 $|N(S)| \geq |S|$ for every $S \subseteq X$ is a necessary condition. Hall proved that this obvious necessary condition is also sufficient as in the following theorem [Hal35].

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 Figure 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 the 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.

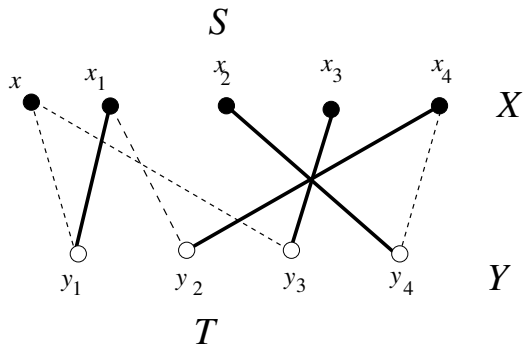


Fig. 5.7 Illustration for the proof of Theorem 5.1.5; $S = \{x, x_1, x_2, x_3, x_4\}$ and $T = \{y_1, y_2, y_3, y_4\}$.

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 Figure 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.

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. Figure 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 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 Figure 5.8(a) is a maximal independent set whereas the independent set shown in Figure 5.8(b) is a maximum independent set. The *independence number* of G, denoted by $\alpha(G)$, the number of vertices in a maximum independent set. For the graph in Figure 5.8, $\alpha(G) = 5$.

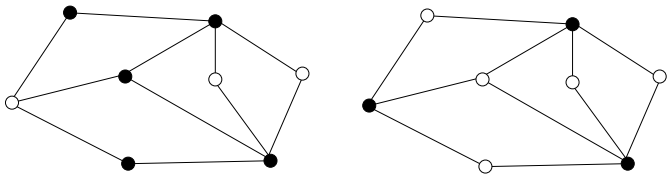


Fig. 5.8 Illustration for independent sets.

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 checkposts 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.

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.*

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 vetices belongs to $V - S$, and hence $V - S$ is not a vertex cover, a contradiction. □

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 Figure 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 Figure 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 locality will have a fire station.

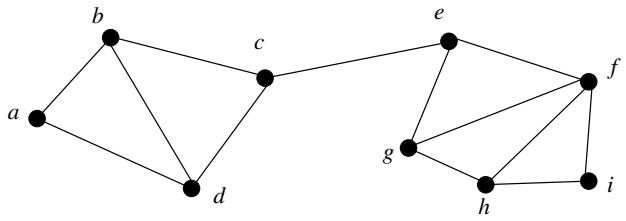


Fig. 5.9 illustration for domination set.

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 [HHS98].

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* $\mathrm{diam}(G)$ *be the diameter of G. Then*

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

Proof. Let x and y be two vertices of G such that $d(x,y) = \mathrm{diam}(G) = k$, and let $P = u_0(= x), u_1, \cdots, u_k(= y)$ 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 dominates 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 $\mathrm{diam}(G)$. 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 [AG07] 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 Figure 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 $H_1, H_2, \cdots, H_k$. Then $\gamma(G) = \sum_{i=1}^{k} \gamma(H_i)$.*

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

We now compute the domination number of the graph in Figure 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 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

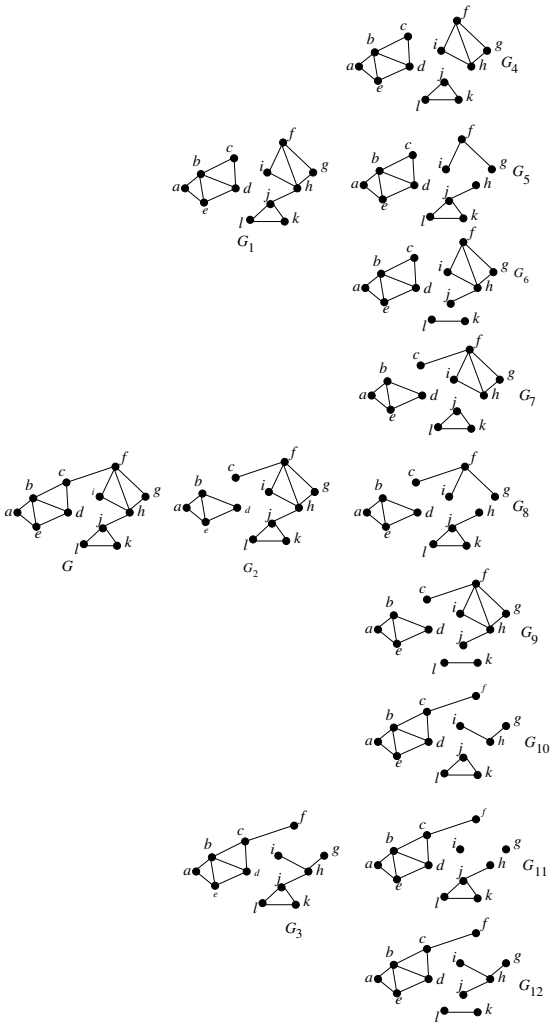


Fig. 5.10 Computation of domination number.

$\gamma(G) = min\{3, 3, 4\} = 3$.

Of course, the algorithm is not efficient. In fact the decision version of the problem is known to be NP-Complete 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 [DW13, YWWY13]. A dominating set D of a graph G is a connected dominating set if the vertices in D induces 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 present Tutte's condition [Tut47] for 1-factor. A connected component H of 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 [Tut47].

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 Figure 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 Figure 5.11(b) satisfies the condition in Theorem 5.5.1 and it has a 1-factor as shown by thick edges.

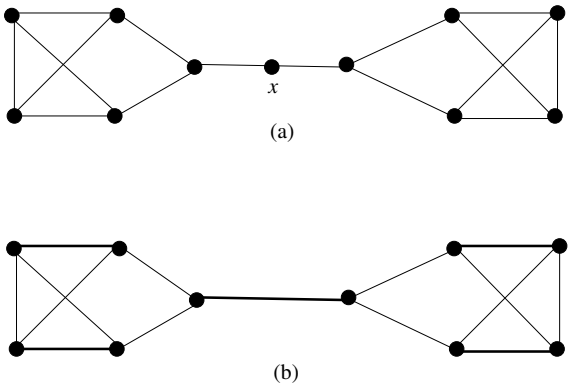


Fig. 5.11 (a) A graph violating the condition in Theorem 5.5.1 and (b) a graph satisfying the condition in Theorem 5.5.1

.

We now see an application of Theorem 5.5.1 in the proof of Theorem 5.5.2 (due to Chungphaisan [BM76]) which gives a necessary and sufficient condition for a tree to have a 1-factor. The proof presented here is due to Amahashi [AK11, Ama85].

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 Figure 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 $v \in V(T)$. 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 Figure 5.12(b).) Therefore e is an edge of F, and

thus F is a 1-factor of G. □

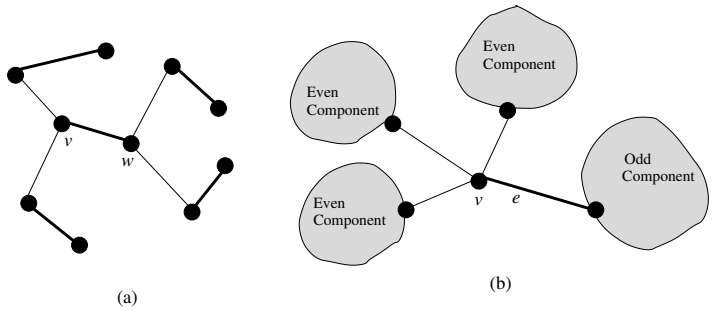


Fig. 5.12 Illustration for the proof of Theorem 5.5.2

.

Bibliographic Notes

The books [AK11, AG07, Wes96] were followed for preparing this chapter. Interested readers can find the book [AK11] very useful for studying matchings, covers and factors.

Exercise

1. Find a maximum matching in each of the graphs in Figure 5.13 and identify whether it has a perfect matching or not.

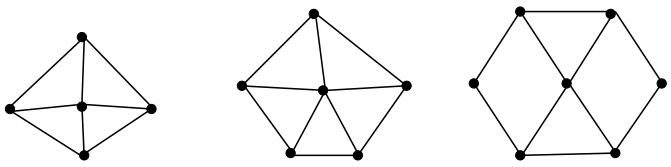


Fig. 5.13 Graphs.

2. Compute the number of perfect matchings in K_n.
3. Write a formal proof of Theorem 5.1.4.
4. Show that every k-regular bipartite graph has a perfect matching.
5. Using Theorem 5.1.5 show that the graph in Figure 5.14 has no matching where all vertices $j_1, j_2, \cdots, j_5$ are saturated.

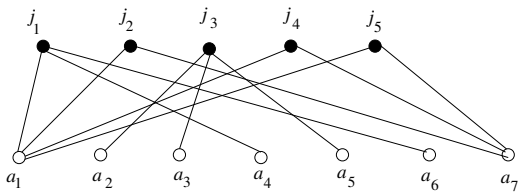


Fig. 5.14 A bipartite graph.

6. Find all maximum independent sets in the graph in Figure 5.6.
7. Find a maximum independent set, a minimum vertex cover and a minimum dominating set in the graph in Figure 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.

Chapter 6

Planar Graphs

6.1 Introduction

Consider a graph of six vertices and eight edges in Figure 6.1(a) where some pair 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 Figure 6.1(b). However it is not possible to redraw the graph in Figure 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.

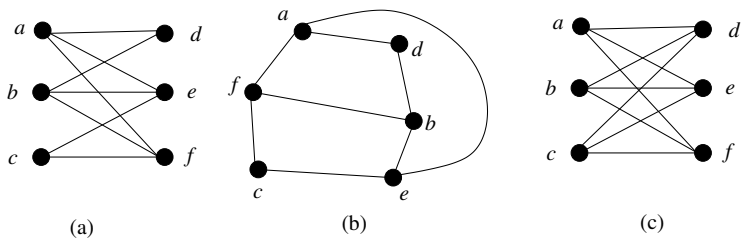


Fig. 6.1 Illustration for planar graph.

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. 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. 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.

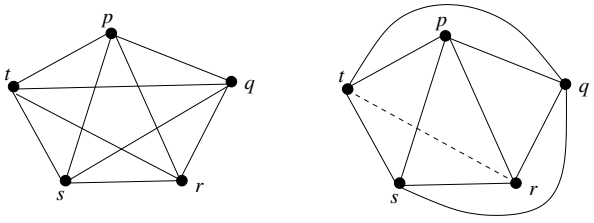


Fig. 6.2 Drawing of K_5

Theorem 6.2.1 *(Kuratowski 1930) 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 Figures 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 Figure 6.3(a), whereas the clockwise ordering is $(c, b), (c, d), (c, e), (c, f)(c, g)$ in Figure 6.3(b).

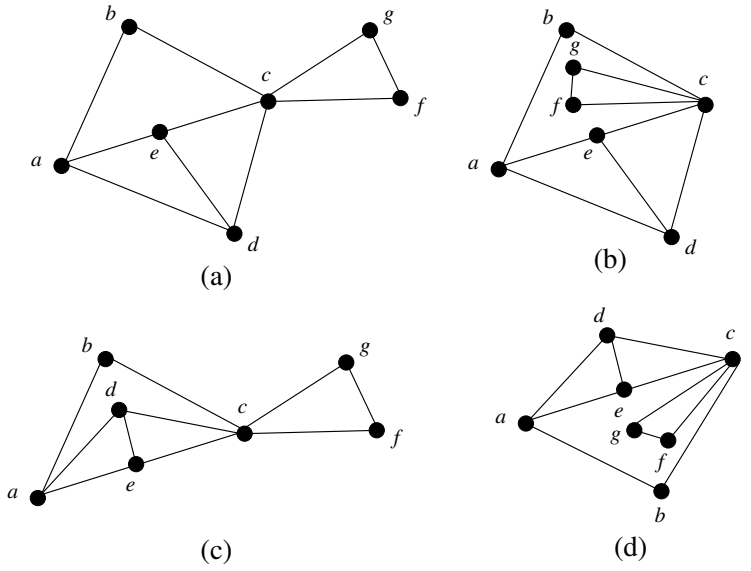


Fig. 6.3 Four planar embeddings of the same planar graph.

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. 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.

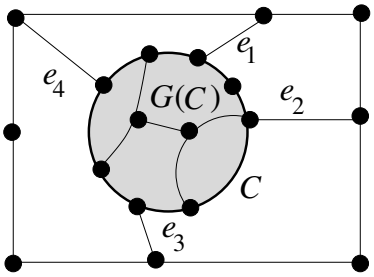


Fig. 6.4 Illustration of $G(C)$ and legs of a cycle C.

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 non-facial cycle in another plane embedding.(Note that the four embeddings in Figure 6.5 are not same but equivalent.) On the other hand the planar embeddings in Figures 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 [NC88]. (We may visualize this uniqeness on the surface of a sphere.) The embedding of the planar graph in Figure 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. Thus we shall assume that G is 2-connected if the embedding is unique. Whitney [Whi33] 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):

(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-

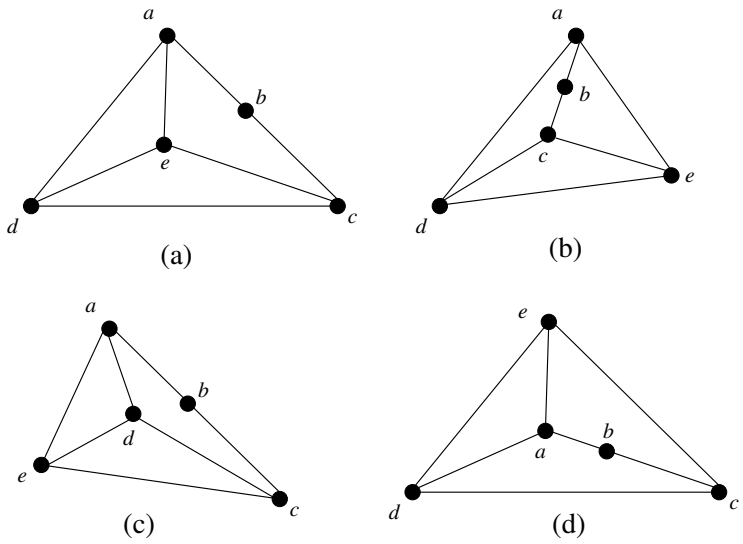


Fig. 6.5 Four equivalent planar embeddings of a planar graph.

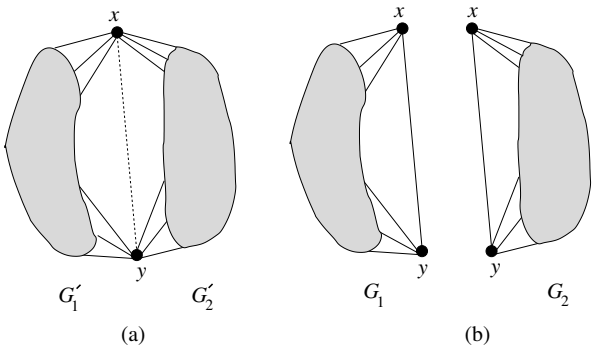


Fig. 6.6 (a) A graph G with a separation pair $\{x, y\}$ where edge (x, y) may exist, and (b) split graphs G_1 and G_2 of G.

components for the cycle C drawn by thick line in Figure 6.7(a) are shown in Figure 6.7(a). We now present the result of Whitney [Whi33] as in the following theorem whose proof is taken from [NC88, NR04].

Theorem 6.3.1 *The embedding of a 2-connected planar graph G is*

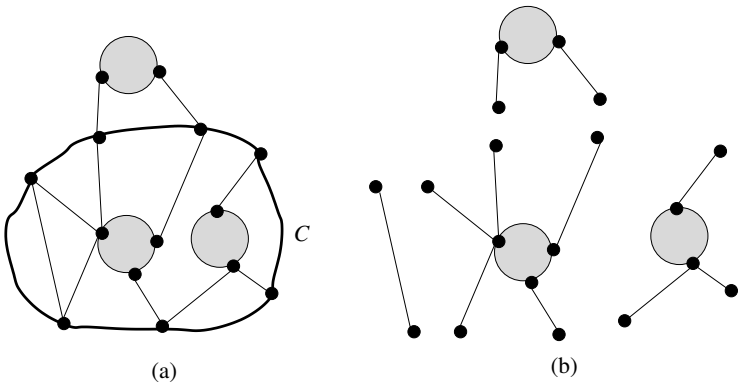


Fig. 6.7 Illustration for C-components.

unique if and only if G is a subdivision of a 3-connected graph.

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 $x_1, x_2, \cdots, x_k$ 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. □

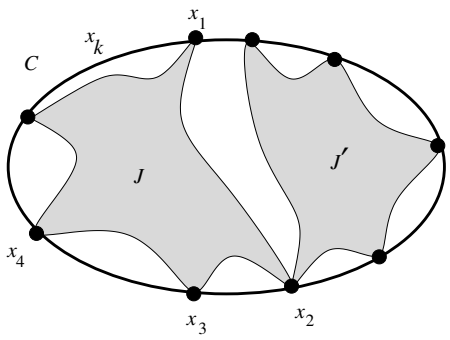


Fig. 6.8 Illustration for the proof of Theorem 6.3.1.

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 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 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.

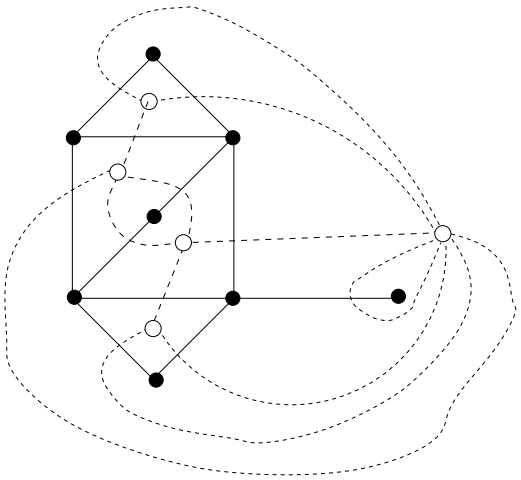


Fig. 6.9 A plane graph G drawn by solid lines and its dual graph G^* drawn by dotted lines.

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

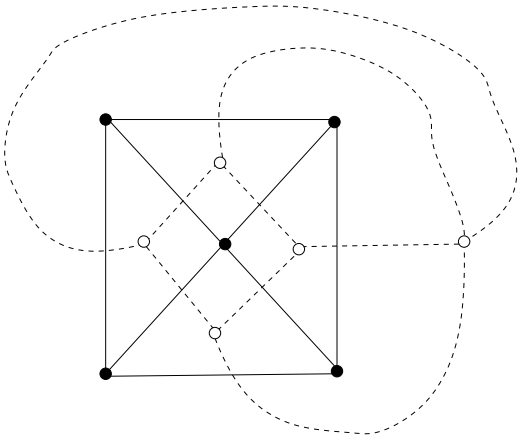


Fig. 6.10 A self-dual plane graph G and its dual graph G^*.

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.

6.4 Thickness of Graphs

In a Printed Circuit Board (PCB) we can not print a circuit with wire crossing. 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 "Jordan 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 (1936), Fáry (1948), and S. K. Stein (1951), 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 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 Fig. 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 polygon P formed by the neighbors of v (See Fig. 6.11(e) and (f)) as follows. Since P has at most five vertices, by art gallery theorem[1] 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 $O(n^2)$ [FPP90, Sch90]. Interested readers can find more on planar graph drawing in [NR04].

Exercise

1. Show that Petersen graph is non-planar. (Use Kuratowsk's theorem.)

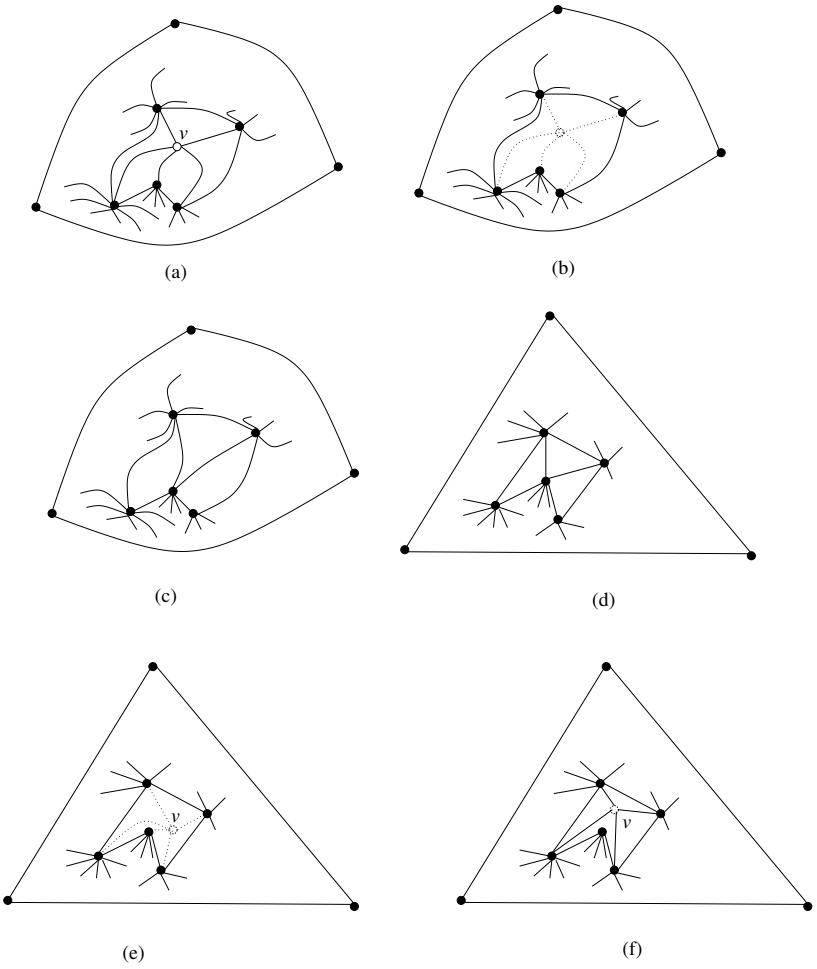


Fig. 6.11 Illustration for straight-line drawing.

2. Using Kuratowsk's theorem show that the two graphs in Figure 6.12 are non-planar.
3. Show that $m \leq 2n - 4$ for a planar bipartite graph of n vertices and m edges.
4. Identify all 3-legged cycles in the plane graph in Figure 6.13. How many

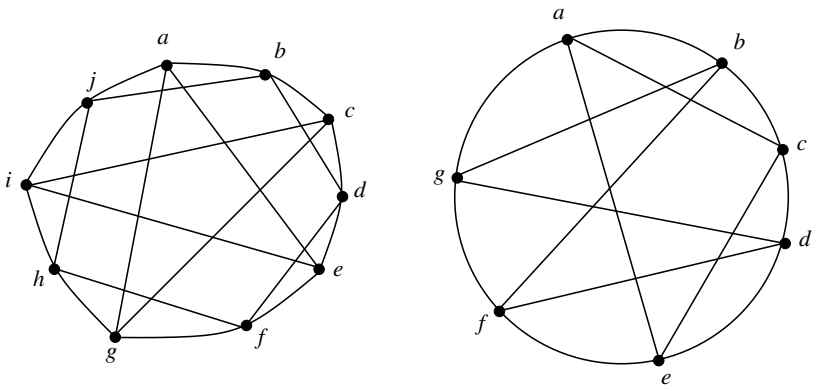


Fig. 6.12 Non-planar graphs.

of them are minimal?

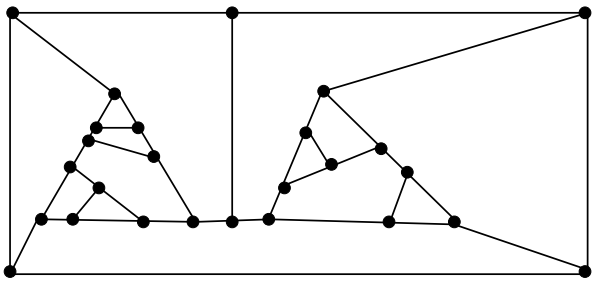


Fig. 6.13 3-leeged cycles.

5. Find a set of three independent 3-legged cycles in Figure 6.13 which are not minimal.
6. Constuct all plane embeddings of each of the graphs in Figure 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?

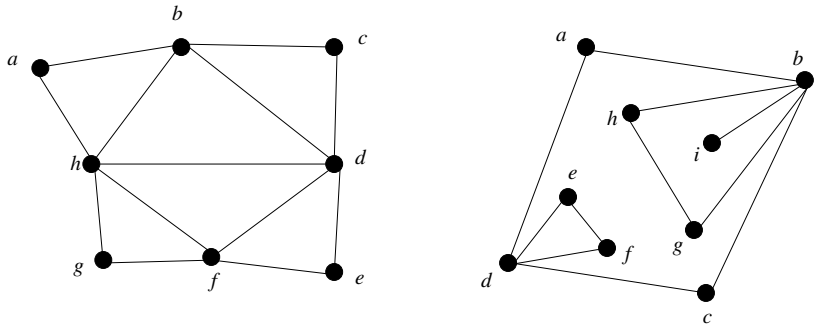


Fig. 6.14 Plane graphs.

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.

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 application in many other areas and has become an active area of reach 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 integers designates colors. The vertex coloring of graph in Fig. 7.1(a) is not optimal since there are 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-cromatic 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 classes*. The *vertex coloring problem*, i.e., coloring vertices of a graph G with $\chi(G)$ colors, is a NP-hard problem.

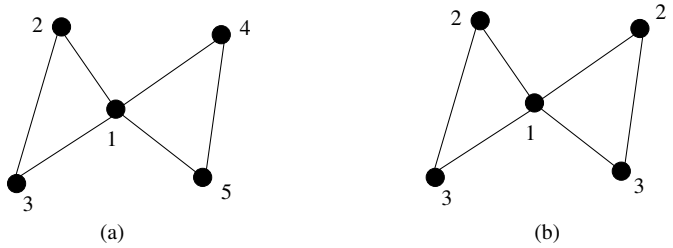


Fig. 7.1 (a) 5-coloring and (b) 3-coloring of a graph.

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 by 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 [MMI72, WP67]. 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.

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.*

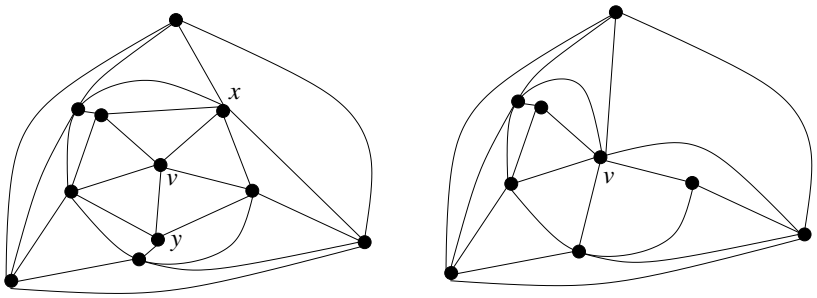
Proof. We prove the theorem by induction on the number n of vertices. The claim is trivially true for simple planar graphs of at most five vertices. Thus suppose that G is a simple planar graph of $n, n > 5$, vertices, and that all simple planar graphs with less than n vertices are 5-colorable. By Corollary 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 five 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 are non-adjacent, as illustrated in Figure 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 Figure 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. □

Based on the proof of Theorem 7.2.4 one can easily design an $O(n^2)$ recursive algorithm which finds a 5-coloring of a planar graph. Linear algorithms are also known for 5-coloring of planar graphs [CNS81b, MST80, Fre84].

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



(a) (b)

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

using a lenghty proof and the conjectured truned into the *4-color theorem.*

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-color 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 [Dir52].

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 is 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 graph 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 [Viz64], 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 $\Delta \leq \chi'(G) \leq \Delta + 1$.*

It is not known which graphs have chromatic index Δ and which have chromatic index $\Delta + 1$. However chromatic index for bipartite graphs is known as in the following theorem [Konig 1916].

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 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 figure 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. □

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 [CH91].

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.*

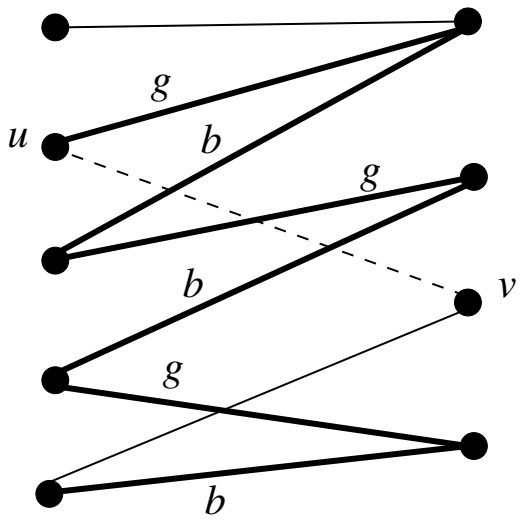


Fig. 7.3 Edge coloring of a bipartite graph.

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 Figure 7.4.) Since we have used n colors, $\chi'(K_n) \leq n$. To show $\chi'(K_n) = n$, 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, 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.

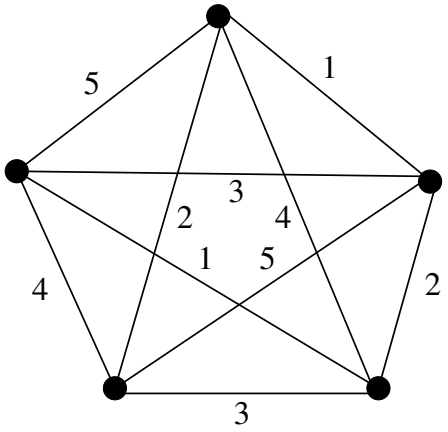


Fig. 7.4 Edge coloring of K_n for odd n.

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 Figure 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. Map coloring can be modeled as the face coloring of a plane graph where the map is represented by a plane

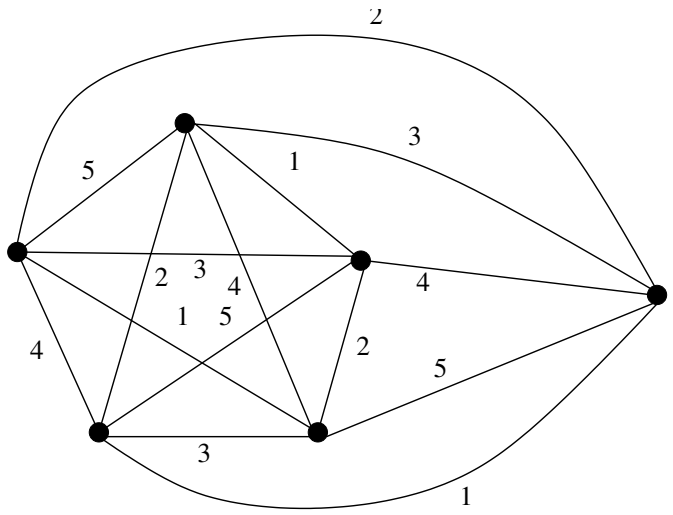


Fig. 7.5 Edge coloring of K_n for even n.

graph and color the faces of the plane graph in such a way that two faces having a conmmon edge receives 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.

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 2 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 [AG07]. 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, as 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 color. 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 [GTMP07, GTPW09] as well as in coding theory [SZ09]. Dujmović *et al.* [DMW05] have used acyclic coloring of planar graphs to obtain upper bounds on the volume of 3-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 [Grun73]. 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 [Och05].

In the rest of this section we will see some preliminary observation on acyclic 3-coloring from [MNWR12]. 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 $d(u_0) \geq 3$, $d(u_1) = d(u_2) = \ldots = d(u_l) = 2$, 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;

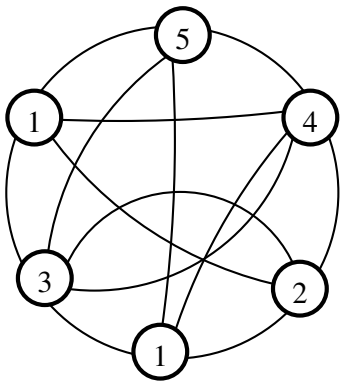


Fig. 7.6 An acyclic coloring of a graph using five colors.

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 Figure 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. 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. □

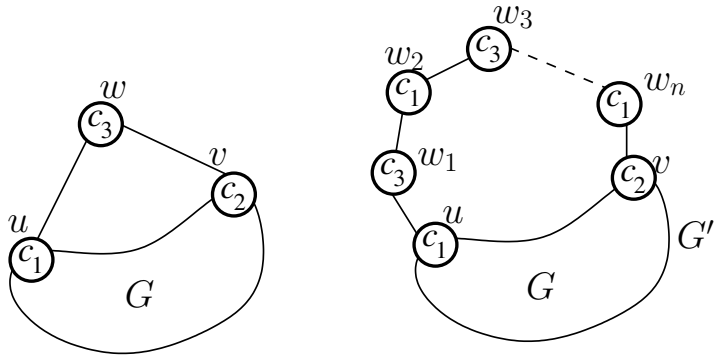


Fig. 7.7 Illustration for the proof of Lemma 7.6.1.

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 vertex 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 Figure 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 $(k-2)+1 = k-1$. □

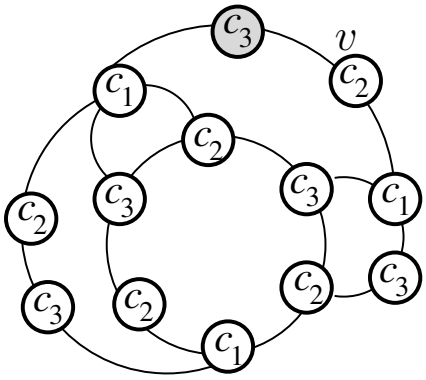


Fig. 7.8 Illustration for the proof of Lemma 7.6.2.

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 [CH91, Wil96] were followed for preparing this chapter.

Exercise

1. Obtain vertex coloring of the graphs in Figure 7.9 with the minimum number of colors.

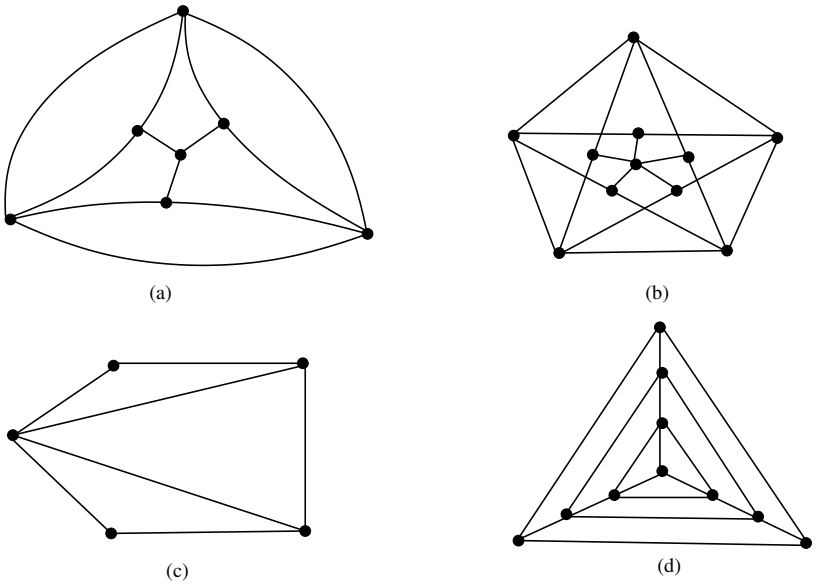


Fig. 7.9 Graphs.

2. Determine the chromatic number of Petersen graph.
3. Show that inner triangulated ladder graphs has chromatic number 3.
4. Show that dual graph of a maximal plane graph is a cubic graph.
5. Let G be a simple planar graph containing no triangle. Then show that $\chi(H) \leq 4$.
6. Show that for a tree of n vertices $P_G(k) = k(k-1)^{n-1}$.

7. Construct a 4-critical graph.
8. Let G be a k-critical graph. Then show that the degree of every vertex of G is at least $k-1$.

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 non-empty 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 Figure 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 Figure 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 similarly to a graph by a diagram, where each arc contains an arrow directing from tail to head, as illustrated in Figure 8.1.

The concept of the degree of a vertex for graphs also extends to digraphs.

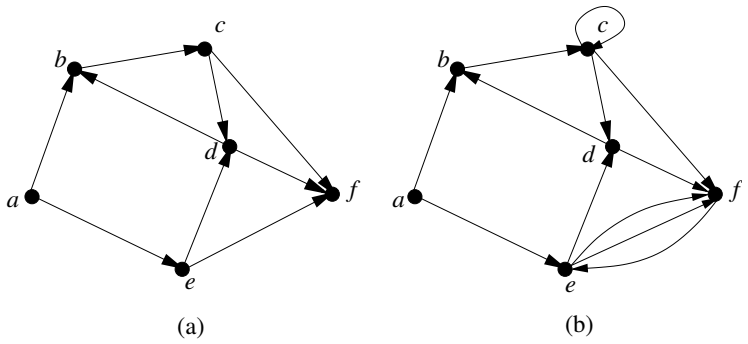


Fig. 8.1 (a) A simple digraph and (b) a multidigraph.

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 Figure 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 withe outdegree 0 is a *sink*. In the digraph in Figure 8.1(a), a is a source and f is a sink. The digraph in Figure 8.1(b) has no sink. We now have the following lemma which is analogous to Lemma 2.2.1.

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

Proof. Every arc is counted exactly once while counted indegrees. Again every arc is counted once while counting outdegree. 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 Figure 8.2(a) is shown in Figure 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 a

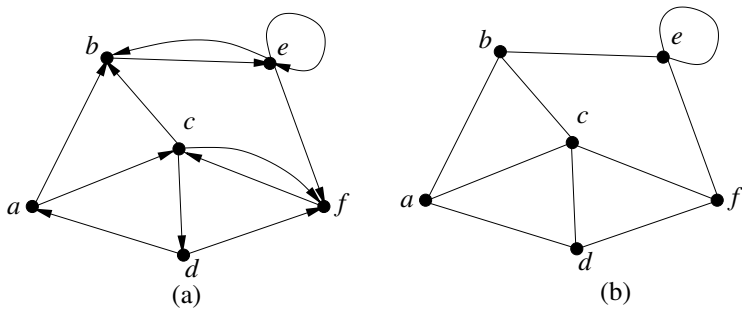


Fig. 8.2 (a) A digraph D and (b) the underlying graph $G(D)$.

symmetric uv if and only if $(u, v) \in E$. The digraph corresponding to G is denoted by $D(G)$. The digraph in Figure 8.3(b) is corresponding to the graph in Figure 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) illustrate two orientation of the graph in Figure 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 nodes 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 $\overleftrightarrow{T}$, is the digraph obtained from D by reversing the direction of each arc of D. Two digraphs are said to be *isomorphic* 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 $v_0v_1, v_1v_2, \cdots, v_{k-1}v_k$. 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

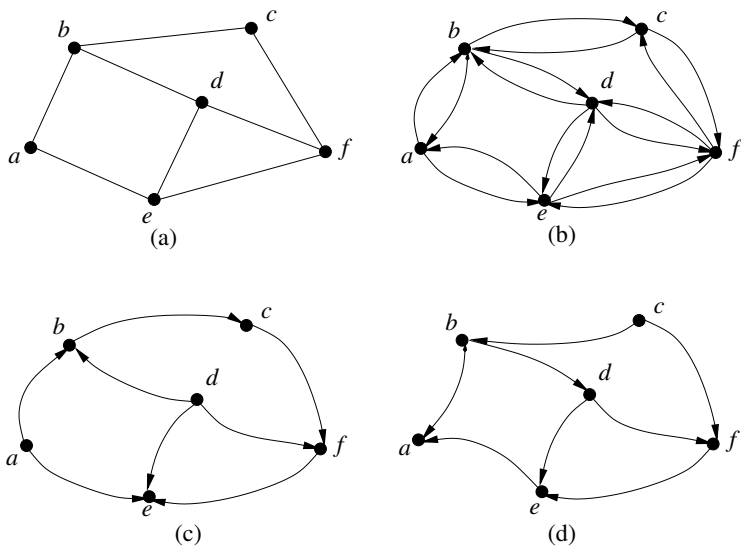


Fig. 8.3 (a) A graph G, (b) the corresponding digraph $D(G)$ and (c)-(d) orientations of G.

reachable from" is reflexive, transitive but not symmetric. A digraph is *strongly connected* if every pair of vertices are reachable from each other. Note that we call a digraph *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 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 graphs.

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 an 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 an Eulerian circuit in D using a method similar to one in the proof of 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 [Mey73] 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 non-adjacent 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 [Red34].

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. Assume 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 $P = v_1, v_2, v_3, \cdots, v_{n-1}$ 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 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 $u, u \neq v_{n-1}$ 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 Figure 8.4. □

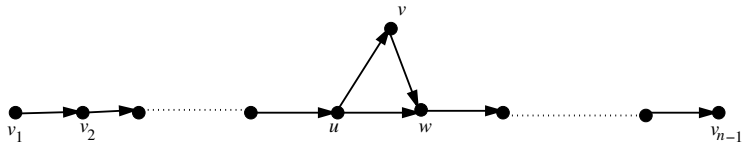


Fig. 8.4 (a) A simple digraph and (b) a multidigraph.

In a tournament T, the outdegree of a vertex represent the score of the corresponding team. Thus the *score sequence* of a tournament is the listing of outdegrees of vertices. It is listed in non-decreasing 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 non-negative 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 Figure 8.5.

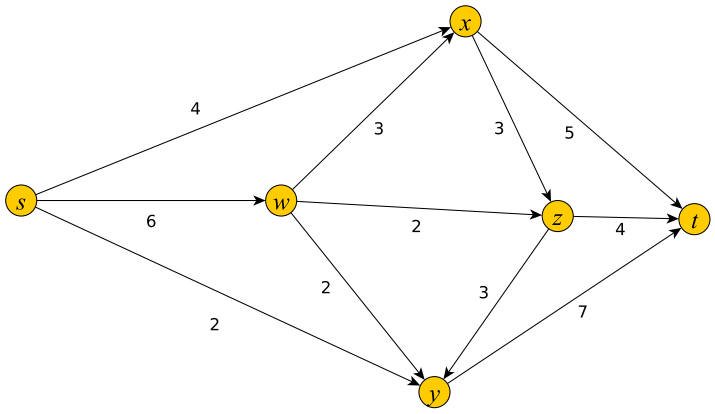


Fig. 8.5 A flow network.

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 in Figure 8.5 is shown in Figure 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 Figure 8.6 is 9. The *maximum flow* for a flow network $\mathcal{N}$ is a flow with the maximum value over all flows for $\mathcal{N}$.

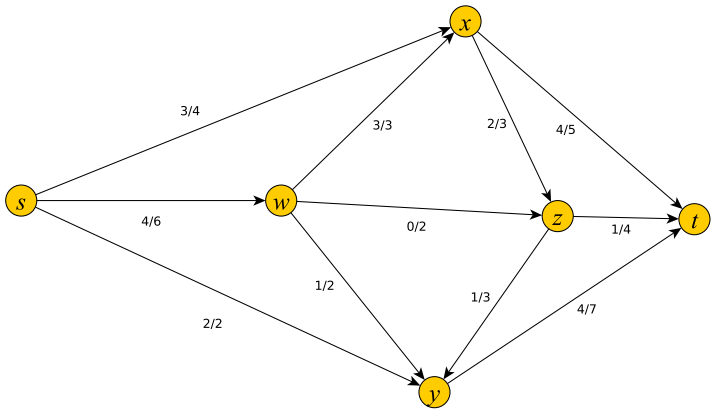


Fig. 8.6 Assignment of flow to a flow network.

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. 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 Figure 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 [FF56].

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 [GT02]. 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 a maximum flow in a network has applications in finding disjoint paths, bipartite matching, air-line scheduling etc.

Exercise

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 Figure 8.7.
3. Find all strongly connected components in the digraph in Figure 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.

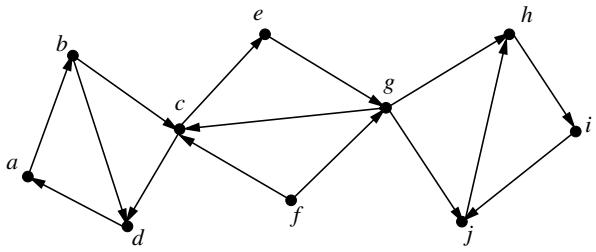


Fig. 8.7 A digraph.

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 Figure 9.1(a) is an outerplanar graph since it has an outerplanar embedding as shown in Figure 9.1(b). An *outer edge* of an outerplane graph is an edge on the outer face of the plane graph and all other edges are *inner edges*.

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 inner faces of G. Figure 9.2 illustrates the weak dual of an outerplane graph where the vertices of the weak dual is 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.*

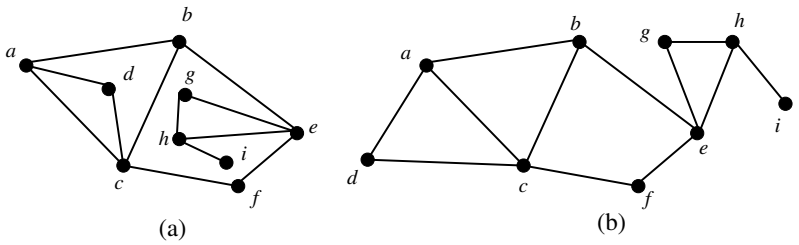


Fig. 9.1 An example of an outerplanar graph.

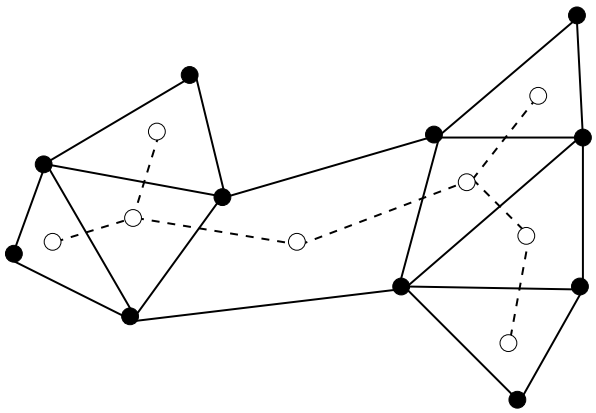


Fig. 9.2 Illustration for the weak dual of a biconnected outerplane graph.

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. □

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 outerplanar graph is a *maximal outerplanar 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 is known [Har72].

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 [JW10] 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 outerplanar graphs of maximum degree 4.

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 consecutive 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 $v_1, v_2, \cdots, v_n$ 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 need to have at least two inner edges incident to it. Since two outer edges are incident to each of

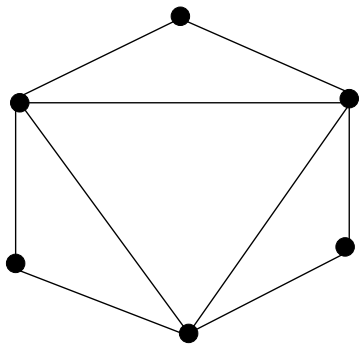


Fig. 9.3 Outerplanar octahedron.

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 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. □

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 triangulated plane graphs called canonical ordering. In 1990, de Fraysseix *et al.* [FPP90] 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 non-consecutive 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 $\pi = (v_1, v_2, \cdots, v_n)$ 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 $v_1, v_2, \cdots, v_k$. 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.* [FPP90].

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

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 $v_{n-1}, v_{n-2}, \cdots, v_3$ in this order, and show that (co1)–(co3) hold for $k = n-1, n-2, \cdots, 3$.

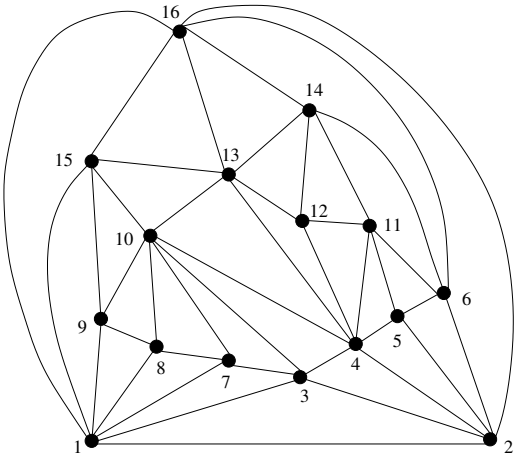


Fig. 9.4 A canonical ordering of a triangulated plane graph of $n = 16$ vertices.

Assume for inductive hypothesis that the vertices $v_n, v_{n-1}, \cdots, v_{k+1}$, $k+1 \geq 4$, have been appropriately chosen, and that (co1)–(co3) hold for 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 $C_o(G_k) = w_1, w_2, \cdots, w_t$, where $w_1 = v_1$ and $w_t = v_2$. If $C_o(G_k)$ has no chord, then any of the vertices $w_2, w_3, \cdots, w_{t-1}$ 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 $w_{p+1}, w_{p+2}, \cdots, w_{q-1}$ 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 $w_{p+1}, w_{p+2}, \cdots, w_{q-1}$ 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.

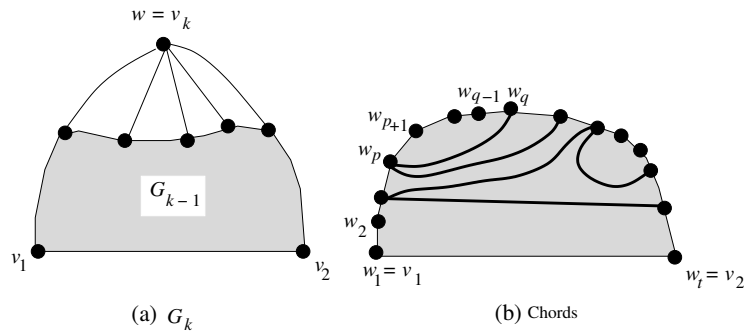


Fig. 9.5 Graph G_k and chords.

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 graph in Fig. 9.6 since the vertex a is outside of the triangle and the vertex e is inside of the 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 plane graph. We show that the number of separating triangles in a triangulated plane graph with n vertices is at most $n - 4$. We also give an example of a class of plane graphs with exactly $n - 4$ separating triangles.

Let T be a separating triangle of a plane graph G. We denote by $G(T)$ the plane subgraph of G inside T including T. A face of G is called an *internal face of* T if it is contained in $G(T)$; otherwise it is called an *external face of* T. Let T_p and T_c be two separating triangles of G. We say that T_p is an *ancestor* of T_c and T_c is a *descendant* of T_p if T_c is contained in $G(T_p)$. Additionally if T_c is not descendant of any descendant separating triangle of T_p, then we also say that T_p is the *parent* of T_c and T_c is a child of T_p. We now have the following lemma, whose proof is trivial.

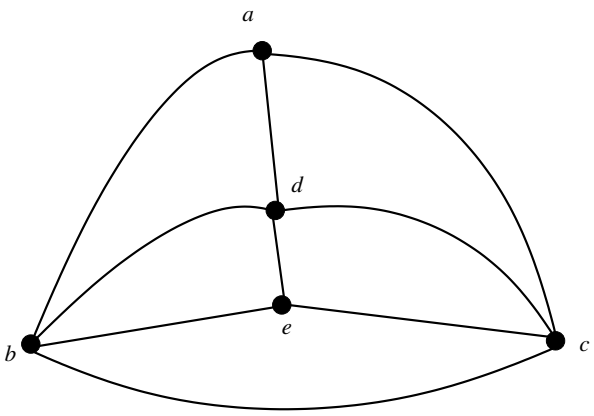


Fig. 9.6 Illustration for Separating triangles.

Lemma 9.3.2 *Let T_1 and T_2 be two separating triangles of a plane graph G. Then exactly one of the following three conditions holds.*

(1) T_1 is a descendant of T_2.
(2) T_2 is a descendant of T_1.
(3) T_1 and 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, as illustrated in Fig. 9.7. We call this tree the *genealogical tree* of G. We now prove the following theorem.

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

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

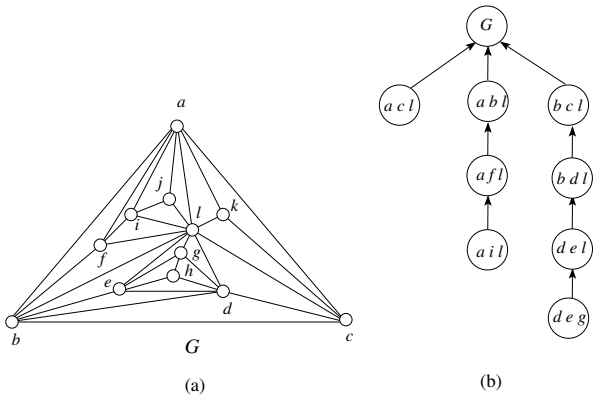


Fig. 9.7 (a) A plane graph G, (b) the genealogical tree of G.

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 $T = uvw$ of G, there are exactly three internal faces f_1, f_2 and f_3 of T, containing the edges (u, v), (v, w) and (w, u), respectively. Furthermore by Lemma 9.3.2, these three internal faces of T are not internal faces for any other separating triangles except for the ancestors and the descendants of T. We first consider the case where 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 T. We next consider the case where T is not a child of the root. In this case, 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 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 $2n-4$ [NR04]. 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 $n-4$.

□

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

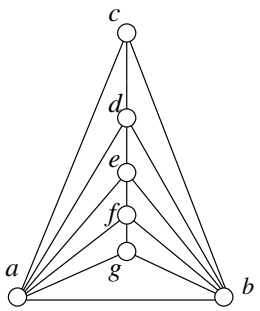


Fig. 9.8 A planar graph with $n-4$ separating triangles.

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:

(1) G is a triangulated plane graph;
(2) 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 Figure 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.

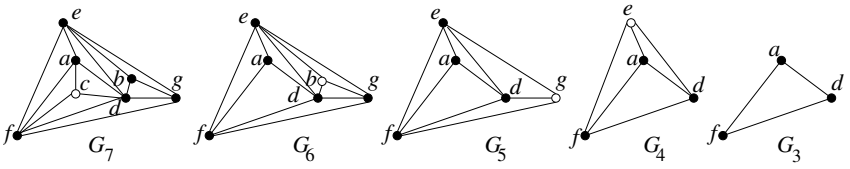


Fig. 9.9 Examples of plane 3-trees.

The following results are known on plane 3-trees.

Lemma 9.3.4 [BV10] *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.*

By Lemma 9.3.4(b) 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.* [MNRA11] 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 Figure 9.10. We call those three triangles the *nested triangles around p*.

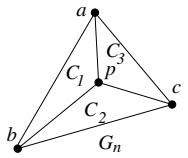


Fig. 9.10 Nested triangles around p.

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).

(1) if $n = 3$, T_{n-3} consists of a single vertex.
(2) 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 counter-clockwise 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 counter-clockwise order.

Figure 9.11 illustrates the representative tree T_{n-3} of a plane 3-tree G_n.

We now prove that T_{n-3} is unique for G_n in the following theorem [MNRA11].

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*

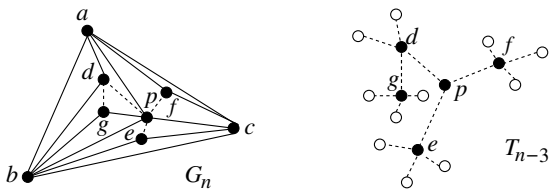


Fig. 9.11 (a) A plane 3-tree G_n and (b) the representative tree T_{n-3} of G_n.

and $2n-5$ *leaves.*

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 9.3.4(b) 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 counter-clockwise 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. □

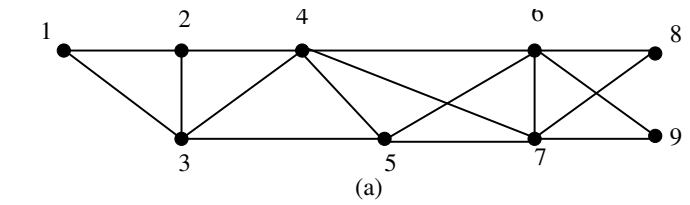
9.4 Chordal Graphs

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 in 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 $v_1, v_2, \cdots, v_n$ in which each v_i is simplicial in the graph induced by $v_i, v_{i+1}, \cdots v_n$. In the chordal graph in Figure 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 ordering of 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 [Spi03]. 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. All sets are subdivided into neighbors and non-neighbors of x in S with neighbors of x is 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 Figure 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



(123456789)
(123458) (67) (9)
(123) (458) (6) (7) (9)
(123) (45) (8) (6) (7) (9)
(12) (3) (4) (5) (8) (6) (7) (9)
(1) (2) (3) (4) (5) (8) (6) (7) (9)

(b)

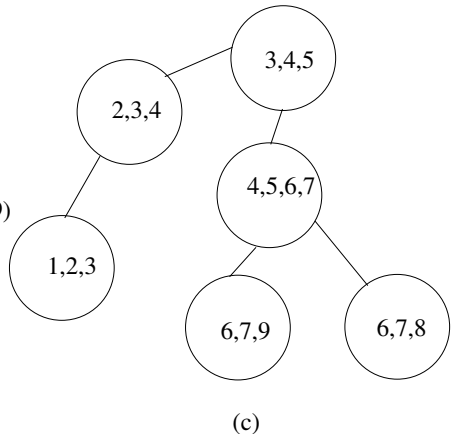


Fig. 9.12 (a) A chordal graph G, (b) step by step illustration for construction of a perfect elimination scheme of G and (c) a clique tree representation of G.

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 Figure 9.12(a) is given in Figure 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 [Dir61].

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. 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 Figure 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 Figure 9.13.

Sufficiency. Let G be a graph where each minimal separator is a clique. Assume that G is not chordal, and let $w, x, y, z_1, \cdots, z_k, w$ 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 Figure 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

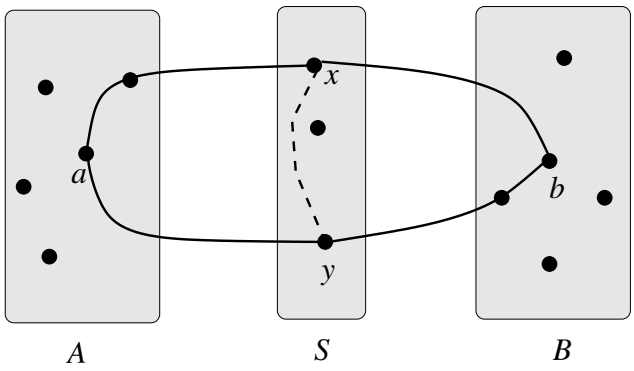


Fig. 9.13 Illustration for the proof of the necessity of Theorem 9.4.1.

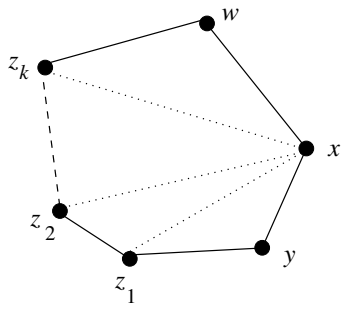


Fig. 9.14 Illustration for the proof of the sufficiency of Theorem 9.4.1.

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.

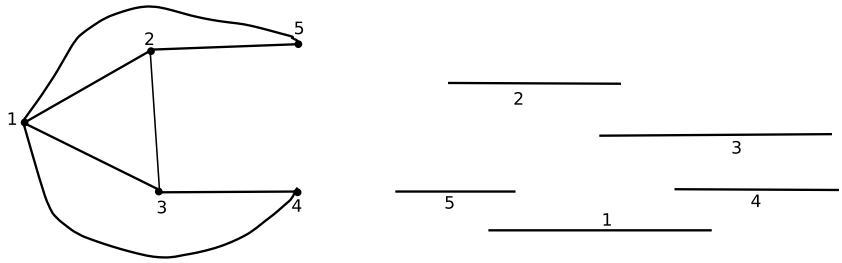


Fig. 9.15 An interval graph with corresponding intervals.

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 [GH64].

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.*

The class of interval graphs is a very nice class of graphs which support many efficient algorithmic techniques [Spi03]. 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 leads to an efficient dynamic programming algorithm.

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.

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 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 $t = t_2$ or $s = s_1 = s_2$ and $t = t_1 = t_2$ [REN05]. A biconnected component of a series-parallel graph is also 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 pair u, v of vertices of a connected graph G is a split pair if 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. 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.

Let G be a biconnected series-parallel graph. Let (u, v) be an outer edge of G. The *SPQ-tree* [DTV99, DT96] $\mathcal{T}$ of G with respect to a reference edge $e = (u, v)$ describes a recursive decomposition of G induced by its split pairs. Tree $\mathcal{T}$ is a rooted ordered 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.

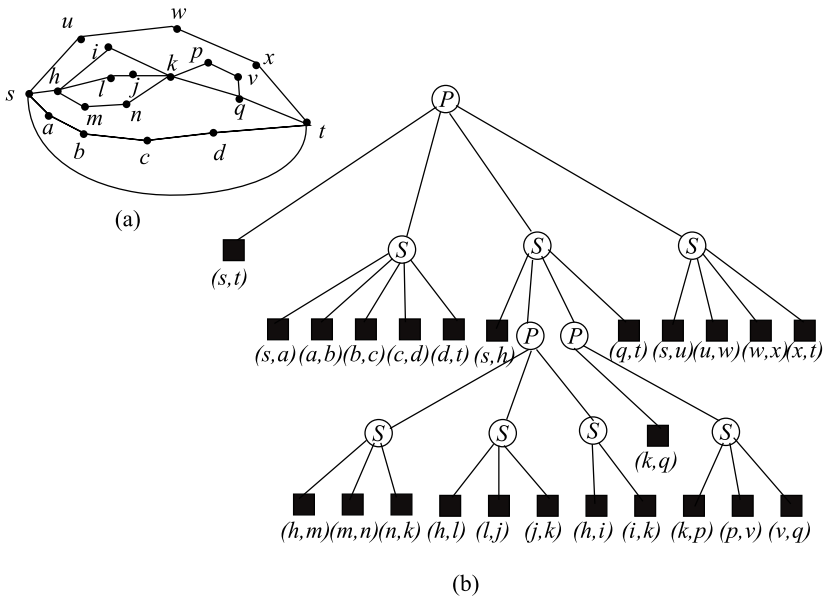


Fig. 9.16 (a) A series-parallel graph G and (b) an SPQ-tree $\mathcal{T}$ of G.

Trivial Case: In this case, G consists of exactly two parallel edges e and e' joining s and t. $\mathcal{T}$ consists of a single Q-node x, and the skeleton of x is G itself. The pertinent graph $G(x)$ consists of only the edge e'.

Parallel Case: In this case, the split pair u, v 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 $e = (u, v)$. The root of $\mathcal{T}$ is a P-node x. The $skeleton(x)$ consists of $k + 1$ parallel edges $e_0, e_1, \ldots, e_k$ joining s and t, where $e_0 = e = (u, v)$ and e_i, $1 \leq i \leq k$, corresponds to G_i. The pertinent graph $G(x) = G_1 \cup G_2 \cup \ldots \cup G_k$ is the union of $G_1, G_2, \ldots, G_k$.

Series Case: In this case the split pair u, v 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 t to s. Then the root of $\mathcal{T}$ is an S-node x. The skeleton of x is a cycle $e_0, e_1, \ldots, e_k$ where $e_0 = e, c_0 = u, c_k = v$, and e_i joins c_{i-1} and $c_i, 1 \leq i \leq k$. The pertinent graph $G(x)$ of node x is the union of $G_1, G_2, \ldots, G_k$. Figure 9.16 shows a series-parallel graph and its SPQ-tree decomposition.

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 $X_1, X_2, \cdots, X_k$ 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 induces a connected subtree of T.

Figures 9.17(b) and (c) illustrates two tree decompositions of the graph in Figure 9.17(a). Observe that the graph in Figure 9.17(a) is a chordal graph and the tree in Figure 9.17(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 opposition is not necessarily true. The tree decomposition in Figure 9.17(c) is not a clique tree.

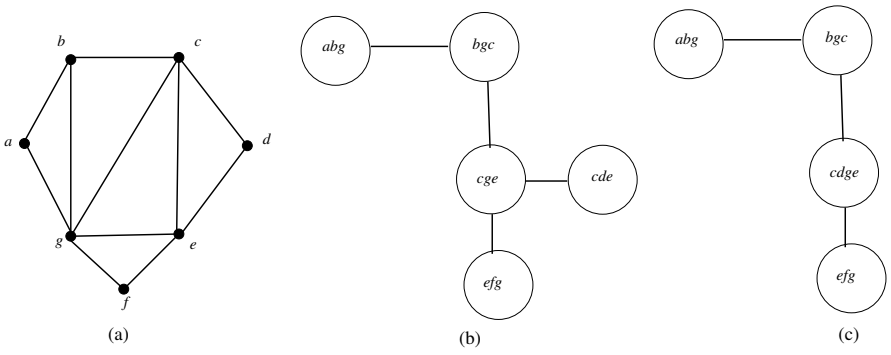


Fig. 9.17 Illustration for tree decomposition.

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

A *path decomposition* of G is a tree decomposition T of G such that T is a path. Figure 9.17(c) illustrates a path decomposition of the graph in Figure 9.17(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.

Exercise

1. Show that every maximal outerplanar graph contains at least two vertices of degree 2.
2. Prove Lemma 9.2.4.
3. Prove Theorem 9.2.5(b)-(d).
4. Find canonical ordering of the graph in Figure 9.18.
5. Identify all separating triangles in Figure 9.18 and construct a genealogical tree of separating triangles.

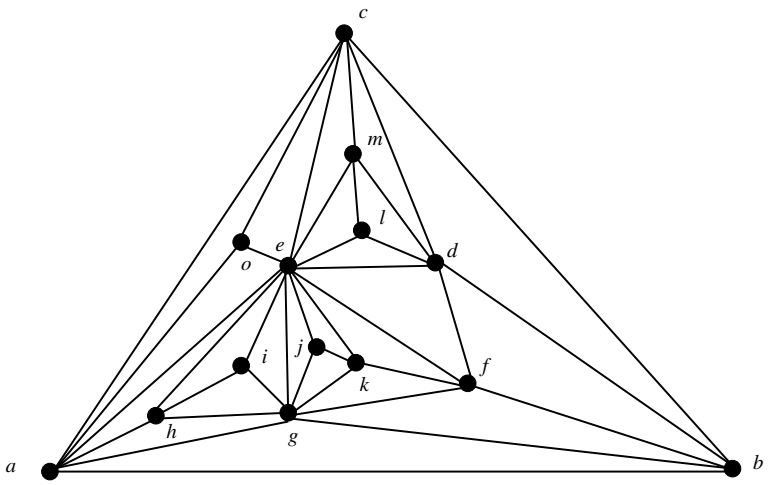


Fig. 9.18 A plane triangulation.

6. Construct a clique tree for the graph in Figure 9.19.
7. Find a perfect elimination scheme for the graph in Figure 9.19.
8. Show that all vertex induced subgraphs of a chordal graph are also chordal graphs.
9. Construct a tree decomposition of the graph in Figure 9.19 which is not a clique tree.
10. Construct a SPQ-tree decomposition of the graph in Figure 9.20.

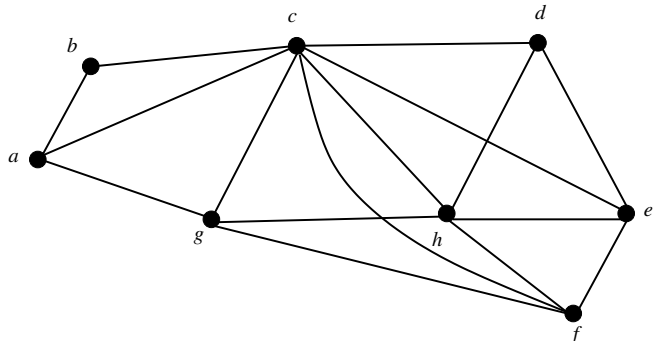


Fig. 9.19 A chordal graph.

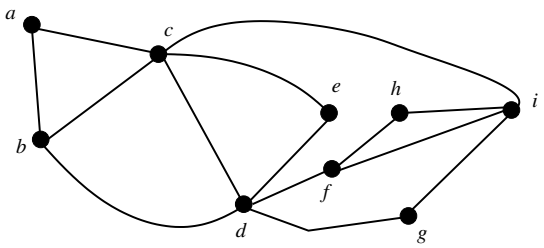


Fig. 9.20 A series-parallel graph.

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 adjacency matrix needs $O(n^2)$ space and 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 [Spi03].

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 will have a total of $2m$ entries for an undirected graph, since each entry appears on two lists. Each entry holds a number in the range $1 \cdots n$, and thus takes $logn$ bits. Thus the total space complexity is $\Theta(mlogn)$ 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 $\Theta(n^2logn)$ bits to represent a complete graph.

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 problem etc. Adjacency matrix and adjacency lists representations may not work fine in all aspects for every classes of graphs.

We take the class interval graphs as an example [Spi03]. 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 Figure 10.1(a). We can not 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 Figure 10.1(c). This assigns to every vertex two integers in the range $1 \cdots 2n$. For each vertex v, we store the labels of the endpoints of v; this uses $O(logn)$ 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 Figure 10.1(d). Thus we can reconstruct the graph from $O(logn)$ bits at each vertex , and interval graphs can be stored in $O(nlogn)$ bits space. Thus this representation is a space efficient

representation with fast adjacency test capability.

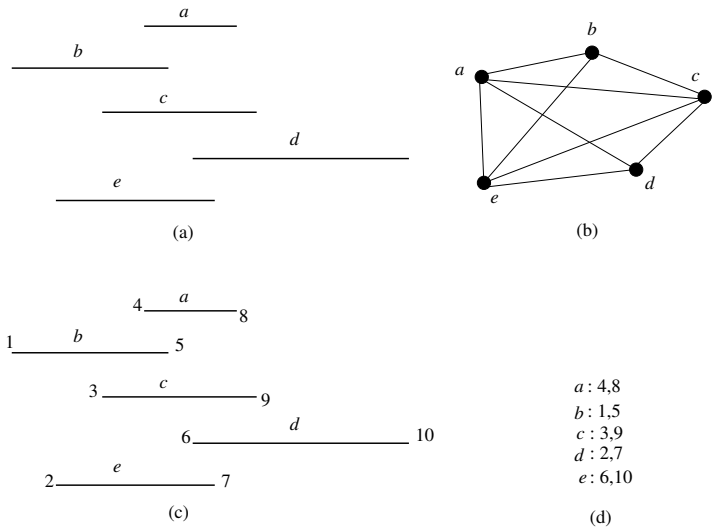


Fig. 10.1 Illustration for an efficient representation of interval graphs: (a) intervals, (b) the interval graph G corresponding to the intervals in (a), and (c) an efficient representation of G.

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 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 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 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 book of Spinrad [Spi03], interested readers can find more 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 intercon-

nections 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 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.

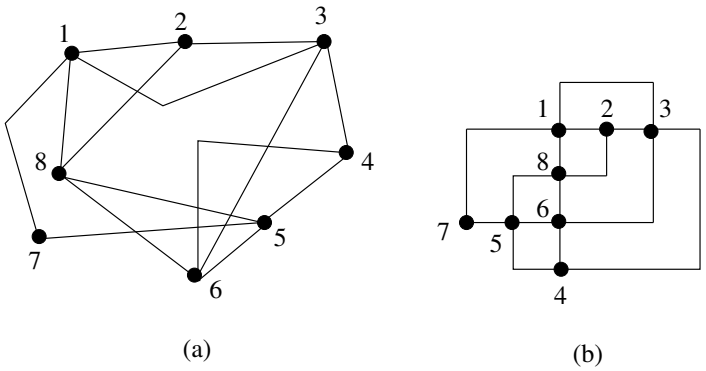


Fig. 10.2 An example of graph drawing in circuit schematics.

The field of graph drawing has been flourished very much in the last 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 [DETT99, KW01, Sug02, NR04, Tam14]. 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 drawing of the planar graph in Figure 10.3(a). In 1990, de Fraysseix *et al.* [FPP90] and Schnyder [Sch90] showed by two different methods that every planar graph of $n \geq 3$ vertices has a straight-line drawing on an integer grid of size $(2n-4) \times (n-2)$ and $(n-2) \times (n-2)$, respectively. A natural ques-

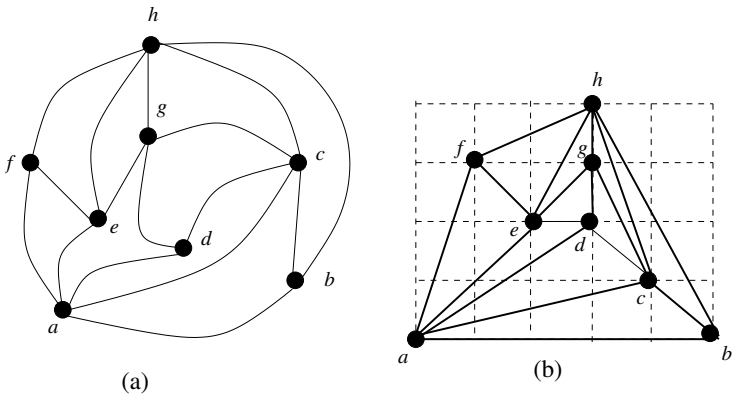


Fig. 10.3 (a) A planar graph G and (b) a straight-line drawing of G.

tion arises: what is the minimum size of a grid required for a straight-line 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 grid drawing [CN98, FPP90]. It has been conjectured that every plane graph of n vertices has

a 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 a straight-line grid drawing with area $(\lceil n/2 \rceil - 1) \times (\lfloor n/2 \rfloor)$ [MNN01]. Garg and Rusu showed that an n-node

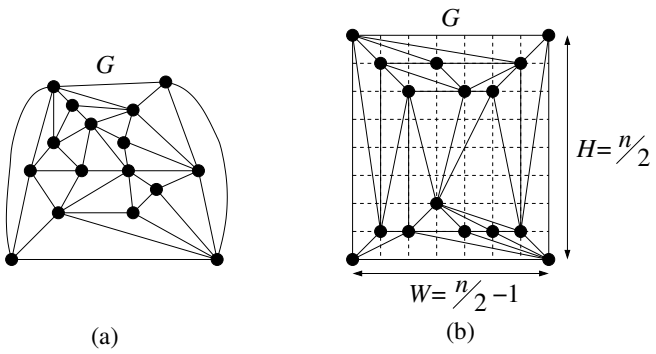


Fig. 10.4 (a) A planar graph G and (b) a straight-line drawing of G on a $(\lceil n/2 \rceil - 1) \times (\lfloor n/2 \rfloor)$ grid.

binary tree has a planar straight-line grid drawing with area $O(n)$ [GR02, GR03b]. 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 [BEG+03]. 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 $O(dn^{1.48})$ [GR03a]. 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 $O(n^{1.48})$ [DF05]. Recently Karim and Rahman [KR09] introduced a new class of planar graphs which has a straight-line grid drawing on a grid of area $O(n)$. They also give 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 Figure 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.

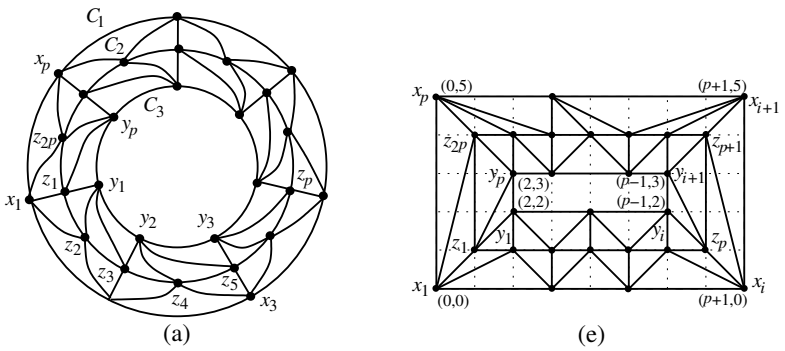


Fig. 10.5 (a) A doughnut graph G and (b) a straight-line drawing of G on a grid of linear area.

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 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 [Tut60] 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 [Tut63], and leads to an $O(n^{1.5})$ time convex drawing algorithm for a planar graph with a fixed embedding. Chiba, Yamanouchi and Nishizeki [CYN84] 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.

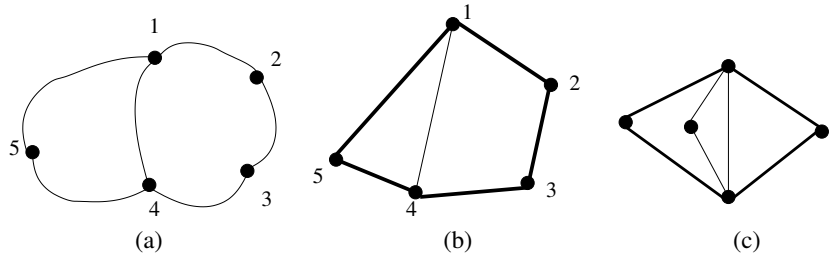


Fig. 10.6 (a) A planar graph G, (b) a convex drawing of G and (c) a planar graph having no convex drawing.

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 [CK97] 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 [MAN06], [MNN06], [ZN10]. 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 [ZN10].

10.3.1.3 *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 10.7(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 Figures 10.7(b). However, G in Figure 10.7(a) does not have a point-set embedding on the point-set S' in Figure 10.7(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.* [IPTT94] 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 [Bos02]; 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 [Cab06] 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 [GHH+07]. Recently, Nishat *et al.* [NMR12] 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 [DM12] proved that the point-set embeddability problem remains NP-complete for 3-connected plane graphs.

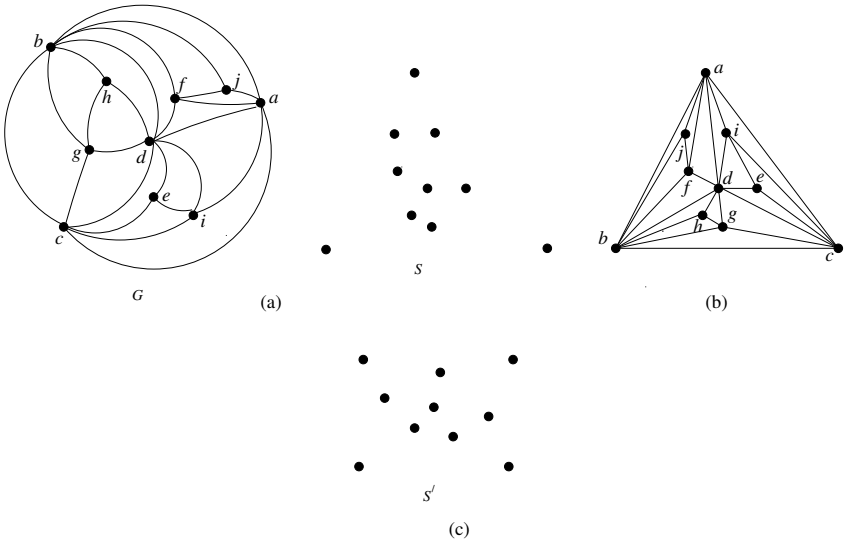


Fig. 10.7 (a) A plane graph G with ten vertices and a set S of ten points, (b) a point-set embedding of G on S and (c) a point-set of ten vertices on which G has no point-set embedding.

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 n-universal point-set of smaller size is an interesting area of research [CHK13, DEL+13]. Instead of point-set, this problem has also been studied on line-sets [DEK+11, HMRS13]. Interesting research problems can be formulated by combining other drawing styles like convex drawing, orthogonal grid drawing with point-set embeddings [DFF+12, DM12].

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 10.8(b) illustrates a simultaneous embedding of the two graphs in Figure 10.8(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 [BKR14]. Readabilty 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 in 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 [BKR14].

10.3.3 *Drawings of Nonplanar Graphs*

In the last subsection 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.

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

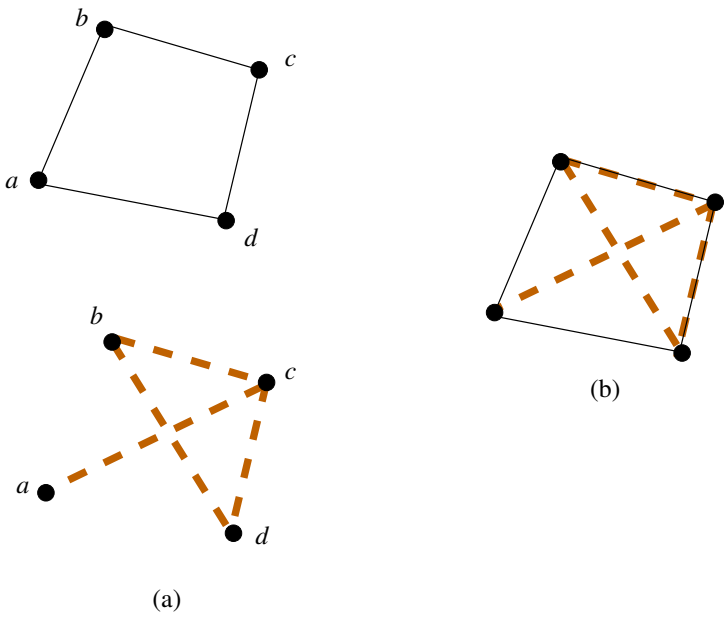


Fig. 10.8 (a) Two graphs, and (b) a simultaneous embedding of the two graphs in (a).

is NP-complete in general [GJ83]. However, interesting eye-tracking experiments by Huang et al. [HHE07, HHE08] indicate that the negative impact of an edge crossing is eliminated in the case where the crossing angle is greater than 70 degrees. 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 10.9 shows a RAC drawing of a non-planar graph. Didimo, Eades and Liotta [DEL11] 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 and Liotta [EL13] 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.* [ABS12] 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 non-trivial nonplanar graphs is an interesting area of research.

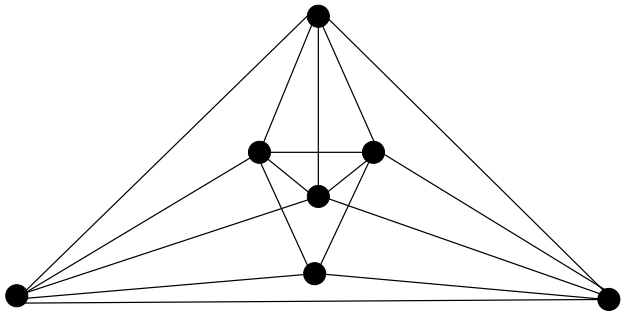


Fig. 10.9 A RAC drawing of a non-planar graph..

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 where the vertical line segment representing an edge must connect the horizontal line segments representing the end vertices. Otten and Wijk [OW78] have shown that every planar graph admits a visibility drawing, and Tamassia and Tollis [TT86] have given $O(n)$ time algorithm for constructing a visibility drawing of a planar graph of n vertices. Dean *et al.* have introduced a generalization of visibility drawing for a non-planar graph which is called bar k-visibility drawing [DEG+07]. 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 incident 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 Figures 10.10(a) and (b), respectively.

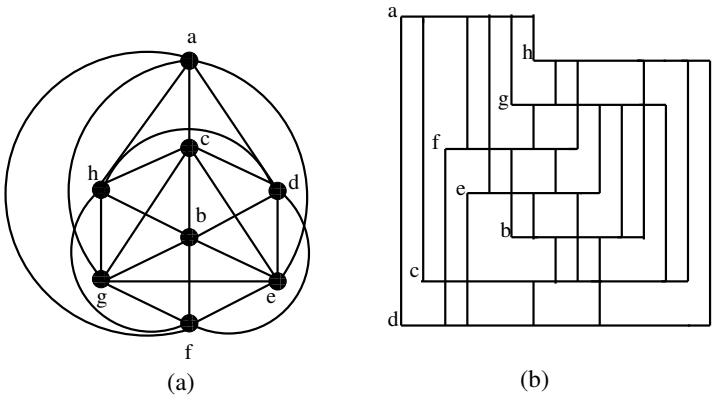


Fig. 10.10 (a) A 1-planar graph G, (b) a bar 1-visibility drawing of G.

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.* [SRRT14] gave linear algorithms for finding bar 1-visibility drawings of some subclasses of 1-planar graphs and conjectured that every 1-planar graph has a bar 1-visibility drawing. The conjecture of Sultana *et al.* triggered the research in this area and several results are published recently [Bra14].

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 technology and sports tournament scheduling. In Section 4.7 we came to know about graceful labeling. Graph colorings studied in Chapter 7 are also graph labellings. 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.

In antibandwidth problem [RSS+09] we are asked to label the vertices in such way that the minimum difference between the labels of two vertices is maximized. More formally, let G be a graph on n vertices. Given a bijection $f : V(G) \to \{1, 2, 3, \ldots, n\}$, 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 Figure 10.11 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 [YJ03], separation number of graphs [LVW84], maximum differential graph coloring problem [HKV10].

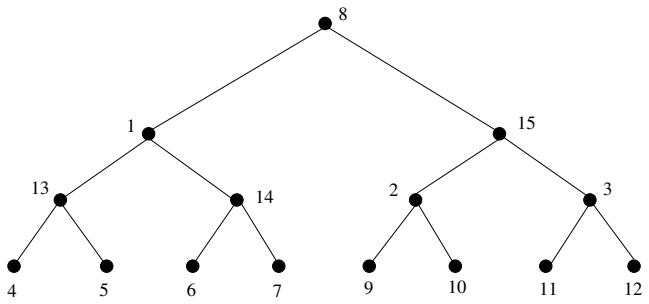


Fig. 10.11 Atibandwidth labeling of a full binary tree.

The antibandwidth problem is NP-hard [LVW84] for general graphs. It is known to be polynomially solvable for the complements of interval, arborescent comparability and threshold graphs [DI99]. Exact results and tight bounds are known for paths, cycles, grid, meshes, hypercubes [YJ03,

RSS+09, HKV10]. For a complete binary tree with h height the antibandwidth is $2^h - 1$ [WJX03] and for complete k-*ary* tree [CMTV09], the antibandwidth is $\frac{n+1-k}{2}$, when k is even. If $P_{m,n}$ is a tree, obtained from a path P_m and m copies of path P_n such that the i^{th} vertex of P_m is adjacent to an end vertex of the i^{th} copy of P_n. Then the antibandwidth of $P_{m,n}$ is $\lfloor \frac{mn}{2} \rfloor$ [WJX03]. But above claim cannot be extended for graphs, when every vertex of P_m has more than one copies of P_n.

Millar *et al.* [MP89] introduced an antibandwidth labeling scheme for forests, which is known as Millar-Pritikin labeling scheme. According to Millar-Pritikin labeling scheme, for a forest G, if the bipartition sets are X and Y, where $|X| \leq |Y|$. Then the antibandwidth of G is at least $|X|$. They also showed that for a balanced bipartite graph ($|X| = |Y|$) Millar-Pritikin labeling scheme produces optimal antibandwidth value. Interestingly, still there is no results for any class of graphs, for which the optimal value of antibandwidth is non-trivial (i.e., not $\lfloor \frac{n}{2} \rfloor$).

Finding the antibandwidth of a graph has several practical applications. 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 [CMTV09]. Given a map, we define the country graph $G = (V, E)$ 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 [HKV10]. Given n transmitters and n frequencies, the frequency assignment problem is to find a bijective frequency assignment where the interfering transmitters have as different frequency as possible, where transmitters are the vertices of graph G and there is an edge between two interfering transmitters [CMTV09].

There are several open problems on antibandwidth labeling [BKKV14].

- 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 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 $u_1, u_2, \cdots, u_k \in V$ and k natural numbers $n_1, n_2, \cdots, n_k$ such that $\sum_{i=1}^{k} n_i = |V|$, we wish to find a partition $V_1, V_2, \cdots, V_k$ 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. A 4-partition of a graph G is depicted in Fig. 10.12, 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 [WK94, WTK95]. The problem is NP-hard in general [DF85], 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, Győri and Lovász independently proved that every k-connected graph has a k-partition for any $u_1, u_2, \cdots, u_k$ and $n_1, n_2, \cdots, n_k$ [G78, L77]. 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 have been known:

(i) a linear-time algorithm to find a bipartition of a biconnected graph [STN90, STNMU90];
(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 [STN+90];
(iii) a linear-time algorithm to find a tripartition of a triconnected planar graph [JSN94];
(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 [NRN97];
(v) a linear-time algorithm for 5-partitioning of a 5-connected internally triangulated plane graph if the five basis vertices are all on the boundary of one face [NN01]; and
(vi) a linear-time algorithm for finding a k-partition of a doughnut graph G with at most two base vertices [KNR09].

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 [WTK95, AR10, AR12]. A *resource k-partitioning* is defined as 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 [WTK95] and in compu-

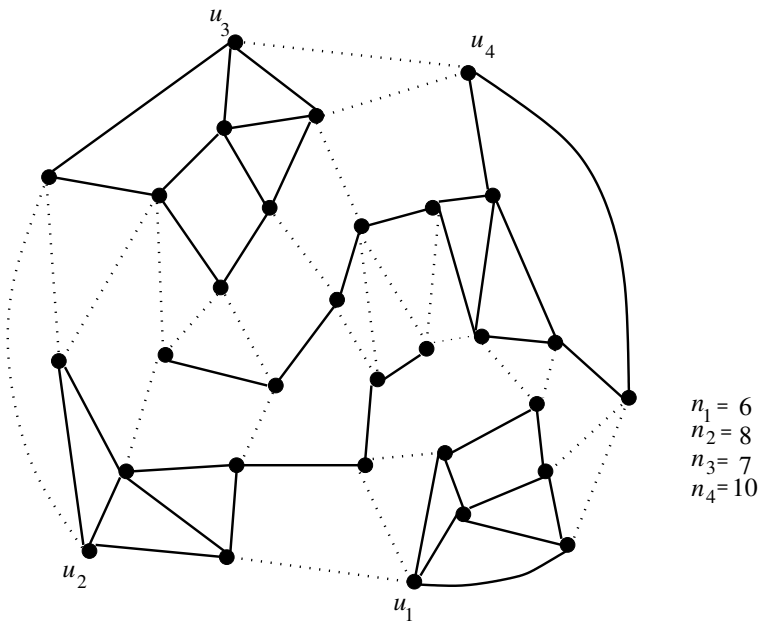


Fig. 10.12 A 4-partitioning of a 4-connected plane graph G.

tational aspects, too. For example, in grid computing we are required to divide a complex task such as computation of fractals into several subtasks and then we wish to delegate each of these subtasks to a computing element in the grid such that a computing element in the grid is not overwhelmed with tasks from other clients. This concept applies to telecommunication networks, fault tolerant systems, various producer-consumer problems and so on [AR12]. Graph partitioning has become very useful concepts for clustering in the area of data mining and wireless sensor networks.

.

10.6 Graphs in Bioinformatics

Graph theory has a glorious history with bioinformatics. Prior to Watson and Crik's 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.

Dealing with the problems of bioinformatics, researchers of graph algorithms usually 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 [JP04]. 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. 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 10.13 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 Figure 10.13(a) and reconstruction of a sequence AGCATTCAGT from a Hamiltonian path in H is illustrated in Figure 10.13(b).

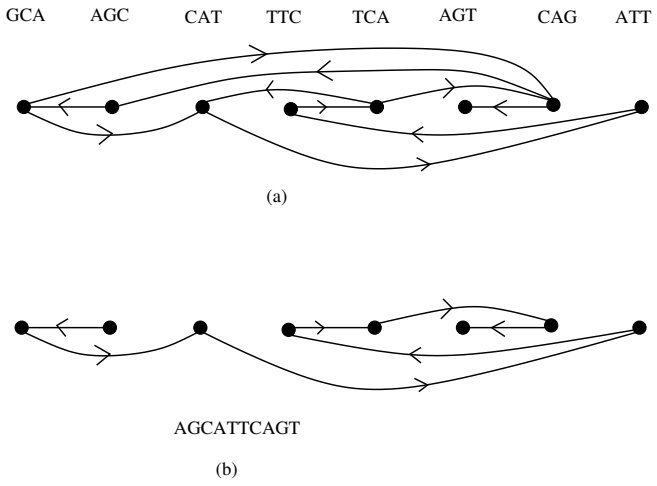


Fig. 10.13 Reconstruct a DNA sequence from its l-mers.

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 which leads to a simple linear algorithm [JP04].

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 [TAM11].

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 [BAT+02]. 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 non-negative 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 non-uniform 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, 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 straight-forward 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 reconstruction of ancestral relationships is one of the fundamental problems in computational biology as widely used to provide both evolutionary and functional insights into biological systems. The evolutionary history of a set of organisms is usually represented by a dendogram called 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. In the phylogenetic tree reconstruction problem, given a set of species/taxa, we want to find a phylogenetic tree that best explains the given data. Due to the difficulty in determining the criteria for an optimal tree, the performance of the reconstruction algorithms is evaluated through comparisons of the output with the known tree. However, as the tree reconstruction problem is proved to be NP-hard under many criteria of optimality and as real phylogenetic trees are of a large dimension, testing these heuristics on real data is difficult. Thus, it is interesting to find efficient ways to sample subsets of taxa from a large phylogenetic tree, subject to some biologically-motivated constraints, in order to test the reconstruction algorithms on the subtree induced by the sample. In this context, Kearney et al. [KMP03] in 2003 introduced the concept of PCG

and showed how to use it to model evolutionary relationships among a set of organisms.

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 10.14(a) illustrates an edge-weighted tree T, and Figure 10.14(b) shows the corresponding PCG G, where $d_{min} = 2$ and $d_{max} = 3.5$. Figure 10.14(c) shows another edge-weighted tree T' such that G is a PCG of T' when $d_{min} = 1.5$ and $d_{max} = 2$.

They hoped to show that every graph is a PCG, but later, Yanhaona et al. [YBR10] constructed a 15-vertex graph that is not a PCG. Several researchers have attempted to characterize pairwise compatibility graphs. Yanhaona et al. [YHR09] proved that graphs having cycles as its maximal biconnected components are PCGs. Salma and Rahman [SRH13] proved that every triangle-free maximum-degree-three outerplanar graph is a PCG. Calamoneri et al. [CPS12] gave some sufficient conditions for a split matrogenic graph to be a PCG.

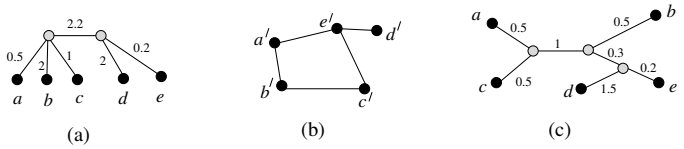


Fig. 10.14 (a) An edge-weighted tree T. (b) A PCG G of T, where $d_{min} = 2, d_{max} = 3.5$. (c) Another edge weighted tree T' such that G is a PCG of T' when $d_{min} = 1.5, d_{max} = 2$.

Finding a pairwise compatibility tree of a given graph appeared to be difficult, even for graphs with few vertices. In 2003, Kearney et al. [KMP03]

showed that every graph with at most five vertices is a PCG. Yanhaona et al. [YBR10] constructed a 15-vertex graph which is not 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. [CFS12] proved that every graph with at most seven vertices is a PCG. Recently Durocher *et al.* [DMR13] have constructed a graph of eight vertices that is not a PCG. They also constructed a planar graph with twenty 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.

- Salma and Rahman [SRH13] proved that every triangle-free maximum-degree-three outerplanar graph is a PCG. Find the larger classes of outerplanar graphs which are PCGs.
- 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?

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 [Erc13]. A wireless sensor network (WSN) consists of a number of sensor nodes spread over a geographic area. Each sensor has wireless communication and signal processing capabilities. Some designated sensor nodes have more advanced communication capabilities which collects all the data from the sensors,

possibly perform further processing of this data, and transfer it to another computer for advanced data processing. Such a designated node is called a *sink* or a *fusion center* of the wireless sensor network. A 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 signal 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 network* since there is no fixed or static infrastructure.

In a 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 Figure 10.15, where nodes send their sensed data to the nearest sensor neighbor 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 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 therefore save energy. WSN designers and implementors are faced with the following challenges [Erc13].

Scalability: A sensor network may consist of hundreds or thousands of nodes. Protocols such as for routing and algorithms for these networks should be scalable.

Fault Tolerance: Failure of a single node should not affect the overall operation of a sensor network, and connectivity of the network should be maintained using new links if required.

Node Deployment: Nodes can be deployed randomly or regularly with the aim of providing maximum coverage of the area to be monitored by the

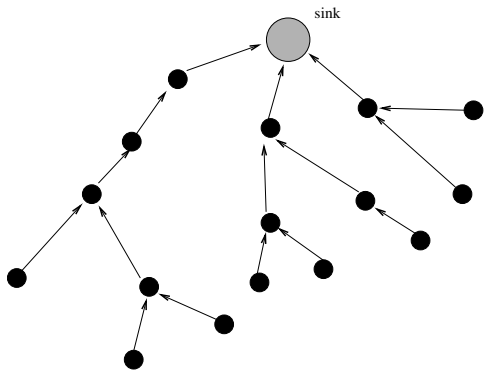


Fig. 10.15 A wireless sensor network.

sensors. Also, the deployment should provide a connected graph of the network.

Power Management: Nodes of a sensor network rely on batteries with limited life-time, and replacing these batteries may be difficult or impossible in many applications where sensors are deployed in harsh conditions. Transmission and reception of messages consume much more energy than local computations, therefore, protocols and algorithms should be designed to work with minimum number of messages. An efficient method to conserve energy is to put some of the sensors into sleep mode when there are not many activities to be sensed. An important requirement in these methods would then be to provide full coverage of the area by the awake sensors.

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 devises are represented by nodes of the graph and two nodes u and v in the graph if the Euclidean distance between the corresponding sensor nodes is at most the transmission radius. In such a model identical transmission capabilities are assumed for each sensor. However transmission range, energy level may vary for the nodes. In the *interfer-*

ence model edges are added considering the interferences of the nodes while transmitting or receiving message.

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 out, a subset of edges are activated dynamically to maintain a shortest spanning tree of the active sensor nodes. A minimal subset of active nodes which which induces connected subgraph and capable of performing special task 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 wireess 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 backbone network.

10.7.2 *Fault Tolerance*

Failure of a single node should not affect the overall operation of a sensor network, and connectivity of the network should be maintained using new links 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 [Kan96, AGHTU08]. 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 whiich 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 $\{V_1, V_2, \cdots, V_k\}$, 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*. They cluster heads dominate the other nodes

within the clusters and hence they are responsible for cluster management functions and for providing hierarchical routing. The responsibility of the cluster heads 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 property of the graph such as the degrees of the nodes, ecentricities of the nodes etc. and 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.