JAMES A. McHUGH

ALGORITHMIC GRAPH THEORY

Algorithmic Graph Theory

James A. McHugh

New Jersey Institute of Technology

PRENTICE HALL, *Englewood Cliffs, New Jersey 07632*

Library of Congress Cataloging-in-Publication Data

McHugh, James A.
Algorithmic graph theory / by James A. McHugh.
p. cm.
Bibliography: p.
Includes index.
ISBN 0-13-023615-2
1. Graph theory. 2. Graph theory--Data processing. I. Title.
QA166.M39 1990
511'.5--dc19 88-30722
CIP

Editorial/production supervision and
interior design: Joe Scordato
Cover design: Lundgren Graphics, Ltd.
Manufacturing buyers: Mary Noonan and Bob Anderson

Dedicated to my parents: Ann and Peter

© 1990 by Prentice-Hall, Inc.
A Division of Simon & Schuster
Englewood Cliffs, New Jersey 07632

All rights reserved. No part of this book may be
reproduced, in any form or by any means,
without permission in writing from the publisher.

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

ISBN 0-13-023615-2

PRENTICE-HALL INTERNATIONAL (UK) LIMITED, *London*
PRENTICE-HALL OF AUSTRALIA PTY. LIMITED, *Sydney*
PRENTICE-HALL CANADA INC., *Toronto*
PRENTICE-HALL HISPANOAMERICANA, S.A., *Mexico*
PRENTICE-HALL OF INDIA PRIVATE LIMITED, *New Delhi*
PRENTICE-HALL OF JAPAN, INC., *Tokyo*
SIMON & SCHUSTER ASIA PTE. LTD., *Singapore*
EDITORA PRENTICE-HALL DO BRASIL, LTDA., *Rio de Janeiro*

Preface

This book covers graph algorithms, pure graph theory, and applications of graph theory to computer systems. The algorithms are presented in a clear algorithmic style, often with considerable attention to data representation, although no extensive background in either data structures or programming is needed. In addition to the classical graph algorithms, many new random and parallel graph algorithms are included. Algorithm design methods, such as divide and conquer, and search tree techniques are emphasized. There is an extensive bibliography, and many exercises. The book is appropriate as a text for both undergraduate and graduate students in engineering, mathematics or computer science, and should be of general interest to professionals in these fields as well.

Chapter 1 introduces the elements of graph theory and algorithmic graph theory. It covers the representations of graphs, basic topics like planarity, matching, hamiltonicity, regular and eulerian graphs, from theoretical, algorithmic, and practical perspectives. Chapter 2 overviews different algorithmic design techniques, such as backtracking, recursion, randomization, greedy and geometric methods, and approximation, and illustrates their application to various graph problems.

Chapter 3 covers the classical shortest-path algorithms, an algorithm for shortest paths on euclidean graphs, and the Fibonacci heap implementation of Dijkstra's algorithm. Chapter 4 presents the basic results on trees and acyclic digraphs, a minimum spanning tree algorithm based on Fibonacci heaps, and includes many applications, such as register allocation, deadlock avoidance, and merge and search trees.

Chapter 5 gives an especially thorough introduction to depth-first search and the classical graph structure algorithms based on depth first search, such as block and strong component detection. Chapter 6 introduces both the theory of connectivity and network flows and shows how connectivity and diverse routing problems may be solved using flow techniques. Applications to reliable routing in unreliable networks and to multiprocessor scheduling are given.

Chapters 7 and 8 introduce coloring, matching, vertex and edge covers, and allied concepts. Applications to secure (zero-knowledge) communication, the design of deadlock free systems, and optimal parallel algorithms are given. The Edmonds matching algorithm, introduced for bipartite graphs in Chapter 1, is presented here in its general form.

Chapter 9 presents a variety of parallel algorithms on different architectures, such as systolic arrays, tree processors, and hypercubes, as well as for the shared memory model of computation. Chapter 10 presents the elements of complexity theory, including an introduction to the complexity of random and parallel algorithms.

I greatly appreciate the help given to me in the preparation of this book by a number of graduate students who read and helped correct earlier versions of the manuscript. These include: Krishna Ayala, Jiann-Ru Chiou, Yaw-Nan Duh, Michael Halper, Shun-Hsien Huang, Pankaj Kumar, and Lai-Wu Luo. A State of New Jersey SBR grant provided support during the initial period of the work. I would also like to thank Jim Fegen and Joe Scordato of Prentice Hall for their help in the development and preparation of this book. Finally, I appreciate the last careful reading of the manuscript by Peter and Jimmy, and the constant support I received from my wife, Alice.

此为试读,需要完整PDF请访问: www.ertongbook.com

Contents

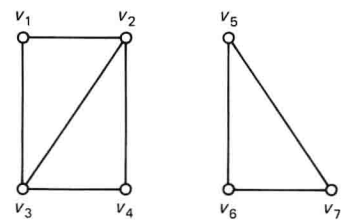
1

Introduction to Graph Theory

1-1 BASIC CONCEPTS

Graphs are mathematical objects that can be used to model networks, data structures, process scheduling, computations, and a variety of other systems where the relations between the objects in the system play a dominant role. We will consider graphs from several perspectives: as mathematical entities with a rich and extensive theory; as models for many phenomena, particularly those arising in computer systems; and as structures which can be processed by a variety of sophisticated and interesting algorithms. Our objective in this section is to introduce the terminology of graph theory, define some familiar classes of graphs, illustrate their role in modelling, and define when a pair of graphs are the same.

Terminology. A *graph* $G(V, E)$ consists of a set V of elements called *vertices* and a set E of unordered pairs of members of V called *edges*. We refer to Figure 1-1 for a geometric presentation of a graph G. The vertices of the graph are shown as points, while the edges are shown as lines connecting pairs of points. The cardinality of V, denoted $|V|$, is called the *order* of G, while the cardinality of E, denoted $|E|$, is called the



$V(G) = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7\}$

$E(G) = \{(v_1, v_2), (v_2, v_4), (v_2, v_3), (v_4, v_3), (v_3, v_1), (v_5, v_6), (v_6, v_7), (v_7, v_5)\}$

Order ($|V(G)|$) = 7

Size ($|E(G)|$) = 8

Number of components = 2

Figure 1-1. Example graph $G(V, E)$.

size of G. When we wish to emphasize the order and size of the graph, we refer to a graph containing p vertices and q edges as a (p, q) graph. Where we wish to emphasize the dependence of the set V of vertices on the graph G, we will write $V(G)$ instead of V, and we use $E(G)$ similarly. The graph consisting of a single vertex is called the *trivial graph*.

We say a vertex u in $V(G)$ is *adjacent to* a vertex v in $V(G)$ if $\{u, v\}$ is an edge in $E(G)$. Following a convention commonly used in graph theory, we will denote the edge between the pair of vertices u and v by (u, v). We call the vertices u and v the *endpoints* of the edge (u, v), and we say the edge (u, v) is *incident with* the vertices u and v. Given a set of vertices S in G, we define the *adjacency set* of S, denoted *ADJ(S)*, as the set of vertices adjacent to some vertex in S. A vertex with no incident edges is said to be *isolated*, while a pair of edges incident with a common vertex are said to be *adjacent*.

The *degree* of a vertex v, denoted by $\deg(v)$, is the number of edges incident with v. If we arrange the vertices of G, $v_1, \ldots, v_n$, so their degrees are in nondecreasing order of magnitude, the sequence $(\deg(v_1), \ldots, \deg(v_n))$ is called the *degree sequence* of G. We denote the *minimum degree* of a vertex in G by $min(G)$ and the *maximum degree* of a vertex in G by $max(G)$. There is a simple relationship between the degrees of a graph and the number of edges.

Theorem (Degree Sum). The sum of the degrees of a graph $G(V, E)$ satisfies

$$\sum_{i=1}^{|V|} \deg(v_i) = 2|E|.$$

The proof follows immediately from the observation that every edge is incident with exactly two vertices. Though simple, this result is frequently useful in extending local estimates of the cost of an algorithm in terms of vertex degrees to global estimates of algorithm performance in terms of the number of edges in a graph.

A *subgraph* S of a graph $G(V, E)$ is a graph $S(V', E')$ such that V' is contained in V, E' is contained in E, and the endpoints of any edge in E' are also in V'. A subgraph is said to *span* its set of vertices. We call S a *spanning subgraph* of G if V' equals V. We call S an *induced subgraph* of G if whenever u and v are in V' and (u, v) is in E, (u, v) is also in E'. We use the notation $G - v$, where v is in $V(G)$, to denote the induced subgraph of G on V-$\{v\}$. Similarly, if V' is a subset of $V(G)$, then $G - V'$ denotes the induced subgraph on $V - V'$. We use the notation $G - (u, v)$, where (u, v) is in $E(G)$, to denote the subgraph $G(V, E - \{(u, v)\})$. If we add a new edge (u, v), where u and v are both in $V(G)$ to $G(V, E)$, we obtain the graph $G(V, E \cup \{(u, v)\})$, which we will denote by $G(V, E) \cup (u, v)$. In general, given a pair of graphs $G(V_1, E_1)$ and $G(V_2, E_2)$, their *union* $G(V_1, E_1) \cup G(V_2, E_2)$ is the graph $G(V_1 \cup V_2, E_1 \cup E_2)$. If $V_1 = V_2$ and E_1 and E_2 are disjoint, the union is called the *edge sum* of $G(V_1, E_1)$ and $G(V_2, E_2)$. The *complement* of a graph $G(V, E)$, denoted by G^c, has the same set of vertices as G, but a pair of vertices are adjacent in the complement if and only if the vertices are not adjacent in G.

We define a *path* from a vertex u in G to a vertex v in G as an alternating sequence of vertices and edges,

$$v_1, e_1, v_2, e_2, \ldots, e_{k-1}, v_k,$$

where $v_1 = u$, $v_k = v$, all the vertices and edges in the sequence are distinct, and successive vertices v_i and v_{i+1} are endpoints of the intermediate edge e_i. If we relax the

此为试读，需要完整PDF请访问：www.ertongbook.com

definition to allow repeating vertices in the sequence, we call the resulting structure a *trail*. If we relax the definition further to allow both repeating edges and vertices, we call the resulting structure a *walk*. If we relax the definition of a path to allow the first and last vertices (only) to coincide, we call the resulting closed path a *cycle*. A graph consisting of a cycle on n vertices is denoted by $C(n)$. If the first and last vertices of a trail coincide, we call the resulting closed trail a *circuit*. The *length* of a path, trail, walk, cycle, or circuit is its number of edges.

We say a pair of vertices u and v in a graph are *connected* if and only if there is a path from u to v. We say a graph G is *connected* if and only if every pair of vertices in G are connected. We call a connected induced subgraph of G of maximal order a *component* of G. Thus, a connected graph consists of a single component. The graph in Figure 1-1 has two components. A graph that is not connected is said to be *disconnected*. If all the vertices of a graph are isolated, we say the graph is *totally disconnected*.

A vertex whose removal increases the number of components in a graph is called a *cut-vertex*. An edge whose removal does the same thing is called a *bridge*. A graph with no cut-vertex is said to be *nonseparable* (or *biconnected*). A maximal nonseparable subgraph of a graph is called a *block* (*biconnected component* or *bicomponent*). In general, the *vertex connectivity* (*edge connectivity*) of a graph is the minimum number of vertices (edges) whose removal results in a disconnected or trivial graph. We call a graph G *k-connected* or *k-vertex connected* (*k-edge connected*) if the vertex (edge) connectivity of G is at least k.

If G is connected, the path of least length from a vertex u to a vertex v in G is called the *shortest path* from u to v, and its length is called the *distance* from u to v. The *eccentricity* of a vertex v is defined as the distance from v to the most distant vertex from v. A vertex of minimum eccentricity is called a *center*. The eccentricity of a center of G is called the *radius* of G, and the maximum eccentricity among all the vertices of G is called the *diameter* of G. We can define the *n*th *power* of a connected graph $G(V, E)$, denoted by G^n as follows: $V(G^n) = V(G)$, and an edge (u, v) is in $E(G^n)$ if and only if the distance from u to v in G is at most n. G^2 and G^3 are called the *square* and *cube* of G, respectively.

If we impose directions on the edges of a graph, interpreting the edges as ordered rather than unordered pairs of vertices, we call the corresponding structure a *directed graph* or *digraph*. In contrast and for emphasis, we will sometimes refer to a graph as an *undirected graph*. We will follow the usual convention in graph theory of denoting an edge from a vertex u to a vertex v by (u, v), leaving it to the context to determine whether the pair is to be considered ordered (directed) or not (undirected). The first vertex u is called the *initial vertex* or *initial endpoint* of the edge, and the second vertex is called the *terminal vertex* or *terminal endpoint* of the edge. If G is a digraph and (u, v) an edge of G, we say the initial vertex u is *adjacent to* v, and the terminal vertex v is *adjacent from* u. We call the number of vertices adjacent to v the *in-degree* of v, denoted indeg(v), and the number of vertices adjacent from v the *out-degree* of v, denoted outdeg(v). The Degree Sum Theorem for graphs has the obvious digraph analog.

Theorem (Digraph Degree Sum). Let $G(V, E)$ be a digraph; then

$$\sum_{i=1}^{|V|} \text{indeg}(v_i) = \sum_{i=1}^{|V|} \text{outdeg}(v_i) = |E|.$$

Generally, the terms we have defined for undirected graphs have straightforward analogs for directed graphs. For example, the paths, trails, walks, cycles, and circuits of undirected graphs are defined similarly for directed graphs, with the obvious refinement that pairs of successive vertices of the defining sequences must determine edges of the digraph. That is, if the defining sequence of the directed walk (path, etc.) includes a subsequence v_i, e_i, v_{i+1}, then e_i must equal the directed edge (v_i, v_{i+1}). If we relax this restriction and allow e_i to equal either (v_i, v_{i+1}) or (v_{i+1}, v_i), we obtain a *semiwalk* (*semipath, semicycle,* and so on).

A digraph $G(V, E)$ is called *strongly connected,* if there is a (directed) path between every pair of vertices in G. A vertex u is said to be *reachable from* a vertex v in G, if there is a directed path from v to u in G. The digraph obtained from G by adding the edge (v, u) between any pair of vertices v and u in G whenever u is reachable from v (and (v, u) is not already in $E(G)$) is called the *transitive closure* of G.

There are several other common generalizations of graphs. For example, in a *multigraph,* we allow more than one edge between a pair of vertices. In contrast, an ordinary graph that does not allow parallel edges is sometimes called a *simple graph.* In a *loop graph,* both endpoints of an edge may be the same, in which case such an edge is called a *loop* (or *self-loop*). If we allow both undirected and directed edges, we obtain a so-called *mixed graph.*

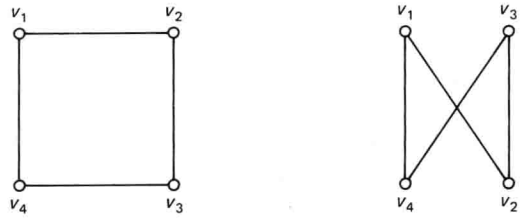
Special graphs. Various special graphs occur repeatedly in practice. We will introduce the definitions of some of these here, and examine them in more detail in later sections.

A graph containing no cycles is called an *acyclic graph.* A directed graph containing no directed cycles is called an *acyclic digraph,* or sometimes a Directed Acyclic Graph (DAG). Perhaps the most commonly encountered special undirected graph is the *tree,* which we define as a connected, acyclic graph. An arbitrary acyclic graph is called a *forest.* Thus, the components of a forest are trees. We will consider trees and acyclic digraphs in Chapter 4.

A graph of order N in which every vertex is adjacent to every other vertex is called a *complete graph,* and is denoted by $K(N)$. Every vertex in a complete graph has the same full degree. More generally, a graph in which every vertex has the same, not necessarily full, degree is called a *regular graph.* If the common degree is r, the graph is called *regular of degree r.*

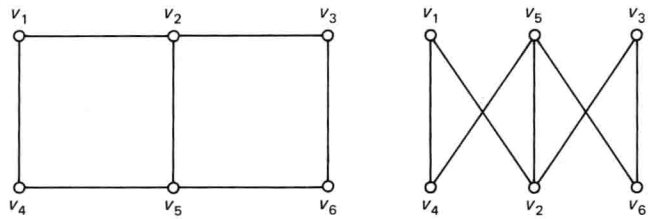
A graph that contains a cycle which spans its vertices is called a *hamiltonian graph.* These graphs are the subject of an extensive literature revolving around their theoretical properties and the algorithmic difficulties involved in efficiently recognizing them. A graph that contains a circuit which spans its edges is called an *eulerian graph.* Unlike hamiltonian graphs, eulerian graphs are easy to recognize. They are merely the connected graphs all of whose degrees are even. We will consider them later in this chapter.

A graph $G(V, E)$ (or $G(V_1, V_2, E)$) is called a *bipartite graph* or *bigraph* if its vertex set V is the disjoint union of sets V_1 and V_2, and every edge in E has the form (v_1, v_2), where $v_1 \in V_1$ and $v_2 \in V_2$. A *complete bipartite graph* is a bigraph in which every vertex in V_1 is adjacent to every vertex in V_2. A complete bigraph depends only on the cardinalities M and N of V_1 and V_2 respectively, and so is denoted by $K(M, N)$. Generally, we say a graph $G(V, E)$ or $G(V_1, \ldots, V_k, E)$ is *k-partite* if the vertex set V is the union of k disjoint sets $V_1, \ldots, V_k$, and every edge in E is of the form (v_i, v_j), for



$V_1 = \{v_1, v_3\}$
$V_2 = \{v_2, v_4\}$

(a) Cyclic and bipartite presentations of $C(4)$.



$V_1 = \{v_1, v_3, v_5\}$
$V_2 = \{v_2, v_4, v_6\}$

(b) Different presentations of a bipartite graph $G(V_1, V_2, E)$.

Figure 1-2. Bipartite and nonbipartite graphs.

vertices $v_i \in V_i$ and $v_j \in V_j$, V_i and V_j distinct. A *complete k-partite graph* is defined similarly. We refer to Figure 1-2 for an example.

Graphs as models. We will describe many applications of graphs in later chapters. The following are illustrative.

Assignment Problem. Bipartite graphs can be used to model problems where there are two kinds of entities, and each entity of one kind is related to a subset of entities of the other. For example, one set may be a set V_1 of employees and the other a set V_2 of tasks the employees can perform. If we assume each employee can perform some subset of the tasks and each task can be performed by some subset of the employees, we can model this situation by a bipartite graph $G(V_1, V_2, E)$, where there is an edge between v_1 in V_1 and v_2 in V_2 if and only if employee v_1 can perform task v_2.

We could then consider such problems as determining the smallest number of employees who can perform all the tasks, which is equivalent in graph-theoretic terms to asking for the smallest number of vertices in V_1 that together are incident to all the vertices in V_2, a so-called covering problem (see Chapter 8). Or, we might want to know how to assign the largest number of tasks, one task per employee and one employee per task, a problem which corresponds graph-theoretically to finding a maximum size regular subgraph of degree one in the model graph, the so-called *matching problem.* Section 1-3 gives a condition for the existence of matchings spanning V_1, and Section 1-5 and Chapters 8 and 9 give algorithms for finding maximum matchings on graphs.

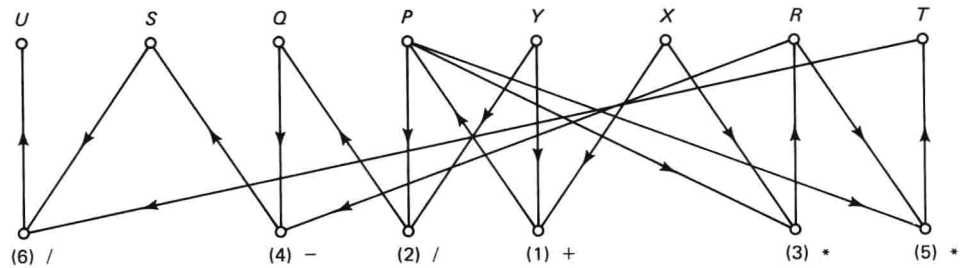
Data Flow Diagrams. We can use directed bipartite graphs to model the flow of data between the operators in a program, employing a *data flow diagram*. For this diagram, we let the data (program variables) correspond to the vertices of one part V_1 of the bigraph $G(V_1, V_2)$, while the operators correspond to the other part V_2. We include an edge from a datum to an operator vertex if the corresponding datum is an input to the corresponding operator. Conversely, we include an edge from an operator vertex to a datum, if the datum is an output of the operator. A bipartite representation of the data flow of a program is shown in Figure 1-3.

A data flow diagram helps in analyzing the intrinsic parallelism of a program. For example, the length of the longest path in a data flow diagram determines the shortest completion time of the program, provided the maximum amount of parallelism is used. In the example, the path $X(+:1)P(\text{DIV}:2)Q(-:4)S(\text{DIV}:6)U$ is a longest path; so the program cannot possibly be completed in less than four steps (sequential operations). A minimum length parallel execution schedule is $\{1\}$, $\{2, 3\}$ concurrently, $\{4, 5\}$ concurrently, and $\{6\}$, where the numbers indicate the operators.

Graph isomorphism. For any mathematical objects, the question of their equality or identity is fundamental. For example, a pair of fractions (which may look different) are the same if their difference is zero. Just like fractions, a pair of graphs may also look different but actually have the same structure. Thus, the graphs in Figure 1-4 are nominally different because the vertices v_1 and v_2 are adjacent in one of the graphs, but not in the other. However, if we ignore the names or labels on the vertices, the graphs are clearly structurally identical. Structurally identical graphs are called Isomorphic, from the Greek words *iso* for "same" and *morph* for "form." Formally, we define a pair of graphs $G_1(V, E)$ and $G_2(V, E)$ to be *isomorphic* if there is a one-to-one correspondence (or mapping) M between $V(G_1)$ and $V(G_2)$ such that u and v

(1) $P = X + Y$
(2) $Q = Y$ div P
(3) $R = X * P$
(4) $S = R - Q$
(5) $T = R * P$
(6) $U = T$ div S

(a) Code sequence.



(b) Data flow diagram for (a).

Figure 1-3. Bipartite data flow model.

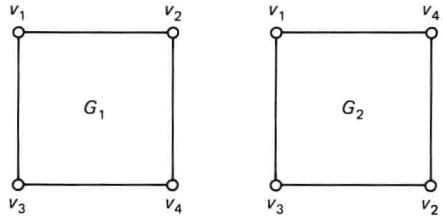


Figure 1-4. Isomorphic graphs.

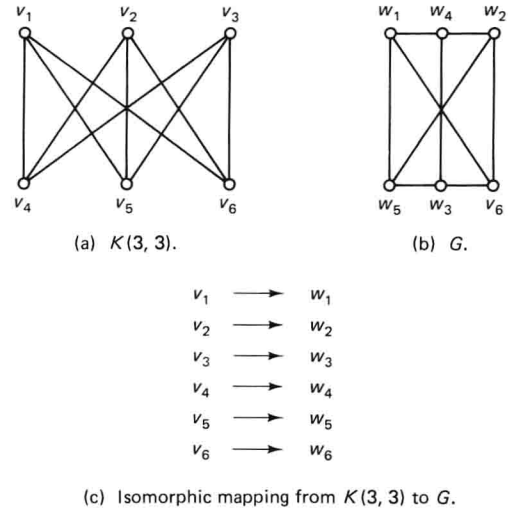


Figure 1-5. A pair of isomorphic graphs.

are adjacent in G_1 if and only if the corresponding vertices $M(u)$ and $M(v)$ are adjacent in G_2. See Figure 1-5 for another example.

To prove a pair of graphs are isomorphic, we need to find an isomorphism between the graphs. The brute force approach is to exhaustively test every possible one-to-one mapping between the vertices of the graphs, until we find a mapping that qualifies as an isomorphism, or determine there is none. See Figure 1-6 for a high-level view of such a search algorithm. Though methodical, this method is computationally infeasible for large graphs. For example, for graphs of order N, there are a priori $N!$ distinct possible 1-1 mappings between the vertices of a pair of graphs; so examining each would be prohibitively costly. Though the most naive search technique can be improved, such as in the backtracking algorithm in Chapter 2, all current methods for the problem are inherently tedious.

A *graphical invariant* (or *graphical property*) is a property associated with a graph $G(V, E)$ that has the same value for any graph isomorphic to G. One may fortui-

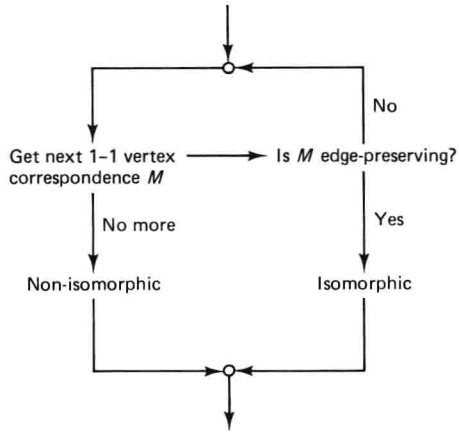
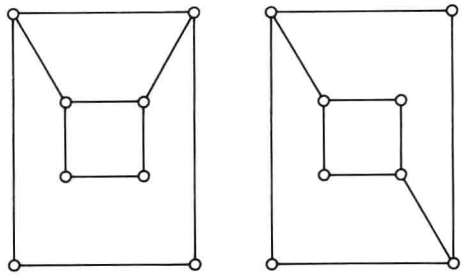
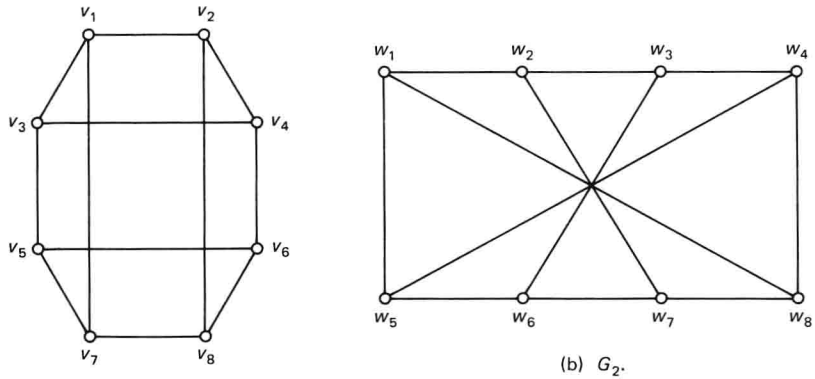


Figure 1-6. Exhaustive search algorithm for isomorphism.



(a) G_1. (b) G_2.

Figure 1-7. G_1 isomorphic to G_2?



(a) G_1. (b) G_2.

Figure 1-8. G_1 isomorphic to G_2?

tously succeed in finding an invariant a pair of graphs do not share, thus establishing their nonisomorphism. Of course, two graphs may agree on many invariants and still be nonisomorphic, although the likelihood of this occurring decreases with the number of their shared properties. The graphs in Figure 1-7 agree on several graphical properties: size, order, and degree sequence. However, the subgraph induced in $G_2(V, E)$ by the vertices of degree 2 is regular of degree 0, while the corresponding subgraph in $G_1(V, E)$ is regular of degree 1. This corresponds to a graphical property the graphs do not share; so the graphs are not isomorphic. The graphs in Figure 1-8 also agree on size, order, and degree sequence. However, $G_1(V, E)$ appears to have cycles only of lengths 4, 6, and 8, while $G_2(V, E)$ has cycles of length 5. One can readily show that G_1 is bipartite while G_2 is not, so these graphs are also nonisomorphic.

1-2 REPRESENTATIONS

There are a variety of standard data structure representations for graphs. Each representation facilitates certain kinds of access to the graph but makes complementary kinds of access more difficult. We will describe the simple static representations first. The linked representations are more complicated, but more economical in terms of storage requirements, and they facilitate dynamic changes to the graphs.

此为试读,需要完整PDF请访问: www.ertongbook.com