Preface

Our objective in writing this book is to present the theory of graphs from an algorithmic viewpoint. We present the graph theory in a rigorous, but informal style and cover most of the main areas of graph theory. The ideas of surface topology are presented from an intuitive point of view. We have also included a discussion on linear programming that emphasizes problems in graph theory. The text is suitable for students in computer science or mathematics programs.

Graph theory is a rich source of problems and techniques for programming and data structure development, as well as for the theory of computing, including NP-completeness and polynomial reduction. This book could be used a textbook for a third or fourth year course on graph algorithms which contains a programming content, or for a more advanced course at the fourth year or graduate level. It could be used in a course in which the programming language is any major programming language (e.g., C, C++, Java). The algorithms are presented in a generic style and are not dependent on any particular programming language. The text could also be used for a sequence of courses like “Graph Algorithms I” and “Graph Algorithms II”. The courses offered would depend on the selection of chapters included. A typical course will begin with Chapters 1, 2, 3, and 4. At this point, a number of options are available. A possible first course would consist of Chapters 1, 2, 3, 4, 6, 8, 9, 10, 11, and 12, and a first course stressing optimization would consist of Chapters 1, 2, 3, 4, 8, 9, 10, 14, 15, and 16. Experience indicates that the students consider these substantial courses. One or two chapters could be omitted for a lighter course. We would like to thank the many people who provided encouragement while we wrote this book, pointed out typos and errors, and gave useful suggestions. In particular, we would like to convey our thanks to Ben Li and John van Rees of the University of Manitoba for proofreading some chapters.

William Kocay
Donald L. Kreher
William Kocay obtained his Ph.D. in Combinatorics and Optimization from the University of Waterloo in 1979. He is currently a member of the Computer Science Department, and an adjunct member of the Mathematics Department, at the University of Manitoba, and a member of St. Paul’s College, a college affiliated with the University of Manitoba. He has published numerous research papers, mostly in graph theory and algorithms for graphs. He was managing editor of the mathematics journal Ars Combinatoria from 1988 to 1997. He is currently on the editorial board of that journal. He has had extensive experience developing software for graph theory and related mathematical structures.

Donald L. Kreher obtained his Ph.D. from the University of Nebraska in 1984. He has held academic positions at Rochester Institute of Technology and the University of Wyoming. He is currently a University Professor of Mathematical Sciences at Michigan Technological University, where he teaches and conducts research in combinatorics and combinatorial algorithms. He has published numerous research papers and is a co-author of the internationally acclaimed text “Combinatorial Algorithms: Generation Enumeration and Search”, CRC Press, 1999. He serves on the editorial boards of two journals. Professor Kreher is the sole recipient of the 1995 Marshall Hall Medal, awarded by the Institute of Combinatorics and its Applications.
3.4.1 An Euler tour algorithm
Exercises
3.5 Notes

4 Trees and Cycles
4.1 Introduction
Exercises
4.2 Fundamental cycles
Exercises
4.3 Co-trees and bonds
Exercises
4.4 Spanning tree algorithms
4.4.1 Prim's algorithm
Data structures
Exercises
4.4.2 Kruskal's algorithm
Data structures and complexity
4.4.3 The Cheriton-Tarjan algorithm
Exercises
4.4.4 Leftist binary trees
Exercises
4.5 Notes

5 The Structure of Trees
5.1 Introduction
5.2 Non-rooted trees
Exercises
5.3 Read's tree encoding algorithm
5.3.1 The decoding algorithm
Exercises
5.4 Generating rooted trees
Exercises
5.5 Generating non-rooted trees
Exercises
5.6 Prüfer sequences
5.7 Spanning trees
5.8 The matrix-tree theorem
Exercises
5.9 Notes

6 Connectivity
6.1 Introduction
Exercises
6.2 Blocks
6.3 Finding the blocks of a graph
Exercises
6.4 The depth-first search
6.4.1 Complexity
Exercises
6.5 Notes

7 Alternating Paths and Matchings
7.1 Introduction
Exercises
7.2 The Hungarian algorithm
7.2.1 Complexity
Exercises
7.3 Perfect matchings and 1-factorizations
Exercises
7.4 The subgraph problem
7.5 Coverings in bipartite graphs
7.6 Tutte’s theorem
    Exercises
7.7 Notes

8 Network Flows
8.1 Introduction
8.2 The Ford-Fulkerson algorithm
    Exercises
8.3 Matchings and flows
    Exercises
8.4 Menger’s theorems
    Exercises
8.5 Disjoint paths and separating sets
    Exercises
8.6 Notes

9 Hamilton Cycles
9.1 Introduction
    Exercises
9.2 The crossover algorithm
9.2.1 Complexity
    Exercises
9.3 The Hamilton closure
    Exercises
9.4 The extended multi-path algorithm
9.4.1 Data structures for the segments
    Exercises
9.5 Decision problems, NP-completeness
    Exercises
9.6 The traveling salesman problem
    Exercises
9.7 The ΔTSP
    Exercises
9.8 Christofides’ algorithm
    Exercises
9.9 Notes

10 Digraphs
10.1 Introduction
10.2 Activity graphs, critical paths
    Exercises
10.3 Topological order
    Exercises
10.4 Strong components
    Exercises
10.4.1 An application to fabrics
    Exercises
10.5 Tournaments
    Exercises
10.6 2-Satisfiability
    Exercises
10.7 Notes

11 Graph Colorings
11.1 Introduction
11.1.1 Intersecting lines in the plane
    Exercises
11.2 Cliques
    Exercises
11.3 Mycielski’s construction
11.4 Critical graphs
    Exercises
11.5 Chromatic polynomials
    Exercises
11.6  Edge colorings  258
11.6.1  Complexity  268
Exercises  269

Page xvii
11.7  NP-completeness  269
11.8  Notes  274

12  Planar Graphs  275
12.1  Introduction  275
12.2  Jordan curves  276
12.3  Graph minors, subdivisions  277
Exercises  282
12.4  Euler’s formula  282
12.5  Rotation systems  284
12.6  Dual graphs  286
12.7  Platonic solids, polyhedra  290
Exercises  291
12.8  Triangulations  292
12.9  The sphere  295
12.10 Whitney’s theorem  297
12.11 Medial digraphs  300
Exercises  301
12.12 The 4-color problem  301
Exercises  305
12.13 Straight-line drawings  305
12.14 Kuratowski’s theorem  309
Exercises  312
12.15 The Hopcroft-Tarjan algorithm  312
12.15.1 Bundles  316
12.15.2 Switching bundles  318
12.15.3 The general Hopcroft-Tarjan algorithm  321
12.16 Notes  325

13  Graphs and Surfaces  327
13.1  Introduction  327
13.2  Surfaces  329
13.2.1 Handles and crosscaps  336
13.2.2 The Euler characteristic and genus of a surface  337
Exercises  340
13.3  Graph embeddings, obstructions  341
13.4  Graphs on the torus  342
Exercises  349
13.4.1 Platonic maps on the torus  349
13.4.2 Drawing torus maps, triangulations  352
Exercises  357
13.5  Graphs on the projective plane  357
13.5.1 The facewidth  364
13.5.2 Double covers  368
Exercises  370
13.6 Embedding algorithms  372
Exercises  381
13.7 Heawood’s map coloring theorem  382
Exercises  384
13.8 Notes  385

14  Linear Programming  387
14.1  Introduction  387
14.1.1 A simple example  387
Graphs and Their Complements

1.1 Introduction

The diagram in Figure 1.1 illustrates a graph. It is called the graph of the cube. The edges of the geometric cube correspond to the line segments connecting the nodes in the graph, and the nodes correspond to the corners of the cube where the edges meet. They are the vertices of the cube.
FIGURE 1.1
The graph of a cube
This diagram is drawn so as to resemble a cube, but if we were to rearrange it, as in Figure 1.2, it would still be the graph of the cube, although it would no longer look like a cube. Thus, a graph is a graphical representation of a relation in which edges connect pairs of vertices.

DEFINITION 1.1: A simple graph $G$ consists of a vertex set $V(G)$ and an edge set $E(G)$, where each edge is a pair \{u, v\} of vertices $u, v \in V(G)$.

We denote the set of all pairs of a set $V$ by $\binom{V}{2}$. Then $E(G) \subseteq \binom{V(G)}{2}$. In

FIGURE 1.2
The graph of the cube
In the example of the cube, $V(G) = \{1, 2, 3, 4, 5, 6, 7, 8\}$, and $E(G) = \{12, 23, 34, 14, 15, 26, 37, 48, 56, 67, 78, 58\}$, where we have used the shorthand notation $uv$ to stand for the pair $\{u, v\}$. If $u, v \in V(G)$, then $u \rightarrow v$ means that $u$ is joined to $v$ by an edge. We say that $u$ and $v$ are adjacent. We use this notation to remind us of the linked list data structure that we will use to store a graph in the computer. Similarly, $u \not\rightarrow v$ means that $u$ is not joined to $v$. We can also express these relations by writing $uv \in E(G)$ or $uv \not\in E(G)$, respectively. Note that in a simple graph if $u \rightarrow v$, then $v \rightarrow u$. If $u$ is adjacent to each of $u_1, u_2, \ldots, u_k$, then we write $u \rightarrow \{u_1, u_2, \ldots, u_k\}$.

These graphs are called simple graphs because each pair $u, v$ of vertices is joined by at most one edge. Sometimes we need to allow several edges to join the same pair of vertices. Such a graph is also called a multigraph. An edge can then no longer be defined as a pair of vertices, (or the multiple edges would not be distinct), but to each edge there still corresponds a pair $\{u, v\}$. We can express this formally by saying that a graph $G$ consists of a vertex set $V(G)$, an edge set $E(G)$, and a correspondence $\psi: E(G) \rightarrow \binom{V(G)}{2}$. Given an edge $e \in E(G)$, $\psi(e)$ is a pair $\{u, v\}$ which are the endpoints of $e$. Different edges can then have the same endpoints. We shall use simple graphs most of the time, which is why we prefer the simpler definition, but many of the theorems and techniques will apply to multigraphs as well.

This definition can be further extended to graphs with loops as well. A loop is an edge in which both endpoints are equal. We can include this in the general definition of a graph by making the mapping $\psi: E(G) \rightarrow \binom{V(G)}{2} \cup V(G)$. An edge $e \in E(G)$ for which $\psi(e) = u \in V(G)$ defines a loop. Figure 1.2 shows a graph with multiple edges and loops. However, we shall use simple graphs most of the time, so that an
edge will be considered to be a pair of vertices. The number of vertices of a graph $G$ is denoted $|G|$. It is called the order of $G$.

**FIGURE 1.3**
A multigraph

The number of edges is $\varepsilon(G)$. If $G$ is simple, then obviously $\varepsilon(G) \leq \binom{|G|}{2}$, since $E(G) \subseteq \binom{V(G)}{2}$. We shall often use node or point as synonyms for vertex.

Many graphs have special names. The complete graph $K_n$ is a simple graph with $|K_n| = n$ and $\varepsilon = \binom{n}{2}$. The empty graph $\overline{K_n}$ is a graph with $|\overline{K_n}| = n$ and $\varepsilon = 0$. $\overline{K_n}$ is the complement of $K_n$.

**FIGURE 1.4**
The complete graph $K_5$

**DEFINITION 1.2**: Let $G$ be a simple graph. The complement of $G$ is $\overline{G}$, where $V(\overline{G}) = V(G)$ and $E(\overline{G}) = \binom{V(G)}{2} \setminus E(G)$. $E(\overline{G})$ consists of all those pairs $uv$ which are not edges of $G$. Thus, $uv \in E(\overline{G})$ if and only if $uv \notin E(G)$. Figure 1.5 shows a graph and its complement.
FIGURE 1.5
A graph and its complement
Figure 1.6 shows another graph and its complement. Notice that in this case, when \( \overline{G} \) is redrawn, it looks identical to \( G \).

In a certain sense, this \( G \) and \( \overline{G} \) are the same graph. They are not equal, since \( E(G) \neq E(\overline{G}) \), but it is clear that they have the same structure. If two graphs have the same structure, then they can only differ in the names of the vertices. Therefore, we can rename the vertices of one to make it exactly equal to the other graph. In the example above, we can rename the vertices of \( G \) by the mapping \( \theta \) given by

\[
\begin{array}{cccccc}
  k & 1 & 2 & 3 & 4 & 5 \\
  \theta(k) & 1 & 3 & 5 & 2 & 4 \\
\end{array}
\]

then \( \theta(G) \) would equal \( \overline{G} \). This kind of equivalence of graphs is known as isomorphism. Observe that a one-to-one mapping \( \theta \) of the vertices of a graph \( G \) can be extended to a mapping of the edges of \( G \) by defining \( \theta(\{u,v\}) = \{\theta(u),\theta(v)\} \).

FIGURE 1.6
Another graph and its complement

DEFINITION 1.3: Let \( G \) and \( H \) be simple graphs. \( G \) and \( H \) are isomorphic if there is a one-to-one correspondence \( \theta: V(G) \rightarrow V(H) \) such that \( \theta(E(G)) = E(H) \), where \( \theta(E(G)) = \{\theta(uv) : uv \in E(G)\} \).

We write \( G \cong H \) to denote isomorphism. If \( G \cong H \), then \( uv \in E(G) \) if and only if \( \theta(uv) \in E(H) \). One way to determine whether \( G \cong H \) is to try and redraw \( G \) so as to make it look identical to \( H \). We can then read off the mapping \( \theta \) from the diagram. However, this is limited to small graphs. For example, the two graphs \( G \) and \( H \) shown in Figure 1.7 are isomorphic, since the drawing of \( G \) can be transformed into \( H \) by first moving vertex 2 to the bottom of the diagram, and then moving vertex 5 to the top. Comparing the two diagrams then gives the mapping

\[
\begin{array}{cccccccc}
  k & 1 & 2 & 3 & 4 & 5 & 6 \\
  \theta(k) & 6 & 4 & 2 & 5 & 1 & 3 \\
\end{array}
\]

as an isomorphism.
FIGURE 1.7
Two isomorphic graphs
It is usually more difficult to determine when two graphs $G$ and $H$ are not isomorphic than to find an
isomorphism when they are isomorphic. One way is to find a portion of $G$ that cannot be part of $H$. For
example, the graph $H$ of Figure 1.7 is not isomorphic to the graph of the prism, which is illustrated in Figure
1.8, because the prism contains a triangle, whereas $H$ has no triangle. A subgraph of a graph $G$ is a graph $X$
such that $V(X) \subseteq V(G)$ and $E(X) \subseteq E(G)$. If $\theta: G \to H$ is a possible isomorphism, then $\theta(X)$ will be a
subgraph of $H$ which is isomorphic to $X$. A subgraph $X$ is an induced subgraph if for every $u,
v \in V(X) \subseteq V(G), uv \in E(X)$ if and only if $uv \in E(G)$.
The degree of a vertex $u \in V(G)$ is $\text{DEG}(u)$, the number of edges which contain $u$. If $k = \text{DEG}(u)$ and
$u \to \{u_1, u_2, \ldots, u_k\}$, then $\theta(u) \to$

FIGURE 1.8
The graph of the prism
$\{\theta(u_1), \theta(u_2), \ldots, \theta(u_k)\}$, so that $\text{DEG}(u) = \text{DEG}(\theta(u))$. Therefore a necessary condition for $G$ and $H$ to be
isomorphic is that they have the same set of degrees. The examples of Figures 1.7 and 1.8 show that this is
not a sufficient condition.

In Figure 1.6, we saw an example of a graph $G$ that is isomorphic to its complement. There are many such
graphs.

DEFINITION 1.4: A simple graph $G$ is self-complementary if $G \cong \overline{G}$.

LEMMA 1.1 If $G$ is a self-complementary graph, then $|G| \equiv 0 \text{ or } 1 \pmod{4}$.

PROOF If $G \cong \overline{G}$, then $\varepsilon(G) = \varepsilon(\overline{G})$. But $E(\overline{G}) = \left(V(\overline{G})\right) \setminus E(G)$, so that $\varepsilon(\overline{G}) = \left(|G|\right) - \varepsilon(G) = \varepsilon(G)$, so

$\varepsilon(G) = \frac{1}{2}\left(|G|\right) = |G|||G| - 1|/4$. Now $|G|$ and $|G| - 1$ are consecutive integers, so that one of them is odd.

Therefore $|G| \equiv 0 \pmod{4}$ or $|G| \equiv 1 \pmod{4}$.

So possible orders for self-complementary graphs are $4, 5, 8, 9, 12, 13, \ldots, 4k, 4k + 1, \ldots$

Exercises
1.1.1 The four graphs on three vertices in Figure 1.9 have 0, 1, 2, and 3 edges, respectively. Every graph on
three vertices is isomorphic to one of these four. Thus, there are exactly four different isomorphism types of
graph on three vertices.

Find all the different isomorphism types of graph on 4 vertices (there are 11 of them). Hint: Adding an edge
to a graph with $\varepsilon = m$, gives a graph with $\varepsilon = m + 1$. Every graph with $\varepsilon = m + 1$ can be obtained in this way.

Table 1.1 shows the number of isomorphism types of graph up to 10 vertices.

Page 6
FIGURE 1.9
Four graphs on three vertices

TABLE 1.1
Graphs up to 10 vertices

<table>
<thead>
<tr>
<th>n</th>
<th>No. graphs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>11</td>
</tr>
<tr>
<td>5</td>
<td>34</td>
</tr>
<tr>
<td>6</td>
<td>156</td>
</tr>
<tr>
<td>7</td>
<td>1,044</td>
</tr>
<tr>
<td>8</td>
<td>12,346</td>
</tr>
<tr>
<td>9</td>
<td>247,688</td>
</tr>
<tr>
<td>10</td>
<td>12,005,188</td>
</tr>
</tbody>
</table>

1.1.2 Determine whether the two graphs shown in Figure 1.10 are isomorphic to each other or not. If they are isomorphic, find an explicit isomorphism.

FIGURE 1.10
Two graphs on eight vertices

1.1.3 Determine whether the three graphs shown in Figure 1.11 are isomorphic to each other or not. If they are isomorphic, find explicit isomorphisms.

1.1.4 Find a self-complementary graph on four vertices.

1.1.5 Figure 1.6 illustrates a self-complementary graph, the pentagon, with five vertices. Find another self-complementary graph on five vertices.

1.1.6 We have seen that the pentagon is a self-complementary graph. Let \( G \) be the pentagon shown in Figure 1.6, with \( V(G) = \{u_1, u_2, u_3, u_4, u_5\} \). Notice that
FIGURE 1.11
Three graphs on 10 vertices
\[ \theta(u_1)(u_2, u_3, u_5, u_4) \] is a permutation which maps \( G \) to \( \overline{G} \); that is, \( \theta(G) = \overline{G} \), and \( \theta(\overline{G}) = G \). \( \theta \) is called a complementing permutation. Since \( u_2u_3 \in E(G) \), it follows that \( \theta(u_2u_3) = u_3u_2 \in E(\overline{G}) \). Consequently, \( \theta(u_3u_2) = u_2u_3 \in E(G) \) again. Applying \( \theta \) twice more gives \( \theta(u_3u_4) = u_4u_3 \in E(\overline{G}) \) and \( \theta(u_4u_3) = u_3u_4 \), which is where we started. Thus, if we choose any edge \( uiuj \) and successively apply \( \theta \) to it, we alternately get edges of \( G \) and \( \overline{G} \). It follows that the number of edges in the sequence so-obtained must be even. Use the permutation \( (1,2,3,4) (5,6,7,8) \) to construct a self-complementary graph on eight vertices.

1.1.8 Can the permutation \( (1, 2, 3, 4, 5) (6, 7, 8) \) be used as a complementing permutation? Can \( (1, 2, 3, 4, 5, 6) (7, 8) \) be? Prove that the only requirement is that every sequence of edges obtained by successively applying \( \theta \) be of even length.

1.1.7 Can the permutation \( (1, 2, 3, 4, 5) \) be used as a complementing permutation? Can \( (1, 2, 3, 4, 5, 6) \) be used as a complementing permutation? Discover what condition this cycle structure must satisfy, and prove it both necessary and sufficient for \( \theta \) to be a complementing permutation.

1.2 Degree sequences

THEOREM 1.2 For any simple graph \( G \) we have
\[ \sum_{u \in V(G)} \text{DEG}(u) = 2\varepsilon(G). \]

Page 9

PROOF An edge \( uu \) has two endpoints. Therefore each edge will be counted twice in the summation, once for \( u \) and once for \( v \).

We use \( \delta(G) \) to denote the minimum degree of \( G \); that is, \( \delta(G) = \text{MIN}\{\text{DEG}(u) \mid u \in V(G)\} \). \( \Delta(G) \) denotes the maximum degree of \( G \). By Theorem 1.2, the average degree equals \( 2\varepsilon/|G| \), so that \( \delta \leq 2\varepsilon/|G| \leq \Delta \).

COROLLARY 1.3 The number of vertices of odd degree is even.

PROOF Divide \( V(G) \) into \( V_{\text{odd}}=\{u \mid \text{DEG}(u) \text{ is odd}\} \), and \( V_{\text{even}}=\{u \mid \text{DEG}(u) \text{ is even}\} \). Then
\[ 2\varepsilon = \sum_{u \in V_{\text{odd}}} \text{DEG}(u) + \sum_{u \in V_{\text{even}}} \text{DEG}(u). \]
Clearly \( 2\varepsilon \) and \( \sum_{u \in V_{\text{even}}} \text{DEG}(u) \) are both even. Therefore, so is \( \sum_{u \in V_{\text{odd}}} \text{DEG}(u) \), which means that \( |V_{\text{odd}}| \) is even.

DEFINITION 1.5: A graph \( G \) is a regular graph if all vertices have the same degree. \( G \) is \( k \)-regular if it is regular, of degree \( k \).

For example, the graph of the cube (Figure 1.1) is 3-regular.

LEMMA 1.4 If \( G \) is simple and \( |G| \geq 2 \), then there are always two vertices of the same degree.

PROOF In a simple graph, the maximum degree \( \Delta \leq |G|-1 \). If all degrees were different, then they would be \( 0, 1, 2, \ldots, |G|-1 \). But degree 0 and degree \( |G|-1 \) are mutually exclusive. Therefore there must be two vertices of the same degree.

Let \( V(G) = \{u_1, u_2, \ldots, u_n\} \). The degree sequence of \( G \) is
\[ \text{DEG}(G) = (\text{DEG}(u_1), \text{DEG}(u_2), \ldots, \text{DEG}(u_n)) \]
where the vertices are ordered so that
\[ \text{DEG}(u_1) \geq \text{DEG}(u_2) \geq \ldots \geq \text{DEG}(u_n). \]
Sometimes it’s useful to construct a graph with a given degree sequence. For example, can there be a simple graph with five vertices whose degrees are \( (4, 3, 3, 2, 1) \)? Since there are three vertices of odd degree, Corollary 1.3 tells us that there is no such graph. We say that a sequence
\[ D = (d_1, d_2, \ldots, d_n), \]
is graphic if
\[ d_1 \geq d_2 \geq \ldots \geq d_n, \]
and there is a simple graph \( G \) with \( \text{DEG}(G) = D \). So \( (2, 2, 2, 1) \) and \( (4, 3, 3, 2, 1) \) are not graphic, whereas \( (2, 2, 1, 1), (4, 3, 2, 2, 1) \) and \( (2, 2, 2, 2, 2) \) clearly are.

Problem 1.1: Graphic

Instance: a sequence \( D = (d_1, d_2, \ldots, d_n) \).

Question: is \( D \) graphic?

Find: a graph \( G \) with \( \text{DEG}(G) = D \), if \( D \) is graphic.

For example, \( (7, 6, 5, 4, 3, 2, 3) \) is not graphic; for any graph \( G \) with this degree sequence has \( \Delta(G) = |G| = 7, \)
which is not possible in a simple graph. Similarly, (6, 6, 5, 4, 3, 3, 1) is not graphic; here we have \( \Delta(G)=6 \), \( |G|=7 \) and \( \delta(G)=1 \). But since two vertices have degree \( |G|-1=6 \), it is not possible to have a vertex of degree one in a simple graph with this degree sequence.

When is a sequence graphic? We want a construction which will find a graph \( G \) with \( \text{DEG}(G)=D \), if the sequence \( D \) is graphic.

One way is to join up vertices arbitrarily. This does not always work, since we can get stuck, even if the sequence is graphic. The following algorithm always produces a graph \( G \) with \( \text{DEG}(G)=D \), if \( D \) is graphic.

**procedure** \( \text{GRAPHGEN}(D) \)

Create vertices \( u_1, u_2, \ldots, u_n \)

**comment:** upon completion, \( u_i \) will have degree \( D[i] \)

\( \text{graphic} \leftarrow \text{false} \) "assume not graphic"

\( i \leftarrow 1 \)

**while** \( D[i] > 0 \)

\[
\begin{align*}
&\begin{cases}
&k \leftarrow D[i] \\
&\text{if there are at least } k \text{ vertices with DEG } > 0 \\
&\text{then} \\
&\text{join } u_i \text{ to the } k \text{ vertices of largest degree} \\
&\text{decrease each of these degrees by } 1 \\
&D[i] \leftarrow 0 \\
&\text{comment: vertex } u_i \text{ is now completely joined}
\end{cases} \\
&\text{else exit} \quad "u_i \text{ cannot be joined}"
\end{align*}
\]

\( i \leftarrow i + 1 \)

\( \text{graphic} \leftarrow \text{true} \)

This uses a reduction. For example, given the sequence

\( D=(3, 3, 3, 3, 3) \)

the first vertex will be joined to the three vertices of largest degree, which will then reduce the sequence to

\( (*, 3, 3, 2, 2, 2) \), since the vertex marked by an asterisk is now completely joined, and three others have had their degree reduced by 1. At the next stage, the first remaining vertex will be joined to the three vertices of largest degree, giving a new sequence \( (*, *, 2, 2, 1, 1) \). Two vertices are now completely joined. At the next step, the first remaining vertex will be joined to two vertices, leaving \( (*, *, *, 1, 1, 0) \). The next step joins the two remaining vertices with degree one, leaving a sequence \( (*, *, *, *, 0, 0) \) of zeroes, which we know to be graphic.

In general, given the sequence

\( D=(d_1, d_2, \ldots, d_n) \)

where

\[
d_1 \geq d_2 \geq \ldots \geq d_n,
\]

the first vertex \( u_1 \) has been deleted. We now do the same calculation, using \( D' \) in place of \( D \).

An excellent data structure for representing the graph \( G \) for this problem is to have an *adjacency list* for each vertex \( v \in V(G) \). The adjacency list for a vertex \( v \in V(G) \) is a linked list of the vertices adjacent to \( u \). Thus it is a data structure in which the vertices adjacent to \( u \) are arranged in a linear order. A node \( x \) in a linked list has two fields: \( \text{data}(x) \), and \( \text{next}(x) \).
Given a node $x$ in the list, $\text{data}(x)$ is the data associated with $x$ and $\text{next}(x)$ points to the successor of $x$ in the list or $\text{next}(x) = \text{NIL}$ if $x$ has no successor. We can insert data $u$ into the list pointed to by $L$ with procedure \textsc{ListInsert()}, and the first node on list $L$ can be removed with procedure \textsc{ListRemoveFirst()}. 

\begin{verbatim}
procedure \textsc{ListInsert}(L, u)
\begin{algorithmic}
  \State \textbf{procedure} NEWNODE()
  \State data\langle x \rangle \leftarrow u
  \State next\langle x \rangle \leftarrow L
  \State L \leftarrow x
end\algorithmic
\end{verbatim}

\begin{verbatim}
procedure \textsc{ListRemoveFirst}(L)
\begin{algorithmic}
  \State x \leftarrow L
  \State L \leftarrow \text{next}(x)
  \State \text{Freenode}(x)
end\algorithmic
\end{verbatim}

We use an array $\text{AdjList}[\cdot]$ of linked lists to store the graph. For each vertex $v \in V(G)$, $\text{AdjList}[v]$ points to the head of the adjacency lists for $v$. This data structure is illustrated in Figure 1.12.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure112.png}
\caption{Adjacency lists of a graph}
\end{figure}

We can use another array of linked lists, $\text{Pts}[k]$, being a linked list of the vertices $u_i$ whose degree-to-be $d_i=k$. With this data structure, Algorithm 1.2.1 can be written as follows:

\begin{algorithm}[h]
\caption{GRAPHGEN(D)}
\begin{algorithmic}
  \State \textbf{comment:} Assume $D$ is not graphic.
  \State \textbf{comment:} Create and initialize the linked lists $\text{Pts}[k]$.
  \State graphic \leftarrow \text{false}
  \For {$k \leftarrow 0$ \textbf{to} $n-1$} \text{Pts}[k] \leftarrow \text{NIL}
  \For {$k \leftarrow 1$ \textbf{to} $n$} \textsc{ListInsert}(\text{Pts}[D[k]], k)
  \Comment{Begin with vertex of largest degree.}
  \For {$k \leftarrow n$ \textbf{downto} 1}
  \DoWhile {\text{Pts}[k] \neq \text{NIL}}
end\algorithmic
\end{algorithm}
\textbf{comment:} These points are to have degree $k$.
\begin{verbatim}
x ← Pts[k]
u ← data\{x\}
\textsc{ListRemoveFirst}(Pts[i])
\end{verbatim}
\textbf{comment:} {Join $u$ to the next $k$ vertices $v$ of largest degree.}
\textbf{comment:} {If this is not possible, then $D$ is not graphic so exit.}
\begin{verbatim}
i ← k
\textbf{for} j ← 1 \textbf{to} k
\begin{verbatim}
while Pts[i] = NIL do \{i ← i - 1 if i = 0 exit
\textbf{x} = Pts[i]
\textbf{v} = data\{x\}
\textsc{ListRemoveFirst}(Pts[i])
\textsc{ListInsert}(\text{AdjList}[u], v)
\textsc{ListInsert}(\text{AdjList}[v], u)
\textsc{ListInsert}(\text{TempList}[i], v)
\end{verbatim}
\textbf{comment:} \{For each such $v$ joined to $u$ if $v$ is on list $Pts[j]$,
then transfer $v$ to $Pts[j - 1]$ \}
\textbf{for} j ← k \textbf{downto} 1
\begin{verbatim}
while TempList[j] ≠ NIL
\textbf{x} = TempList[j]
\textbf{v} = data\{x\}
\textsc{ListRemoveFirst}(TempList[j])
\textsc{ListInsert}(Pts[j - 1], v)
\end{verbatim}
\textbf{comment:} $u$ is now completely joined. Choose the next point.
\begin{verbatim}
\textbf{comment:} Now every vertex has been successfully joined.
\textit{graphic←true}
\end{verbatim}
This program is illustrated in Figure 1.13 for the sequence $D=(4, 4, 2, 2, 2, 2)$, where $n=6$. The diagram shows the linked lists before vertex 1 is joined to vertices 2, 3, 4, and 5, and the new configuration after joining. Care must be
FIGURE 1.13
The linked lists $Pts[k]$. (a) Before 1 is joined to 2, 3, 4, and 5. (b) After 1 is joined to 2, 3, 4, and 5.

Page 15
used in transferring the vertices $u$ from $Pts[j]$ to $Pts[j-1]$, since we do not want to join $u$ to $u$ more than once. The purpose of the list $Pts[0]$ is to collect vertices which have been transferred from $Pts[1]$ after having been joined to $u$. The degrees $d_1, d_2, \ldots, d_n$ need not necessarily be in descending order for the program to work, since the points are placed in the lists $Pts[k]$ according to their degree, thereby sorting them into buckets. Upon completion of the algorithm vertex $k$ will have degree $dk$. However, when this algorithm is done by hand, it is much more convenient to begin with a sorted list of degrees; for example, $D=(4, 3, 3, 3, 2, 2, 2, 2, 1)$, where $n=9$. We begin with vertex $u_1$, which is to have degree four. It will be joined to the vertices $u_2$, $u_3$, and $u_4$, all of degree three, and to one of $u_5$, $u_6$, $u_7$, and $u_8$, which have degree two. In order to keep the list of degrees sorted, we choose $u_8$. We then have $u_1 \rightarrow \{u_2, u_3, u_4, u_8\}$, and $D$ is reduced to $(\ast, \ast, 2, 2, 2, 2, 2, 1, 1)$. Continuing in this way, we obtain a graph $G$.

In general, when constructing $G$ by hand, when $uk$ is to be joined to one of $ui$ and $uj$, where $di=dj$ and $i<j$, then join $uk$ to $uj$ before $ui$, in order to keep $D$ sorted in descending order.

We still need to prove that Algorithm 1.2.1 works. It accepts a possible degree sequence $D=(d_1, d_2, \ldots, d_n)$, and joins $u_1$ to the $d_1$ vertices of largest remaining degree. It then reduces $D$ to new sequence $D'=(d'_2, d'_3, \ldots, d'_n)$.

**THEOREM 1.5 (Havel-Hakimi theorem)** $D$ is graphic if and only if $D'$ is graphic.
**PROOF** Suppose $D'$ is graphic. Then there is a graph $G'$ with degree sequence $D'$, where $V(G') = \{u_2, u_3, \ldots, u_n\}$ with $\text{DEG}(u_i) = d_i'$. Furthermore

$$D' = (d_2', d_3', \ldots, d_n')$$

consists of the degrees

$$\{d_2 - 1, d_3 - 1, \ldots, d_{d_1+1} - 1, d_{d_1+2}, \ldots, d_n\}$$

arranged in descending order. Create a new vertex $u_1$ and join it to vertices of degree $d_2 - 1, d_3 - 1, \ldots, d_{d_1+1} - 1$. Then $\text{DEG}(u_1) = d_1$. Call the new graph $G$. Clearly the degree sequence of $G$ is $D = (d_1, d_2, \ldots, d_n)$.

---

**FIGURE 1.14**

Vertices adjacent to $u_1$

Therefore $D$ is graphic. Now suppose $D$ is graphic. Then there is a graph $G$ with degree sequence $D = (d_1, d_2, \ldots, d_n)$, where $V(G) = \{u_1, u_2, \ldots, u_n\}$, with $\text{DEG}(u_i) = d_i$. If $u_1$ is adjacent to vertices of degree $d_2, d_3, \ldots, d_{d_1+1}$, then $G' = G - u_1$ has degree sequence $D'$, in which case $D'$ is graphic.

Otherwise, $u_1$ is not adjacent to vertices of degree $d_2, d_3, \ldots, d_{d_1+1}$. Let $u_k$ (where $k \geq 2$) be the first vertex such that $u_1$ is not joined to $u_k$, but is joined to $u_2, u_3, \ldots, u_{k-1}$. (Maybe $k = 2$.) Now $\text{DEG}(u_1) = d_1 \geq k$, so $u_1$ is joined to some vertex $x \neq u_2, u_3, \ldots, u_{k-1}$. $u_k$ is the vertex of next largest degree, so $\text{DEG}(u_k) \geq \text{DEG}(x)$. Now $x$ is joined to $u_1$, while $u_k$ is not. Therefore, there is some vertex $y$ such that $u_k \rightarrow y$ but $x \not\rightarrow y$. Set $G \leftarrow G + xy + u_1u_k - u_1x - u_ky$.

The degree sequence of $G$ has not changed, and now $u_1 \rightarrow \{u_2, u_3, \ldots, u_{k+1}\}$. Repeat until $u_1 \rightarrow \{u_2, u_3, \ldots, u_{d_1+1}\}$. Then $G' = G - u_1$ has degree sequence $D'$, so that $D'$ is graphic.

Therefore we know the algorithm will terminate with the correct answer, because it reduces $D$ to $D'$. So we have an algorithmic test to check whether $D$ is graphic and to generate a graph whenever one exists. There is another way of determining whether $D$ is graphic, without constructing a graph.

**THEOREM 1.6 (Erdős-Gallai theorem)** Let $D = (d_1, d_2, \ldots, d_n)$, where $d_1 \geq d_2 \geq \ldots \geq d_n$. Then $D$ is graphic if and only if

1. $\sum_{i=1}^{n} d_i$ is even; and
2. $\sum_{i=1}^{k} d_i \leq k(k - 1) + \sum_{i=k+1}^{n} \text{MIN}(k, d_i)$, for $k = 1, 2, \ldots, n$.

**PROOF** Suppose $D$ is graphic. Then $\sum_{i=1}^{n} d_i = 2\varepsilon$, which is even. Let $V_1$ contain the $k$ vertices of largest degree, and let $V_2 = V(G) - V_1$ be the remaining vertices. See Figure 1.15.
FIGURE 1.15
The vertices $V_1$ of largest degree and the remaining vertices $V_2$

Suppose that there are $\varepsilon_1$ edges within $V_1$ and $\varepsilon_2$ edges from $V_1$ to $V_2$. Then $\sum_{i=1}^{k} d_i = 2\varepsilon_1 + \varepsilon_2$, since each edge within $V_1$ is counted twice in the sum, once for each endpoint, but edges between $V_1$ and $V_2$ are counted once only. Now $\varepsilon_1 \leq \binom{k}{2}$, since $V_1$ can induce a complete subgraph at most. Each vertex $u \in V_2$ can be joined to at most $k$ vertices in $V_1$, since $|V_1| = k$, but $u$ can be joined to at most $\text{DEG}(u)$ vertices in $V_1$, if $\text{DEG}(u) < k$. Therefore $\varepsilon_2$, the number of edges between $V_1$ and $V_2$, is at most $\sum_{v \in V_2} \min(k, \text{DEG}(v))$, which equals $\sum_{i=k+1}^{n} \min(k, d_i)$. This now gives $\sum_{i=1}^{k} d_i = 2\varepsilon_1 + \varepsilon_2 \leq k(k - 1) + \sum_{i=k+1}^{n} \min(k, d_i)$

The proof of the converse is quite long, and is not included here. A proof by induction can be found in the book by Harary [59].

Conditions 1 and 2 of the above theorem are known as the Erdős-Gallai conditions.

Exercises
1.2.1 Prove Theorem 1.2 for arbitrary graphs. That is, prove

\[ \text{THEOREM 1.7} \quad \text{For any graph } G \text{ we have} \quad \sum_{u \in V(G)} \text{Deg}(u) + \ell = 2\varepsilon(G). \]

where $\ell$ is the number of loops in $G$ and $\text{Deg}(u)$ is the number of edges incident on $u$. What formula is obtained if loops count two toward $\text{Deg}(u)$?

1.2.2 If $G$ has degree sequence $D=(d_1, d_2, \ldots, d_n)$, what is the degree sequence of $\overline{G}$?

1.2.3 We know that a simple graph with $n$ vertices has at least one pair of vertices of equal degree, if $n \geq 2$. Find all simple graphs with exactly one pair of vertices with equal degrees. What are their degree sequences?

*Hint*: Begin with $n=2$, 3, 4. Use a recursive construction. Can degree 0 or $n-1$ occur twice?

1.2.4 Program the GRAPHGEN() algorithm. Input the sequence $D=(d_1, d_2, \ldots, d_n)$ and then construct a graph with that degree sequence, or else determine that the sequence is not graphic. Use the following input data:

(a) 4 4 4 4 4
(b) 3 3 3 3 3
(c) 3 3 3 3 3 3 3 3
(d) 3 3 3 3 3 3 3 3
(e) 2 2 2 2 2 2 2 2 2
(f) 7 6 6 6 5 5 2 1

1.2.5 Let $D=(d_1, d_2, \ldots, d_n)$, where $d_1 \geq d_2 \geq \ldots \geq d_n$. Prove that there is a multigraph with degree sequence $D$ if and only if $\sum_{i=1}^{n} d_i$ is even, and $d_1 \leq \sum_{i=2}^{n} d_i$.

1.3 Analysis
Let us estimate the number of steps that Algorithm 1.2.1 performs. Consider the loop structure

for $k \leftarrow n$ downto 1
do while Pts[k] \neq NIL

do { ...}

The for-loop performs $n$ iterations. For many of these iterations, the contents of the while-loop will not be executed, since $Pts[k]$ will be NIL. When the contents of the loop are executed, vertex $u$ of degree-to-be $k$ will be joined to $k$ vertices. This means that $k$ edges will be added to the adjacency lists of the graph $G$ being constructed. This takes $2k$ steps, since an edge $uu$ must be added to both $\text{GraphAdj}[u]$ and $\text{GraphAdj}[u]$. It also makes $\text{DEG}(u)=k$. When edge $uu$ is added, $u$ will be transferred from $Pts[j]$ to $Pts[j-1]$, requiring additional $k$ steps. Once $u$ has been joined, it is removed from the list. Write $\varepsilon = \frac{1}{2} \sum_i d_i$,.