Chapter 1
Graphs

Section 1.0 Introduction

For years, mathematicians have affected the growth and development of computer science. In the beginning they helped design computers for the express purpose of simplifying large mathematical computations. However, as the role of computers in our society changed, the needs of computer scientists began affecting the kind of mathematics being done.

Graph theory is a prime example of this change in thinking. Mathematicians study graphs because of their natural mathematical beauty, with relations to topology, algebra and matrix theory spurring their interest. Computer scientists also study graphs because of their many applications to computing, such as in data representation and network design. These applications have generated considerable interest in algorithms dealing with graphs and graph properties by both mathematicians and computer scientists.

Today, a study of graphs is not complete without at least an introduction to both theory and algorithms. This text will attempt to convince you that this is simply the nature of the subject and, in fact, the way it was meant to be treated.

Section 1.1 Fundamental Concepts and Notation

Graphs arise in many settings and are used to model a wide variety of situations. Perhaps the easiest way to adjust to this variety is to see several very different uses immediately. Initially, let's consider several problems and concentrate on finding models representing these problems, rather than worrying about their solutions.

Suppose that we are given a collection of intervals on the real line, say $C = \{ I_1, I_2, \ldots, I_k \}$. Any two of these intervals may or may not have a nonempty intersection. Suppose that we want a way to display the intersection relationship among these intervals. What form of model will easily display these intersections?

One possible model for representing these intersections is the following: Let each interval be represented by a circle and draw a line between two circles if, and only if, the intervals that correspond to these circles intersect. For example, consider the set

$$C = \{ [-4, 2], [0, 1], (-8, 2], [2, 4], [4, 10) \}.$$

The model for these intervals is shown in Figure 1.1.1.

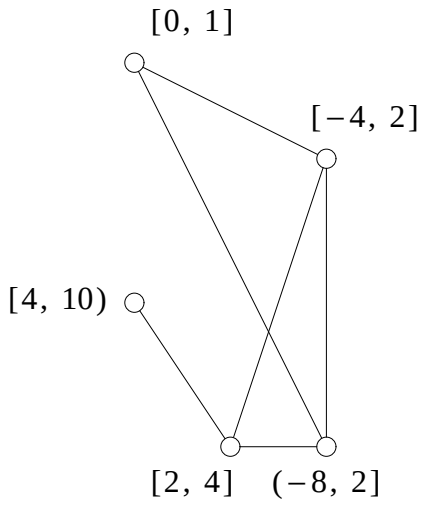


Figure 1.1.1. A model for the intersections of the members of C.

Next, we consider the following old puzzle. Suppose there are three houses (call them h_1, h_2 and h_3) and three utility companies (say gas (g), water (w) and electricity (e)). Our problem is to determine if it is possible to connect each of the three houses to each of the three utilities without crossing the service lines that run from the utilities to the houses. We model this puzzle by representing each house and each utility as a circle and drawing a line between two circles if there is a service line between the corresponding house and utility. We picture this situation in Figure 1.1.2. A solution to this problem would be a drawing in which no lines crossed. The drawing of Figure 1.1.2 is not a solution to the problem, but merely an attempt at modeling the problem.

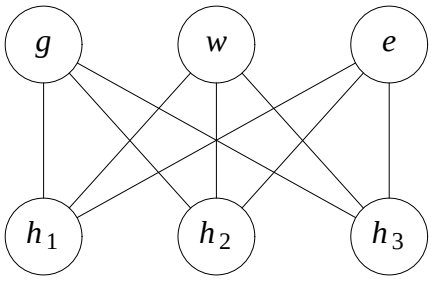


Figure 1.1.2. The three houses and three utilities model.

In our third problem, suppose you are the manager of a company that has four job openings (say j_1, j_2, j_3 and j_4) and five applicants $a_1, \ldots, a_5$ and that some of these applicants are qualified for more than one of your jobs. How do you go about choosing people to fill the jobs so that you will fill as many openings as possible? We picture such a situation in Figure 1.1.3. Again, each job and each applicant can be represented as a circle. This time, a line is drawn from a circle representing an applicant to each of the circles representing the jobs for which the applicant is qualified. A solution to this problem would be a set of four lines joining distinct jobs to distinct applicants, that is, one line joins each job to a distinct applicant. For example, the lines joining j_1 and a_2, j_2 and a_1, j_3 and a_4 and j_4 and a_5 constitute a solution to this problem. Since lines only join jobs to applicants, this is clearly the maximum number of lines possible. Can you find another solution? The real problem is how can we find solutions in general?

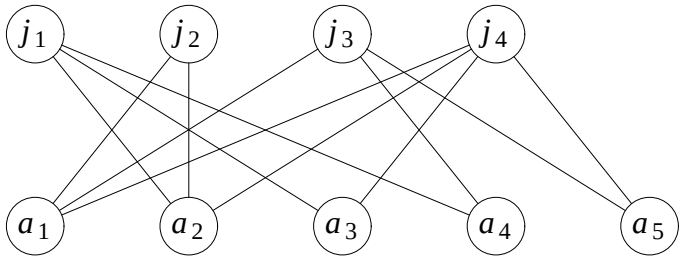


Figure 1.1.3. A job applicant model.

Despite the fact that these problems seem very different, we have used a similar type of diagram to model them. Such a diagram is called a graph. Formally, a *graph* $G = (V, E)$ is a finite nonempty set V of elements called *vertices*, together with a set E of two element subsets of V called *edges.* In our example diagrams, each circle is a vertex and each line joining two vertices is an edge. If the context is not clear, we will denote V or E by $V(G)$ or $E(G)$, respectively, to show they come from the graph G. In Figure 1.1.2, the vertices are h_1, h_2, h_3, g, w, e and the edges are

$$\{ h_1, g \}, \{ h_1, e \}, \{ h_1, w \}, \{ h_2, g \}$$
$$\{ h_2, e \}, \{ h_2, w \}, \{ h_3, e \}, \{ h_3, g \}, \{ h_3, w \}.$$

For simplicity, we will usually denote edges by consecutively listing the vertices at either end. For example, the edge $\{ h_1, g \}$ would be denoted $h_1 g$ or $g h_1$.

One of the beauties of graphs is that they may be thought of in many ways: formally as set systems, geometrically as the diagrams we have presented and algebraically, as we shall see later. Such diverse representations afford us an opportunity to use many tools in studying graphs and to apply graph models in many ways. To do this effectively, of course, we need to build more terminology and mathematical machinery.

Given a graph $G = (V, E)$, the number of vertices in V is called the *order of G* and the number of edges in E is called the *size of G*. They shall be denoted as $|V|$ and $|E|$, respectively. The interval graph of Figure 1.1.1 has order 5 and size 6. If a graph G has order p and size q, we say G is a (p, q) graph. Two vertices that are joined by an edge are said to be *adjacent,* as are two edges that meet at a vertex. If two vertices are not joined by an edge, we say they are *nonadjacent* or *independent.* Similarly, two edges that do not share a common vertex are said to be *independent.* The set of all vertices adjacent to a vertex v is called the *neighborhood of v* and is denoted $N(v)$. An edge between vertices u and v is said to have u (or v) as an *end vertex.* Further, the edge is said to be *incident* with v (or with u) and v is said to *dominate u* (also, u dominates v). The number of edges incident with a vertex v is called the *degree of v* and is denoted *deg v* or by $deg_G\ v$ if we wish to emphasize that this occurs in the graph G. The minimum degree and maximum degree of a vertex in the graph G are denoted by $\delta(G)$ and $\Delta(G)$, respectively. A graph in which each vertex has degree r is called an *r-regular* graph (or simply regular). We now present the theorem traditionally called The First Theorem of Graph Theory.

Theorem 1.1.1 Let G be a (p, q) graph and let $V = \{ v_1, v_2, \ldots, v_p \}$. Then $\sum_{i=1}^{p} deg\ v_i = 2q$. Consequently, any graph contains an even number of vertices of odd degree.

Proof. Since each edge has exactly two end vertices, the sum of the degrees counts each edge exactly twice. Thus, the sum is obtained. Since $2q$ is even, an even number of vertices of odd degree must then be present in the sum. □

We have considered three problems thus far. The drawing of Figure 1.1.1 is a solution to the first problem, and we have an idea of what a solution for one example of the third problem looks like. But the drawing given for the utilities problem does not provide a solution to that problem. This does not mean there is no solution, only that our drawing fails to provide one. What if we try other drawings (see Figure 1.1.4)? One of the interesting features of these drawings is the freedom we have to shape them. There are no restrictions on the size of the vertices or on the length or even the shape of the edges. These drawings are very much free-form. We are also free to choose an entirely different representation for our graph, for example the set representation we used in defining graphs. But this freedom also presents us with some difficulties. If a graph is presented in different ways, how can we determine if the presentations really represent the same graph?

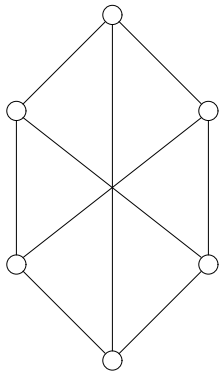


Figure 1.1.4. Another drawing of the house-utilities graph.

Mathematicians use the term *isomorphism* to mean the "fundamental equality" of two objects or systems. That is, the objects really have the same mathematical structure, only nonessential features like object names might be different. For graphs, "fundamentally equal" means the graphs have essentially the same adjacencies and nonadjacencies. To formalize this concept further, we say two graphs G_1 and G_2 are *isomorphic* if there exists a 1-1 and onto function $f : V(G_1) \rightarrow V(G_2)$ such that $xy \in E(G_1)$ if, and only if, $f(x)f(y) \in E(G_2)$ (that is, f preserves adjacency and nonadjacency). We use the function f to express the correspondence between vertices that are "essentially the same" in the two graphs. The function f is called an *isomorphism.*

Example 1.1.1 The two drawings of the house-utilities graph are again shown in Figure 1.1.5. An isomorphism between G_1 and G_2 is determined by the function $f : V(G_1) \rightarrow V(G_2)$ where:

$$f(a) = x, \quad f(b) = r, \quad f(c) = y,$$
$$f(d) = s, \quad f(e) = z, \quad f(g) = t.$$

An isomorphism from G_2 to G_1 is given by f^{-1}, the inverse of f.

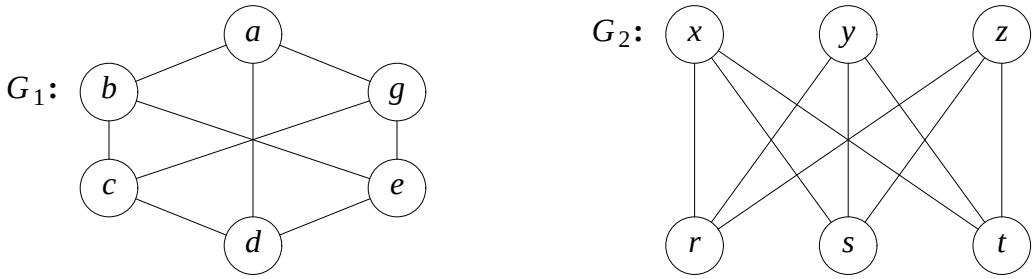


Figure 1.1.5. Two isomorphic graphs.

Can you find other isomorphisms from G_1 to G_2? □

A *subgraph of G* is any graph H such that $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$; we also say *G contains H*. If H is a subgraph of G and $V(H) = V(G)$, we say that H is a *spanning subgraph* of G. A more restricted but often very useful idea is the following: Given a subset S of $V(G)$, the *subgraph induced by S*, denoted <S>, is that graph with vertex set S and edge set consisting of those edges of G incident with two vertices of S.

The graphs in Figure 1.1.6 illustrate these ideas. Since $V(H) = V(G)$, H is a spanning subgraph of G. Also, I is an induced subgraph of G since all edges of G with both end vertices in $V(I)$ are contained in I. However, J is not an induced subgraph of G since the edge from 1 to 5 is in G, but is not in J.

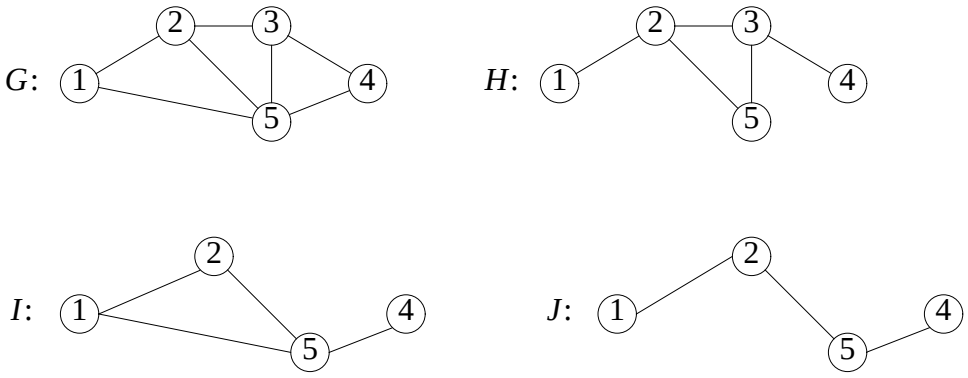
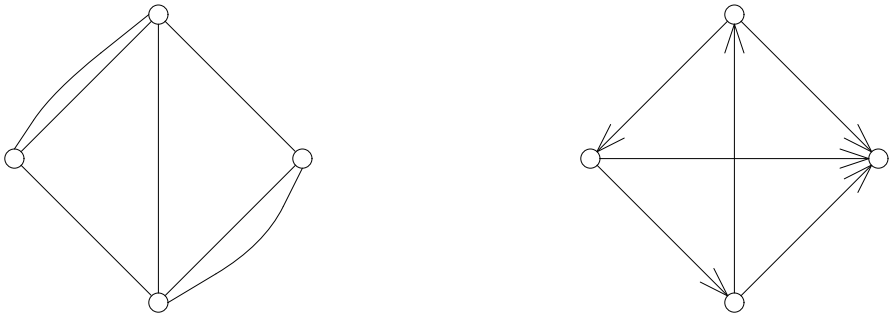


Figure 1.1.6. Spanning subgraph H, induced subgraph I, subgraph J of G.

Several natural and useful variations on graphs will be helpful. The first is the idea of a *multigraph,* that is, a graph with (possibly) multiple edges between vertices. A *pseudograph* allows edges that begin and end at the same vertex (called a *loop*). If we think of the edge between two vertices as an ordered pair rather than a set, a natural direction from the first vertex of the pair to the second can be associated with the edge. Such edges will be called *arcs* (to maintain the historical terminology), and graphs in which each edge has such a direction will be called *directed graphs* or *digraphs.* For digraphs, the number of arcs directed away from a vertex v is called the *outdegree* of v (denoted $od\ v$) and the number of arcs directed into a vertex v is the *indegree* of v (denoted $id\ v$). Often, for emphasis, we denote the arc directed from u to v as $u \to v$. In a digraph, we define the *degree* of a vertex v to be $deg\ v = id\ v + od\ v$. If $u \to v$ is an arc of the digraph, we say that *u dominates v* and that *v* is *dominated by u*. Sometimes we say *u* is *adjacent to v* or *v* is *adjacent from u*.

Clearly, we can produce even more variations such as pseudodigraphs, multidigraphs and pseudomultidigraphs. Although these will not play as significant a role in our study of graphs, at times they will be useful. In this text we will be sure the reader understands the kind of graph under consideration, and the term *graph* will always be as we defined it: finite order, without loops, multiple edges or directed edges.



multigraph digraph

Figure 1.1.7a. A multigraph and a digraph.

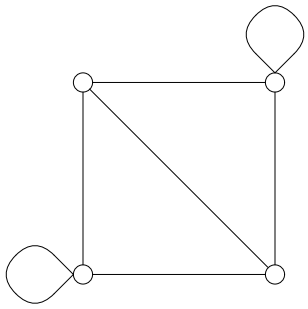


Figure 1.1.7b. A pseudograph.

Section 1.2 Elementary Properties and Operations

A quick inspection of a road map of the southern states shows several important cities and the interstates that connect them. We model a portion of this map in Figure 1.2.1.

It seems natural to think of a graph (or digraph) when trying to model a road system. Vertices represent cities, and edges (or arcs) represent the roads between the cities. In tracing the route from one location to another, we would traverse some sequence of roads, beginning at some starting point, and finally reaching our destination. For example, we could travel from Atlanta to Nashville by first leaving Atlanta and traveling along I75 to Chattanooga, then following I24 to Nashville. Such a model leads us naturally to formally define the following concepts.

Let x and y be two vertices of a graph G (not necessarily distinct vertices). An $x - y$ *walk* in G is a finite alternating sequence of vertices and edges that begins with the vertex x and ends with the vertex y and in which each edge in the sequence joins the vertex that precedes it in the sequence to the vertex that follows it in the sequence. For example, in the graph of Figure 1.2.1, one $b - n$ walk is

$$b, I59, c_2, I75, a, I20, b, I59, c_2, I24, n$$

while another is

$$b, I20, a, I85, c_1, I85, a, I75, c_2, I24, n.$$

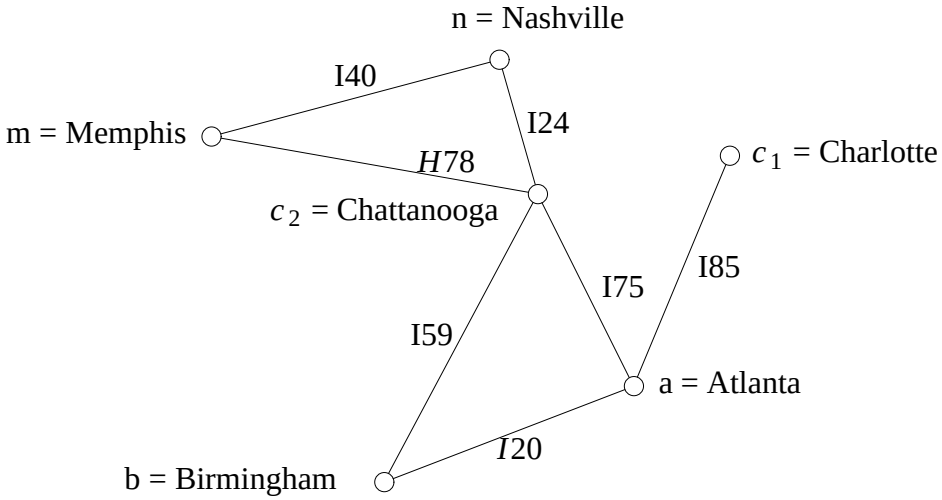


Figure 1.2.1. A model of roads between some southern cities.

The number of edges in a walk is called the *length* of the walk. Note that repetition of vertices and edges is allowed. Ordinarily, we will use a more compact notation for a walk by merely listing the vertices involved, noting that the edge between consecutive vertices (at least in a graph or digraph) is then implied. We will only use the full notation for walks in multigraphs, where there is a choice of edges and this choice is important, or for emphasis of the edges involved. An $x - y$ walk is *closed* if $x = y$ and *open* otherwise. Two walks are equal if the sequences of vertices and edges are identical.

Several stronger types of walks are also important. An $x - y$ *trail* is an $x - y$ walk in which no edge is repeated, and an $x - y$ *path* is an $x - y$ walk in which no vertex is repeated, except possibly the first and the last (if the path is closed). Clearly, a path is also a trail. We consider a single vertex as a trivial path (walk or trail). In the graph of Figure 1.2.1, we see that $a, b, a, c_2, b,$ is an $a - b$ walk of length 4, while a, b, c_2, a, c_1 is an $a - c_1$ trail of length 4, and a, c_2, n, m is an $a - m$ path of length 3.

It is clear that every path is a trail and every trail is a walk and that the converse of each of these statements fails to hold. However, we now present a useful result that relates walks and paths.

Theorem 1.2.1 In a graph G, every $x - y$ walk contains an $x - y$ path.

Proof. Let W be an $x - y$ walk in the graph G. Note that a vertex may receive more than one label if it occurs more than once in W. If no vertex is repeated, then W is

already a path; hence, we assume that at least one vertex is repeated in W. Let i and j be distinct integers with $i < j$ such that $v_i = v_j$. That is, the vertex v_i is repeated as v_j. If we now delete the vertices $v_i , v_{i+1} , \ldots , v_{j-1}$ from W, we obtain an $x - y$ walk W_1 which is shorter than W and has fewer repeated vertices. If W_1 is a path, we are done; if not, we repeat this process to obtain a new $x - y$ walk W_2. If W_2 is a path, we are done; otherwise, repeat this process. Since W is a finite sequence, eventually we must reach the stage where no vertices are repeated and an $x - y$ path is obtained. □

A closed trail is called a *circuit,* while a nontrivial circuit with no repeated vertices (other than the first and last) is called a *cycle.* We allow C_2 as a cycle, but note that it does not occur in graphs. The existence of C_2 is restricted to multigraphs. We do not consider C_1 (a single vertex) as a trivial cycle as this adds more complications than benefits. The *length* of a cycle (or circuit) is the number of edges in the cycle (or circuit). In the graph of Figure 1.2.3, w, x, y, z, w is a cycle of length 4; while t, r, u, t, s, v, t is a circuit of length 6. A graph of order n that consists of only a cycle (or path) is denoted C_n (or P_n) and is called simply an *n-cycle (or n-path).* If a graph contains no cycles it is termed *acyclic.* Cycles and paths are very important ideas and will be studied in much greater detail later. For now, they allow us to continue expanding our terminology. The graph of Figure 1.2.2 is an example of another special class of graphs called *complete graphs,* which contain an edge between all pairs of vertices. We denote the complete graph of order p as K_p.

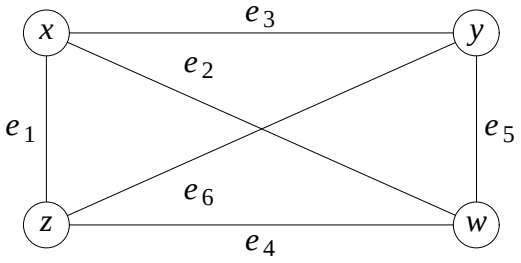


Figure 1.2.2. The complete graph K_4.

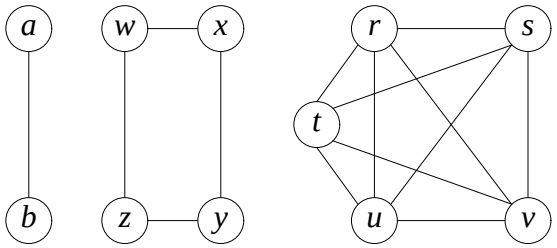


Figure 1.2.3. A disconnected graph *H*.

The graph *H* of Figure 1.2.3 is clearly different from those we considered earlier. For one thing, *H* consists of three "pieces." Each piece should be thought of as being "connected," but *H* should not. We can formalize this idea and obtain a useful way of describing "connected graphs" by using paths. We say a graph *G* is *connected* if there exists a path in *G* between any two of its vertices and *G* is *disconnected* otherwise. Clearly, there is no $a - z$ path in *H*, but there are paths between any two of w, x, y, and z. A *component* of a graph is a maximal connected subgraph. Thus, in this case *H* has three components, a P_2, a C_4 and a K_5.

Connectivity in digraphs is a little more interesting in that there are several possible kinds. A digraph *D* is said to be *strongly connected* (or *strong*) if for each vertex *v* of *D* there exists a directed path from *v* to any other vertex of *D*. We say *D* is *weakly connected* (or *weak*) if, when we remove the orientation from the arcs of *D*, a connected graph or multigraph remains (often, we say that the underlying graph is connected). Of course, *D* is *disconnected* if it is not at least weakly connected. For example, the digraph *E* of Figure 1.2.4 is clearly weakly connected. However, *E* is not strong since there is no directed path in *E* from *x* to any other vertex of *E*. The digraph *D* is also easily seen to be strong.

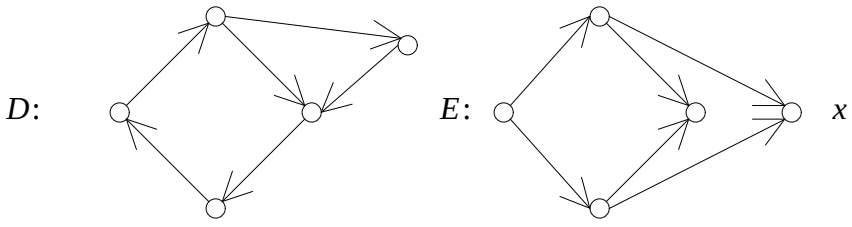


Figure 1.2.4. A strong digraph *D* and weak digraph *E*.

Combining several previous ideas, we define a *tree* to be a connected acyclic graph; a *forest* is an acyclic graph, that is, a graph each of whose components is a tree. (What else would a forest be?) In defining a forest as a collection of trees, what we have done is form a new graph from other graphs. In particular, the *union* of two graphs G_1 and G_2 (denoted $G_1 \cup G_2$) is that graph G with $V(G) = V(G_1) \cup V(G_2)$ and $E(G) = E(G_1) \cup E(G_2)$. If we form the union of m isomorphic copies of the graph G, we denote the resulting graph as mG. As an example of this concept, the graph of Figure 1.2.3 can be denoted as $H = P_2 \cup C_4 \cup K_5$.

Graph union is an example of one graph operation used to form a new graph from other graphs. There are many other graph operations. Perhaps the simplest and most natural graph operation is the following: Consider a graph $G = (V, E)$ and form a new graph $\overline{G}$ where $V(\overline{G}) = V(G)$ and an edge $e \in E(\overline{G})$ if, and only if, e is not in $E(G)$. We call $\overline{G}$ the *complement of G*. In a sense, all we have done is remove all the edges from G and insert those edges that were originally missing from G. It should also be clear that the complement of $\overline{G}$ is the graph G itself.

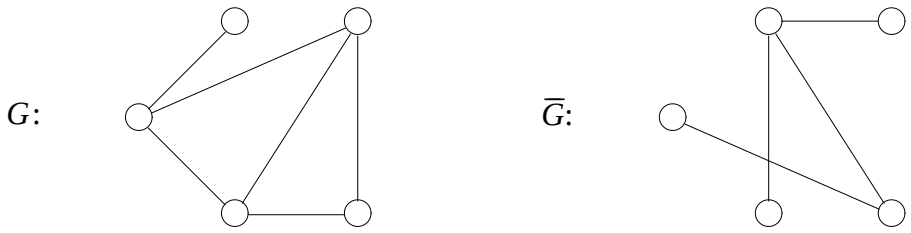


Figure 1.2.5. A graph G and its complement $\overline{G}$.

The *join* of two graphs G and H with disjoint vertex sets, denoted $G + H$, is the graph consisting of $G \cup H$ and all edges between vertices of G and vertices of H. Often, we will have occasion to consider the special case when $H = K_1$ and for simplicity, we will denote this as $G + x$, where K_1 is the vertex x. We define the *complete bipartite* graph $K_{m,n}$ to be the join $\overline{K}_m + \overline{K}_n$. The graph of Figure 1.1.2 is $K_{3,3}$. Complete bipartite graphs are a special case of an important class of graphs. We say a graph G is *bipartite* if it is possible to partition the vertex set V into two sets (called *partite sets*), say V_1 and V_2, such that each edge of G joins a vertex in V_1 to a vertex of V_2. Clearly complete bipartite graphs are bipartite, but complete bipartite graphs contain all possible edges between the partite sets. More generally, a graph G is *n-partite* if it is possible to partition the vertex set of G into n sets, such that any edge of G joins two vertices in different partite sets. Complete n-partite graphs have all possible edges between the partite sets. We denote the complete n-partite graph with partite sets of

order $p_1, p_2, \ldots, p_n$ as $K_{p_1, p_2, \ldots, p_n}$. Can you find a representation for $K_{p_1, p_2, \ldots, p_n}$ as the join of graphs?

On the other hand, at times we will also remove a set S of vertices from a graph G, along with all edges of G incident to a vertex in S. We denote the resulting graph as $G - S$. Again, if $S = \{ x \}$, we denote the resulting graph as $G - x$.

There are several graph operations that result in a graph whose vertex set is the cartesian product of the vertex sets of two graphs. The classic paper of Sabidussi [10] deals with several of these. We shall now consider some of these products.

The *cartesian product* of graphs G_1 and G_2, denoted $G_1 \times G_2$, is defined to be the graph with $V(G_1 \times G_2) = V(G_1) \times V(G_2)$, and two vertices $v = (v_1, v_2)$ and $w = (w_1, w_2)$ are adjacent in the cartesian product whenever $v_1 = w_1$ and v_2 is adjacent to w_2 in G_2 or symmetrically if $v_2 = w_2$ and v_1 is adjacent to w_1 in G_1. Can you show that $P_3 \times P_2$ is isomorphic to $P_2 \times P_3$? In general, is $G_1 \times G_2$ isomorphic to $G_2 \times G_1$?

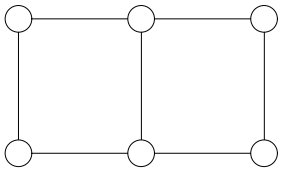


Figure 1.2.6. The cartesian product $P_2 \times P_3$.

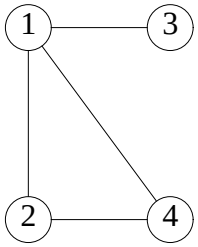
The *lexicographic product* (sometimes called the *composition*) of two graphs G_1 and G_2, denoted $G_1 [G_2]$, also has vertex set $V(G_1) \times V(G_2)$, while (v_1, v_2) is adjacent to (w_1, w_2) if, and only if, either v_1 is adjacent to w_1 in G_1 or $v_1 = w_1$ in G_1 and $v_2 w_2 \in E(G_2)$. In general, is $G_1 [G_2]$ isomorphic to $G_2 [G_1]$?

Figure 1.2.7. The lexicographic products $P_2[P_3]$ and $P_3[P_2]$.

Section 1.3 Alternate Representations for Graphs

Earlier, we considered two ways of representing graphs, as a set system (V, E) and pictorially. Now, we consider several other ways of viewing graphs.

Let $G = (V, E)$ be a (p, q)-graph. Consider the $p \times p$ matrix $A = [a_{ij}]$, where each row (and each column) of A corresponds to a distinct vertex of V. Let $a_{ij} = 1$ if vertex v_i is adjacent to vertex v_j in G and $a_{ij} = 0$ otherwise. Note that $a_{ii} = 0$ for each $i = 1, 2, \ldots, p$. This *adjacency matrix* of G is clearly a symmetric $(0, 1)$-matrix, with zeros down the main diagonal. The adjacency matrix clearly contains all the structural information about G and thus can be used as a representation for G. This representation has several advantages. First, $(0, 1)$-matrices are well-studied objects, and we gain all the power of linear algebra as a tool for studying graphs (although we shall not take full advantage of this here, the interested reader should see [1]). Second, the adjacency matrix representation is a very convenient one for storage within a computer.



$$\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix}$$

Figure 1.3.1. A graph and its adjacency matrix.

We now determine a method for finding the number of $x-y$ walks of a given length in a graph.

Theorem 1.3.1 If A is the adjacency matrix of a graph G with vertices $v_1, v_2, \ldots, v_p$, then the (i, j)-entry of A^n is the number of $v_i - v_j$ walks of length n in G.

Proof. We will apply induction on the length n. For $n = 1$, the result is obvious, as this is just the adjacency matrix itself. Now let $A^{n-1} = [a_{ij}^{n-1}]$ and assume that a_{ij}^{n-1} is the number of distinct $v_i - v_j$ walks of length $n - 1$ in G. Also, let $A^n = [\, a_{ij}^n \,]$. Since $A^n = A^{n-1} A$, we have that

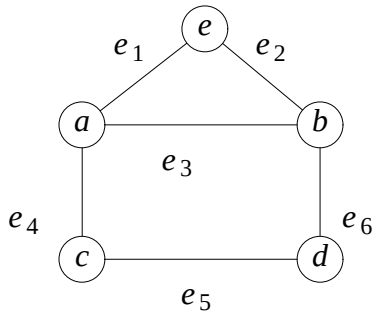
$$a_{ij}^{n} = \sum_{k=1}^{p} a_{ik}^{n-1}\, a_{kj}. \qquad 1.1$$

Every $v_i - v_j$ walk of length n in G must consist of a $v_i - v_k$ walk of length $n - 1$ followed by the edge from v_k to v_j and the vertex v_j. Thus, by induction and equation (1.1), the result follows. □

An idea similar to that of the adjacency matrix is the *incidence matrix.* For a (p, q) graph G, let the $p \times q$ matrix $M = [i_{xe}]$ be defined as follows: $i_{xe} = 1$ if vertex x is incident to edge e and $i_{xe} = 0$ otherwise. Thus, the rows of M correspond to the vertices of G and the columns correspond to the edges. It is easy to see that all the structure of G is contained in M; however, because M is not square, it lacks some of the power of adjacency matrices. Despite this shortcoming, incidence matrices have some valuable uses.

Still other, much simpler representations tend to be useful in computer applications. Probably the most commonly used form is merely to list each vertex in V along with those vertices that are adjacent to the listed vertex. This *adjacency list* contains all the structure of G, but has no extraneous information about nonadjacencies (like the zeros of the adjacency matrix). Nonadjacency is implied by omission from the list. Thus, when using an adjacency list, a program does not need to decide whether or not the next piece of information shows an edge or a nonedge, but rather it just retrieves the next adjacency. Tests with a number of algorithms have shown that adjacency lists will often speed computations. This is especially true in graphs that have many more nonadjacencies than adjacencies (called *sparse* graphs).

Example 1.3.1. The adjacency lists for the graph of Figure 1.3.2 are now shown.



a: b, c, e
b: a, d, e
c: a, d
d: b, c
e: a, b

Figure 1.3.2. A graph and its adjacency lists.

$$\begin{array}{c} \\ a \\ b \\ c \\ d \\ e \end{array} \begin{bmatrix} e_1 & e_2 & e_3 & e_4 & e_5 & e_6 \\ 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 1.3.3. The incidence matrix for the graph of Figure 1.3.2.

Section 1.4 Algorithms

In order to deal with graphs efficiently and actually determine their properties on a computer, we need to develop processes that, when applied to a particular graph, will produce the desired answer. Algorithms are step-by-step procedures for solving problems. Usually, a graph problem will be posed in terms of several *parameters* (or variables). We then describe the problem at hand by giving a specification of the parameters involved and a statement about what constitutes a solution. An *instance* of a problem is obtained when we specify values for the parameters.

When dealing with graphs on a computer, we face several problems, including finding an algorithm that answers the question. But even then, there is no guarantee that we will be able to actually solve the problem. There are several other difficulties that must be faced. The first is the amount of space necessary for the information needed to store the description of the instance of our problem. Such a description might typically be in the form of the graph structure (adjacency matrices, adjacency lists, etc.). We may also need space to keep any partial results or necessary facts. As graphs grow large, this information can simply be too much to deal with. However, since computer memory has become cheaper and external storage (like tapes and discs) can be used, we shall proceed as though space is not the problem of fundamental concern, but rather that we can manage the *space requirements* of our problem.

The problem we will be more concerned with is the *time complexity*, that is, the relative time it will take us to perform the algorithm. We speak of relative time because it is impractical to try to determine the exact running times of algorithms. The computational speeds of machines differ, and the varying skills of programmers can also affect how well an algorithm seems to perform. These concerns make it unreasonable to try to measure time performance exactly. Instead, what we try to measure is the number

of computational steps involved in the algorithm. Since the exact steps we perform might well depend upon the graph being investigated (the particular instance of the problem), ordinarily our estimates are a worst-case measure of performance. Thus, an upper bound on the time complexity of the problem is usually obtained.

Consider as an example the problem of computing the square of the adjacency matrix A of a graph of order p. If we follow the typical rules for matrix multiplication, we perform p multiplications and $p - 1$ additions to compute each entry of A^2. Since there are p^2 entries in A^2, this means the algorithm really requires $p^2 \times (2p - 1) = 2p^3 - p^2$ operations to complete its task. Clearly, as p grows larger, the number of computations we must perform grows even faster.

In judging the quality of an algorithm, we try to measure how well it performs on arbitrary input. In doing this, we try to find an upper bound on the number of computational steps necessary to perform this algorithm. This entails finding some function that bounds the number of computational steps we must perform. As in the description of matrix multiplication, this is usually a function of the *size* of the problem. Here, *size* only refers to the amount of data in the instance of the problem at hand. Since it is clear that we must compute more if we have more data, it only makes sense for us to try to measure our work as a function of the problem size.

Returning to the problem of squaring the adjacency matrix A, if c_1 is the maximum amount of time required to multiply two numbers and c_2 is the maximum amount of time necessary to add two numbers, and c_3 is the maximum amount of time necessary to do all the other steps necessary for us to perform the algorithm (you can think of this as setup time to read in data, etc.), then we can bound the time, $T(p)$, it takes to square the adjacency matrix A (or, for that matter, to perform matrix multiplication of two $p \times p$ matrices) as:

$$T(p) \leq (c_1 p + c_2 (p - 1)) p^2 + c_3 = (c_1 + c_2) p^3 - c_2 p^2 + c_3$$

Thus, the amount of time it takes to square the adjacency matrix is bounded by a cubic function of p, the order of the graph. Since the constants are all dependent on the machines, languages, programmers and other factors outside our control, we simplify our description by saying that the problem of squaring the adjacency matrix of a graph of order p is *On The Order of* p^3, or $O(p^3)$ (read big oh of p^3). Hence, we say that the time complexity of the matrix multiplication algorithm is $O(p^3)$.

Formally, we say that $g(n) = O(f(n))$ if there exist constants k and m, such that $|g(n)| \leq k |f(n)|$ for all $n \geq m$. An algorithm has *polynomial time complexity* if its computational time complexity $T(n) = O(p(n))$ for some polynomial p in the input size n.

Table 1.4.1 compares the times for various bounding functions based on given input sizes. This table assumes that any operation requires .000001 seconds.

	problem size			
time	10	30	50	100
n	.00001 sec	.00003 sec	.00005 sec	.0001 sec
n^2	.0001 sec	.0009 sec	.0025 sec	.01 sec
n^5	.1 sec	24.3 sec	5.2 min	2.7 hrs
2^n	.001 sec	17.9 min	35.7 yrs	2^{48} cent
3^n	.059 sec	6.5 yrs	2 x 10^8 cent	3^{70} cent

Table 1.4.1 Comparisons of several time functions and data sizes.

It is clear that for the same problem size, the smaller the bounding function, the faster the algorithm is likely to run. That is, linear algorithms (those bounded by linear functions) should be "faster" than those with complexity $O(p^5)$, which in turn should be faster than those with complexity $O(p^{50})$ for similar data sets. It is also clear that for some functions (like 2^n), we have no hope of success, even with relatively small data sizes. For example, if the process under consideration has complexity $O(3^n)$, then we can see there is no real hope of success even for values as small as $n = 50$. We call any problem for which no polynomial algorithm can exist an *intractable problem.*

There is yet another kind of problem that at the present time sits between the polynomial problems (those with a known polynomial time solution) and the intractable problems. In order to consider such problems, we must consider *decision problems,* that is, problems that can be answered "yes" or "no". For example, one such decision problem is: Given two graphs G_1 and G_2, does G_1 contain a subgraph isomorphic to G_2?

A problem is said to be *in the class NP* if it can be solved by a nondeterministic polynomial algorithm. We can view such an algorithm as being composed of two parts. The first stage is the *guessing stage.* Here, given an instance *I,* the guessing stage selects some "structure" (graph, subgraph, vertex set, edge set, etc.) as a possible solution. Then, in the second stage, the *checking stage,* the structure is checked to see if it provides a yes or no answer to the decision problem. A nondeterministic algorithm solves a decision problem if it always correctly provides a yes or no answer for the guessed structure (in polynomial time). It should be clear that the set of all problems with polynomial solutions *P* is a subset of *NP*. It is not known if $P = NP$, although many believe that this

is not the case.

There is yet another special type of problem within *NP*, the so called *NP-complete* problems. These problems have been shown to be in a sense the hardest problems in *NP*. That is, a problem *X* is *NP*-complete if a solution for *X* provides a solution for all other problems *Y* in *NP*. By this we mean that there is a polynomial algorithm to "transform" *X* into *Y* (and, hence, to convert a solution of *X* into a solution of *Y*).

A rather interesting theory of computational complexity has arisen around these classes. It is not within the scope of this text to study this theory, but we do point out that there are many graph theoretic problems that are known to be in this elusive class of *NP*-complete problems. The interested reader should see the excellent text of Garey and Johnson [6].

Section 1.5 Degree Sequences

Given any graph, we can easily find the degree of each of its vertices. For example, the graph of Figure 1.3.2 has vertices of degree 2, 3, 3, 2 and 2. Each graph can be associated with such a unique sequence called its *degree sequence.* For convenience, these degrees can be ordered to form a nonincreasing sequence of nonnegative integers. In this case, the sequence 3, 3, 2, 2, 2 is formed. Several interesting questions about degree sequences now come to mind.

The first question you might think of is: Can we reverse this process? By this we mean, given a degree sequence *S*, can we determine a graph with *S* as its degree sequence? Perhaps a better first question is: Can we determine when a sequence of integers represents the degree sequence of a graph? A sequence is said to be *graphical* if it is the degree sequence of some graph. A graph *G* with degree sequence *S* is called a *realization* of *S*. Finally, given a sequence *S* that is graphical, with realization *G*, is *G* uniquely determined, or can there be several nonisomorphic realizations of *S*?

Let’s begin with the question: Are there some obvious restrictions on *S*? Certain conditions are clearly important. First, degrees are nonnegative integers; thus, all the terms of the sequence must be nonnegative integers. Next, if *S* has *p* terms, then no term can be larger than $p - 1$, because no vertex can be adjacent to more than the $p - 1$ other vertices. There are still other conditions that will eliminate some sequences.

Consider the sequence *S*: 1, 1, 1. For *S* to be the degree sequence of some graph, the graph must have exactly three vertices of degree one. But by Theorem 1.1.1, any graph must have an even number of vertices of odd degree, and therefore *S* cannot be a degree

sequence. Remember, Theorem 1.1.1 tells us that the sum of the degrees of the vertices in any graph must be an even number, and hence the sum of the terms of S must be even.

Next, we ask: If the sequence S is graphical, is the graph uniquely determined? That is, must there be only one graph (up to isomorphism) with S as its degree sequence? To answer this question, consider the sequence S: 2, 2, 2, 2, 2, 2. This sequence passes the first test in that all terms are nonnegative integers. It also passes the second test in that it contains only even valued entries. It is easy to find two nonisomorphic graphs that have degree sequence S. The graphs C_6 and $2C_3$ are two such graphs (see Figure 1.5.1). Thus, we have a negative answer to our question, that is, we see that degree sequences do not always provide enough information to uniquely describe a graph.

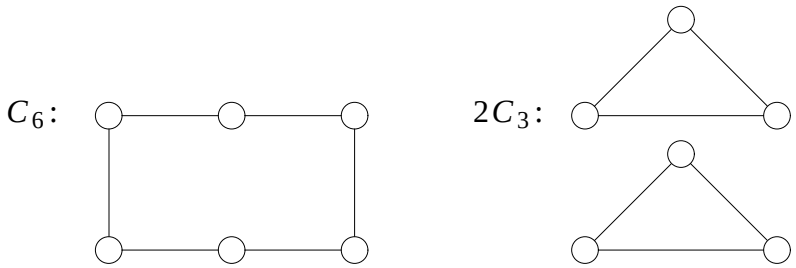


Figure 1.5.1. Two graphs with the same degree sequence.

The most important question raised earlier was: Can we determine when a sequence is graphical? The answer to our question was provided independently by Havel [8] and Hakimi [7].

Theorem 1.5.1 A nonincreasing sequence of nonnegative integers S: $d_1, d_2, \ldots, d_p$ $(p \geq 2,\ d_1 \geq 0)$ is graphical if, and only if, the sequence S_1 : $d_2 - 1, d_3 - 1, \ldots, d_{d_1+1} - 1, d_{d_1+2}, \ldots, d_p$ is graphical.

Proof. Suppose that the sequence S_1 is graphical and let G_1 be a graph of order $p - 1$ with degree sequence S_1. The vertices of G_1 can be labeled as $x_2, x_3, \ldots, x_p$ in such a way that deg $x_i = d_i - 1$ if $2 \leq i \leq d_1 + 1$ and *deg* $x_i = d_i$ if $d_1 + 2 \leq i \leq p$. We can construct a new graph G with degree sequence S by inserting into G_1 a new vertex x_1 and the edges x_1x_j for $2 \leq j \leq d_1 + 1$. The degree of x_1 is d_1, and the degrees of the other vertices are now the remaining values of S. Thus, we have constructed a graph with degree sequence S, and so S is graphical.

Conversely, suppose that S is graphical and among all graphs with degree sequence S, let G be chosen with the following properties:

1. $V(G) = \{ x_1, x_2, \ldots, x_p \}$ and $\deg x_i = d_i, \quad i = 1, 2, \ldots, p.$
2. The sum of the degrees of the vertices adjacent to x_1 is a maximum.

Suppose that x_1 is not adjacent to vertices having degrees $d_2, d_3, \ldots, d_{d_1+1}$, that is, x_1 is not adjacent to the d_1 other vertices of largest degrees. Then there exist two vertices x_i and x_j with $d_j > d_i$ and such that x_i is adjacent to x_1 but x_j is not adjacent to x_1. Since $d_j > d_i$, there exists a vertex x_k such that x_k is adjacent to x_j but not to x_i. Now, removing the edges x_1x_i and x_jx_k and inserting x_1x_j and x_ix_k (see Figure 1.5.2) results in a new graph H with degree sequence S. However, in H the sum of the degrees of the vertices adjacent to x_1 is greater than in G, which contradicts property (2) in our choice of G. Thus, x_1 must be adjacent in G to the d_1 other vertices of largest degree. Now the graph $G - x_1$ has degree sequence S_1, and, hence, S_1 is graphical. □

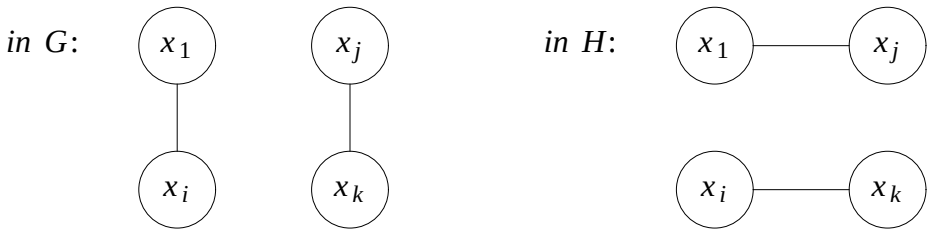


Figure 1.5.2. An edge interchange.

The fundamental step in the last proof was deleting the two edges x_1x_i and x_jx_k and inserting the edges x_1x_j and x_ix_k. This operation left the degrees of the vertices unchanged but varied the structure of the graph, and, it has come to be called an *edge interchange* (see Figure 1.5.2).

The proof of the last theorem essentially provides an algorithm for testing whether a sequence of nonnegative integers is graphical. We begin by applying our earlier tests for ruling out sequences, followed by the reduction from S to S_1. We continue to repeat this process until the sequence being tested fails or until a sequence of all zeros occurs. The sequence of t zeros is graphical because t vertices and no edges suffices. This is called the *empty graph.* We summarize these steps in the following algorithm.

Algorithm 1.5.1 Test for a Graphical Sequence.
Input: A sequence S of nonnegative integers of length p.
Output: Yes if the sequence is graphical, no otherwise.

1. If there exists an integer d in S such that $d > p - 1$, then halt and answer no.
2. If the sequence is all zeros, then halt and answer yes.
3. If the sequence contains a negative number, then halt and answer no.
4. Reorder the sequence (if necessary) so that it is nonincreasing.
5. Delete the first term d_1 from the sequence and subtract one from the next d_1 terms to form a new sequence. Go to step 2.

We can show that Algorithm 1.5.1 actually determines whether the sequence S is graphical. If the sequence halts before we apply step 5, then clearly a determination has been made in step 1, 2 or 3. Thus, we must show that after applying step 5, we will eventually produce a sequence of all zeros or a sequence that contains a negative term (since returning to step 2 means that only test 2 or 3 will halt the algorithm). By the time we reach step 5, we already know that S contains p terms and the largest is at most $p - 1$ (from step 1). In applying step 5, we produce a sequence of $p - 1$ terms with largest value at most $p - 2$. In general, if we apply step 5 a total of k times, we produce a sequence of $p - k$ terms with the largest entry at most $p - 1 - k$. If step 5 were actually applied $p - 1$ times, then the resulting sequence would be a single zero. Hence, we must eventually produce a sequence that has negative terms or we will surely produce a sequence of all zeros.

Once we have applied Algorithm 1.5.1 and determined that a particular sequence S is graphical, we can use the intermediate sequences we constructed to produce a graph with degree sequence S. Let's consider the following example.

Example 1.5.1. Construction of a graph from its intermediate degree sequences.

Suppose we begin with the sequence S_1: 5, 4, 4, 3, 2, 1, 1. Step 1 is satisfied, and we begin the loop of steps 2 – 5. The tests in steps 2 and 3 do not immediately halt us and repeating the loop of steps 2 – 5, we obtain the following collection of intermediate sequences:

$$S_1\text{: } 5,\ 4,\ 4,\ 3,\ 2,\ 1,\ 1$$
$$S_2\text{: } 3,\ 3,\ 2,\ 1,\ 1,\ 0$$
$$S_3\text{: } 2,\ 1,\ 1,\ 0,\ 0$$
$$S_4\text{: } 0,\ 0,\ 0,\ 0$$

Thus, we have determined by Algorithm 1.5.1 that S is graphical. To construct a graph with degree sequence S, we begin with the last sequence S_4. Clearly, S_4 is the

degree sequence of the empty graph on four vertices, say G_4. To proceed from S_4 to S_3, we must introduce a new vertex of degree 2 and produce two vertices of degree 1. That is, we simply undo step 5 of the algorithm. So insert a new vertex v_3 and make it adjacent to any two of the original vertices. Call the resulting graph G_3. Repeat this idea: Insert another vertex v_2 and join it to three vertices in G_3 to obtain the graph G_2. One final repetition of this process produces the graph G_1 with degree sequence S of Figure 1.5.3. □

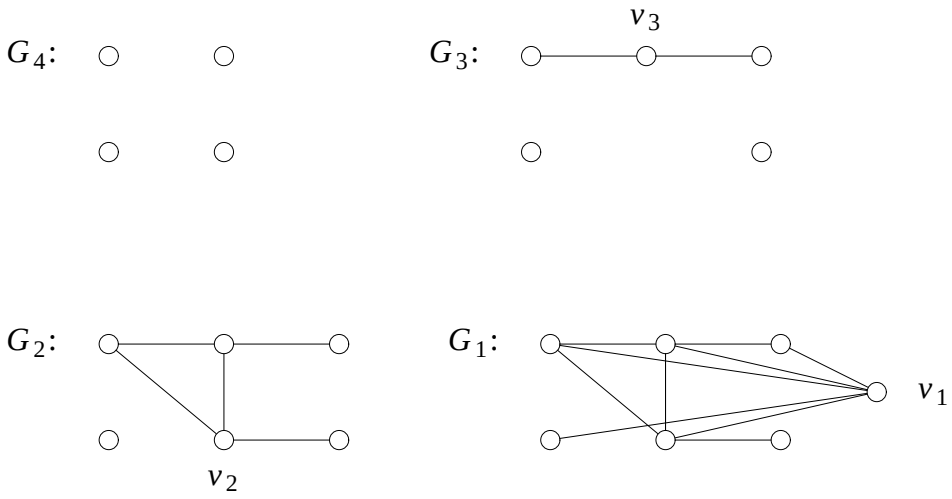


Figure 1.5.3. Reconstruction of a graph with degree sequence *S*.

An interesting result related to this construction and the proof of Theorem 1.5.1 was found independently by Eggleton [2] and by Fulkerson, Hoffman and McAndrew [5].

Theorem 1.5.2 Any realization of a graphical sequence can be obtained from any other realization by a finite sequence of edge interchanges.

An alternate result for testing graphical sequences is due to Erdös and Gallai [3].

Theorem 1.5.3 A nonincreasing sequence of nonnegative integers S: $d_1, d_2, \ldots, d_p$ $(p \geq 2)$ is graphical if, and only if, $\sum_{i=1}^{p} d_i$ is even and for each integer k, $1 \leq k \leq p - 1$, $\sum_{i=1}^{k} d_i \leq k(k-1) + \sum_{i=k+1}^{p} \min\{k, d_i\}$.

Example 1.5.2. The Erdös and Gallai system of inequalities. Suppose we apply the last result to the sequence *S*: 5, 5, 5, 5, 2, 2, 2. The sum of the terms of *S* is even (26), and thus we must examine the system of inequalities.

1. For $k = 1$, $d_1 = 5 \leq 1(0) + \sum_{i=2}^{7} \min\{1, d_i\} = 6.$
2. For $k = 2$, $d_1 + d_2 = 10 \leq 2(1) + \sum_{i=3}^{7} \min\{2, d_i\} = 2 + 10 = 12.$
3. For $k = 3$, $\sum_{i=1}^{3} d_i = 15 \leq 3(2) + \sum_{i=4}^{7} \min\{3, d_i\} = 6 + 9 = 15.$
4. For $k = 4$, $\sum_{i=1}^{4} d_i = 20 > 4(3) + \sum_{i=5}^{7} \min\{4, d_i\} = 12 + 6 = 18.$

Thus, *S* cannot be graphical, because the required inequalities break down when $k = 4$.

We can see exactly why this happens if we think about the way edges must be distributed. When $k = 4$, the first four vertices (call them *U*) all have degree 5, and so we must determine if there is room to absorb all the edges leaving *U*. Even if <U> is complete, there must still be at least eight edges from <U> to the remaining three vertices. But these three vertices all are supposed to have degree 2; hence, they cannot handle the necessary edges. If we were to examine the earlier cases, we would see that numerically there was still a possibility of success. □

Let's consider a similar question for digraphs. First, we need to decide what information we want the degree sequence of a digraph to display. Should we show the outdegrees of the vertices or the indegrees? It actually seems reasonable to display both. Thus, we really want a sequence of ordered pairs, where the first entry of the pair is the indegree of the vertex and the second entry is the outdegree. A sequence S: $(i_1, o_1), (i_2, o_2), \ldots, (i_p, o_p)$ is called *digraphical* if it is the degree sequence of some digraph. Fulkerson [4] and Ryser [9] independently discovered a characterization of digraphical sequences that is reminiscent of our last result.

Theorem 1.5.4 A sequence $S : (i_1, o_1), (i_2, o_2), \ldots, (i_p, o_p)$ of ordered pairs of nonnegative integers with $i_1 \geq i_2 \geq \cdots \geq i_p$ is digraphical if, and only if, $i_k \leq p - 1$ and $o_k \leq p - 1$ for each k, and

$$\sum_{k=1}^{p} i_k = \sum_{k=1}^{p} o_k, \text{ and } \sum_{k=1}^{j} i_k \leq \sum_{k=1}^{j} \min\{ j-1, o_k \} + \sum_{k=j+1}^{p} \min\{ j, o_k \}$$

for $1 \leq j < p$.

Section 1.6 Fundamental Counting

We have already seen several instances where it was necessary to determine how many possible items were involved in the problem under consideration. This is not unusual as we shall often find such counts to be very useful. The simplest and most useful law that aids our ability to count the number of objects in a complicated collection can be seen in the following idea: If we have "many" pigeons and "few" pigeon holes, then some pigeon hole will hold more than one pigeon. We can make this principle more precise with the aid of the following notation. By $\lfloor x \rfloor$ we mean the greatest integer less than or equal to x, and by $\lceil x \rceil$ we mean the least integer greater than or equal to x. Thus, for example, $\lfloor 3.71 \rfloor = 3$, and $\lceil 25.346 \rceil = 26$.

Theorem 1.6.1 (The Pigeon Hole Principle) If m pigeons are placed in k pigeon holes, then one hole will contain at least

$$\lceil m/k \rceil \text{ pigeons.}$$

Proof. Let x be the maximum number of pigeons in any one hole. Then $m \leq kx$, and so $x \geq \frac{m}{k}$. Since x is an integer, we see that $x \geq \lceil m/k \rceil$. □

We shall have occasion to use the Pigeon Hole Principle often. Although it seems obvious and its proof is very easy, it has many important consequences and generalizations. Certainly, its use is by no means restricted to graph theory.

As an illustration of the Pigeon Hole Principle, consider the following claim: In any group of six people, there are either three mutual acquaintances or three mutual nonacquaintances.

We can model this group of people using graphs. Represent each person as a vertex; two vertices are joined by an edge if, and only if, the corresponding people are acquaintances. What graph theoretic property will allow us to verify the claim? Three mutual acquaintances would induce a K_3 in the graph, while three mutual nonacquaintances would induce a $\overline{K}_3$. Thus, we can restate the claim as follows.

Theorem 1.6.2 Any graph on six vertices contains an induced K_3 or an induced $\overline{K}_3$ as a subgraph.

Proof. Let v be any vertex (that is, any person). By the Pigeon Hole Principle, of the remaining five vertices, either three are adjacent to v (acquaintances) or three are not adjacent to v (nonacquaintances). First, suppose that three are adjacent to v. If any two of these neighbors of v are themselves adjacent, then a K_3 is formed. If no pair of these three vertices are adjacent, then a $\overline{K}_3$ is formed by these vertices. A similar argument applies when we assume v has three vertices not adjacent to it, and so the result follows. □

Another example of a fundamental counting technique occurs when we are trying to count the number of available choices we have in performing some graph operation. For example, suppose we want to determine the number, $num(p, G)$, of labeled graphs on a set of p vertices. It is a straightforward matter to see that any one edge can be placed in this set of p vertices in $\binom{p}{2}$ different ways; we simply choose two of the p vertices as the end vertices of the edge. But in how many ways can we place k edges? Since there are $N = \binom{p}{2}$ possible ways to place an edge (the well-known binomial coefficient describing the number of ways of selecting two objects from p objects), there are $\binom{N}{k}$ ways of placing k edges in the p vertices. We can determine the total number of graphs on this set of vertices by summing these values over all possible sizes for our graph, that is:

$$num(p, G) = \sum_{q=0}^{N} \binom{N}{q}.$$

But it is a well-known property of the binomial coefficients that this sum equals 2^N. Hence, we have proved the following result.

Theorem 1.6.3 If $N = \binom{p}{2}$, then there are 2^N labeled graphs on p vertices.

We continue this approach with the following result.

Theorem 1.6.4 The number of subgraphs of K_n isomorphic to P_k is

$$\frac{n\,!}{2(n-k)\,!}.$$

Proof. Recall that P_k is a path on k vertices. Clearly, if $k > n$ there can be no such paths. Thus, suppose that $k \leq n$. If we begin at an arbitrary vertex, there are n choices for the first vertex. Clearly, there are $n - 1$ choices for the second vertex, $n - 2$ for the third vertex and so on. Thus, the number of choices is

$$n(n-1)(n-2)\cdots(n-(k-1)).$$

However, in counting the choices we have actually counted each path twice, since we could simply reverse the order of selection of the vertices and obtain the same path. Thus, the total number of subgraphs of K_n isomorphic to P_k is

$$\frac{n(n-1)\cdots(n-(k-1))}{2} = \frac{n\,!}{2(n-k)!}.\square$$

Exercises

1. Determine as many isomorphisms as you can between the graphs G_1 and G_2 of Figure 1.1.5.
2. Define the complement of G using set differences.
3. Represent $K_{p_1,\,p_2,\,\ldots,\,p_n}$ as the join of graphs.
4. Prove that $G_1 \times G_2$ is isomorphic to $G_2 \times G_1$.
5. Determine a result analogous to Theorem 1.3.1 for digraphs.
6. Give examples to show that there are walks that are not trails and trails that are not paths.
7. Given a (p_1, q_1) graph G_1 and a (p_2, q_2) graph G_2, determine formulas for the order and size of $\overline{G_1}$, $G_1 \cup G_2$, $G_1 \times G_2$ and $G_1\ [G_2]$.
8. Prove or disprove: The graph $G_1\ [G_2]$ is isomorphic to $G_2\ [G_1]$.
9. Prove that if two graphs are isomorphic, then they have the same order and size and degree sequence.
10. Find all nonisomorphic graphs of order 4.

11. Show that two graphs G and H are isomorphic if, and only if, there are two bijections (1-1 and onto functions) $f_1 : V(G) \to V(H)$ and $f_2 : E(G) \to E(H)$ such that $e = uv \in E(G)$ if, and only if, $f_2(e) = f_1(u)\ f_2(v)$.

12. Prove that a (p, q) graph G is a complete graph if, and only if, $q = \binom{p}{2}$.

13. Determine the order and size of $K_{p_1, p_2, \ldots, p_n}$, $n \geq 2$.

14. A (p, q) graph G is *self complementary* if G is isomorphic to $\overline{G}$. Show that if G is self complementary, then $p \equiv 0,1 (\text{ mod } 4)$.

15. Suppose $\Delta(G) = k$. Prove that there exists a supergraph H of G (that is, a graph H that contains G as a subgraph) such that G is an induced subgraph of H and H is k-regular.

16. Prove that if G is a regular nonempty bipartite graph with partite sets V_1 and V_2, then $| V_1 | = | V_2 |$.

17. Determine all nonisomorphic digraphs of order 4.

18. Characterize the matrices that are adjacency matrices of digraphs, that is, those matrices $A(D) = [a_{ij}]$ where $a_{ij} = 1$ if $v_i \to v_j \in E(D)$ and $a_{ij} = 0$ otherwise.

19. Determine which of the following sequences is graphical, and for those that are graphical, find a realization of the sequence.
    a. 5, 5, 5, 3, 3, 2, 2, 2, 2, 2
    b. 7, 6, 5, 5, 4, 3, 2, 2, 2
    c. 4, 4, 3, 2, 1, 0

20. Show that the sequence $d_1, d_2, \ldots, d_p$ is graphical if, and only if, the sequence $p - d_1 - 1, p - d_2 - 1, \ldots, p - d_p - 1$ is graphical.

21. The *degree set* of a graph G is the set of degrees of the vertices of G.

    a. Show that every set $S = \{ a_1, a_2, \ldots, a_k \}$ $(k \geq 1)$ of positive integers with $a_1 < a_2 < \cdots < a_k$ is the degree set of some graph.

    b. Prove that if $u(S)$ is the minimum order of a graph with degree set S, then $u(S) = a_k + 1$.

    c. Find a graph of order 7 with degree set $S = \{ 3, 4, 5, 6 \}$.

22. Show that every graph of order n is isomorphic to a subgraph of K_n.

23. Show that every subgraph of a bipartite graph is bipartite.

24. Let G be a bipartite graph. If A_{21} is the transpose of A_{12}, show that the vertices of G can be partitioned so that the adjacency matrix of G has the following form:

$$\begin{bmatrix} 0 & A_{12} \\ A_{21} & 0 \end{bmatrix}$$

25. Let G be a (p, q) graph and let t be an integer, $1 < t < p - 1$. Prove that if $p \geq 4$ and all induced subgraphs of G on t vertices have the same size, then G is isomorphic to K_p or $\overline{K_p}$.

26. Let G be a (p, q) graph. Show that $\delta(G) \leq \dfrac{2q}{p} \leq \Delta(G)$.

27. Show that the entries on the diagonal of A^2 are the degrees of G.

28. Show that any degree sequence of a nontrivial graph has two equal terms.

29. Show that $d_1, d_2, \ldots, d_p$ is the degree sequence of a multigraph if and only if $\sum_{i=1}^{p} d_i$ is even and $d_1 \leq \sum_{i=2}^{p} d_i$.

30. The *line graph* $L(G)$ of a nonempty (p, q) graph G is that graph with $V(L(G)) = E(G)$ and such that two vertices in $L(G)$ are adjacent if, and only if, the corresponding edges in G are adjacent. If G has degree sequence $d_1, d_2, \ldots, d_p$, determine formulas for the order and size of $L(G)$.

31. Find $L(K_{2,3})$.

32. Assuming no human head has more than 2,000,000 hairs on it, show that there are at least two people in New York City with exactly the same number of hairs on their heads.

33. Show that if the digits $1, 2, \ldots, 10$ are used to randomly label the vertices of a C_{10} (no label is repeated), that the sum of the labels on some set of three consecutive vertices along the cycle will be at least 17.

34. Show that there exist graphs on five vertices that do not contain an induced K_3 or $\overline{K_3}$.

35. If we color the vertices of C_{11} either red, white, blue or green, what can be said about the order of the largest subgraph each of whose vertices has the same color?

36. Prove that in any group of $p \geq 2$ people, there are always two people that have the same number of acquaintances.
37. How many subgraphs isomorphic to $K_{1,3}$ are in K_n?
38. How many subgraphs isomorphic to C_t are in K_n?
39. How many subgraphs isomorphic to C_4 are in $K_{m,n}$?
40. How many subgraphs isomorphic to P_5 are in $K_{m,n}$?
41. If three men check their hats at a club, in how many ways can the hats be returned so that no man receives his own hat?
42. (*) Prove that any sequence of $n^2 + 1$ distinct integers contains either an increasing subsequence of $n + 1$ terms or a decreasing subsequence of $n + 1$ terms.
43. Find a sequence of n^2 ($n \geq 3$) distinct integers that does not contain an increasing subsequence of $n + 1$ terms or a decreasing subsequence of $n + 1$ terms.
44. (*) Prove that if $n + 1$ numbers are selected from the set $\{ 1, 2, \ldots, 2n \}$, then one of these numbers will divide a second one of these numbers.
45. (*) If every vertex of G has degree k, then
    a. k is an eigenvalue of G
    b. if G is connected, then the multiplicity of k is one,
    c. for any eigenvalue λ, $|\lambda| \leq k$.

References

1. Biggs, N. L., *Algebraic Graph Theory, 2nd Edition.* Cambridge University Press, London (1993).
2. Eggleton, R. B., Graphic Sequences and Graphic Polynomials. A report in: *Infinite and Finite Sets,* Vol. 1, ed. by A. Hajnal. Colloq. Math. Soc. J. Bolyai 10 (North Holland, Amsterdam, 1975), 385 – 392.
3. Erdös, P., and Gallai, T., Graphs with Prescribed Degrees of Vertices (Hungarian). *Mat. Lapok* 11(1960) 264 – 274.

4. Fulkerson, D. R., Upsets in Round Robin Tournaments. *Canad. J. Math.*, 17(1965), 957 – 969.

5. Fulkerson, D. R., Hoffman, A. J., and McAndrew, M. H., Some Properties of Graphs with Multiple Edges. *Canad. J. Math.*, 17(1965), 166 – 177.

6. Garey, M. R., and Johnson, D. S., *Computers and Intractability, A Guide to the Theory of NP-Completeness*. Freeman, New York, 1979.

7. Hakimi, S. L., On the Realization of a Set of Integers as Degrees of the Vertices of a Graph. *J. SIAM Appl. Math.*, 10(1962), 496 – 506.

8. Havel, V., A Remark on the Existence of Finite Graphs (Czech.), *Casopis Pest. Mat.*, 80(1955), 477 – 480.

9. Ryser, H. J., Combinatorial Properties of Matrices of Zeros and Ones. *Canad. J. Math.*, 9(1957), 371 – 377.

10. Sabidussi, G., Graph Multiplication. *Math. Z.*, 72(1960), 446 – 457.