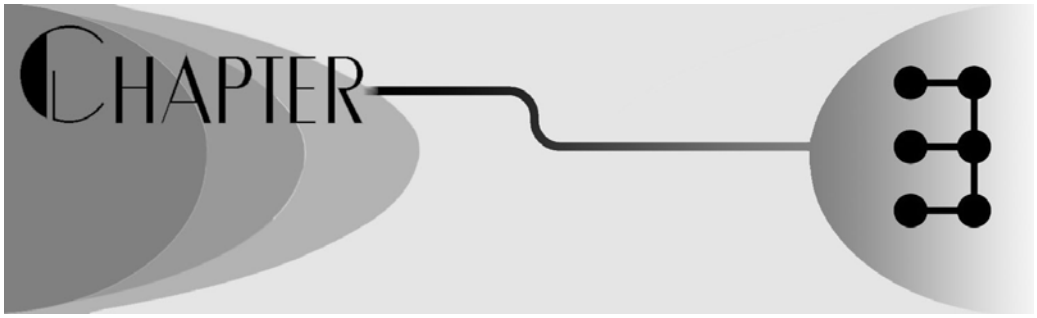


Some Special Properties of Graphs

Chapter Goals

- Define and study Eulerian graphs.
- Examine several important operations on graphs.
- Study bipartite graphs.
- Define and study Hamiltonian paths and cycles.
- Take a look at the well-known traveling salesman problem.

3.1 Introduction

We have seen a number of the basic concepts in graph theory. In this chapter we look at some special properties of graphs. In §3.2 we define and study Eulerian graphs. The techniques involved here were used to solve one of the first problems in graph theory. Next we examine several important operations on graphs in §3.3. An important class of graphs called bipartite graphs is introduced in §3.4. We revisit Eulerian graphs and some of their properties in §3.5. In §3.6 and §3.7 we look at Hamiltonian paths and cycles. Then in §3.8 we present one of the most famous problems in graph theory, the traveling salesman problem.

3.2 Eulerian Graphs

As mentioned in Chapter 1, graph theory was born in 1736 with Euler's famous paper in which he solved the Königsberg bridge problem (Problem 1.1). In the same paper Euler posed and solved a more general problem: For what type of graph G is it possible to find a closed walk running through every edge of G exactly once? Such a walk is now called an *Eulerian circuit*. A graph that consists of an Eulerian circuit is called an *Eulerian graph*. More formally, we have the following definition.

Definition 3.1
Let G be a graph. If some closed walk in G contains every edge in $E(G)$, then the walk is an Eulerian circuit *and the graph G is an* Eulerian graph.

By definition a walk is always connected. Since an Eulerian circuit (which is a walk) contains all of the edges of a graph, an Eulerian graph is always connected except for any isolated vertices. Because isolated vertices do not contribute anything to the understanding of an Eulerian graph, we will assume that Eulerian graphs do not have isolated vertices and are connected.

Now we state and prove an important theorem that will enable us to tell immediately whether or not a given graph is an Eulerian graph.

Theorem 3.2
A connected graph G is an Eulerian graph if and only if the degree of every vertex in G is even.

PROOF: Suppose G is an Eulerian graph. Then it contains an Eulerian circuit (which is a closed walk). In tracing this walk we observe that for each vertex v, the walk contains a pair of edges incident with v—one we "entered" v with and the other we "exited" v with. This is true of all intermediate vertices of the walk. It is also true of the terminal vertex because we "exited" and "entered" the terminal vertex at the beginning and end of the walk. Thus if G is an Eulerian graph, the degree of every vertex is even.

To prove the sufficiency of the condition, assume that all of the vertices of G are of even degree. Construct a walk h starting at an arbitrary vertex v and going through the edges of G such that no edge is traced more than once. We continue tracing from v as long as possible. Because every vertex is of even degree, we can exit from every vertex we enter; the tracing cannot stop at any

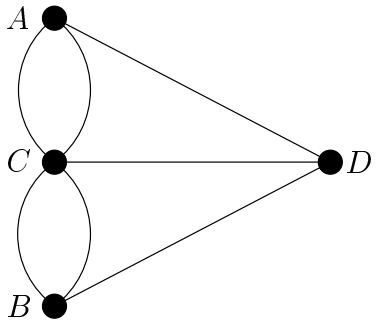


Figure 3.1: Graph of Königsberg bridge problem.

vertex but v. Since v is also of even degree, we shall eventually reach v when the tracing comes to an end. If this closed walk h includes all the edges of G, it follows that G is an Eulerian graph. If not, we remove from G all of the edges in h and obtain a subgraph G' of G formed by the remaining edges. Since the degree of each vertex in G and h is even, the degree of each vertex in G' is also even. Moreover, G' must touch h at least at one vertex u, because G is connected. Starting from u, we can again construct a new walk in graph G'. Since all of the vertices of G' are of even degree, this walk in G' must terminate at vertex u. Notice that this walk in G' can be combined with h to form a new walk, which starts and ends at vertex v and has more edges than h. This process can be repeated until we obtain a closed walk that traverses all of the edges of G. Thus G is an Eulerian graph. □

Looking at the graph of the Königsberg bridge problem shown in Figure 3.1, we find that not all of its vertices are of even degree. Hence, it is not an Eulerian graph. Thus it is not possible to walk over each of the seven bridges exactly once and return to the starting point. Notice that in solving the Königsberg bridge problem Euler proved a much more general result than necessary. In graph theory it often happens that proving something more general is easier than proving the bare minimum.

Eulerian circuits often occur in various puzzles. The common theme of these puzzles is to draw a given picture in one continuous line without retracing and without lifting the pencil from the paper. Two such pictures are shown in Figure 3.2. The drawing in Figure 3.2(a) is called *Mohammed's scimitars* and is believed to have come from the Arabs. The drawing in Figure 3.2(b) is the *star of David*. It is a good exercise to trace these Eulerian graphs.

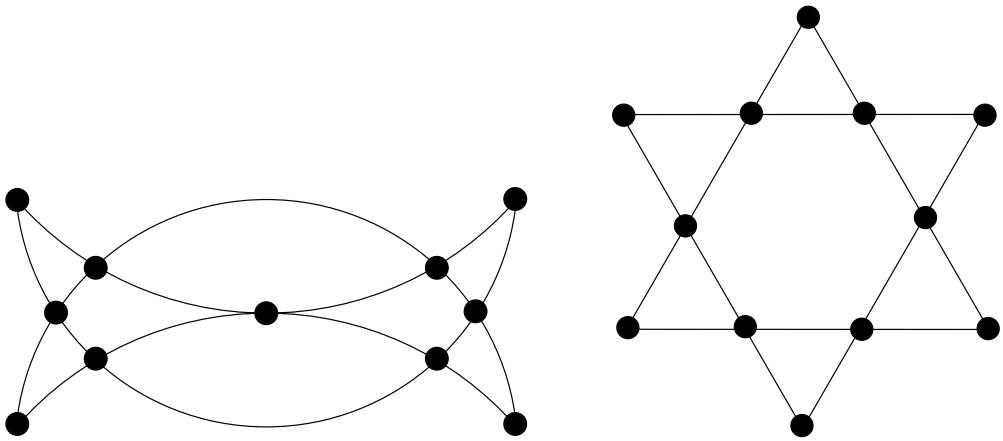


Figure 3.2: Two Eulerian graphs.

In defining an Eulerian circuit some authors drop the requirement that the walk be closed. For example, the walk $(v_1, v_3, v_4, v_1, v_2, v_4, v_5, v_2)$ in Figure 3.3, includes each edge of the graph and does not retrace any edge, but is not closed. The initial vertex is v_1 and the final vertex is v_2. This type of walk is described in the following definition.

Definition 3.3
For a graph G, a trail that includes (traces or covers) all edges of G without retracing any edge is an Eulerian trail. *A graph that has an Eulerian trail is called a* unicursal graph.

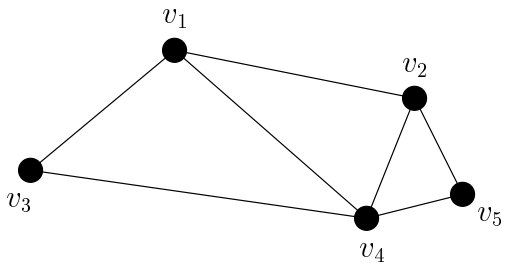


Figure 3.3: Unicursal graph.

Clearly, adding an edge between the initial and final vertices of an Eulerian trail produces an Eulerian circuit. Thus a connected graph is unicursal if and only if it has exactly two vertices of odd degree. This observation is generalized in the following theorem.

Theorem 3.4
Let k be a natural number. If G is a connected graph with $2k$ vertices of odd degree, then there exist k edge-disjoint subgraphs such that

1. *together they contain all of the edges of G and*

2. *each subgraph is a unicursal graph.*

PROOF: Let the odd vertices of G be $v_1, \ldots, v_k; w_1, \ldots, w_k$ in arbitrary order. We add the following k edges to G $\{v_1, w_1\}, \ldots, \{v_k, w_k\}$ to form a new graph G'. Since every vertex of G' is of even degree, G' consists of an Eulerian circuit ρ. If we remove from ρ the k edges just added (no two of these edges are incident on the same vertex), ρ is split into k walks. Each of these walks is an Eulerian trail. That is, the first edge removal leaves a single Eulerian trail; the second edge removal splits that into two Eulerian trails; each successive edge removal splits an Eulerian trail into two Eulerian trails until there are k of them. Thus we have the theorem. □

We shall interrupt our study of Eulerian graphs to define some commonly used graph-theoretic operations. One of these operations is required immediately in the next section while others are used later.

3.3 Operations on Graphs

As with most mathematical entities, it is convenient to consider a large object as a combination of smaller ones and to derive properties of the larger object from those of the smaller ones. This holds true for graphs. Since graphs are defined in terms of sets of vertices and edges, it is natural to employ the set-theoretical terminology to define operations between graphs. The following definition captures a number of important operations.

Definition 3.5
Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be simple graphs. The union *of G_1 and G_2 is the graph*

$$G_1 \cup G_2 = (V_1 \cup V_2, E_1 \cup E_2).$$

If $V_1 \cap V_2 \neq \emptyset$, then the intersection *of G_1 and G_2 is the graph*

$$G_1 \cap G_2 = (V_1 \cap V_2, E_1 \cap E_2).$$

The symmetric difference *of G_1 and G_2 is the graph*

$$G_1 \triangle G_2 = (V_1 \cup V_2, E_1 \triangle E_2).$$

Note, the union, intersection, or symmetric difference of two simple graphs results in a simple graph.

For general graphs the same ideas hold. However, we need to be more careful in our definitions. Figure 3.4 demonstrates how the union, intersection, and symmetric difference of two graphs are defined. As illustrated in the figure, if an edge is in both G_1 and G_2, then the edge's endvertices in G_1 must be the same as in G_2. Otherwise, these three operations cannot be sensibly defined. To be precise, we make the following definition that generalizes Definition 3.5.

Definition 3.6
Let $G_1 = (V_1, E_1, \phi_1)$ and $G_2 = (V_2, E_2, \phi_2)$ be subgraphs of a universal graph $G = (V, E, \phi)$.

The union *of G_1 and G_2 is the graph*

$$G_1 \cup G_2 = (V_1 \cup V_2, E_1 \cup E_2, \phi_{1\cup 2}),$$

where $\phi_{1\cup 2}$ is the restriction of ϕ to $E_1 \cup E_2 \subseteq E$.

If $V_1 \cap V_2 \neq \emptyset$, then the intersection *of G_1 and G_2 is the graph*

$$G_1 \cap G_2 = (V_1 \cap V_2, E_1 \cap E_2, \phi_{1\cap 2}),$$

where $\phi_{1\cap 2}$ is the restriction of ϕ to $E_1 \cap E_2 \subseteq E$.

The symmetric difference *of G_1 and G_2 is the graph*

$$G_1 \triangle G_2 = (V_1 \cup V_2, E_1 \triangle E_2, \phi_{1\triangle 2}),$$

where $\phi_{1\triangle 2}$ is the restriction of ϕ to $E_1 \triangle E_2 \subseteq E$.

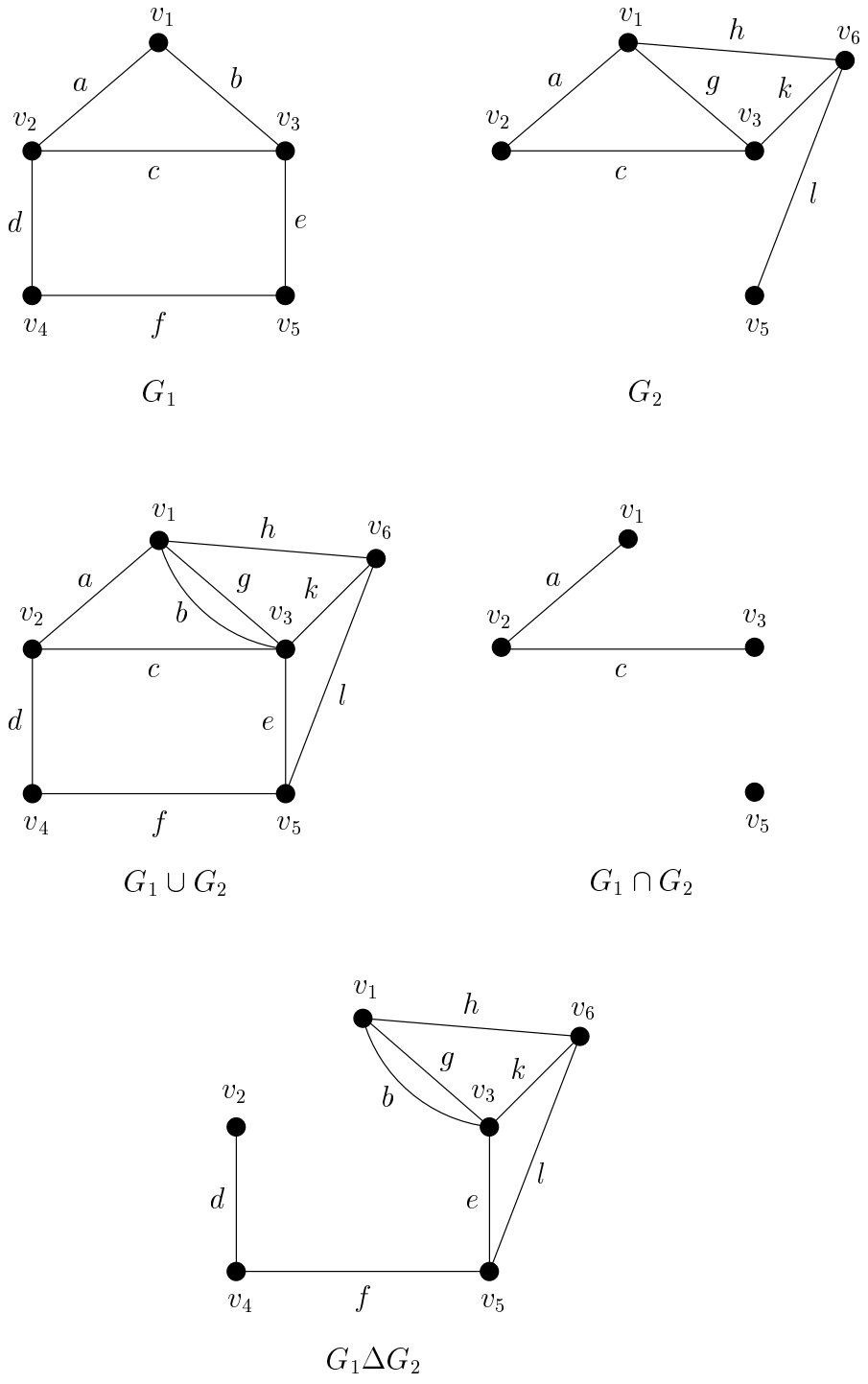


Figure 3.4: Union, intersection, and symmetric difference of two graphs.

Figure 3.4 illustrates Definition 3.6. Note that if we view G_1 and G_2 as simple graphs, the figure also can be used to illustrate Definition 3.5. In both the union and symmetric difference there is only a single edge between v_1 and v_3.

➤ **Note 3.7**

Our assumption stating that both G_1 and G_2 are subgraphs of a universal graph G implies that both ϕ_1 and ϕ_2 agree on $E_1 \cap E_2$. That is, for every $e \in E_1 \cap E_2$ we have

$$\phi_1(e) = \phi_2(e). \tag{3.1}$$

If we had not made the assumption that G_1 and G_2 were subgraphs of G, we would have had to insist that Equation (3.1) hold for all $e \in E_1 \cap E_2$. This condition yields an equivalent definition.

It is obvious from their definitions that union, intersection, and symmetric difference are commutative operations. That is, we have the following:

$$\begin{aligned} G_1 \cup G_2 &= G_2 \cup G_1, \\ G_1 \cap G_2 &= G_2 \cap G_1, \text{ and} \\ G_1 \vartriangle G_2 &= G_2 \vartriangle G_1. \end{aligned}$$

One can also show that these operations are associative. (We ask for the proof that symmetric difference is associative in Exercise 8.) That is, we have the following:

$$\begin{aligned} (G_1 \cup G_2) \cup G_3 &= G_1 \cup (G_2 \cup G_3), \\ (G_1 \cap G_2) \cap G_3 &= G_1 \cap (G_2 \cap G_3), \text{ and} \\ (G_1 \vartriangle G_2) \vartriangle G_3 &= G_1 \vartriangle (G_2 \vartriangle G_3), \end{aligned}$$

The definitions of union, intersection, and symmetric difference of graphs can be extended in an obvious way to include any finite number of graphs. Because the associativity property holds for each of these three operations, it makes sense to drop the parentheses when we are considering an operation on more than two graphs. For instance, $G_1 \vartriangle G_2 \vartriangle \cdots \vartriangle G_n$ is the graph on $V_1 \cup \cdots \cup V_n$

that contains all those edges included in an odd number of the graphs G_i, for $1 \leq i \leq n$.

Note that if G_1 and G_2 are vertex-disjoint, then $G_1 \cap G_2$ is not a graph. Recall graphs must have at least one vertex. If G_1 and G_2 are edge-disjoint, then both $G_1 \cap G_2$ and $G_1 \triangle G_2 = G_1 \cup G_2$ are null graphs having the same number of vertices. For any graph G, we have

$$\begin{aligned} G \cup G &= G \cap G = G, \text{ and} \\ G \triangle G &= N_{|V(G)|}. \end{aligned}$$

Recall that $N_{|V(G)|}$ is the null graph on $|V(G)|$ vertices.

Suppose G' is a subgraph of G. Then by definition $G \triangle G'$ is the subgraph of G that remains after all of the edges in G' have been removed from G. Therefore, $G \triangle G'$ is written as $G - G'$ whenever $G' \subseteq G$. Because of this complementary nature, $G \triangle G' = G - G'$ is often called the *complement* of G' in G.

We have seen how to combine graphs in three different ways. The next definition shows one way of dividing a graph into two pieces.

Definition 3.8
A graph G that is not a null graph is decomposed *into two subgraphs G_1 and G_2 if*

$$\begin{aligned} G_1 \cup G_2 &= G, \text{ and} \\ G_1 \cap G_2 &= N, \end{aligned}$$

where N is some null graph, and neither G_1 nor G_2 is a null graph.

The definition says that every edge of G occurs in either G_1 or G_2, but not both. On the other hand, some of the vertices may occur in both G_1 and G_2. In decomposition isolated vertices are discarded. A graph containing m edges $\{e_1, \ldots, e_m\}$ can be decomposed in $2^{m-1} - 1$ different ways into pairs of subgraphs G_1 and G_2. (Why?)

Graphs also can be decomposed into more than two subgraphs. We require that the subgraphs be pairwise edge-disjoint and collectively include every edge of the original graph.

In graph theory we often need to discuss a graph that has had a vertex or an edge deleted from it. The following definition is useful for doing this.

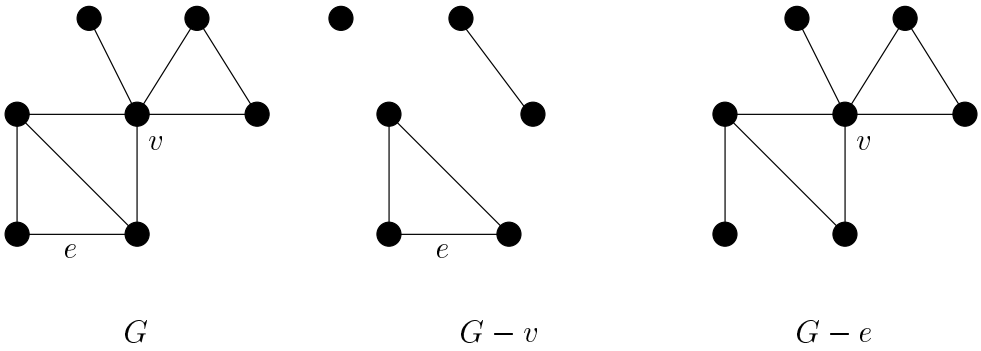


Figure 3.5: Vertex deletion and edge deletion.

Definition 3.9
Let $G = (V, E, \phi)$ be a graph. If v is a vertex in V, then

$$G - v = G[V \setminus \{v\}]$$

is the subgraph of G that is obtained by deleting v from V. When a vertex is deleted, so are all edges incident to it.

Let ϕ' be the restriction of ϕ to $E \setminus \{e\}$. If e is an edge in E, then

$$G - e = G(V, E \setminus \{e\}, \phi')$$

is the subgraph of G that is obtained by deleting e from E. Note that when an edge is deleted, the edge's endvertices are not deleted.

Figure 3.5(a) provides an example of deleting a vertex from a graph and Figure 3.5(b) provides an example of deleting an edge from a graph.

Here are some useful generalizations of Definition 3.9.

➤ **Note 3.10**

1. *From Definition 3.9, deletion of an edge from a graph, we can write*

$$G - e = G \triangle e.$$

In $G \triangle e$ we view the edge e in G as a subgraph including the endvertices of e.

2. *We can delete more than one vertex from a graph. Thus Definition 3.9 can be generalized to define $G - V'$ for any $V' \subseteq V$.*

3. *Definition 3.9 can be generalized to define $G - E'$ for any $E' \subseteq E$. That is, we can delete any set of edges from a graph. Note that if G' is a subgraph of G with edge set $E' \subseteq E$, then $G - G'$, the complement of G' in G, is really the graph $G - E'$.*

Removing a vertex or edge from a graph G can increase the number of components of G.

Definition 3.11
A vertex v is a cut-vertex *if $G - v$ has more components than G. An edge e is a* cut-edge *if $G - e$ has more components than G.*

Next we define the graph obtained by collapsing an edge in it.

Definition 3.12
Let $G = (V, E, \phi)$ be a graph and e an edge of E having distinct endvertices u and v. Let w be a new vertex such that $w \notin V$. Let α be the mapping $\alpha : V \to (V \setminus \{u, v\}) \cup \{w\}$ defined by

$$\alpha(x) = \begin{cases} x & \text{if } x \notin \{u, v\} \\ w & \text{if } x \in \{u, v\} \end{cases}$$

The contraction *of G by e is the graph*

$$G \cdot e = ((V \setminus \{u, v\}) \cup \{w\}, E \setminus \{e\}, \alpha' \circ \phi),$$

where $\alpha'(\{x, y\}) = \{\alpha(x), \alpha(y)\}$ for any vertices x and y.

In Definition 3.12, the graph $G \cdot e$ is formed from G by replacing the vertices u and v by the single new vertex w. Except for the edge e, every edge that was incident to u, v, or both is now incident to the new vertex w. Figure 3.6 provides an example of a graph before and after an edge has been contracted.

➤ **Note 3.13**
Observe that $\alpha' \circ \phi$ is only applied to $E \setminus \{e\}$ but not all of E. Although $(\alpha' \circ \phi)(e) = \{w\}$ is well-defined, this loop is not contained in $G \cdot e$. Hence, we see that a contraction by an edge reduces the number of edges by one and the number of vertices by one. Figure 3.6 illustrates that contraction is easy to understand geometrically. Although formal definitions can be somewhat involved in graph theory, you should always keep the geometrical pictures in mind.

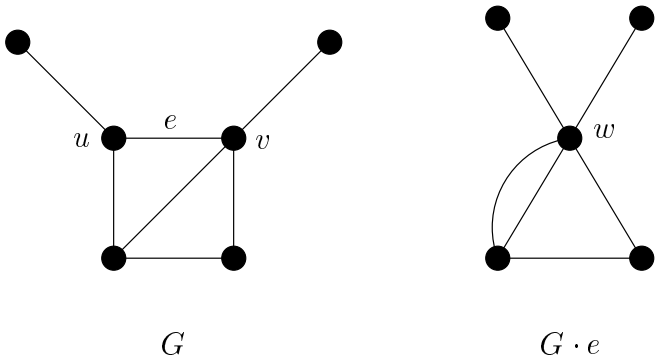


Figure 3.6: Contraction by the edge $e = \{u, v\}$.

We see that if G is a simple graph and e is an edge in G then the contracted graph $G \cdot e$ is not necessarily simple. We can define a related *simple contraction* by an edge e that will produce a simple graph by simply discarding the multiple edges in the contracted graph $G \cdot e$.

Definition 3.14
Let $G = (V, E)$ be a simple graph and e an edge in E having distinct endvertices u and v. Let w be a new vertex such that $w \notin V$. Let α be a mapping defined as follows: $\alpha : V \to (V \setminus \{u, v\}) \cup \{w\}$ is

$$\alpha(x) = \begin{cases} x & \text{if } x \notin \{u, v\} \\ w & \text{if } x \in \{u, v\} \end{cases}$$

The simple contraction *of G by e is the graph*

$$G/e = ((V \setminus \{u, v\}) \cup \{w\}, \alpha'(E \setminus \{e\})),$$

where $\alpha'(\{x, y\}) = \{\{\alpha(x), \alpha(y)\} : \{x, y\} \in E \setminus \{e\}\}$.

Figure 3.7 provides an example of the simple contraction of an edge.

In this section we have examined some of the elementary operations on graphs. These operations provide us with mechanisms for generating additional graphs from existing ones. More complex operations have been defined and are used in graph theory research. For a survey of such operations see Harary and Wilcox [16].

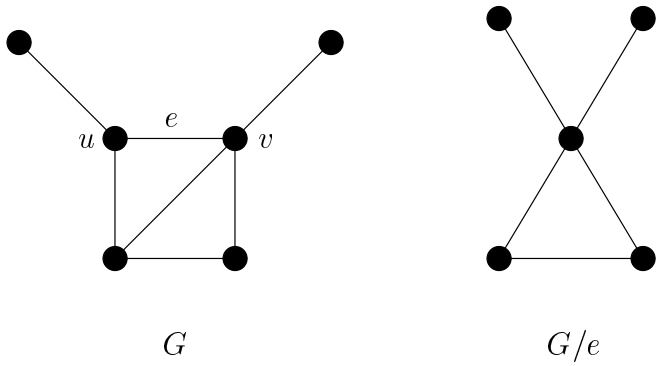


Figure 3.7: Simple contraction by the edge $e = \{u, v\}$.

3.4 Bipartite Graphs

In this section we describe a type of graph that is used in a wide range of applications. We begin with its definition.

Definition 3.15
A graph G is a bipartite graph *if $V(G)$ can be partitioned into two subsets X and Y meeting the following conditions:*

1. $V(G) = X \cup Y$,
2. $X \cap Y = \emptyset$, *and*
3. *both $G[X]$ and $G[Y]$ are null graphs.*

The word "bipartite" means two parts. If you keep this in mind, you will have no trouble remembering Definition 3.15.

➤ **Note 3.16**
We have the following observations.

1. *A bipartite graph with partition $V(G) = X \cup Y$ implies each edge of G has one endvertex in X and the other in Y.*
2. *Every null graph is bipartite; any partition $V(G) = X \cup Y$ demonstrates this fact.*

3. *A bipartite general graph cannot have any loops.*

4. *A graph is bipartite if and only if each of its components is bipartite.*

5. *The complete bipartite graph $K_{m,n}$ is a bipartite simple graph.*

We can see that all cycles in a bipartite graph have even length. This simple property is in fact a characterizing property for bipartite graphs stated in this next theorem.

Theorem 3.17
A graph G is bipartite if and only if each cycle has even length. If G is connected, then the partition $V(G) = X \cup Y$ is uniquely determined.

Before proving this theorem, we need the following observation.

Observation 3.18
If G contains a circuit having odd length, then it contains a cycle having odd length.

PROOF: Assume that $p = (u_0, e_1, u_1, \ldots, e_n, u_n)$ is circuit of shortest possible odd length n. We show that p is actually a cycle. If p is not a cycle, then there are two index numbers j and k such that $0 \leq j < k \leq n$, $(j, k) \neq (0, n)$, and $u_j = u_k$. In this case we have two circuits $p_1 = (u_0, e_1, u_1, \ldots, u_j, e_k, u_{k+1}, \ldots, u_n)$ and $p_2 = (u_j, e_j, u_{j+1}, \ldots, e_{k-1}, u_k)$. The lengths of p_1 and p_2 are both positive and sum up to n. Hence, one of these lengths must be odd. Thus we have a shorter odd circuit than p. This contradicts our initial assumption. Therefore, p must be a cycle. □

➤ **Note 3.19**
There are graphs containing circuits of even lengths that do not contain any cycles of even lengths.

We will use Observation 3.18 to prove Theorem 3.17.

PROOF: (Theorem 3.17) Without loss of generality, we can assume that G is connected. Assume first that $V(G) = X \cup Y$ is a partition of the vertices such that both $G[X]$ and $G[Y]$ are null graphs. Let $p = (u_0, e_1, u_1, \ldots, e_n, u_n)$ be a circuit. Since each edge in G has one endvertex in X and the other in Y, we see that for each i the vertices u_i and u_{i+1} of the circuit p are always in

opposite parts. That is, one is in X and the other is in Y. Since p is a closed trail $u_0 = u_n$ must both be in X or both be in Y. Hence, n must be even.

For the second half of the theorem, assume that each cycle of G has even length. We show that G is bipartite. Choose a fixed vertex $x \in V(G)$. Consider an arbitrary vertex $y \in V(G)$. If p and q are two paths from x to y of respective length n and m, then pq^{-1} is a circuit in G of length $n + m$. Combining our assumption that each cycle of G has even length with the contrapositive of Observation 3.18, we can conclude that $n+m$ is even. Hence, either both n and m are even or they are both odd. Therefore, we can form a partition of $V(G)$ by letting

$$\begin{aligned} X &= \{y \in V(G) : \text{ the length of each path from } x \text{ to } y \text{ is even}\}, \\ Y &= \{y \in V(G) : \text{ the length of each path from } x \text{ to } y \text{ is odd}\}. \end{aligned}$$

Since G is connected, we have $V(G) = X \cup Y$ and $X \cap Y = \emptyset$. Lastly, consider an edge e of G with endvertices u and v. We need to show that they are not contained in the same half of the partition. Suppose p is a path from x to u of length n. If $q = (u, e, v)$, the path pq is a path from x to v of length $n + 1$ that has a different remainder than n divided by two. Hence, one of u and v is contained in X and the other in Y. So, $G[X]$ and $G[Y]$ are both null graphs.

Since G is connected, every two vertices are joined by a simple path. Hence, the partition $V(G) = X \cup Y$ is uniquely determined. □

We can easily extend the definition of a bipartite graph. A graph G for which $V(G)$ can be partitioned into k greater than one sets

$$V(G) = X_1 \cup X_2 \cup \cdots \cup X_k,$$

where $X_i \cap X_j = \emptyset$ for each $i \neq j$ and each of the subgraphs $G[X_i]$ is a null graph is called a *multipartite graph*. However, the nice characterization for bipartite graphs given in Theorem 3.17 can not be extended for k greater than two.

3.5 More on Eulerian Graphs

You now have the background to learn more results on the important topic of Eulerian graphs. We begin with a theorem that characterizes Eulerian graphs.

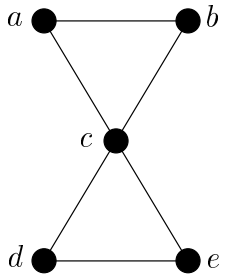


Figure 3.8: Arbitrarily traceable graph from c.

Theorem 3.20
A connected graph G is an Eulerian graph if and only if it can be decomposed into cycles.

PROOF: Suppose graph G can be decomposed into cycles. That is, G is a union of edge-disjoint cycles. Since the degree of every vertex in a cycle is two, the degree of every vertex in G is even. Hence, G is an Eulerian graph by Theorem 3.2.

Conversely, let G be an Eulerian graph. Consider a vertex v_1. There are at least two edges incident at v_1. Let one of these edges be between v_1 and v_2. Since vertex v_2 is also of even degree, it must have at least another edge, say between v_2 and v_3. Proceeding in this fashion, we eventually arrive at a vertex that has previously been traversed. In doing this we form a cycle. Call it Γ. Now we remove Γ from G. All of the vertices in the remaining graph (not necessarily connected) must also be of even degree. From the remaining graph remove another cycle in exactly the same way as we removed Γ from G. We can continue this process until no edges are left. Hence, we have our theorem. □

Consider the graph depicted in Figure 3.8, which is an Eulerian graph. Suppose that we start at vertex a and trace the path going through vertices a, b, and c. Now at c we have the choice of going to a, d, or e. If we take the first choice, we would only trace the circuit going through vertices a, b, c, and a. This is not an Eulerian circuit. Thus starting from a, we cannot trace the entire Eulerian circuit simply by moving along any edge that has not already been traversed. This raises the following interesting question:

> Let G be an Eulerian graph. We wish to obtain an Eulerian trail using the following rule. After arriving at a vertex v in any walk,

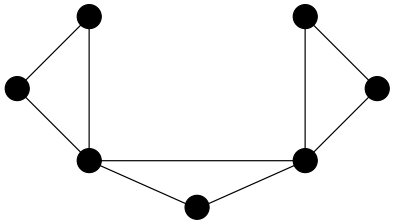


Figure 3.9: Eulerian graph that is not arbitrarily traceable.

one may select *any* edge that has not been previously traversed to continue the walk. What property must the vertex v possess so that an Eulerian graph is produced?

We provide an answer to this question but first we need the following definition.

Definition 3.21
A graph G is said to be arbitrarily traceable from v *if one always obtains an Eulerian trail by randomly traversing from v through all the edges of G, going exactly once along each edge.*

For instance, the Eulerian graph draw in Figure 3.8 is an arbitrarily traceable graph from vertex c, but not from any other vertex. The Eulerian graph shown in Figure 3.9 is not arbitrarily traceable from any vertex. The graph given in Figure 3.10 is arbitrarily traceable from every vertex. This interesting theorem due to Ore [26] answers the question just raised.

Theorem 3.22
An Eulerian graph G is arbitrarily traceable from vertex v in G if and only if every circuit in G contains v.

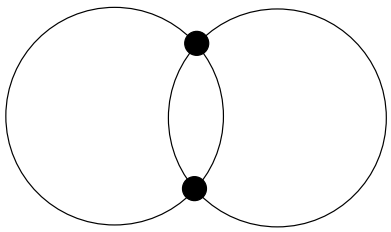


Figure 3.10: Arbitrarily traceable graph from all vertices.

For a proof of the theorem, the reader is referred to Ore [26].

Let G be an Eulerian graph and v a vertex of G. Although G is not necessarily arbitrarily traceable from v, we can still trace through all of the edges of G exactly once and end up at v. Thus we can construct an Eulerian circuit of G. Note, we use // to indicate a comment in an algorithm.

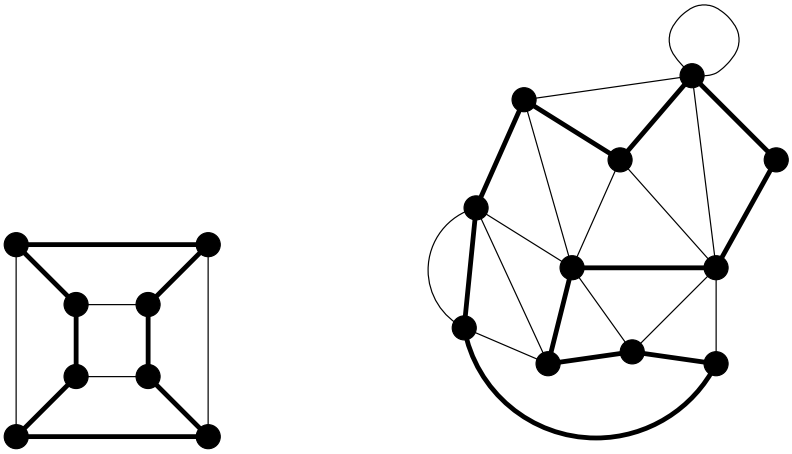
ALGORITHM EULERIAN CIRCUIT shows how to construct an Eulerian circuit of a simple graph G.

```
ALGORITHM EULERIAN CIRCUIT
Input: An Eulerian graph G = (V, E).
Output: An Eulerian circuit of G represented by a list L of vertices.
begin
    currentVertex = v ∈ V;
    L = v;      // vertices are appended to the list L
    vertices = V;
    edges = E;
    while edges ≠ ∅ do {
          pick e = {currentVertex, w} ∈ edges, where
               e is not a cut-edge unless this is the only choice;
          L = Lw;
          currentVertex = w;
          edges = edges − {e};
          }
    output L;
end.
```

The proof that ALGORITHM EULERIAN CIRCUIT actually produces an Eulerian circuit is left to Exercise 17.

3.6 Hamiltonian Paths and Cycles

An Eulerian circuit of a connected graph was characterized by the property of being a closed walk that traverses *every edge* of the graph exactly once. A *Hamiltonian cycle* in a connected graph is defined as a cycle that traverses *every vertex of* G exactly once, except the starting vertex at which the walk also terminates. For example, one gets a Hamiltonian cycle in Figure 3.11(a) by starting at any vertex and transversing along the edges shown in thick lines.



(a) (b)

Figure 3.11: Hamiltonian cycles.

Notice the cycle passes through each vertex exactly once. A Hamiltonian cycle for the graph shown in Figure 3.11(b) is also depicted by thick lines. More formally, we have the following definition.

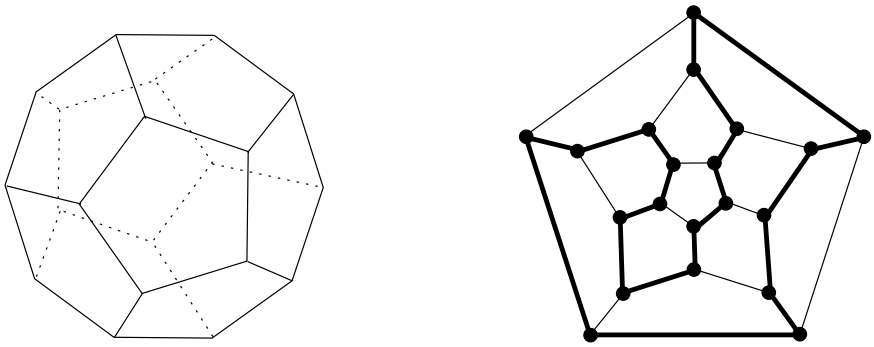
Definition 3.23
A cycle in a connected graph G is said to be Hamiltonian *if it includes every vertex of G. If G has a Hamiltonian cycle, then we say that G is a* Hamiltonian graph.

A path in G is called a Hamiltonian path *if it traverses through every vertex of G.*

Remark 3.24
A Hamiltonian cycle in a graph of n vertices consists of exactly n edges.

Since a Hamiltonian path is a subgraph of a Hamiltonian cycle (which in turn is a subgraph of another graph), every graph that has a Hamiltonian cycle also has a Hamiltonian path. There are, however, many graphs with Hamiltonian paths that have no Hamiltonian cycles (see Exercise 19). The length of a Hamiltonian path (if it exists) in a connected graph of n vertices is $n - 1$.

Obviously, not every connected graph has a Hamiltonian cycle. For example, neither of the graphs shown in Figures 3.8 nor 3.9 has a Hamiltonian cycle. This



(a) (b)

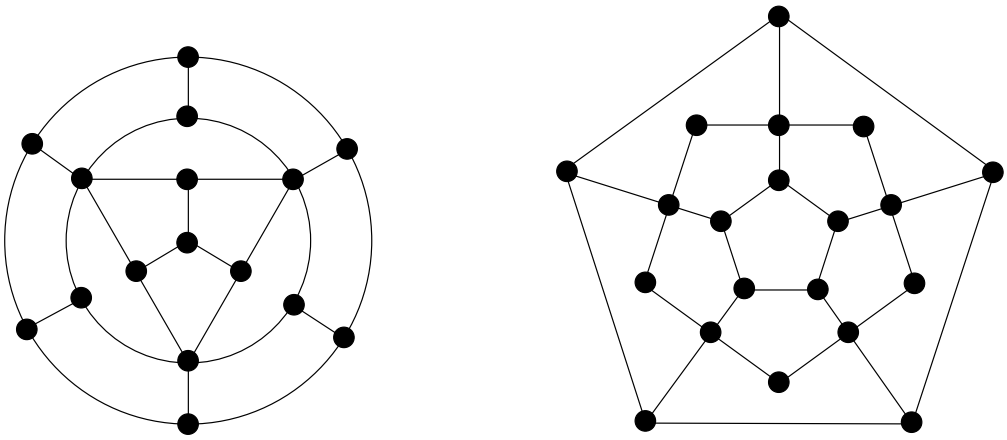
Figure 3.12: Dodecahedron and its graph shown with a Hamiltonian cycle.

raises the question: What is a necessary and sufficient condition for a connected graph G to have a Hamiltonian cycle? This problem, first posed by the famous Irish mathematician Sir William Rowan Hamilton in 1859, is still unsolved. As mentioned in Chapter 1, Hamilton made a regular dodecahedron of wood (see Figure 3.12(a)) each of whose 20 corners was marked with the name of a city. The puzzle is to start from any city and find a route along the edge of the dodecahedron that passes through every city exactly once and returns to the city of origin. The graph of the dodecahedron is given in Figure 3.12(b), and one of many such routes (a Hamiltonian cycle) is shown by thick lines.

The resemblance between the problem of computing an Eulerian circuit and that of computing a Hamiltonian cycle is deceptive. The algorithmic complexity of finding an Hamiltonian cycle in a graph is much greater than that of finding an Eulerian circuit. We revisit this point again in Chapter 15.

Although one can find Hamiltonian cycles in many specific graphs, such as those in Figures 3.11 and 3.12, there is no known criterion to determine the existence of a Hamiltonian cycle in general. However, we will show there are certain types of graphs that always contain Hamiltonian cycles.

In considering the existence of a Hamiltonian cycle (or path), we need only consider simple graphs. This is because a Hamiltonian cycle (or path) traverses every vertex exactly once. Hence, it cannot include a loop or a set of multiple edges. So, before we look for a Hamiltonian cycle in a general graph, we may make the graph simple by removing multiple edges and loops.



(a) (b)

Figure 3.13: Graphs without Hamiltonian cycles.

The following theorem shows us a condition that is necessary for a graph to be Hamiltonian, but it is far from being sufficient. We will discuss some sufficient conditions later.

Theorem 3.25
Let G be a Hamiltonian graph. For all nonempty subsets $S \subseteq V(G)$, we have that the number of components of the graph $G - S$ does not exceed $|S|$.

PROOF: Assume that C is a Hamiltonian cycle of G. Because C is a subgraph of G that includes all the vertices of G, the number of components of $G - S$ is not more than the number of components of $C - S$. Since C is a cycle, the number of components of $C - S$ is not more than $|S|$. Hence, we have the theorem. □

Theorem 3.25 can be used to prove that neither of the two graphs shown in Figure 3.13 has a Hamiltonian cycle nor Hamiltonian path. This is left to Exercise 20.

Before discussing some sufficient conditions for graphs to have Hamiltonian cycles, we consider an important class of Hamiltonian graphs, the complete graphs. Complete graphs of two, three, four, and five vertices are shown in Figure 3.14.

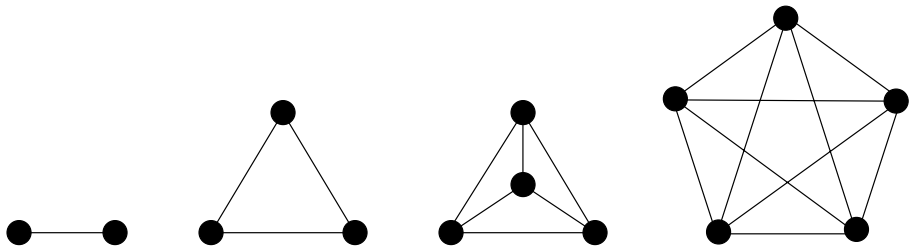


Figure 3.14: Complete graphs of two, three, four, and five vertices.

As in §2.6, a complete subgraph of a given graph is sometimes referred to as a *clique* in the graph. Since every vertex is adjacent to every other vertex, the degree of every vertex is $n-1$ in a complete graph G of n vertices. Also, the total number of edges in a clique having n vertices is $n(n-1)/2$ (see Exercise 12 from Chapter 2).

It is easy to construct a Hamiltonian cycle in a complete graph of n vertices. Let the vertices be numbered $v_1, \ldots, v_n$. Since an edge exists between any two vertices, we can start from v_1 and traverse to v_2, then to v_3, and so on to v_n, and finally from v_n to v_1. This is a Hamiltonian cycle.

3.7 More on Hamiltonian Paths and Cycles

The first issue we want to consider in this section is the number of Hamiltonian cycles in a graph. A graph may contain more than one Hamiltonian cycle. It is interesting to consider all of the edge-disjoint Hamiltonian cycles in a graph. For example, this gives another method for decomposing a graph. Determining the exact number of edge-disjoint Hamiltonian cycles (or paths) in a graph is a difficult problem. Short of an exhaustive search, which would be extremely time consuming, no good algorithms are known for solving this problem. But for complete graphs, the number of edge-disjoint Hamiltonian cycles is given by Theorem 3.26.

Theorem 3.26
For a number n greater than two, the complete graph K_n on n vertices has

$$\left\lfloor \frac{n-1}{2} \right\rfloor$$

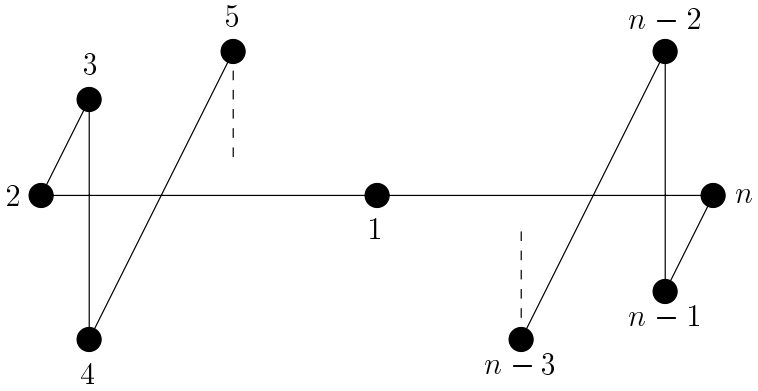


Figure 3.15: Hamiltonian cycle where the number of vertices n is odd.

edge-disjoint Hamiltonian cycles.

PROOF: For simplicity we only prove the theorem in the case when n is an odd integer greater than or equal to three. This case best illustrates the key idea in the proof. The proof for the even case follows similarly.

Assume n is odd. A complete graph G on n vertices has $n(n-1)/2$ edges, and a Hamiltonian cycle in G consists of n edges. Therefore, the number of edge-disjoint Hamiltonian cycles in G cannot exceed $(n-1)/2$. The following shows that there are in fact $(n-1)/2$ edge-disjoint Hamiltonian cycles when n is odd.

The subgraph of a complete graph on n vertices shown in Figure 3.15 is a Hamiltonian cycle. Keeping the vertices fixed on the circle that is centered at one, rotate the polygonal pattern of vertex numbers clockwise by $360/(n-1)$, $2 \cdot 360/(n-1)$, $3 \cdot 360/(n-1)$, ..., and $((n-3)/2) \cdot 360/(n-1)$ degrees.

Observe that each rotation produces a Hamiltonian cycle that has no edge in common with any of the previous ones. Thus we have $(n-3)/2$ new Hamiltonian cycles. Note, all of these are edge-disjoint from the one in Figure 3.15 and also edge-disjoint between themselves. Hence, we have the theorem. □

This theorem enables us to solve the problem of the seating arrangement at a round table that was introduced in Chapter 1 on page 8. Here is how we achieve the solution. To construct G, represent a member x by a vertex and the possibility of x sitting next to y by an edge between x and y. Since every member is allowed to sit next to any other member, G is a complete graph on

nine vertices—nine being the number of people to be seated around the table. Every seating arrangement around the table is clearly a Hamiltonian cycle.

On the first day of their meeting the members can sit in any order, and it will be a Hamiltonian cycle H_1. On the second day, if they are to sit such that every member must have different neighbors, we have to find another Hamiltonian cycle H_2 in G with an entirely different set of edges from those in H_1. That is, H_1 and H_2 need to be edge-disjoint Hamiltonian cycles. From Theorem 3.26 the number of edge-disjoint Hamiltonian cycles in G is four. Therefore, only four such arrangements exist among nine people. The actual seating arrangements can be determined from the proof of Theorem 3.26.

Next we discuss sufficient conditions to ensure that a graph is Hamiltonian.

Suppose G is a simple graph, and u and v are two vertices in G that are not connected by an edge. We denote by $G \cup \{u, v\}$ the simple graph that we get from G by adding an edge between u and v. That is,

$$G \cup \{u, v\} = (V(G), E(G) \cup \{\{u, v\}\}).$$

With this convention in mind, we have the following lemma.

Lemma 3.27 (Bondy, Chvátal 1976)
Let G be a simple graph on n vertices satisfying the following properties:

1. $u, v \in V(G)$,
2. $\{u, v\} \notin E(G)$, *and*
3. $d_G(u) + d_G(v) \geq n$.

G is Hamiltonian if and only if $G \cup \{u, v\}$ is Hamiltonian.

PROOF: Clearly, if G is Hamiltonian, then so is $G \cup \{u, v\}$.

For the other implication, assume that $G \cup \{u, v\}$ is Hamiltonian with a Hamiltonian cycle C. If C does not include the edge $\{u, v\}$, then C is a Hamiltonian cycle in G and we are done. So, assume further that C includes the edge $\{u, v\}$. Let us write C as $C = (v_0, v_1, \ldots, v_n)$, where $v_0 = v_n = u$ and $v_1 = v$. Define the sets U and V by

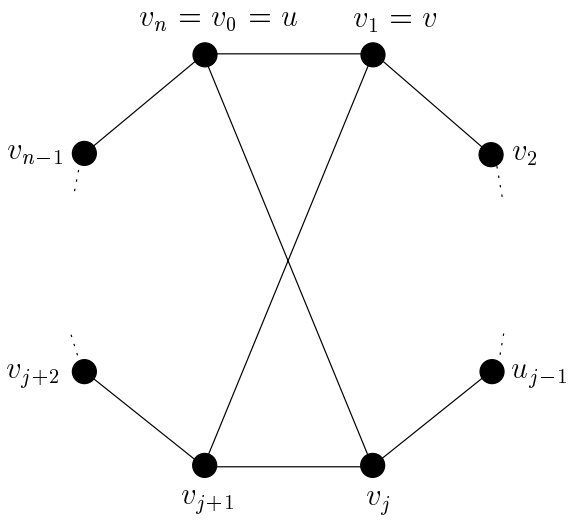


Figure 3.16: C' is a Hamiltonian cycle in G.

$$\begin{aligned} U &= \{i : \{u, v_i\} \in E(G), 2 \leq i \leq n-2\} \text{ and} \\ V &= \{i : \{v, v_{i+1}\} \in E(G), 2 \leq i \leq n-2\}. \end{aligned}$$

Since $\{u, v\} \notin E(G)$, we see that $|U| = d_G(u) - 1$ and $|V| = d_G(v) - 1$. Therefore, we have

$$n - 2 \leq |U| + |V| = |U \cup V| + |U \cap V| \leq (n-3) + |U \cap V|.$$

Hence, we have that $|U \cap V|$ is greater than or equal to one. Thus $U \cap V$ is not equal to the emptyset. (For what follows, it is useful to refer to Figure 3.16.) If $j \in U \cap V$, then we can form a Hamiltonian cycle

$$C' = (v_0, v_j, v_{j-1}, \ldots, v_1, v_{j+1}, v_{j+2}, \ldots, v_{n-1}, v_n)$$

that does not include the edge $\{u, v\}$. This is a Hamiltonian cycle of G. This proves the lemma. □

From Lemma 3.27, we can derive a well-known theorem by Ore [26] as an easy corollary.

Corollary 3.28
Let G be a simple graph on n greater than or equal to three vertices. If

$$d_G(u) + d_G(v) \geq n$$

for every pair of nonadjacent vertices u and v in G, then G is Hamiltonian.

PROOF: Let G_0 equal G. If the vertices u and v are not adjacent in G_0, then we let $G_1 = G_0 \cup \{u, v\}$. By Lemma 3.27, we have that G_0 is Hamiltonian if and only if G_1 is Hamiltonian.

If G_1 is a clique, then it is clearly Hamiltonian. Therefore, G_0 which equals G is also Hamiltonian and we are done. Otherwise, we continue to add edges in this fashion and form a sequence

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

We eventually reach the complete graph K_n. Here each graph G_i is a subgraph of G_{i+1} which has one more edge than G_i. So, G_i is Hamiltonian if and only if G_{i+1} is Hamiltonian. Since K_n is Hamiltonian, then so is G_0 which equals G. □

This simpler version is a direct consequence of Corollary 3.28 due to Dirac [10].

Corollary 3.29
If G is a simple graph on n greater than or equal to three vertices such that

$$d_G(u) \geq \frac{n}{2}$$

for every vertex $u \in V(G)$, then G is Hamiltonian.

➤ **Note 3.30**
Both Corollaries 3.28 and 3.29 provide a mechanical way to determine a sufficient condition for a graph to be Hamiltonian. More importantly though, the way we proved Lemma 3.27 gives us the following algorithm to find a Hamiltonian cycle in a graph G, if G satisfies the conditions of either Corollary 3.28 or Corollary 3.29.

ALGORITHM RESTRICTED HAMILTONIAN CYCLE constructs a Hamiltonian cycle in a simple graph G provided that G satisfies the conditions of either Corollary 3.28 or Corollary 3.29.

ALGORITHM RESTRICTED HAMILTONIAN CYCLE
Input: A simple graph $G = (V, E)$ satisfying the conditions of either Corollary 3.28 or Corollary 3.29.
Output: A Hamiltonian cycle of G.
begin
$G_0 = G$;
$k = ((n-1)n/2) - |E|$;
for $i = 1$ **to** k **do**
// $e_1, \ldots, e_k$ are new edges not in E
$G_i = G_{i-1} \cup G[\{e_i\}]$; // in particular, $G_k = K_n$
let C_k denote an arbitrary Hamiltonian cycle of G_k;
for $i = k$ **downto** 0 **do** {
if $E(C_i) \cap \{e_1, \ldots, e_i\} = \emptyset$
then output C_i;
else let $j \in \{1, \ldots, i\}$ be the largest index such
that e_j is contained in the Hamiltonian cycle C_i;
assume $C_i = (v_0, v_1, \ldots, v_n)$, where $e_j = \{v_0, v_1\}$;
find $l \in \{2, \ldots, n-2\}$ such that
$\{v_0, v_l\}, \{v_1, v_{l+1}\} \in E(G_{i-1})$;
form the Hamiltonian cycle C_{i-1} of G_{i-1} by letting
$C_{i-1} = (v_0, v_l, v_{l-1}, \ldots, v_1, v_{l+1}, v_{l+2}, \ldots, v_n)$;
}
end.

3.8 Traveling Salesman Problem

A problem closely related to the question of Hamiltonian cycles is the *traveling salesman problem*. Here is an intuitive description of the problem. A salesman is required to visit a number of cities during a trip. Suppose we have the distances between all of the cities. In what order should the salesman travel go so as to visit every city precisely once and return to the starting point while traveling the minimum mileage possible?

Representing the cities by vertices and the trips between them by edges, we obtain a saleman's graph. For every edge e_i in this graph, we associate a real number $w(e_i)$ (the distance in miles, say) denoting the cost of the trip between the two cities corresponding to the endvertices of e_i. This graph is called a

weighted graph; $w(e_i)$ being the *weight* of edge e_i.

In the traveling salesman problem, since each city has a trip distance to every other city, we have a *complete weighted graph*. This graph has numerous Hamiltonian cycles. We are to find one with the smallest sum of the weights.

The total number of different (not edge-disjoint) Hamiltonian cycles in a complete graph of n vertices is $(n-1)!/2$. This follows from the fact that starting from any vertex we have $n-1$ edges to choose from the first vertex, $n-2$ from the second, $n-3$ from the third, and so on. Since these choices are independent, there are $(n-1)!$ total. This number is divided by two because each Hamiltonian cycle has been counted twice.

Theoretically, the problem of the traveling salesman can always be solved by enumerating all $(n-1)!/2$ Hamiltonian cycles, calculating the distance traveled in each, and then picking the shortest one. However, even for a modest value of n, the number of computations involved is too great even for any group of computers to solve. Try solving it for the 50 state capitals in the United States, where n equals 50. To date, the largest instances of the traveling salesman problem that have been solved are on the order of 350 cities. This was only accomplished by employing thousands of computers.

Designing an efficient algorithm to find the shortest route is extremely difficult. Despite many attempts, no such algorithm has been found. Since this problem has applications in operations research, some specific large-scale examples have been worked out. Several heuristic methods are also available that approximate the shortest trip (to within a factor of two) but do not guarantee the shortest. In Chapter 15 we have more to say about the algorithmic complexity of the traveling salesman problem.

3.9 Exercises

1. Determine whether the graphs shown in Figure 3.17 are Eulerian. That is, can they be traversed from one vertex through all the edges exactly once ending up at the same vertex? Can you traverse through them if we do not insist upon ending at the same vertex as we started from?

2. Can you draw the graphs shown in Figure 3.18 without lifting your pen or pencil from the paper and without redrawing a line already drawn? Can

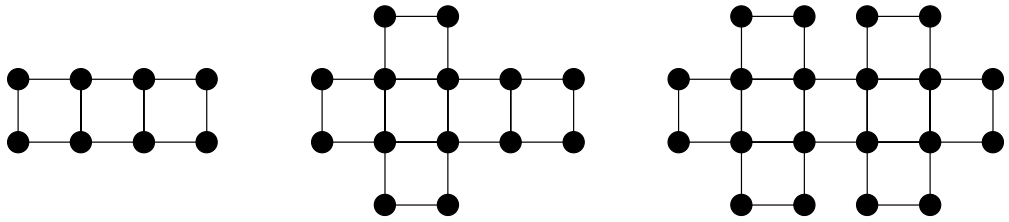


Figure 3.17: Eulerian graphs?

you do it starting and ending at the same point?

3. Prove that if a connected graph is decomposed into two subgraphs G_1 and G_2, then there must be at least one vertex in common between G_1 and G_2.

4. Implement the algorithm implied by the proof of Theorem 3.2 using your favorite programming language. You should be able to enter a graph, suitably coded, as input to your algorithm. Test your program on the graphs shown in Figure 3.17.

5. Draw a graph in which an Eulerian trail is also a Hamiltonian cycle. What can you say about such graphs in general?

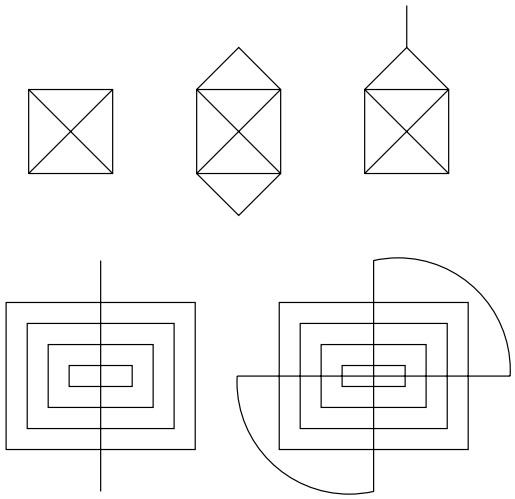


Figure 3.18: Draw these figures using just one line.

6. Is it possible to start from any of the 64 squares of a chess board and move a knight so that it visits every square once and returns to the initial position? If so, give one such tour. [*Hint:* Is the graph of Exercise 7, Chapter 1 on page 32 unicursal?]

7. Let a, b, and c be three distinct vertices in a graph. Assume that there is a path between a and b, and also a path between b and c. Show that there must be a path between a and c.

8. Show that $(G_1 \triangle G_2) \triangle G_3 = G_1 \triangle (G_2 \triangle G_3)$. [*Hint:* Start by showing that $(X_1 \triangle X_2) \triangle X_3 = X_1 \triangle (X_2 \triangle X_3)$ for arbitrary sets X_1, X_2, and X_3.]

9. In a graph G let p_1 and p_2 be two different paths between two given vertices. Prove that the symmetric difference of the paths $p_1 \triangle p_2$ is a circuit or a set of circuits in G.

10. If the intersection of two paths is a disconnected graph, show that the union of the paths has at least one circuit.

11. Draw $K_{2,4}$ in the usual manner and label the edges $1, \ldots, 8$ going from top-to-bottom. Draw the graphs resulting from the contraction of the edges one at a time in increasing numerical order. Repeat the process by deleting edges in the order 2, 4, 6, 8, 1, 3, 5, and 7. Are any of the resulting graphs the same? Explain your answer.

12. For what values of n are the following graphs bipartite: N_n, P_n, C_n, and K_n?

13. For what values of n are the following graphs 3-partite: N_n, P_n, C_n, and K_n?

14. Let n be a fixed natural number greater than two. A *round-robin tournament* (where every player plays against every other) among n players can be represented by K_n. Discuss how you would schedule the tournament to finish it in the shortest possible time.

15. Using your favorite programming language implement an algorithm that determines whether or not a graph is bipartite. You should be able to enter a graph, suitably coded, as input to your algorithm. Test your program on P_6, C_5, K_4, and $K_{3,3}$.

16. You are given a 10-piece domino set whose tiles have the following pairs of dots: $(1,2)$, $(1,3)$, $(1,4)$, $(1,5)$, $(2,3)$, $(2,4)$, $(2,5)$, $(3,4)$, $(3,5)$, and $(4,5)$. Discuss the possibility of arranging the tiles in a connected series such that one number on a tile always touches the same number on its neighbor. [*Hint:* Use K_5 and see if it is an Eulerian graph.]

17. Prove that Algorithm Eulerian Circuit on page 82 outputs an Eulerian circuit when provided with an Eulerian graph as input.

18. Implement Algorithm Eulerian Circuit using your favorite programming language. You should be able to enter a graph, suitably coded, as input to your algorithm. Test your program on the graphs C_5, K_5, and $K_{2,4}$.

19. Draw a graph containing at least five vertices. Construct the graph so it has a Hamiltonian path but not a Hamiltonian cycle.

20. Show that neither of the graphs illustrated in Figure 3.13 is Hamiltonian. [*Hint:* Use Theorem 3.25.]

21. The graph of a *rhombic dodecahedron* (with eight vertices of degree three and six vertices of degree four) is the simple graph G with vertices and edges defined as follows:

$$\begin{aligned} V(G) &= \{u_0,\ldots,u_6\} \cup \{v_0,\ldots,v_6\}, \\ E(G) &= \{\{u_0,u_1\},\{u_0,u_3\},\{u_0,u_5\}\} \\ &\cup \{\{u_1,u_2\},\ldots,\{u_5,u_6\},\{u_6,u_1\} \\ &\cup \{\{v_0,v_2\},\{v_0,v_4\},\{v_0,v_6\}\} \\ &\cup \{\{v_1,v_2\},\ldots,\{v_5,v_6\},\{v_6,v_1\} \\ &\cup \{\{u_1,v_1\},\{u_2,v_2\},\ldots,\{u_6,v_6\}\}. \end{aligned}$$

    Show that G is not Hamiltonian. That is, show it has no Hamiltonian cycle. Does G have a Hamiltonian path? [*Hint:* For the second question, note that G is a bipartite graph. If G has a Hamiltonian path, add an appropriate edge to get a Hamiltonian cycle.]

22. Using the result of Corollary 3.29 show that in a ring of n dancing children, it is always possible to arrange the children so that everyone has a friend at each side if every child enjoys friendship with at least half of the children.

23. Let m and n be positive integers. Let $[m] = \{1, \ldots, m\}$ and $[n] = \{1, \ldots, n\}$. Consider the *grid graph* $[m] \times [n]$ with vertex set $\{(i, j) : i \in [m] \text{ and } j \in [n]\}$ and where we connect every two ordered tuples (i, j) and (i', j') if and only if $|i - i'| + |j - j'| = 1$. Find necessary and sufficient conditions that m and n must satisfy in order for the graph $[m] \times [n]$ to be Hamiltonian.

24. Adopting the terminology from Exercise 23, we can make the following generalization: If k is a positive integer and $n_1, \ldots, n_k$ are also positive integers, the *k-box graph* $[n_1] \times \cdots \times [n_k]$ is the graph with vertex set $\{(i_1, \ldots, i_k) : i_l \in [n_l] \text{ for each } l \in [k]\}$ and where we connect every two ordered k-tuples $(i_1, \ldots, i_k)$ and $(i'_1, \ldots, i'_k)$ if and only if $|i_1 - i'_1| + \cdots + |i_k - i'_k| = 1$. For each fixed k, find necessary and sufficient conditions that the numbers $n_1, \ldots, n_k$ must satisfy in order for the corresponding k-box graph to be Hamiltonian.

25. Prove that a graph G with n vertices always has a Hamiltonian path if every two vertices u and v satisfy

$$d_G(u) + d_G(v) \geq n - 1.$$

[*Hint:* First show that G is connected. Then use induction on the path length of G.]

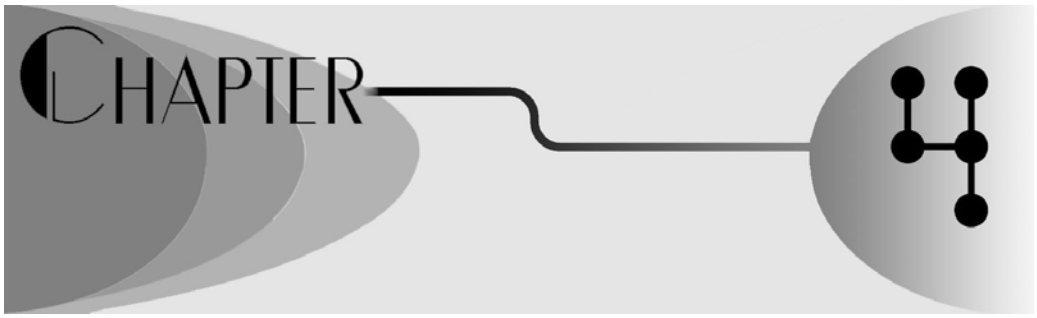
26. Consider an instance of the traveling salesman problem that uses driving distances between cities as the transportation costs between them. What would be a solution for the traveling salesman problem involving the following cities: Boston, Chicago, Los Angeles, Orlando, Providence, Savannah, and Seattle? Use AAA mileage estimates and report the values you used.

27. Consider an instance of the traveling salesman problem that uses Delta Airlines' flying cost between cities as the transportation costs between them. What would be a solution for the traveling salesman problem involving the following cities: Boston, Chicago, Los Angeles, Orlando, Providence, Savannah, and Seattle? Report the dollar values you used for the routes between cities. (If a direct flight between the cities is not available, use the cost of the cheapest alternative route.)

28. A newspaper reporter from the New York Times has contacted you to write a one-page article for the Times that makes the ideas behind the traveling

salesman problem understandable to the masses. Develop such an article being sure to include lots of intuition about the problem.

29. Evaluate the formula $(n-1)!/2$ for the following values: 5, 10, 50, 100, 200, 300, 500, and 1000. Assuming it takes a computer .001 seconds to evaluate a single Hamiltonian cycle in an instance of the traveling salesman problem, how long would it take to run instances of the problem of these sizes? For which values of n would you say this is feasible?



Trees and Forests

Chapter Goals

- Present the concepts of trees and forests.
- Demonstrate why trees are one of the most important types of graphs.
- Discuss the fundamental properties of trees and forests.
- Describe some characterizations of trees.
- Present inductive proofs on trees.
- Examine some important applications of trees.
- Prove the Erdös-Szekeres Theorem.
- Define the notions of distance and centers for graphs in general and specifically for trees.
- Present the concept of rooted trees.
- Define binary and regular binary trees.
- Study path lengths of trees with respect to algorithmic efficiency.

4.1 Introduction

The concepts of trees and forests are probably two of the most important in graph theory—especially for those interested in the applications of graphs. In this

chapter we define the notions of a tree and a forest, and study their properties. We point out some of their applications to searching and sorting, and constructing efficient codes and structures. This chapter provides the proofs of a number of interesting results involving trees including some new proof techniques. We bring some other graph-theoretic terms related to trees into our discussion. Trees are a visual subject. Throughout the chapter, note the use of a wide range of figures for trees.

In §4.2 we introduce some of the basic properties of trees. A number of important characterizations of trees are given in §4.3. In §4.4 we illustrate how to use inductive proofs on trees. The classic Erdös-Szekeres Theorem on subsequences is proved in §4.5. The concepts of *distance* and *centers* with respect to trees are covered in §4.6. In §4.7 we examine how trees can be oriented by introducing *rooted trees*. *Binary trees* are presented in §4.8. This important class of trees is used in many algorithms. *Path length* is important in the analysis of algorithms and is presented in §4.9.

4.2 Trees and Some of Their Basic Properties

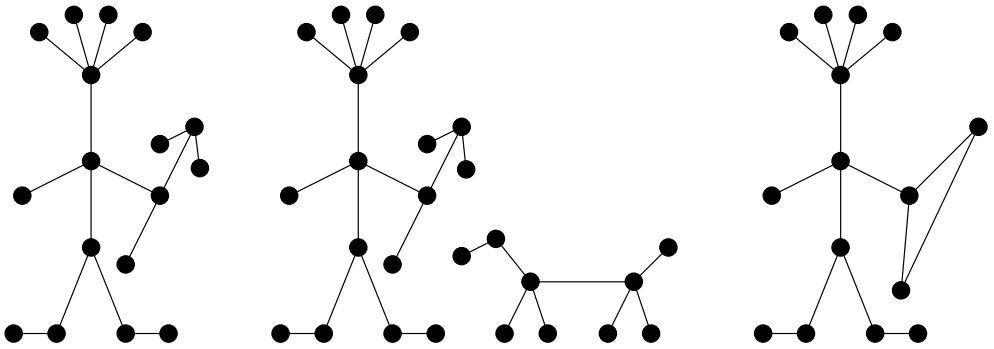
We begin with the definitions of a tree and a forest.

Definition 4.1
A tree *is a connected graph that has no cycle as a subgraph. A* forest *is a simple graph in which every component is a tree.*

Figure 4.1 shows three graphs: (a) is a tree on 17 vertices; (b) is a forest on 26 vertices; (c) is a simple graph on 15 vertices. The last one is neither a tree nor a forest since the bow is a cycle of length three.

By Definition 1.9 on page 13, any graph, including a tree, must have at least one vertex. In this book all trees and forests have finitely many vertices and edges. A tree has neither loops nor multiple edges since every loop is a cycle of length one, and a double edge between a pair of vertices forms a cycle of length two.

For a graph to be a tree is a strong condition. Consider the set of all possible graphs on n vertices. Relatively few of those will be trees. In Figure 4.2 we show all possible trees on six vertices. On the far left of the figure is a path (also commonly referred to as a *chain*) and on the far right of the figure is a *star*. In



(a) (b) (c)

Figure 4.1: (a) A tree. (b) A forest. (c) A graph that is neither a tree nor a forest.

Chapter 5 we examine several counting problems relating to trees. In Chapter 14 we discuss in greater detail how many trees there are on n vertices.

Trees occur in many natural applications including those listed below. The reader is asked to jot down some other applications of trees in the exercises.

The following can be represented by a tree:

- ▷ The genealogy of a family. (In fact, the term *tree* comes from *family tree*.)
- ▷ A river with its tributaries and sub-tributaries.
- ▷ The hierarchy in a business.
- ▷ The hyperlinks between a set of Web pages.
- ▷ The sorting of mail according to zip code and the sorting of integers. (This is called a *decision tree* or *sorting tree*.)

In the following example, we describe the mail sorting application in more detail.

☞ **Example 4.2**
The tree shown in Figure 4.3 can be used to represent the flow of snail mail. A batch of physical mail arrives at some local post office. This post office is represented by vertex N in the figure. The leftmost digit in the zip code is read

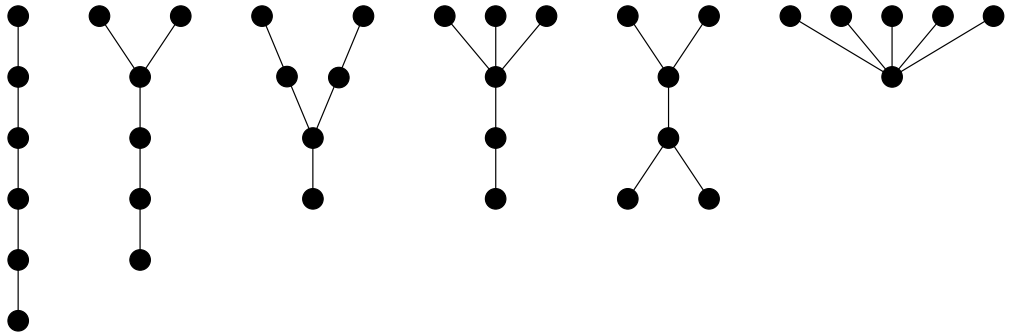


Figure 4.2: All possible trees on six vertices.

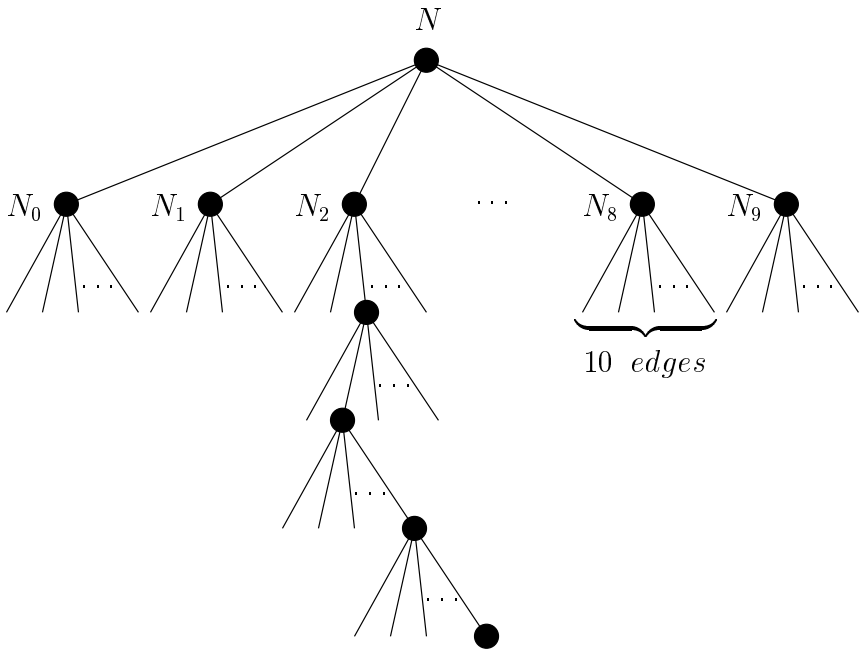


Figure 4.3: A decision tree for mail sorting.

at N. The mail is then divided into ten piles $N_0, N_1, \ldots, N_9$ depending on the value of this digit. These piles are further subdivided into ten additional piles in a similar manner and so on until the mail is subdivided into 10^5 possible piles. The vertices at the bottom of the figure, of which only one is shown, represent an unique five-digit zip code. The vertex shown represents zip code 23199. Once the mail is sorted, it is ready for distribution.

Many sorting problems are simpler than the mail sorting problem described in Example 4.2. In fact, there are often only two alternatives at each intermediate vertex instead of ten as in the mail sorting example. These two choices usually represent a dichotomy such as large or small, good or bad, or 0 or 1. Such a decision tree with two choices at each vertex occurs frequently in computer science and mathematics. We shall study such trees and their applications in §4.8. But first let us obtain a few simple but important theorems on the general properties of trees, starting with an important definition.

Definition 4.3
A vertex u of a simple graph G is called a leaf *if $d_G(u)$ equals one. That is, u has degree one. A vertex, which is not a leaf, is called an* internal vertex.

Many fundamental properties about trees can be derived from the following theorem.

Theorem 4.4
Every tree with at least two vertices has at least two leaves.

PROOF: Consider a tree T having two or more vertices. Let p be the longest path in T. Let u and v be the initial and endvertex of p, respectively. We show that u and v are leaves. That is, $d(u)$ and $d(v)$ both equal one. It suffices to consider just one of the vertices, say u, since the argument would be similar for the other vertex v.

Suppose $d(u)$ is greater than or equal to two. Then there is an edge e, not in the path p, having endvertices u and w in T, where $w \in V(T)$. We now have two cases to consider.

CASE ONE: If the vertex w is not a vertex in the path p, then the composite path $p' = (w, e, u)p$ is a path in G of length one more than that of p. This contradicts the fact that p was a path of longest length in T.

CASE TWO: Suppose the vertex w is a vertex in the path p. If p'' is the path from u to w along the path p, we get a cycle $c = (w, e, u)p''$ of length three or more in T. This contradicts the fact that T is a tree.

Hence, $d(u)$ equals one thereby completing the argument. □

➤ **Note 4.5**
Theorem 4.4 provides a useful tool in proving many properties for trees. In many cases, the fact that every tree on two or more vertices has at least one leaf is all

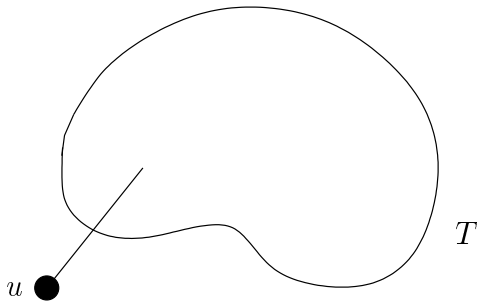


Figure 4.4: Every tree has at least one leaf.

we need to demonstrate a certain property of a tree. Suppose T is a tree having at least one leaf u as shown in Figure 4.4. Notice that the subgraph $T' = T - u$ has no cycles since T has none. Also, T' is connected. Hence, T' is a tree and it has one fewer vertices than T. So many proofs about T can be done inductively by assuming the argument is true for T'.

Here is another useful fact about trees in Lemma 4.6.

Lemma 4.6
A simple graph G on n vertices with k components has $n - k$ or more edges.

PROOF: We use induction on the number of edges of G, m. The statement is clearly true for m equals zero since G is then the nullgraph on n vertices.

Suppose m is greater than one and assume that the assertion is true for all natural numbers less than m. If e is an edge in G, then the simple graph $G - e$ has n vertices, $m - 1$ edges, and $k + \delta$ components, where δ is either zero or one. By the induction hypothesis, we have that $n - (k + \delta) \leq m - 1$. Hence, $n - k \leq m - (1 - \delta) \leq m$ completing the inductive argument. This proves the lemma. □

It is instructive to compare Lemma 4.6 with Theorem 2.14 on page 46. The theorem provides the upper bound on the number of edges in a simple graph on n vertices and k components. Lemma 4.6 provides the lower bound showing that trees are minimally connected in the sense that removing any edge results in two or more components. We will see this even more clearly in Theorem 4.7.

4.3 Characterizations of Trees

Theorem 4.7 contains some of the most frequently used characterizations of trees and we will prove some of these statements using the technique described in Note 4.5.

Theorem 4.7
If T is a simple graph on n vertices, then the following statements are equivalent:

1. *T is a tree.*
2. *T has $n-1$ edges and no cycles of length one or more.*
3. *T has $n-1$ edges and is connected.*
4. *T is connected and each edge is a cut-edge.*
5. *Between every pair of distinct vertices in T there is one and only one path.*
6. *T has no cycles, but if we add one edge to T, then exactly one simple cycle of length one or more is formed.*

PROOF: We prove the theorem by showing the following "cycle" of implications:

$$1 \Rightarrow 2 \Rightarrow 3 \Rightarrow 4 \Rightarrow 5 \Rightarrow 6 \Rightarrow 1.$$

▷ $1 \Rightarrow 2$:

We arge by induction on the number of vertices n. We need to show that T has $n-1$ edges. The statement is clearly true if n equals one. Let T be a tree on n greater than or equal to two vertices. By Theorem 4.4, T has a leaf u. Hence, $T' = T - u$ is a tree on $n-1$ vertices. Therefore, by the induction hypothesis T' has $n-2$ edges. Since u is a leaf, T' has one fewer edges than T. So, T has $n-1$ edges.

▷ $2 \Rightarrow 3$:

We need to show that T is connected. By definition we have that T is a forest. Let k greater than or equal to one be the number of components in T. Every component is a tree and therefore has one fewer edges than vertices as we just saw in proving that condition 1 implies condition 2. Hence, the number of edges in T is $n-k$. Thus $n-k$ equals $n-1$ and we have that k equals one. So, T is connected.

▷ $3 \Rightarrow 4$:

We must show that each edge is a cut-edge. By Lemma 4.6 every connected graph must have at least $n-1$ edges. Therefore, the removal of any edge from T will disconnect it. So, each edge is a cut-edge.

▷ $4 \Rightarrow 5$:

We need to show that there is an unique path between each pair of vertices in T. Since T is connected there is a simple path between every pair of vertices in T. Assume that there are two vertices u and v such that there are two distinct simple paths γ_1 and γ_2 from u to v. We can assume that γ_1 is the simple path of shortest length. Because γ_1 is of shortest length, the edges of γ_2 cannot be a rearrangement of those in γ_1 since otherwise one could further shorten γ_1. Hence, there must be an edge e in γ_2 that is not in γ_1. Now e cannot be a cut-edge since there is a walk in $G-e$ connecting the endvertices of e. This walk is formed by edges from γ_1 and the rest of γ_2. This is a contradiction.

▷ $5 \Rightarrow 6$:

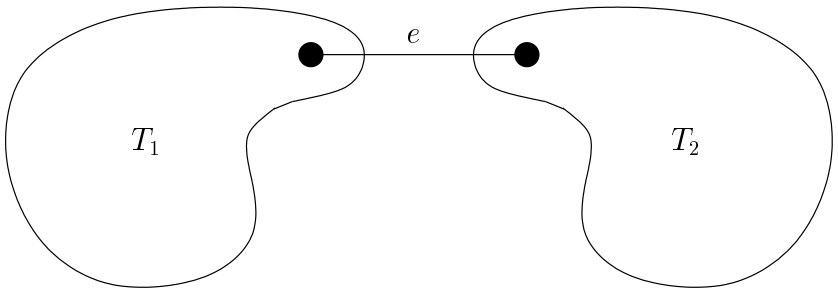
We must show that adding any edge to T adds exactly one simple cycle of length one or more. If T has a simple cycle of length one or more, then for every two vertices on the cycle there are at least two simple paths between them. Now, let u and v be two vertices and γ a simple path from u to v in T. If we add an edge e between u and v, then $c = \gamma(v, e, u)$ is a simple cycle of length one or more in T. If there is another such cycle c', then c' would have to contain e and by removing e from c' one would get yet another simple path from u to v.

▷ $6 \Rightarrow 1$:

We must show that T is indeed connected. Let u and v be vertices in T. By adding an edge e between u and v, we obtain a single cycle of length one or more. This cycle must contain the edge e. The removal of e yields a simple path from u to v. Hence, T is connected and therefore a tree.

□

Depending on the situation, one of the characterizations given in Theorem 4.7 may be preferable to the others.



T

Figure 4.5: The removal of any edge of a tree T yields two trees T_1 and T_2.

4.4 Inductive Proofs on Trees

We say that a graph is *minimally connected* if the removal of any edge disconnects the graph. In view of this, trees on a given set of vertices are precisely the graphs that minimally connect the vertices as we saw in characterization number 4 of Theorem 4.7. Inductive proofs on trees arise in a natural way because every edge is a cut-edge. In an n-vertex tree T is with edge e, the removal of e gives us exactly two trees T_1 and T_2 with $|V(T_1)| + |V(T_2)| = |V(T)|$ as illustrated in Figure 4.5. Regardless of which edge e is chosen, note that both $|V(T_1)|$ and $V|(T_2)|$ are strictly less than $|V(T)|$. Thus any induction hypothesis will apply. We illustrate this inductive proof technique in the following example.

☞ **Example 4.8**
In this example we prove an upper bound on the number of vertices of degree three in a given class of trees.

Suppose n is a natural number greater than one. Consider all trees on n or fewer vertices, where all of the vertices have degree three or less. A very important class of trees called regular binary trees *are among these. We discuss* binary *and regular binary trees further in §4.8. Let $\mathcal{B}_n$ denote the maximum number of vertices of degree three that such a tree can have. We have the following interesting fact:*

$$\mathcal{B}_n \leq \left\lfloor \frac{n}{2} \right\rfloor - 1. \tag{4.1}$$

To prove this, we can use induction on n to show that $\mathcal{B}_n$ is less than or equal to $n/2 - 1$. Then Inequality 4.1 follows directly. The statement $\mathcal{B}_n$ is less than or equal to $n/2 - 1$ is clearly true for n equals two or three, where there is no vertex of degree three.

Assume now that n is greater than three and that the statement $\mathcal{B}_n$ is less than or equal to $n/2 - 1$ is true for all trees on fewer than n vertices. Let T be a tree on n vertices and let $S(T)$ be the set of vertices of T having degree three. We can assume that T has the maximum number of vertices of degree three. That is, $|S(T)|$ equals $\mathcal{B}_n$. We pick $u \in S(T)$ and let its neighbors be u_1, u_2, and u_3. Since every edge is a cut-edge, if the edges $\{u, u_1\}$, $\{u, u_2\}$, and $\{u, u_3\}$ are removed from T, we form three new trees. For $i \in \{1, 2, 3\}$ we add the edge $\{u, u_i\}$ to the tree containing u_i. The tree containing u_i is called T_i and n_i represents the number of vertices in T_i. The subtrees T_1, T_2, and T_3 are blocks *of T. (Blocks are discussed in additional detail in §6.5.) Figure 4.6 illustrates the situation. We note that*

$$n_1 + n_2 + n_3 = n + 2, \tag{4.2}$$

since each of the trees T_1, T_2, and T_3 contains a copy of the vertex u. Since each T_i contains at least two vertices, u_i and the copy of u, we note more importantly that by Equation 4.2, n_i is less than or equal to $n - 2$ for each i.

By the induction hypothesis, each of the trees T_i has $n_i/2 - 1$ or fewer vertices of degree three. Since the set of vertices of degree three in T is a disjoint union of the set of vertices of degree three in each of the trees T_i, together with the vertex u itself, we have

$$\begin{aligned}
\mathcal{B}_n &= |S(T)| \\
&= |S(T_1)| + |S(T_2)| + |S(T_3)| + 1 \\
&\leq \left(\frac{n_1}{2} - 1\right) + \left(\frac{n_2}{2} - 1\right) + \left(\frac{n_3}{2} - 1\right) + 1 \\
&\leq \frac{n}{2} - 1.
\end{aligned}$$

Hence, we have by induction that $\mathcal{B}_n$ is less than or equal to $n/2 - 1$ for any natural number n greater than one. Since $\mathcal{B}_n$ is a natural number, we have

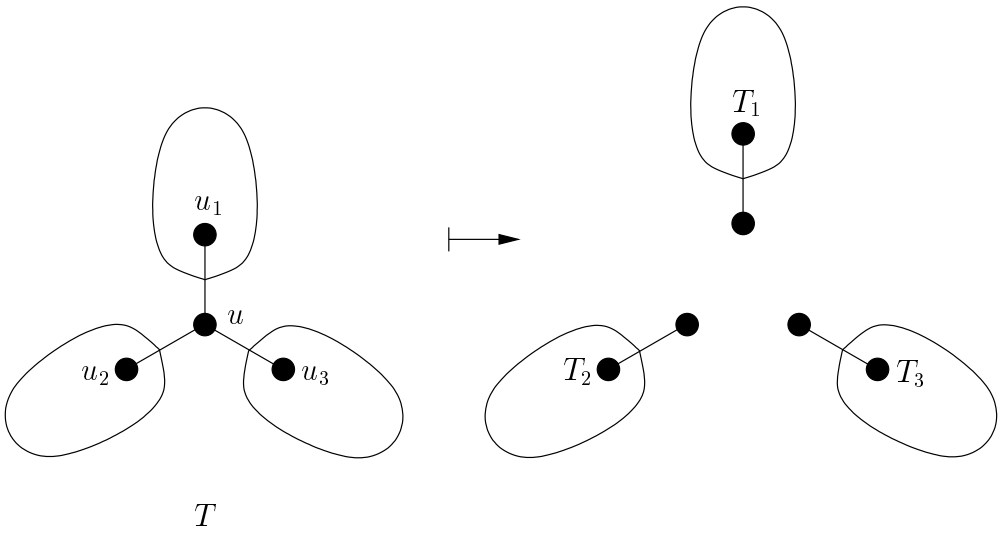


Figure 4.6: From the tree T we obtain three smaller trees T_1, T_2, and T_3.

that $\mathcal{B}_n$ is less than or equal to $\lfloor n/2 - 1 \rfloor$. Because we have in general that $\lfloor x + m \rfloor = \lfloor x \rfloor + m$ for any real number x and any integer m (Can you see why?), it follows that Inequality 4.1 holds for all natural numbers greater than one. It is instructive to compare this result with Observation 4.33 for a regular binary tree on n vertices.

Interestingly, the upper bound of $\mathcal{B}_n$ in Example 4.8 can be reached for an infinite number of natural numbers n. We can form a sequence of trees $T_1, T_2, \ldots, T_k, \ldots$ by initially letting T_1 be a single vertex graph. Next we let T_2 be the graph obtained from T_1 by connecting three leaves to the vertex of T_1. In general, for k greater than or equal to two, we obtain T_{k+1} from T_k by connecting two new leaves to each leaf of T_k as show in Figure 4.7.

By viewing each T_{k-1} as a subgraph of T_k, we notice that every vertex of T_{k-1} has degree three in T_k. If we let t_k be the number of vertices of the tree T_k, we see that T_k has t_{k-1} vertices of degree three plus $t_k - t_{k-1}$ leaves for a total of t_k vertices. By construction, we add two leaves to each leaf of T_k to obtain T_{k+1}. That is, $t_{k+1} = t_k + 2(t_k - t_{k-1})$. Hence, the sequence $(t_k)_{t \geq 1}$ satisfies

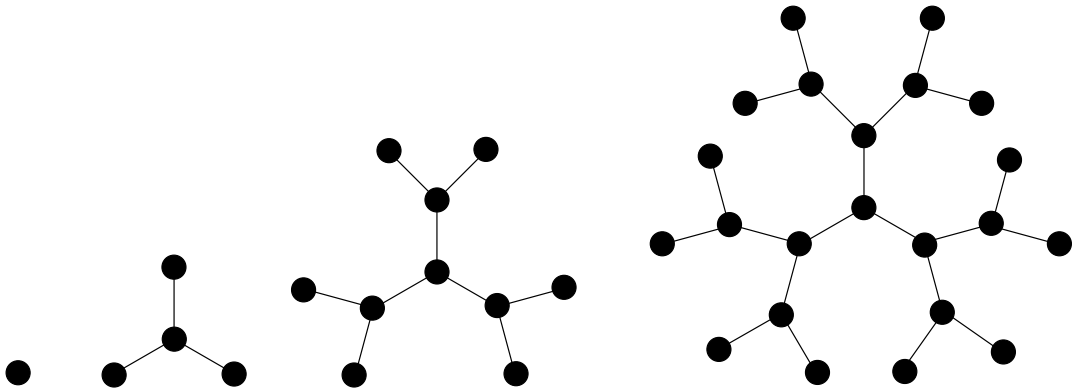


Figure 4.7: The first four trees in the sequence $T_1, T_2, \ldots, T_k, \ldots$

the following *recurrence relation*:

$$\begin{aligned} t_1 &= 1, \\ t_2 &= 4, \\ t_{k+1} &= 3t_k - 2t_{k-1} \text{ for } k \geq 2. \end{aligned}$$

One can now prove by induction that

$$t_k = 3 \cdot 2^{k-1} - 2 \text{ for all } k \geq 1.$$

(In general, to see how one can find explicit formulas for sequences given recursively, we refer the reader to Chapter 5 of [27].) So, we have for each natural number k that

$$|S(T_k)| = t_{k-1} = \left\lfloor \frac{t_k}{2} \right\rfloor - 1.$$

This shows that the upper bound of Inequality 4.1 can be reached for infinitely many n, namely n equals t_k for k a natural number.

In Example 4.8 we do not have complete freedom to choose which edge we can remove to use our induction hypothesis to prove the lower bound given in Inequality 4.1. Since we are not guaranteed the existence of a leaf with a neighbor having degree three, we cannot remove a leaf and directly use the induction hypothesis on the resulting tree.

For a natural number Δ greater than or equal to two, one could consider all trees on n or fewer vertices where each vertex has degree Δ or less. That is, where the maximum degree is Δ. The same argument applies in this more general case as in the case Δ equals three in Example 4.8. (See Exercises 6 and 7.)

4.5 Erdös-Szekeres Theorem on Sequences

We begin this section with an application of *data trees*. The problem we look at is a good exercise in computer programming and is one that occurs often in practice. This leads us into new territory about posets and then to the proof of the classical Erdös-Szekeres Theorem.

Problem 4.9

Increasing Subsequence

A monotonically increasing subsequence *is one in which succeeding numbers are always larger than preceding ones. How can we find the longest monotonically increasing subsequence in a given sequence of integers where no two integers are the same? Suppose the given sequence is* $(4, 1, 13, 7, 0, 2, 8, 11, 3)$. *This sequence can be represented by a tree in which the vertices (except the start vertex) represent individual numbers in the sequence, and the path from the start vertex to a particular vertex* v *describes the monotonically increasing sequence terminating at* v. *A tree that represents information is referred to as a* data tree *by computer scientists.*

As shown in Figure 4.8, our sample sequence contains four longest monotonically increasing subsequences of length four. They are $(4, 7, 8, 11)$, $(1, 7, 8, 11)$, $(1, 2, 8, 11)$, *and* $(0, 2, 8, 11)$. *In our sequence all nodes were distinct, however, this may not always be the case. We can still apply the technique described here to an input where the values are not distinct. Duplicate values will never occur on the same path based on the way our data tree is constructed. Given any sequence it should be clear how to construct its data tree and how to determine the sequence's longest monotonically increasing subsequence.*

As a little digression from our discussion of trees, we will discuss a beautiful result that is directly related to Problem 4.9.

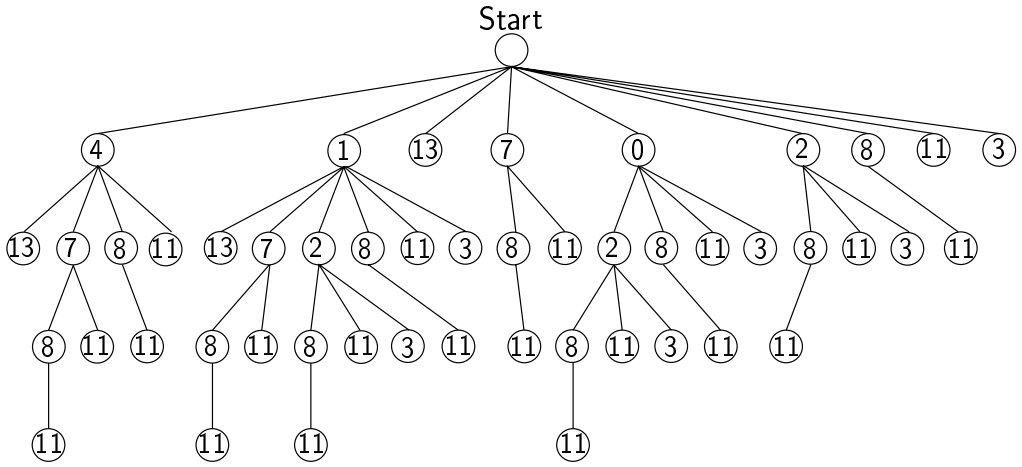


Figure 4.8: The data tree for determining the longest monotonically increasing sequence in $(4, 1, 13, 7, 0, 2, 8, 11, 3)$.

Note that in Problem 4.9 the original sequence had length ten and each of the subsequences had length four. In fact, *every* sequence of distinct numbers of length ten can be shown to have a subsequence of length four that is either strictly increasing or strictly decreasing. Here we use the word "strictly" to denote that the numbers in the subsequence are distinct. The following classic theorem, due to Erdös and Szekeres, generalizes this fact.

Theorem 4.10 (Erdös-Szekeres Theorem)
Every sequence of $n^2 + 1$ distinct integers contains a subsequence of length $n + 1$ that is either strictly increasing or strictly decreasing.

We present two proofs of this theorem. In our first proof we rely on finite posets which were presented in Definition 1.32. The second proof relies on the *Generalized Pigeon Hole Principle*. We begin by defining some new concepts and proving some easy lemmas.

Definition 4.11
Let $(S, \preceq)$ be a finite poset.

1. *Two elements s and s' are said to be* comparable *if either $s \preceq s'$ or $s' \preceq s$ holds. Otherwise, they are called* incomparable.

2. *An element $s \in S$ is called a* minimal *(*maximal*) element if there is no element s' with $s' \preceq s$ (respectively, $s \preceq s'$).*

3. *The set of all minimal (maximal) elements of S is denoted by $\min(S)$ (respectively, $\max(S)$).*

4. *A collection $s_1 \preceq \cdots \preceq s_p$ of distinct elements of S is called a* chain *of length p.*

5. *A collection $s_1, \ldots, s_q$ of distinct elements of S is called an* antichain *of size q if the elements are pairwise incomparable. That is, $s_i \preceq s_j$ does not hold for any natural numbers $1 \leq i \neq j \leq q$.*

As an easy observation, we have the following result.

Observation 4.12
Let $(S, \preceq)$ be a finite poset. The following properties hold:

1. *The subsets $\min(S)$ and $\max(S)$ both form antichains in S.*

2. *For every $s \in S$, there is an $s' \in \min(S)$ and $s'' \in \max(S)$ such that $s' \preceq s \preceq s''$.*

PROOF:

PROPERTY 1: If there were elements s and s' in $\min(S)$ such that $s \preceq s'$, then s' would not be minimal. This contradicts that $s' \in \min(S)$. Similarly, $\max(S)$ has no comparable elements. So, both $\min(S)$ and $\max(S)$ form antichains in S.

PROPERTY 2: We only show the existence of s' since the existence of s'' is achieved in a similar way (see Exercise 13).

If $s \in \min(S)$ then $s \preceq s$ and we are done. Otherwise, let s equal s_0 and find an element s_1 distinct from s_0 with $s_1 \preceq s_0$. If $s_1 \in \min(S)$ then we are done. Otherwise, there is an element s_2 distinct from both s_1 and s_0 with $s_2 \preceq s_1 \preceq s_0$. One can continue in this fashion. However, since S is finite, this process has to stop for some k for the chain $s_k \preceq \cdots \preceq s_1 \preceq s_0$, where there is no element $s'' \in S$ distinct from each s_i with $s'' \preceq s_k$. This means that $s_k \in \min(S)$. □

The following "folklore lemma" is key to proving Theorem 4.14.

Lemma 4.13 (Folklore Poset Lemma)
A poset $(S, \preceq)$ in which the longest chain has length p and the largest antichain has size q has at most $|S| \leq pq$ elements.

PROOF: We prove this by induction on p—the length of the longest chain.

If p equals one, then all of the elements in S are pairwise incomparable since $s \preceq s'$ is a chain of length two. Therefore, q equals $|S|$ proving the lemma in this case.

Let p be greater than or equal to two and assume that the lemma is true for any poset with longest chain of length $p-1$ or less. Let $(S, \preceq)$ be a poset where the longest chain has length p and the largest antichain has size q. Suppose $s_1 \preceq \cdots \preceq s_p$ is a chain in $S' = S \setminus \min(S)$. Then by Observation 4.12, there is an element $s' \in \min(S)$ with $s' \preceq s_1$ yielding a chain in S of length $p+1$ in S. This contradicts our assumption about $(S, \preceq)$. Hence, the length of the longest chain in S' has length $p-1$ or less. Since $S' \subseteq S$, the largest antichain in S' also has size at most q. So, by the induction hypothesis we have that $|S'| \leq (p-1)q$. By Observation 4.12, $\min(S)$ forms an antichain in S and hence by our assumption, we have that $|\min(S)|$ is less than or equal to q. Therefore,

$$|S| = |S'| + |\min(S)| \leq q + (p-1)q = pq.$$

This completes the proof of the lemma. □

We can now prove Theorem 4.14.

PROOF: Let $m = n^2 + 1$ and let $\tilde{a}_m = (a_1, \ldots, a_m)$ be a sequence of distinct integers. We form the set $A_m = \{(i, a_i) : i \in \{1, \ldots, m\}\}$.

Any set S consisting of ordered pairs (s, t) of integers can be made into a poset $(S, \preceq)$ by letting

$$(s, t) \preceq (s', t') \text{ if and only if } s \leq s' \text{ and } t \leq t'.$$

Hence, we can view A_m as the poset $(A_m, \preceq)$.

Note that $((i_1, a_{i_1}), (i_2, a_{i_2}), \ldots, (i_p, a_{i_p}))$ forms a chain in A_m. This means that $a_{i_1} < a_{i_2} < \cdots < a_{i_p}$ is a strictly increasing subsequence in $\tilde{a}_m$. Likewise, $((i_1, a_{i_1}), (i_2, a_{i_2}), \ldots, (i_q, a_{i_q}))$ forms an antichain in A_m which implies that $a_{i_1} > a_{i_2} > \cdots > a_{i_q}$ is a strictly decreasing subsequence of A_m.

Assume that $\tilde{a}_m$ has neither a strictly increasing nor a strictly decreasing subsequence of length $n+1$. In this case the length of the longest chain in A_m

has length p less than or equal to n and the largest antichain has size q less than or equal to n. Hence, by Lemma 4.13 we have that

$$n^2 + 1 = m = |A_m| \leq pq \leq n^2.$$

This is a contradiction. So, $\tilde{a}_m$ must have a strictly increasing subsequence of length $n + 1$ or a strictly decreasing subsequence of length $n + 1$. □

We now state a slight variant of the Erdös-Szekeres Theorem and give Seidenberg's proof of the result [28].

Theorem 4.14 (Erdös-Szekeres Theorem)
If $x_1, x_2, \ldots, x_{mn+1}$ are distinct real numbers then there is either an increasing subsequence of $n + 1$ terms or a decreasing subsequence of $m + 1$ terms.

PROOF: Suppose there is no increasing subsequence of length $n+1$. For each $i \in \{1, 2, \ldots, mn+1\}$, let l_i be the length of the longest increasing subsequence that begins at x_i. Clearly, each l_i falls between 1 and n inclusive. Therefore, by the Generalized Pigeon Hole Principle there exist $x_{i_1}, x_{i_2}, \ldots, x_{i_{m+1}}$ with

$$i_1 < i_2 < \cdots < i_{m+1}$$

such that $l_{i_1} = l_{i_2} = \cdots = l_{i_{m+1}} = l$. By the definition of l, the sequence $x_{i_1}, x_{i_2}, \ldots, x_{i_{m+1}}$ of $m + 1$ terms must be decreasing. □

For similar results on finite posets and related advanced material, we refer the reader to [30].

4.6 Distance and Centers in a Tree

In this section we examine a number of interesting structural properties of trees. The tree shown in Figure 4.9 has four vertices. Intuitively, it seems that vertex b is located more centrally than the other three vertices. We shall explore this idea further and see if there exists a *center* (or *centers*) in a tree. Such a center could play an important role in developing recursive algorithms for problems on trees. For example, repeatedly removing centers would lead to smaller and smaller subtrees. Additionally, such a center could serve an important role in networking applications, where nodes might communicate with each other through the center.

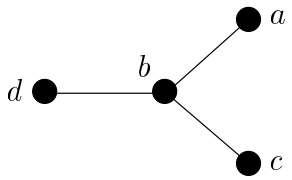


Figure 4.9: A tree with four vertices.

Inherent in the concept of a center is the idea of *distance*. So we must define distance before we can talk formally about a center. In order to speak about distances in a precise manner, we need to define a *metric*[1] on a given set.

Definition 4.15
Let S be a set. A function $D : S \times S \to \mathbb{R}$ is called a metric *on S if it satisfies the following conditions:*

1. $D(s, s) = 0$ *for all* $s \in S$.
2. $D(s, t) > 0$ *for all* $s, t \in S$ *with* $s \neq t$.
3. $D(s, t) = D(t, s)$ *for all* $s, t \in S$.
4. $D(s, u) \leq D(s, t) + D(t, u)$ *for all* s, t, *and* $u \in S$.

The ordered tuple (S, D) is called a metric space.

The function D in Definition 4.15 is really a measure of distance. By viewing it as such, all of the conditions make intuitive sense as we now explain.

1. The distance from any point to itself has a length of zero.
2. The distance between any point and any other point is always positive.
3. The distance from one point to a second is the same as the distance from the second point to the first. Hence, we can talk about the distance *between* points as we do in every day conversation.

[1] A la the *metric* unit system developed in France during the French Revolution to unify all units!

4. The distance from one point to a second point is always less than or equal to the distance between the first point and a third point plus the distance between the third point and the second one regardless of where the third point lies. In other words, one can never take a short-cut from one point to a second point by going through a third point. This inequality is called the *triangle inequality* and must be valid for all metrics D if we want to use D as a measure of distance.

Let us examine how Definition 4.15 applies to graph theory and in particular to trees.

Definition 4.16
Let G be a connected graph. For $u, v \in V(G)$ the distance *between u and v, denoted $D_G(u, v)$, is the length of the shortest path between u and v in G.*

We make a couple of observations about this definition.

➤ **Note 4.17**
If G' is a subgraph of G, then for any vertices u and v in $V(G')$ we have that

$$D_G(u, v) \leq D_{G'}(u, v).$$

We can also consider disconnected graphs and let

$$D_G(u, v) = n$$

whenever the vertices u and v are in different components of G.

Note that the maximum distance between two vertices in a connected graph is $n - 1$. So, any distance value larger than $n - 1$ is sufficient to indicate two vertices are not connected. Some authors set $D_G(u, v)$ equal to ∞ when u and v are not connected.

Note 4.17 shows how the definition of distance can be extended so that the distance between any two vertices is defined for any graph. When there is no ambiguity, sometimes we omit the subscript in D_G and just use D.

In a graph that is not a tree, there are generally several paths between a pair of vertices. We have to enumerate all these paths and find the length of the shortest one to determine the distance between two vertices. There may be several shortest paths between two vertices in a graph. Example 4.18 should clarify these concepts.

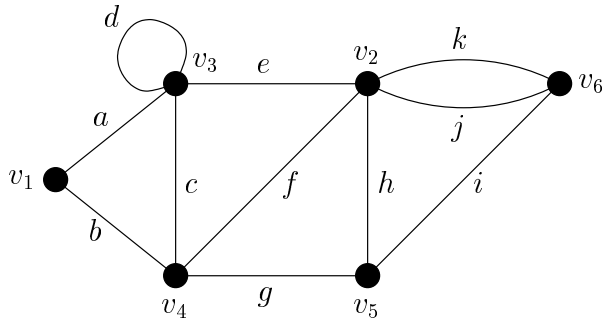


Figure 4.10: The distance between v_1 and v_2 in G is two.

☞ **Example 4.18**
In Figure 4.10 we depict a graph G on six vertices. Here are some of the paths between vertices v_1 and v_2 in G:

- ▷ (v_1, a, v_3, e, v_2),
- ▷ $(v_1, a, v_3, c, v_4, f, v_2)$,
- ▷ $(v_1, b, v_4, c, v_3, e, v_2)$,
- ▷ (v_1, b, v_4, f, v_2),
- ▷ $(v_1, b, v_4, g, v_5, h, v_2)$, *and*
- ▷ $(v_1, b, v_4, g, v_5, i, v_6, k, v_2)$.

There are two shortest paths: (v_1, a, v_3, e, v_2) *and* (v_1, b, v_4, f, v_2). *These paths each have length two. So,* $D_G(v_1, v_2)$ *equals two.*

Several efficient algorithms exist to find a shortest path between two vertices in a graph. The algorithm most commonly implemented, called DIJKSTRA'S ALGORITHM, is discussed in §15.6. By Theorem 4.7, there is only one path between any two vertices in a tree. Thus the determination of distance is much easier. We simply need to find the path between two vertices and determine its length. In the tree shown in Figure 4.9, we have $D(a, b)$ equals one, $D(a, c)$ equals two, $D(c, b)$ equals one, and so on.

Clearly, the distance $D_G(u, v)$ between u and v in a graph G satisfies conditions 1, 2, and 3 of Definition 4.15. Since $D_G(u, v)$ is the length of the

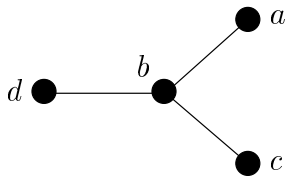


Figure 4.11: Eccentricities of the vertices of a tree: $E(a)$ equals two, $E(b)$ equals one, $E(c)$ equals two, and $E(d)$ equals two.

shortest path between vertices u and v, this path cannot be longer than another path between u and v that goes through a specified vertex w. Hence, $D_G(u,v) \leq D_G(u,w) + D_G(w,v)$. This yields the following observation.

Observation 4.19
Let G be a graph. Define $D_G(u,v)$ equal to n if u and v are in different components. The distance function $D_G : V(G) \times V(G) \to \mathbb{N} \subseteq \mathbb{R}$ between vertices of a graph is a metric.

Referring back to our original topic of the relative location of different vertices in a tree, let us define the term *eccentricity* of a vertex in a graph. This too is a useful concept in graph algorithms.

Definition 4.20
Let G be a graph and $u \in V(G)$.

- ▷ *The* eccentricity $E(u)$ *of u in G is the distance from u to the vertex farthest from u in G. That is,*

$$E(u) = \max(\{D_G(u,v) \mid v \in V(G)\})$$

- ▷ *A vertex with minimum eccentricity in a graph is called a* center.

We consider an example to clarify this definition.

☞ **Example 4.21**
The eccentricities of the four vertices shown in Figure 4.11 are $E(a)$ equals two, $E(b)$ equals one, $E(c)$ equals two, and $E(d)$ equals two. Hence, vertex b is the *center of this tree.*

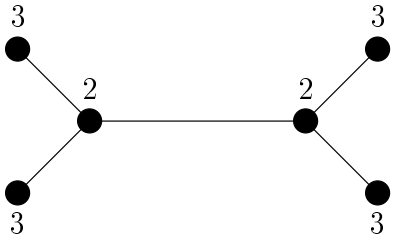


Figure 4.12: A tree with two centers.

On the other hand, consider the tree shown in Figure 4.12. Each of its six vertices is labeled with their eccentricities. This tree has two vertices with minimum eccentricity. Therefore, this tree has two centers.

The reader can easily verify that in general a graph has many centers. For example, in the cycle graph C_n every vertex is a center. König proved the following interesting theorem for trees.

Theorem 4.22
Every tree has either one or two centers.

PROOF: Let T be a tree with two or more vertices. The maximum distance from a given vertex u to any other vertex v occurs only when v is a leaf. By Theorem 4.4, T must have two or more leaves. If we delete all of the leaves from T, it follows by Note 4.5 that the resulting graph T' is still a tree. What can we observe about the eccentricities of the vertices in T'? A little deliberation reveals that the removal of all of the leaves from T uniformly reduces the eccentricities of the remaining vertices (meaning the vertices in T') by one. Therefore, all of the vertices that T had as centers remain centers in T'. From T' we can again remove all of the leaves and get another tree T''. We can continue this process until there is either one vertex, which is the center of T, or two vertices connected by an edge, which are the two centers of T. □

This process is illustrated in Figure 4.13. In this case there were three iterations required and the resulting graph had only a single vertex. Thus the original tree T has just one center.

The next corollary follows directly from the proof of Theorem 4.22.

Corollary 4.23
If a tree T has two centers, they must be adjacent.

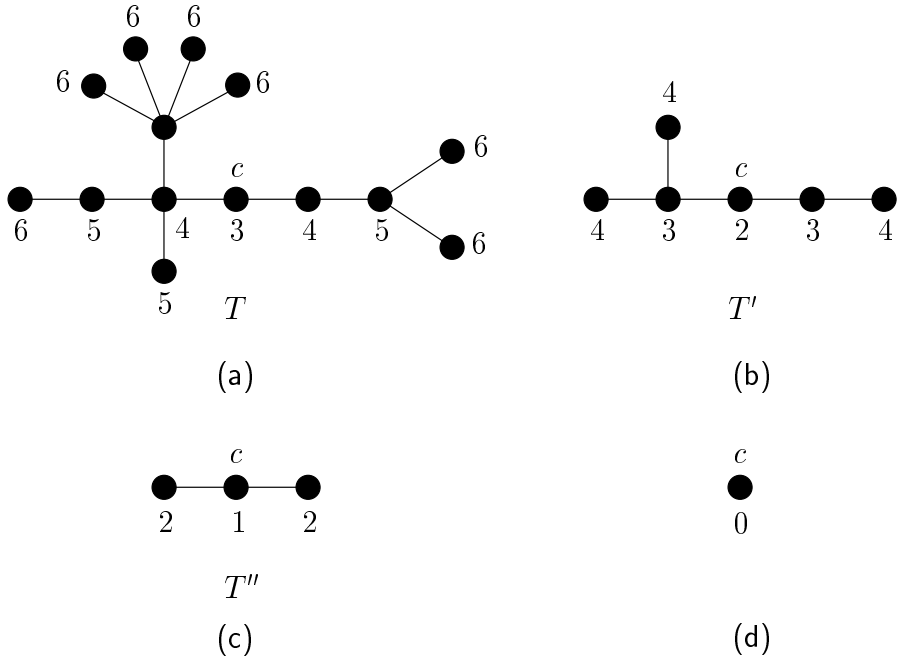


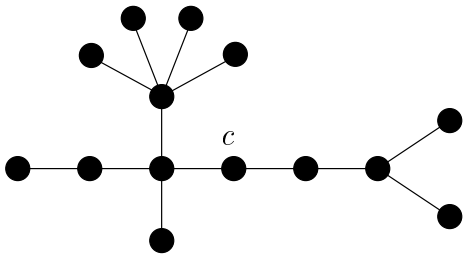
Figure 4.13: Finding a center of a tree.

Next we consider an application of these concepts to sociology. There are many other applications of this material as well, particularly in decomposing trees into subtrees in computer science problems.

☞ **Example 4.24**

Suppose that communication among 14 persons in a society is represented by the graph T shown in Figure 4.14, where the vertices represent the individuals. Two vertices are joined by an edge if those two people communicate. Since the graph is connected, we know that each member can be reached by any other member, either directly or through other members. It is important to note that the graph is a tree and so minimally connected. The group cannot afford to lose any member. If it does, some individual(s) will become isolated.

The eccentricity of each vertex u represents how close u is to the farthest member of the group. In Figure 4.14 the person corresponding to vertex c should be the leader of the group if closeness of communication is the criterion for leadership. Note that any other leader would have more communication hops to some individual. This might be an undesirable situation for a therapy group.



T

Figure 4.14: Finding the leader of a group.

In a tree the eccentricity of a center is sometimes called the *radius*. For instance the radius of the tree T shown in Figure 4.14 is three. On the other hand, the *diameter* of a tree is defined as the length of its longest path. It is left as an exercise for the reader to show that a radius in a tree is not necessarily half of its diameter (see Exercise 21).

4.7 Rooted Trees

In this section we discuss trees having one distinguished vertex called the *root*. There are many applications of rooted trees in computer science, engineering, mathematics, and other fields. We begin by defining rooted trees.

Definition 4.25
Let T be a tree and $r \in V(T)$. A rooted tree *is the ordered tuple (T, r). The vertex r is called the* root *of T.*

For instance, in Figure 4.3 the vertex N is distinguished from the rest of the vertices because this is where the mail sorting begins. We say the mail tree is rooted because of the existence of this root N. Similarly, in Figure 4.8 the "Start" vertex can be considered as the root of this tree.

☞ **Example 4.26**
Figure 4.15 shows all rooted trees on having four vertices.

Generally, the term *tree* means tree without having any root specified. However, in the context of rooted trees, the trees without a distinguished root are sometimes called *free trees* to differentiate them from rooted trees.

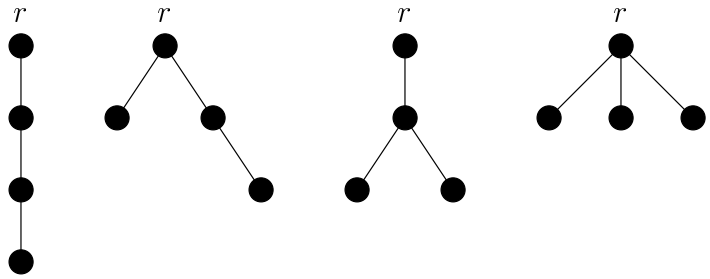


Figure 4.15: All rooted trees with four vertices.

At this point, it will be helpful to introduce some additional terminology to facilitate our discussion on trees.

Definition 4.27
Let (T, r) be a rooted tree with at least two vertices, $u \in V(T)$ be a vertex that is not the root r, and p_u the unique path between r and u in T.

1. *The* parent *of u is the neighbor of u on p_u and is denoted by parent(u).*
2. *Any neighbor of u other than parent(u) is called a* child *of u. The set of children of u is denoted by children(u).*
3. *Two vertices are called* siblings *if they have the same parent.*
4. *Any vertex on p_u other than u is called an* ancestor *of u. The set of ancestors of u is denoted by ancestors(u).*
5. *A vertex v that has u as an ancestor, that is $v \in$ ancestors(u), is called a* descendant *of u. The set of all of the descendants of a vertex u are denoted by descendants(u).*

The terminology in Definition 4.27 is directly related to family trees, where the root represents the oldest person and an edge is always present between a parent and a child. We usually draw the figures of rooted trees from the top downwards as we see in the next example.

☞ **Example 4.28**
Figure 4.16 depicts a rooted tree (T, r). For the vertex $u \in V(T)$, we have the

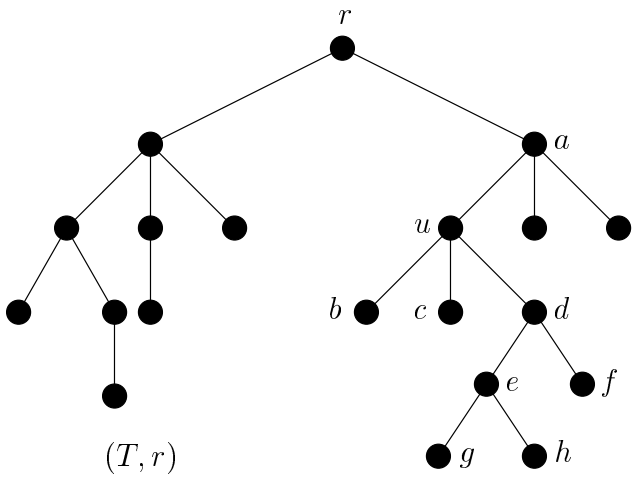


Figure 4.16: A rooted tree and relationships between its vertices.

following:

$$\begin{aligned} \textit{parent}(u) &= a \\ \textit{children}(u) &= \{b, c, d\} \\ \textit{ancestors}(u) &= \{a, r\} \\ \textit{descendants}(u) &= \{b, c, d, e, f, g, h\} \end{aligned}$$

The vertices e and f are siblings. The vertices c and f are not siblings. Note that every vertex in T is a descendant of r. That is, every vertex of T has r as an ancestor. There are no descendants of the leaves of T.

In rooted trees, a vertex's position in the representation of the tree is important when reading from left to right, particularly in *binary trees*. This is highlighted in the following definition.

Definition 4.29
A rooted tree (T, r) in which the order of every set of siblings is specified is called an ordered rooted tree.

Example 4.30 will clarify this definition.

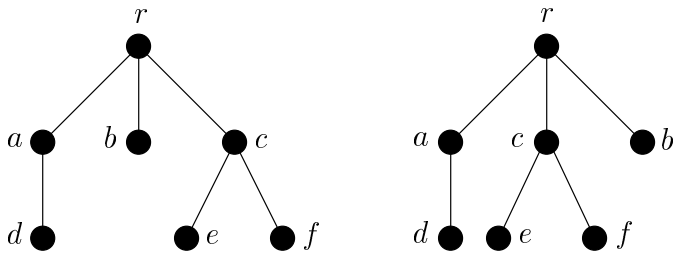


Figure 4.17: Two distinct ordered rooted trees that are identical when viewed as rooted trees.

☞ **Example 4.30**
Figure 4.17 shows two ordered rooted trees. They are the same when considered just as trees or even as rooted trees. They are not the same ordered rooted tree because the order of the siblings $\{a, b, c\}$ *differs. In the first tree the order is* (a, b, c) *but in the second tree the order is* (a, c, b).

In the next section we focus on special classes of ordered rooted trees.

4.8 Binary and Regular Binary Trees

Binary trees are an interesting class of ordered rooted trees because they are used extensively in the study of computer search methods, in modeling problems, and in coding problems.

Definition 4.31
A binary tree *is an ordered rooted tree in which each vertex has at most two children. Each child of a vertex is called either the* left *child or the* right *child. A subtree rooted at the left (right) child of a vertex* u *is known as* u*'s* left subtree *(respectively,* right subtree*).*

Figure 4.18 depicts a sample binary tree. In a binary tree drawing, the position of a vertex in the picture determines whether it is a left or right child. For example, in Figure 4.18 vertex a is a left child of r and vertex b is a right child of r.

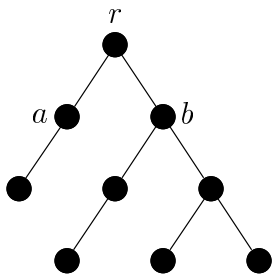


Figure 4.18: A sample binary tree.

In many applications of "binary-like" trees the ordering of the children is *not* important but it is required that each non-leaf vertex have *exactly* two children. For example, various algorithms in computer science require such a tree as input. Usually, we refer to this particular class of trees as the *regular binary trees*. We define them in what follows.

Definition 4.32
A regular binary tree *is a tree meeting the following conditions:*

1. *It has three or more vertices.*
2. *There is exactly one vertex r of degree two.*
3. *All vertices other than r have degree one or three.*

Figure 4.19 is a regular binary tree. According to Definition 4.31, the vertex of degree two is distinct from all other vertices in a regular binary tree, so this vertex can serve naturally as the root of the tree. Adopting this viewpoint, every regular binary tree can be thought of as a restricted, rooted tree.

Two properties of regular binary trees follow directly from the definition.

Observation 4.33
Let T be a regular binary tree on n vertices. We have the following:

1. *The number of vertices of T is odd.*
2. *The number of leaves in T is $(n+1)/2$.*

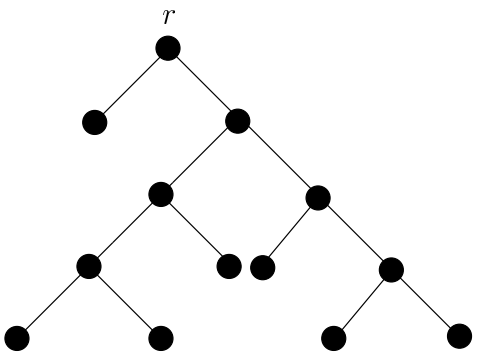


Figure 4.19: A regular binary tree rooted at vertex r of degree two.

PROOF:

PROPERTY 1: Note, there is exactly one vertex of even degree in a regular binary tree. By Corollary 1.25 the total number of all other vertices is even. Therefore, the number of vertices in T is odd.

PROPERTY 2: Let l be the number of leaves in T. There are exactly $n-l-1$ vertices of degree three. Since T is a tree, we have by Theorem 4.7 that T has $n-1$ edges. Therefore, by the Hand Shaking Theorem (Theorem 1.24), the total number of edges is

$$n-1=\frac{1}{2}\sum_{u\in V(T)} d(u)=\frac{1}{2}\left(l+2+3(n-l-1)\right).$$

This yields that l equals $(n+1)/2$. □

It follows from Observation 4.33 that the number of internal vertices in a regular binary tree is one less than the number of leaves. The next concept allows us to talk about the position of a vertex in a rooted tree.

Definition 4.34
Let (T,r) be a rooted tree. For a vertex $u\in V(T)$, the distance $D_T(u,r)$ is called the level *of the vertex u in the rooted tree. The level of u is denoted $l(u)$. The maximum level in (T,r) is called the* height *of (T,r) and is denoted $l_{\max}$. A rooted tree of height h is called an $(h+1)$-*level tree.

For all rooted trees, the root is at level zero. Let us examine the level concept with an example.

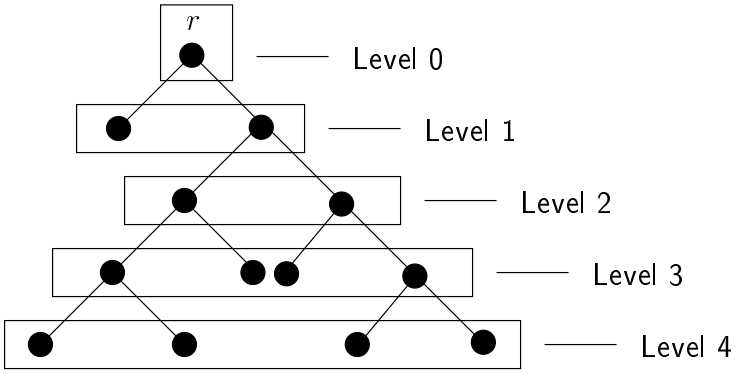


Figure 4.20: A 13-vertex, 5-level regular binary tree.

☞ **Example 4.35**
In Figure 4.20 we show a 13-vertex, 5-level regular binary tree. The height of this tree is four.

One of the most straightforward applications of regular binary trees is in search procedures. Each vertex of the tree represents a test with two possible outcomes. We start at the root. The outcome of the test at the root sends us to one of the two vertices at the next level, where further tests are made and so on. Reaching a specified leaf (the goal of the search) terminates the search. Often in a search procedure, it is important to construct a regular binary tree on n vertices that minimizes the distance between the root and the leaves. This makes for efficient searching. Clearly, only the root is at level zero, level one contains at most two vertices, level two contains at most four vertices, and so on. Therefore, the maximum number of vertices in a $(k+1)$-level regular binary tree is

$$2^0 + 2^1 + 2^2 + \cdots + 2^k = 2^{k+1} - 1.$$

Since k is less than or equal to $l_{\max}$, we have in particular that

$$2^{l_{\max}+1} - 1 \geq n.$$

Since $l_{\max}$ is a natural number, we can deduce that

$$l_{\max} \geq \lceil \log_2(n+1) - 1 \rceil .$$

On the other hand, we can maximize the distance between the root and a leaf of a regular binary tree on n vertices by having exactly two vertices at each

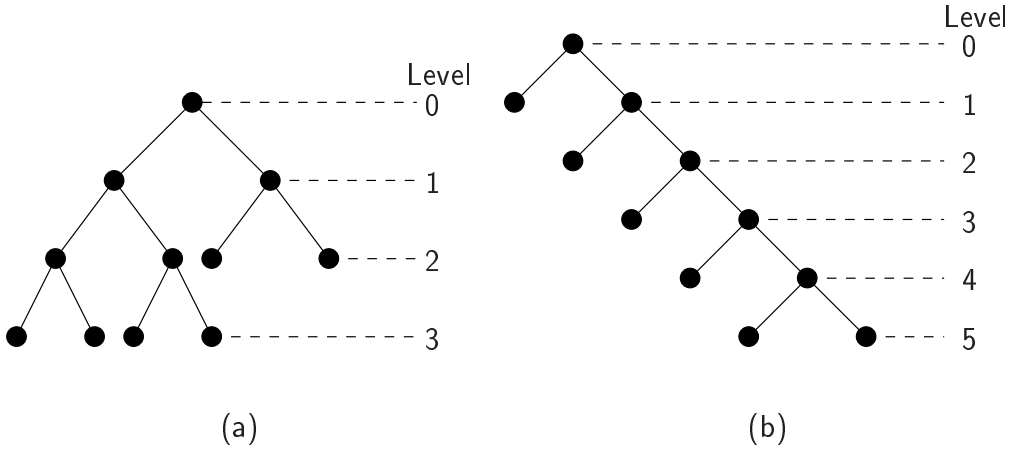


Figure 4.21: (a) The shortest and (b) the tallest 11-vertex regular binary trees.

level except level zero. Therefore,

$$l_{\max} = \frac{n-1}{2}.$$

With these observations in mind, we have the following lemma.

Lemma 4.36
For any regular binary tree (T, r) on n vertices, the height $l_{\max}$ satisfies the following:

$$\lceil \log_2(n+1) - 1 \rceil \leq l_{\max} \leq \frac{n-1}{2}.$$

The next example illustrates trees meeting the bounds of Lemma 4.36.

☞ **Example 4.37**
For n equals 11, regular binary trees realizing the extremes of Lemma 4.36 are shown in Figure 4.21. The first one has height three equal to $\lceil \log_2(11+1) - 1 \rceil$. The second one has height five equals to $(11-1)/2$.

It should be clear that search trees of a low height are more efficient than deeper search trees when the items stored at the leaves are equally likely to be selected. Computer scientists have developed many different types of tree-like data structures for efficient storage and retrieval of information. Some of these structures are *B-trees*, *heaps*, *red-black trees*, *splay trees*, and *2-3 trees*. These concepts can be pursued through any of the algorithms books cited in the references.

4.9 Path Length in a Tree

In the analysis of algorithms, we are sometimes interested in computing the sum of the levels of all of the leaves in a tree T. This quantity is known as the *path length* of T. For example, the path length of the regular binary tree in Figure 4.19 is

$$1 + 4 + 4 + 3 + 3 + 4 + 4 = 23.$$

The path lengths of the trees in Figure 4.21 are 16 and 20, respectively. The path length of a tree is important. This quantity is often directly related to the execution time of an algorithm performed on the tree. Thus it is desirable to construct trees with small path lengths. Our next observation ties in with this issue.

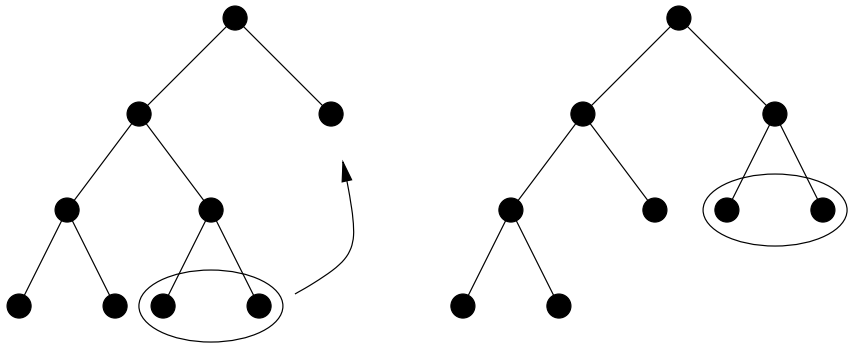
Observation 4.38
Among all regular binary trees on n vertices, the ones with the minimum path length are precisely the regular binary trees with the minimum possible height of $l_{\max} = \lceil \log_2(n+1) - 1 \rceil$ and where all of the leaves are either on level $l_{\max} - 1$ or $l_{\max}$.

PROOF: Let n be given and let k equal $l_{\max}$. Note that there is at least one leaf on level k. Since the tree is a regular binary tree, the leaf has a sibling. Hence, there are at least two leaves at level k.

Assume now that T is a regular binary tree with a leaf on level i less than or equal to $k-2$. If the path length of T is L, then by removing two leaves at level k from T and connecting them to the leaf on level i, we get another binary tree T' having three different leaves. Namely, the two removed leaves, whose level changed from k to $i+1$, and the parent of these leaves that becomes a leaf in T' at level $k-1$. Note, the leaf at level i in T where we connected the two removed leaves is not a leaf in T'. Hence, the path length L' of T' is reduced from T's path length by $2k - 2(i+1)$ and increased by $(k-1) - i$. Therefore, we have that

$$L' = L - (2k - 2(i+1)) + ((k-1) - i) \leq L - 1.$$

This path length is strictly less than that of T. Note that as a result of this procedure, we now have one less leaf on level $i \in \{1, \ldots, k-2\}$. We can continue in this manner until there are no leaves at any level other than the two



T T'

Figure 4.22: The path length of T is greater than that of T'.

deepest levels of the newly formed tree. At this point, we have achieved the minimum path length. □

The next example illustrates one step of the procedure outlined in Observation 4.38 for reducing the path length of a regular binary tree.

☞ **Example 4.39**
In Figure 4.22 the path length of T is

$$L = 3 + 3 + 3 + 3 + 1 = 13.$$

We form a new tree T' by carrying out one step of the procedure described in Observation 4.38. The path length of T' is

$$L' = 3 + 3 + 2 + 2 + 2 = 12 < 13 = L.$$

Notice that T''s path length cannot be reduced any further. Thus the procedure terminates.

More generally, one can consider the concept of *weighted path length*. In some applications, every leaf u_i of a regular binary tree has associated with it a positive real number w_i. Given weights $w_1, \ldots, w_m$ the problem is to construct a regular binary tree having m leaves that minimizes the following quantity:

$$S = \sum_{i=1}^{m} w_i \cdot l(u_i) \tag{4.3}$$

Let us illustrate the significance of this problem with a simple example. An example that every thirsty person, who is stranded in the middle of Orlando on a hot summer day, can relate to.

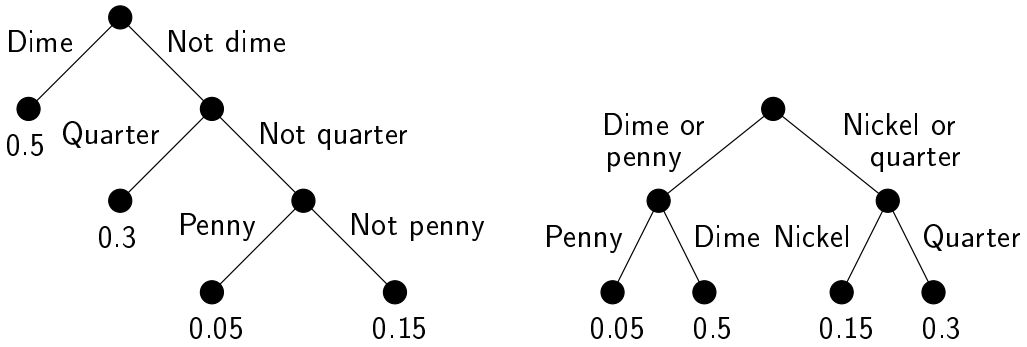
☞ **Example 4.40**

An old-fashioned Coke machine is to identify by a sequence of tests the coin that is put into it. Only pennies, nickels, dimes, and quarters can go through the machine's coin slot. Let us assume that the probabilities of a coin being a penny, a nickel, a dime, and a quarter are 0.05, 0.15, 0.5, *and* 0.30, *respectively. Each test has the effect of partitioning the four types of coins into two complementary sets and asserting the unknown coin to be in one of the two sets. Thus with four coins we have* $2^3 - 1$ *such tests. If the time taken for each test is the same, what sequence of tests will minimize the expected time taken by the Coke machine to identify the coin? (And thus allow you to buy your soda more quickly.)*

The solution requires the construction of a regular binary tree. The tree has four leaves (and therefore three internal vertices) u_1, u_2, u_3, *and* u_4 *with corresponding weights* $w_1 = 0.05$, $w_2 = 0.15$, $w_3 = 0.5$, *and* $w_4 = 0.3$ *such that the quantity* S *of Equation 4.3 is minimized. Let* t *be the time taken for each test. The solution is given in Figure 4.23(a) for which the expected time is* S *equals* $1.7t$. *Contrast this with the second regular binary tree for which the expected time is* S *equals* $2t$. *A general algorithm for constructing a regular binary tree with a minimum weighted path length was given by Huffman [19]. The resulting regular binary tree is called a* Huffman tree.

In this problem of a Coke machine, many interesting variations are possible. For example, not all possible tests may be available, or they may not all consume the same amount of time. The interested reader can explore these variations through the references.

Regular binary trees with minimum weighted path length have also been used in constructing variable-length binary codes, where the letters of the alphabet $(A, B, \ldots, Z)$ are represented by binary numbers. Since various letters have different frequencies of occurrence (frequencies are interpreted as weights w_1, w_2, $\ldots$, w_{26}), a regular binary tree with minimum weighted path length corresponds to a binary code of minimum cost. Such a code is called a *Huffman coding*. For example, the letter "e" occurs much more frequently in the English language than does the letter "x." Therefore, it makes sense to give the letter "e" a smaller code word than the letter "x" when building a good coding. A Huffman



(a) $S = 1.7$ (b) $S = 2$

Figure 4.23: Two regular binary trees with weighted leaves.

code represents the best possible coding in the sense of providing the shortest code words on average.

For more information on minimum-path regular binary trees and their applications, the reader is referred to Knuth [21].

4.10 Exercises

1. Write down three applications of trees to every day issues that were not mentioned already in the text. Describe each application in a couple of paragraphs.

2. Draw all trees of n unlabeled vertices for n equals 1, 2, 3, 4, and 5.

3. Sketch a tree for sorting people based on nine digit social security numbers. How many vertices does such a tree have? How many leaves?

4. Imagine a tree for sorting cars based on license plate numbers. Suppose each license plate has exactly eight symbols and each symbol can be any capital letter or a digit. How many vertices does such a tree have? How many leaves?

5. It can be shown that there are exactly 11 unlabeled trees on seven vertices. (Figure 4.2 does this for n equals six.) Draw these eleven trees making sure that no two are isomorphic.

6. Let Δ be greater than two. Consider all trees on n vertices where each vertex has degree Δ or less. Let $\mathcal{B}_n(\Delta)$ denote the maximum number of vertices of degree Δ that such a tree can have. Prove by induction that

$$\mathcal{B}_n(\Delta) \leq \left\lfloor \frac{n-2}{\Delta - 1} \right\rfloor . \tag{4.4}$$

   [*Hint:* This is a generalization of Example 4.8.]

7. Show that the bound in Exercise 6 is tight for infinitely many positive natural numbers n. Do this explicitly or recursively by constructing a sequence $(t_i(\Delta))_{i \geq 1}$ such that for each $n = t_i(\Delta)$ equality holds in Inequality 4.4.

8. State necessary and sufficient conditions on an ordered n-tuple of positive integers $(d_1, \ldots, d_n)$ such that there is a tree on vertices $u_1, \ldots, u_n$ with $d_T(u_i) = d_i$ for each $i \in \{1, \ldots, n\}$.

9. Let Δ be greater than two. Consider all trees where each vertex has degree Δ or less, and where the longest path is of length l or less. Show that the maximum number $\text{leaves}_{\max}$ of leaves among such trees satisfies the following:

$$\text{leaves}_{\max} \leq \Delta(\Delta - 1)^{\lfloor l/2 - 1 \rfloor}.$$

10. Use a tree to determine all of the strictly decreasing subsequences of the sequence $(8, 9, 10, 11, 5, 6, 7, 1, 2, 3, 4)$.

11. Use a tree to determine all of the strictly decreasing subsequences of the sequence $(5, 7, 3, 8, 6, 7, 5, 3, 0, 9, 5)$.

12. Let $A = \{1, 2, 3, 4, 5, 6, 7, 8\}$. Consider the finite poset $(A, \leq)$. If possible, give two elements that are comparable and two that are incomparable. Specify a minimal and a maximal element. Specify $\min(A)$ and $\max(A)$. Give a chain of longest length and an antichain of largest size. Show how Lemma 4.13 applies to this poset.

13. Let $(S, \preceq)$ be a finite poset. Show that for any $s \in S$ there is an $s' \in \max(S)$ with $s \preceq s'$.

14. Given a sequence of 100 distinct integers, what is the length of the longest strictly increasing or strictly decreasing subsequence you can rest assured it contains? How about for a sequence of 1,037 distinct integers?

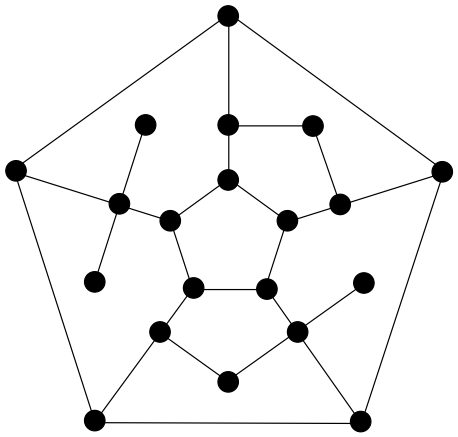


Figure 4.24: Compute the distance between each pair of vertices as requested in Exercise 17.

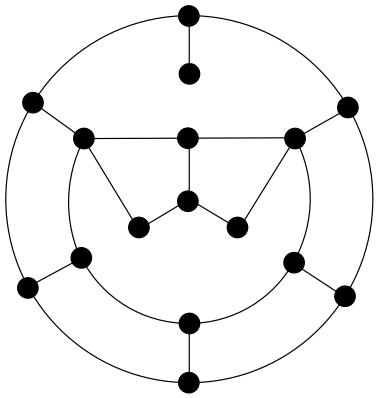


Figure 4.25: Compute the eccentricity as requested in Exercise 18.

15. Is usual addition ($+$) a metric on $\mathbb{N}$? Which properties hold?

16. Define a function on $\mathbb{N}$ that is a metric. Prove your function is a metric.

17. For the graph shown in Figure 4.24 label the vertices and specify the distances between all pairs of vertices.

18. What is the eccentricity of the graph shown in Figure 4.25?

19. For all natural numbers n, what is the eccentricity of K_n? How many centers does K_n have?

20. Suppose you delete a center from a tree thereby separating the tree into several components. Can you prove any general statements about the maximum size of these components in terms of the original size of the tree?

21. Give an example of a tree where the diameter is not twice the radius. Under what condition are the diameter and radius the same?

22. As was done for trees at the end of §4.6, we can define the *diameter* of a graph G by

$$\text{diameter}(G) = \max_{u,v \in V(G)} D_G(u,v) = d.$$

Let Δ greater than two be the maximum degree of a vertex in G. If G is simple and connected show that

$$|V(G)| \leq \frac{\Delta(\Delta-1)^d - 2}{\Delta - 2}.$$

23. Can a tree ever have more parents than children? Explain your answer.

24. Consider the rooted trees where all of the internal vertices have degree Δ greater than one. Such a tree is called a *full Δ-ary tree*. Given that the height of a full Δ-ary tree is h, find a formula for the number of leaves of such a tree and a formula for the number of internal vertices. What happens when Δ is fixed and h grows large?

25. Draw all unlabeled rooted trees on n vertices for n equals 1, 2, 3, 4, and 5.

26. Is it possible to have a tree with more than five vertices such that every vertex except the root and leaves has exactly the same number of ancestors and descendants? Explain.

27. Draw the regular binary trees on nine vertices.

28. Write a definition for regular binary trees that is equivalent to the one given in Definition 4.32 but that is expressed using the concept of internal vertices.

29. Draw all unlabeled regular binary trees with six leaves. Find the path length of each. [*Hint:* Distribute the 11 vertices (because $5 + 6 = 11$) among different levels. Observe that level zero has exactly one vertex, level one has exactly two vertices, level two can have either two or four vertices, and so on. There are six such trees and two of them are shown in Figure 4.21.]

30. Suppose you are given eight coins and are told that seven of them are of equal weight, and that one is either heavier or lighter than the rest. You are provided with an equal-arm balance for comparing coins. You can only make three weighings. Sketch a strategy in the form of a decision tree for identifying the non-conforming coin as well as for finding out whether it is heavier or lighter than the rest. (Can you solve this problem for nine coins instead of eight?)

31. Draw trees having path lengths of 7, 12, and 24.

32. For what natural numbers n can you draw trees having path length n? How about binary trees? How about regular binary trees?

33. Suppose you want to code words over the alphabet $\{a, b, c, d, e, f, g\}$ into the binary alphabet $\{0, 1\}$, where the frequencies that these letters occur in are .15, .1, .1, .1, .3, .1, and .15, respectively. Devise a Huffman code for this problem, draw its corresponding Huffman tree, and code the phrase "feed bad dead cage."