Chapter 4
Trees

4.1 Introduction

"Trees" form an important class of graphs. Of late, their importance has grown considerably in view of their wide applicability in theoretical computer science.

In this chapter, we present the basic structural properties of trees, their centers and centroids. In addition, we present two interesting consequences of the Tutte–Nash–Williams theorem on the existence of k pairwise edge-disjoint spanning trees in a simple connected graph. We also present Cayley's formula for the number of spanning trees in the labeled complete graph K_n. As applications, we present Kruskal's algorithm and Prim's algorithm, which determine a minimum-weight spanning tree in a connected weighted graph and discuss Dijkstra's algorithm, which determines a minimum-weight shortest path between two specified vertices of a connected weighted graph.

4.2 Definition, Characterization, and Simple Properties

Certain graphs derive their names from their diagrams. A "tree" is one such graph. Formally, a connected graph without cycles is defined as a *tree.* A graph without cycles is called an *acyclic graph* or a *forest.* So each component of a forest is a tree. A forest may consist of just a single tree! Figure 4.1 displays two pairs of isomorphic trees.

Remarks 4.2.1. 1. It follows from the definition that a forest (and hence a tree) is a simple graph.
2. A subgraph of a tree is a forest and a connected subgraph of a tree T is a *subtree* of T.

R. Balakrishnan and K. Ranganathan, *A Textbook of Graph Theory,*
Universitext, DOI 10.1007/978-1-4614-4529-6_4,

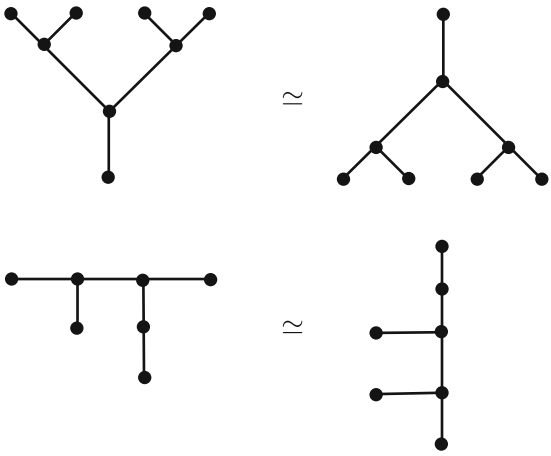


Fig. 4.1 Examples of isomorphic trees

In a connected graph, any two distinct vertices are connected by at least one path. Trees are precisely those simple connected graphs in which every pair of distinct vertices is joined by a unique path.

Theorem 4.2.2. *A simple graph is a tree if and only if any two distinct vertices are connected by a unique path.*

Proof. Let T be a tree. Suppose that two distinct vertices u and v are connected by two distinct u-v paths. Then their union contains a cycle (cf. Exercise 5.9, Chap. 1) in T, contradicting that T is a tree.

Conversely, suppose that any two vertices of a graph G are connected by a unique path. Then G is obviously connected. Also, G cannot contain a cycle, since any two distinct vertices of a cycle are connected by two distinct paths. Hence G is a tree.□

A spanning subgraph of a graph G, which is also a tree, is called a *spanning tree* of G. A connected graph G and two of its spanning trees T_1 and T_2 are shown in Fig. 4.2.

The graph G of Fig. 4.2 shows that a graph may contain more than one spanning tree; each of the trees T_1 and T_2 is a spanning tree of G.

A loop cannot be an edge of any spanning tree, since such a loop constitutes a cycle (of length 1). On the other hand, a cut edge of G must be an edge of every spanning tree of G. Theorem 4.2.3 shows that every connected graph contains a spanning tree.

Theorem 4.2.3. *Every connected graph contains a spanning tree.*

Proof. Let G be a connected graph. Let $\mathscr{C}$ be the collection of all connected spanning subgraphs of G. $\mathscr{C}$ is nonempty as $G \in \mathscr{C}$. Let $T \in \mathscr{C}$ have the fewest number of edges. Then T must be a spanning tree of G. If not, T would contain a cycle of G, and the deletion of any edge of this cycle would give a (spanning)

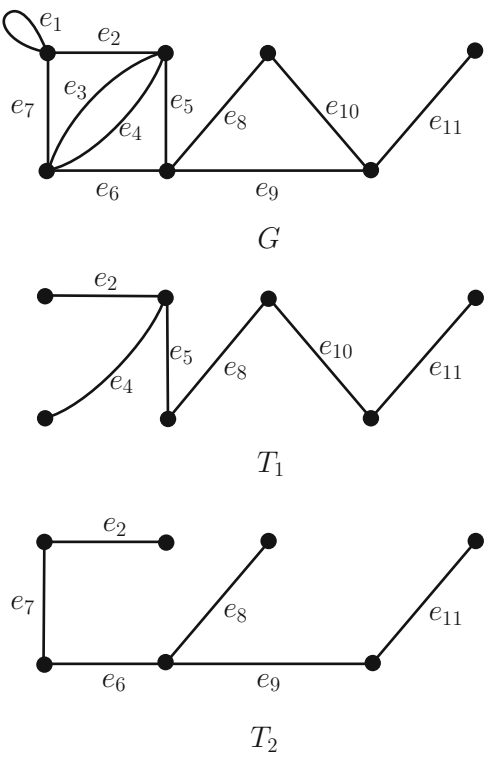


Fig. 4.2 Graph G and two of its spanning trees T_1 and T_2

subgraph in $\mathscr{C}$ having one edge less than that of T. This contradicts the choice of T. Hence, T has no cycles and is therefore a spanning tree of G. □

There is a nice relation between the number of vertices and the number of edges of any tree.

Theorem 4.2.4. *The number of edges in a tree on n vertices is* $n-1$. *Conversely, a connected graph on n vertices and* $n-1$ *edges is a tree.*

Proof. Let T be a tree. We use induction on n to prove that $m = n-1$. When $n = 1$ or $n = 2$, the result is straightforward.

Now assume that the result is true for all trees on $(n-1)$ or fewer vertices, $n \geq 3$. Let T be a tree with n vertices. Let $e = uv$ be an edge of T. Then uv is the unique path in T joining u and v. Hence the deletion of e from T results in a disconnected graph having two components T_1 and T_2. Being connected subgraphs of a tree, T_1 and T_2 are themselves trees. As $n(T_1)$ and $n(T_2)$ are less than $n(T)$, by an induction hypothesis, $m(T_1) = n(T_1) - 1$ and $m(T_2) = n(T_2) - 1$. Therefore, $m(T) = m(T_1) + m(T_2) + 1 = n(T_1) - 1 + n(T_2) - 1 + 1 = n(T_1) + n(T_2) - 1 = n(T) - 1$. Hence, the result is true for T. By induction, the result follows in one direction.

Conversely, let G be a connected graph with n vertices and $n-1$ edges. By Theorem 4.2.3, there exists a spanning tree T of G. T has n vertices and being a tree has $(n-1)$ edges. Hence $G = T$, and G is a tree. □

Exercise 2.1. Give an example of a graph with n vertices and $n-1$ edges that is not a tree.

Theorem 4.2.5. *A tree with at least two vertices contains at least two pendant vertices (i.e., end vertices or vertices of degree* 1*).*

Proof. Consider a longest path P of a tree T. The end vertices of P must be pendant vertices of T; otherwise, at least one of the end vertices of P has a second neighbor in P, and this yields a cycle, a contradiction. □

Corollary 4.2.6. *If $\delta(G) \geq 2$, G contains a cycle.*

Proof. If G has no cycles, G is a forest and hence $\delta(G) \leq 1$ by Theorem 4.2.5. □

Exercise 2.2. Show that a simple graph with ω components is a forest if and only if $m = n - \omega$.

Exercise 2.3. A vertex v of a tree T with at least three vertices is a cut vertex of T if and only if v is not a pendant vertex.

Exercise 2.4. Prove that every tree is a bipartite graph.

Our next result is a characterization of trees.

Theorem 4.2.7. *A connected graph G is a tree if and only if every edge of G is a cut edge of G.*

Proof. If G is a tree, there are no cycles in G. Hence, no edge of G can belong to a cycle. By Theorem 3.2.7, each edge of G is a cut edge of G. Conversely, if every edge of a connected graph G is a cut edge of G, then G cannot contain a cycle, since no edge of a cycle is a cut edge of G. Hence, G is a tree. □

Theorem 4.2.8. *A connected graph G with at least two vertices is a tree if and only if its degree sequence $(d_1, d_2, \ldots, d_n)$ satisfies the condition: $\sum_{i=1}^{n} d_i = 2(n-1)$ with $d_i > 0$ for each i.*

Proof. Let G be a tree. As G is connected and nontrivial, it can have no isolated vertex. Hence every term of the degree sequence of G is positive. Further, by Theorem 1.4.4, $\sum_{i=1}^{n} d_i = 2m = 2(n-1)$.

Conversely, assume that the condition $\sum_{i=1}^{n} d_i = 2(n-1)$ holds. This implies that $m = n - 1$ as $\sum_{i=1}^{n} d_i = 2m$. Now apply Theorem 4.2.4. □

Lemma 4.2.9. *If u and v are nonadjacent vertices of a tree T, then $T + uv$ contains a unique cycle.*

Proof. If P is the unique u-v path in T, then $P + uv$ is a cycle in $T + uv$. It is unique, as the path P is unique in T. □

Example 4.2.10. Prove that if $m(G) = n(G)$ for a simple connected graph G, then G is unicyclic, that is, a graph containing exactly one cycle.

Proof. By Theorem 4.2.3, G contains a spanning tree T. As T has $n(G) - 1$ edges, $E(G)\backslash E(T)$ consists of a single edge e. Then $G = T \cup e$ is unicyclic. □

Exercise 2.5. If for a simple graph G, $m(G) \geq n(G)$, prove that G contains a cycle.

Exercise 2.6. Prove that every edge of a connected graph G that is not a loop is in some spanning tree of G.

Exercise 2.7. Prove that the following statements are equivalent:

(i) G is connected and unicyclic (i.e., G has exactly one cycle).
(ii) G is connected and $n = m$.
(iii) For some edge e of G, $G - e$ is a tree.
(iv) G is connected and the set of edges of G that are not cut edges forms a cycle.

Example 4.2.11. Prove that for a simple connected graph G, $L(G)$ is isomorphic to G if and only if G is a cycle.

Proof. If G is a cycle, then clearly $L(G)$ is isomorphic to G. Conversely, let $G \simeq L(G)$. Then $n(G) = n(L(G))$, and $m(G) = m(L(G))$. But since $n(L(G)) = m(G)$, we have $m(G) = n(G)$. By Example 4.2.10, G is unicyclic. Let $C = v_1v_2 \ldots v_kv_1$ be the unique cycle in G. If $G \neq C$, there must be an edge $e \notin E(C)$ incident with some vertex v_i of C (as G is connected). Thus, there is a star with at least three edges at v_i. This star induces a clique of size at least 3 in $L(G)$ ($\simeq G$). This shows that there exists at least one more cycle in $L(G)$ distinct from the cycle corresponding to C in G. This contradicts the fact that $L(G) \simeq G$ (as G is unicyclic). □

4.3 Centers and Centroids

There are certain parameters attached to any connected graph. These are defined below.

Definitions 4.3.1. Let G be a connected graph.

1. The *diameter* of G is defined as $\max\{d(u, v) : u, v \in V(G)\}$ and is denoted by $\text{diam}(G)$.
2. If v is a vertex of G, its *eccentricity* $e(v)$ is defined by $e(v) = \max\{d(v, u) : u \in V(G)\}$.
3. The *radius* $r(G)$ of G is the minimum eccentricity of G; that is, $r(G) = \min\{e(v) : v \in V(G)\}$. Note that $\text{diam}(G) = \max\{e(v) : v \in V(G)\}$.
4. A vertex v of G is called a *central vertex* if $e(v) = r(G)$. The set of central vertices of G is called the *center* of G.

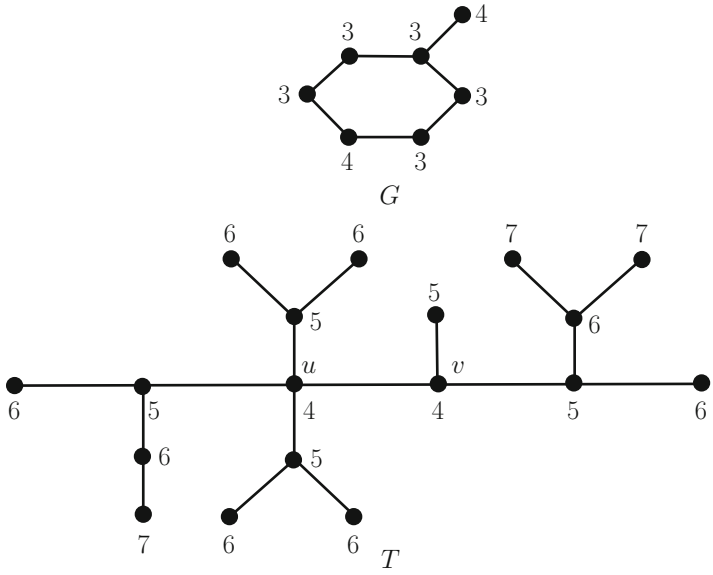


Fig. 4.3 Eccentricities of vertices for graphs G and T

Example 4.3.2. Figure 4.3 displays two graphs T and G with the eccentricities of their vertices. We find that $r(T) = 4$ and $\text{diam}(T) = 7$. Each of u and v is a central vertex of T. Also, $r(G) = 3$ and $\text{diam}(G) = 4$. Further, G has five central vertices.

Remark 4.3.3. It is obvious that $r(G) \leq \text{diam}(G)$. For a complete graph, $r(G) = \text{diam}(G) = 1$. For a complete bipartite graph $G(X, Y)$ with $|X| \geq 2$ and $|Y| \geq 2$, $r(G) = \text{diam}(G) = 2$. For the graphs of Fig. 4.3, $r(G) < \text{diam}(G)$. The terms "radius" and "diameter" tempt one to expect that $\text{diam}(G) = 2r(G)$. But this need not be the case as the complete graphs and the graphs of Fig. 4.3 show. In a tree, for any vertex u, $d(u, v)$ is maximum only when v is a pendant vertex. We use this observation in the proof of Theorem 4.3.4.

Theorem 4.3.4 (Jordan [117]). *Every tree has a center consisting of either a single vertex or two adjacent vertices.*

Proof. The result is obvious for the trees K_1 and K_2. The vertices of K_1 and K_2 are central vertices. Now let T be a tree with $n(T) \geq 3$. Then T has at least two pendant vertices (cf. Theorem 4.2.5). Clearly, the pendant vertices of T cannot be central vertices. Delete all the pendant vertices from T. This results in a subtree T' of T. As any maximum-distance path in T from any vertex of T' ends at a pendant vertex of T, the eccentricity of each vertex of T' is one less than the eccentricity of the same vertex in T. Hence the vertices of minimum eccentricity of T' are the same as those of T. In other words, T and T' have the same center. Now, if T'' is the tree obtained from T' by deleting all the pendant vertices of T', then T'' and T' have the same center. Hence the centers of T'' and T are the same. Repeat the

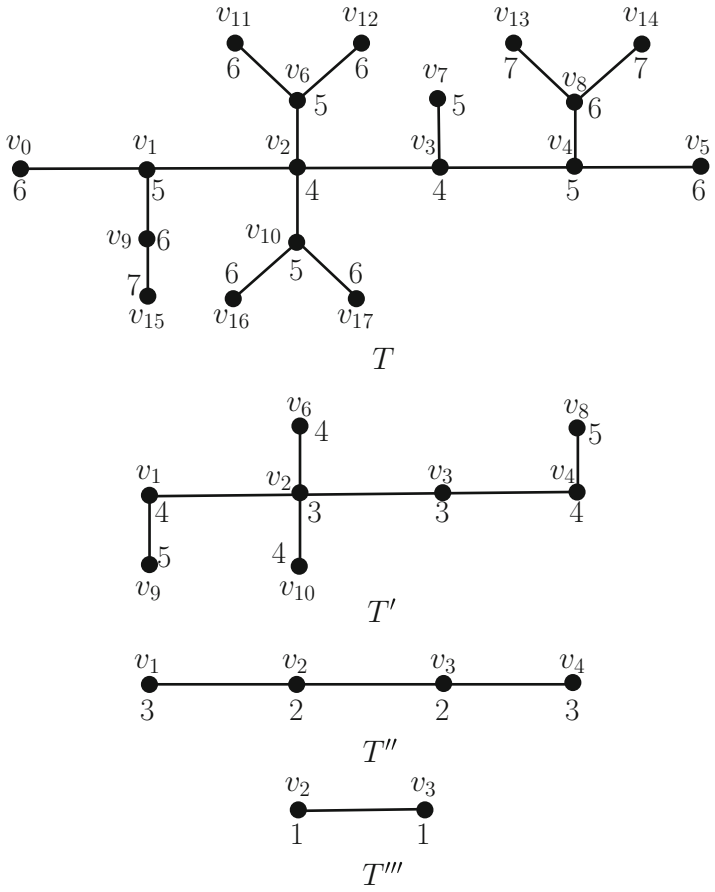


Fig. 4.4 Determining the center of tree T

process of deleting the pendant vertices in the successive subtrees of T until there results a K_1 or K_2. This will always be the case as T is finite. Hence the center of T is either a single vertex or a pair of adjacent vertices. □

The process of determining the center described above is illustrated in Fig. 4.4 for the tree T of Fig. 4.3. We observe that the center of T consists of the pair of adjacent vertices v_2 and v_3.

Exercise 3.1. Construct a tree with 85 vertices that has $\Delta = 5$ and the center consisting of a single vertex.

Exercise 3.2. Show that an automorphism of a tree on an odd number (≥ 3) of vertices has a fixed vertex; that is, for any automorphism f of a tree T with $n = 2k + 1$ ($k \geq 1$) vertices, there exists a vertex v of T with $f(v) = v$. (Hint: Use the fact that f permutes the end vertices of T.)

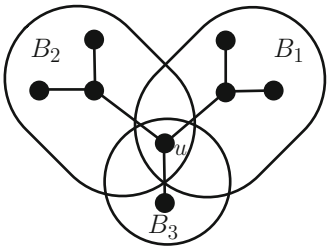


Fig. 4.5 Tree showing three branches at u

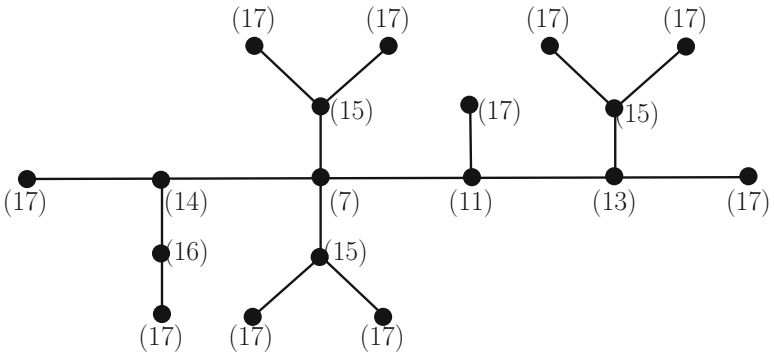


Fig. 4.6 Weights of vertices of a tree

Exercise 3.3. Show that the distinct eccentricities of the vertices of a (connected) graph G form a set of consecutive integers starting from the radius of G and ending in the diameter of G.

Definitions 4.3.5. 1. A *branch* at a vertex u of a tree T is a maximal subtree containing u as an end vertex. Hence the number of branches at u is $d(u)$.

For instance, in Fig. 4.5, there are three branches of the tree at u.

2. The *weight* of a vertex u of T is the maximum number of edges in any branch at u.
3. A vertex v is a *centroid vertex* of T if v has minimum weight. The set of all centroid vertices is called the *centroid* of T.

In Fig. 4.6 the numbers in the parentheses indicate the weights of the corresponding vertices. It is clear that all the end vertices of T have the same weight, namely, $m(T)$.

As in the case of centers, any tree has a centroid consisting of either two adjacent vertices or a single vertex. But there is no relation between the center and centroid of a tree either with regard to the number of vertices or with regard to their location.

Exercise 3.4. Give an example of

(i) A tree with just one central vertex that is also a centroidal vertex;
(ii) A tree with two central vertices, one of which is also a centroidal vertex;
(iii) A tree with two centroidal vertices, one of which is also a central vertex;

(iv) A tree with two central vertices, both of which are also centroidal vertices; and
(v) A tree with a disjoint center and centroid.

Exercise 3.5. Show that the radius of a tree T is equal to $\left\lceil \frac{\text{diam}(T)}{2} \right\rceil$.

Exercise 3.6. Show that in a tree, any path of maximum length contains the center of the tree.

Exercise 3.7. Show that the center of a tree consists of two adjacent vertices if and only if its diameter is even.

4.4 Counting the Number of Spanning Trees

Counting the number of spanning trees in a graph occurs as a natural problem in many branches of science. Spanning trees were used by Kirchoff to generate a "cycle basis" for the cycles in the graphs of electrical networks. In this section, we consider the enumeration of spanning trees in graphs.

The number of spanning trees of a connected labeled graph G will be denoted by $\tau(G)$. If G is disconnected, we take $\tau(G) = 0$. There is a recursive formula for $\tau(G)$. Before we establish this formula, we shall define the concept of *edge contraction* in graphs.

Definition 4.4.1. An edge e of a graph G is said to be *contracted* if it is deleted from G and its ends are identified. The resulting graph is denoted by $G \circ e$.

Edge contraction is illustrated in Fig. 4.7.

If e is not a loop of G, then $n(G \circ e) = n(G) - 1$, $m(G \circ e) = m(G) - 1$, and $\omega(G \circ e) = \omega(G)$. For a loop e, $n(G \circ e) = n(G)$, $m(G \circ e) = m(G) - 1$, and $\omega(G \circ e) = \omega(G)$. Theorem 4.4.2 gives a recursive formula for $\tau(G)$.

Theorem 4.4.2. *If e is not a loop of a connected graph G, $\tau(G) = \tau(G - e) + \tau(G \circ e)$.*

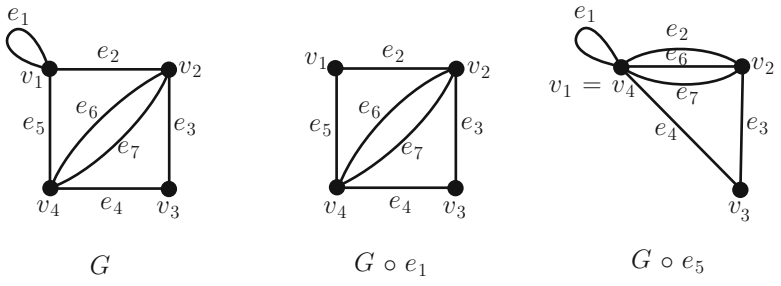


Fig. 4.7 Edge contraction

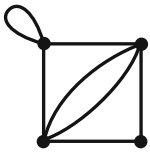
Proof. $\tau(G)$ is the sum of the number of spanning trees of G containing e and the number of spanning trees of G not containing e.

Since $V(G-e) = V(G)$, every spanning tree of $G-e$ is a spanning tree of G not containing e, and conversely, any spanning tree of G for which e is not an edge is also a spanning tree of $G-e$. Hence the number of spanning trees of G not containing e is precisely the number of spanning trees of $G-e$, that is, $\tau(G-e)$. If T is a spanning tree of G containing e, the contraction of e in both T and G results in a spanning tree $T \circ e$ of $G \circ e$.

Conversely, if T_0 is a spanning tree of $G \circ e$, there exists a unique spanning tree T of G containing e such that $T \circ e = T_0$. Thus, the number of spanning trees of G containing e is $\tau(G \circ e)$. Hence $\tau(G) = \tau(G-e) + \tau(G \circ e)$. □

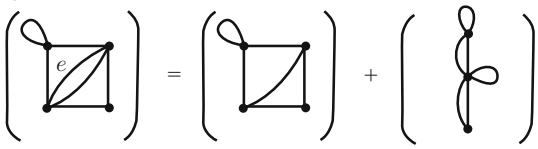
We illustrate below the use of Theorem 4.4.2 in calculating the number of spanning trees. In this illustration, each graph within parentheses stands for the number of its spanning trees. For example,  stands for the number of spanning trees of C_4.

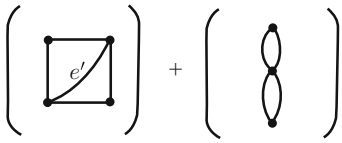
Example 4.4.3. Find $\tau(G)$ for the following graph G:

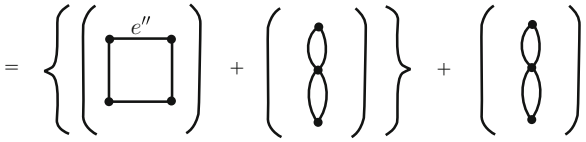


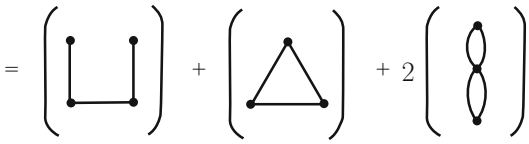
G

Proof.





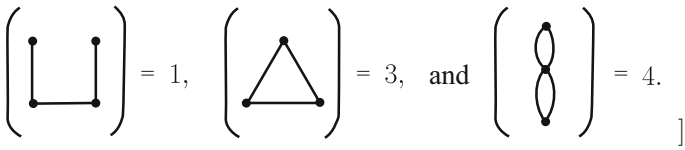




$= 1 + 3 + 2(4)$

$= 12.$

[By enumeration,



]

Hence $\tau(G) = 12$. □

We have seen in Sect. 3.2 that every connected graph has a spanning tree. When will it have k edge-disjoint spanning trees? An answer to this interesting question was given by both Tutte [181] and Nash-Williams [145] at just about the same time.

Theorem 4.4.4 (Tutte [181]; Nash-Williams [145]). *A simple connected graph G contains k pairwise edge-disjoint spanning trees if and only if for each partition $\mathscr{P}$ of $V(G)$ into p parts, the number $m(\mathscr{P})$ of edges of G joining distinct parts is at least $k(p-1)$, $2 \le p \le |V(G)|$.*

Proof. We prove only the easier part of the theorem (necessity of the condition). Suppose G has k pairwise edge-disjoint spanning trees. If T is one of them and if $\mathscr{P} = \{V_1, \ldots, V_p\}$ is a partition of $V(G)$ into p parts, then G must have at least $|\mathscr{P}| - 1$ edges of T. As this is true for each of the k pairwise edge-disjoint trees of G, the number of edges joining distinct parts of $\mathscr{P}$ is at least $k(p-1)$. □

For the proof of the converse part of the theorem, we refer the reader to the references cited.

As a consequence of Theorem 4.4.4, we obtain immediately at least one family of graphs that possesses the property stated in the theorem.

Corollary 4.4.5. *Every $2k$-edge-connected ($k \geq 1$) graph contains k pairwise edge-disjoint spanning trees.*

Proof. Let G be $2k$-edge connected, and let $\mathscr{P} = \{V_1, \ldots, V_p\}$ be a partition of V into p subsets. By hypothesis on G, there are at least $2k$ edges from each part V_i to $V \setminus V_i = \bigcup_{\substack{j=1 \\ j \neq i}}^{p} V_j$. The total number of such edges is at least kp (as each such edge is counted twice). Hence, $m(\mathscr{P}) \geq kp > k(p-1)$. Theorem 4.4.4 now ensures that there are at least k pairwise edge-disjoint spanning trees in G. □

Setting $k = 2$ in the above corollary, we get the result of Kundu.

Corollary 4.4.6 (Kundu [128]). *Every 4-edge-connected graph contains two edge-disjoint spanning trees.*

Corollary 4.4.7. *Every 3-edge-connected graph G has three spanning trees whose intersection is a spanning totally disconnected subgraph of G.*

Proof. Let G be a 3-edge-connected graph. Duplicate each edge of G by a parallel edge. The resulting graph, say, G', is 6-edge connected, and hence by Corollary 4.4.5, G' has three pairwise edge-disjoint spanning trees, say, T_1', T_2', and T_3'. Hence $E(T_1' \cap T_2' \cap T_3') = \phi$. Let T_i, $1 \leq i \leq 3$, be the tree obtained from T_i' by replacing any parallel edge of G' by its original edge in G. Then, clearly, T_1, T_2, and T_3 are three spanning trees of G with $E(T_1 \cap T_2 \cap T_3) = \phi$ because neither an edge of G nor its parallel edge can belong to all of T_1', T_2', and T_3'. □

4.5 Cayley's Formula

Cayley was the first mathematician to obtain a formula for the number of spanning trees of a labeled complete graph.

Theorem 4.5.1 (Cayley [33]). *$\tau(K_n) = n^{n-2}$, where K_n is a labeled complete graph on n vertices, $n \geq 2$.*

Before we prove Theorem 4.5.1, we establish two lemmas.

Lemma 4.5.2. *Let $(d_1, \ldots, d_n)$ be a sequence of positive integers with $\sum_{i=1}^{n} d_i = 2(n-1)$. Then there exists a tree T with vertex set $\{v_1, \ldots, v_n\}$ and $d(v_i) = d_i$, $1 \leq i \leq n$.*

Proof. It is easy to prove the result by induction on n. □

Lemma 4.5.3. *Let $\{v_1, \ldots, v_n\}$, $n \geq 2$ be given and let $\{d_1, \ldots, d_n\}$ be a sequence of positive integers such that $\sum_{i=1}^{n} d_i = 2(n-1)$. Then the number of trees with $\{v_1, \ldots, v_n\}$ as the vertex set in which v_i has degree d_i, $1 \leq i \leq n$, is $\frac{(n-2)!}{(d_1-1)! \ldots (d_n-1)!}$.*

Proof. We prove the result by induction on n. For $n = 2$, $2(n-1) = 2$, so that $d_1 + d_2 = 2$. Since $d_1 \geq 1$ and $d_2 \geq 1$, $d_1 = d_2 = 1$. Hence K_2 is the only tree in which v_i has degree d_i, $i = 1, 2$. So the result is true for $n = 2$. Now assume that the result is true for all positive integers up to $n-1$, $n \geq 3$. Let $\{d_1, \ldots, d_n\}$ be a sequence of positive integers such that $\sum_{i=1}^{n} d_i = 2(n-1)$, and let $\{v_1, \ldots, v_n\}$ be any set. If $d_i \geq 2$ for every i, $1 \leq i \leq n$, then $\sum_{i=1}^{n} d_i \geq 2n$. Hence, there exists an i, $1 \leq i \leq n$, for which $d_i = 1$. For the sake of definiteness, assume that $d_n = 1$. By Lemma 4.5.2, there exists a tree T with $V(T) = \{v_1, \ldots, v_n\}$ and degree of $v_i = d_i$. Let v_j be the unique vertex of T adjacent to v_n. Delete v_n from T. The resulting graph is a tree T' with $\{v_1, \ldots, v_{n-1}\}$ as its vertex set and $(d_1, \ldots, d_{j-1}, d_j - 1, d_{j+1}, \ldots, d_{n-1})$ as its degree sequence.

In the opposite direction, given a tree T' with $\{v_1, \ldots, v_{n-1}\}$ as its vertex set and $(d_1, \ldots, d_{j-1}, d_j - 1, d_{j+1}, \ldots, d_{n-1})$ as its degree sequence, a tree T with vertex set $\{v_1, \ldots, v_n\}$ and degree sequence $(d_1, \ldots, d_n)$, $d_n = 1$, can be obtained by introducing a new vertex v_n and taking $T = T' + v_j v_n$. Hence the number of trees with vertex set $\{v_1, \ldots, v_n\}$ and degree sequence $(d_1, \ldots, d_n)$ with $d_n =$ degree of $v_n = 1$ and v_n adjacent to v_j is the same as the number of trees with vertex set $\{v_1, \ldots, v_{n-1}\}$ and degree sequence $(d_1, \ldots, d_{j-1}, d_j - 1, d_{j+1}, \ldots, d_{n-1})$. By the induction hypothesis, the latter number is equal to

$$\frac{(n-3)!}{(d_1-1)!\ \ldots\ (d_{j-1}-1)!\,(d_j-2)!\,(d_{j+1}-1)!\ \ldots\ (d_{n-1}-1)!}$$
$$= \frac{(n-3)!(d_j-1)}{(d_1-1)!\ \ldots\ (d_{j-1}-1)!\,(d_j-1)!\,(d_{j+1}-1)!\ \ldots\ (d_{n-1}-1)!}.$$

Summing over j, the number of trees with $\{v_1, \ldots, v_n\}$ as its vertex set and $(d_1, \ldots, d_n)$ as its degree sequence is

$$\sum_{j=1}^{n-1} \frac{(n-3)!\,(d_j-1)}{(d_1-1)!\ \ldots\ (d_{n-1}-1)!}$$
$$= \frac{(n-3)!}{(d_1-1)!\ \ldots\ (d_{n-1}-1)!} \sum_{j=1}^{n-1} (d_j - 1)$$
$$= \frac{(n-3)!}{(d_1-1)!\ \ldots\ (d_{n-1}-1)!} \left[\left(\sum_{j=1}^{n-1} d_j \right) - (n-1) \right]$$
$$= \frac{(n-3)!}{(d_1-1)!\ \ldots\ (d_{n-1}-1)!} [(2n-3) - (n-1)]$$
$$= \frac{(n-3)!}{(d_1-1)!\ \ldots\ (d_{n-1}-1)!} (n-2)$$

$$= \frac{(n-2)!}{(d_1-1)!\ \ldots\ (d_{n-1}-1)!}$$
$$= \frac{(n-2)!}{(d_1-1)!\ \ldots\ (d_n-1)!} \text{ (recall that } d_n = 1).$$

This completes the proof of Lemma 4.5.2. □

Proof of theorem 4.5.1. The total number of trees T_n with vertex set $\{v_1, \ldots, v_n\}$ is obtained by summing over all possible sequences $(d_1, \ldots, d_n)$ with $\sum_{i=1}^{n} d_i = 2n-2$. Hence,

$$\tau(K_n) = \sum_{d_i \geq 1} \frac{(n-2)!}{(d_1-1)!\ \ldots\ (d_n-1)!} \text{ with } \sum_{i=1}^{n} d_i = 2n-2$$
$$= \sum_{k_i \geq 0} \frac{(n-2)!}{k_1!\ \ldots\ k_n!} \text{ with } \sum_{i=1}^{n} k_i = n-2, \text{ where } k_i = d_i - 1, 1 \leq i \leq n.$$

Putting $x_1 = x_2 = \cdots = x_n = 1$ and $m = n-2$ in the multinomial expansion $(x_1 + x_2 + \cdots + x_n)^m = \sum_{k_i \geq 0} \frac{x_1^{k_1} x_2^{k_2} \ldots x_n^{k_n}}{k_1! k_2! \ldots k_n!} m!$ with $(k_1 + k_2 + \cdots + k_n) = m$, we get $n^{n-2} = \sum_{k_i \geq 0} \frac{(n-2)!}{k_1! k_2! \ldots k_n!}$ with $(k_1 + k_2 + \cdots + k_n) = n-2$. Thus, $\tau(K_n) = n^{n-2}$. □

4.6 Helly Property

Definitions 4.6.1. A family $\{A_i : i \in I\}$ of subsets of a set A is said to satisfy the *Helly property* if, whenever $J \subseteq I$ and $A_i \cap A_j \neq \phi$ for every $i, j \in J$, then $\bigcap_{j \in J} A_j \neq \phi$.

Theorem 4.6.2. *Any family of subtrees of a tree satisfies the Helly property.*

Proof. Let $\mathcal{F} = \{T_i : i \in I\}$ be a family of subtrees of a tree T. Suppose that for all $i, j \in J \subseteq I$, $T_i \cap T_j \neq \phi$. We have to prove that $\bigcap_{j \in J} T_j \neq \phi$. If for some $i \in J$, tree T_i is a single-vertex tree $\{v\}$ (i.e., K_1), then, clearly, $\bigcap_{j \in J} T_j = \{v\}$. We therefore suppose that each tree $T_i \in \mathcal{F}$ with $i \in J$ has at least two vertices.

We now apply induction on the number of vertices of T. Let the result be true for all trees with at most n vertices, and let T be a tree with $(n+1)$ vertices. Let v_0 be an end vertex of T, and u_0 its unique neighbor in T. Let $T_i' = T_i - v_0$, $i \in J$, and $T' = T - v_0$. (If $v_0 \notin T_i$, we take $T_i' = T_i$.) By the induction assumption, the result is true for the tree T'. Moreover, $T_i' \cap T_j' \neq \phi$ for any $i, j \in J$. In fact, if T_i and T_j have a vertex u $(\neq v_0)$ in common, then T_i' and T_j' also have u in common, whereas

if T_i and T_j have v_0 in common, then T_i and T_j have u_0 also in common and so do T_i' and T_j'. Hence, by the induction assumption, $\bigcap_{j\in J} T_j' \neq \phi$, and therefore $\bigcap_{j\in J} T_j \neq \phi$. □

Exercise 6.1. In the cycle C_5, give a family of five paths such that the intersection of the vertex sets of any two of them is nonempty while the intersection of the vertex sets of all of them is empty.

Exercise 6.2. Prove that a connected graph G is a tree if and only if every family of paths in G satisfies the Helly property.

4.7 Applications

We conclude this chapter by presenting some immediate applications of trees in everyday life problems.

4.7.1 The Connector Problem

Problems 1. Various cities in a country are to be linked via roads. Given the various possibilities of connecting the cities and the costs involved, what is the most economical way of laying roads so that in the resulting road network, any two cities are connected by a chain of roads? Similar problems involve designing railroad networks and water-line transports.

Problems 2. A layout for a housing settlement in a city is to be prepared. Various locations of the settlement are to be linked by roads. Given the various possibilities of linking the locations and their costs, what is the minimum-cost layout so that any two locations are connected by a chain of roads?

Problems 3. A layout for the electrical wiring of a building is to be prepared. Given the costs of the various possibilities, what is the minimum-cost layout?

These three problems are particular cases of a graph-theoretical problem known as the *connector problem*.

Definition 4.7.1. Let G be a graph. To each edge e of G, we associate a nonnegative number $w(e)$ called its *weight*. The resulting graph is a *weighted graph*. If H is a subgraph of G, the sum of the weights of the edges of H is called the *weight* of H. In particular, the sum of the weights of the edges of a path is called the *weight of the path*.

We shall now concentrate on Problem 1. Problems 2 and 3 can be dealt with similarly. Let G be a graph constructed with the set of cities as its vertex set. An edge of G corresponds to a road link between two cities. The cost of constructing a road link is the weight of its corresponding edge. Then a minimum-weight spanning tree of G provides the most economical layout for the road network.

We present two algorithms, Kruskal's algorithm and Prim's algorithm, for determining a minimum-weight spanning tree in a connected weighted graph. We can assume, without loss of generality, that the graph is simple because, since no loop can be an edge of a spanning tree, we can discard all loops. Also, since we are interested in determining a minimum-weight spanning tree, we can retain, from a set of multiple edges having the same ends, an edge with the minimum weight, and we can discard all the others.

First, we describe Kruskal's algorithm [127].

4.7.2 Kruskal's Algorithm

Let G be a simple connected weighted graph with edge set $E = \{e_1, \ldots, e_m\}$. The three steps of the algorithm are as follows:

Step 1 : Choose an edge e_1 with its weight $w(e_1)$ as small as possible.

Step 2 : If the edges $e_1, e_2, \ldots, e_i$, $i \geq 1$, have already been chosen, choose e_{i+1} from the set $E \backslash \{e_1, e_2, \ldots, e_i\}$ such that

(i) The subgraph induced by the edge set $\{e_1, e_2, \ldots, e_{i+1}\}$ is acyclic, and

(ii) $w(e_{i+1})$ is as small as possible subject to (i).

Step 3 : Stop when step 2 cannot be implemented further.

We now show that Kruskal's algorithm does indeed produce a minimum-weight spanning tree.

Theorem 4.7.2. *Any spanning tree produced by Kruskal's algorithm is a minimum-weight spanning tree.*

Proof. Let G be a simple connected graph of order n with edge set $E(G) = \{e_1, \ldots, e_m\}$. Let T^* be a spanning tree produced by Kruskal's algorithm and let $E(T^*) = \{e_1, \ldots, e_{n-1}\}$. For any spanning tree T of G, let $f(T)$ be the least value of i such that $e_i \notin E(T)$. Suppose T^* is not of minimum weight. Let T_0 be any minimum-weight spanning tree with $f(T_0)$ as large as possible.

Suppose $f(T_0) = k$. This means that $e_1, \ldots, e_{k-1}$ are in both T_0 and T^*, but $e_k \notin T_0$. Then $T_0 + e_k$ contains a unique cycle C. Since not every edge of C can be in T^*, C must contain an edge e'_k not belonging to T^*. Let $T'_0 = T_0 + e_k - e'_k$. Then T'_0 is another spanning tree of G. Moreover,

$$w(T'_0) = w(T_0) + w(e_k) - w(e'_k). \tag{4.1}$$

Now, in Kruskal's algorithm, e_k was chosen as an edge with the smaller weight such that $G[\{e_1, \ldots, e_{k-1}, e_k\}]$ was acyclic. Since $G[\{e_1, \ldots, e_{k-1}, e'_k\}]$ is a subgraph of the tree T_0, it is also acyclic. Hence,

$$w(e_k) \leq w(e'_k), \tag{4.2}$$

Table 4.1 Mileage between Indian cities

	Mumbai	Hyderabad	Nagpur	Calcutta	New Delhi	Chennai
Mumbai (M)						
Hyderabad (H)	385					
Nagpur (N)	425	255				
Calcutta (Ca)	1035	740	679			
New Delhi (D)	708	773	531	816		
Chennai (Ch)	644	329		860	1095	

and therefore from (4.1) and (4.2),

$$\begin{aligned} w(T_0') &= w(T_0) + w(e_k) - w(e_k') \\ &\leq w(T_0). \end{aligned}$$

But T_0 is of minimum weight. Hence, $w(T_0') = w(T_0)$, and so T_0' is also of minimum weight. However, as $\{e_1, \ldots, e_k\} \subset E(T_0')$,

$$f(T_0') > k = f(T_0),$$

contradicting the choice of T_0. Thus, T^* is a minimum-weight spanning tree of G. □

When the graph is not weighted, we can give the weight 1 to each of its edges and then apply the algorithm. The algorithm then gives an acyclic subgraph with as many edges as possible, that is, a spanning tree of G.

Illustration The distances in miles between some of the Indian cities connected by air are given in Table 4.1.

Determine a minimum-cost operational system so that every city is connected to every other city. Assume that the cost of operation is directly proportional to the distance.

Let G be a graph with the set of cities as its vertex set. An edge corresponds to a pair of cities for which the ticketed mileage is indicated. The ticketed mileage is the weight of the corresponding edge (see Fig. 4.8).

The required operation system demands a minimum-cost spanning tree of G. We shall apply Kruskal's algorithm and determine such a system. The following is a sequence of edges selected according to the algorithm.

$$HN,\ HCh,\ HM,\ ND,\ NCa.$$

The corresponding spanning tree is shown in bold lines, and its weight is $255 + 329 + 385 + 531 + 679 = 2,179$.

We next describe Prim's algorithm [159].

$HN, HCh, HM, ND, NCa.$

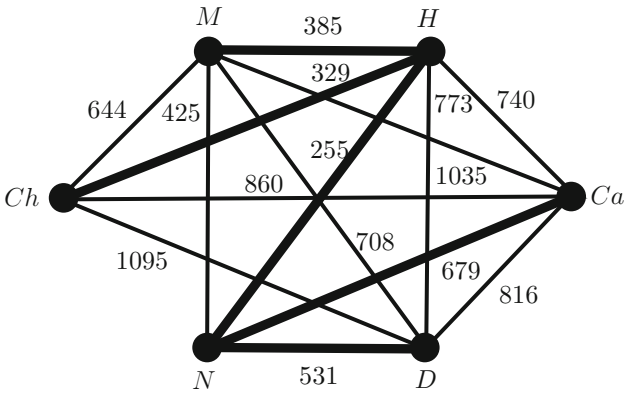


Fig. 4.8 Graph of mileage between cities

4.7.3 Prim's Algorithm

Let G be a simple connected weighted graph having n vertices. Let the vertices of G be labeled as $v_1, v_2, \ldots, v_n$. Let $W = W(G) = (w_{ij})$ be the weight matrix of G. That is, W is the $n \times n$ matrix with

(i) $w_{ii} = \infty$, for $1 \leq i \leq n$,
(ii) $w_{ij} = w_{ji} =$ the weight of the edge (v_i, v_j) if v_i and v_j are adjacent,
(iii) $w_{ij} = w_{ji} = \infty$ if v_i and v_j are nonadjacent.

The algorithm constructs a minimum-cost spanning tree.

Step 1: Start with v_1. Connect v_1 to v_k, where v_k is a nearest vertex to v_1 (v_k is nearest to v_1 if v_1v_k is an edge with minimum possible weight). The vertex v_k could be easily determined by observing the matrix W. Actually, v_k is a vertex corresponding to which the entry in row 1 of W is minimum.

Step 2: Having chosen v_k, let $v_i \neq v_1$ or v_k be a vertex corresponding to the smallest entry in rows 1 and k put together. Then v_i is the vertex "nearest" the edge subgraph defined by the edge v_1v_k. Connect v_i to v_1 or v_k, according to whether the entry is in the first row or kth row. Suppose it is, say, in the kth row; then it is the (k, i)th entry of W.

Step 3: Consider the edge subgraph defined by the edge set $\{v_1v_k, v_kv_i\}$. Determine the nearest neighbor to the set of vertices $\{v_1, v_k, v_i\}$.

Step 4: Continue the process until all the n vertices have been connected by $(n-1)$ edges. This results in a minimum-cost spanning tree.

Proof of correctness: Let T be a tree obtained by applying Prim's algorithm. We want to show that T is a minimum-weight spanning tree (that is, an optimal tree)

of G. We prove by induction on $n = |V(G)|$. Suppose $e = v_1v_2$ is an edge of least weight incident at v_1.

Claim. There exists a minimum-weight spanning tree of G that contains e. To see this, consider an optimal tree T' of G. Suppose T' does not contain e. As T' contains the vertex v_1, T' must contain some edge f of G incident at v_1. By Prim's algorithm, $w(e) \leq w(e')$ for every edge e' of G incident at v_1 and consequently, $w(e) \leq w(f)$, where w denotes the weight function. Hence, the spanning tree $T'' = T'+e-f$ of G has the property that $w(T'') = w(T')+w(e)-w(f) \leq w(T')$.

As T' is optimal, T'' is also optimal. But T'' contains e. This establishes our claim.

Let $G' = G \circ e$, the contraction of G obtained by contracting the edge e. Every spanning tree of G that contains e gives rise to a unique spanning tree of G'. Conversely, every spanning tree of G' gives rise to a unique spanning tree of G containing e.

Let S denote the set of vertices of the tree T_p (with $e \in E(T_p)$) grown by Prim's algorithm at the end of p steps, $p \geq 2$, and S' denote the set of vertices of $T'_p = T_p \circ e$. Then $[S, V(G)\backslash S] = [S', V(G')\backslash S']$. Therefore, an edge of minimum weight in $[S, V(G)\backslash S]$ is also an edge of minimum weight in $[S', V(G')\backslash S']$. As the final tree T is a Prim tree of G, the final tree $T \circ e$ of G' is a Prim tree of G'. But then G' has one vertex less than that of G and so $T \circ e$ is an optimal tree of $G \circ e$. Consequently, T is an optimal tree of G. □

Illustration Consider the weighted graph G shown in Fig. 4.8. The weight matrix W of G is

$$W : \begin{array}{c} \\ M \\ H \\ N \\ Ca \\ D \\ Ch \end{array} \begin{array}{c} \begin{array}{cccccc} M & H & N & Ca & D & Ch \end{array} \\ \begin{bmatrix} \infty & 385 & 425 & 1035 & 708 & 644 \\ 385 & \infty & 255 & 740 & 773 & 329 \\ 425 & 255 & \infty & 679 & 531 & \infty \\ 1035 & 740 & 679 & \infty & 816 & 860 \\ 708 & 773 & 531 & 816 & \infty & 1095 \\ 644 & 329 & \infty & 860 & 1095 & \infty \end{bmatrix} \end{array}$$

In row M (i.e., in the row corresponding to the city M, namely, Mumbai), the smallest weight is 385, which occurs in column H. Hence join M and H. Now, after omitting columns M and H, 255 is the minimum weight in the rows M and H put together. It occurs in row H and column N. Hence, join H and N. Now, omitting columns M, H, and N, the smallest number in the rows M, H, and N put together is 329, and it occurs in row H and column Ch, so join H and Ch. Again, the smallest entry in rows M, H, N and Ch not belonging to the corresponding columns is 531, and it occurs in row N and column D. So join N and D. Now, the lowest entry in rows M, H, N, D and Ch not belonging to the corresponding columns is 679, and it occurs in row N and column Ca. So join N and Ca. This construction gives the same minimum-weight spanning tree of Fig. 4.8.

Remark. In each iteration of Prim's algorithm, a subtree of a minimum-weight spanning tree is obtained, whereas in any step of Kruskal's algorithm, just a subgraph of a minimum-weight spanning tree is constructed.

4.7.4 Shortest-Path Problems

A manufacturing concern has a warehouse at location X and the market for the product at another location Y. Given the various routes of transporting the product from X to Y and the cost of operating them, what is the most economical way of transporting the materials? This problem can be tackled using graph theory. All such optimization problems come under a type of graph-theoretic problem known as "shortest-path problems." Three types of shortest-path problems are well known:

Let G be a connected weighted graph.

1. Determine a shortest path, that is, a minimum-weight path between two specified vertices of G.
2. Determine a set of shortest paths between all pairs of vertices of G.
3. Determine a set of shortest paths from a specified vertex to all other vertices of G.

We consider only the first problem. The other two problems are similar. We describe Dijkstra's algorithm [52] for determining the shortest path between two specified vertices. Once again, it is clear that in shortest-path problems, we could restrict ourselves to simple connected weighted graphs.

4.7.5 Dijkstra's Algorithm

Let G be a simple connected weighted graph having vertices $v_1, v_2, \ldots, v_n$. Let s and t be two specified vertices of G. We want to determine a shortest path from s to t. Let W be the weight matrix of G. Dijkstra's algorithm allots weights to the vertices of G. At each stage of the algorithm, some vertices have permanent weights and others have temporary weights.

To start with, the vertex s is allotted the permanent weight 0 and all other vertices the temporary weight ∞. In each iteration of the algorithm, one new vertex is allotted a permanent weight by the following rules:

Rule 1: If v_j is a vertex that has not yet been allotted a permanent weight, determine for each vertex v_i that had already been allotted a permanent weight,

$$\alpha_{ij} = \min\{\text{old weight of } v_j,\ (\text{old weight of } v_i) + w_{ij}\}.$$

Let $w_j = \text{Min}_i\, \alpha_{ij}$. Then w_j is a new temporary weight of v_j not exceeding the previous temporary weight.

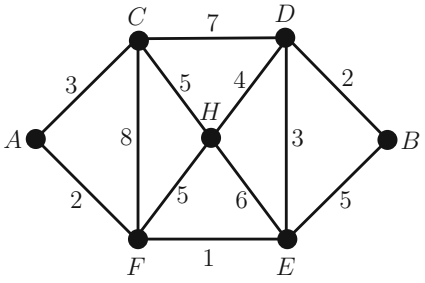


Fig. 4.9 Graph of mileage between cities

Table 4.2 Steps of algorithm for shortest path from A to B

	A	B	C	D	E	F	H
Iteration 0	[[0]]	∞	∞	∞	∞	∞	∞
Iteration 1	[0]	∞	[3]	∞	∞	[[2]]	∞
Iteration 2	[0]	∞	[[3]]	∞	[3]	[[2]]	[7]
Iteration 3	[0]	∞	[[3]]	[10]	[[3]]	[[2]]	[7]
Iteration 4	[0]	[8]	[[3]]	[[6]]	[[3]]	[[2]]	[7]
Iteration 5	[0]	[8]	[[3]]	[[6]]	[[3]]	[[2]]	[[7]]
Iteration 6	[0]	[[8]]	[[3]]	[[6]]	[[3]]	[[2]]	[[7]]

Rule 2: Determine the smallest among the w_j's. If this smallest weight is at v_k, w_k becomes the permanent weight of v_k. In case there is a tie, any one vertex is taken for allotting a permanent weight.

The algorithm stops when the vertex t gets a permanent weight.

It is clear from the algorithm that the permanent weight of each vertex is the shortest weighted distance from s to that vertex. The shortest path from s to t is constructed by working backward from the terminal vertex t. Let P be a shortest path and p_i a vertex of P. The weight of p_{i-1}, the vertex immediately preceding p_i on P, is such that the weight of the edge $p_{i-1}p_i$ equals the difference in permanent weights of p_i and p_{i-1}.

We present below an illustrative example. Let us find the shortest path from A to B in the graph of Fig. 4.9.

We present the various iterations of the algorithm by arrays of weights of the vertices, one consisting of weights before iteration and another after it. Temporary weights will be enclosed in squares and the permanent weights enclosed in double squares. The steps of the algorithm for determining the shortest path from vertex A to vertex B in graph G of Fig. 4.9 are given in Table 4.2.

In our example, B is the last vertex to get a permanent weight. Hence the algorithm stops after Iteration 6, in which B is allotted the permanent weight. However, the algorithm may be stopped as soon as vertex B gets the permanent weight.

The shortest distance from A to B is 8. A shortest path with weight 8 is $A\ F\ E\ D\ B$.

4.8 Exercises

8.1. Show that any tree of order n contains a subtree of order k for every $k \leq n$.

8.2. Let u, v, w be any three vertices of a tree T. Show that either u, v, w all lie in a path of T or else there exists a unique vertex z of T which is common to the u-v, v-w, w-u paths of T.

8.3. Show that in a tree, the number of vertices of degree at least 3 is at most the number of end vertices minus 2.

8.4. Show that if G is a connected graph with at least three vertices, then G contains two vertices u and v such that $G - \{u, v\}$ is also connected.

8.5. * If H is a graph of minimum degree at least $k-1$, then prove that H contains every tree on k vertices. (Hint: Prove by induction on k.) (See [88].)

8.6. Prove that a nontrivial simple graph G is a tree if and only if for any set of r distinct vertices in G, $r \geq 2$, the minimum number of edges required to separate them is $r - 1$. (See E. Sampathkumar [168].)

8.7. Show that a simple connected graph contains at least $m-n+1$ distinct cycles.

8.8. Prove that for a connected graph G, $r(G) \leq \text{diam}(G) \leq 2r(G)$. (The graphs of Fig. 4.3 show that the inequalities can be strict.)

8.9. Prove that a tree with at least three vertices has diameter 2 if and only if it is a star.

8.10. Determine the number of spanning trees of the two graphs in Fig. 4.10:

8.11. If T is a tree with at least two vertices, show that there exists a set of edge-disjoint paths covering all the vertices of T such that each of these paths has at least one end vertex that is an end vertex of T.

8.12. Let T be a tree of order n with $V(T) = \{1, 2, \ldots, n\}$, and let A be a set of transpositions defined by $A = \{(i, j) : ij \in E(T)\}$. Show that A is a minimal set of transpositions that generates the symmetric group S_n.

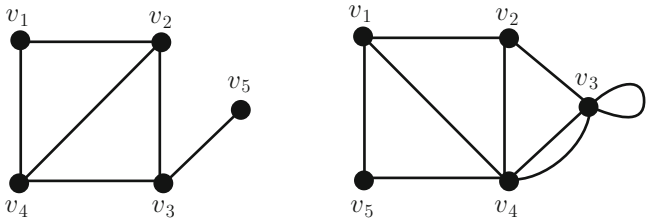
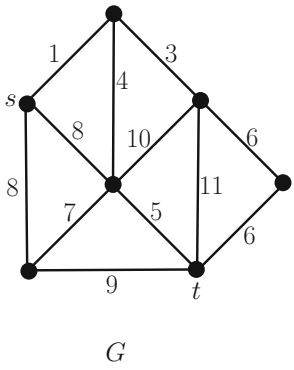


Fig. 4.10

Fig. 4.11



8.13. For the graph G of Fig. 4.11, determine two distinct minimum-weight spanning trees using

(i) Kruskal's algorithm,
(ii) Prim's algorithm.

What is the weight of such a tree? Also, determine a minimum-weight s-t path using Dijkstra's algorithm.

8.14. Apply Prim's algorithm to the illustrative example given in Sect. 4.7.3 by starting from the third row of the weight matrix.

8.15. If G is a connected weighted graph in which no two edges have the same weight, show that G has a unique minimum-weight spanning tree.

8.16. Establish the correctness of Dijkstra's algorithm.

Notes

In 1847, G. R. Kirchoff (1824–1887) developed the theory of trees for their applications in electrical networks. Ten years later, in 1857, the English mathematician A. Cayley (1821–1895) rediscovered trees while he was trying to enumerate the isomers of the saturated hydrocarbons C_nH_{2n+2} (see also Chap. 1). Since then "trees" have grown both vertically and horizontally. They are widely used today in computer science.

There is also a simpler (?) proof of the converse part of Theorem 4.4.4 using matroid theory (see pp. 126–127 of [191]).

Corollary 4.4.6 is due to Kilpatrick [122], but the elegant proof given here is due to Jaeger [114]. The proof of Cayley's theorem presented here (Theorem 4.5.1) is based on Moon [142], which also contains nine other proofs. The book by Serre [170] entitled *Trees* is mainly concerned with the connection between trees and the group $SL_2(Q_p)$.

For general algorithmic results related to graphs, see [3, 72, 153].