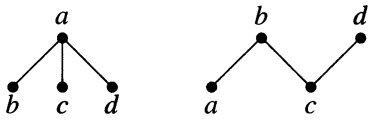
2.2. Spanning Trees and Enumeration

There are $2^{\binom{n}{2}}$ simple graphs with vertex set $[n] = \{1, \ldots, n\}$, since each pair may or may not form an edge. How many of these are trees? In this section, we solve this counting problem, count spanning trees in arbitrary graphs, and discuss several applications.

ENUMERATION OF TREES

With one or two vertices, only one tree can be formed. With three vertices there is still only one isomorphism class, but the adjacency matrix is determined by which vertex is the center. Thus there are three trees with vertex set [3]. With vertex set [4], there are four stars and 12 paths, yielding 16 trees. With vertex set [5], a careful study yields 125 trees.

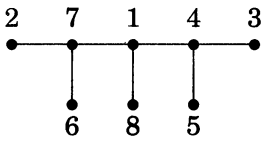


Now we may see a pattern. With vertex set $[n]$, there are n^{n-2} trees; this is **Cayley's Formula**. Prüfer, Kirchhoff, Pólya, Renyi, and others found proofs. J.W. Moon [1970] wrote a book about enumerating classes of trees. We present a bijective proof, establishing a one-to-one correspondence between the set of trees with vertex set $[n]$ and a set of known size.

Given a set S of n numbers, there are exactly n^{n-2} ways to form a list of length $n-2$ with entries in S. The set of lists is denoted S^{n-2} (see Appendix A). We use S^{n-2} to encode the trees with vertex set S. The list that results from a tree is its **Prüfer code**.

2.2.1. Algorithm. (Prüfer code) Production of $f(T) = (a_1, \ldots, a_{n-2})$.
Input: A tree T with vertex set $S \subseteq \mathbb{N}$.
Iteration: At the ith step, delete the least remaining leaf, and let a_i be the *neighbor* of this leaf. ■

2.2.2. Example. After $n-2$ iterations, only one of the original $n-1$ edges remains, and we have produced a list $f(T)$ of length $n-2$ with entries in S. In the tree below, the least leaf is 2; we delete it and record 7. After deleting 3 and 5 and recording 4 each time, the least leaf in the remaining 5-vertex tree is 4. The full code is (744171), and the vertices remaining at the end are 1 and 8. After the first step, the remainder of the Prüfer code is the Prüfer code of the subtree T' with vertex set $[8] - \{2\}$.



If we know the vertex set S, then we can retrieve the tree from the code a. The idea is to retrieve all the edges. We start with the set S of isolated vertices. At each step we create one edge and mark one vertex. When we are ready to consider a_i, there remain $n - i + 1$ unmarked vertices and $n - i - 1$ entries of a (including a_i). Thus at least two of the unmarked vertices do not appear among the remaining entries of a. Let x be the least of these, add xa_i to the list of edges, and mark x. After repeating this $n - 2$ times, two unmarked vertices remain; we join them to form the final edge.

In the example above, the least element of S not in the code is 2, so the first edge added joins 2 and 7, and we mark 2. Now the least unmarked element absent from the rest is 3, and we join it to 4, which is a_2. As we continue, we reconstruct edges in the order they were deleted to obtain a from T.

Throughout the process, each component of the graph we have grown has one unmarked vertex. This is true initially, and thus adding an edge with two unmarked endpoints combines two components. After marking one vertex of the new edge, again each component has one unmarked vertex. After $n - 2$ steps, we have two unmarked vertices and therefore two components. Adding the last edge yields a connected graph. We have built a connected graph with n vertices and $n - 1$ edges. By Theorem 2.1.4B, it is a tree, but we have not yet proved that its Prüfer code is a. ∎

2.2.3. Theorem. (Cayley's Formula [1889]). For a set $S \subseteq \mathbb{N}$ of size n, there are n^{n-2} trees with vertex set S.

Proof: (Prüfer [1918]). This holds for $n = 1$, so we assume $n \geq 2$. We prove that Algorithm 2.2.1 defines a bijection f from the set of trees with vertex set S to the set S^{n-2} of lists of length $n - 2$ from S. We must show for each $a = (a_1, \ldots, a_{n-2}) \in S^{n-2}$ that exactly one tree T with vertex set S satisfies $f(T) = a$. We prove this by induction on n.

Basis step: $n = 2$. There is tree with two vertices. The Prüfer code is a list of length 0, and it is the only such list.

Induction step: $n > 2$. Computing $f(T)$ reduces each vertex to degree 1 and then possibly deletes it. Thus every nonleaf vertex in T appears in $f(T)$. No leaf appears, because recording a leaf as a neighbor of a leaf would require reducing the tree to one vertex. Hence the leaves of T are the elements of S not in $f(T)$. If $f(T) = a$, then the first leaf deleted is the least element of S not in a (call it x), and the neighbor of x is a_1.

We are given $a \in S^{n-2}$ and seek all solutions to $f(T) = a$. We have shown that every such tree has x as its least leaf and has the edge xa_1. Deleting x leaves a tree with vertex set $S' = S - \{x\}$. Its Prüfer code is $a' = (a_2, \ldots, a_{n-2})$, an $n - 3$-tuple formed from S'.

By the induction hypothesis, there exists exactly one tree T' having vertex set S' and Prüfer code a'. Since every tree with Prüfer code a is formed by adding the edge xa_1 to such a tree, there is at most one solution to $f(T) = a$. Furthermore, adding xa_1 to T' does create a tree with vertex set S and Prüfer code a, so there is at least one solution. ∎

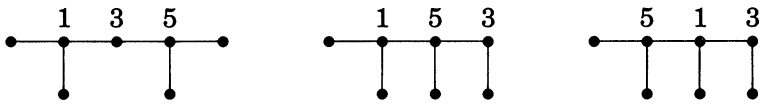
Cayley approached the problem algebraically and counted the trees by their vertex degrees. Prüfer's bijection also provides this information.

2.2.4. Corollary. Given positive integers $d_1, \ldots, d_n$ summing to $2n - 2$, there are exactly $\frac{(n-2)!}{\prod(d_i-1)!}$ trees with vertex set $[n]$ such that vertex i has degree d_i, for each i.

Proof: While constructing the Prüfer code of a tree T, we record x each time we delete a neighbor of x, until we delete x itself or leave x among the last two vertices. Thus each vertex x appears $d_T(x) - 1$ times in the Prüfer code.

Therefore, we count trees with these vertex degrees by counting lists of length $n - 2$ that for each i have $d_i - 1$ copies of i. If we assign subscripts to the copies of each i to distinguish them, then we are permuting $n - 2$ distinct objects and there are $(n - 2)!$ lists. Since the copies of i are not distinguishable, we have counted each desired arrangement $\prod(d_i - 1)!$ times, once for each way to order the subscripts on each type of label. (Appendix A discusses further aspects of this counting problem.) ∎

2.2.5. Example. *Trees with fixed degrees.* Consider trees with vertices $\{1, 2, 3, 4, 5, 6, 7\}$ that have degrees $(3, 1, 2, 1, 3, 1, 1)$, respectively. We compute $\frac{(n-2)!}{\prod(d_i-1)!} = 30$; the trees are suggested below. Only the vertices $\{1, 3, 5\}$ are non-leaves. Deleting the leaves yields a subtree on $\{1, 3, 5\}$. There are three such subtrees, determined by which of the three is in the middle.



To complete each tree, we add the appropriate number of leaf neighbors for each non-leaf to give it the desired degree. There are six ways to complete the first tree (pick from the remaining four vertices the two adjacent to vertex 1) and twelve ways to complete each of the others (pick the neighbor of vertex 3 from the remaining four, and then pick the neighbor of the central vertex from the remaining three). ∎

SPANNING TREES IN GRAPHS

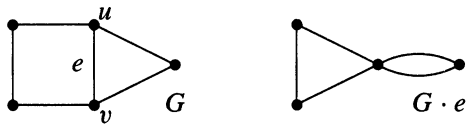
We can interpret Cayley's Formula in another way. Since the complete graph with vertex set $[n]$ has all edges that can be used in forming trees with vertex set $[n]$, the number of trees with a specified vertex set of size n equals the number of spanning trees in a complete graph on n vertices.

We now consider the more general problem of computing the number of spanning trees in any graph G. In general, G will not have as much symmetry as a complete graph, so it is unreasonable to expect as simple a formula as for K_n, but we can hope for an algorithm that provides a simple way to compute the answer when given a graph G.

2.2.6. Example. Below is the kite. To count the spanning trees, observe that four are paths around the outside cycle in the drawing. The remaining spanning trees use the diagonal edge. Since we must include an edge to each vertex of degree 2, we obtain four more spanning trees. The total is eight. ■

In Example 2.2.6, we counted separately the trees that did or did not contain the diagonal edge. This suggests a recursive procedure to count spanning trees. It is clear that the spanning trees of G not containing e are simply the spanning trees of $G - e$, but how do we count the trees that contain e? The answer uses an elementary operation on graphs.

2.2.7. Definition. In a graph G, **contraction** of edge e with endpoints u, v is the replacement of u and v with a single vertex whose incident edges are the edges other than e that were incident to u or v. The resulting graph $G \cdot e$ has one less edge than G.



In a drawing of G, contraction of e shrinks the edge to a single point. Contracting an edge can produce multiple edges or loops. To count spanning trees correctly, we must keep multiple edges (see Example 2.2.9). In other applications of contraction, the multiple edges may be irrelevant.

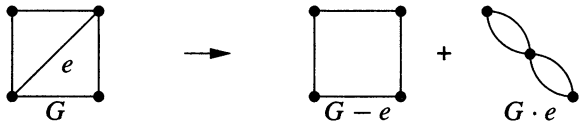
The recurrence applies for all graphs.

2.2.8. Proposition. Let $\tau(G)$ denote the number of spanning trees of a graph G. If $e \in E(G)$ is not a loop, then $\tau(G) = \tau(G - e) + \tau(G \cdot e)$.

Proof: The spanning trees of G that omit e are precisely the spanning trees of $G - e$. To show that G has $\tau(G \cdot e)$ spanning trees containing e, we show that contraction of e defines a bijection from the set of spanning trees of G containing e to the set of spanning trees of $G \cdot e$.

When we contract e in a spanning tree that contains e, we obtain a spanning tree of $G \cdot e$, because the resulting subgraph of $G \cdot e$ is spanning and connected and has the right number of edges. The other edges maintain their identity under contraction, so no two trees are mapped to the same spanning tree of $G \cdot e$ by this operation. Also, each spanning tree of $G \cdot e$ arises in this way, since expanding the new vertex back into e yields a spanning tree of G. Since each spanning tree of $G \cdot e$ arises exactly once, the function is a bijection. ∎

2.2.9. Example. *A step in the recurrence.* The graphs on the right each have four spanning trees, so Proposition 2.2.8 implies that the kite has eight spanning trees. Without the multiple edges, the computation would fail. ∎



We can save some computation time by recognizing special graphs G where we know $\tau(G)$, such as the graph on the right above.

2.2.10. Remark. If G is a connected loopless graph with no cycle of length at least 3, then $\tau(G)$ is the product of the edge multiplicities. A disconnected graph has no spanning trees. ∎

We cannot apply the recurrence of Proposition 2.2.8 when e is a loop. For example, a graph consisting of one vertex and one loop has one spanning tree, but deleting and contracting the loop would count it twice. Since loops do not affect the number of spanning trees, we can delete loops as they arise.

Counting trees recursively requires initial conditions for graphs in which all edges are loops. Such a graph has one spanning tree if it has only one vertex, and it has no spanning trees if it has more than one vertex. If a computer completes the computation by deleting or contracting every edge in a loopless graph G, then it may compute as many as $2^{e(G)}$ terms. Even with savings from Remark 2.2.10, the amount of computation grows exponentially with the size of the graph; this is impractical.

Another technique leads to a much faster computation. The Matrix Tree Theorem, implicit in the work of Kirchhoff [1847], computes $\tau(G)$ using a determinant. This is much faster, because determinants of n-by-n matrices can be computed using fewer than n^3 operations. Also, Cayley's Formula follows from the Matrix Tree Theorem with $G = K_n$ (Exercise 17), but it does not follow easily from Proposition 2.2.8.

Before stating the theorem, we illustrate the computation it specifies.

2.2.11. Example. *A Matrix Tree computation.* Theorem 2.2.12 instructs us to form a matrix by putting the vertex degrees on the diagonal and subtracting

the adjacency matrix. We then delete a row and a column and take the determinant. When G is the kite of Example 2.2.9, the vertex degrees are 3, 3, 2, 2. We form the matrix on the left below and take the determinant of the matrix in the middle. The result is the number of spanning trees! ∎

$$\begin{pmatrix} 3 & -1 & -1 & -1 \\ -1 & 3 & -1 & -1 \\ -1 & -1 & 2 & 0 \\ -1 & -1 & 0 & 2 \end{pmatrix} \rightarrow \begin{pmatrix} 3 & -1 & -1 \\ -1 & 2 & 0 \\ -1 & 0 & 2 \end{pmatrix} \rightarrow 8$$

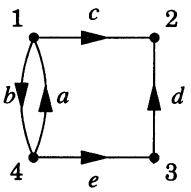
Loops don't affect spanning trees, so we delete them before the computation. The proof of the theorem uses properties of determinants.

2.2.12. Theorem. (Matrix Tree Theorem) Given a loopless graph G with vertex set $v_1, \ldots, v_n$, let $a_{i,j}$ be the number of edges with endpoints v_i and v_j. Let Q be the matrix in which entry (i, j) is $-a_{i,j}$ when $i \neq j$ and is $d(v_i)$ when $i = j$. If Q^* is a matrix obtained by deleting row s and column t of Q, then $\tau(G) = (-1)^{s+t} \det Q^*$.

Proof*: We prove this only when $s = t$; the general statement follows from a result in linear algebra (when the columns of a matrix sum to the vector 0, the cofactors are constant in each row—Exercise 8.6.18).

Step 1. If D is an orientation of G, and M is the incidence matrix of D, then $Q = MM^T$. With edges $e_1, \ldots, e_m$, the entries of M are $m_{i,j} = 1$ when v_i is the tail of e_j, $m_{i,j} = -1$ when v_i is the head of e_j, and $m_{i,j} = 0$ otherwise. Entry i, j in MM^T is the dot product of rows i and j of M. When $i \neq j$, the product counts -1 for every edge of G joining the two vertices; when $i = j$, it counts 1 for every incident edge and yields the degree.

$$M = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{pmatrix} -1 & 1 & 1 & 0 & 0 \\ 0 & 0 & -1 & -1 & 0 \\ 0 & 0 & 0 & 1 & -1 \\ 1 & -1 & 0 & 0 & 1 \end{pmatrix} \end{array}$$



$$Q = \begin{pmatrix} 3 & -1 & 0 & -2 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ -2 & 0 & -1 & 3 \end{pmatrix}$$

Step 2. If B is an $(n-1)$-by-$(n-1)$ submatrix of M, then $\det B = \pm 1$ *if the corresponding $n-1$ edges form a spanning tree of G, and otherwise* $\det B = 0$. In the first case, we use induction on n to prove that $\det B = \pm 1$. For $n = 1$, by convention a 0×0 matrix has determinant 1. For $n > 1$, let T be the spanning tree whose edges are the columns of B. Since T has at least two leaves and only one row is deleted, B has a row corresponding to a leaf x of T. This row has only one nonzero entry in B. When computing the determinant by expanding along this row, the only submatrix B' with nonzero weight in the expansion corresponds to the spanning subtree of $G - x$ obtained by deleting x and its incident edge from T. Since B' is an $(n-2)$-by-$(n-2)$ submatrix of the incidence matrix for an orientation of $G - x$, the induction hypothesis yields $\det B' = \pm 1$. Since the nonzero entry in row x is ± 1, we obtain the same result for B.

If the $n-1$ edges corresponding to columns of B do not form a spanning tree, then by Theorem 2.1.4C they contain a cycle C. We form a linear combination of the columns with coefficient 0 if the corresponding edge is not in C, $+1$ if it is followed forward by C, and -1 if it is followed backward by C. The result is total weight 0 at each vertex, so the columns are linearly dependent, which yields $\det B = 0$.

Step 3. Computation of $\det Q^*$. Let M^* be the result of deleting row t of M, so $Q^* = M^*(M^*)^T$. If $m < n-1$, then the determinant is 0 and there are no spanning subtrees, so we assume that $m \geq n-1$. The Binet–Cauchy Formula (Exercise 8.6.19) computes the determinant of a product of non-square matrices using the determinants of square submatrices of the factors. When $m \geq p$, A is p-by-m, and B is m-by-p, it states that $\det AB = \sum_S \det A_S \det B_S$, where the summ runs over all p-sets S in $[m]$, A_S is the submatrix of A consisting of the columns indexed by S, and B_S is the submatrix of B consisting of the rows indexed by S. When we apply the formula to $Q^* = M^*(M^*)^T$, the submatrix A_S is an $(n-1)$-by-$(n-1)$ submatrix of M as discussed in Step 2, and $B_S = A_S^T$. Hence the summation counts $1 = (\pm 1)^2$ for each set of $n-1$ edges corresponding to a spanning tree and 0 for all other sets of $n-1$ edges. ∎

DECOMPOSITION AND GRACEFUL LABELINGS

We consider another problem about graph decomposition (Definition 1.1.32). We can always decompose G into single edges; can we decompose G into copies of a larger tree T? This requires that $e(T)$ divides $e(G)$ and $\Delta(G) \geq \Delta(T)$; is that sufficient? Even when G is $e(T)$-regular, this may fail (Exercise 20); for example, the Petersen graph does not decompose into claws.

Häggkvist conjectured that if G is a $2m$-regular graph and T is a tree with m edges, then $E(G)$ decomposes into $n(G)$ copies of T. Even the "simplest" case when G is a clique is still unsettled and notorious.

2.2.13. Conjecture. (Ringel [1964]) If T is a fixed tree with m edges, then K_{2m+1} decomposes into $2m+1$ copies of T. ∎

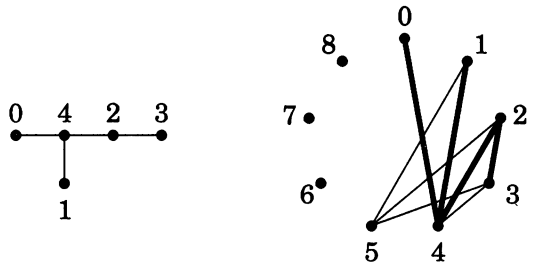
Attempts to prove Ringel's conjecture have focused on the stronger **Graceful Tree Conjecture**. This implies Ringel's conjecture and a similar statement about decomposing complete graphs of even order (Exercise 23).

2.2.14. Definition. A **graceful labeling** of a graph G with m edges is a function $f\colon V(G) \to \{0, \ldots, m\}$ such that distinct vertices receive distinct numbers and $\{|f(u) - f(v)| : uv \in E(G)\} = \{1, \ldots, m\}$. A graph is **graceful** if it has a graceful labeling.

2.2.15. Conjecture. (Graceful Tree Conjecture—Kotzig, Ringel [1964]) Every tree has a graceful labeling. ∎

2.2.16. Theorem. (Rosa [1967]) If a tree T with m edges has a graceful labeling, then K_{2m+1} has a decomposition into $2m+1$ copies of T.

Proof: View the vertices of K_{2m+1} as the congruence classes modulo $2m+1$, arranged circularly. The *difference* between two congruence classes is 1 if they are consecutive, 2 if one class is between them, and so on up to difference m. We group the edges of K_{2m+1} by the difference between the endpoints. For $1 \le j \le m$, there are $2m+1$ edges with difference j.

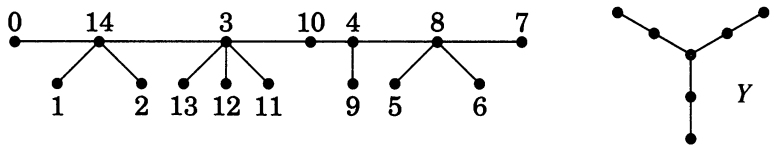


From a graceful labeling of T, we define copies of T in K_{2m+1}; the copies are $T_0, \ldots, T_{2m}$. The vertices of T_k are $k, \ldots, k+m \pmod{2m+1}$, with $k+i$ adjacent to $k+j$ if and only if i is adjacent to j in the graceful labeling of T. The copy T_0 looks just like the graceful labeling and has one edge with each difference. Moving to the next copy shifts each edge to another having the same difference by adding one to the name of each endpoint. Each difference class of edges has one edge in each T_k, and thus $T_0, \ldots, T_{2m}$ decompose K_{2m+1}. ∎

Graceful labelings are known to exist for some types of trees and for some other families of graphs (see Gallian [1998]). It is easy to find graceful labelings for stars and paths. We next define a family of trees that generalizes both by permitting the addition of edges incident to a path.

2.2.17. Definition. A **caterpillar** is a tree in which a single path (the **spine**) is incident to (or contains) every edge.

2.2.18. Example. The vertices not on the spine of a caterpillar (the "feet") are leaves. Below we show a graceful labeling of a caterpillar; in fact, every caterpillar is graceful (Exercise 31). The tree Y below is not a caterpillar. ∎



2.2.19. Theorem. A tree is a caterpillar if and only if it does not contain the tree Y above.

Proof: Let G' denote the tree obtained from a tree G by deleting each leaf of G. Since all vertices that survive in G' are non-leaves in G, G' has a vertex of degree at least 3 if and only if Y appears in G. Hence G has no copy of Y if and only if $\Delta(G') \leq 2$. This is equivalent to G' being a path, which is equivalent to G being a caterpillar. ∎

BRANCHINGS AND EULERIAN DIGRAPHS (optional)

Tutte extended the Matrix Tree Theorem to digraphs. His theorem reduces to the Matrix Tree Theorem when the digraph is symmetric (a digraph is *symmetric* if its adjacency matrix is symmetric, and then it models a graph). There is a surprising connection between this theorem and Eulerian circuits.

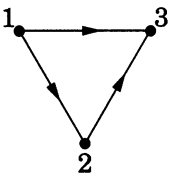
2.2.20. Definition. A **branching** or **out-tree** is an orientation of a tree having a root of indegree 0 and all other vertices of indegree 1. An **in-tree** is an out-tree with edges reversed.

A branching with root v is a union of paths from v (Exercise 33). Each vertex is reached by exactly one path. The analogous result holds for in-trees; an in-tree is a union of paths to the root, one from each vertex.

We state without proof Tutte's theorem to count branchings.

2.2.21. Theorem. (Directed Matrix Tree Theorem—Tutte [1948]) Given a loopless digraph G, let $Q^- = D^- - A'$ and $Q^+ = D^+ - A'$, where D^- and D^+ are the diagonal matrices of indegrees and outdegrees in G, and the i, jth-entry of A' is the number of edges from v_j to v_i. The number of spanning out-trees (in-trees) of G rooted at v_i is the value of each cofactor in the ith row of Q^- (ith column of Q^+). ∎

$$Q^- = \begin{pmatrix} 0 & 0 & 0 \\ -1 & 1 & 0 \\ -1 & -1 & 2 \end{pmatrix}$$



$$Q^+ = \begin{pmatrix} 2 & 0 & 0 \\ -1 & 1 & 0 \\ -1 & -1 & 0 \end{pmatrix}$$

2.2.22. Example. The digraph above has two spanning out-trees rooted at 1 and two spanning in-trees rooted at 3. Every cofactor in the first row of Q^- is 2, and every cofactor in the third column of Q^+ is 2. ∎

Isolated vertices don't affect Eulerian circuits. After discarding these, a digraph is Eulerian if and only if indegree equals outdegree at every vertex and the underlying graph is connected (Theorem 1.4.24). Such a digraph also is strongly connected, which allows us to find a spanning in-tree. We will describe Eulerian circuits in terms of a spanning in-tree.

2.2.23. Lemma. In a strong digraph, every vertex is the root of an out-tree (and an in-tree).

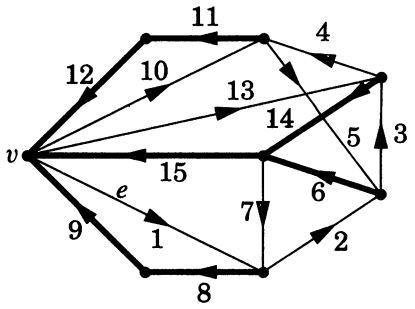
Proof: Consider a vertex v. We iteratively add edges to grow a branching from v. Let S_i be the set of vertices reached when i edges have been added; initialize $S_0 = \{v\}$. Because the digraph is strong, there is an edge leaving S_i (Exercise 1.4.10). We add one such edge to the branching and add its head to S_i to obtain S_{i+1}. This repeats until we have reached all vertices.

To obtain an in-tree of paths to v, reverse all edges and apply the same procedure; the reverse of a strong digraph is also strong. ∎

The lemma constructively produces a *search tree* of paths from a root. The next section discusses search trees in more generality.

2.2.24. Algorithm. (Eulerian circuit in directed graph)
Input: An Eulerian digraph G without isolated vertices and a spanning in-tree T consisting of paths to a vertex v.
Step 1: For each $u \in V(G)$, specify an ordering of the edges that leave u, such that for $u \neq v$ the edge leaving u in T comes last.
Step 2: Beginning at v, construct an Eulerian circuit by always exiting the current vertex u along the next unused edge in the ordering specified at u. ∎

2.2.25. Example. In the digraph below, the bold edges form an in-tree T of paths to v. The edges labeled in order starting with 1 form an Eulerian circuit. It leaves a vertex along an edge of T only where there is no alternative. If the ordering at v places 1 before 10 before 13, then the algorithm traverses the edges in the order indicated. ∎



2.2.26. Theorem. Algorithm 2.2.24 always produces an Eulerian circuit.

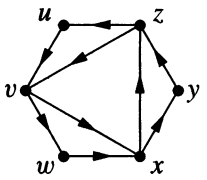
Proof: Using Lemma 2.2.23, we construct an in-tree T to a vertex v. We then apply Algorithm 2.2.24 to construct a trail. It suffices to show that the trail can end only at v and does so only after traversing all edges.

When we enter a vertex $u \neq v$, the edge leaving u in T has not yet been used, since $d^+(u) = d^-(u)$. Thus whenever we enter u there is still a way out. Therefore the trail can only end at v.

We end when we cannot continue; we are at v and have used all exiting edges. Since $d^-(v) = d^+(v)$, we must also have used all edges entering v. Since

we cannot use an edge of T until it is the only remaining edge leaving its tail, we cannot use all edges entering v until we have finished all the other vertices, since T contains a path from each vertex to v. ■

2.2.27. Example. In the digraph below, every in-tree to v contains all of uv, yz, wx, exactly one of $\{zu, zv\}$, and exactly one of $\{xy, xz\}$. There are four in-trees to v. For each in-tree, we consider $\prod(d_i - 1)! = (0!)^3(1!)^3 = 1$ orderings of the edges leaving the vertices. Hence we can obtain one Eulerian circuit from each in-tree, starting along the edge $e = vw$ from v. The four in-trees and the corresponding circuits appear below. ■



In–tree has	Circuit
zu & xy	$(v, w, x, z, v, x, y, z, u)$
zu & xz	$(v, w, x, y, z, v, x, z, u)$
zv & xy	$(v, w, x, z, u, v, x, y, z)$
zv & xz	$(v, w, x, y, z, u, v, x, z)$

Two Eulerian circuits are the same if the successive pairs of edges are the same. From each in-tree to v, Algorithm 2.2.24 generates $\prod_{u \in V(G)}(d^+(u) - 1)!$ different Eulerian circuits. The last out-edge is fixed by the tree for vertices other than v, and since we consider only the cyclic order of the edges we may also choose a particular edge e to start the ordering of edges leaving v. Any change in the exit orderings at vertices specifies at some point different choices for the next edge, so the circuits are distinct. Similarly, circuits obtained from distinct in-trees are distinct. Hence we have generated $c\prod_{u \in V(G)}(d^+(u) - 1)!$ distinct Eulerian circuits, where c is the number of in-trees to v.

In fact, these are all the Eulerian circuits. This yields a combinatorial proof that the number of in-trees to each vertex of an Eulerian digraph is the same. The graph obtained by reversing all the edges has the same number of Eulerian circuits, so the number of out-trees from any vertex also has this value, c. Theorem 2.2.21 provides a computation of c.

2.2.28. Theorem. (van Aardenne-Ehrenfest and de Bruijn [1951]). In an Eulerian digraph with $d_i = d^+(v_i) = d^-(v_i)$ the number of Eulerian circuits is $c\prod_i(d_i - 1)!$, where c counts the in-trees to or out-trees from any vertex.

Proof: We have argued that Algorithm 2.2.24 generates this many distinct Eulerian circuits using in-trees to vertex v (starting from v alond e). We need only show that this produces all Eulerian circuits.

To find the tree and ordering that generates an Eulerian circuit C, follow C from e, and record the order of the edges leaving each vertex. Let T be the subdigraph consisting of the last edge on C leaving each vertex other than v. Since the last edge leaving a vertex occurs in C after all edges entering it, each edge in T extends to a path in T that reaches v. With $n - 1$ edges, T thus forms an in-tree to v. Furthermore, C is the circuit obtained by Algorithm 2.2.24 from T and the orderings of exiting edges that we recorded. ■

EXERCISES

2.2.1. (−) Determine which trees have Prüfer codes that (a) contain only one value, (b) contain exactly two values, or (c) have distinct values in all positions.

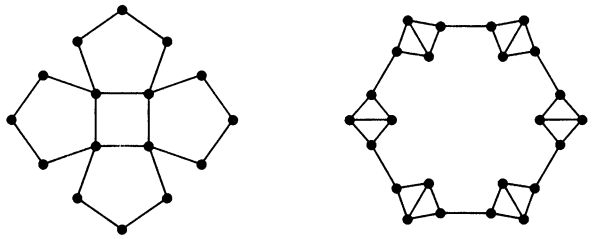
2.2.2. (−) Count the spanning trees in the graph on the left below. (Proposition 2.2.8 provides a systematic approach, and then Remark 2.2.10 and Example 2.2.6 can be used to shorten the computation.)

2.2.3. (−) Let G be the graph on the right above. Use the Matrix Tree Theorem to find a matrix whose determinant is $\tau(G)$. Compute $\tau(G)$.

2.2.4. (−) Let G be a simple graph with m edges. Prove that if G has a graceful labeling, then K_{2m+1} decomposes into copies of G. (Hint: Follow the proof of Theorem 2.2.16.)

• • • • •

2.2.5. The graph on the left below was the logo of the 9th Quadrennial International Conference in Graph Theory, held in Kalamazoo in 2000. Count its spanning trees.



2.2.6. (!) Let G be the 3-regular graph with $4m$ vertices formed from m pairwise disjoint kites by adding m edges to link them in a ring, as shown on the right above for $m = 6$. Prove that $\tau(G) = 2m8^m$

2.2.7. (!) Use Cayley's Formula to prove that the graph obtained from K_n by deleting an edge has $(n-2)n^{n-3}$ spanning trees.

2.2.8. Count the following sets of trees with vertex set $[n]$, giving two proofs for each: one using the Prüfer correspondence and one by direct counting arguments.

a) trees that have 2 leaves.

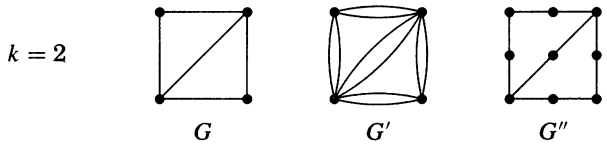
b) trees that have $n-2$ leaves.

2.2.9. Let $S(m,r)$ denote the number of partitions of an m-element set into r nonempty subsets. In terms of these numbers, count the trees with vertex set $\{v_1,\ldots,v_n\}$ that have exactly k leaves. (Rényi [1959])

2.2.10. Compute $\tau(K_{2,m})$. Also compute the number of isomorphism classes of spanning trees of $K_{2,m}$.

2.2.11. (+) Compute $\tau(K_{3,m})$.

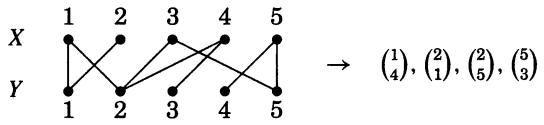
2.2.12. From a graph G we define two new graphs. Let G' be the graph obtained by replacing each edge of G with k copies of that edge. Let G'' be the graph obtained by replacing each edge $uv \in E(G)$ with a u,v-path of length k through $k-1$ new vertices. Determine $\tau(G')$ and $\tau(G'')$ in terms of $\tau(G)$ and k.



2.2.13. Consider $K_{n,n}$ with bipartition X, Y, where $X = \{x_1, \ldots, x_n\}$ and $Y = \{y_1, \ldots, y_n\}$. For each spanning tree T, we form a list $f(T)$ of ordered pairs (written vertically). Having generated part of the list, let u be the least-indexed leaf in X in the remaining subtree, and similarly let v be the least-indexed leaf in Y. Append the pair $\binom{a}{b}$ to the list, where a is the index of the neighbor of u and b is the index of the neighbor of v. Delete $\{u, v\}$. Iterate until $n-1$ pairs have been generated to form $f(T)$ (one edge remains). Part (a) shows that f is well-defined.

a) Prove that every spanning tree of $K_{n,n}$ has a leaf in each partite set.

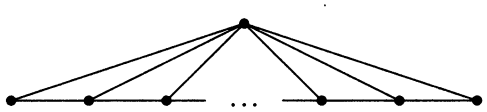
b) Prove that f is a bijection from the set of spanning trees of $K_{n,n}$ to $([n] \times [n])^{n-1}$ Thus $K_{n,n}$ has n^{2n-2} spanning trees. (Rényi [1966], Kelmans [1992], Pritikin [1995])



2.2.14. (+) Let $f(r,s)$ be the number of trees with vertex $[n]$ that have partite sets of sizes r and s (with $r+s=n$). Prove that $f(r,s) = \binom{r+s}{s} s^{r-1} r^{s-1}$ if $r \neq s$. What is the formula when $r = s$? (Hint: First show that the Prüfer sequence for such a tree will have $r-1$ of its terms from the partite set of size s and $s-1$ of its terms from the partite set of size r.) (Scoins [1962], Glicksman [1963])

2.2.15. Let G_n be the graph with $2n$ vertices and $3n-2$ edges pictured below, for $n \geq 1$. Prove for $n > 2$ that $\tau(G_n) = 4\tau(G_{n-1}) - \tau(G_{n-2})$. (Kelmans [1967a])

2.2.16. For $n \geq 1$, let a_n be the number of spanning trees in the graph formed from P_n by adding one vertex adjacent to all of $V(P_n)$. For example, $a_1 = 1$, $a_2 = 3$, and $a_3 = 8$. Prove for $n > 1$ that $a_n = a_{n-1} + 1 + \sum_{i=1}^{n-1} a_i$. Use this to prove for $n > 2$ that $a_n = 3a_{n-1} - a_{n-2}$. (Comment: It is also possible to argue directly that $a_n = 3a_{n-1} - a_{n-2}$.)



2.2.17. Use the Matrix Tree Theorem to prove Cayley's Formula.

2.2.18. Use the Matrix Tree Theorem to compute $\tau(K_{r,s})$. (Lovász [1979, p223]—see Kelmans [1965] for a generalization)

2.2.19. (+) Prove combinatorially that the number t_n of trees with vertex set $[n]$ satisfies the recurrence $t_n = \sum_{k=1}^{n-1} k\binom{n-2}{k-1} t_k t_{n-k}$. (Comment: Since $t_n = n^{n-2}$, this proves the identity $n^{n-2} = \sum_{k=1}^{n-1} \binom{n-2}{k-1} k^{k-1}(n-k)^{n-k-2}$.) (Dziobek [1917]; see Lovász [1979, p219])

2.2.20. (!) Prove that a d-regular simple graph G has a decomposition into copies of $K_{1,d}$ if and only if it is bipartite.

2.2.21. (+) Prove that $K_{2m-1,2m}$ decomposes into m spanning paths.

2.2.22. Let G be an n-vertex simple graph that decomposes into k spanning trees. Given also that $\Delta(G) = \delta(G) + 1$, determine the degree sequence of G in terms of n and k.

2.2.23. (!) Prove that if the Graceful Tree Conjecture is true and T is a tree with m edges, then K_{2m} decomposes into $2m-1$ copies of T. (Hint: Apply the cyclically invariant decomposition of K_{2m-1} for trees with $m-1$ edges from the proof of Theorem 2.2.16.)

2.2.24. Of the n^{n-2} trees with vertex set $\{0, \ldots, n-1\}$, how many are gracefully labeled by their vertex names?

2.2.25. (!) Prove that if a graph G is graceful and Eulerian, then $e(G)$ is congruent to 0 or 3 mod 4. (Hint: Sum the absolute edge differences (mod 2) in two different ways.)

2.2.26. (+) Prove that C_n is graceful if and only if 4 divides n or $n+1$. (Frucht [1979])

2.2.27. (+) Let G be the graph consisting of k 4-cycles with one common vertex. Prove that G is graceful. (Hint: Put 0 at the vertex of degree $2k$.)

2.2.28. Let $d_1, \ldots, d_n$ be positive integers. Prove directly that there exists a caterpillar with vertex degrees $d_1, \ldots, d_n$ if and only if $\sum d_i = 2n - 2$.

2.2.29. Prove that every tree can be turned into a caterpillar with the same degree sequence using 2-switches (Definition 1.3.32) such that each intermediate graph is a tree.

2.2.30. A bipartite graph is *drawn on a channel* if the vertices of one partite set are placed on one line in the plane (in some order) and the vertices of the other partite set are placed on a line parallel to it and the edges are drawn as straight-line segments between them. Prove that a connected graph G can be drawn on a channel without edge crossings if and only if G is a caterpillar.

2.2.31. (!) An *up/down labeling* is a graceful labeling for which there exists a *critical value* α such that every edge joins vertices with labels above and below α. Prove that every caterpillar has an up/down labeling. Prove that the 7-vertex tree that is not a caterpillar has no up/down-labeling.

2.2.32. (+) Prove that the number of isomorphism classes of n-vertex caterpillars is $2^{n-4} + 2^{\lfloor n/2 \rfloor - 2}$ if $n \geq 3$. (Harary–Schwenk [1973], Kimble–Schwenk [1981])

2.2.33. (!) Let T be an orientation of a tree such that the heads of the edges are all distinct; the one vertex that is not a head is the *root*. Prove that T is a union of paths from the root. Prove that for each vertex of T, exactly one path reaches it from the root.

2.2.34. (*) Use Theorem 2.2.26 to prove that the algorithm below generates a binary deBruijn cycle of length 2^n (the cycle in Application 1.4.25 arises in this way).

Start with n 0's. Subsequently, append a 1 if doing so does not repeat a previous string of length n, otherwise append a 0.