Counting Spanning Trees[*]

Bang Ye Wu Kun-Mao Chao

1 Counting Spanning Trees

This book provides a comprehensive introduction to the modern study of spanning trees. A spanning tree for a graph G is a subgraph of G that is a tree and contains all the vertices of G. There are many situations in which good spanning trees must be found. Whenever one wants to find a simple, cheap, yet efficient way to connect a set of terminals, be they computers, telephones, factories, or cities, a solution is normally one kind of spanning trees. Spanning trees prove important for several reasons:

1. They create a sparse subgraph that reflects a lot about the original graph.
2. They play an important role in designing efficient routing algorithms.
3. Some computationally hard problems, such as the Steiner tree problem and the traveling salesperson problem, can be solved approximately by using spanning trees.
4. They have wide applications in many areas, such as network design, bioinformatics, etc.

Throughout this book, we use n to denote the number of vertices of the input graph, and m the number of edges of the input graph. Let us start with the problem of counting the number of spanning trees. Let K_n denote a complete graph with n vertices. How many spanning trees are there in the complete graph K_n? Before answering this question, consider the following simpler question. How many trees are there spanning all the vertices in Figure 1?

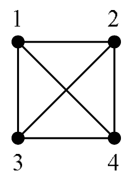

Figure 1: A four-vertex complete graph K_4.

The answer is 16. Figure 2 gives all 16 spanning trees of the four-vertex complete graph in Figure 1. Each spanning tree is associated with a two-number sequence, called a Prüfer sequence, which will be explained later.

Back in 1889, Cayley devised the well-known formula n^{n-2} for the number of spanning trees in the complete graph K_n [1]. There are numerous proofs of this elegant formula. The first explicit

[*]An excerpt from the book "Spanning Trees and Optimization Problems," by Bang Ye Wu and Kun-Mao Chao (2004), Chapman & Hall/CRC Press, USA.

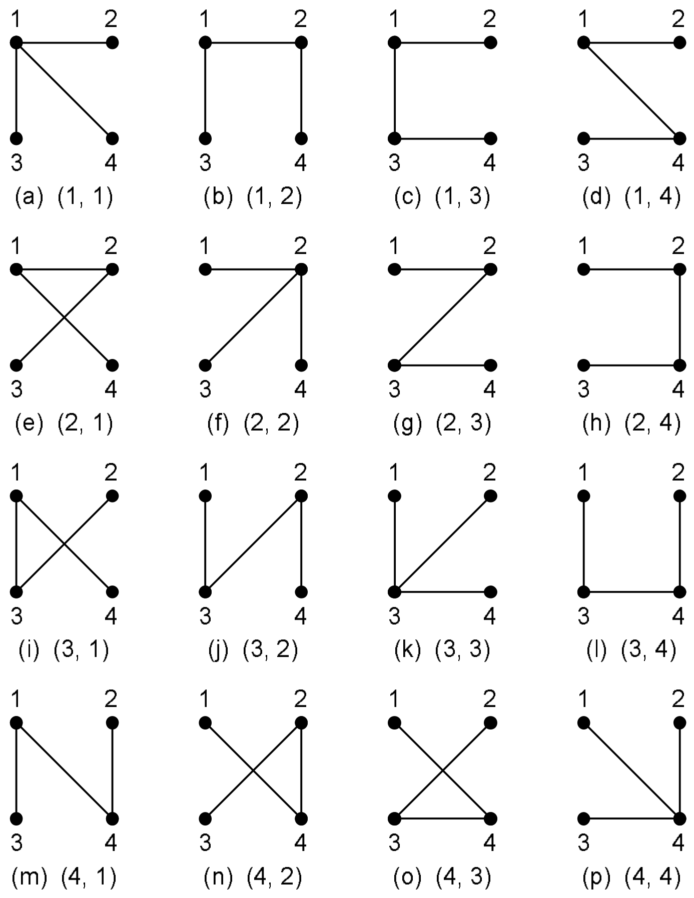

Figure 2: All 16 spanning trees of K_4.

combinatorial proof of Cayley's formula is due to Prüfer [2]. The idea of Prüfer's proof is to find a one-to-one correspondence (bijection) between the set of spanning trees of K_n, and the set of Prüfer sequences of length $n-2$, which is defined in Definition 1.

Definition 1: A Prüfer sequence of length $n-2$, for $n \geq 2$, is any sequence of integers between 1 and n, with repetitions allowed.

Lemma 1: There are n^{n-2} Prüfer sequences of length $n-2$.

Proof: By definition, there are n ways to choose each element of a Prüfer sequence of length $n-2$. Since there are $n-2$ elements to be determined, in total we have n^{n-2} ways to choose the whole sequence. □

Example 1: The set of Prüfer sequences of length 2 is $\{(1,1), (1,2), (1,3), (1,4), (2,1), (2,2), (2,3), (2,4), (3,1), (3,2), (3,3), (3,4), (4,1), (4,2), (4,3), (4,4)\}$. In total, there are $4^{4-2} = 16$ Prüfer sequences of length 2.

Given a labeled tree with vertices labeled by $1, 2, 3, \ldots, n$, the Prüfer Encoding algorithm outputs a unique Prüfer sequence of length $n-2$. It initializes with an empty sequence. If the tree has more than two vertices, the algorithm finds the leaf with the lowest label, and appends to the sequence the label of the neighbor of that leaf. Then the leaf with the lowest label is deleted from the tree. This operation is repeated $n-2$ times until only two vertices remain in the tree. The algorithm ends up deleting $n-2$ vertices. Therefore, the resulting sequence is of length $n-2$.

```
Algorithm: PRÜFER ENCODING
Input: A labeled tree with vertices labeled by $1, 2, 3, \ldots, n$.
Output: A Prüfer sequence.
    Repeat $n-2$ times
        $v \leftarrow$ the leaf with the lowest label
        Put the label of $v$'s unique neighbor in the output sequence.
        Remove $v$ from the tree.
```

Let us look at Figure 2 once again. In Figure 2(a), vertex 2 is the leaf with the lowest label, thus we add its unique neighbor vertex 1 to the sequence. After removing vertex 2 from the tree, vertex 3 becomes the leaf with the lowest label, and we again add its unique neighbor vertex 1 to the sequence. Therefore, the resulting Prüfer sequence is $(1, 1)$. In Figure 2(b), vertex 3 is the leaf with the lowest label, thus we add its unique neighbor vertex 1 to the sequence. After removing vertex 3 from the tree, vertex 1 becomes the leaf with the lowest label, and we add its unique neighbor vertex 2 to the sequence. Therefore, the resulting Prüfer sequence is $(1, 2)$.

Now consider a more complicated tree in Figure 3. What is its corresponding Prüfer sequence?

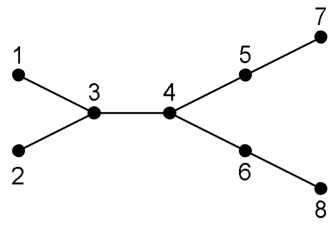

Figure 3: An eight-vertex spanning tree.

Figure 4 illustrates the encoding process step by step. In Figure 4(a), vertex 1 is the leaf with the lowest label, thus we add its unique neighbor vertex 3 to the sequence, resulting in the Prüfer sequence under construction, denoted by P, equals to (3). Then we remove vertex 1 from the tree. In Figure 4(b), vertex 2 becomes the leaf with the lowest label; we again add its unique neighbor vertex 3 to the sequence. So we have $P = (3, 3)$. After removing vertex 2 from the tree, vertex 3 is the leaf with the lowest label. Since vertex 4 is the unique label of vertex, we get $P = (3, 3, 4)$. Repeat this operation a few times until only two vertices remain in the tree. In this example, vertices 6 and 8 are the two vertices left. It follows that the Prüfer sequence for Figure 3 is $(3, 3, 4, 5, 4, 6)$.

It can be verified that different spanning trees of K_n determine different Prüfer sequences. The PRÜFER DECODING algorithm provides the inverse algorithm, which finds the unique labeled tree T with n vertices for a given Prüfer sequence of $n-2$ elements. Let the given Prüfer sequence be $P = (p_1, p_2, \ldots, p_{n-2})$. Observe that any vertex v of T occurs $\deg(v) - 1$ times in $(p_1, p_2, \ldots, p_{n-2})$, where $\deg(v)$ is the degree of vertex v. Thus the vertices of degree one, i.e., the leaves, in T are those that do not appear in P. To reconstruct T from $(p_1, p_2, \ldots, p_{n-2})$, we proceed as follows. Let V be the vertex label set $\{1, 2, \ldots, n\}$. In the i^{th} iteration of the **for** loop, $P = (p_i, p_{i+1}, \ldots, p_{n-2})$. Let v be the smallest element of the set V that does not occur in P. We connect vertex v to vertex

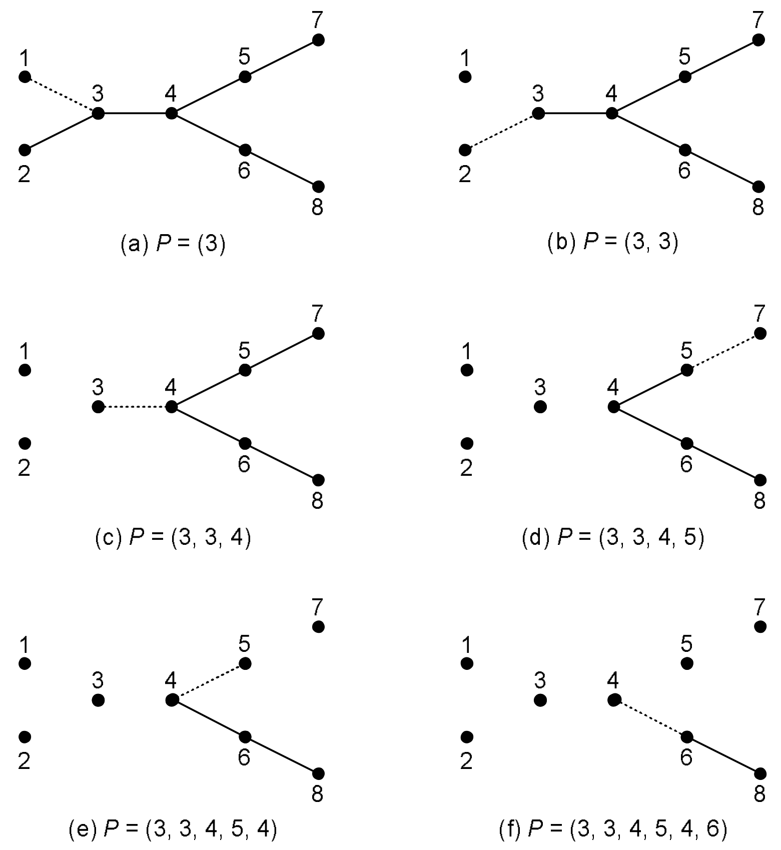

Figure 4: Generating a Prüfer sequence from a spanning tree.

p_i. Then we remove v from V, and p_i from P. Repeat this process for $n-2$ times until only two numbers are left in V. Finally, we connect the vertices corresponding to the remaining two numbers in V. It can be shown that different Prüfer sequences deliver different spanning trees of K_n.

```
**Algorithm:** PRÜFER DECODING
**Input:** A Prüfer sequence $P = (p_1, p_2, \ldots, p_{n-2})$.
**Output:** A labeled tree with vertices labeled by $1, 2, 3, \ldots, n$.
    $P \leftarrow$ the input Prüfer sequence
    $n \leftarrow |P| + 2$
    $V \leftarrow \{1, 2, \ldots, n\}$
    Start with $n$ isolated vertices labeled $1, 2, \ldots, n$.
    **for** $i = 1$ **to** $n-2$ **do**
        $v \leftarrow$ the smallest element of the set $V$ that does not occur in $P$
        Connect vertex $v$ to vertex $p_i$
        Remove $v$ from the set $V$
        Remove the element $p_i$ from the sequence $P$
        /* Now $P = (p_{i+1}, p_{i+2}, \ldots, p_{n-2})$ */
    Connect the vertices corresponding to the two numbers in $V$.
```

To see how the Prüfer Decoding algorithm works, let us build the spanning tree corresponding to the Prüfer sequence $(3, 3, 4, 5, 4, 6)$. Figure 5 illustrates the decoding process step by step. Initially, $P = (3, 3, 4, 5, 4, 6)$ and $V = \{1, 2, 3, 4, 5, 6, 7\}$. Vertex 1 is the smallest element of the set V that does not occur in P. Thus we connect vertex 1 to the first element of P, i.e., vertex 3 (see Figure 5(a)). Then we remove 3 from P, and 1 from V. Now $P = (3, 4, 5, 4, 6)$ and $V = \{2, 3, 4, 5, 6, 7, 8\}$. We connect vertex 2 to vertex 3. Repeat this operation again and again until only two numbers are left in V (see Figure 5(g)).

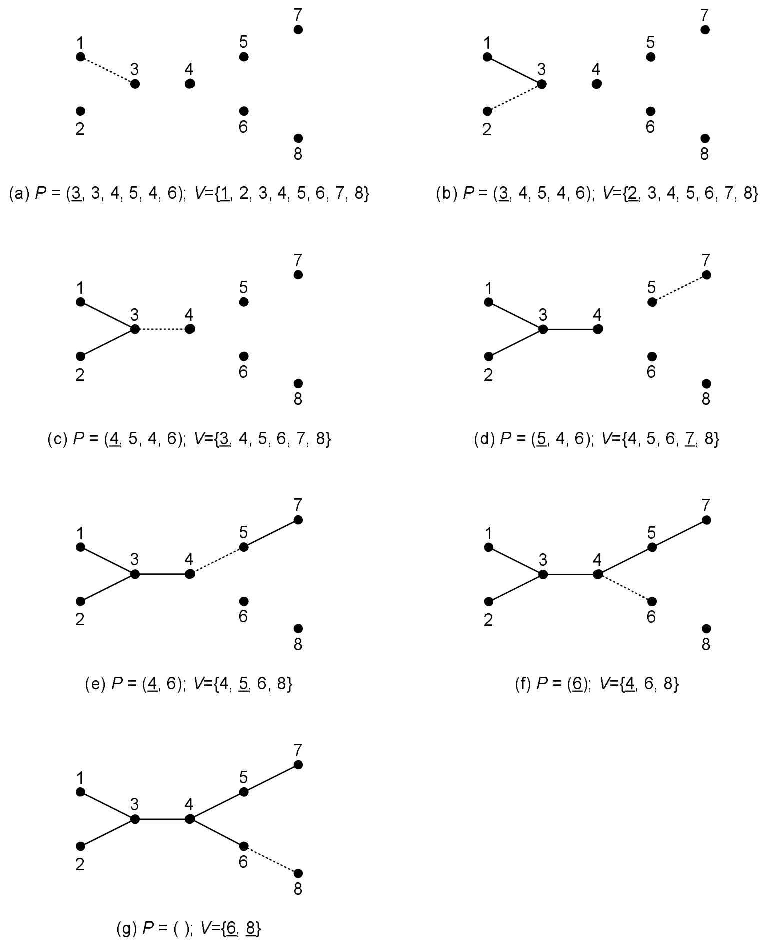
(a) $P = (\underline{3}, 3, 4, 5, 4, 6)$; $V=\{\underline{1}, 2, 3, 4, 5, 6, 7, 8\}$
(b) $P = (\underline{3}, 4, 5, 4, 6)$; $V=\{\underline{2}, 3, 4, 5, 6, 7, 8\}$
(c) $P = (\underline{4}, 5, 4, 6)$; $V=\{\underline{3}, 4, 5, 6, 7, 8\}$
(d) $P = (\underline{5}, 4, 6)$; $V=\{4, 5, 6, \underline{7}, 8\}$
(e) $P = (\underline{4}, 6)$; $V=\{4, \underline{5}, 6, 8\}$
(f) $P = (\underline{6})$; $V=\{\underline{4}, 6, 8\}$
(g) $P = (\ )$; $V=\{\underline{6}, \underline{8}\}$

Figure 5: Recovering a spanning tree from a Prüfer sequence.

We have established a one-to-one correspondence (bijection) between the set of spanning trees of K_n, and the set of Prüfer sequences of length $n - 2$. We obtain the result in Theorem 2.

Theorem 2: The number of spanning trees in K_n is n^{n-2}.

It should be noted that n^{n-2} is the number of distinct spanning trees of K_n, but not the number of nonisomorphic spanning trees of K_n. For example, there are $6^{6-2} = 1296$ distinct spanning trees of K_6, yet there are only six nonisomorphic spanning trees of K_6.

In the following, we give a recursive formula for the number of spanning trees in a general graph. Let $G - e$ denote the graph obtained by removing edge e from G. Let $G\backslash e$ denote the resulting graph after contracting e in G. In other words, $G\backslash e$ is the graph obtained by deleting e, and merging its ends. Let $\tau(G)$ denote the number of spanning trees of G. The following recursive formula computes the number of spanning trees in a graph.

Theorem 3: $\tau(G) = \tau(G - e) + \tau(G \backslash e)$

Proof: The number of spanning trees of G that do not contain e is $\tau(G - e)$ since each of them is also a spanning tree of $G - e$, and vice versa. On the other hand, the number of spanning trees of G that contain e is $\tau(G \backslash e)$ because each of them corresponds to a spanning tree of $G \backslash e$. Therefore, $\tau(G) = \tau(G - e) + \tau(G \backslash e)$. $\square$

References

[1] A. Cayley. A theorem on trees. *Quart. J. Math.*, 23:376–378, 1889.

[2] H. Prüfer. Never beweis eines satzes über permutationen. *Arch. Math. Phys. Sci.*, 27:742–744, 1918.