GRAPH THEORY – LECTURE 4: TREES

Abstract. §3.1 presents some standard characterizations and properties of trees. §3.2 presents several different types of trees. §3.7 develops a counting method based on a bijection between labeled trees and numeric strings. §3.8 showns how binary trees can be counted by the Catalan recursion.

Outline

3.1 Characterizations and Properties of Trees
3.2 Rooted Trees, Ordered Trees, and Binary Trees
3.7 Counting Labeled Trees: Prüfer Encoding
3.8 Counting Binary Trees: Catalan Recursion

1. Characterizations of Trees

Review from §1.5 *tree* = connected graph with no cycles.

Def 1.1. In an undirected tree, a ***leaf*** is a vertex of degree 1.

1.1. **Basic Properties of Trees.**

Proposition 1.1. *Every tree with at least one edge has at least two leaves.*

Proof. Let $P = \langle v_1, v_2, \ldots, v_m \rangle$ be a path of maximum length in a tree T. Etc. □

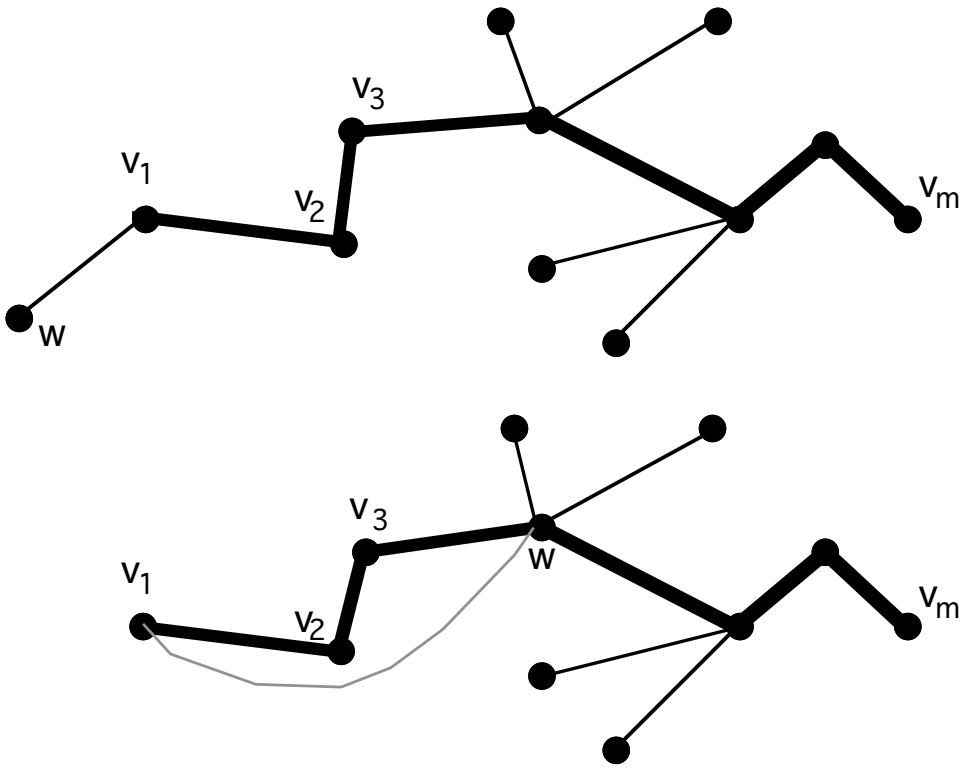


Figure 1.1: The two cases in the proof of Prop 1.1.

Corollary 1.2. *If the minimum degree of a graph is at least 2, then that graph must contain a cycle.* □

Proposition 1.3. *Every tree on n vertices has exactly $n-1$ edges.*

Proof. By induction using Prop 1.1. □

Review from §2.3 An acyclic graph is called a ***forest***.

Review from §2.4 The **number of components** of a graph G is denoted $c(G)$.

Corollary 1.4. *A forest G on n vertices has $n - c(G)$ edges.*

Proof. Apply Prop 1.3 to each of the components of G. □

Corollary 1.5. *Any graph G on n vertices has at least $n - c(G)$ edges.* □

Six Different Characterizations of a Tree

Trees have many possible characterizations, and each contributes to the structural understanding of graphs in a different way. The following theorem establishes some of the most useful characterizations.

Theorem 1.8. *Let T be a graph with n vertices. Then the following statements are equivalent.*

(1) *T is a tree.*
(2) *T contains no cycles and has $n-1$ edges.*
(3) *T is connected and has $n-1$ edges.*
(4) *T is connected, and every edge is a cut-edge.*
(5) *Any two vertices of T are connected by exactly one path.*
(6) *T contains no cycles, and for any new edge e, the graph $T+e$ has exactly one cycle.*

Proof. See text. □

The Center of a Tree

Review from §1.4 and §2.3

- The ***eccentricity*** of a vertex v in a graph G, denoted $ecc(v)$, is the distance from v to a vertex farthest from v. That is,

$$ecc(v) = \max_{x \in V_G} \{d(v, x)\}$$

- A ***central vertex*** of a graph is a vertex with minimum eccentricity.
- The ***center of a graph*** G, denoted $Z(G)$, is the subgraph induced on the set of central vertices of G.

In an arbitrary graph G, the center $Z(G)$ can be anything from a single vertex to all of G.

However, C. Jordan showed in 1869 that the center of a tree has only two possible cases. We begin with some preliminary results concerning the eccentricity of vertices in a tree.

Lemma 1.9. *Let T be a tree with at least three vertices.*

- *(a) If v is a leaf of T and w is its neighbor, then*

$$ecc(w) \;=\; ecc(v) - 1$$

- *(b) If u is a central vertex of T, then*

$$deg(u) \;\geq\; 2$$

Proof. (a) Since T has at least three vertices,

$$deg(w) \;\geq\; 2$$

Then there exists a vertex $z \neq v$ such that

$$d(w, z) \;=\; ecc(w)$$

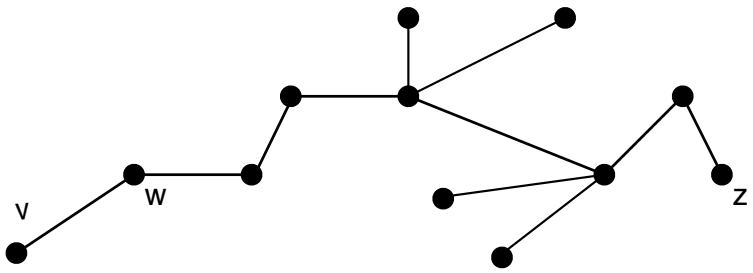


Figure 1.-2:

But z is also a vertex farthest from v, and hence

$$ecc(v) \;=\; d(v, z) \;=\; d(w, z) + 1 \;=\; ecc(w) + 1$$

(b) By Part (a), a vertex of degree 1 cannot have minimum eccentricity in tree T, and hence, cannot be a central vertex of T. □

Lemma 1.10. *Let v and w be two vertices in a tree T such that w is of maximum distance from v (i.e., $ecc(v) = d(v, w)$). Then w is a leaf.*

Proof. Let P be the unique v-w path in tree T. If $deg(w) \geq 2$, then w would have a neighbor z whose distance from v would equal $d(v, w) + 1$, contradicting the premise that w is at maximum distance. □

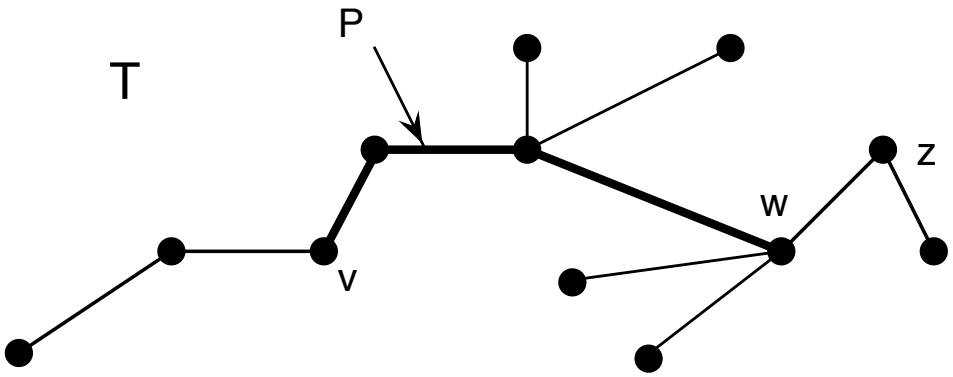


Figure 1.-3:

Lemma 1.11. *Let T be a tree with at least three vertices, and let T^* be the subtree of T obtained by deleting from T all its leaves. If v is a vertex of T^*, then*

$$ecc_T(v) = ecc_{T^*}(v) + 1$$

Proof. Let w be a vertex of T such that

$$ecc_T(v) = d(v, w)$$

By Lemma 1.10, vertex w is a leaf of tree T and hence, $w \notin V_{T^*}$ (as illustrated in Figure 1.3). It follows that the neighbor of w, say z, is a vertex of T^* that is farthest from v among all vertices in T^*, that is, $ecc_{T^*}(v) = d(v, z)$. Thus,

$$ecc_T(v) \;=\; d(v, w) \;=\; d(v, z) + 1 \;=\; ecc_{T^*}(v) + 1$$

□

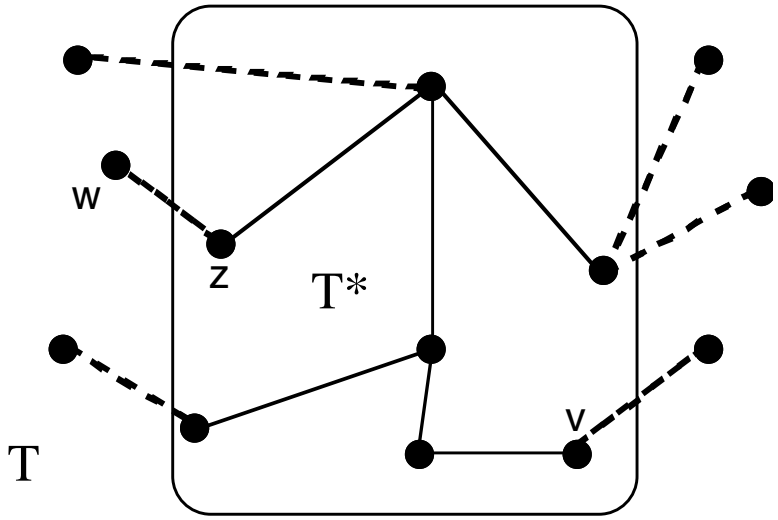


Figure 1.3:

Proposition 1.12. *Let T be a tree with at least three vertices, and let T^* be the subtree of T obtained by deleting from T all its leaves. Then*

$$Z(T) \;=\; Z(T^*)$$

Proof. From the two preceding lemmas, we see that deleting all the leaves decreases the eccentricity of every remaining vertex by 1. It follows that the resulting tree has the same center. □

Corollary 1.13 (**Jordan**, 1869)**.** *Let T be an n-vertex tree. Then the center $Z(G)$ is either a single vertex or a single edge.*

Proof. The assertion is trivially true for $n = 1$ and $n = 2$. The result follows by induction, using Proposition 1.12. □

Tree Isomorphisms and Automorphisms

Example 1.1. The two graphs in Fig 1.4 have the same degree sequence, but they can be readily seen to be non-isom in several ways. For instance, the center of the left graph is a single vertex, but the center of the right graph is a single edge. Also, the two graphs have unequal diameters.

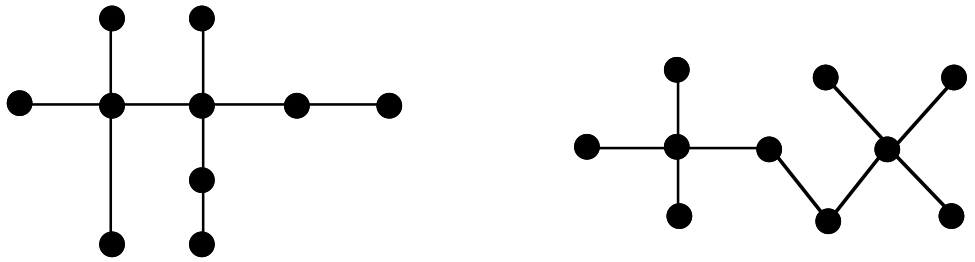


Figure 1.4: Why are these trees non-isomorphic?

Example 1.2. The graph shown in Figure 1.5 below does not have a non-trivial automorphism because the three leaves are all different distances from the center, and hence, an automorphism must map each of them to itself.

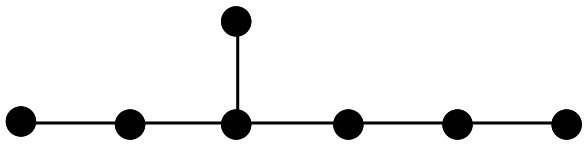


Figure 1.5: A tree that has no non-trivial automorphisms.

Remark 1.1. There is a linear-time algorithm for testing the isomorphism of two trees (see [AhHoUl74, p84]).

2. Rooted, Ordered, Binary Trees

Rooted Trees

Def 2.1. A ***directed tree*** is a directed graph whose underlying graph is a tree.

Def 2.2. A ***rooted tree*** is a tree with a designated vertex called the ***root***. Each edge is implicitly directed away from the root.

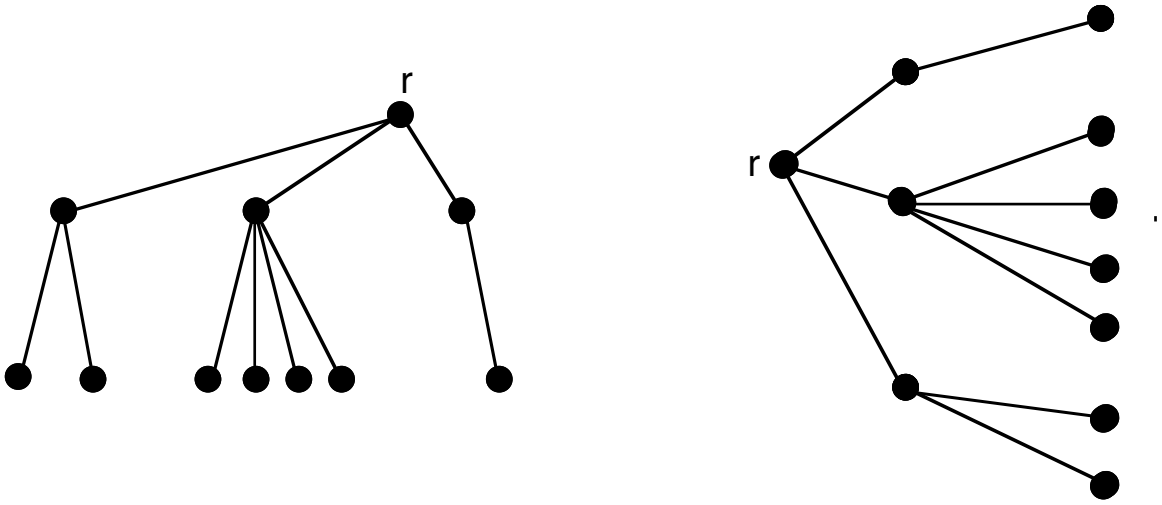


Figure 2.1: Two common ways of drawing a rooted tree.

Rooted Tree Terminology

Designating a root imposes a hierarchy on the vertices of a rooted tree, according to their distance from that root.

Def 2.3. In a rooted tree, the ***depth*** or ***level*** of a vertex v is its distance from the root, i.e., the length of the unique path from the root to v. Thus, the root has depth 0.

Def 2.4. The ***height*** of a rooted tree is the length of a longest path from the root (or the greatest depth in the tree).

Def 2.5. If vertex v immediately precedes vertex w on the path from the root to w, then v is ***parent*** of w and w is ***child*** of v.

Def 2.6. Vertices having the same parent are called ***siblings***.

Def 2.7. A vertex w is called a ***descendant*** of a vertex v (and v is called an ***ancestor*** of w), if v is on the unique path from the root to w. If, in addition, $w \neq v$, then w is a ***proper*** descendant of v (and v is a proper ancestor of w).

Def 2.8. A ***leaf*** in a rooted tree is any vertex having no children.

Def 2.9. An ***internal vertex*** in a rooted tree is any vertex that has at least one child. The root is internal, unless the tree is trivial (i.e., a single vertex).

Example 2.2. The height of this tree is 3. Also,

- r, a, b, c, and d are the internal vertices;
- vertices e, f, g, h, i, and j are the leaves;
- vertices g, h, and i are siblings;
- vertex a is an ancestor of j; and
- j is a descendant of a.

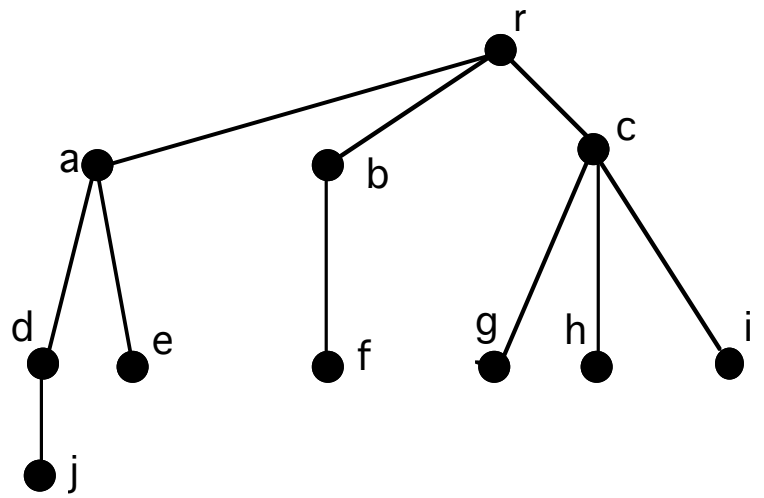


Figure 2.6:

Many applications impose an upper bound on the number of children that a given vertex can have.

Def 2.10. An m***-ary tree*** ($m \geq 2$) is a rooted tree in which every vertex has m or fewer children.

Def 2.11. A ***complete*** m***-ary tree*** is an m-ary tree in which every internal vertex has exactly m children and all leaves have the same depth.

Example 2.3. Fig 2.7 shows two *ternary* (3-ary) trees; the one on the left is complete; the other one is not.

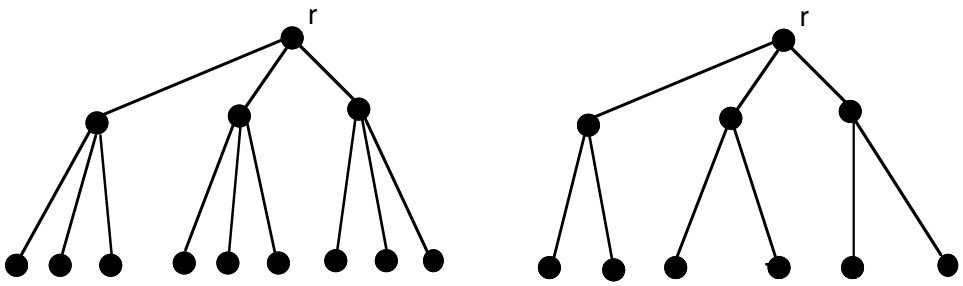


Figure 2.7: Two 3-ary trees: one complete, one incomplete.

Isomorphism of Rooted Trees

Def 2.12. Two rooted trees are said to be ***isomorphic as rooted trees*** if there is a graph isomorphism between them that maps root to root.

Example 2.4. There are more isomorphism types of rooted trees than there are of trees.

Figure 2.8: Isom trees need not be isom as rooted trees.

Ordered Trees

Def 2.13. An ***ordered tree*** is a rooted tree in which the children of each vertex are assigned a fixed ordering.

Def 2.14. In a ***standard plane drawing*** of an ordered tree,

- the root is at the top,
- the vertices at each level are horizontally aligned, and
- the left-to-right order of the vertices agrees with their prescribed order.

Remark 2.1. In an ordered tree, the prescribed *local ordering* of the children of each vertex extends to several possible *global orderings* of the vertices of the tree. One of them, the *level order*, is equivalent to reading the vertex names top-to-bottom, left-to-right in a standard plane drawing. Level order and three other global orderings, *pre-order*, *post-order*, and *in-order*, are explored in §3.3.

Example 2.6. The ordered tree on the left stores the expression $a * b - c$, whereas the one on the right stores $c - a * b$.

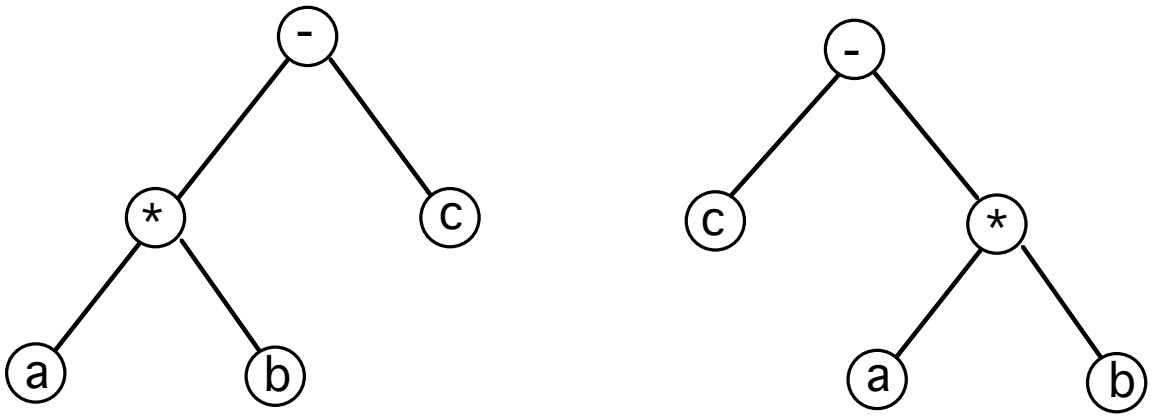


Figure 2.10: Two trees: isom as rooted, not as ordered.

Binary Trees

Def 2.15. A ***binary tree*** is an ordered 2-ary tree in which each child is designated either a ***left-child*** or a ***right-child***.

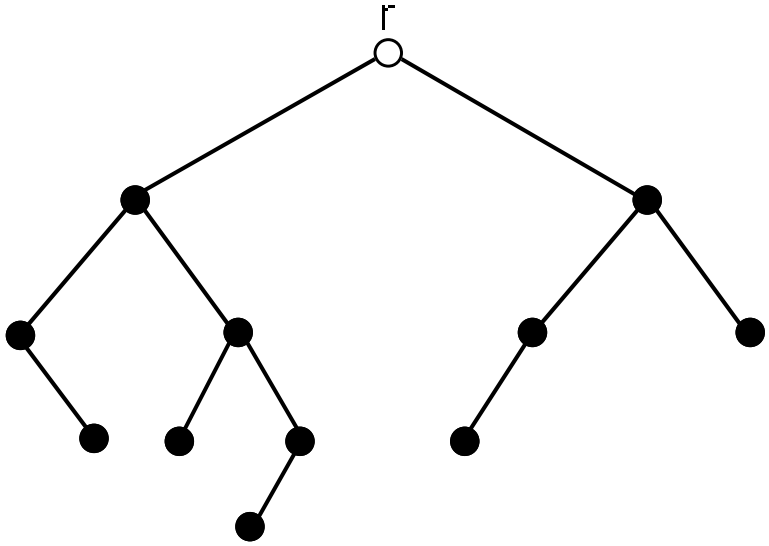


Figure 2.11: A binary tree of height 4.

Def 2.16. The ***left (right) subtree*** of a vertex v in a binary tree is the binary subtree spanning the left (right)-child of v and all of its descendants.

The mandatory designation of left-child or right-child means that two different binary trees may be indistinguishable when regarded as ordered trees.

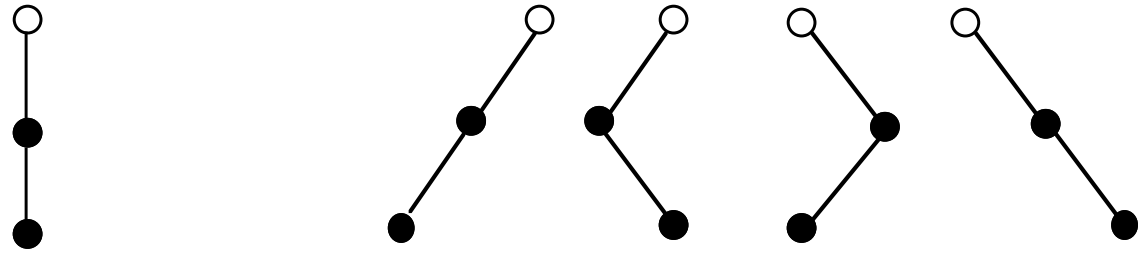


Figure 2.12: One ordered tree yielding four binary trees.

Theorem 2.1. *The complete binary tree of height h has $2^{h+1}-1$ vertices.* □

Corollary 2.2. *Every binary tree of height h has at most $2^{h+1}-1$ vertices.* □

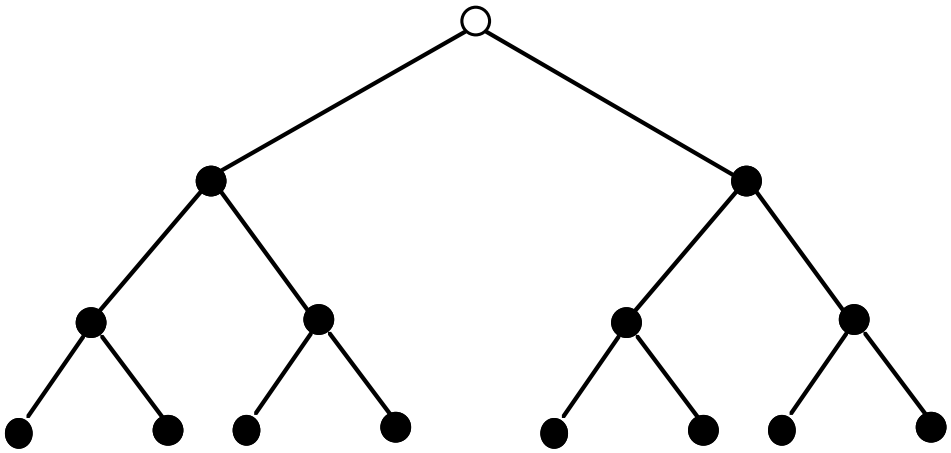


Figure 2.13: Complete binary tree of ht 3 has 15 vertices.

7. Counting Labeled Trees

The number of n-vertex labeled trees is n^{n-2}, for $n \geq 2$, and is known as ***Cayley's Formula***.

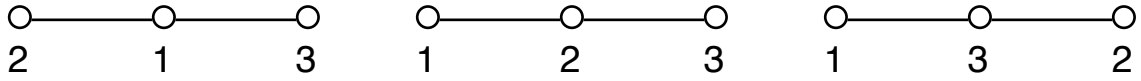
Figure 7.1: Two different labeled trees, isom as unlabeled.

Sometimes, labeled graphs are merely graphs with labels. At other times, they are a class of graph objects whose isomorphisms must preserve the labels.

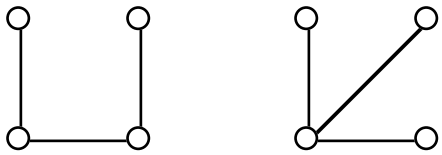
> Cayley's formula counts the number of classes of n-vertex trees under the equivalence relation of label-preserving isomorphism.

SUPPLEMENT: EXAMPLES OF COUNTING TREES WITH CAYLEY'S FORMULA

EXAMPLE: $T_3 = 3^{3-2} = 3$

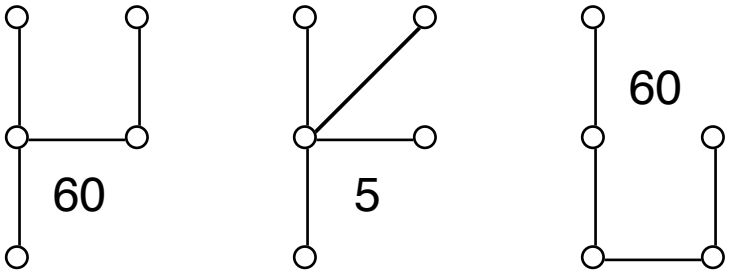


EXAMPLE: $T_4 = 4^{4-2} = 16$



12+4=16

EXAMPLE: $T_5 = 5^{5-2} = 125$



60+5+60=125

Prüfer Encoding

Def 7.1. A ***Prüfer sequence*** of length $n-2$, for $n \geq 2$, is any sequence of integers between 1 and n, with repetitions allowed.

Table 7.1: Prüfer Encoding

Algorithm: Prüfer Encoding
Input: an n-vertex tree with std 1-based vertex-labels.
Output: a Prüfer sequence of length $n-2$.
Initialize T to be the given tree.
For $i = 1$ to $n-2$
Let v be the 1-valent vertex with the smallest label.
Let s_i be the label of the only neighbor of v.
$T := T - v$.
Return sequence $\langle s_1, s_2, \ldots, s_{n-2} \rangle$.

Example 7.1.

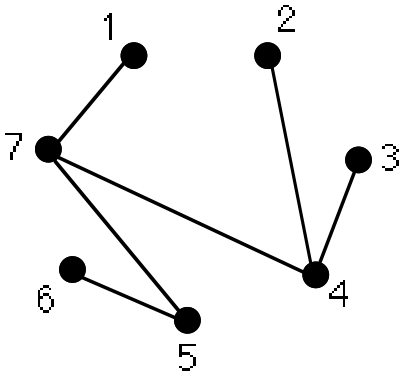


Figure 7.2: A labeled tree, to be encoded into a Prüfer sequence S.

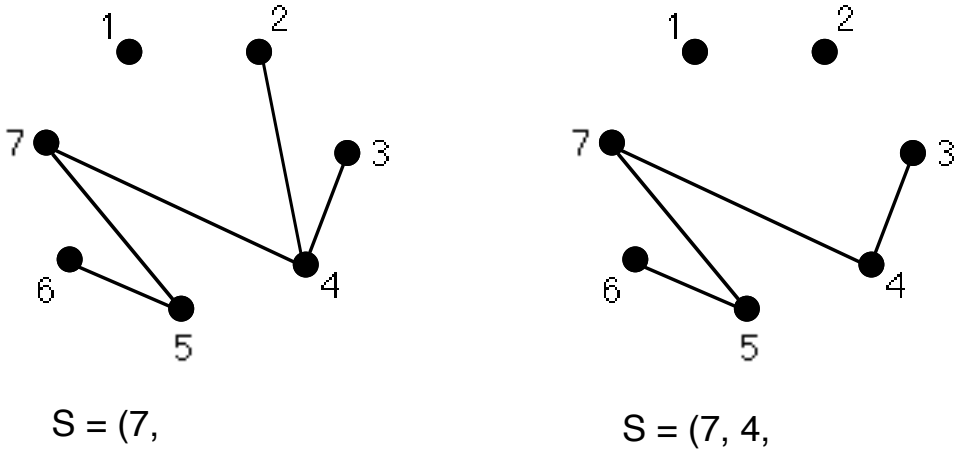


Figure 7.3: First two iterations of the Prüfer encoding.

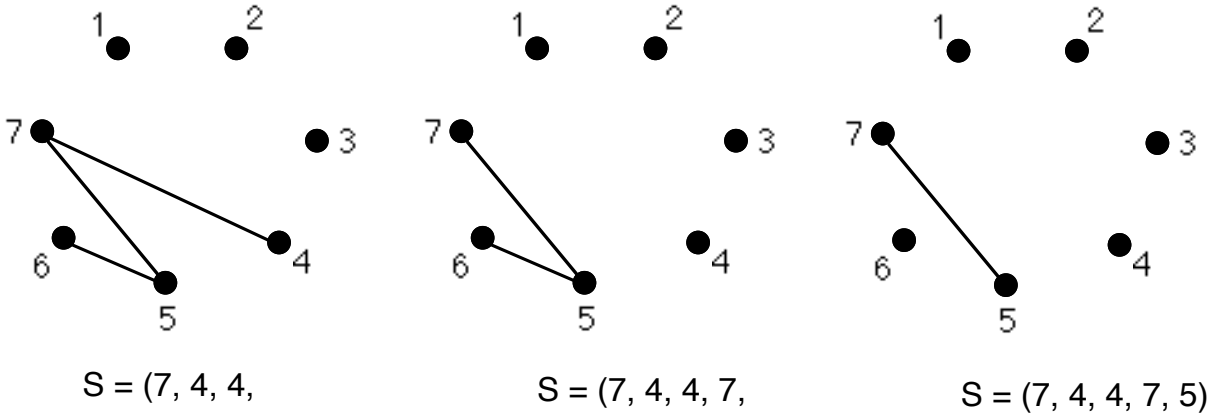


Figure 7.4: Last three iterations of the Prüfer encoding.

Notice for the above example that the degree of each vertex is one more than the number of times that its label appears in the Prüfer sequence. The next result shows this is true in general. In the proof, $l(u)$ denotes the label on vertex u.

Proposition 7.1. *Let d_k be the # occurrences of the label k in a Prüfer encoding sequence for a labeled tree T. Then the degree of vertex k in the tree T equals $d_k + 1$.*

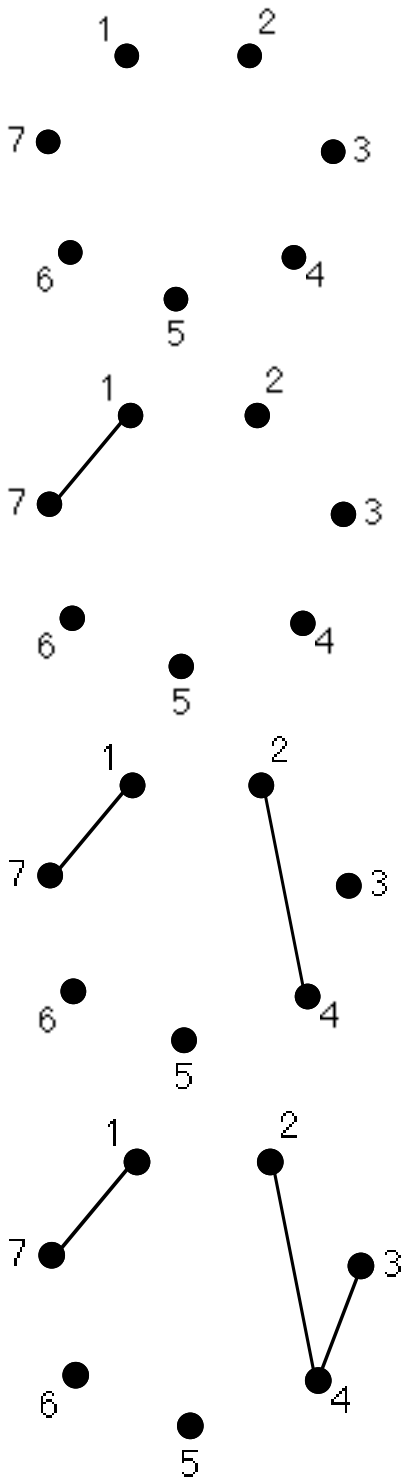
Proof. See text. □

Prüfer Decoding

Table 7.2: Prüfer Decoding

Algorithm: Prüfer Decoding
Input: a Prüfer sequence of length $n-2$.
Output: an n-vertex tree with std 1-based vertex-labels.
Initialize list P as the Prüfer input sequence.
Initialize list L as $1, \ldots, n$.
Initialize forest F as n isolated vertices, labeled 1 to n.
For $i = 1$ to $n-2$
Let k be the smallest # in list L that is not in list P.
Let j be the first number in list P.
Add an edge joining the vertices labeled k and j.
Remove k from list L.
Remove the first occurrence of j from list P.
Add an edge joining the vertices labeled with the two remaining numbers in list L.
Return F with its vertex-labeling.

Example 7.2. Decoding for the input seq 7, 4, 4, 7, 5.

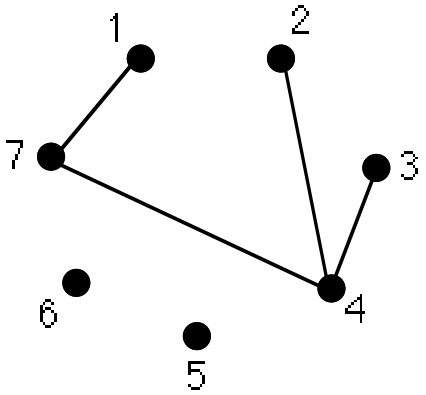


L = 1, 2, 3, 4, 5, 6, 7 P = 7, 4, 4, 7, 5

L = 2, 3, 4, 5, 6, 7 P = 4, 4, 7, 5

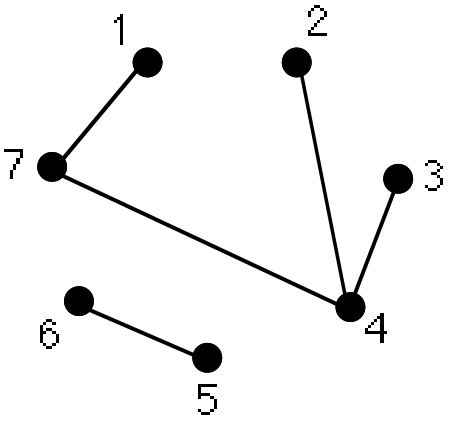
L = 3, 4, 5, 6, 7 P = 4, 7, 5

L = 4, 5, 6, 7 P = 7, 5

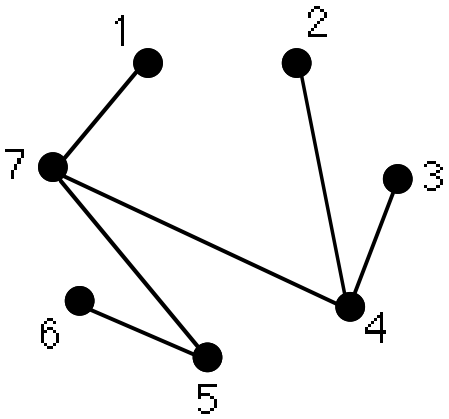


L = 5, 6, 7

P = 5



L = 5, 7



Theorem 7.4 (***Cayley's Tree Formula***)**.** *The number of different trees on n labeled vertices is n^{n-2}.*

Proof. Details in text. □

Remark 7.1. A slightly different view of Cayley's Tree Formula is that it gives us the number of different spanning trees of the complete graph K_n. The next chapter is devoted to spanning trees.

8. Counting Binary Trees

Let b_n denote the number of binary trees on n vertices. Then

$$b_n = b_0 b_{n-1} + b_1 b_{n-2} + \cdots + b_{n-1} b_0$$

This *recurrence relation* is known as the ***Catalan recursion***, and the quantity b_n is called the $\boldsymbol{n^{\text{th}}}$ ***Catalan number***.

Example 8.1. Applying the Catalan recursion to the cases $n = 2$ and $n = 3$ yields $b_2 = 2$ and $b_3 = 5$. Figure 8.1 shows the five different binary trees on 3 vertices.

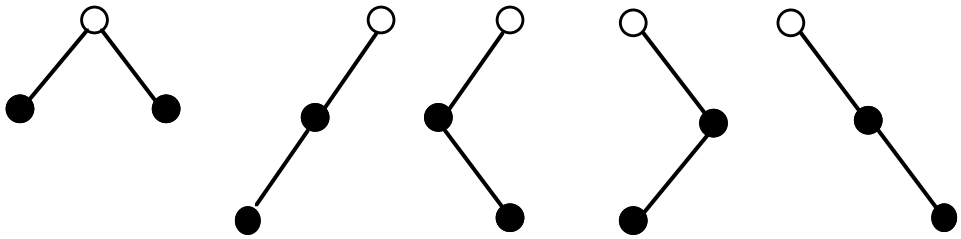


Figure 8.1: The five different binary trees on 3 vertices.

With the aid of generating functions, it is possible to derive the following *closed formula* for b_n.

Theorem 8.1. *The number b_n of different binary trees on n vertices is given by*

$$b_n = \frac{1}{n+1}\binom{2n}{n}$$

□

9. Supplementary Exercises

Exercise 3 What are the minimum and maximum number of vertex orbits in an n-vertex tree?

Exercise 16 Prove that there is no 5-vertex rigid tree.

Exercise 17 Prove that there is no 6-vertex rigid tree.