

# Trees

## 1 Introduction

Trees are very special kind of (undirected) graphs. Formally speaking, a **tree** is a *connected graph* that is *acyclic*.<sup>1</sup> This definition has some drawbacks: given a graph it is not trivial to verify whether it is a tree or not. We next give various characterizations, ending with a characterization due to Euler.

**THEOREM 1** (Tree Characterizations). *The following definitions are all equivalent for a graph  $G = (V, E)$ .*

1.  $G$  is a tree.
2. (Path Uniqueness) For every pair of vertices  $v, w \in V$  there exists exactly one path from  $x$  to  $y$ .
3. (Minimally Connected)  $G$  is connected and deleting any of its edges gives rise to a disconnected graph.
4. (Minimally Cycle Free)  $G$  contains no cycles and any graph  $G'$  obtained from  $G$  by adding an edge from  $\binom{V}{2} \setminus E$  contains a cycle.
5. (Euler's result)  $G$  is connected and the number of edges is one less than the number of vertices, i.e.,  $|E| = |V| - 1$ .

*Proof.* (1)  $\implies$  (2): If not then let  $p, q$  be two paths between  $v$  and  $w$ . Consider the first vertex where  $p$  and  $q$  differ, say  $v'$  (there must be such a vertex). Let  $w'$  be the first vertex where  $p$  and  $q$  again meet  $v'$ .

Then we have a cycle:  $v \overset{p,q}{\sim} v' \overset{q}{\sim} w' \overset{p,q}{\sim} w$ .

(2)  $\implies$  (3): Suppose deleting an edge between two vertices  $v, w$  does not give us a disconnected graph. Then that means there is another path connecting  $v$  and  $w$ . But this cannot be since there is a unique path between any two vertices of a tree.

(3)  $\implies$  (4):

(1)  $\implies$  (5): We will prove it by induction on the number of vertices. We first claim that a tree with at least two vertices always contains at least two **leaves**, i.e., vertices with degree one. If this is true then the proof is straightforward: Delete a leaf and the corresponding edge and we are left with a tree with  $|V| - 1$  vertices and  $|E| - 1$  edges; by induction hypothesis,  $|E| - 1 = |V| - 1 - 1$ , which implies (5). Now to show our claim. Let  $v, w$  be two vertices in the tree that are connected by the longest path in the tree, i.e., they are the furthest apart; these exists because our tree is finite. We claim that both of them are leaves. If not, let's say  $v'$  be the neighbour of  $v$  not on the path connecting  $v$  and  $w$ . Then clearly  $v', w$  are further apart than  $v, w$  giving us a contradiction.

(5)  $\implies$  (1): This direction is similar to the one above. Again, if we can show that there is a leaf then deleting that leaf and the corresponding edge gives us a graph that satisfies Euler's constraint, and hence by induction is a tree. Thus adding the leaf back doesn't destroy the tree property. To prove the claim, we know that the sum of the degrees of all the vertices is  $2|E| = 2|V| - 2$ . Thus, as the graph is connected, the average degree of a vertex is strictly less than two, so there must be vertices with degree exactly one.

**Q.E.D.**

---

<sup>1</sup>Connectedness means that any pair of vertices is connected by a sequence of edges in the graph. Acyclic means that there is a unique path between all pairs of vertices

## 2 Isomorphism for Rooted Trees

The problem of detecting tree isomorphism is a special case of graph isomorphism. For the general problem, we don't know any polynomial time solutions. However, for the special case of tree isomorphism there are polynomial time algorithms, the earliest being by Aho, Hopcroft, Ullman. Here we deal with the isomorphism of **rooted trees**,  $(T, r)$  where  $T$  is a tree and  $r$  is a special vertex called the root. Such trees can be considered in **levels**, where at the first level only the root is present, on the next all neighbours to the root, on the next level all of their neighbours except the ones already seen. In other words, **level  $i$**  contains all vertices at distance  $i$  from the root. We can talk of a traversal of the tree using standard graph traversal algorithms (BFS, DFS) and associate with each vertex  $v$  a subset of its neighbours called the **descendants**, which are the vertices reachable from the root  $r$  on a path traversing through  $v$ .

Testing isomorphism requires us to develop isomorphism invariants, i.e., a property which is the same for all graphs iff the graphs are isomorphic to each other. We start with guesses for a good invariant until we have the right one.

First observe that an isomorphism preserves edge incidences, so if we start from the root then we would expect to see the same number of levels and equal number of vertices at each level. Is this criterion isomorphism invariant? The answer is no, as Figure 1 illustrates.

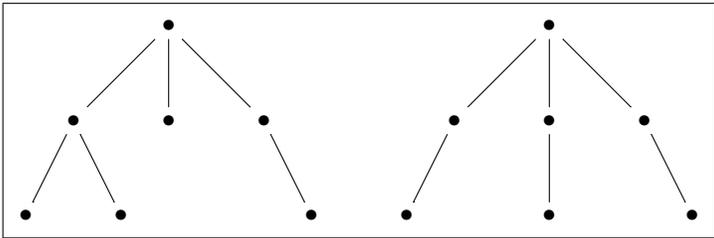


Figure 1: Two non-isomorphic graphs with equal number of vertices at each level

The above failure emphasizes that we not only want the same number of vertices at each level but also that each vertex at each level has the same “spectrum”, which is roughly the number of outgoing edges. More precisely, the **degree spectrum** of a tree is a sequence of non-negative integers  $(d_i)$  where  $d_i$  is the number of vertices with  $i$  outgoing edges when we traverse the tree from the root down. Is the degree spectrum at each level of the tree is an isomorphism invariant? The answer is no, as Figure 2 illustrates. The problem

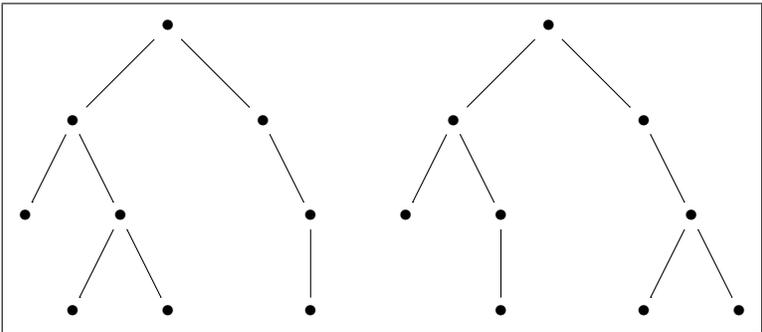


Figure 2: Two non-isomorphic graphs with the same degree spectrum at each level

with our definitions of isomorphism invariants is that they are treating each level separately. What we need is an invariant that for each vertex at every level captures the subtree rooted at the vertex completely, i.e., each vertex should carry the complete structural information of the subtree rooted at that vertex. If we have such an invariant then we can do a comparison level by level. One starting point is to associate a tuple with every vertex in the tree: the leaves get the tuple  $()$ , and a vertex gets the tuple  $(L)$ , where  $L$  is the

concatenated list of the tuples corresponding to its children. A simple post-order traversal can yield us the tuple corresponding to each vertex in the tree. However, we have still not accounted for the *ordering* in which the children's tuples are concatenated to form the parents tuple. To do that we replace each left-paren 1 and each right paren with 0. Thus we have associated a binary number with each vertex, and we claim that this gives us an invariant. When constructing the binary number corresponding to a vertex, we concatenate the numbers corresponding to its children in descending order. Let the number so obtained be the **canonical number** of the tree.

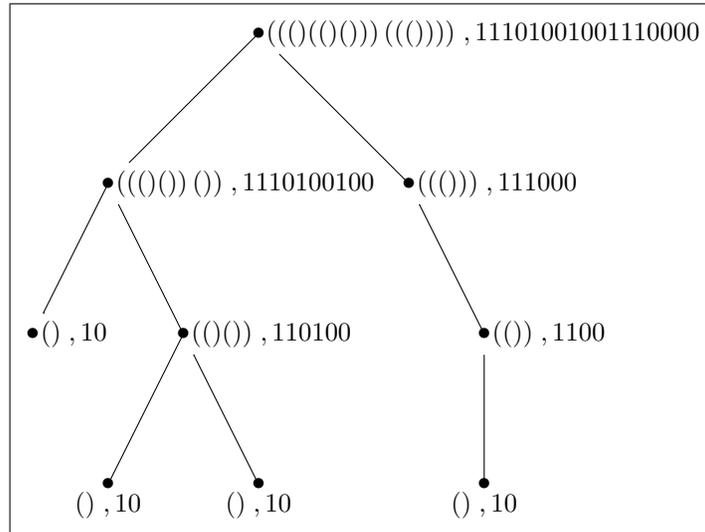


Figure 3: The tuple and the binary number associated with the vertices of the tree

We claim the following:

**THEOREM 2.** *The canonical number of a rooted tree is an isomorphism invariant, i.e.,  $(T_1, r_1) \equiv (T_2, r_2)$  iff their canonical numbers are the same.*

*Proof.* ( $\Rightarrow$ ) The proof is by induction on the level number; the base case is obvious. Since any isomorphism permutes the children of the root, the ordering of the canonical numbers of the children is unique up to permutations that permute the children with the same canonical number. By the inductive hypothesis, their canonical numbers are isomorphism invariant, thus the canonical number associated with the root is isomorphism invariant as well.

( $\Leftarrow$ ) Again the proof is by induction on levels. If we look at the list of the canonical numbers of the two roots  $r_1$  and  $r_2$ , then up to permutations within same canonical numbers, the two lists are the same. By induction hypothesis, we know that the sub-trees corresponding to the children of  $r_1$  and  $r_2$  are isomorphic. Since the ordered list of canonical numbers is same for  $r_1$  and  $r_2$ , and the subtrees of their childrens are isomorphic, it follows that the trees  $T_1$  and  $T_2$  are isomorphic. **Q.E.D.**

The algorithm to construct the canonical number is straightforward. Do a post-traversal of the tree and at each vertex  $v$  do the following:

1. sort the canonical numbers  $b_1 \geq b_2 \geq \dots \geq b_k$  of the children of  $v$  (here  $k := d_v$ , the outdegree of  $v$  wrt, say, a DFS traversal from the root),
2. assign the canonical number for  $v$  as  $1b_1b_2 \dots b_k0$ .

The traversal of the tree takes  $O(|V|)$  time. What's the work done at each node? The work at each node involves sorting a list of binary numbers. To sort binary numbers of lengths  $\ell_1, \dots, \ell_k$  we can do the following: first pad the numbers with leading zeros so that all of them have the same length  $\ell$ ; then put the numbers them into two bins based upon their most significant bit, then refine these two bins wrt the second significant

bit, and so until we have sorted the numbers. What is the complexity of sorting  $k$  numbers of lengths  $\ell$ ? Since we look at all the bits of the  $k$  numbers exactly once, the complexity of sorting binary numbers of length  $\ell$  is  $O(k\ell) = O(d_v\ell)$ . The worst-case length of canonical numbers is  $2n$  (at every level we add a one and zero, and in the worst case there are  $n$  levels). Thus the worst-case complexity at a vertex  $v$  is  $O(nd_v)$ . Thus the overall complexity is  $O(n\sum_v d_v) = O(n|E|) = O(n^2)$ .

Can we improve upon this? Do we have to carry the complete structure of the subtree rooted at a vertex to its parent? Also, if the trees are not isomorphic, then can we detect it earlier than comparing the canonical number of the roots? The answer to the last question is yes – if we compare the canonical numbers level by level then we can detect non-isomorphism earlier. More precisely, we assign the canonical number to each vertex in the tree, and corresponding to each tree  $T$  we form a list  $T(i)$  of the sorted canonical numbers of the vertices at level  $i$ ; all of this can be done in  $O(|V|^2)$  time. After this, we proceed to compare the trees level by level. If at the  $i$ th level the two lists  $T_1(i)$  and  $T_2(i)$  do not match then we declare that the trees are not isomorphic. The overall complexity in the worst case still is quadratic; nevertheless, a failure can be detected earlier. However, we can improve the worst case complexity as well. Since we are proceeding level-by-level, if we have detected isomorphism till level  $(i + 1)$  then at level  $i$  we do not need to carry the canonical numbers of level  $(i + 1)$ . *We only to have ensure that the isomorphic subtrees at level  $(i - 1)$  are assigned the same numbers.* Based upon this idea Aho, Hopcroft and Ullman gave the following procedure:

1. Assign 1 to all leaves of the trees  $T_1$  and  $T_2$ .
2. Suppose inductively that we have assigned some integers to the vertices of  $T_1$  and  $T_2$  at level  $(i + 1)$ . Let  $L_1$  be the list of vertices at level  $(i + 1)$  of  $T_1$  sorted in increasing order according to the integers assigned to them; similarly  $L_2$  for  $T_2$ .
3. We next assign a tuple of integers to the vertices at level  $i$  as follows: scan the list  $L_1$  from left to right and for each vertex  $v$  in the list look at its parent at level  $i$ ; insert the integer corresponding to  $v$  in the tuple corresponding to its parent. Thus for each vertex  $w$  at level  $i$  we have a tuple  $(j_1, \dots, j_k)$ ,  $j_1 \leq j_2 \leq \dots \leq j_k$ . We sort these tuples in lexicographic order to get a list  $S_1$ ; since we might have tuples with different sizes, we impose the criterion that tuples of smaller size are lexicographically smaller than tuples of larger size. Similarly create a list  $S_2$  for  $T_2$ . If  $S_1$  and  $S_2$  are not identical then the two trees are not isomorphic. Otherwise, we assign 1 to all vertices  $w$  at level  $i$  corresponding to the first distinct tuple in  $S_1$ , assign 2 to all vertices corresponding to the second distinct tuple in  $S_1$  and so on; similarly for  $T_2$ . Update  $L_1$  to be the list of vertices of  $T_1$  at level  $i$  sorted according to the numbers assigned to them; similarly update  $L_2$  for  $T_2$ , and proceed inductively.
4. If eventually the tuple associated with the roots are the same then the two trees are isomorphic.

Complexity: Is  $O(|V|)$  since lexicographic sort of strings of lengths  $\ell_1, \dots, \ell_k$  with entries at most  $m$  takes  $O(m + \sum_{i=1}^k \ell_i)$  time. In our case,  $m = n$  and  $\sum_i \ell_i = \sum_{v \in V} d_v$ , where  $d_v$  is the number of descendants of a vertex  $v$  in the rooted tree. But we know that the latter sum is twice the number of edges of the tree and hence is  $O(|V|)$ .

See Aho-Hopcroft-Ullman p. 84 for more details.

### 3 Enumeration of Trees

Cayley's first paper on trees (where he also introduced the notation) in 1857 is called "On the theory of the analytic forms called trees" and arose as a study of differential operators (from a problem of Sylvester). His concern in this paper was on counting the number,  $A_n$ , of *rooted trees* with  $n$  edges (branches according to Cayley's terminology). The idea is to construct a rooted tree  $T$  with  $n$  edges from trees with fewer edges. How can we do that? The tree  $T$  can have one edge from the root, and then it has to connect to a rooted tree with  $n - 1$  edges;  $T$  can also have two edges from the root to two roots trees with  $p$  and  $q$  edges, where  $p + q = n - 2$ ; similarly, if there are three edges from the root then they connect to three rooted trees with edges  $p, q, r$  such that  $p + q + r = n - 3$ ; so on, when all the  $n$  edges are connected from the root to  $n$  leaves. Let  $j_i$  be the number of trees with  $i$  edges; note that there are  $A_i$  ways of choosing each such tree. Then the

total number of edges contributed to  $T$  by these  $j_i$  trees is  $j_i(i+1)$ , where  $j_i \cdot i$  comes from each of the tree with  $i$  edges and  $j_i$  comes from the edges connecting these trees to the root. Thus

$$n = \sum_{i=0}^n j_i(i+1).$$

Let  $A(x) := \sum_n A_n x^n$ . Then the equation above suggests that the coefficient of  $x^n$  on the LHS comes from combining the convolution of series of the form

$$(1 + x^{i+1} + x^{2(i+1)} + \dots + x^{j_i(i+1)} + \dots) = \frac{1}{1 - x^{i+1}}.$$

This is what we had for partitions of  $n$ . However, the key difference here is that each of the  $A_i$  trees with  $i$  edges can contribute such a term to  $T$ . Thus we have

$$\sum_n A_n x^n = \frac{1}{(1-x)} \frac{1}{(1-x^2)^{A_1}} \frac{1}{(1-x^3)^{A_2}} \frac{1}{(1-x^4)^{A_3}} \dots = \prod_{i \geq 1} \frac{1}{(1-x^{i+1})^{A_i}}.$$

From this equation we can recursively compute each  $A_n$  from  $A_1, \dots, A_{n-1}$ .

### 3.1 Labelled trees

The second result that Cayley claimed (but did not give a convincing proof) was that the number  $T_n$  of trees that can be formed from  $n$  labels is  $n^{n-2}$ . A standard proof by Prüfer shows that each labelled tree can be bijectively mapped to sequences  $(a_1, \dots, a_{n-2})$ , where  $a_i \in [n]$ ; clearly there are  $n^{n-2}$  such sequences which gives the proof.

The bijection is as follows: Given a tree  $T$ , take the leaf  $b_1$  with the smallest label, let  $a_1$  be the unique node connecting to  $b_1$ ; let  $T'$  be the tree obtained by deleting  $b_1$  from  $T$ ; then pick  $b_2$  the leaf with the smallest label in  $T'$ , let  $a_2$  be the unique node connecting to  $b_2$ ; recursively proceed in this manner to get  $a_3, a_4$  so on; in the end we are left with two nodes connected by an edge, assign them the empty string.

In the converse: Given  $(a_1, \dots, a_{n-2})$ , find the smallest number in  $[n]$ , say  $b_1$  that does not appear in the sequence; connecting  $a_1$  and  $b_1$ , and delete  $b_1$  from  $[n]$  and  $a_1$  from the sequence; proceed recursively until we have exhausted the complete sequence.

Another combinatorial proof is due to Joyal (see PTB). Consider the tuple  $(T, a, b)$ , where  $T$  is a labelled tree with labels in the set  $[n]$ , and  $a, b \in [n]$  are two nodes from the tree (think of them as the left-end and right-end of the tree). Let  $T'_n$  be the number of such tuples. We have  $T_n$  choices for the tree  $T$  and  $n^2$  choices for  $a, b$ ; thus  $T'_n = T_n n^2$ . Therefore, if we can show that  $T'_n = n^n$ , then we have the desired formula for  $T_n$ . This would follow if we can show a bijection between  $T'_n$  and functions from  $[n] \rightarrow [n]$ .

Let  $f : [n] \rightarrow [n]$  be a function. Corresponding to  $f$ , we can naturally associate a directed graph  $G_f$  whose vertices are numbers in  $[n]$ , and there is an edge from  $i$  to  $j$  if  $j = f(i)$ . Note that  $G_f$  may not be connected. However, each component of  $G_f$  must contain a cycle, since there are equal number of vertices and edges in each component; moreover, there can only be one cycle in each component, because outdegree of all vertices is one. Our aim is to remove all these cycles. The idea is to take all the vertices that are on some cycle and put them on a path  $P$  in the tree that we want to construct (some suitable order has to be chosen); the rest of the graph is acyclic, and that we can just attach to the path  $P$  that we constructed. What do cycles mean in terms of the function? Let  $M$  be the union of all the vertices in  $G_f$  that are part of a cycle. Then  $f$  restricted to  $M$ , denoted as  $f|_M$ , forms a bijection from  $M$  to  $f(M)$ . How do we construct a triplet  $(T, a, b)$  corresponding to  $f$ ? Let  $m_1 < m_2 < \dots < m_k$  be the elements of  $M$  ordered naturally. Then we form a path connecting  $f(m_i)$  to  $f(m_{i+1})$ ,  $i = 1, \dots, k-1$ , in  $T$ ,  $a := f(m_1)$ , the left end, and  $b := f(m_k)$ , the right end. The remaining portion of  $T$  is obtained from  $G_f$  by attaching to  $f(m_i)$  the subgraph, if any, in  $G_f$  that is not part of the cycle on which  $f(m_i)$  lies. This is illustrated in Figure 4(a) for the function:

$$f := \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 4 & 2 & 5 & 5 & 1 & 8 & 9 & 9 & 10 & 8 \end{pmatrix};$$

the set  $M = \{1, 2, 4, 5, 8, 9, 10\}$  and

$$f|_M = \begin{pmatrix} 1 & 2 & 4 & 5 & 8 & 9 & 10 \\ 4 & 2 & 5 & 1 & 9 & 10 & 8 \end{pmatrix}.$$

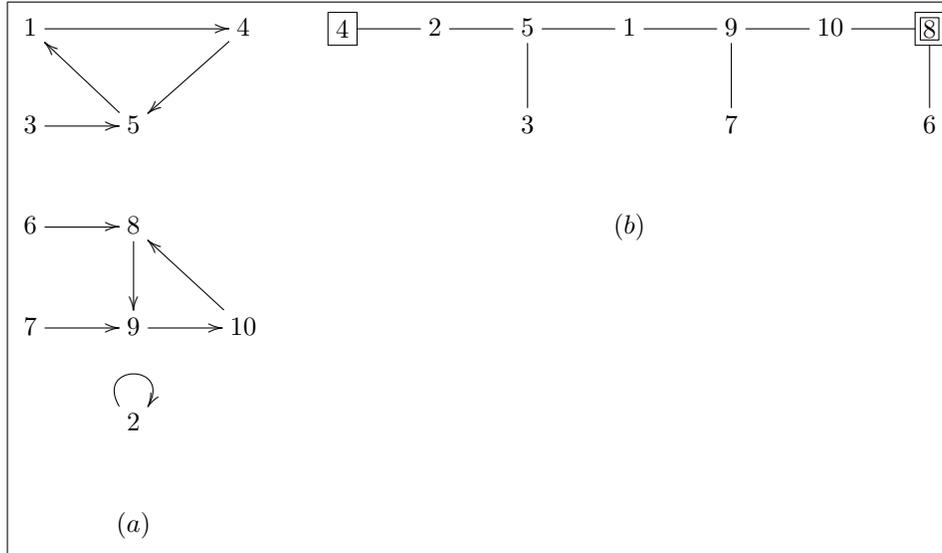


Figure 4: (a) The directed graph  $G_f$ ; (b) The tree  $T$ , where 4 is the left end and 8 the right end.

For the converse, given  $(T, a, b)$ , let  $P = (p_1, \dots, p_k)$  be the ordered tuple denoting the unique path from  $a$  to  $b$ , therefore,  $p_1 = a$  and  $p_k = b$ . Let  $M = \{m_1, \dots, m_k\}$  be the set of elements of  $P$  ordered in increasing order, thus,  $m_1 < m_2 < \dots < m_k$ . The bijection  $f|_M$ , which is the restriction of the desired  $f$  to  $M$ , is defined as  $f|_M(m_i) = p_i$ . For the remaining  $i$ ,  $f(i)$  is the first element on the path from  $f(i)$  to  $P$ . Suppose the tree is as shown in Figure 5. Then  $P = (4, 5, 1, 9, 10, 8)$ ,  $M = \{1, 4, 5, 8, 9, 10\}$ ,

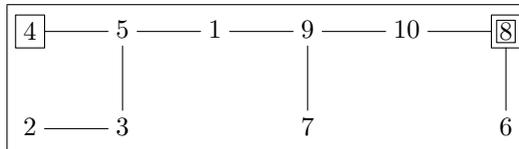


Figure 5: A tree  $T$ , where 4 is the left end and 8 the right end.

$$f|_M = \begin{pmatrix} 1 & 4 & 5 & 8 & 9 & 10 \\ 4 & 5 & 1 & 9 & 10 & 8 \end{pmatrix}$$

and  $f(2) = 3$ ,  $f(3) = 5$ ,  $f(5) = 7$  and  $f(6) = 8$ . It is clear that this procedure when applied to Figure 4(b) yields the corresponding function  $f$ .