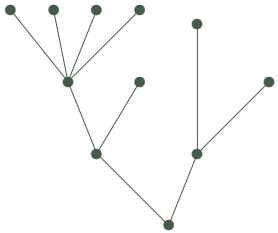
5 Basics of Trees

Trees, actually acyclic connected simple graphs, are among the simplest graph classes. Despite their simplicity, they still have rich structure and many useful application, such as in data structures.



Brief outline of this lecture

- Definition and basic properties of trees (acyclic connected graphs).
- Rooted and ordered trees, search trees.
- Efficient isomorphism testing for trees.
- Spanning trees in a graph, enumeration of spanning trees.

5.1 Definition and properties of trees

Definition 5.1. A tree is a connected graph T without cycles.

A graph whose connected components are trees is called a *forest*. □

Notice that a tree must be simple since a loop is a cycle of length one and parallel edges form a cycle of length two. More basic properties of trees follow...

Lemma 5.2. *A tree on more than one vertex contains a vertex of degree 1 (a "leaf").* □

Proof: We consider a longest possible path in a tree T. Since there are no cycles in T, the endvertices of this path must have no other edges incident with them (hence the degree one). □ □

Theorem 5.3. *An n-vertex tree has exactly $n-1$ edges for $n \geq 1$.*

Proof: By induction on n... □

A one-vertex tree has $n - 1 = 0$ edges, OK. □

Let a tree T have $n > 1$ vertices. By Lemma 5.2, T has a vertex v of degree 1. Denote by $T' = T - v$ the subgraph obtained from T by deleting v. Then T' is also connected and acyclic, and hence T' is a tree on $n-1$ vertices. By induction assumption, T' has $n-1-1$ edges, and so T has $n-1-1+1 = n-1$ edges. □

Theorem 5.4. *Any two vertices of a tree are connected via exactly one path.* □

Proof: Since a tree T is connected by the definition, any two vertices u, v are connected via a path.

If there were two distinct paths P_1, P_2 between u, v, then their symmetric difference (w.r.t. edge sets) — a subgraph $H = P_1 \Delta P_2$ of nonempty edge set — has all degrees even and not all zero. On the other hand, H as a subgraph of a tree is acyclic, and hence its components are trees (not all isolated vertices), and by Lemma 5.2 there has to be a vertex of degree 1 in H, a contradiction. □ □

Corollary 5.5. *If a single new edge is added to a tree, this results in creation of exactly one cycle.*

Proof: If u, v are not neighbours in a tree T, then the new edge uv forms one cycle with the unique $u-v$ path from Theorem 5.4. □ □

Theorem 5.6. *A tree is a minimal connected graph (on a given vertex set).* □

Proof: We prove, from the previous properties, the two directions of this claim:

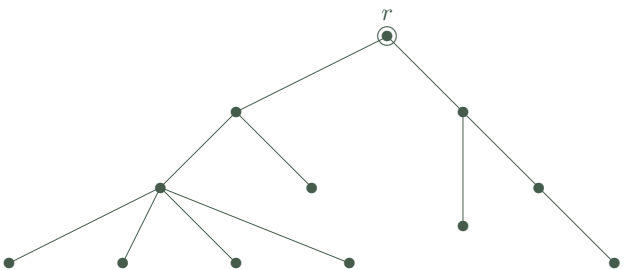
- If T is a tree, then removal of any edge from T disconnects it.
- Conversely, if a connected graph had a cycle, then it would not be minimal among its connected subgraphs.

□

5.2 Rooted trees

The introduction of *rooted trees* is motivated primarily by practical applications in which a "tree data structure" has to be grasped at some vertex (the root), and also by historical use of family-trees. Actually, the traditional family-trees motivated much terminology of graph trees. □

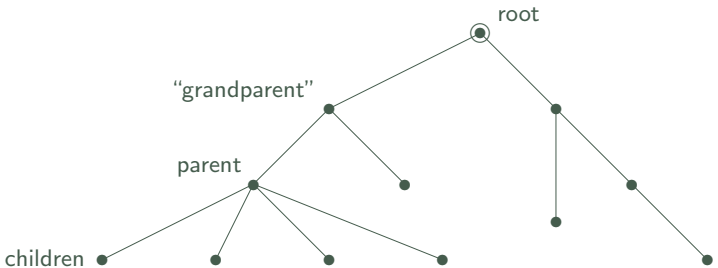
Definition 5.8. A rooted tree is a tree T
together with a highlighted *root* $r \in V(T)$, i.e. shortly a pair (T, r).



Interestingly, trees in computer science grow top-to-bottom...

Definition: Consider a rooted tree (T, r) and its vertex v. Let u denote the neighbour of v on the unique path towards the root r. Then u is the *parent* of v, and v is a *child* (descendant) of u. □

The root has no "ancestors".



□

Definition: A vertex of degree 1 in a tree is called a *leaf*.

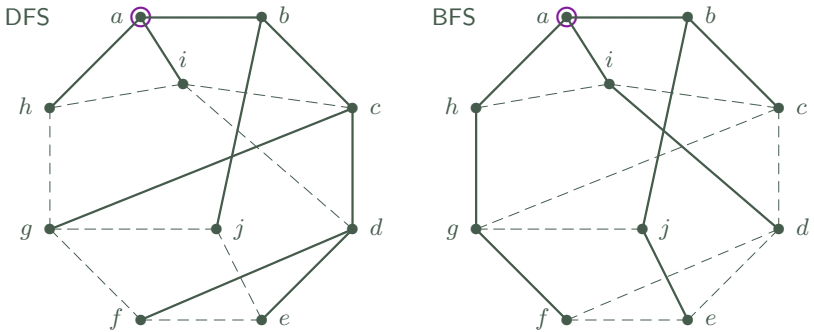
Strictly saying, a root of degree 1 is also a leaf, but we shall use this term carefully for roots to avoid confusion.

Notice that a leaf (which is not a root) has no descendants.

DFS and BFS trees of a graph

Every instance of the graph seach Algorithm 2.4 defines a *search tree* S_G of the graph G as follows:

- the vertices of S_G are all of G (a "spanning" tree),
- an edge $f \in E(G)$ is in S_G iff one end of f has been just discovered while processing f.

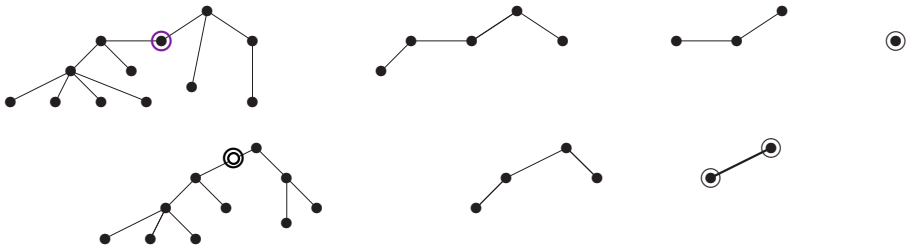


Center of a tree

Definition: The *center* of a (nonempty) tree T is the vertex or the edge found by the following recursive procedure:

- If T is a single vertex or a single edge, then this (T) is the center. □
- Otherwise, we create a smaller tree $T' \subset T$ by removing all leaves of T at the same time. This T' is nonempty, and we determine the center of T' recursively. The center of T is identical to this center of T'. □

Example 5.9. *The definition of a tree center is illustrated by the two examples:*



□

Lemma 5.10. *The vertices of the center of a tree T (according to the previous definition) are the same as those of a center defined in Section 3.1.* □

`Proof` : If the center is found in t iterations of the definition, then T contains a path P of length $2t$ or $2t+1$, and the center U is formed by the "middle" vertex or edge of P. Hence the radius of T is at least t or $t+1$, respectively. On the other hand, the vertices of U have this excentricity, and so they are in the center of T in the sense of 3.1. □

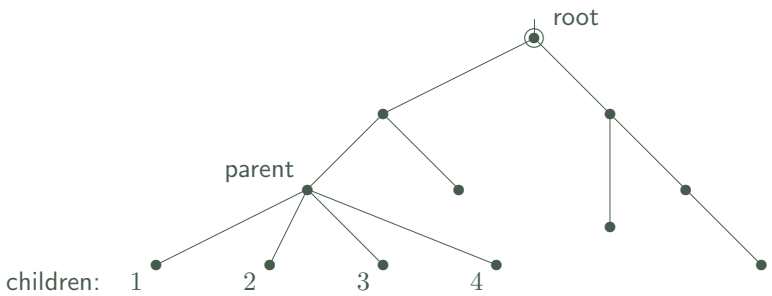
Conversely, assume a vertex $v \in V(T)$ having excentricity $\mathrm{rad}(T)$. If $v \notin U$, then, since there is a unique path connecting v to some vertex of P, the distance of v to one of the ends of P must be larger than $\mathrm{rad}(T)$, a contradiction. Therefore, U is exactly the center of T in the sense of 3.1. □

Fact. If one needs to determine a unique root of a tree, the center is a good choice. (If the center is an edge, then the root may be a new vertex subdividing this edge.)

Ordered rooted trees

Definition: A rooted tree (T, r) is *ordered* if the orderings of the children of each its vertex are prescribed.
A rooted ordered tree is also called a *planted tree*. □

A rooted ordered tree can be visualized as a tree drawn in the plane without "crossings" and with a marked root. The cyclic ordering of the child edges (against the parent edge / the root mark) then gives the tree ordering of the children.



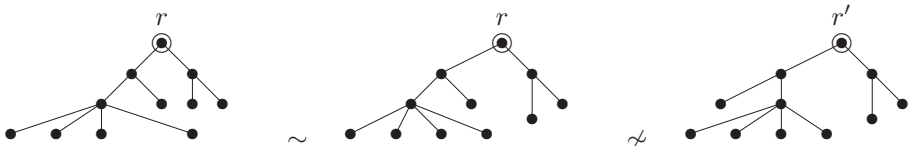
5.3 Tree isomorphism

The isomorphism problem of trees, as a special instance, has already been defined and studied on general graphs. Tree isomorphism problem, however, has one additional interesting aspect — that there is an efficient and straightforward solution to it. □

Definition: Two rooted trees (T, r) and (T', r') are *isomorphic* if there is an isomorphism between T and T' such that r is mapped onto r'.

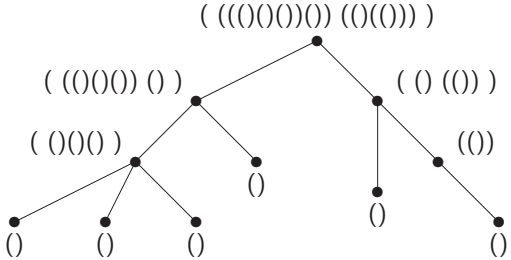
□

Definition: Two rooted ordered trees are isomorphic if there exists a rooted isomorphism between them which preserves the orderings of children at each vertex.



Coding of ordered rooted (planted) trees

Definition:



The *code* of a rooted ordered tree is determined recursively from the codes of the subtrees of its root, concatenated in a prescribed order and enclosed in a pair of parentheses. □

Remark: Instead of '(' and ')', one may use other pair of symbols such as '0' and '1'.

Lemma 5.11. *Two rooted ordered trees are isomorphic if and only if their codes are identical as strings.*

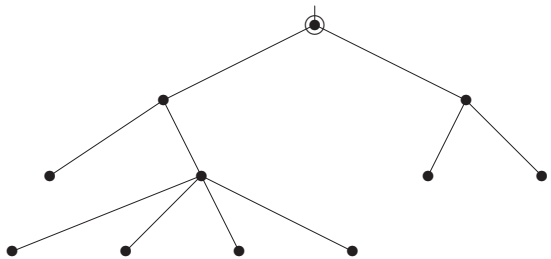
Proof: Easily by induction on the depth... □

Example 5.12. *Draw a planted tree with the following code*

$$(\,(\,()\,(\,()\,()\,()\,()\,)\,)\,(\,(\,()\,()\,)\,)\,)\,.\,\square$$

If s is the code of a rooted ordered tree T, then—as easily follows from the definition—T can be drawn using the following procedure:

- Reading the first char '(', we put the pen on the paper (marking the root). $\square$
- Reading every next '(', we draw the edge to the next child of the current vertex. $\square$
- Reading each ')', we return the pen to the parent. In the root, we lift the pen and are finished. $\square$



$\square$

Isomorphism testing for trees

The core idea behind the tree isomorphism testing algorithm is to compare their codes which are rooted in the centers and ordered lexicographically among the children.

Algorithm 5.13. Testing isomorphism of two trees.

```
input < trees T and U;
if (|V(T)|!=|V(U)|) return 'Not isomorphic.';
(T,r) = rooted_at_center(T);   (U,s) = rooted_at_center(U);
k = minimal_code(T,r);         m = minimal_code(U,s);
if (k==m as strings) return 'Isomorphic.';
else return 'Not isomorphic.'; □

Function minimal_code(tree X, root r) {
    if (|V(X)|==1) return "()";
    d = number of components of X-r, i.e. subtrees of r;
    for (i=1,...,d) {
        Y[i] = i-th connected component of X-r;
        s[i] = i-th neighbour of r, i.e. the root of the subtree Y[i];
        k[i] = minimal_code(Y[i],s[i]);
    }
    sort as k[1]<=k[2]<=...<=k[d] lexicographically;
    return "("+k[1]+...+k[d]+")";
}
```

The proof

A mathematical proof of Algorithm 5.13 is given as follows.

Theorem 5.14. *Let T, U be two trees on the same number of vertices, and let (T', r) and (U', s) be their rooted trees obtained in the first step of Algorithm 5.13 (r, s are the centers of T, U). Then:*

a) T and U are isomorphic if, and only if, (T', r) is isomorphic to (U', s).

b) (T', r) is isomorphic to (U', s) if, and only if,

$$\texttt{minimal\_code}(\texttt{T}', \texttt{r}) = \texttt{minimal\_code}(\texttt{U}', \texttt{s}).$$

Proof (sketch): Claim (a) immediately follows from the unique choice of a center.

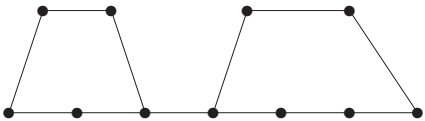
Claim (b) is proved by induction on the depth of our rooted trees (T', r) and (U', s). If their depths differ, then they are clearly not isomorphic. On the other hand, two rooted trees of depth 0 are always isomorphic with the code '()'. So let (T', r) and (U', s) be both of depth $\ell > 0$. If their codes in the algorithm are identical, then (T', r) and (U', s) are clearly isomorphic.

Conversely, if (T', r) and (U', s) are isomorphic, then there exists a "matching" bijection between the subtrees of their roots, and hence the codes of matching subtrees are identical by the inductive assumption. Since the codes of the subtrees are sorted in the same way, we get $\texttt{minimal\_code}(\texttt{T}', \texttt{r}) = \texttt{minimal\_code}(\texttt{U}', \texttt{s})$. □

5.4 Spanning trees in graphs

Definition 5.15. **A spanning tree** of a connected graph G is a subgraph T of G such that T is a tree and $V(T) = V(G)$. □

Example 5.16. *How many distinct spanning trees are contained in this graph?*



□

We argue as follows; which edges have to be removed from our graph to obtain a tree? Easily, one from each of the two cycles. Hence, by the principle of independent choices, we have got $5 \cdot 6 = 30$ spanning trees. □

The following claim is considered a "beautiful gem" of graph theory.

Theorem 5.17. (Cayley)
The complete graph K_n for $n > 0$ contains exactly $\boldsymbol{n^{n-2}}$ distinct spanning trees. □

Even more generally, we have:

Definition. The *Laplace matrix* $\boldsymbol{Q} = (q_{ij})_{i,j=1}^{n}$ of an n-vertex graph G is defined:

- $q_{ii} = d_G(i)$ (vertex degree),
- $q_{ij} = 0$ for non-adjacent vertices $i \neq j$,
- $q_{ij} = -1$ for adjacent vertices $i \neq j$. □

Theorem 5.18. *Let $\boldsymbol{Q}$ be the Laplace matrix of a graph G, and let a matrix $\boldsymbol{Q}'$ result from $\boldsymbol{Q}$ by erasing the first row and the first column. Then the number of distinct spanning trees in G equals the determinant $|\boldsymbol{Q}'|$.*

The proof is, however, quite difficult and using advanced linear algebra.