Lecture 15 - Trees, Recursive Definitions, and Structural Induction

6.042 - April 8, 2003

Note: These lecture notes are supplemented by Appendix B of *Introduction to Algorithms* by Cormen, Leiserson, Rivest, and Stein, which is on the course web site.

1 Trees

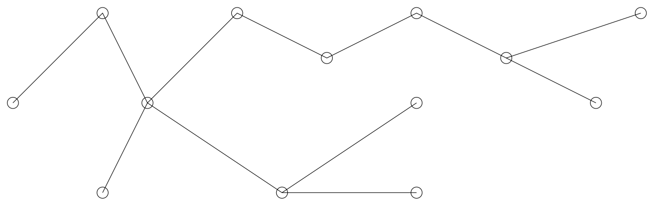
The first part of this lecture concerns *trees*, one of the most important kinds of graph used in computer science.

Definition 1 *A* tree *is a connected, acyclic, undirected graph.*

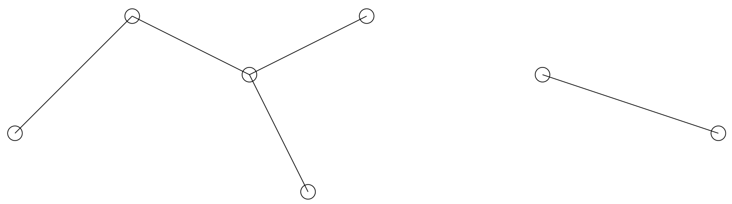
The terms in this definition require some explanation themselves. A *path* from vertex x_0 to vertex x_n is a sequence of edges (x_0, x_1), (x_1, x_2), ..., (x_{n-1}, x_n). (However, a path is often specified by giving the sequence of vertices traversed, which is $x_0, x_1, \ldots, x_n$ in this case.) A graph is *connected* if there is a path between every pair of vertices. A *simple path* is a path in which no edge appears more than once. A *cycle* is a simple path that starts and ends at the same vertex. A graph containing no cycles said to be *acylic*.

These formal definitions may be confusing, but the underlying idea of a tree is really simple, as the following examples show.

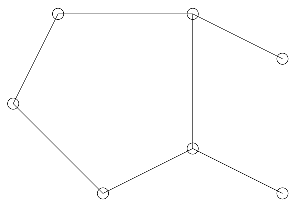
Example 1. The graph below is connected because there is a path between every pair of vertices. It is acylic, because there are no cycles. Because this graph is connected and acyclic, it is a tree.



Example 2. This undirected graph is *not* a tree. The graph is acylic, but not connected. In particular there is no path from a vertex in the left component to any vertex in the right component.



Example 3. This graph is also *not* a tree. It is connected, but not acyclic.



Several special types of trees are particularly important in computer science: binary trees, rooted trees, positional trees, etc. In this lecture, however, we shall consider trees with no special restrictions on their structure. When there is risk of confusion with a special kind of tree, we'll call these *free trees*.

1.1 Equivalent Definitions

There are remarkably many different, yet equivalent, ways to define a tree. Many of these alternate definitions are listed in the theorem below. (The proof of this theorem is tedious and given in the supplemental reading.) You can check that each alternate definition holds for the graph in Example 1, but is violated by the graphs in Examples 2 and 3.

Theorem 1 *Let $G = (V, E)$ be an undirected graph. The following statements are equivalent.*

1. *G is a tree (as defined above).*
2. *Every pair of vertices in G is connected by a unique simple path.*
3. *G is connected, but removing any edge disconnects G.*
4. *G is connected and $|E| = |V| - 1$.*
5. *G is acylic and $|E| = |V| - 1$.*
6. *G is acylic, but adding any edge creates a cycle.*

1.2 The Diameter of a Graph

Graphs are often adorned with functions defined on their vertices or edges. For example, suppose that we want to study airline routes. We might use a graph where vertices represent cities and edges represent direct flights. We could then incorporate flight times into our model by defining a function mapping each edge of the graph to the duration of the corresponding flight. This notion leads to another slew of definitions.

Definition 2 *An* edge-weighted graph *is a graph $G = (V, E)$ together with a function $w : E \longrightarrow \mathbb{R}$ that assigns a weight $w(e)$ to each edge $e \in E$.*

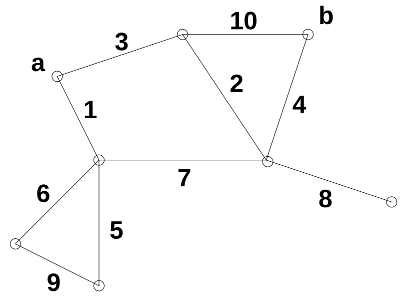
Strictly speaking, if (u, v) is an edge in the graph, then its weight should be denoted $w((u, v))$. But the double parentheses are annoying, so we'll write $w(u, v)$ instead. Also, if (u, v) is not an edge of G, then we say $w(u, v) = \infty$.

Definition 3 *In an edge-weighted graph, the* length *or* weight *of a path p consisting of edges $e_1, \ldots, e_k$ is denoted $w(p)$ and is defined by:*

$$w(p) \quad = \quad \sum_{i=1}^{k} w(e_i)$$

Definition 4 *In an edge-weighted graph, the* distance *between vertices u and v is denoted $d(u, v)$ and is defined to be the length of the shortest path from u to v.*

Example 4. Below is an example of an edge-weighted graph. The weight of each edge is noted beside it. The length of the shortest path from vertex a to vertex b is $d(a, b) = 3 + 2 + 4 = 9$. Notice that this is not the path with the fewest edges.



Distances between vertices in an edge-weighted graph obey the triangle inequality:

Fact 1 (Triangle Inequality) *If u, v, and x are vertices in an edge-weighted graph, then:*

$$d(u, v) \quad \leq \quad d(u, x) + d(x, v)$$

Intuitively, this says that the absolute shortest path from u to v is no longer than the shortest path from u to v that also visits x on the way. Remember that $d(u, v)$ is the length of the shortest path between u and v, which is not necessarily the weight of the edge (u, v). For example, in the graph above, the distance between the vertices joined by the weight 10 edge is only $2 + 4 = 6$.

Definition 5 *The* diameter *of an edge-weighted graph is the distance between the two most-distant vertices.*

As a special case, the diameter of a disconnected graph is said to be ∞. Finding the diameter of the graph in Example 4 is not so easy! Some hunting around shows that it is $6 + 1 + 3 + 2 + 8 = 20$.

1.3 Finding the Diameter of a Tree

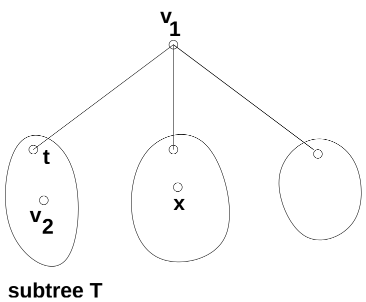
There is a simple procedure for finding the diameter of a tree.

1. Pick any vertex $v_1 \in V$.
2. Find a vertex $v_2 \in V$ that is maximally distant from v_1.
3. Find a vertex $v_3 \in V$ that is maximally distant from v_2.
4. The diameter of the graph is $d(v_2, v_3)$.

Theorem 2 *The diameter-finding procedure always gives a correct answer.*

Proof. The proof uses contradiction and the well-ordering principle. Let $G = (V, E)$ be the smallest tree (with respect to number of vertices) for which the procedure gives the wrong answer. Note that we can not have $|V| = 1$; in that case, the procedure correctly outputs 0.

Selecting a vertex $v_1 \in V$, partitions the remainder of the graph into subtrees. The situation is depicted in the figure below.



Let T denote the subtree containing vertex v_2, and let t be the vertex in T that is adjacent to v_1. Let vertices x and y be the endpoints of a true diameter; that is, choose x and y so that $d(x, y)$ is equal to the diameter of the graph.

First, we show that at least one of the vertices x and y is not in subtree T. Suppose that x and y are both in T. Then the procedure would fail on the graph consisting of only the subtree T. (If the procedure initially picks node t, then v_2 is selected as before. Since the graph is smaller, the procedure must then select some v_3' such that $d(v_2, v_3') \leq d(v_2, v_3)$. Thus, since the procedure gave a too-small answer for G, it also gives a too-small answer for the graph consisting of only subtree T.) Since G is defined to be the smallest graph for which the procedure fails, this gives a contradiction. Therefore, at least one of x and y is not in subtree T.

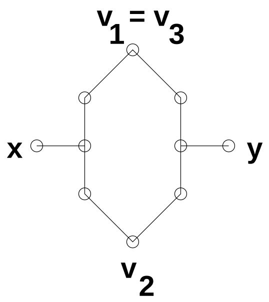
Without loss of generality, suppose that x is not in T. Then we can argue as follows:

$$\begin{aligned} d(v_2, v_3) &\geq d(x, v_2) \\ &= d(x, v_1) + d(v_1, v_2) \\ &\geq d(x, v_1) + d(v_1, y) \\ &\geq d(x, y) \end{aligned}$$

The first step uses the fact that v_3 is defined to be a vertex maximally distant from v_2. In the second step, we use the fact that there is a unique path between vertices in a tree. (See Theorem 1, part 2.) In this case, the unique path from x to v_2 must pass through v_1, since v_2 is in subtree T and x is not. The third step uses the fact that v_2 is a vertex maximally distant from v_1. The final step uses the triangle inequality.

Now we have a contradiction, because the procedure does not fail on G after all. The distance between v_2 and v_3 is as great as the distance between x and y, the endpoints of a true diameter. Therefore, our original supposition was incorrect, and so the procedure always gives the right answer. □

This procedure does not necessarily work for graphs that are not trees. In the example below, assume that all edges have weight 1. The procedure might select vertices v_1, v_2, and v_3 as shown and conclude that the diameter of the graph is 4. However, the distance between vertices x and y is 6.



2 Recursive Definition

Many sets, functions, and other mathematical objects can be defined recursively. For example, we defined the Fibonacci numbers recursively as follows:

$$
\begin{aligned}
F_0 &= 0 \\
F_1 &= 1 \\
F_n &= F_{n-1} + F_{n-2} \qquad\qquad \text{(for } n \geq 2\text{)}
\end{aligned}
$$

Similarly, the Peano axioms define the natural numbers recursively by asserting that zero is a number and that every number has a successor.

Many interesting sets can be expressed using a three-part recursive definition.

1. A *base clause* specifies a set of *base elements* that belong to S.

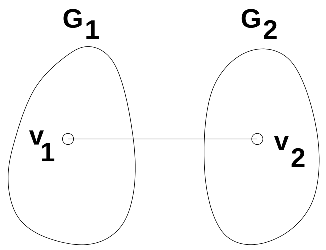
2. An *inductive clause* defines ways of combining elements of S to obtain new elements of S.

3. An *extremal clause* states that the only elements of S are those that can be produced by finitely-many applications of the preceding two clauses. The extremal clause is usually understood to be present and not stated explicitly.

As an example, we can define the set of trees recursively as follows:

1. A graph with a single vertex is a tree:

2. Suppose that $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are trees and that V_1 and V_2 are disjoint. Let v_1 be a vertex in V_1, and let v_2 be a vertex in V_2. Then $G = (V_1 \cup V_2, E_1 \cup E_2 \cup \{(v_1, v_2)\})$ is also a tree:



We can prove that this definition of a tree is equivalent to the six previously given. We won't, because it is tedious, but we could. No, really.

Sometimes one wants to prove that all the elements in an inductively-defined set have a property P. This can be done with a *structural induction* argument, which has two parts:

1. Show that all the base elements have property P.
2. Show that inductively-generated elements have property P, assuming that the elements they're generated from have the property.

These two steps suffice to show that every element in the set has property P. The reason is that any such argument can be translated into a traditional induction on the number of applications of the inductive clause. The use of structural induction is illustrated in the proof of the following mini-theorem.

Theorem 3 *Let $G = (V, E)$ be a tree. Then $|E| = |V| - 1$.*

Proof. The proof is by structural induction. Let P be the property that the number of vertices exceeds the number of edges by 1. First, suppose that G consists of a single vertex. Then $|V| = 1$ and $|E| = 0$, and so G has property P. Now suppose that $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are trees with property P. Let $G = (V, E)$ be the tree obtained from G_1 and G_2 via the inductive clause. Then we have:

$$
\begin{aligned}
|E| &= |E_1| + |E_2| + 1 \\
&= (|V_1| - 1) + (|V_2| - 1) + 1 \\
&= (|V_1| + |V_2|) - 1 \\
&= |V| - 1
\end{aligned}
$$

The first step uses the definition of G. The second uses the fact that G_1 and G_2 have property P. We simplify in the third step, and the final step again uses the definition of G. Thus tree G has property P as well. Thus, we have shown that every tree has property P by structural induction. $\square$