Introduction to Graph Theory	Instructor: Padraic Bartlett

Lecture 3: Trees!

Week 1 — *Mathcamp 2011*

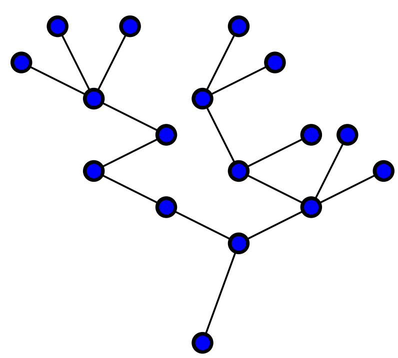
We closed our last lecture with a discussion of bipartite graphs; there, we came up with a remarkable characterization (a graph is bipartite if and only if it doesn't have a subgraph isomorphic to an odd-length cycle) for this entire family of graphs – a powerful result!

In graph theory in general, proving powerful results like the above can only happen when we're focusing on a specific family of graphs. As we established in the first lecture with the degree-sum formula, we *can* come up with properties that are true about all graphs; however, (because there are **so many** different kinds of graphs,) coming up with really interesting statements about all graphs will usually be impossible. Therefore, what we'll usually do in this course is study specific families of graphs – like, say, bipartite graphs! – and come up with specific properties for those families! In today's lecture, for example, we will turn our attention to the family of **tree** graphs, which we can classify quite thoroughly:

1 Basic Properties

Definition. If a graph G has no subgraphs isomorphic to cycles, we call G **acyclic**. A **forest** is another word for an acyclic graph; similarly, a **tree** T is a graph that's both connected and acyclic. In a tree, a **leaf** is a vertex whose degree is 1.

Example. The following graph is a tree:



Trees have a number of properties, which we quickly state here:

Proposition 1 *If G is a tree, G is bipartite.*

Proof. We proved in our last lecture that a graph is bipartite if and only if it has no subgraphs isomorphic to cycles of odd length. If G is a tree, none of its subgraphs are isomorphic to cycles of any length at all (and in specific none of the odd ones;) therefore, G is bipartite, as claimed.

Proposition 2 *Every finite tree T with at least two vertices has at least two leaves.*

Proof. Take any tree T on n vertices, and look at all of the paths in T that don't repeat any vertices. Because there are finitely many vertices and edges in T, there are finitely many such paths; therefore, there must be a **maximum length** M that these paths can reach. Let P be such a path, and let v_0, v_M be its two endpoints.

We claim that these two endpoints are both leaves. Indeed, if one of them wasn't a leaf, then there would have to be at least two edges leaving that vertex, one of which was not used by our maximal path P. If that edge went back to some other vertex in P, it would make a cycle (which our graph T doesn't have, as it's a tree); if it went to some vertex that's not in P, adding that edge and vertex to our path would a path of length $M+1$, which is strictly longer than our maximum-length path P (and is therefore impossible.) Thus, both of these endpoints are leaves.

Proposition 3 *Deleting a leaf from a tree on n vertices produces a new tree on $n-1$ vertices.*

Proof. Take any tree T on n vertices, let $l \in V(T)$ be a leaf of T, and let $u, v \in V(T)$ be a pair of distinct vertices, neither of which are L.

Because T is connected, there must be a path P from u to v: pick this path P so that it doesn't repeat any vertices or edges. (You showed that we can do this on the HW.) We know that this path cannot involve l (because if it went to l, it would have to travel along the one edge that goes to l twice); therefore, if we delete l and its edge from our graph, we still have a path from u to v. Therefore, the graph $T \setminus \{l\}$ is still connected. Because deleting an edge cannot create a cycle, this means that the graph $T \setminus \{l\}$ is a tree.

Proposition 4 ***For a graph G on n vertices, the following statements are equivalent[1]:***

1. ***G is a tree.***
2. ***G is connected and has $n-1$ edges.***
3. ***G has $n-1$ edges and no cycles.***
4. ***For every pair u, v in $V(G)$, there is exactly one path from u to v that doesn't repeat any vertices.***

Proof. HW!

2 How Many Trees Exist on n Vertices?

Given a property – say, being bipartite, or being a tree – a question we often like to ask in graph theory (and the field of combinatorics in general!) is *how many* objects there are that satisfy that property. For example, one question we might ask is the following:

[1] A series of true-false statements are called **equivalent** if one of them being true means that all of the others are true. For example, the two statements "n is odd" and "$n+1$ is even" are equivalent: whenever one of them is true, the other must be true as well.

Question 5 *How many distinct trees are there on n vertices?*

(By "distinct," we mean "distinct as graphs" – i.e. we will regard two trees as being the same if and only if they are identical, not if they're just isomorphic to each other. To denote this, we will say that we're asking for "distinct labeled trees," to emphasize that we're caring about these as graphs, not just as graphs up to isomorphism.)

There are a number of beautiful proofs that answer this question (one of which we'll see later in the spectral theory course!) We study one such proof here:

Theorem 6 *(Cayley) There are n^{n-2} distinct labeled trees on the vertex set $\{1, 2, \ldots n\}$.*

Proof. A trick we often employ in mathematics is the art of counting a quantity in two ways (like in the degree-sum proof.) This trick is how we'll prove this claim: specifically, we're going to show that there is a bijection[2] between

- the collection of trees on $\{1, 2, \ldots n\}$, and
- the sequences of length $n - 2$ of numbers from the set $\{1, 2, \ldots n\}$.

Because there are n^{n-2} such sequences (this is pretty easy to see: each entry in our sequence has n choices and we're picking $n - 2$ entries,) if we can find such a bijection, it will prove that there are n^{n-2} trees.

How can we turn a tree into a sequence $\{a_1, \ldots a_{n-2}\}$ of numbers $1 \ldots n$? The trick, here, is the following ingenious algorithm discovered by the mathematician Heinz Prüfer in 1918:

1. As input, take a tree on n vertices, with its vertices labeled with the numbers $\{1, \ldots n\}$.
2. Look at the leaves of our tree; these are all labeled with distinct numbers. Let l be the leaf of our tree with the smallest possible label, and let v be l's only neighbor.
3. Delete l from our tree, mark it as "finished," and write down v's label as the first entry in our sequence.
4. If our tree doesn't consist of a single edge, repeat this process! I.e. go to step 2, where we look at all of the leaves and pick the one with the smallest label, then to step 3, where we delete this leaf and put the label of its only neighbor as the next entry in our sequence, then return here again.
5. If our tree does consist of a single edge, then only two vertices remain (both of which are not marked "finished,") and we've created a sequence of length $n - 2$. Stop.

[2] A **bijection** between two collections A, B of objects is a map that sends each element of A to an element of B, in a way such that each element of B is matched to exactly one element of A. Intuitively, a bijection is just a way of "matching up" elements of A and B. Notice that if there's a bijection between A and B, then A and B have the same number of elements (because each element of A has a corresponding unique element in B, and vice-versa.

This process turns trees into sequences of numbers in $\{1, \dots n\}$ of length $n-2$. To show that it's a bijection, we simply need to create an **inverse** process to the Prüfer algorithm: i.e. a process that will take any sequence $(a_1, a_2, \dots a_{n-2})$ of elements from $\{1, \dots n\}$ and create a tree, such that putting that tree into the Prüfer algorithm will return the same sequence.

To do this, let $A = (a_1, a_2, \dots a_{n-2})$ be any sequence of numbers from $\{1, \dots n\}$. As well, create a list $\mathcal{E} = (e_1, \dots e_n)$ of integers, where each e_i is equal to the number of times the number i occurs in A, plus one. (The idea of this list $\mathcal{E}$ is that it is counting the number of edges that each node i in our tree will be incident with.) We then proceed by the following algorithm:

1. Let i be the first non-struck-out entry in A. Strike out the first entry from A.

2. Let j be the smallest index in $\{1, \dots n\}$ such that $e_j = 1$. Mark j as "finished," draw an edge connecting i to j, and decrement e_i, e_j both by 1. (Question: why is this operation well-defined? Prove it to yourself!)

3. If there are any non-zero indices in $\mathcal{E}$, go to 1 and repeat this process.

4. Otherwise, there are no elements left in A, and thus only two elements left in T. Connect these two elements with an edge.

To see that this process always creates a tree T, simply notice that

- the process above starts off with T as a graph where each connected component contains an unfinished vertex, and

- each stage consists of joining two unfinished vertices in distinct components and marking one vertex as finished: this decreases the number of connected components by 1 and leaves one unfinished vertex in each conected component.

- Because the process above runs for $n-1$ steps, creates $n-1$ edges in T, and we start with n connected componenets, it produces a graph with $n-1$ edges on n vertices that's connected. By our earlier proposition, this is a tree.

Finally, to prove our claim, it suffices to show that these two algorithms are inverses of each other: i.e. that taking any sequence, applying the above inverse map, and then applying the Prüfer map will result in the same sequence.

Proving this is a HW exercise! (But it's not hard: consider those "finished" labels we attached to vertices in the Prüfer algorithm and its inverse step.)