Math 3322: Graph Theory[1] Mikhail Lavrov

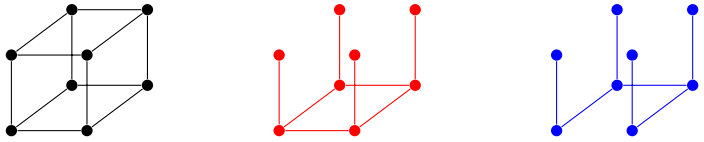
Lecture 10: Trees and spanning trees

September 18, 2023 *Kennesaw State University*

1 Spanning trees

What does it take to connect a graph?

We have seen many examples of connected graphs. For example, the cube graph (shown in the first diagram below) is connected:



But not all the edges of the cube graph are necessary to have a connected graph. For example, we can remove all edges between the four vertices in the "top half" of the cube, and the result is still connected, because those vertices can still get to each other through the bottom half. (Second diagram.)

Even that is not the best we can do. Remove any one of the edges in the bottom half, and the result is still connected: the bottom half of the cube forms a path subgraph. Now we have a connected subgraph of the cube graph that cannot lose any more edges and still remain connected. (Third diagram.)

In general, we say that a graph T is a **tree** if it is a *minimally connected* graph: T is connected, but for every edge $e \in E(T)$, the subgraph $T - e$ is no longer connected. Every edge of a tree is absolutely necessary to keep the tree connected.

The graph we drew in the third diagram above is a tree; moreover, it is what we call a spanning tree of the cube graph. A **spanning tree** of a graph G is a subgraph T which is a tree and satisfies $V(T) = V(G)$: T includes all the vertices of G.

The term **spanning subgraph** is sometimes used to mean a subgraph that has all the vertices of the original graph. Using this term, we can define a spanning tree of G as a spanning subgraph of G which is a tree.

Theorem 1.1. *A graph G is connected if and only if it has a spanning tree.*

Proof. Let G be any connected graph. To find a spanning tree T of G, we will delete edges of G one at a time until we get a tree.

[1]This document comes from the Math 3322 course webpage: http://facultyweb.kennesaw.edu/mlavrov/courses/3322-fall-2023.php

This can be done essentially any way you like. Suppose we have ended up at an intermediate graph H (some spanning subgraph of G) and H has an edge e such that $H - e$ is still connected. Then just delete edge e and keep going. (If there are many options for e, pick any of them.)

Eventually, we stop (because there are only finitely many edges to delete), and the only way this process can stop is when deleting any edge would disconnect the remaining graph. That is exactly what it means to be a tree: we have arrived at a spanning tree of G.

In the other direction, suppose G has a spanning tree T. Then any two vertices v, w of G are also vertices of T, and T is connected, so there is a $v - w$ walk in T. That walk is also a $v - w$ walk in G, because T is a subgraph of G. Therefore G is connected. □

Theorem 1.1 is a useful distinguishing property of connected graphs; it's much easier to use than the best tools we had before. Checking the definition of a connected graph boils down to checking that for any two vertices v, w, there is a $v - w$ walk: we need to find many different objects in the graph.

A spanning tree of G, on the other hand, is a *single* object which is enough to demonstrate that G is connected. All connected graphs have them—no other graphs do.

2 Special spanning trees

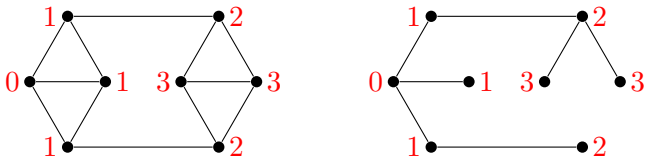
There are a few special spanning trees that are worth mentioning.

2.1 Breadth-first search trees

We have already seen an algorithm for finding distances in a connected graph. When we want to find distances from vertex v, we begin by labeling v with 0. Then, we label all its neighbors with a 1. Then, we go through all of the 1 vertices, and label all of their unlabeled neighbors with a 2. Then we repeat this process: go through all the vertices labeled i, and give all their unlabeled neighbors the label $i + 1$. This ends when we label the entire graph: the labels are distances to v.

In the course of doing this process, we can find a spanning tree of G. Whenever we label a new vertex x, we do it because it's the neighbor of an already labeled vertex y. In such a case, include edge xy in the tree T.

Here's an example. On the left is the completed distance labeling of a graph G. On the right is the spanning tree of G we get. (Note that this tree depends also on the order in which we look at vertices. Here, we looked at the top vertex with label 2 before the bottom vertex. So the top vertex got to be the one with edges to the vertices with label 3.)



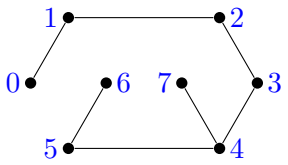
The spanning tree we get in this way is called the **breadth-first search** tree (BFS tree) of G.

2.2 Depth-first search trees

The **depth-first search** tree (DFS tree) comes from another way to explore a graph.

Starting at a vertex v, you just keep walking to new vertices for as long as you can. If you get to a vertex, and it has neighbors you haven't seen yet, you go to those vertices. If you get to a vertex and it *doesn't* have unvisited neighbors, you backtrack: retreat the way you came. You might have to backtrack several times before you get to a vertex with unvisited neighbors. If you end up backtracking all the way to v, you know you're done: there's nothing left to explore.

The DFS tree is the tree formed by all the edges you take over the course of this exploration. For example, here is one possible DFS tree of the graph we used as an example earlier:



Each vertex is labeled with the step at which it is added to the tree. In this particular example, we only had to backtrack once: to visit the vertex labeled 7.

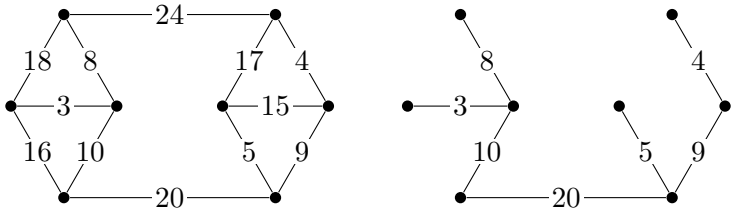
2.3 Minimum-cost spanning trees

In many applications, each edge of a graph has an associated cost. For example, maybe we're trying to build a rail network to connect the cities in a country, and the cost of an edge is the cost of building a railway between those two cities.

A minimum-cost spanning tree is the spanning tree whose edges have the least total cost. This is the cheapest way to connect the graph (any spanning graph which is not a tree will have unnecessary edges that can be removed to get something cheaper).

Many algorithms exist to find a minimum-cost spanning tree. For example, we can go from the edges in order from most expensive to cheapest, and delete whatever we can: delete every edge you come across that does not disconnect the remaining graph. This is a "greedy" algorithm: it does the best thing it sees in the moment, with no regard to how this affects future choices. But it turns out to succeed for this problem.

Here's an example of a graph with edge costs, and its only minimum-cost spanning tree. (Check to make sure that you understand the greedy algorithm above by trying it yourself, and seeing if you get the same tree.)

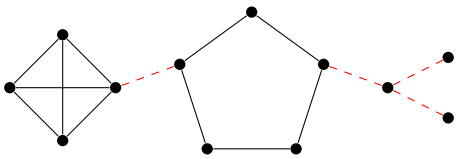


3 Bridges

Theorem 1.1 will be very useful to us—eventually. To get as much as possible out of this theorem, we need to learn more about trees. Fortunately, there are lots of things to learn.

To begin with, let's take another look at the way we found a spanning tree in the proof of Theorem 1.1. We kept deleting edges if deleting them would not disconnect the graph, until there were no more edges left that we could delete. What sort of edges do we get in this way?

In a connected graph G, an edge e is called a **bridge** if $G - e$ is not connected. For example, in the graph below, the dashed edges are the bridges:



Not all graphs have any bridges at all. However, as we delete edges of our graph, this might cause some of the remaining edges to become bridges. Eventually, we can't delete any more edges because all of the remaining edges are bridges. That's what it means to be a tree! (An equivalent phrasing of our definition is "T is a tree if all edges of T are bridges.")

So how do we tell if an edge is a bridge?

Theorem 3.1. *In a connected graph G, edge vw is a bridge if and only if it is not on any cycles.*

Proof. Suppose edge vw is part of a cycle: $(v_1, v_2, \ldots, v_k, v_1)$ where $v_i = v$ and $v_{i+1} = w$. Our basic idea: edge vw is not necessary because we can always go "the long way" around this cycle.

Since G is connected, there are walks between every pair of vertices. In all of those walks, whenever edge vw is used (whenever the walk goes $\ldots, v, w, \ldots$) we can replace it by the longer sequence $v_i, v_{i-1}, \ldots, v_1, v_k, v_{k-1}, \ldots, v_{i+1}$. (If a walk goes $\ldots, w, v, \ldots$, use this sequence in reverse.)This gives walks between every pair of vertices, none of which use edge vw, so it gives walks between every pair of vertices in $G - vw$. Therefore $G - vw$ is still connected, and vw is not a bridge.

In the other direction, suppose vw is not a bridge. Then $G - vw$ is connected, and in particular, $G - vw$ contains a $v - w$ path: $(v_0, v_1, \ldots, v_\ell)$ with $v_0 = v$ and $v_\ell = w$. Therefore G contains the cycle $(v_0, v_1, \ldots, v_\ell, v_0)$ whose last edge is the edge vw. □

4 Properties of trees

What does Theorem 3.1 tell us about trees? In a tree T, every edge is a bridge, so no edge is part of any cycles. Therefore a tree T has no cycles at all.

In fact, this is another defining property of trees. If a graph G is connected and has no cycles, then by Theorem 3.1, every edge of G must be a bridge: deleting any edge of G disconnects it. Therefore G is a tree. We have shown the following:

Proposition 4.1. *A graph T is a tree if and only if it is connected and **acyclic** (has no cycles).*

There are many equivalent properties of trees. Here is one more:

Proposition 4.2. *A graph T is a tree if and only if it is acyclic, but adding any edge would create a cycle.*

Proof. Suppose T is a tree; by Proposition 4.1, we already know it is acyclic. Suppose v, w are *not* adjacent in T; we want to show that $T + vw$ (the graph we get if we add edge vw to T) has a cycle. Well, vw cannot be a bridge of $T + vw$: deleting it gives us the connected graph T. So by Theorem 3.1, $T + vw$ has a cycle (containing vw).

Suppose T is acyclic, and adding any edge would create a cycle. We want to prove that T is connected: then we can use Proposition 4.1 and conclude that T is a tree.

Suppose v, w are vertices of T; we want to know that there is a $v - w$ walk in T. If vw is an edge, then (v, w) is such a walk. If vw is not an edge, then $T + vw$ has a cycle. That cycle must include vw (because otherwise, the cycle would have existed in T). As in the proof of Theorem 3.1, deleting vw from that cycle leaves a $v - w$ path, finishing our proof that T is connected. □

Proposition 4.2 is description of trees that, in a way, is the opposite of the definition of a tree. The definition says that T is a tree if and only if it is minimally connected: connected with as few edges as possible. Proposition 4.2 says that T is a tree if and only if it is maximally acyclic: acyclic with as many edges as possible.

5 Practice problems

1. Let's find some spanning trees in the cube graph.

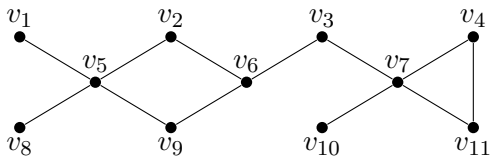
   (a) Find a breadth-first search tree of the cube graph. *(Check: your tree should have degree sequence* $3, 3, 2, 2, 1, 1, 1, 1$*.)*

   (b) Find a depth-first search tree of the cube graph. *(Check: you should backtrack either once or not at all; in the second case, your tree will be isomorphic to the path graph* P_8*.)*

   (c) Recall that the cube graph has vertices $000, 001, 010, \ldots, 111$ where two vertices are adjacent if they differ only in one position.

   Suppose that an edge whose endpoints differ in the i^{th} position has cost i. Find a minimum-cost spanning tree. *(Check: your total cost should be* 11*.)*

2. Let G be the graph shown below:



   (a) Identify all the bridges in G. *(Check: there should be* 5 *bridges.)*

   (b) Find all the possible spanning trees of G. *(How does the answer to (a) help here?)*

3. Prove that *every* n-vertex graph with the degree sequence $n-1, 1, 1, \ldots, 1, 1$ is a tree. What does such a graph look like?

4. Find a connected 3-regular graph G which has a bridge!

   (Hint: it's not as easy as it seems; you'll need at least 10 *vertices. If* e *is a bridge, think about the degree sequences of the components of* $G - e$*.)*

5. It's even harder to find a 4-regular graph which has a bridge.

   (a) Let G be a 4-regular graph in which edge vw is a bridge. Then $G - vw$ should have two components: one containing v, and one containing w. Describe the degree sequences of each component.

   (b) Conclude that this can't happen: a 4-regular graph cannot have a bridge.

6. We have described how to get the breadth-first search tree of G, but we did not prove that it is actually a tree.

   (a) Prove that as we build up the BFS tree, whenever we add an edge xy, this cannot create a cycle. *(It follows that the BFS tree is acyclic.)*

   (b) Prove that the BFS tree (of a connected graph) is connected. Conclude that it's a tree!

7. Prove that T is a tree if and only if between any two vertices of G, there is **exactly one** path. This is another of the many characterizations of trees!