## DRAFT

Last time we discussed graphs. Today we continue this discussion, and focus on graphs of a special kind, trees. Afterwards, we start discussing applications of graphs in computer science. We talk about graph coloring today, and we'll mention finite state machines, finite automata, and regular expressions in the next lecture.

### 20.1 Trees

Before we describe trees, we revisit the notion of a path.
We need the concept of a simple path. A simple path is just like any other path, except every edge appears on it at most once, and the only vertex that can be visited twice is the start vertex, but only if the second visit is at the very end of the path. If the start vertex is the same as the end vertex, the path is called a simple cycle.
Example 20.1: Consider the graph in Figure 20.1. We show some simple paths as well as some paths that are not simple.

(a) A simple path from 0 to 3 .

(c) A path from 0 to 1 that is not simple. The last vertex is a previouslyvisited vertex, but not the start vertex.

(b) A simple cycle that starts and ends at 1 .

(d) A path from 0 to 3 that is not simple. The vertex 1 appears twice. The edge taken after the first visit to 1 is $(1,2)$.

Figure 20.1: Examples illustrating the definition of a simple path. The arrows give the direction each edge is traversed in.

In a simple graph, there is at most one edge between any one pair of vertices. Thus, we can list the vertices on the path instead of listing the edges. If the path goes from vertex $u$ to vertex $v$ in one step, there is a unique edge this path can follow to achieve that. Hence, we can recover the sequence of edges from the sequence of vertices visited on the path.

For example, we could describe the path in Figure 20.1a using the sequence 0, 1, 2, 5, 4, 3 instead of the sequence $\{0,1\},\{1,2\},\{2,5\},\{5,4\},\{4,3\}$.

And now we are ready to define what a tree is.
Definition 20.1. $A$ tree is a connected graph without simple cycles.
We show three graphs in Figure 20.2. Only the first graph is a tree. The remaining two are not. One is not connected and one is not acyclic.

(a) A tree

(b) Not a tree because the graph has two connected components.

(c) Not a tree because there is a simple cycle in this graph. The edges on that cycles are highlighted using thick blue lines.

Figure 20.2: Examples illustrating the definition of a tree.
Now the graph in Figure 20.2a doesn't exactly look like a tree. So let's redraw it to make it look more like a tree. Pick an arbitrary node (in this case we pick the only one of degree 4), and consider it the root of the tree. Draw its neighbors above it, then draw the neighbors of the neighbors (except for the root which has already been drawn) one more level above, and keep going until the entire graph is drawn. We show this process in Figure 20.3, with the graph taking a tree form in Figure 20.3b.

Every vertex of degree 1 in a tree is called a leaf. For example, the leaves in Figure 20.3b are vertices $0,4,6,8$, and 9 .

We declared the vertex 3 the root of the tree in Figure 20.3b, but that is only a product of how we drew the tree. To give the root some significance, we make all the edges directed, so that they all point away from the root. The resulting tree, called a rooted tree, is shown in Figure 20.3c. Now the concept of a root makes sense. It is the only vertex with in-degree zero. All the other vertices have in-degree 1.


Figure 20.3: Pick node 3 as the root. Then draw all its neighbors in the next level. One level above, draw the neighbors of the root's neighbors. Keep going in this fashion until the entire picture is drawn.

### 20.1.1 Properties of Tress

Given their special structure, we can prove more facts about trees. In particular, we describe some relationships between the vertex and edge sets of trees.

Proposition 20.2. Let $T=(V, E)$ be a tree. Then the following are true.
(i) If $|V| \geq 2, T$ has at least two leaves.
(ii) There is exactly one simple path between any two vertices.
(iii) If $V$ is nonempty, $|V|=|E|+1$.

We make some remarks about individual parts of Proposition 20.2, and then prove the individual parts.

If there is just one vertex in the graph, we consider the graph a tree if there are no edges, and we call the only vertex a leaf in that case. If there are at least two vertices, the characterization of the number of leaves from part (i) applies.

Proof of part (i) of Proposition 20.2. Consider a simple path $P$ of maximum length in the tree $T$. Say $P$ starts at $u$, ends in $v$, and the vertices on the path are $u=v_{0}, v_{1}, \ldots, v_{r-1}, v_{r}=v$. We claim that $u$ and $v$ have degree 1 , which means they are leaves.

Suppose $u$ does not have degree 1. Then there is another edge besides $\left\{u, u_{1}\right\}$ that's incident on $u$, say $\{w, u\}$. This edge does not appear on the path $P$ because $P$ is simple. There are two cases to consider, and each of them leads to a contradiction.

Case 1: $w$ does not appear on the path $P$. Then the path $w, u, v_{1}, \ldots, v_{r}, v$ is a simple path longer than $P$, which is a contradiction because $P$ is the longest simple path in $T$.

Case 2: $w$ appears on the path $P$, say $w=v_{i}$. Then the path $w=v_{i}, v_{i+1}, \ldots, v_{r}, v_{i}$ is a simple cycle in $T$. But since $T$ doesn't contain any simple cycles, this is a contradiction.

Part (ii) of Proposition 20.2 actually gives us an equivalent definition of a tree. That is, a connected graph with exactly one simple path between any two vertices is a tree. We leave the proof of this fact as an exercise to the reader.

Because $T$ is connected, there is a path between any two vertices in $T$. We first show that this implies the existence of a simple path between any two vertices in $T$. For example, consider the path in Figure 20.1d. We could turn it into a simple path by omitting the cycle $1,2,5,4,1$, and just going to 3 the moment we reach 1 for the first time. In fact, this strategy works in general. If a cycle is part of a path from $u$ to $v$, the part of the path without the cycle is a shorter path from $u$ to $v$. After removing some number of cycles in this fashion, we end with a simple path.

Lemma 20.3. Let $G=(V, E)$ be a graph. If there is a path from vertex $u$ to vertex $v$, there is also a simple path from vertex $u$ to vertex $v$.

Proof. We give a proof by induction on the path length.
In the base case, consider a path of length zero. This has only one vertex on it, namely the starting vertex. Thus, no vertices repeat, and the path is simple.

Now consider a path of length $n+1$ from $u$ to $v$ in some graph. Suppose that for every $x, y \in V$ if there is a path from $x$ to $y$ of length $n$ or less, then there is also a simple path from $x$ to $y$.

Let $e_{1}, e_{2}, \ldots, e_{n+1}$ be the edges on the path from $u$ to $v$. If this path is simple, there is nothing to prove. Otherwise there is some vertex $w$ that appears twice so that it is the endpoint of $e_{i}$ and $e_{j}$ with $i<j$. Then the edges $e_{i+1}, e_{i+1}, \ldots, e_{j}$ form a cycle that starts and ends at
$w$. Moreover, the path $e_{1}, \ldots, e_{i}$ is a path from $u$ to $w$ and $e_{j+1}, \ldots, e_{n+1}$ is a path from $w$ to $v$. Then the sequence of edges $e_{1}, \ldots, e_{i}, e_{j+1}, \ldots, e_{n+1}$ is a path from $u$ to $v$. The path has length $i+(n+1)-(j+1)+1=i+n-j+1$. Now $j \geq i+1$, so the length is at most $i+n-(i+1)+1=n$. The induction hypothesis then implies there is a simple path from $u$ to $v$.

Proof of part (ii) of Proposition 20.2. Consider any two vertices $u$ and $v$ in $T$. There is a path from $u$ to $v$ in $T$ because $T$ is connected. Then there is a simple path from $u$ to $v$ by Lemma 20.3.

What remains to show is that there are not multiple simple paths from $u$ to $v$. For the purpose of contradiction, assume there are different simple paths, $P$ and $Q$, from $u$ to $v$. The path $P$ goes through vertices $u=v_{0}, v_{1}, \ldots, v_{r-1}, v_{r}=v$. The path $Q$ starts at $v$, and at some point has to deviate from $P$. Say that at $v_{i}$ for some $i<r, Q$ goes to $w_{1} \neq v_{i+1}$ next. At some point $Q$, has to rejoin $P$ because both $P$ and $Q$ end at $v$. Say that $Q$ continues from $v_{i}$ by visiting vertices $w_{1}, w_{2}, \ldots, w_{s}$ that don't appear on $P$, and rejoins $P$ after $w_{s}$ by going to $v_{j}$ with $0 \leq j \leq r$ and $i \neq j$ (the inequality holds because otherwise $Q$ would not be a simple path). We show that the vertex $v_{j}$ is involved in a simple cycle, which is a contradiction. There are two cases to consider.

Case 1: $j<i$. Consider the path from $v_{j}$ to $v_{i}$ using $Q$ and then the path from $v_{i}$ to $v_{j}$ using $P$. Observe that $w_{k}$ does not appear on the path $P$ for any $k \in\{1, \ldots, s\}$ by assumption. Thus, the vertices $v_{j}, v_{j+1}, \ldots, v_{i}, w_{1}, \ldots, w_{s}$ are all distinct, and are, therefore, part of the simple cycle $v_{j}, v_{j+1}, \ldots, v_{i}, w_{1}, \ldots, w_{s}, v_{j}$.

Case 2: $j>i$. Consider the path from $v_{i}$ to $v_{j}$ using $P$ and then the path from $v_{j}$ to $v_{i}$ obtained by following $Q$ backwards from $v_{j}$ (which we can do because edges don't have direction). Observe that $w_{k}$ does not appear on the path $P$ for any $k \in\{1, \ldots, s\}$ by assumption. Thus, the vertices $v_{j}, v_{j+1}, \ldots, v_{i}, w_{1}, \ldots, w_{s}$ are all disjoint vertices, and are part of the simple cycle $v_{i}, v_{i+1}, \ldots, v_{j}, w_{s}, \ldots, w_{1}, v_{i}$.

Let us stress again that there are two common strategies for proving a fact about graphs by induction. Either induct on the number of vertices or the number of edges. We choose the former for the next proof.

Proof of (iii) of Proposition 20.2. We give a proof by induction on the number of vertices.
For the base case, consider a tree with one vertex. This tree has no edges because the only possible edge would be a self-loop, and this edge would form a simple cycle. Thus, in the base case, we have $|V|=1$ and $|E|=0$, so $|V|=|E|+1$ holds.

Now consider a tree with vertex set $V$, and assume that every tree with $|V|-1$ vertices has $|V|-2$ edges. Note that $|V| \geq 2$ in this case, so $T$ has a vertex of degree 1 by part (i) of Proposition 20.2. Call this vertex $v$.

Consider the subgraph $T^{\prime}=\left(V^{\prime}, E^{\prime}\right)$ of $T$ obtained from $T$ by removing $v$ and its incident edge. Suppose that $u$ is the vertex connected to $v$ by this edge. The situation is shown in Figure 20.4.


Figure 20.4: The tree $T$ in the proof of part (iii) of Proposition 20.2.
We claim that $T^{\prime}$ is a tree. First, $T^{\prime}$ doesn't contain any simple cycles because it's a subgraph of $T$, which does not have any simple cycles. Second, consider any two vertices $x, y \in V^{\prime}$. Since $x, y \in V$, there is a simple path from $x$ to $y$ in $T$ by part (ii) of Proposition 20.2. We claim that
this path consists only of vertices in $V^{\prime}$. If not, it would have to contain $v$. But the only way to get to $v$ is from $u$, and the only way to continue the path from $v$ is to go back to $u$. But this would traverse the edge $\{u, v\}$ twice, which would contradict the fact that the path is simple.

We have shown that $T^{\prime}$ is connected and doesn't have any simple cycles, so $T^{\prime}$ is a tree. Moreover, it consists of $\left|V^{\prime}\right|=|V|-1$ vertices, so $\left|V^{\prime}\right|=\left|E^{\prime}\right|+1$ by the induction hypothesis. Since $\left|V^{\prime}\right|=|V|-1$ and $\left|E^{\prime}\right|=|E|-1$, we also get that $|V|=\left|V^{\prime}\right|+1=|E|^{\prime}+1+1=|E|+1$, which completes the proof.

We mentioned that part (ii) of Proposition 20.2 gave an equivalent definition of a tree. You may ask whether any graph satisfying part (i) or part (iii) is a tree. The answer is no. For example, the graph in Figure 20.5a has two vertices of degree 1 but also contains a simple cycle, and the graph in Figure 20.5b satisfies the conclusion of (iii), but is not connected. In fact, any graph $G=(V, E)$ with $|V|=|E|+1$ that is not a tree is disconnected and contains a simple cycle.


Figure 20.5: Parts (i) and (iii) of Proposition 20.2 are not equivalent definitions of a tree.

### 20.1.2 Spanning Trees

Suppose Madison gets hit by a giant blizzard. The road cleaning crew needs to decide which roads to plow first so that Madisonians can get from every intersection to any other intersection in town. (We assume that they can walk through the snow to the nearest intersection and can reach their actual destination by walking through the snow from the intersection nearest to it.)

Model the streets of Madison as a graph $G$. Intersections are vertices, and two vertices $u$ and $v$ are connected by an edge if there is a direct road segment between the intersections that correspond to $u$ and $v$ (that is, there are no other intersections on the way between those two intersections).

Now consider any connected subgraph $C$ of $G$ consisting of all vertices of $G$. If the road cleaning crew plows all the road segments corresponding to edges in $C$, they will have accomplished their goal. Now consider the situation when $C$ is a tree. This tree consists of all vertices of $G$. Such a tree is called a spanning tree of $G$. The road cleaning crew can just find a spanning tree of $G$ and plow the road segments corresponding to the edges in that spanning tree.

So, why should the road cleaning crew be able to find a spanning tree in the first place? The key observation is that the graph $G$ is connected. If it were not connected, some Madisonians would be cut off from some parts of Madison, which would be a rather sorry state of affairs. Let's start with a formal definition of a spanning tree, and then prove a theorem that tells us the road cleaning crew can find a spanning tree.

Definition 20.4. Let $G=(V, E)$ be a graph, and let $T=\left(V, E^{\prime}\right)$ be a connected subgraph of $G$. If $T$ is a tree, we call it a spanning tree of $G$.

Theorem 20.5. Every connected graph $G$ has a spanning tree.

Proof. Start with the graph $T=G$. Now keep removing edges from $T$ until it becomes a tree.
If $T$ is a tree, it is a spanning tree of $G$. Otherwise $T$ has a simple cycle in it. Pick any edge on that cycle, and remove it from $T$ to obtain a graph $T^{\prime}$. We claim that $T^{\prime}$ is connected. This follows by an argument in the proof of Euler's formula given in the last lecture. Also, $T^{\prime}$ still consists of all vertices of $G$. Now replace $T$ with $T^{\prime}$, and look for another edge to remove from $T$.

The process of removing edges stops when $T$ has $|V|-1$ edges by part (ii) of Proposition 20.2. At that point, $T$ is a spanning tree because it consists of all vertices of $G$ and is a tree.

If a graph $G$ is not connected, it doesn't have a spanning tree because there is a pair of vertices in $G$ without a path between them. No subgraph of $G$ can contain a path between those two vertices either, so $G$ does not have a spanning tree.

The best we can hope for if $G$ is not connected is to find a spanning tree for each connected component of $G$. This is possible by the theorem we just proved. The collection of those spanning trees is called a spanning forest of $G$.

### 20.2 Applications of Graphs

We have already seen some applications of graphs. For example, we used topological sorting to come up with a valid procedure for getting dressed in the morning, and we used spanning trees to help the Madison street cleaning crew rescue the city after a major blizzard. Let's now discuss some computer science oriented applications of graphs.

### 20.2.1 Graph Coloring

Every program written in a high-level language such as C needs to be compiled down to something a computer can run on its CPU. There may be many variables in a program, but the CPU can only store a constant number of them in its registers. One problem the compiler faces is to allocate a register on the CPU for each variable. It can reuse registers if the lifetimes (or scopes if you want) of two variables do not overlap. That is, if the lifetimes of two variables overlap, those two variables need to be stored in different registers. But if the lifetimes don't overlap, the compiler can use the same register for the variable that just started existing and the variable that already ceased to exist. If the compiler fails to allocate registers in the fashion we just described, it will be necessary to copy some data back and forth between the main memory (RAM) and the CPU's registers, which is going to slow down the execution of the program.

Let's model the situation the compiler is facing as a graph. There is a vertex corresponding to each variable, and two vertices are connected with an edge if their corresponding variables' lifetimes overlap. Our goal is to label each vertex with a name of a register so that no pair of vertices connected by an edge has the same label.

This is exactly the problem we face in graph coloring. There, we are given a graph, and our goal is to color its vertices so that if two vertices are connected by an edge, they don't have the same color. Of course, the goal is to use as few colors as possible. So just think of register names as colors, and register allocation becomes graph coloring.

Definition 20.6. Let $G=(V, E)$ be a graph, and $C$ a set of "colors" with $|C|=k$. A total function $f: V \rightarrow C$ is a $k$-coloring of $G$ if $(\forall u, v \in V)\{u, v\} \in E \Rightarrow f(u) \neq f(v)$.

Definition 20.7. The chromatic number of a graph $G$, denoted $\chi(G)$, is the smallest number $k$ for which $G$ has a $k$-coloring.

We characterize graphs with low chromatic numbers (up to 2), and discuss some other results about the chromatic number of various graphs.

First, if $\chi(G)=0$, we are not allowed to use a single color to color vertices of $G$. But every vertex must have some color, so the only graph that has chromatic number zero is the empty graph (a graph with zero vertices and no edges).

Now suppose $\chi(G) \leq 1$. Since we are allowed to use only one color, there cannot be any edges in the graph, for if there were an edge, both of its endpoints would be colored the same.

The situation becomes more interesting when $\chi(G) \leq 2$. First let's look at some incorrect characterizations of graphs whose chromatic number is at most 2. These incorrect characterizations come from various attempts to extend the characterization of graphs with chromatic number 1. A graph with chromatic number 1 has no edges, so one would be tempted to say that a graph with chromatic number 2 has at most 1 edge. Also, every vertex in a graph with chromatic number 1 has degree 0 , so we may think that all vertices in a graph with chromatic number 2 must have degree at most 1. Neither of those two claims gives a full characterization of graphs with chromatic number 2. For example, the graph in Figure 20.6 has more than one edge, and also contains a vertex of degree more than 1.


Figure 20.6: A 2-colorable graph. The vertex in the middle is colored white and all the other vertices are colored black. No edge has the same color on both ends, so the coloring shown here is a valid 2 -coloring.

Suppose we have two colors, red and green, at our disposal. Each vertex gets one of those colors. Regardless of whether the color assignment is a coloring, it induces a partition of the vertex set. One partition class is the set of vertices colored red, and the other partition class is the set of vertices colored green. If the color assignment is actually a 2 -coloring, there are no edges between a pair of vertices colored by the same color. But then the graph is bipartite. Let $V_{\text {left }}$ be the set of red vertices, and $V_{\text {right }}$ be the set of green vertices. Since the coloring is valid, there are no edges that are subsets of $V_{\text {left }}$ or $V_{\text {right }}$.

Conversely, if a graph is bipartite, it is 2-colorable. Just color all vertices in $V_{\text {left }}$ red and all vertices in $V_{\text {right }}$ green. Since all edges involve one vertex of $V_{\text {left }}$ and one vertex of $V_{\text {right }}$, their endpoints have different colors. Thus, we have proved the following theorem.

Theorem 20.8. A graph is 2 -colorable if and only if it is bipartite.
There is a more general correspondence between a $k$-coloring of a graph $G$ and a partition of its vertex set, $V$, into $k$ subsets $V_{1}, \ldots, V_{k}$ such that no edge is a subset of $V_{i}$ for any $i \in\{1, \ldots, k\}$ (we call such a graph $k$-partite).

If the graph has a $k$-coloring, we can define partition classes by grouping together all vertices with the same color, i.e., $V_{c}=\{v \in V \mid v$ is colored with color $c$ in the coloring $\}$. Note that no edge in $G$ can be a subset of $V_{c}$ because that would make both of its endpoints have the same color.

Conversely, if we have a partition of $V$ into sets $V_{1}$ through $V_{k}$, we can color all vertices in $V_{1}$ with one color, all vertices in $V_{2}$ with another color, and so on.

The case with $k=2$ is interesting because we can easily tell whether a graph is bipartite. We have no good characterization of $k$-partite graphs for $k \geq 3$.

There is no good characterization of graphs with chromatic number at most 3. In fact, a good characterization that is computable by an efficient algorithm would be a major breakthrough. The problem of deciding whether a graph is 3 -colorable is equivalent to the satisfiability problem we mentioned in Lecture 2. Thus, if we could efficiently decide whether a graph is 3 -colorable, we could efficiently decide satisfiability and thousands of other problems that have been proved to be equivalent to satisfiability. We do not prove the equivalence of deciding 3 -colorability and the satisfiability problem in this course. This result is proved in most introductory courses on algorithms or theory of computation.

There is also no good characterization of graphs with chromatic number at most 4, but an important class of graphs has chromatic number at most 4.

Theorem 20.9 (Four color theorem). If $G$ is a planar graph, $\chi(G) \leq 4$.
The original proof of the four color theorem has an interesting place in history because it was done by a computer. The authors of the proof, Appel and Haken, designed a program that generated a list of cases such that every planar graph falls in at least one of them. The program then verified for each case that it was four-colorable. Of course, since the proof was done by a computer, it is necessary to prove correctness of the program. Since then, the proof has been simplified. In particular, the number of cases was reduced and it is now possible, but still tedious, to verify the entire proof by hand.

The four color theorem has an interesting consequence. Consider a map in which every country is colored so that two countries that share a border have different colors. Note that we can draw the map in a plane. So place a vertex in each country, and connect vertices corresponding to two countries that share a border with edges (here it is important that they share a border and not just one point in the corner). This gives us a planar graph. The four color theorem then tells us we can color the resulting graph with four or fewer colors. Now just color a country on the map with the color of its corresponding vertex in the graph to get the desired coloring of the map. We illustrate this process in Figure 20.7.


Figure 20.7: Four-coloring a map

