[SQUEAKING]

[RUSTLING]

[CLICKING]

ERIK DEMAINE: Welcome back to 6.1200. I hope you all had a good spring break. Today, this week, we continue our segment on graph theory. I'm going to do more graphs. Today is undirected graphs. And next class will be directed graphs.

But remember, an undirected graph, or the full mouthful is simple undirected graph is what we defined a couple of lectures ago. We have a set of vertices, V. We have a set of edges, E. Vertices can be anything, but the edges have to be pairs of vertices. And those vertices have to be distinct. So the set of edges is always a subset of this. This would be the maximum possible set of edges. You have two-item set u, v for all vertices u and v, where u does not equal v.

So today I'm just going to remove-- omit these adjectives and just say graph. But whenever I say graph today, I really mean simple undirected graph. And we're going to walk around in graphs today. So let's define a notion of walk.

This is going to be similar to an execution in state machines. Next lecture will be even more like that. But for today a walk is a sequence of vertices. I'm going to write that sequence with arrows, but it's really just a list. It's a finite sequence, let's say, where each consecutive vi to vi plus 1 is an edge in the graph. So this is like a valid transition in the state machine world. So this is like i equals 0, 1 up to k minus 1 is when that needs to apply.

So if you have-- let me draw a graph here. It's one of the first graphs I ever drew in my life. I still remember it for reasons we will talk about in a moment. Here's a graph with 5 vertices, a bunch of edges. And let me get a color. So for example, this is a path, which is-- sorry-- is a walk, which I would write a to b to d to e to b back to d up to c. That's writing it with commas. You can also write it with arrows. It's the same thing. So that's an example of a walk.

And for some terminology-- it would be helpful-- we're going to call this a walk from v0 to vk. V0 is just the first vertex in the sequence. vk is the last sequence. But I'm defining what from and to means. And we'll define the length of this path to equal k. k is the number of edges in this sequence, not the number of vertices. The number of edges is one fewer than the number of vertices. So in this example, we have 1, 2, 3, 4, 5, 6 edges. So this is a length 6 walk.

In particular, we allow length to equal 0. What does a length zero walk look like? Well, it's a vertex, and that's it. You just stay there. So c by itself is a zero length walk. e by itself is a zero length walk. Not so exciting yet.

Let's define a related notion, which is path. And this is a special kind of walk with no repeated vertices or edges. So in particular, not this path-- this walk is not a path, because it visits some vertices multiple times. It visits b twice. You can see that in this sequence. It also visits some edges multiple times. b to d is repeated.

So if I redraw this graph, an example of a path between the same two vertices would be this one. This is still starting at a and ending at c. But now it's shorter, and it doesn't repeat any vertices.

In particular, with walks, if you don't repeat a vertex, you probably also don't repeat an edge. But I think it's helpful here to highlight we really don't want to repeat vertices or edges. This will get a little more subtle when we talk about closed walks in a little bit.

So why might we care about walking in a graph? Lots of practical motivations. Every time you, I don't know, look up something in Google Maps, you're finding a path from a to b. Maybe you want the shortest path, or the least energy path, or so on. So we care about paths a lot. When we're finding routes in transportation networks, we care about them when we're finding routes on the internet or on a computer network. In both those cases, the vertices are places you can be, and the edges are the routes, like the streets you can take, for example.

In a social network, if the vertices are people and the edges are friendships, then you might care about degrees of separation between people, like a famous experiment where they asked people to send physical letters to people they knew on a first name basis. This is like in the '60s-- Milgram. And they found that the distance between two people is only about six. These days it's about five. We're getting more connected, despite more people. Anyway, those are some motivations.

Also, in things like the 8 puzzle that we talked about some time ago, you can think of each of these configurations as vertices in a graph. And the edges are transitions. And because every time I make a move, I can also unmake that move. This is an undirected graph. Everywhere I can go, I can immediately go back.

And you might care about, what does this graph look like? It's big. It's got like 100,000 vertices. So I can't draw it here. But you can explore it by moving around. Can you get everywhere? That's a good question. And that's the next topic.

Connectivity is what this lecture is mostly about. So let's talk about connectivity. So we'll call two vertices connected if there's a walk between them.

I'd like to say there's a walk between them, which is not caring about which one you start at and which you end at. But the way we define walk is it starts at one place and it goes to the other. Of course, if v0 to vk is a walk, then also vk backwards to v0 is also a walk. If I flip this sequence, if I reverse it, it's still a walk, because edges are sets. They don't care about which order you follow them in. So if you can go from vi to vi plus 1, you can go from vi plus 1 to vi in an undirected graph.

So this is equivalent to there existing a walk from v to u. It doesn't matter which one you put on the left and which you put on the right. These are the same thing. Technically, this is a theorem. You have to prove it. And the proof is just reverse the walk.

If I have a walk from u to v, I can reverse it. And I get a walk from v to u. If I have a walk from v to u, I can reverse it and get a walk from u to v. So in either direction, these two things are equivalent. And so we just say there's a walk between u and v, or we say u and v are connected, or v and u are connected-- all the same thing.

There are some other equivalent properties to this. So I want to write some more. The other big one is there exists a path from v to u. This is perhaps a good moment to mention-- so we're using the words walk and path here in one consistent way throughout this class.

But there are other classes, such as the very next class, 6.121, where this is called path, and this is called simple path. So just a warning-- there's two sets of terminologies for whether a walk is allowed to-- walks can always repeat. Vertices, paths, sometimes they can. Sometimes they can't. And in this class, they cannot. So we're going to use walk to mean the general thing, and path to mean the specific thing where you don't repeat any vertices or edges.

So I claim there's a walk from u to v if and only if there's a path from u to v. Let's prove this. So remember, how do we prove that two things are equivalent? How do we prove an if and only if? We do it in two parts. We prove left to right and right to left. So I'm going to do right to left first, because this is the easy one. Why is it, I ask you, if I have a path from u to v, is there necessarily a walk from u to v? Any suggestions? Yeah.

AUDIENCE: By definition, a path is a walk.

ERIK DEMAINE: A path is a walk. A path is a special kind of walk. So if I have a path, then definitely I have a walk, because that path is the walk. So I would usually write a path is a kind of walk. So that's the easy direction.

The more impressive thing is that if there's a walk, then there's actually-- not only is there a walk, but there's a walk that doesn't repeat any vertices or edges. This is trickier. And I'm going to tell you how to find it, because it needs a nifty idea.

But first, we write the template, which is we want to prove that if there's a walk from u to v, then there's a path. So let's assume there's a walk from u to v. And let's give it-- I'm not going to give it a name yet. So there exists a walk. There might be many walks. I want to try to choose not necessarily a particular one, but a more restricted type of walk that's possible, because I know that these at least exist.

So I want to take a shortest walk-- shortest in terms of length, so minimum possible length. Among all walks, I want one of them that minimizes the length of the walk. We know there exists some walk. So there is some solution here because you can't go below a walk of length zero. There's some shortest.

And this one I'm going to give a name. Let's say it starts at u, which is also called v0. Then it goes to v1, then it goes to v2, and so on, until it reaches vk, which equals v. So it's a walk from u to v, using some number of edges in between. But now I know it's shortest.

So I claim that this is a path from u to v. I claim that there are no repeated vertices or edges in this sequence. In particular, I claim that there are no repeated vertices. So let's prove the claim, so maybe as a picture first.

So suppose that-- let's do it by contradiction. Let's suppose that the path does repeat a vertex. So it starts at u, and then it goes-- which is v0, then it goes to v1 and so on. Maybe at some point after going for a while, it comes back to a vertex it's been to before proceeding to vk over here, which is v. This is what a walk looks like that repeats at least one vertex. I claim this is the general picture, but you can write it down in words.

Then the idea is, well, if the walk did this-- maybe I should draw this as a continuous path in red. If the walk did this, where it repeats this vertex twice, why not just do this? That's a shorter path. And that contradicts that we started with the shortest path. If this path is strictly shorter than this one, that's a contradiction. Write that down a little more carefully.

So for contradiction, let's say that vi equals vj for some i less than j. So this would be both vi and vj are the same vertex in this sequence. The vertex gets repeated. That's the general situation. I can assume i is less than j, just by relabeling i and j, swapping i and j if I need to.

And now the idea is that v0 up to vi, which equals vj, and then going directly to vj plus 1, vj plus 2, and so on, up to vk is a shorter walk between the same two vertices, still from u to v. And so that is a contradiction. And so that ends the proof of the claim. And that ends the proof of this claim, that if there's a walk, I can turn it into a path. I take the shortest walk. And it turns out that must be a path-- proof by contradiction.

So a bunch of work there. But now, for connectivity's sake, we can use walk when it's convenient. We can use path when it's convenient. For connectivity, these two notions are interchangeable. We can use whatever is nicest.

Let me mention a couple other fun properties of connectivity over here. So this one, walk from u to v versus v to u is a property we call symmetry. And there are two other properties that are nice to know. One is called reflexivity, or this is the reflexive property. Every vertex is connected to itself. Why? Because we allowed zero length walks and paths. Starting at v and doing nothing, that is a valid walk. So that's a walk from v to v.

And another useful fact is transitivity. Connectedness is a transitive relation. If we have u and v are connected and we have v and w are connected, then we know u and w are connected. Why? This is a particularly nice proof by picture. Please don't think that proof by picture is a valid proof method, but I'll tell you the words in a second.

Suppose we have vertices u, v, and w. And there's some path from u to v. And there's some path from v to w. Well, then there's a path from u to w. You just concatenate the paths, or concatenate the walks. At this point, we could assume we had paths initially, but when we glue them together, it won't be a path anymore necessarily. It might be a walk. There might be some repetitions here and here. But for the easy proof, just take two walks. Concatenate them. It's still a walk. Therefore, u and w are connected.

Next week we'll talk about these properties-- symmetry, reflexivity, and transitivity-- even more, so this is kind of a foreshadowing of that. But it lets us define some cool concepts about connectivity. So first, we're going to call a graph connected if u and v are connected for all vertices u and v.

So we sometimes call this pairwise connected. Pairwise is a useful notion when you have something that only makes sense on pairs of elements, but you want it to apply to the whole structure, in this case, the whole graph, not just pairs of vertices. Then you just put a for all over the two things. And that would be pairwise connected.

But it really doesn't matter how you-- connected is kind of a robust notion because of things like transitivity. In an undirected graph, it's the same to say that every pair of vertices is connected, as it is to say that there's some vertex that every other vertex is connected to because of transitivity. We know if you can get from a to b and from b to c, then you can also get from a to c. So this is why connectivity is the right notion. Even more interesting is a notion called connected components.

Maybe let me draw an example before I define this carefully. So this graph, for example, is connected. From every vertex you can get to any other. Let's draw a disconnected graph. I should have it over here somewhere. Let's say I've got this guy here, and maybe just a vertex and some edge, a, b, c, d, e, f, g.

So this is one graph. It has six vertices, four edges. And it's, obviously, not connected. There's no walk, for example, from here to here, or from here to there, b to f, c to e, whatever. But it's still possible to get from a to c, a to b, a to d. So a can get to everyone in this chunk. And that chunk we call a connected component. So in this graph, there are three connected components. These are the mutually reachable regions of the graph.

So to define that precisely, I'm going to introduce a new term, which is induced subgraph. So what does induced subgraph mean? It's a notion. Given one graph, it defines a graph that lives inside the original graph. It's a subset of the vertices and edges. That's what subgraph means.

And it's defined by a set of vertices. So it's the subgraph induced by a particular set of vertices. In this case, the vertex set is all the vertices connected to v. This is a nice shorthand for saying, give me all the u, where u and v are connected. And as we said, it doesn't matter whether I say u and v or v and u-- same thing.

And the edge set is determined by this word induced. When I say induced, I mean I just specify the vertices, and then I want all the edges that could possibly make sense. So in other words, I want to take all of the edges u, v that were in my original edge set with the property that both u and v are in my new vertex set.

So for example, in this graph, if I take the subgraph induced by a, b, c, d, that means I take these four vertices and none of the others. And I take all the edges that are among those four vertices. So that would be three of them. These three edges-- a to b, b to c, and b to d-- are the three edges in my induced subgraph a, b, c, d.

If I just take the subset f and g, these are all the vertices connected to f, say. Then my subgraph will just be this picture with these two vertices and one edge connecting them. So that's what induced subgraph means.

And so in particular, if I take the induced subgraph for vertices connected to a particular vertex, I get what I call a connected component. So this is one connected component here. This is another one. And this is another one. And those are the three connected components of the graph. And in general, the connected components partition the graph.

Partition means, in this case, that every vertex or edge is in exactly one connected component. So everybody is represented exactly once. This is just a fact which we will take as given. It, in some sense, follows from the symmetry, reflexivity, and transitivity of the connectivity relation. Cool. So that's connected components.

Again, many possible motivations for connected components. You would hope that the road network of North America, the continental-- let's say continental United States is connected. It probably isn't because of some weird things, disconnected things going on with Canada.

Or you might wonder, is the state space of this eight puzzle connected? Can you get from any configuration to any other? We already know the answer is no. We proved you can't get from an even permutation to an odd permutation.

So this example happens to be even. You can see the number of inversions written down here. It's always even. And if I take any other permutation that's even, it turns out it's possible to get there. We didn't prove that. What we did prove is that if we take a different configuration where the number of inversions is odd, all of these configurations cannot get to-- they're in a different connected component from this one.

And so it turns out the eight puzzle has exactly two connected components, one for the even permutations, one for the odds. The Rubik's cube, for example, depending on how you define it, has 12 connected components. In general, you might care about how many different types of states there are for a given puzzle. So that's a brief tour of connectivity.

The next concept is-- maybe I'll start with-- right now, we're just walking aimlessly from one vertex to another. Let's think about closing a loop. What if I-- I guess it's even more aimless. I started a vertex. And what if I want to come back to the same vertex? This is what we call a closed walk.

So if a walk happens to start and end at the same vertex, if v0 equals vk, then we call it closed. And so in particular, we could try to apply the closed notion to a path. That would be kind of nice. And we're going to give that a name, which is cycle. A cycle is roughly-- this is not going to be quite correct. Start with a closed path.

Can someone tell me what would be wrong with this definition? If I say, OK, I want to close-- so closed is an adjective that means you start and end at the same vertex. If I apply it to path, that means a walk, let's say, from some vertex v to the same vertex v with no repeated vertices or edges. Any problems with that? Yeah.

AUDIENCE: The last part, [INAUDIBLE] the one you start with, it's going to be repeated.

ERIK DEMAINE: Right. The one you start with is going to be repeated, because that's what we wanted. We want the start and the end to be the same. So there is no closed path. Actually, there's kind of one closed path, which is the path that just starts at a vertex v and doesn't go anywhere. That I also want to exclude. This is not what I want to call a cycle.

So there are two things. I want a closed path of length more than zero. You're not allowed to just stay where you are. And I need to add this exception that the first vertex equals the last vertex. I don't want any other vertices to be the same.

So this is-- now this is sort of a correct definition, except I'm not a big fan of definitions with exceptions. So I will write another equivalent and less controversial definition, which is let's start with our notion of walk. And we want a closed one. And so that implies that the first vertex equals the last vertex. And we want no other repeated vertices or edges.

And whereas with walk, it was-- or with path, it wasn't that big a deal that I said no repeated edges, because if you repeat an edge, you also repeat its endpoints. So repeating an edge implies repeating vertices. But I put them both in there, just for emphasis.

Here it actually matters-- say, a path with no-- a closed walk with no other repeated vertices or edges. There's another thing that I want to exclude, which is suppose I have an edge. And I get my red color. And I start at this vertex. I go to the other one. And I come back.

You might say, oh, that looks like a length 2 cycle. But it's not. I don't want to call this a length 2 cycle. And it's not allowed by this definition, because it repeats an edge. It doesn't repeat any vertices, other than the ones it's supposed to. It starts-- the sequence here is u, v, u. It looks like an emoji.

So there's no repeated vertices, except the first 1 equaling the last one. But there is a repeated edge. And so I don't allow this. This is not a cycle. So in fact, in an undirected graph, a cycle has to have length at least three. The definition says it has to be at least one. Oh, I probably need of length greater than 0 here again.

Cool. So cycle is sort of the analog of path. And closed walk is the analog of walk. Obviously, it's the closed version where you come back to where you started. Cycle is the best analog we have to a closed path. And these are nice in a certain sense, which I want to talk about.

So let's talk about a notion which is an Euler walk. And then we'll get to the notion of a closed walk.

So an Euler walk is a particular kind of nice walk that visits all of the edges exactly once. That's the definition. Let's think about graphs where you can take a walk, starting somewhere, ending somewhere, and visit every edge exactly once.

For example, this graph has such a walk, the first graph that I drew. This is why I was fascinated with it, reading some *Scientific American* article by Martin Gardner. You can draw this graph without picking up your pen. You pen down, you draw, and then you pen up. And that got all of the edges. I didn't draw the vertices, but hopefully you can imagine that. Maybe you should draw the vertices first, and then you can draw all the edges in one swoop. So this is kind of a nice notion, if you like drawing things with pen or pencil or stylus, or whatever.

It's actually a very old notion. It goes back to Euler. Unsurprisingly, this guy Leonhard Euler, who you know from the number E. That's E for Euler. In 1736, he drew this rather beautiful diagram of some islands and bridges in Konigsberg, which then was in Germany, now is in Russia. And it has-- this place, the city is famous for very beautiful bridges, as many cities are. But this one has a lot of them and some non-trivial, interesting topology.

And so he wondered-- the question was posed, can you visit all of the bridges without any repeats? It seems natural. And if you think about it, this is the vertices are like islands. The edges are bridges. And you want to visit all the edges exactly once. So a natural problem, also, if you're sightseeing. And just for fun, because I dug up this 1736 paper, here are some more beautiful figures from that paper, thinking about all the different cases.

So he characterized-- and this is like the first paper, or one of the very first papers in graph theory. This started the whole world of graph theory we're talking about. He's characterizing when this is possible and when it isn't. It turns out, in this original example, it's not possible.

Today some of these bridges have collapsed or been destroyed, so maybe it's possible. I don't know about current day Konigsberg, but that's-- what we're going to talk about now is, when do Euler walks exist? Why was it possible for that house graph, but not possible for this graph? This is the complete graph on four vertices. And you can't draw this in a single path.

Why is my red chalk always the farthest possible point? So let's try. This will not be a proof that you can't do it, but we will shortly prove that you can't do it in one stroke. So I can go here, here, here, here, here. It's pretty good, but I missed this edge. This edge is uncovered.

And it turns out it's not possible for that graph. It was possible for this graph. Even better would be, I guess, in the case-- in the tourism case, let's go back to Konigsberg for a second. Suppose your hotel is in island A. You start there. You'd kind of like to end at the same hotel without having to repeat a bridge again. So that would be an Euler closed walk.

And that we call an Euler tour, just because that's more common terminology for it. So that's really where you start. So closed means that you start and end at the same place. Euler means you visit every edge exactly once, every edge of the graph. So that's even nicer.

It turns out that's not possible in this graph. If you look at it, I started at this vertex here. And I ended at this vertex here. And that was the best I could do. I couldn't actually start and end at the same vertex.

If, for example, I added another vertex and a couple more edges, then it would be possible for me to complete the tour like this. So the new graph I drew here with six vertices instead of five, that one has an Euler tour. The original one just had an Euler walk. And this one has neither.

So how can we characterize which graphs these are? So we're not going to read Euler's paper. It's in Latin-- a bit hard for me to read. We'll write our own proof. This will be the main theorem of today. Though, we have a fun topic afterwards as well.

So first, let's think about the tour case. That turns out to be cleaner. So a graph has an Euler tour if and only if it's connected. That's certainly necessary because we said it should visit every vertex. You don't have to include this as part of the definition, but it makes the characterization cleaner.

And we also need that every vertex has even degree. For an Euler tour, for a closed walk, we need every vertex to have even degree. Let's check that in our examples. This graph, remember, the degree of a vertex is how many edges are incident to it. So the degree of this vertex is 3. The degree of this vertex is 3. This graph is symmetric, so all the vertices have degree 3. 3 is an odd number, and not even.

In this graph, this is degree 2. This is degree 4. This is degree 4. So far, everything is even. This one is degree 3. This one is degree 3 in the original graph. If I add this dotted part, then I convert this into degree 4, and degree 4, and degree 2 down here. And so then all the vertices have even degree. So it seems to be working, but let's not do a proof by example. Let's prove it in general.

Again, the top level statement here is an if and only if. So we need to prove each direction separately. Let's start with the easy direction, which is left to right. So if I have an Euler tour-- let's give it a name. Consider an Euler tour v0 to v1 to vk. And vk equals v0 because it's a closed walk.

Now I want to look at-- let's basically count edges incident to vi or some vertex. I think that's an unnecessary sentence. Let me rewind. Let's look at each visit. vi minus 1 to vi to vi plus 1 to a vertex vi.

So in this path, I'm just looking at a little piece of the path of two consecutive edges from vi minus 1 to vi to vi plus 1. And I'd like this to work even with wraparound. Because this is a closed tour, we start at v0 and we end at v0. So we do some stuff. And then we end up back at v0.

And so I'd like to just consider three consecutive vertices on this tour. Sorry. Wrong directions. This way, including possibly v0. So to do that, I want to treat this modulo, k plus 1, I guess. And this also modulo or no modulo k, actually, because vk equals v0. So if I work modulo k with these indices, for every vertex i, I look at the one previous possibly wrapping around to vk minus 1 or vi plus 1, possibly wrapping around back to v0, then this is a well-defined notion for every vertex on this visit.

When I look at this, that visit to vi, in particular, that's visiting a vertex. And it also includes or visits two edges incident to vi, sort of "what goes in must come out." So if I'm looking at a vertex here, vi, this is the previous one. Oh, this is the previous one. i minus 1 to vi plus 1.

But by visiting vi, I visited one edge coming into vi. And I visited one edge coming out of vi. And so that's two. And maybe this path visits this vertex vi, again, as some other vertex vj.

But every time it visits a vertex, it visits two edges incident to it. So that means the total number of edges visited is going to be even, because I just keep adding two, and then two more and two more. So total number of edges, I guess, visited edges incident to any vertex must be even.

And how many edges incident to that vertex do I visit? Well, all of them, because an Euler walk has to visit every edge exactly once by the end. So this total number of visited edges incident to a vertex, that's exactly the degree of the vertex. It's exactly the total number of incident edges. So this shows that if you have an Euler tour, all the degrees must be even. What goes in must come out.

It also has to be connected, just because it has to visit every vertex. And so by visiting every vertex in particular, that is a walk between any pair of vertices. Or you take pieces of that walk, and you can go from any vertex to any other. So that's one direction. That's the easy one. Let's do the hard one.

And although I won't be explicit about it, this proof also, in some sense, gives you a strategy for finding these tours. They're not too hard to find. Next page.

So now we want to go from right to left. So we get to assume that the graph is connected and that all the vertices have even degree. So we say, all degrees even. And we want to show that the graph has an Euler tour.

For this proof-- so there's one clever idea, and then the rest is just mechanical. Whereas before in this proof that I just erased, in order to convert a walk into a path, we took the shortest path, the shortest walk, and the shortest walk ended up having to be a path.

For an Euler tour, we kind of want to do the opposite. I want the longest walk that doesn't repeat any edges. The longest walk that doesn't repeat any edges, that turns out it will visit every edge exactly once. So for that, it's maybe useful to define another term, which is trail. We're really not going to use this much in the class, so feel free to forget it, other than in the context of this proof, which is a walk with no repeated edges.

So in other words, it visits every edge at most once. And then we're going to take the longest trail. So subject to every edge being visited at most once, I want to visit as many edges as possible. And that turns out it will be an Euler walk.

So let's take the longest trail in the graph. And I'll call it T. And it's going to, say, start at vertex v0, go to v1, and go to vk. I need to prove two things about this trail. In order to be an Euler tour, it has to be Euler. It has to visit every edge exactly once, and it has to be closed.

It turns out it's helpful to prove closed first. So first claim is that this trail T is closed. So we start at v0. We end at vk. Closed means that v0 equals vk. So if we do-- let's do a proof by contradiction.

Suppose v0 does not equal vk. And I want to get a contradiction, which basically means that somehow I want to show the trail was not actually longest in that case. So again, we have some-- we start at v0. We do some wiggly path. And we end at vk, and supposing these are different.

I'd like to claim that I can make this walk longer. In particular, I claim that there's an edge out of vk that I haven't visited yet, which means I could extend the path by one. That's my claim. And I'm going to do that by counting the edges at vk. How many have I used?

Very similar to this proof actually-- so vk, so this is another sequence from v0 to vk, except now we have not equals at the end. So let's think about all the visits to vk by this-- I should really be pointing at this. So here's vk. We know there's a visit at the end. Possibly it's visited in between. Maybe some of the vi's equal vk. We know v0 doesn't equal vk, but some of the others might.

Every time I visit one of those, it looks exactly like this. In other words, I visit two edges incident to vk. So let's say for every vi that's equal to vk we visit two edges incident to vk. So that's an even number in total. This is for i not equal to k.

But then there's this last visit at the end. And this one is special because we stopped. There's no edge after vk because this doesn't loop around. So there's an edge before coming into vk, but none after. So that means, I guess, for i equal k, for that visit to vk, we visit one edge incident to vk.

So in this trail how many edges incident to vk did we visit? An odd number, because we had an even number for all the earlier visits, and then we had one more at the end, so even plus one is odd. So that means T visits an odd number of edges incident to vk.

But we assume that the degrees are even. And that means there's at least one more edge that we assumed here. All degrees are even. So in particular, vk's degree is even. And so that means there exists an edge incident to vk not visited by T.

But then we can extend T. It can go from v0 up to vk, and then plus whatever one more edge. So if I call this edge here, vk comma vk plus 1, because I haven't defined a vk plus 1 yet, then this will become a strictly longer trail because that edge wasn't visited before, but now it is. This is a longer trail-- contradiction.

What did it contradict? I assumed here that v0 does not equal vk. So in fact, we learned that v0 equals vk. In other words, T is closed. So I proved this claim. I haven't yet proved the whole thing I want. So at this point, I've proved that my walk is closed. But I still need to prove that it's Euler. Those are the two things I need to show. So let's do that next.

So second claim-- and this will be the last one for this proof. So T visits every edge. I'm going to give it a name, u, v, in the edge set of the graph. So this is what I want, because, by definition, because I started with a trail, it has no repeated edges. So if this is every edge at most once, and this is saying that it visits every edge at least once, and therefore, it, in fact, visits every edge exactly once. And that's the definition of Euler.

So that's what I need to prove. Again, proof by contradiction-- not sure why, but today has lots of contradictions in it. It seems to be the easiest way. Suppose not-- this is a very concise way to say, suppose that it doesn't visit some edge u, v, and e. I'll write it out. It doesn't visit some edge u, v, and e.

Again, I want to get a contradiction to T being the longest trail. So I'm going to somehow try to make T longer. So it doesn't visit u, v, but it visits some stuff, v0 to vk. So let's try to connect them together.

So we've got-- here's the edge u, v that's not covered. And then over here, we have v0. Some crazy path that can revisit vertices and whatever. It's a walk. Sorry. I keep saying path when I mean walk. vk-- oh, actually, at this point, we know that vk equals v0. So it ends where it started, but somehow it doesn't use this edge. It might use these vertices. It might not.

But what I'd like to do is connect v to v0, which is also vk. Why can I do that? How do I know? I claim there's a walk. Yeah.

AUDIENCE: Because the graph is connected.

ERIK DEMAINE: Because we assume the graph is connected. Way up here-- assume connected. That's part of this characterization. So when we're going from right to left, we get to assume the graph is connected. Therefore, all pairs of vertices have a walk.

So in particular, these two vertices have a walk. Now, it's not part of the trail. It's just another walk. So this thing over here is T. And this thing I'm going to call w. So let's write down some words. g is connected. So there exists a walk w-- or sorry. You'll notice I called it x. I guess x for connection from v to vk.

I think I really want this to be a path, if you don't mind. We know for connectivity's sake this is the same thing. If there's a walk, there's also a path. Now, let's assume, furthermore, this is a path, in particular because I want it to be a trail. So I don't want it to repeat any edges. And a path won't repeat vertices or edges. So that's a little better. It's even better.

And I'm going to give this path some words. So it starts at v, which we're going to call x0. And then it goes to x1, and so on to xl, I guess, which is vk. It ends at vk, which is also v0.

Now, I claim I can make the path longer. Sorry. I can make the original trail T longer as follows. I'm going to start at u. I'm going to go to v. I'm going to follow x. And now I'm at the loop. And then I want to loop around. That doesn't quite work, because, plausibly, some of the vertices along x here, maybe there's some vertex here, which is actually on this path-- sorry-- on the trail T.

I want to give this vertex a name. So the whole path is called x. This vertex I want to call x sub i. So let's let x sub i, which is also some v sub j be the first vertex of x that's on T. We start at some vertex. It might-- it may be the very first vertex v is actually on T. So then x sub i is actually x sub 0.

But we know eventually we get on to T, because the last vertex on this path x-- this is the path x. We know the last vertex is vk, and that's on T. So somewhere along the way-- it could be the first vertex; it could be the last one; it could be anywhere in between-- we intersect T. And that's going to be my path.

So I'm going to start at u, go to v, follow x until I reach T, and now T is a loop-- is a closed walk. And I'm going to follow that walk as it was before. I wish I didn't draw this quite so long-- exactly what I had before. It visits exactly the same edges I had before. We walk, so we can stop here.

I claim that this red walk that I drew is a longer trail. Why is it longer? Because it has this edge now, u, v, which it didn't before. And it has all the edges we had before. Maybe it has some more here, but it also might not. Maybe these two vertices are the same, but we've got at least plus 1 here. And we visited all the edges we had before.

And because of the way we chose xi, all of these edges were not used before. So it's still a trail. And so I guess what do I write? u to v, which is also known as x0. Then we go to xi. xi equals vj. And then we go to vj plus 1, and so on, around T. In the notes, I wrote it explicitly. You go from vj up to vk. And then you loop around. And you go from v0 up to vj again. And that will be all the edges we had before. So it's longer.

And this will be a trail because we didn't-- all the edges that we added were not in the original trail T. And because we assumed a path here, there weren't any duplicate edges within x. And so I think we're good. And that proves a contradiction.

Because we assumed that T was the longest trail, therefore, T actually visits every edge in the graph. It also visits every vertex at least once, just because the graph is connected. If you visit all the edges and your graph is connected, then you definitely visit all the vertices. So that proves this theorem. Any questions? A lot of details there. That's one of our bigger proofs, but it's pretty cool.

Let me-- this was Euler tours. You might also wonder about Euler walks. So we've now shown in particular that this graph with the extra vertex here is indeed Eulerian. I mean, I also drew it, but we now understand why.

When I remove this vertex, not all the vertex degrees were even. Two of them were odd. It turns out that's the right answer. So corollary, remember, means a theorem you can prove from the previous theorem, or follows fairly easily.

So I claim that a graph has an open Euler walk. Open here just means not closed. So we've already characterized the situation of closed Euler walks. That's this theorem. That's the Euler tour. So what's left is to characterize when there's an open Euler walk, where you start and end at different vertices.

So let's ignore-- let's suppose that we don't have a closed Euler walk, but we do have an open one. I claim this is possible if and only if the graph is connected as before, and exactly two vertices have odd degree.

Intuitively, those odd degree vertices are where you start and where you end. If you look at this example with this erased, I started here, which had degree 3. I walked around, and I ended here, which also had degree 3. So the odd degree is essentially to compensate for starting and ending at those vertices. And that turns out to be all you can do.

So if you have a graph with all even degree vertices, then there's an Euler tour. And if you have a graph with two odd degree vertices, then there's an Euler walk. If you have more than two odd degree vertices, like here, there's neither. What if I have one odd degree vertex? Yeah.

AUDIENCE: That's impossible.

ERIK DEMAINE: That's impossible. What are you talking about? Because the number of odd degree vertices is even. I think you probably showed that in recitation, or even lecture. I forget. It follows from the handshaking lemma from lecture. So those are your options. Cool.

Let me sketch a proof of this. And then I want to go to the last topic, which is trees. So I want to-- the easy direction, as before, if you have an Euler walk, you're still connected. And if you do the counting right, you can see the number of degree 2 vertices should be odd, because it's just like this proof.

But now it's not symmetric. We can no longer do these things modulo k, which was really nice. Every visit looked the same, because now there's-- v0 and vk are visited in a different way from every other vertex. And just like we said before, the last vertex, the last visit has only one edge coming into it. So that makes this one odd. And the first vertex has only one edge going out of it, so it makes that vertex odd. So this is easy. The proof is in the notes if you're interested.

The interesting part is the other direction. I mean, this is roughly as before. Let's suppose that we have exactly two vertices of odd degree. I want to claim there's an open Euler walk. And I want to do it without repeating this giant proof. That's a lot of work to-- I mean, you could repeat it and be more careful. And you can prove the Euler walk thing.

But in fact, there's a way to convert this problem into the Euler tour problem. So I'm going to do it by picture. Suppose you have a graph. And you have two vertices, u and v, which are odd degree. Maybe they're also just odd in general, but their odd degree.

And now I'm going to modify the graph. I'm going to construct a new graph, G prime. So this graph that I drew is G here. And in G prime, I'm going to add a new vertex x and two edges, from u to x and x to v. Why did I do that? Because now all the degrees are even.

In G, the degree of u was odd. And in G prime, it's one larger. So now it's even. Same for v. And x has degree 2. So now everything's even. So that means G prime has an Euler tour by the previous theorem. So that was-- I don't have to repeat the proof. I can apply the theorem as a whole.

And now I want to convert this tour into an Euler walk in G. How do I do that? I just erase these two edges. The Euler tour visits every edge in G prime exactly once. And it's a loop. So in particular, at some point, it crosses over, let's say, from u to x. And then it can't go back along the same edge, because you can't repeat edges.

So then it has to go-- let's say it goes from u to x to v, or it might go from v to x to u at some point in the closed walk, in the tour. And when it does that, I want to-- instead of going from somewhere to u to x to v, I'm going to start at v. And then I'm going to keep going along the tour until I get back to u. And that will include everything in the tour, except for these two edges. And so that, actually, will be an Euler walk in G.

So that maybe went a little fast there, but hopefully that's fairly intuitive. This is a very powerful idea called reduction, where you take your problem that you want to solve and you convert it into a problem that you already know how to solve. So we already-- we just solved Euler tours. And now we want to solve open Euler walks. And we could do it by adding-- augmenting the graph a little bit, applying the theorem, and then we need to make sure that the result from that theorem is actually good enough for what we need. We can convert our Euler tour into an Euler walk.

But because tours are cyclic in nature, you can rotate it to start at v and keep going and visit everything and end at u. Cool. So that also proves that you start and end at the two odd degree vertices in G, which is nice. Questions?

Let's go back to-- back to trees. So we talked about walks, paths, cycles, connectivity, Euler things. And the last topic is trees. That's a very sad board.

I guess now for something completely different. Now we know how to draw graphs without raising our pen. I define something called a tree, which you've probably heard the word tree many times in the past, not just nature trees, but also computer science trees are all over the place, like your directory tree. I don't know. There are trees everywhere. If you've done any data structures, you know about trees, tree data structures.

So what is the mathematical notion of a tree? It's a graph. And again, today, we're using simple undirected graphs. And trees always require that. It's a graph that is both connected and acyclic. Acyclic just means no cycles.

What's a cycle? It's been a while. Remember, a cycle is a closed path of positive length. So it doesn't repeat any vertices or edges, except that the very first vertex equals the last vertex.

So if there are no cycles, there's no way to do this. Then we call it a tree. And some examples of trees. Sure. Just a path is a tree. This is a special kind of tree we call a star-- one vertex connected to a bunch of things. In general, it looks something like this.

If you've seen computer science trees, this can be a little bit different. So the only thing we're requiring here is that they're connected. So this is a tree. This is a tree. But together, this is not a tree. It's actually what we call a forest. This would be a forest with three trees in it. Or individually, these are each a tree graph.

Let's prove some properties about trees because they're so useful. Just nice to work with. In particular, there's a very useful notion about trees called leaves. You can guess where this came from, from real trees.

But the idea is very simple. Just a leaf is a vertex of degree 1. You can apply this to any graph. Sometimes you have vertices degree 1. Sometimes you don't. But trees have leaves. Shocking news. So this is-- I'll call it the leaf lemma. It's not quite true that all trees have leaves. They have to start growing for a little while first.

And in mathematics, this is every tree of-- I guess width is better-- at least two vertices has at least two leaves. So as long as your graph-- as long as your tree is not this graph-- maybe I'll add it to my list of examples. These were all trees. Also, single vertex is a tree, but it doesn't have any edges. And it has degree 0 vertex, no degree 1 vertices.

So other than this exceptional case, every other tree has at least two leaves. Where are the leaves in these graphs? Here's a leaf. Here's a leaf. Here's a leaf. Here's a leaf. This one just has two. This one has a whole bunch of leaves.

Let's sketch a proof of this lemma, which is a familiar technique. Take a longest something. In this case, the something is path. Remember, a path is a walk that doesn't repeat any vertices or edges. So I want to take a longest path in one of these graphs.

For example, if I take this, this turns out to be a longest path in this graph, length 4. I believe you cannot find a path of length 5 here. You can find walks that are arbitrarily long, but paths, you can only find length 4 in this particular graph.

I claim that the first vertex and the last vertex in that path must be leaves. So ends must be leaves. And they must be distinct. Why are they distinct? Well, this is a path. And a path can't repeat vertices. So unless the path has length zero, the start vertex and the end vertex are different.

It's not length zero, basically, because we're not in this case. In this case, there's no edges. You can't follow anything. But as long as there's at least one edge, then you know there is a path of length at least one. And so the start vertex and the end vertex will be different.

And furthermore, I claim they must be leaves. Basically, if this wasn't-- if this vertex didn't have degree 1, so there wasn't just this one way to exit, there'd be another edge incident to that vertex. And then you could have made a longer path, so a contradiction.

I'm going to call it there. And you can read the notes for more details. And instead, I'm going to tell you what you can do with these leaves. Why are they useful? Every tree except the one vertex tree has a leaf. And if you've ever taken care of a plant, sometimes it's useful to prune the plant. It grows better. In our case, pruning is a very useful operation that lets us use induction.

We've seen some inductions for graphs. And trees are really nice to induct on, because they have leaves to trim. So this is a theorem for any leaf l in a tree T, T minus l is a tree. What is T minus l? This is the graph induced by all the vertices minus l.

So in other words, it's the graph you get if you take the tree and you clip off the leaf and its incident edge. That's the negative version. The positive version is you say, I keep everything except l. And I take all the edges among those vertices that do not include l. That will end up having-- let's call this T prime.

The number of edges in T prime is going to be 1 less than the number of edges in T. And the number of vertices is also going to be one less, because we removed one vertex and one edge. So here's our tree. Here's a leaf.

This whole picture is T. And this inner picture is T prime. This is a leaf l. So T prime or T minus l is this smaller thing. A very useful thing-- in general, if you remove a vertex from a graph, you also remove all of its incident edges.

So this is a thing which is pretty easy to prove, because what we're saying is suppose that T is connected and acyclic. The claim is that T prime is also connected in acyclic. And you can just show that that's true. It's not very hard.

Again, see the notes. And what this enables to do, together, the leaf lemma and this tree-pruning lemma let you do induction over trees in a really nice way. So let's argue-- let's see an example of that with something called the size of the tree.

Any tree with n vertices has n minus 1 edges always. If you look at these examples, the one vertex tree has zero edges. The four vertex path has three edges. The six vertex star has five edges. This thing has one fewer edge than vertices. I won't count. I guess there are three kinds of mathematicians in the world. Those who can count, and those who can't. So I'm the latter kind.

So finally, we get to an induction. We haven't done induction all day. It's terrible. Finally, we can do induction. So we're going to-- remember, whenever you do an induction over graphs, you're really just doing regular induction or strong induction over numbers. And so in this case, we're going to induct over the number of vertices in the graph.

So our induction hypothesis is this, that for a number n, P of n is the statement that any tree with n vertices has exactly n minus 1 edges. I should write exactly here. And we're going to prove it by induction. So we probably need a base case.

What is the smallest possible thing that n can be? 1. Remember, we forbid zero vertex graphs. You have to have at least one vertex to be a graph. So in n equals 1, that means there's only one graph with n equals 1, and it has zero edges. Great.

Well, my vertex looks a lot like my number 0. Sorry. So let's do the induction step. Now we're going to use all of these beautiful lemmas that we have about leaves. So let's suppose that n is bigger than 1. In other words, n is at least two. And therefore, the leaf lemma applies. It says if you have n at least two, you have at least two leaves. I actually only need one of them.

But let's say, by leaf lemma, there exists a leaf. And let's call it l. Now, I'm going to apply this tree pruning lemma that says now I have a leaf. I can remove it. Clip it. Take a new graph, T prime, which is everything in T except l.

So by tree pruning T prime, which is T minus l, is a tree. Not only is it a tree. It's a smaller tree. It has exactly n minus 1 vertices. So I can apply induction, because in their induction step, we get to assume that this property is true for all smaller n. That's strong induction. In particular, it's true for n minus 1. That's weak induction.

So by the induction hypothesis, we know that T prime has one fewer edge than vertex, and it has n minus 1 vertices. And so it has n minus 2 edges. And T prime was everything in T except for one edge, that measly edge connected to l. So T has one more edge than T prime. So it has n minus 1 edges. And that's what we needed to show.

This is very nice for inducting over trees. The idea is if you have a tree, you're going to do induction. You should take a leaf and prune it. Usually, that's what you want to do. Sometimes it's useful to know there's two of them.

You could take any vertex and remove it. But if I took a vertex in the middle of my tree, it might disconnect into multiple trees. And then I have to do a lot more work. By choosing a leaf, when I remove it, I guarantee I'm still a tree. And so I can still apply induction. It's very helpful. That's lots of fun things about trees.

It turns out the reverse is true. If you have any connected graph with one fewer edge than vertex, it must be a tree. Trees also have unique paths between any two vertices. And any graph that has unique paths between any two vertices is a tree. Other fun things, like they're the minimal connected graphs or maximal acyclic graphs-- lots of fun properties, which you can find in the notes or the textbook. That's it for trees. Next time we'll do directed graphs.