PROFESSOR: Directed acyclic graphs are a special class of graphs that really have and warrant a theory of their own. Of course, "directed acyclic graphs" is lot of syllables, so they're called "DAGs" for short. OK. So here's why they come up all the time.

Let's look at a diagram that may be familiar to you. This shows the prerequisite structure of required courses in the 6-3 program of MIT Electrical Engineering and Computer Science department. There are similar charts for the other sub-majors of EECS and in other departments as well. So what does it mean?

Well, let's look at this vertex corresponding to the first term calculus class 18.01. And there's an edge that points to 6.042. And that's because, if you look at the catalogue, 6.042 lists 18.01 as a prerequisite. If you look at the algorithms-- introductory algorithms-- class 6.006, you'll find, if you look in the catalog, that it has listed two prerequisites, 6.042 and 6.01. And the fact that they're explicitly listed in the catalog as prerequisites is why there's an arrow from 6.01 to 6.006 and from 6.042 to 6.006.

Now when you're planning on when you want to take 6.006, of course, you have to attend to not just the fact that you have to take 6.042 first and 6.01 first, but you've got to take the prerequisites of those prerequisites first. So you really have to take 18.01 before you can take 6.006. And you need to take 8.02 before you can take 6.006. There are corequisites here. Let's just ignore those and pretend that they were prerequisites, because they're another kind of arrow that needn't distract us.

OK. So that's what this diagram is telling us. And this is a DAG. It's simply a bunch of vertices, the course labels in rectangular boxes, and directed arrows showing catalog listings. And what I said was that when you're planning your course work, you're really interested in the indirect prerequisites.

So "one class, u, is an indirect prerequisite of another class, v" means that there's a sequence of prerequisites starting from u and going to v. It means that you really have to have taken u some time before you took v. And that's a crucial fact and thing that you need to take account of when you're planning a course schedule.

So in terms of graph digraph language, "u is an indirect prerequisite of v" just means that there's a positive length walk that goes from u to v in the digraph. In this case, we're talking about the 6-3 digraph of prerequisites. So "there's a positive length walk from 18.01 to 6.006" means that you really have to have taken 18.01 before you take 6.006.

And of course, we're talking, then, about the positive length walk relation D plus of the digraph D. If D is the digraph shown in the prerequisite chart-- direct prerequisite chart-- then we're interested in D plus. And u D plus v just means there's a positive length walk-- that's what the plus is for-- going from to u to v.

Now what happens if you have a closed walk? Well, a closed walk is a walk that starts and ends at the same vertex. And we can ask this question-- suppose there was a closed walk that started at 6.042 and ended at 6.042. How long does it take to graduate then?

Well, it takes a long time. Because you can't take 6.042 until you've taken 6.042 and you're never going to be able to take it. That's a bad thing. We definitely don't want the prerequisite structure of courses in a department to have a closed walk of positive length. And in fact, there's a faculty committee that checks for this kind of thing.

Bugs like this occasionally creep in when some busy curricular office of a department is planning a complicated program with dozens, if not 100 courses. And the Committee on Curricula's job is to check for that kind of thing. There's a whole staff that does it. I used to be the chair of that committee. And we did spend a lot of time with proposals from departments and making sure that those proposed course requirements satisfied faculty rules.

OK. So a special case of a closed walk is a cycle. A cycle is a walk who's only repeat vertex is its start and end. Let me remark, because we keep talking about positive length cycles, that a single vertex all by itself is a length-0 cycle. So you're never going to be able to get rid of length-0 cycles, because they're the same as vertices. But positive length cycles, you can hope to ensure are not there.

So if you're going to represent a cycle as a path, you'd show the sequence of vertices and edges, v0, v1, v2, where the understanding is that all of the vertices from v0 up to v n minus 1 are different-- that's what makes it a cycle-- except that the last vertex, v0, is a repeat of the first one. That's the one repeat that's allowed in a cycle. So it's natural to draw it in a circle like this where you start at v0, you follow the edges around from v1 to v i plus 1 all the way back around to v0. And that's kind of what a cycle is going to look like.

So we have a very straightforward lemma about cycles and closed walks, namely that the shortest positive length closed walk from a vertex to itself-- "it's closed" means it starts and ends at v-- is a positive length cycle starting and ending at v. And the reasoning and proof is basically the same proof that said that the shortest walk between one place and another is a path from one place to the other.

The logic is that if I have a closed walk from v to v and there was a repeat in it other than at v, I could clip out the piece of the walk between the repeat occurrences and I'd get a shorter walk. So the shortest closed walk can't have any repeats. It's got to be a positive length cycle.

So a directed acyclic graph now is defined simply as a digraph that has no positive length cycles. It's acyclic, no positive length cycles. And of course, we can equally well define it, since cycles are a special case of closed walks and closed walks of positive length imply cycles, as a digraph that has no positive length closed walk.

Some examples of DAGs that come up-- well, the prerequisite graph is going to be one. And in general, any kind of set of constraints on tasks, which ones you have to do before you do other ones, is going to be defining a DAG structure. One that you might not have thought of is, the successor function defines a relation on the integers, say, going from n to n plus 1. So I'm going to have an arrow that goes directly from n to n plus 1.

And what's the walk relation then, the positive length walk relation, in this graph? Well, there's a positive length walk from n to m precisely when n is less than m. So the successor DAG, it's paths represent the less than relation. And of course, less than, it doesn't have any cycles. Because if a is less than b, you're never going to get around from b back to something that's less than it, like back to a. So there can't be any cycles in the successor DAG. And that's why it is a DAG.

Another similar one is the proper subset relation between sets. So here I'm going to draw an arrow from this set to that set if this set is contained in that set but they're not equal. So {a, b} is a subset of {a, b, d}, but {a, b, d} has this extra element, d. So the left-hand set is a proper subset of the right-hand set. And I'm going to draw an arrow there.

And by the same reasoning, there can't be any cycles in this graph-- a positive length cycle-- because if there was, it would mean that the set had to be a proper subset of itself, which doesn't happen. So this would be another basic example of a DAG. And I hope you begin to see, from these examples, why DAGs are really all-pervasive in mathematics and in other areas and why they merit attention.

So when we're looking at a DAG though, we're basically usually interested in just the walk relation of the DAG. So if we're only interested in the walk relation of the DAG, then it would be typically the case that many different DAGs are going to have the same walk relation. And it's natural to ask, what's the most economical one. Is there a minimum, say, DAG that defines a given walk relation?

So let's look at this example. Here's a simple DAG. And you can check that there are no cycles in it. What's the smallest DAG with the same relation as this one? And the way I can get it is by going through the edges one at a time and asking whether I can get rid of the edge because it's not contributing anything.

So look here. There's a path from a to e that goes through b. Well, that tells me that having this direct edge from a to e is not contributing anything in terms of connectedness. And that means I could get rid of it and I'm still going to wind up with the same possibility of walking from one place to another. Because I can always walk from a to e going through b instead of going directly from a to e. I didn't need that edge.

Another example is, here's a walk from a to d that goes through c. There's no need for me to walk directly from a to d. As long as I'm walking, I can take the longer walk and get rid of the short circuit from a to d. Likewise, if I look at this path from c to d to f, I don't need that edge from c to f. And as a matter of fact, now if I look at this length 3 path from a to c to d to f, there's no need for me-- in order to get from a to f, there's no need for me-- to take the direct edge. I can get rid of that too. It's kind of a redundant extra edge.

Finally, if I look at the path from b to d to f, I can get rid of the direct edge from b to f. And at this point, I'm done. I'm left with a set of edges called covering edges, which have the property that the only way to get from one vertex to another is going to have to be to use a covering edge to the target for vertex. Or more precisely, the only way to get, say, from a to b is going to be to use that covering edge.

If there was any other path that went from a elsewhere and got back to b without using this edge, then it wouldn't be a covering edge anymore. The fact that it's a covering edge means that if you broke it, there's no way anymore to get from a to b. So that's the definition of covering edges and you'll do a class problem about them, more precisely, in a minute.

So the other edges are unneeded to define the walk relation. And all we need to keep are the covering relations to get the minimum representation of the walk relation in terms of a DAG.