[SQUEAKING]

[RUSTLING]

[CLICKING]

ZACHARY ABEL: Good morning. Good morning. Welcome. Welcome to lecture 11 of 6 120, 6 1,200. I never know how to pronounce it. It's one of them, probably.

We have finished our number theory unit, and we're now starting a new unit on graphs, graph theory. This is a new topic, a new kind of mathematical abstraction that we'll see really interesting uses for across this class, across algorithms, across other contexts as well. Before we dig into what they're good at, let's talk a bit about what they are, how they're defined.

So I've written an example of a graph up here. Totally clear, right? It's easy to read and comprehend, but let me define it for you.

So a simple graph. Here, let's say a simple undirected graph, which are the only kind of graphs we'll be talking about for the next couple of lectures, is defined as a pair. So the graph G is a pair of V, E, where V is a nonempty set of what we call nodes, also known as vertices.

And E is a set of two element subsets of V, not all of them, but some of them, possibly none of them. Could be all of them as well, any amount that you want. So a graph is just a set V, where we call the objects vertices or nodes. I'm sorry. These are edges.

So it's a set V of nodes and a set E of pairs of nodes. And those pairs are called edges. So in its purest set theoretic form, here is a graph. Here's a set of vertices. Here's a set of some of the two element subsets of V. Of course, I don't really think about the graph in this form.

Let me give you an alternative way to express this graph up here. 1, 2, 3, 4, 5, 6, 7. These are going to be my nodes. This is a. This is b, c, d, e, f, and g. And anytime we have an edge, let's connect them by drawing an arc between them. So we have ab, ac, and bc. We have these, these, these, this, this, and this.

This is a visual representation of this graph, but it's conveying the same information. This is a lot easier to look at, in my opinion, but they're both conveying the same information. This is the general idea of a graph. It's a set of vertices, which are the dots, and edges, which are the segments connecting them. Does that make sense? Wonderful.

I want to mention that there are some subtle differences in the definitions in other sources. Sometimes when people are talking about undirected graphs, some people will allow what are called parallel edges, which is when you have two different edges connecting the same pair of vertices, or self-loops, which is when you have a single edge connecting a vertex back to itself. And we are disallowing both of those.

So we will never be working with graphs like this that have parallel edges or that have self-loops. So that is what people usually mean by simple. And it's why all of the undirected graphs we work with are going to be simple.

Likewise, some sources allow the vertex set to be empty. I stipulated up there that v should be a nonempty set. So we're just disagreeing on a single graph that has zero vertices and then necessarily zero edges because there's nothing to connect.

The reason we are not allowing that for our purposes is that often it's going to be a counterexample to a lot of the theorems we want to state. We would always have to say for every nonempty graph, for every graph with a positive number of vertices, something like that, and it just gets cumbersome. So we chose this definition because it makes that more convenient. Not really a big deal, not really something we'll need to think about much. It's just a part of the definition to be vaguely aware of in the back of your mind.

We do allow the set of edges to be empty. For example, here is a graph with five vertices and zero edges. Let's call them vertices 1, 2, 3, 4, and 5. Five vertices, zero edges, that's totally fine. That's a valid graph. So this has an empty set for the set of edges, which is fine. All right?

So what are graphs good at? Graphs are great at representing two-way relationships between pairs of objects. Let me give some real-world examples, like friendship. You can imagine the graph of all the friends in the world.

Here, let's see, the friend graph. The vertices are going to be the set of people. Each person is a vertex. The set of edges is pairs of friends.

Now, I'm making an assumption here that friendship always goes both ways. If I'm friends with you, then you're friends with me. No unrequited friendships here. For example, Facebook friendships always go both ways. You're either friends on Facebook, or you're not. So the graph of Facebook friends is a concrete representation of this kind of thing, which can be interesting.

Let's see. Another example is the brain. The brain is just a network of neurons. The neurons are the vertices, neurons and connections. Some neurons connect to neighboring neurons, and it's those neurons and their connections that somehow creates cognition and memory and intelligence. And I wish we understood how. But at least it's a graph.

Also, we have the internet. Most network connections on the internet go both ways. Network connections. So the servers are the vertices. And which servers can talk to which other servers are the edges. And most communications have to go both ways because you have to communicate back and forth and make sure that each of you trusts the other party before you actually send any information over. So you have to go both ways to make that connection first, and then you can send information in whatever direction you want.

A non-example is hyperlinks on websites. If site A has a link that takes you to site B, there isn't necessarily a link back. So this would not be an example of an undirected graph. That would be a directed relationship, which we're going to talk about later after break. But that's not an example for today. Similarly, Twitter followers is a directed relationship. If I follow you, you don't necessarily follow me. So that would be a directed graph, which will fit into lectures after break, not this one. All right.

And graphs-- I said earlier, graphs are a really useful abstraction. And I claim that they're useful because they strike a nice balance between simplicity and complexity. On one hand, graphs are simple to define and describe. They're simple conceptually. It's just a set of objects and a set of pairwise relationships. Easy to define. So as we just saw, there are lots and lots of real scenarios that can be modeled faithfully as a graph. On the other hand, they're complex in that there's a rich mathematical theory of graphs.

There are lots of powerful tools and definitions and theorems that we can bring to bear anytime we're trying to analyze a graph. And therefore, we can apply all of that to all the real-world scenarios that we have been able to model as a graph. So it's a really nice balance between the two. So let's dig into this rich mathematical theory and see what tools and theorems we can develop.

So the next definition is vertex degree. Sorry. The next definition is adjacency. Let's see.

Two vertices, u and v, are adjacent if and only if they're connected by an edge, i.e. u, v is one of our edges. So their vertices are adjacent if they're connected by an edge. We'll say that-- let's see-- an edge u, v has endpoints u and v, just the two endpoints of the edge. And we'll say that edge u, v is incident to u and v. So a vertex and an edge are incident if that edge has that vertex as one of its endpoints.

These words sound a little alike. Adjacent is used for two vertices that have an edge between them. Incident is used for an edge and a vertex, where the vertex is an endpoint of the edge. But useful terminology.

And that lets us define the next thing, which is for a vertex v, the degree of v is the number of edges incident to v. So the degree of a vertex is the number of edges it has. So what's the degree of e? Just yell it out. 3. yeah. 1, 2, 3.

What's the degree of c? 4. Degree of f? 2. What's the degree of this vertex 2? 0. It doesn't have any edges. Excellent. Degree is just the number of incident edges. Wonderful.

You can understand some, not everything, but some things about a graph just from the degrees of its vertices. For example, that graph way over there on the left, the degrees of its vertices are-- let's go through them all. We've got 2, 2, 4. 3, 3, 2, 2, in that order. I think I got that right.

Sometimes this is called the degree sequence. That's less important than the concept of degree itself. But we can play with this concept a little bit. For example, does there exist a graph with degree sequence 2, 2, 1? Sorry, that's supposed to be a question mark. There we go. Does there exist a graph with degree sequence 2, 2, 1? So three vertices, two have degree 2. One has degree 1. Think about that for a second, and then we'll try it together.

OK, I see someone violently shaking their head. I think they're cracking their neck. Let's try it. So 2, 2, 1 means we need three vertices-- 1, 2, 3. The first vertex has degree 2. Well, there's really only one way to do that. It has to connect here and here.

The next vertex also has to have degree 2. Well, that means it also has to connect there. The only way for it to get a second edge. Then the third vertex is supposed to have degree 1, which it doesn't. So we're sad it didn't work.

You can think of this, by the way, as a proof by contradiction. If there were a graph that satisfied this, then it would have to look like this. But this graph doesn't work, so nothing works. Sound good?

OK, but what if we asked a more complicated version of the question? Does there exist a graph with degree sequence 2, 2, 2, 2, 2, 1? This time there would need to be six vertices. And now there are lots of ways that we could choose two edges for each of those vertices. So there are lots of cases that we would need to try out in our proof by contradiction. So maybe a different strategy would work better.

Does there exist a graph with this degree sequence? Yes.

AUDIENCE: Does the sum always has to be even?

ZACHARY ABEL: No, because the sum always has to be even. Interesting. Why does the sum have to be even?

AUDIENCE: If it's an edge of [INAUDIBLE] another vertex [INAUDIBLE].

ZACHARY ABEL: All right, so if an edge connects to one vertex, then it also connects to another vertex. So each edge is somehow counted twice. Yeah, that's exactly the right intuition for the following theorem. In fact, that's the correct proof for the following theorem.

We've got the handshake lemma. Let's see. For a graph G, which is vertices and edges, we have the sum of vertices in the set of vertices of the degree of that vertex equals. So if you take all the degrees and add them all up, your colleague here claimed that what you're going to get is two times the number of edges.

And we can see that for this example, if we add this up 4, 8, 14, 16, 18, and there are 9 edges, that checks out. And the reason this is true is also exactly what your colleague said earlier. Over here, we're adding the degree of every vertex, which means we're basically counting every endpoint of all the edges.

And every edge has two endpoints. So every edge was counted twice in this count on the left. So that's the handshake lemma.

It's called handshake lemma because if you imagine a room full of people and some pairs of them shake hands, then if each person counts the number of handshakes they had, and you add that all up, that equals two times the number of handshakes, because each handshake had two hands involved. It's exactly the same explanation and proof. It's just with hands instead of pairs-- instead of half edges.

All right, so does this help us with that? Can someone tell me why there doesn't exist a graph with this degree sequence? Yes?

AUDIENCE: Because the sum is odd.

ZACHARY ABEL: Yeah, because the sum is odd. This would have to have, what, 11 divided by 2 edges, which isn't great. I don't know how to draw a half edge in these graphs, so there is no such graph. Awesome.

Here's one more application of the handshake lemma. Let's see. What is the maximum number of edges in an n vertex graph? What is the maximum number of edges in an n vertex graph?

Well, each vertex has degree at most n minus 1, because at most, it connects to all the other vertices. There are no other vertices to connect to. You can't connect to yourself. You can't repeat any connections. Once again, that's what we meant by simple graph, and all our graphs are simple. So every vertex has degree at most n minus 1.

So the sum of the degrees is at most n times n minus 1. But the sum of the degrees is equal to 2 times the number of edges. So the number of edges is at most n times n minus 1 divided by 2. Didn't have to do any fancy induction. Didn't have to compute some sum. I wonder if this equals a sum that we've seen before.

So we have an upper bound. The number of edges is at most that. And it turns out this is achievable. Achieved by the complete graph Kn, which is the graph with n vertices and all possible edges.

So this vertex connects to everyone. This vertex connects to everyone. This connects to everyone, connects to everyone. I think I got them all. So this is a drawing of K6. There are six vertices, and everyone connects to everyone.

This graph indeed has every vertex with degree n minus 1, so it has exactly this many edges. The K for Kn, this is standard. You'll see Kn everywhere. I think it comes from German. It kind of sounds like complete graph, right? But we'll be using this notation a lot.

All right. Let me ask another question. So most of us here are MIT students. I know we have some cross-registered students. Welcome. I'm glad you're here.

For the MIT students in the room, do you have any friends at Harvard? Yeah? Awesome. OK, so I got at least one hand. I got more than one hand.

But let me ask, how many Harvard friends do you think the typical MIT student has? Don't answer that. It's rhetorical. What about the reverse question? How many MIT friends does the typical Harvard student have? Which one's bigger?

Let's see. So on one hand, we've got the average number of Harvard friends for the typical MIT student, or the average number of MIT friends for the typical Harvard student. Which one's bigger? And how could we know? How can we decide this?

Maybe I could do a big survey. I could sample a bunch of MIT students, sample a bunch of Harvard students. Take the average of whatever is reported, and take that as ground truth. That is how a lot of empirical statistics ends up working.

Can anyone suggest a better way? Yes?

AUDIENCE: So we know that every friendship is two-way, and every [INAUDIBLE] has a bigger undergrad population at MIT. So you divide it all by a big number, so [INAUDIBLE].

ZACHARY ABEL: All right. So let me rephrase that. So we know that every friendship goes both ways. There's one MIT student and one Harvard student. Again, we're assuming friendship is two-way, no unrequited friendships. Can't the two schools just get along?

But because they're the same number of friendships on both sides, when we're taking the average, we're dividing by the number of Harvard students and dividing by the number of MIT students. But Harvard has more undergrads than MIT, so we've divided by a bigger number. So the MIT average should be bigger.

Whew, that was a lot. Thank you for saying all that. Let's unpack it a little more slowly. But you are absolutely correct in every step of your reasoning.

So what I'm going to do is let's draw a graph where the vertices are going to equal the set of MIT undergrads and the set of Harvard undergrads. And I want to emphasize that these are two disjoint sets. No student is enrolled in both schools. We're not counting cross-registration. That's why I mentioned that earlier.

And sometimes to emphasize that these two sets are disjoint, sometimes you put a little dot inside the union sign. So our set of vertices is the disjoint union of MIT students and Harvard students. And our edge set is going to be the set of-- let's see-- m, h, where m is an MIT student, h is a Harvard student, and m and h are friends.

And we can draw an example of what this might look like. So here's all of my Harvard vertices on the left. Here's all of my MIT students on the right. Here, we have, I don't know, Bill and Helen and Natalie and Neil and TS. On the right, maybe we have Buzz and Kenji and Lisa, and these lists go on. This is just an example. I don't know if there are actually students with these names.

And some pairs of them are friends. Maybe these are friends and these and these are friends. And Buzz is just friends with most people. And TS only likes Lisa among this graph. Some pairs of them are friends. Here's our graph.

Again. there are more vertices, more edges, but here's an example of our graph. And this graph has a special property called bipartite. This is a bipartite graph.

Let's see. Definition. G is bipartite when it's possible to partition V as the disjoint union of two sets. I'm going to call them left and right. That's common notation for bipartite graphs, such that every edge has one endpoint in L and one endpoint in R.

So you have some vertices on the left, some vertices on the right. And every edge crosses between them. One endpoint on the left, one endpoint on the right. These graphs are called bipartite. And the one we just drew certainly has that property. Sound good? With me so far? Awesome.

And now, we have a bipartite version of the handshake lemma. Let's see. Handshake lemma bipartite, which says that if you add up all the degrees just on the left, the vertices in the left side degree of V, well, every edge has one endpoint on the left and one endpoint on the right. So this is just equal to the number of edges.

And likewise, you can do the same thing on the right. So all the degrees on the left add to the number of edges. All the degrees on the right add to the number of edges.

But remember, we are interested in the averages. The average degree on the left versus the average degree on the right. All right? So we've got, let's say, average degree on the MIT side divided by the average degree on the Harvard side.

Well, this is equal to the total degree on the MIT side. So just adding up all the degrees of the MIT students divided by the number of MIT students. And this denominator is the total degree on the Harvard side divided by the number of Harvard students. That's what we mean by average. Add them up, divide by the number. Add them up, divide by the number.

But we just said that the total degree on the M side and the total degree on the H side, both of those are equal to just the number of edges. So this is the number of edges divided by the size of M divided by the number of edges divided by the size of H. The E's cancel. This just becomes the size of H divided by the size of M.

All right, we've now caught up to your comment. Thank you so much. And now, we can just ask how many students there are in each school? This is about-- let's see, I wrote something down-- about 7,200 divided by 4,600. Those are close anyway. They might be a bit out of date. But this is about 1.6.

So the average MIT student has about 1.6 times more Harvard friends than the number of MIT friends the average Harvard student has. And that's not any judgment about behavior on the two sides of the schools. It's just a fact about averages after knowing a little bit of graph theory. But our number is bigger, so we're better.

But, so already, we've just defined a couple of things. And we've already been able to prove some non-obvious theorems and facts again. They still look like toy examples. But there are real scientific studies that have studied questions like that and not known the graph theory, and then came up with conclusions that were very obviously wrong to anyone who did know the graph theory.

For example, there's a very famous study out of UChicago from 1994 called "The Social Organization of Sexuality, Sexual Practices in the US." So it was a big sweeping study about sexual practices and behavior and health across the country.

They had thousands of participants. They asked lots and lots of questions and came up with lots of statistics and measurements and trends, and published all their results in a big 700-page write-up, a very famous study.

And out of all the questions they asked, I'm going to single out one of them for the single purpose that their results are obviously wrong because of exactly this argument. So one question they asked specifically to their male and female participants. They asked the men how many female partners they've had. They asked the women how many male partners they've had.

Don't worry. They had lots of other questions about other kinds of relationships. They had lots of other questions for their non-binary participants as well. Like I said, I'm just singling out this one of many questions.

So what's the average number of female partners a man has had? What's the average number of male partners a woman has had? And according to their reports and surveys, the men reported about 1.74 times as many partners as the women.

Let's see. So the average degree of the men was equal to about 1.74 times the average degree of the women. A different study, this time from ABC News in 2004, got a similar result, but instead of 1.74-- so sorry. This was from UChicago. ABC News approximated this number to be about 3.3.

Yet another study from the National Center of Health, this time from 2007, got this number to be about 1.75. When, of course, we know the exact answer. This ratio should be the number of women divided by the number of men because of the specific question they were asking about this subset of relationships among this subset of the population, they put themselves in a bipartite scenario where this theorem applies. And the correct answer is just the number of women divided by the number of men.

Let's see. The number of women divided by the number of men, which-- according to the 2010 census-- was about 157 million over 152 million, which is about 1.03. The correct answer is 1.03. And it's not a question of behavior. It's a question of just averages. There is nothing to learn from this ratio, except this is how many men and women are in the country. Not a very interesting example for a behavioral study, if you ask me.

I'm not going to speculate on why these numbers don't match reality. I don't know. Think about it yourselves if you want. Are there questions about this example? Or anything else we've talked about so far?

All right, let's move on to another topic in graph theory, another common application. So instead of measuring affinities or friendships among pairs of objects, let's measure conflicts between them. So here's an example graph. Let's define the vertex set of our graph to be the set of all MIT classes with a final exam. And E is going to be pairs of classes, C1, C2, that have someone enrolled in both.

So you're the registrar. Your goal is to assign a time for the final exam for every class. And if some student is taking two different classes and those classes are scheduled for the same time slot or overlapping time slots, then that student can't be in two different places at the same time, so you have a conflict.

So this graph here is measuring all of the conflicts among final exam times. There's an edge precisely when the two classes can't be scheduled for overlapping times. Does that make sense so far? Cool. So edges are conflicts in exam scheduling.

And let me draw an example. Let's see. So in this example, MIT only has five courses-- 6,3700, 6.2000, 6.1200, 6.3000, and 6.4100. So this is some fake approximation of what this graph might look like.

So this says that there's someone taking our class and 6.2000. There's at least one student taking both of these classes. There's no one taking 6.3700 and 6.3000. So for example, these two are allowed to be scheduled at the same time. So this is listing all of our conflicts. And this brings us to the graph coloring problem.

So we say that a proper K coloring of a graph G is defined to be-- I'll write it down mathematically first and then explain it in words-- a function f from the set of vertices to a set C of colors, where there are at most K colors. And for all edges, u, v, fu does not equal fv. So in other words, we have some set of colors available. And we want to assign a color to every vertex.

And we want to assign a color to every vertex so that no two adjacent vertices are ever assigned to the same color. In this example, you can think of colors as time slots for final exams. If two vertices are given the same color, that means that they'll be administered at the same time. And this condition that edges have to have two different colors on their endpoints means that any time there's a scheduling conflict, they will always be given different colors, which means they'll be given different time slots.

So this is what we mean by a proper K coloring. It's an assignment of a color to every vertex from a possible choice of K different colors, where the two endpoints of every edge are assigned different colors. And as an example, so let's try to color this graph. Let's say I'm going to color this red and this green and this blue and this yellow and this purple. Is that a proper coloring?

Yeah. Yeah. Every edge has two different colors on its endpoints because all the colors are distinct. This is a proper five coloring. Can be five-colored. All right?

But that means we would need five different time slots for the final exams. And we'd like to be a little more efficient. Can we color this with just four colors? Yes, I claim we can. Let's make that red instead of purple.

Now, this class and this class are both red, but because there's no edge between them, that's fine. All of the edges have two different colors on their endpoints. And that's all we need. All right? So this can be four-colored.

Can we do better? Can we get away with just three colors? All right, interesting. I'm seeing one head shake and one head nod. So yes, please.

AUDIENCE: How about making the yellow red.

ZACHARY ABEL: Make yellow red.

AUDIENCE: And then making that bright red like a blue or green?

ZACHARY ABEL: OK, we'll make this blue or green. Yeah. All right, so we've got red, green, blue, red, blue. So we've got two reds, but there's no edge between them, so we're fine. And we've got two blues, but there's no edge between them, so we're fine. This is absolutely a valid three-coloring. Can be three-colored.

Can this be 17-colored? Is there a proper 17 coloring for this graph? This is a bit of a weird edge case, but yes, this set C says that there have to be at most 17 colors available.

Now, we don't have to use all of those colors. So if we had a 17-color palette and we found this proper coloring, that would be considered a valid 17-coloring. So this is a proper 17-coloring of the graph.

But more directly, it's a proper three-coloring because usually we're trying to minimize the number of colors. All right? And we can make that notion a concrete definition. The chromatic number of a graph G, the chromatic number is denoted chi of G-- that's a lowercase Greek chi-- is the smallest number of colors needed for a proper coloring of G. All right?

So the chromatic number asks what's the fewest colors you can get away with. In our scheduling problem, what's the fewest number of time slots that we need to cover all of the final exams? For this graph, what is its chromatic number? Have we shown that it's three?

First of all, is it five? Is the chromatic number of this graph five? Why not?

AUDIENCE: [INAUDIBLE] smaller.

ZACHARY ABEL: Because we can get away with fewer colors. That's right. It's not five because we can do fewer. It's not four because we can do fewer. Is it three? I don't know yet. Can we get away with fewer colors? Well, let's see.

Can it be two-colored? Is there a proper coloring with just two colors for this graph? And in this case, there isn't, because this triangle right here shows us that all three of these vertices need different colors.

You can think of this as a proof by contradiction as well. Assume it's possible to color with just two colors. Well, then, well, these three vertices would have a conflict among them somehow, which is a contradiction. So cannot be two-colored.

And these two statements together can be colored with three colors. Cannot be colored with two colors. This is what we need to confirm that the chromatic number is three. Three is the smallest because three works and fewer doesn't work.

And I want to emphasize that, in general, to show that the chromatic number of G is some value K, usually we need both an upper bound and a lower bound. We need an upper bound, which is that chi of G is less than or equal K. And how do we prove that? Usually by giving a K coloring. If you know it's possible to color in K colors, then the chromatic number has to be K or smaller. Give a K coloring.

But you also need a lower bound that chi of G has to be greater than K minus 1. And how do you prove that? Well, you have to prove that it can't be K minus 1 colored, usually by contradiction.

So I just wanted to emphasize that the definition of chromatic number as the smallest thing that works, there are two parts to that. It works, and nothing smaller works. Make sense? Cool.

All right. Yes. So graph coloring. We've seen one application, which is the registrar scheduling final exams. They actually do something exactly like this, by the way. This is why final exam schedules don't come out until a couple of weeks into the semester because they need to see who's enrolled in which pairs of classes, so we can try to schedule those apart from each other. They're trying to find a coloring with few colors.

In practice, if they were looking for an actual proper coloring with no conflicts whatsoever, it probably wouldn't fit into exam week. So instead, they're looking to minimize the number of conflicts, which is a slightly different problem, but still an interesting one. But there are lots of scenarios where you're trying to avoid conflicts, just like the registrar is trying to avoid conflicts, so other coloring applications.

A really common one, and one that's appropriate for this computer science class, is register allocation. When designing compilers, you have multiple values that you're trying to compute, different operations you need to do with them. Yes?

AUDIENCE: Does the elements of C matter or just absolute size? Because you can put green in the left corner like there's a bunch of different.

ZACHARY ABEL: That's a great question. Thank you for asking. Do the elements of C matter or just the size of C? C is the set of colors available.

So if you change the set C, then you're giving a different coloring because your colors mean different things. But you're not going to change how many colors are needed. You're not going to change the chromatic number if you change the set of colors available. And you said we could have put green here instead of here?

AUDIENCE: Yeah.

ZACHARY ABEL: Yeah, like we could have made-- the reds could instead be green. And this green could be red, or this green could be yellow. Now, we've got some green, some blues, and a yellow. Still three colors, different coloring because we're assigning different colors to them, but still the same number of colors. And often, for the applications we're about to talk about, the number of colors is all that matters, because the colors just mean these are the different classes that you need to put them in, and then you do something with each class.

So this application, register allocation. When you're computing multiple values, your processor has a small number, like 16 or 32, registers, which are really, really fast memory that it can do operations on. And in fact, it has to bring data into a register before it can compute on them usually. And some values you're computing and you need lots of different pieces to go into that computation. Some values, well, you're done with it. You don't need to remember it anymore. So let's evict that and put something else in that register.

And ideally, you can figure out a way to schedule all of your values in the different registers so that everything fits in the registers and everything's fast, instead of having to spill them out into RAM. You don't need to understand compilers or exactly what I'm saying here. The point is, your variables, the values you're computing, have different lifetimes, times when they need to still exist, and other times when you're allowed to forget them.

And compiler writers often draw exactly the conflict graph between these lifetimes. This value has to exist at the same time that this value also has to exist, so they can't be in the same register, because then one would overwrite the other. And if you can write down your conflict graph and find a coloring with a small number of colors, ideally smaller than the number of registers available, then everything works in registers, and you're really fast.

Another one is radio tower frequencies. If each tower in a radio network needs to broadcast with some frequencies, well, if two towers use the same frequency that can reach the same places, then they're going to conflict. You're going to get static in the noise. You're getting noise from two different places at once. So any conflicting towers need to use different frequencies.

And this is, again, a graph coloring problem. Draw your conflict graph. Color it with a small number of colors. Each color is a different frequency. And since different towers that are assigned the same color don't have an edge in the conflict graph, it means they don't conflict, and so they're allowed to use the same frequency just fine.

In general, where did I put this list? Way over here. No, come back. Let's see, map coloring. A while ago, Eric told us about the four-color theorem. That every map that you draw on a sheet of paper, you can color all the regions in different colors so that two neighboring regions never have the same color. And four colors is always enough.

This is just saying that certain kinds of graphs can always be four-colored. Not every graph, but certain kinds of graphs, the so-called planar graphs can always be four-colored. This is the language that that proof is written in. It's a theorem about graph theory. It uses graph theory. But it's an example, we've already seen, which is nice.

In general, any time you're trying to avoid pairwise conflicts, graph coloring is a great tool to reach for. Because it's such a great tool to reach for in many contexts, wouldn't it be great if there was a really fast, simple algorithm that says, here's a graph. Here's a target number of colors, please give me a proper coloring that uses this number of colors. So an algorithm to K color any input graph, or decide it's not possible.

If I'm the registrar, and I know I have-- what is it-- five days of exams, two sessions each. So I have 10 different exam sessions. I really want a 10-coloring of my graph. And so I want to ask, please give me a 10-coloring of this conflict graph. And then, that's my schedule. Is there an algorithm to do that?

Likewise, instead of asking give me 10 colors, you can ask the algorithm to compute chi of G. I don't know in advance how many colors I'm expecting. Just give me the best coloring. Give me a coloring that uses the fewest number of colors. Is there an algorithm to do that? That would be really great in lots of applications.

Unfortunately, I'm talking in hypotheticals here because these algorithms very, very likely don't exist. These problems are what we call NP-- well, this problem is NP-complete. This one is possibly even harder. At least, this problem is NP-complete. Like I said, this one might be even harder than NP.

What does this mean? I don't care so much. I don't really want to dig into what NP-complete means right now, or NP-hard, or any of those things. What I will say is that there are thousands upon thousands of computational problems, like graph coloring, like the traveling salesman problem, if you've heard of that one, it's pretty famous. Like Sudoku on n squared by n squared boards instead of just 9 by 9. There are thousands upon thousands of these problems that computer scientists have decided are all equally difficult.

So if we were able to find an efficient algorithm to solve any one of them, like to solve the K coloring problem, then that would immediately give us an algorithm to solve all the other ones efficiently as well. And all the problems that are easier than those.

A notable problem that we've already talked about is factoring large numbers. This is known to be in NP-- it's not known whether it's NP-complete. It's in fact thought to be even easier than most of these. So it's thought that factoring numbers is somehow easier than K coloring your graph.

Also, by the way, even if we're just asking for three colors, does this graph have a coloring with just three colors. That's already NP-complete. That's already too hard, and we don't believe there's going to be an efficient algorithm for it.

So if you solve this easier problem, factoring large numbers, if you find an efficient algorithm for that, well, you'll be able to break all the public RSA keys. Read everyone's encrypted messages. Steal all the electronic money, including the real currency, from electronic banks. Run from all of the world governments that are trying to hire and/or silence you.

But if you solve the even harder problem of three-coloring a graph, you get all that plus $1 million prize for solving a famous millennium problem. So you've got choices. Point is-- graph coloring is a hard computational problem. And most computer scientists strongly believe that there is no efficient algorithm for graph coloring. And so, at least for very large graphs, it's not realistic to ask for a coloring with a specific number of colors or the best possible coloring, which means that our applications are a little bit sad because sometimes we really want optimal colorings.

Thankfully, in practice, optimal colorings aren't really necessary. Sometimes we can get away with good enough. And so for all of these NP-complete or NP-hard problems, the best you can hope for is an algorithm that's good enough. Usually, the answer it gives us a little worse, but it runs faster. And let's see an algorithm for that to give us some colorings of graphs, even if they're not always optimal colorings.

All right. So we have a greedy coloring algorithm. And here is the idea of the algorithm. So first of all, we're going to order the vertices-- V1, V2, Vn. So we're going to pick some ordering of the vertices.

We're going to order the colors. I'm going to use my favorite color palette-- 1, 2, 3, and so on. Those are my colors. And then, for each Vi in order-- so we'll look at V1, then V2, then V3 and so on, and we'll assign each of them a color one at a time. Assign Vi the smallest color that doesn't introduce conflicts.

So when we get to vertex i, we're going to ask, does color one work? Does color two work? Does color three work?

And the first time we get to a color that is valid, that it doesn't have a neighbor of that same color, we're going to assign it that color and then move on to the next vertex. Yeah, it's called greedy because you're always picking the smallest color you can at any one time. It doesn't necessarily mean that you're going to get an optimal coloring. In fact, usually you won't, at least for big graphs where it's hard to find optimal colorings.

Let's run through this algorithm. Let's see it happening just to get a more concrete feel for what we mean. Here we go. So here's my graph. No, pause. Thank you.

Six vertices-- 1, 2, 3, 4, 5, 7 edges. And this is the vertex order we're going to use. We could choose any vertex order we want. But we're going to look at the vertices in this order and always choose the smallest color that's valid.

So the first color is fine because there are no possible conflicts. No one else has colors yet. Second vertex, the first color is no good because it already has a neighbor that's been assigned that color. So instead of the color triangle, we have to go up to color square. All right?

Now, let's go to vertex three. Our first color is valid this time, so it gets color one. Still with me? All right, vertex four can't use color one but can use color two.

Vertex five, let's take a look. Vertex five can't use the first color. Can't use the second color, but can use the third color, so it gets assigned to the third color. And finally, vertex six gets assigned the first color. And we have our valid coloring or proper coloring. Does that algorithm make sense? Awesome.

One thing I want to mention is that different vertex orderings will often lead to different colorings and often lead to colorings with a different number of colors. So here with this ordering, we got a coloring with three colors. We got a proper three-coloring.

But if instead we use this ordering, where all I've done is changed the order of vertices one, five, and six, this is a different vertex order on the same graph, so let's run the algorithm again in this ordering. And let's see what coloring it gives us. So first color is fine here. First color is fine here.

This one needs the second color because it already has a neighbor with the first color. Stop me if I'm going too quickly. The next one has neighbors with colors one and two, so it needs the third color.

The fifth vertex has a neighbor with color one, so it needs the second color. And then, finally, vertex six has a neighbor with color one and a neighbor with color two and a neighbor with color three, so it in fact needs a fourth color. So with this different vertex ordering on the same graph, it gave us a four-coloring instead of a three-coloring. All right? So some orderings are better than others.

Yeah? All right. Does this algorithm actually work? Does it always give a proper coloring?

So yes, in this case, it does. And an easy way to see that is that the way the algorithm is designed, it never introduces conflicts. Always after we've assigned the next vertex, we never have an edge that has two endpoints of the same color because that's kind of the point of the algorithm.

So we never introduce a conflict on any edge. So when we get to the end, we haven't introduced any conflicts. So there aren't any conflicts. That's kind of a state machine style proof of that algorithm. Yes?

AUDIENCE: Could you try ordering like the middle one once you get the optimal [INAUDIBLE]?

ZACHARY ABEL: Good question. If you try all the orderings and then take the best coloring you get, will that be optimal? So there are two questions in there.

Can you get all colorings from this algorithm? And the answer is no. You can't get all colorings from this algorithm. For example, colorings that never use color one you can't get from this algorithm because you're always going to use color one on the first vertex.

But the other question, can you always get some coloring that uses the minimal number of colors? Even if you can't get all different colorings, can you at least get all different numbers of colors? And that answer is yes. It takes a little bit of thought to prove, but the answer is yes.

So if you do try all n factorial vertex orderings and take the minimal coloring, you will, in fact, get the chromatic number. But n factorial is a runtime that is not considered efficient for things like NP and NP hardness. So that is absolutely an algorithm to compute chromatic number, just not an efficient one. Yeah, great question.

Let me prove one thing about this greedy algorithm. As we said, it's not always going to give the smallest colorinc depending on what order you pick. But we can prove something about the quality of colorings we get. And here is that theorem.

So for all graphs G, for all vertex orderings, if the max degree of our graph is less than or equal d-- by max degree here, I mean, look at the degrees of all the vertices and take the largest one. So the maximum of the degrees of the vertices. So if the max degree is at most d. Said differently, if all vertices in the graph have degree at most d, then the greedy algorithm returns a proper coloring with at most d plus 1 colors.

So if all the vertices of the graph have degree at most d, then the greedy algorithm will use at most d plus 1 colors. Does this statement make sense? So in this example, all the degrees are three or less. So we're guaranteed to produce a coloring that uses at most four colors.

And I want to emphasize this depends on the degrees of the vertices, not the number of vertices. If this graph hac thousands and thousands of vertices, but all the degrees were three or less, we would still get a coloring with at most four colors. All right? It's not necessarily optimal. In this graph, it wasn't optimal. We found a three-coloring, whereas this only promises a four-coloring, but at least it's some measure of quality of the results we get out of this greedy algorithm. Does the claim make sense? Wonderful. Let's prove it.

All right. We're going to prove this by induction. But induction on what? We might think, OK, well, there's a d in the theorem statement, so what if we induct on d? Well, that means at some point we know something about all graphs whose degrees are at most eight. And we need to prove something about all graphs whose degrees are at most nine.

I don't really see how to get from one to the next. That doesn't seem easy to me, right? Graphs with max degree nine look very different from graphs with max degree eight.

If you have a graph with max degree nine, there might be many, many, many vertices with degree nine. But to get anything useful out of our inductive hypothesis, we would need all degrees to be eight or less. There is a disconnect there. I don't know how to make that proof work.

So let's see induction on what? We just said induction on d may be not such a good idea. Often when proving a statement about graphs by induction, there are two common strategies to try. You can try inducting on the number of vertices or inducting on the number of edges. I'll explain what that means in just a second.

But these are going to be your best friends. If you're trying to prove a thing about graphs by induction, first, try proving it by induction on the number of vertices. Then try proving it by induction on the number of edges. If both of those don't work, I don't know. Cry. That's what I would do.

But let's try this strategy. Let's induct on the number of vertices in our graph. And here's what that means. So we're going to let p of n be the statement that all graphs with n vertices-- so all graphs with exactly n vertices-- and max degree at most d result in at most d plus 1 colors from the greedy algorithm.

So this is what we mean by we're inducting on the number of vertices. Because n, the variable we're going to be inducting on, is measuring the number of vertices of our graph. So if we think about induction as proving things one step at a time, we're going to first prove something about all graphs with one vertex, then all graphs with two vertices, then all graphs with three vertices, then all graphs with four vertices. This is fun. It's 5, 6, 7.

We're going to build up one size at a time. And that's how our induction is going to go. All graphs with n vertices is the key phrase here that makes this inducting on the number of vertices. If you want to do induct on the number of edges, you would say all graphs with n edges or something like that. All right? Does this setup make sense?

So we've defined Pn here. Our goal-- we want to prove for all n greater or equal 1 P of n is true. Yeah? So for any number of vertices, all graphs with that many vertices do the thing. So together, all of these Pn's, starting at P1 and going upward, include all finite graphs. Does that make sense?

I failed to mention earlier, but most of the graphs we're talking about in this class are going to be finite. So if I don't specify, go ahead and assume it's finite. So this will cover all of the finite graphs. So it'll cover all the graphs we care about. And we'll end up proving our theorem.

All right, so let's actually prove our theorem. Base case P of 1. Well, if there's only one vertex, we know what the greedy coloring is going to do. Well, let's see, max degree is zero because there are no edges if there's only one vertex.

Greedy coloring uses one color. One is 0 plus 1, which is what was promised. So P of 1 is fine. Let's do the inductive step, P of n-- so here, assume P of n. We want to show P of n plus 1.

And I really want to emphasize one point right here. Very often in this class, students come away from the next couple of lectures with the belief that induction on graphs is different. It follows different rules. It's backwards. We're going from n plus 1 to n or something like that.

And I really want to emphasize induction on graphs is using regular induction in the way we're used to. We're not changing the rules here. What changes is that what it means to prove P of n implies P of n plus 1 is a little different from what we're used to. We're still proving that P of n implies P of n plus 1, because that's what induction tells us to do.

But if we use our proof outlining techniques from the first couple lectures, and really unpack what it means to prove that this implies that, what it tells you to do is often a little bit different from what your intuition might tell you you're supposed to do. So it's doubly important when inducting on graphs to lean heavily on our proof outlining skills. Make sure that we're setting up our proof correctly to prove Pn implies Pn plus 1, and not something that we just think means Pn implies Pn plus 1.

So let's do that now. We're assuming P of n. In other words, we're assuming all n vertex graphs with min degree at most d produce at most d plus 1 colors. That's P of n. We're assuming P of n.

And we want to show that all n plus 1 vertex graphs with min degree at most d produce at most d plus 1 colors. All I've done is write out what Pn and Pn plus 1 mean in context. With me so far?

All right, and here's where the confusing bit happens, or at least the non-intuitive bit. Our task is we want to prove that all n plus 1 vertex graphs do some stuff. So the theorem we're trying to prove is a for all. All n plus 1 vertex graphs do something.

But we know how to prove a for all. We start with a generic thing in that set, and we prove the conclusion for that thing. So this for all immediately tells us that the next line of our proof should probably look something like this.

Suppose G is an arbitrary n plus 1 vertex graph. Yes?

AUDIENCE: For writing out the meanings of P of n and P of n plus 1, did you mean to say max degree instead of min degree?

ZACHARY ABEL: Absolutely. I meant to say max degree instead of min degree. That is a typo. Thank you so much. I guess it's a write-o because I'm not typing.

AUDIENCE: For both?

ZACHARY ABEL: For both, yes. Thank you. Thank you for catching that.

All right, is everyone OK with the fact that this for all, all n plus 1 vertex graphs tells us that this next line should start with suppose G is an n plus 1 vertex graph? Because this is the part that many students get wrong. They will often say suppose G is an arbitrary n vertex graph. And then they go on to try to prove stuff about some n plus 1 vertex graphs, but they haven't shown that they're really getting all of them. It's this for all that tells us we should start like this and no other way.

All right, so what do we do next? So vertices V1, Vn, Vn plus 1, so we have whatever vertex order we're using. Now, the greedy algorithm will color V1 through Vn as if Vn plus 1 and its edges didn't exist. The algorithm never looks at edges that connect to vertices that haven't been colored yet.

So as far as the greedy algorithm is concerned, coloring vertices V1 through Vn, so all but one of the vertices in our graph, is going to look just like coloring an n vertex graph. Yeah? So this, what's called a sub-graph on V1 through Vn, if we take just the first n vertices, make a sub-graph of just its vertices as our vertex set and the edges between them as our edge set with all their edges, this sub-graph also has max degree d. Oh, sorry.

We're supposed to prove something about all n plus 1 vertex graphs with max degree d. So assume G is some n plus 1 vertex graph with max degree d. Then, when we look at the sub-graph, where we're only taking some of the vertices and some of the edges, well, degree can't go up when you do that. So this sub-graph also has max degree at most d.

OK. By P of n, because we assumed P of n, these n vertices will use at most d plus 1 colors. We have an n vertex graph that we're talking about. We know it satisfies the property, so we get the conclusion because of our assumption. So it uses at most d plus 1 colors.

So now, all we need to talk about is the last step of the greedy algorithm. Last step for Vn plus 1. Well Vn plus 1 has degree at most d. So it has at most d colors that conflict.

So one of the first d plus 1 colors will be sufficient. Uses 1 through d plus 1 uses one of these colors because you can only have d conflicts because this has degree at most d. Therefore, the last color also uses one of the first d plus 1 colors. So the whole coloring uses that many colors. And we're done with the proof.

Did that make sense? If it didn't, come up, talk to me afterward. I'm happy to stay and answer questions. Thank you so much. We'll see you on Thursday.