[SQUEAKING]

[RUSTLING]

[CLICKING]

JASON KU: OK, let's get started. Welcome to the 12th lecture of 6.006. This is our second lecture talking about weighted graphs, and in particular, weighted shortest paths, algorithms. Last time we talked about weighted graphs. This is a kind of a generalization of what we mean by distance in an unweighted graph instead of each edge having a weight of 1, essentially. We generalize that to be any integer.

And last time, we showed how to solve shortest single-source shortest paths in a graph that doesn't have cycles even if it has 0 or negative weights in linear time using an algorithm called DAG relaxation. We also showed in that lecture how in linear time, if we are given the shortest path weights to all the things reachable in finite-- or with shortest path distance that's finite, we can construct a shortest paths tree from those weights in linear time.

So this is motivating why we're not really going to talk about parent pointers for the next couple of lectures. We're just going to concentrate on the shortest path weights. And so today, we're going to be talking about our most general algorithm we'll be showing for solving single source shortest paths, in particular in graphs that could contain cycles and could have negative weights.

So just to recap our little roadmap here, single source shortest paths in linear time. Last time we discussed another linear time algorithm, DAG relaxation. And today we're going to be talking about Bellman-Ford, which isn't limited to asymptotic graphs. In particular, there could be negative weight cycles in our graph.

If it has cycles, if it has negative weights, the worry is that we could have negative weight cycles, in which case there-- if a negative weight cycle is reachable from our source, then the vertices in that cycle and anything reachable from that cycle will potentially have an unbounded number of edges you need to go through. There's not a bound on the number of edges for a shortest path, because I could just keep going around that cycle as many times as I want and get a shorter path. And so we assign those distances to be minus infinity.

So that's what we're going to do today in Bellman-Ford. In particular, what we're going to do is compute our shortest path distances, the shortest path waits for every vertex in our graph, setting the ones that are not reachable to infinity, and the ones that are reachable through a negative weight cycle to minus infinity, and all other ones we're going to set to a finite weight.

And another thing that we might want is if there's a negative weight cycle in the graph, let's return one. So those are the two kinds of things that we're trying to solve in today's lecture. But before we do that, let's warm up with two short exercises. The first one, exercise 1, given an undirected graph, given undirected graph G, return whether G contains a negative weight cycle.

Anyone have an idea of how we can solve this in linear time, actually? In fact, we can do it in order E. No-- yes, yes. Reachable from S. I guess-- let's just say a negative weight cycle at all. Not in the context of single-source shortest paths.

AUDIENCE: Detect whether there's a negative weight edge?

JASON KU: Ah. Your colleague has determined an interesting fact about undirected graphs. If you have a negative weight edge in an undirected graph, I can just move back and forth along that edge. That's a cycle of length 2-- or I guess three vertices back to where we came from. There is of negative weight, because I'm just traversing that weight over and over and over again.

So the question of single-source shortest paths of finding negative weights is not particularly interesting in the undirected case. What I can do is just for every negative weight edge, undirected edge in my graph, I can just find the readability from the vertices-- the endpoints of that edge and label them as minus-- basically if the connected component containing S has a negative weight edge, then everything in the graph is accessible from a negative weight cycle.

So this is not such an interesting problem. And so we're going to restrict our discussion today to directed graphs. So this is if and only if exists negative weight edge. OK, exercise 2, kind of a little preview for what's to come, we're actually not going to show you an algorithm directly that meets this Bellman-Ford running time, V times E. What instead we're going to show you is an algorithm that solves single-source shortest paths in--

So given an algorithm, Alg A, solves single-source shortest paths in order V times V plus E time. OK, what is that? That's V squared plus V times E. That's close to what this V times E is. That's what we're going to show you. But if I had such an algorithm, can anyone tell me a single-source shortest paths algorithm-- how we can use this algorithm to solve single-source shortest paths in just V times E time? Show how to solve SSSP in order the V times E. I guess we can put a dot there as well.

So this is a little tricky. It's kind of related to the difference we had between the reachability problem and the single-source shortest paths problem that we saw last lecture. When are these asymptotically different in their upper bound is when V is asymptotically larger than E. But the connected component containing S can have at most E vertices, or order E. It can actually have at most E plus 1 vertices, because otherwise it wouldn't be connected.

So, what we can do if we had such an algorithm, we could first, when we're giving our graph, explore everything in the graph using BFS or DFS, find all the things reachable from S, and then just throw away everything else. Now I have a graph for which V is asymptotically no bigger than E, and then we can use this algorithm to solve single-source shortest paths in V times E time. I'm not going to write all that down here. You can see it in the notes. Yeah?

AUDIENCE: Does this work if your graph isn't simple?

JASON KU: Does this work as your graph isn't simple? I haven't thought about it. We are not going to talk about non-simple graphs in this class, but probably not because you've got a lot of edges. Though in our class if we're talking about single-source shortest paths, if we have multiple edges between two vertices, we can just take the minimum weight one because it's never better to take the larger ones. Does that answer your question? Great.

All right. So those are our warm-ups, that's our goal. We need to find an algorithm for single-source shortest paths. And general graphs, graphs with-- potentially graphs with cycles, and negative weights, and solve it in this V times linear kind of time. That makes sense? All right.

So first, before we get to the algorithm, we're going to discuss a little bit about simple short-- about shortest paths in general. If we didn't-- the problem here is negative weights. How do we find-- if we had negative weight cycles, there seems to be these problems, because we could have minus infinities is in our deltas.

But if we didn't have negative weights, I'd like to assert to you that our shortest paths, even if there are negative weights, are going to be simple. They won't repeat vertices. So that's the first thing we're going to show you. Let's see. Simple shortest paths. OK.

So, claim. I'm going to give my claims numbers today just because I'm going to have a lot of them. If my shortest path distance from S to some vertex is finite, meaning it's not infinite or minus infinite-- some finite value, there exists a shortest path-- a shortest S to V path that is simple. And remember, simple means not going through a vertex more than once. All right. How are we going to prove this?

Well, consider if this claim was not true. If every shortest path contained a cycle, essentially. It repeated a vertex. Then my path looks something like this. I mean, there's some vertices along here, and then I go to V. So here's S, and then there's some cycle I repeat, some vertex. I'm going to call this cycle C.

Now what do I know about this path? I know that it has-- it's a shortest path and it has finite weight. So in particular, this path-- this delta distance is not minus infinity. But if this is not minus infinity, what do I know about the weight of this cycle?

AUDIENCE: It's not negative.

JASON KU: Yeah. It can't be negative. Because if it was negative, I could keep going around this cycle, and this would have a non-finite weight. Shortest path distance from S. So I know this is-- can't be negative, so it must be 0 or positive. But if it's 0 or positive and this is a shortest path-- went through this cycle, then I could remove it, and now I have a new path with one fewer cycle. I could just keep doing this to create a simple path. So that checks out. OK.

So, that's interesting. If it's simple, what do we know about the number of edges in a simple shortest paths? How many could there possibly be? How long in number of edges could a simple shortest path be? If I can't repeat vertices, I can have at most vertices on my simple path, which means I can use at most V minus 1 edges-- fence posting.

So, simple paths have at most V minus 1 edges. That's a nice little thing I'd like to box off. That's a really nice property. So while a shortest path here could have an infinite number of edges, if the shortest path distance is finite, I know I only have to check paths that use up to V minus 1 edges. In particular, this is finitely bounded in terms of the number of paths I have to consider. It's exponential, potentially, but at least it's finite. The other way, I potentially had to check every possible path of which there could be infinite if there's cycles in my graph.

OK. So I have an idea. What if I could find shortest-path distances by limiting the number of edges I go through? So not the full shortest path distance from S to V, but let's limit the number of edges I'm allowed to go through, and let's talk about those shortest-path distances, just among the paths that have at most a certain number of edges. I'm going to call this k-edge distance. And I'm just going to provide a little notation here. Instead of having a delta, I'll have a delta k here. That means how many edges I'm limited by.

So from S to V is shortest S to V path using at most k edges. Short-- weight-- weight of a. Shortest path, shortest S to V path using at most k edges. And these notions seem somewhat symmetric. If I were able to compute this thing for V minus 1, then-- for all the vertices, then if the distance is finite, then I'll have successfully computed the real shortest paths because of this statement.

Now that doesn't mean that if this is-- it doesn't mean the other way. If this is minus infinity, if the shortest-path distance is minus infinity, it doesn't say anything about what this is. It just says the shortest path using at most V minus 1 vertices, using at most V minus 1 edges would be whatever this is. But really, the shortest path length needs to consider an infinite number of edges. So it doesn't really tell us much about that.

But for the finite ones it does. It works well. And so if we are able to compute this thing for k equals V minus 1 in a graph that doesn't contain negative weight cycles, we'd be done. I claim to you a stronger statement, that if the shortest path using at most V edges from s to v Is less than-- strictly less than delta of V minus 1-- this is all in the subscript here. Basically this is the shortest-path distance of any simple path, and possibly ones that also contain cycles, but definitely it includes all the simple paths.

If there's a shorter path to my vertex that goes through more than V minus 1 edges, that this path can't be simple, because it goes through a vertex more than once. Otherwise it would be included in this distance set. So if this is the case, and I found a shorter path to V that uses v edges-- yeah, that use V edges, that path can't be simple, which means that path or some path there contains a negative weight cycle.

So if this is true, then I know that the real shortest-path distance from S to V must be minus infinity. I'm going to call such a vertex a witness. If we can find a vertex that has this property-- I mean, I haven't shown you how to compute these things yet, but if I were able to find a vertex V-- and these are capital V's if you're having trouble. This V is different than this V, this is cardinality.

If we can find such a vertex V, that certifies that there is a negative weight cycle in our graph. So I'm going to call V is a witness. OK. So, if this property is true, it's a witness and it definitely has this. Is it possible, you think-- I'm going to claim to you that it's possible that a vertex could have minus infinite distance but not have this property halt. I could probably give you an example-- I don't have one off the top of my head right now, but that's possible.

You could imagine, there might be no path going to a vertex on a negative weight cycle that goes through V exactly V edges. It might go through more edges, a shorter one. So this equation would be inequality and would not certify that this is true. But I claim to you, if a vertex has this property, if it's its shortest path distances minus infinite, then it must be reachable from a witness. So that's the claim. If delta S, V is minus infinity, then V is reachable from a witness. Reachable from a vertex that has this property-- that has this property.

And if it's reachable from something that has minus infinity shortest pathway, then I can take that path go to my reachable vertex, and that's also minus infinite path. OK. So how do we prove this? Well, let's consider-- let's I'm going to state a somewhat stronger statement that we'll prove instead.

It suffices to prove that every negative weight cycle contains a witness. If we are to prove that, then every vertex with this property, every vertex with this property is reachable from a negative weight cycle by definition. So, if we can prove that every-- prove every negative weight cycle contains witness.

If we can prove that every negative weight cycle contains a witness, then every vertex reachable from one of those witnesses-- in particular, reachable from the negative weight cycle-- has shortest distance minus infinity, and that should prove the claim. This thing has to be reachable from a negative weight cycle. And so if we prove negative weight cycles contain witnesses, then all of these vertices are reachable from a witness. OK, great, great. Confusing myself there for a second. OK.

So let's consider a negative weight cycle. NG. Here's a directed negative weight cycle. Recall. This will be my negative weight cycle C. All of the sum of the edges in this thing, the weights has negative weight. And I'm going to have a little bit notation-- if I have a vertex V here, I'm going to say that its predecessor in the cycle, I'm just going to call it V prime. That's just some notation.

All right. So, if I have computed these shortest-path distances to every vertex in my graph, shortest-path distance going through at most V vertices and the shortest path distance going through at most V minus 1 vertices, then I know the following thing holds. Delta V going from S to V for any vertex in my cycle can't be bigger than delta V minus 1 from S to U plus the weight-- sorry, not U-- V prime, its predecessor, plus the weight going from the predecessor to my vertex.

Why is that? Why is that? Because this is the weight of some vertex-- this is the weight-- the shortest-path distance to my predecessor using one fewer edge. And so this in particular is the weight of some path that uses V edges. So if this is the shortest such path distance, this has to upper bound it at least-- at most. Yeah?

AUDIENCE: Is that the triangle inequality?

JASON KU: That is a statement of the triangle inequality, thank you. All right. So, yes, this is just by triangle inequality. OK. Now what we can say is, let's take this equation summed over all vertices in my cycle. So I'm just going to add summation here of all vertices in my cycle of this whole thing. I'm going to do that out a little bit neater.

Summation of delta, not d. Delta V S, V. I guess I don't need this open parentheses. Equals-- or less than or equal to sum of V and C of delta V minus 1 V prime. And here, I'm summing over V and C, and this is just my notation for the predecessor.

And then I'm going to sum over the weights in my cycle V and C. These are the sum of the weights in my cycle. Well, what do I know about this cycle? This is just the weight of C. The weight of C-- that's awful handwriting. C, what do I know about the weight of the cycle? It's negative.

So, this is less than 0, which means that if I remove this, this needs to be a strict equality. But if the sum of all of these is strictly less than the sum of all these, we can't have none of the vertices in my graph satisfying-- not satisfying this property. If all of them are not witnesses, then this thing is bigger than this thing-- at least as big as this thing for every vertex in my cycle, which is a contradiction.

So, the claim holds, if we have a negative infinite shortest-path distance, then V is reachable from a witness. So it suffices for us to find all the witnesses, find all the vertices reachable from the witnesses, and then mark them as minus infinity. Does that make sense? OK. So, now we finally are able to get to our algorithm. Bellman-Ford.

And what I'm going to show you today is a little different than what is normally presented as Bellman-Ford. The original Bellman-Ford algorithm does something a little different. And because it does something a little different, which we'll talk about at the end, it's a little hairier to analyze. I'm going to show you a modification that is a little easier to analyze and has this nice property that we're going to be able to use the algorithm to give us a negative weight cycle if it exists.

So, we're going to say this is maybe a modified Bellman-Ford. And the idea here is to make a vertex associate-- make many versions of a vertex. And I want this version of the vertex to correspond to whether I came here using 0 edges, 1 edge, 2 edges, 3 edges-- I have a different vertex version of the vertex for each one of these-- for a path going through, at most, a certain number of edges.

OK. So this is an idea called graph duplication. Idea, graph duplication. And this is a very common technique for solving graph-related problems. Because essentially what I get to do is I get to store information. If I'm having different versions of a vertex, I can have that vertex correspond to reaching that vertex in a different state.

So that's what we're going to do here. The idea here is make V plus 1 levels-- basically duplicate vertices in our graph-- where vertex Vk in level k represents reaching vertex V using at most k edges.

OK, so this definition seems similar to what we're doing up here. If we have vertices that have this property, then their shortest paths in this new graph might correspond to these k edge distances. And really, the name of the game here is to compute these two for every vertex, because then we can-- then if d is finite, delta is finite, then this guy will be the length of our shortest path. And if they are different, that will be a witness and we can explore from it.

So-- and if we connect edges from one level to only higher levels, basically levels with a higher k, then this graph is going to be a DAG. Whoa. That's cool. Why is that cool? Because we saw how to solve single-source shortest paths in a DAG and linear time.

Now this graph that we're going to construct is going to have V plus 1 levels. So could have-- our graph kind of explodes V times. We're going to do that in a second. I'm going to be more precise with what I mean there. But if we're multiplying our graph V plus 1 times, then the size of our graph is now V times larger. Now that doesn't-- that's not so hard to believe. But if we made our graph V times larger and we ran a shortest path algorithm in linear time with respect to that graph, then that graph has something like size V times V plus E size. That looks familiar, maybe? That's this running time. So if we can find an algorithm that runs in that running time, we can get down to V times E.

So let's try to do that. Here's the transformation I'm going to show you. I'm going to show you first with an example. Here's an example of a directed graph that does contain a negative weight cycle. Can anyone find it for me? bcd. Has weigh minus 4 plus 3 minus 1. It has a minus 2 total weight. So that's a negative weight cycle.

So in order to take shortest paths from a, I will want to say at the end of my algorithm, this better be 0, and all of these better be minus infinity. So that's what I want in my algorithm. So what's my algorithm going to be? I'm going to make V plus 1 copies of this graph, and I'm going to kind of stretch it out. OK.

So here, I have V 0, 1, 2, 3, 4-- there are four vertices in my graph. So this is 1, 2, 3, 4, 5 copies of my graph. I have a version of vertex a for each one of those copies, a version of vertex b for each of those copies, c and d, et cetera. So I have this nice grid of vertices. And I'm not going to put any edges within a layer, within a level. Because then-- I mean, this graph has cycles. And I don't want cycles in my graph.

What I'm going to do instead is for every edge in my original graph-- for example, the edge from a to b, I'm going to connect it to the b in the next level. So a0 is connected to b1 with an edge weight of minus 5, just like in the original. And I'm going to do that for every edge in my graph, and I'm going to repeat that down all the way.

In addition, I'm going to add zero-weight edge from a0 to a1 or from every vertex all the way down the line. These are all zero-weight edges corresponding to-- I'm not going to traverse an edge, I'm just going to stay at this vertex. That's going to allow us to simulate this at most k edges condition.

Now if you take a look at paths in this graph from a0, our starting vertex, clearly none of the other vertices in that level are reachable from a0, just as we want. Because the shortest-path distance to any of these vertices using at most 0 edges should be infinite. I can't get there in 0 edges.

But then any path in this graph using at most k edges is going to correspond to a path from a0 to a vertex in that level, the corresponding level. So for example, if I had a-- if I was looking for paths 2b using at most three edges, any path-- a path from a0 to b3 in this graph would correspond to a path in this graph that uses at most three edges. so Let's find such a path.

So going from a0, b1, stay at b1-- stay at b, sorry. Yeah, that's a path using fewer than three edges-- or at most three edges. But there's another path here. Where is it? Going from a, a, a to b-- OK, that's not such an interesting one. That's the same path. So I might have more than one path in here corresponding to a path in there, but my claim is that any path in here corresponds to a path in here.

So what's a path of length? 3, that's non-trivial. Yeah, a to c to d to b. So a to c to d to b. Yeah, that's a path. And basically, because I constructed this so that the edges always moved from level to level, as I traverse these edges, I always change levels. Yeah?

AUDIENCE: But my original graph doesn't have these self-loops with 0 weight.

JASON KU: Yes. My original graph doesn't have an edge from a to a. That's true. I'm using these edges to correspond to-- I'm deciding not to take an edge. It's not that I'm like doing any work here, I'm just staying there for a state. And that's what's going to allow me to get this at most edges.

All right. So, this is the graph construct. Hopefully you understand that we made these V layers. This is V. And a vertex-- we made V copies of every vertex and connected them using edges in this way. OK. So, first step of Bellman-Ford is construct this graph. So, Bellman-Ford, construct G prime as described above. It has how many vertices? V times V plus 1. V times V plus 1 vertices.

And how many edges? Well, I have one edge for outgoing edge for each vertex corresponding to just staying in the same place. So that's V squared vertices-- I mean edges. And then I have one edge-- for every edge in my graph, I have-- sorry. I have a V minus 1-- sorry. Just V. I have V edges for every edge in my graph.

So that means-- so this is the number of vertices. And V times V plus V times E, this is V V plus E. All right, cool. So that's how many edges I have. So constructed in that way, it's a DAG. If we only have edges going to increasing levels, then this thing can't have cycles, because otherwise that would mean there would be an edge pointing backwards. And we didn't construct any of those.

All right. So we construct this graph G prime. We can do that in linear time with respect to these things. I just go through all the edges, I make these edges, and I make these vertices. It doesn't take anything-- I just do it naively. Right I can do that in time V times V plus E asymptotically. OK.

Now I run DAG relaxation, our nice algorithm we had last time, from-- in there was a0. I'm going to say it's S0, our source. Our source vertex. Single source shortest paths. So that I compute delta of S0 to Vk for all k and-- what is it? 0 to V. That's what single source shortest paths does. It computes for me this distance from my source-- at some source to every other vertex in the graph. And so in particular I get these. Well, that is all of them.

Then for each vertex V, set-- the thing I'm going to return, d-value, S to V, equal to the shortest-path distance I got from DAG relaxation to a particular vertex. V V minus 1. Why am I doing this? I'm setting it to be the shortest-path distance to the guy in the second-to-last row here or column in my modified graph.

The hope is that this distance in my DAG corresponds to this distance in my original graph. The distance to V using at most V minus 1 edges. So that's the claim-- that's a claim we're going to prove in just a second. I'm going to write it down just so that we have-- just to continue our train of thought.

Claim, delta S0 Vk equals delta k, the k edge distance, from S to V. That's what we want to claim. That would then-- what would that mean, then? That would mean that I'm correctly setting the shortest-path distance here for all vertices whose distances finite. Great. I mean, I set values to things where they're not finite, where they're minus infinity also, but in particular I set the ones correctly if they're finite. OK.

So the last thing we need to do is deal with these minus infinity vertices. But we know how to do that. We just look at the witnesses. Because we've computed this value for k equals V equals V minus 1, and if that claim over there is true, then those shortest-path distances are the same as these k edge shortest-path distances. And we can just, for every vertex, we compare these things. If this is satisfied, we got a witness. OK.

So for each witness U and V where delta S0 U V is less than, strictly, S0 U V minus 1-- that's the definition of a witness here, close the parentheses. Then for each vertex V reachable from U set-- sorry, d, is what we're returning, d of S, V equal to minus infinity. That's the end of the algorithm. Basically I'm looking for all the witnesses. For each witness, I find all of the vertices reachable from it and set it to minus infinity just as we argued before. OK.

So, it remains to prove this claim. How do we prove this claim? Well, we can induct on k. Is this true for k equals 0? Yeah. We kind of already argued it over here when we are talking about our initialization step or what DAG relaxation does. It'll set this to be the shortest path from this guy to all these vertices. These aren't reachable from here, and so these are infinite. And that one's 0.

So the base case-- so induct on k base case, k equals 0? Check. That's all good. Now we-- in our inductive step, let's take a look at the shortest-path distance from 0-- from S0 to V of k prime for some k prime. And the assumption is, the inductive hypothesis is that this distance is the k edge distance for all k prime less than-- I mean all k less than k prime.

Well, kind of by definition of a shortest path, this is the minimum overall incoming vertices of the shortest path from S0 to U of k prime minus 1 plus the weight of the edge from U of k prime minus to Vk prime for all Uk prime minus 1 in the adjacencies, the incoming adjacencies of Vk prime.

OK, what does this mean? I'm just saying in my graph G prime, a shortest path to this vertex needs to go first through some vertex in the layer before it, which is one of these. And in particular, I'm only connected to things adjacent to me. That's all this is saying. I have to go through that vertex and take some shortest path to one of those previous vertices.

Now s actuality, these adjacencies, I constructed them to be similar to the adjacencies in my original graph in addition to one edge coming from my original vertex, from vertex V. So this is the same as the minimum of this set delta S0. Same thing. Plus W U, V for all U in the adjacent-- incoming adjacencies of my original vertex.

In addition to one more term. What is that other term? These are all of the things corresponding to my incoming edges in my original thing, but I also have that one edge coming from the V before it. So this is-- I'm going to union-- union-- union this with delta S0 V of k prime minus 1. Awful. I think there's another one here. This is S0 of k prime minus 1. I"m not going to rewrite it.

OK. Then by induction, this thing and this thing must be the edge shortest paths using k minus 1 vertices. And then that's just the statement of what the shortest path should be using at most k prime edges going from S to V. So, these things are the same as we claimed. Yay, check.

All right. And then it's not such-- it's kind of a trivial leap, then, to say that at the end of Bellman-Ford, these guys-- sorry, the things that we return, these guys, are the shortest-path distances, because here, if they're finite, we set them to their true shortest-path distance; and if they're minus infinity, that invariant means that these things correspond to exactly this claim over here. It's a witness. And then finding all the vertices reachable from those witnesses, we set all of the infinite ones to be minus infinity as desired.

OK, so what's the running time of this thing? Well, we had to construct this graph, so we had to take that time. We ran DAG relaxation, that takes the same amount of time. For every vertex, we did order V at work. And then for each witness, how many could there be? And most, V. Checking the reachability of each vertex, that can be done in how long? Order E time. Because we don't need to consider the things that aren't connected to S-- or aren't connected to the witness.

So this thing takes order V times E work. So we're upper-bounded by this time it took to construct the original graph and by the claim we had before, that takes V times E time. OK. So that's Bellman-Ford. I'm just going to leave you with two nuggets. First, the shortest path, if for any witness-- let's say we have a witness here. Do I have any witnesses here? I didn't fill in all these. But is there a vertex on this cycle that goes through who has the shortest path? That goes through four vertices that's smaller than any other. OK.

I can go from a to c to b to d to b to c. And you can work out this algorithm-- I have it in the notes that you can take a look at. This will actually have a shorter path for vertex-- sorry. It'll have a shorter path for vertex b. a to b to c to d to b. Thank you. That's a path of length 4, of four edges. That has shorter path than any path that has fewer edges. In particular, there's only one other path to b using fewer than four-- there's two other paths. One path of length-- that has one edge, that has weight minus 5, and one path-- this path that has weight 9 minus 1 is 8.

Whereas this path, minus 5, minus 4, 3, minus 1 has minus 10 plus 3 is minus 7, which is shorter than minus 5. So b and d is a witness. And if we actually take a look at that path through this graph, going from a to b to c to d back to b we see that there's a negative weight cycle in this graph. b to c to d to b.

And indeed, that's always the case for our witnesses. You can see a proof of that in the notes, and you can see in recitation a little space optimization to make us not have to construct this entire graph on the fly, but actually only use order V space while they're going. OK. So that's Bellman-Ford. Sorry for running a little late.