PROFESSOR: Good morning everyone, let's get started. So part two of our two lecture sequence on network flow. So all the pain involved in the concepts and notation from Tuesday turns into algorithmic fun today. But we will do a little bit of a review just to make sure we're all on the same page with respect to all of these notions that we talked about on flow networks. So we call that a flow network is simply a directed graph, GVE, and each edge is going to have two numbers associated with it.

If you see something like 1:3, this means that this is the flow, and that's the capacity. We have a basic constraint dissociated with the flow not exceeding the capacity on any given edge. And we also have the laws of conservation that say that other than the sink in the source, which are the two distinguished vertices in G V E, you're going to have a situation where for any intermediate vertex, all of the flow entering the vertex has to leave the vertex. And with those two constraints you need to find the max flow in your flow network. The flow value corresponding to this potential max flow, or some flow, is F equaling f of s of V. And so s is the source. V corresponds to all of the vertices.

And you're looking at pushing flow from the source vertex to any of the vertices, and thus the V includes t, as well. And that this is essentially a flow value, and we showed-- this is implicit summation notation, we showed, using a little bit of algebra, that this equals f of V t. And that's going to hold because of flow conservation primarily. So that was not particularly surprising. Something that was a little more surprising, or is a little more surprising, is that you can have an arbitrary cut in the flow network, and a cut corresponds to any partition. S, T, such that s belongs to S, the source belongs to S, and the sink belongs to T, and that's essentially a cut.

And so obviously there could be exponentially many cuts in a given flow network. But the amazing thing is that you can show a lemma that says that, regardless of what the actual flow values are in any of these edges-- across any cut you're going to see this flow. And this is true for any cut S, T. The corollary to this lemma is that the flow is going to be less than or equal to the capacity of any cut. And that simply comes from the edge constraints. You know that the flow value corresponding to any particular edge has to be less than the capacity of that edge. So you obey that constraint and you just look at the edges that go from S to T, sum up those capacities, and your maximum flow or any flow is bounded, [INAUDIBLE], by that quantity. And that's our c S T.

So you now have a relationship between the minimum capacity cut and the max flow. We didn't actually exploit that in the sense of the algorithm itself, the operation of the algorithm. But we're going to actually exploit the notion of a cut when we prove this algorithm that I got to late on Tuesday's lecture that corresponds to the Ford-Fulkerson algorithm. I showed you the execution of it. In order to prove that it's correct we're going to need this notion of that bound on the flow value given any cut in the network. So the max flow related to min-cut.

But before we get to that. Just to catch you up on the algorithm that we actually got to, the Ford-Fulkerson algorithm, that was based on the notion of a residual graph. This is now a new graph, G f, which is V, E f, so it depends on the flow. You computed based on G, based on the given flow f. And this graph has edges that have strictly positive residual capacities which are defined as the C f u, v equals C u,v minus f u, v. And this needs to be strictly greater than 0 in order for the edge to exist in G of f.

So if this is zero then you don't have an edge between u and v in G of f. If it's one or greater assuming integral flow values, you will. And in the residual graph-- two more quick definitions. One of which you saw last time, and another that I just verbalized, is the notion of an augmenting path. And this is any path from s to t in G f. So if there's a path from S to T in G f you do not have a maximum flow. There is an augmenting path. You're going to be able to increase the flow corresponding to the f value, that gave you your residual network, G S of f.

And how much can you increase this flow by? How much can you increase f by? That is termed the residual capacity. And the residual capacity of an augmenting path is simply C f of P, that's what we're going to call it, C f of P, equals the minimum value of the residual capacities along the path. So this would be C f u,v, and that's pretty much it.

And some examples should make this clearer. If we look at what the actual algorithm is, and go through a simpler example than we did last time, but it'll give you a sense of how augmentations are going to change the flow. How augmenting paths appear and eventually disappear as the flow increases and reaches the maximum flow.

So this simply corresponds to, as I said, walking through the augmenting path, looking at each of the residual capacities, and picking the Min value. So the Ford-Fulkerson algorithm, which is the first algorithm for max flow continues to be used, sets the flow on each of the edges to be zero. So you just set everything to zero. Clearly satisfies flow conservation, satisfies all the edge capacities. And you have literally three lines of code here, while an augmenting path in G f exists.

So there's a lot going on here, you have to compute G of f, I mean it's three lines of code, but the sub routines calls, which are fairly complicated, you have to compute g of f, given f, and initially it's all zeros. And then eventually it's going to change, or immediately after you discover an augmenting path. And now you have discovered an augmenting path. So you have to now do either breadth-first search or depth-first search, in order to discover whether this is a path from s to t because that's the definition of an augmenting path.

And then the last line of code says, augment f by C f of p. And again this is fairly involved. You have to take the path in G of f. You're going to increase the values of the edges that correspond to G of f translating those edges back to G. And there's possibly an inversion in direction that is associated with that translation. And so if you take a really straightforward example. Let's say if you have G is s, and this one vertex here, a, and then you have t. And right now I have 1:2 and 1:4. I now am going to compute G of f. So G of f is going to be-- I'm just going to write down s, a, and t because the vertices are all the same. The edges are obviously different.

Am I going to have an edge from s to a? Yes. And what number should I put on that edge? 1, and that's because it's 2 minus 1, not because it's just plain 1. And this edge here is going to have a residual capacity of 1, and that just comes directly from that 1. And what that means is, and this is important, what that means is that, in effect, this residual capacity says in this direction you still have a capacity of 1. Which means that in this direction you could reduce by one.

OK, that's the important thing to remember. And over here I'm going to have-- this way I'm going to have 3 and coming back I'm going to have 1. All right, so straightforward example but I think evocative in the sense that it applies the concepts that we've seen so far other than cuts. And we'll get to that. And what happens here, is there a path from s to t? Absolutely. And what is the residual capacity of the path from s to t?

It's one because you've got a 1 here, and you got a 3 here, you got to take the Min, and it's 1.

Right, I guess that's the number of choice today, one. But once you do that and you find that, you say, OK, what I'm going to do is-- and this is going to be pretty straightforward-- I'm going to look at these edges, and I know that I have c of t being a one corresponding to this value. And I'm going to go add one to those corresponding edges because these are the edges that actually shows in terms of the augmenting path. So the augmenting path is that.

And so this turns into s to a to t, and that becomes 2:2. This becomes 2:4 and that's it. So I now have G with the new f. So I can now think about what is G of f1 correspond to. The G is separate from f, so in some sense the G stayed the same, but the f changed, as you saw here. And now G f1 looks like s, a, t. What do I do between s and a?

Well, I only have something from a to s. I don't have anything from s to a because the residual capacity in this direction, thanks to the fact that the flow has saturated the edge capacity, is actually zero. But I do have an edge this way. And what is the value for that edge? It's 2. And over here, what are the numbers for the top and the bottom? 2 and 2.

And now I look at G f1, and I've gone through one iteration of this pseudo code, and I now try to find an augmenting path in G f1. And is there a path from s to t? No, which means I'm done. This is the max flow. You believe me? For this example, you believe me. You believe me, like, for every example? No, absolutely not because you haven't done a proof! We haven't done a proof! That's why you shouldn't believe me.

So we got to do a proof. So how is it that we know, given all of this, it's not because I told you so, that when we've converged. When we get out of this while loop, that we have a max flow. Looks pretty good. Worked for that example and it's going to work for every example, but that's not a proof.

So now we have to sort of take in all of the notions that we've talked about here, including primarily the notion of cuts. And the proof, the max flow, min-cut theorem, which is going to show this key result that we require, which is that when we terminate in the Ford-Fulkerson algorithm, we're going to have a max flow. And that's the reason why it's a maximum flow algorithm. If you don't have that proof, you don't have a max flow algorithm.

So hopefully all of that is clear. Pipe up if you have questions. And let's write out the max flow min-cut theorem, which I mentioned of it last time but never really got to even stating, but today we're going to state it and prove it. So this is an interesting theorem. I mean it's a legendary theorem. And it's not your usual theorem in the sense that it makes a simple statement. It actually makes three statements. It says the following are equivalent, and it's got three things which are the following. F equals c S T for some cut, S,T. This says that some cut is saturated. Right, so that's just the statement.

The second statement, which is equivalent to the first one, says that f is a maximum flow. So f being a maximum flow means there's some cut that's saturated. If there's some cut that's saturated for a given flow, you have a max flow. And then the last one, which is important for our Ford-Fulkerson algorithm, is that f admits no augmenting paths.

And so if I want to show that the Ford-Fulkerson algorithm terminates with a max flow, tell me, based on implications-- like i implies j, for numbers i and j, what it is that I need to prove. We're going to prove a bunch of other things along the way but crucially what is the i implies j that I want if I want to claim that the Ford-Fulkerson algorithm terminates with the max flow.

Yeah, you over there.

AUDIENCE: 3 implies 2.

PROFESSOR: 3 implies 2. That's right, 3 implies 2, exactly right. 3 implies 2. So we need to do 3 implies 2. Now of course, the theorem says that, 1 implies 2, 2 implies 3, 3 implies 1, et cetera. There's a lot of implications here. What we are going to do is we're going to show that 1 implies 2, 2 implies 3, and 3 implies 1. And that pretty much takes care of everything.

And it turns out the reason we do this is simply because it makes for the simplest proofs. 1 implies 2 and 2 implies 3 are one- liners, and that 3 implies 1 is a little more interesting and involved, but it's a little bit easier than directly doing 3 implies 2. So that's the way we're going to do this. You could certainly play around and do other things. So any questions so far? OK, so let's go ahead and do this.

All right, we should be able to knock these two, 1 implies 2 and 2 implies 3, in just about a minute each. So I want to show 1 implies 2, and essentially what I want to say here is, if I've saturated a particular cuts capacity then I have a max flow. And really, I mean this comes from the definitions, since I'm going to have f less than or equal to the c S, T, and this is simply because of edge capacity constraints. This is just edge capacities for any cut S T, the assumption that f equals c of S T implies f is a maximum flow because f can be increased.

And that's basically it. Right, so this is pretty easy. Next one is easy as well. 2 implies 3. If the rare and augmenting path-- so I'm going to do this by contradiction-- if the rare and augmenting path, the flow value can be increased. Because remember the augmenting path corresponds to a path with strictly positive residual capacities. The only reason there's an edge in there is because you have a greater than zero residual capacity.

So that means that the flow value could be increased by some small amount corresponding to C f u v, and each of these quantities that are in here-- I'm sorry, corresponding to min C f u v, that's the residual capacity. But you know that each of these capacities, C F u v, are strictly greater than zero. So the min clearly is strictly greater than zero. It might be some tiny quantity but it's greater than zero. Which means that the flow value could be increased contradicting the maximality of f.

So a little proof by contradiction gives you 2 implies 3. So, so far, so good. This is primarily based on the definitions of augmenting paths, residual capacities, et cetera. Now the fun starts.

Now we want to show really, I mean this is truly what the max flow min cut is because I really didn't have cuts up there. So what's min-cut about that, right? So I had a question at the end of Tuesday's lecture on how come the algorithm didn't use cuts. And, well, the algorithm doesn't use cuts, but showing that the algorithm converges to the max flow uses the notion of cuts.

So what we want to do is do 3 implies 1, and that's going to take care of what we want. And we're going to say, well, suppose f admits no augmenting paths. So what does that mean? Well, that means that you can't reach t from s. If you reach t from s, there's an augmenting path, so you can't make that move or make that path. So let's go ahead and, sort of, look at this a little more carefully and figure out what's causing this lack of connectivity.

There's a lack of connectivity between s and t. You can't actually get there. So there's like a chasm, there's a gap. And so what's causing that? And what we're going to do is figure out what the cut of the partition is that could be the reason why you have this lack of connectivity.

So we're going to define S which is exactly going to be our S from our favorite s t cut definition, but bear with me for just a second. Because I will tell you what S is. There exists a path in G f from s to u. So what I'm doing is I'm collecting the reachable vertices from S. And I know that I'm not going to have t, small t, in this set because f admits no augmenting paths so I can't possibly reach t from s. But I can collect all of the vertices that that can be reached from s in this given G of f and put them all into S.

And then my T definition is simply V minus S. And the key observation which I already have made but important to emphasize is that small t belongs to T, and obviously s belongs to S, therefore S T is a cut. It satisfies the definition of a cut. So we've set ourselves up. And then we have a key observation to make that I'm going to try and extract out or you. Let me move over here.

All right, so what I want to do is I want to draw this out, and I want to look at this as this is S, that is T. I know that S is over here, are s is over here, t is over here.

I'm going to pick some vertex v that's over here, some vertex u that's over here. I know that there's a path, and I'm writing this is a dotted path because this is a path in G of f. We're going to actually move between a G of f and G in this proof. So that's the only hard part of this proof. The movement between g and G of f. The edges are different between g and G of f. Got to keep that in your head. Because we're going to make arguments about that.

I know that all of the vertices that are on the left hand side are reachable from s, so I just picked an arbitrary vertex u, and I know that there's a path in G of f because that's the way I defined it from s to u. I also know that there is no path in G of f from u to v because if there was a path v would have been on this side, correct? So there's clearly no path in g of f from u to v. That's why v is over here.

But let's say that I'm picking something where there's an edge in G, the original flow network from u to v. So the edge that I've drawn here, the dark edge, is an edge between u and v in G. And that edge had a certain capacity. It existed. The reason it existed is because it had a nonzero capacity, correct? Because we originally defined our flow network to be something where-- if you only put edges in there, in the original flow network if the capacities were greater than zero. So that edge in there, in G had some capacity c u v which was greater than zero.

But this edge does not exist in G of f, correct? So what can I say?

Go ahead.

AUDIENCE: [INAUDIBLE]

PROFESSOR: Exactly. In the original graph, we have saturated this edge-- that was my best throw of the term. I like that. And you guys didn't notice, but maybe it'll be on video. Actually, it won't be on video unfortunately. But so I got this edge here in the original graph and I didn't get this dotted edge here because the residual capacity was zero. The original capacity was nonzero. The residual capacity is zero. The only reason that happens is because C f u v is zero since if C f u v greater than zero then v would belong to S, not v belonging to T as assumed. So that's essentially what we've determined here.

So this means that f of u v equals c of u v simply because c f u v equals c u v minus f u v which is 0. All right, so that's the statement I can make. Now, I did not have any constraint on u and v. I just said u belongs to s and v belongs to t. And for each of those things, I know that I can't have edges in the residual network between u and v which means that any edge that exists-- maybe there's no edge in the original network, but if there was an edge in that original network it's saturated according to this argument.

So every edge from S to T in the original network is saturated, and that essentially means that I've gotten to the capacity of my cut. That's essentially what that means. If I've saturated every edge, I've reached that capacity. OK, so that's it. All you do once you recognize that you had an arbitrary choice of u and v, you just say summing over all u belonging to s and v belong to t yields f of S T equals c of S T. All right, I'm on a roll not just with Frisbees. I finished the proof with a finger to spare.

So f of S T equals c of S T. All right, so that's exactly what we want. We are saying f of S T is obviously a cardinality of f so I've shown this thing over here. So that's why the Ford-Fulkerson algorithm works. It's because of this analysis that the Ford-Fulkerson algorithm works.

So are we done? What are we missing in algorithm design, our algorithm analysis? Not you, yet.

AUDIENCE: [INAUDIBLE]

PROFESSOR: Sorry?

AUDIENCE: A runtime?

PROFESSOR: Runtime, good runtime. I know you have too many Frisbees. You probably do, too. But so we haven't done-- we've done correctness, we've done conversions. We haven't done a complexity analysis. And so there's some bad news here. And erasing a pivotal moment in 6 over 6.

Why is this a pivotal moment in 6 over 6? It's on the Frisbee, exactly. This picture-- maybe not exactly that picture. It's not quite that picture but it's the same graph. It's a pivotal moment because this picture was famous because it was in a textbook but now it's iconic because it's on a Frisbee. We immortalized this picture.

So this is an example of really a failure of the Ford-Fulkerson algorithm. So it's kind of a bad example on this Frisbee but it's going to lead us to better algorithms. So what I have here is a weird flow network that has these strange capacities. I mean it's a fairly straightforward flow network. But at the as of the moment it obviously satisfies a flow conservation because all of the flows are 0 and therefore edge capacities. But we're going to now take a pathological execution of the Ford-Fulkerson algorithm.

Which, by the way, did not specify how you are going to select the augmenting path. It just said, find an augmenting path. And in passing, I said, overuse depth-first search, we'll use breadth-first search, and kind of hand wave my way through that. Now it turns out you can't hand wave when you want to run algorithms, right. You eventually have to code them up, you got to pick something. So people did. People picked different ways of selecting augmenting paths in the Ford-Fulkerson framework.

And they discovered that some of them would work beautifully and some of them would fail miserably on networks that were as small as this, and they'd just take forever. And this is before complexity analysis, kind of maybe before asymptotic complexity really came into vogue. So they just had empirical analysis to tell them that some techniques for discovering augmenting paths work better than other techniques on an empirical basis. So what could go wrong with Ford-Fulkerson on this example? What selection strategy corresponding to the augmenting path could cause Ford-Fulkerson to have a huge number of iterations? Go ahead.

AUDIENCE: So each time you pick an augmenting path you pick one that goes to the center.

PROFESSOR: Exactly, and so let's walk through a couple here. So the first one, let's just say this was a and b, to make it easy. So what is the first path that we picked? What is it--? Go for it. You're not off the hook yet.

AUDIENCE: You pick s to a to b to t.

PROFESSOR: Yeah, and then when you pick s to a to b to t, I'm not going to draw G of f for this, it's pretty clear that G of f is going to have-- I'm sorry, you end up picking s to a to b to t corresponding to the G of f that you've created. Right, so the G of f might have something like s to a-- there's only going to be an edge this way that's going to have 10 raised to 9. And then this way it's going to have 1, and that's going to b, and over here to t it's going to have 10 raised to 9. It's also going to have 10 raised to 9 this way, and it's going to have 10 raised to 9 this way.

And the reason the gentleman was laughing is because we ended up picking this path. S in G of f-- this is G of f just to be clear-- you pick s a b t. And so what you end up doing is you end up making this 1, making this 1, and coming over here making this 1. Wow! OK, you did increase the flow. Very good, made progress.

Now what happens? When you do this, certainly you'll get a different G of f. And the G of f is going to turn into-- well, this gets a little more interesting here. This is 10 raised to 9 minus 1. This is 10 raised to 9-- no, that doesn't change. This is still 10 raised to 9. This is 10 raised to 9 minus 1. And that this edge here also changes because basically what ends up happening is you end up going this way with 1. Did I get that right? Yeah? OK, good.

So now what's the bad path? s b a t. So s b a t would now say, oh, what I'm going to do is, I'm going to go ahead and make this 1. What do I do with that?

AUDIENCE: Make it 0.

PROFESSOR: Make it 0, exactly. Make it 0, and then this 1 becomes a 1. Shall we keep going? No. So 2 billion iterations for a graph with four vertices. Now that's performance for you. So this particular, potentially a depth-first Ford-Fulkerson on this example, depending on the ordering of vertices it's certainly quite possible that you would get to this particular order in the depth-first search on G of f.

The computers is dumb, it does what you tell it to do, it's doing depth-first search. It's producing these paths, and you end up with 2 billion iterations for this. And how many iterations do you really need if you did it right? Two. So that's a billion factor slowdown.

So this is a pathological example, a simple pathological example, to just show you what the problem is. But you can imagine that if you use depth-first search you might be a factor of five slower on average than if you use some of the technique. And a factor of 5 is nothing to be scoffed at, especially if you're running for minutes. And, you know, back in the day computers were horribly slow. So how is this problem fixed?

Well, any number of ways. But the first real way that took into account asymptotitc complexity, did analysis, and did all of that was due to Edmonds and Karp, which is a few years after Ford-Fulkerson. In fact, several years after Ford-Fulkerson.

And their contribution was not as much a new algorithm, though it is called Edmonds-Karp algorithm. It's really a small change to Ford-Fulkerson. You might want to think about it as the Edmonds-Karp analysis, which was kind of important, and we're not going to do that here. But you'll see it in section on Friday.

But the Edmonds-Karp algorithm slash analysis is that, if you use a breadth-first augmenting path, it's obviously just as easy to discover complexity-wise as the depth-first path. Then you could get a polynomial bound that did not depend on the capacities for the complexity and therefore iteration count of Ford-Fulkerson. So that was their contribution.

And so the breadth-first path is the shortest path in G f from s to t if you count where each edge has weight 1. So in that case you would get your two iterations. You'd get the shortest path of weight 2. Whenever I say weight, I mean just counting the number of edges. And these algorithms they actually recognized that breadth-first implementations of Ford-Fulkerson were the ones that ran faster than all the other ones. And then they said, wait, what's going on here? This looks pretty good.

And so they went off and did a fairly sophisticated analysis that showed that you had, and as I said you'll see that in section, that you only need order VE augmentations in the worst case. So this is what we know and love with respect to algorithmic complexity. Order VE augmentations, in the worst case, provided you used breadth-first augmentation. And so what is the overall complexity of the algorithm, let's call it the Edmonds-Karp algorithm, with this breadth-first search it would be-- order?

VE square, you could think about it as, let's assume that e is greater than V, and so we just say order V squared. So that was the Edmonds and Karp's contribution. This mean that an amazing amount of work over the past 30, 40 years on improving that bound. [? Denic ?] independently arrived at a similar analysis in bound, along with Emonds and Karp. They were the first to give polynomial bounds on max flow. And there was an algorithm that came out of MIT partly, or at least MIT alums, where you improve that. And so until about 2011-- and there's a lot of different algorithms depending on whether you're taking into account the value of the capacity, and you want to think about this is as some max value, and if you want to include that in the complexity then things change a bit. If you can limit that.

But assuming that those capacities are essentially unbounded, then the fastest algorithm in 2011 was King, Raul, and Tarjan. And that runs in order VE log of E divided by V log V V. Obviously, we need something for the log here. So that's kind of, pretty good here, right, because this is pretty small. If you really think about it, if E's large for dense graphs this is going to be a small constant. So that's pretty good. So that was the fastest until 2011.

Very recently, Orlin, who's at MIT, came up with an order v e algorithm and there are many variants here that use fast matrix multiply. And then just this past September, Aleksander Madry joined our department, and he has a variant algorithm that is better in some sense, not in the pure sense of V and E but there's a couple of constraints associated with it that are very mild. But it's in that sense better than order VE and I won't get to that. In the context of 6 over 6 we are obviously going to stick with Edmonds-Karp and Ford-Fulkerson in terms of what you're responsible for, but this as been a rich area of research for decades. All right, yeah.

[INAUDIBLE]

Oh, simply because the breadth-first augmentation takes order e. I'm just saying it takes order E time. So this was the order VE augmentations, in the worst case, each augmentation takes order E time, because breadth-first search takes order E time assuming that E is greater than V and I just added the E square factor. Thanks for asking. I'm sure a lot of other people had the same question. Yes, I just sort of skipped over that a little too quickly. That's why you get the order VE square.

And so that's a significant reduction if you assume that constant factors don't explode going from Edmonds-Karp to King, Raul, and Tarjan, and certainly Orlin, So now we got algorithms, we've proven them correct, we've done some analysis.

At tomorrow, in section, you're going to see, at the analysis that gives you the order VE. Which is a half hours worth of pain, should I say. Uh, no, a half hours worth of excitement! That corresponds to the order VE, that bound that we have. All right, so it's not that difficult. It's about the level of this thing over here, and maybe a little bit more involved. So, yeah?

AUDIENCE: [INAUDIBLE]

PROFESSOR: Breadth-first search takes order v plus e time. On a graph, breadth-first search is order v plus e. And I'm getting a little lazy here, and I'm paying for it, by putting in the v e square. But I should really have been-- if I wanted to be a stickler, I would have put in v plus e, thank you.

So where was I? So we got an algorithm now. You know, we've got a bunch of algorithms. Network flow is this amazing hammer that is being used in a wide variety of things. And it's been used for matching, bipartite matching, all sorts of things. My favorite example is something that is related to sports, I should say, which is a baseball elimination. So what is this application about?

You see charts that correspond to how your team is doing. Standings, right? Go to ESPN and click on standings for baseball, for example. It doesn't-- now you see pre season standings but let's assume it's August or something-- and you click on standings. And you see a chart that corresponds to games played, games won, games lost, games remaining, and so on and so forth.

And back in the day when they were just winners corresponding to each division you had to win your division to make the playoffs. So you had to be the best. Now if you ended up being tied, you ended up having to play an elimination game. But the whole purpose of this analysis is deciding whether you still have a chance to make the playoffs or not.

So the goal here is you want an algorithm that is going to look at the standings and decide if your team is alive or not. Is there a chance of God is great, whatever, everything goes your way, angels in the outfield, right? Are you going to make the playoffs? And so you can be very optimistic.

And you might think that this is a straightforward, I'm going to add up a couple numbers and figure this out. It turns out it's not quite that simple though sportswriters would love to think that. That it's as simple as adding up a bunch of numbers. And obviously, there's network flow lurking in here somewhere. Otherwise, I wouldn't be talking about it here.

So this is not quite historical in terms of the standings. I tweaked the numbers a little bit in order to make this example a little more interesting. But roughly speaking, August 30th, 1996, this was the standings corresponding to the American League East. And now this is 1996 so the teams are a little bit different from what you have. So New York, unfortunately, was leading the division even back then. Still had Baltimore. Then there was Boston. And then was

Toronto. And then this is back in 1996, and Detroit was part of the [INAUDIBLE] back then. This was before realignment.

So I just called these one, two, three, four, and five. Those are the positions. All right, so Boston was in third place. And so, I'm going to say these are w i, which are the wins, so far. l i, which are the losses. And r i, which are remaining games to play. And This is usually what you see. I mean this is the simple version of the standings and I'm just going to give you most of these, I should say. And I'm not going to write down what Detroit is, simply because I'm going to vary that number. Because it gets interesting as that number varies.

And each of these teams had 28 games to play. OK so that's usually what you see when you see a snapshot. OK now I'm going to also add rij, which are the games that these teams play against each other. And the reason is simple. This linear correlation between win loss column, specifically corresponding to games that Boston plays with New York. No ties in baseball, so Boston wins, New York loses and vice versa.

So I want those numbers as well and these are things that you typically don't see but they're going to be important, because they do determine whether your team is alive or eliminated. So 5 The reason I have dashes here is because the team doesn't play itself in the diagonal.

AUDIENCE: Is that the number of games?

PROFESSOR: Yes that is absolutely the number of games that they're going to play against each other. All right so I've got small number. I mean 5 plus 7, 12. Plus 4. What is that? 16. 16 plus 3 is 19. There's 28 games left to play, there's obviously games that are being played against other teams outside the division. And so this is just rij, OK?

So that's the story here. And I should say that when I talk about this, what are these corresponding to? Well this corresponds to NY. This column corresponds to Baltimore this corresponds to Boston, same order, this corresponds to Toronto. This column corresponds to Detroit.

So otherwise it's under specified. Any questions about the table here? If you want to know, the Bo Sox did not make the playoffs that year.

So all that most sportswriters do, most people do, is say Team i is eliminated. If wi plus ri is strictly less than wj for sum j. So you're just going to say, OK you know what? If I've won all my games, I'm still not going to make it to this team that's already won 75 games, right?

And so clearly I'm done. That team has already won more games than I could possibly win. So that's an easy one. And so this is strictly less, because we're just talking about elimination, total elimination. You get your summer vacation, or your fall vacation.

So let's just say that you had Detroit, and you're talking about w 5 years. You're talking about this number here, is 46. I5 is 88. So they're having a bad season here. And you're going to say 46 plus 28 equals 74, which is less than 75. Therefore in this scenario, Detroit is eliminated. All makes sense?

So this is sufficient, but I claim it's not necessary. So you might actually have false hope if you are a *Mirror* sportswriter, who only knows to add up these numbers and compare it with others. But if you were a 6046 student, and you went through these wonderful lectures on network flow, you know much more.

And we're not quite there yet in terms of requiring the power of network flow. But I ask, what if w5 were 47? Can you make an argument which clearly doesn't work here, with this naive argument, you have 75 equals 75. So you still have hope. But can you make an argument that if w5 were 47, given the whole table-- that's the hint, the whole table, that Detroit was actually eliminated.

Worth the Frisbee for sure, right? Over there. I saw you first.

AUDIENCE: It looks like the only way that Detroit can get more wins than New York is if they win all their games and New York loses all their games?

PROFESSOR: Yes.

AUDIENCE: and if New York loses all their games, they're going to lose 7 games to Boston And Boston will have 76 wins.

PROFESSOR: Exactly right. So that's a great argument. All right, could you stand up? All right. You catch well too. So that's basically the essence of the issue. There's stuff going on which corresponds to teams playing each other that can affect the outcome. OK so let me write out what the gentleman just said.

Still eliminated, boohoo, because w5 plus r5 equals 75 now. 47 plus 28 is 75. But either New York or Baltimore-- I mean there's many ways you could show this. But we'll just take-- this is not exactly what the gentleman said. But this is what I had here, so let me just write that out. Either New York or Baltimore will win 76 games. Since they play each other 5 times. OK?

So now I'm going to do some more sophisticated analysis and I got what I wanted. But let's say that w5 was 48. So this was 76 and it kind of worked out for both of the examples. The one with Boston and the 7 games, that I got as an answer. And this one that I just wrote down. But what if w5 equals 48? Is Detroit eliminated?

How many people think Detroit is eliminated? All right, a bunch of you. You want to explain why?

AUDIENCE: So New York, Baltimore, and Boston are guaranteed to get a total of at least 14 more wins. Because of their total number of wins that they're allowed to have without any of them getting to 77 is only going to be 13. So it will be 1 plus 5 so that's too many. But they're not actually eliminated, because this is from the Wild Card round anyway.

PROFESSOR: I didn't like the last part of your answer. Did, but the other question is, did people understand the first part of his answer? That was complicated. I mean it was the right analysis. Great job. So, we want a systematic way so we can run an algorithm so we need to bother with this type of analysis.

And you're exactly right. It turns out that I skipped this little detail, because it was inconvenient, which is that in '95, a wild card was introduced. So we could think about this as elimination from being the division winner. So that's an aside. Thanks for pointing out, it's important to know sports history. Really important. Almost as important as 046.

But basically it gets more and more complicated as you get closer and closer to these edge cases. And so you can do an analysis, and I'll actually describe roughly what your analysis was. What's your name?

AUDIENCE: Alexander.

PROFESSOR: Alexander's analysis was. But I don't want to do that yet. I'll do that at the end if you have time. I want to give you a general purpose technique that obviously is going to use max flow to solve this problem. All right? So we're going to set this up. And so that's actually one of the nice things about maximum flow. How you can translate. You can sort of create these networks from tables or what have you, and add capacities to edges, find max flows, and solve these problems.

So this is a good example of that. So let's go ahead and do that I am going to draw a flow network based on this table. It's going to have a source and a sync that are basically these dummy nodes that are essentially going to source about infinite flow. But the capacities of the edges are going to come from that table. And the edges themselves are straightforward in the sense that the graph is going to look the same based on the number of teams, and the particular team that you're analyzing.

So this graph is trying to determine-- this is a flow network to determine if team 5, Detroit, is eliminated. This is just the flow network. I'm going to have a bunch of edges. I'm going to just draw this once and we'll be moving things around on this. 2 to 3, 1 to 4, 2 to 4, and lastly 3 to 4. And the reason you don't see 5 here is because this is an analysis for team 5.

So the other 4 teams show up here. They have edges going to t. s is over here, and it's got a bunch of edges going to all of these nodes. What are these pairs? As you can imagine, these pairs correspond to the games that each of these teams that are inside the circle play against each other. So 3 plays 4 a certain number of times. According to that table it's 4 times.

So I'm going to put a 4 in here. This is 4 as well. 4, 5, 7, 2. OK? And the edges in between here are going to have capacities of infinity and how are these edges structured? So far I've just explained how the left-hand side works. These edges have a capacity of infinity. 1 2 goes to 1, 1 2 goes to 2. 1 3 goes to 1, 1 3 goes to 3. And that's pretty much it.

So that's where these edges are. That's how these edges are introduced. And all of these edges have a capacity of infinity. 2 to 4, 3 to 4, and that. So far, it's pretty straightforward. There's one last thing that we need to do, which is add capacities to these edges. This is actually crucial. It turns out that we have to add capacities such that this max flow is going to represent elimination.

So these capacities have to be chosen. And in particular the equation for this is simply w5 plus r5 minus w1. In this case if it's 48, I'm asking the question for 48 for w5 plus 28 minus 75. So that's 1. And then this thing here would be w5. I'll just write that out. w5 plus r5 minus w2, hopefully you can read that, and that's a 5 and then similarly this is a 7, and that's 13.

And you might ask, what does this represent? Well it kind of represents what you think it represents if you just look at that equation. It says how much can I push through here before I get greater than w1. So what is the capacity? What's the difference here? if I have a 1 here, that means that I still see w5 plus r5, I'm 1 greater than w1.

So in particular, if I want to write this out, the capacities are the number of games team i can win, and not have more wins than team 5. So this is a statement about team 1 when I see that over there. Our team 2 for the other edge. And I'm just trying to figure out how much can I allow, with respect to this other team, to determine elimination for the particular team that I'm looking at, team 5 or not.

So in particular, let's say that team i wins one particular game. So it goes up to it goes up to 76. And if it goes up to 76, it does not necessarily have more wins then team 5, because team 5 can go up to 76 as well. That's pretty much it, and the same thing for team 2, team 3, and team 4.

So the intuition here is the following. So what we're going to do is compute the max flow and look for a certain property on that network. And that's going to tell us if team 5 is eliminated or not for the particular choice of wi of 48, because that's how I constructed this particular example.

So the intuition is, assume team 5 wins all remaining games. That makes perfect sense. This is the best case scenario. You're trying to figure out, in the best case scenario, are you eliminated or not? Is team 5 eliminated or not? And what you want to do is divvy up the remaining games. And that corresponds to sending flow through these edges.

So flow through these edges corresponds to assigning wins to teams 1, 2, 3, and 4. That's the key. Divvy up the remaining games, so all teams have less than or equal to w5 plus r5 wins.

So what does it mean if all teams have less than or equal to w5 plus r5 wins? If you can divvied up such that that's the case, what does that mean for team 5? Team five is still in the game. Team 5 is not eliminated. So if you can do this, then team 5 is not eliminated

If you can't team 5 is eliminated. Because some team is going to get 77 wins. And so that analysis that Alexander carried out wasn't a flow analysis. But somehow, that's got to be in here. That type of analysis has got to be in here to show that some team is going to get to 77 wins. Because you know, well thanks to Alexander, that Detroit is going to be eliminated here.

But we want to show that in terms of the max flow on this network. So there is an associated theorem that I'll write. We've got enough time here. We just have to find the max flow for this.

And that should take just a minute. But I want to set it up so that's pretty much the end. Finding the max flow, what that means.

But the theorem is, which is essentially a more precise restatement of this observation, is that team 5-- obviously you can do this for any team-- is eliminated. If and only if the max flow does not saturate all edges leaving the source. I.e. in this case, the max flow is strictly less than 26, which corresponds to the sum of those things. 5 plus 7 plus, yadda yadda yadda.

And so what does it mean to saturate all of the edges? It means that all the games have been played. All the relevant games here that affect team 5 have been played. So you want to play all the games, because that's the end game, if you will. But basically what that intuition here is that saturation of those edges corresponds to playing all remaining games. And the argument is not a proof if you can't play all the remaining games, without exceeding the capacity of i to t edges, team 5 is eliminated.

So you have to saturate, because you have to play all the games. And when you do that saturation, obviously your flow network has to satisfy its laws corresponding to the capacity that we have in there. And it's quite possible that this is going to cause a restriction over here that corresponds to requiring that not all the games we play in order for Detroit to survive. and that's basically the game.

So saturating the network means playing all the games. And if you get a max flow, to be more concrete, if you can get a max flow here that's less than 26, then you've saturated all the edges and this means that you played all the games. And you found this team that beats you. That beats team 5.

So what happens over here? Let's take a look at this and all you have to do here, is to find the max flow and then we can go back to the statement of the theorem. So I want to find the min cut corresponding to this, which is going to tell me what the max flow is. I could go around for focus in on this, and start with 0 and augment, and so on and so forth.

But there's another way which is not necessarily algorithmic, but eyeballs. Use your eyeballs and find me the min cut here. Which has the minimum capacity. The min cut, the minimum capacity is going to tell me what my max flow is. We know that from the max-flow min-cut theorem. So I want to cut this network into capital S and capital T and find the max flow.

So any ideas? It's a little hard. So I see some people waving their hands. All right, let me just do that since we're running out of time. It's kind of cool. So I'm going to use a different color. We're going to go like that, yeah. and then veer over this way, and then jump up like that, and then come over like that.

And this is S. And on this side is T. So I got small s and capital S, small t and cap T. But I got this 4 in cap T. Notice that I got 3 to 4, 2 to 4, 1 to 4 in cap T and I got these other ones in cap S.

So all I've done here, forget baseball elimination, is find a min-cut. And if you look at what the value of this min-cut is, the value of the min-cut is simply the capacities of the edges that go from S to T. So it's simply the S to T edges and summing over the capacities. And if you take a look, obviously S to T, you've got 4 plus 4 plus 4. So you've got 4 plus 4 plus 4, corresponding to this one, this one, and that one.

Do I need to add this edge in here? Over here? Where does this edge go? From this to over there? That goes from T to S. So that's good because that has a capacity of infinity. That would cause trouble. And so the other edges are I got 1, 5, and 7. So I need this one, that one, and that one. So 4 plus 4 plus 4, plus 1 plus 5 plus 7 equals 25.

And so this implies elimination, because what I've done is found the max flow, which is strictly less than 26. Which means that I have not saturated all of the edges that come out of s. I have not been able to play all the games. And so if I'd played all the games, I'd be exceeding some capacity constraint. And if I push more flow in there, I'd be exceeding some capacity constraint on this side. And that would imply that Detroit is eliminated, because some team would have 77 wins.

All right, so hopefully you've got the gist of it. I'll put this in the notes. Take a long, hard look at it. That particular values aren't important. The framework of translation is important. And you can certainly ask questions of your TAs on Friday. And I'll stick around here for questions.