PROFESSOR: Morning everyone. I hope you all had a great spring break. And during which I hope you can have more of 6.046.

Today we're going to look at network flow and two things in network flow. The first one is Edmonds-Karp algorithm, and after that, we're going to look at two applications. In particular, they are called bipartite matching and cover.

OK. So we'll start with a simpler algorithm, which is Ford-Fulkerson. Can someone remind us what Ford-Fulkerson does? Go ahead.

AUDIENCE: Every time you have a augmenting path, it gets rid of it.

PROFESSOR: It gets rid of it? Yep. What's your name, by the way?

AUDIENCE: Michelle.

PROFESSOR: Michelle. OK. So, Michelle gave a very brief description. But let's be a little bit more detailed. So we have a graph network flow G, and the first thing it's going to do is that given the flow, it will transform G from a residual graph. Everyone remembers that? Then F-F algorithm will find a path going from source to sink in this residual graph, and then augment this path. Augmenting just means increasing the flow on that path. By how much?

AUDIENCE: The minimum edge on the path [INAUDIBLE].

PROFESSOR: Yeah, the minimum edge on the path. Let me just define that to be the capacity of the path, which is the capacity of the weakest link. OK. Then what's the last step we're missing here? So after augmenting, our flow changed from f to f prime. And then we're going to look back. We have f prime, then Gf prime, find a path in Gf prime, then continue from there.

So this algorithm, in some sense, is not even a very detailed algorithm, because it doesn't say how to find a path in the second step. And that's indeed a problem, because we have seen a pathologically bad case where each capacity is a billion, and the correct way to do it is simply

augmenting this path and then that path-- oh sorry, there's edge one-- then we'll be done. But the pathologically bad case is we keep going through this middle edge.

OK. So Edmonds-Karp is an improvement to this algorithm. Let me just write it here. Its first step, third, and fourth steps are actually exactly the same. No change from Ford-Fulkerson. The only thing it does is that it finds a special path. Did Srini cover this in the lecture? What path Edmonds-Karp finds? I see some people nodding. OK. Go ahead.

AUDIENCE: He ran the first search from the source to the sink in Gf, take the shortest path.

PROFESSOR: OK. Does everyone agree with that? So Edmonds-Karp finds the shortest path. From-- I guess, can you see that part? No? Oh, I'm sorry about that.

Find the shortest path. Here, shortest just means least number of hops. Here, in this pathologically bad example, Edmonds-Karp would do much better, because it would find this path, instead of this weird path. Because that path, its distance is 2, and the one going across the middle is 3. OK. Now we're going to prove that this algorithm runs in order VE squared. So OK. So one step back. Do we have to prove that is correct? That it indeed finds the maximum flow?

AUDIENCE: No, you don't.

PROFESSOR: Yeah. I claim I don't have to do that, because it's the same thing as Ford-Fulkerson. You can find any path. Now, we want to show-- we want to bound this run time. So any idea, high-level idea, how we're going to do that? In which several steps? OK. So, OK, a simpler question. What's the complexity of this second step, find the shortest path from source to sink? Go ahead.

AUDIENCE: OV plus E.

PROFESSOR: OV plus E. And in this case, actually, V is always less than E, so I can simply say O of E. Now, how long does it take to augment that path, if we have found it?

AUDIENCE: O of E.

PROFESSOR: O of E. That's correct. But our claim, we can also say it's O of V, because the shortest path at most has three hops. And then, from f to f prime-- OK. So I think in some sense, these two are the same steps. Also O of V, and then updating the rescue graph is also O of V. So one of

these iterations-- each iteration takes how long?

AUDIENCE: O of E.

PROFESSOR: O of E. Right. This is the most expensive step in one iteration. OK. So what else do we have to do to prove this bound?

AUDIENCE: Number of iterations.

PROFESSOR: We need to show that the number of iterations is O of V times E. If we can show that, then we are pretty much done. OK? OK. Now, to prove that bound, we first need a lemma, which we call monotonic. So let me define delta f v. This is the length of the shortest path from s, from the source, to v, which is a node in a graph, the length of that in rescue graph of f. Is that definition clear? And this lemma says delta f v, for any vertex v, does not decrease. So it can only monotonically increase.

OK. We are going to prove that. So we'll prove by contradiction. Assume this is not a case. That means there is some v that in f prime, which is the new residual graph, its shortest path from the source in the new residual graph is less than the old one. Right? We are going to derive a contradiction to that. So there may be many such vertices that drop. But if there's any, I'm going to define v to be the one with smallest delta in f prime. OK. Any questions about this step? So there could be like v1, v2, v3-- that all make this happen. I'm going to choose the one among those that has the smallest distance from the source. OK?

Now here is my source. We have a path to v in graph Gf prime. OK. So this is a path. Then I can always find a predecessor, which I'll call u here. If this is the shortest path to v, what can I say about delta of v and delta of u? What can I say about these two quantities?

AUDIENCE: Can v decrease? Can v decrease one of them, at least?

PROFESSOR: This one is greater than that?

AUDIENCE: One of them must be--

PROFESSOR: One of them is greater than the other, and that's always true, right? For any two?

AUDIENCE: v is u plus 1.

PROFESSOR: OK. Michelle said v is u plus 1. Does that make sense to everyone? That's the shortest path to

v. It goes through u. OK. That's correct. So OK, maybe I wasn't clear about that. I defined u to be v's predecessor on the shortest path. OK? Then definitely delta f prime of v is delta f prime u plus 1. Now, I'll use u to be my step stone back to the original graph, f. I want to say something about this quantity and delta u in graph f. What can I say about that? This is a tricky part.

AUDIENCE: It's greater than v?

PROFESSOR: This one is greater than that? Or the entire thing is greater than that? OK. I'm going to claim this one is greater than delta f u plus 1. By which I'm claiming this quantity is larger than that. Can anyone give a reason why I can claim that? So recall how I defined v.

AUDIENCE: Can you repeat how you defined v?

PROFESSOR: Good question. So v-- so there are several, probably several v's that have delta f prime less than delta f. I'm going to define v to be the one with smallest delta f. Among all those nodes that have a job in delta f, in delta. This is probably a tricky part. Then by definition, since delta f u is less than delta f v, right, and I defined v to be the one with the smallest delta that satisfies that. So all the u's-- so u is a predecessor of v, so u shouldn't be one of those nodes that have a drop in delta. So I know this is probably a tricky part. Yeah, I'll stop for questions and make sure we resolve this part before we move on. How many people get it? OK. Only two. That's not good. OK.

AUDIENCE: I'm confused about how v can be the one with the smallest delta f if you have a predecessor with a smaller delta f.

PROFESSOR: OK. So v to be the smallest-- the one with smallest f such that delta f prime is less than delta f.

AUDIENCE: OK.

PROFESSOR: OK, maybe, yeah, that's why I confused you guys. Yeah. Sorry about that. So we have a bunch of nodes who have a drop in delta, and I defined v to be the one among them that has the smallest delta f prime. Question?

AUDIENCE: Sorry, I'm lost at what delta f prime is versus delta.

PROFESSOR: OK. Delta f of a node is the shortest path from source to that node in G of f, which is the residual graph given a flow. So f is, well, some flow, and f prime is the flow after we

augmenting a certain path. So f prime is one step after f. OK. How many people get that now? Still not everyone. OK. Any questions about that? How many people still haven't got that? OK, so some people-- it's like Schrodinger's cat. It's in the middle state. Well, then I'll have to move on, and I'll assume you all get that one.

As the last step, so of course we will ask, what's the relation between this guy and delta f of v? Because in the end, we want to show a contradiction that delta f v is probably greater or equal to delta f of v. OK. This is another tricky part.

So let me ask maybe a simpler question now. So that is our G of f prime. So we also have G of f. It has the source, some u here, v there, sink there. Does this edge exist in this graph? So I know if that edge exists, because I defined u to be the predecessor of v. But u, v is in that graph doesn't necessarily mean u, v is in the old graph. It could certainly exist, but is it possible that this edge didn't exist? How many people would say that edge definitely exists? How many people would say maybe it doesn't exist? OK. It doesn't matter, because we can prove both cases.

So let's say case one, u, v is indeed in the original graph, G of f. OK. In that case, can I say something about that last step?

AUDIENCE: It's greater or equal to delta f v.

PROFESSOR: Greater or equal to delta v. OK. Why is that?

AUDIENCE: Because v was on the shortest path. Well-- yeah. So the shortest path, either passed through-- yeah. The shortest path to v either passed through u or didn't.

PROFESSOR: Correct. Does everyone get that? So the shortest path in this graph to v is not necessarily this one, right? But if it's some other one-- OK, so, in case one I'm assuming this edge exists, right? Then no matter what the shortest path is, it's definitely shorter than I first go to-- from s to u and then u to v. Right? The shortest path between s to v is definitely shorter than I first go from s to u and then u to v. Make sense? OK. So case one, then, is a contradiction, because we showed that delta f prime of v is greater or equal to delta f of v. Any questions about that?

OK. Case two, u, v is not in Gf. OK, so, how can that even happen? Is it possible that this edge didn't exist?

AUDIENCE: It could be a backwards-facing edge that you add [INAUDIBLE].

PROFESSOR: Great point. That's exactly right. So it is possible that this edge didn't exist, but only appeared after we go from f to f prime. How can that happen? That must mean we are augmenting a path that goes right through it. Right? But this edge doesn't exist, so we cannot be augmenting a path that crosses this way. We must be augmenting a path that goes like that. First from s to v, and then v to u, and then u to t. If we are augmenting such a path, then we're going to remove this edge from v to u, but we will add u to v to the next graph. OK? Did everyone get that?

So now, here-- well, formally I'm going to claim if we assume u, v is not in G of f, and we also know u, v is in Gf prime. What does that mean? These two can only be caused by the fact that u, v is in G of f. Not only in that graph-- it must be on the path we are augmenting. Make sense? OK. So given that, can we say something about delta f of u and delta f of v? So here is our p. Right? This entire thing here. And p is the augmenting path.

AUDIENCE: And f u is just f v plus 1.

PROFESSOR: Correct. Right. So if that is the augmenting path in Edmonds-Karp, we are looking for the shortest path. Right? So v is the predecessor of u, then delta of u is delta of v plus 1. That means that quantity, here, is delta f of v plus 2. OK? So it's also a contradiction to what we assumed. OK. So that proves our lemma of every node's delta monotonically increases.

Now here, I'm going to show that-- our final theorem, that we have at most VE number of iterations. So the way we're going to show that is we are going to define a capacity of the path. It must be the capacity of its weakest link. We're going to define its weakest link is u, v. OK? And we're going to show that u, v-- we'll call that critical edge-- we're going to show u, v can be critical only O of V times. If that holds for every edge, then I claim all the edges combined can only be augmented O of VE times.

Now, how do we show that? So again, let's assume we are augmenting a path that goes through u to v. If we do that, by our algorithm, we will get rid of this path, and have an edge backwards. We go from f to some other f prime. Now, when can I augment u, v again? It can only happen if at some point, I come back to augment the path going from v to u, because that will eliminate this back edge, and and adds back our edge u, v. OK. Let's see what happens there.

Now I'll call this graph f, this graph f prime. So in f prime, we know we are augmenting the path

from v to u. Right. The same argument, delta u, is delta v plus 1 in f prime. Correct? And we know this one doesn't drop, right? So it's greater or equal than delta f of v plus 1. OK? And we also know in f, v is the predecessor of u. So this one is equal to delta f of u plus 2. What does that mean? It means in between two times u, v is augmented, delta of u must increase by at least 2.

Then how large can delta of a certain node possibly be? It's definitely bounded by v. So then I claim this particular edge, u, v, can only be involved, can only be critical edge, O of V times. Actually, half of V times. Strictly. So then, every edge combined, there can only be half of V times E number of augmentations. OK. Any questions about that part? About the entire proof? If not, we'll move on.

OK. This is not part of the required in the recitation, but I'll quickly say a few words about an even better algorithm, Dinic, which was in O of V squared times E. So this is an improvement to Edmonds-Karp, and its idea is that I'm going to find all the shortest paths in one go from s to t. I'm going to augment all of them at the same time. Then, because they are all shortest paths, when I augment that, each path will be broken. Maybe I'll get rid of this edge, this edge, and that edge. And the thesis is that the shortest path will increase.

So here, the shortest path is 3, but I have destroyed all the shortest path of length 3, so I'm going to shortest path of 4. If they do that, you can bound the number of iterations to O of V, because your shortest path can be, at most, V. But each iteration is slightly more complicated, because you need to find all the shortest paths, and it happens that they can show-- you can find it in V times E. That gives the V square of E.

That's not the important part. The more interesting part is that, actually, the author of this algorithm, his name is Dinitz. But his algorithm is very hard to understand. Yeah, nobody got that. And there's some other guy whose name I think is Even. He understood the problem and started advertising to people and giving lectures on Dinitz's algorithm. So he popularized the algorithm, but unfortunately he got the name of the author wrong. So that's why this algorithm is henceforth called Dinic's algorithm. This is useful to know. Why? Because you can tell this story to other people so they will assume that you know a lot about Dinic's algorithm, while in fact, you probably don't. And that's exactly what I'm doing here. OK.

Now let's look at one application. Bipartite matching. OK. So the problem is that we have several person and several tasks. And let me get rid of one person, because I don't want to

draw that many stuff. So we have a graph like that. And each edge-- if there is an edge, that means this person is capable of performing that task. And the problem is that-- find the matching, which means the assignment from people to tasks, such that we get as many tasks done as possible. OK? So one person can only do one task, and one task only needs one person. So here you can see a bad matching, which is I assign this first guy to task one, then no one can do task two. So if I'm smarter, I'll assign one of these guys to that task and have this person handle the other task. Is the problem clear?

So this is called bipartite graph, because-- well, which means you can partition a graph into two parts, and within each part, there's no edge connecting any pair of vertices. And you can also define this problem for a general graph. The goal is the same. Find a subset of edges such that no two edges are connected to the same vertex. But we are going to look at bipartite graph. And we claim in bipartite graph, this can be solved using max flow. OK. I'll give you one minute to think about that, how to transform that problem to a max flow problem. Any ideas?

OK. A hint. I will add the source here, and the sink there, and I'll have created these edges. OK. How do I restrict the capacity of all the edges such that I can guarantee no person is taking two tasks, and no task is assigned to two people?

AUDIENCE: Make all of them weight 1.

PROFESSOR: Make all of them weight 1. OK. You're right. We do that-- 1, 1, 1, 1, 1, and everything here is weight 1, then that's definitely correct. What's your name?

AUDIENCE: Calvin.

PROFESSOR: Calvin? OK. If we do that, then, well, because each person only has one incoming edge-- oh, sorry, maybe I should have drawn the arrows here. Each person only has one incoming edge, so it cannot take care of two tasks at the same time. Right? Same thing for each task. It only has one outgoing edge, so it cannot be taken care of by multiple people.

OK. So if we find-- if the max flow is k, that means we can find the max matching is also k. We can get k tasks done. It's very easy to see because we can have a cut here, and if the max flow is k, that clearly k tasks are taken care of. OK. Any questions about that?

AUDIENCE: So, the question is can we do all the assignment [INAUDIBLE]?

PROFESSOR: Find an assignment from people to tasks to get as many tasks done as possible. So how many

tasks can we handle? OK. So I'm doing a different topic from the recitation notes. In the recitation notes, we are considering another problem called bipartite cover, which is exactly the same thing as that.

So cover, also, bipartite graph, is defined too as, let's find several vertices in this graph such that each node, at least-- OK, let me color several nodes in this graph-- such that each edge is connected to at least one dark node. OK. And this, the nodes I colored, is called a cover. I want to find a minimum cover. You can of course cover all the nodes, that trivially holds, but we are looking for the minimum cover. And the claim is that min cover is k if and only if max flow-- sorry, max matching-- is k.

Why is that? Because in that matching-- if we have a matching, then we have a set of disjointed edges, right? No two edges are connected to the same node. So if we want to cover these edges, we at least have to add one of these guys-- color one of these two, and one of these two. Right? So if we have k matching edges, then we at least need k nodes to cover them. For example, I can cover them like that. But this is not foolproof, because I also need to show I can indeed find a cover that is k. OK. Let me think about whether I should prove that. OK. I'm going to give it a try. If you don't get that, that's fine.

AUDIENCE: Can you say again what cover is?

PROFESSOR: OK. Cover--

AUDIENCE: [INAUDIBLE]?

PROFESSOR: Cover is a set of nodes such that every edge in the graph is connected to at least one of the nodes, in a cover. OK? So this is not a cover, because this edge is not covered. So in this case I probably have to add that as well. But clearly there's a better cover, even a smaller cover, which is those two. Right? Yeah.

So how are we going to transform that matching to cover? So let me first give a matching here. One max matching should be something like this. I have this edge and one of those two. That is a matching. Right? That is a max matching.

No questions about that? Let me get rid of all this. One way to transform this into a cover is that I will first color these two guys. I colored the nodes on the left where they are connected to a matching edge. After that, I'm going to start from this one and jump between nodes, taking an alternating path. Meaning I'll take an unmatched edge, and then take a matched edge, and

then take an unmatched edge again, but there's no such thing.

OK. And if I take this jump, I will swap them, make that dark. OK. This graph is a little strange. Ah, OK. Now I claim this is a cover. Is that? Yeah, that is one, right? And if there is another unmatched edge going out, then I'll keep taking that alternating path. But I have to stop, because there's no such edge anymore.

How can I say-- why I can prove-- why can I claim this is a cover? So clearly if this is a cover, then it's a cover of size k, right? So we have proved the entire thing. Then we're going to consider several things separately.

So I'm going to first claim if I have a matched edge, then it's definitely covered, because one of its end points is dark. Right. We only do a switching between dark and white if it's a matched edge itself. Right. So I only need to show that this thing doesn't happen. There's an edge that's unmatched, and two of its endpoints are both white. OK. I'm going to first claim this node cannot have a matched edge. Because if it does, then this is an alternating path, and I'm going to switch this too. OK. Make sense? OK. So this doesn't exist. So it can only have an unmatched edge.

Now what can I say about this node? I claim this node needs at least one matched edge, because if it doesn't, right, then I should add this guy into my matching. It doesn't violate any constraint. I didn't add that because there's probably a matched edge connecting to this guy. Then I'm going to-- so, then this guy has to be dark, because this one is not. Then how did this one become dark? It must have come from some other guy on the left, right? So there's an alternating path going back and forth. Something like this. There is an alternating path, starting from left, but ends here. Because we showed that this guy is not connecting to any matched edges.

In that case, what I'm going to do is match this edge, match this edge, match this edge, then unmatch these two edges. And that's a larger matching. Because I removed the two, I added three. That means a max matching of k leads to a minimum cover of k as well, which also means max flow of k in our network flow.

So this part-- this equivalence is not required. You should know this proof, the proof of Edmonds-Karp, and know that a matching and cover can be solved by network flow, and that Dinitz's name is spelled not as Dinic. OK. And that's everything for today.