**SRINIVAS DEVADAS:** Good morning, everyone. So we have a singleton lecture today on linear programming, which is general purpose optimization technique that you can use to solve a whole bunch of problems, including ones that we've seen in 6.046 and previously in 006.

And most recently, we looked at max flow. We wouldn't have had to go through all of that pain we went through to derive a max flow algorithm if you had a linear programming package handy and all you wanted to do was find the optimum solution. You could have just run the linear program with an appropriate input, of course, that is derived from the flow network. And you'd get your optimal solution. And we'll spend a couple of minutes on that as we look at the power of linear programming in today's lecture.

But it's not just max flow. You could do shortest paths. You could do multi commodity max flow, which is more complicated than max flow and a variety of other problems. So that's that good news.

The bad news is that the algorithms for linear programming are a heck of a lot more complicated than max flow. And you can imagine that that would be the case, because it's a more general purpose and more powerful technique. The history really is that it was an open problem. Up until 1979, people did not know if linear programming was polynomial-time solvable until Khachiyan came up with this ellipsoid method, and then there's been progress sense.

But the algorithm we're going to describe today and execute on a example input is a simplex algorithm-- the simplex algorithm-- that runs in worst case exponential time. But it's a very efficient in practice. And it's held its ground, even with the advent of more efficient, from a theoretical sense, polynomial-time algorithms, namely the ellipsoid method, which actually is not that efficient in practice and new interior point methods.

So a little bit of context, let's just dive into an example of optimization in the context of politics and see how you could formulate this particular problem as a linear program. So how does

politics work? You buy elections, right? So you don't want to spend a lot of money, so you want to minimize the amount of money that is required to buy an election.

And the way you buy an election is, well, campaign, but you advertise. And it doesn't matter facts that are relevant. As long as you get to the right demographic with the right message, let's assume that you're going to win the election. So that's our mathematical abstraction pf campaigning and politics all in 30 seconds.

So how to campaign to win an election. And as I said, we're going to advertise. But you do have a little bit of work to do here. That's why you need your campaign manager. And this manager is going to estimate votes obtained per dollar spent.

But that dollar is spent advertising in support of a particular case, or a particular issue. And contradictions are allowed. As long as you're sending different messages to different demographics, you're all good, right? You're assuming that people don't watch more than one channel. So you're a *Fox News* guy or you're a *MSNB* guy. You don't do both.

So now, we get at this estimate. And it turns into a table. And so you have your policy, and you've got your demographic. So you got urban, now think Detroit, suburban, I guess, you could Lexington where you live, and rural-- I really don't have any idea what that means. But presumably, there's places.

And here's our policy. You want to build roads-- kind of boring, but some people are interested in roads-- gun control, of very sensitive, farm subsidies-- you know who's interested in that-- and gasoline tax-- kind of more, this hits your pocket, so more or less everybody's interested in that.

So you tell the urban guys you want to build roads, and they don't like you. So you get a minus 2 there. So this can go, you advertise, and it hurts you. You lose votes.

Tell the suburban people-- well, typically, it's a situation where you have these nice cars, and you don't like potholes, so you like roads if you live in suburbia. Rural people have 4-by-4's. They don't particularly care. They don't care as much.

Gun control-- well, you can imagine that urban people like that. Suburban people are, hm, OK, meh. And the rural people hate it, right? You do not want to advertise on gun control in the rural areas.

Farm subsidies-- so like, don't want to deal with that. What is a farm? And the rural people love it, right? And then gasoline tax, well, urban people are commuting. And, well, they typically don't have a lot of money. So there you go.

And those are the numbers. I'm not going to justify every number here. But you could put whatever numbers you what, I mean. So let's move on. This is just a table.

It could have positive numbers. It could have negative numbers. And you still want to win this election. Regardless of how crazy the demographics are, how crazy your electorate is, you want to win the election. So as long as you have a great campaign manager, who can get you this table, it's all mathematical here on out.

You've just got to figure out how you're going to win a majority. And you could argue that all you want is to win the election. We're going to do something slightly different, which is something that's obviously going to guarantee victory. But you want to win a majority in every demographic. As the tables may be off by little, you want to be careful.

So the last thing, of course, in order to estimate how much money you need is the population here. So that's votes obtained per dollar spent. So you're getting-- it's $10 a vote, it's $5 a vote, et cetera. And so we need to translate that to votes, because that's the dollars. And you got your population here corresponding to each of these areas. And that's what you got here.

Majority, we'll assume you win in the case of ties, just to keep these numbers easy, so that's just divided by 2. So that's what you got so far. And you want to win by spending the minimum amount of money. So that's our optimization problem.

So we can take this, and we can convert it to a set of linear equations. And that's going to create our linear program-- our first linear program. And this is our algebraic setup. And so let-- we need some variables here, so let x1, x2, x3, x4 denote the dollars spent per issue.

So you got those four issues up there. So let me write that out. It's important to make sure you know what I'm talking about with respect to a particular issue. So those are our four issues. And those are our four variables. So this linear program has four variables. You're trying to discover the values of these variables to optimize, minimize your cost function.

The second thing that a linear program has-- and pretty much the only other thing a linear program has-- are constraints. And these constraints are also linear. It gets much more complicated if you have quadratic constraints, and we won't go there. These constraints that

I'm going to write correspond to this statement here that says you want to majority in each demographic.

So you can imagine that because you have three demographics, you're going to have three constraints. You could have written this differently. There's just any number of variants here. And you'll get the sense of that as we go to other examples. But we'll just stick to one variant here.

So now, I want to translate everything that I've written in English over there into algebra. And so I got my minimization criterion-- minimize x1 plus x2 plus x3 plus x4-- subject to minus 2x1 plus 8x2 plus 0x3 plus 10x4 greater than or equal to 50,000. And this represents the requirement that I want a majority in the first demographic, namely the urban demographic. And so I want at least 50,000 votes there.

And I need to spend the money corresponding to the values of x1 through x4 in such a way that I get those 50,000 votes. And that represents that, and it's just reading off the minus 2, 8, 0, and 10 from the urban column. So those numbers that you see here correspond to the column, because I'm talking about the urban demographic. And you can imagine that the next constraint is going to correspond to the middle column, and the third to the third.

So I'll just write that out. I will call this constraint the constraint number 1-- I might refer to it later-- 5x1, plus 2x2 plus 0x3, plus 0x4 greater than or equal to 100,000. Call this number two. And then finally, 3x1 minus 5x2 plus 10x3.

And that's our set of constraints, but there's one more little issue that we have to be careful about, if you're being precise, and that is that there's no notion of un-advertising. And so you're going to spend positive dollars. And so x1 through x4 is greater than or equal to 0.

So that's our first linear program. And it came from this particular problem. It'd be wonderful-- and that's exactly what we're going to do for the rest of the lecture-- if we could solve this linear program and any possible linear program in an efficient way.

And so the number of variables is small n. And you can imagine that the number of constraints here, just talking about these constraints, are m constraints. And you certainly want at run time that is polynomial in n. That's our goal here.

And as I mentioned early on, it was unclear for the longest time-- well, at least not until 1979,

but people had been thinking about it for a long time before that-- as to whether there was a general algorithm that would solve any linear program in time-polynomial in n. And that was resolved in '79 by Khachiyan. We'll look at a algorithm as simplex that in the worst case runs exponentially in n but is simpler to describe and is very efficient in practice.

So in our particular problem, this one, it turns out you-- and I'm actually going to come back to this in a second-- but I will just tell you that the optimum for this particular linear program with these particular numbers correspond to these numbers here. So you want to spend something of the order of $20,000 for-- so there's 100 here, so take away the two 0's-- so spend something of the order of $20,000 for the first issue, building roads, spend a bit of money for the second issue, ignore the third corresponding to farm subsidies, and spend a bit of money for the gasoline tax issue.

And these numbers aren't important, other than the fact that they happen to be optimum. So if you add up these numbers, then x1 plus x2 plus x3 plus x4 is something of the order of $21,000-- $27,000, excuse me, though I'm writing it out as this fraction. So important consideration here is that these values xi are real numbers.

That's it. It's not that they have to be integral. Clearly, there were fractions here for the optimum, some of them anyway. But in general, linear programming says the variable values are real.

There's also integer linear programming, which is NP complete, which adds the additional constraint that the xi values are integral. So it turns into a harder problem. You got polynomial-time solvable if the xi are real. You got NP complete, which Eric is going to talk about on Thursday, if the values are forced to be integral. So this extra constraint makes things worse from a complexity standpoint. We won't talk about ILP anymore for the rest of this lecture.

So I will come back to this. And I'll talk about how we can show that this is optimum without actually going into a deep algorithmic dive. But what I want to do just before that is to give you the general formulation of a linear program. It's called the standard form in CLRS, also called the general form.

In some cases, we'll look at the standard form for LP. And I want to pop up a level about this example and give you the general setting. And we'll focus in on the general setting for the most part.

But what I have here is I can either minimize or maximize-- we had a minimization problem-- for the political problem, minimize the linear objective function subject to linear inequalities or equations. And the variables, think of x as a vector, it's a column vector, or x1, x2, all the way to xn. And the objective function is c times x, so that's c1x1 dot, dot, dot, cnxn. And we just had all the coefficients being a 1 over there.

And the inequalities, they're the fun part, you can represent them as a matrix A, so A times x less than or equal to b. And notice that this is the standard form that I'm talking about. And now, I have diverged from what I had here, because I had greater than or equal to over here.

So it turns out, you'll see linear programs in different settings. Sometimes, you'll have minimization. Sometimes, you'll have maximization. Sometimes, you'll have less than constraints, less than or equal to constraints. Sometimes, you'll have greater than or equal to constraints. Sometimes, you'll have equality constraints.

We'll spend a little bit of time talking about how you can transform any given linear program into a standard form. So our standard form is going to be something that maximizes the objective function. So these are our inequalities, and they're represented as less than or equal to. That's the standard form.

And you want to maximize c times x-- again, max for standard-- such that these set of inequalities told Ax less than or equal to b and x greater than or equal to 0. So for each of the values that correspond to the variables, you want these variables to be non-negative in the standard form. And you want less than or equal to corresponding to each of the inequalities-- not equal to, not greater than or equal to, but less than or equal to. And you have this linear cost function, where you could have arbitrary coefficients, but you're maximizing it.

So that's it. So it's all about polarities, not much more than that. It's just about polarities. And if you get a linear program, a specific linear program that doesn't conform to this-- we'll spend a few minutes talking about conversions-- and it's going to be fairly straightforward. May not be immediately obvious, but we'll get to that. Any questions so far?

So I want to go back to this claim here, where I said this is optimum. Now without actually describing an algorithm to you, I'm going to be able to show you, convince you, that this value here corresponding to whatever it is, 3 million, 3.1 million, is in fact optimum. And this is something I could do, because linear programming has this powerful notion of duality.

So what is that? Well, let's just first look at our specific example here. And I'll give you a very specific observation. I'm going to give you what you can think of as a certificate of optimality. I'm going to give you a certificate of optimality for that set of numbers.

And here's how I'm going to do it. So is there a short certificate? I can imagine giving you a long proof that a particular linear programming algorithm always gives you the minimum answer, the optimum answer, walk through that, executive that algorithm on this particular example, and then you're convinced, of course, that the solution is going to be optimum.

But for this specific example, I want to give you a certificate. This certificate isn't going to work for other examples. It's going to short, because it only works with this example. But it won't work for other ones.

And so how do I do that? So the answer is, in fact, there is a certificate that shows that the LP solution is optimal. And consider that I compute this particular algebraic quantity, where all I've done is I've taken these three equations and I've multiplied them by these magical constants.

And so I'm not going to tell you how we get this certificate of optimality. But I'm going to give you the certificate. And it's going to be clear that it's a certificate of optimality. And if I take these three equations here, 1, 2, and 3-- actually, I refer to 1, 2, and 3, they refer to the equations. These are equations or constraints.

And so I take that. And obviously, if I have a bunch of equations and I multiply them out, I can certainly add them up, and I get another equation at the end of it. And it's all going to be linear.

And if I do that, I get x1 plus x2 plus 140 divided by 222 times x3 plus x4. So that's what happens to the left hand side. And the right hand side is-- you'll recognize this quantity-- five 0's divided by 111. That's what you get.

So now, can someone tell me why in the last step, why this is a certificate of optimality-- the fact that obviously, this is all algebra, once I've discovered the coefficients-- so now that I've done this, why have I just shown you that 3.1 million divided by 111 is, in fact, optimum? Can someone tell me this? Look at what you have on the left hand side. No? Yeah.

**AUDIENCE:** Any other solution would cost more than the amount put in spent.

**SRINIVAS DEVADAS:** Any other solution, but I want you to relate that to-- what am I spending?

**AUDIENCE:** You're spending 3,100,000-- like, it's the same thing.

**SRINIVAS DEVADAS:** Yeah, but I mean, this was a claim. This was a claim-- and at this point, an unproven claim. It was an unproven claim. Yeah, go ahead.

**AUDIENCE:** You know that the left hand side of that inequality is less than or equal to x1 plus x2 plus x3 plus x4.

**SRINIVAS DEVADAS:** Correct. And what is x1 plus x2 plus x3 plus x4, to be clear?

**AUDIENCE:** The thing you're trying to minimize?

**SRINIVAS DEVADAS:** Yeah, exactly, the thing you're trying to minimize. Exactly. You're almost there. but the key observation here is that x1 plus x2 plus x3 plus x4 is greater than or equal to x1 plus x2-- because all of these are positive quantities, remember-- 140 divided by 222, that's less than 1, x3 plus x4, correct? So given that, I can say that this is greater than our equal to 3,100,000 111. It's because of this observation that it's a certificate of optimality.

She has her head down, OK. Great. So that's pretty cool. Just cooked up these coefficients from somewhere, pulled them out of a hat, you're all convinced now that the value we got was optimum. Did not run an algorithm. Maybe I ran an algorithm-- of course, you ran an algorithm to get those coefficients, right? Well, how did those coefficients appear?

So we're not going to spend a whole lot of time on this. You'll see this likely in a problem set or perhaps in section. But in general, I won't worry too much about duality, other than knowing the concept. And this notion of LP duality essentially says that what just happened wasn't a coincidence. You can do this all the time.

There will always be, for a linear program, a short certificate of optimality that corresponds to some set of coefficients that you can do this particular math with by taking these linear constraints, multiplying them out, adding them up together, and showing that you have a lower bound on-- in the case of this problem-- you can't get lower than this. And therefore, for a minimization problem, when you reach that, you clearly found the optimum.

And that's the notion of LP duality. And the basic theorem-- and this is really more as an FYI, we won't prove this theorem-- is that if you had the standard form for the LP, which I'm just writing down again here, where you had Ax less than b, x vector greater than or equal to 0. So

that's identically what I had up here, or done here, corresponding to the standard LP form.

Well, there's a dual-- this is what's called the primal form. Usually, if you don't say it, you think of it as the primal form. And if it's dual, you call it a dual.

And this is primal form of LP. This is a dual form of LP, or dual LP. And the dual LP flips everything. And it's not just negation, but transposed, and the actual variables also swap functionalities. So it's really pretty cool.

So your max turns into a min. The c gets replaced by the b, which is on the right hand side of the inequalities. And your constraints are A transpose, y greater than or equal to c. So there's a flip there as well. And y is greater than or equal to 0.

So there's a bunch of things that's going on here. And these two problems end up being equivalent-- the primal and the dual, you can always do this. And essentially, what is happening here is that you're solving these two problems simultaneously. And there's lots of algorithms that keep flipping between these two forms for efficiency.

But ultimately, what ends up happening is you see that the actual constraints that you had here corresponding to the b constraints turn into-- the b ends up in the cost function here. And that's essentially what's happening out here with respect to multiplying these equations out with particular coefficients. So as I said, this is really more as an FYI.

This is an obviously interesting and an interesting proof of optimality, which is a different kind of proof from proving an algorithm correct and applying that proof to a particular instance. That's the kind of thing that happens in LP, especially when you flip from primal and dual forms. So I'll leave it at that.

What I'd like to do is give you a sense of how we can convert to standard form, so you can apply an algorithm that-- for example, you have a program and it only requires standard form. It runs on standard form. Let's go over it really quickly. This is not going to take very long-- a translation from different kinds of LPs-- and we had a slightly different here for our political problem that had a minimization-- and how would we convert that to standard form.

So it's probably just one conversion here that's tricky. So suppose I want to minimize minus 2x1 plus 3x2, and I want to convert it to standard formal. All I have is a standard LP solver. What do I do?

It should be easy. What do I do if I had a solver that was maximizing, but I want to minimize a quantity? Just switch the signs, right? So negate to 2x1 minus 3x2 and maximize. So that was easy.

This is a tricky one. Suppose xj does not have a non-negativity constraint. So it just happens to be the case that it's not dollars but it's some other quantity that can go negative. It might be profit or loss. So that quantity represents profit and loss, so it could go negative if it's loss.

So I don't have this constraint in my original problem specification. But my standard form and my LP solver requires this entire vector to be non-negative. So I got a problem here. I can't use my standard solver, because of this non-negativity constraint. So how do I fix that? How do I turn it into a problem that allows the standard solver to be used? Yeah, go ahead.

**AUDIENCE:** You can break it up into two variables, like xj1 and xj2, so xj1 minus xj2 equals xj, and both could have [INAUDIBLE].

**SRINIVAS DEVADAS:** Perfect, great, that's good. Here you go.

So what you do here is take xj and replace it with, let's say, xj prime minus xj double prime. And you have xj prime greater than or equal to 0, xj double prime greater than or equal to 0. But depending on the particular values in whatever solution you're exploring are the final solution, you may have an actual xj value that's negative or positive. So you added an extra variable here to your linear program.

And a couple more real quick, suppose that I have an equality constraint corresponding to x1 plus x2 equals 7. What do I do with an equality constraint where I have x1 plus x2 equals 7? Yeah, go ahead.

**AUDIENCE:** You can say x1 plus x2 is greater than or equal to 7, and x1 plus x2 is less than or equal to 7.

**SRINIVAS DEVADAS:** No, you can't do less than or equal to.

**AUDIENCE:** But then you can flip the signs to--

**SRINIVAS DEVADAS:** Ah, then you could flip the signs. So you have two steps there, good. So your less than or equal to needs another multiplier. So what you end up doing is something like x1 plus x2

greater than or equal to 7. And then you need-- if you do what the gentleman just said-- and flip the sign, you get minus x1 minus x2 greater than or equal to minus 7.

Is that right? No? I messed up? Oh, I want less than or equal to. You're right, you're right. So I need less than or equal to-- that's right, of course, thank you. So I need less than or equal to in both places.

So that's the standard form. I needed less than or equal to. Good. What you've done is increased the number of constraints by one. Did I get this right the second time? All right.

So that's pretty much it. The last thing, which I won't really write out, is, which we've done here already, greater than or equal to constraint translated-- I won't give you an example of this; we have this already-- translates to less than or equal to by minus 1 multiplied. So we have to invoke that in order to do the equality anyway.

So you're in business. If you have a standard LP solver, you can take pretty much any optimization problem that is linear in terms of its objective function and has linear constraints, and you can transform it into LP. If you had non-linear constraints, there's lots of work that goes on in linearizing those constraints and using LP solvers.

It's a very practical thing to do. It may be something you'll end up doing, invoking these powerful LP solvers-- typically, they're commercially available; the best ones are commercial-- and use it to solve your particular problem. It turns your algorithm design problem into a reduction. And so you'll spend really the next couple of weeks thinking about reductions.

We'll start that up right now, where we'll take existing combinatorial problems, for which we already know algorithms for, and you're going to reduce them to LP. Just to give you a sense of what the power of LP is, but this notion of reduction is very powerful, you can use it to do complexity proofs. Here, we're just using it as a convenience in today's lecture to use our LP hammer.

So let's say that I have our favorite problem of the week, namely max flow. And I want to convert that to LP. So go back a week ago, and right about this time a week ago, and we'd set up the max flow problem.

And let's assume that we went back there. And we didn't talk about augmenting paths, and we didn't talk about residual capacities or min-cut or anything like that, but we knew LP already. And we just want to solve max flow using LP. So let's do that.

So this is maximum flow. I'm not going to bother with converting to standard form. We know how to do that, given what we just did here, over there. So I'll just do whatever I want to keep things simple.

Max flow is obviously a maximization problem. And using the same notation we've used, it's not going to look like Ax and b, just because I want you to recall what max flow is. And we're going to translate that. And the values of these variables-- or the names of these variables, whether they're x or f, it should really matter. We know how to do LP at this point-- we know how to formulate LP, I should say, at this point. And we're assuming that we have an LP solver.

So what I want to do here with the maximum flow problem is maximizing the flow value. And it's simply, you grab the source, you have a variable associated with the flow from the source to every other vertex of v. And you have to maximize that. So that was the setup for max flow. I'm not changing that.

What do you think the three constraints, or whatever set of constraints that we have here, are going to correspond to the LP? You spent a week on max flow, looked at the problem set, what constraints am I going to have to put up there? I'm going to have to put up capacity constraints. That's an obvious one.

What is another one? Conservation constraints. All flow entering a node that is not the source or the sink has to leave it. In the original network, is there a concept of negative flow? No, you will define it going in the other direction. So we did talk about negative numbers, et cetera, but you're going to have positive quantities, especially if you look at net flow, the version of the flow that we zoomed in on in the Tuesday lecture from last week.

And you also have-- in the general setting, you're going to have these skew symmetry constraints as well. So the three things that you need here are skew symmetry, conservation, and capacity. So you have such that f u, v equals minus f v, u for all u, v belonging to V.

And depending on the kind of network that you have, if you constrain it to a sudden type that you don't have these two-way edges, you could certainly remove some, if not all, of these skew symmetry constraints. Important ones are conservation and capacity. And this should seem pretty familiar to you.

But the key-- the reason I'm writing these all out is primarily to ensure that you understand that these are all linear constraints. So that's pretty much the only thing that you need to observe here. Obviously, these constraints you've seen many a time from the two lectures last week. But notice that they're all linear. And finally, this one is f u, v less than or equal to c of u, v for all u, v belonging to cap V.

So this is f. That's a variable that's less than another constant, clearly linear. Doing a bunch of sums here. I could obviously have multipliers, scalar multipliers. In this case, for conservation, I don't have scalar multipliers, but clearly linear. Skew symmetry, got two variables in here. One of them is a negation of the other, clearly linear, so that's why this is an LP.

And so you might say, well, I know better, max flow is much more efficient than any LP solver that's out there. And you would be right. If you have a max flow problem of this variety, it's difficult to imagine that you would get performance, empirical performance from running an LP solver.

But this generalization of max flow that's a multi-commodity max flow, where you just don't have one commodity flowing through. You may be counting cars and trucks on a road, or there's two different kinds of liquid flowing through the same pipe, whatever, gas or liquid. And so when you have multiple commodities, you may have a linear but more complicated cost function that's a function of the flow of each of the commodities. And they may have a certain weight associated with them. So there's a lot of things that could be more general-- there could be more general settings corresponding to max flow.

And I'll just leave you with the thought that you could simply have two commodities. And we'll just call them 1 and 2. And so now, you have the f1's and the c1's and the f2's and the c2's. Each commodity has to be conserved.

But what about the capacity? What do you think happens with the capacity? Let's just assume these are two different kinds of cars. So what would the capacity constraint look like? Yeah.

**AUDIENCE:** You can say either c1 or f1 plus f2 is-- for each edge, you can add them together or you might take the linear [INAUDIBLE].

**SRINIVAS DEVADAS:** Exactly, that's right. So good point. It may be the case that I have distinct capacities. And in fact, if you have completely disjoint problems, you're right in that you can solve them separately.

But actually, the more interesting case would be that you have a single capacity c, so you'll have-- let me just write this out here. If in fact you had two distinct things, so if you had f1, c1, f1, c2, the question is, do you have two distinct, disjoint optimizations, in which case you just use max flow twice.

On other hand, what's more interesting really-- and I should've used this example for starters-- but here's a better one. You have two commodities and a single capacity. So the road is a good example. Both the cars and the trucks share the same road. It has a certain capacity.

And now, your capacity constraint is looking like f1 plus f2 over here. And that's the flow through the particular edge uv. So you have something like f1 u, v plus f2 u, v is less than or equal to c u, v for this total capacity. And that's pretty much it.

So that is linear. The nice thing is that it's linear. You could put weights on it. If you wanted to claim that a particular commodity 1 uses up-- because it's a truck, it uses up more space on the road. And you can accommodate fewer of them. You could put a multiplier in there. Still say it's linear.

So that's the power of by having an LP engine. You could translate problems that are not exactly max flow, that are multi-commodity flow. You may have additional linear constraints that you could add, and you could still use your LP package. So that's the reason why this is interesting and powerful.

So that is kind of an obvious, corresponding to max flow. Let's look at something that's a little less obvious. And it's going to be a little tricky to convert the shortest path problem to LP, not a lot of work but one little observation that's going to be important to make in order for the whole thing to flow through or actually work out.

So we all know the shortest path problem. We want to find-- let's just call it the single source shortest path problem. You have a specific source. That's going to turn into the point from which you're going to start computing the distance. That's what Dijkstra does, and that's Bellman-Ford does.

And so this is from vertex x-- s, excuse me, s. And what I want to do here is obviously set it up as a set of linear constraints. If I have dv corresponding to the distance from the source-- so dv represents the distance from the source-- and eventually I want dv to be the shortest distance from the source.

That's our notation for shortest paths. dv represents an existing path-- it may not be the shortest path-- from s to v, the value of that. But dv monotonically decreases as you run through. It's initially infinity in Dijkstra going back to Dijkstra. And then we shrink it through a process of relaxation.

Now I want to try and model that-- I want to try and model all of this as an LP. So it's not immediately obvious-- the thing that the flow networks had, where we had these constraints. We have capacity constraints and conservation constraints. And we could turn that constraint into an inequality. And it was pretty smooth. It's pretty easy.

So what I need to do here with shortest paths is something that's a little more subtle. So what basic constraint do I have in a shortest path algorithm? What's an inequality-- you remember an inequality from shortest paths that we kept talking about? The triangle inequality.

So we're going to have to go with the triangle inequality and take the triangle inequality and use that to create an LP formulation of shortest paths. In particular, what we have here is that I could write dv minus du is less than or equal to w u, v for all u, v belonging to E. And that's the triangle inequality. And I'm going to have d of s equals 0. That's the only thing that I start with.

And so what's happening out here is simply that there's different ways of getting to v. And my shortest path is going to be the best way of getting to v. So in particular, the way you want to think about this is that if I have a v and I can get to it from, let's just say, u1 and u2. And maybe the source is over here. And these are the only two edges that can get to v.

So I'm just looking at a fairly limited setting. u1 and u2 are going to have to be the two vertices. One of these two is going to get me to v. And I got w u1, v here. And I got w u2, v over here.

And so what this is saying is, I'm going to have to write this out for each of these edges. For each of these edges, I'm going to have this constraint. And that's says that the dv value, if I want the shortest path, should obey both of these constraints. And if I want to obey both of these constraints, one of them is going to be my limiting constraint. And I'm going to get the min of those two. Correct?

So in effect what this translates to is that it's an AND, right? So dv have minus du1 is less than or equal to w u1, v. dv minus du2 is less than or equal to w u2, v. That's an AND, because I'm putting both of those constraints in here.

And that essentially means that dv is going to be the min of the two quantities-- the du1 quantity plus the w u1, v, and the du2 quantity plus the w u2, v. That make sense? Ask me questions if this is unclear.

So that simply corresponds to the fact that I'm doing an AND over here. I'm adding all of these constraints in there. So I'm applying the triangle inequality to every edge, to every relationship between a vertex that has a path ending at it. And you're pushing it forward to this vertex v, all the different ways that you can get to v. In this case, there is two sets of ways-- one from u1 and one from u2. And the last step is a minimization step.

So you think you're done or we're done, but we're not quite done, because what's missing here in terms of my formulation of LP? What else do I have to do here? Well, sure, non-negative-- let's do that. Sorry?

**AUDIENCE:** Objective.

**SRINIVAS DEVADAS:** Objective, who said objective? You again? So we are missing an objective function. Now shortest path is what kind of problem again? Short means minimum? Minimum, height whatever.

So do I put a-- what happens if I put a minimum in here? And let's say that I do something like sigma over V, cap V, dv, because I want to minimize-- or I could pick a particular one. I could pick a particular single source, single destination, and I put a minimum there.

What happens? Does this work? What's the solution to this problem, if I minimize the distance? 0, because the zero value is going to work. So there's something-- I haven't put in the constraint that I do want a path. I do want a path from s to v for any v that matches one of the quantities in the min, because the min says I have equality.

The big issue here is, this is a less than or equal to, and that's why the min doesn't work in the objective function. It's a less than or equal to. But this min over here, which is the definition of a shortest path, is saying that it's either equal to this quantity or equal to that quantity. There's an equality over here that is missing from this side. And that's the key observation.

Once you observe that, that need equality for one of these constituent quantities of the min, then you'll see that what you have to do is simply change this min to a max. So you say, well, how the heck did a min get changed into a max? And I'm not sure I'm going to convince every on of you in the next minute or so.

But the bottom line is, it comes down to I do have a min already in my inequalities, because I'm ANDing each of these inequalities, and I'm putting down each of those inequalities in there. So each of them is going to force me to find the best solution, because they're going to constrain me to not go via u2 if u1 is better, because the other constraint corresponding to u1 is going to force me down. So there's an additional min in there, because the ANDing of the less than or equal tos.

And then in order to actually force the equality for one of those, I need to push up as hard as I can, or as high as I can. So think about it. Play around with a couple of examples. Choose a simple example for starters. And you'll see that this is the correct formulation.

So you can see that it's not completely clear in some cases how to transform problems to LP. But even in those cases, sometimes you can. So there's just a ton of different problems, a good skill to have to be able to take combinatorial optimization engines, like LP or even max flow, and be able to translate problems to them. It's something that you'll probably do if you stick to algorithms in your careers or exploiting available algorithm packages.

So the last thing I want to do here for the rest of the time is to give you some sense for how an LP program is actually optimized. How can you possibly take the standard LP formulation, which is a general setting. You know nothing about shortest paths, let's assume, nothing about max flow. It's not about a specific problem. It's about the general setting.

How can we solve the general setting, because that was the theme here anyway. You had this engine, and you want to use this engine. But now, how do you build this engine?

So what we're going to do is look at a fairly simple example of the simplex algorithm. And this algorithm is in the textbook. And it'll be in my notes. So I'll get as far as I can. It's not that complicated to describe, especially from an example standpoint. But I may not get through all of the steps to get you the optimum for this particular example given how much time we have.

The most important concept in simplex is yet another form of representation for simplex, which says that you can represent the LP, not in standard form, but in slack form. So I'm going to tell you what slack form is. And then what we're going to do is, the flow of simplex, algorithmic flow, is to convert one slack form into and equivalent.

Obviously, you don't want to do something that's incorrect, but it has to be an equivalent slack

form, whose objective value has not decreased and has likely increased. So you're guaranteed that the objective value has not decreased. You're not guaranteed that it's increased.

And then we're going to keep going till the optimal solution becomes obvious. And you might say, how is this obvious? That's the reason why I talked about the short certificate of optimality. It's definitely a relationship between the termination of simplex and the fact that you can now say, hey, I know I'm done here, it's kind of obvious that I can't do any better. And hopefully, you'll see that by the end of this lecture in this simple example.

So that's it. It's an iterative algorithm. It's exponential, unfortunately, because this takes m plus n, choose n iterations in the worst case, where n is the number of variables and m is the number of constraints. Most of the time, it does a lot better, but that's the only bound that you can actually prove in the worst case. And so you're stuck with an exponential algorithm if you're using simplex worst case.

We won't actually do much analysis on simplex. It's really out of scope for 046 in terms of the analysis. The actual algorithm is certainly within scope.

So what I want to do is give you some sense for what the slack form looks like. And we'll do a couple of iterations of simplex. And we'll get as far as we can before the end the lecture.

So we'll take a different example from our political example. It's similar in size. And I want to explain to you what the slack form is and why it's interesting.

So what I want to do is maximize 3x1 plus x2 plus x3 subject to the constraints that x1 plus x2 plus 3x2 is less than or equal to 30, 2x1 plus 2x2 plus 5x3 is less than or equal to 24, 4x1 plus x2 plus 2x3 less than or equal to 36, and then non-negativity constraints, x1, x2, x3 greater than or equal to 0. So that's our example problem. I'll leave it up there.

You're going to convert this to slack form. And so what is the slack form? We're going to introduce an additional number of variables that correspond to the number of equations that we have. So we're going to introduce, in this case, three new labels, because I have three equations.

And the slack for this problem looks like this. I'm going to have z equals 3x1 plus x2 plus x3, same as before. And then I'm going to have variables that represent-- these are called basic variables. And the original variables are called non-basic variables.

So I'm going to add three basic labels, x4, x5, and x6, corresponding to these three constraints. And they're going to represent slack in the sense that they're going to correspond to how much slack you have in the inequalities that you have in the original problem. So if x4 happens to be 0, then you're jammed up.

You have no slack, because x1 plus x2 plus 3x3 equals 30. And increasing any one of them will violate the constraint. So that's just simply the notion of slack. It's how much room do you have.

x5 is 24 minus 2x1 minus 2x2 minus 5x3. And the last one is 36 minus 4x1. This is very mechanical up to this point. And so I'll call this set of equations, equation I.

And what I'm going to do is I'm going to now work on a space that corresponds to x1, x2, x3, x4, x5, x6. So I'm going to have these solutions that now have six values associated with them, as opposed to just three values, because I've added three variables, the basic variables, to my non-basic set. So the original variables are non-basic, just to differentiate.

So so far it's just set up its definitions. We can think about up running through iterations of simplex. It takes about three iterations here in order to get to the point where the optimum is obvious. So you're going to convert through three slack forms. And then finally, when you get to this fourth slack form, you see that you have an optimal solution.

And how does that work? You're going to have the notion of a basic solution, where we set all non-basic variables to 0. So in this case, what we're going to have-- and then we're going to compute values of the basic variables.

So our objective function here is going to be 3 times 0 plus 1 times 0 plus 2 times 0, which is obviously 0. And the values x4, of course, is going to be-- because all of these are 0-- is going to be 30. x5 is going to be 24. And x6 is going to be 36.

So this is kind of a trivial starting point. So you can think of this as 0, 0, 0, the solution that you're looking at, 30 24, and 36. So that's our starting point, which doesn't really tell you much.

But now comes the key step, where we're going to do something that's called pivoting. And in pivoting, you're going to swap a basic able with a non-basic variable. It is a step that requires some intelligence. But the basic step is a swap.

So one of the basic variables is going to become a non-basic variable and vice versa. And how do we select this? Initially, you can kind of do this in an arbitrary way. It gets a little more subtle as you go along. You don't want to do things in a random way.

But let's just start with what pivoting actually does in a more generic setting. The basic step is select a non-basic variable-- let's call it x sub e-- who's coefficient in the objective function is positive. You can always redefine things if you can't find something like that. But we won't go there.

And then what we're going to do is we're going to increase the value of x as much as possible. And we always have constraints, of course. So we can do that without violating any of the constraints.

And then at this point, variable xe becomes basic. So it's going to turn into the left hand side. It's going to move over. You're going to swap-- the x1 might be over here, over here, and you got to rewrite these equations so the x1 becomes basic, for example, over to the left hand side, and the variable that it replaced goes over to the right hand side.

So you can think of this as Gaussian elimination, except with inequalities. There's definitely relationships there, if you recall your Gaussian elimination. If you don't, don't worry about it.

So xe becomes basic, and some other variable becomes non-basic. The values of other basic variables and the objective function may change. So we'll do one pivot step at least, so you get a sense for the algebra involved. And it becomes a little more concrete. To motivate you further, you'll be doing this in problem set 8 on a different example.

So what did I do here is we're going to select a non-basic variable. So let's just select x1, lexicographic order or numeric order, let's select x1 is selected. That's the non-basic variable. And what I want to do is increase the value of x1. So I want to increase it without violating constraints.

Now, which of these constraints do you think is going to cause trouble first with respect to increasing the value of x1? x1 is now 0. We're at ground level, we have all 0, things are feasible.

Now, as I start increasing x1, remember, you have non-negativity constraints associated with each of the x4 values as well. That's what these basic variables represent. So don't forget the fact that the constraints are violated. You need all the xi's to be greater than or equal to 0.

That was true for the non-basic variables. It's also true for the basic variables. So a violation of a constraint implies that one of the currently basic variables goes negative. That's exactly equivalent to the original inequality not being satisfied.

So which of these constraints, do you think, is going to cause trouble here? Just look at it, and should be able to look at the three equations and tell me, as I increase the value of x1, where am I going to hit my limit? x6, yes, absolutely correct.

So that's because of the minus 4 here. This is a big multiplier. I got a minus 1 here, a minus 2 here, and a minus 4 here. And if we just look at 4, the magnitudes, 4 is bigger than 2 and is bigger than 1, so it's going to be that third constraint.

So third constraint-- you can obviously compute this numerically or mechanically. It was just easier to do this in this example by eyeballing it. And so third constraint is the tightest one. And it limits how much we can increase x1. So I'm going to do my second step up there, which corresponds to rewriting x1 as these other variables.

And now, I got x1 on the left hand side and x6 on the right hand side. And now, it's just merely a matter of substitution. Once I've done this, I'm just going to jam through and go in and I'm going to rewrite the other equations with x6 on the right hand side. And that is, I'm going to replace x1 with the eyeball equation. And that's really a simple substitution.

So at this point, what's happened is, because the third constraint representing x6 was the one that was chosen, what's happened is that x1 and x6 are going to interchange roles. x1 was non-basic, it's going to become basic. x6 was basic and is going to become non-basic. And that's basically the essence of the simplex algorithm.

The iteration and then the convergence and all of that is, as I mentioned, going to require getting to a point where the optimality is obvious, but we won't be doing any proofs corresponding to conversions for simplex or any other specific LP techniques in this class. Maybe some constraint techniques, I take back what I said, but certainly not for simplex.

But I just want to give you a sense for the flow here. And so let's just go through this last thing here in terms of finishing off the pivot xi's. And I want to show you what the equations look like. And if you just keep doing that, at some point, you'll converge.

And so what happens here is you have z equals 27 plus x2 divided by 4 plus x3 divided by 2

minus 3x6 divided by 4. x1 equals-- so there's a bunch of algebra here that I'm obviously skipping over, but it's simple algebra. And I have x4 over here, 21 minus 3x2 divided by 4 minus 5x3 divided by 2 plus x6 divided by 4, and then one last thing here.

So that's my pivot step, that I've flipped x1 and x6. So now you ask, what was the point? What was the point of this? Well, the point of this was that you actually increased the objective function value and, in this particular case, quite significantly, while maintaining correctness.

And so let me just make these observations and conclude here, because then that gets us to the point where you've seen the details of one pivot step. And you can imagine applying it over and over to a convergence. And let's just look at the original basic solution, which as you recall was 0, 0, 0, 30, 24, and 36. And this is simply x1 through x6.

This original basic solution suddenly satisfies these equations-- equations II, if you just put them in there. And it makes sense that it will have the objective value of 0, given equivalents, but you can verify that by saying that you have 27-- the original had the objective value of 0, because all the x sub 1, x2, x3 were 0, so that was an easy check in the set of equations corresponding to the first set, which I've erased at this point. But no matter.

And if you look at what I have here, I have 36 equals 0. So the objective value here is 0. It matches what you had before. But the basic solution for II, I'm going to set the non-basic values.

So what is a solution here? The non-basic values are 0. So the solution is going to be 9, because 9 is non-basic. x1 is now non-basic. x2 and x3 are still basic. And then I have 21 and 6. And x6 now has become basic.

So the way I get this solution is simply by plugging in 0's on this aside. And I get 9, 21, and 6, because I just plugged in 0's on the right hand side. So that's how I got a new solution.

And if you look at the objective value for this, the objective value, you can look at this objective value simply by looking at the original problem. And the original problem had 3x1 plus x2 plus x3 as the objective value. And so if you go off and you see, well, that you had 0's for the other ones, but you have 3 times 9, so we have an objective value of 27.

So this flip of our pivot basically got you from an objective value of 0, while maintaining correctness, to an objective value of 27. And you can look at this in the notes or in CLRS, but

you have to do two more pivots corresponding to two other variables-- the same grungy stuff that I went through here, substitution after selection. And the objective value is going to increase.

And you might ask, how do I know that I'm done? And so that was the last thing here, which is increase the value until it becomes obvious-- no, this was pivoting, I'm sorry-- increase the value pivot until it becomes obvious what the optimum is. And what ends up happening is you end up with an objective function.

In this case, the objective function mind you, this thing over here is the objective function. And notice that it has a negative coefficient on the variable that was actually the part of the first pivot. So x1 and x6 were a part of the first pivot. And this got a negative coefficient.

So what ends up happening here is you end up getting something like 30 minus something minus something minus something, where you have xi values over here. And when you set these to be 0, that's the best you can do, because these are all negative quantities. So I'll just leave you with that. Hopefully, you understood how we do the pivoting-- sub through it three times, and then you get that objective function, and the optimum value is 30. See you next time.