PROFESSOR: Ok then, let's start with problems. And as we find concept issues, we'll get to concept issues. So, we talked about shortest-path, but we talked about shortest-path in a very odd way, right? I'm a coder. So for me I'm used to, all right I go to one of these lectures, I hear a problem, then I get out of the lecture with an algorithm and the running time, right? This time we got out of the lecture with no algorithm and no running time. So, what's the point?

The point is that we learned some analysis tricks that we can use for any shortest-path algorithm. And the advantage of knowing that-- by the way too easy to forget, at least until you see the first real algorithms-- if you know that, if you have to modify an algorithm, you can still use the same analysis to prove that your new algorithm is going to be correct and that it's going to be fast. They're really nice tools to have, especially if you have to do that on a problem set or on an interview, or who knows, maybe even on a quiz. We'll see if that happens or not.

We didn't talk about any shortest-path algorithms. So, we're not going to assume shortest-path algorithms. Instead we're going to use them as if we had them.

No, wait. I don't like that. That's kind of annoying. Let's not do that. Let's do something else. We didn't talk about the shortest-path algorithm in lecture. Let's build one right now, how about that?

Let's turn to the graph. Vertices and edges, right? Let's see if I can make this work.

Suppose we have a graph. We have V vertices, E edges and all the edges have a weight that's non-negative. Sorry, let's make it positive just to make things easy. It's smaller than W, so it's not too big. And they're all integers. Go.

There's brute-force, OK. So, brute-force would mean what? Enumerate all the paths? OK. Let's go over something a little bit better.

Does anyone remember the algorithm that was taught in class? The structure? No. Klaus mentioned Dijkstra and mentioned Bellman-Ford. Let's write them up here. And they gave us the running times, which we'll try to remember in a bit. But, we don't know algorithms, so no Dijkstra, no Bellman-Ford for us. What do we know?

Let's start with a simpler case. What if there are no weights here. What if the graph looked like this. You're making my life hard. Like this. No costs. All the edges have the same costs. What would we do to solve the problem?

Say we want the shortest path from A to E.

AUDIENCE: BFS.

PROFESSOR: BFS. How's everyone feeling about BFS? What does BFS do, really quickly? How do you run BFS?

AUDIENCE: Take a starting node and then you can search all it's neighbors.

PROFESSOR: OK. Pick a starting node. Then, look at its neighbors?

AUDIENCE: And then you enter it through each of its neighbors to find its neighbors and so forth.

PROFESSOR: Sounds an awful lot like BFS. What's the difference between them?

AUDIENCE: Well, you go through all of these neighbors before you start going through any of it. I mean, you go through all As neighbors before you take a neighbor and go through all of its neighbors.

PROFESSOR: Ok, that's the difference. And it makes a huge difference as we've seen in topological sorting last time.

So we have a list and we're going to use it to keep track of the nodes that we need to visit. We start with A. We numerate all of As neighbors. And the ones that haven't

been seen get stuck at the end of the list. A, B, C. A is out of the list.

Then we take out the first node from the list, B in this case. We numerate all its neighbors, A, C, D, E. Out of all these, we take the neighbors that we haven't seen and we put them at the end of the list.

take out C, we look at its neighbors. We've seen them. We take out D, we look at his neighbors. We've seen them. We take out E, we look at all its neighbors. We've seen them. So this is BFS.

BFS has this concept of levels. And another way to look at levels is, if you start at A, and you draw a circus of radius one, it's going to have all the nodes that are distance 1 from 8. Right? Exactly one edge.

Then you draw a circus of radius 2 and you get all the nodes that you can reach in two edges. And BFS calls these circles levels. A is at level 0. This is level 1. This is level 2. If you look at this list and you write down the levels, you have 0, 1, 1, 2, 2.

If you had another node out here, say F, this would be discovered when we get to D. It would go all the way here at the end, and this is level 3. In BFS the levels are always increasing.

If you keep track of parent pointers, the parent pointer of a node, at some level, will always point to a node at the previous level. So, in BFS we get the shortest paths in terms of edge count. Make sense? This is a recall, right? It is not new stuff. Everyone happy with it?

If it wouldn't have costs, we could use BFS to compute shortest paths. But, we have costs. How do we deal with that?

Intuitively, I'd like to make BFS go across this edge faster than it would go across the edge. If I could do that, if this guy could get stuck in the queue before this guy, I would be happy. Intuitively, that's what I'd like to do. How do we do that?

AUDIENCE: Sort before adding?

PROFESSOR: Sort--

AUDIENCE: Sort your neighbors sorts.

PROFESSOR: Sort? OK.

AUDIENCE: What if you kept track of it in a menu? Kept track of your queue as a priority queue.

PROFESSOR: Congratulations. You deserve a Turing Award. You have discovered Dijkstra. OK, let's go for something simpler. I just say this is Dijkstra. We'll talk about it in lecture on Tuesday. Very good.

AUDIENCE: Like, have an adjacency list and, I think using what he said. Just sort it by cost--

PROFESSOR: OK, let's see how this would work. We have A, and we're pulling A out of the list. And we see its neighbors are C and B. Right? The distance to C is 2. The distance to B is 4. So we start with A. Does this mean that I'm going to put in B before C? I'm going to put in C and then B?

AUDIENCE: No, it doesn't really matter because at the end you just sort the part. I guess you could add in the costs. Like associate the costs of B with 4 and C with 2 within your adjacency list?

PROFESSOR: So, that's how we would keep track of costs, right? One way.

AUDIENCE: --for the queue would be A and then the value would be like a [INAUDIBLE] B and its cost, and the [INAUDIBLE] C and its cost.

PROFESSOR: So you're saying that what I would have in this list is B with a cost of 4? And C with a cost of 2? Presumably, the other way around, right? C goes first?

Like This? And then I take out C, I explore the neighbors so and so forth? The moment I did this I already lost the battle. Because, see this path A, C, B, length 3? If I put both of them in the queue, then the parent pointers are going to look like this. So, I already have a bad path from B to A.

AUDIENCE: OK, well don't add that node then.

PROFESSOR: OK, then how do we do it?

AUDIENCE: I guess you can talk to each of them and see which one's lower and then, only add that one if it's the lowest. So it's more like depth search then--

PROFESSOR: Well, you two need to talk together because you're getting to Dijkstra, too. You're both going to rediscover Dijkstra.

So that's great, but I'm looking for something simple. This is too complicated. If you're going to do Dijkstra now, then what are going to do on lecture on Tuesday?

AUDIENCE: More Dijkstra problems?

PROFESSOR: Well, I think Shreeny wants to talk about Dijkstra, so let's not get there.

AUDIENCE: How do you do this without doing Dijkstra?

PROFESSOR: Good. That's exactly what I want to know.

[LAUGHTER]

So, let me give you another hint. The first hint was that I want to go through this edge faster than I would go through this edge. The other hint is that we're allowed to change the graph.

AUDIENCE: Is this what we talked about in lecture, or are we thinking of something else?

PROFESSOR: He might have talked about that at the end of lecture.

AUDIENCE: Were you putting the current weight for each node inside the circles?

PROFESSOR: No.

AUDIENCE: You could add it and then if you could find a better path to B? With rows relative to A, then you could just delete that path? And B to 4? So at the beginning you add them all, but then--

PROFESSOR: You're risking to have exponential running time.

AUDIENCE: (LAUGHS)

PROFESSOR: Those are all valid optimizations under some constraints. Like, this one's good if you don't have too much memory, but the time goes up.

I want my graph to have edges with no costs, because if I have costs, this is not going to work. BFS is not going to work. We can't tweak it. If you could tweak it then that would be another Turing Award.

AUDIENCE: So, create dummy nodes?

PROFESSOR: Create dummy nodes. How do I do that?

AUDIENCE: Put a band of node in between A and C?

PROFESSOR: OK. Why am I doing this? So now I have two edges and they have costs.

AUDIENCE: One. Then you put three nodes in between A and B.

PROFESSOR: OK. And now here I have four edges, right? 1, 1, 1, 1. There was a 4 here before and there was a 2 here before. Do you guys see what's happening?

So, instead of one edge of cost C, I have C edges. So, now BFS is going to take C steps to go through that edge. And all the edges now have the same cost. This looks like a problem that BFS can solve. We know it can solve. We proved that it can solve it. Well, CLRS did. We take their word for granted, sort of. This works right? Is everyone happy with it? Does it make sense?

AUDIENCE: So basically, we get through C before you'll get to B on the other path.

PROFESSOR: Yep. This is the path of length 3, so this node is going to get in the queue first. Then this node. Then these two nodes. And they're going to get in the queue from C.

Let's see what happened here. This is really important because this is what you're going to be doing in real life. Chances of having to build a new algorithm? Kind of slim. Why don't you want to do that? Say you take Dijkstra and you tweak it a little

bit. If you did that, you have to go through the analysis and prove the running time again, prove the correctness again. Time-consuming, error-prone, it's really hard. However, what happened here is that we have some problem that we want to solve.

Say we have the raw input for that problem that we're trying to solve. That is this graph that I draw the board. And we apply some transform and we get an input that's suitable for an algorithm that we already know. I'm going to take a shortcut here. We apply the transform, right? We took this graph and we built a new graph in. And this is hopefully easy. Right? Take an edge and split it up. Put in fake nodes. This is easy. We know how to code that.

And now we have this algorithm that you already know, and we're treating it like a black box. We know it's correct. We know it's running time. We're not questioning them. We're taking them straight from the textbook. They have to be right. By the way, what's the running time of BFS?

AUDIENCE: V plus E?

PROFESSOR: Very good.

Then BFS is going to give us a path, right? And the path is going to look like this. It's going to be A, fake node, C, D, fake node, E. If you're giving this back to the guy who gave you the problem, they're going to be like, what are these fake nodes? I don't know anything about them. This doesn't make sense.

You need to take this raw output and you need to transform it again. Based on what you did this transformation, you have to read out the output. You have to interpret it. So, this is either interpret or readout. And then you get a nice output that's the output to your original problem.

This process here is the most likely way in which you'll use your 6006 knowledge. And this is replaced by any algorithm that we taught. This is replaced by any real life problem that you'll encounter.

There's one missing step here. We know the running time for this. Let's find out the

running time for the whole thing. In order to do that, we have to figure out given this original graph. Let's say that now, in the original graph, we have V prime vertices, E prime edges, and W prime weights. We have to convert this into V plus E. We have to see when we make this transform, how many vertices do we get? How many edges do we get in terms of the original graph so that we can compute the running time?

In the original graph, V prime vertices, E prime edges, W prime weights. In this new graph, how many edges do I have, at most?

AUDIENCE: Something times E?

PROFESSOR: Yep. Perfect. Each edge has cost at most W. That means we're going to split it up into most W edges.

How many vertices are we going to have?

AUDIENCE: W prime?

PROFESSOR: W prime, E prime.

AUDIENCE: Plus V prime?

PROFESSOR: Plus V prime. Don't forget about the original ones. They're still there. So, total running time for BFS on this new graph is?

AUDIENCE: --E prime.

PROFESSOR: Put it the other way around, plus W prime, E prime. This is the running time that we have. Now, does anyone remember Dijkstra's running time and Bellam-Ford's running time? So what is the running time for Dijkstra?

AUDIENCE: V log V plus E?

PROFESSOR: Almost.

AUDIENCE: V log V?

PROFESSOR: Oh, wait. I think you're right. V log V plus--

AUDIENCE: E?

PROFESSOR: I already named this from CRS? Or is this what we got? Entry is a fancy key to give this running time.

AUDIENCE: [INAUDIBLE].

PROFESSOR: You can get this, but you need the really fancy data structure for it. In real life, the running time looks more like E log V. And how about Bellam-Ford?

So let's take this running time for Dijkstra because this is what you're going to implement in real life. And we can argue about that after next lecture for now. Take my word for it.

And let's compare it to what we got here. V is smaller than E in most same cases. Let's say that this is actually W prime, E prime. If you compare this with this, you'll see that, if W is smaller than log V, well, smaller equal, then this algorithm is not worse than Dijkstra. It's on par with Dijkstra.

For graphs with small costs, we discovered an algorithm that solves shortest paths. You walked into this recitation knowing no algorithm to solve shortest path. Now we have an algorithm. Already in a much better position. Right?

And we didn't invent anything new. We don't have to prove correctness. All we did was one of these transformations. How's everyone feeling? Those things makes sense? Everyone happy?

Let's talk about another problem that also uses this structure. Suppose we have the same graph that we had before, or actually, any graph with V vertices, E edges. Now we know how to compute shortest paths. So we can assume these as black boxes. We're going to use these as they are.

We have a graph with V vertices and E edges. Let me copy the costs really quickly.

Suppose this is the highway system, right, like in Google Maps. And we have two brothers. They start off from a place and they have to end up in another place. They're going to drive together. They want this to be sort of fair. Every time they go between two cities they're going to switch seats so that none of them drives too much.

So say the brothers are Tim and Jim. The names are wrong. This is actually a problem from last year's quiz. So, by the way, real problem. You can pay attention now.

These are two brothers. They're going to alternate. Right? One of them is going to drive across one road, then the other one, then the first one, then the second one. So on and so forth.

Now, two Tim has a much better sense of direction than Jim. That's just how things are. And if you ever driven long distances, the hardest thing to do is to get from the city to the highway. Like, if you have to drive from here to New York. Hardest thing to do? Get out of Boston and onto the highway. Then, at the end, the hardest thing to do? Get from the highway into where you want to go through New York traffic. Everything else? Piece of cake.

We want Tim to handle both of these situations. The driving path must start with Tim and must end with Tim. Otherwise, God knows, they're going to crash.

First off let's convert this into graph terms. And then let's solve it.

AUDIENCE: You might convert what's already in the graph.

PROFESSOR: I have this constraint that they're going to switch seats and that Tim has to start and Tim has to end. How do I phrase this in mathy terms? In graph terms?

AUDIENCE: We're starting it where and ending where?

PROFESSOR: We're given where in the graph. We're starting at S, ending at T. Some points in the graph.

AUDIENCE: And people have to drive with it an entire edge?

PROFESSOR: Ya.

AUDIENCE: So basically, Jim has to-- whoever the good driver has to-- to drive on the edge that's adjacent to E and S.

PROFESSOR: Yep. OK.

AUDIENCE: You physically need, like an odd number of edges.

PROFESSOR: Yep. So I want the shortest path with an odd number of edges. Why odd number of edges? If we look at the edges, Tim's going to take the first one. Then Jim, then Tim, then Jim, then Tim, then Jim. No matter how many I have here, it has to end with Tim. So, I'm going to have an odd number of letters here, therefore, odd number of edges. Right? So I want the shortest path that has an odd number of edges. And as the hint we're going to use that trick over there. Yes?

AUDIENCE: But, the shortest path isn't necessarily like the fairest, right? Like, in terms of distributing driving?

PROFESSOR: One of them is going to drive one more road, such is life. Sorry, one more edge segment. Tim's going to drive three times, Jim is going to drive two times.

AUDIENCE: Right, but like, in terms of the weights?

PROFESSOR: Oh, yeah. We don't care about that. Jim doesn't know where he's going anyways.

AUDIENCE: Just a minimum, we have to have like three paths, right? We don't just want Tim to get out to the highway and be like, oh look our journey's not that far--

PROFESSOR: So I went to the path with the smallest cost. So, smallest total weight.

AUDIENCE: In theory it could be length, too. Right?

PROFESSOR: Well, if it's length, too, then it's not going to be good because they're going to crash.

AUDIENCE: Oh, I see. They have to switch every time.

PROFESSOR: Tim can't drive for two consecutive edges.

AUDIENCE: You could use breadth-first search and, once you get to the end point, or made all paths that are [INAUDIBLE] and then, of those paths, go through to see which one's the lowest cost?

PROFESSOR: You're thinking enumerate all paths of even length? Just to make sure I got it. So this is cool because it let's me show something else that's cool. So, enumerate all paths of even length. Something else I heard, enumerate all the shortest paths. For both of these, let's look at this graph. S, T, all the paths have length 1. How many ways are there to get from S to T?

AUDIENCE: Two to three.

PROFESSOR: Two to three. Two ways to go across this diamond. Two ways to go across this one. Two ways to go across this one. Right? 2 times 2 times 2? 8. If I add another diamond, how many paths? One more?

So the number of even paths, or the number of shortest paths, the number of whatever paths I can think about is, order of? Come on, guys. Math. Rrrr! Go!

[LAUGHTER]

AUDIENCE: 2 to a number of diamonds?

PROFESSOR: Which is? Roughly to the vertices. Exponential in the number of vertices. Not very good, not very good. We can't do this. So, it's better that we talked about it here, then that you try to do it on the quiz and you have to discover this on your own, right? Good comment.

AUDIENCE: Didn't understand what you meant by that lecture. Now it makes sense.

PROFESSOR: That's completely different.

AUDIENCE: That wasn't the thing he was talking about?

PROFESSOR: That's different.

AUDIENCE: OK.

PROFESSOR: That's where relaxation can go wrong. So if you show that if you relax the edges in a random order, you can have an exponential number of steps. This shows that there are an exponential number of paths for any graph, no matter how good your algorithm is.

So we're not going to be able to enumerate them. Let's try something else.

AUDIENCE: You have another suggestion? Just look at all the--

PROFESSOR: So, you're doing this. You're looking here. Everyone's looking here. How about this?

AUDIENCE: We should do that?

PROFESSOR: Ther's that, right? There's a transform. You need to transform your graph in some way, then run the algorithm. And then reading off the outputs should be easy. I claim that the hard step is the transform step, not the read off step.

Let me give you one more piece of intuition. We're going to keep track of states, somehow. Remember Rubik's cube? We had one vertex for each state. A state representing the configuration of the cube.

Here, suppose I'm looking at the path from S to D. The problem that I have with this is that, if I have an algorithm that tells me the shortest path from S to D is something, I don't know if it's even or if it's odd. It's not keeping track of that. It's missing some state. The state that it's missing is whether the path that I used to get here is even or odd length. I should do something to the graph to keep track of this state.

AUDIENCE: Can you BFS it once? To plan-- You BFS once. Then you that B and C are all like 108, and then you know that E and D are 208?

PROFESSOR: B 208 if I take this path. It might be 108, might be 208. But, this is good. This is what

you'll do on a quiz, right? You come up with an idea and then you're going to think about it for a bit. If you get stuck, try to convince yourself that it's wrong. If things get too hard, discard and look somewhere else. This is thought process, right? You think of something, backtrack, backtrack, backtrack, and eventually you get to the right solution.

Let's go over the solution for this one, and then I'll give you a hard one that is sort of like this. Then I'll want the solution from you for that one.

I'm going to draw this graph in a bigger version, so I'll have to erase this. So this is a cool concept. You will be able to do a lot of things with it.

AUDIENCE: Is this one solvable by Dijkstra?

PROFESSOR: Ya.

AUDIENCE: Just curious.

PROFESSOR: Yeah, we'll run Dijkstra on it after we transform it. These are big nodes! Why am I doing it this way?

AUDIENCE: (LAUGHS) Is that like, Seattle, Portland, San Francisco?

PROFESSOR: It's because I'm going to need room for copies. I'm going to make a copy of each node to keep track of state.

AUDIENCE: Gosh.

PROFESSOR: So, instead of one node I'll have two nodes for each node. And I have two nodes because this is what lets me keep track of state, odd path or even path. Let me erase the edges because that's not how it's going to work, actually.

Instead of having a node A, I'm going to have a node A even. And I'm going to have a copy, A odd. E even, I'm going to have to copy, V odd. C even, copies to C odd. D even, make a copy to D odd. Wait, this is E, sorry. Sorry, my bad.

[QUIET TALKING]

Suppose I got to A using an even length path. If I take the edge from A to B, I will get to B using an even length path, or an odd length path?

AUDIENCE: Odd.

PROFESSOR: So, if I get to A using an even length path, then from there I'll get to B using an odd length path. Sorry, four. Using this exact road.

AUDIENCE: A really messed-up graph.

PROFESSOR: That's a nice way to name it.

[WOMAN LAUGHING]

I've heard worse. If I get to A using an odd number of edges, how many edges will I have and I get to B?

AUDIENCE: [INAUDIBLE].

PROFESSOR: So this edge became these edges here. Now let's do the edge between B and E. I get to be using an even number of edges. I take B. Do I get to the even or to the odd?

AUDIENCE: Odd.

PROFESSOR: I get to B using an odd path. What are the lengths of these paths?

AUDIENCE: 5.

PROFESSOR: Cool. God, this is starting to look ugly, you're right.

[LAUGHTER]

Let's do A, C and stop there. A even is connected to?

AUDIENCE: Odd.

PROFESSOR: C odd, by a path the length. And A odd is connected to C even by a path of length.

Is this starting to make sense? So all the edges are going to look like this.

AUDIENCE: So are those like two independent graphs?

PROFESSOR: Well, I have edges connecting them. But, I like your question.

The reason this is ugly is because I'm looking at a 2D projection of a 3D graph. If I had a holographic display, which would be a lot nicer, then I would do something along these lines. Two pieces of paper. I would draw the original graph, and call it the even graph. Put it here. I would draw the graph again, call it the odd graph, and put it here. So this is my 2D graph. This is my map, the original map. And this is state. The third dimension is state. When I'm at a node, and I used an even number of edges to get here, I'm going to go take one edge and go to another node using an odd number of edges. So all the edges are going to go from here to here, and from here to here. Does it makes more sense now?

So the reason that looks ugly is because you're seeing the graph like this. If you could see like this, and maybe play with it a little bit, it would make more sense. It wouldn't look as ugly.

There are two graphs, the even graph and the odd graph. And the edges between them represent you doing something. When you take the highway, you transition from an even state to an odd state, and from an odd state to an even state. So that's why the edges are the way they are.

Now let me see if you guys get it. What node do I start from? What node do I want to end up in?

AUDIENCE: A, and you're ending up at B? [INAUDIBLE].

PROFESSOR: OK.

AUDIENCE: Oh, there's two--

PROFESSOR: I heard three answers. One of them is correct. Do you want to vote or do you guys want to figure it out? Talk it out? Fight?

AUDIENCE: A even, E odd.

PROFESSOR: We start from A even and we end up at E odd. Why do I started at A even?

AUDIENCE: We've seen exactly zero cities when we started out.

PROFESSOR: Yep, so the original path has length the 0, which happens to be even. And we want to end up with an odd length path. So if we go from here to here-- see how because we have two nodes out of each original node, the path is going to alternate between even and odd. And is going to keep track of the state that they didn't have before, and it's going to just do the right thing. It's magic! It works!

Let's see what is the running this new algorithm. Say my original graph had V and E. How many new vertices do I have?

AUDIENCE: Two kinds.

PROFESSOR: How many edges?

AUDIENCE: Twice?

PROFESSOR: Each edge is copied exactly twice. Which algorithm am I going to use? All my weights are positive because they're the time it takes to drive somewhere, the distance or something. So which algorithm do I use?

AUDIENCE: Breadth-first search?

PROFESSOR: We can't use breadth-first search because you have weight. So breadth-first search is not going to find the right answer.

AUDIENCE: [INAUDIBLE].

PROFESSOR: Oh, you want to use this thing.

AUDIENCE: Yes.

PROFESSOR: Well, so we don't have to worry about that because now we put the shortest path

algorithm in the black box and we have them.

AUDIENCE: Oh, so we can use them now.

PROFESSOR: We use these black boxes. But what we need to know is when to use this black box and when is this black box. Which boxes faster, by the way? Please, please give me the right answer.

AUDIENCE: Dijkstra?

PROFESSOR: All right! Dijkstra is faster.

[LAUGHTER]

So whenever we can use Dijkstra we will use Dijkstra. When can't we use Dijkstra?

AUDIENCE: When it's negative, [INAUDIBLE].

PROFESSOR: Negative--

AUDIENCE: Weights.

PROFESSOR: Negative weights. If I have weights greater or equal to 0, I'm happy I can use Dijkstra. If I have weights that are smaller than 0, well, whatever. It's going to be slower, but I can still solve it.

AUDIENCE: It's like arbitrary negative weights.

PROFESSOR: Yep. Arbitrary negative weights. So all the edges here are positive, so I'm going to use--

AUDIENCE: Dijkstra.

PROFESSOR: Dijkstra! Very good. So the running time will be, order of--

AUDIENCE: V log V?

PROFESSOR: V log V plus E. If you're a theory person, if you're in real life it's E log V.

This is exactly the running time for the original graph. The transformation only increased the graph size by a constant factor. So, same running time. Pretty good, right?

OK, let's do one more problem. Let's do the hard problem now. Are you guys getting ready for the hard problem?

AUDIENCE: If we hadn't seen this probably wouldn't have thought of it. I wouldn't even think about this.

PROFESSOR: Well, that's why we taught you this.

So I wanted to do two things. I wanted to show you that trick, which is a cool trick, keep track of state by multiplying the vertices that you have. And I wanted to go through some wrong solutions and figure out why they're wrong so you can develop an intuition. Because, when you're going to get a new problem, you're going to think of something. Unless you're really good, unless you are destined to win a Turing Award of some sort, the first solution will probably be wrong. Even if you're destined to win a Turing Award, the first thing that comes to mind will probably be wrong. I'm willing to bet Dijkstra didn't think of Dijkstra off the top of his head when he heard about the problem.

What you want to do is get an intuition. So you think of a solution and it starts getting complicated, or things start looking wrong, you want to back out and think of something else. The more things we can consider, higher chances that you're going to stumble upon the correct solution. So that's why we are building an intuition for bad solutions.

[LAUGHTER]

I'm not just saying, hey, give me a solution and now I'll tell you it's wrong. It's not just to embarrass you or something. There's that intuition building step. That's really important.

There's this network that we keep talking about. There's this highway network,

except this time it's a bit more complicated. So for each edge I have two things. I have a fuel cost, which is constant. A fuel cost is a function of the length of the road. But now, I have these realistic highways where I have traffic. If you try to go from Boston to New York, 2 AM? Three hours. Rush hour? Six hours. So we have to keep track of this in some way. Well we're going to split up the day into minutes. Say you have 10 minutes in a day. Can anyone tell me what M is, really quickly? Not a number, a formula, something.

AUDIENCE: What's 3,600 times 24? And that's seconds.

PROFESSOR: All right. So this is how many minutes in a day. This is how many minutes we have in a day, right? For each edge, we're going to have a function that's the time cost of the edge. So it's the time cost of this edge, and it's going to say, if I start at a certain time, it's going to take this many minutes to go across the edge.

For each highway I know how much time it's going to take to go across it, given when I start. And I know much fuel I'm going to consume. By the way, edges are directed now, just to make our life easy. This graph is going to be like this. Let's see if I can get this right.

AUDIENCE: [INAUDIBLE].

PROFESSOR: So the real reason I'm having this is the time to go across the highway might be different, depending which way you go.

AUDIENCE: Oh--

PROFESSOR: Getting into Boston in the morning versus getting into Boston in the evening. Which one's easier? Sorry, versus getting out of Boston in the morning. Intuitively, I'd say it's probably harder to get in in the morning because people are going to work. And it's easier to get out. But ya, I don't know either.

AUDIENCE: The 95 one's a hurdle.

[LAUGHTER]

PROFESSOR: We have an answer.

[LAUGHTER]

Our edges are going to be oriented. I want to find an itinerary that satisfies two constraints.

AUDIENCE: [INAUDIBLE].

PROFESSOR: Ya, that's not good, right? New York and New York.

[WOMAN LAUGHING]

So I want to see how fast I can get to my destination. That's my top priority. If I get to New York at 5:00 PM, I want to get there at 5:00 PM. I don't want to get there at 5:01, for example. But if I have three ways to get there at 5:00 PM, I want to choose the way that's the most eco-friendly. So, the least amount of fuel.

So out of all possible ways, the fastest way? Out of all possible fastest ways, the way that consumes the least fuel? Go. You have two minutes. Maybe five.

AUDIENCE: And we have the same costs for the--

PROFESSOR: Each edge always has the same fuel cost. But, depending on when you start, going across it is going to take a different time, too.

AUDIENCE: And we don't know what those times are?

PROFESSOR: They're going to be different for each edge.

AUDIENCE: One guess, take Dijkstra. Find the shortest path length and we look for all path lengths that are that length. I don't know.

PROFESSOR: So we have the issue that-- I might still have it. Oh no, never mind. Sorry, I erased it so it's not on your mind anymore. We have those diamonds that show you that there might be an exponential number of paths with the same length, or the same

distance.

AUDIENCE: And it doesn't help that we know we should cut off the--

PROFESSOR: Nope. So we need to do the process that I just erased. Transform the graph in. The thing is, now you're missing state, right? This misses state. This tells you how many edges you have to get from source to destination. But we're not keeping track of some state that these vital. Transform the graph to keep track of state, run, say Dijkstra, and then interpret the output.

What state do we need? Let's think of that.

AUDIENCE: Fuel cost, I think.

PROFESSOR: Sorry?

AUDIENCE: The two costs is the same for each edge. It's just the time is different. To be in traffic.

AUDIENCE: But once we have found the fastest spot that we wanted to see cut into our fuel cost range and which one do we want to take. Thinking about the priority, our priority is time. So we want to save fuels as a state, to check it?

PROFESSOR: The problem is I don't even know if fuel costs are integers. I can't keep fuel as a state. How many copies would I have? For each vertex how many copies would I have if fuel is a state? Who knows?

So let's try something else. But, you're on the right track. Let's take a variable and make it state.

AUDIENCE: So for every vertex can you keep track of what's the shortest time it took to get there?

PROFESSOR: What's the shortest time it took to get there? How would you keep track of that? Would you make that state, or how would you keep track of that? I'm not disagreeing. I'm trying to understand your--

AUDIENCE: So for example, let's say if you have B, right? Let's say it takes like one hour to go from A to B.

PROFESSOR: It depends when you start, by the way. How much time it takes depends on when you start. If you start at 8:00 AM, it might be an hour. If you start at 9:00 AM it might be two hours.

AUDIENCE: Do you need to have a finite number of states?

PROFESSOR: It's nice if it's not infinite, but it doesn't have to be constant, like we had before. It can be more than constant, for sure. So what are you thinking?

AUDIENCE: I want to use time as a state.

PROFESSOR: I like that idea. Let's try to do it.

We're going to use time as a state. How many vertices am I going to have for each vertex? So for each original vertex, how many vertices am I going to have in my new graph?

AUDIENCE: Three or four [INAUDIBLE].

AUDIENCE: The sum of all times?

PROFESSOR: So if I say, hey, I'm at this vertex at this time, then I'm going to have M vertices for each original vertex, right?

AUDIENCE: Right.

PROFESSOR: I promise that the resolution of time is minutes. I promise that this thing gives you an integer number of minutes.

AUDIENCE: Can it be put into days, though?

PROFESSOR: Suppose we can get from source to destination in one day. So let's see how it builds this graph first, and then we can figure out the rest of the things.

So for each node I'm going to make M copies of it. Suppose in my original graph I had a node. I'm going to have M copies of that node. And each copy has the original vertex, and the time when I'm there. Right? There are V of these. And there are M of these. Yes?

AUDIENCE: It totally makes sense. It's a great idea.

PROFESSOR: Yeah, you had it! Great idea. So far so good.

[LAUGHTER]

AUDIENCE: Now they need to be connected to the appropriate next one.

PROFESSOR: Let's hear how we do that. Suppose we have an edge from U to V. How are going to connect this? How are we going to transform this? How many edges are going to make from that edge?

AUDIENCE: Do you have to do it from every start time? I guess. So you'd write M edges.

PROFESSOR: M edges. From U and a start time to V and what?

AUDIENCE: T plus T C of T.

PROFESSOR: T plus T C of going through that edge. Right?

AUDIENCE: Right.

PROFESSOR: T C are going from U to V at the stop. Ya, like this. So you start from a time. So the edge points from the time that you start up until the time when you'd be able to finish. So, up until the time where you'd be off the highway and die in the next city.

And we need one more type of edges. There's one more tiny trick. If I'm somewhere in Massachusetts, and I know that it's really bad now. I prefer to wait out for a few minutes, go to a bar, and then come and drive-- OK not to a bar-- go to a food place, then drive later.

[LAUGHTER]

AUDIENCE: --vertical edges that--

PROFESSOR: --represent waiting.

AUDIENCE: You have to add one minute between them.

PROFESSOR: How do I do that?

AUDIENCE: From U to U with a cost of 1.

PROFESSOR: So from U to U at 20 plus 1. I like this better. What's the fuel cost of this?

AUDIENCE: Zero.

PROFESSOR: So here this edge had fuel cost at C, Then the new edge is going have fuel cost for the ones on the top.

AUDIENCE: Is it proportional to minutes? Speed?

PROFESSOR: It's the same. F C stays the same, no matter when we go through the highway if C is a function of the distance of the road.

This edge becomes M edges with the same cost. And then I have to have vertical waiting edges. If we had the holographic display that I talked about earlier you'd have M sheets of paper this time. You start at time 0 at your source and then your edges go represent the moves that you can make. So, you could start in Boston at 8:00 AM. And you could take I-90 and end up in-- I don't know Massachusetts city names-- Albuquerque at 9:00 AM?

[LAUGHTER]

AUDIENCE: Amherst.

PROFESSOR: Amherst, OK. So you started in Boston 8:00 AM, you end up at Amherst at 9, 10:00 AM. Right? So this is one edge. The edges represent the moves that you could make in this graph. Does this makes?

AUDIENCE: I don't understand the two last lines there. I don't really get what we were doing

there [INAUDIBLE].

PROFESSOR: Let's see what we're trying to do here. We're saying that, if I'm a node U, and I'm starting at time T.

AUDIENCE: Ya, so that's like one piece of paper.

PROFESSOR: Yep. I'm going to go on the road, right? I'm going to go on the road from U to V. So where am I going to arrive?

AUDIENCE: There at V at some time plus that.

PROFESSOR: Now we have to figure out what time we're going to arrive at. The time that we arrive at is the original time, plus whatever time I'm going to spend on the road. What's the time I spend on the road? It's not constant because I have that timetable that includes traffic.

So this is what this tells me. This is what this big, ugly formula is all about. It says the timetable for this edge assuming you start at time T. That's all there is. Nothing else.

AUDIENCE: What's the second line, then?

PROFESSOR: So the second line is waiting. So if I don't have this then I'm constrained in that I have to drive all the time. I go from here to Amherst, then I have to go from Amherst to somewhere else, and keep going, keep going, keep going.

AUDIENCE: [INAUDIBLE].

PROFESSOR: Yeah, this is where I wait for a minute. If I wait for a minute, I don't consume any fuel and I go from time T to time T plus 1.

AUDIENCE: OK.

PROFESSOR: Does this make sense? Do we want to analyze the running time for this really quickly?

AUDIENCE: Sure.

PROFESSOR: How many edges?

AUDIENCE: E times M?

PROFESSOR: OK, let's do vertices because vertices is quick.

AUDIENCE: V times M?

PROFESSOR: So, edges is, you're saying, V times M? Almost.

AUDIENCE: Pause there.

PROFESSOR: So there's a pause, right? How many pause edges do I have?

AUDIENCE: V-- M.

PROFESSOR: This is how many vertices? This is how many edges? Plug this into which algorithm, Dijkstra or Bellam-Ford?

AUDIENCE: Dijkstra.

PROFESSOR: Done! Yay, we solved the hard problem.

AUDIENCE: How do you ensure that the time's right? Do you have to go through and see if there's a path?

PROFESSOR: How do we read the solution? That's good. That's a good question. I like that.

So, we're going to have M vertices of the destination. Alright? Let's see how they're going to look like. First off, if you start at 8:00 AM, maybe you're not going to make it to New York at 8:01 AM. The vertex that says New York at 8:01 AM is probably going to have a cost of plus infinity. First off the really early times are going to have a cost of plus infinity. Not going to happen. Then, at some point, the cost is going to become finite. That's the fastest way you can get from Boston to New York. using the least amount of fuel. So the cost that you have there is the answer to our problem.

AUDIENCE: So basically, when you get there you have to enter it through all M.

PROFESSOR: All M vertices that correspond to the destination.

The costs are going to look like this. They're going to be infinity, infinity, infinity all the way up, until some point in here you're going to have your final answer. The cost to get there the fastest-- see, you can get there at 3:00 PM. This is how much fuel you have to spend to get there at 3;00 PM. But, if you're willing to wait until 3:01 PM, you're going to have the fastest cost you can have for that. If you're willing to wait until 3:02, you're going to have the answer for that, too.

So here you're going to get the whole trade-off curve of, if you're willing to wait for a few minutes, or if you're willing to wait for an extra hour, how much fuel you can save. So, I think that's cool about your questions. You had a question, too.

AUDIENCE: Why did we have V times M in the E prime expression?

PROFESSOR: First off, we have E times M, right? We're good with these. These are the waiting edges.

AUDIENCE: So we are waiting after every minute?

PROFESSOR: The waiting edges are vertex time to vertex time plus 1. How many vertices? V. How many times?

AUDIENCE: M.

PROFESSOR: So it's V times M. Because at each minute, you can be at a certain city and decide to wait one more minute to stay the same city. OK Does this make sense? Thank you guys! I'm really happy we solved the hard problem! That's good.