PROFESSOR: It was written about Route 66, which used to be the main highway between Chicago, Illinois and Los Angeles, California.

Very famous highway because anybody who wanted to go across country always took route 66 because it was the shortest way to go.

And the question is, how do you find the shortest path?

Not just any old path or a good path, but how do you find the very shortest path?

And that'll be the subject that we're going to discuss today.

But route 66, I lament its passing, but it's been largely replaced by the interstate highway system that was created by President Eisenhower.

Guess why?

Let's see, maybe the ROTC people know.

You know why Eisenhower created the interstate highway system?

Well, in public affairs, of course, there's always a distinction to be made between the explanation and the reason.

The explanation was-- AUDIENCE: To move weapons across the country?

PROFESSOR: Well, to move nuclear weapons across the country.

Let's put it in, sightly, more benign terms.

Eisenhower had observed that the German army was able to move its troops rapidly, even though we bombed their railroads into oblivion because of their auto bond.

So Eisenhower conceived that if there were ever an invasion in the United States, we too would want to be able to move our forces around on a highway system.

And consequence of that, we have a pretty good highway system and pretty awful railroad system.

It's interesting.

I'm a beneficiary of that, in a funny way, because I'm from East Peoria, Illinois.

And I was surrounded by the factors of Caterpillar Tractor Company, which made all of the tractors that built all of

those roads.

So my high school spent money like water from that huge tax base of all those factors.

Anyway, today we want to find the very best path, instead of just a good path.

And like the last time, we'll deal with, both, an example that we can set our program to work on.

By the way, can you find the shortest path between S and G?

Would you like to bet your life on the shortest path between S and G?

Probably, not.

With your eye, you can find a good path.

But you can't find the best possible path.

Today, what we're doing is probably not modeling any obvious property of what we have inside our heads.

But being able to find the best path is part of the skill set that anybody who's had a course artificial intelligence would be expected to have.

So we're going to look at it, even though it's not like many of the things we do.

A model of something that's, probably, going on in your head.

So we're going to use, both, this example from Cambridge and our Blackboard example.

Let's see, we have to caution ourselves.

Tanya, is search about maps?

No, it's about what?

Starts with a C. And the next letter is H. And it ends up being choice.

So we're talking about choice.

Not about maps.

Even though our examples are drawn from maps because they're convenient, they're visual, and helps understand the concepts behind the algorithms I'm talking about.

So let's start off by looking at our classroom example.

And I did something today that I neglected to do last time.

And that's talk to you about what I meant by heuristic distance.

It's those pink lines that I just drew on the map.

We're talking about the distance as the crow would fly between two places, even though there's no road that goes between those two places.

So in general, and we discussed last time, it's best to get yourself into a place that's close, as the crow flies, to your goal.

And of course, that's a heuristic and it can get you in trouble because it's not always true.

It would appear that being at node E is a good place to be because it's not very far from G. But in that particular case designed to illustrate the point, being close is, actually, not a good thing because it's a dead end.

But in general, it's a good thing to be close.

And we talked last time about hill climbing and beam search, being close was the objective of those kinds of searches.

And at one point, in a beam search illustration, we had C, B, A, and D. We had paths terminating at all four of those nodes as candidates for the next round of search.

And we decided on the basis of these airline distances to keep D and B, and reject A and C because they're further away as the crow flies.

Now, I repeat this even though many of you have had this fixed already in your tutorials because we're going to need this concept of heuristic distance today.

And I wanted to be sure that that point has been clarified.

So now, with this smaller map I imagine you can do, by eye, a determination of what the shortest path is.

What is it, Juana?

Can you help me out with that?

AUDIENCE: S, A, D, G.

PROFESSOR: S, A, D, G. And if you add up those distances, the distance is 11 along that path that goes from S, first to A, and then to D, and then from D to G. So Juana asserts that that is the best path.

And we're going to treat Juana as an Oracle because we're going to follow, in our initial attempt to understand these algorithms, a very important principle of problem solving.

And that is that, if you want to solve a problem, the easiest way is, usually, ask somebody who knows the answer.

Or Google, which also, probably, knows the answer.

So in this particular case, we believe that Juana knows the answer.

And she said that the shortest path is S,A, D, G, and its path length is 11.

But we don't trust her because we're applying to the same medical school and she may be trying to screw us.

[LAUGHTER] PROFESSOR: So we're going to be very cautious about accepting her answer until we've checked it to make sure that she hasn't attempted to delude us.

So how do we go about doing that?

Well, one way to do that is to check to be sure that all other possible paths that we could develop end up being, for sure, longer than the one that Juana has told us about.

So she's told us about S, A, D, G. And it has a total path length of 11.

And now what I'm going to do is I'm just going to develop the rest of this tree-like diagram.

But what I'm going to do is, I'm not going to do it in a British Museum or random way.

What I'm going to do is, I'm going to look at the choice that corresponds the shortest path that can be extended.

So the shortest path that can be extended is this one right here.

The one that just has the starting node in it.

And I could have gone this other way to B. And if I go that other way to B, then the path length along that side is 5.

And likewise, if I look at the path that terminates in A, that has a path length of 3.

So now I've got two choices.

A and B. I've got choices that extend beyond those two places.

So I'm always going to extend the one that has the shorter length.

So in this case, that would be the path that goes from S to A.

So if I from S to A, I don't have to go to D. I can also go to B. And if I go to B, then the accumulated path length is S, A, B. That's 7.

And know that we're talking now about the path, like the accumulated path length, that we've traveled so far.

Last time we were talking a lot about distances to the goal.

Heuristic estimates of how far we are from the goal.

Now we're doing exactly the opposite.

We're not considering how far we've got to go.

We're only thinking about how far we've gone so far.

So now, repeating these steps again.

I've got 7 and 5.

So I'll go over and consider the choices that go through the B node on the path S, B. And that gives me S, B, A and S, B, C. And what are those path lengths?

Well, let's see.

S, B, A would be 9.

And S, B,C would be 9.

And now the shortest path is this one over here.

So I extend that.

I go S, A, B. S, A, B. The only place I can go is C. That adds another 4.

So that's 11.

And what do I know about that path?

I don't have to take that any further, right?

Because the path length, since I've gone on that path already, is equal to the path length that Juana has told me gets me to the goal.

So it'll be foolhardy to carry on because, presuming that these lengths are all non-negative, I can't do any better.

And I can't even do as well, unless I've got a length that has 0 length.

So now that I have that idea, I can quickly finish up by saying, well, let me consider these two paths.

S, B, A can only go to D. And if I go to D, that adds 3.

9 plus 3 is 12.

Nothing else can happen there because that's 12 and I've got a path of a goal that's 11.

C, I can only go to E. It's a dead-end but I don't have to think about that because I know that the accumulated distance along this path is 6 plus 9.

That's 15.

So all of these need not be extended any further because their length, accumulated so far, is equal to or less than a length of a goal.

So I've checked the Oracle.

And although we're applying to the same medical school, Juana has told me the truth.

So now, unfortunately, Juana's not always around.

And I don't always have an Oracle.

So I'm going to have to have some way of finding the shortest path without that Oracle that I can check against.

Let's see.

What can I do?

Maybe I can do the same thing I just did.

Always extend the shortest path so far and hope that I run into the goal at some point.

And then I have to ask myself the question how much extra work did I need to do when I don't have the Oracle?

Let's just try it and see what happens.

You don't have that path to start.

So I just have S. This distance is 0.

I can go either to A or B. If I go to A, I've got a distance of 3.

Here, I've got a distance of 5.

I'll extend the path that goes S, A. That can either got to B or D. Going to B or D gives me 7 that way.

S, A, D gives me 6.

So looking across all of these and extending the shortest path so far takes me back over to S, B. So I extend those.

S, B takes me to A or C.

And those, in turn, have total accumulated path lengths of 9 and 9.

Now the shortest one is S, A, D. You see the pattern.

Now let's see.

I haven't found the goal yet.

So I can ask myself the question is any of the work that I've done so far wasted?

No because all of the paths that I've got so far are shorter than the path of the goal because the goal hasn't shown up.

So when I do my oracle checking after I found the goal, none of that work's going to be wasted.

So in the end, I don't, actually, need the Oracle.

I could just develop this graph by extending the shortest path, so far, until I hit the goal.

And then, perhaps, do a little remaining checking to make sure that all the other paths extend with a length that's greater than the path of the goal.

So if those words are confusing, let's carry on with the algorithm, and I think it'll be clearer.

So let's see.

We've got the 7, 6, and two 9s.

We're going to extend the one that's 6.

That gets this to the goal.

Boom, we've got it.

And we've got a path length of 11.

Note, though, we can't quit because we have to be sure that all other paths are longer than 11.

So now we have to carry on with the same algorithm that we started with.

The Oracle checking algorithm.

And when we do that, we look for this shortest path, so far, that has not been extended.

That's B, S, A, B. That goes to C. That's 11.

So we're done there.

A goes to D. That adds 3.

That's 12.

C goes to E. That adds 6.

That's 15.

And sure enough, we're done.

OK?

Elliot?

AUDIENCE: Does it know that there's know that there isn't a chance that you could have a zero distance extension from the [INAUDIBLE]?

PROFESSOR: The question is, does it know that there's no zero distance length that's coming up.

That's an implementation detail.

This guarantees you'll find a path that's as short as any that you can possibly find.

But there might be others if they're zero-length lengths.

As long as they're non-negative lengths, we're safe.

We've got a shortest path.

So that was easy.

And now we can repeat the exercise with our more complicated map of Cambridge.

First of all, let's do depth first just to recall what that looks like.

That is, certainly, not a short path.

So let's try this idea, which, by the way, bares the label branch inbound.

Let's try branch inbound on the same map.

And there it goes.

Each of those little flickers is trying another path.

So you can see it's working it's guts out to find the shortest path.

It's almost there but it's almost a pathological case.

Or it's almost doing British Museum.

There it's finally found the shortest path.

Now there are some things we can ask about that.

But first of all, before I ask anything about it, I'd like to get the flow chart up on the board because we're going to decorate that flow chart, a little bit, as we go.

So the first thing we do is initialize queue.

Then we're going to test first path on the queue.

Then we might be happy because we might be done.

We might have a shortest path to the goal.

Actually, that's not quite true, is it?

We can't really quit until every other path is it.

Well, that's interesting.

If the first element on the queue gets us all the way to the goal, and we sorted our queue by path length, are we through as soon as that first element on the queue gets us to the goal?

Yeah because every other path must have been sorted beyond it.

And therefore, it can't offer us a shorter path to the goal.

So if the first path is a path to the goal we're done.

Alas, it usually isn't.

So we'll extend first path.

We're going to put all those extensions back on the queue, and then we're going to sort them.

So that's, pretty much, the same as we did last time.

We're always going to put the elements back on the queue.

We're going to look at the first element the queue and see if it's a winner.

If it is we're done.

If it's not, we're going to extend it.

And then go back in here and try again.

Well, sort of.

But we noted that this did a awful lot of work because if we look at those statistics up there, it put 1,354 paths onto the queue.

That's the N queueing part.

And then it extended 835,000 paths that had come to the front of the queue.

Now I'd like to give you an aside because it's easy to get confused about N queueing and extending.

In all of the searches we did last time, it would have been perfectly reasonable to keep a list of all the paths that we had put onto the queue.

An N queueing list.

And never add a path to our queue if it terminates in a node that some other path terminate in that has already gone to the queue.

What I said last time was let us keep track of the things that have been extended and not extend them again.

So you can either keep track of the nodes that have been extended and not extend them again.

Or look at the paths with nodes that terminate, and blah, blah, blah and been put on the queue, the queued ones.

And not put things back on the queue again.

And I think, last time, I may have put a column in there that said N queued.

It should have been extended.

Even though N queued worked last time, only extended works this time because we want to be sure that anything we extend is a short path.

So the N queued idea doesn't work, at all, for these optimal paths.

So now I want to come back over here off the side bar and say that we're keeping track of all of the nodes, all of the paths that end in nodes unless they have already been extended beyond that particular place.

So we need to decorate our algorithm here and say test first path and extend the first path if not already extended.

Because you can see that in the example I had, so far, we did that same silliness that we talked about last time.

We extended paths that went through A more than once, like so.

Would it ever make sense to extend this path?

No because we've already extended a path that got there with less distance.

Will it ever make sense to extend this path?

No because we've already extended another path that gets to be by a shorter distance.

So if we keep an extended list, we can add that to branch inbound to our advantage.

So let's see how that would work on the classroom example.

And then we'll do Cambridge.

So this is bridge inbound, plus an extended list.

And I do mean extended.

Not in the N queued list.

N queued list won't work here.

So let's see, I start off the same way as I did before.

S goes to either A or B. That's a length of 3.

That's a length of 5.

So I extend A. That goes to either B or D. But B is as if it wasn't there at all.

Oh, sorry.

Hang on.

B goes there.

And those path lengths are 7 and 6.

And now I look around on the board, and I say what is the shortest path so far?

And it's B. So I extend that to get to A and C with path lengths of 9 and 9.

And what's the shortest one next?

It's D. And that goes to G. And the path length is 11.

And what's the shortest one on the board?

The one that has to be extended next.

That's this one that gets to B. But I've already extended a path that get to B. So I don't, actually, do that extension.

So I've saved some work.

But I've got to go over here and do these two now.

But wait.

I've already extended B. I've already extended A, so I don't have to do that one either.

The only one I have to do is the one that goes to C. And that those then to E with a path length of 15.

And I'm done.

So if you compare this one with a previous one, you can see that there might be vast areas of this tree that are pruned away and don't have to be examined all.

So now, just for the sake of illustrating that, I would like to keep track of just one of those statistics.

The number of extensions.

So for this particular example, case one, the number of extensions was 835.

Why don't you see if you can guess to yourself what it would be if I use this concept of an extended list.

See, I'm not going to extend anything I've already extended because it's guaranteed to have a longer path length then something that already got to that same place.

So it makes no sense to do it.

So let me change the type to branch-and-bound with an extended list.

I'm going to turn the speed down a little bit so we can watch it.

It might take the rest of the hour.

Who knows?

Still doing a lot of work.

Still examining a lot of paths.

Well, look at that.

Instead of 835 extensions it only did 38.

So that's a pretty substantial savings.

And you would never not want to do that.

So note that that's a layering on top of branching out.

That's not a different algorithm.

It's an adjustment improvement to the algorithm, and it makes it more efficient.

So this whole thing is based on what I call the dead horse principle.

As soon as we figure out that a path that goes to a particular place can't possibly be the winning path, we get rid of it, and don't bother extending it.

It's a dead horse principle.

But if we look at this example, what's the shortest possible length of a path that's already gone from S to B?

What do you think, Tanya?

Well, first of all, it can't be less than 5 because we've already gone that distance.

So when I say what's the shortest length of any path that there could possibly be that goes from S to D. We know it's at least 5.

But can we say something more about it?

Especially, when we look at these airline distances, and note that this airline distance is 6, and that's a little more than 7, and that's a little more than 7.

So what do you think?

So it's gone from S to B, and the question is what's the shortest path that could possibly be that had started out going from S to B?

11 right?

Because we can't have a path that's shorter than the airline distance.

If there were a straight line road from A to G, its length would be 6.

But there isn't.

So that gives us a lower bound on the distance that we have along that path.

So we're using the accumulated distance, plus the airline distance, to give us a lower bound on the path that we've started off on that goes from S to B.

Once again, let's solidify a little bit by simulating the search and seeing how it turns out.

Not just I did last time, I'm going to forget that I've got an extended list.

I don't want to carry both of those things around with me at the same time.

So forget that we've got an extended list.

We'll bring all those back together a little later.

So we're going to forget what we just did there.

And instead we're just going to use this concept of an airline distance and see what happens.

As before we start with a starting node.

We have two choices as always.

We can go to A or B. And the accumulated distance, if we go to A, is 3.

And then accumulated distance if we go to B is 5.

But now we're going to add in the airline distances.

So the airline distance from A to G is a little more than 7, which is 10 plus.

The airline distance from B to G is exactly 6.

So that gives us 11.

And following the procedure we've all been using already so far, we're going to extend the path that seems to have the shortest potential.

Now it's the shortest potential distance S to G. So that must be this one here.

So from A we can go to B or D. The accumulated distance S, A, B, is 7.

The airline distance is 6, so that's equal to 11.

Standard arithmetic 13.

The distance S, A, D. That is 6 plus a little more than 7.

So what's the accumulated distance?

S, A, D is 3 plus 3 is 6.

AUDIENCE: [INAUDIBLE].

PROFESSOR: What?

AUDIENCE: The airline distance from D would be 5.

PROFESSOR: Would be 5, right.

So airline distance, in this case, is the same as the actual distance.

So the accumulated distance is 6.

The actual distance is 5.

So that's equal to 11.

So now I've got two 11's on the board.

And simulating what we'd ask you do on a quiz, we don't know which of those is going to be better.

They've got a tie score.

So what we're going to do is we're going to choose the one that's lexically least.

So B comes before D. So we'll expand B. And that can go to either A or C. And we have to calculate the best possible distance that goes along those paths.

The accumulated distance S, B, A. S, B, A is 9.

So that's 9 plus 7 plus.

That's 16 plus.

This has an accumulated lead of distance of 9.

Also plus 7 plus.

Also 16 plus.

Well, now let's see.

Things are shaping up pretty well because this one has the lowest score so far.

We extend that to G. And now the accumulated distance is 11.

The airline distance is 0, so that's 11.

And that's smaller than everybody else.

So we've got.

So now compare this one with our branch inbound graph.

And you see, once again, we've done considerably less work.

And that, in many practical cases, means that instead of taking more than the remaining lifetime of the universe to complete the calculation, it can happen in a few seconds.

But let's see how it works on the example.

So I'm not going to use the extended list.

I'm just going to use this idea of using a lower bound on the distance remaining, the airline distance, and see what

happens.

So this time, the number of extensions is 70.

So it didn't do quite as well as working alone as the extended list did working alone.

So we immediately conclude that the extended list is more useful than using one of these lower bound heuristics.

By the way, this is called an admissible heuristic.

If the heuristic estimate is guaranteed to be less than the actual distance, that's called an admissible heuristic.

Admissible because you can use it for this kind of purpose.

So it looks like the day extended list is a more useful idea than the admissible idea.

Right?

What do you think about that, Brett?

Am I hacking?

Am I joking?

AUDIENCE: I think you're judging prematurely.

PROFESSOR: Why am I judging prematurely?

What do you think it might depend on?

AUDIENCE: The fact that we're using extensions and the extended list pretty much guarantees you can only extend each node once.

PROFESSOR: Well, Brett has sad something unintelligible that I can't think how to repeat.

What he meant to say, though, was that-- [LAUGHTER] PROFESSOR: --in these cases, it almost always depends on the problem itself.

If you change the problem, you may get a different result.

So why don't we change the problem and see if we get a different result?

So instead of starting on the extreme left, let's start in the middle and see what happens.

So I'll readjust my starting position to be right there.

Oops, that's the wrong adjustment.

And we might as well start by getting our baseline branch-and-bound without anything.

And for that one, maybe, we'll speed it up a little bit.

So that gives us 57 extensions.

It's an easier problem.

So let's try it with the admissible heuristic.

That went too fast.

Wow, still pretty fast.

Six extensions.

What do you think this number's going to be?

Closer to six or closer to 57?

Better than six?

Worse than six?

Well, let's think.

What we're going to do is we're going to just not repeat any movements through the same node again.

But it's not going to do something very important for us.

It's not going to keep us out of the left side because it has no idea of the remaining airline distance to the goal.

So let's see if that's true It sure is.

Look at that.

It is, foolishly, spending a lot of its' time doing something we would never do.

Namely, looking over there on the left side.

So this time, the number of extensions is 35.

So in case two, the admissible heuristic does very much better.

In case one, the extension thing does much better.

But wait a minute, would we ever not want to use both at the same time?

We wouldn't want to use just one of these, right?

They both have the possibility of doing us a lot of good.

So maybe if we put them in harness together, we'll get something that's even better.

And when we do that-- see here, the extended list is a layer on top of branch and bound.

The admissible heuristic is another layer on top of branch and bound.

If we put those together, we get something called A star.

So A star is just branch and bound, plus an extended list, plus and admissible heuristic.

So let's go back to our original problem and try A star on that.

We're running this at a pretty slow speed because we're expecting it to be a lot more efficient than the original branch and bound.

And sure enough it is.

The number of extensions is 27.

So look at that.

A lot better than either of those working independently.

Now I can stick the thing in the center and see what happens then.

So in this particular case, the extended list didn't, actually, help us because our admissible heuristic was channeling us so tightly toward the goal it didn't matter.

So it all depends on the nature of the space that you're trying to explore.

By the way, you know how the whole works, right?

So what you want to do is you want to extend the first path and sort.

But not just by accumulated distance.

Sort by accumulated distance plus admissible heuristic.

But what are the theoreticians?

You must be complaining.

Sort's expensive.

Do we actually need to sort?

No, we don't actually need to sort.

What do we to do?

AUDIENCE: We just need to keep track of what's the minimum.

PROFESSOR: We just need to keep track of what's the minimum.

So we don't need to, actually, do that sort.

That's an unnecessary computation.

So instead, we can test, not the first path but the shortest path.

And now you have it.

Now you have the whole of A star.

And now you can go home, but I don't think you should because I'm about to show you that this idea of admissibility, actually, leads to certain screw cases that we're very fond of asking about on exams.

So it turns out that the admissible heuristic, in certain circumstances, could get you into trouble.

It doesn't look like it could because, logically, nothing I've said seems strange or questionable.

But that's because I've been working with a map.

And it turns out that if you work with a map then admissibility is a perfectly sound way of doing an optimal search.

But, Travis, is search just about maps?

No, search is not just about maps.

So we may have non-Euclidean arrangements that will cause us trouble.

So I'd like to illustrate that with the following example.

It's not going to be a large map or a large graph.

S, then go up here to A or down here to B. Then they merge at C. And then they go out here to the goal, G.

And the actual distances are 1, 1, 1, and 10.

And over here, we'll make that 100.

So it's a kind of oddly constructed map, but it's there because we need a pathological case to illustrate the idea.

Now that's the actual distances.

And if we did branch and down with an extended list, everything would work just fine.

But we're not.

We're going to use an admissible heuristic.

And we're going to say that this guy has an estimated distance to the goal of 100.

This guy is 0.

And this guy is 0.

Now, 0 is always an underestimate of the actual distance to the goal, right?

So I'm always free to use 0.

Is that 100 OK?

Yeah because the actual distances is 101, so it's less than that the actual distance.

So it's OK as an admissible heuristic.

So these numbers that I put up here, together, constitute an admissible heuristic set of estimates to the goal.

So now, let's just simulate A star and see what happens.

So first of all, you start with S, and that can either go to A or B. The actual distance is 1 plus an estimate on the remaining distance.

That gives us 100 plus 100.

That's equal to 101.

If we go to B instead, the actual distance is 1 plus the heuristic's distance is 0, so that's equal to 1.

OK, good.

So now we know that we always extend the shortest path so far.

Did I goof this, or are you asking a question?

AUDIENCE: [INAUDIBLE]?

PROFESSOR: Yeah, when I say actual, it's the actual distance that you've traveled.

AUDIENCE: But that's [INAUDIBLE].

PROFESSOR: So wait a second.

If I go from S to A, the actual distance I've traveled is 1.

AUDIENCE: I meant like, does the map-- PROFESSOR: So now I'm taking the sum of the actual distance, plus the estimated distance to go.

AUDIENCE: All right.

I'm just wondering if the original map has to be [INAUDIBLE].

PROFESSOR: See this is not a map.

She was asking if the map has to be geometrically accurate.

See, this could be a model of something that's not a map.

And so, I'm free to put any numbers on those links that I want, including estimates, as long as they're underestimates of the distance along the lengths.

So this tells me that my estimated distance here, so far, is 1.

So I'll, surely, go down here to C. And if I go to C, then my accumulated distance is 11.

And my estimate of the remaining distance is 0.

So that's a total of 11.

So now I'm following my heuristic again and saying what's the shortest path on a base of the accumulated distance plus the estimated distance?

Here, the accumulated distance plus the estimated distance is 101.

Here, it's only 11.

So plainly, I extend this guy.

And that gets me to the goal.

And the total accumulated distance is now 111 plus 0 equals 111.

And that's not the shortest path, but wait.

I still have to do my checking, right?

I have to extend A. I when I extend A, I get to B. And now, when I get to B that way, my accumulated distance is 2 plus my-- oh, sorry.

S, A, C.

My accumulate distance it 2.

My estimated distance is 0, so that's equal to 2.

So I'm OK because I'm still going to extend to this guy, right?

Wrong.

I've already extended that guy.

So I'm hosed.

I won't find the shortest path because I'm going to stop there.

And I'm going to stop there because this is an admissible heuristic and that's not good enough unless it's a map.

It's not good enough for this particular case because this is not geometric.

This cannot be done as a map on a plane.

So that's a situation where what I've talked to you about, so far, works with branch-and-bound.

Works with branch-and -bound plus an extended list.

But doesn't work when we added an admissible heuristic.

So if we're going to do this in general, we need something stronger than admissibility, which works only on maps.

And so the flourish that I'll tell you about here in the last few seconds of today's lecture is to add a refinement as follows.

So far, we've got admissibility.

And if we want to write this down in a kind of mathematical notation, we could say that it's admissible if the estimated distance between any node X and the goal is less than or equal to the actual distance between X and the goal.

That's the definition of admissible.

As long as heuristic does that it's admissible.

And A star works if it's a map.

But for that kind of situation where it's not a map we need a stronger condition, which is called consistency.

And what that says is that the distance between X and the goal minus the distance between some other node in the goal, Y. Take the absolute value of that.

That has to be less than or equal to the actual distance between X and Y.

So this heuristic satisfy the consistency condition?

Well, let's see.

Here the guess is 100.

Here it's 0.

So the absolute difference is 100.

But the actual distance is only 2.

So it satisfies admissibility, but it doesn't satisfy consistency.

And it doesn't work.

So you can almost be guaranteed we'll give you a situation where if you use an admissible heuristic you'll lose.

And if you use a consistent heuristic, you'll still win.

So how can we bring this back into the fold?

Well, we can't use that heuristic.

It's no good.

But if this heuristic estimate of the goal were 2, then we'd be OK because then it would still be admissible.

But it would also be consistent.

So the bottom line is that you now know something you didn't know when you started out two lectures ago.

You now know how MapQuest and all of its' descendents work.

Now you can find an optimal path, as well as a heuristically good path.

You see that if you don't do anything other than branch and bound it can be extremely expensive.

And you can even invent pathological cases where it's exponential and the distance to the goal.

So because it can be so computationally horrible, you want to use every advantage you can, which, generally, involves using an extended list.

As well as-- no laptops, please.

It still holds.

No smoking, no drinking, and no laptops.

So you're going to use all the muscles you can.

And those muscles include using an extended list and an admissible or consistent heuristic, depending on the circumstances.

And so, I think we'll conclude there since our time is up.

And Elliot, you can ask a question after class.

Why don't you come up and ask it now?