PROFESSOR: This week's problem is from the 2008 quiz one on search. It is motivated by the search of evil overlord, Mark Vader, who is shopping for a new evil stronghold. He starts from his current stronghold, which is S, the Depth-First Search Star. Now the Depth-First Search Star has the following qualities. It has a small, thermal Exhaust Pipe Weakness. It has the quality of That's No Moon, but it does not have a Race of Enslaved Minions, or a Secret Escape Route, or Sharks with Laser Beams.

Now when I originally wrote this problem in 2008, there was another quality that very important. It was a giant Laser, and it did have that property, as did Dr. Evil's Moon Base, but that was removed to make it easier to solve. So we're going to solve it without the giant laser, that we can just have the Sharks with Laser Beams.

So Mark Vader has gone to Ackbar's Emporium of New Evil Strongholds, which are listed on the left, and he's trying to figure out the best way to get from his current stronghold to his goal stronghold, the 603 Fortress, which has the admirable qualities of no small thermal Exhaust Pipe Weakness, and it still has That's No Moon. It has a Race of Enslaved Minions, and a Secret Escape Route, and Sharks with Laser Beams. So it's got everything you would want and no weakness.

However, he can only move between fortresses that have exactly one difference. Fortunately, Mark remembers how to perform the search techniques he learned in 6.034 from his mentor, Emperor Palpatine. So we've got several fortresses here. We've got the Depth-First Search Star. We've also got Shaoul Ghoul, which has the qualities, basically, of That's No Moon, and none of the other qualities. Dol Guldur here, has got the Exhaust Pipe Weakness somehow, despite being Sauron's stronghold. It's not a moon, and it has those enslaved minions. I guess the Orcs are enslaved.

Moonraker, here, only has the Exhaust Pipe Weakness, so it's not very good. But James Bond didn't need to deal with stuff. Zeal Underwater Palace has That's No Moon and a Secret Escape Route. Zeromus' Lunar Core has an Exhaust Pipe Weakness and a Race of Enslaved

Minions. Whalers of the Moon Ride has the Exhaust Pipe Weakness, Race of Enslaved Minions, and Sharks with Laser Beams.

6.03 Fortress we've already been over. Atlantis has all qualities except for a Secret Escape Route. Willy Wonka's Factory has some of everything. It is, after all, Willy Wonka's Factory. Highlight, the Race of Enslaved Minions, the Oompa Loompas. And Dr. Evil's Moon Base is only missing That's No Moon, because it is a moon, because it's a moon base.

So if you got all the references that were used in making those fortresses, I'm sorry. There's nothing I can do for you. Now, Mark, being a clever evil overlord, realizes that he can produce a graph of the exploration choices with edges joining strongholds that differ by just one feature. So although this is a graph here, as Patrick often says-- and you get to hear again right now-- search is about choice, not just about maps.

So we're not moving around anywhere in the real world, but we certainly are going to move around this graph to pick between these many stronghold choices. How can we decide where to travel? Well, we're about to find out. So we've got this lovely graph right here, and now we're going to get to do depth-first search on the graph. Now there's a lot of ways to do search, and you've seen Patrick do it, you may have seen some people do it in recitation. And I'm going to offer you guys a unique opportunity to see me do search in one of several ways, or possibly more.

I have the tried and true method. You have the queue, or agenda, or whatever you want to call it. You keep track of it at every level. You make damn sure that everything you're doing is right. It takes a long time, and you get the answer right. Or there's one where you only work with the goal tree, and you draw it really fast, and you might make a mistake, but it's going to solve it more quickly.

So who would rather see the reliable, but slower, approach? Who would rather see the faster, but less reliable, approach? All right. Monte Carlo people here, rather than Las Vegas. If you don't know what that is, you'll learn it in algorithms class. All right. Then there's a third approach that I probably won't show you unless there's overwhelming favor for it. That is the approach that is manically fast and will solve depth-first search in no time at all, but it is very likely to make mistakes.

That is the approach that I use when solving depth-first search. I've used it enough times that I don't make mistakes that often, but even I still do. So does anyone want to see that one? OK.

Not that many people. Well, it doesn't take much time, but I'm just afraid that if I show it you, that you'll be like, ooh, that's the only way to do it. So we'll save that for later if we have more time. It essentially, for people that are interested, involves just using your chalk or finger to trace through the tree really fast and figure out what it's doing by drawing little tiny lines. It is very fast, but it's not very accurate. And you have shown, basically, no work if you get it wrong, so shame on you. OK.

Well, let's do the somewhat faster way where we don't draw out the entire agenda, and then people who were the almost nobody who raised their hand for that way can check it out in tutorial the way that you do it using the agenda. So let's do it using the goal tree. So it's a bit faster that way. So we're going to start, starting at S, the start node, and going to G, the goal node. That's a standard notation, but make sure that when you're taking the test, you check to make sure where the start node is, where the goal node is.

It's pretty much an asshole move to have a G node that isn't the goal node, but sometimes there won't be an S node. That's a pretty good clue that something else is the start node. Now there's a few little white star ideas that we have, or silver star ideas, for search. One is lexicography. What it lexicography? Who cares? Well, the most famous lexicographer I know of is Samuel Johnson, who wrote a really famous dictionary. And the only reason we care about it in this class is because you'll always see the instructions "break ties in lexicographic order." And then you might be like, wow, that's really rather a wordy word, or a sesquipedalion word, or whatever really long word you want to use to describe the word "lexicographic." What the heck does it mean?

Basically, it means alphabetical order, like you would do in a dictionary. So for instance, in alphabetical order, A comes before B. So you would go to A before B. No, even beyond that point, there's several ways to do a lexicographic tiebreak. And it's not always consistent between algorithms how you would do a lexicographic tiebreaker. In the most recent time that Patrick was kind of randomly talking to the staff, he made it clear that the way he would like us to do it, and therefore, the way we'd like you to do it, is to tiebreak based on the very latest nodes at the end.

So you might say, that's not really lexicographic. Wouldn't SAB come before SGA in the dictionary? And my answer to you is, yes, it would. Yes, yes, it definitely would. However, that is apparently what we're doing, according to the last thing that Patrick said, and we will keep you updated if he decides to go back to dictionary order.

So that's lexicography. You might be wondering what the picture here is. Well, this is our friend, the ouroboros. Long has it been a symbol in alchemy of infinity, eternity, or maybe a sort of infinite energy engine, turning iron into gold. But today, unfortunately, the ouroboros is an endangered species because no biting your own tail in 6.034. A lot of people mess up from this. It's an honest mistake. I mean, we've gotten some emails from staff members who couldn't solve one of the problems because they forgot about this. So it's OK if you forget, but try your best to remember. No biting your own tail is the only smart thing our system does.

You'll remember last week when we were talking about rules, and we were saying that the rule system is not a smart creature. It doesn't know that "not Polly is dead" should be the same as "Polly is not dead", or something like that. It's too dumb to figure that out. Well, this system is also dumb, but one thing it knows is that Patrick really hates if the same node appears twice within the same path. And it will destroy that immediately, before even adding it to the queue. It's gone. It's not considered. It's out of there. So very important, sad ouroboros, no biting your own tail.

All right. So now that I've gotten through that, let's actually solve the problem with some depth-first search. How will we do that? Well, we're not going to use the queue, as popular demand, we're going to use a goal tree. So we'll start at node S. What are our choices at node S? Well, I'm going to force you guys to help. So it won't be quite as fast, but it'll be fun. So what are our choices at node S? You.

AUDIENCE: Me?

PROFESSOR: Yes.

AUDIENCE: A or B?

PROFESSOR: Not quite, and this is something I like about this problem. You got most of them. Do you see that there might be another choice? Everyone? Yeah, C. This is a big problem that has happened on a few different quiz problems where there's a sort of a grid that looks like a tree, or a graph that looks like a tree, where people aren't as willing to go up as they are to go down. You can see it on some of the other past quiz problems too.

Make sure that you check the connectivity, and also note that unless otherwise specified-- and I don't think we've really done this much-- the little connecting edges in our graphs are

bidirectional. You'll see a big arrow, and probably instructions written at least eight font sizes higher than all the rest of them, and bold, if we're going to do something different for this quiz. It happens that they do it sometimes, where I'll just say in recitation, we never do this. And then on the quiz, they do it.

But it will be in giant bold letters, which also leads me to another silver star idea that I forgot to tell you guys last week. So I hope people didn't ditch out on me, like, oh, Mark's boring. We won't come next week. And then miss this one because it just came to my mind. That is, read the instructions. It is a very important thing for the quizzes, and Patrick will, in a later lecture, tell you guys, ask why five times, and he'll explain why that's a gold star idea.

My parallel to this is read the instructions five times. Maybe not five, maybe three or four, but at least three. Read the instructions. Read them again. Then read them a third time only paying attention to whatever is bold and eight sizes larger than all the other ones, cause there's going to be something there that's that size. And it going to be the thing that everyone misses. So make sure you're not everyone, and you read the instructions a lot of times.

But anyway, so yeah, we've got it right. S goes to A, B, and C. OK. We're looking at A, B, and C as possibilities. Everyone, help me make the lexicographic tiebreak. Where do we go?

AUDIENCE: A.

PROFESSOR: A. That's right. Actually, I'll call on you guys for other ones, but that first step is pretty simple. We could have everyone do it together. Everyone, A only leads to?

AUDIENCE: D.

PROFESSOR: That's right. Why doesn't it lead to S as well?

AUDIENCE: [CHATTER]

PROFESSOR: No biting your own tail. Everyone's right. Good job, everyone. All right. S leads to these three. You go to A, go to D. Dead end. Backtrack. OK. When we backtrack, we backtrack up to A. There's no other children. We backtrack up to S. Where will we go now?

AUDIENCE: B.

PROFESSOR: B. It's the next one alphabetically. All right. B can't go to S, so it only goes to?

AUDIENCE: H.

PROFESSOR: Yes. When we're at H, we can go to?

AUDIENCE: [CHATTER]

PROFESSOR: F or I. That's right. You guys get this. Depth-first search is easy. All right. But we choose? F. That's right. When we're at F, we can go to? E and J. That's right. We'll choose? E. When we're at E, we can go to? We can go to C. It's not on this particular path. People are correct when they say C. When we go to C, we can go to nowhere. We're dead. Backtrack.

We backtrack. We can't do anything at E. We go to F. Now I forgot to tell you guys, but this is an important note. Someone's going to get this wrong. It's going to be one of you. Look around you, all through the room. At least one of you is going to do this. So, now I'm going to tell you not to do this, and then hopefully, that will still be true, but for fewer people.

When I went to D, and then I backtracked, and I went to S, how many times did I backtrack? Once. I backtracked once. This would have been more obvious if we were doing the really slow boring agenda way, or queue way, because when we got to SAD, and we expanded it, and there was nothing left, we throw it off the top of the queue, and go to the next thing on the queue. And it turns out the next thing on the queue is SB. That step was only taken once. Since we're using the goal tree, which is faster, it looked like we did two, but we didn't. And if you use the queue, you'll see that because it'll go S, and then we expand that, SASBSC. SBIC OK. We expand SA. SADSBSC. We expand SAD. It's dead. SBSC. One backtrack. We're back at SB.

So, so far we've done two, not four. It's pretty intuitive to say that you did two there. It turns out you didn't because of the algorithm that's backing what lets us do this goal tree search. So try to make sure you're not the one who says two. And I, on the other hand, will try to make sure that we don't take off too many points if you do. Question?

AUDIENCE: So any time you backtrack, regardless of how much it chains, any backtracking is just considered once.

PROFESSOR: It is always going to be considered one step. It's possible to backtrack two times in a row. Like, if B didn't go to anything, instead of going to H, then we might backtrack twice before we got to SC. Generally, anytime you draw a swizzle, if you're like me, and you draw a swizzle-- I suggest the swizzles. They're very nice little things. But any time you draw a swizzle on your

graph, you have backtracked once. You could even write BT next to the swizzle, and go back and count those, or even just count the swizzles at the end. You've got it. You got the answer. Because they often ask, how many times have you backtracked?

All right. So good questions, everyone. So anyway, we backtracked from C. Nothing at E. We go back. We were at F. So we were at SBHF. We go to J. It's the only choice. At J, we can only go to?

AUDIENCE: I.

PROFESSOR: And at I, we can only go to? And we win. It's not an optimal search, so as soon as we see anything with a G on the queue, boom. Winner. All right. And we're done. We did it. That wasn't too bad. For those of you who are vaguely interested in seeing how I would do it the really super fast way, it goes something like this.

All right. ABC. A comes first. Only D. Backtrack. All right. B comes first. H. F comes first. E. C. Nothing. Backtrack. We came that way. JIG. So that's the really fast way to do depth-first search. Don't do that, kids. We don't like-- well, maybe you can. If you get it right, we're not going to take off points usually. Generally, when we say draw the goal tree below, it's just assigned partial credit.

However, we are pretty strict about that. If anything is wrong, except for maybe, like, OK, you write everything exactly right, and forget G at the end. If anything that's not completely understandable is wrong, you will probably lose all of the points, and it'll be a lot of points if you don't draw the goal tree. I will emphasize drawing the goal tree is a good idea.

All right. So now let's do a breadth-first search. Before we do the breadth-first search on this tree, I will tell you guys that there is also a fast way to do the breadth-first search which is less risky, and it really depends on how nice they are about what they ask for. In this case, the breadth-first search question asks, what path does Mark find using breadth-first search? Rather than saying, what nodes does he expand in order, or anything like that.

That is important. If that question is asked you, there is a trick that will let you solve it very, very quickly. In fact, faster than depth-first search. You can solve it by inspection in about 30 seconds. Does anyone know what the answer is? What path did he find?

AUDIENCE: [CHATTER]

PROFESSOR: That's correct. So the answer is? On this graph, the answer is?

AUDIENCE: SBHIG.

PROFESSOR: SBHIG. That's the answer. You would have your five points. Did people see that? It doesn't always work, so we're going to actually solve it. But did everyone see that sometimes you can get away with not doing it? Because breadth-first search is guaranteed to give you the path, as we heard correctly, with the least number of jumps, and if there are more than one that tie with the least number of jumps, you can just lexicographically figure it out, in this case with actual dictionary order.

But SBHIG is the only one. Let's do an actual breadth-first search though, so we can feel better about ourselves. OK. So you've got S, and S goes to, as we saw before, ABC. Maybe that doesn't have to go quite so high. All right. As we saw before, S goes to ABC. And you already told me that A goes to D, and you told me that B goes to H. But what does C go to? E. That's right.

As you can see, we're expanding it level by level, left to right. All right. So SAD, were does SAD go to? SAD goes to nowhere. It's dead. SBH. Now, wait. You might say, wait a minute. Patrick said that we're using this weird dictionary order where E is at the end that comes before H. That's our tiebreak order, but it turns out that breadth-first search and depth-first search don't sort in any way. It's very important. They don't sort the paths that are currently on the queue.

So you're going to just go left to right, left to right, left to right. And only at each node are you going to break ties in lexicographic order. All right? So SBH. H goes to, as we already know, F and I. E, we think we already know, but we don't quite because this is E coming from the other direction. E, this time goes to F. That's right. Well, actually, you guys do already know it. All right. Great.

Now we come over here. HF goes to, as we already know, E and J. I goes to, as we don't already know, HI goes to G and J. That's right. And as it turns out, by an implementation detail, we're done. Questions? Does it not expand depth at all? Now this is an implementation detail. It's perfectly sane and reasonable to make a breadth-first search that likes to finish its entire level that it's working on. However, in our implementation, and we would have seen this if we had been pedantic and drawn out the entire queue-- that's another reason why drawing out the entire queue is, as I said, more reliable.

In our implementation, since it's not an optimal search, the moment anywhere on the queue, you add something with a G at the end, you finish. And because of the fact that the way breadth-first search does its mojo, is that it adds everything to the end of the queue. That's how it does it level by level, right? It adds it to the end of the queue, instead of the front. Well, then you'll add it to the end of the queue, and then you will have it on the queue with a G. So you won't have to do SCEF. Another question?

AUDIENCE: I'm just wondering, so the breadth-first search, there was no backtracking [INAUDIBLE] had no other--

PROFESSOR: Ah. That's a good question. So the question was, so for the breadth-first search, there was no backtracking. D died. Why didn't we backtrack or something? The answer to that one is, for breadth-first search, backtracking doesn't really, it isn't really a thing, like it is for depth-first search. Why? Well, because breadth-first search, we're sending our infinite monkeys down every path. In depth-first search, we're really focused in now. We're like, we want to get there. We want to get there. Ooh, this way, this way, this way, this way. And might have gone the wrong way, and then we hit a dead end, we're like, oh, crap. And we go backwards.

But for breadth-first search, we really are like an evil overlord. And we're like Mark Vader saying, storm troopers, go every direction. And then from there, go every other direction. And so, even though, yes, when we got to D, some of the storm troopers hit a dead end, and probably it was reflective, and they shot it, and hit themselves or something like that. Meanwhile, the storm troopers we sent to B and C are still fine, so we don't need to backtrack.

AUDIENCE: Cause there are, like, other troops going down the tree.

PROFESSOR: Yeah, there are other troops going down the tree in every direction. Whereas in the depth-first search, we only sent-- we were like, there are definitely no droids on this escape pod. Send everyone this straight direction, and we only sent them to A and D. And then, so we had to backtrack because we hit a dead end. Does that make sense? Another question?

AUDIENCE: [INAUDIBLE]

PROFESSOR: So the question is, in this breadth-first search, did lexicographic order ever come into play? The answer is yes, in a very subtle and sneaky way. Which is, I wrote E before J, F before I, and G before J. If I was not using lexicographic order, it might have been reasonable to write, for instance, when I was expanding I, to write J before G because J was higher up on the tree

or something like that.

But the only way it came into play is that I wrote them left to right in alphabetical order every time, and I wrote ABC. Another question?

AUDIENCE: We see the F node twice. Suppose that the goal node wasn't reached [INAUDIBLE], then you would have visited F again.

PROFESSOR: And we would have visited F again. That's correct. The question is, F is listed twice. So if we hadn't reached the goal node, let's say that down there, after I is Z, and after Z is the goal node, then would we have visited F again? The answer is yes. As I said, this is the approach that throws your troops every possible way. So there's storm troopers going from E to F, and there's storm troopers going from H to F, and they're going everywhere.

Now there is a way to cut down on this. You could do breadth-first search with an extended list. If you did do breadth-first search with an extended list, that would be sort of equivalent to as soon as you expand one node, as soon you send storm troopers out of I to look at G and J, one of them stays at I, and if any other storm troopers come to I, and they're like, we want to see what's past I, he's like, no, no, no. We've already sent troops past I. We've got this. Go back home. It's OK. There are no rebels here, but it might be Han Solo dressed up as a storm trooper, as we'll see in the next problem. Where our extended list screws us over.

But for this problem, do you see what I mean? With an extended list, we could avoid this because the extended list basically says, once I've expanded and searched past here, don't do it again. But if you don't have one, yeah, you'll do F twice. In fact, didn't we do something twice here? Oh, it turns out we didn't. But we easily could have. We almost did E twice. We almost did F twice. It turns out we didn't, but we could have. Another question?

AUDIENCE: [INAUDIBLE] for the implementation of the queue that when you expand I, that both G and J were added simultaneously and in lexicographic order? Or is there a piece where it would be [INAUDIBLE] order, G got added before J, so hence, it stopped [INAUDIBLE]?

PROFESSOR: Ha. That is a very good question that provokes a minor change. The question is, when you expand SBHI, is SBHIG added before SBHIJ because of lexicographic ordering, or are they both added at the same time? The answer is, simultaneous. When you do the expansion, you instantly receive say, a list maybe, or whatever data structure you like, containing all of the children, and they get appended to the queue at once in order.

So you will receive all of them. I suppose you could create some kind of search problem where you have, like, an adversary node, like a wumpus, or you're hunting a wumpus. And if you don't want to add the exact wumpus to your search tree, or you lose. You only want to add the one next to the wumpus. I don't know. I'm making this up. And unfortunately, you would have to add all the children at once. Our current algorithm does not add one at a time.

All right. So we've done a breadth-first search. Great. Now we're going to do some optimal search. That is going to require me to draw a slightly different graph. I will draw it on what is left of the bottom here. And then we'll solve it on what is left of the bottom here. All right. Mark has his new stronghold, and he wants to invade parallel universes. Now he's programmed his evil supercomputer to find the shortest path of jumps from his starting universe to the goal universe, G. It takes a certain amount of energy to move from universe to universe based on the differing factors between those universes.

Like maybe in one universe, a butterfly didn't flap its wings in China, whereas in the other universe, sentient reptiles rule the earth. And so, Mark's supercomputer has tried to create a heuristic value that determines how different the universes are to guess, sort of, how much energy it's going to take to get from his start universe, S, to his goal universe, G, given that currently his armies are at a particular universe.

In other words, OK, it's not A you put in search, but dammit, we're going to have a heuristic distance to the goal. And we're going to have a graph with distances. So let's see what it looks like. All right. And down here is G. So this otherwise unassuming heart shape hides an evil invasion force. So let's see.

The distance here is 100. The distance here is 3, 4. The distance here is 4. The distance here is 50, 50, 14, 4, 16, 16. Oh sorry, these are connected. 30 and 10. The heuristic values are 0 at the start node, 50 at B, 60 at A, 55 at C, 50 at D, 56 at E, 50 at F, 0 at G because it's the goal node, 39 at H. I'll draw the little H smaller. And 0 at I. Well, after all, I goes right to G. It can be 0. It'll be great. OK.

Now, this time Mark is trying to conserve the energy of his parallel universe jump. That, understandably, takes a lot of energy. So he needs to program in the shortest number of universe jumps that will get him to the goal. He doesn't mean the least number of jumps. He means the least amount of energy on these edges. He's not interested in getting to this new world, and not having enough energy left to blast the crap out of it.

So we need to find the shortest path. So first Mark programs a simple branch and bound search. He adds in an extended list to his branch and bound just to make it a little bit faster. As usual, he breaks ties of equal length in lexicographic order. So let's list the nodes that Mark's computer adds to the extended list in order. Distances are shown next to the edges. Ignore the number in parentheses for now. They are heuristics, and we're not using them.

So that brings me to another point that I'd like to drive home before we go for the home stretch and solve these problems. What the heck is the difference between branch and bound and A*? OK. I like to liken it to the following. I probably should have, I'll make this a silver star idea. OK. The silver star idea is pizza. It doesn't really look like pizza, but that's the silver star idea. Branch and bound is a cheese pizza. It's simple. If you order it for a large group event of college students, they will eat it, and things will be OK.

Now A* search is some kind of meat lover's or supreme pizza. Maybe a meat lover's pizza. It's got all these extra toppings. A lot of people are really going to like it. It might be better. But then you've got a vegetarian, and everything's screwed up. So basically, A* is just branch and bound with some extra toppings added on. In this case, one of the toppings is an extended list, an extended list which we'll see keeps track of where we've passed through and already expanded out, and it never goes back.

The other topping that we're going to add is a heuristic. A heuristic tells us about how far we think we have left until we're at the end, so we don't go through really short paths that go completely in the wrong direction. All right? Between the two of those, we get our supreme pizza, but sometimes, as we'll see, they sort of mess each other up. So one of the toppings just doesn't go well with the other one. Maybe someone doesn't Hawaiian. They think that the ham doesn't go well with the pineapple.

So let's do a branch and bound that just has an extended list. Maybe we've got some green peppers on this pizza. This is going to be safe. All right. So we're going to list the nodes as to the extended list, and the way we're going to do that is, well, you guys said you like goal tree more than queue, so let's do it with a goal tree. So we've got S, and as we already know, S is the only path. Our current length at S is 0. That's the lowest of all of our lengths on the tree cause it's the only length.

So we go to A, B, and C. It may be hard to see, so the length of SA is 100. The length of SB is--

AUDIENCE: You mean ABEF?

AUDIENCE: Yeah. [CHATTER]

PROFESSOR: Oh. You're right. It's a different tree. I was doing A, B, and C from the other tree. Thank you, friends, for correcting my foolishness. The length of SB is 3. It is a different tree. I was trying to save work that wound up creating more because someone's going to be confused by this. So the length of A, B, E, and F are 100, 3, 14, and 14. Which one do we choose? Lexicographically, it's A. We choose that one, right?

AUDIENCE: [CHATTER]

PROFESSOR: F is going to be 14. That' a 1. Sorry.

AUDIENCE: 4. 4.

PROFESSOR: Oh, it's 4? I wrote this problem. I should know it. You're right. F is 4. My apologies once more. That's what I get. All right. I'm going to stand over here from now on when I write over there. OK. So which one of these are we going to choose? Even with the 4, we're going to choose B. That's right. Lexicographic be damned. It's only for tiebreaks. We want the shortest path. Our special thing with branch and bound is we take the currently shortest path, whatever it is. Great.

So we expand B. Fortunately, and I'm pretty sure I've got this correct this time-- oh wait, I've got an even better idea. I'll just take this little hand sheet with me, and then I don't have to look at that at all. OK. So once we expand B, B goes to D. Our path length from B to D is 4, so what will we write next to little G here? 7. That's right. We add them all together. I'm going to ask you guys again the next time we do this with the heuristics, which is going to be the next part of the problem, and someone's going to give me the wrong answer next time. So stay tuned for that.

All right. So we've got SBD. We've got all of these. Where do we go next? F. That's right, because currently 4 is the shortest because you guys corrected me correctly. So SF. 4 is the shortest. F only goes to H. And what do we write next to the H? 20. That's right. 16 plus 4, 20. All right. Where's our current shortest? It's the D, SBD. All right? So SBD. D only goes to I, and we've got a 57.

Wait a minute. I want to get this problem right, so we better actually write the extended list, [INAUDIBLE] because that's the only thing they're asking us for. So first we extended S. Then we extended B. Then we extended F. Then we extended D. All right. So far, so good. And I have the answer key, so I know we're doing it right. OK. What are we doing next? E. That's right. So we extend E. E goes to H. And when E goes to H, we've got a length of 30. All right.

Who's our winner now? Which H? SFH. That's right. SFH is length 20. Oh yeah, of course I should write an E in here. I always forget to do that. SFH is length 20, so we will extend H, and I'm going to write it here preemptively. When we extend H over here, H only goes to I, with length 50. Great. What's the next shortest? The next shortest is the other H. However, will we expand it? You guessed that because I asked that question in that way. You knew the answer was no. Why don't we expand it? It's already on the extended list. That's right.

So since it's already on the extended list, this one dies a horrible death. I like writing an X through it instead of writing a swizzle at the bottom. You can do whatever you want. All right, it's gone. It's not a choice. What's the next best one? SFHI. That's right. That goes to G, and the length is 60. And we've extended I. All right. Question is, are we done?

The people who say no, I like you. You're smart. You realize that just because the G is on there, that we can't end. However, the people who said yes, you are either oblivious or really, really smart, and I'm going to choose to assume you're all really, really smart because the really, really smart people said, OK. Yes, we're not quite done just because we added a G. We still have to check to make sure there are no lengths with shorter path, but look. The only one with a shorter path is I, which is already on the extended list. So actually, we are done. Double check. We've got it.

So these are the paths we extended in order, and our final path is? Everyone.

AUDIENCE: [CHATTER]

PROFESSOR: That's right. SFHIG. I claim that is the correct path. However, Mark is frustrated by branch and bound's speed. I don't know. I wasn't that frustrated. Seemed pretty good. But Mark is frustrated by branch and bound's speed, so he reprograms his computer to use A*. Mark counts the number of subspace anomalies between each universe and the goal, and uses this count as the heuristic for A*. These numbers are in parentheses. Hopefully, you can read them. Yes? Oh, we've got a question. Right here?

AUDIENCE: So, given the implementation, you said that you expand all possible nodes. So why doesn't I go to C and D, as well as [INAUDIBLE]? Like, why doesn't it expand to [INAUDIBLE]?

PROFESSOR: Ah. That is a very good question. And the answer is, a very simple answer. You guys tricked me. No, but I should have been able to figure it out. It does go to C and D. The correct tree, which we wouldn't have lost points for having the incorrect tree here cause we did get the correct answer, yes. The reason why it is, that same reason that very first time I asked someone something, he didn't remember the C. It's easy to forget to go up the tree. It does actually have a C and D. However, they are horrendous paths. They are 100 and 100 on their path length, and so it doesn't matter. But you were correct. The official answer we had up there is wrong.

AUDIENCE: Would the D get added to this list of children that's already in the extended list?

PROFESSOR: Good question. The question is, would the D get added to the children? After all, it's already in the extended list. The answer is, we search for, and remove, and kill all of the attempts to extend something that's already on the expanded list at the time we try to expand it. The time we try to expand it is only when it's on the front of the queue because it's the currently shortest path. So that means they get added. It's just that when it comes time to expand it, it will get crossed off no matter what. Turns out, it escaped execution because of the fact that we never expanded it.

AUDIENCE: So H should go to E as well?

PROFESSOR: So H should go to E as well, is the question. The answer is yes. H should go to E as well. A lot slipped past me this time. H should go to E as well, with a length of 36, and it dies there. This one will actually be checked, so it actually does make a difference. If we ask how many times was a node executed due to already being on the extended list? Very good. Very good notice. It should be on there. We'll get it right next time.

All right, everyone. So I'm working together with you. I made the mistake too. Easy one to make. It can mess you up. It didn't this time. We're going to get it-- question?

AUDIENCE: If G's the goal at the end, and you get G in your outcomes, and you know the number that's shorter than everything else, do you have to actually extend at G?

PROFESSOR: So, the question is, do you actually extend G? Should we even put G in the extended list? The answer is, the answer to that question is, it is a matter of taste. And in the questions where we,

in my opinion, rather foolishly, asked how many nodes were extended, rather than ask you to write them out, we generally accept the answer where you didn't extend or where you did extend G. It's an implementation detail. You can either have a fail safe that, as soon as it sees G at the beginning of the list, says, we're out. We're not going to extend. We're done. We win. Or a fail safe that, when you're about to extend something, and you go into the extension process, and it sees that it ends in G, that it wins.

So it is a matter of taste. If you kind of like to watch your little guy doing the search win, and have an S and a G, you can put it on. If you don't, you can not put it on. We won't take off points for whether or not it has a G at the end because clearly, if you did all the other crazy stuff correct, well, you could have written in a G if you wanted to. I think, unless there's someone who mis-solves the problem right at the end, is like, oh, who cares about G? Oh no, it's stuck. It's a dead end. We lose. But that would be probably pretty rare.

So let's solve the A*. So I'm not going to make the same mistake twice. We go from S to ABEF. And S, by the way, had a value of 0. All right. What is the value at A? Well, I think it's 160. The question is, how do we calculate this? Well, it's the path that we've traveled so far plus the heuristic value at the final node. This is why someone's going to give me the wrong answer at BD, but let's see.

So we have 160 here. OK. So SB. What is the heuristic value here? We've got 3 for the path, and 50 for the heuristic, so it's 53. SE. What have we got here? We've got 14 for the path, 56 for the heuristic, so it's 70. All right. SF. We've got 4 for the path, 50 for the heuristic, 54. OK. Who's our winner? It's B again. Barely, but it is the winner. Extended list up here. SB. All right. B, as we saw, goes only to D.

What is the value that I should write here?

AUDIENCE: [CHATTER]

PROFESSOR: OK. I am happy I heard all the things that I expected to hear. I heard the correct answer, which is 57. I also heard someone say 107. So why is it 57, and not 107? Someone does it this way every time. Do not add up all the heuristics along the way. I will try to explain to you why you would never want to do that. The heuristic value at any given node says, given that I'm here, how much work do I think I have left to get to the end?

All right? It's sort of like, let's guess the last few nodes in the path that we haven't done out yet.

So you can see why it would be bad to add that for every node in your list because then you're double counting all of the last nodes. So add the path so far to the very final heuristic. That's 3 plus 4 plus the 50 that's with D is 57. So our current winner, then, is 54, with F. Just the same as last time. F goes to H. And what's our total value at H? 20 plus, that's a 39, 59. So who wins now? D, with 57.

But we extended F, yes. All right. So D was 57. All right. So D was not one of the ones that I tricked myself with. D only goes to I. What's our value at I? I think our value is 57 because I has a heuristic of 0. Yes. I has a heuristic of 0. Our value at I is 57. OK? And I extended D. Who's the winner now? I? OK. So I is the winner. With I as the winner, we extend I. I goes to C, D. Ha ha. Good call, whoever back there figured out my secret error before. C, D, and G.

AUDIENCE: [CHATTER]

PROFESSOR: C, H, and G? Oh, you're right. We already went to D. Aha. C, H, and G. You're correct. Everyone see that? C, H, and G. Absolutely right. I goes to C, H, and G. So the path to C, H, and G is a hard path. To C, we got, what did we have? 57 plus 50 is 127. Oh, and by the way, C, G, and H. We've got to do it in lexicographic order. So 127 to C. To G, we have 67. And to H, we have 87.

So who wins now? H. H, with 59. That's right. H, with 59. And as we already know, cause it was the winner last time, H, with 59, goes to I, which has, I believe, 50. Well, it's still gets added. Remember? We're only going to kill it when it gets extended. So who's the shortest? So you were a step ahead, whoever asked me that question. Who's the shortest? H also goes to E. Absolutely right. H also goes to E. I was getting ahead of myself here. H also goes to E with a length of extra 16. So we've got 20 plus 16 is 36 plus 56 is 92. All right. That's good.

So who's the shortest? I is the shortest. Do we extend it? No. It's on the extended list. All right. Who's the shortest now? It's hard to read, but the shortest is this 67 on the G. So we're done. We win. Our path is SBDIG. Yeah, for A*. It gets the right answer, right? No. Unfortunately not. So what happened here? Why did we not get the right answer? Be as specific as possible. What are people saying about heuristics?

AUDIENCE: [CHATTER]

PROFESSOR: All right. People are saying the heuristic must be consistent. That is both correct and specific, so I applaud you. Too easy to say the heuristic was inadmissible. It actually was completely

admissible everywhere. So that leads us into our very last point for the day. What the hell are these heuristic consistency and admissibility things, and why do I care? Well, the reason why you care is many fold, one of which is that it's almost guaranteed to be on the quiz. But what are they?

Admissibility is a check at every point to make sure your heuristic at that point, which is supposed to be an estimate of how much work you have left to do, is always an underestimate or an accurate estimate. It can never be an overestimate. Why not? Well, as Patrick showed you with his mostly correct example in lecture, if it's an overestimate, it's never going to expand that node cause it's going to think, you know, if you write one million as a heuristic, it's going to think it needs to do one million after it's done going that way. It's not going to want to go that way. All right?

So it's always got to be an underestimate or an equal estimate. But let's say that you, at the quiz, and you forget Mark told me this. You didn't bring any notes. You're just having a brain freeze. You're like, oh no. Is it supposed to be an overestimate or an underestimate? I can't remember which one. How can I figure it out? How can I figure it out? I propose you the following calm, soothing mantra slash sutra that will help protect you.

Think to yourself of the following question. We know that A* sometimes messes up with the heuristic. Does branch and bound always get it right? Yes. What heuristic values does branch and bound add everywhere? 0. Right? It has no heuristic. It essentially adds 0. Is 0 an overestimate or an underestimate? An underestimate. Therefore, since branch and bound always works, the one you're looking for is an underestimate.

And I know from when I was taking the class, I always had a moment where I had to spend, like, two minutes convincing myself, OK, I'm going for underestimate, not overestimate. This will help. It will take fewer than two minutes to do that.

So what is consistency? Consistency is a little bit stronger of a claim. When you claim that a graph is consistent, what you're saying is, between any two nodes, or to do it more simply, between any two adjacent nodes, the distance between the heuristics is less than the distance between the nodes. In other words, admissibility is a sort of, like, consistency between every node in G. Whereas consistency is consistency between every node and every other node. All right? Consistency is a stronger claim. Any graph that's consistent is always admissible. Any graph that is inadmissible is always inconsistent. That's the contrapositive. Question?

AUDIENCE: Less than or less than or equal to?

PROFESSOR: Equal to is still OK. Equal to is always OK. That's great. It's a perfect estimate. But never greater. Now, if a graph is inconsistent, will you lose? Will you lose? Why is it called admissible, then, if it's sometimes not admissible? Why do we lose? The answer is the extended list, and you see that here. If a graph is admissible, you will always get the right answer unless you use an extended list because you're checking from every node to the goal node. And you're sure that your estimates are right. But if estimates within nodes aren't correct, you might go through them out of order, and that violates your assumption that you made when you decided to use the extended list that you would always go through the sub-graphs in order.

This graph is very expertly crafted to do that to you cause I might as well be sort of a bottleneck, goal node, and I has an inconsistency. There's a few other inconsistencies in this graph, but they don't do anything to you. In fact, even inadmissibilities sometimes don't mess you up. So you can't just say, blindly, it's inadmissible, so it will never work. It might work.

It turns out the only inconsistency that matters here is not the inconsistency between, say, S and F. That doesn't turn out to matter. It's the inconsistency between some of these nodes, including I and H, specifically.

So have a good weekend. Come in on Monday and Tuesday to tutorial. Ask about the queue method, or other methods. Ask about anything that's niggling in your brain. But hopefully, you guys have got this. You're going to do fine.