PROFESSOR: I want to talk about two things, maybe, depending on the level of interest. We can talk a little bit about PSET five and the things that I think are weird in the coding part of PSET five. By the way, how many people started PSET five?

[LAUGHTER]

AUDIENCE: I downloaded it.

PROFESSOR: Hmm, maybe we are not going to talk about P set five. You guys need to start early!

AUDIENCE: I did.

PROFESSOR: You did. Yeah. I guess I can't punish you for everyone else. So P set five, and I can talk about graphs. I can talk about whatever didn't make sense in class and I can talk about background stuff for graphs.

So how do people feel? Who wants talk about the PSET that we haven't read?

[LAUGHTER]

Who wants to talk about graphs?

AUDIENCE: It's really easy. We took 604 two.

PROFESSOR: Yeah, if you took 604 two, nothing here is new. I mean nothing in the recitation is new. We're going to take it to new heights of graph knowledge, and you'll be able to do many more cool things that you're able to do before.

OK, so PSET. For people who started early, what was the gist of the coding assignments?

AUDIENCE: So we have to speed up something. When I ran the tests-- do I say what I got as results?

PROFESSOR: The time isn't on the PSET so you can say what you got as --

AUDIENCE: Pardon?

PROFESSOR: The time isn't on the PSET so you can say what you got as a time.

AUDIENCE: OK, the slowest was like add for me.

PROFESSOR: You should look to see add is a valid answer. Look at the questions and see if add is a valid answer.

AUDIENCE: OK.

PROFESSOR: It's not like what was lowest function, right? There's more text there. You should read the rest of the text and see if add makes sense as a valid answer.

AUDIENCE: Sure. OK.

PROFESSOR: What you're doing-- big picture-- is you have some processor, that's not a real processor, and it can do arithmetic with bytes and 16-bit words. And we give you the basic operations and then they give you a library for large number arithmetic because, guess what? 16 bytes addition, subtraction? Not going to cut it for science applications are for cryptography, what we want to play with.

What are the basic operations that that processor can do? Front of the PSET, so even if you didn't get to the coding assignment you can still tell me.

AUDIENCE: Divide, and zeros, I think. Zeros.

PROFESSOR: Plus, minus, multiply, divide, and mod. Let's start to these.

So you have two primitives in that processor. You have have bytes, which are basically 8-bit digits. There's 200. The range is 0 to 256. And the word is 16-bit, from 0 to the 16 minus 1. If you care to know this is 6, 5, 5, 3, 5. So, if you take two

bytes and add them together what do you get?

AUDIENCE: [INAUDIBLE].

PROFESSOR: Not right. It's a trick question. You get word. The processor cannot do simple math on bytes. It converts them up to words and all the math happens on words.

Now if you guess two words, and add them up together what do you get?

AUDIENCE: Two words?

PROFESSOR: You do if we're not that nice.

AUDIENCE: A world in carry bit?

PROFESSOR: Sorry, you just get word. At least nobody said a byte. I was like, please don't say byte, please don't say byte. You said carry bit. Why do we care about the carry bit? What is the problem if you do addition this way?

AUDIENCE: If you're going too high. If the highest bit is one on both of them. Then it's like overflow, kind of.

PROFESSOR: Yep. Suppose we're trying to add 2 to the 16 minus 1 plus 2 to the 16 minus 1. By the way, does anyone remember hex notation? Hexadecimal? OK, people who started the PSET remember. That's good. It will be easy to write things hex for the PSET.

If you try to add these numbers you're going to get 1, F, F, F, E. You can use the math here to see that this is more than the words. So you would like to know that this thing overflows, right? You like to know that there is. Well we don't give you that. All you get to this. So addition happens, modulo 2 to the 16. If you want to be able to detect overflow what they have to do?

AUDIENCE: Just tests those bits.

PROFESSOR: That one way of doing it. It would take quite a few instructions, though.

If you want to do overflow detection, the easiest way I think of doing it this to use this form. So suppose you have two bytes. This is the maximum value in a byte, right? 255, 2 to the 8 minus 1. If you have 2 bytes and you add them all together you're going to get 1, F, E. Right? It's just like the case here, except you have a fewer F's. Where is the carry here?

AUDIENCE: Could be the 1, right?

PROFESSOR: Yep. This guy here is the carry and this guy's the low result.

Does anyone know how these are called? If you have a word and you have two bytes, what is the first byte, what is the second bytes?

AUDIENCE: [INAUDIBLE].

PROFESSOR: Most significant byte, M, S, B, and least significant byte, L, S, B. If you want to figure out your carries, then you should do your math this way. You're going to have byte 1 plus byte 2. Add them together. And then you call L, S, B to get the byte result. And then you call M, S, B to get the carry. This is addition. Everyone with me so far? By the way, are these numbers signed or unsigned?

AUDIENCE: [INAUDIBLE].

PROFESSOR: No negative numbers, right? So no support for negative numbers. Everything is going to be positive. And we're going to build everything we need off of positive numbers because they're easier to deal with.

What if I have two words and I subtract them? What do I get?

AUDIENCE: Word.

PROFESSOR: OK. What if I have 0 minus 1? What do I get?

AUDIENCE: 0.

PROFESSOR: Not quite.

AUDIENCE: If you're not using signs? Do you get O, 1?

PROFESSOR: O, 1's, OK.

AUDIENCE: Ya. Wait--

PROFESSOR: So, O 1's would be this, F, F, F, F.

AUDIENCE: That's assuming if you signed then. It will overflow. That's a really big number then.

PROFESSOR: So fancy CS, 2's complement. For people who think in math mode, what is this?

AUDIENCE: I think.

PROFESSOR: It's minus 1 mod 2 to the 16.

Remember, when we were doing modular arithmetic before, to figure out a number's multiplicative inverse? And when we were doing rolling hashes? We didn't want negative numbers. We made them positive. So minus 1 lot to the 16 is 2 to the 16. Minus 1, and comes out to this value. So you get minus 1, it's just that you have to know your basis.

So we got the addition and subtraction. Let's talk about multiplication. What do you do for multiplication? What can you multiply? Two bytes, very good. Just two bytes. What do you get? A word. Thank you. You multiply two bytes, you get a word. What if there's overflow? It's a trick question.

AUDIENCE: There's no overflow.

PROFESSOR: There is no overflow. So two bytes, 2 to the eighth minus 1 times 2 to the eighth minus 1, is 2 to the 16 minus 2 to the 8th minus 2 to the eighth plus 1. You can say this safely without doing the arithmetic, right? This is how much you can hold, in other words. This is how much you can holding in a byte.

AUDIENCE: Why do you minus 1?

PROFESSOR: Ya, minus 1. There's a minus 1 here, and there's a minus 1 here, but I can't find it

here, so roughly this. But there is an overflow.

AUDIENCE: But where did you get the minus 1 from?

PROFESSOR: Where do you get the-- so, this is how much you can hold in a byte. Byte has 8 bits, right? From 0 0 to F F. This is 2 to the eighth minus 1.

AUDIENCE: OK.

PROFESSOR: OK, so multiplication? There is an overflow. I don't have to deal with that. You guys have to figure out how to deal with it.

How about division? What can you divide?

AUDIENCE: Some words?

PROFESSOR: Almost.

AUDIENCE: Words like--

PROFESSOR: A word divided by a byte. What do you get out of it?

AUDIENCE: Words?

PROFESSOR: That would be nice. Nope. Sorry, you get a byte. And if you do module, you also get a byte.

What if there's overflow in the division? What happens? What would you expect to happen?

AUDIENCE: We don't have any best flowing point numbers, so, we shouldn't get overflow, right?

PROFESSOR: How about 2 to the 16 minus 1 divided by 1? So what do get?

AUDIENCE: Same thing?

PROFESSOR: Will this fit in the byte?

AUDIENCE: Yes? No.

[LAUGHTER]

No. Well, that time I don't get a word.

PROFESSOR: You don't get a word. I promise you get a byte. What will that byte be?

AUDIENCE: It's going to be modulo something. Most significant--

PROFESSOR: That's reasonable, right? Modulo to 56 because that's what it can carry.

What about modulo? What happens there if you get overflow?

AUDIENCE: The same thing. It's just going to loop, right? 0?

PROFESSOR: So if you do a modulo, this is going to be, at most, 255 right? If you do modulo 255, the remainder is going to be between 0 and 254. Will that ever overflow byte? No overflow. No need to worry about it. Word addition can overflow. Subtraction can overflow. Multiplication doesn't overflow. Division overflows, modulo doesn't.

AUDIENCE: What do you mean, like will it see the space confined?

PROFESSOR: Yes. It sees the space of the result. Because here, if you're adding two words, the result exceeds the space of a word. Which is what you get.

AUDIENCE: Is it just in all cases it's just [INAUDIBLE].

PROFESSOR: Exactly. So the reason you have to deal with this weird system is this is, pretty much, how Intel does their arithmetic. If you look at old school, 16-bit Intel processors, you have registers and you have exactly these operations. For newer processors, there are 32 bits, but then it's just, you write more F's on the board and you get the same thing.

AUDIENCE: Is it most significant bit, or most significant byte?

PROFESSOR: Most significant byte. On a real processor, you have registers that are the size of a word and then you can pull out the bytes in constant time. What constant do we have? These are the operations. What constants do we have? Only two constants,

0 and 1. What if you want to get something bigger? What if you want to get 2? How do you get 2?

AUDIENCE: Two 1's? 1 plus 1?

PROFESSOR: Yep. What if you want to get this number?

AUDIENCE: Do that several times?

[LAUGHTER]

PROFESSOR: It's kind of painful to type, even if you copy paste. There's a method on Word called from-bytes. And it takes an M, S, B and an L, S, B and it gives you a word. So what would I give it?

AUDIENCE: [INAUDIBLE].

PROFESSOR: Let's give it a 1 and let's give it a 0. So this will get 1, 0, 0 and then minus 1. Yes? OK.

Intel is pretty nice about constants, but some other processors aren't. So I figured, why not let's get you acquainted to these kind of tricks, to too. Let's make the PSETS more interesting. All right, so any questions on this fake processor that you have to code for?

AUDIENCE: Can you explain the last one again?

PROFESSOR: Here?

AUDIENCE: Yeah, Word from-bytes.

PROFESSOR: So in Word from-bytes a word is 2 bytes. One next to the other. It gives you the first byte, and gives you the second byte. In this case, the first byte is 1. The second byte is 0. Right? So 2 bytes. A 1 and a 0. How do you write this in hex? 1, 0, 0. One byte, second byte. And then I subtract 1 and I get this.

While I erase the board, I want you guys to think of graph questions because this is

the other topic we're looking at today. What was unclear about the lecture? What is unclear about graphs in general? Do guys remember the handshaking lemma? What does it mean? How do you prove it? Things of that sort. What do you guys want to cover today?

AUDIENCE: Handshaking, that's the thing where you have a bunch nodes and they're all connected, like they're all in a closed graph, that the number of edges is equal to twice the number of vertices?

PROFESSOR: OK.

AUDIENCE: That's all it is, right?

PROFESSOR: The number of vertices? No.

AUDIENCE: The number of handshakes that occurred are twice the number of edges. You said the number of vertices, wait--

PROFESSOR: No, vertices and edges are not related in here.

AUDIENCE: Yeah, think of a triangle, right? That's three edges, three vertices. Which is like not two times, which is what you said. It's the number of degrees is twice the number of edges. That's what I was thinking of then.

PROFESSOR: All right, so let's start with simple stuff. What's a graph?

AUDIENCE: Nodes and edges?

PROFESSOR: All right. Fancy word for nodes?

AUDIENCE: Vertex.

PROFESSOR: Vertex. Vertices and edges. How do I draw vertices? How do I draw edges?

AUDIENCE: Circles and lines.

PROFESSOR: Circles and lines.

AUDIENCE: 0's and 1's. Possibly arrows.

PROFESSOR: Possibly arrows. I like that. You want to get fancy.

[LAUGHTER}

PROFESSOR: When do I draw an edge as a line? When do I draw it as an arrow?

AUDIENCE: Directed under.

PROFESSOR: Which one's which?

AUDIENCE: Directed as an arrow. Character number.

PROFESSOR: Cool. Let's draw this graph here. Yeah, looks like a pretty boring graph. Is this graph connected or not?

AUDIENCE: Yes.

PROFESSOR: What's a connected graph?

AUDIENCE: I think that we can get to any other node following some path.

PROFESSOR: There's a path from any to any other node. How do I make this unconnected using the least amount of effort? It's a bit of a trick question. Any guess is fine.

AUDIENCE: What was the question again?

PROFESSOR: How do I make this unconnected using the least amount of effort on my behalf?

AUDIENCE: Just add a node?

PROFESSOR: I heard you move pages. I don't like you racing because I don't like the-- So I like that answer out of the two I heard. Now it's not a connected graph. How many connected component does it have? Two and five.

AUDIENCE: Or one connected component, but there are two components?

PROFESSOR: OK, what's a connected component?

AUDIENCE: It's like in that part of the graph the neck follows the prognito beacon and puts them in two parts [INAUDIBLE] two connected components--

PROFESSOR: So, a connected component is a bunch of vertices such that you can get from one vertex to all the other vertices. If the graph is undirected, that's true for any vertex in the set. We also want the sets to be maximal because, if we say this is a connected component, it's not very useful. Noticing that this whole thing is a connected component is useful. So this is one component. This is the other component. Make sense?

So, we have the world. We have cities. You can bike from a city to another city and that's it. No other a route of transportation. How many connected components?

AUDIENCE: Seven?

PROFESSOR: OK. Roughly seven. Why? OK, so it only seven?

AUDIENCE: Seven continents?

PROFESSOR: OK. Let's say roughly seven. So this is what I wanted. I wanted some thinking. If you try to get from a continent to another continent, presumably you'd go through some patch of sea. Otherwise, why are they calling them continents? Does anyone want to give out another answer? I asked this in the previous section and some people there give me some very precise answers that are not continents. This is what I had in mind, by the way. As far as I'm concerned, this is a good answer. Come on guys, world. Two things, islands and Europe and Asia are connected, so you can go from one to the other. They're weird. Why are they separate continents? I have no idea. The geography people might--

AUDIENCE: [INAUDIBLE].

PROFESSOR: Sorry?

AUDIENCE: You roll mountains?

PROFESSOR: OK, so it would be a pain bike through them, but presumably you can. If you have a robot that doesn't get tired or something you can bike.

AUDIENCE: It doesn't mind falling off cliffs.

PROFESSOR: The answer, depending on my geography, we know is somewhere between 7 and 10,000, or whatever the number of islands is. There are a lot of tiny island somewhere, right? Anyways, between seven and a big number. These are connected components in the world. What's the degree of A? 2. What's the degree of the D?

AUDIENCE: [INAUDIBLE].

PROFESSOR: Very good. Let's make this. The degrees of F, G, H are 2, 2, 2. The degree of C is 2. The degree of B is?

AUDIENCE: --2.

PROFESSOR: Thank you. And the degree of E is?

AUDIENCE: 2.

PROFESSOR: If you add up all these numbers together what you get? 18. I can't do math, so I couldn't possibly have added all these numbers together, right? I used something else. How many edges do I have in the graph? 1, 2, 3, 4, 5, 6, 7, 8, 9. Do you see your connection here?

AUDIENCE: Yeah.

PROFESSOR: This is the handshaking lemma. That's all there is to it. So if you look at the degrees of a node each edge adds one to two degrees. For instance, this edge adds one to C's degree and adds one to D's degree.

If you're a math person and you write this up, you have to write sums. You have big sums using intimidating notation, so it's not as obvious. But this is really all there is to it. Each edge contributes one to two nodes degrees. If you add up all the

degrees, you're going to get to times the number of edges. So far so good?

What if we have directed graphs? What if I had this? A, B, C, D, E, F. What is the degree of A?

AUDIENCE: 2?

PROFESSOR: Not quite. Sorry, that was trick question. A doesn't have a degree. If you have directed graphs, you don't have degrees. You have in degrees and out degrees. Now that I've said that you have no idea, right? What's the out degree of A?

AUDIENCE: [INAUDIBLE].

PROFESSOR: Sometimes this is known as the degree of edges. This is 2. What's the in degree of A?

AUDIENCE: [INAUDIBLE].

PROFESSOR: So A has two edges going out, 0 edges going in. How about D? What's the out degree of D?

AUDIENCE: [INAUDIBLE].

PROFESSOR: Come on guys. Don't scare me. (LAUGHS) What's the in degree of D?

AUDIENCE: 2?

PROFESSOR: 2, very good.

PROFESSOR: So what's the equivalent of the handshaking lemma on oriented graphs?

AUDIENCE: The sum of the in degrees and out degrees?

PROFESSOR: OK, what about them?

AUDIENCE: They're equal to twice the number of vertices?

PROFESSOR: Not quite. They're equal to? You had 80 percent of the answer.

AUDIENCE: Oh. (LAUGHS)

PROFESSOR: So the sum of the in degrees and the sum of the out degrees.

AUDIENCE: So add them up?

PROFESSOR: If you add them up, you will get two times the number of edges. That is correct. But, I want something more--

AUDIENCE: They equal each other?

PROFESSOR: Yep. This is cooler, right? So why is that? Does anyone see why that's the case? Each edge-- Come on, guys. So what does each edge do?

AUDIENCE: [INAUDIBLE].

PROFESSOR: So if I look at this edge here, this edge is going to contribute 1 to B's out degree, and 1 to D's in degree. Each edge contributes 1 to an out degree and 1 to an in degree. That means that total sum of out degrees equals the total sum of in degrees. Both of them are E and they combined to E. Sorry, eyed I don't know why that didn't click to me right away.

AUDIENCE: OK.

PROFESSOR: The intuition behind this is, that for every node, if you take an edge you're going to get somewhere else. If the sum of the out degrees was bigger, then you have a black hole somewhere. If you go on an edge you don't come back. Same four in degrees.

OK, how do we represent graphs?

AUDIENCE: I think we just did.

PROFESSOR: Sure, if you're drawing them on the board that's what you do, but if you're in Python what do you do?

AUDIENCE: You can have a link list.

PROFESSOR: Of?

AUDIENCE: Of each node having its neighbors linked with.

PROFESSOR: OK, so I have one big link list? Or how does this work?

AUDIENCE: You'd have a starting point of some sort. A starting node. Then from that node you can have its neighbors connected to it. So it was a link list , I guess.

PROFESSOR: I wasn't precise. That is not trivial to build. By the time we build that we're done with this recitation.

What do you get is the vertices and the edges. We want an easier representation that just looks at the vertices and build something, then looks at the edges and builds something.

AUDIENCE: You could have a table of values. Like, A has these neighbors-- a dictionary.

PROFESSOR: Let's go for that.

So we have a dictionary. For each vertex you have the list of vertices that are connected to it. What's the list for A?

AUDIENCE: B and C?

PROFESSOR: OK, what's the list for B?

AUDIENCE: A, D, and E.

PROFESSOR: All right. For Python this would be a dictionary, right? So how much total space does this take?

AUDIENCE: The number of nodes?

PROFESSOR: The number of--

AUDIENCE: --nodes. Well, then there's also the space you made for the list, though.

PROFESSOR: If these were actual slots in an array-- so this would be an array-- I would have the number of nodes. You're giving away the answer to my next question. So I have these slots here. This would be an array. It's order V just to store this.

The thing in Python is that have these dictionaries that are fancy hashes where they grow as you need them. For example, you have 10,000 vertices but you don't have any edge. Your dictionary size is going to be order 1 because it only grows as you add edges to it. So this is assuming that I don't store empty lists. If I have a stray node here, if I have a node I-- say this is I-- if I don't store anything, I don't have to pay for it. If I store an empty list here, then I have to pay for it. There is an order V component that you mentioned.

AUDIENCE: Let's say, if there's a graph there, everything is not connected so there are a bunch of words instead. Then in that case it would be order V, right?

PROFESSOR: It's order V if we store empty lists for the nodes that don't have edges.

AUDIENCE: Right. And if none of the nodes have edges, they're all unconnected--

PROFESSOR: On the other hand, if I don't store anything for the nodes that don't have edges, it's order 1.

AUDIENCE: If you get no edges do we do an empty list or do we not store it?

PROFESSOR: Depends on what you want. So if we store empty lists you're going to have an order V cost here. But your code is going to be simpler, presumably, because you don't have to check if something is or isn't in the dictionary.

How about this stuff? What's the total size of this stuff here?

AUDIENCE: [INAUDIBLE].

PROFESSOR: OK. Order E. How many elements do I have in here in total? In this thing here. So what's the sum of the length of the lists?

AUDIENCE: Average value of E times the number of vertices?

PROFESSOR: Let's go over something else.

AUDIENCE: The degree--

PROFESSOR: All right. That's what I wanted to hear. A lists its neighbors, right? The number of neighbors that A has is the degree of A. B lists its neighbors, so this is the degree of B. If you sum up over all of them, what is the sum of all the agrees in the graph?

AUDIENCE: 2, E.

PROFESSOR: 2, E. We learned about this, right? The handshaking lemma, 2, E. So what is the total cost for storing this data structure?

AUDIENCE: [INAUDIBLE].

PROFESSOR: So V plus E, assuming empty lists.

Let's look at another data structure for storing things in a graph. So instead of using lists, let's use a matrix. A, B, C, D, E on top, and A, B, C, D, E on the left. Let's pretend our graph is just that component over there. Otherwise, it just gets big and there's no extraneous site.

Does anyone know how this is called? If I put numbers here, does anyone know what this is called?

AUDIENCE: Is that a matrix?

PROFESSOR: There's a fancy name for it. It ends with matrix. OK. I don't think we taught it to you, so don't worry. The fancy name is maybe it's misspelled, but something that looks and sounds like this., adjacency matrix. Misspelled?

AUDIENCE: [INAUDIBLE].

PROFESSOR: I hope someone will help me. OK so if this is an adjacency matrix, what should this element between B and A tell me? If it's an adjacency matrix, it better tell me if they're adjacent. They're adjacent if they have an edge between them. For A and B, what does it happen to be?

AUDIENCE: One?

PROFESSOR: OK. A and A? What do we do?

AUDIENCE: What with 1? It's adjacent to itself.

PROFESSOR: What's easier for the algorithm that you're trying to implement?

AUDIENCE: 1?

PROFESSOR: So it doesn't really matter. We use 1's most of the time, but sometimes it's easier to use a 0.

AUDIENCE: Then you can get to A from A, right?

PROFESSOR: Yeah. So that's why you'd use a 1. Sometimes, though, you don't want to in code. So someone dictate this to me. Or, everyone dictate this to me so I know it makes sense. A and C, is there an edge between them?

AUDIENCE: Yes.

PROFESSOR: What do I write?

AUDIENCE: 1.

PROFESSOR: A and D? A and E?

AUDIENCE: 0.

PROFESSOR: B, A?

AUDIENCE: 1.

PROFESSOR: B, C?

AUDIENCE: 0.

PROFESSOR: B, D?

AUDIENCE: 1.

PROFESSOR: The E?

AUDIENCE: 1.

PROFESSOR: OK. C, A?

AUDIENCE: 1.

PROFESSOR: C, B?

AUDIENCE: 2.

PROFESSOR: C, D?

AUDIENCE: 1.

PROFESSOR: C, E?

AUDIENCE: 0.

PROFESSOR: All right. Now I'm going to use the bits you gave me to come to get this. Let's see if I can do it correctly.

And D? I'm not looking at the graph, by the way, I'm trusting you. So you better give me the right answer. D?

AUDIENCE: 1.

PROFESSOR: How many 1's do I have in this?

AUDIENCE: [INAUDIBLE].

PROFESSOR: To see if you guys are thinking.

AUDIENCE: 2, E? [INAUDIBLE].

PROFESSOR: Each edge contributes to two 1's, right? And that accounts for most of the 1's.

AUDIENCE: O plus V.

PROFESSOR: So this is how many 1's I have. How many 0's do I have?

AUDIENCE: V squared minus 2, E plus V?

PROFESSOR: This is a non-boring way of asking how many elements I have in this matrix. This is what I was looking for.

So V columns, right? Zeros V squared's total elements. How much memory do I need to store this?

AUDIENCE: V squared.

PROFESSOR: V squared. what? If I want to store it as compactly as possible? I want to pack these as tight as I can.

AUDIENCE: You mean you need the entire array arranged?

PROFESSOR: Yes, so what do I store? V squared what?

AUDIENCE: Oh, you mean units.

PROFESSOR: Yeah. What's the unit?

AUDIENCE: 1 bit?

PROFESSOR: All right! So V squared bits Whereas, this is V plus E words because you have to store pointers everywhere here. So sometimes if your graph is really dense you might prefer this representation. Let me see, how much do I have? Oh, plenty of time.

Who remembers breadth-first search? Yes?

AUDIENCE: Basically you just start at some node and you check all its neighbors, and check all those neighbors.

PROFESSOR: All right.

In breadth-first search we have a graph. What did I draw there? A. B, E. So suppose this is our graph. And I do a breadth-first search starting at A. How does that work?

I started with the list of the nodes that I'm going to visit. I initialization it with A because this is the only node that I know about. What happens next?

AUDIENCE: It goes with B and C.

PROFESSOR: All right. So I take out of the list the first thing that I can. This is my current node. You said I visit B and C because they're the neighbors. Right. I took A out of the list. A was definitely visited. And I'm visiting B and C. What do I do when I visit them?

AUDIENCE: Put them in the list.

PROFESSOR: Put them in the list. This means I discovered them and I'm going to visit the later. So I discovered them and I know of their existence. What happens next?

AUDIENCE: Check if B is what you want.

PROFESSOR: So B gets out of the list, and what do I do?

AUDIENCE: Discover it's neighbors.

PROFESSOR: All right, so its neighborhoods are A, B, and E. What Now?

AUDIENCE: If they haven't already been on that list then add a [INAUDIBLE].

PROFESSOR: A was already visited. D and E weren't. D and E, then what happens?

AUDIENCE: Then you check C's neighbors.

PROFESSOR: Any neighbor's that I haven't seen? Nope. Then?

AUDIENCE: D?

AUDIENCE: D and?

AUDIENCE: E.

PROFESSOR: OK. And then?

PROFESSOR: I guess you could go to F, but it's not connected to anything. But how did you get A then, right?

PROFESSOR: So A is the first node. I started with A because I said I'm doing a breadth-first search starting from A. So BFS starts from somewhere. A BFS has a source. We'll see why that matters in a bit.

So you ever get to see F?

AUDIENCE: No.

PROFESSOR: Nope. So if I start at A what are the nodes that I visit and what are the nodes that I don't visit?

AUDIENCE: Well you visit all the ones in the connected graph.

PROFESSOR: All right, in the connected component. So the whole graph can be connected or not connected.

The nodes that I visit are the nodes in A's connected component because, by definition, those are the nodes that I can reach from A. If I have many kind of the components and I use BFS starting from one node, I might not visit the entire graph.

How do I keep track of what node that I've discovered and haven't discovered? What data structure do I use for these smiley faces?

AUDIENCE: [INAUDIBLE].

PROFESSOR: So for the smiley faces. For this thing I probably want a queue. We'd teach you something simpler, but most of the time when you write the code you actually use queue. What do I want for the smiley faces?

When I pull a node out, say I pulled out B, and I see A, D, E, I want to know that I visited A and I didn't visit the D and E. You guys remember that we didn't visit D and E, right, by the time we got to 2? OK. So I want to be able to check whether I visited

a node or not really fast. What data structure should I use for the smileys?

AUDIENCE: A hash table.

PROFESSOR: A has table. Cool. What's a hash table in Python?

AUDIENCE: A dictionary.

PROFESSOR: A dictionary.

All right, so this is going to be a seen, which is a dictionary, and it maps vertices to maybe true.

AUDIENCE: [INAUDIBLE].

PROFESSOR: Yep. Python has a set, right? So you can use that.

The smileys needs to be some sort of a dictionary. So given that, and assuming that this is a queue that we can extract and you can start from in constant time, what is the running time of BFS? Let's make life easy. Let's assume the graph is connected and let's assume just doesn't exist. BFS was the running time.

AUDIENCE: [INAUDIBLE] visiting.

PROFESSOR: So you visit every node once, so it's at least V. That's a start. Now what do you do when you visit the node?

AUDIENCE: Check out all the--

PROFESSOR: All the neighbors, right? Given the node, you have to go to the data structure that you have that's either this or this, and you have to get a list of neighbors. If I have this data structure, how fast do I get a list of neighbors for a node?

AUDIENCE: Order of the degree?

PROFESSOR: Yep. Order of the degree of the node. That's good. How about this data structure?

AUDIENCE: Same-- oh, no that's order of V.

PROFESSOR: Yep. So in this data structure, for example, C only has two edges coming out of it. But I have to go through the entire lining of the matrix to see where I have my 0's and where I have my 1's. In order to list the neighbors, here is straight up order V, whereas this is order of the degree of the node that I'm looking for.

So for this, where it's nice and simple, order of V, what is the total running time?

AUDIENCE: Wait, because it's B plus B. Oh no, it's the same one.

PROFESSOR: So for each node I have to?

AUDIENCE: V squared.

PROFESSOR: OK. So we're up to V squared already. What else do we need to do? Anything that they need to do on a node is order of V, so V squared is going to be the total running time. That's it. So if we use an adjacency matrix we get order of V squared as the total running time.

What about if we lists, what the running time? I'll give you a hint. You have to use amortized analysis. Shivers anyone?

[LAUGHTER]

We know it's order V because we're going to visit every node. And you guys didn't see what I was going to write here.

[LAUGHTER]

OK. So for every node I go through the neighbors and I do something to them. Right? I check if they're in seen, if not then I add them to the list. So for every node the work is?

AUDIENCE: it's degrees.

PROFESSOR: Yep. So if I look at all the nodes?

AUDIENCE: That's E.

PROFESSOR: Yep.

For every node I have to look at its neighbors and I have to see which neighbors are in seen. For the neighbors that are not in seen I have to add to my queue. Checking if a neighbor is in seen or not is order 1. Adding it to the queue is order 1. So the total work for a node is order of neighbors because of the adjacency list, not matrix.

If I look at the entire graph, here I can't do a local analysis like I did before. If I look at the entire graph and I look at all the nodes slash vertices, then the total work is the sum of their degrees. For each node is degree. For total work, sum of the degrees. And I have that nice handshaking lemma that says that the sum of the degrees is 2, E. So order E. So total running time? V plus E.

OK, can I say that the running time is order E? With no V's?

AUDIENCE: No, because you have to look at every--

PROFESSOR: Is it?

AUDIENCE: V greater than [INAUDIBLE] in case of not connected components?

AUDIENCE: [INAUDIBLE].

AUDIENCE: It's possible that you could have more vertices than edges--

PROFESSOR: OK. So it's possible that I have more vertices than edges. I agree with that, but I do anything to the vertices that I haven't seen?

So this is subtle. The difference between this and this is a matter of how you implement everything. In CLRS they assume that the seen is an array. So all of their nodes have numbers from 1 to V. So then they initialize an array, everything is false, and then they set the true elements.

In Python we can use a dictionary and not initialize it with anything. So if you do it that way in Python and code carefully, you can get to order E. The parts of the

graph that you don't discover, you don't have to pay for them. If you look at CLRS code, it is definitely V plus E. The difference between this and this depends on the code.

What's the point of BFS, by the way? Why do we care? What does it give us?

AUDIENCE: The shortest word path.

PROFESSOR: The shortest path from point?

AUDIENCE: From your start to where you're going.

PROFESSOR: So it gives you the shortest path from the node that I'm starting the BFS from. So only from this node? If I start BFS at A it's going to give me the length of the shortest path from A to which node?

AUDIENCE: [INAUDIBLE].

PROFESSOR: All the nodes?

AUDIENCE: Yeah.

PROFESSOR: All the nodes that are visited by BFS are reachable and we have a path from them. How do you compute this distance?

Let's see, so what's the distance from A to A?

AUDIENCE: 0?

PROFESSOR: 0. When I start from A and I see that its neighbors are B and C, what's the distance from A to B and from A to C?

AUDIENCE: [INAUDIBLE].

PROFESSOR: OK. Now I look at B. What's the distance between A and A? A is B's neighbor.

AUDIENCE: With A and A?

PROFESSOR: Yeah. So I have three neighbors. B has three neighbors, A, D, and E. So I care about the distances between A and A, D, E. Distance A, A, distance A, D, and distance A, E. This one is 0. How about the distances between A and D and A and E? They're 2.

So the way I would compute these distances is that I start with A being 0. The distance from A to A is 0. And then when I look at a node, when I discover the neighbors, all the neighbors that don't have smiley faces on them get the node's distance plus 1. You can get to them by getting in the current node and then traversing an edge.

When you do that it's important that you don't update the distances of the nodes that have smiley faces. If you do, you're going to say that the distance from A to A is 2 and all hell breaks lose from there.

AUDIENCE: Wait, why would you say it would be 2?

PROFESSOR: If I forget the fact that it has a smiley face. So if I go through all these neighbors and I say the distance from A to B is one then the distance from A to all of B's neighbors has to be 2. That would be wrong.

AUDIENCE: Wait, from A to all of--

PROFESSOR: B's neighbors.

AUDIENCE: From A to all of-- Oh, because A is one Of--

PROFESSOR: B's neighbors.

AUDIENCE: Yeah OK.

PROFESSOR: The smileys tell me which vertices I've already seen and I've already, presumably computed distances for them. We don't want to update those. OK this is BFS in essence. One question. Between Facebook and Twitter, which one's directed and which one's undirected?

AUDIENCE: Facebook is undirected.

PROFESSOR: OK. Facebook is undirected. Why is it undirected?

AUDIENCE: [INAUDIBLE].

AUDIENCE: [INAUDIBLE].

AUDIENCE: When you follow someone they don't necessarily follow you.

PROFESSOR: OK. So this is Twitter, right? Twitter, directed because of follows.

Has anyone used Facebook recently? Did you guys see there's a new option?

AUDIENCE: The little scroll bar on the side?

PROFESSOR: Subscribers.

AUDIENCE: Oh, ya.

PROFESSOR: OK, so how do subscribers work?

AUDIENCE: You subscribe. It's like google plus.

[LAUGHTER]

PROFESSOR: OK. Directed are undirected? If I subscribe to you do you have to subscribe to me?

AUDIENCE: Directed.

PROFESSOR: So which one is it?

AUDIENCE: I guess Facebook is kind of directed now because you can unsubscribe from people.

PROFESSOR: Is it?

So Facebook has two graphs in it. They happen to have the same vertices. The people are the vertices in both graphs. But the friends relationship defines an

undirected graph. The subscribers relationship defines a directed graph. The graphs are completely different. And there are two graphs. That's the right way to reason about them. that's why it was slightly tricky.

Can someone think of a cool way to use BFS on Facebook?

AUDIENCE: That's networks, right? Figuring out how many people are in the first degree or second degree. MySpace was really into that.

PROFESSOR: Lincoln also does that, right? How many people are your friends? How many people are your friend's friends? So and so forth.

Now suppose you want to get to someone in Facebook and you don't know them directly. They're not your friends. Presumably, you want to get to them through the minimum amount of effort. So you want to see do you have a friend that knows them? If not, do you have a friend that knows a friend that knows them? Do you have a friend that knows a friend that knows a friend that knows them? So and so forth. So BFS will give you that minimum path.

OK. do the graphs make sense? So by the way, the BFS on Facebook is just the beginning of a ton of cool things you can with graph algorithms.