PROFESSOR: I'm going to review what we did with the craft inequality just a little bit, because evidently a number of people were confused about this. I'm going to put a little more notation in with it. For some people, notation helps. For other people, it hinders things. But after you've thought about it a little bit, a little more notation can certainly be helpful.

What we're trying to do in this craft inequality is, we're thinking of a set of symbols where, supposing that there's a codeword for each symbol, c of x is the codeword for symbol x, which is a string of binary digits. y1 up to y sub n. In the world of two-toed sloths, the representation of numbers that sloths use is binary, base 2. And therefore, they use the number associated with some sequence of bits like this would be the sum of y sub i, times 2 to the minus i.

In other words, it's the same thing as a decimal, except it's in a world where people have two fingers instead of ten. There's an interval associated with this number, also. And the interval is what you would get if you took an arbitrary real number and rounded it down to l sub i. Well, in this case to m binary digits. So that interval, then, is the number itself. And the other side of the interval is that number itself -- the two-toed sloth didn't like what I was going to say about it, and it changed my slide.

So, is that extra factor the 2 to the minus n, there. In other words, any number in that range, if you round it down to m significant binary digits is going to the this number here.

Well the point of this, if the number, namely, this base 2 expansion of a number y prime is in this interval, then y is going to be a prefix of y prime. Let me just give you some examples of that because saying it in words is confusing, the idea is very simple.

Suppose you have a binary string, 011. That corresponds to the number 3/8. Namely, 1/4 plus 1/8. And the interval there is going to be the interval from 3/8, including 3/8, up to 1/4. But not including 1/4. Namely, 1/4 will be represented as -- the sloth really is hitting hard this morning. Maybe I'm the sloth. OK. There we go. From 3/8 to 1/2. Namely, 1/2 is just 1. Nothing more than that. Or it could be 10 or 100, and so forth.

So then, as an example, 011 is going to be a prefix of all of these quantities here. That's a prefix of 0111 because 0111 is 3/8 plus 1/16, which is in that interval we're talking about.

0110 is a more interesting case. Because 0110 is itself, the number associated with it, is just 3/8. But what this is saying is, if you take the same number but expand it to four digits, according to what this says, r of y like is then in this interval. And therefore this prefix situation holds. So this is a prefix of this.

Why do I make it so complicated? I make it so complicated because I want to talk about the length of that interval. And the length of these intervals is denoted in this diagram here. Anytime I have a number expressed to n binary digits, it covers an interval of 2 to the minus n. And because of this prefix property, as soon as one number covers an interval, no other number have its base in that interval. In other words, all of these intervals have to be disjoint, exactly as is indicated here. So when you add up the size of all these intervals, you have to get something less than or equal to 1. And that's the proof of the craft inequality.

So, let's go on and talk about discrete source probabilities. Now the two-toed sloth has gotten his machine out there, he's really mad at me this morning.

If we try to model English text, you know that some letters are far more probable than others. Namely, if you take an enormous amount of English text and you measure the relative frequency with which each letter occurs, you'll get more or less stable relative frequencies if you take enough text. And these letters are far more probable than these letters. So that gives you part of a model. You can also say that successive letters are going to be very dependent.

Namely, t is very often followed by h, and h is often preceded by t. q u is even more of a case here, because as far as English language words are concerned, u always follows q. Some letter strings are words. Other letter strings are not words. There are constraints on grammar. And what is really the clincher, which says there's no way you're going to model English in any sensible nice way, is meaning. And even worse than that, depending on who writes the English, it might have meaning or it might not have meaning. And for those English texts that don't have any meaning, the entropy is going to depend very much on whether the meaningless text is written by a salesperson or it's written by James Joyce. And in one case you have -- well, in one case you have an enormous amount of freedom in what this sequence of letters are. And in the other case you have letters which are very, very constrained.

So what's the point of this? The point of this is, if you're interested in trying to find the source coding method for English, what you don't want to do is to start out trying to get the best statistical model of English that you can. Because it's a losing proposition. And by trying to do that, you'll spend all your time trying to get the model. And you won't get any insight into what you ought to do as far as source coding is concerned.

This is pretty much true throughout all of technology. You don't solve problems by first getting too far into the details of what the problem is, before you start thinking about structures of possible solutions. In other words, we always deal with technological problems by dealing with toy problems first.

Now, there's a difference between engineers who worry about toy problems. Because engineers, if they hate theory, usually, don't say what the toy problem is. But they have that toy problem very firmly in the back of their mind, because of all their experience. Theoreticians, on the other hand, make their models very, very explicit. They don't often like to say that they're toy models because DARPA doesn't tend to support things that are working on toy models. So they try to conceal this. And often they just hide the fact that they're using a model. So, all of this becomes very complicated. But, for you, if you're trying to do either engineering or mathematics or teaching, or just be a sensible person.

When you're dealing with these problems, be explicit about what your models are. Try to understand the toy problems before you understand the more complicated problems. If you understand that out of this course, it'll be a worthwhile course for you. And, of course, you won't understand it until you get lots more experience. But believe me, that's the way it is.

OK, so that's the whole point of this. You want to start with simple toy models. And I'm not just justifying the fact that we're going to study this incredibly simple model here, which is a toy model. But by studying this you will see that everything else follows. If you read Claude Shannon's work on information theory, this, in fact, is where he started. He started with a beautiful description of trying to model the English language. Finally wound up with talking about this. The conclusions he drew, from studying these discrete memory sources, lead to his general theorems about data compression on sources. They led to his general theorems about the capacity of channels. They led to the idea that you want to separate source coding from channel coding, and finally, they led to all of the modern ideas that we have about quantization. In other words, the simple ideas you get out of this generalize directly to everything that's known about information theory.

So. Enough philosophy, let's get on with the business of what we're trying to do. A discrete memoryless sources has the following properties. The source output has to be an unending sequence, x1, x2, x3, blah, blah, blah, of random letters drawn from a finite alphabet, x. In other words, we are taking these real sources. And we're saying, let's make them not real now. Let's put a probability measure on them. And in this probability measure, one of the things that the probability measure will do will be to describe the probability on each one of these letters and the sequence in which it's coming out of the source.

Each source output, x1, x2, blah, blah, blah, is selected from a common alphabet. Namely, if you're using English on one letter of the sequence, you're going to use English on every letter of the sequence. Going to use a common probability measure, with some probability mass function p sub x of x. This notation means this is the probability mass function for the chance variable X. A chance variable is like a random variable, except the objects are not necessarily numbers. The objects can be anything. So, a chance variable is a generalization of a random variable. So this probability mass function talks about the probability of each of the symbols in this alphabet x.

Then the final thing is, each source output, x sub k, is statistically independent of all other source outputs, x1 up x k minus 1, and x k plus 1 on to forever. This is a nice example, because if you're going to specify a source probabilistically, you have to somehow find a way of explaining what the probability of every possible event within this source is. This is an easy way of doing it. You say they're independent. And then you can find the probability of anything you want to find. So that's a generic way of putting probability measures on things.

So then, we want to go into the idea of prefix-free codes for these discrete memory of the sources. We've already talked about prefix-free codes. We talked about the craft inequality. You might have thought it was a little bit strange talking about the strictly combinatorial property of codes without talking at all about the probabilities, which are the things that led us into talking about these codes in the first place. Namely, we want to use unequal length, variable length codes. Because of the fact that some letters are more likely than other letters. And eventually we'll be using them because of all these constraints between different words.

So, for notation, let l of x be the length of the codeword for letter x in, the alphabet capital X. ok so that's the same as this y1 y2 up to y sub n. At some strength of binary symbols. Capital L of x is a random variable, where capital L of x is equal to little l of x, where capital X equal to x. Now, what the heck does that mean? It's just notation. In other words, what we're starting out with is this ensemble of letters. We have a probability assignment on each letter in that alphabet. And then what we would like to talk about is a length function on those letters. So we have little l of x, which is defined for each x.

We then want to talk about this as a random variable. Because when we choose some random letter, x, little x, of this ensemble, capital X, l of x becomes a random variable. We will always, in this course, use capital letters to talk about random variables. And we will always use little letters to talk about things which are not random variables. Excuse me, not not random variables but random variables or chance variables. I think we can probably leave it open now. It seems as if the sloth has gone away.

So then we want to talk about the expected value of the length. You talk about the expected value of something, you're talking about the expected value of a random variable. We will also denote the expected value of this random variable L with a bar over it, which is the sum over the letters in the alphabet, of p of x times L of x.

So all this is what you would do anyway if you never thought about this. Until, at some point, when you're taking a quiz or something and start to get confused and say, what is this stuff all about? I don't have any idea what this means after you've written five pages of stuff. So, it's worthwhile spending a little bit of time sorting that out.

So, L bar is the number of encoder output bits per source symbol. In some strange sense. Namely, it's this expected value. Now, to finish this off, if we want to look at the number of binary digits to come out of the source when a long sequence of letters come out of the source -- A long sequence of letters. x1, x2, x3, and so forth, come out of the source. They go into the encoder. The encoder is mapping each letter that comes out of the source into this codeword c of x. So we have a sequence of codewords which are all concatenated together. And, therefore, the total number of binary digits which has come out of the source corresponding to these n symbols that have come out of the source, is the sum of L of x1 plus L of x2 plus of x3 plus L of x4, and so forth. So we have a sum of independent random variables.

Now, what do you know about sums of independent random variables? Well, the one thing you ought to know about, and which should be stamped on your brain because it's the central thing that makes any probabilistic theory make sense, it's the only way that we can ever understand our environment. You look at the past. You try to figure out from the past what's going on in the future. And the only way you can do that, the only tool you have, really, is this law of large, numbers. Which says, when you see a long sequence of things, from that long sequence of things, you sort of figure out what's going on. If you're dealing with a random variable, the thing you do is add up all of these numbers. You divide by the total number of them that you have. And that gives you the expected value. It gives you a typical value. What the law of large numbers really says is, if you look at the sum of binary digits out of this encoder, over a very long period of time, divide by the total number of symbols, that's a random variable again. And this random variable is, with high probability, going to be very, very close to this expected value, which is this quantity here. In other words, the ensemble average, which is this, is going to be very close to the time average. And the time average, now, is a random variable. And that's what the law of large numbers says.

You see, the problem that we all have, dealing with real world problems, is that there's nobody to tell us this is what the ensemble is. Unless you believe somebody that doesn't know. And the only real evidence that you have is the actual sequence. And from the actual sequence, you then look at what happens for this particular sequence. You then build a model. And your model, by definition, has the expected value of L going to be equal to the expected value in the model that you've chosen.

So. What's your objective? Your objective in trying to form a prefix-free code, then, is to find a set of integers, L of x, which satisfy the Kraft inequality. And they minimize L bar. In other words, what we're trying to do is, we're trying to choose a code which minimizes the expected length of the code. Which is really, over a long period of time, going to minimize the number of binary digits that come out of the source encoder. What we want to do is to choose these integers to minimize this.

So what we're going to do now is, suppose our alphabet is just 1, 2, up to capital N.

What am I doing here? I'm saying, we don't care about what these symbols are called, anyway. It's totally irrelevant what the names of the symbols are. So I will name them 1, 2, up to capital N. Probability mass function, then, I can denote as p sub 1 up to p sub capital N. In other words I've gotten rid of all these axes that were lousing up our equations all along. Now, I'll denote the unknown lengths by l1 up to l sub M. So the problem is, somebody gives you this set of numbers. p1 to p sub M, which is a PMF. In other words, these numbers add up to 1. And tells you, I want a prefix-free code which minimizes this expected length. Namely, the expected value corresponding to these lengths here. So, the expected length, to minimize it, what we want to do is to minimize over the choice of l1 up to l sub M. Subject to the craft inequality, we want to minimize this expected value.

So we have a nice, clean, mathematical problem now. We want to minimize this sum, subject to this constraint. And the constraint includes the fact that all of these things have to be integers. Well, for those of you who have studied minimization, there's a funny thing in here. Because integer minimization problems tend to be very, very nasty. And, therefore, you look at this and you say, this is probably something I'm going to have trouble solving. Strangely enough, it isn't. But you would think it probably is something which will be hard to solve.

So, since integers louse up minimization problems, what do we do? Well, we say, just for fun let's try to solve this problem without the integer constraint on it. Let's see what that leads to, and see if we can do anything with that. So we say, OK, let's try to minimize this function here over the integers l1 to l sub M, subject to this constraint. So we're minimizing this, subject to this constraint.

Now, an easy way to do that -- yes. AUDIENCE: Are you

saying that side length is not a fixed probability.

PROFESSOR: No, I still have these fixed probabilities. I still have p1 up to p sub M, as known probabilities. But I'm going to say, let's suppose I can choose a length which is two point five bits instead of two bits.

AUDIENCE: You're saying the shortest length [UNINTELLIGIBLE]

PROFESSOR: Well, we're going to wind up there eventually. But for now, all I want to do is to look at this problem. If I start out by saying, assign shortest lengths to the biggest probabilities, I have two problems. One is, it's a little hard to prove to you that I want to do that. Although we'll do that later today. And the other is, it doesn't really give you the general properties that we want to know about this. So, for those two reasons, I want to just attack this as a straightforward mathematical problem. If you're a computer scientist, this looks strange. Because computer scientists like to attack problems by algorithms. Analog engineers like to attack problems by writing a complicated formula, and taking derivatives, and all sorts of things like that. We're going to be doing both of those things in this course. And you'll see that both of them lead to certain advantages.

And here, we're taking, where the analog engineer's approach is saying, suppose this is a bunch of numbers. I want to minimize this function over a set of numbers, l1 up to l sub capital M. So, how do I do that? Well, this guy Lagrange, he was a great mathematician. He was also a great mathematician early enough that he could do some really trivial things and become famous for them. Just like Kraft that we were talking about before. But, unlike Kraft, Lagrange really did a lot of other very important things. And what Lagrange said was the following: Well, suppose I want to minimize this sum. And I want to have this constraint added in. Sort of what I want to do, then, is to minimize a weighted sum of this, which is what I'm interested in, and this. In other words, if I minimize this weighted sum here of these two things, I'm going to wind up with some sort of value for this. And some sort of value for this. By changing lambda, then, which stands for Lagrange, he was also clever in making himself famous that way, by changing lambda, I can change the balance between how important these two things are. And as I change the balance between how important they are, when I change it to just the right place, I'm going to have this constraint here satisfied with equality. So that's the whole idea of Lagrange minimization.

So we take this function. How do you now minimize a function of multiple variables?

Well, again it's a messy problem. But the first thing you can try to do is find a stationary point. So, let's always do the easy thing first. We take the partial derivative of this function here, with respect to l sub i. That's what we're trying to minimize. And what we get is p sub i minus lambda times the natural log of 2, times 2 the minus l sub i.

I'm not very good at differentiation any more, so I only differentiate things which are easy. And that's easy. I want to find a stationary point, so I set this equal to 0. That makes the problem worse, because now I have a function of lambda and also all of these l sub i's. But now I choose l sub i, so that I satisfy the constraint. Namely, I choose lambda to satisfy this equation here. When I choose lambda to satisfy this equation here, what I get is p sub i is equal to 2 to the minus l i, and therefore, l sub i is equal to minus log p sub i. In other words, I have this equation here. What happens when I sum this equation over i? Let's look at it.

We sum this over i. The sum of p i over i is 1 minus lambda times natural log of 2 times sum of 2 to the minus l sub i. And I want to make this equal to 1. So what I get is 1 is equal to this times lambda natural log of 2. So, I hope that choosing lambda equal to 1 over natural log of 2 is what I want to do. And when I do that, this becomes 1 here. And I just have 2 to the minus lambda i is equal to 1. OK, good.

And then, going back to this equation, p sub i is equal to 2 to the minus l sub i. OK, this is this arithmetic. I mean, if you don't follow what I'm doing, just look at it later and you'll find that there's nothing really there.

So we wind up with these lengths being equal to the negative of the binary logarithms of these probabilities. It's only a stationary point. We don't know whether it's a minimum yet. And, unfortunately, we also have the problem of, they might not be integers. But, anyway, what we wind up with, then, if we ignore looking at these problems for the time being, is that the lengths are going to be equal to this. The expected value of the lengths is then going to be equal to the sum, over i, of minus p sub i times the logarithm of p sub i.

When Shannon saw this, and when various other people saw it, they said, gee, this looks like the entropy of statistical mechanics. So let's call this quantity entropy. For no better reason than that. And it would probably have been far better if they called it something else. Because for years, there were physicists and philosophers trying to figure out what the deep relationship was between statistical mechanical entropy and information theoretic entropy. And there probably is such a relationship, but the relationship is far more complicated than understanding information theory. And it's far more complicated than understanding statistical mechanics. So I advise you to not worry about that one until after you understand what it means in an information theoretic sense.

So h of x is what we call the entropy of the random variable x. And it really is the entropy associated with these logarithms of p sub i. So when you take functions of a random variable, a random variable carries along a lot of baggage with it. Including the probabilities of everything. And when you take the expected value of a random variable, the individual values of the sample points of that random variable are important. And the probabilities are important.

Here we have something even stranger. Because it's only the probabilities that have anything to do with it. And this makes sense. We already said that these symbols have nothing to do with this problem we're dealing with. You can call the symbols whatever you want to call them. And, therefore, the only thing of any interest to us is these probabilities that we're dealing with. So H of x is a function only of these probabilities. It's the expected value of minus log p sub i. This is called entropy, and in fact we will find out very shortly that it really is the minimum number of bits per source symbol needed to represent the source. In other words, when we generalize the problem from just plain ordinary garden-variety oh prefix-free codes, we will find that this number is really what characterizes the whole problem for discrete memory-less sources. So let's go on and say more about that.

Let's say something about bounds on the entropy. First, what's the relationship between the entropy and this minimum of the expected length that we started to talk about? And I claim that H of x is less than or equal to L min, which is less than the entropy plus 1. And why is that?

We already have the machinery to see this. We almost have the machinery to see this. Namely, we have solved this minimization problem. We've only found a stationary point, but we've ignored the fact that we have an integer constraint. So, if you allow me to say for the time being that, in fact, when we solve the problem without worrying about integers, that it actually gives me a minimum, then in fact this follows very easily. Because what I'm going to do is to find those optimal lengths, which are non-integers. And then I can solve the prefix condition and get a code by simply increasing each of those numbers to the next integer up. In other words, I can take the ceiling function of each of those real numbers to get an integer. When I take the ceiling function, 2 to the minus l sub i is going to go down. So the craft inequality is still satisfied. So the entropy has to be less than or equal to this average. Has to be less than H of x plus 1.

So the average is equal to H of x if and only if each these probabilities it an integer power of 2 to start with. In other words, the solution I came up with before is that the length I wanted should be equal to minus the logarithm to base 2, of p sub i. So if p sub i is already a power of 2 then I'm home free. Because I just pick that length to be minus log of p sub i, and it happens to be an integer. And I don't have to round it up.

So if I let l1 to lM be these codeword lengths -- well, here's where I'm going to prove this to you. And the proof is the following: I want to prove that H of x is less than or equal to l min, which I'll just call here L bar. So H of x minus L bar is equal to, this is the entropy. This is the expected length here. I can rewrite this as the sum of p sub i times logarithm of 2 to the minus l sub i divided by p sub i. That's just arithmetic. l sub i is equal to to logarithm of 2 to the l sub i, that's equal to minus the logarithm of 2 the minus l sub i. So I get this.

There's an inequality. Hate to call it an inequality, it's so trivial. Here's the point 1. If you plot a natural log of x. And if you compare it with the function x minus 1, you can see that natural log of x is less than or equal to x minus 1.

Now, this an inequality which happens to be very useful in information theory. I would claim that any inequality that you can prove in information theory, by any means at all, I can prove using this inequality and nothing else. And I've believed that for 50 years and nobody's proven me wrong yet.

And also, this is something you can draw and remember. So it's simple. So the idea, then, is since this sum log of 2 to the minus l over p sub i is less than or equal to the natural log of 2 times the natural logarithm of this. You just, here we go. For any u greater than 0, natural log of u is less than or equal to this. So the logarithm to the base 2 of u is less than or equal to the logarithm to the base 2 of e, which is some number, times u minus 1. With equality at u equals 1. So this is less than or equal to this. And look how nice that is. The p sub i's cancel out and you get the sum over i, of 2 the minus l i minus p sub i is less than or equal to 0. An equality occurs if, and only if, p sub i is equal to 2 to the minus l i. OK?

So that's all there is to it. And that establishes that -- well, establishes part of this theorem here. And the other part we already established, And if you don't believe me, the notes do it more carefully.

Well, this left us a serious problem unknown. Which is, how do you actually solve this integer minimization problem. How do you solve it if you have a big, long, complicated source with lots of probabilities in it? And everybody thought it was hopeless. Even Shannon thought it was hopeless. And Shannon sort of figured out ways to approach this problem. He said, well, you want to have about half the probability. Starting with 1, about half the probability starting with 0. So he would divide up the symbols in the alphabet, so he could come as close is possible to half of them being up here and half of them coming down here. And he would continue to do that, I mean, I don't usually like to write on the blackboard, but he would start out generating a code like this. And this would be approximately 1/2. This is approximately 1/2. And then he would take these symbols. Split them, again, in probability.

And everybody was starting the problem over here, and trying to generate a code working their way out. Well, Dave Huffman was a graduate student at the time. and he took Bob Fano's graduate course in information theory, I think a year or so later than Kraft did. And Bob Fano assigned as a homework problem, how do you solve this problem? Sneaky guy. And he was very amazed when Dave Hoffman came in next day and said, oh, it's easy, you do it this way.

So the question is, how did he do it? Well, Huffman, instead of looking at the problem from here out, looked at the problem from here in. He was -- I mean, this was before there was anything called computer science. But he thought like a computer scientist did. In other words, he thought algorithmically. And he also thought in terms of discrete problems. And therefore, he looked for properties that these optimum codes should have. And it was neat. So, he started out with a limit. He said, an optimal code has to have the property that a p i is greater than p sub j. Then the optimal length associated with p sub i, namely the optimal length of the i'th codeword, had to be less than or equal to the length of the j'th codeword.

And you can see this by saying, well, suppose that's not true. Suppose that p i is greater than p j. And also, li is greater than lj. And then you say, OK, take this situation. We will interchange those two codewords in the code. And we'll look at what that does to the average. And if you work that through, you find out that since what you've done is, you've shortened the codeword associated with this and lengthened the codeword associated with this. You have changed the average length to make it smaller.

Now, let me warn you about something. When you start looking at these properties, the most confusing thing is what happens when two probabilities are the same, or when two lengths are the same. And I would advise you to just ignore that problem, until you get an idea of what's going on. Namely, assume that all lengths are different. All probabilities are different. And then it's easy to see what's going on. And when you get all done, go back and straighten out the cases where things are equal. And I think the notes do this carefully. If you read books on information theory, about half of them do it carefully, and about half of them don't. So you should be suspicious. But anyway, that's one of those trivialities that you just have to sort out for yourself.

OK. The next lemma is optimal prefix-free codes are full. We talked about what a full code is. When you draw this binary graph for it, you don't have any nodes in a binary graph -- you don't have any leaves that are not associated with codewords. Because if you do, we showed you that shortened the codeword of the part of the tree on the other side of that leaf. In other words, if you have something here, if this is a codeword and this is not a codeword, then you just get rid of this, and bring that back here.

But this is a whole tree stemming off here, you do the same thing. You take this whole tree and you bring it into there, and you throw this away. So, optimal prefix-free codes are full. So far there's nothing to this.

The next part of it is the sibling of a codeword. And what's a sibling? Well, we used to call it a brother. But then couldn't do that because we have to call it a brother or sister. And that got to difficult. So people invented the word sibling, to talk about a brother or a sister. So the sibling of a codeword for is the string form by changing the last bit. In other words, in this family tree here, the sibling of this is this. The sibling of this is this. The sibling of this is this. So, a leaf can have a sibling which is an intermediate node, and vice versa.

So then he said, the sibling of a codeword is a string formed by changing the last bit. I think he probably said the brother, but, anyway. For optimality, the sibling of each maximum length codeword is another codeword. Now, that's a really simple one. If I make this a codeword, and this is the maximal length codeword in this code I'm talking about, this can't be an intermediate node because then there would have to be longer codewords. And it can't be empty because these optimal codes are all full. And therefore, this has to have a sibling which is also a codeword. So the longest codewords have to have siblings.

Well, that's easy enough. Incidentally, one of the problems that you have in proving this sort of thing is, what happens if you have zero-probability letters. Well, we just get rid of that problem and say, well, there aren't any zero-probability letters. Because if we want to come up with a sensible model for something, we're not going to create a codeword for something that can't happen. So, there are no zero-probability letters in this alphabet. I mean, if you want to put them in, it just complicates the whole thing. And you can do it.

Then, finally, there's this lemma which says, there is an optimal prefix-free code in which, after you order the probabilities of all of the messages, namely you order p1 to be greater than or equal to p2, greater than or equal to p sub m. In other words, we just rename the letters in the alphabet, so that letter m is less likely than letter m minus 1, and so forth. Back to 1. 1 is the most probable, m is the least likely.

Well, we've already concluded that we want to assign the longest messages to the least probable codewords. And this says, take the two least probable codewords and we can always make an optimal code in which those two codewords are siblings. And the reason for that is, one of them is not going to be longer than the other or else you can shorten the code by interchanging things.

So there is an optimal prefix-free code in which the codeword for m minus 1. And the codeword for m are maximal length and they're siblings. So the Huffman algorithm first combines these two. And then looks at the reduced tree with m minus 1 nodes. Let me show you an example of that.

So it starts out. Here, I've ordered the probabilities associated with a set of symbols. The symbols are 1, 2, 3, 4, 5. The two least likely messages are 0.1 and 0.15. Obviously, I could've interchanged these two if I want to. But why interchange them? So I say, OK, the last digit on this one, I'm going to assign to be a 0. The last digit on this, I'm going to assign to be a 1. And the important thing is, I'm going to make them siblings in this tree. And what I'm going to do now, terribly complicated thing, instead of building a tree from left to right, I'm going to build a tree from right to left. So when I get all done with the tree it's going to come in like this. And what I'm doing is starting out at the end, to start to build the end of the tree. And what happens after I go through this first step is, I say, OK there is an optimal code. In which these two quantities are siblings of maximal length. I now want to form an optimal code for these probabilities here.

So, I go back and I iterate again. And I said, OK, if I have these probabilities here, what's the optimal code. Well, I could reorder the things. But now I know that the only thing I'm interested in is the two least likely symbols in this new alphabet here. Which is 0.2 and 0.15. So I combine those together. I tie them together as siblings in this last generation, however it works out.

So then I have an alphabet of size three. And then down here, I have these two things tied together. These two things tied together. So I have a node of probability 0.25. I have a node of probability 0.35, and I have a node of probability 0.4. I take the two least likely, and I tie them together. And then I have two nodes left, one with probability 0.6 and one with probability 0.4. And I tie them together. And, presto, I have my whole code, except for flipping it over, to go from left to right if you like. Codes that go from left and right, instead of right to left.

OK. I have swindled you. How have I swindled you? I mean, I've swindled you a little bit by talking about these things that might be equal or not equal. And that's not important. You can sort that out on your own. There's a very important swindle I pulled. And what's that? What's very incomplete in this argument? This part is fine. Nothing wrong here. We have a lemma which says, you can find an optimal code by tying these two things together. Yeah?

AUDIENCE: [UNINTELLIGIBLE] combine those two [UNINTELLIGIBLE] combination.

PROFESSOR: You're saying, how do I know to combine these two? OK, which means what? Yeah.

AUDIENCE: [UNINTELLIGIBLE] you've just added the probabilities --

PROFESSOR: I've just added those two probabilities. So I have a new ensemble where I have four probabilities, 0.25, 0.15, 0.2, and 0.4. And that's fine. I still have these things. No, there's no independence involved here at all. I mean, I started out with five letters. Which are disjoined. I now have four letters that are disjoined. What have I done? Yeah.

AUDIENCE: [UNINTELLIGIBLE]

PROFESSOR: Yes. Yeah. I have assumed, now, that once I get these four symbols, if I have those four symbols, I can form an optimal code for those four symbols in which these two symbols get tied together. But how do I know that an optimal code for this reduced set of probabilities is also an optimal code for the original problem? I have tied these two things together. I know there's an optimal code in which these two things are tied together. I then have four symbols. I want to find a code for those four symbols. But I assume that the optimal code for these four symbols, when I break apart these two things, gives me an optimal code for five symbols.

That's the sort of thing I want you people to start catching onto immediately. I want you to start asking those nasty questions. And those nasty questions are the things that say, OK, how do I know that this works? In other words, you're not here to learn these algorithms. I can tell you what the algorithm is in an instant. You can do the algorithm. A computer can do the algorithm about three thousand times faster than you can. And you can be replaced by a computer, if you only learn the algorithms. You can program the algorithm. You can probably find the computer that can program the algorithm too. And there's no need to program it more than once. So that after you've done that, you are useless again. So the only thing that's worthwhile for you is to be able to spot these problems and to understand what's going on.

So. How do I know that this first optimization leads to the second optimization. After combining these two least likely codewords, or siblings, we've gotten a reduced set of probabilities. In this problem here, what we've done, the reduced set of probabilities are 0.4, 0.2, 0.15, and 0.25. Why does finding the optimal code for this reduced set result in an optimal code for the original set? That's really the question that we're asking.

Well, it's not hard. If you take any code for the reduced set, let's call the reduced set x prime, set of probabilities. Let the expected length of that be l prime. It's not necessarily an optimal code, but it's any old code that I generate. Any old code I generate for L prime, I can now take that code for l prime and I can expand it out to a code for L. Namely, I have this code here this, this, this, and that's the expanded -

- and now I can expand it into a code for the original set, by adding on this and this, as leaves on this. This leaf here then becomes an intermediate node. And I add two extra leaves to it. OK, well, it's not hard.

The expected length for this code, for these five letters, I claim, is equal to the expected length for this reduced code, this, this, this, and this. Plus one extra digit for this. Plus one extra digit for this. So the expected length for L is the expected length for L prime plus 0.15 plus 0.1. Which says the following: if I want to minimize this, and I know that this has to be equal to this, and these two numbers are fixed, I can't change them. I can minimize this, by minimizing this. And that's the final step in the whole argument.

And what's peculiar is that everybody learns the Huffman algorithm. And what Huffman did, which was really very smart, was to sort out this issue. And I can teach this to a hundred classes, and nobody will ever point out to me that there's a logical flaw in the whole argument. And you can look at most books on information theory and they never point out that there's that logical flaw there, either.

So, anyway, that's the end of Huffman's algorithm. You can see when you look at this that this is really an extraordinarily easy thing to do. I mean, you can take an alphabet of several thousand symbols. All you have to do is order them. Tie the least two likely together. Assign a 1 and a 0 to them. Then, stick that into an ordered list again. Take the two least probable. Tie them together. Stick it into an ordered list again. And, if you have some minimal knowledge of data structures, you can do this with essentially on the order of one operation for each letter in this alphabet. So it really isn't a very difficult sum.

So here's an integer problem which is really easy to solve. And the way to solve it is to look at the problem in the opposite way from what everybody else has looked at it in. Does this say you want to ignore everything that everybody else has done, and go your own way? Not quite. But it says that's one of the things you ought to try, if you find that everybody is doing something one way and you can find another way to look at, that's very rich. It might turn out to be nothing but it might turn out to be something very worthwhile.

Let's now talk about this quantity, entropy. And for every chance variable, x, if that chance variable, x, is discrete and has a finite number of elements in it, so I'm talking about a chance variable x, what's a chance variable have tagging along after it? It has a set of n probabilities tagging along after it. That's what a chance variable is. A chance variable is not just the alphabet. A chance variable is the alphabet plus the probabilities. That's why you then talk about it having an entropy. And the entropy is the expected value of, minus the logarithm, of this PMF function. So, in fact, this is an unusual statistic in the sense that it has nothing to do with the symbol values, and everything to do with just the probabilities of the symbol values. And as we go on, you'll see that in fact this is a very important property of it. And dealing with the logarithms of these symbol values is, in fact, a much more worthwhile thing to do than dealing with the probabilities of the symbols.

Now, let me pause again and see if anybody can have any idea of why logarithms of probabilities might be more significant than probabilities. And think of what we're going to be doing here. We're taking a sequence of letters. When I take a sequence of letters, what's the probability of the sequence of letters? If they're IID. Namely, we're looking --

AUDIENCE: [UNINTELLIGIBLE]

PROFESSOR: It's the product of those probabilities. Now, if you agree with me that the probability theory is concerned 50% with the law of large numbers and 50% with everything else all put together, why is the logarithm of a probability important?

AUDIENCE: [UNINTELLIGIBLE]

PROFESSOR: You change your product to a sum, yes. If you have a product of probabilities, you can talk about a sum of the logarithms of probabilities. That's why entropy is important in statistical mechanics. It also is, fundamentally, the reason why entropy is important in information theory. Is because what you're almost always interested in is a product of probabilities. And when you're interested in a product of probabilities and you want to use the law of large numbers, you turn that product of probabilities into a sum of the logarithms of probabilities. Fundamental idea. Shannon took eight years sorting all this out. And Shannon was by far the smartest person I've ever met. I mean, the problems that we worry about, he just, bip. Solves it with no effort at all. This one took them a while to sort out. It also took him a while to sort out the fact that once he sorted this out, he could sort out all of the other problems. As far as communications was concerned. So was quite important.

I mean, I can tell you one of the peculiar things about Shannon. Just from the first time I ever talked to him about a technical problem. I'd just become a faculty member here. And his office was about five doors down from mine. And one day I screwed up my courage to go down and talk to the guy about a problem I was working on. And I thought it was a really neat problem. It had all sorts of pieces to it, all sorts of bells and whistles on it. And I started to explain it to him. And he said, well, can we look at a slightly simpler case where you throw out this part of it, you throw out one bell. Then he'd throw out a whistle. Then he'd throw out a bell. And I was going along with this and saying, yeah, I guess we could. We could. We can throw out all of these things without really losing the essence of the problem.

And, finally, I started to get discouraged. Because this really neat research problem, this really important research problem, was turning a toy problem which was almost trivial. It had nothing to do, it seemed, with anything. And finally we got down to a certain point. And I said, yeah, but this is trivial, the solution is this. And he said, yeah. And then we started putting back all the pieces. And his genius was, he knew which things to throw out. So that each of the things we threw out, we could put them back in again. When we got done, the research problem was trivial. And his genius was in finding the right trivial example to look at. So, in fact, what you always want to look at, in the communications field -- and in most fields, I think -- is finding the really simple way of looking at something. Which means you have to throw out most of the nonsense.

So, in this case, it's looking at entropy, which is the logarithm of a probability assignment. And you want to look at that because the logarithm of a probability assignment lets you add the logarithms of probabilities. Use the law of large numbers. And then you can talk about sequences of elements.

Properties of entropy. For a discrete random chance variable. We have m elements in the alphabet. First thing is that the entropy is always greater than or equal to 0. Why is that? I'll let you figure it out. Why is the logarithm of a problem, minus the logarithm of a probability, greater than or equal to 0? Why is it non-negative? Yeah.

AUDIENCE: [UNINTELLIGIBLE]

PROFESSOR: Probabilities are always less than or equal to 1, yes. So this quantity here is always greater than or equal to 0. Because the logarithm of 1 is equal to 0. We have a quality here if x is deterministic. Which is just a special case there. Where you have an ensemble of one element and it has probability 1. Or, in fact, at this point you could add in things which have zero probability. Well, that's a little bit tricky. Because you add in something that's zero probability. And the logarithm of 0 is infinity. So you're dealing with the expected value of a bunch of infinities, which each occur with zero probability. And you're forced to say, well, I think that 0 times log of 0 is equal to 0. And in fact, epsilon times log of episilon goes to 0 as epsilon goes to 0. But you save yourself a lot of worry by just leaving out things of zero probability.

So H of x is greater than or equal to 0. We have equality if x is deterministic. H of x is less than or equal to log of m. The quality, if x is equiprobable. And how do I know that? I look at this again. I'm not going to prove it here but, essentially, this follows from saying that the natural logarithm of something is less than or equal to that something minus 1. And you take the difference of the entropy, and log of m. And, presto, it gives you the result that you want.

So you've got the most entropy, if everything is equiprobable. For any code satisfying the Kraft inequality, the entropy is less than or equal to L bar. Well, that's what we already proved. Mainly in the middle of the lecture, we showed that for any code to satisfied the Kraft inequality, the entropy was always less than or equal to L bar, because the entropy is what you get if you minimize the expected length without the integer constraint. And L bar is what you get -- well, L bar min is what you get when you minimize it with the integer constraint. I mean, you don't bother about minimizing it. You get something bigger than L min. So this is less than or equal to the length of any codeword.

For the very best codeword, for the very best code, the expected value of the minimum is less than or equal to the entropy plus y. And you get that just by adding to each non-integer length the ceiling function. Which gives you, at most, one extra digit for each codeword. Now, here's the more interesting one. For independent chance variables, x and y, here's where the nice part about notation comes along. What's the entropy of x y? Well, what do I mean by x y first? I have a chance variable, x. And this chance variable x has an alphabet associated with it. x1 up to x sub m. I have a chance variable y. It has an alphabet associated with it. What's the sample space, what's the set of events corresponding to the chance variable x y? By x y, I mean a chance variable whose elements are the possible values of both x and y.

So, I'm talking about the joint ensemble of x and y. I have a bunch of possible values for that. And those possible values, if I have m possibilities for each, I have m squared possible values for the two of them. So I'm talking about the expected value of minus the logarithm of the probability of x and y. In other words, I am trying to take the -- let me write it out. I'm probably given conniptions to -- no? OK.

I want to take p of x y of symbol x y, times minus the logarithm to the base 2 of p sub x y of x y. That's what this means if I write it out.

Well, this probability here is p sub x of little x, times p sub y of little y. Why is that? Because I'm assuming that they're independent of each other. And, therefore, the probability of two of them is the product of the probabilities, times minus log to the base 2. So if p sub x of x minus logarithm to the base 2 of p sub y of y. And I'm summing this over all x and all y.

And the more sophisticated way to write this -- things I say in lecture, you don't have to copy down because they're always in the notes. If they're not in the notes, it's probably wrong anyway, so. So this expected value is expected value of the logarithm of the probability of x y. Which is the expected value of the logarithm of p of x times p of y. And, since I have a logarithm of a product, that's the expected value of minus log p of x minus log p of y, which is the entropy of x plus the entropy of y. In other words, when I have a joint ensemble of even more independent quantities, The entropy the sequence is equal to the sum of the entropies of the individual elements in that sequence.

Well, that's all I wanted to talk about today. If any of you have any questions to ask, you should ask them now. I went through Huffman coding pretty quickly, because it's something where you have to do some exercises on it to sort it out. And I didn't want to do any more than that.