[SQUEAKING]

[RUSTLING]

[CLICKING]

PETER SHOR: OK, so today we're going to be talking about error correcting codes. So on Thursday, Professor Moitra told you about Shannon's noisy coding theorem. And I don't know if he drew this picture or not. It's not in the notes. And it's a picture everybody draws when they talk about Shannon's noisy coding theorem.

So you have some message, message drawn from m, which is some set of messages. And now you encode it. And now you have a noisy channel. And you get c tilde, or you get the received code word, which maybe I should call it c tilde. And now you decode and you get something which I'll call m tilde.

And Shannon, if the noise at the rate which I want to say is log k over-- what is the rate? It's k over n, right-- so these messages are k bits, and this code is n bits-- is less than the channel capacity, there exists a code such that as n goes to infinity and k over n remains the same, the probability that m tilde equals m goes to 1.

So in the limit of long block lengths-- I don't know how much coding theory terminology Ankur used, but n is called the block length-- then the error rate goes to 0. So that is the noisy-channel coding theorem. So did Professor Moitra put this diagram on the board? OK. Oh, well.

And Shannon used random codes to prove his theorem. And these are totally impractical for large n. Because what you have is you have a list of code words, which is exponential in n, and you get a received word. And you want to find which of these code words it's closest to. And we don't know any way of doing this that is substantially better than going through the entire list of code words, computing the distance, and finding the shortest one. And in fact, you can show that this is an NP-hard problem, which means that there probably isn't such a way.

So what you need is you need to find specific codes with practical decoding algorithms. And today and Thursday, I will tell you about two of these codes. And they will basically stand in for the whole class of linear codes.

So, Shannon proved his theorem in 1948. And two years later, Hamming found the first error-correcting code. Hamming, 1950, actually found a whole class of error-correcting codes. But the first part of the talk, I'll just talk about the simplest one in this class-- can encode four bits into seven bits and correct one error.

So if you have, I don't know, a million bits that you want to send, and their error rate is 1 in 100, then this is not going to work because you can break the million bits into blocks of four and transmit each of them in seven bits. But chances are very high that in these million bits you will find one block of four which has two errors in it, which means it will not be corrected. So what you really would like is you would like a way of encoding, say, 1,000 bits into 1,500 bits that correct 10 errors. And we will get to something like that with Reed-Solomon codes, which we'll talk about on Thursday.

But what is the Hamming code? So I'm going to give you, first, a very simple description of the Hamming code, and then a more complicated one. And the simple description, we're going to start with a Venn diagram with four bits-- well, with three regions. And these regions are actually going to correspond to specific bits of this code.

OK. Now, let me see if I can do this. I want to do this. So let's take the four bits to be bit a, bit b, bit c, and bit d. So these are the message bits. And now what we're going to do is add three parity check bits. Add these-- 4 plus 3 is 7, which is why we're encoding four bits into seven bits. And these bits are going to give you the error-correcting ability.

So let's say this is a plus b. This is a plus b plus d. This is b plus c plus d. And this is a plus c plus d, where the plus is exclusive r, so binary addition. Example, let's take 1, 0, 1, 1 to be the message. So, 1, 0, 1, 1. I'm sorry, 1, 0, 1, 1.

So now we want to compute the parity-check bits. So what we do is we put a bit in here so that the sum of the bits in this circle is even. So that would be 0, because 1 plus 1 is 0. And similarly, this is 0, because 1 plus 1 is 0. And this is 1 because 1 plus 1 plus 1 is 1. And we know to add a 1 to make it 0. OK. So our code is 1, 0, 1, 1, and now 0, 0, 1.

OK. Now let's send it through the channel, and we will change one of the bits. Does anybody have a suggestion for which bit we want to change? Who wants to change the second bit? OK, we'll do that. So it's 1, 1, 1, 1, 0, 0, 1.

So what we do now is we compute. Let's call these a, b, c, d, e, f, g. And this was a, b, c, d, e, f, g, I believe. Yeah. So now we compute a plus d plus c plus g, a plus c plus d plus g. That is 0, so this circle is correct.

Now we compute a plus e plus b plus d, a plus e plus b plus d, that's 1. So this circle is incorrect. And b plus c plus d plus f, b plus c plus d plus f, that-- b, c, d, f-- That's also equal to 1. So this circle is also incorrect.

And so now we know the incorrect bit is not in this region, but is in the intersection of this region and this region. And that means it is b. So the received word is not incorrect. Bit is b. And now we change b, so we get 1, 0, 1, 1, 0, 0, 1. Right. And while you might have noticed that the message was the first four bits, so this is the message. OK.

So now I'm going to tell you about general linear codes, and then general Hamming codes. A linear code over a field F is a collection of length n strings in F. So, if c1, c2 are in the code-- I should probably call this a code C-- then c1 plus c2 is in the code. And if c1 is in the code and alpha is in F, alpha times C1 is in the code. So any linear combination of code words is also a code word.

And it's fairly easy to see that this holds for this Hamming code. Because if you have a set of bits that satisfies this, so the sum of these, sum of these, and sum of these is all 0, and you have another set of bits that satisfies this, then when you take the sum of these sets of bits, the sum of things within this circle is the sum of the things from the first code word in this circle, plus the sum of things in the second code word in this circle, and 0 plus 0 equals 0. So it's also a code. So we have this is a linear code.

By linear algebra, every code C see has a generating matrix G such that C is the row sums of G. So now I'm going to write down the Hamming code in a different way. And actually, you're not going to get quite the same-- well, I mean, I'm going to write down the Hamming code in a different way. And let's see, we have 1, 0, 0, 0.

So this is a first bit. So now the first two bits of the parity check matrix are both 1 here, so 1, 0, 0, 0, 1, 1, 0. So when you put these bits in the Venn diagram, they will be-- assuming I've done this right-- they will satisfy the parity checks. And then there's 0, 1, 0, 0. And that's just 0, 1, 1 and 0, 0, 1, 0. And this is going to be 1, 0, 1 and 0, 0, 0, 1, 1, 1, 1, because this fourth bit is the center region of the Venn diagram. And that means all three parity checks need to be 1 for them to satisfy that. So G is equal to this.

And there are 16 code words in C-- in C. And what do I want to say? And then the row sums of G and 0, 0, 0, 0, 0, 0 is in C. And actually, the all-zeros vector is a code word of every linear code, because we know that if c1 is in C and alpha is in F, then alpha c1 is in C. And if alpha is 0, then alpha c1 is the all-zeros vector. So every linear code has the all-zeros vector in it. OK, correct. Oh, OK.

We need some more definitions before I can continue. The Hamming weight of a vector-- and let me see if I can find what-- w of v is the number of non-- wait, I did this wrong. The Hamming weight w of v of a vector v is the number of non-zeros v contains. Let v equal 0, 1, 3, 2, 0, 4, 1, say.

The Hamming weight is the number of non-zeros-- with 1, 2, 3, 4, 5-- is equal to 5. The Hamming distance of v and w is the number of places where v and w differ.

So let's pick an easier example-- 0, 1, 3 and-- OK. I want to use the same notation for Hamming distances in the notes. So this is d sub H of v, w.

And 1, 1, 2 is-- well, how many places do they differ? Anybody?

AUDIENCE: 2.

PETER SHOR: 2. So call this c1 and c2. So the Hamming distance between c1, c2 is equal to 2. And it's easy to see that the Hamming weight of c1 minus c2 is equal to the Hamming distance between c1 and c2. Because if two bits are the same, then their difference is 0. And if two bits are different, then their difference is non-zero. So here, c1 minus c2 is equal to minus 1, 0, 1 has Hamming weight 2.

So theorem, a code can correct. Let d of a code be minimum over c1, c2 in the code c1 not equal to c2, the Hamming distance between c1 and c2. So it's the smallest Hamming distance between two non-equal vectors. Then the code can correct. t is equal to dH of c over 2 floor errors. And what's the proof? Well, let's assume-- I'm sorry, I wrote this formula down wrong-- d minus 1 over 2 errors.

Suppose it cannot, then, well, there's a received word, a received word c tilde such that, well, the two code words could have resulted in c tilde. So I mean, this is just saying that if you can correct every word with errors to a unique code word, assuming there are fewer than t errors, then the code can correct t errors.

So if it cannot correct t errors, there must be some word c tilde so that there are two code words which that are in Hamming distance t of C tilde. So t is equal to c tilde. Or actually, t is greater than or equal to the Hamming distance between c tilde and c1, and t is greater than or equal to the Hamming distance between c tilde and c2.

Then the distance between c1 and c2 must be less than or equal to the distance from c1 to c tilde, plus the distance between c tilde and c2. And this is just because Hamming distance is a metric, which means if the distance between c1 and c tilde is, say, t, and the distance between c tilde and c2 is t, then the distance between c1 and c2 is at most 2t.

But this is less than or equal to 2t by these inequalities. And that is less than or equal to, I guess, the Hamming distance for the code. But we know by the definition of the Hamming distance from the code that it's the minimum distance between two code words. So this is a contradiction. I should say less than here, because t was 1 less than the distance of code divided by 2. OK.

For a linear code, let d equal the minimum over all code words in the code of the Hamming weight. What did we use for Hamming weight? w-- OK, so w of c. And we have to specify that c is not equal to 0 here, so the minimum weight of any non-zero code word. Then c can correct d minus 1 over 2 errors.

Proof. Well, we're going to use this theorem. Might as well call this theorem 2. Proof. So the minimum over c1 and c2 in C of the distance between the two code words is equal to the minimum c in C, the weight of c. Because, well, why? Because the weight of c1 minus c2 is equal to the Hamming distance between c1 and c2. So for linear code c, you want c to be able to correct d minus 1 over 2 errors.

So the generating G for Hamming code was G equal 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, and 0, 0, 0, 1, 1, 1, 1. OK, I want to claim you can define a Hamming code with block length 2 to the k minus 1. n equals 2 to the n minus k minus 1.

So here for this Hamming code k, which is the number of bits it encodes is 4. And the block length is 7, which is 2 to the 7 minus 4. That's 8 minus 1 is 7. And how do we do that?

Well, we're going to do the same thing as we did before-- 1, 1, 1, 1, 1, 1, 1, 1. So that you see is-- and then, OK, so we put the identity matrix here. And now we're going to put all binary strings of length s with at least two 1's, so 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1. So I guess there are six here, then four, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1. So this is 11, and this is 15.

So here, k equals 11, n equals 15. k equals-- well, there are 2 to the s strings of length s total. And if we remove the ones with 1, 1 and the ones with 0, 1's, you get 2 to the s minus s minus 1.

And n has s more columns than-- I mean, so this identity matrix, has an equal number of rows and columns. So we have s more columns than k. So that's k plus s equals 2 to the s minus 1. So this is the 15, 11 Hamming code.

And now we have to explain why does it correct Why does it correct one error? Need to show all row sums, minimum weight 3. Are all non-zero row sums have minimum weight 3.

So let's start, start with easy case. All rows have minimum weight 3. Or rather, all rows have weight at least 3. Why is that? Well, let's pick any row. Let's say this one. The identity matrix has 1 bit in it, and the parity check bits have at least 2 bits in them. 1 bit, parity check bits, 2 bits.

Any sum of two rows-- well, let's take two rows and look at their sum. There are two different ones in the identity matrix part. So we need at least one 1 in the parity check bits. But we arrange these parity check bits so that they were all different. And when you take two non-different strings and you sum them mod 2, you don't get 0. You get at least 1.

So here, the identity matrix part has two bits, and the parity check, at least 1 bit. I should have said at least 2 bits there. And any sum of three rows or more, the identity part has at least three bits.

So the last thing I want to do today is tell you how to correct errors in linear codes, because.

AUDIENCE: I have a question.

PETER SHOR: Yeah.

AUDIENCE: Why is that k 2 to the third minus y minus 1?

PETER SHOR: OK. So we're looking at all binary strings of length s with at least two 1's. So how many binary strings of length s are there? There are 2 to the s binary strings of length s. And now we have to subtract out the ones that have 1, 1 or 0, 1's. There are s 1's with 1, 1, because the 1 can be in any of the s positions. And there's one string with 0, 1's, which is all-zero strings. So the number of binary strings with at least two 1's is 2 to the s minus s minus 1.

And then that's the number of rows in this matrix. To get the number of columns in this matrix, well, we just take the number of rows, because the identity matrix has an equal number of rows and columns, and add s. And that's 2 to the s minus 1.

So this is the class of Hamming codes. I should say something about them later. But let me first tell you how to correct errors in linear codes.

OK. Oh, wait, I want to pull this down. Well, to work with the code, need to encode a message, identify positions with error. And once you've identified the positions with the errors, you can fix them and then decode. And we defined generator matrices, generator matrix for code G equals code is set of all row sums.

To encode, use C equals m times G. So m is the message, c is the code word, and G is a generator matrix. So if m equals 1, 0, 0, 1, mG is the sum of the first row and the last row, because this is the first one and the last one. So it's 1, 0, 0, 1, 0, 0, 1. So encoding is easy.

To decode, we need the parity check matrix. H is defined as-- well, we want GH equals 0. And we want H to have maximum rank so that GH, H maximum rank with this condition.

OK, let's leave this up. And I will say something. For all codes, all linear codes, we can find a generator matrix G, G of the form 1, 1, 1, 1, 1, 1. So this is the identity matrix and some matrix here s. So you see, here the second part is s and this first part is the identity.

And by general linear algebra configurations proof, proof is really Gaussian elimination. You take the code and you do Gaussian elimination on it, and then you rearrange the order of the coordinates to get this identity. And I'm not going to go into further details, because they're not that important for this, and they're not in the notes either.

And I want to claim that here H is equal to minus s I, where you have a minus s here and an I of the right form. So H is equal to minus 1 minus 1, 0.

I feel stupid putting these minus 1's on because minus 1 equals 1-- there's another row here, minus 1, minus 1, minus 1, 1, 1, 1-- because minus 1 equals 1 in binary. So this really could be simplified to this. But for non-binary codes, you really need the minus s.

And you can see that-- oh, well, I will explain. And G equals I s. And note this I is a different size identity matrix than this I.

But anyway, GH is equal to I s minus s I. And when you do matrix multiplication here, you get I times minus s plus s times I equals I times minus s, plus s times I equals 0. So GH is 0, which was our condition for parity check matrix.

And you can also see that H is the maximum rank matrix such that GH equals 0, because, well, that's linear algebra 2. I mean, I want to say H transpose generates the perpendicular space to G. And the dimension of H transpose plus the dimension of G is equal to n, the dimension of the entire space.

And this is true for any linear spaces, even those over finite fields, even though G intersect H transpose might be non-empty. So I mean, you guys are used to thinking of linear algebra over the reals. And then you have a space s and a perpendicular space s perp. And s intersect s perp is just 0. This isn't true here.

Let's see. Look at the last row in G. It's perpendicular to itself. So the last row in G is also in H, in the row space of H transpose. Because everything that's perpendicular to all the guys in G is in H transpose. And you can see that the last row is perpendicular to all the other rows in G.

So there's some linear combination of columns in H that gives you that. And you can see it's the sum of all three columns, because you need 1, 1, 1 in the last three entries in the column. And then the sum of the first three is the sum of these four columns is 0, 0, 0, 1. So a space and its perpendicular space can intersect, but that doesn't matter. So this is the parity check matrix.

And now suppose we have a linear code with G and H are the generators matrix and the parity check matrix. Encode, we get mG error. We have mG plus e.

And the next step is compute the syndrome of c tilde, and that's c tilde times H. This is mG plus e times H is equal to mGH plus eH, which is just eH. Because remember, G times H was 0.

And this is called syndrome. And it's called the syndrome because it's what you use to diagnose the error. So syndrome depends only on the error and not on the encoded code word, not on the message. OK.

How do you go from s to e? Well, it's not easy. Depends on the code. There is no efficient algorithm for taking a syndrome for an arbitrary code and computing the error. You have to do something that is codependent, and we're not going to-- I mean, there could be lots and lots of different ways going from the syndrome to the error, depending on the code. And some of them can be quite complicated. Some of them are easy. For Hamming codes, it's very easy.

Code, assume error has 1 bit, or Hamming weight 1. So 1 bit equal 1, and all the others are 0-- 0, 0, 0, 1, 0, 0, 0. Let's make this the kth bit times the Hamming matrix is kth-- not Hamming matrix-- times the parity check matrix. So if you have 1, if you have a vector with 1 and you multiply it by the matrix, you just get the kth row of the matrix.

So because the Hamming code can only correct 1 bit errors, we only need to worry about the case where the error e has 1 bit in it. And that case is easy to decode. So let's do an example, because I think we have just enough-- well, we probably have more than enough time for an example, but we should do an example.

Encode 1, 1, 1, 0 in our 7-bit Hamming code. So there is our 7-bit Hamming code up there. We sum the first three rows, because we have the first three entries of our message are 1. m equals 1, 1, 1, so that's, well, it's 1, 1, 1, 0, 0, 0, 0.

Add error, and you get, let's say, e equals 0, 0, 0, 0, 0 1, 0. We get 1, 1, 1, 0, 0, 1, 0. So now we have mG plus e times H. So that's the sum of the first three rows of H. And that's zeros. And then add the fifth row of H, that's 0, 1, 0 equals 0, 1, 0.

And this is the syndrome. 0, 1, 0 is the sixth row of H. I think I said fifth here, but I meant sixth-- the sixth row of H, error in position 6, 0, 0, 0, 0, 0, 1, 0. And this was c tilde. So c tilde plus e is equal to 1, 1, 1, 0, 0, 0, 0. And this first bit is the message.

Now there's one more section in the notes about Hamming codes. I don't want to get into Reed-Solomon codes this time, because I want to do them all next time. And that's about perfect codes.

A perfect code is one all of whose syndromes correspond to an error of t bits or fewer. If it's a t error-correcting perfect code, all of the syndromes correspond. So the binary, the Hamming codes are perfect codes.

So how many syndromes do they have? Well, their syndromes are length s bits. So there's 2 to the s syndromes. And the code has 2 to the s minus 1 bits.

So I want to say, first, the zero syndrome corresponds to no error. And every other one of these syndromes corresponds to exactly 1 bit. So there's exactly as many syndromes as bits. And you can ask, are there any other perfect codes?

So to have a perfect code, you need all the possible errors. So let's say L equals 0 through t. How many ways are there of getting an error of exactly L bits? It's n choose L is equal to 2 to the something to the number of bits in the syndrome.

So this is a condition you need for perfect codes. And it's satisfied. Or this is a solution you need for binary perfect codes, for Hamming codes. And it's also satisfied, if I'm remembering these numbers right, n equals 23 and t equals 3. So there's exactly one solution, other than the one you get for Hamming codes.

And remarkably, maybe it's remarkably, this is something called the Golay code. And I should have looked this up before coming to class, but I didn't get around to it. This was discovered by Golay sometime after Hamming's paper. But the construction had already been found by a football-- although you should call that soccer-- enthusiast who I think was finished in coming up with a betting system for soccer games-- football games, I should say.

So this is actually a remarkable mathematical construction, which is related to finite simple groups. But I don't have time, I mean, we don't have the mathematics to go into that, and I certainly don't have time to go into that. But this is the one other binary perfect code.

There's also a code over z sub 3 called the ternary-Golay code, where you have a 3 to the s here. And, I guess, a 3 to the something on this one as well, 3 to the L, I think. But anyway, if you want to use perfect code-- I mean, so at first, after discovering the Hamming code, you would say, well, perfect codes are exactly what we want to use, but there aren't enough of them.