PROFESSOR: Welcome to class again. This time it's not Professor Forney it's me, so my name is Ralf Koetter. You guys have some substantial chalk here at MIT. And I'm visiting here from the University of Illinois, so Professor Forney thought I could teach this class here. All right, let's see. So I understand that last time, last Wednesday, you went through all the finite field stuff, meaning, so you know what that would mean, the finite field. There's p elements, p to the m elements. Whatever q you have here, is a power of a prime in order to be a field. So this one, as a notation, is a ring of polynomials. You've seen that too.

So I assume you know everything about finite fields that you will need to know here, at least, except for one more theorem which Professor Forney told me he did not cover. And this is the fundamental theorem of algebra. I have to write a little bit smaller with this thing here, otherwise I'll run out.

AUDIENCE: [UNINTELLIGIBLE PHRASE]

PROFESSOR: Oh, I know, that's probably better. Better. With the algebra, at least that's what it's often called, and really, about 60 percent of all the proofs in algebra eventually boil down to this here. And what it says is, polynomial of degree m, f beta equals zero, at most, m of beta. At least, that's one way to formulate it. Let me see. So that's fine. So what it says is a polynomial of degree m has at most m roots. Once you all have seen that, probably one way or another, but because of its importance, I want to emphasize it once more.

Do we need a proof of this? In true MIT spirit we do. And the proof would go something like this. You look at problem number one in your homework assignment, and from problem number one, I could prove that here, too, but since it's in the homework, I won't.

You can write the following given any beta. Write f of x as f of x is equal to plus alpha. Alphas are the field so that's by some sort of long division you get to that. That's what I'm not going to prove. Then f of beta is equal zero is the same as

saying that alpha is equal to zero. So if either is a root of the polynomial, zero, it follows that f of x has this thing here as a factor, this h of x, x minus beta, where because of the degree properties of polynomials, h of x is m minus 1. And so the rest follows by induction.

So basically, then we can prove that this polynomial has, at most, m minus 1 roots, and so on. And you can descend this route, and so the rest follows by induction. In particular we can say if f of x has m distinct roots beta one, beta m, then it factors completely into the linear factors like this. So I just wanted to quickly state the fundamental theorem of algebra, since we need it in a proof later on, and I think you didn't go through it.

OK. So last time, you learned everything about fields, finite fields, extension fields, so chapter eight is pretty much what we have to cover now. What is the whole idea of chapter eight? It's linear codes, codes, MDS codes, and redundant codes. Oh, by the way, do you have any questions about this here? That in any way? It's pretty straight, right?

OK, so I understand in chapter six or so, you had already linear codes over the binary fields. So let's just define codes over a larger field, formally, a linear code C of length n. No subspace of Fn. So whatever the field is. So F could be any extension field, could be the binary field, so it really generalizes a definition of code, of what a linear code is.

OK, so it's a subspace. What can be derived from that? Since it's a subspace, it's a group. And then we can derive minimum distance properties. So let's first define it again, since it's slightly different than the definition for binary codes. Between Fn, say Fqn. So I denote the vectors with an underscore. I think in the notes, it's boldface notation, so translate that online as I go here. The distance between two words x and y, given as dx, the number of positions that x_i is unequal to y_i.

AUDIENCE: What's the subscript?

PROFESSOR: There, a q. Oh, this is another thing I should warn you about. My handwriting is

bound to deteriorate during class. So I usually start out reasonably okay, towards the end of the class it's -- I tell my students to throw little pieces of chalk at me when it gets too bad and I'm not facing them, so please just say something if it gets too bad.

So distance is defined as that, quickly. So it doesn't really matter what the values are here. The x_i and the y_i could assume different values. It's a somewhat coarse measure for the real, the difference between code words, or difference between words. Why do you think I say it's coarse? In digital communications in particular? Good question, right? In the end, we want to map that into a modulation scheme. In the end, we want to map our codes that we are deriving here into modulation schemes. In the end, we want to embed them into some Euclidean space. Now, different elements of our alphabet we will map to different elements in Euclidean space.

So basically, approximating their distance relation in Euclidean space, which we are really interested in with the Hamming distance here is pretty coarse, but we can do, so we do that. It's an approximation, at least. That clear? All set? All right.

AUDIENCE: [UNINTELLIGIBLE PHRASE] the Hamming distance [UNINTELLIGIBLE] the same as the Euclidean distance?

PROFESSOR: Well, it depends on the modulation scheme. It very much depends on the modulation scheme. If you have a 8-PSK scheme, where you would label, put in the words, here, with three bit symbols, or with the symbol from F8, then it's definitely different. It's definitely different. So if you do anti-polar signaling, then it's directly reflected. OK, I'm starting to digress already.

So just for completeness, minimum distance, minimum Hamming, of a code subset Fqn is d as a minimum code of dxy, and they have to be different, it's the same as before. So now if I claim that -- so the minimum distance of a code is also given by the minimum between 0 and x in the code 0 and x, and this is minimum of the Hamming weight of x, and you could do 0 x in the code. So that's all old stuff. I just write it down so we get started here.

Is that clear, from the group property, why this would be true? So if you just take this, we can add basically x to both x and y, just translating the whole relation to somewhere else. So in particular, we translate it here, once we have it here, than the distance between 0 and x is just the weight.

OK. So far, so good. Now what is next? Generate a matrix. This is not really in the notes, but I think it's important. So see, the code here is a subspace. It's a linear space, so it has a generator, it has generators, k generators. So let g1 be k, write this off the code. So as a basis of the vector space, that this would be a basis of the vector space, any basis would be fine here. Then C may be defined as all the x in Fqn such that x is sum over -- what do I call it -- fi gi, where fi is in Fq.

And the reason I introduce this, we can -- this is just the definition of a space, right? That's clear. So if you have these generators, you find a generator matrix, uh-oh, it already starts. Let me -- matrix g which contains, as a m matrix containing the rows gi. So the i-th row in the generator matrix is just gi. Then you also can write as x is equal to f times g, f element Fqk, or just the same statement as this one, so nothing has happened.

So basically, the reason I did that, I wanted to introduce the term generator matrix, which is sort of important. And one more property of this orthogonal complement of C, of the code. So what does that mean? So the orthogonal complement of the code you could write as Fqn such that sum of x_i y_i is equal to 0. The sum is obviously over the field for all y in the code. What's the dimension of this, of the orthogonal complement?

AUDIENCE: n minus k.

PROFESSOR: n minus k, clearly, because we have ambient space is n dimensional, we impose k linear constraints on this, by the k generators, so the k dimensions of, take note, by the generators drop out. So the dimension of the orthogonal complement is n minus k.

So what else do we need to say about this? C is called the dual code for this

reason. C is called dual code. In particular, it's a code that's a linear space. It's a subspace of Fqn again, it's a code. So it's just as nice a code as C at this point in time at least. So it's called a dual code. To C, if it is a code, it has a generator matrix.

Let h be a generator matrix for C dual. So in particular, we could define C dual now, for example, by the equivalent of this relation here. But because it's a dual code, we now also can define the original code in an equivalent way such that x times h transpose is 0. We could define our original code C either as the image of a matrix g, of a generator matrix g, or as a kernel of a parity-check matrix h. So h is a ...WRITING ON BOARD... for C. So that's all pretty much straight linear algebra, and I'm sure you've seen that in many different places. Any questions about any of this?

AUDIENCE: So the addition of the dual [UNINTELLIGIBLE PHRASE] the summation [UNINTELLIGIBLE PHRASE] equals 0 for all [INAUDIBLE] other than x, right? [UNINTELLIGIBLE PHRASE]

PROFESSOR: Oh no, no, no, it doesn't have to be different from x. If y is in the code, if y is in C, then x has to be orthogonal to it. They can be the same vector, in particular, if you have binary vectors, an even made binary vector is orthogonal to itself. It's a little bit odd, but that's the magic of finite fields. OK. Good. So these are codes, now we could stop. We have defined the object, and obviously it exists, because we could just write something down and it exists. So once we have defined it, the next question is, what sort of codes do exist? So that's what we're going to do next.

First, question one. Codes do, what type of codes do exist? So which codes do you know?

AUDIENCE: [INAUDIBLE]

PROFESSOR: You know Reed-Muller codes, you know probably sporadic binary codes that are out there. These are all binary codes. So what type of codes exist over larger fields? Many, many classes. There exists the equivalent of the Reed-Muller codes, there exist QRE Reed-Muller codes, and there exist generalized Reed-Muller codes, and,

and, and, and, and. But we are interested in a very special class today, which is MDS codes. It stands for Maximum Distance Separable. It's a strange name. There's no particular reason for MDS. But, let's see what we can do with that.

What type of codes do exist? So we have parameters of codes -- oh, I think you write the curly bracket, right -- n, k and d. So that would mean a code of length n, dimension k, and distance d. And let me add something to it a q, if you want to emphasize that this is a query field. So are all numbers possible here? What do we have, a 20, 19, 17 code over, I don't know, over F8. Is this possible? What would you think? No?

AUDIENCE: [INAUDIBLE]

PROFESSOR: It's not possible. It doesn't seem likely. What conflicts here, is the dimension and the distance. If you get a large dimension, in particular, if we would make this 20, what would that mean? It would mean we have to take the entire space. If you take the entire space, then the minimum weight word is 1. So this is possible. You know this is possible. If you drop this by 1, that seems very unlikely that we would get a 17 here.

But what do we get here? 2. You get a 2 because that's what we can achieve with a single parity-check code. The parity-check code doesn't have to be restrained to binary. Why? Why would it be restrained to binary? You could just, the set of all vectors let's define the single parity-check codes s p c, q, as the set of all vectors such that sum of the x_i is equal to 0. So could we have a word of weight 1 in here? Obviously not, right? If it has a weight 1, how would it add up to 0? Because one position would never cancel with any other position. So the minimum weight is 2 here, and we get a distance of 2.

So what's the next one? It's tempting to say 3, right? 3, but this is very much a question, now. Because this is not as easy to come by as a single parity-check. And that's what we're going to do next. We're going to define bounds on the maximum distance that a code can have altogether.

OK, so let's do the following. Which parameter is possible? OK. So let's assume you have a code, an n,k,d code, and now we want to find a relation, a bound between n, k and d. How do we do this? Any ideas? Let a computer run for eternity and find all possible codes? No, no, no, no. We don't do this. We wouldn't get far.

Let's assume we have an n,k,d code. What does that mean? Well, let's write the code words all down in a huge matrix, so each row in this matrix corresponds to one code word. So this has a length n, this is q to the k, q is whatever the alphabet is of the code in question, and now we say it's an n,k,d code. What that means, it means, among other things is, say we delete, just punch out, d minus 1 positions. We punch out d minus 1 positions of all code words and we look at the code that remains. You guys don't have colored chalk here, huh? We look at the code that remains, it means we look at this part of the matrix. Is that clear, what I'm doing here?

So if the code indeed had distance d, can there be any two rows equal in this part? Remember, we punch out all d minus 1. Can there be any rows in this part that are equal? No, right? Couldn't be. They all have to be different. What does that mean? They all have to be different, but how many different tuples can we have in this part? Well, we have at most q to the n minus d minus 1. That's the length here. This n minus d minus 1. Different tuples.

So how can we patch that together into a relation on the parameters? It basically says, q, this is q to the k. q to the k is upper bounded by this. It's upper bounded by this. And let me take the logarithm on here, and we get this relation.

That's a first incarnation of the tension that we get on code construction, on codes. And bound on this, at least. If you choose d large, the distance large, then k has to go. If you choose k large, the distance cannot be very large. So this is where we, for the first time, see this tension. And it's also important, I'm sorry that I run around like this here, n has to be at least k plus d, k plus d minus 1. So here, you see this 28 in 3, it would just satisfy this. It would just satisfy this.

So do we know it exists? No. No, why would it? So far, we only have looked at this here, and so, if it would exist, it would have to satisfy that. But there's no reason to

assume it exists. At the moment, at least. OK. This is called the Singleton bound. Any code over any field, phi n, this relationship on the parameters. Good. Any code satisfying and bound with equality is called MDS. So we have an MDS code if and only if it satisfies the Singleton bound with equality. That's the definition of MDS codes. And now it makes maybe a little bit more sense to talk about Maximum Distance Separable codes, well, in a sense, they have the maximum distance among all codes. You find all codes with the given n and k, if they're MDS, they have the maximum distance.

OK, let's think about this a little here.

AUDIENCE: [INAUDIBLE] dependence on q [UNINTELLIGIBLE]?

PROFESSOR: Yeah, there's a very strong dependence on q. The bound, not. The bound has no dependence on q. If the guys exist or not, that's very much dependent on q. We'll get to that.

AUDIENCE: [INAUDIBLE] when the q is large, we have more options to [UNINTELLIGIBLE]?

PROFESSOR: Absolutely. Absolutely. For binary, there's a very simple argument to show that there are no binary MDS codes except for the parity-check codes and the repetition codes and trivial code. So say we have a binary code, a binary n,k,d code with a generator matrix. So what could the generator matrix be? There will be an identity part, and then there will be the rest of the generator matrix, and how could we possibly fill that in, in order to make it MDS? Because this is n, this is k, and we see in order to make it MDS, every single row has to have all entries equal to 1. Because if not all entries are equal to 1, here, then we immediately have exhibited a code word with a weight less than n minus k plus 1.

So OK, we know the first row has to have all 1's. Because now, the weight of this row is exactly on the MDS [INAUDIBLE]. What about the next one? The next one, same thing. All entries have to be 1. But now we see the problem, right? Now we add those two guys, it should again be a code word, and we have a grade two code word.

So this is, in a nutshell, to prove that there are no binary MDS codes except the trivial ones. So the trivial ones are n, n1, n, n minus 1, 2 and n1, n. These are the trivial ones. The space itself, so it's in a parity-check code, and the repetition code. These are the only binary MDS codes. And the argument is roughly there.

OK, where was I? Yeah, let's think about this a little bit more. And we are getting to exactly your question about the [UNINTELLIGIBLE]. This here has to hold, this argument has to hold regardless of which d minus 1 positions we punch out. This argument has always to hold. Which means, think about it, it's an enormously strong combinatorial condition on the code.

So you have a code, that means you have a code, you write it in a matrix like this, all the code words. You punch out an arbitrary collection of d minus 1one positions, and the rest, the remaining positions, have to make up the entire space here. The entire space Fqn minus to that right exponent. This is a very -- think about it, I mean, just writing down this is an enormously strong combinatorial condition. So that will actually lead to the codes existing only for a very, very special, for a subset of field sizes. In particular, like you said, we have to have enough freedom in the field size to fill up this matrix to satisfy this.

OK, before we get to that, before I say a word about the field size, let me formalize what I just said here, namely, that all of the other positions have to be exactly the q to the n minus d minus 1, different tuples. And the definition, let the code with q to the k, code words over alphabet Fq. Let subset of the positions in C, i is called an information set if C constrained to i runs exactly through all the q to the k, runs through all the q to the k. Fqk.

So what it means is, you have a code, and you have a subset of positions, maybe this one, this one, this one, this one. This is a subset of positions if the code words. So if the matrix that remains after you take out the punctured columns, runs through all the q to the k elements of Fqk, then this is an information search.

AUDIENCE: [INAUDIBLE]

PROFESSOR: Constrained to i, because i has size k. i is just -- its just about enough to describe every code word, if the restraint of C to the set would indeed be giving a unique vector for each code word. The reason to call it -- so this is the definition of information set. The reason to call it an information set, it's pretty straight, right? Why is it called an information set? Because it's enough, right? Because it's enough.

If you know exactly the value of a code word in these positions, then it is enough to recover the entire code word. When some genie tells you, gives you a code world which was corrupted by noise or something, but tells you, these k positions are OK. That's enough, that's all you need. That's an information set. You can recover the information from them.

Actually, it is an application that pops up sometimes. That somehow, you get side information about some positions in the code word indeed being correct, and others not. And others you don't know about. So that's the information set, with respect to our MDS code. So with respect to our MDS code, a corollary of the thing that involved any k positions in an MDS code, an information set. So any k positions on information set. It's a really strong property. Really strong combinatorial property. OK, so far, so good.

This is so strong, this property, that we can say something about these codes even without even knowing if they exist. So, so far, we have talked about these codes as if we knew they existed. Well, it's not entirely trivial, since we know those guys here exist. So it's not entirely empty, we're not out in cuckoo space, here. But do any other one exist, except for those? That's the question. We don't know that yet. We will show in a little while that they do, but we don't know that yet. But the interesting part is that we can derive properties of those codes without even knowing they exist. And how do we do that?

For example, we want to derive the following property, how many words of weight d exists in linear MDS code? One could ask that, right? If they exist, they're nice, and if they exist, we also want to know how many words do exist at minimum distance. Because that translates, again, directly into union bound arguments later on, and

probability of error. So that's a good question. How many words of weight d exist in a MDS code? Let's call this n d, and we want to know how many. So I'll let you think about this for a sec while I erase the board, and then somebody will tell me the answer.

So how can we think about this? Let's try to do a similar argument as this one. Let's look at a single word, and let's assume that d positions, we ask the questions does there exist a code word within the first d positions? It is equivalent to the question, does there exist a code word that covers exactly all d positions? Any set of d positions.

AUDIENCE: 0 everywhere else?

PROFESSOR: And 0 everywhere else. Why is that nice? If you could prove that, that there exists a word for all d positions, because then, we pretty much know what happens. Then we know that, well, if this is true, then there are n choose d ways to choose those d positions. And then within those d positions, and since it's a linear code, we can multiply with the q minus 1 on the repeated element. So if we can choose our d positions arbitrarily, then this is the number over words at distance d.

So let's look at a word, and let's, without loss of generality, assume it's a first d positions. So the first d positions. So in particular, these would be the first d minus 1 positions, which would mean that this have length k. So if we have an MDS code, this is an information set. So if this is an information set, then we can fill up this thing with just about anything we want. So we choose this information set to be equal to 1. This is how we choose this information set, and by the property of MDS code, we are guaranteed that there exists a code word which is this part in the information set.

But we also are guaranteed it's a weight d word, right? The minimum distance is d, that means all of these m entries here, they must all be non-zero, in this part. Otherwise, it wouldn't have weight d. OK? And there we have it. That was all we needed to show. Right? Because now we have shown that there exists a word of weight d in the first d positions. Is that clear?

Let's try again. I will say the same words. Maybe it becomes clearer by that. Let's look at a code word. This is a generic code word, at first, and we want to answer the question, does there exist a code word within, which has support only in the first d positions? So does there exist a code word which is non-zero here, up to d, and which is zero everywhere else? That's the question we want to answer.

OK. Now here's what we do. We look at this road and say, you know what? Let's look at the last k positions, which have an overlap of 1 with this word here, because it's an MDS property. So we have this relation between n, k, and d. And since any k positions in the word are in information set, so we can choose whatever we want in this part, and this is what we choose. By the property of MDS codes, this corollary, we are guaranteed there exists the code word which in the second half of the code word looks like this.

And in the first half, it looks different. There's something else here, and I say, well it cannot have any 0 in here, because then it would have weighed less that d. So it has non-zeros here. So indeed, we have shown the existence of a code word which has non-zeroes in the first d positions. Very simple. And that was without loss of generality. You could make the same argument for any d positions.

What have we shown? We have shown that indeed, we can choose any d positions in the code to support the minimum weight code word of weight d. This is how many ways we can choose this, then we have to multiply it with q minus 1. All non-zero field elements. The reason is, we might have chosen this, or we might have chosen omega or omega squared here, or just the multiples, the scalar multiples of it.

Interesting, right? This property, that any k positions on information set is really strong enough to prove the -- actually, it's strong enough to prove the entire weight distribution of an MDS code.

AUDIENCE: [INAUDIBLE] [UNINTELLIGIBLE]?

PROFESSOR: No, no, no, why no, no, no, no. So then you would get too much. If you write q minus 1 to the d, then you would want to multiply each position with a different

value. That would imply that there's more than one code word in the first d positions. More than one code word so that they are not scalar multiples of each other. If that would be true, then you could find a linear combination which is still 0 in this part, but has additional 0 here somewhere. But if that is true, then we don't have enough distance anymore. Then it's not an MDS code.

All right, so it's indeed q minus 1. Within each d positions, we have one dimensional space. It's just one dimension.

AUDIENCE: [INAUDIBLE] far off minimum weight code words?

PROFESSOR: Yeah, yeah, definitely. Any other code must have less, so it would have less. But every other code would have a smaller minimum distance.

AUDIENCE: [INAUDIBLE]. Suppose we let the last k minus 1 position zero, and the one before that, [UNINTELLIGIBLE PHRASE]. And you said that we can do it for any of [UNINTELLIGIBLE] total field?

PROFESSOR: Sure.

AUDIENCE: Since it's a linear code, some of those code words should be in the linear code, right?

PROFESSOR: Sure.

AUDIENCE: So because it's a field, also we are going to [INAUDIBLE] there exists an inverse [UNINTELLIGIBLE]?

PROFESSOR: Absolutely.

AUDIENCE: So if we add those two code words, we should have all zero, [UNINTELLIGIBLE] k minus 1, and have inverse at the one before. We get that code word which has a minimum weight, which is less that the one we have here?

PROFESSOR: Good question. There is a trick out. There's a way out of this. Great argument. But there's a trick out.

AUDIENCE: There's gotta be an upper bound

PROFESSOR: No, no, there is a trick, there is a way out here. Namely, so let's put it like this. Right here we put in a 1, just for simplicity, let's assume all the other positions are also 1. And then you say, this would be another code word, which has here an omega. I say, you know what? What's going to happen? All the other positions are going to be omega 2. There's no way to combine these two guys to get an additional 0, unless you get all 0's. Unless you get to 0. That's what I said, it's a one-dimensional space in these positions. When it's a soft code .

AUDIENCE: [UNINTELLIGIBLE PHRASE]

PROFESSOR: It tells you that if you write down the minimum weight code words in the q minus 1 times d matrix, that is, you have a Latin square, basically. That's what it tells you. There's in no position, if you have anywhere in here an element alpha and element omega, the same omega pops up nowhere else. There's ramifications of MDS codes in combinatorics left and right, so this would be a Latin square. You know, you can learn a lot a lot about MDS codes if you think a little bit about that, and about combinatorics altogether. OK, where was I? So we know that's fun. And actually, in the homework, you going to do n d plus 1. So the next one. But once you do n d plus 1, do all of them. In a sense it's just inclusion and exclusion from then on. The first one is sort of the toughest one, the rest is inclusion exclusion. And just for the heck of it, when you go home and do the homework, write them all out. It's a pretty looking formula, in the end.

OK, so far, so good. So we have still talked about MDS codes without knowing if they exist. Except for the trivial ones here. And the existence of MDS codes is actually not known for which parameters they exist. So I give you a research problem. The research problem is the main conjecture on MDS codes. And it's always sort of tricky. When a research problem has a name, then that signifies danger. Then it means that it's not trivial.

The question is, for which k d and q, for which sets of parameters n k d q, do MDS codes exist? And the conjecture this is that n k q, because in MDS code we can

actually get rid of the d here. e, the longest length of an MDS code. The longest length of an MDS code, I mentioned k over an alphabet of size q. The conjecture is that n, k, d is less than q plus 1 for k at least 2, unless -- I always have to look that up -- I think 2q. And k plus 1 for k greater than q. We talk about it in a second, except that n, there's a 3, 2 to the s. So if the alphabet is a power of 2, alphabet size an extension field of 2, basically. q plus 2 and q minus 1 q to the s. q plus 2. OK, so this is the main conjecture on MDS codes. Basically, it says that the length can essentially be as large as the alphabet size, but not larger.

AUDIENCE: [INAUDIBLE]

PROFESSOR: This q, yeah? Oh, yeah, n k q, sorry. It doesn't make sense otherwise.

So the lengths can be in the same order of magnitude as the alphabet size. That gives enough room, enough choices, to fill up this matrix with the information set, with the MDS property on the information sets. This is the parity-check code, this row is just taken out as a trivial code. And then, the demon of mathematics conspired that this would also be true. So if you have an extension field of 2, and you want to give it a dimension three, MDS code, they exist for q plus 2. Right. There are, of course, reasons for this, but they go pretty deep, why they exist for those parameters, and this is just mysterious. One can give reasons, so on another hand, it's just so, right.

There are exceptionally enough that they have names. The first one is the Hexacode, it's something with a generator matrix, and this goes over F4. So that's an MDS code of length six, so this is a n6, 3, 4, MDS code over alphabet size 4. That's the first one, in that sequence here. Anyway.

Otherwise, we have this conjecture. If you solve this, you are going to be rich and famous, you're going to live in Hollywood, and maybe, maybe not. But you're going to be probably not rich, you're going to be famous about a couple of hundred people who know about this MDS conjecture, but very smart people have been looking for this for a long, long time. OK.

All right, 20 minutes left. So it's better we define, we make sure those codes exist. Do we have any question about this MDS conjecture? OK, last 20 minutes, let's at least make sure those things exist. Reed-Solomon codes. So Reed-Solomon codes cover this case. They are examples of codes which lie, which satisfy this equality.

OK, so how do we define Reed-Solomon codes? Now, just in a true mathematician spirit, write down consider the following. Consider the following code. See? Beta 0 beta q minus 1. The beta i are the distinct field elements, the distinct elements in the finite field. f is a polynomial. f is a polynomial, and the degree is less than k.

OK, good. So we have defined a code. So what that means is we start from polynomials. The set of all polynomials of degree at most k. So what do we know about that set? It's a vector space, right? The set of all polynomials of degree at most k. We can add them to get a polynomial of degree at most k, we can multiply them with a scalar to get a polynomial with degree at most k. It's a vector space.

So we take this vector space and evaluate for any element in that vector space. This element in all non-zero elements of the field and we get a code. We get a set of vectors, so we get a set of vectors, and that --

AUDIENCE: [INAUDIBLE]

PROFESSOR: Yeah, I took all elements. Why not? Why not all elements? Strictly speaking, I should have taken one more in order to get the one here. We can talk about that in a sec. But this one more element would be -- so it's a code. First of all, it's a code. Right? We all see it's a code.

And once you see it's a code, we ask, what are the parameters? The parameters. So length, length is the easy one. Well, it's q. What is dimension? Dimension of C. What's the dimension? It's a little bit tricky, that question. I actually, at Illinois, we have to take a class on teaching. How to become an effective teacher. And one of the things they told us is that if you ask a question, you have to wait for 12 seconds to get an answer. So what's the dimension? There you go.

This mapping, this mapping from a vector space to a vector space. This mapping,

also called evaluation map, is a linear map. It's a linear map, meaning that, well, let's start differently. Let's start differently. Do any two polynomials map to the same code word? That you know. That you cannot. Are there any two codes, two polynomials, so are there f of x, g of x, such that f of beta 0, so that they coincide in all positions? No, then they would be the same, right?

And the reason is because if there would be something like that, then you could just look at h is f of x minus g of x, which is just another polynomial of degree k. And this would have to vanish in all positions. If k is less than q, it could not possibly vanish in all positions, because then the polynomial of degree k would vanish in more than k positions. Fundamental theorem of algebra. The very beginning.

So the dimension of C is indeed k, the same as the dimension of this vector space. The dimension of the vector space of polynomials of the degree k minus 1. And the distance, if k is less than q, the distance is equal to q. The distance, what is it? Same argument, roughly the same argument. I think that's a linear code, so if it's a linear code, the minimum distance of the code is the same as the minimum weight of a non-zero word. What's the minimum weight of a non-zero word? These are polynomials of degree k minus 1. What's the minimum weight of a non-zero word?

Well, we start out with the weight 1, and whenever the polynomial evaluates to 0, one of the weights drops out. So I claim the minimum distance as the minimum weight, weight of the non-zero word, and this is n minus, well, if any of these polynomials vanishes in all, it vanishes in at most, k minus 1 positions. At most, k minus 1 of these vectors here, of these entries, is equal to 0. So it drops by, at most, k minus 1. Drops by at most, k minus 1.

And there we have it. There we have it. There we have, oh, this is q. There we have it. There we have that the minimum distance of the code satisfies this equation.

AUDIENCE: [INAUDIBLE]

PROFESSOR: What?

AUDIENCE: The dimension?

PROFESSOR: The dimension. So, it's the same argument, roughly. So I say, the dimension, so let's just say the size of the code is q to the k. When is the size of the q to the k, if no two elements in the space evaluate to the same code word? But if two of them would evaluate to the same code word, then we would less size than the vector space had. But if two of them evaluate to the same code word, that means this is true for all four positions. Then we could define a polynomial h of the degree k minus 1. which disappears in more than k minus 1 positions. I mean, all positions. Cannot be, hence the size of the code is q to the k, so this is a linear map, dimension is k.

OK. So cool. So we have it, right? We have our MDS codes. They exist. Here they are. They are Reed-Solomon codes. Not all MDS codes are Reed-Solomon codes, but the ones we are interested in, they are.

AUDIENCE: [INAUDIBLE]

PROFESSOR: Well, the distance is at least this, but the MDS bounds is at most this, so it's equal to this. But the MDS bounds, so the MDS bound has this is. So with that. So it's indeed, they lie exactly bang on to this. There are MDS codes, Reed-Solomon codes. So that is good. So we know what they are.

So incidentally, where do you think this one more point is that you would evaluate our polynomials in? You've heard about projective geometries? There's one more point, it's infinity. You have, basically, if you look at the numbers, in order to close it up, you want to add infinity to that, too. In order to get this one more, this one addition in length, you want to evaluate this also at infinity. You will have opportunity to do that in the homework. I looked at the homework and I was pleased to see this problem there. I hope you will be pleased, too.

OK, all right. Any questions about this? Let's see what else I wanted to say. Because it just gives me a few minutes to talk about a few properties of Reed-Solomon codes, a few properties of Reed-Solomon codes. And what did I want to say there? On nested codes, so an RS code with parameters n k, maybe we define

them [UNINTELLIGIBLE] like this. q is properly contained, k minus 1, minus 1. This is pretty straight from the definition of RS codes. The set of polynomials of degree at most k minus 1 contains the set of polynomials of degree at most k minus 1. So they are nested codes, property one. You will see this is important, that they are nested codes, for various constructions where Reed-Solomon codes take part in later on.

A punctured RS code is again an MDS code. Why is that so? Why is that so? Well, you see it? Say if you puncture a Reed-Solomon code. That means we just choose to not evaluate our code in this, this position. And this field element. Well, we just drop that coordinate. Does anything change in the arguments we have made? Well, the length is now 1 less, the dimension, well, the dimension is still the same, as long as k is not larger than the length of the code. The distance, still the same as the length, the distance is at least the length minus the number of 0's. So that equation still holds.

Well, but that's all we needed. Still MDS code. So there was really no -- it was not important. It was not important if you took all field elements, or a subset of the field elements with MDS property. That has nothing to do with it. In particular, we often in the end, we often will drop the 0 element. We often choose not to evaluate these polynomials in the 0 of the field.

A punctured Reed-Solomon code is an MDS code. So what else did I want to say about this? What else did I want to say about this? A generator matrix. How would a generator matrix look like? Yeah, how would it look like? Basically, we can come from here, right? We can take the generators of that space. So basically, we say that one -- generate the set of polynomials, that vector space of polynomials with -- so this is the basis of that vector space. So if we map that basis, then we get a basis of the image of the mapping. And the mapping of that basis would give this. So we evaluate the function 1 in all field elements -- gives us 1.

We evaluate the function x in all field elements. This gives us the next generator of the Reed-Solomon code. Well, 0 gives 0, 1 gives, oh, let's write like this. We evaluate it in all field elements. These are all the field elements. The next one, and

this goes up to beta -- OK, so this would be a generator matrix. That's fine. So now, in order to make things a bit more interesting, do you have to stop five minutes early? We just started five minutes late? OK then, I think that's over. I think it's over.

One more thing for you guys to think about until you reach home, then the rest you do next time. So let beta 0 be equal to 0 beta 1, or beta i equal to omega i minus 1 where omega is primitive in the field. Then we can write the matrix v of omega. I tend to see that the first k columns, the first k rows of this matrix would be a generator matrix of a Reed-Solomon code. Of course it's the same as [UNINTELLIGIBLE].

If we now delete the first position, we erase the first, we puncture the first position all out, and we look at the rest of the matrix. This factor of the matrix. Does this remind anybody of anything? It's a DFT, it's a Fourier transform. And that's what we start with next time. So think about why this is a Fourier transform. And maybe that's a nice analogy. So we get the distance. The distance is at least something, which means it's not impulsive. It's not a single 1 somewhere. The vector that we get is not impulsive. Maybe it has something to do with the bandwidth constraint and the frequency domain. That's what you have to think about on the way home, and that's it. Thanks so much.