Problem 6, EGMO 2013 and coding theory

Zsuzsa Baran

Snow White and the Seven Dwarves are living in their house in the forest. On each of 16 *consecutive days, some of the dwarves worked in the diamond mine while the remaining dwarves collected berries in the forest. No dwarf performed both types of work on the same day. On any two different (not necessarily consecutive) days, at least three dwarves each performed both types of work. Further, on the first day, all seven dwarves worked in the diamond mine.*
Prove that, on one of these 16 *days, all seven dwarves were collecting berries.*

Just reading this problem it is not really clear what to expect - it is number 6, so it should probably be difficult, but the text itself looks relatively tame and it even has dwarves in it... This problem is interesting in itself, but as it will turn out, its solution also has some nice connection to coding theory. Let us first briefly present (or recall) the solution of the original problem and then talk a bit more about its context.

Solution:[1] We should start by establishing some notation.

We know that on each day each dwarf performed one of two given types of works, so we could enumerate the dwarves 1 to 7 and encode the work assignment of a given day by a sequence of 7 characters where the kth character denotes the type of work the kth dwarf was doing that day. It will be convenient to use characters 0 and 1, say 0 standing for work in the mine and 1 meaning berry collection in the forest.

Using this language, we can phrase the problem as follows. We have 16 sequences of length 7 where each entry is 0 or 1 such that any two sequences differ in at least 3 entries and the first sequence is 0000000. We wish to show that one of the sequences is 1111111.

Let $V = \{0,1\}^7$ denote the set of $0-1$ sequences of length 7. Let us call the 16 sequences induced by the work assignments *day-sequences*. Let the number of 1s in a sequence be called the *weight* of that sequence and let us say that a sequence $x \in V$ *covers* $y \in V$ if they differ in at most one entry. Note that every $x \in V$ covers exactly 8 sequences. Also note that any two day-sequences differ in at least 3 entries, so they cover disjoint sets of sequences. Therefore the 16 day-sequences together cover $16 \cdot 8 = 2^7$ sequences, i.e. the whole of V.

We know that 0000000 is a day-sequence, therefore there is no day-sequence of weight 1 or 2. Therefore all $\binom{7}{2} = 21$ sequences in V of weight 2 should be covered by day-sequences of weight 3. Each day-sequence of weight 3 covers exactly 3 sequences of weight 2. Therefore there should be 7 day-sequences of weight 3.

Then out of the $\binom{7}{3} = 35$ sequences of weight 3 only the 7 day-sequences are covered so far. The remaining 28 should be covered by day-sequences of weight 4. Each day-sequence of weight 4 covers 4 sequences of weight 3, hence there should be exactly 7 day-sequences of weight 4. Note that these together cover all the sequences of weight 5, so there cannot be any day-sequences of weight 5 or 6. So the last day-sequence should have weight 7, i.e. it should be 1111111. This proves the required statement.

Remark: Note that we did not actually prove that 16 day-sequences with the required property exist, we only proved that *assuming* they exist, sequence 1111111 should be one of them. From the above solution with a bit of work we can find 16 day-sequences with the required properties:

[1]Heavily based on solution 1 from the EGMO 2013 solutions document https://www.egmo.org/egmos/egmo2/solutions.pdf

0000000	1110000	1001100	1000011	0101010	0100101	0010110	0011001
1111111	0001111	0110011	0111100	1010101	1011010	1101001	1100110

We will not prove it here, but this collection is actually unique up to reordering the entries.

Broader context: You may have come across the above collection of 16 sequences before as they form the so-called Hamming(7,4) code. Let us take a step back and talk a bit about codes in general.

A basic problem coding theory is concerned about is transferring some messages by sending binary bits. We choose a certain *code length* n and a set of possible messages A and to each message in A we assign a $0-1$ sequence of length n, called a *codeword*.

For example, we can let the code length be 2, the set of possible messages be $A = \{yes, no, maybe\}$ and let 00 mean *yes*, 11 mean *no* and 01 mean *maybe* (sequence 10 has no meaning). When we talk about a code, we often do not actually write out what the possible messages are and which codeword is assigned to which, just give the set C of codewords, i.e. the set of sequences that carry meaning. In the above example the set of codewords would be $C = \{00, 01, 11\}$.

There are two competing properties we would really like our code to have. Firstly we want it to be efficient, i.e. be able to send many messages with a given code length. Secondly we want it to be robust to potential errors in the transmission, i.e. we want our messages to still be decodable if a few bits get lost or change on the way.

If we just want to be able to send as many different messages as possible, we can let each of the 2^n sequences encode a different message (i.e. let $C = \{0,1\}^n$), but then losing even one bit will make our message undecodable and changing a bit will completely change the message.

If we want our code to be very robust to errors, we can for example choose to send two different messages only, one encoded by a sequence of 0s and the other encoded by a sequence of 1s (i.e. $C = \{00\dots0, 11\dots1\}$). Then changing up to $\lfloor\frac{n-1}{2}\rfloor$ bits we can still correctly recover the original message, but the obvious drawback is that we used n bits to send one of two possible messages, i.e. 1 bit worth of information.

In practice we usually want something inbetween these two extremes, but it depends on the exact context (e.g. how noisy is the channel where we transmit) where we wish to find the balance.

We measure how efficient our code is by its *rate*. A code that transmits one of 2^k possible messages [2], i.e. k bits worth of information in codewords of length n is said to have rate $\frac{k}{n}$.

We can measure the reliability of a code by looking at what is the maximum number of bits that can change so that we can still correctly decode the message.[3] For example a 2-error correcting code means that if any of up to two bits change, we can still decode. Notice that being d-error correcting means exactly that any two codewords differ in at least $2d+1$ entries. Another way of saying that is that the *minimal distance* of the code is at least $2d+1$, where the minimal distance means the smallest distance appearing between two codewords and the *distance* of two sequences is defined simply as the number of entries where they differ.

We can see that the 16 day-sequences from the problem form a code of length 7 with minimal distance 3. This means that the code has rate $\frac{4}{7}$ and it is 1-error correcting.

[2] or anything between 2^{k-1} and 2^k

[3] If we receive a sequence that is not a codeword, we will try to decode it as the codeword that differs from it in the least possible number of entries.

Is it the best we can get? And what does even best mean? We cannot ask for large rate AND large error correction rate at the same time, but we may ask that if we want a given code length n and given error correction d, then what is the maximal rate we can achieve. Or vice versa, if we want a given code length n and a given rate, what is the maximal error correction we can hope for? While there are no exact answers in general, there are some bounds and one of them looks a lot like what we did in the solution of the dwarf problem.

Say we want our code to have length n and be d-error correcting. Let us say that a sequence $x \in \{0,1\}^n$ covers $y \in \{0,1\}^n$ if they differ in at most d entries. We can note that any given sequence covers exactly $1 + \binom{n}{1} + \binom{n}{2} + \ldots + \binom{n}{d}$ sequences and also that no sequence can be covered by two different codewords in a d-error correcting code. So the codewords cover disjoint sets and together they can cover at most 2^n sequences. This means that the number of codewords can be at most $\frac{2^n}{1+\binom{n}{1}+\binom{n}{2}+\ldots+\binom{n}{d}}$.[4]

This bound often cannot be achieved, as often the fraction is not even an integer, but in case of 1-error correcting codes of length 7 this bound gives 16, which is achieved by our code of day-sequences. So our code is optimal in the sense that it contains as many codewords as possible given its length and error correction.

Another interesting property of our code worth discussing is that it is a so-called *linear code*. Linear codes are a special type of code that are often used because they have nice structure and are much easier to handle than just an arbitrary collection of sequences.

For any two sequences in $\{0,1\}^n$ we can define their sum by simply adding them entry-wise and then taking the results mod 2. (So for example 1011+1100=0111.) A linear code of length n is some set $C \subset \{0,1\}^n$ of codewords such that the sum of any two (not necessarily different) codewords is also a codeword.[5] For example we can easily check that

$$C = \{0000, 1001, 0110, 1111\}$$

is a linear code of length 4.

Note that in our example 0000 was a codeword. This is not a coincidence – since the sum of any sequence with itself will be $00\ldots0$, any linear code should contain $00\ldots0$ as a codeword. We may also notice that once we knew 1001 and 0110 were codewords we could be sure that 1111 is also a codeword, since it is their sum.

It is also true in general that if we knew some codewords are in our linear code, we also know that their sums are also in the code. Therefore we can describe a linear code without needing to list all its elements, by giving only a few well-chosen codewords. In general it is always possible to choose some subset B of the codewords such that any codeword can be written as the sum of some elements of B (so we can obtain all elements of C just from B), but no element of B can be written as the sum of some other elements of B (so we cannot throw out an element of B and still have this property). A set B like that is called a *basis* for the code C. The basis B is not unique[6], e.g. in our above example we could have chosen $B = \{1001, 0110\}$ or $B = \{1001, 1111\}$ or $B = \{0110, 1111\}$. But it is true that all possible bases B have the same size k, called the *rank* of the code.

If we have a code of length n and rank k, how many elements will it have? We know every codeword is the sum of some basis elements and there are 2^k possible

[4] Then of course the rate can be at most $\frac{1}{n}\log_2\left(\frac{2^n}{1+\binom{n}{1}+\binom{n}{2}+\ldots+\binom{n}{d}}\right)$. This is called the Hamming bound.

[5] For those familiar with linear algebra: C is a linear subspace of the vector space $\mathbb{F}_2^n$.

[6] unless our code C has only two elements

ways to write down some sum of elements of B (including the empty sum which is defined to be $00\dots 0$). But will these summations all give different sequences or is it possible that some sequence in C is counted twice, because it can be written in two different ways as sums of basis elements? It turns out that the 2^k sums all have different results, so if a code has rank k, then it will have exactly 2^k elements.[7]

E.g. in our above example with basis $B = \{1001, 0110\}$ the possible sums are the empty sum $= 0000$; 1001; 0110; $1001 + 0110 = 1111$. So indeed C has rank 2 and the number of its elements is $2^2 = 4$.

We had a lot of definitions, so let us just recap what we had so far. A linear code of length n is given by a set $C \subset \{0,1\}^n$ of codewords such that the sum of any two codewords is also a codeword. For any linear code C there exists some well-chosen subset $B \subset C$ such that all elements of C can be written as the sum of some codewords in B, but no element of B can be written as the sum of some other elements of B. Such a B is called a basis of the code C and in general it is not unique, but all bases have the same size, called the rank of the code C. Each element of C can be written uniquely as a sum of basis elements, therefore if the rank of C is k, then C will have 2^k elements.

So one reason why linear codes are nice is that we do not have to list all their elements, it is enough to write down a basis. We often write the sequences in the bases under each other to form a table, called the *generator matrix* of the code. This table will have as many columns as the code length and as many rows as the rank of the code.

Our code of day-sequences is also a linear code and a possible basis is

$$\{1110000, 1001100, 0101010, 1101001\}$$

with corresponding generator matrix $G = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}$.

Another nice thing about linear codes is that we do not have to check all pairs of sequences to find the minimum distance, it is enough to check the distance of each sequence from $00\dots 0$, i.e. the weight of each sequence. Do you see why? (Why is it the case that if a linear code contains sequences x and y which are of distance d from each other then there will also be a sequence z which is distance d away from $00\dots 0$?)

Finally let us talk about two more ways to think about our code.

We know it has 16 elements, i.e. it can transmit 4 bits worth of information, but by looking at the generator matrix or the collection of codewords, we cannot really see 4 bits where that information lives.

By closer inspection though we may notice the following. If we want to write down a codeword x, we can choose x_3, x_5, x_6 and x_7 arbitrarily, but then our hands are tied in choosing the rest of the entries. If x is a codeword, then x_1 should be the sum of x_3, x_5 and x_7, while x_2 should be $x_3 + x_6 + x_7$ and x_4 should be $x_5 + x_6 + x_7$, all of the sums considered mod 2 of course.

For example we can choose x_3 to be 1 by adding the first row of the generator matrix to the sum and we can choose it to be 0 by not adding the first row. (For convenience let us just call the rows of the generator matrix b^1, b^2, b^3, b^4.) Similarly x_5 is 1 if and only if b^2 is included in the sum, x_6 is 1 if and only if b^3 is included

[7]Can you actually prove this? Can you prove that a linear code cannot have for example (non-zero) elements x, y, z, w, u such that $x + y + z = w + u$? Once we have that it follows that a linear code with a basis of size k has 2^k elements and using this we can conclude that all bases of a linear code have the same size. It is a more difficult problem to show that a basis even exists...

and the x_7 is 1 if and only if b^4 is included. Then when we want to find x_1, each of b^1, b^2 and b^4 adds 1 to it and similarly for the 2nd and 4th entries.

So we can say that entries 3, 5, 6, 7 are where we are actually sending our message and the rest of the entries are just their to add extra checks to make the code more robust to errors, they are so-called parity check bits.

Now let us see the last, somewhat related way to look at our code.

We have noticed that the codewords are exactly those $x \in \{0,1\}^7$ where

$$\begin{aligned} x_1 &= x_3 + x_5 + x_7 \pmod 2 \\ x_2 &= x_3 + x_6 + x_7 \pmod 2 \\ x_4 &= x_5 + x_6 + x_7 \pmod 2. \end{aligned}$$

We could rearrange these equations as

$$\begin{aligned} x_1 + x_3 + x_5 + x_7 &= 0 \pmod 2 \\ x_2 + x_3 + x_6 + x_7 &= 0 \pmod 2 \\ x_4 + x_5 + x_6 + x_7 &= 0 \pmod 2 \end{aligned}$$

which if you are familiar with matrices can be written as

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \pmod 2.$$

The matrix on the left is called the *parity check matrix* of the code. In general the codewords of a code with parity check matrix H are exactly those $x \in \{0,1\}^n$ which satisfy $Hx = 0 \pmod 2$. We can check that any code defined with a parity check matrix is linear, but the converse is also true, any linear code can be defined via a parity check matrix!

Notice that our code had length 7 and its generator matrix had 4 rows, whereas its parity check matrix had 3 rows. This makes sense: we are sending 7 bits in total, we had 4 bits to choose freely and 3 bits completely constrained. In general it is also true that if the codewords are length n and the rank is k, then we have 'k bits worth of free choice', so we have '$(n-k)$ bits worth of constraints', the parity check matrix has $(n-k)$ rows.

So giving its parity check matrix is yet another way to specify a linear code.

By now you may have wondered where the name Hamming(7,4) is coming from and may have guessed - correctly - that 7 refers to the code length and 4 refers to the rank.[8] There are more Hamming codes with other lengths and ranks, which are defined via their parity check matrix. For any $d \geq 3$ the Hamming$(n, n-d)$ code, where $n = 2^d - 1$ has a $d \times n$ parity check matrix whose columns are exactly the non-zero element of $\{0,1\}^d$. (The code is only defined up to reordering the entries of the codewords.)

We can see that our code also satisfies this definition and with some work can check that each of these Hamming codes has minimum distance 3.

There are numerous other things to say about the collection of these 16 day-sequences, but we will conclude our discussion here. We deviated very far from the original EGMO problem, but I hope if nothing else, it could show how it is a part of a much larger system and how revisiting this problem years later - like I did - might make you think 'Oh wait, I actually know what this is!'.

[8]Hamming refers to the American mathematician Richard W. Hamming.

Exercises from the text:

1. Show that if B is a basis for code C, then any element of C can be written uniquely as a (possibly empty) sum of elements of B. Deduce that if B has k elements then C has size 2^k.
2. Use this to show that all bases of C have the same size.
3. Show that if a linear code contains two sequences of distance d, then it also contains a sequence of weight d.
4. Show that any code defined via a parity check matrix (i.e. C consists of all x satisfying $Hx = 0$ for some given matrix H) is linear.