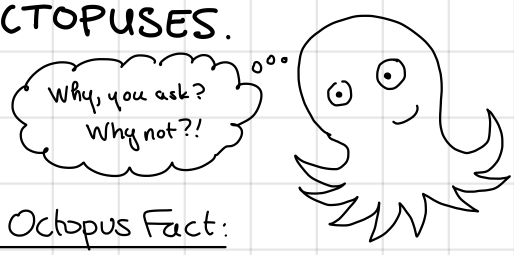
CS250/EE387 - LECTURE 1 - LOGISTICS + BASICS

AGENDA

1. Logistics
2. Course Pitch
3. Basic problem in coding theory
4. Formal definitions
5. Rate vs. Distance: Hamming bound

During this course, we will (at least in the lecture notes...) be learning a little bit about OCTOPUSES.



Today's Octopus Fact:

Octopuses have three hearts! One of the hearts is inactive when the octopus is swimming, so it tires out faster when swimming than when crawling.

① LOGISTICS

- **COURSE ELEMENTS**
  - Pre-recorded videos, with corresponding lecture notes
  - In-class exercises, meant to practice, reinforce, and extend material in the videos/notes.
  - 3 HW assignments
  - Final project
- **CLASS MEETINGS**
  - This is a "flipped" class — watch the videos before class and come to class ready to engage!

SEE COURSE WEBSITE FOR MORE DETAILS! Also for the schedule, materials, assignments, etc.

② COURSE PITCH

"$\underbrace{\text{ALGEBRAIC}}_{\text{II}}$ $\underbrace{\text{ERROR CORRECTING CODES}}_{\text{I}}$"

I. Error correcting codes are a fundamental tool for →
- communication
- storage
- complexity theory
- algorithm design
- cryptography
- pseudorandomness
- etc...

II. Algebraic techniques are a fundamental tool for designing ECCs.

Basically, this course is about the following fact:

LOW-DEGREE POLYNOMIALS
DON'T HAVE TOO MANY ROOTS.

As we will see, this fact is stupidly useful throughout CS and EE. (In particular, in these places)

[Course Pitch ctd.]

In this class we will discuss:

What are codes?
- Basics of Error Correcting Codes: combinatorial bounds + existential results
- Some basic abstract algebra [finite fields – nothing fancy]

How do we get them?
- The classic polynomial codes: Reed-Solomon and Reed-Muller
- [If time we will mention] fancier polynomial codes: Multiplicity Codes, Folded RS codes

How do we use them?
- Algorithms for manipulating these codes in various settings: Unique decoding, list decoding, local decoding

Why do we care?
- Applications!

In this class we will NOT discuss:

- Nitty gritty details of any one application (this is a THEORY course)
- LDPC codes, Turbo codes, Raptor Codes, Fountain Codes, ...

[See Montanari's course EE 388 for all that good stuff.]

At the end of this course:

YOU SHOULD HAVE THE TOOLS TO USE ERROR-CORRECTING CODES (and the algebraic tools behind them) IN YOUR OWN RESEARCH/LIFE.

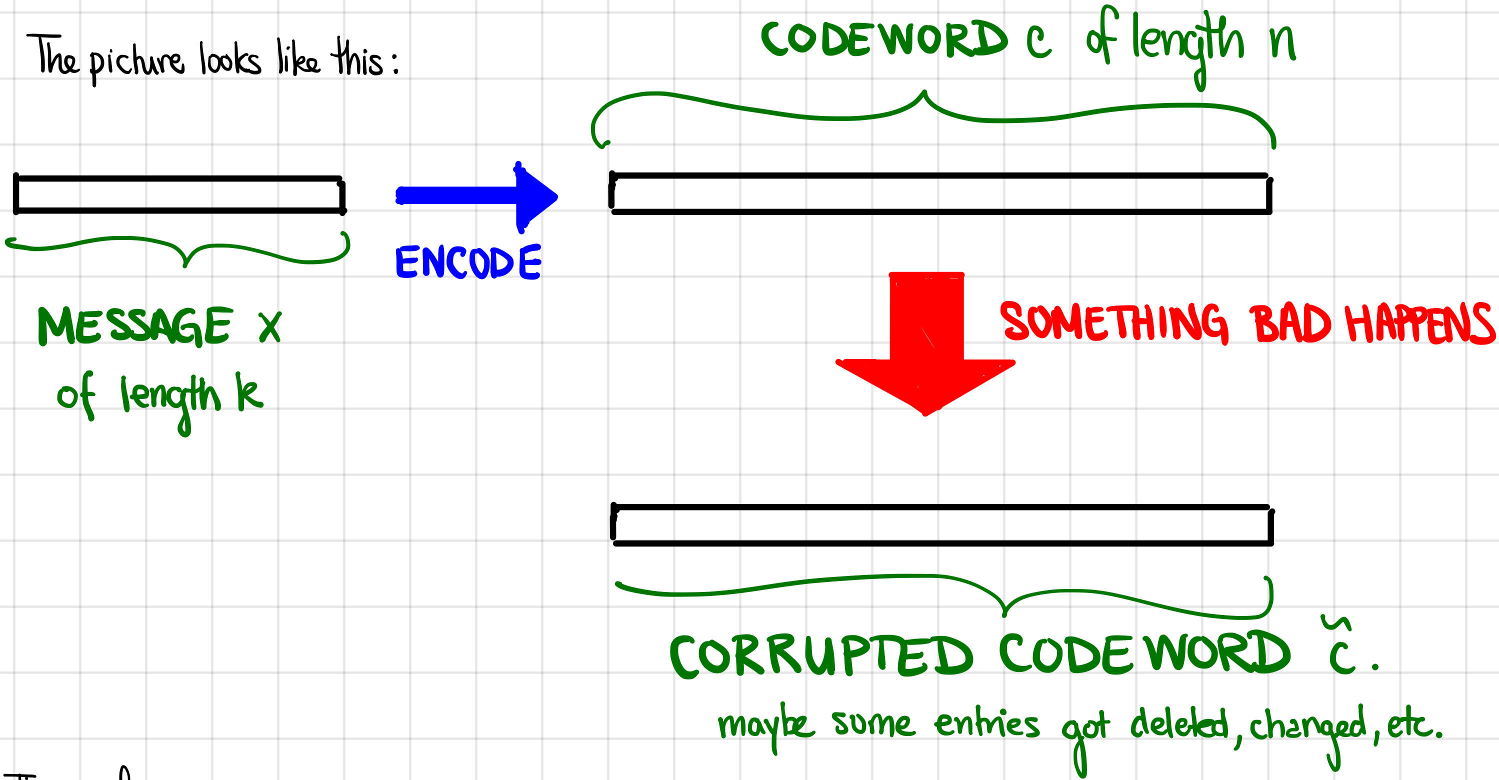
That means:

- Enough familiarity with terminology, constructions, algorithms, and notions of decoding to pick up a research paper and understand it.

- Exposure to lots of examples of how ECCs can be useful in a wide variety of settings.

③ The BASIC PROBLEM in CODING THEORY

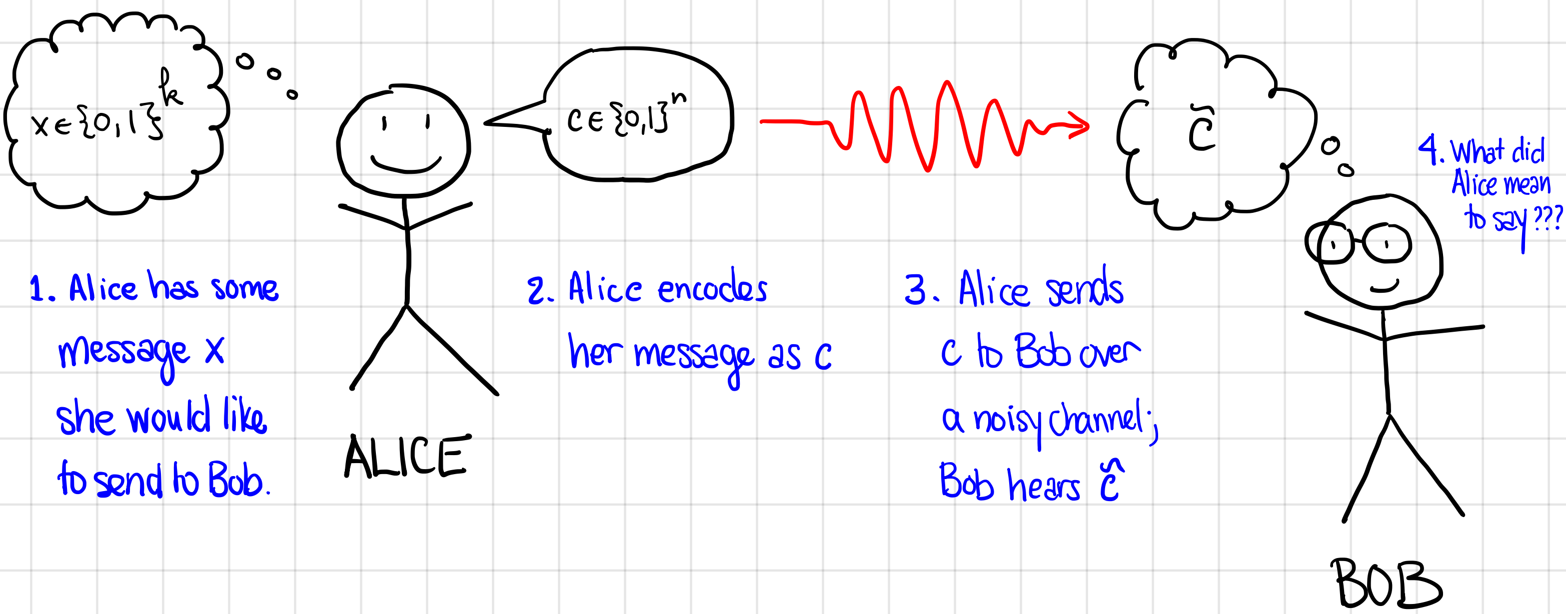
The picture looks like this:



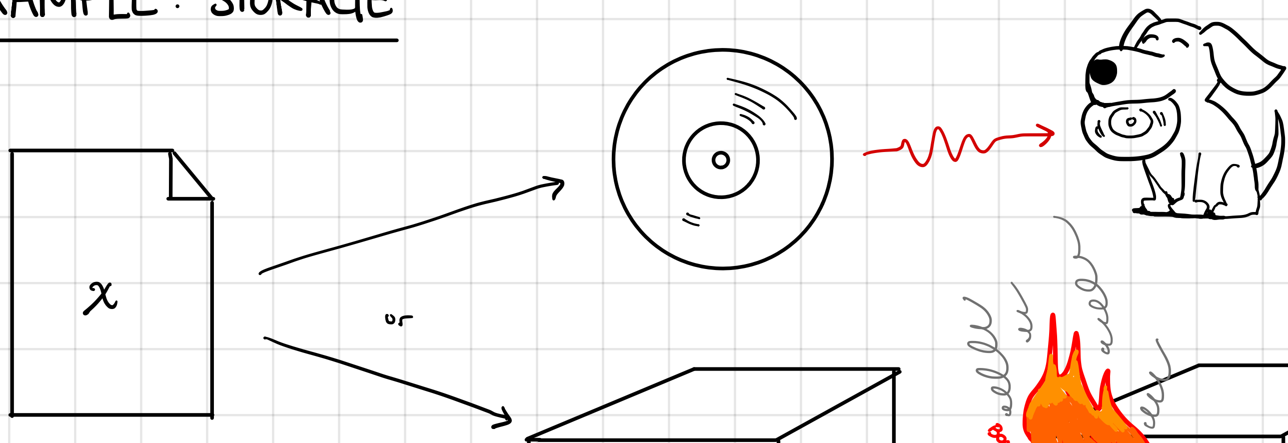
The goal:

$$\boxed{\text{GIVEN } \tilde{c}, \text{ FIND (SOMETHING ABOUT) } x.}$$

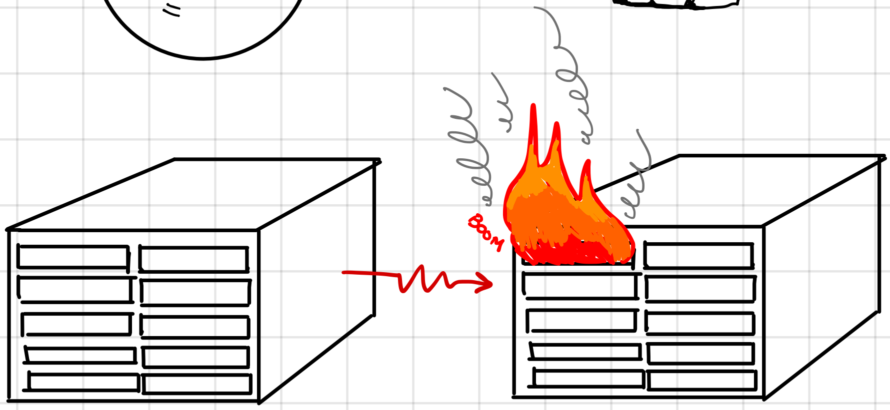
EXAMPLE: COMMUNICATION



EXAMPLE: STORAGE



1. Suppose x is a file.



2. Encode x as a codeword c.

3. c is stored; say on a CD or in a RAID array... → but something BAD happens.

5. I still want x!

THINGS WE CARE ABOUT:

① We should be able to handle the SOMETHING BAD, whatever that means.
② We should be able to recover WHAT WE WANT TO KNOW about x.
③ We want to MINIMIZE OVERHEAD: k/n should be as big as possible.
④ We want to to all this EFFICIENTLY.

QUESTION What are the trade-offs between ①-④?

It depends on how we model things:

- What is the SOMETHING BAD?
- What exactly do we WANT TO KNOW?
- What counts as EFFICIENT? What kind of access do we have to $\tilde{c}$?

Today we'll look at one way of answering these questions.
There are many legit ways, and we will see more throughout the quarter.

④ FORMAL DEFINITIONS

Let Σ be any finite set and let $n>0$ be an integer.

DEF. A <u>CODE</u> $\mathcal{C}$ of <u>BLOCK LENGTH</u> n over an <u>ALPHABET</u> Σ is a subset $\mathcal{C} \subseteq \Sigma^n$. An element $c \in \mathcal{C}$ is called a <u>CODEWORD</u>.

Sometimes I will say "length" instead of "block length."

So far, this is not a very interesting definition.

EXAMPLE 1. $\mathcal{C} = \{$ HELLOWORLD, BRUNCHTIME, ALLTHETIME $\}$ is a code of block length 10 over $\Sigma = \{A, B, \ldots, X, Y, Z\}$.

This is not a very interesting code, although it does capture the vast majority of my thoughts.

EXAMPLE 2.

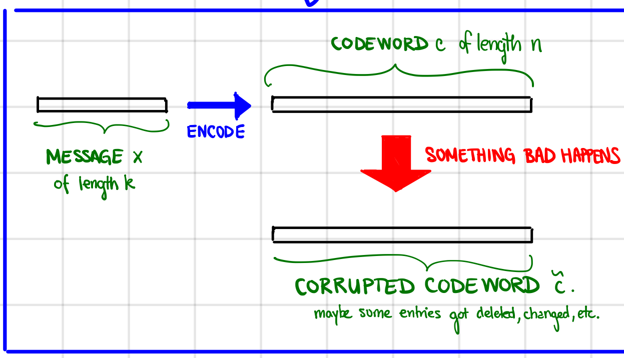
$$\mathcal{C} = \left\{ \begin{array}{c} (0,0,0,0) \\ (0,0,1,1) \\ (0,1,0,1) \\ (0,1,1,0) \\ (1,0,0,1) \\ (1,0,1,0) \\ (1,1,0,0) \\ (1,1,1,1) \end{array} \right\}$$

is a code of length 4 over $\Sigma = \{0,1\}$.

If $\Sigma = \{0,1\}$, we say $\mathcal{C}$ is a **BINARY CODE**.

This second example is a bit more interesting.

What does this have to do with the picture from before? [this one]



Consider the map $\mathrm{ENC}: \{0,1\}^3 \to \{0,1\}^4$ given by:

$$\mathrm{ENC}: \underbrace{(x_1, x_2, x_3)}_{\text{message } x} \longmapsto \underbrace{(x_1, x_2, x_3, x_1+x_2+x_3 \bmod 2)}_{\text{codeword } c}$$

For example, $\mathrm{ENC}((0,1,1)) = (0,1,1,0)$.

Then $\mathcal{C} = \mathrm{Im}(\mathrm{ENC})$. That is, $\mathcal{C}$ is the set of all codewords that could be obtained using this encoding map.

The second example can actually be used to fix bad stuff.

Suppose you see:

$$0 \quad \quad 0 \quad 1$$

The SOMETHING BAD that happened obscured this entry.

This is called an <u>ERASURE</u>.

We know which bit got erased, but we don't know what its original value was.

What is the missing bit?

It must be a 1, since $0 + \quad + 0 = 1 \mod 2$.

Suppose instead you see:

$$0 \quad 0 \quad 0 \quad 1$$

Then we know SOMETHING went wrong (at least one bit was flipped) but we are not sure what it was.

This is called an <u>ERROR</u>.

One bit may have been changed, but we don't know which one.

We say that the code in EXAMPLE 2 can $\begin{cases} \text{CORRECT one ERASURE} \\ \text{— or —} \\ \text{DETECT one ERROR} \end{cases}$

But it <u>cannot</u> CORRECT one ERROR. Let's see a code that can.

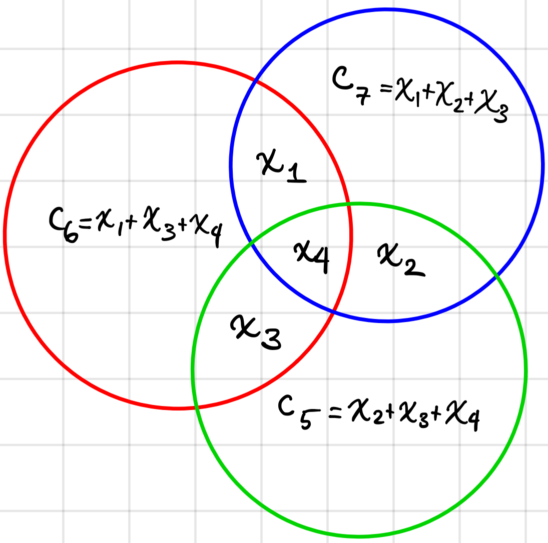
<u>EXAMPLE 3</u>. Consider the encoding map $ENC: \{0,1\}^4 \to \{0,1\}^7$

all mod 2

$$ENC: (x_1, x_2, x_3, x_4) \longmapsto (x_1, x_2, x_3, x_4, x_2+x_3+x_4, x_1+x_3+x_4, x_1+x_2+x_4)$$

Let $\mathcal{C} = \operatorname{Im}(ENC)$. So $\mathcal{C} \subseteq \{0,1\}^7$, aka $\mathcal{C}$ is a BINARY CODE of LENGTH 7.

ANOTHER WAY to VISUALIZE this CODE:



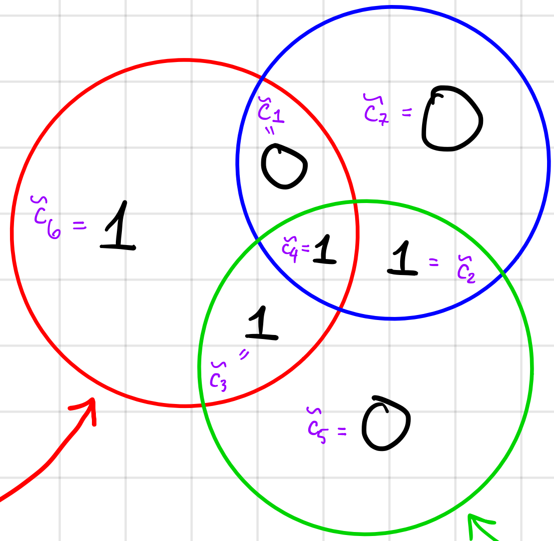
Put the message x_1, x_2, x_3, x_4 in the middle, and then the circles tell you how to fill in the rest.

PUZZLE: I took some $c \in \mathcal{C}$ and flipped at most one bit, to obtain:

$$\tilde{c} = (0 \;\; 1 \;\; 1 \;\; 1 \;\; 0 \;\; 1 \;\; 0)$$

What is c?

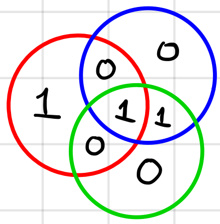
ALTERNATIVE LOOK at the PUZZLE:



SOLUTION: In a legit codeword, all three circles should sum to 0. Here, both the green circle and the red circle are messed up.

The thing those circles have in common is $\tilde{c}_3$. So if I flip $\tilde{c}_3$ I should get a legit codeword again:

$\hat{c} = (0, 1, 0, 1, 0, 1, 0)$ or



And, $\hat{c}$ is the ONLY solution because flipping any other bit would mess up other circles.

Hooray! That works. But it seems pretty ad hoc. For the rest of this lecture and some of next one, we'll try to introduce some formalism to make this solution seem less ad hoc. At the same time we will flesh out what we care about for ECCs.

these things.

1. We should be able to handle the SOMETHING BAD, whatever that means.
2. We should be able to recover WHAT WE WANT TO KNOW about x.
3. We want to MINIMIZE OVERHEAD: k/n should be as small as possible.
4. We want to to all this EFFICIENTLY.

First some definitions:

DEF. The HAMMING DISTANCE between $x, y \in \Sigma^n$ is

$$\Delta(x,y) := \sum_{i=1}^{n} \mathbb{1}\{x_i \neq y_i\}$$

The RELATIVE HAMMING DISTANCE between $x, y \in \Sigma^n$ is

$$\delta(x,y) := \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}\{x_i \neq y_i\} = \frac{\Delta(x,y)}{n}.$$

Notice: this is a metric. In particular, it obeys the triangle inequality.

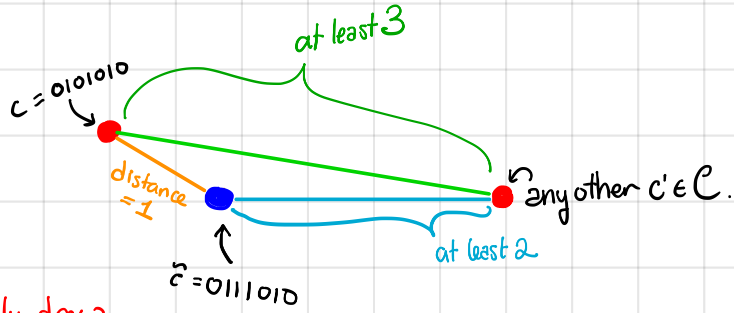
DEF. The MINIMUM DISTANCE of a code $C \subseteq \Sigma^n$ is

$$\min_{\substack{c \neq c' \\ \text{in } C}} \Delta(c, c')$$

Sometimes I'll just call this "distance."

CLAIM The code in EXAMPLE 3 has minimum distance 3.

If the CLAIM is true, it explains why that code can correct one error:



DISCLAIMER

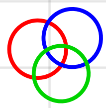
I will frequently draw pictures as though Hamming distance is Euclidean distance, and $\{0,1\}^n$ is $\mathbb{R}^2$.

Indeed, if $\Delta(c,c') \geq 3 \ \forall\, c \neq c' \in \mathcal{C}$, then

$$\Delta(\tilde{c}, c) = 1 \quad \Longrightarrow \quad \Delta(\tilde{c}, c') \geq 2 \quad \forall\, c' \in \mathcal{C} \text{ other than } c.$$

by the triangle inequality. Thus, the "correct" codeword $c \in \mathcal{C}$ is uniquely defined by "the one that is closest to $\tilde{c}$."

To prove the CLAIM:

- You can probably convince yourself by staring at  (in the same way we convinced ourselves we could always fix one error).
- But we'll see a much less ad hoc way to establish distance after we build up some machinery for **LINEAR CODES** in Lecture 2, so let's put it aside for now.

The **POINT** of this dicussion was that:

MINIMUM DISTANCE is a reasonable proxy for robustness.

aka, { ① We should be able to handle the SOMETHING BAD, whatever that means.
② We should be able to recover WHAT WE WANT TO KNOW about x.

That is,

- In EXAMPLE 2, the code had minimum distance 2 (check this!) and could CORRECT 1 ERASURE and DETECT 1 ERROR.
- In EXAMPLE 3, the code had minimum distance 3, and could CORRECT 1 ERROR.

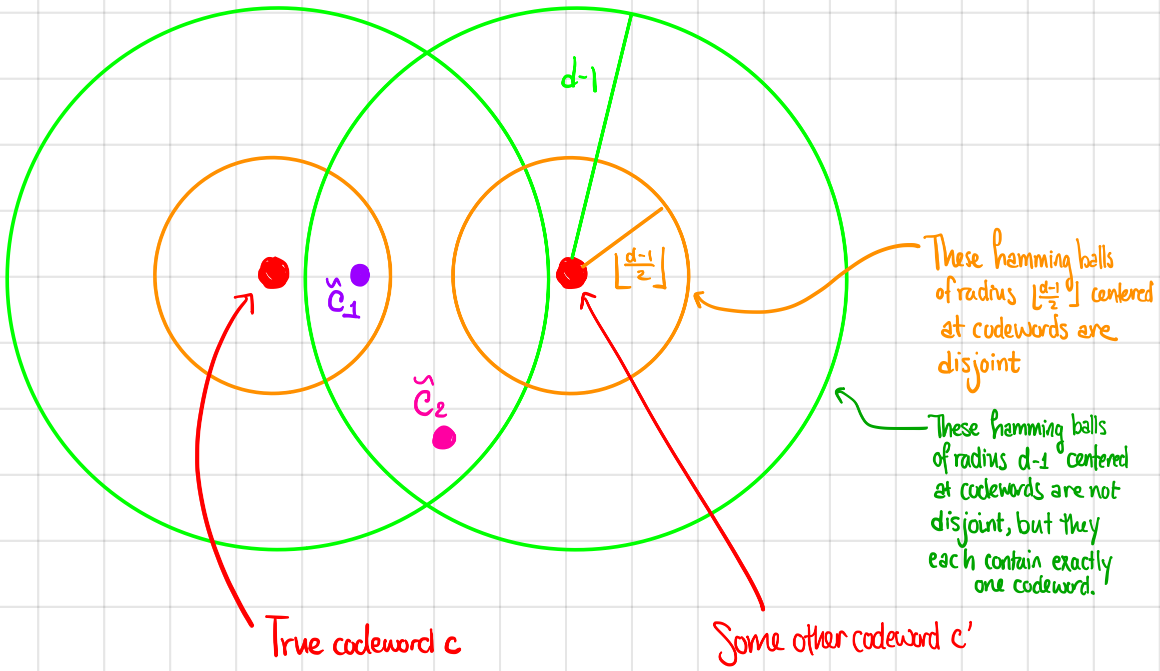
More generally, a code with distance d can:

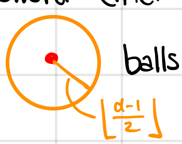
- correct $\leq d-1$ erasures
- detect $\leq d-1$ errors
- correct $\leq \left\lfloor \frac{d-1}{2} \right\rfloor$ errors

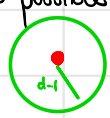
For these two, the (inefficient) algorithm is: "if you see $\tilde{c}$, return $c \in C$ that's closest to $\tilde{c}$."

For this one, the (inefficient) alg. is: "If $\tilde{c} \notin C$, say that something is wrong."

The picture looks like this:



- If c is the "correct" codeword and $\leq \left\lfloor \frac{d-1}{2} \right\rfloor$ errors are introduced, we may end up with $\tilde{c}_1$. Since all the  balls are disjoint, we can find c from $\tilde{c}_1$.

- However, if $\leq d-1$ errors are introduced, we may end up with $\tilde{c}_2$. Now it's possible that $\tilde{c}_2$ came from c or that it came from c'; we can't tell. However, since each  ball doesn't contain any codeword other than its center, we can tell that something went wrong.

Returning to this, we can now clarify the first two things.

(from earlier)

THINGS WE CARE ABOUT:

1. We should be able to handle the SOMETHING BAD, whatever that means.
2. We should be able to recover WHAT WE WANT TO KNOW about x.
3. We want to MINIMIZE OVERHEAD: k/n should be as small as possible.
4. We want to to all this EFFICIENTLY.

If we want:

1. We should be able to handle $\lfloor \frac{d-1}{2} \rfloor$ WORST-CASE ERRORS or $d-1$ WORST-CASE ERASURES
2. We want to recover ALL OF x (aka correct the errors or erasures)

Then we should say 1 & 2 We want MINIMUM DISTANCE d.

Next we will move on to 3.

ASIDE. A natural question at this point is, "what if I don't want to handle worst-case errors/erasures?" For example, if my code has minimum distance d, and I have two codewords:

$$c = (0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0) \in \{0,1\}^n$$
$$c' = (\underbrace{1\ 1\ 1\ 1}_{d}\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0) \in \{0,1\}^n$$

Then if an adversary chooses to flip the first two bits, we'd be confused. But instead say two bits get flipped at random. The probability we get confused is $\frac{\binom{d}{2}}{\binom{n}{2}}$ which might be quite small!

The random-error model (also called the "Shannon model" or "Stochastic model") is natural and important! We will discuss it a little bit in this class. However, most of our focus will be in the worst-case model (also called the "Hamming model" or "adversarial model."

Moving on to (3), what do we mean by "overhead"?

DEF. The **MESSAGE LENGTH** (sometimes called **DIMENSION**) of a code $\mathcal{C}$ over an alphabet Σ is defined to be $k = \log_{|\Sigma|} |\mathcal{C}|$.

This definition makes sense with our operational understanding:

$$\underbrace{\left(\text{message } x \text{ of length } k \text{ over } \Sigma\right)}_{|\Sigma|^k \text{ possibilities}} \xmapsto{\text{ENC}} \underbrace{\left(\text{codeword } c \in \mathcal{C}\right)}_{|\mathcal{C}| \text{ possibilities}}$$

So $|\Sigma|^k = |\mathcal{C}|$ aka $k = \log_{|\Sigma|} |\mathcal{C}|$.

DEF. The **RATE** of a code $\mathcal{C} \subseteq \Sigma^n$ with block length n over an alphabet Σ is

$$R = \frac{\log_{|\Sigma|} |\mathcal{C}|}{n} = \frac{\text{message length } k}{\text{block length } n}$$

So if R is close to 1, that's **GOOD**. Not much overhead.
And if R is close to 0, that's **BAD**. Lots of overhead.

DEF. A code with distance d, message length k, block length n, and alphabet Σ is called a $(n, k, d)_{|\Sigma|}$ code.

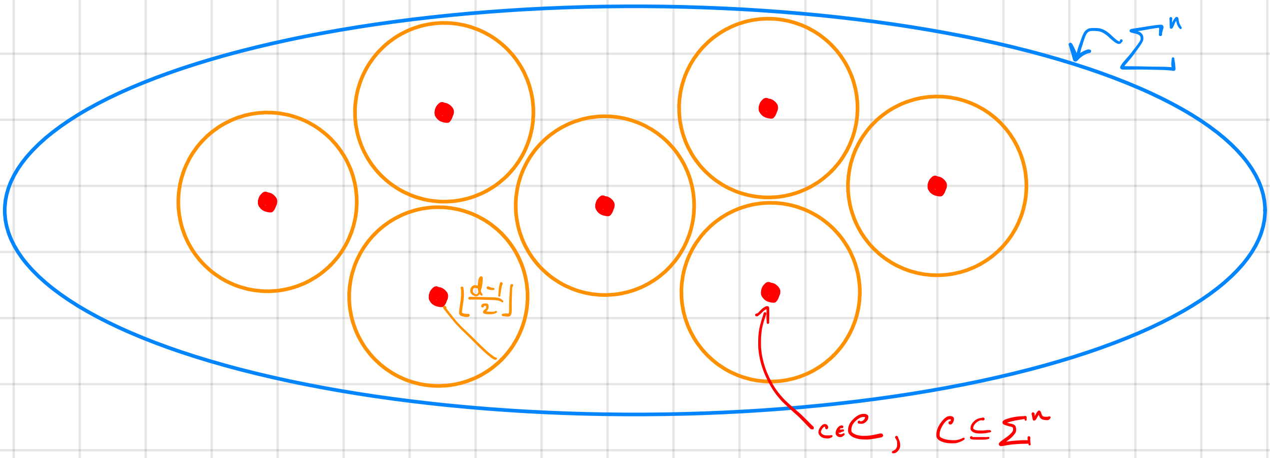
QUESTION. WHAT IS THE BEST TRADE-OFF BETWEEN RATE AND DISTANCE?

This question is still open for binary codes!
But there's lots we do know.

5 RATE vs. DISTANCE: HAMMING BOUND.

What is the best trade-off between rate and distance we can hope for?
The HAMMING BOUND gives one bound on this.

Let's return to the picture we had before, with disjoint Hamming balls of radius $\lfloor \frac{d-1}{2} \rfloor$:



- We have $|\mathcal{C}|$ disjoint Hamming balls of radius $\lfloor \frac{d-1}{2} \rfloor$.
- There can't be too many of them or they wouldn't all fit in Σ^n.

To be a bit more precise:

DEF. The HAMMING BALL in Σ^n of radius e about $x \in \Sigma^n$ is

$$B_{\Sigma^n}(x,e) := \{ y \in \Sigma^n : \Delta(x,y) \leq e \}.$$

The VOLUME of $B_{\Sigma^n}(x,e)$ is $\mathrm{Vol}_{|\Sigma|}(e,n) := |B_{\Sigma^n}(x,e)|$

Notice that $|B_{\Sigma^n}(x,e)|$ does not depend on x.

Say that $|\Sigma| = q$. Then

$$\mathrm{Vol}_q(e,n) = \underbrace{1}_{\vec{0}} + \underbrace{\binom{n}{1}(q-1)}_{\text{all the elements of } \Sigma^n \text{ of weight } 1} + \underbrace{\binom{n}{2}(q-1)^2}_{\text{all the elements of } \Sigma^n \text{ of weight } 2} + \cdots + \underbrace{\binom{n}{e}(q-1)^e}_{\text{all the elements of } \Sigma^n \text{ of weight } e.}$$

Notes:

- Sometimes I will drop the "Σ^n" from the $B_{\Sigma^n}(x,e)$ notation
- Sometimes I will write $B_{\Sigma^n}(x, e/n)$ if it's more convenient to talk about relative distance.

So that means that if a code $\mathcal{C} \subseteq \Sigma^n$ has distance d and message length k, where $|\Sigma| = q$,

$$\underbrace{|\mathcal{C}| \cdot \mathrm{Vol}_q\left(\left\lfloor \frac{d-1}{2} \right\rfloor, n\right)}_{\text{total volume in the } \odot \text{'s}} \leq \underbrace{q^n}_{\text{total volume in } \Sigma^n}$$

so taking logs of both sides,

$$\underbrace{\log_q(|\mathcal{C}|)}_{\log_q(|\mathcal{C}|) = k} + \log_q\left(\mathrm{Vol}_q\left(\left\lfloor \frac{d-1}{2} \right\rfloor, n\right)\right) \leq n$$

$$\Rightarrow \boxed{\text{Rate} = \frac{k}{n} \leq 1 - \frac{\log_q\left(\mathrm{Vol}_q\left(\left\lfloor \frac{d-1}{2} \right\rfloor, n\right)\right)}{n}}$$

This is called the HAMMING BOUND.

Back to EXAMPLE 3, which was a $(\underset{n}{7}, \underset{k}{4}, \underset{d}{3})_{\underset{q}{2}}$ code

- We have $\left\lfloor \frac{d-1}{2} \right\rfloor = 1$
- $\mathrm{Vol}_2(1, 7) = 1 + \binom{7}{1} \cdot 1 = 8$
- So

$$\frac{k}{n} \leq 1 - \frac{\log_2(8)}{7} = 1 - 3/7 = 4/7.$$

- And in fact $\frac{k}{n} = 4/7$, so in this case the Hamming bound is tight!

Notes about this example:

- When the Hamming bound is tight, we say the code is PERFECT.
- EXAMPLE 3 (which is perfect) is a special case of something called a HAMMING CODE.
- You will explore this more in in-class exercises and on homework.

⑥ RECAP

Now we understand the first 3 of our desiderata:

THINGS WE CARE ABOUT:

THESE THREE
① We should be able to handle the SOMETHING BAD, whatever that means.
② We should be able to recover WHAT WE WANT TO KNOW about x.
③ We want to MINIMIZE OVERHEAD: k/n should be as large as possible.
④ We want to to all this EFFICIENTLY.

That is, (for now), our goal is to design codes $\mathcal{C} \subseteq \Sigma^n$ so that:

- The DISTANCE of $\mathcal{C}$ is as large as possible.
- The RATE of $\mathcal{C}$ is as close to 1 as possible.

Even without the algorithmic considerations, understanding the trade-off between rate and distance turns out to be a fascinating combinatorial question!

In fact, for binary codes ($|\Sigma| = 2$), this question is **STILL OPEN!**
(We saw that EXAMPLE 3 was optimal for $n=7$ and $k=4$, but what about in general?)

Next time, we'll give an overview of abstract algebra, and then give some more definitions that will further de-ad-hoc-ify EXAMPLE 3.

That's it for today.

QUESTIONS to PONDER:

① How would you generalize the code in EXAMPLE 3 to larger n?

② What is the best bound you can come up with on the rate of a code $C \subseteq \{0,1\}^n$ with distance d?