AICC II

Simon Lefort Laura Paraboschi

February 2024

1 Entropy and Data Compression

1.1 Sources and Entropy

Let S be a **random variable**, a function that associates a **real number** to each **outcome**, s.

For example, a random variable can be X, that counts the number of 6 after 2 throws.

$$H(s) = \begin{cases} 0 & \text{if } s \text{ is "we got no 6"} \\ 1 & \text{if } s \text{ is "we got a single 6"} \\ 2 & \text{if } s \text{ is "we got two 6"} \end{cases} \tag{1}$$

Let A be **the alphabet** for this symbol (all the possible outcomes). Therefore, there are $|A|^n$ possible values for n successive symbols.

Support The support of a random variable is all the outcomes s such that $p(S = s) \equiv p_S(s) > 0$.

Entropy The entropy of a random variable represents the level of "uncertainty", "surprise" related to the variable's possible outcomes.

$$H_b(S) := -\sum_{S \in A} p_S(s) log_b(p_S(s))$$

In this course, b will almost always be 2, because we are interested in binary representation of data.

Note that we can also compute the expected value of $-log(p_S(S))$.

$$\begin{aligned} H(S) &= \mathbb{E}[-log_b(p_S(S))] \\ &= \sum_{s \in A} p_S(s) \cdot -log_b(p_S(s)) \end{aligned} \tag{2}$$

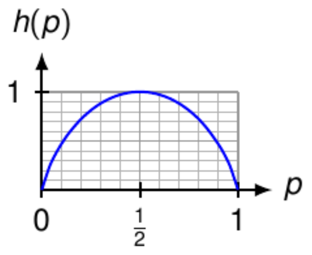


Figure 1: when $|A| = 2, h(p_s(s))$ is entropy

Uniform distribution We say that a random variable is uniformly distributed if:

$$\forall s \in A, p(s) = \frac{1}{|A|}$$

In that case the entropy of A is the maximum, $log(|A|)$.
Indeed:

$$0 \text{ (no randomness) } \leq H_b(S) \leq log_b|A| \text{ (max randomness)}$$

IT-Inequality (useful for demos) the **IT-inequality** states that:

$$log_b(r) \leq (r-1)log_b(e)$$

with equality iff $r = 1$.

Sequence of random variables A sequence of random variable is finite $(S1, ..., S_n)$ or infinite $S_1, S_2, ...$. The probability that the sequence happens is denoted by $p_{S_1,...S_n})$. We can then compute the entropy $H(S_1, ..., Sn_))$.
Note:

$$H(S_1, S_2, ..., S_n) \leq H(S_1) + ...H(S_n)$$

with equality iff $S_1, ..., S_n$ are independent. Seems "intuitive" because there is less "surprise" if you know one outcome is "linked" to another one.

Marginal distribution , or $p_X(x)$, can be calculated by summing over all the outcomes of Y the value of $p_{X,Y}(x/y)$.

Joint distribution , often represented as a table, gives us the probability that $X = x$ and $Y = y$.

1.2 Source Coding

Encoder An encoder is defined by:

- the input alphabet $\mathbf{A}$
- the output alphabet $\mathbf{D}$ (typically $0, 1$)
- the codebook $\mathbf{C}$ which consists in sequences of D
- a one-to-one function $\Gamma : \mathbf{A}^k \to \mathbf{C}$

Prefix-free codes are codes where no cdeword is the prefix of another word.

Unique decodability is verified if every concatenation of codewords can a unique parsing into a sequence of codewords (*can you decode with only one solution?*)

- fixed-length code $\implies$ uniquely decodable
- prefix-free $\implies$ uniquely decodable
- prefix-free $\equiv$ instantaneous
- reverse is prefix-free $\neg \implies$ instantaneous

Kraft-McMillan theorem Let $I_1, ..., I_m$ be the respective codewords lengths.

$$\text{D-ary code uniquely decodable} \implies D^{-I_1} + ... + D^{-I_m} \leq 1$$

(intuition: comes from the number of used leaves)

Kraft-McMillan theorem 2 if the positive integers $I_1, ..., I_m$ satisfy the Kraft's inequality for some integer D, there exists a D-ary prefix-free code.
To build it, draw a tree and order codewords lengths in increasing order.

Average codeword length takes into account the probability a character appears.

$$L(S, \Gamma) := \mathbf{E}[l(S)] = l_a \cdot p(S = a) + l_b \cdot p(S = b) + ... + l_z \cdot p(S = z)$$

$$\Leftrightarrow \mathbf{L(S, \Gamma)} = \sum_{\mathbf{s} \in \mathbf{A}} \mathbf{p_s(s)} \cdot \mathbf{l(\Gamma(s))}$$

Unit for codeword length is "code symbols" (or bits in most cases in this class).

Shannon codes are prefix-free codes where the length of each symbol is calculated this way:

$$l(\Gamma(s)) = \lceil -log_D(p_S(s)) \rceil$$

These are clearly **not optimal** codes (but they are still uniquely decodable and easy to create). Also, the following identity applies:

$$H_D(S) \leq L(S, \Gamma_{Shannon}) < H_D(S) + 1$$

proof lower bound: $H_D(S) - L(S, \Gamma) \leq 0$ (+ IT ineq.)
proof upper bound: as we use $\lceil -log_D(p_S(s)) \rceil < -log_D(p_S(s)) + 1$

Huffman codes construction, unlike Shannon codes, start from the leaves. We write the probability of each symbol, and recursively join the ones with the smallest probability. Such codes are **optimal** (the best we can have). They are not necessarily unique.

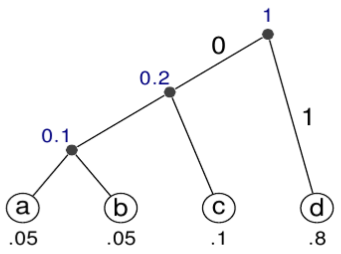


Figure 2: 0.05/0.05 then 0.1/0.1 then 0.2/0.8

When using D-ary code, at the beginning, we need $k(D-1) + D$ leaves.

Path-Length lemma states that you can compute the average length of a code by simply summing over the probabilities of the intermediate nodes *(here, 0.1 + 0.2 + 1 = 1.3).*

Compressing long strings , concatening Huffman codes n times no longer produce the best result. We need to make a new Huffman codes for **blocks** of n letters and associated probabilities. The per letter average codeword-length bcomes (in a 2-letter example):

$$\frac{H_D(S_1, S_2)}{2} \leq \frac{L((S_1, S_2), \Gamma_{SF})}{2} < \frac{H_D(S_1, S_2)}{2} + 1$$

for $S_1, S_2, ..., S_n$ (dependent OR independent):

$$\frac{H_D((S_1, S_2, ..., S_n)}{n} \leq \frac{L((S_1, S_2, ..., S_n), \Gamma_{SF})}{n} < \frac{H_D((S_1, S_2, ..., S_n) + 1}{n}$$

or, if $S_1, S_2, ..., S_n$ are independent and identically distrib. (IID):

$$H_D(S) \leq \frac{L((S_1, S_2, ..., S_n), \Gamma_{SF})}{n} < H_D(S) + \frac{1}{n}$$

(because $H(S_1) = H(S_2) = ... = H(S_n) = nH(S)$)

Therefore when compressing long strings, *"it's no longer entropy and entropy +1, entropy is it"* - Gastpar

IID Source from now on we will think about sources, that emits an (possibly) infinite sequence of symbols. IID sources are **independent** from each other and **identitcally distributed** according to a fixed distribution $p_S(s)$ (not really the case in the real world).

Thus : let S be the infinite sequence produced by an IID source S.

- by encoding blocks of symbols into D-ary codewords, the average codeword-length per symbol of a uniquely decodable code can be made as close as desired to $H_D(S)$
- no uniquely decodable D-ary code can achieve a smaller average codeword-length

$$H(S_{\text{IID source}}) = log(|\text{IID source}|)$$

1.3 Conditional Entropy

In real world sequences, strings are not i.i.d.

Conditional probability gives the probability of the event $X = x$ given that $Y = y$ has occured

$$p_{X/Y}(x/y) = \frac{p_{X,Y}(x,y)}{p_Y(y)}$$

Average conditional entropy is the average entropy of a variable X knowing Y (but not knowing exactly what little y is).

$$H(X/Y) = \sum_{y \in Y} p(Y = y)(\sum_{x \in X} p(x/y) \cdot \log_D(p(x/y)) \leq H(X)$$

$$\Leftrightarrow \sum_{y \in Y} H(X/Y = y)p(Y = y)$$

with equality iff X, Y are independent:

$$\text{X, Y independent} \Leftrightarrow H(X/Y) = H(X) \Leftrightarrow p(x,y) = p(x)p(y)$$

Conditional entropy bound can we say that **with a fixed little y**:

$$H_D(X/Y = y) \leq_? H_D(X)$$

It would seem intuitive, but no!

- Let X : 1 if I follow the lecture, 0 otherwise
- I come to the campus 99% of the time
- I follow the lecture 100% of the time if I come to the campus
- I follow the lecture 50% of the time if I do not come to the campus
- Let Y : 1 if my car is broken, 0 otherwise
- The entropy of X is lower than the entropy of X knowing Y = 1.

Conditional entropy of f(x) , where f is a deterministic function

$$H(f(x)/x) = 0$$

Chain rule

$$H_D(X,Y) = H_D(X) + H_D(Y/X)$$
$$\Leftrightarrow H_D(Y,X) = H_D(Y) + H_D(X/Y)$$
$$\Leftrightarrow H(X/Y) = H(X,Y) - H(Y)$$

therefore:

$$H_D(S_1, ..., S_n) = H_D(S_1) + H_D(S_2/S_1) + ... + H_D(S_n/S_1, ..., S_{n-1})$$

Entropy rate à quel point quand on ajoute une lettre à notre code ça fait descendre vers l'avg code length vers l'entropie ?

Sunny-Rain source probability of next symbol only depends on the previous one

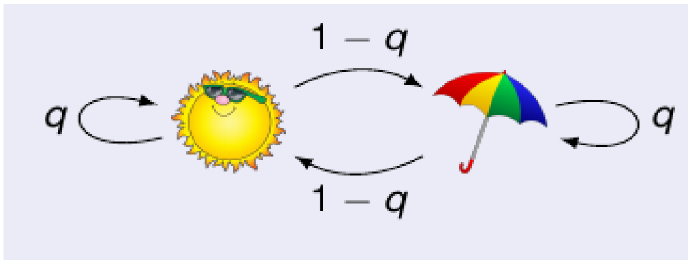


Figure 3: Enter Caption

Compute probability of c swaps starting with a specific R or S.

$$p_{s_1,s_2,\dots,s_n}(s_1,\dots,s_n) = \frac{1}{2}\cdot q^{n-1-c}(1-q)^c$$

$(1-q)$ represents the swaps

Distribution of a random variable X is now denoted as $p_X(\cdot)$.

Regular source $\mathbb{S} = (S_1, S_2, \dots)$ if:

$$H(\mathbb{S}) = \lim_{n\to\infty} H(S_n)$$

$$H^*(\mathbb{S}) \text{ (entropy rate) } = \lim_{n\to\infty} H(S_n|S_1, S_2, \dots, S_{n-1})$$

exist and are finite.

$$H^*(\mathbb{S}) \leq H(\mathbb{S})$$

For regular sources:

$$H^*(\mathbb{S}) = \lim_{n\to\infty} \frac{H(S_1,\dots,S_n)}{n}$$

Stationary source if statistics, probabilities never change over time.

Known bounds on factorial

$$\frac{n^n}{e^{n-1}} \leq n! \leq \frac{n^{n+1}}{e^{n-1}}$$

1.4 Entropy and algorithms

Algorithmic performance can be bounded using entropy (for example, we know we would need at least X questions before finding a word correctly, on average for the 20 questions game).

2 Cryptography

Monoalphabeticalphabet cipher is a unique bijection between a fixed alphabet and a new alphabet of the same size.

Polyalphabetic cipher (like Vigenere) uses multiple substitution tables (a key specify which table is used for which position in the message).

Perfect secrecy , when knowing the cryptogram does not give you any guess about the plaintext, they are statistically independent (where T is the plaintext, K is the key) and it implies:

$$H(T) \leq H(K)$$

Symmetric-key cryptosystem is one for which both ends use the same key (for encryption then decryption). Problem: in a n-user network, each user needs n-1 keys to communicate. It's very expensive to distribute securely keys.

One-way function ex. $g \to g^a \mod p$. For reverse, it's a discrete logarithm problem to compute a such that $g^a \mod p = A$

Exchanging keys (Diffie and Hellman) We agree on a p prime and a generator (see below). Let a the secret known by A, and b the secret known by B $\in \{1, 2, 3, ..., p-1\}$

- g^a is sent to B
- g^b is sent to A

therefore, the attacker knows g^a, g^b but can not compute g^{ab} while A and B can.

Exchanging plaintext same as before, but now A sends :

- g^a
- $t \cdot g^{ab} \mod p$

B needs to compute C s.t. $C \cdot g^{ab} \mod p = 1$. So then B can retrieve $t = C \cdot g^{ab} \cdot t \mod p$

2.1 Number theory

Euclidian division , $a = m \cdot q + r$, r has to be positive $\in \{0, 1, 2, 3, 4, 5, ..., m-1\}$ if $a \mod m$ is non negative, then $r = a \mod m$, otherwise $r = a \mod m + m$

Congruence , $a \equiv b \mod m$ if $m|a - b \Leftrightarrow a - b = m \cdot k$. It is an **equivalent** relation!

$$a + b \mod m = a + (b \mod m) \mod m$$
$$a(b \mod m) \mod m = ab \mod m$$
$$a^n \mod m = (a \mod m)^n \mod m$$

Important:

$$xa \equiv xb \mod z \not\Longrightarrow a \equiv b \mod z$$

but

$$xa \equiv xb \mod z \Longrightarrow yxa \equiv yxb \mod z$$

if you can find y s.t. $yxb = 1$ then it implies $a \equiv b \mod z$

Equivalence classes s.a. $[0]_4, [1]_4, etc.$

Euler's totient function is used when we want to compute b such that $a^b \equiv 1 \mod n$.

$$\text{Let } n = \prod_{i=1}^{r} p_i^{k_i}$$
$$\text{Let } \varphi(n) = \prod_{i=1}^{r} (p_i - 1) p_i^{k_i - 1}$$

According to Euler's theorem, **if a and n are relatively prime**:

$$a^{\varphi(n)} \equiv 1 \mod n$$

It represents the number of numbers co-prime with n before it (e.g. for 6, there is only 1 and 5, so $\varphi(6) = 2$).
if p is prime:

$$\phi(p^k) = p^k - p^{k-1}$$

if p and q are prime:

$$\phi(pq) = pq - p - q + 1 = (p-1)(q-1)$$

Division rules (stupid, but still great to recall) , a number can be divided by:

- 9 if the sum of **all its numbers** can be divided by 9
- 8 if the sum of **its last 3 numbers** can be divided by 8
- 6 if it is divisible by 2 and by 3
- 4 if the sum of **its last 2 numbers** can be divided by 4

Fast exponentiation is used when you want to compute a^b fastly. For instance 3^45.

- First, decompose b (binary): $45 = 32 + 8 + 4 + 1$
- Then, compute 3^2. Then square. Again. Again. Until you arrive to the value of 3^{32}.
- Then, compute $(3)^1 \cdot (3^2)^0 \cdot (3^4)^1 \cdot (3^8)^1 \cdot (3^{16})^0 \cdot (3^{32})^1$

Mod 97 - 10

- append 00 to your number
- compute the remainder after division by 97
- compute the check digits c = 98 - r
- replace 00 with this two-digit check number
- by construction, this new number will always be equal to 1 mod 97.
- $(100n + 98 - (100n \mod 97))$

multiplicative inverse if there exists a multiplicative inverse for $a, \mod b$, then $ax \mod b$ has a unique solution. Otherwise that's not necessarily the case:

$$[3]_9 x \equiv [3]_9$$

$$x \text{ can be } [1]_9, [4]_9, [7]_9$$

Bézout (inverse) a is inversible mod m iif $\exists x$ such that:

$$ax \equiv 1 \mod m$$

$$\Leftrightarrow ax - 1 = mk$$

$$\Leftrightarrow ax - mk = 1$$

Therefore a is inversible $\mod m$ iif $\exists(x, y)$ such that $ax + my = 1$ (Bezout).

$$\Leftrightarrow GCD(a, m) = 1$$

If there is an inverse mod m, it is unique. or use $[b]_m^{\phi(m)-1}$

$gcd(a,b)$	$a = bq + r$	q	$u = \tilde{v}$	$v = \tilde{u} - q\tilde{v}$

Divisible if a/c and b/c and $gcd(a, b) = 1$ then ab/c (think about common prime factors to demonstrate it).

$\mathbb{Z}/\mathbb{Z}m^*$, we only keep elements of $\mathbb{Z}$ such that they do not have any common factor with m (co-prime). Therefore, when we multiply the numbers between them, the final result will always be co-prime with m. Therefore $\mathbb{Z}/\mathbb{Z}m^*$ is a commutative group.

Order of an element a , the smallest positive integer k such that $a \cdot a \cdot ... \cdot a = e$, where e is the identity. It always exists for finite commutative groups.?
Any integer p that also satifies this is a multiple of the order of a. The order of a must divide the number of elements in the commutative group G.

Generator recall the fact that for Diffie Hellman we needed to pick a generator (a number such that when you exponentiate then modulo reduce, it gives you all the elements of the group). It is now clear to see that we should pick a number of the group with the order equal to n!

If we have a generator g and an index i, on peut retrouver l'ordre de l'élément g^i in the cyclic group: $ord(g^i) = \frac{ord(g)}{gcd(ord(g),i)}$

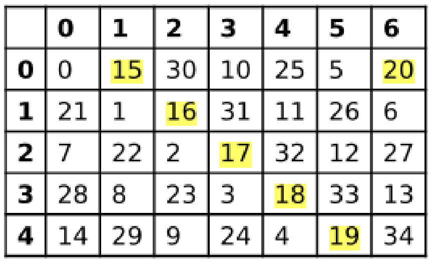
Isomorphism two groups are isomorphic if they have the same set of orders.

Cyclic groups have a generator (for instance Z,Z5) $\{g, g^2, g^3, .., g^G = e\}$, G being the number of elements in the group. Otherwise if $<$ G, then we do not get all the elements in the group

2.1.1 Method: CRT

We create a table that, given two modulo m and n, provides the modulo according to $m \cdot n$.

To fill it more quickly, think about the diagonals.



	0	**1**	**2**	**3**	**4**	**5**	**6**
0	0	15	30	10	25	5	20
1	21	1	16	31	11	26	6
2	7	22	2	17	32	12	27
3	28	8	23	3	18	33	13
4	14	29	9	24	4	19	34

Here we see that if we have a number congruent to 3 modulo 7 and congruent to 1 modulo 5, then it will be congruent to 31 modulo 35.

We define $\phi(n) : \mathbb{Z}/m_1m_2\mathbb{Z} \rightarrow \mathbb{Z}/m_1\mathbb{Z} \times \mathbb{Z}/m_2\mathbb{Z}$ as the function that, for a certain n in the table, gives the modulos in the two other lands. $\phi^{-1}(a, b)$, on the contrary, gives us back the two modulos in the original lands.

2.1.2 Commutative Groups

(note) Commutative group : a set endowed with a binary operation that satisfies the following properties:

Method: Checking if a Group $G(\{a, b, c\}, \star)$ is Commutative

1. Find the identity element ϕ such that $\forall x, x \star \phi = x$.
2. Find the inverse element α for each element such that $\forall x, x \star \alpha = \phi$.
3. Verify that $\forall x, y, z \in G$:
   - $x \star y \in G$
   - $(x \star y) \star z \equiv x \star (y \star z)$
   - $x \star y \equiv y \star x$

(note) Fields : Fields are commutative. They have two operations, addition and multiplication. Every non-zero element in a field has a multiplicative and additive inverse. (e.g., $\mathbb{Z}/p\mathbb{Z}$ is a field (m has to be prime because every element needs a multiplicative inverse), but also infinite fields like $\mathbb{Q}$ or $\mathbb{R}$).

(note) Finite (or Galois) Field Fields : Fields are commutative. They have two operations, addition and multiplication. Every non-zero element in a field has a multiplicative and additive inverse. E.g., $\mathbb{Z}/p\mathbb{Z}$ is a field (p has to be prime because every element needs a multiplicative inverse)). F_p is isomorphic to $\mathbb{Z}/p\mathbb{Z}$.
see https://www.youtube.com/watch?v=yBVqk4YM2VY

(note) characteristics : the characteristic of a field is the smallest positive integer n such that $n \cdot 1 = 0$. If no such integer exists, the field has characteristic 0. (e.g. F_9 has characteristic 3

Cyclic Groups : a group is cyclic if there exists an element g in the group such that every element in the group can be written as a power of g. The order of a cyclic group is the number of elements in the group.

Method: Checking if a Group $G(\{a, b, c\}, \star)$ is Isomorphic to $H(\{e, f, g\}, \circ)$

- Create a table of the two groups for each operation.
- Look for elements that behave the same way (e.g., are they their own inverse?), and match them together.

- Then, for the remaining elements, check if they have the same order.

  The order of an element x in a group G is such that $x^{\text{order}} = \text{identity}$. If two elements have the same order, then "they are isomorphic".

  If two groups do not have the same cardinality, they are not isomorphic.

Usually, when we write $(\mathbb{Z}/2\mathbb{Z}, +)\times(\mathbb{Z}/3\mathbb{Z}, +)$, we use the operations component-wise.

2.2 RSA

2.2.1 Choosing the Parameters

We choose two large prime numbers p and q. We calculate $m = p \cdot q$, so we can easily calculate $\Phi(m) = (p-1)(q-1)$.

The foundation of RSA is that we want $[t^{e \cdot d}]_m = [t]_m$.

So, $e \cdot d = \Phi(m) \cdot q + 1$, and more generally:

$$e \cdot d + \Phi(m) \cdot k = 1 \quad (1)$$

Let's choose a p and a $q \to m = p \cdot q \to \Phi(m) = (p-1)(q-1)$.
Let's choose any e such that $\gcd(e, \Phi(m)) = 1$, see condition (1).
Now we need to find a d that works with our parameters, knowing that we want $e \cdot d \equiv 1 \pmod{\Phi(m)}$, see condition (1).

2.2.2 Finding d Faster

The problem is that it takes a long time to do this. We can calculate it faster by setting:

$$k = \operatorname{lcm}(p-1, q-1)$$

This way, we just need to set $e \cdot d \equiv 1 \pmod{k}$, see condition (1).
Why does it work? $\operatorname{lcm}(p-1, q-1)$ is a divisor of $(p-1)(q-1) = \Phi(m)$ (it is the smallest number that can be divided by both).

For example:

$$10 \cdot 60 = 160 = 2^5 \cdot 5 \quad \operatorname{lcm}(10, 60) = 2^4 \cdot 5$$

So if $e \cdot d \equiv 1 \pmod{k}$, it means that it is also $e \cdot d \equiv 1 \pmod{\Phi(m)}$.

Similarly, the number used for encoding must be coprime with k.

3 Signature

We send t, the plain text and $[t^d]_m$. e is available to the public. They can compute $[t^{d^e}]_m = [t]_m$ to check whether they get the right plain text again, to see if you are the real sender.

Because we do not want to send too much data (sending the plaintext twice!), we hash the plaintext with a public function to reduce its size, then compute $[h^d]_m$. The goal for attackers is to seek for a plaintext that would give the same hash.

4 Channel coding

Linear code is a subspace of $\mathbb{F}^n$ (a vector space).

Generating matrix is a matrix that, when multiplied by a message, gives the codeword. It is a $k \times n$ matrix. We can build it by taking k linearly independent codewords (as we know from algebra, any k linearly independent vectors can generate a k dimensional space).

Generating matrix for systematic code is a matrix that has the identity matrix as its first k columns. It is very useful to build a parity-check matrix.

When we don't have a systematic code, we can always transform it into one by using the Gauss-Jordan elimination method (then no need to update the parity-check matrix), or by swapping the columns of the generating matrix (then we need to also swap the columns of the parity-check matrix).

Parity-check matrix is a matrix that, when multiplied by a codeword, gives us a syndrome (that is 0 if the codeword is correct).

bit-rate : $\frac{log_D(M)}{n}$, D is the size of the alphabet, M is the number of codes in the codebook, n is the size of the codes in the codebook

Singleton's bound $d_{\min} \leq n - k + 1$ (make sure to use the right D, the size of the alphabet)

Minimum distance decoding , we receive a codeword that goes through an error channel, and we pick the code in the codebook with the minimum hamming distance with the errored code.

Let p be the maximum weight, the maximum number of errors we can make.

- Detection: $p < d_{\min}$
- Correction of erasures: $p < d_{\min}$
- Correction: $p < \frac{d_{\min}}{2}$ (we must stay within the ball of a code)

Reed-Solomon codes are a family of MDS codes. They are defined by a generator polynomial. It goes from $\mathbb{F}^k$ to $\mathbb{F}^n$, where n is the size of the codewords. The generator polynomial is a polynomial of degree $k - 1$.

- Select $a_1, a_2, ..., a_k$ in $\mathbb{F}$
- Select your message $u_1, u_2, ..., u_k$
- Create a polynomial $u(x) = u_1 + u_2x + ... + u_kx^{k-1}$

- Evaluate this polynomial at $a_1, a_2, ..., a_n$ to get the codeword
- Then, when you want to recover the message, you can use the Lagrange interpolation method, and from k points, you can recover the polynomial.

Method: create a polynomial that goes through k points $f(a_1) = 1$ $f(a_2) = 2$ $f(a_3) = 3$

- $\frac{(x-a_2)(x-a_3)}{(a_1-a_2)(a_-a_3)} \cdot 1$
- $+ \frac{(x-a_1)(x-a_3)}{(a_2-a_1)(a_2-a_3)} \cdot 2$
- $+ \frac{(x-a_1)(x-a_2)}{(a_3-a_1)(a_2-a_1)} \cdot 3$

so it cancels all the other terms except the one we want to keep.