Number Base Case
$N = d_n R^n + d_1 R^1 + d_0 R^0$
the d specifies the Number system -> d_2 == binary
can also be written as R_2
This can also be used to expand numbers:
$N_{10}255 = 2 * 10^2 + 5 * 10^1 + 5 * 10^0$
$N_2110 = 1 * 2^2 + 1 * 2^1 + 0 * 2^0 => N_{10}6$

Quantities:
N -> natural numbers | | **Z** -> full numbers
Q -> rational numbers | | **R** -> real numbers

Common number systems:
Decimal: $N_{10} = n * 10^n ..0 * 10^0$
Binary: $N_2 = n * 2^n ..0 * 2^0$
$2^{10} = 1024, 2^9 = 512, 2^8 = 256, 2^7 = 128, 2^6 = 64,$
$2^5 = 32, 2^4 = 16, 2^3 = 8, 2^2 = 4, 2^1 = 2, 2^0 = 1$
Hexadecimal: $N_16 = n * n^{16} ..0 * 16^0$
notation: 0 1 2 3 4 5 6 7 8 9 A B C D E F
$16^5 = 1048576, 16^4 = 65536, 16^3 = 4096, 16^2 = 256,$
$16^1 = 16, 16^0 = 1$

Modulo
8 mod 4 = (8) -> 0 , 8 mod 3 = (6) -> 2 , 8 mod 5 = (5) -> 3
if x<y in x mod y then the result will always be x!
any negative numbers can be considered as NOTnegative
aka only absolute values! modulo deals with |x|
many programming languages actually do not follow this!
they have their own implementation of modulo.
$5 \equiv 3 \bmod 2$ -> as 5 mod 2 = 1 and 3 mod 2 = 1

Codeword length
Byte = 8 bit || Word = 16 or 32 bit
TCP packet = 1024 bit

Cyclic group
Es sei $F(a) = a^3 + a + 1 = 0$,
- Dann können wir zunächst festhalten
- a = a
- $a^2 = a^2$ aber
- $a^3 = a+1$
- $a^4 = a(a+1) = a^2 + a$
- $a^5 = a(a^2 + a) = a^3 + a^2 = a^2 + a+1$
- $a^6 = a(a^2 + a +1) = a^3 + a^2 + a = a+1 + a^2 + a = a^2 + 1$
- $a^7 = a(a^2 + 1) = a^3 + a = a+1 + a = 1$
- $a^8 = a$: der Zyklus beginnt von vorne!

{**0, 1, *a*, *a*², *a*+1, *a*² + a, *a*² + *a*+1, *a*² + 1** }
{000, 001, 010, 100, 011, 110, 111, 101}

WHAT THE FUCK

Result Quantity the result of all possible outcomes
it is denoted with: Ω
A single element of the result list is: ω -> $\omega \in \Omega$
The list of results is $|\Omega|$
Example Dice roll: $\Omega = \{1, 2, 3, 4, 5, 6\}$

Probability: $P(A) = \frac{\text{best results}}{\text{all results}} = \frac{|A|}{|\Omega|} = \frac{|A|}{n}$
So what is the probability of rolling a 6?
$P(\text{desired number to roll}) = \frac{\text{only 1 good result!}}{\text{6 possible results}} = \frac{1}{6}$
hence the chance is 1 in 6
Why this complicated method? You can modify desired results! just change the A in P(A)!

Inverse Probability: P(inverse) = 1 - P(A)
dice -> $1 - \frac{1}{6} = \frac{5}{6}$

Addition rule:
$P(A \cup B) = P(A) + P(B) - P(A \cap B)$
!!The last part is needed, as otherwise the number would exceed the possible states!!
$P(A \cup B \cup C) = P(A) + P(B) + P(C) - P(A \cap B) - P(A \cap C) - P(B \cap C) + P(A \cap B \cap C)$

Amount of possibilities:
ordered probes with replication:
2 coins, head and tail, possibilities? **k**=head/tail=2 **n**=coins=2
$\Omega = n^k = 2^2$

ordered probes without replication:
5 dices. How many combinations?
dice numbers = **n** = 6 (1-6), dice amount = **k** = 5
possibilities = $\Omega = \frac{n!}{(n-k)!} = \Omega = \frac{6!}{(6-5)!} = 720$
Or this:
$\Omega = \Pi_n^{n-k+1} n = \Pi_6^{6-5+1} 6 = 2 * 3 ... 5 * 6 = 720$

unordered probes wihout replication:
25 players, each should only play once with the other.
$\Omega = \frac{n!}{k!(n-k)!}$ -> $\frac{25!}{2!(25-2)!}$ -> $\frac{\text{too big}}{\text{too big}} = 300$
as you can see the bottom is a BIG calculation, so
$\Omega = \frac{\Pi_n^{n-k+1} n}{k!} -> \frac{\Pi_{25}^{25-2+1} 25}{2!} -> \frac{24 * 25}{2} = 300$
Note that **k** can also be defined as the length of the tuple we want to receive.
-> (Player,Player) - > 2

Source to Sink Information

Nachricht (Darstellung & Bedeutung)	redundant	nicht-redundant
irrelevant	Zeichenvorrat bei Quelle und Senke verschieden	
relevant	vorhersagbar	Information

Entropy

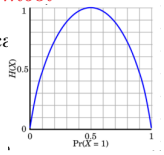
information content
this essentially just us how many bits are needed
k is base state count -> bit = 2
and **N** is the full number of states
example: list True,False,True,False 4 states total, base 2.
$H_0 = log_k(N)[k]$ -> $H_0 = log_2(4)[bit] = 2$

information flow
essentially information content over time
$H_0^* = \frac{log_2(N)}{\tau}[\frac{bit}{s}]$

information quantity / Surprise
$I(x_k) = -log_2(P(x_k))[bit]$

Entropy (Surprise per element)
0 means no symbols. 1 means perfect balance 50-50
$H(X) = \Sigma_{k=1}^N P(x_k) * I(x_k)[\frac{bit}{symbol}]$
where X is the list of symbols

Sink Redundance / Code Reducnc
$R_Q = H_0 - H(X)[\frac{bit}{symbol}]$
$R_c = L - H(X)[\frac{bit}{symbol}]$



Code Word Length
$L(x_k) = \text{rounded}(I(x_k))[bit]$

Median Code Word Length
$L = \Sigma_{k=1}^N P(x_k) * L(x_k)[\frac{bit}{symbol}]$

Entropy of the entire Code
$H_c(X) = \Sigma_{k=1}^N P(x_k) * L(x_k)[\frac{bit}{symbol}]$

H_c can be a real number -> $H_c \in \mathbb{R}$

Für jede beliebige zugehörige Binärcodierung mit Präfixeigenschaft ist die mittlere Codewortlänge nicht kleiner als die Entropie $H(X)$:	Für jede beliebige Quelle kann eine Binärcodierung gefunden werden, so dass die folgende Ungleichung gilt:
$H(X) \leq L$	$H(X) \leq L \leq H(X) + 1$

Sink without memory
$P(x_k, y_k) = P(x_k) + P(y_i)$

Sink with memory
$P(x_k, y_i) = P(x_k) + P(x_k|y_i)$

Entropy without memory / Combined Entropy
$H(H,Y) = \Sigma_{x_k}^N \Sigma_{y_i}^N P(x_k, y_i) * (-log_2(P(x_k, y_i))$
or: $H(X,Y) = H(X) + H(Y)$

Entropy with memory
$H(H,Y) = \Sigma_{x_k}^N \Sigma_{y_i}^N P(x_k) *$
$P(x_k, y_i) * (-log_2(P(x_k) * P(x_k|y_i))$

Encoding of Symbols
- **Ordne die Zeichen gemäss ihrer Auftrittswahrscheinlichkeit**
- **Die beiden Zeichen mit der kleinsten Auftrittswahrscheinlichkeit haben die gleiche CW-Länge l_N**
- **Sei l_N die mittlere CW-Länge für eine Quelle mit N Zeichen und l_{N-1} die mittlere CW-Länge für den Fall, dass die beiden letzten zu einem einzigen Zeichen zusammengefasst werden, dann gilt:**

$L_N - (p(x_{N-1}) + p(x_N)) \cdot L(x_N) = L_{N-1} - (p(x_{N-1}) + p(x_N)) \cdot (L(x_N) - 1)$
$\Rightarrow L_N = L_{N-1} + p(x_{N-1}) + p(x_N)$

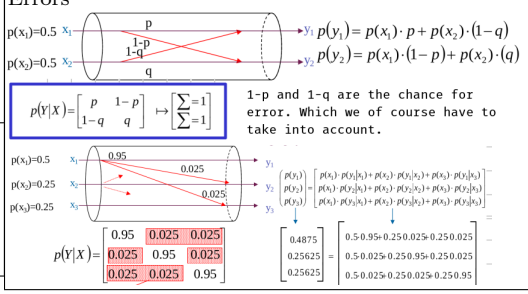
1	2	3	4	5	6	7	8	9
0.22	0.19	0.15	0.12	0.08	0.07	0.07	0.06	0.04
1	2	3	4	8	9	5	6	7
				0	1			
0.22	0.19	0.15	0.12	0.1		0.08	0.07	0.07
1	2	3	6	7	4	8	9	5
			0	1		0	1	
0.22	0.19	0.15	0.14		0.12	0.1		0.08
1	2	8	9	5	3	6	7	4
		00	01	1		0	1	
0.22	0.19	0.18		0.15		0.14		0.12

continue this pattern until every symbol has a code
note the extra 0 on every step

Run Length Encoding RLE/RLC
Quelltext w: Agggbbehfffgggg => |w|=15
Codiert w_c: A3g2beh3f4g => $|w_c|$= 11
A+3xg+2xb+e+h+3xf+4xg
shortening of length by compressing repetition.

Encoder and Decoder
You need to either choose 1 or 0 as the starting bit. After that the decoder can print out the correct code.

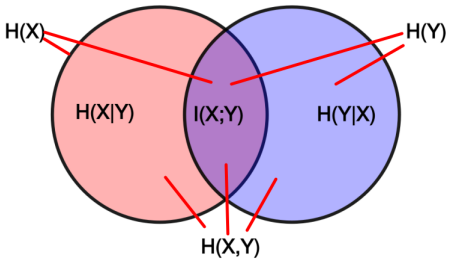
Chiffre text
You can "encrypt" your data by shifting the codes by a certain amount.
In the caesar chiffre this is done with the number 4. a -> e
Please do not use this, use RSA or other algorithms.

Errors


Conditional Entropy -> Entropy of Y given X
$H(Y|X) = \Sigma_{k=1}^N \Sigma_{i=1}^N P(x_k, y_i) * (-log_2(\frac{P(x_k, y_i)}{P(x_k)}))$

Chain Rule
$H(Y|X) = H(X,Y) - H(X) \;||\; H(Y \backslash X)$

Bayes Rule
$H(Y|X) = H(X|Y) - H(X) + H(Y) \;||\; H(Y \backslash X)$



Transinformation
likelyhood of information being correct at arrival.
$T = H(X) - H(X|Y) \;||\; H(Y) - H(Y|X)$
or: $|(X;Y)$

Hamming distance / distance to next valid codeword
$h = Min_{i,j}(d(x_i, x_j))$

error detection distance
the amount of bits that differ from input to output
$e^* = h - 1$

error correction distance for h even
$h = 2e + 2$ -> $e = \frac{h-2}{2}$

error correction distance for h uneven
$h = 2e + 1$ -> $e = \frac{h-1}{2}$

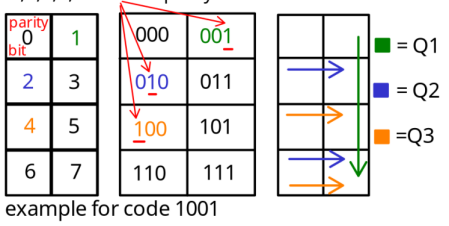
Consider the valid input either 111 or 000.
The Hamming distance **h** is therefore 3 bits.
The detection distance e^* is 3 - 1
Due to h being uneven, the correction distance **e** is $\frac{h-1}{2}$
which results in 1.

tighly packed coderoom
n = dimension of code
m = dimension of messages $2^m * \Sigma_{w=0}^e \binom{n}{w} \leq 2^n$
k = dimension of control -> n = m + k
The code is considered to be tightly packed
if the equation has the result 2. aka == not smaller.

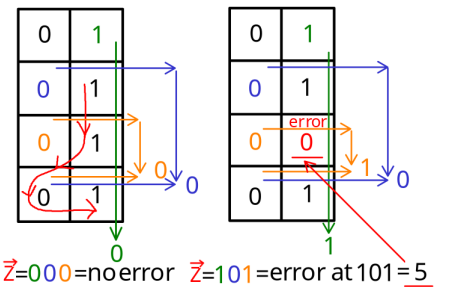
m=2 k=1

x_1	x_2	x_3	
0	0	0	0 = (0 + 0) mod 2 = 0 OK
0	1	1	1 = (0 + 1) mod 2 = 1 OK
1	0	1	1 = (1 + 0) mod 2 = 1 OK
1	1	0	0 = (1 + 1) mod 2 = 0 OK
0	0	1	$x_3 = (x_1 + x_2) \bmod 2$
0	1	0	1 = (0 + 0) mod 2 = 0 NOT OK
1	0	0	0 = (0 + 1) mod 2 = 1 NOT OK
1	1	1	0 = (1 + 0) mod 2 = 1 NOT OK
			1 = (1 + 1) mod 2 = 0 NOT OK

Hamming Codes
The hamming code is very easy to implement
$\Sigma_i x_i * \vec{P}_i \equiv \vec{0} mod 2$
The syndrome $\vec{Z} = \Sigma_i x_i * \vec{P}_i mod 2$
1,2,4,8,16...2^x are parity checks



example for code 1001



$\vec{Z}$=000=no error $\vec{Z}$=101=error at 101=5

note that the 001 010 100 of the parity checks are simply the unit vector $\vec{0}$!!!

parity checks needed:

$par = log_2(\text{bit amount of code})$

1101 = 4 bits -> 3 parity checks
as 4 can be displayed by 3 bits -> 100

OR we can use XOR

write down all the positions with 1 in binary:

001
011
101

error

001
011

errors can be represented by either removing one bit from the calculation -> setting it to 0 or adding it, setting it to 1

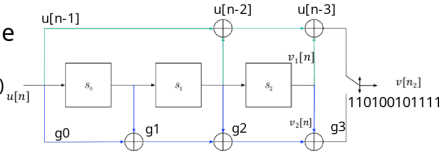
⊕111 = 000

⊕111 = 101

convolutional code / folding code

encode

101(000)



110100101111

u[n]	s_0	s_1	s_2	$v_1[n]$	$v_2[n]$	$v[n_2]$
-	0	0	0	-	-	-
1	0	0	0	1	1	11
0	1	0	0	0	1	01
1	0	1	0	0	0	00
0	1	0	1	1	0	10
0	0	1	0	1	1	11
0	0	0	1	1	1	11
-	0	0	0	-	-	-

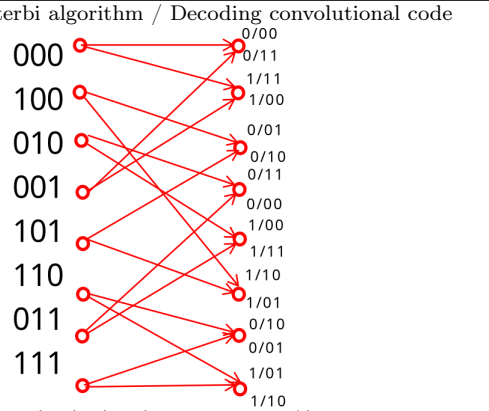
this algorithm is heavily dependent on the configuration, aka the more s states you have the more bits you get as the output.

$\{u[n]\} = (u_1, u_2, \ldots, u_n)$

$\{v[n]\} = \left\{\left[g^0 u_{[n]} + g^1 u[n-1] + g^2 u[n-2] + g^3 u[n-3]\right], g^0 u[n+1-m], \ldots,\right\}$

$\{v[n]\} = \sum_{m=0}^{M} g^m \, u_{[n-m]}$

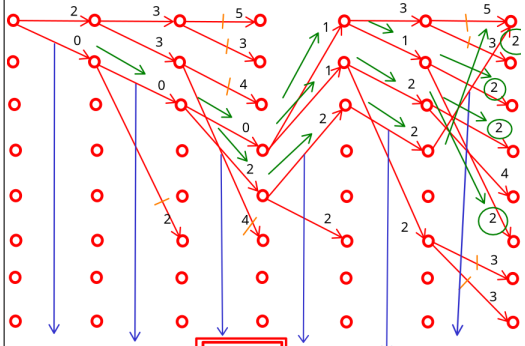
Viterbi algorithm / Decoding convolutional code



now check what the output was with the values of the changes above the 1/10… 0/00…

consider the output from above with 2 errors: 1101**1**1101**1**1

the numbers on the line represent the difference to the code given
->Hamming Distance



1/11 0/01 1/00 0/10 0/11 0/11

!!corrected error!!

-> 101000

There is a big problem with this algorithm, as you can see there were multiple ways of solving it, and the others would NOT have given the same code. So multi error can still be tricky.

Input - Output ratio for convolutional codes

$R = \frac{input}{output}$ in example $R = \frac{1}{2}$

RSA

RINKEL DONE

