La compression

- Les fichiers sont tous numériques (son, vidéo) et les volumes sont de plus en plus importants.
- Il faut optimiser le stockage **ET** la transmission, donc la taille des données.
- La *théorie de l'information* (Shannon) permet de formaliser le problème de la représentation de l'information.
- Il y a deux grandes familles d'algorithmes de compression :
  1. Algorithmes **avec perte** d'information.
     Le reconstruction des données n'est pas parfaite. Ces méthodes sont surtout utilisées pour le codage du son, d'images et de vidéos. Ils exploitent les caractéristiques de l'oeil et de l'oreille afin que l'information perdue reste imperceptible à ces sens.
     Exemples : les format `mp3`, `JPEG` et `MPEG`.
  2. Algorithmes **sans perte** d'information.
     Applicables à tout type de données. Les performances sont souvent moins bonnes que pour les méthodes avec perte, mais on est sûr de récupérer les données initiales.

Codage de données

- Soient $\Omega = \{\omega_1, \ldots, \omega_N\}$ des données (lettres, sons) à coder.
  Un code binaire est une application C de Ω dans l'ensemble des mots binaires de longueur finie :

$$C : \Omega \longrightarrow A^+ = \bigcup_{k \geq 1} \{0,1\}^k = \{0, 1, 00, 01, 10, 11, \ldots, 101001, \ldots\}$$

- Propriétés :
  1. le code doit être **uniquement décodable** :
     $\forall x, y \in \Omega : x \neq y \Rightarrow C(x) \neq C(y)$, càd., que C est injective ;
  2. le code doit être un **code préfixe** :
     aucun code $C(x)$ n'est le préfixe d'un autre code $C(y)$, pour tout $y \neq x$.

  Ces propriétés permettent un décodage facile.

Codage de données

- *Exemples :*

Ω	C_1	C_2	C_3	C_4	C_5
ω_1	00	0	00	000	1
ω_2	10	01	01	011	01
ω_3	110	001	10	101	001
ω_4	00	000	11	110	000

  Seuls les codes C_3, C_4 et C_5 vérifient les propriétés.

  La suite 0000111001010001 représente

  $\omega_1\omega_1\omega_4\omega_3\omega_2\omega_2\omega_1\omega_2$ dans le code C_3,
  $\omega_4\omega_2\omega_1\omega_1\omega_2\omega_4\omega_1$ dans le code C_5.

- Pour $\omega \in \Omega$, on note $l_C(\omega)$ le nombre de bits de ω dans le code C, *i.e.* la longueur de $C(\omega)$.

  Exemple : $l_{C_3}(\omega_1) = 2$, $l_{C_4}(\omega_1) = 3$ et $l_{C_5}(\omega_1) = 1$.

Codage de données

- *Exemple :* pour coder le texte $\omega_1\omega_2\omega_1\omega_1$
  le code C_3 utilise 8 bits, le code C_4 12 bits et le code C_5 5 bits.
  Le code C_5 est donc le moins coûteux pour ce texte.
- Pour formaliser ceci, on affecte à chaque élément $\omega \in \Omega$ une **probabilité** ou **fréquence d'apparition** $p(\omega) \in [0, 1]$, on définit alors la longueur moyenne du code C :

$$L(C) = \sum_{\omega \in \Omega} p(\omega) l_C(\omega)\,.$$

- *Exemple* :
  - Pour tout quadruplet $(p_1, p_2, p_3, p_4) \in (\mathbb{R}_+)^4$, avec $\sum_{i=1}^{4} p_i = 1$, on a $L(C_3) = 2$ et $L(C_4) = 3$.
  - Une probablité adaptée au texte $\omega_1\omega_2\omega_1\omega_1$ est $(3/4, 1/4, 0, 0)$, on a alors $L(C_5) = \frac{3}{4} * 1 + \frac{1}{4} * 2 = 1,25$.

Codage de données

1. Le modèle ci-dessus suppose que les apparitions des ω sont *indépendantes*.

   Mais, par exemple en français, on peut trouver la suite de lettres “texte” mais pas “eettx”.
   Les formats `gzip` et `rar` utilisent des méthodes combinées, plus performantes, qui codent en partie des suites finies de Ω.

2. Le code C_4 est toujours le plus coûteux, mais il permet la *détection d'erreurs* de transmission.
   - En effet, $C_4(\omega)$ est déduit de $C_3(\omega)$ de façon à ce que la somme des bits soit paire.
   - Lorsque dans un code on perd un seul bit, on détecte facilement l'erreur, mais on ne peut pas la corriger.
   - *Exemple :* 001 est certainement pas un mot du code C_4, il y a une erreur de transmission. Mais 001 peut être le code de ω_1 de ω_2 ou de ω_3 !
     Ainsi, si l'on reçoit le message 001 110, il peut être le code de $\omega_1\omega_4$, $\omega_2\omega_4$ ou $\omega_3\omega_4$.

Codage de Huffman

- D.A. Huffman, 1925-1999, États-Unis, travaux en théorie de l'information et codage.
  Codage découvert en 1951, lors d'un projet de fin d'études.
- L'idée du codage de Huffman est proche de celle utilisée dans le code Morse : statistiques d'apparition des caractères.
- Inconvénients : il faut connaître les fréquences des caractères et fournir le code avec le texte.
- Avantages : si C est un codage de Huffman et $\tilde{C}$ un codage binaire uniquement décodable, alors $L(C) \leq L(\tilde{C})$.
  Ceci se démontre sous l'hypothèse de données composées d'une suite de caractères indépendants.
- Le codage de Huffman est peu utilisé tout seul.
  En général, on le combine avec d'autres méthodes, par exemple dans `.gzip`, `.png`, `.jpeg`,`.mp3`.

Algorithme de Huffman

Données : Lettres $\{\omega_1, \ldots, \omega_N\}$ et probabilités associées $\{p_1, \ldots, p_N\}$.

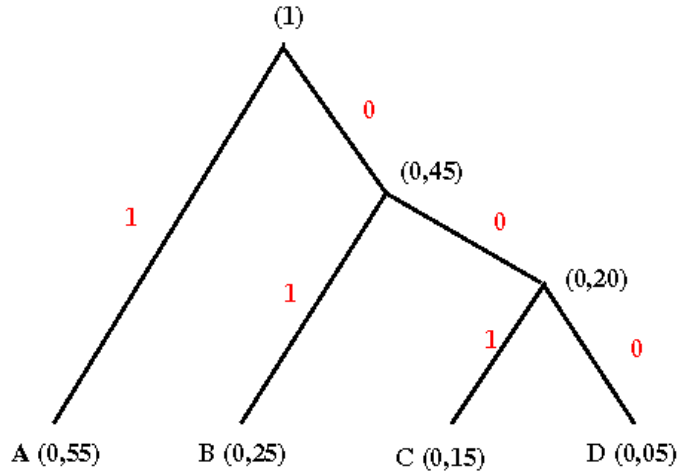
1. Créer une liste des lettres classée dans l'ordre des probabilités décroissantes.
   Chaque lettre est une feuille de l'arbre.
2. Tant qu'il a plus d'un élément dans la liste, répéter :
   - 2.1 associer les deux noeuds de plus petite probabilité, la probabilité du nouveau noeud est la somme des probabilités des enfants ;
   - 2.2 supprimer de la liste les deux noeuds réunis et les remplacer par le nouveau noeud ;
   - 2.3 réordonner la nouvelle liste.
3. Le dernier noeud de la liste sera la racine de l'arbre.
4. Associer le code "1" à la branche de gauche et le code "0" à la branche de droite.
5. Remonter l'arbre à partir de la racine, en ajoutant à chaque fois au code un "0" ou un "1", suivant la branche suivie.
   En arrivant à la feuille, on a le code binaire de la lettre associée.

Algorithme de Huffman

- L'arbre est habituellement représenté la racine en haut et les feuilles en bas :
  on lit le code en allant de la racine vers la feuille.
- C'est le fait d'aller de la racine vers les feuilles qui permet d'obtenir un code préfixe.
- L'arbre, et donc le code, que l'on obtient n'est pas unique.
- Pour décoder à partir de l'arbre, il suffit de remonter depuis la racine jusqu'au feuilles, en suivant les branches en fonction de la suite des "0" et "1".
- *Exemple 1* : On considère les lettres Ω ={A, B, C, D} auxquelles on associe les probabilités
  $p_1 = 0,55$, $p_2 = 0,25$, $p_3 = 0,15$ et $p_4 = 0,05$.

Algorithme de Huffman. Exemple 1

1. Trier la liste.
2. Relier les noeuds deux à deux en regroupant les fréquences d'apparition les plus faibles. Dans cet exemple, l'ordre ne change pas.
4. Coder les branches, p.ex. "1" à gauche et "0" à droite.



5. Lire les codes, de la racine vers les feuilles :
Code(A)=1, Code(B)=01, Code(C)=001 et Code(D)=000 .

Algorithme de Huffman. Exemple 1

- On a Code(A)=1, Code(B)=01, Code(C)=001 et Code(D)=000

  C'est un code préfixe à longueur variable, uniquement décodable pour l'ensemble de lettres Ω ={A,B,C,D}.
- *Exemple* de décodage 1011001 = 1 01 1 001
  ce qui est le code de ABAC
- La longueur moyenne du code est
  $L(\text{Code}) = 0,55 * 1 + 0,25 * 2 + 0,15 * 3 + 0,05 * 3 = 1,65$
- Un code de longueur fixe aura une longueur moyenne de 2.
  D'où une compression moyenne de 17,5% .
- Soit le code définie par
  Code2(A)=00, Code2(B)=01, Code2(C)=10 et Code2(D)=11 .
  Pour ABAC on a le codage sur 8 bits 00010010,
  ce qui fait une compression réelle de 12,5% .

  Que peut-on dire des mots AA et DC ?

Algorithme de Huffman. Exemple 2

On considère l'ensemble Ω={ espace, A, C, D, E, R, S, T }
avec les probabilités

ω	espace	A	C	D	E	R	S	T
$p(\omega)$	0,26	0,06	0,15	0,04	0,24	0,08	0,12	0,05

1. Ordonner la liste de façon décroissante :
   0,26(esp) 0,24(E) 0,15(C) 0,12(S) 0,08(R) 0,06(A) 0,05(T) 0,04(D)

2.1 Créer un nouveau noeud, de probabilité la somme des anciennes probabilités :
0,26(esp) 0,24(E) 0,15(C) 0,12(S) 0,08(R) 0,06(A) 0,05(T) 0,04(D)

0,09 (T-D)

2.2-3 Supprimer les anciens noeuds et réordonner la liste :
0,26(esp) 0,24(E) 0,15(C) 0,12(S) 0,09(T-D) 0,08(R) 0,06(A)

Algorithme de Huffman. Exemple 2

2. Créer le nouveau noeud, supprimer les anciens, réordonner :
0,26(esp) 0,24(E) 0,15(C) 0,12(S) 0,09(T-D) 0,08(R) 0,06(A)

0,14 (R-A)

2. Créer le nouveau noeud, supprimer les anciens, réordonner :
0,26(esp) 0,24(E) 0,15(C) 0,14 (R-A) 0,12(S) 0,09(T-D)

0,21 (S-(T-D))

2. Créer le nouveau noeud, supprimer les anciens, réordonner :
0,26(esp) 0,24(E) 0,21(S-(T-D)) 0,15(C) 0,14 (R-A)

0,29 (C-(R-A))

Algorithme de Huffman. Exemple 2

2. Créer le nouveau noeud, supprimer les anciens, réordonner :
0,29(C-(R-A)) 0,26(esp) 0,24(E) 0,21(S-(T-D))

0,45 (E-(S-(T-D)))

2. Créer le nouveau noeud, supprimer les anciens, réordonner :
0,45 (E-(S-(T-D))) 0,29(C-(R-A)) 0,26(esp)

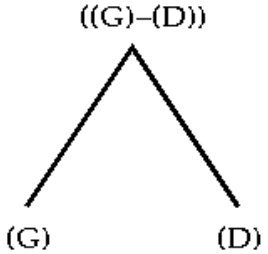
0,55 ((C-(R-A))-esp)

2. Créer le nouveau noeud, supprimer les anciens, réordonner :
0,55 ((C-(R-A))-esp) 0,45 (E-(S-(T-D)))

1 (racine= (((C-(R-A))-esp)-(E-(S-(T-D)))))

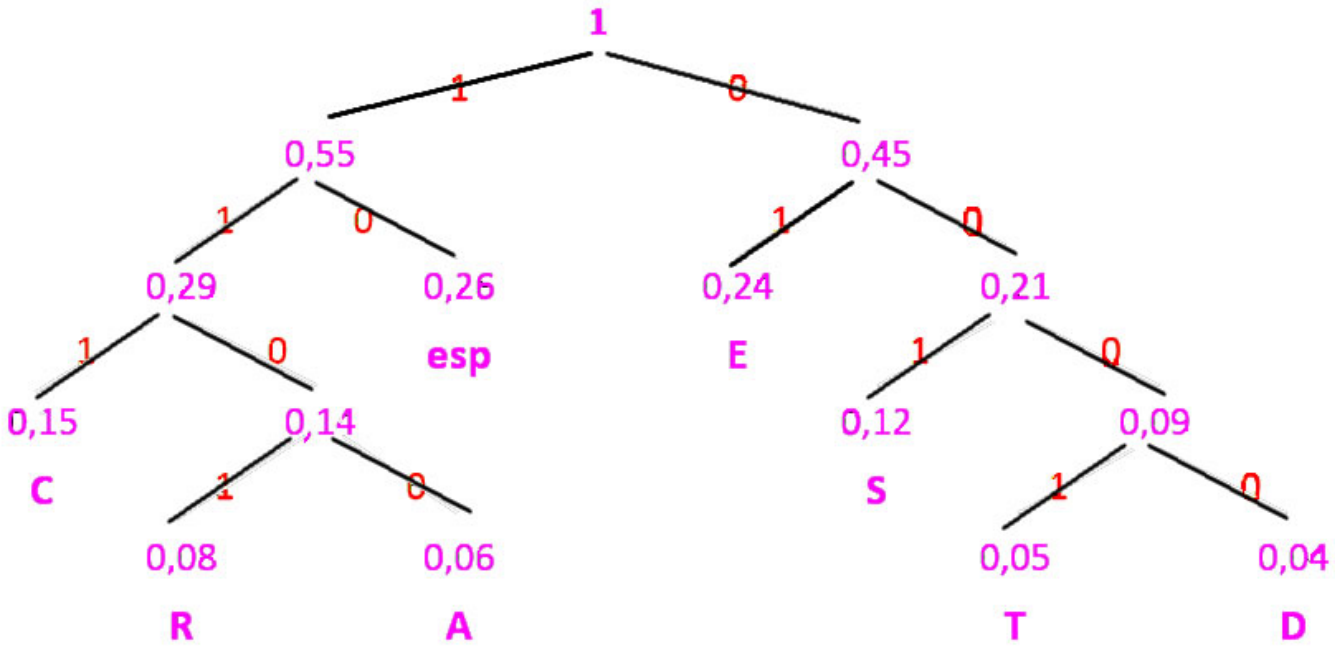
Algorithme de Huffman. Exemple 2

- La racine est composée de (((C-(R-A))-esp)-(E-(S-(T-D))))
- On construit l'arbre binaire, à partir de la racine, en interprétant (G)-(D) comme étant un noeud ayant comme enfant gauche (G) et comme enfant droit (D) :



- Une fois l'arbre binaire obtenu, on code les branches : "1" à gauche et "0" à droite.
- Pour obtenir les codes des éléments de Ω il suffit de lire l'arbre de la racine vers les feuilles.

Algorithme de Huffman. Exemple 2



L'arbre binaire avec les codes binaires et lettres/probabilités.

Algorithme de Huffman. Exemple 2

ω	espace	A	C	D	E	R	S	T
$p(\omega)$	0,26	0,06	0,15	0,04	0,24	0,08	0,12	0,05
code	10	1100	111	0000	01	1101	001	0001

- La longueur moyenne du code est 2,730
  $= 2 * (0,26 + 0,24) + 3 * (0,15 + 0,12) + 4 * (0,08 + 0,06 + 0,05 + 0,04)$
- En théorie de l'information, on montre que la borne inférieure théorique de la longueur moyenne d'un code pour Ω est l'entropie de Ω, définie par

$$H(\Omega) = - \sum_{\omega \in \Omega, p(\omega) > 0} p(\omega) \log_2(p(\omega)) .$$

Or ici $H(\Omega) = 2,714$, l'algorithme de Huffman s'approche donc très bien de cette valeur optimale.

Algorithme de Huffman. Exemple 2

- Décodage facile quand l'arbre est donné : il suffit de descendre de la racine de l'arbre vers une feuille, en suivant les bits.
- Décodez le message suivant :
  0000010011111100110100010100110000111011100111011000
  0001001100111111001101000100110

  On obtient : DESCARTES TRACES DES ECARTS
- Le décodage se fait sans ambiguïté car aucun code n'est le préfixe d'un autre.
- Le décodage peut être fait au fur et à mesure, sans attendre la fin du message.
- Si les "poids" des lettres représentent leur fréquence d'apparition dans le texte, la construction garantit que les symboles de plus forte probabilité ont les codes les plus courts.

Codage de Gray

- Frank Gray, Etats-Unis, brevet déposé en 1951.
- Le code de Gray est également appelé *code binaire réfléchi*.
- C'est un codage binaire dans lequel deux valeurs successives ont des codes qui diffèrent que d'un seul bit.

décimal :	0	1	2	3	4	5	6	7
binaire :	000	001	010	011	100	101	110	111
code de Gray :	000	001	011	010	110	111	101	100

- Utilité :
  lorsque l'on transmet des données dont les valeurs ne varient que d'une unité, le code de Gray permet un codage très facile.

  En effet, on ne change qu'un bit à la fois et
  on peut détecter s'il y a des erreurs de transmission.

Codage de Gray

Applications :

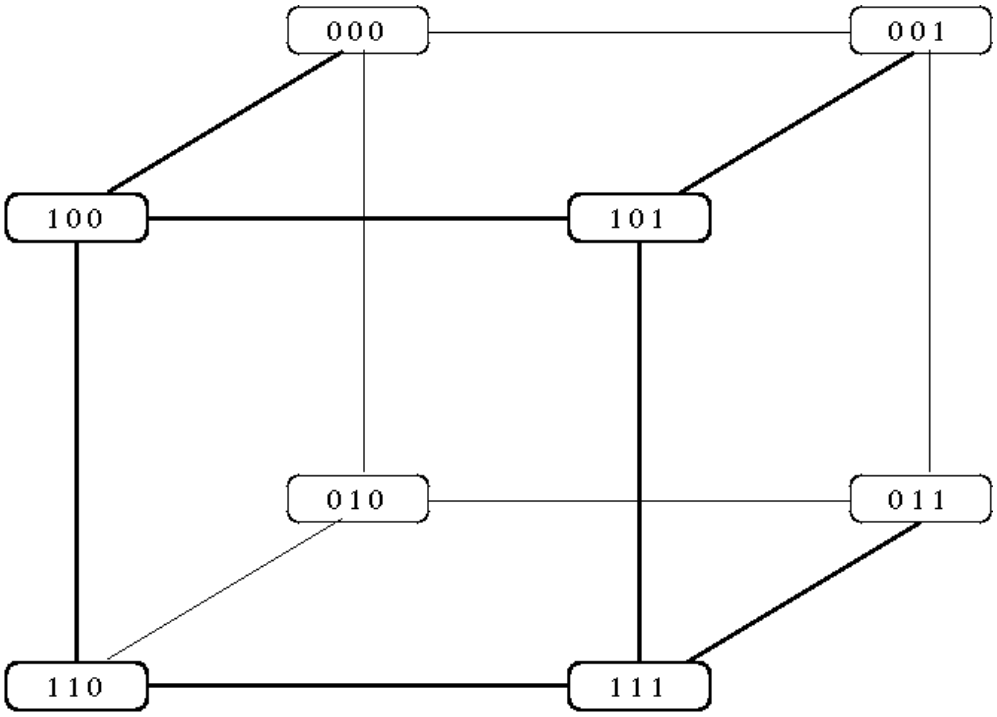
1. L'ingénieur Émile Baudot a utilisé un codage de ce type dès 1878 en télégraphie.
2. F. Gray voulait coder des signaux analogiques en digital et minimiser les erreurs lors de la conversion.
3. Dans les convertisseurs analogique/digital actuels, on utilise toujours le code de Gray.
4. On représente les états de capteurs de positions : règles optiques, roues codeuses, souris mécaniques.
5. Tables de Karnaugh.
6. Permet la résolution de problèmes tels les *Tours de Hanoi* ou *Jeu du baguenaudier (anneaux chinois)*.

⋮

Codage de Gray

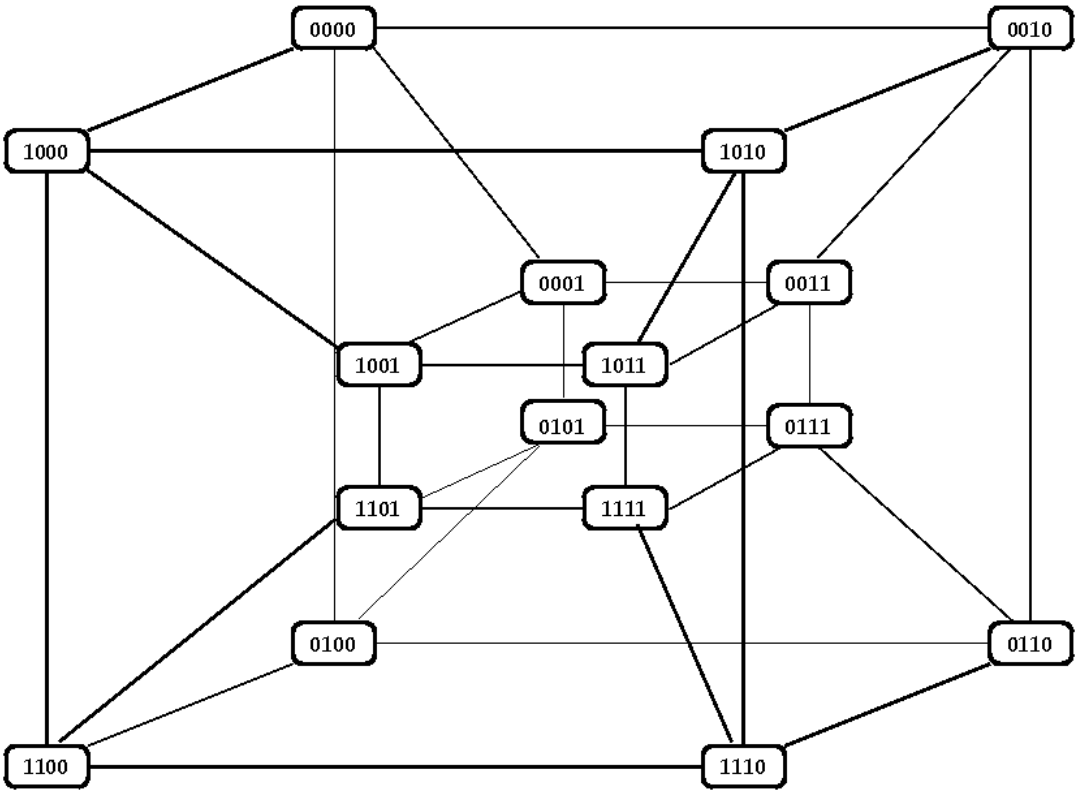
Le code de Gray sur n bits se construit à partir de l'hypercube de $\mathbb{R}^n$ dont les sommets sont les mots binaires de longueur n.

Pour $n = 3$ on a :



Codage de Gray

Pour $n = 4$:



Calculer sans retenue

- Rappel : pour représenter un nombre en base b, on utilise l'alphabet $\{0, 1, \ldots, b-1\}$.
  On faisant la somme de deux chiffres, on est en général obligé d'utiliser le principe de position pour représenter le résultat.
  Exemples : en base $b = 10$, on a $7 + 8 = 15$,
  en base $b = 2$, $1 + 1 = 10$.
  En utilisant le 0 et le principe de position, on peut noter n'importe quel entier grâce aux chiffres $\{0, 1, \ldots, b-1\}$.
- Mais dans certaines applications, il faut tenir compte de la périodicité de ce qui est représenté. On compte
  1. les heures de la journée de 1h à 24h ;
  2. les jours de la semaine de lundi (1) à dimanche (7) ;
  3. les angles sont mesurés de 0° à 360° ;
  4. le signal d'un phare passe de façon périodique toutes les x sec. ;
  5. une roue mécanique à m dents a fait un tour entier quand les m dents se retrouvent dans leur position initiale ;

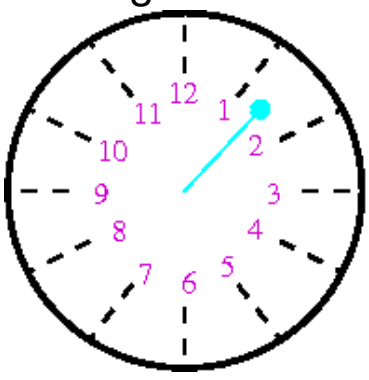
  :

Calcul modulaire

- Pour formaliser ces problèmes, on utilise la *théorie de nombres*, plus précisément l'arithmétique modulaire, encore appelée le calcul modulaire.
- En Europe ce sont *Euclide* (approx. de -325 à -265) et *Diophante d'Alexandrie* (approx. de 210 à 284) et en Chine *Sun Zi* (vers 300) et *Qin Jiushao* (approx. de 1202 à 1261), qui étudient des problèmes de ce type.
- Applications aujourd'hui :
  Calcul de clés de contrôle, codes correcteurs, cryptographie, ...
- Définitions : soient a et b deux entiers relatifs, on dit que
  1. a et b sont premiers entre eux si leur plus grand commun diviseur est 1. On note $\text{pgcd}(a, b) = 1$;
  2. a divise b s'il existe $k \in \mathbb{Z}$ tel que $b = ka$. On note $a|b$.

Calcul modulaire : exemple

Horloge avec une aiguille et 12 graduations de 1 à 12 :



- Ici 12 est identique à 0h et 24h,
  tandis que 18h est identifié à 6h,...
- De façon générale, pour $r \in \{1, \ldots, 12\}$,
  r *représente* les entiers de la forme $n = r + 12k$ où $k \in \mathbb{Z}$.
- En remplaçant 12 par 0, donc pour $r \in \{0, 1, \ldots, 11\}$,
  on peut dire que r représente les entiers n tels que le reste de la division euclidienne de n par 12 est r. C'est l'ensemble

$$\{\ldots, r-48, r-36, r-24, r-12, r, r+12, r+24, r+36, r+48, \ldots\}$$

Calcul modulaire : propriétés

Soit $p \in \mathbb{N} \setminus \{0, 1\}$, alors

1. a et b sont congrus modulo p, si et seulement si $a - b$ est un multiple de p, c'est-à-dire si $p|(a - b)$. On note $a \equiv b \mod p$.
2. On a $a \equiv b \mod p$ si et seulement si a et b ont le même reste dans la division euclidienne par p. Soit r ce reste, alors $a = kp + r$ pour $k \in \mathbb{Z}$ et $r \equiv a \mod p$.
3. Si $a \equiv b \mod p$ et si $b \equiv c \mod p$, alors $a \equiv c \mod p$.
4. Si $a_1 \equiv b_1 \mod p$ et $a_2 \equiv b_2 \mod p$, alors $a_1 + a_2 \equiv b_1 + b_2 \mod p$, $a_1 a_2 \equiv b_1 b_2 \mod p$, et pour tout entier $n \in \mathbb{Z}$, $na \equiv nb \mod p$.
5. Tout entier $m \in \mathbb{Z}$ a un *représentant* dans $F_p = \{0, 1, \ldots, p - 1\}$. Les calculs modulo p sur les entiers peuvent ainsi se ramener aux calculs modulo p dans F_p.

Calcul modulaire

- *Exemple :* calculer $x \equiv 23^6 \mod 7$ et $y \equiv 233^6 \mod 7$, or $23^6 = 148035889 = 7 * 21147984 + 1$ et $233^6 = 160005726539569$! mais modulo 7 on a : $23^6 \equiv 2^6 = 2^3 \cdot 2^3 \equiv 1 \cdot 1$
- On représente les opérations dans F_p grâce à des tableaux :
  - Pour $p = 2$, $F_2 = \{0, 1\}$ et

+	0	1
0	0	1
1	1	0

*	0	1
0	0	0
1	0	1

  - Pour $p = 3$, $F_3 = \{0, 1, 2\}$ et

+	0	1	2
0	0	1	2
1	1	2	0
2	2	0	1

*	0	1	2
0	0	0	0
1	0	1	2
2	0	2	1

  - Pour $p = 4$, $F_4 = \{0, 1, 2, 3\}$ et

+	0	1	2	3
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

*	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	0	2
3	0	3	2	1

Calcul modulaire : clé de sécurité

- Pour éviter de erreurs lors de la saisie d'un numéro, comme un chiffre erroné ou la permutation de chiffres, on adjoint souvent une **clé de sécurité**.
- *Exemple :* numéro INSEE ou «numéro de sécurité sociale»
  Il est composé de 13 chiffres, $s = s_{12}s_{11}\ldots s_0$,
  et permet de coder le sexe, l'année, le mois, le département de naissance,*etc.* d'un individu
  et d'une clé de sécurité $c(s)$ à deux chiffres calculé comme suit

$$c(s) = 97 - (s \mod 97) \in \{1, \ldots, 97\}.$$

  Le calcul modulo le nombre premier 97 permet de détecter des erreurs de saisie dans s (*cf.* TD).
- D'autres exemples sont les codes barres EAN 13, le code ISBN,...

Calcul modulaire : Algorithme RSA

- Proposé par Rivest, Shamir et Adleman en 1977.
- Algorithme de cryptographie asymétrique,
  très utilisé dans le commerce électronique (protocole SSL).
- C'est un algorithme à clé publique CPub pour chiffrer
  et à clé privée CPriv pour déchiffrer un message :
  - Alice engendre les clés CPub et CPriv ;
  - Alice envoie CPub à Bob ;
  - Bob utilise CPub pour coder son message **M** en **C(M)**,
    il envoie **C(M)** à Alice par un canal non sécurisé ;
  - Seule Alice peut décoder **C(M)** grâce à CPriv, et retrouver **M**.

Calcul modulaire : Algorithme RSA

- Calcul des clés :
  1. Choisir deux nombres premiers distincts (et grands) p et q ;
  2. Calculer $n = pq$ et $\varphi = (p-1)(q-1)$;
  3. Choisir e, avec $1 < e < \varphi$ et $\text{pgcd}(e, \varphi) = 1$;
  4. Déterminer l'unique entier d vérifiant $e\,d \equiv 1 \mod \varphi$.

  Alors CPub = (n, e) et CPriv = d
- Chiffrement du message :
  Si M est un entier à chiffrer, $0 \leq M < n$, alors $C(M) \equiv M^e \mod n$.
- Déchiffrement du message :
  Si $C(M)$ est un entier chiffré, alors $M \equiv C(M)^d \mod n$.
- La force de RSA est qu'il n'existe actuellement pas d'algorithme rapide pour factoriser un entier n grand en ses facteurs.
  Un record datant de 2009 a permis de factoriser un nombre de 768 bits.
  Or les clés RSA sont habituellement de longueurs 1 024, 2 048 ou 4096 bits...

Calcul modulaire : Factorisation RSA-768

Le nombre *RSA-768* a 232 décimales en base 10 et 768 bits en base 2. Il a été factorisé en 2009, après 31 mois de calculs avec plusieurs centaines de machines, en deux nombres premiers de 116 décimales.

RSA-768

= 123018668453011775513049495838496272077285356959533479219732245215172640050726365751874520219978646938995647494277406384592519255732630345373154826850791702612214291346167042921431160222124047927473779408066535141959745985690214341 3

= 334780716989568987860441698482126908177047949837137685689124313889828837938780022876147116525317430877378144679994893

$\times$

367460436667995904282446337996279526322791581643430876426760322838157396665112792333734171433968102700927987363089 17

Calcul modulaire : Factorisation RSA-768

Publication : *Factorization of a 768-bit RSA modulus*
in *Cryptology ePrint Archive : Report 2010/006*
par *Kleinjung, Aoki, Franke, Lenstra, Thomé, Bos, Gaudry, Kruppa, Montgomery, Osvik, te Riele, Timofeev, Zimmermann.*
Extrait :

> *Our computation required more than* 10^{20} *operations. With the equivalent of almost 2000 years of computing on a single core 2.2GHz AMD Opteron, on the order of* 2^{67} *instructions were carried out. The overall effort is sufficiently low that even for short-term protection of data of little value, 768-bit RSA moduli can no longer be recommended. This conclusion is the opposite of the one arrived at on [39], which is based on a hypothetical factoring effort of six months on 100 000 workstations, i.e., about two orders of magnitude more than we spent.*

Voir aussi `http://fr.wikipedia.org/wiki/RSA-768`

Deuxième partie II

La logique