Table des matières

Notions de modélisation et complexités

1 Introduction

Rapport du jury 2015 : "La ligne directrice du calcul formel est la recherche de l'effectivité, puis de l'efficacité (souvent en temps, parfois en espace) du calcul algébrique,.."
"Plus largement, une réflexion minimale sur les ordres de grandeur (est-ce qu'un calcul faisable représente $10^1, 10^{10}, 10^{100}, 10^{1000}$ opérations élémentaires) permettrait souvent de mieux restituer les problèmes soulevés par un texte, ou de proposer des valeurs de paramètres réalistes quand ce sujet n'est pas évoqué par le texte."
"Les candidats ayant le réflexe de se saisir, seuls, d'une question de complexité, sont perçus très positivement par le jury."

Définition. — Une structure algébrique (groupe,anneau,corps,espace vectoriel,...) est dite **effective** si l'on dispose :
- d'une structure de données pour en représenter les éléments,
- d'algorithmes pour en effectuer les opérations et pour y tester l'égalité et autres prédicats.

- L'anneau $\mathbb{Z}$ est effectif. Le corps $\mathbb{Q}$ et les corps finis $\mathbb{F}_p, \mathbb{F}_q$ sont effectifs.
- Si K est un corps effectif, alors K^n et $M_n(K)$ sont effectifs. Une clôture algébrique $\overline{K}$ est alors elle aussi effective.
- Si A est un anneau effectif, alors $A[X]$ est effectif.

Règles générales de complexité :
- Le temps d'exécution d'une séquence d'instructions est la somme des temps de chaque instruction.
- Le temps d'un branchement est le temps du pire cas.
- Le temps d'une boucle est le produit du nombre de passages dans la boucle par les instructions de la boucle, si les instructions prennent des entrées de taille constante.
- Le temps d'un programme est minoré par le temps d'affichage du résultat.

Pourquoi s'intéresser à la complexité ?
- Pour évaluer les ressources nécessaires à priori.
- Pour comparer deux algorithmes traitant d'un même problème.
- Pour savoir ce qui est possible ou non à priori. (utile en cryptographie)

Capacités actuelles de calcul : 2^{60} opérations en temps raisonnable, et 10^{10} octets de mémoire.
Elles augmentent d'année en année et sont à prendre en compte dans la résolution de problèmes/cryptographie.

2 Complexités et algorithmes usuels

2.1 Tri de listes et remarques

- L'algorithme naïf a une complexité $O(n^2)$. La complexité optimale est de $O(n\ln(n))$.
- L'algorithme de tri-fusion a une complexité $O(n\ln(n))$.

Remarque. — La somme/comparaison de deux entiers $\leq n$ demande $O(\log(n))$ opérations.
Si l'on note m le nombre de bits de ces entiers, ces opérations demandent $O(m)$ opérations. Cette complexité linéaire en la taille des entrées, et les complexités de toutes les autres opérations que l'on voudra faire sur des entiers/polynômes seront plus élevées que cela (quadratique, polynômiale, sous-exponentielle, voire exponentielle dans des cas naïfs).
Rien qu'une complexité en $O(n^3)$ est déjà gênante en termes de calculs si n peut être vraiment très grand dans les cas concrets considérés. (matrices stochastiques ou polynômes de très grande taille par exemple)

2.2 Exponentiation rapide

Pour $(G, \times)$ un groupe et $x \in G$, on veut calculer x^n.
L'algorithme naïf demande n multiplications dans G.
L'exponentiation binaire utilise l'écriture binaire de n : $n = (n_{l-1}, \ldots, l_0)$ afin de calculer x^n.
On peut la coder par la gauche ou par la droite.
L'exponentiation binaire par la gauche commence avec $b = 1$, et lit les l_i de gauche à droite.
Si $l_i == 1$ alors $b = b^2$ et $b = b \times x$. Si $l_i == 0$, alors $b = b^2$.
Cela demande au plus $2.\log_2(n)$ multiplications dans G, donc $O(log(n))$ multiplications.

2.3 Calcul de complexité d'algorithmes récursifs

Théorème. Théorème du maître ou Diviser pour mieux régner —
Si la fonction C du côut de calcul est croissante et satisfait : $C(n) \leq mC(\frac{n}{p}) + T(n)$
pour p un entier, $\forall n = p^e$, et pour T fonction positive tq $T(n) \leq a.n^{C(1)}$, alors pour $k := C(1)$, on a :

- $C(n) = O(n^k)$ si $m < p^k$
- $C(n) = O(n^k.\log(n))$ si $m = p^k$
- $C(n) = O(n^{\log_p(m)})$ si $m > p^k$

Cela se démontre en écrivant par récurrence : $C(n) \leq \sum_{i=0}^{e-1} m^i.T(p^{e-i}) + m^e.C(1)$ et en distinguant les 3 cas.

Exemple. — Les algorithmes de tri-fusion, de création d'arbre, d'évaluation multi-point, d'interpolation, de multiplication de Karatsuba, de Transformée de Fourier Rapide, ou de factorisation de polynômes sur des corps finis/$\mathbb{Z}$ utilisent ce principe pur diminuer leur complexité.

2.4 Algorithmes probabilistes

Certains algorithmes vont avoir recours à des tirages aléatoires d'éléments d'un ensemble pour être efficaces.
On suppose toujours que l'on peut obtenir de bons jets aléatoires pour un coût fixe, ce qui fait que la classe des algorithmes aléatoires contient des algorithmes plus efficaces que ceux déterministes.
On en distingue trois types :

- **Las Vegas** : Peut ne pas s'arrêter mais s'arrête toujours sur la bonne réponse.
- **Monte Carlo** : S'arrête toujours. Si la réponse est affirmative alors elle est bonne, mais peut se tromper dans le cas d'une réponse négative.
- **Atlantic City** : S'arrête toujours. Peut se tromper, mais la proba d'une mauvaise réponse est plus faible que la proba d'une bonne réponse.

Exemple. — Avec un algorithme de type Las Vegas on peut déterminer plus rapidement une racine carrée de $a \in \mathbb{F}_p$ pour $p \equiv 3\ mod(4)$. (Si $p \equiv 1\ mod(4)$ alors $a^{\frac{p-1}{2}}$ convient.)

2.5 Inversion de matrice, déterminant, systèmes linéaires

- Pour $A \in M_n(K)$, on calcule en général A^{-1} et det(A) par Pivot de Gauss.
  Le pivot de Gauss nécessite $O(n^3)$ opérations dans K.
- Pour un système linéaire Ax=b, si $A \in S_n^{++}(\mathbb{R})$, on peut construire une suite $(x_k)_k$ qui va converger et stationner en $A^{-1}b$ en au plus n étapes avec la méthode du gradient à pas conjugué. (On peut aussi utiliser la méthode du gradient à pas optimal)

3 Tests de primalité

Tester si un nombre est premier en un temps raisonnable n'est pas simple à mettre en place.
Tout comme la factorisation de polynômes, les meilleurs algorithmes connus sont probabilistes.
Pour des nombres entiers très grands (de l'ordre de 500 bits), on va souvent se limiter à l'utilisation d'un test de primalité probabiliste avec suffisamment de tirages pour que la probabilité d'erreur soit infime afin de pouvoir considérer le nombre qui a résisté au test comme premier et l'utiliser dans des applications cryptographiques sans inquiétude.

Définition. Nombres de Carmichael — Un nombre entier n est de Carmichael si pour tout entier a premier avec n, on a $a^n \equiv a$ mod(n).

Proposition. — Tous les nombres premiers sont de Carmichael, et un nombre de Carmichael est forcément impair.
Un nombre entier n est de Carmichael si et seulement si n est sans facteur carré et si pour tout p facteur premier de n, $p-1|n-1$.
Il existe une infinité de nombres de Carmichael. (1992)

Les nombres $561 = 3.11.17$ et $1105 = 5.13.17$ sont les deux plus petits nombres de Carmichael non premiers.

Définition. Test de Solovay-Strassen — Soit $n \geq 3$ impair et a un entier premier avec n.
On dit que n vérifie la condition de Solovay-Strassen en base a ssi : $a^{\frac{n-1}{2}} \equiv (\frac{a}{n})$ mod(n) Où $(\frac{a}{n})$ désigne le symbole de Jacobi.

Proposition. — Les nombres premiers vérifient toujours la condition de Solovay-Strassen.
Si n n'est pas premier, alors n ne vérifie pas la condition de Solovay-Strassen en base a pour au moins la moitié des $a \in \{2, \ldots, n-1\}$.
Dans le cas où a n'est pas premier avec n, un calcul de pgcd permet de montrer que n n'est pas premier.
Sinon, on montre que l'ensemble des a tels que n est de Solovay-Strassen en base a est un sous-groupe strict de $(\mathbb{Z}/n\mathbb{Z})^*$, donc de cardinal au moins deux fois moindre.
Algorithmiquement, on va tirer un certain nombre de a aléatoirement dans $\{2, \ldots, n-1\}$ et vérifier si n est de Solovay-Strassen pour chacun de ces a.
La complexité de bons algorithmes de test de Solovay-Strassen est de $O(k.\log(n)^3)$, où k est le nombre maximum de fois où l'on tire aléatoirement un entier a pour calculer un symbole de Jacobi avec lui. La probabilité de faux-positif du test est alors inférieure à $\frac{\log(n)}{2.2^k}$.
Dans les applications en cryptographie, si l'on choisit une valeur suffisamment grande de k, par exemple 100, la probabilité pour que l'algorithme se trompe est si petite que l'on peut considérer le nombre qui a résisté au test comme premier et l'utiliser dans des applications cryptographiques sans inquiétude.

Définition. Test de Miller-Rabin —
Soient e,m des entiers avec m impair, et soit $n = 2^e m+1$. On dit que m vérifie la condition de Miller-Rabin en base a si :

- ou bien $a^m \equiv 1$ mod(n)
- ou bien il existe un $0 \leq f < e$ tel que $a^{2^f m} \equiv -1$ mod(n)

Proposition. — Les nombres premiers vérifient toujours la condition de Miller-Rabin.
Si n vérifie la condition de Miller-Rabin en base a, alors $a^{n-1} \equiv 1$ mod(n).
Si n est de Miller-Rabin en base a, alors il est de Solovay-Strassen en base a.
Le test de Miller-Rabin est ainsi plus efficace que celui de Solovay-Strassen : Si n n'est pas premier, alors il est de Miller-Rabin pour au plus un quart des $a \in \{2, \dots, n-1\}$.
Algorithmiquement, on va tirer un certain nombre de a aléatoirement dans $\{2, \dots, n-1\}$ et vérifier si n est de Miller-Rabin pour chacun de ces a.
La complexité de bons algorithmes de test de Miller-Rabin est de $O(k.\log(n)^3)$, où k est le nombre maximum de fois où l'on tire aléatoirement un entier a pour calculer un symbole de Jacobi avec lui.
La probabilité de faux-positif du test est alors inférieure à $\frac{\log(n)}{2.4^k}$.
Le test de Miller-Rabin est très utilisé en cryptographie asymétrique pour engendrer les grands nombres premiers nécessaires pour le chiffrement RSA, et pour beaucoup des utilisations du chiffrement El Gamal ou de l'échange de clés Diffie-Hellman.

Les tests de Solovay-Strassen et de Miller-Rabin sont des tests probabilistes de type Monte-Carlo.

4 Opérations sur les polynômes

4.1 Evaluation d'un polynôme

- Pour $P \in \mathbb{R}[X]$ de degré n, l'évaluation classique demande $2n-1$ multiplications et n additions. On cherche à diminuer le plus possible le nombre de multiplications.
- **Méthode de Horner** : On écrit $P(X) = ((...(a_nX + a_{n-1})X + a_{n-2})...)X + a_0$, et l'évaluation demande n multiplications et n additions.
- Si $P(X) = a_0\Pi(X - a_i)$, on a besoin de n multiplications et n additions.
- Pré-processing : On cherche à exprimer $P(X)$ sous une autre forme pour diminuer le nombre de multiplications nécessaires, quitte à avoir quelques additions en plus.
- L'algorithme de Belaga demande $\lfloor \frac{n+1}{2} \rfloor + 1$ multiplications et n+1 additions.
- On montre de plus qu'il n'existe aucun algorithme d'évaluation avec moins de $\lfloor \frac{n+1}{2} \rfloor$ multiplications ou moins de n additions.
- **Forme orthogonale** : $P(X) = \sum_{k=0}^{n} b_k Q_k(X)$ avec $Q_{i+1} = (A_i.X + B_i)Q_i(X) - C_i.Q_{i-1}(X)$, $A_i \neq 0$, $Q_0 = 1$,$Q_{-1} = 0$.
  Cette méthode requiert 3n-1 multiplications et 3n-1 additions. On choisit en général les polynômes de Chebychev comme famille de polynômes orthogonaux.

Deux autres données importantes dans l'évaluation polynômiale sont la **stabilité** et le **conditionnement**.
Un algorithme est dit numériquement instable s'il créer des erreurs d'arrondis qui augmentent de manière incontrôlée.
Il est dit mal conditionné si une petite incertitude sur les données peut créer une grande incertitude sur le résultat.
L'algorithme de forme orthogonale avec les polynômes de Chebychev est le seul de ces algorithmes à être bien conditionné et stable, il est donc souvent plus intéressant à utiliser.

4.2 Multiplication de polynômes

- Pour un anneau A donné, on note $M(N)$ le nombre d'opérations dans A nécessaires pour multiplier deux polynômes de degré $\leq N$.
- L'algorithme de multiplication naïf demande $O(N^2)$ opérations.
- La méthode de Karatsuba demande $O(N^{\log_2(3)})$ opérations.
- La FFT (Fast Fourier Transform) demande $O(N\log(N)\log(\log(N)))$ opérations.

Ces résultats de complexité ont des analogues pour la multiplication d'entiers :
- On peut multiplier des entiers écrits avec N bits par :
- L'algorithme de multiplication naïf en $O(N^2)$ opérations.
- La méthode de Karatsuba en $O(N^{\log_2(3)})$ opérations.
- L'algorithme de Schönhage-Strassen en $O(N\log(N)\log(\log(N)))$ opérations.

Application. Somme de fractions rationnelles de degré 1—
On veut calculer P, Q tels que $\sum_{i=0}^{n-1} \frac{c_i}{X-a_i} = \frac{P}{Q}$.
On procède de façon récursive :

- Entrée : $(a_i)_{0\leq i\leq n-1}$,$(c_i)_{0\leq i\leq n-1}$ (Avec n une puissance de 2)
- Sortie : $\frac{P}{Q}$ ($= \sum_{i=1}^{n} \frac{c_i}{X-a_i}$).
- 1) Si n=1, renvoyer $\frac{c_0}{X-a_0}$.
- 2) Sinon, calculer récursivement $S_1 = \sum_{i=0}^{\frac{n}{2}-1} \frac{c_i}{X-a_i} = \frac{P_1}{Q_1}$ et $S_2 = \sum_{i=\frac{n}{2}}^{n-1} \frac{c_i}{X-a_i} = \frac{P_2}{Q_2}$
- 3) Calculer $S = S_1 + S_2 = \frac{P_1Q_2+P_2Q_1}{Q_1Q_2}$.
- 4) Renvoyer S.

La complexité $T(n)$ vérifie : $T(n) \leq 2T(\frac{n}{2}) + O(M(n))$, et le théorème du maître nous permet de trouver que $T(n) = O(M(n)\log(n))$.

Remarque. — La multiplication d'entiers est un problème très similaire à la multiplication de polynômes à coefficients entiers, la décomposition d'entiers en base a étant très similaire à l'écriture d'un polynôme. Des méthodes similaires s'appliquent pour la multiplication d'entiers, bien qu'il faille tenir compte des retenues.

Application. Algorithme de Karatsuba —
On peut raffiner l'algorithme naïf de multiplication en remarquant que pour $P = p_0 + p - 1X$ et $Q = q_0 + q_1X$, le produit $R = PQ = r_0 + r_1X + r_2X^2$ peut-être obtenu en seulement 3 multiplications avec : $r_0 = p_0q_0$, $r_2 = p_1q_1$, et $r_0+r_1+r_2 = P(1)Q(1) = (p_0+p_1)(q_0+q_1) \Rightarrow r_1 = (p_0+p_1)(q_0+q_1)-r_0-r_2$.
On a quelques additions supplémentaires, mais la perte d'une multiplication implique le gain d'un exposant dans la complexité de l'algorithme de Karatsuba par rapport à l'algorithme naïf.

- Entrée : P,Q polynômes de degré au plus $n-1$. (avec n une puissance de 2)
  On pose $k = \lceil \frac{n}{2} \rceil$.
- Sortie : R=PQ
- 1) Si n=1, renvoyer PQ
- 2) Décomposer $P = P_0 + X^kP_1$, $Q = Q_0 + X^kQ_1$.
- 3) Calculer $A_1 = P_0Q_0$ et $A_2 = P_1Q_1$ récursivement.
- 4) Calculer $A_3 = P_0 + P_1$ et $A_4 = Q_0 + Q_1$
- 5) Calculer $A_5 = A_3A_4$ récursivement
- 6) Calculer $A_6 = A_5 - A_1$, et $A_7 = A_6 - A_2$.
- 7) Renvoyer $A_1 + A_7X^k + A_2X^{2k}$

Le nombre d'opérations $K(n)$ de cet algorithme vérifie : $K(n) \leq 3K(\frac{n}{2}) + 4n$, donc le théorème du maître nous dit que $K(n) = O(n^{\log_2(3)}) = O(n^{1,59})$

4.3 La Transformée de Fourier Rapide ou FFT

L'idée de cet algorithme est basée sur le fait que l'évaluation en n+1 points $(a_i)_i$ est un isomorphisme linéaire de $K_n[X]$ vers K^{n+1}, et que pour P,Q de degrés $\leq \frac{n}{2}$, on peut déterminer PQ en calculant les $P(a_i)$, $Q(a_i)$, puis les $P(a_i)Q(a_i)$, et en interpolant ces valeurs en les a_i.
L'opérateur d'évaluation multi-points, dont la réciproque est l'interpolation, permet de linéariser la multiplication de polynômes.

Ici, n sera une puissance de 2.
Les a_i ne sont pas choisis quelconques. Pour les déterminer, on utilise pour cela une racine primitive n-ième de l'unité : un $w \in A$ tel que $w^n = 1$ et tel que $w^t - 1$ n'est pas un diviseur de 0 $\forall 1 \leq t < n$.
Avec un tel w, on pose alors $a_i = w^i$.

Définition. —
L'opération $P \in A[X] \mapsto (P(1), P(w), \ldots, P(w^{n-1})$, pour w racine primitive n-ième de l'unité, s'appelle la **transformée de Fourier discrète**.

Application. Algorithme de calcul de la transformée de Fourier discrète—
- — Entrée : $P(X) = p_0 + \cdots + p_{n-1}X^{n-1}$ et $1, w, w^2, \ldots, w^{n-1}$ les puissances d'une racine n-ième primitive de l'unité. (avec n une puissance de 2)
- — Sortie : $(P(1), P(w), \ldots, P(w^{n-1})$
- — 1) Si n=1, renvoyer p_0.
- — 2) Sinon, soit $k = \frac{n}{2}$. Calculer :
  $R_0(X) = \sum_{j=0}^{k-1}(p_j + p_{j+k})X^j$ et $\overline{R}_1(X) = R_1(wX) = \sum_{j=0}^{k-1}(p_j - p_{j+k})w^j X^j$
- — 3) Calculer récursivement $R_0(1), R_0(w^2), \ldots, R_0((w^2)^{k-1})$ et $\overline{R}_1(1), \overline{R}_1(w^2), \overline{R}_1((w^2)^{k-1})$
- — 4) Renvoyer $R_0(1), \overline{R}_1(1), R_0(w^2), \overline{R}_1(w^2) \ldots, R_0((w^2)^{k-1}), \overline{R}_1((w^2)^{k-1})$.

Remarque. — Cet algorithme récursif est basé sur le fait que $X^n - 1 = (X^{\frac{n}{2}} - 1)(X^{\frac{n}{2}} + 1)$.
En écrivant alors $P(X) = Q_0(X)(X^k-1)+R_0(X) = Q_1(X)(X^k+1)+R_1(X)$, on a alors $P(w^{2r}) = R_0(w^{2r})$ et $P(w^{2r+1}) = R_1(w.w^{2r})$ car $(w^{2r})^k = 1$ et $(w^{2r+1})^k = -1$.

D'une part, les calculs n'impliquent ici que des multiplications avec les w^i que l'on a pré-calculés, et d'autre part, les polynômes R_0,R_1 s'expriment très facilement à partir des coefficients de P.
Au plus $\frac{3n}{2}\log(n)$ multiplications apparaissent dans la Transformée de Fourier discrète.

Proposition. — La Transformée de Fourier discrète en w est un morphisme de A-modules $A[X]/(X^n - 1) \to A^n$ avec pour multiplication dans A^n la multiplication coordonnée par coordonnée.
Sa matrice dans les bases canoniques des espaces est la matrice de Vandermonde de $\{1, w, \ldots, w^{n-1}\}$, V_w.
Un calcul montre alors que $V_{w^{-1}}.V_w = n.I_n$.
Ainsi, si 2 est inversible dans A, n est inversible dans a comme puissance de 2, et la Transformée de Fourier discrète en w est inversible, d'inverse la Transformée de Fourier discrète en w^{-1} multipliée par n^{-1}.

L'interpolation des $P(1), P(w), \ldots, P(w^{n-1})$ pour retrouver P se ramène donc à une Transformée de Fourier discrète en les w^{-i}, w^{-1} étant bien elle aussi une racine primitive n-ième de l'unité.

Proposition. Algorithme de FFT—
- — Entrée : P,Q dans $A[X]$ tels que $\deg(P) + \deg(Q) < n$, w une racine primitive n-ième de l'unité. (n une puissance de 2, 2 inversible dans A)
- — Sortie : PQ
- — 1) Pré-calculer les w^i.
- — 2) Calculer les $P(1), P(w), \ldots, P(w^{n-1})$ et les $Q(1), Q(w), \ldots, Q(w^{n-1})$ par Transformée de Fourier discrète en w.
- — 3) Calculer les $r_i = P(w^i).Q(w^i)$.
- — 4) Pré-calculer les w^{-i}
- — 5) Pour $R(X) = r_0 + r_1X + \cdots + r_nX^n$, calculer les $R(1), R(w^{-1}), \ldots, R(w^{-(n-1)})$ par Transformée de Fourier discrète en w^{-1}.
- — 6) Poser $S(X) = s_0 + \cdots + s_nX^n$ avec $s_i = R(w^{-i})$.
- — 7) Calculer $n^{-1}.S(X)$.
- — 8) Renvoyer $n^{-1}.S(X)$.

Théorème. — Si 2 est inversible dans l'anneau a et n une puissance de 2, étant donnée w une racine primitive n-ième de l'unité dans A, le produit de deux polynômes P et Q tels que $\deg(P) + \deg(Q) < n$ peut être calculé en $\frac{9}{2}n\log(n) + O(n)$ opérations dans A.
Au plus n produits seulement sont entre deux éléments de A qui ne sont pas des puissances de w.

Remarque. — Pour pouvoir appliquer la Transformée de Fourier rapide avec n assez grand dans un $\mathbb{Z}/p\mathbb{Z}$, on va chercher p premier de la forme $p = l2^e + 1$ (**nombre premier de Fourier** d'exposant e) avec e "suffisamment grand".
Par exemple, $29.2^{57} + 1$ est un tel nombre premier. Donc, dans $\mathbb{Z}/4179340454199802089\mathbb{Z}$, on dispose de racines primitives 2^{57}-ièmes de l'unité (21 en est une), ce qui permet de multiplier des polynômes de degrés colossaux par un algorithme en $O(n\log(n))$ opérations.
De plus, on peut trouver une racine primitive de l'unité grâce à un élément primitif de $(\mathbb{Z}/p\mathbb{Z})^*$ que l'on peut déterminer avec un algorithme probabiliste. (la probabilité qu'un élément de $\mathbb{F}_p$ tiré au hasard soit primitif tend vers $\frac{6}{\pi^2}$)

Remarque. Algorithme de Schönhage-Strassen —
Quand les racines primitives de l'unité font défaut, on peut quand même faire fonctionner les idées à base de transformée de Fourier du moment que 2 est inversible en étendant l'anneau A de manière judicieuse. Grâce aux racines virtuelles n-ièmes de l'unité, du moment que 2 est inversible dans A, l'algorithme de Schönhage-Strassen permet de multiplier des polynômes de $A[X]$ de degré au plus n en $O(n\log(n)\log(\log(n)))$ opérations dans A.

Une version de cet algorithme dans des anneau de type $\mathbb{Z}/(2^{2^k}+1)\mathbb{Z}$ permet de multiplier des entiers avec la même complexité. (La gestion des retenues peut être programmée sans augmenter la complexité globale)

Remarque. — On peut aussi étendre l'idée de la FFT au cas où 3 est inversible (FFT triadique), et plus généralement à un anneau quelconque. On obtient alors un algorithme de complexité $O(n\log(n)\log(\log(n)))$.

4.4 Division euclidienne de polynômes

Pour effectuer des divisions euclidiennes rapides, on va s'aider de $K[[X]]$, l'anneau des séries formelles.
Pour F une série formelle, on dit que $F = O(X^n)$ si $F(X) = X^n.H(X)$ avec $H \in K[[X]]$.
On a le lemme suivant :

Lemme. — Soit F une série formelle de terme constant inversible et G une série formelle telle que $G - \frac{1}{F} = O(X^n)$, avec $n > 0$.
Alors $N(G) := G + G(1 - GF)$ vérifie $N(G) - \frac{1}{F} = O(X^{2n})$.

L'itération provient de la méthode de Newton appliquée à $f(u) = \frac{1}{u} - F$, et la démonstration se fait en considérant H telle que $GF = 1 - HX^n$.
Il découle de ce lemme un algorithme récursif de calcul du développement en série formelle de $\frac{1}{F}$ jusqu'à l'ordre n, dont le nombre de calculs $C(N)$ vérifie : $C(N) \leq C(\frac{N}{2}) + 2.M(N) + aN$ avec a une constante.
La démonstration de la prop 1 nous permet de trouver que $C(N) = O(M(N))$.

- Pour F et G polynômes unitaires de $K[X]$ de degrés m et n, la méthode de div.eucl. de F par G naïve demande $O(n(m-n))$ opérations.
- Le Lemme apporte une méthode plus rapide : Calculer $X^m.F(\frac{1}{X})$ et $X^nG(\frac{1}{X})$ (inverser l'ordre des coefficients)
  Calculer $\frac{X^m.F(\frac{1}{X})}{X^nG(\frac{1}{X})}\ mod(X^{m-n+1})$ (une inversion de série formelle et un produit)
  On en déduit le quotient Q car $deg(Q) \leq m-n$, et on en déduit R en calculant $F - QG\ mod(X^n)$.
- Cette méthode nécessite $O(M(n))$ opérations.
- Remarque : Pour $A, B \in K[X]$ de degré au plus d, avec $B(0)$ inversible, on peut calculer $\frac{A}{B}$ $mod(X^N)$ pour $N \geq d$ en $O(\frac{N}{d}M(d))$ opérations.

4.5 Interpolation et évaluation multi-points

On cherche à construire $P \in K[X]$ tel que $P(x_i) = y_i \ \forall 1 \geq i \geq n$.

Remarque. — L'évaluation multi-points d'un polynôme P de degré n en n points se fait naïvement en n évaluations coûtant $O(n)$ opérations dans K, soit $O(n^2)$ opérations dans K (coût quadratique).
L'évaluation en n points sur $\mathbb{R}_n[X]$ et l'interpolation en n points sont des applications linéaires qui sont inverses l'une de l'autre via le théorème chinois.

De plus, $P \bmod (X - a_i)$ étant une constante, multiplier deux polynômes P et Q dans une base d'interpolation revient à faire n multiplications de nombres. On cherche donc des méthodes rapides d'interpolation et d'évaluation multi-points afin d'avoir des méthodes plus rapides pour multiplier deux polynômes entre eux.

- L'interpolation la plus basique se fait selon la base canonique X^k, où l'on cherche à inverser l'isomorphisme chinois qui à $P \in \mathbb{R}_n[X]$ envoie $(P \bmod (X - a_i))_i$.
  La matrice de l'isomorphisme chinois est une matrice de Vandermonde en les a_i, et inverser cet isomorphisme revient à inverser une matrice de Vandermonde, ce qui demande $O(n^3)$ opérations.
- **L'interpolation de Lagrange** : se fait en considérant $Q(X) = \Pi_j(X - x_j)$ puis $b_i(X) = \frac{Q(X)}{X - x_i}$. On a alors $P(X) = \sum_i \frac{y_i}{b_i(x_i)}.b_i(X)$.
  On peut facilement faire varier les y_i mais une variation d'un x_i ou l'ajout d'un nouveau point demande à recalculer tous les polynômes.
  L'algorithme basique basé sur l'interpolation de Lagrange nécessite $O(n^3)$ opérations dans K pour calculer P.
- **Interpolation de Newton** se fait en écrivant $P(X) = \lambda_0 + \lambda_1.(X - x_1) + \cdots + \lambda_{n-1}(X - x_1)\dots(X - x_{n-1})$.
  On a alors $\lambda_0 = P(x_1) = y_1$, et pour $P_1(X) = \frac{P(X) - \lambda_0}{X - x_0}$ on a $P_1(x_i) = \frac{y_1 - y_i}{x_1 - x_i}$ et donc P_1 est le poly interpolateur des $P_1(x_2), \dots, P_1(x_n)$ en les $x_2, \dots, x_n$.
  Cela fournit alors un algorithme récursif dont le coût total est de $O(n^2)$ opérations dans K.
- Dans le cas de polynômes à coefficients entiers, le théorème des restes chinois donne d'autres méthodes d'interpolation.

Théorème. — L'évaluation et l'interpolation peuvent se faire en $O(M(n)\log(n))$ opérations dans K.

Exemple. — Pour A une matrice de taille $n \times n$ dont les coefficients sont des polynômes à une indéterminée de degré au plus d, det(A) est un polynôme de degré $\leq nd$.
Calculer det(A) directement par un Pivot de Gauss est trop lourd en calculs (les fractions à stocker deviennent rapidement importantes car vivant dans $K[X]$ et non seulement dans K).
A la place, on va calculer det(A) par interpolation en se donnant $nd + 1$ points $(x_i)_i$ de K et en évaluant les coefficients de A afin de calculer le déterminant de l'évaluée en chaque x_i.

Application. — Algorithme d'évaluation rapide :

- Entrée : $(a_i)_{1 \leq i \leq n}$, et P. (Avec n une puissance de 2)
- Sortie : $(P(a_0), \dots, P(a_{n-1})$.
- 1) Si n=1, renvoyer P. (P sera de degré 0 à ce moment-là)
- 2) Sinon, calculer $P_0 = P \ mod((X - a_0)\dots(X - a_{\frac{n}{2}-1}))$ et $P_1 = P \ mod((X - a_{\frac{n}{2}})\dots(X - a_{n-1}))$, puis faire un calcul récursif de $L_0 = (P_0(a_0), \dots, P_0(a_{\frac{n}{2}-1}))$ et $L_1 = (P_1(a_{\frac{n}{2}}), \dots, P_1(a_{n-1}))$.
- 3) Renvoyer L_0, L_1.

La complexité $T(n)$ vérifie : $T(n) \leq 2T(\frac{n}{2}) + O(M(n))$, le théorème du maître nous permet de trouver que $T(n) = O(M(n)\log(n))$.
On peut alors se servir d'un arbre, de l'évaluation rapide, et de la somme de fractions rationnelles de degré 1 afin d'obtenir un algorithme d'interpolation rapide d'une complexité en $O(M(n)\log(n))$.

5 Approximation de racines de polynômes avec la méthode de Newton

Théorème. Méthode de Newton polynômiale— Soit $P \in \mathbb{R}[X]$ à racines simples $\lambda_1, \ldots, \lambda_n \in \mathbb{C}$.
Si l'une des racines λ_i de P de module maximal est réelle positive, alors il existe $r > 0$ tel que $\forall x_0 \in]\lambda_i - r, +\infty[$, la suite récurrente définie par $x_{n+1} = x_n - \frac{P(x_n)}{P'(x_n)}$ converge vers λ_i.
Cette convergence est linéaire, puis quadratique dès que $x_n \in]\lambda_i - r, \lambda_i + r[$.

Si l'on se donne un polynôme P, alors $\frac{P}{pgcd(P,P')}$ a les mêmes racines que P et est à racines simples.
Si P est de plus à racines réelles, on peut soit intervertir l'ordre de ses racines avec $x \mapsto x + a$ et $x \mapsto \frac{1}{x}$ afin d'avoir un polynôme Q dont la plus grande racine est l'image de la r-ième racine de P, ou bien évaluer P en certains points afin de localiser grossièrement λ_r grâce aux changements de signe de la courbe de P, puis débuter une méthode de Newton une fois suffisamment proche de λ_r.
On peut donc bien utiliser cette méthode pour approximer à vitesse quadratique n'importe quelle racine d'un polynôme P.

Remarque. — Si P est dans $\mathbb{Q}[X]$ et si $x_0 \in \mathbb{Q}$, alors tous les x_n sont dans $\mathbb{Q}$, ce qui permet d'approcher de façon rationnelle des nombres algébriques.
On peut ainsi savoir si un $a \in \mathbb{N}$ est la puissance n-ième d'un entier en appliquant la méthode de Newton à $P(X) = X^n - a$ afin d'approcher sa racine réelle positive à l'entier le plus proche, et en testant si cet entier élevé à la puissance n est égal à a.

Toutefois, dans le cas de la recherche d'une racine entière d'un polynôme $P \in Z[X]$, on peut chercher une racine modulo p de P, a_1, puis une racine modulo p^2, a_2, telle que $a_2 \equiv a_1 mod(p)$, et ainsi de suite, jusqu'à ce que l'on atteigne un l tel que P ne s'annule pas sur $[p^l, +\infty[$.
On utilise alors la **Méthode de Newton algébrique**.

5.1 Calcul d'inverse avec la méthode de Newton algébrique

Proposition. — Soit A un anneau commutatif unitaire. Soient $a, p, x \in A$ tels que $ax \equiv 1\ mod(p)$.
Alors $a(2x - x^2a) \equiv 1\ mod(p)$.

Proposition. — Soit A un anneau commutatif unitaire. Soient $a, p, x_0 \in A$ tels que $ax_0 \equiv 1\ mod(p)$, $l \in \mathbb{N}^*$, et $r := \lceil \log_2(l) \rceil$.
On définit $\{x_i\}_{1 \leq i \leq r}$ par : $x_{i+1} = 2x_i - ax_i^2\ mod(p^{2^i})$.
Alors $a.x_r \equiv 1\ mod(p^l)$.

On peut ainsi calculer l'inverse de a modulo p^l en $O(M(l))$ opérations, avec $M(l)$ le coût de la multiplication d'entiers de taille $\leq 2^l$.

5.2 Algorithme de Newton-Hensel et calcul de racines n-ièmes

Lemme. — Soit A un anneau commutatif unitaire. Soient $f \in A[X]$, $a, p \in A$ tels que $f(a) \equiv 0 mod(p)$ et $f'(a)$ inversible modulo p d'inverse s.
Alors : $f(a - f(a)s) \equiv 0\ mod(p^2)$.

Théorème. — Soit A un anneau commutatif unitaire. Soient $f \in A[X]$, $x_0, p \in A$ tels que $f(x_0) \equiv 0 mod(p)$ et $f'(x_0)$ inversible modulo p d'inverse s.
Soit $l \in \mathbb{N}^*$.
Si on a : $\begin{cases} f(x) \equiv 0 \bmod(p^l) \\ f(y) \equiv 0 \bmod(p^l) \\ x \equiv y \equiv x_0 \bmod(p) \end{cases}$ alors $x \equiv y \bmod(p^l)$.

La **Méthode de Newton p-adique** donne ainsi une seconde façon de savoir si un entier a est la puissance n-ième d'un entier.
Pour se faire, on se ramène à n impair et on extrait de a la plus grande puissance de 2 qui la divise afin d'avoir a impair, afin que $f(x) = x^n - a$ vérifie : $f(1) \equiv 0\ mod(2)$ et $f'(1) = n \neq 0\ mod(2)$.
Soit l le plus petit entier non-nul tel que $a < 2^{nl}$.
Si l'algorithme de Newton-Hensel peut être itéré jusqu'à 2^l pour avoir un entier x^* tel que $(x^*)^n - a \equiv 0$ mod(2^l), alors x^* est une racine n-ième de a dans $\mathbb{Z}$.

6 Localisation de racines

Le calcul exact des racines d'un polynôme P est toujours possible si $deg(P) \leq 4$.
On a des formules algébriques pour exprimer ces racines si $deg(P) = 2$ ou 3, mais qui s'en fout ?
La théorie de Galois nous permet de montrer qu'il existe des polynômes de tout degré ≥ 5 dont les racines ne peuvent pas s'exprimer à l'aide de sommes,produits, et extractions de racines p-ièmes appliquées aux coefficients de P.

6.1 Cas des polynômes à coefficients complexes

Soit $P(X) = X^n + a_{n-1}X^{n-1} + \cdots + a_0 \in \mathbb{C}[X]$ de racines $\lambda_1, \ldots, \lambda_n$.

Proposition. — En notant $r_0 := \max_{1\leq k\leq n}(|\lambda_k|)$ on a que $r_0 \leq 1 + sup_k(|a_k|)$.
En considérant $X^n P(\frac{1}{X})$ on obtient de même : $\inf_k(|\lambda_k|) \geq \frac{|a_0|}{1+\max_k(|a_k|)}$.

On introduit par la suite la mesure M(P) d'un polynôme P afin d'étudier la distance entre ses racines.
La mesure d'un polynôme à coefficients entiers permet aussi d'obtenir des majorations sur la taille de ses facteurs irréductibles.
Et il existe des méthodes pour approximer M(P).

Définition. — Pour $Q(X) = a_n X^n + a_{n-1}X^{n-1} + \cdots + a_0 \in \mathbb{C}[X]$ de racines $\lambda_1, \ldots, \lambda_n$,on définit :
$\|Q\|_2 := \sqrt{\sum_k |a_k|^2}$, $\|Q\|_1 := \sum_k |a_k|$, $\|Q\|_\infty := \max(|a_0|, \ldots, |a_n|)$, $M(Q) := |a_n|\Pi_k(\max(1, |\lambda_k|)$.

Lemme. — On a : $\|Q\|_\infty \leq \|Q\|_2 \leq \|Q\|_1 \leq (n+1)\|Q\|_\infty$
et $2^{-n}\|Q\|_1 \leq M(Q) \leq \|Q\|_2$ (inégalité de Landau)

Comme M est multiplicative, on trouve alors que $\|P\|_\infty\|Q\|_\infty \leq 2^{n+m}.\sqrt{n+m+1}\|PQ\|_\infty$
On peut approcher $M(Q)$ par $\|Q_m\|_2^{2^{-m}}$ où $Q_m(X) := a_n^{2^m}\Pi_k(X - \lambda_k^{2^m})$.

Proposition. Séparation des racines—
Pour P à racines simples, on pose $\delta := \min_{i\neq j}|\lambda_i - \lambda_j|$.
Pour $disc(P) := a_n^{2n-2}\Pi_{i<j}(\lambda_i - \lambda_j)^2 = Res_X(P, P')$ le discriminant de P, on a :
$\delta \geq \sqrt{\frac{disc(P)}{n^{n+2}}}.\|P\|_2^{1-n}$.

Dans le cas où P est à coefficients entiers et à racines simples, $disc(P) \geq 1$, ce qui permet de simplifier l'expression précédente.
Ces résultats ne permettent pas de localiser les racines. Il faut un analogue des suites de Sturm pour cela. (Méthode de l'indice de Cauchy)

Remarque. — Il existe des polynômes irréductibles de $Z[X]$ dont les racines sont arbitrairement proches.
Pour $a \geq 10$, on pose $P(X) = X^n - 2(aX - 1)^2$. Le critère d'Eisenstein appliqué à P pour $P = 2$ nous dir que P est irréductible.
Or, pour $h := a^{-\frac{n+2}{2}}$ on remarque que $P(\frac{1}{a} - h) < 0$, $P(\frac{1}{a}) > 0$, et $P(\frac{1}{a} + h) < 0$, donc P a deux racines distinctes dans $]\frac{1}{a} - h, \frac{1}{a} + h[$.

6.2 Cas des polynômes à coefficients réels

Pour les polynômes réels, on dispose d'autres méthodes pour localiser les racines.

Proposition. Règle de Newton—
Soit $P \in \mathbb{R}[X]$ de degré n, et soit $L \in \mathbb{R}$ tel que $P^{(i)}(L) \geq 0$, $\forall 0 \leq i \leq n$.
Alors toute racine réelle de P est majorée par L.

Proposition. Règle de Lagrange-MacLaurin—
Soit $P(X) = X^n + a_{n-1}X^{n-1} + \cdots + a_0 \in \mathbb{R}[X]$, tel que pour un $m \leq n-1$, on ait $a_{n-1}, \ldots, a_{n-1-(m-1)}$ positifs.
Alors pour $A := \max(-a_{n-1-m}, \ldots, -a_0)$, toute racine réelle de P est strictement majorée par $1 + A^{\frac{1}{m}}$.

Exemple. — Pour $P(X) = X^6 - 12X^4 - 2X^3 + 37X^2 - 10X - 10$, alors la règle de Newton donne L=$\sqrt{8}$ et la règle de Lagrange-Maclaurin donne L=$1 + \sqrt{12}$.

Théorème. Suites de Sturm—
Soient $a < b$ et $P_0, \ldots, P_s$ des polynômes réels tels que :
- — $P_0(a)P_0(b) \neq 0$
- — P_s ne s'annule pas sur $[a, b]$
- — Pour $0 < j < s$ et $c \in [a, b]$ tels que $P_j(c) = 0$, alors $P_{j-1}(c)P_{j+1}(c) < 0$.
- — Pour $c \in]a, b[$ tel que $P_0(c) = 0$, alors $P_0(x)P_1(x)$ est du signe de $x - c$ au voisinage de c.

Pour $c \in [a, b]$ on note $V(c)$ le nombre de changement de signes (sans compter les 0) dans la suite $P_0(c), \ldots, P_s(c)$.
Alors le nombre de racines distinctes de P_0 sur $[a, b]$ est $V(a) - V(b)$.

Ainsi, pour un polynôme P à racines simples (quitte à considérer $\frac{P}{pgcd(P,P')}$, on peut construire une suite de Sturm pour P en posant $P_0 = P$, $P_1 = P'$, et $P_{i+1} \equiv -P_{i-1} \bmod(P_i)$.
Cet algorithme est similaire à l'algorithme d'Euclide pour P et P' et s'arrête donc, son dernier résultat non-nul étant $P_s = \pm pgcd(P, P') = \pm 1$, et cette suite est bien une suite de Sturm pour P. Et si l'on prend un intervalle contenant toutes ses racines réelles (via une majoration de $P(X)$ et $P(-X)$), alors on peut localiser toutes les racines de P par dichotomie en évaluant la suite de Sturm obtenue précédemment en des points de $[a, b]$.
On ne doit calculer qu'une seule fois une suite de Sturm pour P, et les évaluations sont plutôt rapides car on ne regarde que le signe des $P_k(c)$ pour un c donné afin de compter le nombre de changements de signes.

6.3 Cas des polynômes à coefficients entiers

Le lemme de Gauss nous permet de savoir que l'étude de l'irréductibilité d'un polynôme à coeffs entiers sur $\mathbb{Z}$ ou sur $\mathbb{Q}$ est identique si les coefficients de ce polynôme sont premiers entre eux dans leur ensemble.

Corollaire. — Soient P,Q $\in \mathbb{Z}[X]$ de degrés m,n tels que Q divise P. Alors $\|Q\|_\infty \leq 2^n \|P\|_2$.

Théorème. — Soit $F(X) = P(X)^n + pQ(X) \in \mathbb{Z}[X]$ avec p premier, $n \geq 1$, P irréductible sur $\mathbb{F}_p$,
et P ne divisant pas Q sur $\mathbb{F}_p$.
Alors F est irréductible sur $\mathbb{Z}$.

Corollaire. Critère d'Eisenstein—
S'il existe un p premier qui divise les a_i et tel que p^2 ne divise pas a_0, alors P est irréductible sur $\mathbb{Z}$.

Remarque : Si $P \in \mathbb{Z}[X]$ est de coeff dominant non-divisible par un p premier et est irréductible sur $\mathbb{F}_p$, alors il est irréductible sur $\mathbb{Z}$.

7 Factorisation de polynômes sur des corps finis

On ne connaît pas d'algorithme déterministe de factorisation en temps polynômial sur les $\mathbb{F}_q$.
Cependant, on a des algorithmes probabilistes de factorisation en temps polynômial.

Théorème. — Soit $P \in \mathbb{F}_q[X]$ de degré d.
P est irréductible ssi : $\begin{cases} X^{q^d} \equiv X \bmod(P) \\ \forall p \text{ premier tq } p|d,\ pgcd(X^{q^{d/p}} - X, P) = 1 \end{cases}$

La factorisation se déroule en 3 étapes :

— On se ramène à un polynôme sans facteurs carrés.
Considérer $pgcd(P, P')$ n'est pas suffisant dans le cas où $P = Q_1(X)^p.Q_2(X)^{p^3}$. car $P' = 0$, et $P = Q^p$ avec à nouveau $Q' = 0$.

— On se ramène à des polynômes dont les facteurs irréductibles ont le même degré.
Comme $X^{q^r} - X = \Pi_{\deg(Q)|r, Qirred} Q(X)$, on peut calculer les $\Psi_r(X) = \Pi_{\deg(Q)=r, Qirred} Q(X)$ et calculer les $pgcd(P, \Psi_r)$ pour tout $r \leq d$. (En calculant les pgcd et en divisant P par ceux-ci, on va souvent s'arrêter pour un r bien plus petit.)
Il y a une probabilité $> \frac{1}{2}$ que la factorisation soit finie à cette étape.

— On finit ensuite de factoriser chaque polynôme ayant des facteurs irréductibles de même degré, puis on retrouve la puissance de chaque facteur irréductible dans le polynôme initial.
Deux algorithmes fréquemment utilisés sont Berlekamp et Cantor-Zassenhaus.

7.1 Algorithme de Berlekamp

Soit $Q \in F_q[X]$ sans facteurs carrés : $Q = \Pi_{i=1}^r Q_i$.
L'application Ψ_Q : $\begin{array}{ccc} \mathbb{F}_q[X]/(Q) & \to & \mathbb{F}_q[X]/(Q) \\ P & \mapsto & P^q - P \end{array}$ est $\mathbb{F}_q$-linéaire, et l'étude de son noyau va permettre de factoriser Q.

Proposition. — Un polynôme P de degré $\leq \deg(Q) - 1$ est dans le noyau de Ψ_Q ssi P mod$(Q_i) \in \mathbb{F}_q$.
(P est une constante modulo chaque Q_i)

Pour $G_i := \frac{Q}{Q_i}.Q_i'.(\frac{1}{Q'}) \in \mathbb{F}_q[X]/(Q)$, on remarque alors que $Q_j|G_i$, $\forall j \neq i$ et que $Q_i|G_i - 1$.
Par linéarité, et par la proposition les G_i forment donc une base du noyau de Ψ_Q.
On remarque alors que $dim(Ker(\Psi_Q)) = r$.

Proposition. — Si $P \in Ker(\Psi_Q)$ est non-constant, alors les constantes P mod(Q_i) ne sont pas toutes identiques.
On a ainsi un $\lambda \in \mathbb{F}_q$ tel que $pgcd(Q, P - \lambda)$ est un diviseur strict de Q.

Application. Scindage de Q :

— Entrée : $Q \in \mathbb{F}_q[X]$ de degré n, sans facteurs carrés.
— Sortie : Un facteur non-trivial de Q si Q n'est pas irréductible.
— Calculer la matrice de l'application Ψ_Q dans la base canonique $1, X, \ldots, X^{n-1}$.
— Si le rang de Ψ_Q est n-1, alors retourner le fait que Q est irréductible.
— Choisir un élément P non-constant dans le noyau de Ψ_Q.
On trouve une base de Ker(f) par le pivot de Gauss.
— Calculer les $pgcd(Q, P - a)$ pour tout $a \in \mathbb{F}_q$ jusqu'à trouver un diviseur non-trivial de Q. La complexité est linéaire en q à cause de cette étape.

Cet algorithme calcule un facteur non-trivial de Q en $O(n^w + qM(n)\log(n))$ opérations.
(le $w > 0$ vient de la partie d'algèbre linéaire de l'algorithme, suivant la méthode pour trouver un élément du noyau convenable)
Comme la complexité est linéaire en q, l'algorithme devient impraticable dès que q atteint quelques miliers.
On ne connaît pas d'algorithme de factorisation déterministe de complexité polynômiale en n et log(q).

7.2 Algorithme de Cantor-Zassenhaus

Application. Factorisation de Q :

— Entrée : $Q \in \mathbb{F}_q[X]$ de degré n sans facteurs carrés et dont tous les facteurs irréductibles ont même degré $\frac{n}{r}$. (On connaît le degré des facteurs irréductibles de par la 2e étape de factorisation)
— Sortie : la factorisation en irréductibles $Q = \Pi_{i=1}^{r} Q_i$
— 1) Si r=1, retourner Q.
— 2) Répéter : a) Choisir P de degré $d \leq n-1$ d'une façon aléatoire.
b) Calculer $H_1 = pgcd(Q, P)$. Si H_1 non-constant, alors aller à l'étape 3.
c) Calculer $H = P^{\frac{q^d-1}{2}}$ mod(Q) si q est impair, et $H = Tr_{\frac{n}{r}}(P)$ mod(Q) si q est une puissance de 2. d) Calculer $H_1 = pgcd(Q, H-1)$. Si H_1 non-constant, alors aller à l'étape 3.
— 3) Calculer $H_2 = Q/H_1$ et appeler récursivement l'algorithme pour H_1 et H_2.
— Retourner la réunion H_1, H_2.

Grâce à l'étape probabiliste, cet algorithme factorise un polynôme de degré n en $O(n^2 + n\log(q))$ opérations si q est impair.

7.3 Factoriser dans $\mathbb{Z}[X]$

— Se ramener à un polynôme de contenu 1 en factorisant par le contenu.
— Se ramener à un polynôme sans facteurs carrés avec $pgcd(P, P')$.
— Choisir un petit p premier ne divisant pas $Disc(P)$. (afin que P mod(p) n'ait pas de facteurs carrés)
— Factoriser P mod(p).
— En déduire la factorisation de P mod(P^m) pour $m >> 0$ grâce à la méthode de Newton-Hensel. (méthode de Newton p-adique)
— Pour tous les regroupements possibles dans la factorisation mod(p^m), relever les coefficients dans $\mathbb{Z}$ et chercher si on a trouvé un diviseur de P.

8 Résultant

8.1 Relations coefficients-racines. Polynômes symétriques

Définition. — Un polynôme $P \in K[X_1, \ldots, X_n]$ est symétrique si $\forall \sigma \in \Sigma_n$, P est stable par σ :
$P(X_{\sigma(1)}, \ldots, X_{\sigma(n)}) = P(X_1, \ldots, X_n)$.
Pour $n \geq 1$ fixé et pour tout $1 \leq k \leq n$, on définit les polynômes symétriques élémentaires :

$$\Sigma_k(X_1, \ldots, X_n) := \sum_{1 \leq i_1 < \cdots < i_k \leq n} X_{i_1} \ldots X_{i_k} \tag{1}$$

Théorème. — Tout polynôme symétrique de $K[X_1, \ldots, X_n]$ s'écrit de façon unique comme un polynôme en les polynômes symétriques élémentaires Σ_k.

Remarque. — Pour $P(X) = X^n + a_{n-1}X^{n-1} + \cdots + a_0 = \Pi_{i=1}^{n}(X - \alpha_i)$, on a :
$a_j = (-1)^{n-j}\Sigma_{n-j}(\alpha_1, \ldots, \alpha_j)$, pour tout $0 \leq j \leq n-1$.

Théorème. Formules de Newton— Soit $P_k(X_1, \ldots, X_n) := X_1^k + \cdots + X_n^k$.
Alors on a : $\forall k \geq n : P_k - \Sigma_1.P_{k-1} + \Sigma_2.P_{k-2} + \cdots + (-1)^n\Sigma_n.P_{k-n} = 0$
$\forall 1 \leq k \leq n : P_k - \Sigma_1.P_{k-1} + \cdots + (-1)^{k-1}\Sigma_{k-1}.P_1 + (-1)^k.\Sigma_k.k = 0$

8.2 Le résultant de Sylvester

Définition. — Soient $f(X) = a_m X^m + \cdots + a_0$, $g(X) = b_n X^n + \cdots + b_0$ deux polynômes sur un corps K. Lorsque $f(X)$ et $g(X)$ possèdent un facteur commun non-trivial h : $f = f_1.h$ et $g = g_1.h$, alors l'équation $uf + vg = 0$, $\deg(u) < \deg(g)$ et $\deg(v) < \deg(f)$ possède comme solution le couple $(u, v) = (g_1, -f_1)$.
Réciproquement, puisque $g|uf$ et $\deg(u) < \deg(g)$, l'existence d'un couple solution non-nul implique l'existence d'un facteur commun non-trivial entre f et g.
En projetant cette équation dans la base canonique de $K_m[X]\times K_n[X]$, l'existence d'une solution non-nulle est équivalente au fait que l'application linéaire associé soit non-injective, ce qui est équivalent à la nullité du déterminant de la matrice de cette application linéaire dans la base canonique de $K_m[X] \times K_n[X]$, de dimension $m + n$.
Ce déterminant est appelé le résultant de f et g et est noté $Res_X(f, g)$. C'est un élement de K.

Le résultant de f et g se calcule naïvement en $O((n + m)^3)$ opérations par Pivot de Gauss.

Théorème. — Le résultant possède les propriétés suivantes :
- $Res_X(f, 0) = 0$
- $Res_X(f, g) = (-1)^{mn} Res_X(g, f)$
- $Res_X(f(X + a), g(X + a)) = Res_X(f, g)$, pour tout $a \in K$.
- Si $m \leq n$ et si h est le reste de la division de g par f, on a $Res_X(f, g) = a_m^{n-m} Res_X(f, h)$.
- $Res_X(f, g_1 g_2) = Res_X(f, g_1)Res_X(f, g_2)$.
- $Res_X(f, g) = 0$ ssi f et g ont un facteur commun non-trivial ou si f ou g est nul.
- Le résultant de f et g ne change pas si l'on passe sur L une extension de corps de K.
- Pour $\overline{K}$ une clôture algébrique de K, et $(\alpha_i)_i$,$(\beta_j)_j$ les racines de f et de g, on a :
  $Res_X(f, g) = a_m^n \Pi_{i=1}^m g(\alpha_i) = (-1)^{mn} b_n^m \Pi_{j=1}^n f(\beta_j) = a_m^n b_n^m \Pi_{i=1}^m \Pi_{j=1}^n (\alpha_i - \beta_j)$.
  $Res_X(f, g)$ est un polynôme homogène en les α_i, β_j de degré mn.

Remarque. — Le résultant est un polynôme entier en les coeffs de f et de g.
Cela permet donc de le calculer dans tout anneau commutatif unitaire.
Si l'on suppose A factoriel, alors en utilisant le théorème précédent pour $Frac(A)$, on garde la propriété que f et g ont un facteur commun ssi $Res_X(f, g) = 0$, ce qui est équivalent à avoir une racine commune dans une extension algébrique de $Frac(A)$.
Cette démarche permet de traiter le cas des polynômes en plusieurs variables, car si $f, g \in K[X_1, \ldots, X_n]$, l'anneau $A := K[X_1, \ldots, X_{n-1}]$ est factoriel et l'on peut voir f et g comme polynômes en X_n à coefficients dans A.
On peut ainsi définir le résultant en X_i de deux polynômes de $K[X_1, \ldots, X_n]$ sans problèmes.
Ce résultant est alors un polynôme de $K[X_1, \ldots, X_{i-1}, X_{i+1}, \ldots, X_n]$.

Proposition. Lien avec le PGCD — Soient $f, g \in K[X]$.
Alors il existe $A, B \in K[X]$ tel que $Af + Bg = Res_X(f, g)$.
De plus, les coefficients de A,B sont des polynômes entiers en les coefficients de f et g.
Ainsi, f et g sont premiers entre eux ssi $Res_X(f, g) \neq 0$.

Grâce à cette proposition, on peut démontrer que pour D l'ensemble des matrices diagonalisables de $M_n(\mathbb{C})$, $\overset{\circ}{D}$ est l'ensemble des matrices diagonalisables à valeurs propres distinctes.

Application. Discriminant — Pour $P(X) = a_m X^m + \cdots + a_0$ sur un anneau factoriel A, on définit le discriminant de P par : $Res_X(P, P') = (-1)^{\frac{m(m-1)}{2}} a_m.Disc(P)$. C'est un élément de A.
En considérant $(\alpha_i)_i$ les racines de P dans un corps de décomposition de P sur $Frac(A)$, on trouve alors : $Disc(P) = \Pi_{i<j}(\alpha_i - \alpha_j)^2$.
Ainsi, $Disc(P) = 0$ ssi P a une racine double.
On peut aussi définir $Disc(P)$ par sa deuxième expression et remarquer que $\Pi_{i<j}(X_i - X_j)^2$ est un polynôme symétrique afin de voir que $Disc(P)$ est un polynôme symétrique en les racines de P, donc un polynôme en les coefficients de P de par les relations coefficients-racines.
De plus, pour tout $P \in \mathbb{C}[X]$, $Disc(P)$ est invariant :
s$\forall M \in Gl_2(\mathbb{C})$, $M = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$, $(cX + d)^{\deg(P)}.P(\frac{aX+b}{cX+d}) = P(X)$.

8.3 Théorie de l'élimination

On se donne $f, g \in K[X_1, \ldots, X_r]$. On écrit alors :
$f(X) = \sum_{i=0}^{m} f_i X_r^i$ et $g(X) = \sum_{j=0}^{n} g_j X_r^j$ avec $f_m g_n \neq 0$.

Théorème. — On a l'alternative suivante :
- Si $Res_{X_r}(f, g) \equiv 0$, alors f et g ont un facteur commun.
- Sinon, l'équation $Res_{X_r}(f, g)(X_1, \ldots, X_{r-1}) = 0$ est une équation en $r - 1$ variables.

En pratique, on dispose souvent de r équations à r variables. En calculant des résultants successivement, on peut échelonner notre système d'équations en fonction du nombre d'inconnues, et on peut alors résoudre la dernière équation de façon approchée puisqu'elle ne comporte qu'une variable.
Cependant, contrairement à la résolution par pivotage des équations linéaires, ici, le premier cas du théorème nous dit qu'une solution $(x_0, \ldots, x_r)$ du système va donner naissance à un facteur non trivial, et que donc x_0 va être la racine de la dernière équation.
De nouvelles racines peuvent toutefois apparaître dans les résultants.

Proposition. — Si $(\alpha_1, \ldots, \alpha_r)$ est un zéro commun de f et de g, alors :
$Res_{X_r}(f, g)(\alpha_1, \ldots, \alpha_{r-1}) = 0$.

Théorème. — On suppose connu un $(\alpha_1, \ldots, \alpha_{r-1})$ tel que $Res_{X_r}(f, g)(\alpha_1, \ldots, \alpha_{r-1}) = 0$.
Si $f_m((\alpha_1, \ldots, \alpha_{r-1}))g_n((\alpha_1, \ldots, \alpha_{r-1})) \neq 0$, alors il existe α_r tel que $(\alpha_1, \ldots, \alpha_r)$ soit un zéro commun de f et de g.

8.4 Applications

Application. Intersection de courbes dans $\mathbb{R}^2$ —
Si l'on considère $f(X, Y) = X^4 + Y^4 - 1$ et $g(X, Y) = X^5Y^2 - 4X^3Y^3 + X^2Y^5 - 1$,
alors leurs lieux d'annulation définissent deux courbes de R^2.
Les points (x, y) dans l'intersection de ces deux courbes doivent vérifier $Res_X(f, g)(y) = 0$.
$Res_X(f, g)(Y)$ est un polynôme de degré 28 ici, et on peut compter et approcher ses solutions réelles avec la méthode de Newton polynômiale afin de trouver 4 solutions réelles.
En reprenant $f(X, Y)$ et en évaluant Y en chacune des 4 solutions, on trouve alors 16 candidats.
En évaluant $g(X, Y)$ en chacun de ces candidats, on peut alors déterminer lesquels sont des points d'intersection des deux courbes et ainsi trouver tous les points d'intersection de ces deux courbes.

Application. Intersection de courbes dans $\mathbb{R}^2$ —
On se donne C une courbe plane paramétrée de façon rationnelle : $C = \{(\frac{a(t)}{b(t)}, \frac{c(t)}{d(t)}) \in \mathbb{R}^2, t \in \mathbb{R}\}$.
Les points (x,y) sont sur la courbe C ssi $b(t)x - a(t) = d(t)y - c(t) = 0$, ce qui fournit une équation implicite de C : $f(x, y) = Res_t(b(t)x - a(t), d(t)y - c(t)) = 0$.

Application. Formule de Héron —
Pour ABC un triangle de côté abc, on cherche à exprimer $S := Aire(ABC)$ en fonction de a,b,c.
Quitte à composer par des rotations et translations, on suppose que $A = (0, 0)$,$B = (a, 0)$,$C = (x, y)$.
Ainsi, (x,y,S) est solution du système : $\begin{cases} P(X, Y) = (a - X)^2 + Y^2 - b^2 = 0 \\ Q(X, Y) = X^2 + Y^2 - c^2 = 0 \qquad R(Y, S) = aY - 2S = 0 \end{cases}$
On élimine alors les variables X,Y en calculant $R_1(Y) = Res_X(P, Q)$ et $R_2(S) = Res_Y(R, R_1)$.
On a $R_2(S) = -2a^4c^2 + a^6 2a^4b^2 + a^2b^4 - 2a^2b^2c^2 + a^2c^4 + 16S^2a^2$, ce qui donne la formule de Héron :
$S^2 = \frac{1}{16}(a + b - c)(a + b + c)(a - b + c)(a - b - c)$

Application. Problème de la cinématique inverse —
Dans le plan, on fixe les points $A = (-1, 0)$ et $B = (1, 0)$. On se donne alors trois longueurs a,b,c et on cherche les points C,D tels que BC=a,CD=b,DA=c. Les coordonnées des points C,D sont ainsi solutions de 3 équations en a,b,c, et l'utilisation du résultant nous permet d'obtenir des équations permettant d'approcher numériquement toutes les positions que peuvent prendre les points C et D.
Ce problème est par exemple important dans la conception d'essieux de trains.

Application. Nombres algébriques —
Soient $a, b \in \mathbb{C}$ algébriques sur $\mathbb{Q}$ de polynômes minimaux sur $\mathbb{Q}$ f et g.
Si on note p (resp q) le polynôme minimal de a+b (resp ab), alors on a :
$p(Y) = Res_X(f(X), g(Y-X))$ et $q(Y) = Res_X(f(X), g(Y/X)X^{\deg(g)})$.
Pour a et b les racines réelles de $f(X) = X^5 - 2$ et $g(X) = X^3 + X + 7$, on a par exemple :
$q(X) = X^{15} - 70X^{10} + 984X^5 + 134456$

9 Codes correcteurs d'erreurs

Les codes correcteurs ont été introduits pour corriger les erreurs de transmission ou de lecture de données numériques, ou les erreurs survenant au cours de leur inscription sur un support physique (bande, CD) ou encore lorsque les données subissent une altération sur le support de stockage.
Voici quelques domaines où ils sont appliqués : Transmissions spatiales, minitel, codes barres, disque compact et DVD, communications par internet,...

Les messages transmis sont supposés découpés en blocs (ou mots) de longueur n écrits avec l'alphabet $\{0, 1\}$.
Un code binaire C de longueur n est un sous-ensemble de $\{0, 1\}^n$ contenant tous les mots que l'on peut transmettre.
C est alors une partie de $\mathbb{F}_2^n$.

Définition. — On définit alors la **distance de Hamming** entre deux mots x et y de $\mathbb{F}_2^n$ par :
$d(x, y) := Card(\{i \text{ tq} x_i \neq y_i \})$.
C'est bien une distance sur $\mathbb{F}_2^n$.

La distance minimum du code C est alors $d_C := \min_{x \neq y \in C}(d(x, y))$.

Application. — Lorsqu'un mot $c \in C$ est émis, et que le mot $r \in \mathbb{F}_2^n$ est reçu après d'éventuelles erreurs, on décode le mot r selon le **principe du maximum de vraisemblance** : on décode r comme le mot de C à distance minimum de r.

Définition. — On dit que C est un **code k-correcteur** quand toute erreur portant sur au plus k bits est corrigée correctement.

Ainsi, C est k-correcteur ssi toutes les boules de centre $x \in C$ et de rayon k sont disjointes, ssi $d_C \geq 2k+1$.
Un code correcteur C de distance d_C va donc toujours corriger correctement toute erreur portant sur au plus $\lceil \frac{d_C - 1}{2} \rceil$ bits.

9.1 Codes linéaires

Comme le calcul de la distance minimale porte sur tous les $x \neq y$, et comme $d(x, y) = d(x - y, 0)$, on préfère se restreindre au cas où C est un sous-espace vectoriel de $\mathbb{F}_2^n$. Dans ce cas-ci, $d_C = \min x \in C, x \neq 0(d(x, 0))$.
Un tel code C est appelé un **code linéaire**
Pour $Card(C) = 2^k$, le rapport $\frac{k}{n}$ est appelé **taux de transmission** du code.

Définition. — On regroupe les trois paramètres n, k, d_C d'un code linéaire C en disant que C est de type (n, k, d_C).

Proposition. —
- — Borne de singleton : On a toujours $d_C + k \leq n + 1$(on ne peut pas avoir un code très fourni dans $\mathbb{F}_2^n$ qui corrige beaucoup d'erreurs)
- — Borne de Hamming : La boule $B(0, r)$ contient $R = \sum_{l=0}^{n} \binom{n}{k}$ mots.
  Ainsi, pour C code de $\mathbb{F}_2^n$ tel que $d_C = 2r + 1$, C possède au plus $\frac{2^n}{R}$ éléments.

Proposition. — Pour C un code linéaire de type (n, k, d), on définit le code étendu $\widetilde{C}$ comme le s-ev de $\mathbb{F}_2^{n+1}$ composé des $(x_1, \ldots, x_{n+1})$ vérifiant : $(x_1, \ldots, x_n) \in C$ et $\sum_{i=1}^{n+1} x_i = 0$.
$\widetilde{C}$ est alors un code linéaire de type $(n + 1, k, d_C + 1)$ si d_C est impair et $(n + 1, k, d_C)$ si d_C est pair.

9.2 Matrice génératrice d'un code linéaire

On peut se donner un sous-ev de $\mathbb{F}_2^n$ par une famille libre de taille k.
Une matrice génératrice G d'un code linéaire C est une matrice dont les lignes forment une base de C.
Une matrice génératrice G est donc de taille $k \times n$ et de rang k.

Définition. — Si la matrice G est de la forme $(I_k|P)$, on dit que le codage est **systématique** : les k premiers bits portent l'information et les n-k suivants sont de la redondance.

Définition. — On dit que deux codes linéaires de $\mathbb{F}_2^n$ sont équivalents si l'on obtient l'un à partir de l'aure par une permutation des coordonnées.

Ces codes ont alors le même cardinal et la même distance minimale.

Définition. — La **matrice de contrôle** d'un code C est la matrice génératrice du code dual :
$C^\perp = \{y \in \mathbb{F}_2^n, \forall c \in C, < y, c >= 0\}$

Proposition. — Soit H une matrice de contrôle du code linéaire C.
La distance minimum d_C est le plus grand entier k tel que $k-1$ colonnes de H sont toujours linéairement indépendantes.

9.3 Quelques codes linéaires

Exemple. Code de Hamming—
Un code de Hamming de paramètre $r \geq 2$ est un code linéaire de longueur $n = 2^r - 1$ tel que sa matrice de contrôle $H(r)$ de r lignes et $2^r - 1$ colonnes ait toutes ses colonnes distinctes deux à deux et non-nulles.
A permutation près, on peut supposer que la i-ième colonne de $H(r)$ représente l'écriture binaire de i sur r bits.
Les codes de Hamming sont de distance $d_C = 3$, ce qui n'est pas une très grande distance de correction d'erreurs, mais ils sont très faciles à décoder.
De plus, ce sont des codes parfaits : La réunion pour x dans C des $B(x, 3)$ donne $\mathbb{F}_2^{2^r-1}$ tout entier.

Exemple. Codes cycliques— Ce sont les codes linéaires stables par l'automorphisme cyclique de décalage des coordonnées : $(x_1, \ldots, x_n) \mapsto (x_n, x_1, \ldots, x_{n-1})$
En identifiant cet automorphisme à la multiplication par X dans $\mathbb{F}_2[X]/(X^n - 1)$, un code cyclique C est alors un idéal de $\mathbb{F}_2[X]/(X^n - 1)$.
Cela revient donc à considérer un idéal de $\mathbb{F}_2[X]$ contenant $(X^n - 1)$, càd un idéal engendré par un diviseur g de $X^n - 1$ car $\mathbb{F}_2[X]$ est euclidien.
Ce diviseur g de $X^n - 1$ tel que $(\overline{g}) = C$ est appelé *polynôme générateur* de C.
Si $g(X) \neq X^n - 1$, une base de C est alors $g(X), Xg(X), \ldots, X^{n-1-\deg(g)}g(X)$, et C est de dimension $k = n - \deg(g)$.
Le procédé de codage $\mathbb{F}_2^k \to C$ se fait de la comme suit : le vecteur $(x_1, \ldots, x_k)$ est codé par le polynôme $c = c_I - c_R$, où $c_I(X) = x_1X^{n-1} + \cdots + x_rX^{n-r}$ et où c_R (de degré $< n - k$) est le reste de la div.eucl. de c_I par g. (c_I porte l'information et c_R la redondance)
Si on suppose que n est premier avec 2, alors $X^n - 1$ est à racines simples dans son corps de décomposition.
Notons K le corps de décompo de $X^n - 1$ sur $\mathbb{F}_2$ et soit α une racine primitive n-ième de l'unité dans K.
$P := Irr(\alpha, \mathbb{F}_2)$ a alors pour degré l'ordre de 2 dans $(\mathbb{Z}/n\mathbb{Z})^*$ et ses racines sont $\alpha, \alpha^2, \alpha^4, \ldots, \alpha^{2^{r-1}}$.
Donc $[K : \mathbb{F}_2] = r$, et on sait de plus que le polynôme cyclotomique Φ_n se factorise sur $\mathbb{F}_2$ en un produit de polynômes irred de degré r, dont P fait partie.
Le polynôme générateur g va ainsi être déterminé par ses racines dans K, qui forment un sous-ensemble de $\{1, \alpha, \ldots, \alpha^{n-1}\}$.
Pour $U \subset \mathbb{Z}/n\mathbb{Z}$, $g_U = \Pi_{i \in U}(X - \alpha^i)$, g_U est à coeffs dans $\mathbb{F}_2$ ssi U est stable par la multiplication par 2 dans $\mathbb{Z}/n\mathbb{Z}$. (car $\mathbb{F}_2$ est l'ensemble des racines de $X^2 - 1$)
On a ainsi une bijection entre les codes cycliques de longueur n et les parties de $\mathbb{Z}/n\mathbb{Z}$ stables par la multiplication par 2.

On peut alors montrer que si U contient s entiers consécutifs, alors le code cyclique de polynôme générateur g_U est soit nul, soit a une distance de Hamming $\geq s+1$.

Exemple. Codes BCH—
Les codes BCH (Bose-Chaudhuri-Hocquenghem) sont des codes cycliques particuliers. La famille des codes BCH contient les codes de Reed-Solomon qui servent à la lecture des CD.
Les codes BCH binaires primitifs sont de longueur $n = 2^r - 1$.
L'entier r est ainsi l'ordre de 2 dans $(\mathbb{Z}/n\mathbb{Z})^*$, et le corps K de décomposition de $X^n - 1$ sur $\mathbb{F}_2$ est $\mathbb{F}_{2^r}$.
Tous les calculs de décodage vont se faire sur K, que l'on voit aussi comme un $\mathbb{F}_2$-ev de dimension r.
Soit α une racine primitive n-ième de l'unité dans K, que l'on se donne par son poly minimal P sur $\mathbb{F}_2$.
Tout élément de K^* est donc un α^i, et s'écrit de manière unique comme combinaison linéaire à coeffs dans $\mathbb{F}_2$ de $1, \alpha, \ldots, \alpha^{r-1}$.
Un code BCH de longueur $n = 2^r - 1$ et de distance prescrite $\delta \leq n$ est le code cyclique de polynôme générateur g_U où U est la plus petite partie de $\mathbb{Z}/n\mathbb{Z}$ contenant $\{1, 2, \ldots, \delta - 1\}$ et stable par multiplication par 2.
Ainsi, un polynôme $c = x_{n-1}X^{n-1} + \cdots + x_0 \in \mathbb{F}_2[X]$ appartient à ce code si et seulement si
$c(1) = c(\alpha) = c(\alpha^2) = \cdots = c(\alpha^{\delta-1}) = 0$.
On peut trouver des exemples de codes BCH explicites dans le Demailly (p.213,p.238,p.240).

Exemple. Codes de Golay—
Ce sont des codes linéaires étendus que l'on note G_n.
Le code de Golay G_{24} a été utilisé pour la transmission de photos par les sondes Voyages I et II (1979-1981).
La matrice génératrice du code G_{24} est $G = (I_{12}|A)$ avec A = $\begin{pmatrix} 0 & 1 & 1 & \ldots & 1 & 1 \\ 1 & & & & & \\ \vdots & & & B & & \\ 1 & & & & & \end{pmatrix}$
et $B \in M_{11}(\mathbb{F}_2)$ une matrice circulante, càd : $B_{i+1,j} = B_{i,\tau(j)}$ où $\tau \in S_{11}$.
On a donc $B_{i,j} = B_{0,\tau^i(j)}$,avec ici : $\tau = (11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1)$ et $(B_{0,j})_j = (1\ 1\ 0\ 1\ 1\ 1\ 0\ 0\ 0\ 1\ 0)$.
Le code G_{24} est de type (24,12,8), ce qui en fait un très bon code correcteur dans le contexte car sa distante de Hamming est proche de la borne de Singleton.

10 Cryptographie

Définition. — La cryptographie est l'ensemble des principes, méthodes et techniques dont l'application assure le chiffrement et le déchiffrement des données, afin de préserver la confidentialité et l'authenticité de ces données.

Exemple. – Le chiffrement de César : On décale les lettres de l'alphabet de 3 lettres vers la droite : $(ATTAQUEZ \mapsto DWWDTXHC)$. Le chiffrement de César est une translation sur $\mathbb{Z}/26\mathbb{Z}$.

Le point commun des méthodes de cryptographie d'avant 1945 (jusqu'à la machine Enigma) est le fait que la méthode de chiffrage est secrète.
L'inconvénient c'est que la méthode de chiffrement n'est pas secrète pour ceux qui l'utilisent, donc une personne trahissant le secret (traître/torturé) ruine la sécurité du système.

Remarque. — Ceci est un principe de Kerchoffs : "La sécurité d'un système de chiffrement ne doit résider que dans la clé est non dans le procédé de chiffrement."

Vient alors la cryptographie à clé secrète, où deux correspondants chiffrent et déchiffrent leur message selon un procédé public, mais grâce à une clé de chiffrement que eux seuls possèdent.
Cette méthode est très rapide, mais il faut que les correspondants s'échangent la clé, et il faut aussi que chaque couple de correspondants aient une clé de chiffrement différente.
Exemples : DES (jusqu'en 1990), triple-DES, AES (standard), masque jetable.

10.1 Cryptographie à clé publique

Le modèle qui se développe alors et que l'on utilise couramment (cartes de crédit, authentification sur internet,...) est la cryptographie à clé publique.

Définition. — Pour un cryptographie a clé publique, son procédé de chiffrement est connu de tous, mais la clé de chiffrement et de déchiffrement ne sont pas les mêmes.
De plus, la clé de déchiffrement ne doit pas être trouvée à partir de la clé de chiffrement en un temps polynômial. ⇒ On peut ainsi distribuer publiquement la clé de chiffrement, et garder secrètement la clé de déchiffrement.

Remarque. — La cryptographie à clé publique est basée sur des fonctions "sans-retour" (Diffie,Hellman,1976) : des bijections f pour lesquelles calculer f(x) est simple, mais pour lesquelles calculer $f^{-1}(x)$ est très compliqué. (Un peu comme un piège à écrevisses).
Cette méthode résout les soucis d'échange et de gestion des clés de chiffrement de la méthode à clé privée, mais en contrepartie les calculs en sont alourdis.

En pratique, on mélange cryptographie à clé publique et à clé privée, comme dans le SSL : La crypto à clé publique est utilisée pour échanger une clé de session, puis la crypto à clé privée utilise cette clé de session pour chiffrer les communications.

Remarque. — Les méthodes de crypto à clé publique les plus connues sont :
- Le RSA : 1978, basé sur la difficulté de factoriser des nombres, cryptage très rapide, utilisé quasiment partout, clés de chiffrement plutôt grosses (1000 bits), les attaques sont de complexité sous-exponentielle
- Le logarithme discret sur les corps finis : les attaques sont de même complexité que le RSA, mais les clés de chiffrement sont plus petites
- Le logarithme discret sur des courbes elliptiques : aucune attaque connue en complexité sous-exponentielle, des clés de chiffrement plus petites (250 bits), implique des mathématiques plus poussées, ouvre à de nouvelles possibilités, recommandé pour le moment par la NSA (on ne connait pas encore d'attaque contre ce code programmable efficacement sur ordinateur quantique)
- Le logarithme discret sur des courbes hyperelliptiques : mêmes avantages que le log discret sur courbes elliptiques, pas encore breveté, implique des mathématiques encore plus poussées.
- Le NTRU : basé sur les réseaux (problème de la recherche du vecteur le plus proche), très jeune, possédant déjà des attaques majeures, mais sans aucun algorithme d'attaque en temps polynômial sur ordinateur quantique
- Et quelques autres : le problème de Knapsack ; McEliece (codes correcteurs d'erreur), Paillier (pour les votes électroniques), HFE (systèmes d'équations non-linéaires).

10.2 La méthode RSA

A ne pas confondre avec un mode de vie développé par la maître course, le système de chiffrement à clé publique RSA a été présenté en 1978 par Rivest, Shamir, et Adelman, et est entièrement basé sur la difficulté de factoriser des nombres entiers.

Définition. — La clé privée est constituée de deux nombres premiers assez gros p et q (environ 500 bits chacun, avec certaines conditions), et d'un entier d premier avec (p-1) et (q-1) (d est inversible dans $\mathbb{Z}/(p-1)(q-1)\mathbb{Z}$).
La clé publique est $n := p.q$ ainsi que e l'inverse de d modulo $(p-1)(q-1)$.

La clé publique est facile à obtenir à partir de la clé privée (algorithme d'Euclide étendu), mais si l'on se donne e et n, il est très difficile de retrouver d,p, ou q.

Proposition. Procédé de chiffrement —

- — Imaginons A détienne la clé privée et que B veuille envoyer un message à A en utilisant la clé publique que A lui a fournie.
- — B choisit son message m, un élément $\mathbb{Z}/n\mathbb{Z}$, et calcule son message chiffré $c := m^e$ mod(n)
- — B envoie c à A.
- — A reçoit c et calcule c^d mod(n)$\equiv m^{e.d}$ mod(n)$\equiv m$ mod(n), ce qui lui permet de retrouver m.
- — Comme A est la seule personne à connaître d, elle est la seule à pouvoir déchiffrer les messages chiffrés.

Remarque. — Les deux propriétés arithmétiques derrière cette méthode sont le théorème d'Euler (Pour $pgcd(a,n) = 1$, $a^{\varphi(n)} \equiv 1$ mod(n), avec ici $\varphi(n) = (p-1)(q-1)$) ainsi que le théorème des restes chinois (pour a,b premiers entre eux, on a un isomorphisme d'anneaux $\mathbb{Z}/ab\mathbb{Z} \simeq \mathbb{Z}/a\mathbb{Z} \times \mathbb{Z}/b\mathbb{Z}$).

On peut alors montrer que pour tout $m \in \mathbb{Z}/pq\mathbb{Z}$, $m^{k.(p-1)(q-1)+1} \equiv m$ mod(n), $\forall k \geq 0$. (même si m n'est pas premier avec p.q)
On peut de plus avoir un déchiffrement plus rapide en calculant $c^{d \bmod(p-1)}$ mod(p) et $c^{d \bmod(q-1)}$ mod(q) pour retrouver c^d mod(n) via le théorème des restes chinois. (méthode 4 fois plus rapide car p,q ont deux fois moins de bits que n et car d mod$(p-1)$, d mod$(q-1)$ ont deux fois moins de bits que d).

Pour générer une clé privée et une clé publique pour l'algorithme RSA, il y a besoin de deux nombres premiers très grands p et q et d'un entier d premier à (p-1)(q-1).
Pour trouver de tels nombres premiers, on choisir des nombres au hasard jusqu'à en trouver un qui soit premier. Il faut pour cela utiliser des tests de primalité, qui sont relativement lents. La proportion de nombre premiers plus petits que n étant équivalente à $\frac{n}{log(n)}$, trouver de nouveaux nombres premiers encore plus grands nécessite encore plus de calculs.
Une fois p et q choisis, on choisit e (qui sera dans la clé publique) pour calculer facilement d par algorithme d'Euclide étendu. On prend en général e=3 (e=2 ne marchant pas), ou e=65537. (car $65537 = 10000000000000001_2$ a une écriture binaire pauvre en 1, ce qui garantit un calcul de c^e par exponentiation binaire très rapide, et car e=3 peut apporter des failles)

Application. Cassage du RSA—
Pour ce qui est du cassage d'une clé privée, il faut réussir à calculer p ou q ou d.
Si l'on connaît n et e, alors connaître p $\Leftrightarrow$ connaître q $\Leftrightarrow$ connaître d.
Donc le cassage se ramène bien exclusivement à des tentatives de factoriser n=pq.
Voici les méthodes générales de cassage du RSA :

- — Attaque par force brute : Essayer de diviser n par tous les entiers $\leq \sqrt{n}$: $O(\sqrt{n})$ tests nécessaires.
- — Crible d'Eratosthène : Si $2 \nmid m$ alors on ne teste pas m : $O(\frac{\sqrt{n}}{\log(n)})$ tests nécessaires.
- — Crible quadratique : Utiliser des extensions de corps finis de degré 2 pour trouver des entiers x et y tels que $x^2 \equiv y^2$ mod(n) $\Rightarrow pgcd(x-y,n)|n$: Complexité sous-exponentielle $O(e^{\log(n)^{1/2}\log(\log(n))^{1/2}})$
- — Crible par les corps de nombres : Une généralisation du crible quadratique utilisant des extensions de corps finis de plus haut degré pour trouver x et y.
  Complexité sous-exponentielle en $O(e^{(\frac{64}{9})^{13}\log(n)^{1/3}\log(\log(n))^{2/3}})$

Application. Failles potentielles du RSA—
Dans le cas où e=3, si l'on connaît m^3 mod(n_1), m^3 mod(n_2), m^3 mod(n_3), pour $(n_1,3),(n_2,3),(n_3,3)$ trois différentes clés publiques, alors on peut facilement calculer m^3 mod$(n_1n_2n_3)$ avec l'isomorphisme chinois, donc connaître m^3 car $m^3 < n_1n_2n_3$, et enfin utiliser la méthode de Newton polynômiale pour retrouver m la racine entière positive de $X^3 - m^3$.

De plus, si différentes personnes conçoivent des clés privées (p,q,d) et (p_1,q_1,d_1) avec $p = p_1$ et $q \neq q_1$ (ou vice-versa), alors on peut casser ces deux clés privées en même temps en calculant $pgcd(pq,p_1q_1) = p_1 = p$.
(Beaucoup de petits sites utilisant le RSA vont simplement demander (2^k).nextprime() pour trouver p et q, ce qui fait que cette situation peut facilement arriver)

Si au contraire on cherche à prendre d aussi petit que possible pour avoir un déchiffrement plus rapide, l'attaque de Wiener permet de retrouver d très rapidement en utilisant le développement en fraction continue de $\frac{e}{n}$. Cette attaque ne marche que si $q < p < 2q$ et si $d < \frac{1}{3}.n^{\frac{1}{4}}$ (peut être étendu à $d < n^{0292}$).

Remarque. — La taille des entiers n=pq que l'on a réussi à factoriser à progressé bien plus vite que ce qu'aurait dit la "fameuse" loi de Moore (On factorisait des n à 120 bits en 1993, et des n à 231 bits en 2010, au lieu de 137 bits selon la loi de Moore). Cela est dû à des améliorations importantes dans l'algorithme de factorisation.

10.3 Le logarithme discret

Définition. — Soit G un groupe, et $g \in G$ d'ordre fini l. Soit $H =< g >$.
Alors, pour tout $h \in H$, $\exists! n \in \{0, \dots, l-1\}$ tel que $h = g^n$.
Un tel n est appelé le logarithme discret de h en base g et est noté $\log_g(h)$, et est déterminé modulo l.

Exemples : $(\mathbb{Z}/n\mathbb{Z}, +)$, $(\mathbb{F}_q^*, \times)$, une courbe elliptique, le Jacobien d'une courbe hyperelliptique,...

Le but est de trouver un groupe pour lequel le calcul du logarithme discret est difficile, afin de l'utiliser en cryptographie.

Echange de clé de Diffie-Hellman :
- Données publiques : un groupe G et $g \in G$ d'ordre l.
- A choisit un nombre a dans $\{1, \dots, l-1\}$, calcule g^a, et envoie g^a à B.
- B choisit un nombre b dans $\{1, \dots, l-1\}$, calcule g^b, et envoie g^b à A.
- A peut ainsi calculer $(g^b)^a = g^{ab}$, et B peut ainsi calculer $(g^a)^b = g^{ab}$.
- A et B partagent ainsi le secret commun g^{ab}.
- Un espion connaîtrait G,g,g^a, et g^b, mais ne pourrait déduire g^{ab} sans résoudre un problème de logarithme discret.

Définition. — Un algorithme de calcul de logarithme discret est dit générique s'il n'utilise que les opérations suivantes :
- Le produit de deux éléments
- L'inverse d'un élément
- Le test d'égalité

En d'autres mots, cet algorithme peut être utilisé sur n'importe quel groupe.

Théorème. Théorème de Shoup—
Soit p le plus grand facteur premier divisant l l'ordre de g. Alors, calculer le logarithme discret associé à g en utilisant un algorithme générique nécessite au moins $O(\sqrt{p})$ opérations dans le groupe $< g >$.

Application. — La méthode "Pas de bébé, pas de géant" (Shanks) :
- On se donne $hin < g >$ et on cherche n tel que $h = g^n$.
- Soit $s = \lceil \sqrt{l} \rceil + 1$. Il y a $u, v < s$ tels que n=u+vs.
- On a alors : $h = g^{u+vs} \Rightarrow h = g^u.(g^s)^v \Rightarrow h(g^{-1})^u = (g^s)^v$
- On va alors calculer et stocker en mémoire les $h(g^{-1})^u$ pour $0 \leq u < s$, ainsi que g^s
- Puis, pour v allant de 0 à (s-1), on calcule $(g^s)^v$, et on le compare aux $h(g^{-1})^u$.
  S'il y a un u tel que $(g^s)^v = h(g^{-1})^u$, on s'arrête. Sinon on passe au v suivant.
- Cet algorithme nécessite $2\sqrt{l}$ opérations dans G.
  En contrepartie, il nécessaire de stocker $\sqrt{l}$ éléments de G en mémoire.

Exemple. — On prend $G = \mathbb{F}_p^*$ avec p=83. $Card(G) = 82 = 2 * 41$. On prend g=3, d'ordre 41.
On veut calculer $\log_3(30)$. Pour cela, on calcule s=7, puis $3^{-1} = 28$ mod(83), et $3^7 = 29$ mod(83).
La méthode "Pas de bébé, pas de géant" trouve n=3+4*7=31 en 10 étapes, là où une recherche par forc brute demanderait 31 étapes.

Pour un groupe de cardinal proche de 2^{80} (niveau de sécurité d'environ 40 bits), une opération réalisée par un PC récent sur ce groupe prend environ $10\mu s$. Donc 2^{40} opérations prennent environ 4 mois, on peut donc attaquer un logarithme discret sur un tel groupe en un temps raisonnable.
Cependant, stocker un élément de G nécessaire 80 bits, soit 10 octets. Les 2^{40} calculs dans G vont donc demander environ 10.000 GB de mémoire, qui doit être de la mémoire vive.
Le problème ici est la limitation de stockage en mémoire des données.

Application. — La méthode ρ de Pollard

— Cette méthode est basée sur le paradoxe des anniversaires : Si l'on tire au hasard deux éléments de G, le nombre de tirages avant une collision est d'environ $\sqrt{\frac{\pi.l}{2}}$.
— On réalise alors une marche aléatoire $w_{i+1} = \Phi(w_i)$ jusqu'à ce qu'une collision arrive, càd losque l'on aie μ et τ tels que $w_\mu = w_{\mu+\tau}$.
On aura alors $\mu\ \tau\ \sqrt{\frac{\pi.l}{8}}$.
— Afin de stocker moins d'information en mémoire, on utilise une astuce : si $i = k\tau$ et $i \geq \mu$, alors $w_i = w_{2i}$.
Ainsi, on démarre avec w_0 et $z_0 = w_0$, puis on avance avec $w_{i+1} = \Phi(w_i)$ et $z_{i+1} = \Phi(\Phi(z_i))$, et on compare w_{i+1} avec $z_{i+1} = w_{2i}$.
— Le stockage en mémoire est très faible, et on a toujours $O(\sqrt{l})$ opérations dans G.
Cependant, chaque itération demande d'appliquer 3 fois Φ.

Application au logarithme discret : Pour $w_i := g^{a_i}h^{b_i}$, on a $w_i = w_j \Leftrightarrow h = g^{\frac{a_j-a_i}{b_i-b_j}}$, donc $n = \frac{a_j-a_i}{b_i-b_j}$ mod(l).
Et une telle marche aléatoire est facile à programmer en parallèle sur plusieurs ordinateurs. Un logarithme discret sur une courbe elliptique à 109 bits (donc 55 bits de sécurité) a été cassé en 2002 avec cet algorithme en utilisant 10.000 ordinateurs en parallèle pendant 549 jours.

Remarque. — Résumé des contraintes sur G :

— G doit contenir un sous-groupe cyclique d'ordre premier p sur lequel le problème de logarithme discret sera appliqué.
— Si l'on veut n bits de sécurité, il faut que p aie 2n bits.
— On cherche des groupes pour lesquels il n'y a pas d'attaque qui soit meilleure que les attaques dites génériques.

Un candidat pour G : $\mathbb{F}_p^*$

— $\mathbb{F}_p^*$ est de cardinal p-1, et est cyclique. C'est un candidat naturel pour G.
— L'algorithme de calcul d'index peut déterminer un logarithme discret sur un tel groupe en un temps sous-exponentiel.
Un niveau de sécurité de 80 bits implique que $p \sim 2^{1024}$ (même niveau de sécurité que le RSA)
— En pratique, on choisit p un nombre premier à 1024 bits tel que p-1 est divisible par un nombre premier l de 160 bits.
Dans ce cas, les opérations ont lieu dans $\mathbb{F}_p^*$ mais les clés (exposants) vivent dans $\mathbb{Z}/l\mathbb{Z}$.
— Les clés sont alors plus petites que pour le RSA (160 bits au lieu de 1024 bits).

D'autres candidats sont les $\mathbb{F}_{2^n}^*$, pour lesquels le calcul d'index marche de la même façon : 1024 bits sont nécessaires pour avoir 80 bits de sécurité.
Pour les courbes elliptiques et les courbes hyperelliptiques pour lesquelles personnes ne connaît d'attaques plus efficaces que les attaques génériques, 160 bits sont suffisants pour 80 bits de sécurité.

Par rapport à la méthode RSA, le chiffrement avec le logarithme discret fournit des clés plus petites, un déchiffrement plus rapide (des exposants de 160 bits au lieu de 1024 bits), une génération de clé triviale, mais un chiffrement plus lent.

Commandes Sage

Voir : "Aide-mémoire Sage" de G.Lepetit, Mars 2017.

- Objets usuels
- Listes et temps de calcul
- Graphes
- Arithmétique
- Polynômes
- Matrices, systèmes linéaires