Licence d'informatique. Algorithmique et programmation. Cours avancé

Jacques Stern

Année 2010–2011

Table des matières

Cours 1 (5 octobre)

Algorithmes : conception et évaluation

— Algorithmes approchés et analyse amortie

1.1 Algorithmes approchés

1.1.1 Présentation

Le but de cette partie est de présenter des exemples d'algorithmes permettant de résoudre de manière approchée un problème d'optimisation dont le problème de décision associé est connu comme NP-complet. Le problème choisi est le "bin packing" (en français problème de *placement* ou *empaquetage*). L'intérêt du problème est double. D'une part, il existe des algorithmes pratiques et efficaces avec une *garantie de performance*, ce qui veut dire que la solution trouvée par l'algorithme est dans un rapport constant de l'optimum. D'autre part, il existe des algorithmes (théoriques) donnant à ce rapport une valeur aussi proche que possible de 1. En d'autres termes, même si le problème est NP-complet, on peut s'approcher autant qu'on veut de l'optimum par un algorithme polynomial. Toutefois, le degré du polynôme qui mesure la complexité augmente avec la précision choisie.

Le problème du "bin packing" s'énonce de la manière suivante : étant donnés un entier n et un ensemble $U = (u_1, \cdots, u_n)$ de n objets, auxquels est associée une taille $s(u_i) \in [0, 1]$, trouver une partition de U en k sous-ensemble X de U, k étant choisi minimum et chaque ensemble X de la partition tel que

$$\sum_{i \in X} s(u_i) \leq 1$$

En d'autres termes, on place les objets u_i dans des boîtes de taille 1 et on cherche à minimiser le nombre de boîtes nécessaires.

1.1.2 Algorithmes pratiques

L'algorithme le plus simple est "first fit" ou FF : on prend les objets de U l'un après l'autre et on place chaque objet dans la première boîte possible. En notant $X_j[\ell]$ l'ensemble des objets placés dans la j-ième boîte après qu'aient été traités les objets $u_1, \cdots u_{\ell-1}$, on choisit donc le plus petit indice j, tel que

$$\sum_{i \in X_j[\ell]} s(u_i) + s(u_\ell) \leq 1$$

et on ajoute u_ℓ à $X_j[\ell]$ pour former $X_j[\ell+1]$, les autres $X_q[\ell]$, $q \neq j$, étant inchangés.

Lemme 1 *Soit I une instance du problème "bin packing" et soit $OPT(I)$ l'optimum correspondant ; on a*

$$FF(I) \leq 2.OPT(I) + 1$$

Preuve On note d'abord que, puisque pour chaque élément X_j de la partition optimale, on a :

$$\sum_{i \in X_j} s(u_i) \leq 1,$$

il vient

$$\sum_{i=1}^{n} s(u_i) \leq \sum_{j} \sum_{i \in X_j} s(u_i) \leq k = OPT(I).$$

Par ailleurs, on ne peut trouver dans la partition obtenue par l'algorithme FF deux boîtes X_j et $X_{j'}$, $j < j'$, à moitié vides, c'est à dire telles que

$$\sum_{i \in X_j} s(u_i) \leq 1/2;$$

en effet, au moment où un premier objet est placé dans $X_{j'}$, la place disponible dans X_j permet l'ajout de cet objet, contredisant ainsi la règle de placement de l'algorithme. En notant j_0 l'indice de l'unique boîte à moitié vide, si elle existe, a donc pour $j \neq j_0$:

$$\sum_{i \in X_j} s(u_i) \geq 1/2.$$

En sommant, il vient :

$$\sum_{i=1}^{n} s(u_i) \geq \sum_{j \neq j_0} \sum_{i \in X_j} s(u_i) \geq (k-1) \times 1/2 = \frac{FF(I)-1}{2}.$$

Finalement, la somme $\sum_{i=1}^{n} s(u_i)$ est minorée par $\frac{FF(I)-1}{2}$ et majorée, ainsi qu'on l'a vu plus haut, par $OPT(I)$. Il vient

$$\frac{FF(I)-1}{2} \leq OPT(I),$$

et donc

$$FF(I) \leq 2.OPT(I) + 1.$$

On peut en fait démontrer par une combinatoire un peu plus fine que $FF(I) \leq 17/10.OPT(I) + 2$ (voir [2]). En effectuant, préalablement à l'exécution de l'algorithme FF, un tri rangeant les objets par taille décroissante, on améliore la garantie de performance qui devient $FFD(I) \leq 11/9.OPT(I)+2$.

1.1.3 Un algorithme théorique

Le résultat exposé dans cette section est dû à de la Vega et Lueker [5]. Il apparaît également dans [4].

Théorème 1 *Pour tout ε, $0 < \varepsilon \leq 1/2$, il existe un algorithme polynomial A_ε qui calcule pour toute instance I du problème bin packing une solution utilisant un nombre de boîtes majoré par $(1 + 2\varepsilon)OPT(I) + 1$.*

La preuve se fait en plusieurs étapes.

On commence par se restreindre aux instances telles que $s(u_i)$ soit toujours $\geq \varepsilon$ et ne prenne que K valeurs distinctes, ε et K étant fixés. On pose $M = \lfloor 1/\varepsilon \rfloor$. Pour tout placement, chaque boîte contient au plus M objets. En notant $t_1, \cdots, t_K$, les valeurs possibles prises par s, on définit le type d'une boîte comme la suite $n_1, \cdots, n_K$, chaque n_i étant le nombre de fois où un objet de taille k_i apparaît dans la boîte. Le nombre de types possibles de boîtes est majoré par une constante T qui est le nombre de possibilités de répartir $\leq M$ objets en K sous ensembles. On peut voir que cette constante est le coefficient du binôme $\begin{pmatrix} M+K \\ K \end{pmatrix}$, ce que le lecteur est invité à établir à titre d'exercice.

On dit que deux placements sont équivalents, si l'un est obtenu à partir de l'autre en appliquant à U une permutation qui respecte la taille $s(u)$ des

objets. A tout placement, on peut associer la fonction définie sur U par le type de la boîte dans laquelle un objet est placé. Cette fonction est constante sur chaque classe et elle est injective sur les classes d'équivalence. En effet, on peut l'inverser à équivalence près en associant à chaque boîte un ensemble d'objets ayant les tailles et les répétitions de tailles prescrites par le type de la boîte. Finalement, le nombre de classes est majoré par le nombre de façons de répartir n objets en T, et donc par $\begin{pmatrix} n+T \\ T \end{pmatrix}$, qui est un polynôme en n. L'optimum peut ainsi être trouvé par recherche exhaustive sur ce nombre polynomial de classes.

La seconde étape se restreint toujours aux instances telles que $s(u_i)$ soit $\geq \varepsilon$ mais abandonne la condition sur le nombre de valeurs distictes. On commence par trier les objets de U en ordre croissant de taille et on crée $K = \lceil 1/\varepsilon^2 \rceil$ groupes d'objets consécutifs ayant tous $Q = \lfloor n\varepsilon^2 \rfloor$ éléments exactement, sauf le dernier groupe moins nombreux. En arrondissant par excès chaque valeur de s à la taille du plus grand objet de chaque groupe, on est ramené à une instance J du cas précédent.

Lemme 2 *On a* $OPT(J) \leq (1+\varepsilon).OPT(I)$.

Preuve du lemme On introduit l'instance J' où chaque valeur de s est arrondie par défaut à la taille du plus petit objet de chaque groupe. A partir d'un placement optimal pour I, on obtient directement un placement pour J' ayant le même nombre de boîtes, par simple arrondi des tailles. On a donc :

$$OPT(J') \leq OPT(I).$$

De même, à partir d'un placement optimal pour J', on définit un placement pour J en remplaçant dans chaque boîte chacun des Q objets d'un groupe donné par les Q objets du groupe précédent. Cette opération, qui est le cœur de la preuve est possible car la taille (dans J) de chaque remplaçant est $\leq$ à celle (dans J') de l'objet qu'il remplace. Bien entendu, l'opération de remplacement ne peut s'appliquer aux objets du premier groupe, pusqu'il n'y a pas de groupe précedent. On place ces objets dans Q boîtes supplémentaires. Finalement,

$$OPT(J) \leq OPT(J') + Q \leq OPT(I) + Q \leq OPT(I) + n\varepsilon^2.$$

On note enfin que, en suivant l'argument de la preuve du lemme 1, on a $OPT(I) \geq \sum_i s(u_i)$ et que cette quantité est $\geq n\varepsilon$, puisque les objets ont une taille $\geq \varepsilon$. En reportant dans l'inégalité ci-dessus, il vient, comme annoncé dans le lemme :

$$OPT(J) \leq OPT(I) + \varepsilon.OPT(I).$$

On peut maintenant présenter l'algorithme d'approximation polynomial A_ε :

1. enlever les objets de taille $\leq \varepsilon$,
2. appliquer l'algorithme défini à la seconde étape à l'instance J, obtenu en arrondissant par excès,
3. restaurer les tailles originales des objets,
4. placer les objets de taille $\leq \varepsilon$ par FF.

Si aucune boîte supplémentaire n'est créée à l'étape 4, le lemme 2, montre qu'on a $A_\varepsilon(I) \leq (1+\varepsilon)OPT(I)$. Sinon, toutes les boîtes, sauf éventuellement la dernière sont remplies à au moins $1-\varepsilon$. Il vient, par un argument analogue à celui du lemme 1 :

$$(1-\varepsilon)(A_\varepsilon(I)-1) \leq \sum_i s(u_i) \leq OPT(I),$$

ce qui donne

$$A_\varepsilon(I) \leq \frac{OPT(I)}{1-\varepsilon} + 1.$$

Comme ε est $\leq 1/2$, on peut majorer $\frac{1}{1-\varepsilon}$ par $1+2\varepsilon$ On obtient ainsi :

$$A_\varepsilon(I) \leq (1+2\varepsilon).OPT(I) + 1.$$

1.2 Analyse amortie

1.2.1 Présentation

Le but de cette partie est de présenter le principe de l'analyse *amortie*. Dans certains cas, l'analyse de la complexité d'un algorithme par majoration du coût dans le cas le pire n'est pas significative. d'autres mesures sont possibles :

- l'analyse en moyenne, qui évalue l'espérance mathématique du temps de calcul, pour une distribution de probabilité donnée sur les instances,
- l'analyse amortie qui évalue la complexité cumulée d'un grand nombre d'exécutions successives de l'algorithme.

On étudie un algorithme très simple qui effectue des insertions et des suppressions dans un tableau et gére l'allocation mémoire. Cet algorithme opére comme suit sur le tableau T, la taille du tableau $size$, l'indice de la première place disponible num :

1. *initialisation.* Un tableau de taille 1 avec $num = 0$ est créé ;

2. *insertion non critique* si $num < size$ un objet est inséré à la première place disponible ; num est incrémenté d'une unité ;
3. *insertion critique* si $num = size$ un tableau de taille $2.size$ est créé ; les objets de l'ancien tableau sont recopiés ; un nouvel objet est inséré à la première place disponible ; num est incrémenté d'une unité ;
4. *suppression non critique* si $num \geq size/4$ l'objet en position num est supprimé ; num est décrémenté d'une unité ;
5. *insertion critique* si $num = size/4$ un tableau de taille $size/2$ est créé ; les objets de l'ancien tableau sont recopiés ; l'objet en position num est supprimé ; num est décrémenté d'une unité ;

Le résultat qu'on se propose d'établir est le suivant :

Théorème 2 *Le coût d'une suite de m insertions et suppressions successives à partir d'une initialisation est un $O(m)$.*

Preuve On introduit une fonction potentiel $\Phi(T)$ égale à
$\Phi(T) = 2.num - size$, si $num \geq size/2$
$\Phi(T) = size/2 - num$, si $num \leq size/2$.

Lemme 3 *Si c_i est le coût de la i-ème opération d'insertion ou de suppression et si Φ_i est la valeur du potentiel après cette i-ème opération, on a :*

$$-1 \leq c_i - (\Phi_i - \Phi_{i-1}) \leq 3.$$

Avant d'établir le lemme, on observe que le théorème s'en déduit facilement. En effet, en sommant les inégalités ci-dessus pour les différentes valeurs de i, on obtient

$$-m \leq \sum_{i=1}^{m} c_i - \Phi_m + \Phi_0 \leq 3m.$$

Comme Φ_m est majoré par la taille $size$ du tableau T après les m opérations et que cette taille ne peut dépasser la plus petite puissance de deux qui excède m, et donc pas $2m$, on a bien

$$\sum_{i=1}^{m} c_i = O(m).$$

Reste à établir le lemme. Il y a quatre cas à traiter. On se borne à en examiner deux.
insertion critique : Le potentiel avant l'exécution d'une telle insertion est $2.num - size = size$. Après doublement de la taille et insertion, le potentiel

devient $2(size + 1) - 2.size = 2$. La différence de potentiel $\Delta\Phi$ est donc $size - 2$. Quant au coût c de l'opération, il se compose du coût de copie du plus petit tableau au plus grand, soit $size$, et de l'insertion proprement dite de coût unitaire. On a ainsi $c - \Delta\Phi = 3$.
suppression non critique : La quantité $size$ retant inchangée, la différence de potentiel est de -2 ou $+1$ suivant la partie de la fonction linéaire par morceaux Φ qui est utilisée. Quant au coût c de l'opération, il est unitaire. On a ainsi $c - \Delta\Phi = 3$ ou 0.

Bibliographie

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*, 2nd ed. Cambridge, Massachusetts : M.I.T. Press, 2001.

[2] D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM J. Comput.*, 3 :256–278, 1974.

[3] M. R. Garey and D. S. Johnson. *Computers and Intractability.* Freeman, 1979.

[4] V. V. Vazirani. *Approximation Algorithms*, Springer (2001)

[5] W. de la Vega, G. Lueker. Bin packing can be solved within $1 + \varepsilon$ in linear time. *Combinatorica*, 1(4) :349-355, 1981.

Cours 2 (12 octobre)

Tri et hachage

— Tri de Shell

2.1 Le tri de Shell

2.1.1 Présentation

Le tri de Shell est ainsi appelé à la suite du nom de l'auteur de l'article [3] où il a été introduit. L'idée est simple : on se donne une suite décroissante d'entiers appelés *incréments*, soit $h_t, h_{t-1}, \cdots, h_1$, et on trie successivement, pour i allant de t à 1, un tableau donné en prenant les éléments "de h_i en h_i", c'est à dire en triant séparément les tableaux extraits dont les indices valent 1 modulo h_i, 2 modulo h_i, et ce jusqu'à h_i modulo h_i. D'un point de vue heuristique, il apparaît que les tris "grossiers", réalisés avec des incréments de taille élevés, réduisent la complexité des tris "fins" exécutés ultérieurement avec des incréments petits.

De manière surprenante, il n'est pas simple de donner à cette observation heuristique un contenu mathématique précis, comme on va le voir dans la suite. L'exposé qui suit s'inspire largement de Knuth [4].

2.1.2 Le théorème de commutation

Le résultat qui suit montre que la succession de tris fait sens dans la mesure où un tri ne détruit pas les progrès obtenus précedemment. Soit k un entier, on dit qu'un tableau est k-trié si les tableaux extraits en prenant les éléments de k en k sont triés. En notant le tableau A, cette définition exprime qu'on a $A[i] \leq A[i+k]$, pour tous entiers i tels que i et $i+k$ soient des indices du tableau. Un k-tri est un algorithme qui retourne un tableau k-trié.

Théorème 3 *Soient $k > h$ deux incréments. En appliquant un h-tri à un tableau k-trié, on obtient un tableau qui est à la fois k-trié et h-trié.*

La preuve se fonde sur le lemme suivant :

Lemme 4 *Soient m, r, $n \geq r$, des entiers. Soient $(x_1, \cdots, x_{m+r})$ et $(y_1, \cdots, y_n)$ deux suites telles que l'on ait pour tout j, $1 \leq j \leq r$, $y_j \leq x_{m+j}$. Si l'on trie séparéméent les deux suites, alors, en notant $(\overline{x}_1, \cdots, \overline{x}_{m+r})$ et $(\overline{y}_1, \cdots, \overline{y}_n)$, les suites triées obtenues, on a $\overline{y}_j \leq \overline{x}_{m+j}$.*

Preuve du lemme Dans la suite triée, $\overline{x}_{m+j}$ domine les éléments $\overline{x}_i$ pour $i \leq m+j$, qui sont au nombre de $m+j$. Parmi ces $m+j$ éléments, il y en a au moins j qui proviennent d'éléments de la suite initiale de la forme $x_{m+\ell}$. Ces j éléments dominent respectivement les y_ℓ correspondants. Il y a donc j indices ℓ de la suite non triée $(y_1, \cdots, y_n)$ inférieurs ou égaux à $\overline{x}_{m+j}$. Donc le j-ième élément de la suite triée $\overline{y}_j$ est bien $\leq \overline{x}_{m+j}$.
Preuve du théorème A partir du lemme, il est facile d'établir le théorème. En effet, il s'agit de montrer que dans le tableau final $\overline{A}$, on a $\overline{A}[i] \leq \overline{A}[i+k]$. En notant $u = i \bmod h$ et $v = i+k \bmod h$; puisque $u - v = k \bmod h$ et que $|u - v| \leq h < k$, on peut poser $m = \frac{u-v+k}{h}$. On considère alors la suite la suite des éléments du tableau initial pris de h en h qui contient $A[i+k]$, soit $x_\ell = A[v + \ell h]$. L'élément x_m vaut $A[u+k]$ et, comme l'indice $u+k$ est $\leq$ à l'indice $i+k$, on reste bien dans le tableau. On considère de même la suite $y_\ell = A[u + \ell h]$ des éléments du tableau initial pris de h en h qui contient $A[i]$. On note que

$$y_\ell = A[u + \ell h] \leq A[u + k + \ell h] = A[v + (\frac{u - v + k}{h} + \ell)h],$$

soit $y_\ell \leq x_{m+\ell}$, cette inégalité valant pour autant que les deux termes y_ℓ et $x_{m+\ell}$ soient définis, soit pour ℓ au plus égal à un entier r. On se trouve dans les conditions du lemme. En posant $\ell = \frac{i-u}{h}$, il vient :

$$\overline{A}[i] = \overline{y}[u + \ell h] \leq \overline{x}_{m+\ell} = \overline{A}[v + (\frac{u - v + k}{h} + \frac{i - u}{h})h] = \overline{A}[i + k],$$

ce qui achève la démonstration.

2.2 Tris de Shell à deux passes

2.2.1 La suite d'incréments $(2, 1)$

On considère ici le cas le plus simple du tri de Shell, consistant en un 2-tri suivi d'un 1-tri. La question qui se pose est de mesurer le gain sur

le tri le plus élémentaire par insertion. La complexité de cet algorithme de tri, quand il est appliqué à un tableau donné A réalisant une permutation des n premiers entiers est égal au nombre d'inversions de cette permutation, ce qui donne dans le plus mauvais cas $\frac{n(n-1)}{2} \simeq \frac{n^2}{2}$. Effectuer deux 2-tris à la place (par insertion également), réduit cette complexité d'un facteur multiplicatif 2. Reste à examiner la complexité résiduelle du tri appliqué à une permutation 2-triée, ce qui est un joli problème combinatoire. On va voir que cette complexité est en moyenne $O(n^{3/2})$, mais ce n'est pas si simple...

2.2.2 Le nombre d'inversions d'une permutation 2-triée

Il y a en tout $\left(\begin{array}{c} n \\ \lfloor n/2 \rfloor \end{array} \right)$ permutations 2-triées des entiers de 1 à n. En effet, pour définir complètement une permutation 2-triée il suffit de préciser les $\lfloor n/2 \rfloor$ entiers qui sont en position paire.

Par ailleurs, une représentation graphique d'une permutation 2-triée est possible sous forme d'une "escalier" construit à partir du tableau A qui définit la permutation. Cet escalier est formé de sommets P_k obtenus en examinant les entiers successifs k et en partageant l'ensemble $\{1, \cdots, k\}$ en deux sous-ensembles Odd_k et $Even_k$ respectivement formés des entiers j placés en position impaire et paire dans le tableau A. Le sommet P_k est égal à $(|Even_k|, |Odd_k|)$, où $|X|$ désigne le nombre d'élements de X. L'escalier est formé en partant de $(0, 0)$ et en joignant les sommets correspondant à des k consécutifs. En se reportant aux transparents du cours - en page 13 - on trouve une représentation de l'escalier correspondant au tableau

$$2, 1, 3, 4, 6, 5, 7, 10, 8, 11, 9, 12, 14, 13, 15.$$

On constate que l'arête correspondant à k va vers la gauche si k est en position paire et vers le bas si k est en position impaire. L'escalier *régulier*, correspondant à la permutation identité est en pointillés. Il se trouve que le nombre d'inversions est égal à la surface enserrée entre l'escalier régulier et l'escalier correspondant à la permutation. Pour le voir, on imagine qu'on place successivement les entiers k dans le tableau A, soit en position paire soit en position impaire. On observe que pour k impair, on a :

1. Si $Even(k-1) = Odd(k-1)$, l'adjonction d'un k en position paire ou impaire ne produit pas de nouvelle inversion avec les ℓ déjà placés.
2. Si $Even(k-1) > Odd(k-1)$, l'adjonction d'un k en position paire ne produit pas de nouvelle inversion avec les ℓ déjà placés, tandis que l'adjonction d'un k en position impaire produit un nombre de de nouvelles inversions égal à $Even(k-1) - Odd(k-1)$.

3. Si $Even(k-1) < Odd(k-1)$, l'adjonction d'un k en position impaire ne produit pas de nouvelle inversion avec les ℓ déjà placés, tandis que l'adjonction d'un k en position paire produit un nombre de de nouvelles inversions égal à $Odd(k-1) - Even(k-1) - 1$.

En interprétant géométriquement, on constate que seules les arêtes verticales allant de P_{k-1} à P_k situées en dessus de l'escalier, ainsi que les arêtes horizontales allant de P_{k-1} à P_k situées en dessous de l'escalier, apportent une contribution au nombre d'inversions et que cette contribution est égale à la surface de de la bande horizontale (respectivement verticale) allant de l'arête P_{k-1}, P_k à l'arête de l'escalier régulier de même ordonnée (respectivement de même abscisse).

On laisse au lecteur le soin de justifier le même résultat pour k pair.

Muni de cette interprétation géométrique, on peut calculer le nombre de permutations dont l'escalier passe par l'arête verticale allant de (i,j) à $(i,j+1)$. On trouve

$$\left(\begin{array}{c} i+j \\ i \end{array} \right) \left(\begin{array}{c} n-i-j-1 \\ \lfloor n/2 \rfloor - i \end{array} \right).$$

Le premier coefficient du binôme vient de la manière de placer i arêtes horizontales et j arêtes verticales pour former un escalier allant de $(0,0)$ à (i,j). Le second vient de la manière de placer $\lfloor n/2 \rfloor - i$ arêtes horizontales et $\lceil n/2 \rceil - j - 1$ arêtes verticales pour former un escalier allant de $(i, j+1)$ à $(\lfloor n/2 \rfloor, \lceil n/2 \rceil)$.

Chaque arête contribuant à la surface de la bande horizontale allant jusqu'à l'escalier régulier par $|i-j|$, le nombre total d'inversions de toutes les permutations 2-triées est pour n pair, $n = 2q$:

$$A_{2q} = \sum_{i=0}^{q} \sum_{j=0}^{q} |i-j| \left(\begin{array}{c} i+j \\ i \end{array} \right) \left(\begin{array}{c} 2q-i-j-1 \\ q-j \end{array} \right),$$

et pour n impair, $n = 2q+1$:

$$A_{2q+1} = \sum_{i=0}^{q} \sum_{j=0}^{q} |i-j| \left(\begin{array}{c} i+j \\ i \end{array} \right) \left(\begin{array}{c} 2q-i-j \\ q-j \end{array} \right),$$

Il se trouve que A_n a la forme simple $\lfloor n/2 \rfloor \times 2^{n-2}$, mais que ce n'est pas trop simple à établir.

2.2.3 Une forme close pour le nombre d'inversions d'une permutation 2-triée

On commence par noter, en échangeant i et j, que $2A_{2q}$ vaut

$$\sum_{i=0}^{q}\sum_{j=0}^{q}|i-j|\left(\begin{array}{c} i+j \\ i \end{array}\right)\left(\begin{array}{c} 2q-i-j-1 \\ q-j \end{array}\right)+\sum_{i=0}^{q}\sum_{j=0}^{q}|i-j|\left(\begin{array}{c} i+j \\ i \end{array}\right)\left(\begin{array}{c} 2q-i-j-1 \\ q-j-1 \end{array}\right).$$

Par une formule standard, ceci mène à :

$$\sum_{i=0}^{q}\sum_{j=0}^{q}|i-j|\left(\begin{array}{c} i+j \\ i \end{array}\right)\left(\begin{array}{c} 2q-i-j \\ q-j \end{array}\right),$$

et donc $A_{2q+1} = 2A_{2q}$.

En utilisant maintenant la symétrie en i, j, on a

$$A_{2q} = \sum_{j\leq i\leq q}(i-j)\left(\begin{array}{c} i+j \\ i \end{array}\right)\left(\begin{array}{c} 2q-i-j \\ q-j \end{array}\right);$$

Il reste donc à étudier la suite u_n définie par :

$$u_n = \sum_{j\leq i\leq n}(i-j)\left(\begin{array}{c} i+j \\ i \end{array}\right)\left(\begin{array}{c} 2n-i-j \\ n-j \end{array}\right).$$

On introduit la série $\varphi(z) = \sum_{n=0}^{\infty} u_n z^n$.

Lemme 5 *On a* $\varphi(z) = \frac{z}{(1-4z)^2}$.

En développant en série le second membre, il vient :

$$\varphi(z) = \frac{z}{4}\frac{d}{dz}(\sum_{q=0}^{\infty}4^q z^q),$$

ce qui donne immédiatement $u_q = q4^{q-1}$. Cette égalité permet de trouver la formule close proposée plus haut pour A_n.

Le lemme est établi en plusieurs étapes :
Etape 1. Decomposition en série de $\frac{1}{\sqrt{1-4z}}$.
La fonction $(1-4z)^{-1/2}$ se décompose en une série de terme général

$$(-1/2)(-3/2)\cdots\left(-\frac{2i-1}{2}\frac{(-4z)^i}{!i}z^i\right),\cdots$$

lequel s'écrit

$$\frac{1.3.\cdots.(2i-1)2^i z^i}{!i} = \frac{1.3.\cdots.(2i-1)2.4.\cdots.(2i)z^i}{(!i)^2} = \left(\begin{array}{c} 2i \\ i \end{array}\right) z^i.$$

avec la convention $!0 = 1$.

Etape 2. Calcul de la série $\sum_{i=1}^{\infty} \left(\begin{array}{c} 2i+s \\ i \end{array}\right)$. Notant $f_s(z)$ cette somme, on sait déjà que $f_0(z) = \frac{1}{\sqrt{1-4z}}$.

A partir de l'égalite $\left(\begin{array}{c} 2i+1 \\ i \end{array}\right) = \frac{1}{2}\left(\begin{array}{c} 2i+2 \\ i+2 \end{array}\right)$, on déduit que $f_1 = \frac{1}{2}\frac{f_0(z)-1}{z}$. Notant $u = \sqrt{1-4z}$, on trouve $f_1(z) = \frac{1}{u}\left(\frac{1-u}{2z}\right)$.

On se propose de généraliser la formule

$$f_s(z) = \frac{1}{u}\left(\frac{1-u}{2z}\right)^s.$$

A partir de l'identité

$$\left(\begin{array}{c} 2i+s \\ i \end{array}\right) = \left(\begin{array}{c} 2i+s-1 \\ i \end{array}\right) + \left(\begin{array}{c} 2i+s-1 \\ i-1 \end{array}\right),$$

on déduit que $f_s(z) = f_{s-1}(z) + zf_{s+1}(z)$. Il suffit donc de montrer que la formule proposée satisfait la même récurrence. On considère :

$$\frac{1}{u}\left(\frac{1-u}{2z}\right)^{s-1} + z\frac{1}{u}\left(\frac{1-u}{2z}\right)^{s+1}.$$

En mettant en facteur $f_0(z) = \frac{1}{u}\left(\frac{1-u}{2z}\right)^s$, on trouve la somme :

$$\frac{2z}{1-u} + z\frac{1-u}{2z}.$$

En observant que $u^2 = 1-4z$ et donc que $4z = 1-u^2$, on peut transformer cette somme en

$$\frac{1+u}{2} + \frac{1-u}{2} = 1,$$

ce qui donne la relation cherchée.

Etape 3. Fin du calcul. On a à calculer

$$\varphi(z) = \sum_{n=0}^{\infty} \sum_{\leq j \leq n} (i-j) \left(\begin{array}{c} i+j \\ i \end{array}\right) \left(\begin{array}{c} 2n-i-j \\ n-j \end{array}\right) z^n.$$

En introduisant $s = i - j$ et $t = n - j$, il vient :

$$\sum_{i=0}^{\infty}\sum_{s=0}^{\infty}\sum_{t=0}^{\infty} s \begin{pmatrix} 2i+s \\ i \end{pmatrix} \begin{pmatrix} 2t+s \\ t \end{pmatrix} z^s z^i z^t.$$

Pour s fixé, la somme des termes où z^s apparaît est le produit de s et de deux séries, chacune de somme $f_s(z) = \frac{1}{u}\left(\frac{1-u}{2z}\right)^s$. On a donc à sommer

$$\sum_{s=0}^{\infty} \frac{1}{u^2} s\alpha^s,$$

avec

$$\alpha = z.\left(\frac{(1-u)^2}{4z^2}\right) = \frac{(1-u)^2}{4z} = \frac{1-u}{1+u}.$$

Sachant que $\sum_s s\alpha^s$ vaut $\frac{\alpha}{(1-\alpha)^2}$, on trouve pour $\varphi(z)$:

$$\frac{1}{u^2}\frac{1-u}{1+u}(\left(1-\frac{1-u}{1+u}\right)^{-2} = \frac{1}{u^2}\frac{1-u}{1+u}(\left(\frac{2u}{1+u}\right)^{-2},$$

soit enfin

$$\frac{1}{4u^4}(1+u)(1-u) = \frac{z}{(1-4z)^2}.$$

2.2.4 Performance optimale du tri à deux passes

La nombre moyen d'inversions d'une permutation 2-triée est, d'après ce qui précède,

$$\lfloor n/2 \rfloor \frac{2^{n-2}}{\begin{pmatrix} n \\ \lfloor n/2 \rfloor \end{pmatrix}}.$$

En utilisant la formule de Stirling on constate que c'est asymptotiquement $\sqrt{\pi/128}n^{3/2}$, qui est bien comme annoncé en $O(n^{3/2})$.

On démontre plus généralement que le nombre moyen d'inversions d'une permutation h-triée est majoré par un équivalent de $\sqrt{\pi}/8n^{3/2}h^{1/2}$ (cette majoration est un peu moins fine que celle ci-dessus pour $h = 2$). Il en résulte qu'un tri de Shell à deux passes correspondant à la suite d'incréments $(h, 1)$ a une complexité majorée par

$$\frac{h}{2}(\frac{n}{h})^2 + \sqrt{\pi}/8n^{3/2}h^{1/2}.$$

En equilibrant les deux termes on est amené à poser $n = (\frac{n}{4\pi})^{1/2}$, ce qui donne une complexité en $O(n^{5/3})$.

2.3 Tris à plus de deux passes

2.3.1 Incréments premiers entre eux

L'utilisation d'incréments premiers entre eux se révèle partculièrement efficcace à cause du résultat suivant.

Théorème 4 *Soient k et h deux incréments premiers entre eux. Si un tableau A est à la fois h trié et k trié, alors pour tout couple d'indices $i < j$ tels que la différence $j - i$ soit au moins $(k-1)(h-1)$, on a $A[i] \leq A[j]$.*

Le théorème est conséquence du lemme suivant :

Lemme 6 *Soient h et k deux entiers positifs premiers entre eux. Tout entier $n \geq (h-1)(k-1)$ s'écrit sous la forme $ah + bk$ pour des entiers positifs ou nuls a, b convenables.*

Preuve du lemme On pose $a = nh^{-1} \bmod k$ (c'est là qu'on utilise l'hypothèse). L'entier a est compris entre 0 et $k-1$ et donc $a(h-1)$ est majoré par $(k-1)(h-1)$ et aussi par n. Il en résulte que $ah \leq n + a < n + k$. Comme en outre $n - ah$ est divisible par k, on en déduit que ah est $\leq n$. La différence $n - ah$ est divisible par k donc de la forme bk.
Preuve du théorème A partir du lemme, il est facile d'établir le théorème. En effet, en écrivant $j-i$ sous la forme $ah+bk$, $a \geq 0$, $b \geq 0$, on peut considérer la suite d'indices $i, i+h, \cdots i+ah$, puis $i+ah, i+ah+k, \cdots, i+ah+bk$. Chaque saut se fait en croissant car les tableaux sont h-tris et k-triés. Donc $A[i] \leq A[j]$.

2.3.2 Incréments de la forme $2^s - 1$

On considère ici la suite d'incréments $h_s = 2^s - 1$, pour s décroissant de $\lfloor \log n \rfloor$ à 1. On pose $t = \lfloor \log n \rfloor$ et on majore la complexité des $t/2$ premières passes par la somme

$$\sum_{s=t/2}^{t} \frac{h_s}{2} \left(\frac{n}{h_s} \right)^2$$

qui est un $O(n^2 \sum_{s=t/2}^{t} \frac{1}{2^s})$, soit donc un $O(n^2/2^{t/2}) = O(n^{3/2})$.

Reste les $t/2$ dernières passes. On note que h_{s+2} et h_{s+1} sont premiers entre eux. Le lemme 6 montre que, lorsque l'on aborde le h_s-tri, les inversions correspondent à des indices i et j distants de $(h_{s+2} - 1)(h_{s+1} - 1)$ au plus. Comme on prend les indices de h_s en h_s, les indices j qui produisent une

inversion avec un indice i au sein d'un même tableau extrait sont dans un ensemble qui a $\frac{h_{s+2}h_{s+1}}{h_s}$ éléments. En faisant varier i, on voit que l'ensemble des h_s-tris a une complexité cumulée majorée par $\frac{nh_{s+2}h_{s+1}}{h_s}$, qui est un $O(n.2^s)$. En sommant on obtient un $O(n.\sum_{s=1}^{t/2} 2^s)$ qui est un $O(n.2^{t/2}) = O(n^{3/2})$.

La complexité du tri de Shell pour la suite d'incréments $2^s - 1$ est donc en $0(n^{3/2})$. Ce résultat a été prouvé dans [2]. Le problème général de la complexité du tri de Shell reste ouvert.

Bibliographie

[1] D. Knuth. *The Art of Computer Programming*, Volume 3 : Sorting and Searching. Addison-Wesley, Reading, MA, 1973.

[2] A. Papernov and G. Stasevich. A method for information sorting in computer memories. *Problems Inform. Transmission*, 1(3) :63–75, 1965.

[3] D. L. Shell. A High-Speed Sorting Procedure. *CACM*, 2(7) :30-32, July 1959.

Cours 3 (19 octobre)

Recherche de motifs

— Algorithmes d'alignement en bio-informatique

3.1 Extraction et alignements pour deux suites

3.1.1 Motivation bio-informatique

De nombreux problèmes de bio-informatique conduisent à des algorithmes ayant pour but l'extraction de motifs communs à deux ou plusieurs suites de symboles et l'alignement de suites par adjonction d'espaces :

- Reconstruction de séquences DNA à partir de fragments,
- stockage et recherche dans les bases de données de séquences DNA,
- comparaison de séquences.

Le but de cette section et de la suivante est de donner un aperçu de ces algorithmes.

3.1.2 Algorithme d'extraction de sous-suite

Soient $X = (x_1, \cdots, x_m)$ et $Y = (y_1, \cdots, y_n)$ deux suites de symboles. Une suite extraite U de X est obtenue à partir d'une suites d'indices $j_1, \cdots, j_k$, $1 \leq j_i \leq m$ en posant $U = (x_{j_1}, \cdots, x_{j_k})$. On appelle plus longue suite extraite commune (PLSEC) à X et Y une suite extraite à la fois de X et Y de longueur k maximale.

L'extraction d'une PLSEC est un exemple typique de *programmation dynamique*. On construit un tableau $c[i, j]$ qui contient la valeur optimale de la longueur d'une PLSEC pour des suites X_i et Y_j obtenues par troncature, c'est à dire que $X_i = (x_1, \cdots, x_i)$, $0 \leq i \leq m$ et de même pour Y_j (X_0 ou Y_0

est la suite vide). Le tableau est rempli par récurrence en utilisant le lemme suivant :

Lemme 7 *Soient* $i, j > 0$.
i) Si $x_i = y_j$, *alors* $c[i,j] = c[i-1, j-1] + 1$;
ii) Si $x_i \neq y_j$, *alors* $c[i,j] = \max\{c[i, j-1], c[i-1, j]\}$.

La preuve du lemme est facile. Le lecteur pourra se reporter au cours d'algorithmique de Cormen et al. [3]. En maintenant dans un second tableau l'indication $\nwarrow$, $\leftarrow$, $\uparrow$ de la provenance de l'optimum (de $c[i-1, j-1]$, de $c[i-1, j]$, ou de $c[i, j-1]$), il est immédiat d'écrire un algorithme qui calcule non seulement la longueur de la PLSEC, soit $c[m, n]$, mais également une PLSEC de longueur maximale. Un tel algorithme est reproduit dans les transparents du cours, en page 9.

3.1.3 Algorithme d'alignement de deux suites

Soient deux suites S et T et soit "-" un symbole qui n'apparaît ni dans S, ni dans T. Un *alignement* est formé de deux suites $\overline{S}$ et $\overline{T}$, de même longueur obtenues respectivement à partir de S et T en insérant dans chacune des symboles "-" supplémentaires. On peut définir le coût d'un alignement à partir d'une fonction *score* $\sigma(a, b) \geq 0$, définie sur les couples de symboles : le coût d'un alignement est la somme des scores des lettres alignées. Le problème posé est celui de la recherche d'un algorithme qui calcule le coût minimal $Opt - align(X, Y)$ d'un alignement de X et Y. Il est résolu en calculant par programmation dynamique le coût $V[i, j]$ de l'alignement de suites tronquées et en utilisant le lemme suivant :

Lemme 8 *Soient* $i, j > 0$.

$$On\ a\ V[i,j] = \min \left\{ \begin{array}{c} V[i-1, j-1] + \sigma(s_i, t_j) \\ V[i, j-1] + \sigma(s_i, -) \\ V[i-1, j] + \sigma(-, t_j) \end{array} \right\}.$$

La preuve du lemme est immédiate. Comme dans le cas de la PLSEC, on peut obtenir un alignement optimal en maintenant dans un tableau l'indication $\nwarrow$, $\leftarrow$, $\uparrow$ de la provenance de l'optimum.

3.1.4 Pénalité pour les espacements

Il existe naturellement de nombreuses variantes de l'algorithme présenté à la section précédente. L'une d'elles, liée à des contraintes issues de la

bio-informatique, tient compte d'une *pénalité* s'appliquant aux longs espacements. En faisant le choix d'une pénalité linéaire de la forme $W_g + qW_s$, où q désigne le nombre d'espaces consécutifs insérés et où W_s, W_g sont ≥ 0, on est amené à une récurrence plus complexe : l'algorithme maintient outre $V[i,j]$, trois tableaux $E[i,j]$, $F[i,j]$, $G[i,j]$ contenant l'optimum respectif correspondant à chacun des trois cas suivants :

1. optimum $E[i,j]$ du coût d'un alignement où le symbole "-" n'est pas inséré à la fin du mot X_i, mais est inséré à la fin du mot Y_j ;
2. optimum $F[i,j]$ du coût d'un alignement où le symbole "-" est inséré à la fin du mot X_i, mais n'est pas inséré à la fin du mot Y_j ;
3. optimum $G[i,j]$ du coût d'un alignement où le symbole "-" n'est inséré ni à la fin du mot X_i, ni à la fin du mot Y_j ;

Le lemme qui suit permet l'écriture d'un algorithme qui procède par récurrence. On suppose que la pénalité prend complètement en compte le rôle des alignements : en d'autres termes $\sigma(a,-) = \sigma(-,a) = 0$.

Lemme 9 *Soient* $i,j > 0$.
$V[0,0] = G[0,0] = 0$,
$V[i,0] = E[i,0] = W_g + iW_s$,
$V[0,j] = F[0,j] = W_g + jW_s$.
De plus,
i) $V[i,j] = \min\{E[i,j], F[i,j], G[i,j]\}$.
ii) $E[i,j] = \min\{E[i,j-1] + W_s, V[i,j-1] + W_s + W_g\}$.
iii) $F[i,j] = \min\{F[i-1,j] + W_s, V[i-1,j] + W_s + W_g\}$.
iv) $G[i,j] = V[i-1,j-1] + \sigma(s_i, t_j)$.

Preuve : La preuve est une simple étude de cas. Par exmple, pour *ii)*, on observe que l'optimum du coût d'un alignement où le symbole "-" n'est pas inséré à la fin du mot X_i, mais est inséré à la fin du mot Y_j peut provenir d'un alignement optimum de X_{i-1} et Y_j du même type, avec un coût supplémentaire de W_s, ou d'un alignement optimum de l'un des autres types, avec un coût supplémentaire de $W_g + W_s$. Ceci conduit à la formule

$$E[i,j] = \min\{E[i,j-1] + W_s, F[i,j-1] + W_s + W_g, G[i,j-1] + W_s + W_g\}.$$

La formule plus simple du lemme est obtenue en observant que le minimum est inférieur à $E[i,j-1] + W_s + W_g$ et donc qu'on ne change pas sa valeur en le remplaçant par

$$\min\{E[i,j-1] + W_s, V[i,j-1] + W_s + W_g\}.$$

3.1.5 Gestion de l'espace

Tous les algorithmes décrits précédemment ont une complexité en temps et en espace quadratique en $O(mn)$. Si une telle complexité n'est pas critique s'agissant du temps, elle peut l'être, notamment dans les applications bio-informatiques, s'agissant de l'espace.

On présente dans cette section un algorithme récursif qui opére en espace linéaire. On se place dans le contexte de l'alignement de deux suites développé dans la section 3.1.3, mais la méthode s'applique *mutatis mutandis* aux autres algorithmes. L'idée est de calculer l'optimum (ici un minimum) simultanément en tronquant les suites au début ou à la fin, c'est à dire - dans le second cas - en lisant les suites à l'envers.

Lemme 10 *Soient $V[i,j]$, $V'[i,j]$ les valeurs optimales de l'algorithme de programmation dynamique tronquant respectivement les suites à la fin et au début. On a :*

$$V[m,n] = \min_{0 \leq k \leq n} \{c[\lceil m/2 \rceil, k] + c'[\lfloor m/2 \rfloor, n-k].$$

Preuve : L'inégalité $\leq$ est claire : à partir d'un alignement de $X_{\lceil n/2 \rceil}$ et Y_k et d'un alignement de $X^{\lfloor n/2 \rfloor}$ et Y^{n-k}, on forme par concaténation un alignement de S et T (où S^i désigne le mot formé en ne conservant que les i derniers symboles de S). Pour établir l'inégalité inverse, on considère un alignement optimal et on choisit k^* de manière à ce que cet alignement mette en regard $X_{\lceil n/2 \rceil}$ et Y_{k^*}.

On observe ensuite que, pour u donné, le calcul d'un vecteur formé des scores optimaux d'un alignement de S_u et T_v pour v allant de 0 à m, ne nécessite qu'un espace $O(n)$. En effet, lors du "remplissage" dynamique, ligne par ligne du tableau des valeurs $V[i,j]$, il est seulement nécessaire de conserver en mémoire deux lignes consécutives du tableau. Bien évidemment, on perd ainsi la possibilité de retourner un alignement optimal. On peut toutefois, compte tenu du lemme, calculer en espace $O(n)$ l'entier k^* qui réalise le minimum du second membre de la formule donnant $V[m,n]$.

L'algorithme récursif $Rec - Opt - align(X,Y)$ retourne un alignement optimal de X et Y, en opérant en espace linéaire comme suit :

1. Calculer en espace $O(n)$ l'entier k^* qui réalise le minimum du second membre de la formule donnant $V[m,n]$; sauvegarder le résultat et libérer la mémoire.
2. Appeler $Rec - Opt - align(X_{\lceil m/2 \rceil}, Y_{k^*})$; sauvegarder le résultat et libérer la mémoire.

3. Appeler $Rec - Opt - align(X^{\lfloor m/2 \rfloor}, Y^{n-k^*})$; sauvegarder le résultat et libérer la mémoire.
4. Concaténer les deux alignements obtenus aux étapes 2 et 3 ; libérer la mémoire et retourner le résultat.

3.2 Alignement de plusieurs suites

Etant données plusieurs suites et "-" un symbole qui n'apparaît dans aucune de ces suites, un *alignement* de ces suites est obtenu en insérant dans chacune des symboles "-" supplémentaires de manière à ce que les suites produites soient toutes de même longueur. Comme dans le cas de deux suites on peut attribuer un coût à un alignement multiple à partir d'une fonction *score* $\sigma(a, b)$. On se restreindra au cas où σ est symétrique ($\sigma(a, b) = \sigma(b, a)$), vérifie l'inégalité triangulaire ($\sigma(a, c) \leq \sigma(a, b) + \sigma(b, c)$) et est telle que $\sigma(a, a) = 0$. Egalement, on supposera qu'il n'y a pas de pénalité pour les espacements.

Dans cette section, on s'écarte des notations utilisées auparavant en notant S_i la i-ème suite. En prenant les suites deux à deux, on peut considérer le coût de l'alignement correspondant soit $D(S_i, S_j)$. Le coût SP (sum of pairs) est la somme

$$\sum_{i<j} D(S_i, S_j).$$

On peut également proposer une autre mesure - nommée erreur de consensus ou EC- calculée à partir d'un alignement en déterminant une suite S représentant le "consensus" qui minimise :

$$\sum_{i} D(S, S_i).$$

Les problèmes de décision associés à la recherche d'optimum pour SP ou EC sont NP-complets. Il est donc naturel de s'intéresser à des algorithmes approchés.

3.2.1 L'algorithme Center-Star

Cet algorithme calcule pour chaque indice t, la somme

$$\sum_{i} D(S_i, S_t),$$

et choisit un indice t^* minimisant cette somme. Quitte à renuméroter les indices, on peut supposer que $t^* = 1$. A partir de S_1, l'algorithme construit

par récurrence un alignement des u premieres suites. Au moment d'ajouter S_u, l'algorithme considère l'alignement entre S_1 et S_u produisant la valeur minimum $D(S_u, S_1)$, et insére des "-" de façon à recaler les positions de S_1 dans cet alignement et dans l'alignement des $u-1$ premières suites obtenu récursivement.

La complexité de l'algorithme est en $O(k^2n^2)$ où k est le nombre de suites et n la longueur maximale. Le terme n^2 provient de l'algorithme qui calcule $D(S_i, S_t)$ par programmation dynamique. Le terme k^2 est lié à la taille de l'ensemble dans lequel varients les paires $\{i, j\}$.

Théorème 5 *La performance pour SP de l'algorithme Center-Star est dans un rapport* $\frac{2(k-1)}{k} < 2$ *de l'optimum.*

Preuve : Soit μ le coût de l'alignement optimal pour SP. On considère l'alignement optimal et on note $d^*(i, j)$ le coût induit de l'alignement de S_i et S_j. De même, on note $d(i, j)$ le coût induit de l'alignement de S_i et S_j résultant de la solution trouvée par l'algorithme. Par optimalité, on a :

$$\mu = \sum_{i<j} d^*(i, j) \leq \sum_{i<j} d(i, j).$$

Par l'inégalité triangulaire, pour i et $j \neq 1$, on a :

$$d(i, j) \leq d(i, 1) + d(1, j).$$

En sommant, on obtient

$$2\sum_{i<j} d(i, j) = \sum_{i\neq j} d(i, j) \leq 2(k-1)\sum_{i>1} d(i, 1).$$

Compte tenu de la stratégie de l'algorithme, la somme apparaissant dans le dernier terme, soit $\sum_{i>1} d(i, 1)$ est égale à $\sum_{i>1} D(S_i, S_1)$ et est majorée par les sommes analogues $\sum_{i\neq j} D(S_i, S_j)$ pour j fixé. En prenant la moyenne, on trouve pour l'optimum SP un majorant de la forme

$$\frac{(k-1)}{k}\sum_{i\neq j} D(S_i, S_j) = \frac{2(k-1)}{k}\sum_{i<j} D(S_i, S_j).$$

En utilisant la majoration $D(S_i, S_j) \leq d^*(i, j)$, on trouve finalement :

$$\mu \leq \sum_{i<j} d(i, j) \leq \frac{2(k-1)}{k}\mu,$$

ce qui achève la preuve.

3.2.2 Alignement avec consensus

Etant données plusieurs suites, on peut, reprenant les notations de la section précédente, chercher à minimiser la somme $\sum_i D(S, S_i)$. On appelle suite de Steiner une suite S^* réalisant l'optimum. A partir d'une telle suite S^*, il est facile, en s'inspirant de l'algorithme Center-Star de construire un *alignement avec consensus* de ces suites en insérant dans chacune et dans S^* des symboles "-" supplémentaires de manière à ce que les suites produites $\overline{S_i}$ soient toutes de même longueur et qu'elles réalisent avec une suite $\overline{S^*}$ obtenue à partir de S^* un alignement de coût $D(S^*, S_i)$.

Il est difficile de calculer S^* efficacement. Toutefois, on a :

Théorème 6 *La suite S produite par l'algorithme Center-Star est telle que la somme $\sum_i D(S, S_i)$ est dans un rapport $\frac{2k-1}{k} < 2$ de la valeur optimale obtenue pour une suite de Steiner S^*.*

Preuve : Soit μ^* la valeur de $\sum_i D(S^*, S_i)$, S_t la suite réalisant le minimum de $D(S^*, S_i)$ et S_c la suite retournée par l'algorithme Center-Star. Compte tenu de la stratégie de cet algorithme, on a :

$$\sum_{i \neq c} D(S_c, S_i) \leq \sum_{i \neq t} D(S_t, S_i).$$

En utilisant l'inégalité triangulaire, le second terme peut être majoré par

$$\sum_{i \neq t} D(S_t, S^*) + D(S^*, S_i) \leq (k-1) D(S_t, S^*) + \mu^*.$$

Par définition de S_t, on a $D(S_i, S^*) \geq D(S_t, S^*)$ pour tout i et donc, en sommant $\mu^* \geq k D(S_t, S^*)$. En reportant dans l'inégalité ci-dessus, il vient :

$$\sum_{i \neq c} D(S_c, S_i) \leq \mu^* (1 + \frac{k-1}{k}),$$

ce qui, compte tenu de l'inégalité $\mu^* \leq \sum_{i \neq c} D(S_c, S_i)$, conduit au résultat attendu.

Bibliographie

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*, 2nd ed. Cambridge, Massachusetts : M.I.T. Press, 2001.

[2] R. Shamir Algorithms for Molecular Biology : Course Archive. `http ://www.math.tau.ac.il/ rshamir/algmb.html`

Cours 4 (26 octobre)

Arbres

— Structures de données fusionnables

On a vu dans le cours différentes structures d'arbre permettant de gérer un ensemble de données et d'opérer des opérations de recherche, d'insertion et de suppression d'un élément repéré par une clé, tout en maintenant le caractère équilibré des arbres et en assurant ainsi que chaque opération a une complexité $O(\log n)$, où n est le nombres de nœuds de l'arbre.

Ces structures en revanche ne permettent pas facilement la fusion de deux ensembles de données, du moins pas avec la même complexité. D'autres structures y parviennent et l'objet de ce chapitre est d'en présenter le principe.

4.1 Arbres binomiaux et tas binomiaux

4.1.1 Arbres binomiaux

On définit un arbre binomial de manière récursive : un arbre à un seul sommet B_0 est un arbre binomial et on forme l'arbre binomial B_n à partir de deux copies de l'arbre binomial B_{n-1} en ajoutant un unique lien faisant de la racine du premier le fils aîné de la racine du second. On pourra se reporter à la page 25 des transparents du cours ou à l'ouvrage [3] pour une figure.

Le lemme suivant résume les propriétés des arbres binomiaux, établies par des récurrences simples laissées au lecteur.

Lemme 11 *L'arbre binomial B_n*
i) a exactement 2^n nœuds,
ii) a exactement hauteur n,
iii) a exactement $\binom{n}{i}$ nœuds de profondeur $i < n$, et en particulier exac-

tement n fils de la racine,
iv) est obtenu à partir de la suite $B_{n-1}, \cdots, B_0$, prise dans cet ordre, en faisant de la suite des racines de ces arbres, prise dans le même ordre, la suite des fils d'un nouveau nœud, lequel devient la racine de l'arbre obtenu.

4.1.2 Tas binomiaux

Un *tas binomial* est une structure de données formée d'un suite d'arbres binomiaux dont les racines constituent une liste chaînée. Chaque nœud est de plus étiquété de telle manière que, dans chaque arbre binomial l'étiquette d'un père est inférieure à celle de son fils. Enfin, pour chaque entier i, il n'y a qu'un seul arbre binomial B_i au plus.

On note que si I est l'ensemble des indices des arbres binomiaux qui apparaissent dans le tas binomial, on a, en notant n le nombre de nœuds du tas,

$$n = \sum_i 2^i.$$

On reconnaît la décomposation de n en base 2 et I est donc l'ensemble des indices où un 1 apparaît dans la décomposition binaire de n. Ceci montre que le nombre d'éléments de I est $< \lceil \log n \rceil$ et le lemme 11 montre qu'il en est de même pour la hauteur de chacun des arbres du tas.

4.1.3 Fusion de deux tas binomiaux

Pour fusionner deux tas binomiaux, donnés par un pointeur vers la liste chaînée des racines, on fusionne les deux listes en ordre croissant des indices des arbres binomiaux. L'indice est exactement le nombre de fils de chaque nœud racine, c'est à dire le *degré* de la racine, ce qui permet de le calculer, à moins qu'on ne préfère maintenir ce degré dans un champ attaché à chaque nœud.

Le problème c'est que l'on obtient pas ainsi en général un arbre binomial. En effet on peut avoir deux arbres de même indice i, correspondant au cas où i apparaît à ma fois dans la décomposition binaire des entiers n et m représentant respectivement le nombre d'éléments de chacun des tas à fusionner.

Pour résoudre ce problème, on s'inspire de l'addition binaire avec "retenue". On parcourt la suite des racines des arbres binomiaux de la liste fusionnée et on réduit le nombre des arbres par les règles suivantes :

1. un unique arbre d'indice i reste inchangé,

2. deux arbres successifs d'indice i sont groupés en un arbre d'indice $i+1$ dont la racine correspond à celui des deux arbres dont l'étiquette est minimale,
3. trois arbres d'indice i sont remplacés par deux arbres, l'un d'indice i et l'autre d'indice $i+1$, en laissant le premier inchangé et en appliquant aux deux derniers la même méthode que pour la règle 2,

L'aspect algorithmique de cet algorithme de réduction est détaillé dans [3]. Le principe est de maintenir un pointeur qui avance dans la liste. Bien entendu, un arbre d'indice $i+1$ créé par la règle 2 ou 3 peut être ensuite regroupé avec un autre pour créer un arbre d'indice $i+2$. Toutefois, il est facile de voir qu'aucun autre cas que ceux mentionnés ci-dessus n'apparaît. Comme on ne fait que parcourir la liste obtenue par fusion des deux listes chaînées initiales, en réaffectant un nombre borné de pointeurs pour chaque nœud, on a une complexité en $O(n)$ où n est la taille du plus grand des deux tas.

4.1.4 Autres opérations

Les autres opérations se ramènent à la fusion : l'insertion se fait par fusion avec un tas réduit à un nœud unique ; la recherche du minimum correspond au parcours de la liste chaînée des racines ; la suppression de ce minimum passe par la suppression du nœud correspondant dans la liste chaînée des racines, la création d'un second tas à partir des fils de la racine, lesquels sont naturellement chaînés par le pointeur d'un nœud à son frère, et la fusion des deux tas ainsi obtenus.

Il reste la suppression d'un élément non nécessairement minimal, accédé par un pointeur : l'étiquette de ce nœud est changée en $-\infty$ et la propriété de tas est restaurée par un algorithme standard, analogue à ceux présentés dans la leçon numéro 2 sur les tris. L'élément - qui est devenu d'étiquette minimale - est alors supprimé. Tous les algorithmes décrits précédemment ont une complexité $O(n)$.

4.2 Tas de Fibonacci

Un *tas de Fibonacci* est une structure de données formée d'un ensemble d'arbres dont les racines constituent une liste cyclique doublement chaînée. Les fils de chaque nœud sont de même organisés en une liste cyclique doublement chaînée. Chaque nœud est également étiqueté de telle manière que, dans chaque arbre l'étiquette d'un père soit inférieure à celle de son fils. Enfin, dans un tas de Fibonacci, certains éléments sont marqués, d'autres non. On accède à la structure par un pointeur vers l'élément minimal.

Toutefois, toute structure ayant les propriétés décrites ci-dessous n'est pas nécessairement un arbre de Fibonacci. En effet, il n'y a pas à proprement parler de définition "synthétique" et ne sont des tas de Fibonacci que les structures qui peuvent être récursivement créées à partir d'un tas vide, par les opérations d'insertion, de suppression du minimum, de décrémentation d'une clé et de fusion.

La complexité des opérations mentionnées ci-dessus se mesure par référence à un potentiel, comme expliqué dans la leçon 1 : il s'agit donc de complexité amortie. Le potentiel est défini par la quantité $t + 2m$, où t est le nombre total d'arbres dans le tas et m le nombre de nœuds marqués.

4.2.1 Les opérations simples

Les opérations d'insertion, de recherche du minimum sont de complexité amortie $O(1)$.

Insertion L'insertion d'un nouvel élément se fait sur la liste doublement chaînée par exemple à gauche de l'élément minimal. Le pointeur vers cet élément minimal est réaffecté si nécessaire. Le nouvel élément est non marqué.

Recherche du minimum La recherche du minimum se fait par "lecture directe" en déréférençant le pointeur vers cet élément minimal.

Fusion La fusion se fait en fusionnant les deux listes doublement chaînées des racines. Les potentiels s'ajoutant, le coût amorti est bien en $O(1)$.

4.2.2 L'opération d'extraction du minimum

L'opération d'extraction du minimum commence par insérer les fils du nœud supprimé dans la liste doublement chaînée des racines. Le pointeur vers l'élément minimal est réaffecté si nécessaire.

Cette opération est suivie d'une opération de "consolidation" destinée à assurer que toutes les racines ont même degré (c'est à dire même nombre de fils). Cette opération est représentée en page 35 des transparents de cours. Elle est analogue à l'opération de réduction du tas obtenu après fusion de deux tas binomiaux. On recherche deux éléments de même degré dans la liste doublement chaînée des racines et, à chaque fois qu'on en trouve deux, on les regroupe exactement comme on a regroupé dans la section 4.1.3 deux arbres binomiaux, ce qui a pour effet d'augmenter d'une unité le degré de la racine.

On supprime également la marque de celle des racines placée "sous" l'autre, si elle est marquée.

On organise l'algorithme de consolidation à partir d'un tableau de pointeurs A indicé par tous les degrés possibles en parcourant la liste chaînée des racines en en regroupant les arbres à l'aide des règles suivantes :

1. quand on rencontre un arbre dont la racine est de degré d avec $A[d] = NIL$, on affecte $A[d]$ à cet arbre ;
2. quand on rencontre un arbre dont la racine est de degré d avec $A[d] \neq NIL$, on regroupe $A[d]$ et cet arbre, on libère le pointeur $A[d]$ et on "propage les retenues" en traitant par la règle 2 l'arbre de degré $d+1$ ainsi obtenu.

Le coût de la première étape est un $O(D)$ où D est le degré maximal à considérer. Le coût de la seconde est un $O(D+t)$, puisque chaque regroupement diminue d'une unité le nombre de racines dans la liste des racines, ce en partant de $t+D-1$. Le potentiel quant à lui passe de $t+2m$ à au plus $D+2m$: en effet, à la fin de la consolidation, il n'y a plus que D éléments au plus dans la liste des racines. Le coût amorti est donc un $O(D)$.

4.2.3 L'opération de décrémentation d'une clé

L'opération de décrémentation de la clé d'un nœud x qui n'est pas dans la liste des racines, commence par placer ce nœud dans la liste des racines, avec à sa suite son sous arbre. La marque du nœud ainsi déplacé est supprimée si elle était présente.

Cette opération est suivie d'une opération en "cascade" consistant à déplacer aussi dans la liste des racines le père du nœud x, son grand-père etc., tant qu'ils sont marqués. La marque du nœud ainsi déplacé est supprimée si elle était présente. Lorsqu'un ascendant non marqué est rencontré, la cascade s'arrête et l'ascendant en question est marqué.

Une fois la cascade terminée, le pointeur vers l'élément minimal est réaffecté si nécessaire.

Le coût total de ces trois étapes est un $O(c)$ où c est le nombre d'ascendants touchés par la cascade. Le potentiel, qui est initialement $t+2m$ voit la valeur de t augmenter de c et la valeur de m diminuer de c. Le bilan est une diminution de c puisque m compte double, ce qui compense bien le coût. Finalement, en ajustant les constantes, on trouve un coût amorti $O(1)$.

4.2.4 Complexité amortie des opérations

On a vu que toutes les opérations avaint une complexité amortie $O(1)$ sauf la consolidation, qui a une complexité amortie $O(D)$. Reste donc à borner D.

Théorème 7 *Soit x un nœud de degré k d'un tas de Fibonacci. Alors la taille du sous arbre issu de x est minorée par la $k+1$-ième valeur F_{k+1} de la suite de Fibonacci.*

On note qu'il résulte de ce théorème que pour un tas qui a n éléments, tous les nœuds ont au plus degré k, où k est le plus grand entier tel que $F_{k+1} \leq n$. Comme F_{k+1} est minoré par φ^{k-1}, φ représentant le nombre dor, on en conclut que D est bien un $O(\log n)$.

Le théorème est quant à lui conséquence du lemme suivant.

Lemme 12 *Soit x un nœud de degré k d'un tas de Fibonacci et soit $y_1, \cdots, y_k$ la suite de ses fils ordonnée suivant le moment où ils ont été liés à x lors de l'exécution des algorithmes opérant sur les tas de Fibonacci ; alors, le degré de y_i est minoré par $i-2$.*

Preuve du lemme Au moment où y_i est lié à x, x a pour degré $i-1$ au moins, puisque $y_1, \cdots, y_{i-1}$ ont été liés auparavant. De plus, y_i a même degré que x. Donc y_i a degré $\geq i-1$. Pour conclure, il suffit d'observer qu'un nœud ne perd des fils que lors de la "cascade" qui commence par le déplacement d'un élément consécutif à une décrémentation de sa clé et se poursuit avec le déplacement de son père, son grand père etc. Un père qui perd un fils est marqué. S'il devait perdre un autre fils, il serait placé dans la liste des racines. Comme y_i est fils de x, il n'est pas dans ce cas et a donc au moins $i-2$ fils.

On revient maintenant à la preuve du théorème. Soit s_i la taille maximum du sous-arbre issu d'un nœud de degré i d'un tas de Fibonacci. On a $s_0 = 1$. On a de plus, d'après le lemme précédent,

$$s_k \geq 2 + \sum_{i=2}^{k} s_{i-2}.$$

En utilisant l'hypothèse de récurrence, il vient

$$s_k \geq 2 + \sum_{i=1}^{k-1} F_i = 1 + \sum_{i=0}^{k-1} F_i$$

Il est bien connu (et facile à vérifier si on ne le sait pas) que $1 + \sum_{i=0}^{k-1}$ vaut F_{k+1}, ce qui achève la preuve du lemme.

Bibliographie

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*, 2nd ed. Cambridge, Massachusetts : M.I.T. Press, 2001.

[2] J. Vuillemin. A data structure for manipulating priority queues *Communications of the ACM* Volume 21 (4), 309–315, 1978.

[3] M. L. Fredman and R. E. Tarjan . Fibonacci heaps and their uses in improved network optimization algorithms, *Journal of the ACM*, Volume 34 (3) 596–615 , 1987.

Cours 5 (2 novembre)

Graphes

— Graphes d'expansion

Dans cette leçon, on établit un lien entre les propriétés d'expansion d'un graphe et les valeurs propres de sa matrice d'adjacence. On étudie aussi les applications algorithmiques de graphes ayant de bonnes propriétés d'expansion.

5.1 Graphes d'expansion et valeurs propres

5.1.1 Définitions

Les graphes étudiés dans cette section sont non orientés. Etant donné un ensemble de sommets W d'un graphe G, on note $|W$ le nombre d'éléments de W et on appelle voisins de W l'ensemble des sommets liés à un élément de W par une arête de G.

Un graphe à n sommets admet la *constante d'expansion* c, si pour tout ensemble de sommets W tel que $|W| \leq n/2$, l'ensemble des voisins de W a au moins $c|W|$ éléments.

5.1.2 Lien avec la seconde valeur propre

On dit qu'un graphe est d-régulier si chaque sommet a exactement d voisins. La théorie qui suit présente, dans le cadre simplifié des graphes réguliers, le lien profond qui unit les propriétés d'expansion d'un graphe aux valeurs propres de sa matrice d'adjacence A. On note que la matrice A est symétrique et qu'elle admet d comme valeur propre, puisque la matrice $A - dI$ a la somme de toutes ses colonnes nulle et est donc non inversible. C'est la

plus grande valeur propre : en effet, pour tout $\lambda > d$, la matrice $A - \lambda I$ est à diagonale dominante, c'est à dire que, sur chaque ligne, la somme des coefficients non diagonaux est majorée en valeur absolue par le coefficient diagonal. Il est bien connu - et le lecteur peut le démontrer à titre d'exercice - qu'une telle matrice est non inversible. Il n'est pas exclu que d soit valeur propre multiple, mais dans ce cas, il n'est pas difficile de voir - par la même méthode que pour l'exercice suggéré ci-dessus - que G se décompose en un certain nombre de sous-graphes sur lesquels d est valeur propre simple, chaque sommet d'un sous-graphe n'étant lié qu'à des sommets du même sous-graphe. Dans ce qui suit, pour simplifier, on se restreindra implicitement à des graphes d-réguliers, n'admettant pas de décompostion en sous-graphes, c'est à dire pour lesquels d est valeur propre simple.

Théorème 8 *Soit G un graphe d-régulier et λ la seconde plus grande valeur propre de la matrice d'adjacence A, alors G admet la constante d'expansion $\frac{d-\lambda}{2d}$.*

La preuve de ce théorème repose sur le lemme suivant.

Lemme 13 *Soit X le complément de W, b et c le nombre d'éléments de W et X respectivement ; alors, le nombre $e(W, X)$ d'arêtes joignant un sommet de W à un sommet de X est minoré par $\frac{(d-\lambda)|W||X|}{n}$.*

Pour déduire le théorème du lemme, il suffit d'observer que chaque arête allant d'un sommet de W à un sommet de X a pour extrémité un voisin de W. Comme un tel voisin peut être atteint par d arêtes, on en déduit que le nombre de voisins est minoré par $|W|\frac{(d-\lambda)|X|}{dn}$. Pour $|W| \leq n$, on a $\frac{|X|}{n} \geq \frac{1}{2}$, d'où le résultat.

Preuve du lemme On considère un vecteur x à n coordonnées indexées par les sommets de G. Les coordonnées x_u de x dont l'indice u est dans W sont égales à $-c$ et celles x_v dont l'indice v est dans X sont égales à b. On observe que x est orthogonal au vecteur propre $\mathbf{1}$ correspondant à la valeur propre d, dont toutes les coordonnées sont égales à 1.

Il est bien connu qu'une matrice symétrique réelle est diagonalisable et que ses sous espaces propres sont deux à deux orthogonaux. En appliquant cette proprété à la matrice $d.I - A$, dont toutes les valeurs propres sont $\geq d - \lambda$ et en considérant le sous-espace propre orthogonal au vecteur propre $\mathbf{1}$, auquel x appartient, il vient

$$((d.I - A)x, x) \geq (d - \lambda)||x||^2 = (d - \lambda)(bc^2 + cb^2).$$

Par ailleurs, le produit scalaire $((d.I-A)x,x)$ est donné par la formule matricielle habituelle ${}^t(x)(d.I-A)(x)$, et on voit facilement que ce produit vaut :

$$\sum_u dx_u^2 - 2\sum_{A[u,v]=1} x_u x_v,$$

soit en notant E l'ensemble des arêtes de G

$$\sum_{\{u,v\}\in E} (x_u - x_v)^2,$$

soit encore $e(W,X)(b+c)^2$.

Il vient

$$e(W,X) \geq (d-\lambda)\frac{bc^2+cb^2}{(b+c)^2} = (d-\lambda)\frac{bc}{n},$$

qui est l'inégalité à démontrer.

Une réciproque du théorème ci-dessus a été établie par Alon [1] : elle affirme que, pour un graphe d qui admet c comme constante d'expansion, la seconde valeur propre est $\leq d - \frac{c^2}{4+2c^2}$.

5.2 Applications algorithmiques

5.2.1 Marche aléatoire dans un graphe

Soit G un graphe d régulier. Une *marche aléatoire* de longueur t dans le graphe G est obtenue

- en choisissant aléatoirement une origine u_0,
- en choisissant aléatoirement, étant donnée la i-ème valeur u_i, $i < t$, un sommet u_{i+1} voisin de u_i.

Théorème 9 *Soit G un graphe d-régulier et λ la seconde valeur propre en valeur absolue de la matrice d'adjacence ; soit W un ensemble de sommets de G ; on note c la quantité $\frac{|W|}{n}$ et on suppose que l'inégalité*

$$((1-c)d^2+\lambda^2)^{1/2} \leq \frac{d}{2^{1/4}}$$

est satisfaite ; alors, la probabilité qu'une marche aléatoire de longueur t n'ait aucun élément dans W est majorée par $2^{-t/4}$.

Ce théorème est conséquence du lemme suivant.

Lemme 14 *Le nombre de marches aléatoires de longueur t qui évitent W est majoré par $n(1-c)^{1/2}((1-c)d^2+\lambda^2)^{t/2}$.*

Le nombre total de marches aléatoires de longueur t étant $n(d)^t$ la probabilité est, d'après le lemme, majorée par

$$\frac{n(1-c)^{1/2}((1-c)d^2+\lambda^2)^{t/2}}{n(d)^t},$$

ce qui, d'après l'hypothèse du théorème, est bien borné par $2^{-t/4}$.

Preuve du lemme : On considère la projection P qui à un vecteur x à n coordonnées indexées par les sommets de G associe le vecteur obtenu en annulant toutes les coordonnées qui n'appartiennent pas à W. Le nombre n_u de chemins de longueur i qui évitent W et dont l'extrémité est un sommet donné u s'interprète, avec les notations de la section 5.1.2, comme la coordonnée d'indice u du vecteur $(PA)^iP\mathbf{1}$. Pour le voir, on procède par récurrence. Le cas $i=0$ est clair. Pour passer de i à $i+1$, on observe que, par hypothèse de récurrence, le vecteur y dont les coordonnées sont n_u, est donné par $y=(PA)^iP\mathbf{1}$. Chaque marche aléatoire de longueur $i+1$ évitant W est définie en prolongeant d'un dernier choix une marche de longueur i. Celles qui atteignent un sommet donné v sont donc en nombre

$$\sum_{\{u,v\}\in E} n_u = \sum_{A[u,v]=1} n_u = \sum_u A[u,v]n_u.$$

On reconnaît là la v-ième coordonnée de $A(PA)^iP\mathbf{1}$, et on multiplie par P pour annuler les coordonnées qui n'appartiennent pas à W, afin de tenir compte de ce que le dernier choix doit lui aussi éviter W.

Pour évaluer n_u, on procède un peu comme dans la preuve du lemme 13 et on établit que l'application linéaire PA a une norme ℓ_2 majorée par $((1-c)d^2+\lambda^2)^{1/2}$. Pour cela, on étudie séparément l'action de PA sur le sous-espace propre engendré par le vecteur propre $\mathbf{1}$ correspondant à la valeur propre d et sur son orthogonal. Sur le sous espace engendré par $\mathbf{1}$, la norme ℓ_2 est $d\sqrt{1-c}$, comme on le voit par un calcul direct. Sur l'orthogonal, les valeurs propres de A sont majorées en valeur absolue par $|\lambda|$ et donc - comme on le voit immédiatement en diagonalisant la matrice- la norme de A est majorée par $|\lambda|$. Comme celle de P est majorée par 1, il vient $||PA|| \leq |\lambda|$. En sommant les deux applications, on trouve bien, par l'inégalité de Schwarz, une norme majorée par $((1-c)d^2+\lambda^2)^{1/2}$.

Il résulte du calcul qui précède que la norme ℓ_2 de $(PA)^tP\mathbf{1}$ est au plus $((1-c)d^2+\lambda^2))^{t/2}||P\mathbf{1}||$. En fait, c'est la norme ℓ_1 qu'il faut majorer mais l'inégalité de Schwarz donne pour cette denière un majorant de la forme

$$\sqrt{n}((1-c)d^2+\lambda^2))^{t/2}\sqrt{(1-c)n},$$

ce qui achève la preuve du lemme.

5.2.2 Constructions explicites et applications

Les constructions explicites connues de graphes d'expansion, dues à Lubotzky, Phillips, Sarnak [4] et Margulis [5], sont trop difficiles pour être exposées ici. On se contentera de mentionner qu'il est possible de trouver des familles explicites de graphes d'expansion de degré d avec $|\lambda| \leq 2\sqrt{d-1}$ pour toute valeur de d égale à $p+1$, p étant un nombre premier congru à 1 modulo 4. Par exemple, pour $d = 30$, on obtient $|\lambda| \leq 2\sqrt{29}$ et donc

$$\left(\frac{1}{2}d^2 + |\lambda|^2\right)^{1/2} \simeq 23.79 \leq \frac{d}{2^{1/4}} \simeq 25.23.$$

Les conditions du théorème sont donc satisfaites dès que $|W|$ est au moins $n/2$. Il n'est pas possible de choisir n quelconque mais cela est peu restrictif car, pour une fonction ε tendant vers zéro, il existe un n dans tout intervalle $[m, (1+\varepsilon)m]$.

A partir des paramètres explicités ci-dessus, il est donc possible d'exécuter un algorithme probabiliste dépendant du choix d'une unique variable aléatoire suivant une loi uniforme sur les entiers $1, \cdots m$, et dont la probabilité de succès est - pour fixer les idées - $1/2$, en utilisant une marche aléatoire de longueur t. La probabilité d'échec est alors bornée par $2^{-t/4}$. On peut aussi répéter $t/4$ fois l'algorithme en faisant des tirages indépendants pour obtenir la même probabilité d'échec. La difference est le nombre de bits d'aléas dans chaque cas : il est de $O(t \log m)$ dans le second cas et de $O(\log m + t)$ dans le premier, ce qui est bien moins asymptotiquement. Même s'il n'a pas de conséquence pratique, il s'agit là d'un résultat algorithmique profond.

Bibliographie

[1] N. Alon. Eigenvalues and expanders, *Combinatorica* 6(1986), 83–96.

[2] N. Alon, and J. H. Spencer. *The Probabilistic Method*, John Wiley & Sons, Inc., New York, 1992.

[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*, 2nd ed. Cambridge, Massachusetts : M.I.T. Press, 2001.

[4] A. Lubotzky, R. Phillips, and P. Sarnak. Ramanujan graphs, *Combinatorica*, 8 (1988), 261–277.

[5] G. A. Margulis. Explicit group-theoretic constructions of combinatorial schemes and their applications in the construction of expanders and concentrators. *Problemy Peredachi Informatsii*, 24(1), 51–60, 1988.

Cours 6 (9 novembre)

Flots

— Méthode de flot bloquant

Dans cette leçon, on cherche à améliorer les algorithmes de flot maximum présentés dans le cours de base et notamment l'algorithme d'Edmonds-Karp, basé sur la recherche d'un chemin de longueur minimal dans le graphe résiduel. Cette amélioration passe par la notion de flot bloquant. On applique ensuite les résultats obtenus à la recherche de couplages dans les graphes bipartis.

6.1 Flots bloquants

6.1.1 Définitions

On conserve les notations du cours de base, que l'on rappelle. Soit G un graphe orienté. On note E l'ensemble de ses arêtes et V l'ensemble de ses sommets. On désigne par m et n respectivement le nombre de ses arêtes et le nombre de ses sommets. On suppose toujours $m \geq n$. Dans toute la suite, on distingue deux sommets s et t de G, qu'on nomme respectivement *source* et *but* et on munit G d'une fonction *capacité* $C(u, v)$ qui prend comme arguments deux sommets u et v liés par une arête et retourne une valeur **entière** strictement positive. Par convention, on étend C à tous les couples de sommets en posant $C(u, v) = 0$ si (u, v) n'est pas une arête.

On rappelle qu'un flot sur G est une fonction f qui prend comme argument deux sommets u et v et retourne une valeur **entière**, positive ou négative, qui satisfait les trois propriétés suivantes :
antisymétrie $f(u, v) = -f(v, u)$ *quels que soient* u *et* v.

conservation *Pour tout sommet u autre que s et t, on a*

$$\Delta_f(u) = \sum_v f(u,v) = 0$$

contrainte de capacité $f(u,v) \leq C(u,v)$ *quels que soient u et v.*

La *valeur* $|f|$ du flot f est la quantité $\sum_v f(s,v)$, dont il est facile de vérfier qu'elle vaut aussi $\sum_u f(v,t)$. On dit qu'un couple de sommets (u,v) est saturé si $f(u,v) = C(u,v)$. Le graphe résiduel de f admet V comme ensemble de sommets et pour arêtes les couples de sommets (u,v) non saturés par f. On le notera R_f dans toute la suite en omettant l'indice f s'il n'y a pas ambiguïté. Le graphe résiduel R_f est muni d'une capacité résiduelle définie par $C(u,v) - f(u,v)$. Un *chemin améliorant* est un chemin de s à t dans le graphe résiduel. Soit $d_f(s,u)$ ou simplement $d(s,u)$ la fonction distance du sommet s au sommet u dans le graphe résiduel, chaque arête comptant pour une unité. Le *graphe de niveau* L_f admet comme arêtes celles des arêtes (u,v) de R_f qui vérifient

$$d(s,u) + 1 = d(s,v) < \infty$$

Ce graphe est sans cycle et il est immédiat de vérifier que tout chemin améliorant n'emprunte que des arêtes de L_f. Il est tout aussi immédiat d'observer qu'un simple parcours en largeur du graphe R_f permet le calcul de L_f, avec une complexité $O(m)$.

Un flot f sur un graphe G est dit *bloquant* si tout chemin de s à t (dans G) a au moins une de ses arêtes saturée. Le principe des algorithmes de flot bloquant est de calculer un flot bloquant f' du graphe L_f pour la capacité résiduelle et d'augmenter f en $f + f'$. On converge ainsi plus rapidement vers le flot maximal.

6.1.2 Algorithme de calcul d'un flot bloquant

L'algorithme de calcul d'un flot bloquant opère sur des variables globales qui représentent :

1. un flot f sur G,
2. un graphe H, sur les mêmes sommets que G,
3. un chemin p de G d'origine s représenté sous forme d'une liste chaînée avec accès en temps constant à l'extrémité $last(p)$ de p.

Cet algorithme met en œuvre quatre fonctions.

INITIALISER
$p \leftarrow [s]$;

RECULER
$v \leftarrow last(p)$;
si $v = s$ **alors retourner**(échec) **sinon**
supprimer v de p ;
$u \leftarrow last(p)$;
supprimer (u, v) de H ;
retourner(succès) ;

AVANCER
$v \leftarrow last(p)$;
si v n'a pas d'arête sortante dans H **alors retourner**(échec) **sinon**
calculer une telle arête sortante (v, w) ;
ajouter w à la fin de p ;
retourner(succès) ;

AUGMENTER
$v \leftarrow last(p)$;
calculer Δ minimum de $C(u, v) - f(u, v)$ quand (u, v) décrit les arêtes de p ;
supprimer de H les arêtes où le minimum est atteint ;
pour chaque arête (u, v) de p ajouter Δ à $f(u, v)$ et $-\Delta$ à $f(v, u)$;

L'algorithme lui-même est décrit par :

$H \leftarrow G$; $f \leftarrow 0$;
INITIALISER
test-fin $\leftarrow$ succès ;
tant que (test-fin = succès) **faire**
test-av $\leftarrow$ succès ;
tant que (test-av = succès) **faire** test-av $\leftarrow$ AVANCER ;
si $last(p) \neq t$ **alors** test-fin $\leftarrow$ RECULER **sinon**
AUGMENTER ;
INITIALISER ;

On prouve d'abord que, s'il se termine, l'algorithme calcule un flot bloquant f. Si ce n'est pas le cas, il existe un chemin q de s à t dont aucune arête n'est saturée. Ce chemin q ne peut emprunter uniquement des arêtes du graphe défini par la valeur de H à la fin du programme. En effet, l'algorithme ne se termine que si un appel de la fonction $RECULER$ retourne une valeur d'échec pour la variable $test - fin$, ce qui ne se produit que si elle est appelée avec une valeur de p telle que $last(p) = s$. La fonction $RECULER$ n'est elle-même appelée que si la fonction $AVANCER$ retourne une valeur d'échec pour la variable $test - av$, ce qui dans le contexte implique qu'aucune arête du graphe défini par la valeur finale de H ne sort de s. Les arêtes du

chemin q ne sont donc pas toutes dans H. Soit (u, v) la première arête de q à avoir été supprimée de H. Il y a deux cas à considérer :
premier cas : cette arête a été supprimée par la fonction $AUGMENTER$. Cette fonction a également saturé l'arête (u, v) en augmentant la valeur courante du flot f. D'où une contradiction.
deuxième cas : cette arête a été supprimée par la fonction $RECULER$ à un moment où, par définition de (u, v), l'arête suivante du chemin q, soit (v, w) était encore dans H. Or, la fonction $RECULER$ n'est appelée que si la fonction $AVANCER$ retourne une valeur d'échec pour la variable $test - av$, ce qui dans le contexte implique qu'aucune arête du graphe défini par la valeur courante de H ne sort de v. D'où, à nouveau une contradiction.

On montre ensuite la terminaison : chaque exécution des deux fonctions $RECULER$ ou $AUGMENTER$ supprime au moins une arête ; il y en a donc au plus m. Les exécutions consécutives de $AVANCER$ sont en nombre au plus $n-1$, puisque chacune augmente la longueur du chemin p d'une unité et que, le graphe L_f étant sans cycle, cette longueur ne peut dépasser $n-1$. Le nombre total des appels de fonction est donc en $O(m)$ pour les fonctions $RECULER$ ou $AUGMENTER$ et en $O(mn)$ pour la fonction $AVANCER$.

La complexité est également en $O(nm)$ si on prend soin de représenter les graphes par des structures de données "mixte" composées d'une matrice d'adjacence, d'une liste d'adjacence et de pointeurs indiquant, pour chaque arête (u, v), la position de v dans la liste d'adjacence de u. Une telle structure de donnée permet de réaliser en temps constant les opérations de base telles que : suppression d'une arête ou accès à une arête issue d'un sommet donné. En effet, $RECULER$ et $AVANCER$ ont alors un coût constant, tandis que $AUGMENTER$ a un coût linéaire en n.

6.1.3 Calcul d'un flot maximum

Pour calculer un flot maximum, on itère le calcul d'un flot bloquant sur le graphe flot graphe L_f pour la capacité résiduelle et l'augmentation de f en $f + f'$. On arrête l'itération lorsque le flot est maximum, ce que l'on constate - par exemple - par la condition $d(s, t) = \infty$.

Lemme 15 *Soit R' le graphe résiduel du flot $f + f'$ et $d'(s, u)$ la fonction distance à la source pour ce graphe R'. Pour tout sommet u on a $d(s, u) \leq d'(s, u)$.*

Preuve : L'argument est analogue à celui donné dans le cours pour l'algorithme d'Edmonds-Karp. Si la conclusion du lemme est en défaut, on choisit un contre-exemple tel que l'entier k

$$k = d'(s, u) < d(s, u)$$

soit minimal. On note que k est différent de zéro et on désigne par w un sommet juste avant u dans un chemin de R' de longueur minimale k allant de s à u. Par minimalité, on a

$$k - 1 = d'(s, w) \geq d(s, w).$$

On distingue deux cas :
premier cas : l'arête (w, u) est présente dans R_f. L'inégalité triangulaire donne alors

$$d'(s, u) = d'(s, w) + 1 = k \geq d(s, w) + 1 \geq d(s, u).$$

D'où une contradiction.
deuxième cas : l'arête (w, u) est absente de R_f. Puisque c'est une arête de R' c'est que le flot bloquant f' a emprunté (avec une valeur positive) l'arête inverse (u, w), laquelle appartient donc à L_f. Il vient, puisqu'on se trouve sur un graphe de niveau :

$$d(s, u) = d(s, w) - 1 \geq d'(s, w) - 1 = k - 2,$$

ce qui contredit, là encore, l'hypothèse $d'(s, u) < d(s, u)$.

Lemme 16 *On conserve les notations du lemme précédent. On a alors* $d(s, t) < d'(s, t)$.

Preuve : l'inégalité au sens large résulte du lemme précédent. S'il y a égalité, on choisit un chemin $p = (u_1, \cdots u_k)$ de longueur minimale allant de s à t dans le graphe R'. Ce chemin n'est pas un chemin de R_f. En effet, comme le flot est bloqant, p aurait au moins une arête saturée pour le flot f', ce qui veut simplement dire qu'elle n'est pas une arête de R' ! Soit i le plus grand tel que (u_i, u_{i+1}) ne soit pas une arête de R_f. cette arête étant dans R', le flot bloquant f' a emprunté (avec une valeur positive) l'arête inverse (u, w), laquelle appartient donc à L_f. Il vient, puisqu'on se trouve sur un graphe de niveau :

$$d(s, u_{i+1}) = d(s, u_i) - 1 \leq d'(s, u_i) - 1 = i - 1.$$

Comme le chemin p n'emprunte à partir de u_{i+1} que des arêtes de R_f, on déduit de ce qui précède :

$$d(s, t) \leq d(s, u_{i+1}) + d(u_{i+1}, t) \leq i - 1 + k - i - 1 = k - 2,$$

ce qui donne la contradiction cherchée.

Le lemme montre que l'on ne peut itérer que $n - 1$ fois au plus un algorithme de flot bloquant, puisque la distance $d(s, t)$ est majorée par $n - 1$. Le calcul d'un flot maximum par une méthode de flot bloquant peut donc être réalisée avec une complexité $O(mn^2)$, ce qui consitue un progrès vis à vis de la complexité $O(mn^2 + nm^2)$ de l'algorithme de Edmonds-Karp.

6.2 Graphes unitaires et applications

6.2.1 Définitions

Un graphe G muni d'une capacité C est *unitaire* s'il n'admet pas d'arête allant de s à t et si tout sommet u différent de s et t est d'un des deux types suivants :
type I *: u admet au plus arête entrante (c'est à dire une arête de la forme (v, u)) et cette arête, si elle existe, est de capacité 1,*
type II *: u admet au plus arête sortante (c'est à dire une arête de la forme (u, v)) et cette arête, si elle existe, est de capacité 1.*

On se donne un flot f sur un graphe unitaire. Pour un sommet de type I, le flot sur l'arête entrante ne peut valoir que zéro ou un, et dans ce dernier cas, il existe une unique arête sortante sur lequel il vaut 1 également. Toujours dans le cas où le flot vaut 1 sut l'arête entrante, celle-ci disparaît du graphe résiduel, mais l'inverse de l'unique arête sortante sur lequel le flot vaut 1 apparaît comme unique arête entrante avec capacité 1. Le sommet reste donc de type I dans le graphe résiduel.

De même, pour un sommet de type II, le flot sur l'arête sortante ne peut valoir que zéro ou un, et dans ce dernier cas, il existe une unique arête entrante sur lequel il vaut 1 également. L'arête sortante disparaît du graphe résiduel, mais l'inverse de l'unique arête entrante sur lequel le flot vaut 1 apparaît comme unique arête sortante avec capacité 1. Le sommet reste donc de type II dans le graphe résiduel. Le graphe résiduel est donc également unitaire.

Lemme 17 *Soit G un graphe unitaire. Un chemin améliorant de longueur ℓ admet au moins $\lfloor \frac{\ell}{2} \rfloor$ arêtes* critiques*, c'est à dire d'arêtes dont la capacité résiduelle réalise le minimum de $C(u, v) - f(u, v)$ quand (u, v) décrit les arêtes du chemin améliorant.*

Preuve du lemme : comme le graphe résiduel est unitaire, les observations qui précèdent l'énoncé du lemme montrent qu'un flot de valeur 1 sur un chemin améliorant sature l'unique arête entrante d'un sommet de type I et l'unique arête sortante d'un sommet de type II, soit une arête sur deux, d'où le résultat.

Lemme 18 *Soit G un graphe unitaire, f un flot sur G et d la distance de s à t dans le graphe résiduel R. Tout flot sur R a une valeur majorée par $\frac{n-2}{d-1}$.*

Soit f' un flot sur le graphe résiduel R. Comme R est unitaire, les observations qui précèdent l'énoncé des deux lemmes montrent que l'ensemble des arêtes de

G sur lesquelles f vaut 1 peut être décomposé en une réunion de chemins de s à t et de cycles, ces chemins et ces cycles ayant leurs ensembles de sommets respectifs deux à deux disjoints, si l'on ne tient pas compte des extrémités s, t des chemins. La valeur $|f'|$ du flot f' est exactement le nombre de chemins de s à t dans cette décomposition. Comme chaque chemin a $d-1$ sommets internes au moins (c'est à dire distincts de s, t), il y a au plus $\frac{n-2}{d-1}$ chemins, ce qui établit le résultat annoncé.

Le lemme précédent montre qu'un algorithme de flot bloquant pour le calcul d'un flot maximum ne peut itérer plus de $2\lceil\sqrt{n-2}\rceil$ fois le calcul du flot bloquant. En effet, le lemme 16 montre qu'après $\lceil\sqrt{n-2}\rceil$ itérations, $d(s,t)$ vaut au moins $\lceil\sqrt{n-2}\rceil+1$ et donc, d'après le lemme précédent, le flot maximum sur le graphe résiduel vaut au maximum $\sqrt{n-2}$, valeur qui ne peut donner lieu qu'à $\lceil\sqrt{n-2}\rceil$ itérations supplémentaires. En conclusion, pour un graphe unitaire, un algorithme de flot bloquant itére $O(\sqrt{n})$ fois le calcul d'un flot bloquant.

Le calcul du flot bloquant lui même est rendu plus efficace dans le cas d'un graphe unitaire. En effet, la contribution cumulée totale des exécutions des fonctions $AUGMENTER$ a dans ce cas un coût $O(m)$. En effet, cette contribution c est proportionnelle à la somme des longueurs ℓ_j des chemins définis par les valeurs courantes p_j de la variable p au moment des appels à la fonction $RECULER$. D'après le lemme 17, au moins $\frac{\ell_j-1}{2}$ arêtes de p_j sont saturées et supprimées de H par la fonction $AUGMENTER$, soit en tout

$$\frac{\sum_j \ell_j}{2} - O(m).$$

Cette quanté étant majorée par m, on en déduit que $\sum_j \ell_j$ et donc c sont en $O(m)$.

Le nombre des appels à la fonction $RECULER$ reste en $O(m)$, de même que le coût cumulé de leurs exécutions.

Enfin, le coût cumulé des exécutions de la fonction $AVANCER$ est aussi un $O(m)$, puisque tout arête ajoutée à p par cette fonction, avec un coût $O(1)$ est ensuite supprimée par $RECULER$ ou $AUGMENTER$, avec le même coût.

On déduit de ce qui précède que le calcul d'un flot bloquant sur un graphe unitaire est en $O(m)$. Le calcul d'un flot maximum par une méthode de flot bloquant est donc finalement de complexite $O(m\sqrt{n})$ pour les graphes unitaires.

6.2.2 Couplage dans les graphes bipartis

Un graphe biparti admet une partition de l'ensemble V de ses sommets en deux sous ensembles X et Y de façon que toute arête (u, v) ait son origine u dans X et son extrémité v dans Y. Un *couplage* est un ensemble d'arêtes d'un graphe biparti ayant deux à deux des origines et des extrémités distinctes. Un couplage est *optimal* si le nombre des arêtes est maximum. Il est immédiat de transformer la recherche d'un couplage optimal en un problème de flot maximal. Il suffit d'ajouter deux sommets s et t et des arêtes allant de s aux sommets de X et des sommets de Y à t, comme représenté en page 20 des transparents du cours, et en munissant toutes les arêtes de la capacité 1. Le graphe obtenu est unitaire.

En utilisant cette transformation, on en déduit que le calcul d'un couplage parfait est réalisable par un algorithme de complexite $O(n^{5/2})$, résultat obtenu initialement dans [2].

Bibliographie

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*, 2nd ed. Cambridge, Massachusetts : M.I.T. Press, 2001.

[2] J. E. Hopcroft, and R. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.* 2 (1973), 225-231.

[3] R.E. Tarjan. *Data Structures and Network Algorithms.* Society for Industrial & Applied Mathematics (1983).

Cours 7 (16 novembre)

Entiers

— Tests de primalité ; sommes de quatre carrés

Dans cette leçon, on présente des algorithmes efficaces pour reconnaître si un nombre entier est premier et pour obtenir la décomposition d'un entier en somme de quatre carrés, suivant le résultat démontré en 1770 par le mathématicien français Joseph Louis Lagrange. Les algorithmes présentés sont probabilistes et efficaces.

7.1 Symboles de Legendre et de Jacobi

7.1.1 Définitions

Soit p un nombre premier impair, a un entier premier avec p. Le symbole de Legendre $(a|p)$ vaut (-1) si a est un carré modulo p (on dit aussi un résidu quadratique modulo p) et -1 sinon. Afin que le symbole soit défini dans tous les cas, on pose $(a|p) = 0$ si p divise a.

Lemme 19 *Soit p un nombre premier impair, a, b des entiers positifs ou négatifs. On a :*
i $(a|p) = (-1)^{\frac{p-1}{2}}$ *;*
ii $(ab|p) = (a|p)(b|p)$ *;*

Preuve : On rappelle que l'anneau des entiers modulo p est un corps et que les éléments a du groupe multiplicatif des éléments non nuls ont chacun un ordre, qui est le plus petit entier d tel que $a^d = 1 \bmod p$, et qui divise le nombre d'éléments du groupe, soit $p - 1$. Si maintenant a est d'ordre d et u est un entier, $0 \leq u < d$, les éléments $a^u \bmod p$ sont distincts et solutions de l'équation $x^d = 1 \bmod p$. Comme l'anneau des entiers modulo p est un corps,

il n'y a pas d'autre solution et donc les éléments d'ordre d sont inclus dans le groupe engendré par a. De plus, a^u engendre le même groupe que a si et seulement si x est premier à d (c'est à dire inversible modulo d), ce qu'on voit facilement par des calculs simples sur les exposants de a. Il y a donc $\varphi(d)$ tels générateurs, où φ est la fonction d'Euler définie par :

$$\varphi(n) = \prod_i (p_i - 1) p_i^{e_i}$$

les p_i décrivant l'ensemble des facteurs premiers de n, et e_i désignant l'exposant de p_i dans la factorisation de n.

En notant N_d le nombre de sous-groupes engendrés par un élément d'ordre d, il vient :

$$p - 1 = \sum_{d|p-1} N_d \varphi(d)$$

et chaque N_d est au plus égal à 1. En réalité, tous sont égaux à 1, ce qu'on voit en établissant, pour n quelconque, l'égalité

$$n = \sum_{d|n} \varphi(d)$$

laquelle est prouvée en examinant successivement le cas où n est premier, puissance d'un nombre premier, puis par récurrence sur le nombre de facteurs prmiers de n, en utilisant la multiplicativité de n, laquelle découle du théorème des restes chinois. Les détails sont laissés au lecteur qui pourra se reporter à un cours de théorie des nombres, par exemple [5].

Ayant établi qu'il existait un et un seul sous-groupe engendré par un élément d'ordre d (et donc par $\varphi(d)$ tels éléments), pour chaque diviseur d de $p-1$, on en déduit en particulier qu'il y a $\varphi(p-1)$ générateurs du groupe des éléments non nuls modulo p, et donc que ce groupe est cyclique. Revenant à l'enoncé du lemme à démontrer, on choisit un générateur b du groupe. Les carrés sont les puissances de b d'exposant pair. Pour un tel élément $a = b^{2u} \bmod p$, on a $a^{\frac{p-1}{2}} = b^{u(p-1)} = 1 \bmod p$. Pour un non carré, $a = b^u \bmod p$, avec u impair, on a $a^{\frac{p-1}{2}} = (b^{\frac{p-1}{2}})^u \bmod p$. Or $b^{\frac{p-1}{2}}$ n'est pas égal à 1 modulo p, puisque b est d'ordre $p-1$. Comme la carré de cette quantité vaut 1, c'est donc qu'il s'agit de l'autre racine de l'unité dans un corps à p éléments, soit -1. Ceci achève la preuve de l'assertion *i* et l'assertion *ii* s'en déduit facilement.

7.1.2 Loi de réciprocité quadratique de Gauss

On va maintenant établir la célèbre loi de réciprocité quadratique de Gauss publié pour la première fois par Gauss dans ses *Disquisitiones arith-*

meticae en 1801 (voir [3]).

Théorème 10 *(***Loi de réciprocité quadratique de Gauss***) Soient p et q deux nombres premiers impairs, on a*

$$(p|q)(q|p) = (-1)^{\frac{p-1}{2}\frac{q-1}{2}}.$$

On a également $(2|p) = (-1)^{\frac{p^2-1}{8}}$.

On commence par établir le lemme suivant :

Lemme 20 *(***Lemme de Gauss***) Soit p un nombre premier impair, a un entier premier avec p ; on pose $a_j = (ja) \bmod p$; alors, $(a|p)$ vaut $(-1)^n$, où n est le nombre d'indices j tels que $a_j > p/2$.*

Preuve : On énumère les entiers $a_j > p/2$ sous la forme $r_1, \cdots, r_n$ et les entiers $a_j < p/2$ sous la forme $s_1, \cdots, s_k$. On note que les r_i sont deux à deux distincts, car les a_j le sont, de même pour les s_j. De plus, une somme $r_i + s_j$ n'est jamais nulle, car la somme des a_j correspondants est de la forme $(j_1 + j_2)a$ avec $j_1 + j_2 < p$. Les s_j et les $p - r_i$ sont donc un réarrangement des entiers $1, \cdots, (p-1)/2$. Prenant leur produit, il vient

$$\begin{aligned}
\prod_{j=1}^{(p-1)/2} a_j &= a^{\frac{p-1}{2}} \prod_{j=1}^{(p-1)/2} j \\
&= (a|p) \prod_{i=1}^{n} p - r_i \prod_{j=1}^{k} s_j \\
&= (a|p)(-1)^n \prod_{j=1}^{(p-1)/2} j,
\end{aligned}$$

d'où le résultat.

On établit ensuite le lemme suivant :

Lemme 21 *Soit p un nombre premier impair, a un entier impair et premier à p ; alors $(a|p)$ est la somme modulo 2 des entiers $\lfloor ja/p \rfloor$ quand j décrit l'ensemble des entiers $1, \cdots, (p-1)/2$.*

Preuve : On note S la somme modulo 2 des entiers $\lfloor ja/p \rfloor$. On observe que $ja = \lfloor ja/p \rfloor . p + (ja) \bmod p$. En faisant la somme de tous les ja, il vient

$$a \sum_{j=1}^{(p-1/2} j = Sp + \sum_{i=1}^{n} r_i + \sum_{j=1}^{k} s_j.$$

Par ailleurs, la somme

$$\sum_{i=1}^{n}(p-r_i)+\sum_{j=1}^{k}s_j$$

vaut $\sum_{j=1}^{(p-1/2}j=\frac{(p-1)(p+1)}{8}$. Par addition des égalités, on obtient :

$$(a+1)\frac{(p-1)(p+1)}{8}=np+Sp+2(\sum_{j=1}^{k}s_j).$$

Les entiers a et p étant impairs, il vient :

$$m+S=0 \bmod 2,$$

ce qui donne le résultat attendu,

On note que la seconde assertion du théorème résulte de la preuve ci-dessous : il suffit de considérer le cas où a vaut 2 au lieu d'être impair ; on déduit des calculs que $S+n$ vaut $\frac{(p-1)(p+1)}{8}$ modulo 2. En observant que $a_j=2j$ est $<p$, on observe que chaque terme de la somme S est nul, ce qui suffit pour conclure.

Il reste à achever la preuve de la loi de réciprocité quadratique. On considère l'ensemble Σ formé des couples (x,y), $1\leq x\leq (p-1)/2$, $1\leq y\leq (q-1)/2$. On note que Σ a exactement $\frac{(p-1)(q-1)}{4}$ éléments. On partage Σ en deux sous ensembles S_1 et S_2 suivant que $qx>py$ ou non. On note que pour x fixé, le nombre d'éléments de S_1 dont la première coordonnée est x est exactement $\lfloor qx/p\rfloor$. Le nombre d'éléments de S_1 est ainsi la somme considérée dans le lemme précédent pour calculer le symbole $(q|p)$. De même, le nombre d'éléments de S_2 est-il la somme considérée dans le lemme précédent pour calculer le symbole $(p|q)$. On en déduit que le produit des deux symboles vaut $(-1)^{|\Sigma|}=(-1)^{\frac{(p-1)(q-1)}{4}}$, ce qui est le résultat à établir.

7.1.3 Symbole de Jacobi

Soit a et n des entiers, n étant impair et positif. On écrit n sous forme d'un produit de facteurs premiers non nécessairement distincts $n\prod_{i=1}^{k}q_i$. Le symbole de Jacobi $(a|n)$ est alors défini par

$$(a|n)=\prod_{i=1}^{k}(a|q_i),$$

où les quantités $(a|q_i)$ sont des symboles de Legendre.

Lemme 22 *Soient a b des entiers, n, m des entiers impairs positifs ; on a :*
i $(ab|n) = (a|n)(b|n)$;
ii $(a|nm) = (a|n)(a|m)$;
iii $((-1)|n) = (-1)^{\frac{n-1}{2}}$
iv $((2)|n) = (-1)^{\frac{n^2-1}{8}}$
v $(m|n)(n|m) = (-1)^{\frac{(n-1)(m-1)}{4}}$

Preuve : La preuve repose sur des calculs simples. Les points *i* et *ii* sont immédiats. Le point *iii* et le point *iv* reposent sur les égalités :

$$\frac{ab-1}{2} = (\frac{a-1}{2} + \frac{b-1}{2}) \bmod 2,$$

$$\frac{a^2b^2-1}{8} = (\frac{a^2-1}{2} + \frac{b^2-1}{2}) \bmod 2,$$

pour a et b impairs. Le point *v* repose sur la loi de réciprocité quadratique et l'égalité

$$\sum \frac{p_i - 1}{2} = \frac{m-1}{2}$$

où m est impair et $m = \prod p_i$ est la décomposation de m en facteurs premiers non nécessairement distincts. Les détails sont laissés au lecteur.

La loi de réciprocité quadratique a une conséquence algorithmique tout à fait remarquable, qui est l'existence d'un algorithme de complexité cubique (en la taille des entiers) qui calcule le symbole de Jacobi $(m|n)$. De fait, le calcul suit essentiellement un algorithme de pgcd. On note $Jacobi(m, n)$ l'algorithme qu'on appelle pour n impair et $0 < m < n$ (le cas m négatif s'en déduisant à l'aide des règles *i* à *iii* du lemme ci-dessus). L'algorithme est récursif et procède comme suit :

1. Extraire de m la plus grande puissance de 2, $m = 2^h m'$, m' impair ; si h est impair, calculer $t = (-1)^{\frac{n^2-1}{8}}$;
2. Si $m' = 1$, retourner t ; sinon calculer $t = t.(-1)^{\frac{(n-1)(m'-1)}{4}}$ et $n' = n \bmod m'$;
3. Si $n' = 0$ retourner 0 ; sinon, appeler récursivement $Jacobi(n', m')$; retourner $t.Jacobi(n', m')$.

7.2 Tests de primalité

7.2.1 Test de Solovay et Strassen

Le test de Solovay et Strassen a été publié dans [9]. Il prend en entrée un entier positif impair dont on veut connaître la primalité et itère au plus k fois le test suivant :

1. choisir a au hasard entre 1 et n-1
2. calculer le symbole de Legendre $x = (a|n)$ et la quantité $y = a^{\frac{n-1}{2}} \bmod n$; si $x \neq y$ retourner "composé".

Si le test n'a pas retourné "composé" après k itérations, l'entier est déclaré "probablement premier".

Théorème 11 *Soit n un entier positif impair. Si le test de Solovay et Strassen retourne "composé", alors n n'est pas premier. Si inversement n n'est pas premier, alors la probabilité prise sur les choix successifs des aléas qu'il soit déclaré "probablement premier" est $\leq 1/2^k$.*

Avant d'établir la preuve du théorème, on observe que les calculs effectués à l'étape 2 sont de complexité cubique, si l'on se réfère à l'algorithme de calcul du symbole de Jacobi, proposé à la section précédente et qui repose sur la loi de réciprocité quadratique de Gauss.

On observe également que la qualification "probablement premier" est - dans un sens mathématique - dénuée de sens : un nombre est premier ou non ! La qualification "composée" constitue une preuve de non primalité, ainsi qu'on va le voir. La qualification "probablement premier" est, en un certain sens, une *vérité heuristique*, dont on peut parfaitement se contenter, spécialement si k est grand.

Preuve : Si le test retourne "composé", alors n ne peut être premier, puisque, si c'était le cas, le symbole de Jacobi coïnciderait avec le symbole de Legendre et serait donc calculé par la formule $(-1)^{\frac{n-1}{2}}$. Si maintenant, n est composé, alors l'ensemble des entiers a, qui sont inversibles modulo n et tels que $(a|n) = a^{\frac{n-1}{2}} \bmod n$ est un sous-groupe du groupe des éléments inversibles modulo n. Ce sous-groupe est propre. On suppose le contraire et on distingue deux cas suivant que que n est ou non divisible par le carré d'un nombre premier p.

Dans le premier cas, on choisit un générateur g du groupe des éléments inversibles modulo p et on calcule $g^{p-1} \bmod p^2$. Si cette quantité est différente de 1, on pose $h = g$; sinon, on pose $h = g(1 + p)$. On note que $h = g \bmod p$. Il s'ensuit que l'ordre de h modulo p^2 est un multiple de l'ordre de g modulo p, soit $p - 1$. Par ailleurs, dans le second cas, on a :

$$h^{p-1} = (1 + p)^{p-1} \bmod p^2 = 1 + p(p - 1) \bmod p^2.$$

Donc, dans les deux cas, h^{p-1} n'est pas égal à 1 modulo p^2. Il en résulte que l'ordre de h modulo p^2 qui divise l'ordre du groupe des éléments inversibles modulo p^2, soit $p(p-1)$ est en fait égal à $p(p-1)$. Par le théorème des restes chinois, il existe un entier a tel que a soit égal à h modulo p^2 et a soit égal à 1 pour tous les autres nombres premiers de la factorisation de n. La quantité $a^{\frac{n-1}{2}}$ n'est pas égale à 1 ni -1 : en effet, son carré a^{n-1} vaudrait alors 1 modulo p^2, donc $n-1$ serait multiple de l'ordre $p(p-1)$ de a. On aurait ainsi trouvé un diviseur commun p à n et $n-1$.

Dans le second cas, on peut écrire n comme produit d'un nombre premier p et d'un entier $m > 1$ non divisible par p. On utilise à nouveau le théorème des restes chinois : soit u un non résidu quadratique modulo p et v un carré modulo m. Soit a congru à u modulo p et à v modulo m. Par hypothèse, $a^{\frac{n-1}{2}}$ vaut $(a|p)(a|m) = (u|p)(v|m) = -1$. Soit maintenant, toujours par le théorème des restes chinois, b tel que $b = 1 \bmod p$ et $b = a \bmod m$. On a $b^{\frac{n-1}{2}} \bmod p = 1$ et $b^{\frac{n-1}{2}} \bmod m = a^{\frac{n-1}{2}} \bmod m = -1$ et donc $b^{\frac{n-1}{2}} \bmod n$ ne peut être égal ni à 1 ni à -1 et donc n'est pas une valeur d'un symbole de Jacobi, d'où la contradiction cherchée.

Finalement, l'ensemble des entiers a, qui sont inversibles modulo n et tels que $(a|n) = a^{\frac{n-1}{2}} \bmod n$ a au plus $\frac{\varphi(n)}{2}$ éléments, ce qui est inférieur à $\frac{n}{2}$

7.2.2 Test de Rabin

Le test de Rabin a été publié dans [7], mais il est analogue à un autre test proposé antérieurement par Miller [6], dans un cadre non probabiliste, fondé sur l'hypothèse de Riemann généralisée.

Le test prend en entrée un entier positif impair n dont on veut connaître la primalité et itère au plus k fois le test suivant :

1. choisir a au hasard entre 1 et n-1 ;
2. calculer l'exposant d de n dans la décomposition en facteurs premiers de $n-1$;
3. calculer $x_0 = a^{\frac{n-1}{2^d}} \bmod n$ par l'algorithme des carrés répétés ; calculer ensuite $x_i = x_{i-1}^2 \bmod n$ pour i de 1 à d ;
4. si $x_0 \neq 1 \bmod n$ et $x_i \neq -1$ pour $i = 0, \cdots d-1$, alors retourner composé.

Si le test n'a pas retourné “composé” après k itérations, l'entier est déclaré “probablement premier”.et

En utilisant l'exponentiation modulaire par carré répété, le temps d'exécution de cet algorithme est cubique.

Théorème 12 *Soit n un entier positif impair. Si le test de Rabin retourne "composé", alors n n'est pas premier. Si inversement n n'est pas premier, alors la probabilité prise sur les choix successifs des aléas qu'il soit déclaré "probablement premier" est $\leq 1/4^k$.*

Preuve : On se borne ici à établir que, si le test retourne "composé", alors n ne peut être premier, et on renvoie le lecteur à l'article [7] pour l'implication inverse. Si le test retourne composé et si la valeur de x_d est différente de 1, alors on a $x^{n-1} \neq 1 \bmod n$. Comme tous les éléments a compris entre 1 et $n-1$ sont tels que $a^{n-1} = 1 \bmod n$ lorsque n est premier, on peut bien conclure que ce n'est pas le cas. Si maintenant $x_d = 1$, on choisit le plus grand indice i tel que $x_i \neq 1$ et on observe que x_i est une racine carrée de l'unité qui n'est égale ni à 1 ni à -1. Ceci ne peut se produire dans le cas où n est premier car l'anneau des entiers modulo n est alors un corps, dans lequel l'éqaultion $x^2 = 1$ n'a que deux racines. La preuve est ainsi achevée.

7.2.3 Tests déterministes

Les deux tests présentés dans les sections précédentes sont - comme on l'a expliqué - de nature probabiliste. De fait, si l'on admet l'hypothèse de Riemann généralisée, la méthode de Rabin peut être rendue déterministe : plus précisément, on peut prouver que, si l'on a testé toutes les valeurs de a comprises entre 1 et $2(\log n)^2$, et que n n'a pas été déclaré "composé", alors il est premier. En l'absence d'hypothèse supplémentaire, l'existence d'un test déterministe de complexité polynomial est un résultat établi en 2002 par Agrawal, Kayal et Saxena[1].

7.3 Décomposition d'un entier en somme des quatre carrés

Dans ce paragraphe, on va, après quelques rappels sur la division des des polynômes et des entiers de Gauss, étudier un algorithme probabiliste permettant d'obtenir la décomposition d'un entier en somme de quatre carrés. L'exposé suit l'article de Rabin et Shallit [8].

7.3.1 Division des polynômes et des entiers de Gauss

L'opération de division euclidienne des polynômes à coefficients dans un corps est bien connue. Elle s'applique en particulier à des polynômes à coefficients entiers modulo p, où p est premier : le quotient de A par B, est un

polynôme Q et le reste R un polynôme de degré strictement inférieur à celui de B. La complexité de la division ainsi pratiquée est celle de n opérations arithmétiques sur les entiers $\leq p$, où n est le degré des polynômes donnés, soit $O(n(\log p)^2)$, avec les algorithmes habituels.

A partir de la division avec reste, on peut construire un algorithme d'Euclide de complexité $O(n^2(\log p)^2)$, qui calcule le pgcd de deux polynômes.

L'anneau des entiers de Gauss est formé des nombres complexes de la forme $x + iy$, avec x et y entiers. Dans cet anneau, les éléments inversibles sont ceux dont le module $|x + iy| = \sqrt{x^2 + y^2}$ vaut 1, soit 1, -1, i et $-i$. Une opération de division euclidienne avec reste peut être définie en général : $a + ib$ et $x + iy$ étant donnés, on peut trouver $p + iq$ et $r + is$ tels que

$$a + ib = (p + iq)(x + iy) + r + is,$$

avec $|r + is| < |x + iy|$.

Pour calculer $p + iq$ et $r + is$, on peut considérer le nombre complexe à coefficients rationnels $(a + ib)(x + iy)^{-1}$, soit $u + iv$ et calculer les entiers p et q les plus proches de u et v. le module de la différence $(u + iv) - (p + iq)$ est au plus $\frac{1}{\sqrt{2}}$ ce qui donne le résultat annoncé avec $|r + is| \leq \frac{1}{\sqrt{2}}|x + iy|$. La complexité de la division ainsi pratiquée est celle des opérations arithmétiques sur les entiers, soit quadratique en la taille avec les algorithmes habituels.

A partir de la division avec reste, on peut construire un algorithme d'Euclide qui calcule le pgcd de deux entiers de Gauss. Sa complexité est celle de $O(\log N)$ opérations arithmétiques, où N est un majorant des modules des entiers considérés.

7.3.2 Sommes de deux carrés

On étudie d'abord le cas d'un nombre premier p congru à 1 modulo 4 et on donne une version algorithmique du théorème de Fermat-Lagrange.

Théorème 13 *Soit p un nombre premier congru à 1 modulo 4 ; il existe un algorithme probabiliste de complexité moyenne $O(logp)$ opérations arithmétiques sur les entiers $\leq p$, qui décompose p en somme de deux carrés, c'est à dire qui calcule x et y tels que $x^2 + y^2 = p$.*

Preuve : On pose $p = 4k + 1$. Le symbole de Legendre $(-1|p) = (-1)^{\frac{p-1}{2}} = (-1)^{2k} \bmod p$ vaut 1, et donc (-1) est un carré modulo p. Soient u_0 et u_1 ses racines carrées. Soit b aléatoire ; l'application $b \to \frac{u_0+b}{u_1+b}$ est une bijection de l'ensemble des entiers modulo p - privé de $-u_1$ - dans lui même - privé de 1. Avec une probabilité qui est essentiellement $1/2$, $u_0 + b$ et $u_1 + b$ ont donc des symboles de Legendre différents, par exemple $(u_0 + b|p) = 1$ et

$(u_1 + b|p) = -1$. Le nombre $u_0 + b$ est alors la seule racine commune aux deux polynômes $(u-b)^2+1$ et $u^{2k}-1$: en effet $u_1 + b$ est l'autre racine de $(u-b)^2+1$ mais sa valeur en le second polynôme est -2. En calculant le pgcd des deux polynômes, on trouve le polynôme $u - (u_0 + b)$ et donc la valeur de u_0. Il est à noter que la complexité de ce calcul de pgcd est moindre que dans le cas général : en effet, il s'agit simplement de calculer, par la méthode des carrés répétés, la valeur modulo $(u-b)^2+1$ de u^{2k} ; on ne manipule ainsi que des polynômes de degré 2, ce qui ne produit, pour chacune des $O(\log p)$ itérations, qu'un nombre constant d'opérations arithmétiques.

Une fois trouvé un entier u tel que $u^2 + 1 = 0 \bmod p$, on calcule le pgcd de p et $u + i$ dans les entiers de Gauss, toujours en $O(\log p)$ opérations arithmétiques. Soit $x + y$ le résultat du calcul de pgcd. La quantité $x^2 + y^2$ divise à la fois p^2 et $u^2 + 1 < p^2$. Elle est donc égale à 1 ou p. reste à voir, pour achèver la preuve du théorème, que p et $x+iy$ ne peuvent être premiers entre eux. Si c'est le cas, alors p divise $u^2 + 1 = (u + i)(u - i)$ et est premier avec $u + i$. Il divise donc $u - i$ et donc le carré de son module, soit p^2 divise $u^2 + 1 < p^2$, ce qui donne bien une contradiction.

7.3.3 Sommes de quatre carrés

On établit maintenant le résultat suivant :

Théorème 14 *Il existe un algorithme probabiliste de complexité moyenne $O((\log n)^2 \log\log n)$ opérations arithmétiques sur les entiers $\leq n$, qui décompose un entier n en somme de quatre carrés.*

Preuve : On suppose d'abord n pair de la forme $2(2k + 1)$. On applique alors un théorème profond de Linnik [4], qui prouve que le nombre de premiers $p < n$ tels que $n - p$ soit somme de deux carrés est $> \frac{An}{\log n \log\log n}$, pour une constante A. On observe qu'une telle somme de carrés est nécessairement composée du carré d'un nombre pair et du carré d'un nombre impair. Le premier carré est congru à 0 modulo 4 et le second à 1. Comme n est congru à 2 modulo 4, p est bien congru à 1 modulo 4 et on peut donc lui appliquer la méthode de la section précédente. En prenant au hasard x et y, $< \sqrt{n}$, on trouve donc - avec en moyenne $O(\log n \log\log n)$ essais - un nombre premier $p = n - x^2 - y^2$, qui se décompose en deux carrés en $O(\log n)$ opérations arithmétiques. Il faut prendre garde qu'en s'appliquant à un entier non premier, la méthode de la section précédente ne garantit pas la terminaison. On doit donc - par exemple - terminer l'algorithme dès que le choix de l'aléa b ne conduit pas à une valeur unique d'une racine carrée de -1 modulo p.

Si n est impair, on applique la méthode précédente à $2n$, ce qui permet d'écrire $2n = x^2 + y^2 + u^2 + v^2$. Comme $2n$ est congru à 2 modulo 4 et que

les carrés modulo 4 valent 0 ou 1, il y a deux carrés de nombres pairs et deux carrés de nombres impairs, par exemple x et y sont pairs et u, v impairs. On écrit alors

$$n = ((x-y)/2)^2 + ((x+y)/2)^2 + ((u+v)/2)^2 + (u-v)/2)^2.$$

Enfin, si la décomposition de n en facteurs premiers admet une puissance de 2 d'exposant ≥ 2, on extrait la plus grande puissance de 2 d'exposant pair et on se trouve ainsi ramené à l'un des deux cas précédents.

Bibliographie

[1] M. Agrawal, N. Kayal and N. Saxena. PRIMES is in P, preprint , 2002.

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*, 2nd ed. Cambridge, Massachusetts : M.I.T. Press, 2001.

[3] C. F. Gauss. *Disquisitiones Arithmeticae*, Springer, 1986.

[4] Ju. V. Linnik. An asymptotic formula in an additive problem of Hardy-Littlewood, *Izv. Akad. Nauk SSSR. Ser. Mat.*, 24, 1960, 629–706.

[5] I. Niven, H. S. Zuckerman, H. L. Montgomery. *An Introduction to the Theory of Numbers*, 5th edition, Wiley, 1991.

[6] G L. Miller. Riemann's Hypothesis and Tests for Primality, *Journal of Computer and System Sciences* 13 (3), 1976, 300–317.

[7] M. O. Rabin, Probabilistic algorithm for testing primality, *Journal of Number Theory* 12 (1), 1980, 128–138.

[8] M. O. Rabin, J. O. Shallit. Randomized algorithms in number theory, *Comm. Pure Appl. Math.* 39 (1986), 239–256.

[9] R. M. Solovay and V. Strassen. A fast Monte-Carlo test for primality, *SIAM Journal on Computing* 6 (1), 1977, 84–85.

Cours 8 (23 novembre)

Transformation de Fourier rapide

— Algorithme de multiplication rapide des entiers

Dans cette leçon, on présente l'algorithme de Schönhage et Strassen, qui permet de multiplier deux entiers de n bits en $O(n \log n \log\log n)$ opérations. Cet algorithme a été publié dans [4]. On trouve un exposé dans l'ouvrage [1].

8.1 Transformation de Fourier dans un anneau

8.1.1 Définitions

Soit n un entier positif. On appelle racine primitive n-ième de l'unité dans un anneau, un élément ω de cet anneau, tel que :
i $\omega^n = 1$
ii Pour tout indice i, $i = 1, \cdots, n-1$,

$$1 + \omega^i + \omega^{2i} + \cdots + \omega^{(n-1)i} = 0.$$

On note que, si n est pair, et si n est inversible dans l'anneau, on obtient $\omega^{n/2} = -1$ e, faisant $i = n/2$ dans l'équation *ii* ci-dessus.

En reprenant l'algorithme de FFT et de FFT inverse exposés dans le cours, le lecteur pourra vérifier que, dès que n est une puissance de 2, inversible dans un anneau donné, et dès qu'on dispose d'une racine primitive n-ième de l'unité dans cet anneau, on peut implanter ces deux algorithmes.

8.1.2 Nombres de Fermat

On appelle nombre de Fermat un entier m de la forme de la forme $m = \omega^{n/2} + 1$, avec ω et n puissances de 2.

Lemme 23 *Dans l'anneau des entiers modulo m, ω est racine primitive n-ième de l'unité et n est inversible.*

Preuve : Pour voir que ω est racine primitive n-ième de l'unité, on observe d'abord que $\omega^{n/2} = m - 1$ et donc que $\omega^n = 1 \bmod m$. On évalue ensuite $P(\omega^j)$ où P est le polynôme $P(t) = 1 + t + \cdots + t^{n-1}$, dont il est facile de voir que c'est le produit des binômes $1 + t^\ell$ pour $\ell = 1, \cdots, 2^{k-1}$, k étant l'entier tel que $n = 2^k$. On constate que $P(\omega^j)$ est nul en isolant les puissances de 2 dans j, soit $j = u2^t$, $t < k$, u impair, et en fixant i de manière que l'exposant de 2 dans la décomposition en facteurs de $j2^i$ soit exactement $k - 1$. Il vient $1 + (\omega^j)^{2^i} = 1 + (\omega^{n/2})^u = 0 \bmod m$, ce qui implique bien que $P(\omega^j)$ vaut zéro.

Reste à voir que $n = 2^k$ est inversible. En observant que ω^n excède n et qu'il s'agit de puissances de deux on en déduit que n divise ω^n, soit $\omega^n = nv$ pour un entier v. Modulo m, v est l'inverse de n, ce qu'il fallait démontrer.

8.1.3 Convolution cyclique de polynômes à coefficients entiers

Soit A et B deux polynômes de degré $n - 1$

$$A(x) = a_0 + a_1 x + \cdots + a_{n-1} x^{n-1},$$

$$B(x) = b_0 + b_1 x + \cdots + b_{n-1} x^{n-1}.$$

Il est possible de calculer le produit AB modulo $x^n - 1$, appelé également convolution cyclique, comme polynôme à coefficients dans l'anneau des entiers modulo m en opérant comme suit :

1. Calculer les transformées de Fourier $\hat{A}$, $\hat{B}$ de A et B,
2. calculer le produit terme à terme de $\hat{A}$ et $\hat{B}$,
3. appliquer la transformation de Fourier inverse au résultat précédent.

La complexité de chaque étape est :

1. $O(n \log n)$ opérations de coût $O(\log m)$ pour l'étape 1,
2. n multiplications modulo m pour l'étape 2,
3. $O(n \log n)$ opérations de coût $O(\log m)$ pour l'étape 3.

Le coût modéré assigné aux opérations de FFT vient de ce que ce sont des additions ou des multiplications modulo m par des puissances de deux, qui ont un coût seulement linéaire. En supposant que ω est petit, disons majoré par 256, on obtient une complexité qui est $O(n^2 \log n) + n\mu(m)$, où $\mu(m)$ désigne le temps de calcul d'une multiplication modulo m.

Le calcul ci-dessus est exact, c'est à dire fournit le produit de A et B modulo x^{n-1} au sens des entiers, si les coefficients a_i et b_j sont majorés en valeur absolue par M et si $2nM^2 \leq m$. En effet chaque coefficient de la convolution cyclique

$$\sum_{i+j=u \bmod n} a_i b_j$$

est alors majoré par $m/2$ et donc parfaitement déterminé par son reste modulo m. De même, le produit est exact si les a_i et b_j sont positifs majorés par M et si $nM^2 \leq m$. Dans ce dernier cas, si M est seulement majoré par $\sqrt{m}$, il est possible de déterminer le produit exact en calculant également la valeur modulo n de AB modulo $x^n - 1$, ce qui correspond à calculer directement les valeurs modulo n de chaque coefficient de la convolution cyclique. Les entiers n et m étant premiers entre eux, le théorème des restes chinois permet de calculer la valeur modulo mn d'un coefficient c de la convolution cyclique à partir de ses valeurs c_m et c_n modulo m et n.

En termes de complexité, le surcoût attaché à l'application du thérème des restes chinois est négligeable : comme m-1 est divisible par n, il est facile de voir qu'un coefficient c est calculé par $c_m + m((c_n - c_m) \bmod n)$.

Le surcoût attaché aux "petites multiplications", c'est à dire aux multiplications modulo n est *a priori* non négligeable puisqu'il correspond à $O(n^2)$ multiplications de nombres de $\log n$ bits. Le lemme suivant montre qu'on peut cependant rendre cette complexité moindre que celle des étapes 1 à 3 du calcul modulo m.

Lemme 24 *Il existe un algorithme de calcul des petites multiplications de complexité* $0(n^2)$.

Preuve : L'idée est d'évaluer $(A \bmod n)(B \bmod n)$ au point $\beta = 3\lceil \log n \rceil$. Il s'agit d'une simple multiplication de deux entiers, qui peut être réalisée en temps $O(n^{\log 3})$ par l'algorithme récursif vu en cours (algorithme de Karatsuba). Cette complexité est bien un $O(n^2)$. Par ailleurs AB s'écrit

$$\gamma_0 + \gamma_1 x + \cdots + \gamma_{2n-2} x^{2n-2},$$

où chaque γ_u est le coefficient

$$\sum_{i+j=u} a_i b_j,$$

lequel est majoré par n^3. Par suite, dans le calcul, de les termes $\gamma_u(2^\beta)^u$ ont des décompositions binaires qui ne se chevauchent pas. L'écriture binaire du produit $(A \bmod n)(B \bmod n)$ évalué en β donne donc celle des coefficients γ_u par lecture directe. Les valeurs de la convolution cyclique modulo n s'en déduisent par la formule $c_u = \gamma_i + \gamma_{u+n} \bmod n$.

8.1.4 Produit de polynômes modulo $x^n + 1$

La méthode ci-dessus peut s'adapter *mutadis mutandis* au calcul de deux polynômes modulo $x^n + 1$. On explique les principales modifications à la méthode de la section précédente. On suppose que ω est une puissance paire de deux, ce qui permet de définir $\rho = \sqrt{\omega}$. On peut alors effectuer le changement de variable $x = \rho y$ et obtenir deux polynômes en y, soit $\overline{A}(y) = A(\rho y)$ et $\overline{B}(y) = B(\rho y)$. Le calcul de $\overline{A}.\overline{B}$ modulo $y^n - 1$ et modulo $m = \omega^{n/2} + 1$ donne un résultat $\overline{C}(y)$. On définit alors $C(x) = \overline{C}(\frac{x}{\rho})$. On a, en calculant dans l'anneau des entiers modulo m :

$$\overline{A}(y).\overline{B}(y) = \overline{C}(y) \bmod y^n - 1.$$

Soit,

$$A(\rho y).B(\rho y) = C(\rho y) \bmod y^n - 1,$$

d'où, par changement de variable,

$$A(x).B(x) = C(x) \bmod (\frac{x}{\rho})^n - 1.$$

La quantité ρ^n valant -1, le calcul est bien modulo $x^n + 1$, comme requis.

On note que le surcoût en termes de complexité est lié aux changements de variable : comme ρ est une puissance de deux, chaque coefficient est calculé en temps linéaire ce qui donne au plus un coût supplémentaire $O(n^2)$, qui est moindre que le coût des FFT.

Il y a aussi des modifications dans la détermination du calcul exact. En effet, les coefficients du produit modulo $x^n + 1$ sont donnés par une fomule différente :

$$\tilde{c}_u = \sum_{i+j=u} a_i b_j - \sum_{i+j=n+u} a_i b_j.$$

Cette formule permet de localiser les valeurs des coefficients dans un intervalle de longueur nM^2, lorsque les a_i et les b_j sont positifs majorés par M. Cet intervalle dépend de u et est de la forme $[-(n-u-1)M^2, (u+1)M^2]$. Cette localisation permet de calcul de $\tilde{c}_u$ à partir de ses valeurs modulo m et n. Les détails sont laissés au lecteur.

8.1.5 Conclusion

En résumé, on a montré dans cette section comment calculer par FFT de manière exacte le produit de deux polynômes de degré n modulo $x^n - 1$ ou $x^n + 1$ lorsque n est une puissance de 2 et que les coefficients des deux polynômes sont majorés par une valeur $M \leq \sqrt{m}$, où m est le nombre de Fermat $\omega^{n/2} + 1$, ω étant aussi une puissance de 2.

En supposant que ω petit, par exemple majoré par 256, le calcul a une complexité qui est $O(n^2 \log n) + n\mu(m)$, où $\mu(m)$ désigne le temps de calcul d'une multiplication modulo m.

8.2 Multiplication rapide des entiers

L'algorithme de Schönhage et Strassen prend en entrée un entier q puissance de 2 et deux entiers poitifs u, $v \leq 2^q$ et forme le produit $uv \bmod 2^q + 1$. Sa complexité est en $O(q \log q \log\log q)$. En prenant q plus grand que le nombre de bits de u et v, on obtient un algorithme exact de multiplication rapide des entiers. Il est à noter que cet algorithme semble complexe mais qu'il a été implanté (voir [5]). La performance devient meilleure que celle de l'algorithme récursif de Karatsuba à partir de 20000 chiffres décimaux environ.

8.2.1 Algorithme de Schönhage et Strassen

L'algorithme est récursif. On va d'abord expliciter la récursivité dans le cas où q est un carré, c'est à dire une puissance paire de 2 ; on écrit $q = n^2$ et on pose $m = \omega^{n/2} + 1$, avec $\omega = 16$.

Partant de l'écriture binaire de u et v sur q bits, ce qui laisse de côté les cas simples où l'un des entiers u, v vaut 2^q, on forme n blocs de n bits chacun. Ceci revient à considérer deux polynômes $A(x)$ et $B(x)$ de degré $n - 1$ à coefficients positifs majorés par $M = 2^n$, soit :

$$A(x) = a_0 + a_1 x + \cdots + a_{n-1} x^{n-1},$$

$$B(x) = b_0 + b_1 x + \cdots + b_{n-1} x^{n-1},$$

tels que $u = A(2^n)$ et $v = B(2^n)$. On constate que M est majoré par $2^n \leq \sqrt{m} \simeq \sqrt{16^{n/2}} = 2^n$. On est donc précisément dans les conditions de la section précédente, résumées dans le paragraphe 8.1.5, et on peut donc calculer le produit AB modulo $x^n + 1$ avec une complexité $O(n^2 \log n) + n\mu(m)$, où $\mu(m)$ désigne le temps de calcul d'une multiplication modulo m. On observe également que ce produit, évalué au point 2^n, vaut $A(2^n)B(2^n)$, à un multiple

près de $(2^n)^n + 1$. Il fournit donc la valeur de $uv \bmod 2^{n^2}$, c'est à dire de $uv \bmod 2^q$, comme attendu. Enfin, les multiplications modulo m sont gérées en appelant récursivement l'algorithme.

En conclusion, lorsque $q = n^2$, la complexité $\mu(2^q+1)$ de la multiplication rapide modulo $2^q + 1$ vérifie l'équation récursive :

$$\mu(2^q + 1) = O(n^2 \log n) + n\mu(2^{2n} + 1).$$

On examine maintenant les modifications à apporter dans le cas où q est une puissance impaire de 2, qu'on écrit $q = 2n^2$. On pose $h = 2n$, $m = \omega^{n/2} + 1$, avec $\omega = 256$. Les polynômes $A(x)$ et $B(x)$ sont définis en formant, dans l'écriture binaire de u et v sur q bits, n blocs de h bits chacun. On constate que M est majoré par $2^h \leq \sqrt{m} \simeq \sqrt{256^{n/2}} = 2^{2n} = 2^h$. On est donc à nouveau dans les conditions de la section précédente, résumées dans le paragraphe 8.1.5, et le reste de l'argumentation est inchangée.

Lorsque $q = 2n^2$, la complexité $\mu(2^q) + 1$ de la multiplication rapide modulo $2^q + 1$ vérifie l'équation récursive :

$$\mu(2^q + 1) = O(n^2 \log n) + n\mu(2^{4n} + 1).$$

8.2.2 Complexité de l'algorithme

On pose $u_q = \frac{\mu(2^q+1)}{q}$. Les deux équations récursives de la section précédente deviennent :

$$u_q = O(\log q) + 2u_{2n}$$

correspondant au cas $q = n^2$, et

$$u_q = O(\log q) + 2u_{4n}$$

correspondant au cas $q = 2n^2$. On pose $\theta(q) = 2n$ si $q = n^2$ et $\theta(q) = 4n$ si $q = 2n^2$, ce qui permet d'évaluer la profondeur de la récursivité.

Lemme 25 *On a $\theta^p(q) \leq 16$ dès que p est $\geq \log\log q$.*

Preuve : Soit $q = 2^k$; $\lambda(k) = \log(\theta(2^k))$. Il vient, si k est pair, $\lambda(k) = \frac{k}{2} + 1$, et si k est impair $\lambda(k) = \frac{k}{2} + \frac{3}{2}$. Dans tous les cas, on a $\lambda(k) \leq \frac{k}{2} + \frac{3}{2}$, ce qui donne récursivement

$$\lambda^p(k) \leq \frac{k}{2^p} + 3(\frac{1}{2^p} + \cdots + \frac{1}{2}) \leq \frac{k}{2^p} + 3,$$

qui est bien ≤ 4 dès que $p \geq \log k$, ce qu'il fallait démontrer.

On déduit du lemme, avec les mêmes notations, que l'on a :

$$u_q \leq O(k + 2\lambda(k) + \cdots + 2^{p-1}\lambda^{p-1}(k)) + 2^p u_{\theta^p(q)}.$$

Prenant $p = \lceil \log k \rceil$, on constate que le premier terme du majorant est un $O(kp)$ et le second un $O(k)$. On a ainsi établi que $u_q = O(k \log k)$ et donc que $\mu(2^q + 1)$ est bien - comme annoncé - en $O(q \log q \log\log q)$.

8.2.3 Amélioration de l'algorithme de Schönage et Strassen

En 2007, Martin Fürer a - plus de 35 ans après les résultats de Schönage et Strassen, amélioré la meilleure borne de complexité pour un algorithme de multiplication de deux entiers de n bits. Ce résultat peut être trouvé dans [3]. La borne de Fürer est en $O(n \log n 2^{\log^* n})$, où $\log^n$ désigne le nombre d'itérations de la fonction log nécessaires pour atteindre une valeur ≤ 1. La fonction $\log^* n$ croit moins vite que $\log n$, $\log\log n$ etc.

Bibliographie

[1] A. V. Aho, J. E. Hopcroft and J. D. Ullman . *The Design and Analysis of Computer Algorithms*, Addison Wesley, 1974.

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*, 2nd ed. Cambridge, Massachusetts : M.I.T. Press, 2001.

[3] Martin Fürer. Faster integer multiplication, Proceedings of STOC 2007, 57-66.

[4] A. Schonhage and V. Strassen. Schnelle Multiplikation grosser Zahlen, *Computing* 7, 1971, 281-292.

[5] Paul Zimmermann. Implantation de l'algorithme de Schönhage en C avec la bibliotheque BigNum.
`http ://www.loria.fr/ zimmerma/bignum/schon.bignum.c`

Cours 9 (30 novembre)

Algèbre linéaire et géométrie des nombres

— Algorithme LLL de réduction des réseaux

Dans cette leçon, on présente l'algorithme de réduction des réseaux de Lenstra, Lenstra et Lovász [3], qui fournit une base d'un réseau ayant des vecteurs qui sont tous relativement courts. Cet algorithme permet notamment de calculer un vecteur du réseau qui est dans un rapport constant de l'optimum. Ce rapport reste exponentiel en la dimension du réseau et la question de savoir si on peut garantir un rapport polynomial est ouverte. L'algorithme LLL est présenté dans l'ouvrage [1].

9.1 Réseaux à coordonnées entières

9.1.1 Définitions

Un réseau est un sous-groupe discret d'un espace $\mathbf{R}^n$. Dans le point de vue algorithmique adopté dans cette leçon, on ne s'intéresse qu'aux réseaux à coordonnées entières, qui sont des sous-groupes de $\mathbf{Z}^n$. Une base d'un réseau est une suite de vecteurs linéairement indépendants $(b_1, \cdots, b_k)$, qui engendre le réseau par combinaisons linéaires à coordonnées entières. Le nombre d'éléments k d'une base est la dimension du réseau. A toute base est associée une matrice B dont les k colonnes sont les vecteurs de base. Le déterminant de la matrice symétrique définie positive $\tilde{B}B$ est un entier positif et sa racine carrée est notée $\Delta(B)$. C'est en fait un invariant du réseau : deux bases se correspondent par une matrice de changement de base U de dimension $(k \times k)$, à coordonnées entières, dont l'inverse V est également à coordonnées

entières. Posant $B' = BU$, il vient $\Delta(B') = det(\tilde{U}\tilde{B}BU)^{1/2} = |det(U)|\Delta(B)$. En prenant l'identité analogue pour V, on en déduit que $|det(U)||det(V)|$ vaut 1, et donc que les deux déterminants $det(U)$ et $det(V)$ valent ± 1. La matrice U est dite *unimodulaire* et on a bien $\Delta(B') = \Delta(B)$.

On peut donc définir le déterminant d'un réseau L, soit $\Delta(L)$ comme celui d'une quelconque de ses bases... pour autant qu'on sache que tout réseau admet une base. Ce n'est pas un résultat complètement évident en général, mais - dans le cas simple des réseaux à coordonnées entières - on peut considérer les familles libres ayant le plus grand nombre d'éléments k et en choisir une $(b_1, \cdots, b_k)$ de déterminant minimum $\Delta(B)$, puisque le carré du déterminant est entier. Si une telle famille n'engendre pas le réseau, on prend b hors du sous-réseau engendré par la base et on écrit

$$b = \sum_i^k \alpha_i b_i$$

où les α_i sont rationnels. Le vecteur b' défini par

$$b' = b - \sum_i^k \lfloor \alpha_i \rfloor b_i = \sum_i^k (\alpha_i - \lfloor \alpha_i \rfloor) b_i$$

reste dans le réseau et a toutes ses coordonnées < 1. En extrayant de la famille $(b, b_1, \cdots, b_k)$ une famille libre $(b', b_1, \cdots, b_{i-1}, b_{i+1}, \cdots, b_k)$, on obtient un déterminant qui vaut $|\alpha_i - \lfloor \alpha_i \rfloor|\Delta(B)$, contradiction.

Un plus court vecteur d'un réseau L est un vecteur non nul $b \in L$ dont la norme euclidienne $||b||$ est minimale. Le théorème de Minkowski s'enonce comme suit :

Théorème 15 *Soit S une partie convexe symétrique de l'espace engendré par un réseau L de dimension k. Si le volume de S est $> 2^k\Delta(L)$, alors S a un point non nul dans L.*

Il résulte de ce théorème que le plus court vecteur d'un réseau est de longueur majorée par $\frac{2}{V_k^{1/k}}\Delta(L)^{1/k}$, où V_k est le volume de la boule unité d'un espace de dimension k. Toutefois, la preuve du théorème de Minkowski est non constructive et l'algorithme LLL est, d'une certaine façon, une version algorithmique du théorème.

9.1.2 Orthogonalisation de Gram-Schmidt

Etant donnée une base $(b_1, \cdots, b_k)$ d'un réseau, on peut lui appliquer l'orthogonalisation de Gram-Schid vue dans le cours de base. On obtient une

suite de vecteurs $(b_1^*, \cdots, b_k^*)$, à coefficients rationnels, deux à eux orthogonaux, tels que, pour tout $i \leq k$, $(b_1, \cdots, b_i)$ et $(b_1^*, \cdots, b_i^*)$ engendrent le même sous-espace. ces vecteurs, ainsi que les coefficients μ_{it}, $1 \leq t \leq i-1$, sont définis par :

$$b_i = b_i^* + \sum_{t=1}^{i-1} \mu_{it} b_t^*$$

ce qui donne

$$\mu_{it} = \frac{(b_i, b_t^*)}{||b_t^*||^2}$$

Les coefficients μ_{it} sont ceux d'une matrice triangulaire supérieure T, dont la i-ème colonne est composée des coefficients $(\mu_{i1}, \cdots, \mu_{ii-1}, 1)$, et qui permet de calculer la matrice B^* des b_i^* en fonction de la matrice B des b_i par la formule $B = B^*T$. Cette formule montre que le déterminant du réseau $\Delta(L)$ est égal à la racine carrée du déterminant de la matrice $\tilde{B^*}B^*$, soit au produit $\prod_{i=1}^k ||b_i^*||$. Le carré de ce produit, soit $\prod_{i=1}^k ||b_i^*||^2$, *a priori* rationnel est donc en fait un entier.

Conservant les notations ci-dessus, on considère un entier $i \geq 2$ et on note L_i le réseau de dimension $i-1$ engendré par $(b_1, \cdots, b_{i-1})$. Son déterminant est $\Delta_i = \prod_{j=1}^{i-1} ||b^*j||$ et son carré est un entier. Toutefois, ce déterminant n'est pas un invariant du réseau. C'est précisément ce qui sera le fondement de l'algorithme LLL. Le lemme suivant précise rôle de Δ_i et sera utile dans la suite.

Lemme 26 *Chaque coordonnée de b_i^* est un rationnel dont le dénominateur divise Δ_i^2 et chaque coefficient μ_{it}, $t \leq i-1$, est un rationnel dont le dénominateur divise Δ_{t+1}^2.*

Preuve : Le cas $i = 1$ est clair en considérant que Δ_i vaut 1 et l'on se restreint dans la suite à $i \geq 2$. Si l'on note B_i la matrice dont les colonnes sont les vecteurs $(b_1, \cdots, b_{i-1})$, alors, d'après ce qui a été vu dans le cours de base, b_i^* est est exactement donné par $(I - B_i^+)(b_i)$, où B_i^+ est le pseudo-inverse de B_i, soit $(\tilde{B_i}B_i)^{-1}B$. Toutes les matrices du second membre sont à coordonnées entières et l'inversion introduit un dénominateur qui est le déterminant de $(\tilde{B_i}B_i)$ soit Δ_i^2. Ceci établit la première assertion du lemme.

Pour la seconde on considère la formule

$$\mu_{i+1,t} = \frac{(b_{i+1}, b_t^*)}{||b_t^*||^2.}$$

On sait que les coordonnées de b_t^* ont un dénominateur qui divise Δ_t^2 soit $\prod_{j=1}^{t-1} ||b_j^*||^2$. Tenant compte de la division par $||b_t^*||^2$, le dénominateur devient un diviseur de $\prod_{j=1}^{t} ||b_j^*||^2$ soit Δ_{t+1}^2.

9.2 Algorithme LLL

9.2.1 Présentation

L'algorithme LLL prend en entrée une base $(b_1, \cdots, b_k)$ d'un réseau à coordonnées entières et calcule une base dite *réduite au sens de Lovász*. Une telle base a les deux propriétés suivantes :

- tous les coefficients μ_{it}, $1 \leq t < i \leq k$ sont tels que $|\mu_{it}| \leq 1/2$
- pour tout indice i, $1 \leq i \leq k$ on a $||b_i^* + \mu_{i,i-1} b_{i-1}^*||^2 \geq 3/4||b_{i-1}^*||^2$.

La condition de Lovász a une interprétation géométrique : elle exprime que la projection $\pi(b_i)$ de b_i sur le plan engendré par les vecteurs b_i^* et b_{i-1}^* est de norme $\geq \sqrt{3/4}||b_{i-1}^*||$. Le lecteur pourra établir à titre d'exercice que cette condition, jointe à la condition $|\mu_{it}| \leq 1/2$ implique que l'angle entre $\pi(b_i)$ et b_{i-1}^* est supérieur ou égal à $\arctan(\sqrt{2})$. La condition de de Lovász exprime donc que le parallélépipède engendré par les vecteurs de la base n'est pas trop "pointu". Cette intuition conduit aux propriétés suivantes.

Lemme 27 *Soit B une base $(b_1, \cdots, b_k)$ réduite au sens de Lovász. On a*
i $\Delta(L) \leq \prod_{i=1}^{k} ||bi|| \leq 2^{k(k-1)/2}\Delta(L)$,
ii Pour tout $t < i$, $||b_t^*|| \leq 2^{(i-1)/2}||b_i||$,
iii $|b_1| \leq 2^{(k-1)/4}\Delta(L)^{1/k}$,
iv Pour tout x non nul on a : $||b_1|| \leq 2^{(k-1)/2}||x||$.

Preuve : La propriété $|\mu_{it}| \leq 1/2$ implique

$$||b_i^* + \mu_{i,i-1} b_{i-1}^*||^2 \leq ||b_i^*||^2 + 1/4||b_{i-1}^*||^2,$$

ce qui, joint à la condition de Lovász, donne

$$3/4||b_{i-1}^*||^2 \leq ||b_i^*||^2 + 1/4||b_{i-1}^*||^2,$$

ou encore

$$||b_{i-1}^*||^2 \leq 2||b_i^*||^2.$$

ce qui donne par récurrence, pour $t < i$: $||b_t^*||^2 \leq 2^{i-t}||b_i^*||^2$. Le point *ii* du lemme est conséquence immédiate de cette inégalité. Pour le point *i*, on part de

$$b_i = b_i^* + \sum_{t=1}^{i-1} \mu_{it} b_t^*$$

soit, en utilisant le fait que les μ_{it} sont majorés par $1/2$:

$$||b_i||^2 \quad \leq \quad ||b_i^*||^2 + \sum_{t=1}^{i-1} 1/4||b_t^*||^2$$

$$
\begin{aligned}
&\leq ||b_i^*||^2(1 + 1/4\sum_{t=1}^{i-1} 2^{i-t}) \\
&\leq 2^{i-1}||b_i^*||^2.
\end{aligned}
$$

On a alors :

$$
\begin{aligned}
\Delta(L)^2 &= \prod_{i=1}^{k} ||b_i^*||^2 \\
&\leq \prod_{i=1}^{k} ||b_i||^2 \\
&\leq \prod_{i=1}^{k} 2^{i-1}||b_i^*||^2 \\
&= 2^{k(k-1)/2}\Delta(L)^2.
\end{aligned}
$$

Le point *iii* est onbtenu en faisant le produit des inégalités $||b_1||^2 \leq 2^{i-1}||b_i^*||^2$. Enfin, le point *iv* part d'un vecteur quelconque $x = \sum_i \alpha_i b_i$. En considérant le dernier indice j tel que le coefficient α_j correspondant est non nul, il vient $x = \sum_{i=1}^{j} \alpha_i b_i$, ce qui dans la base orthogonale peut s'écrire $x = \sum_{i=1}^{j} \beta_i b_i^*$, avec des β_i rationnels mais $\alpha_j = \beta_j$ entier. Il vient $||x|| \geq |\alpha_j|^2||b_j^*||^2 \geq 2^{-(j-1)}||b_1^*||^2$, ce donne bien $||b_1|| \leq 2^{(n-1)/2}||x||$, comme attendu.

9.2.2 Description et complexité de l'algorithme

Pris à un haut niveau d'abstraction, l'algorithme LLL prend en entrée une base $(b_1, \cdots, b_k)$ d'un réeseau L et opère comme suit :

1. Appliquer à la base $(b_1, \cdots, b_k)$ l'orthogonalisation de Gram-Schmid.
2. Rendre les μ_{it} de valeur absolue $\leq 1/2$ en remplaçant b_i par $b_i - \lambda b_t$, où λ est l'entier le plus proche de μ_{it}
3. S'il existe un premier entier $i \geq 2$ tel que la condition de Lovász d'indice i $||b_i^* + \mu i, i - 1 b_{i-1}^*||^2 \geq 3/4||bi - 1 * ||^2$ ne soit pas satisfaite, échanger b_{i-1} et b_i et retourner à l'étape 1, sinon retourner la valeur courante de la base $(b_1, \cdots, b_k)$.

Pour préciser un peu, quelques observations sont utiles.
Remarque 1 : Il est préférable de procéder aux opérations 1, 2 et au test 3 incrémentalement, c'est à dire d'initialiser une variable i à 1 et de calculer successivement b_i^* et les coefficients μ_{it} (en utilisant les valeurs déjà connues), de rendre les μ_{ij} de valeur absolue $\leq 1/2$, jusqu'à trouver un indice pour lequel la condition de Lovász n'est pas satisfaite.

Remarque 2 : Pour chaque $i \geq 2$, la seconde étape doit procéder par t décroissant, c'est à dire par une boucle : pour t allant de $i-1$ à 1 remplacer b_i par $b_i - \lceil \mu_{it} \rfloor b_t$. Tout d'abord, comme cette suite d'opérations ne modifie aucun b_j d'indice $< i$, elle ne change pas les coefficients μ_{jt} antérieurement calculés et dûment rendus de valeur absolue $\leq 1/2$. Avant chaque affectation de b_i correspondant à l'indice t, on a :

$$\frac{(b_i - \lceil \mu_{it} \rfloor b_t, b_t^*)}{||b_t^*||^2} = \mu_{it} - \lceil \mu_{it} \rfloor,$$

ce qui montre que la nouvelle valeur de b_i est bien telle que μ_{it} est $\leq 1/2$. La réaffectation de b_i ne change pas b_i^* puisqu'elle ajoute à b_i un élément de l'espace engendré par $(b_1, \cdots, b_{i-1})$. Enfin, chaque nouvelle affectation ne modifie pas les coefficients μ_{iu} d'indice $u > t$ puisque'elle ajoute à b_i un élément de l'espace engendré par $(b_1, \cdots, b_t)$, dont le produit scalaire avec b_u^* est nul. A la fin de la boucle, tous les μ_{it} ont été rendus de valeur absolue $\leq 1/2$.
Remarque 3 : Après l'échange éventuel de b_{i-1} et b_i à l'étape 3, il n'est pas nécessaire de reprendre le calcul de l'orthogonalisation de Gram-Schmid à son début puisque les vecteurs $(b_1, \cdots, b_{i-2})$ restent les mêmes ; on peut ainsi réinitialiser à l'indice $i-1$.

En termes de complexité, l'étape 1 nécessite, pour i fixé, $O(k)$ calculs de coeffients μ_{it} en mode rationnel. Chaque calcul consiste principalement en l'évaluation d'un produit scalaire de vecteurs de dimension n, soit en tout $O(kn)$ opérations en mode rationnel. Les autres opérations, calcul de b_i^*, réduction des μ_{it} ne changent pas cet ordre de grandeur. On suppose, ce qui sera démontré juste après, que l'algorithme s'arrête et on note N le nombre d'échanges opérés à l'étape 3. En analysant la version incrémentale de l'algorithme, on constate qu'elle est contrôlée par un indice i qui varie entre 2 et k, que l'indice vaut 1 au début de l'algorithme, k à la fin et qu'il est diminué N fois. C'est donc qu'il a été augmenté $N+k$ fois, d'où une complexité qui est un $O(knN)$ opérations en mode rationnel.

Lemme 28 *L'algorithme LLL se termine et le nombre N d'échanges qu'il réalise est en $O(k^2 \log M)$, où M est un majorant des normes euclidiennes des vecteurs d'entrée de l'algorithme.*

Preuve : On reprend les notations de la section 9.1.2 et on considère la suite des carrés des déterminants $\Delta_1^2, \cdots, \Delta_{k+1}^2$ atachés à une base $(b_1, \cdots, b_k)$ et on note que :

– Les opérations de l'étape 2 ne changent pas la valeur des b_i^*.

– L'échange éventuel de b_{i-1} et b_i à l'étape 3 ne modifie pas les réseaux L_j, pour $j = 1, \cdots, i-1$, ni pour $j = i+1, \cdots k+1$ et donc pas non plus les Δ_j^2 correspondants.
– L'échange de b_{i-1} et b_i à l'étape 3, remplace b_i^* par $\pi(b_i)$, projection de b_i sur l'espace engendré par b_i^* et b_{i-1}^*. Puisque la condition de Lovász n'est pas satisfaite, on a $||\pi(b_i)||^2 < 3/4||b_i^*||^2$ et donc la valeur de $\Delta_i^2 = \prod_{j=1}^{i-1} ||b_j^*||^2$ est diminuée d'un facteur 3/4 au moins.

En utilisant la majoration $\Delta(L_i) \leq \prod_{j=1}^{i-1} ||b_j||$, établie au point *i* du lemme 27, il vient $\Delta(L_i)^2 \leq M^k$. Les $\Delta(L_i)^2$ étant des entiers, ils ne peuvent chacun diminuer d'un rapport 3/4 qu'un nombre de fois qui est un $O(k \log M)$, soit en tout $O(k^2 \log M)$, ce qu'il fallait démontrer.

9.2.3 Majoration des rationnels

Il résulte de la section précédente que la complexité de l'algorithme LLL est - en conservant les notations ci-dessus - de $O(k^3 n \log M)$ opérations en mode rationnel. Pour terminer d'évaluer cette complexité et, en particulier, pour voir qu'elle est bien polynomiale, il reste à majorer numérateurs et dénominateurs des rationnels manipulés. S'agissant des dénominateurs, le lemme 26 montre qu'ils restent majorés par les quantités Δ_i^2, qui sont, on vient de le voir, en $O(M^k)$. Le lemme suivant permet de boir que cet ordre de grandeur reste essentiellement le même pour les numérateurs.

Lemme 29 *Les rationnels qui apparaissent dans les calculs de l'algorithme LLL admettent des numérateurs et des dénominateurs dont la taille est majorée par un $O(k \log M)$.*

Preuve : Le résultat est déjà connu pour ce qui est des dénominateurs, qui sont majorés par un $O(M^k)$. On note ensuite que chaque quantité $||b_i^*||$ ne peut que rester constante ou diminuer d'un facteur au moins $\sqrt{3/4}$. Toutes ces quantités restent donc majorées par M. Après les étapes 1,2,3, les valeurs de b_i sont données par

$$b_i = b_i^* + \sum_{t=1}^{i-1} \mu_{it} b_t^*$$

avec $|\mu_{it}| \leq 1/2$. Il s'ensuit que $||b_i||$ est majoré par un $O(kM)$. Au cours de l'étape 1, les quantités μ_{it} sont calculées par

$$\mu_{it} = \frac{(b_i, b_t^*)}{||b_t^*||^2}$$

On a donc $\mu_{it} \leq ||b_i||/||b_t^*||$. La quantité $||b_t^*||^2$ a un dénominateur majoré par Δ_t^2, ce qui permet de la minorer par $1/\Delta_t^2$. De ce qui précède, on déduit que

μ_{it} est majoré par un $O(kM^{k/2})$ soit un $O(M^k)$. Les dénominateurs des μ_{it} étant majoré par un $O(M^k)$, leurs numérateurs sont majorés par un $O(M^{2k})$. Reste à borner la variation des b_i à l'étape 2, les remplacements successifs de b_i par $b_i - \lceil \mu_{it} \rfloor b_t$ conduisent à ajouter à un vecteur majoré par un $O(kM)$ des quantités chacune majorée par un $O(kM^{k+1})$. L'opération étant répétée au plus k fois, on aboutit à un majorant qui est un $O(k^2M^{k+1})$. En prenant les logarithmes la preuve du lemme est achevée.

En conclusion, on voit, en assignant une complexité quadratique aux opérations sur les rationnels, que la complexité de l'algorithme LLL est en $O(nk^5(\log M)^3)$. Dans la pratique, on opére en mode flottant et non en mode rationnel. Ceci conduit à des implantations tout à fait performantes, comme celle de Victor Shoup [5], même si leur terminaison n'est pas, *stricto sensu*, assurée.

Bibliographie

[1] H. Cohen. *A Course in Computational Algebraic Number Theory*, Springer, 1993.

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*, 2nd ed. Cambridge, Massachusetts : M.I.T. Press, 2001.

[3] A. K. Lenstra, H. W. Lenstra Jr. and L. Lovász. Factoring polynomials with rational coefficients, *Mathematische Annalen* 261 (1982), 515–534.

[4] Hermann Minkowski. *Geometrie der Zahlen*, Teubner, Leipzig, 1896 ; republié par Johnson, New York, 1968.

[5] V. Shoup. A Library for doing Number Theory, www.shoup.net/ntl/

Cours 10 (14 décembre)

Programmation linéaire

— Algorithme de Khachyan

Dans cette leçon, on présente l'algorithme de l'ellipsoïde de Khachyan, qui établit que le problème de la programmation linéaire admet un algorithme de complexité polynomiale. L'algorithme de Khachyan a été initialement présenté dans l'article [3]. On pourra également se reporter à l'ouvrage [4]. Un autre algorithme polynomial pour la programmation linéaire, plus pratique au demeurant, a été proposé quelques années plus tard par Karmarkar [2]

10.1 Ellipsoïdes

10.1.1 Définitions

Soit B une matrice symétrique définie positive de taille $n \times n$ et soit t un élément de l'espace $\mathbf{R}^n$. On note $E(B,t)$ l'ensemble

$$E(B,t) = \{x | (x-t)^t B^{-1}(x-t) \leq 1\},$$

qu'on appelle *ellipsoïde* centré en t et de matrice B. Ecrivant $B = QQ^t$ et posant $u = Q^{-1}(x-t)$, on constate que $E(B,t)$ est l'image de la boule unité par l'application affine inversible $u \to t + Qu$. Il en résulte que le volume de l'ellipsoïde $E(B,t)$ est

$$Vol(E(B,t)) = V_n \sqrt{det(B)}$$

où V_n désigne le volume de la boule unité de $\mathbf{R}^n$, soit $\frac{\pi^{n/2}}{\Gamma(n/2+1)}$. On laisse au lecteur le soin de préciser comment définir Q en diagonalisant la matrice B, dont les valeurs propres sont réelles positives, dans une base orthonormale, c'est à dire associée à une matrice de changement de base U telle que $U^t U = I$.

10.1.2 Prototype de l'itération

On considère la matrice diagonale M dont les termes diagonaux sont $\frac{n^2}{(n+1)^2}$ pour le premier et $\frac{n^2}{n^2-1}$ pour les autres et le vecteur z dont la première coordonnée est $-\frac{1}{n+1}$ et les autres sont nulles. On note P la matrice diagonale telle que $PP^t = M$. L'ellipsoïde $E(M, z)$ est l'image de la boule unité par l'application affine inversible $u \to z + Pu$.

Lemme 30 *L'ellipsoïde $E(M, z)$ contient l'intersection de la boule unité et du demi-espace défini par $x_1 \leq 0$. Son volume est majoré par $e^{-\frac{1}{2(n+1)}} V_n$.*

Preuve du lemme : On écrit l'inéquation qui exprime l'appartenance d'un point de coordonnées $(x_1, \cdots, x_n)$ à l'ellipsoïde $E(M, z)$ sous la forme

$$\frac{(n+1)^2}{n^2}(x_1 + \frac{1}{n+1})^2 + \frac{n^2-1}{n^2}\sum_{i=2}^{n} x_i^2 \leq 1.$$

Cette inégalité est équivalente à

$$\frac{2(n+1)x_1^2 + 2x_1(n+1) + 1}{n^2} + \frac{n^2-1}{n^2}\sum_{i=1}^{n} x_i^2 \leq 1.$$

Le premier terme est majoré par $\frac{1}{n^2}$, car la quantité $2(n+1)(x_1^2 + x_1)$ est négative ou nulle pour $x_1 \leq 0$ et $|x_1| \leq 1$; le second par $\frac{n^2-1}{n^2}$ lorsque le point $(x_1, \cdots, x_n)$ est dans la boule unité. L'inclusion cherchée s'ensuit.

Quant à l'inégalité sur les volumes, elle résulte de ce que

$$det(B) = (\frac{n^2}{n^2-1})^{n-1}(\frac{n}{n+1})^2.$$

En utilisant l'inégalité $1 + u \leq e^u$, il vient

$$det(B) \leq e^{\frac{1}{n+1}} e^{-\frac{2}{n+1}} \leq e^{-\frac{1}{n+1}},$$

ce qui donne le résultat attendu.

10.2 L'algorithme de Khachyan

L'algorithme de Khachyan trouve un point d'un ouvert convexe borné S défini par un certain nombre d'inégalités **strictes** de la forme $(a_i)^t x < b_i$, et inclus dans la boule centrée à l'origine et de rayon R. En se reportant aux transparents du cours, on voit qu'on y note A la matrice dont les lignes sont les $(a_i)^t$. L'algorithme construit itérativement une suite d'ellipsoïdes $E(B_j, t_j)$, de volume décroissant, qui contiennent S.

10.2.1 Stratégie générale de l'algorithme

L'algorithme se compose des étapes suivantes :

1. Poser $j := 0$, $t := 0$, $B := Diag(1/R)$, où $Diag(1/R)$ désigne la matrice diagonale dont tous les termes diagonaux valent $1/R$.
2. Si la valeur courante t_j de t appartient au convexe S, retourner t_j.
3. Sinon, choisir un vecteur ligne a de A tel que $a^t t_j \geq b$, où b est la coordonnée correspondante de l'inégalité de définition de S.
4. Soir B_j la valeur courante de B. Construire un ellipsoïde $E(B_{j+1}, t_{j+1})$ de plus petit volume que $E(B_j, t_j)$ qui contient l'intersection de $E(B_j, t_j)$ et du demi-espace défini par $a^t(y - t_j) < 0$. La nouvelle valeur de t est t_{j+1} et la nouvelle valeur de B est B_{j+1}.
5. Incrémenter j et retourner à l'étape 2.

10.2.2 Les formules de mise à jour

Dans cette section, on va expliciter les formules de mise à jour incluses dans les transparents du cours et qui peuvent apparaître quelque peu mystérieuses. Le principe est de se ramener par des applications affines inversibles à la situation prototype de la section 10.1.2. Dans ce qui suit, on omet l'indice de j lorsqu'il résulte clairement du contexte.

Soit s un élément de l'espace $\mathbf{R}^n$. La symétrie relative à l'hyperplan orthogonal à s est définie par

$$\Sigma(x) = x - 2s.\frac{s^t x}{||s||}.$$

La matrice de cette application est $I - 2\frac{ss^t}{||s||}$ et donc cette application est sa propre transposée, $\Sigma = \Sigma^t$. De plus, $\Sigma^{-1} = \Sigma$ et Σ préserve le produit scalaire.

On note Σ la symétrie relative à l'hyperplan "médiateur" de v et w où $v = Q^t a$ et $w = ||v||e_1$ est colinéaire au premier vecteur de base. Les vecteurs v et w étant de même norme, cet hyperplan est défini par l'équation $v^t x = w^t x$ et est orthogonal à la différence $w - v$. On observe que Σ échange v et w. Dans le cas où v et w seraient égaux, on peut choisir pour Σ n'importe quelle symétrie relative à un hyeplan contenant $v = w$.

On considère maintenant l'application affine inversible obtenue en composant dans cet ordre

1. l'application prototype de la section 10.1.2, $u \to z + Pu$,
2. la symétrie Σ,

3. l'application $u \rightarrow t + Qu$, où t est la valeur courante du centre de l'ellipsoïde après j itérations et Q la valeur courante de la matrice telle que $B_j = QQ^t$.

Cette application transforme la boule unité en un ellipsoïde de volume moindre qui est précisément $E(t_{j+1}, B_{j+1})$. Cet ellipsoïde contient S. En effet l'application prototype transforme la boule unité en un ellipsoïde qui contient l'intersection de la boule unité et du demi-espace défini par $e_1^t x < 0$. La symétrie Σ laisse invariante la boule unité et transforme le demi espace défini par $e_1^t x < 0$ en le demi espace défini par $(Q^t a)^t x < 0$. Enfin, l'application $t + Qu$ transforme la boule unité en un ellipsoïde qui contient S et le demi-espace défini par $(Q^t a)^t x < 0$ en le demi-espace défini par $(Q^t a)^t Q^{-1}(y - t_j) < 0$, soit $a^t QQ^{-1}(y - t_j) < 0$, soit encore $a^t y < a^t t_j$, qui, par construction, contient également E.

Reste à voir la correction des formules proposées dans les transparents de cours. En appliquant la composition décrite ci-dessus, il vient pour t_{j+1} :

$$t_{j+1} = t_j + Q\Sigma(-\frac{w}{(n+1)||v||}) = t_j - \frac{1}{n+1} Q\Sigma\Sigma^t Q^t a.$$

En notant que $\Sigma\Sigma^t$ est l'identité, que QQ^t vaut B, et que $||v||^2 = a^t QQ^t a = a^t Ba$, on trouve bien la formule donnée dans les transparents du cours, soit

$$t_{j+1} = t_j - \frac{1}{n+1} \frac{B_j a}{\sqrt{a^t B_j a}}.$$

De même, en appliquant la compostion décrite ci-dessus, il vient pour B_{j+1} :

$$B_{j+1} = Q\Sigma P(Q\Sigma P)^t = Q\Sigma PP^t\Sigma^t Q^t.$$

On décompose PP^t en la somme de $\frac{n^2}{n^2-1} I$ et d'une seconde matrice ayant pour seul terme non nul le premier terme diagonal qui vaut $\frac{n^2}{n^2-1}(\frac{n-1}{n+1} - 1)$. La contribution globale du premier terme est, après simplifications

$$\frac{n^2}{n^2-1} QQ^t = \frac{n^2}{n^2-1} B_j.$$

La contribution du second se calcule à partir de l'élément w défini plus haut, par la formule

$$\lambda Q\Sigma w w^t \Sigma^t Q^t$$

en prenant pour λ la valeur

$$\lambda = -\frac{n^2}{n^2-1} \frac{2}{n+1} \frac{1}{||v||^2} = -\frac{n^2}{n^2-1} \frac{2}{n+1} \frac{1}{a^t B_j a}.$$

On observe que $\Sigma w = Q^t a$, ce qui donne une contribution

$$\lambda (B_j a)(B_j a)^t$$

soit finalement

$$-\frac{n^2}{n^2-1}\frac{2}{n+1}\frac{(B_j a)(B_j a)^t}{a^t B_j a},$$

qui, ajoutée à la première contribution, donne bien la valeur cherchée.

10.2.3 Terminaison de l'algorithme

En l'état, ce qui précède n'établit pas formellement que l'algorithme de Khachyan se termine en temps polynomial. Ce qu'on peut observer est que le volume des ellipsoïdes diminue d'un facteur multiplicatif $e^{-\frac{1}{2(n+1)}}$ à chaque étape. Ceci résulte du lemme et de la construction itérative se ramenant au prototype par une application affine inversible. Si le convexe S est non vide, il existe une boule de rayon r incluse dans S. Comme par ailleurs on a supposé que S était inclus dans une boule de rayon R, l'itération ne peut se poursuivre si l'on atteint

$$j \geq 2n(n+1)(\ln R - \ln r).$$

10.3 Conclusion : algorithme de complexité polynomiale pour la programmation linéaire

A partir des résultats précédents, on souhaite proposer un algorithme de complexité polynomiale pour résoudre un problème de programmation linéaire, qui consiste à déterminer le maximum de la fonction

$$\sum_{i=1} n c_i x_i$$

sous les m contraintes

$$\sum_i m a_{ij} x_i \leq b_i,$$

pour des variables non négatives $x_i \geq 0$.

On note N un majorant des coefficients $|a_{ij}|$. Comme indiqué dans le cours de base, il existe une quantité L, polynomiale en $\log M$, m et n telle que

– Les solutions de base ont des coordonnées qui sont des rationnels dont le numérateur et le dénominateur sont majorés par 2^L.

– L'optimum de la focntion d'objectif, sil existe, est également un rationnel dont le numérateur et le dénominateur sont majorés par 2^L.
– Deux solutions de base, dont les fonctions d'objectif diffèrent de moins de 2^{-2L} ont même valeur de la fonction d'objectif.

La preuve se fonde sur la résolution du système linéaire obtenu en choisissant les variables de base par les formules habituelles par déterminants extraits et sur la majoration des déterminants par la formule d'Hadamard.

On peut également établir par les mêmes méthodes le lemme suivant

Lemme 31 *Un système donné d'inégalités*

$$\sum_i ma_{ij}x_i \leq b_i$$

a une solution si et seulement si le système associé d'inégalités strictes

$$\sum_i ma_{ij}x_i < b_i + \varepsilon$$

a une solution pour $\varepsilon = 2^{-2L}$.

On résout un programme linéaire par recherche dichotomique sur l'optimum de la fonction d'objectif, en appelant un nombre polynomial de fois un algorithme qui décide si un convexe borné défini par des inégalités au sens **large** est non vide. On se ramène à un convexe ouvert défini par des inégalités au sens **strict** par le lemme. On conclut en utilisant pour chaque appel l'algorithme de Karmarkar.

Les détails de la preuve du lemme ci-dessus, facile mais pas complètement directe, et de la construction dichotomique sont laissés au lecteur, qui pourra se reporter à l'ouvrage [4].

Bibliographie

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*, 2nd ed. Cambridge, Massachusetts : M.I.T. Press, 2001.

[2] N. Karmarkar. A New Polynomial Time Algorithm for Linear Programming, Combinatorica, 4(4), 1984, 373–395.

[3] L.G. Khachyan. A polynomial algorithm in linear programming, Translated in Soviet Mathematics Doklady, 20 (1979) 191–194.

[4] C. H. Papadimitriou, K. Steiglitz. *Combinatorial Optimization : Algorithms and Complexity*, Prentice-Hall, 1982

Cours 11 (4 janvier)

Polynômes à une variable

— Algorithme de Berlekamp

Dans cette leçon, on présente l'algorithme de Berlekamp [1], qui calcule la factorisation d'un polynôme dont les coefficients sont des entiers modulo p, ainsi que l'algorithme probabiliste de Cantor-Zassenhaus [2], qui résout le même problème plus efficacement.

11.1 L'algorithme de Berlekamp

11.1.1 Réduction sans facteur carré

Tout polynôme à coefficients dans le corps $\mathbf{F}_p$ à p éléments admet une factorisation unique. Supposons qu'un facteur W apparaisse à une puissance ≥ 2 dans la factorisation de U, soit

$$U = V^i W.$$

Prenant la dérivée, il vient :

$$U' = iV^{i-1}W + V^i W'$$

ce qui montre que V^{i-1} divise U' et donc le pgcd de U et U', soit D. Il en résulte que le quotient U/D est sans facteur carré.

Le raisonnement ci-dessus est en défaut si D est nul. Ceci se produit lorsque le polynôme dérivé U' est lui-même nul, c'est à dire si chaque coefficient non nul u_i de U est d'indice $i = 0 \bmod p$. Le polynôme U s'écrit alors $\overline{U}(X^p)$ pour un polynôme convenable $\overline{U}$. Il suffit donc de factoriser $\overline{U}$, qui est de degré moindre et de noter que

$$U = \overline{U}(X^p) = (\overline{U}(X))^p.$$

On se ramène ainsi - récursivement - au cas sans facteur carré.

11.1.2 Point de vue algébrique

On suppose désormais U sans facteur carré et on considère sa décomposition

$$U = U_1 ... U_r$$

en facteurs irréductibles distincts. Le théorème des restes chinois s'applique et exprime que l'anneau quotient $\mathbf{F}_p[X]/U$ est isomorphe au produit $\prod_{i=1}^{r} K_i$, où K_i est le corps $\mathbf{F}_p[X]/U_i$. L'application linéaire

$$V \longrightarrow V^p - V \bmod U$$

est définie sur l'anneau quotient et l'isomorphisme mis en évidence ci-dessus permet de conclure que son noyau est de dimension r. En effet, V est élément du noyau si et seulement si, pour chaque indice $i \leq r$, on a $V^p - V = 0 \bmod V_i$. Or, dans le corps K_i seuls les scalaires sont solutions de l'équation $x^p - x = 0$. Il en résulte que le noyau est isomorphe au produit des sous-corps $\mathbf{F}_p$ de chaque K_i, ce qui montre bien que sa dimension est r.

Lemme 32 *Etant donné un polynôme non constant H du noyau. Alors U s'écrit*

$$U = \prod_{s \in \mathbf{F}_p} pgcd(U, H - s)$$

et cette factorisation n'est pas triviale, en ce sens qu'un des polynômes au moins du produit n'est ni constant ni de même degré que U.

Preuve du lemme : En substituant H dans l'équation

$$X^p - X = \prod_{s \in \mathbf{F}_p} (X - s),$$

il vient

$$H^p - H = \prod_{s \in \mathbf{F}_p} (H - s)$$

et donc, somme U divive $H^p - H$,

$$U = \prod_{s \in \mathbf{F}_p} pgcd(U, H - s).$$

Pour terminer, il suffit d'observer que si la factorisation est triviale, il y a une unique valeur de s telle que U divise $H - s$, ce qui implique que H est constant modulo U et fournit bien une contradiction.

11.1.3 L'algorithme de Berlekamp

L'algorithme de Berlekamp opère comme suit

1. Former la matrice Q de l'application $V \longrightarrow V^p - V \bmod U$;
2. Calculer une base $V_1, \cdots, V_r$ du noyau de $Q - I$ en prenant pour V_1 un polynôme constant ;
3. Poser $E = \{U\}$; $j = 1$; $k = 1$;
4. Tant que $j \leq r$ ou $k \neq r$ faire
   $j = j + 1$;
   pour chaque B de E
   retirer B à E ; $k = k - 1$;
   pour $s = 1$ à p
   $F = pgcd(B, V_j - s)$;
   si $\delta(F) \geq 1$, ajouter F à E ; $k = k + 1$;

Avant d'estimer la complexité de cet algorithme, on montre qu'il produit effectivement la factorisation cherchée. L'algorithme maintient un ensemble de polynômes E dont le produit est U et remplace chaque élément B de E par une factorisation. Ce dernier point résulte du lemme 32, où plutôt d'une variante du lemme qui s'applique à tout diviseur B de U. La factorisation est non triviale si H est constant modulo B. Si deux facteurs U_i et U_j ne sont pas distingués par l'algorithme, ce dernier retourne un élément B de E divisible à la fois par U_i et U_j.

A ce point, on choisit un V_ℓ tel que

$$V_\ell \bmod U_i \neq V_\ell \bmod U_j.$$

Un tel V_ℓ existe : sinon, par linéarité, on a

$$V \bmod U_i = V \bmod U_j,$$

pour tout élément V du noyau de $Q - I$, ce qui contredit le fait que ce noyau est de dimension r. Soit $s = V_\ell \bmod U_i$; dans la phase de l'algorithme où ℓ est traité, on a à calculer $F = pgcd(B, V_\ell - s)$. Ce polynôme F ne peut être égal à B car V_ℓ n'est pas constant modulo B, puisque ses valeurs modulo U_i et U_j diffèrent. Cette contradiction achève la preuve de correction de l'algorithme.

11.1.4 Complexité de l'algorithme

L'étape 1 de l'algorithme requiert le calcul récursif des polynômes $X^k \bmod U$, pour $k = 1, 2p, \cdots (n-1)p$. Les opérations dans le corps $\mathbf{F}_p$ se font en compexité $O((\log p)^2)$ et le calcul de $X^p \bmod U$ par un algorithme de carré

répété est en $O(n^2(\log p)^3)$, le terme n^2 provenant des élévations au carré ou des multiplications ou encore de la réduction modulo U de polynômes de dégré $2n$ qui sont les résultats de calculs intermédiaires. Ce calcul fait, celui des puissances successives de ce polynôme est en $O(n^3(\log p)^2)$.

L'étape 2 est une classique opération d'algèbre linéaire. La méthode du pivot de Gauss s'applique, avec complexité $O(n^3(\log p)^2)$.

Enfin, l'étape 3/4 nécessite au plus rp calculs de pgcd de polynômes de degré au plus n, soit un $O(rpn^2(\log p)^2)$. Globalement, l'algorithme n'est pas polynomial à cause du facteur p et la question de l'existence d'un algorithme de factorisation de polynômes modulo p, qui opère en temps polynomial reste ouverte.

11.2 Algorithme de Cantor-Zassenhaus

L'algorithme de Cantor-Zassenhaus opère comme suit

1. Former la matrice Q de l'application $V \mapsto V^p - V \bmod U$;
2. Calculer une base $V_1, \cdots, V_r$ du noyau de $Q - I$;
3. Poser $E = \{U\}$; $j = 0$; $k = 1$;
4. Tant que $k \neq r$ faire
   j=j+1 ;
   pour chaque B de E
   choir au hasard $H = \sum_{i=1}^{r} s_i V_i$;
   $F = pgcd(B, H^{\frac{p-1}{2}} - 1)$;
   si $F \neq B$ et F est non constant, retirer B de E et y ajouter F et B/F ; $k = k + 1$;

Toute la question est de déterminer la probabilité que le calcul du pgcd d'un polynôme B non irréductible et de $H^{\frac{p-1}{2}} - 1$ scinde le polynôme B.

Lemme 33 *Etant donné un polynôme H du noyau de Q et un facteur non irréductible B de U dont deux facteurs irréductibles distincts sont U_i et U_j, alors le pgcd de B et $H^{\frac{p-1}{2}} - 1$ distingue U_i et U_j, en ce sens que l'un divise ce pgcd et l'autre pas, si et seulement si la résiduosité quadratique de H est différente modulo U_i et U_j.*

Preuve du lemme : Par l'isomorphisme mis en évidence dans la section 11.1.2, H est un scalaire de $\mathbf{F}_p$ modulo tout polynôme U_i. Ce scalaire est - ou non - un résidu quadratique selon qu'il est - ou non - solution de l'équation $x^{\frac{p-1}{2}} - 1 = 0$, c'est à dire selon que $H^{\frac{p-1}{2}} - 1$ est - ou non - divisible par U_i. La preuve du lemme s'en déduit immédiatement.

Le lemme précédent montre que la probablité qu'un choix aléatoire de H ne scinde pas un facteur non irréductible donné B est majorée par la probabilité de choisir au hasard deux résidus quadratiques ou deux non résidus quadratiques, soit

$$(\frac{p-1}{2p})^2 + (\frac{p+1}{2p})^2,$$

quantité majorée par $\frac{19}{36}$ dès que p est ≥ 3. On laisse au lecteur le soin de conclure que l'espérance du nombre j de calculs de pgcd est un $O(r \log r)$, ce qui conduit bien à un algorithme probabiliste polynomial.

Bibliographie

[1] E. Berlekamp. Factoring polynomials over finite fields, Bell System Tech. J. 46 (1967), 1853–1859.

[2] D. Cantor and H. Zassenhaus. A new algorithm for factoring polynomials over finite fields, Math. Comp. 36 (1981), 587–592.

[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*, 2nd ed. Cambridge, Massachusetts : M.I.T. Press, 2001.

[4] D. Knuth. *The Art of Computer Programming*, Volume 2 : Seminumerical Algorithms. Addison-Wesley, Reading, MA, 1969.