ANALYSE d'ALGORITHMES
(Amphi0)

Philippe Flajolet

Le 30 septembre 2004

Il s'agit d'un résumé succint du cours d'introduction à l'analyse d'algorithmes au sein du MPRI, 2ème année. Ça a duré 2 heures 30 environ, avec env. 40 étudiants.

1 Analyse d'algorithmes

La théorie de la complexité traite de la classification des problèmes de calcul en grandes classes, comme P (ce qui se calcule en temps polynomial, $O(n^k)$), EXP (ce qui se calcule en temps exponentiel, $O(e^{n^k})$), ou encore NP dont le statut, en l'etat de nos connaissances est "intermédiaire". Cette classification est précieuse. Par exemple, sont du coté exponentiel (NP ou EXP) de nombreux problèmes d'optimisation combinatoire, le cassage de systèmes cryptographiques, les problèmes durs du calcul formel (systèmes polynomiaux et bases de Gröbner), etc. La classification mérite d'être affinée notamment pour les problèmes dans P d'utilisation courante (voire universelle) en informatique, comme: tri, recherche dans un dictionnaire, recherches de points dans des espaces multidimensionnels et géométrie algorithmique, algorithmique de base du calcul formel, etc.

On analyse les *algorithmes* plutot que les programmes, afin d'obtenir des résultats largement indépendants des calculateurs et des compilateurs. Soit un algorithme $\mathbf{A}$ opérant sur des données ou entrées $\mathcal{E}$. On se fixe une mesure de complexité τ, usuellement un décompte d'instructions. Par exemple, $\mathbf{A}$ est un algorithme de tri (tri bulle, tri-fusion, tri-rapide, etc); alors $\mathcal{E}$ est par exemple $\cup\mathbb{R}^n$ (on trie des réels = des flottants). On a toujours une notion de *taille*, notée n; par exemple pour les tris par comparaisons, il s'agira de la dimension du tableau à trier. On est donc dans la situation suivante: Soit

$$\tau_{\mathbf{A}}(e) = \text{le coût } \tau \text{ de l'algorithme } \mathbf{A} \text{ sur l'entrée } e \in E.$$

Caractériser la dépendance de τ en fonction de la taille $n = |e|$ des données.

La question ci-dessus mérite d'être précisée. On introduit:

- L'**analyse dans le meilleur cas**: il s'agit de déterminer

$$\tau^{\min}_{\mathbf{A}}(n) := \min_{|e|=n} \tau_A(e).$$

Cette mesure “optimiste”est peu indicative de la vraie complexité d'un algorithme. Par exemple, sur un tableau déjà trié, un algorithme de tri n'a rien à faire; il va en général s'en rendre compte tout de suite et la complexité dans le meileurs cas sera juste $O(n)$ pour tout le monde, ou à peu près.

- L'**analyse dans le pire cas**: il s'agit de déterminer

$$\tau_{\mathbf{A}}^{\max}(n) := \max_{|e|=n} \tau_A(e).$$

Cette mesure est utilise, surtout dans des contextes comme le temps réel, le calcul parallèle, les systèmes embarqués et synchrones. Mais il arrive souvent qu'un algorithme soit, dans 99.9999%, des cas mille fois plus rapide que dans le cas le pire. On a alors intérêt à prendre un léger risque occasionnel et préférer de fonder une classification sur les cas les plus “typiques”.

- L'**analyse en moyenne**: il s'agit de déterminer

$$\overline{\tau}_{\mathbf{A}}(n) := \operatorname*{Moyenne}_{|e|=n} \tau_A(e).$$

Cette quantité ($\overline{X}$ indique un moyenne, c-à-d. une espérance mathématique). Elle est censée représenter le cas “typique”de l'algorithme. Comme on le verra, on peut l'enrichir de diverses informations supplémentaires.

Par exemple, le tri utilisé dans de nombreux systèmes (`sort` en Unix) repose sur le Tri-Rapide (ou *Quicksort*). Cet algorithme est en $O(n \log n)$ en moyenne et $O(n^2)$ das le cas le pire. On connait par contre des algorithmes, Tri-Fusion ou Tri-par-Tas qui sont en $O(n \log n)$ dans le cas le pire, et paraissent meilleurs vis à vis de ce critère. On préfère néammoins le Tri-Rapide car il est dans l'immense majorité des cas environ deux fois plus rapide que les autres. L'analyse (Knuth, Sedgewick dans les années 1970) nous révèle en effet que la constante cachée dans le $O(n \log n)$ du coût moyen est environ deux fois meilleure que celle de ses rivaux. Donc, c'est cet algorithme qui a été préféré (cf Bentley).

Revenons à l'analyse en moyenne. Pour le tri par exemple, on se ramène de $\mathbb{R}^n$ à $\mathcal{E}_n$ qui va être l'ensemble des permutations de taille n. (Il suffit de ne considérer que les types d'ordre sous-jacents; pour les tris par comparaison, c'est raisonnable). Comment s'y prendre?

Tout d'abord, il faut savoir combien il y a d'entrées possibles de taille n. On note ainsi systématiquement

$$E_n = \|\mathcal{E}_n\| = \|\{e \mid |e| = n\}\|.$$

Pour le tri, $E_n = n! = 1 \cdot 2 \cdots n$ (la factorielle). Ensuite, on peut par exemple chercher à déterminer les quantités plus raffinées

$$E_{n,k} == \|\{e \mid |e| = n,\ \tau_{\mathbf{A}}(e) = k\}\|$$

Combien y-a-t'il d'entrées de taille n dont le coût est égal à k? Si on sait faire, on a alors déterminé la **probabilité** que le coût d'une entrée n soit égal à k. Le coût moyen

est alors (formule usuelle des espérances)

$$\overline{\tau}_{\mathbf{A}}(n) = \sum_k k \frac{E_{n,k}}{E_n} \frac{1}{E_n} \left(\sum_k k E_N n, k \right).$$

Tout ceci nous montre déjà que

Analyser un algorithme = Compter des configurations combinatoires.

On a un problème à un indice dans le cas de E_n ou encore dans le cas de la somme de la dernière équation, laquelle représente le *coût cumulé* de l'algorithme sur toutes les entrées de taille n. Si on réussit même à déterminer les $E_{n,k}$ on a une *analyse en distribution*. On a alors accès des informations comme la variance, laquelle mesure la dispersion d'une variable aléatoire:

$$\mathbb{V}\mathrm{ar}(X) = \mathbb{E}\left((X - \mathbb{E}(X))^2\right) = \mathbb{E}(X^2) - E(X)^2.$$

Ce qui est de fait plus suggestif, c'est l'*écart-type*:

$$\sigma(X) := \sqrt{\mathbb{V}\mathrm{ar}(X)},$$

lequl est dans la bonne échelle. Ici le moment d'ordre deux qui intervient dans la variance sera

$$\sum_k k^2 \frac{E_{n,k}}{E_n}.$$

En résumé, l'analyse d'algorithmes au delà du cas le pire, peut mettre en jeu:

- La moyenne, ou espérance mathématique des coûts;
- la variance, ou son copain, l'écart type, qui mesure la dispersion;
- la distribution des coûts qui en précise le profil;
- éventuellement, les grandes déviations qui quantifient le risque d'observation d'un coût anormalement loin de la moyenne.

C'est sur cette base du coût moyen qu'on choisit Quicksort, comme on l'a dit. Surtout que l'analyse plus fine (variance, distribution, grandes déviations) montre que pour $n \geq 1000$ et pour unen implantation convenable, la probabilité de s'écarter de plus de 30% de la moyenne est moins que 10^{-50}. A peu près autant de chances que de voir votre ordinateur se changer par effet quantique en une roue de gruyère ou un sandwich saucisson-beurre-cornichon!

Enfin, il y a une chose dont on n'a pas encore parlé, l'**asymptotique**. L'analyse d'un algorithme risque de nous donner des sommes, peut-être très complexes exprimant son coût moyen. On aimerait disposer d'une bonne echelle, d'une sorte de thermomètre pour comparer tout le monde. Déjà, combien y a t-il de permutations de n éléments? Réponse: $n! \equiv 1 \cdot 2 \cdots n$. Mais encore. Il y a la formule du mathématicien James Stirling (premièer moitié du 18e siècle):

$$n! \sim n^n e^{-n} \sqrt{2\pi n}.$$

On sait tout de suite (avec son ordinateur) que par ex. $100!$ est de l'ordre de 10^{158} ou $1000!$ de l'ordre de 10^{2568}. Plus proche encore de nos problèmes, l'analyse du TriRapide dans le cas moyen nous dit que le nombre de comparaison vaut (en moyenne) une quantité qui fait intervenir

$$2nH_n \qquad \text{où} \quad H_n := 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}.$$

Mais encore? On compare H_n à

$$\int_1^n \frac{dt}{t} = \log n.$$

De fait

$$H_n = \log n + \gamma + O\left(\frac{1}{n}\right),$$

où $\gamma = 0.57\ldots$ est la constante d'Euler. Voilà qui est très précis et commode.

Résumé: *Dans la suite on analysera des algorithmes plutôt que des programmes. L'analyse (moyenne, distribution) est pertinente. Elle est d'ailleurs encore plus pertinente, voire nécessaire, pour les algorithmes probabilistes ou "randomisés"—je préfère dire aléatoirisés—dans lesquels l'ordinateur se (com)plait à jouer aux dés.*

La discussion qui précède a montré que l'analyse d'algorithmes en ce sens nécessite des méthodes pour effectuer des dénombrements (ou énumerations, comptages) combinatoires et de l'asymptotique:

Analyse d'algorithmes $\rightsquigarrow$ analyse combinatoire + analyse asymptotique.

Eh bien ce qui va nous occuper une bonne partie du temps, c'est un cadre, celui de la *combinatoire analytique* qui permet précisément de faire cela de manière systématique en utilisant ce qu'il faut de maths.

2 La combinatoire analytique: dénombrements

On va présenter vu d'avion (l'analyse d'algorithmes vue du ciel!) quelques pièces du puzzle. Ca servira à donner une idée et mettre en place les choses.

On veut déterminer des suites de nombres (f_n) qui comptent des mots, des arbres, des permutations, etc. On définit la *fonction génératrice ordinaire* ("Ordinary Generating Functions", OGF):

$$f(z) := \sum_{n \geq 0} f_n z^n.$$

Exemple, Euler et Segner dès 1753 veulent compter combien il y a de triangulations (à nombre maximal, n, de triangles) d'un n-gone. Soit T_n ce nombre. On décide par commodité $T_0 = 1$ (la triangulation vide) et on se convainc de la récurrence

$$T_n = \sum_{k=0}^{n-1} T_k T_{n-1-k}.$$

On a un triangle “racine”, par ex, celui déterminé par les cotés de numéro 0, 1. Puis une triangulation a gauche et une autre à droite. Ca nous permet de calculer assez vite (en temps $O(n^2)$ en fait), les valeurs initiales:

$$1, 1, 2, 5, 14, 42, 132, 429, \dots .$$

Bon, mais est-ce qu'il y a plus de triangulations que de permutations? La récurrence quadratique sur les T_n donne par un petit calcul l'équation quadratique de $T(z)$:

$$T(z) = 1 + T(z)^2.$$

(Preuve on multiplie par z^n les deux membre de la relation et on effectue $\sum_n$; ça ressemble au produit de polynômes.) Ça se résout bien et

$$T(z) = \frac{1 - \sqrt{1 - 4z}}{2z}.$$

Là on se souvient du binome de Newton (formule de Taylor aussi),

$$(1 + x)^\alpha = 1 + \frac{\alpha}{1}x + \frac{\alpha(\alpha - 1)}{2!}x^2 + \cdots .$$

Donc avec $\alpha = \frac{1}{2}$ et $x = -4z$, par un nouveau petit calcul,

$$T_n = \frac{1}{n+1}\binom{2n}{n}.$$

En particulier:

$$T_{n+1} = \frac{4n = 2}{n+1}T_n.$$

Et asymptotiquement? Par Stirling et recalcul:

$$T_n \sim \frac{4^n}{\sqrt{\pi n^3}}.$$

Réponse: il y en a bien moins que de permutations, $T_{100} \approx 10^{57}$, etc. Au passage on a utilisé la notation internationales des binomiaux:

$$\binom{a}{b} := \frac{a!}{b!(b-a)!}.$$

Marche arrière on réfléchit. On compare la spécification (un peu comme une sorte de définition de type en programmation) des triangulations ($\mathcal{T}$) et l'équation de la fonction génératrice $T(z)$. Ô miracle:

$$\mathcal{T} = \square + \mathcal{T}\nabla\mathcal{T}.$$
$$T(z) = 1 + zT(z)^2.$$

On observe un dictionnaire: $\square$ devient 1; le triangle atomique ∇ devient z; coller ensemble (produit cartésien) devient un produit de séries.

Théorème 1: On a un dictionnaire **dictionnaire** dans le genre:

ϵ(objet vide)	1	Sequence	$\frac{1}{1-\cdot}$
$\mathcal{Z}$(atome)	z	Ensemble	Exp
$\times$	$\times$	Cycle	Log

Là, Exp et Log désignent une variante de l'exponentielle classique $(\exp(f) = e^f)$ et du logarithme ($\log$) respectivement. On appelle ça des opérateurs de Pólya.

Sequence: on aligne les objets comme dans une liste, on forme un train; Ensemble: on les mets dans un grand sac, l'ordre ne compte pas; Cycle: on considère les arrangements circulaires, comme dans une liste circulaire.

Pièce 2. *Pour les objets étiquetés, c'est des séries exponentielles.*

Dans la vie (combinatoire) il y a pas mal d'objets qui se composent d'atomes distingués les uns des autres, ce quón peut voir comme des numéros collés. C'est aussi utile aux structures d'ordres. Par exemple, une permutation comme

$$\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 2 & 5 & 3 & 6 & 1 & 4 \end{pmatrix},$$

peut être vue comme un graphe linéaire étiqueté

$$\mapsto 2 \leftarrow 5 \leftarrow 3 \leftarrow 6 \leftarrow 1 \leftarrow 4.$$

C'est très utile pour prendre en compte des structures ordonnées et ça sert par exemple pour gérer les files de priorité (arbres croissants, tas ou "heaps"), analyser le tri, etc.

Bon, ces choses se comptent avec des séries exponentielles définies comme associant à une suite (f_n) la série

$$\widehat{f}(z) = \sum_{n \geq 0} f_n \frac{z^n}{n!}.$$

On appelle cela la fonction génératrice exponentielle (EGF en anglais)

Theorème 2: On a une notion de produit étiqueté adaptée (détails ommis) et un dictionnaire un peu différent mais du même style qu'avant.

[Exemple non traité: les permutations alternates et $\tan(z)$.]

Pièce 3. *Pour les paramètres, on va aux fonctions génératrices bivariées.*

Souvenez vous qu'on aimerait bien avoir accès à des choses comme les $E_{n,k}$. Il s'agit du nombre de bêtes de taille n ayant un certain paramètre (combinatoire, χ, ou un coût, τ) égal à k. Facile: on définit (cas ordinaire) a partir d'une suite à deux indices la série génératrice bivariée (BGF en anglais)

$$f(z, u) = \sum_{n,k} f_{n,k} u^k z^n.$$

Bonne nouvelle: pour les paramètres gentils (techniquement ceux qui sont additifs ou encore "hérités"), on a encore une version du dictionnaire. C'est ca le **Théorème 3**, lequel existe en version ordinaire et en en version exponentielle.

A propos, on voulait par exemple faire de l'analyse en moyenne. Si on la la BGF, c'est facile: (i) le nombre d'objets de taille n vaut

$$\mathrm{coeff}[z^n]\,[f(z,u)]_{u=1}\,.$$

La moyenne du paramètre ou coût vaut:

$$\frac{\mathrm{coeff}[z^n]\,\left[\frac{d}{du}f(z,u)\right]_{u=1}}{\mathrm{coeff}[z^n]\,[f(z,u)]_{u=1}\,.}$$

(Preuve: $\frac{d}{du}u^k = ku^{k-1}$.)

[Exemple non traité: le nombre moyen de cycles dans une permutation de taille n; on trouve H_n.]

$\star\star\star\star\star\star$

Revenons encore sur nos pas. On a suggéré ce qui s'appelle des *méthodes symboliques*. Ça permet de compiler quasi-automatiquement une description d'un type combinatoire en une famille d´equations reliant des fonctions génératrices. Quand ça s'applique, *on ramène donc compter à spécifier (programmer) dans un langage combinatoire qui est un fragment rudimentaire de la théorie des ensembles*.

Un exemple important: les langages réguliers sont définis par expressions régulières et l'union combinatoire ressemble à $\cup$ (alias $+$); la concaténation ressemble à $\times$, la formation de séquences à l'opération $\star$ de Kleene. Cette analogie fonctionne dans le cas d'expressions régulières non ambiguës. Donc tout ce qui est de ce ressort possède une fonction génératrice rationnelle. On peut la produire à vue d'après la spécication.

3 La combinatoire analytique: l'analyse

Revenons aussi a l'exemple des nombres de Catalan (T_n). On a réussi l'analyse asymptotique par une chaîne de petits calculs: expliciter la fonction génératrice; développer et exprimer les coefficients (style formule de Taylor); enfin approximer l'expression exacte. Pour des cas plus difficiles, c'est trop fragile comme méthode ou bien trop compliqué. De fait les fonctions génératrices sont les objets centraux de la théorie. On aimerait bien les faire parler directement. La clef gît dans l'analyse complexe, c-à-d., des fonctions définies sur $\mathbb{C}$.

Pièce 4. *Les fonctions analytiques entrent en jeu. On peut faire de l'analyse des coefficients directement sur les fonctions en utilisant leurs singularités.*

On va commencer par une définition: soit f définie d'un ouvert $\Omega \subset \mathbb{C}$ dans les complexes $\mathbb{C}$. On dit que f est analytique en $z_0 \in \Omega$ si autour de z_0 et dans un petit disque, existe une représentation de f en série (de genre "Taylor"):

$$f(z) = \sum_{n\geq 0} c_n (z - z_0)^n.$$

Puis, f est analytique dans Ω si f est analytique en tout z_0 de Ω. Si on se restreint à un bout compact de Ω, il suffit même d'un nombre fini de petits disques. On a donc un très bon accès à f.

Une notion essentielle est celle de **singularité**: c'est un point du bord de l'ouvert de définition de la fonction au delà duquel la fonction ne peut aller tout en gardant son caractère analytique. Par exemple $1/(1-z)$ est singulière en $z=1$, car la fonction devient infinie; $\sqrt{1-4z}$ est singulière en $z=1/4$ car elle cesse d'y être dérivable, etc.

Voyons la Pièce 4 sur l'exemple des fractions rationnelles. J'observe d'abord que (r entier):

$$\mathrm{Coeff}[z^n]\frac{1}{(1-z)^r} = \binom{n+r-1}{r-1} \sim \frac{n^{r-1}}{(r-1)!}.$$

Ça se déduit facilement du développement de Newton ou encore développement binomial à exposant négatif. Alors on peut faire des choses comme estimer directement

$$\mathrm{Coeff}[z^n]\frac{1}{((1-z)^3(1-2z)^2(1-3z)}.$$

La base technique de tout ça n'est rien d'autre que le développement en éléments simples de la fraction rationnelle. Puis on contemple la chose au fond des yeux...

Les pôles plus loin contribuent exponentiellement moins. A distance égale de l'origine, il y a un poids qui est donné par l'exposant. De toutes façons on verra ça bientôt en détail dans la première partie du cours.

Ce qui nécessite la théorie des fonctions d'une variable complexe et qui est non trivial, c'est le fait que les formules de passage se généralisent. Par exemple, le calcul de $[z^n]1-z)^{-r}$ donné à la base pour r entier s'etend à r réel quelconque. De la sorte on peut voir que la série des triangulations et sa composante $\sqrt{1-4z}$ donnent lieu à

$$\sqrt{1-4z} \qquad \rightsquigarrow \qquad C\frac{4^n}{n^{3/2}}.$$

Cette traduction systématique vaut pour toute fonction qui a le même type de singularité.

Pièce 5. *On détecte de l'universalité. Par exemple, le nombre T_n d'arbres de taille n à degrés dans un ensemble D fixé vérifie immanquablement une estimation du genre*

$$T_n \sim C \cdot A^n n^{3/2}.$$

Les paramèters principaux vérifient de ce fait des **lois** *prévisibles.*

Expliquons! Si je prends n'importe quelle règle finie de formation d'arbres qui contraint les degrés, alors la singularité est toujours de type $\sqrt{Z}$. Plus précisément, soit D un ensemble fini de degrés permis et $\mathcal{T}$ la classe des arbres correspondants. (Par ex., $D=\{0,2\}$ donne les arbres binaires; $D=\{0,1,2\}$ les arbres d'expression avec des opérateurs unaires comme $\sqrt{\cdot}$ ou $\log$; les conditionnelles donnent des arbres de syntaxe ternaires, etc.) La fonction génératrice verifie une équation implicite

$$T(z) = z\phi(T(z)), \qquad \phi(y) := \sum_{d\in D} y^d.$$

Pour des arbres où $D=\{0,11,17\}$ par ex., $T(z)$ est une fonction algébrique de degré 17 mais sans expression "par radicaux". Néammoins, sa singularité intéressante (on

dit "dominante") va toujours être de type $\sqrt{Z}$. Ceci, même dans le plan complexe. Là on est dans des propriétés qui permettent l'analyse de singularité et justifient la forme donnée ci-dessus, $C \cdot A^n n^{3/2}$.

Il y a plus (pire?): la hauteur de tels arbres est invariablement en $\sqrt{n}$, en moyenne et en probabilité. La longueur de cheminement (=le coût total d'accès à tous les sommets) en $O(n^{3/2})$. Il y a une proportion asymptotiquement constante de feuilles, etc. Ca s'applique par exemple aux arbres de termes en calcul formel et en compilation. On peut même analyser le coût de règles de dérivation formelle: on obtient $n^{3/2}$ si on recopie les sous-expresions; $O(n)$ si on partage—en moyenne, toujours.

Pièce 6. *Il y a universalité parmi les classes d'arbres associées à des structures ordonnées. Par exemple, la profondeur moyenne (coût d'une recherche) est en* $O(\log n)$.

Disons juste que ceci concerne par exemple les fameux *arbres binaires de recherche*, mais comprend aussi les arbres quadrants de la recherche multidimensionnelle et de la géométrie algorithmique, les varaintes optimisées du TriRapide (médiane de 3, etc) et bien d'autres. Techniquement, on a des équations différentielles pour les dénombrements, des singularités prévisibles, ergo ....

Enfin, il y a une classe de problèmes importants qui se ramènent aux fonctions d'un ensemble fini.

Pièce 7. *Les caractéristiques de base des fonctions d'un ensemble fini dans lui-même sont du ressort de l'analyse de singularités des fonctions génératrice d'arbres (variante en* $\sqrt{Z}$*).*

Ca se relie par exemple à un algorithme de factorisation entière, celui de Pollard. Le résultat est une méthode qui factorise $N = pq$ en environ $\sqrt{p}$ où p est le plus petit des deux facteurs premiers de N. Ainsi, si $n \approx 10^{40}$, on peut le factoriser par un programme de moins de 10 lignes en environ 10^{10} opérations arithmétiques, soit quelques secondes de CPU(!).

4 Mots et bioinformatique

Une question importante est de repérer si l'apparition du motif TACTAC sept fois dans une chaîne sur l'alphabet $\{\mathsf{A}, \mathsf{C}, \mathsf{G}, \mathsf{T}\}$ de longueur 312000 est ou non "attendue".

Pièce 8. *Les mots, motifs et expressions régulières sont du ressort de la théorie de la combinatoire analytique, i.e., méthodes symboliques et analyse de pôles. On prédit très bien, sous modèle simple de l'aléa, ce qu'on doit attendre et au contraire ce qui est significatif".*

Donc on distingue le bruit statistique inévitable. (Par exemple, plus de 3% des gens dont le nom commence par la lettre "H" sont pauvres, mais ce n'est pas significatif car statistiquement "inévitable"; par contre la proportion de riches en région parisienne n'est pas statistiquement attendue donc significative.)

Techniquement, ceci repose sur une chaine qui relie les fractions rationnelles, les expresions régulières, les automates, et le "dictionnaire"du début du cours. Pour les automates, on a une formulation matricielle qui fait intervenir le spectre de matrices positives. Ceci est du ressort de la théorie dite de Perron-Frobenius laquelle se retrouve

dans l'étude des chaines de Markov (Automate fini = mémoire finie du passé = chaîne de Markov), donc on intersecte le cours de Réseaux.

Pièce 9. *Les méthodes de combinatoire (bijections, dénombrements) conduisent aussi à des algorithmes de génération aléatoire, c'est-à-dire, de simulation, efficaces.*

Mentionnons ici la "méthode récursive"implantée dans la bibliothèque `Combstruct` de Maple. Ceci s'applique en particulier à des familles très variées d'arbres. Par exemple, des arbres qui apparaissent en phylogénie: penser à l'arbre des espèces depuis Darwin!

Pièce 10. *Les sous-séquences communes donnent lieu à une riche combinatoire et une riche algorithmique.*

Comparer des morceaux d'ADN revient à se poser la question (voir Pièce 8 supra): à quel point deux suites aléatoires se ressemblent-elles? Une mesure de ressemblance est la longueur de la plus grande sous-séquence commune. On peut aussi vouloir imposer des bornes supérieures sur les "gaps", etc. Certains de ces problèmes se traitent par les méthodes du cours. Certains problèmes sont des questions ouvertes depuis plusieurs décennies.

5 Dictionnaires et information

On a vu l'arbre binaire de recherche. Mais si on veut gérer un dictionnaire de mots, c'est peu efficace: on passe son temps a recomparer les débuts des mots. Il a même été montré récemment (Janson et Fill) que le coût moyen d'une recherche parmi n mots est en $(\log n)^2$. Dans le même temps, on gaspille de la mémoire.

La structure d'arbre digital ou "trie"[treille] est comme un dictionnaire où on aurait mis un onglet pour les mots commençant par A, idem pour B, C, etc. Pour les mots commençant par A (ou B, etc) pas besoin de répéter le A initial. Même chose avec des sous-onglets. On gagne alors en mémoire de stockage, mais aussi en vitesse d'xaccès.

Théorème 11. Pour un alphabet binaire avec probabilités p, q de chaque lettre, la profondeur moyenne vaut

$$\sim \frac{1}{H} \log n,$$

où $H = -p \log p - q \log q$ est la fonction d'entropie.

L'analyse pour ce résultat fait appel aux séries génératrices, notamment exponentielles. On a des récurrences plus compliquées que les récurrences diviser-pour-régner habituelles. L'analyse complexe sert aussi.

Là on voit qu'on touche à la théorie de l'information. Mentionnons juste alors quelques connections fascinantes:

(a) Les algorithmes de compression par dictionnaire, comme celui de Lempel et Ziv, sont utilisés par `gzip`, etc. Ils compriment un texte typiquement dans un facteur de 3. Leur analyse, comme leur implantation d'ailleurs, se relie a celles des arbres digitaux et aux motifs.

(*b*) Les arbres digitaux servent de modèle à des protocoles de communication (le "protocole en arbre"), à des problèmes d'algorithmique distribuée, voire même à des algorithmes en fouille de données (estimation de cardinalités sur données massives). D'aucuns considèrent que c'est le processus le plus important de l'informatique...

(*c*) La théorie de l'arbre digital se relie puissamment à la théorie des systèmes dynamiques.

Une ou deux conférences-séminaires compléteront ce cours. Signalons au passage la possibilité (l'importance) d'expérimenter avec **le calcul formel**.