TD Algorithmique n°4

PTSI B Lycée Eiffel

13 novembre 2012

Commençons par préciser un peu des notions évoquées la semaine dernière concernant l'efficacité des algorithmes. Écrire des algorithmes qui tournent c'est bien, écrire des algorithmes qui font le boulot rapidement c'est mieux. Pour cela, on étudie ce qu'on appelle la complexité de l'algorithme, qui peut être de deux natures différentes :

- la complexité en espace : moins on utilise de mémoire, mieux c'est. Il faut donc éviter dans la mesure du possible de définir 38 variables dans un programme quand 12 étaient suffisantes, ou de créer un tableau de 12 000 cases si 1 000 suffisent. Nous ne nous intéresserons pas vraiment à ce genre de problèmes.
- la complexité en temps : plus le programme est rapide, mieux c'est. Pour cela, il faut essayer d'effectuer le moins d'opérations possibles pour résoudre notre problème. C'est cette notion que nous allons détailler un peu plus aujourd'hui.

Pour mesurer la complexité d'un algorithme, on essaie de rapporter le nombre d'opérations effectuées à la quantité de données que l'algorithme doit traiter. Nous continuerons à travailler avec des listes cette semaine, nous noterons donc n le nombre d'éléments de la liste intervenant dans notre programme. On dira algorithme effectuant des opérations sur cette liste est :

- linéaire si le nombre d'opérations effectuées est proportionnel à n (ainsi, si on multiplie la taille des données par 10, l'agorithme mettra 10 fois plus de temps à tourner).
- quadratique si le nombre d'opérations est proportionnel à n^2 (si on multiplie n par 10, le programme mettra 100 fois plus de temps à faire le boulot).
- cubique si le nombre d'opération est proportionnel à n^3, puis plus généralement polynômial s'il est proportionnel à n^p pour un certain entier p.
- exponentiel si le nombre d'opérations est proportionnel à α^n pour un réel $\alpha > 1$. Ce genre d'algorithme n'est étudié qu'en pratique, car le temps de calcul explose très vite avec la taille des données.

Prenons un exemple, un algorithme met 1 seconde à trier les données d'un tableau contenant 10 éléments, pour trier un tableau de 100 éléments, il mettra :

- 10 secondes s'il est linéaire
- 100 secondes s'il est quadratique
- environ 23 secondes si le nombres d'opérations est proportionnel à $n\ln(n)$ (cas fréquent)
- 1 000 secondes (un peu plus d'un quart d'heures) s'il est cubique
- 1.1^{90} secondes (environ une heure et demie) s'il est exponentiel de base 1.1.
- 2^{90} secondes (un peu plus de 39 milliards de milliards d'années) s'il est exponentiel de base 2.

Nous allons essayer de comparer différents algorithmes cherchant à résoudre un même problème : le tri des données (des nombres dans notre cas) d'un tableau à n éléments, qu'on cherche à replacer dans l'ordre croissant. Pour cela, la seule opération qu'on effectuera sera la comparaison de deux éléments du tableau (on néglige les opérations de déplacements d'éléments à l'intérieur du tableau). Voici quelques possibilités, pour chacune votre mission consiste à écrire une procédure en Maple effectuant le tri (comme dans le TD précédant, les procédures prendront comme argument le tableau et son nombre d'éléments), et surtout à essayer de compter le nombre de comparaisons effectuées.

Tri par sélection

Une méthode très intuitive pour trier un tableau est la suivante : on commence par chercher le plus petit élément de notre tableau, et on le place en tête, puisqu'il sera le premier élément du tableau trié. On recommence avec les éléments restants.

Tri par insertion

Encore un algorithme intuitif et un peu naïf. On créée une copie du tableau et on y place le premier élément du tableau à trier. Puis on place le deuxième élément au bon endroit dans le nouveau tableau (devant le premier ou derrière selon qu'il est plus petit ou plus grand), on ajoute ensuite le troisième élément etc.

Tri à bulles

Le nom ridicule de cet algorithme tient à une comparaison assez douteuse avec un phénomène visible dans les aquariums (ou les bouteilles de Coca) : les plus grosses bulles remontent plus vite que les petites. Ici, c'est un peu la même chose : on fait « remonter » les plus gros éléments en premier. En pratique, on effectue l'algorithme suivant : on parcourt le tableau à trier de gauche à droite, en comparant chaque élément à l'élément qui le suit. Si l'élément en question est plus grand que son successeur, on les échange. Une fois le parcours terminé, on recommence, mais en se dispensant de comparer les deux derniers éléments de tableau (pourquoi ?). Puis on continue à faire des parcours en diminuant d'un le nombre de comparaisons à chaque fois.

Tri fusion

Cet algorithme se base sur le principe, fréquemment utilisé en informatique, « diviser pour régner ». On procède en pratique de la façon suivante : on sépare le tableau en deux sous-tableaux de taille égale, on trie chacun de ces deux tableaux, puis on les fusionne. En quoi consiste cette fusion ? On compare le premier élément de chacun des deux tableaux, on sélectionne le plus petit (qui sera forcément le premier élément du tableau final), on l'enlève de son tableau et on recommence. On obtient ainsi, à partir de deux sous-tableaux triés de taille n chacun, un tableau trié de taille $2n$. Poussons le principe un peu plus loin. Pour chacun des deux sous-tableaux, on va les trier en effectuant sur eux un tri fusion, et ainsi de suite récursivement. On découpe en fait nos tableaux jusqu'à obtenir des tableaux de taille 1 (qui se trient tout seuls), puis on fait des fusions successives.

Tri rapide

Le tri rapide (quicksort en anglais) est réputé être l'algorithme le plus utilisé au monde. Il se base également en partie sur le principe diviser pour régner, mais n'utilise pas de fusion. C'est en fait très simple : on choisit un élément dans le tableau, et on le compare à tous les autres, en formant deux tas : ceux plus petits que lui et ceux plus grands que lui. Il ne reste plus qu'à trier ces deux tas en utilisant la même méthode, l'élément ayant servi de pivot se trouvant au milieu des deux sous-tableaux une fois le tri terminé. Cet algorithme souffre d'un léger défaut : son efficacité dépend fortement du choix du pivot à chaque étape. Si par malheur on tombe sur le plus grand ou le plus petit élément du tableau on fait une étape pour rien. En pratique, on prend usuellement le premier élément du tableau, en supposant que statistiquement ce sera un bon pivot. Essayez de compter le nombre de comparaisons dans le cas où on tombe en gros tout le temps sur un élément situé au milieu du tableau trié.

Tri par casiers

C'est une méthode de tri extrêmement efficace (elle ne nécessite essentiellement aucune comparaison) mais qui ne fonctionne qu'avec des entiers ou assimilés (en tout cas un ensemble fini pas trop gros). Supposons qu'on veuille trier un tableau d'entiers tous compris entre 1 et n, avec n connu à l'avance ou calculé en cherchant le maximum du tableau ($n-1$ comparaisons. On crée un tableau de taille n rempli de 0, et on parcourt ensuite le tableau à trier, en incrémentant pour chaque élément la case du tableau correspondant à la valeur de l'élément. À la fin, il suffit de reconstituer un tableau contenant le bon nombre de fois chaque valeur ayant été incrémentée. On peut en fait utiliser des variantes de cet algorithme dans des cas plus généraux : on veut trier des noms (dans l'ordre alphabétique), on crée un tableau de 26^2 cases correspondant à chaque couple de lettres possibles et on place chaque mot dans la case correspondant à ses deux premières lettres. Il ne reste ensuite plus qu'à trier les éventuels mots commençant par les mêmes lettres, ce qui met assez peu de temps comparé au tri global du tableau.

Comparaisons

Si vous êtes courageux, faites à la main les étapes du tri à bulle, du tri fusion et du tri rapide pour trier le tableau suivant : $[4; 13; 11; 6; 9; 1; 7; 2; 10; 16; 14; 3; 8; 5; 15; 12]$, et compter le nombre de comparaisons pour chaque méthode.

Pour le tri d'un tableau de 10000 éléments, on met environ 2 minutes en utilisant un tri à bulles (119.1 secondes avec un tableau aléatoire), 98.1 secondes avec un tri par sélection, 62.5 secondes avec un tri par insertion, 0.109 secondes pour un tri fusion et 0.063 secondes pour un tri rapide.