Complexité
TD 6 - Correction

Master 1 Informatique

Serge Miguet, Guillaume Metzler, Tess Masclef
Institut de Communication (ICOM)
Université de Lyon, Université Lumière Lyon 2
serge.miguet@univ-lyon2.fr guillaume.metzler@univ-lyon2.fr
tess.masclef@univ-lyon2.fr

Etude des nombres premiers

1) Proposez un algorithme ayant pour complexité $\mathcal{O}(p)$ permettant de déterminer si le nombre p est premier.

L'approche naïve dont il est fait mention ici consiste à tester tous les entiers de 2 à $p-1$ afin de voir si l'un d'entre eux divise p. Si un tel entier existe, le nombre p n'est pas premier, dans le cas contraire p est premier.

2) Quelle est la complexité de cet algorithme, exprimée en fonction de la taille des données d'entrée (où n est le nombre de bits de p lorsqu'il est écrit en base 2) ?

L'algorithme précédemment proposé consiste à effectuer un nombre de division de l'ordre de p afin de tester si un tel entier p est premier, cette complexité est donc linéaire.

Regardons maintenant la représentation binaire du nombre p et supposons qu'il soit écrit sur $n = \log_2(p)$ bits afin de se rapprocher du langage machine.
Avec cette approche, nous devons également tester la divisiblité de p par tous les entiers inférieurs à p, *i.e.* il faut tester toutes les combinaisons de 0 et 1 sur les n bits. On a donc une complexité en $\mathcal{O}(2^n)$, elle est donc exponentielle avec cette approche binaire.

3) Montrez qu'une petite astuce permet de réduire la complexité de cet algorithme $\mathcal{O}(\sqrt{p})$.

Dans la première question, nous avons proposé un algorithme qui consiste à tester tous les entiers compris entre 2 et p. Est-ce réellement nécessaire ? Non !

En effet, on pourrait commencer par faire un test simple, vérifier si p est pair ou impair, s'il est pair il n'est évidemment pas premier donc on s'arrête.
Cela permet d'avoir une complexité qui reste linéaire mais divisée par 2 (inutile de tester tous les diviseurs pairs), *i.e.* en $\mathcal{O}\left(\frac{p}{2}\right)$.

De plus, supposons que p ne soit pas premier, *i.e.* il existe des entiers a et b (plus petits que p) tels que

$$p = a \times b.$$

On peut alors dire $a \mid p$ (lire a *divise* p) et $b \mid p$.

Dans ce cas, on a nécessairement $a \geq \sqrt{p}$ et $b \leq \sqrt{p}$ (ou inversement, faire un petit raisonnement simple pour s'en convaincre.)

Il n'est donc pas nécessaire de tester tous les entiers de 2 à $p-1$ mais on peut s'arrêter à $\sqrt{p}$. Inutile de tester au délà de cette valeur afin d'éviter des tests redondants. On réduit ainsi la complexité en $\mathcal{O}(\sqrt{p})$.

En combinant les deux règles, on peut avoir une complexité en $\mathcal{O}\left(\frac{\sqrt{p}}{2}\right)$.

4) Quelle est la complexité de l'algorithme qui consisterait à utiliser l'algorithme de la question 3 pour déterminer l'ensemble des nombres premiers inférieurs à p.

Notre algorithme précédent à une complexité de $\mathcal{O}\left(\frac{\sqrt{p}}{2}\right)$ pour tester si un nombre donné p est premier. Si on doit effectuer ce même test pour tous les entiers qui lui sont inférieurs, notre complexité globale est de l'ordre de $\mathcal{O}\left(\frac{p\sqrt{p}}{2}\right)$.

5) Le crible d'Eratosthène est un algorithme plus efficace pour résoudre le problème de la question 4 : il construit la liste de tous les entiers inférieurs à p, puis élimine tous les multiples de 2, puis tous les multiples de 3, puis tous les multiples du premier nombre non encore éliminé et ainsi de suite jusqu'à ce que le premier nombre non éliminé soit supérieur à $\sqrt{p}$. Justifier son fonctionnement, écrire cet algorithme en méta-langage algorithmique (de manière itérative ou de manière récursive), puis évaluer sa complexité.

Imaginons que l'on dispose de la liste des entiers allant de 2 à p sotckés dans une liste représentant les candidats potentiels au fait d'être un nombre premier. Le premier

élément de la liste : 2, est un nombre premier, donc tous les multiples de 2 ne sont pas des nombres premiers on peut donc les supprimer de la liste actuelle. Le deuxième élément de cette nouvelle liste est alors un nombre premier. On supprimera de cette liste tous les éléments qui lui sont multiples pour obtenir une nouvelle liste. On procède de la même manière avec le troisième élément de la liste nouvellement créée et ainsi de suite, jusqu'à ce que le nouvel élément d'une liste considérée soit strictement plus grand que $\sqrt{p}$.

6) Un nombre parfait est un nombre égal à la somme de ses diviseurs stricts (1 compris, mais lui-même exclu). Proposez un algorithme, pour trouver tous les nombres parfaits inférieurs à n et évaluez sa complexité.

On peut commencer par donner des exemples de tels nombres comme 6 et 28. En effet, nous avons

$$6 = 1 \times 2 \times 3 = 1 + 2 + 3$$

et

$$28 = 7 \times 2^2 \times 1 = 14 + 7 + 4 + 2 + 1.$$

On vous laissera devinier le suivant, attention, ils deviennent rapidement très grands ! Mais essayons de donner un pseudo-code de la procédure. Celui ci devra déterminer l'ensemble des diviseurs de l'entier étudié ainsi que la somme de ces diviseurs.

Ainsi, étant donné un entier n, nous devons tester tous les entiers de 2 à $\sqrt{n}$ divisent n, ce qui fait une première procédure de complexité $\mathcal{O}\left(\frac{\sqrt{n}}{2}\right)$ en nombre d'opérations. Ensuite, si un tel entier k existe dans cet intervalle, alors l'entier n/k divise aussi n, on a donc une complexité globale de $\mathcal{O}\left(\sqrt{n}\right)$ à cette étape.

Nous avons maintenant extrait tous les diviseurs de n dont il faut faire la somme (qui implique au plus $2\sqrt{n}$ diviseurs et tester si cette dernière est égale à n (coût de 1). Cette étape a donc une complexité en $\mathcal{O}\left(2\sqrt{n}+1\right)$.

Ainsi le processus global aurait une complexité en $\mathcal{O}(3\sqrt{n}+1)$. Si nous devions ensuite répéter le processus pour tous les entiers inférieurs à n, nous avons une complexité global en $\mathcal{O}(3n\sqrt{n}+n) = \mathcal{O}(3n\sqrt{n})$.

7) Des nombres amis sont deux nombres dont chacun est égal à la somme des diviseurs stricts de l'autre. Proposez un algorithme, pour trouver tous les couples nombres amis inférieurs à n et évaluez sa complexité.

Regardons un petit exemple. Les entiers 220 et 284 sont des nombres amis. Ecrivons la liste des diviseurs pour s'en convaincre, on pourra par exemple parcourir la liste des entiers de 2 à $\sqrt{220}$ pour déterminer l'ensemble des diviseurs de 200:

$$220 : 1, 2, 4, 5, 10, 11, 20, 22, 44, 55, 110,$$

dont la somme est égale à 284. De même pour le nombre 284

$$220 : 1, 2, 4, 71, 142,$$

dont la somme est bien égale à 220.

Une procédure pour détecter les nombres amis consisterait à démarrer de façon analogue à celle utilisée pour détecter les nombres parfaits. Dans le cas où n n'est pas parfait, il convient de calculer la somme des diviseurs de la somme des diviseurs de n et de tester si cette somme est égale à n.