Un algorithme pour la partie d'un entier première à un autre

Jean-François Burnol
4 avril 2018

On utilise parfois la construction suivante. Soit A et B deux entiers strictement positifs et $M = \mathrm{PPCM}(A, B)$. On peut écrire M sous la forme $M = A_1 B_1$ avec $A_1 \mid A$, $B_1 \mid B$, et $\mathrm{PGCD}(A_1, B_1) = 1$. Par exemple il suffit de prendre pour A_1 le produit indicé par les nombres premiers :

$$A_1 = \prod_{\substack{p \mid M \\ \mathrm{ord}_p(M)=\mathrm{ord}_p(A)}} p^{\mathrm{ord}_p(A)}$$

Si l'on pose $X = M/A$, alors le A_1 ci-dessus est le plus grand diviseur de A premier avec X.

D'où ce problème :

Trouver un algorithme efficace qui étant donnés A *et* X *renvoie le plus grand diviseur* A_1 *de* A *qui est premier avec* X.

Si $X = p$ est un nombre premier suffit de diviser A par X jusqu'à ce ne soit plus possible. Mais déjà si $X = p^3$ par exemple il faut réfléchir plus. Deux idées viennent à l'esprit :

1. calculer les $D_n = \mathrm{PGCD}(A, X^n)$, critère d'arrêt $D_{n+1} = D_n$, renvoyer A/D_n,
2. ou remplacer A par $A/\mathrm{PGCD}(A, X)$ jusqu'à ce que $\mathrm{PGCD}(A, X) = 1$.

Les deux approches sont correctes. Elles évitent de factoriser A et/ou X car on sait que factoriser est coûteux.

On peut améliorer la première en faisant tous les calculs modulo A, car ça ne change pas les D_n (qui à propos forment une suite croissante pour la divisibilité, ultimement stationnaire). Mais une fois qu'on calcule modulo A, autant remplacer à chaque fois X par son carré, ça ira plus vite.

Finalement on peut combiner les deux approches, d'où l'algorithme que je propose.

Je vous laisse une dernière chance d'y réfléchir. Au verso, une implémentation en Python.

```
def partiepremiere(A, X):
    """Renvoie A1 tel que A = A1 * A2 avec A1 le plus grand entier premier
    avec X qui divise A.

    Autrement dit A2 est la partie de A dont les facteurs premiers divisent X.

    :param int A, X: des entiers (strictement positifs).
    :rtype: int
    :return: le A1 décrit ci-dessus.

    Exemples :

    >>> partiepremiere(1000, 378)
    125
    >>> partiepremiere(378, 1000)
    189

    """
    D = pgcd(A, X)
    while D > 1:
        A = A // D
        X = X ** 2 % A
        D = pgcd(A, X)
    return A
```

Un exemple (cuisiné) :

```
>>> partiepremiere(209497687909679241537850000, 855623548458571629338624 0)
1276667
```

Dans cet exemple les facteurs premiers ont moins de six chiffres, donc l'approche par factorisation ne serait pas très coûteuse. Voici les factorisations (avec Maple).

```
> ifactors(209497687909679241537850000);
[1, [[2, 3], [5, 4], [7, 1], [11, 1], [19, 1], [29, 1], [331, 1], [66821, 3]]]

> ifactors(855623548458571629338624 0);
            [1, [[2, 14], [5, 1], [11, 1], [52183, 3], [66821, 1]]]

> ifactors(1276667);
                  [1, [[7, 1], [19, 1], [29, 1], [331, 1]]]
```