Introduction à l'algorithmique **Automne 2017**

IFT2125-6001 TA: Maëlle Zimmermann

Démonstration 8

1

Question: Soient $a, b \in \mathbb{N}$ et $d = pgcd(a, b)$.

1. Montrer qu'il existe $s, t \in \mathbb{Z}$ tels que $sa + tb = d$.
2. Donner un algorithme efficace afin de calculer s, t et d à partir de a et b. L'algorithme ne doit pas calculer d avant de calculer s et t.
3. Soient $a, b \in \mathbb{N}$ tels que $b > 1$ et $pgcd(a, b) = 1$. Donner un alqorithme efficace qui calcule $s \in \mathbb{Z}$ tel que $sa \bmod b = 1$.

Solution:

1. Supposons sans perte de généralité que $a \geq b$ et montrons la proposition par induction sur b.

   Cas de base: $b = 0$: Nous avons $1 \cdot a + 0 \cdot b = a = pgcd(a, b)$

   Etape d'induction: $b > 0$: Posons $w = a \mod b$. Notons que $w < b$, ainsi par hypothèse d'induction, il existe $s', t' \in \mathbb{Z}$ tels que $s'b + t'w = pgcd(b, w)$. Selon la preuve que l'algorithme d'Euclide marche, nous avons $pgcd(a, b) = pgcd(b, w)$. Ainsi $s'b + t'w = pgcd(a, b) = d$.

   Posons $s = t'$ et $t = s' - (a \div b)t'$. Nous obtenons:

$$\begin{aligned} sa + tb &= t'a + (s' - (a \div b)t')b \\ &= s'b + t'(a - (a \div b)b) \\ &= s'b + t'(a \mod b) \\ &= s'b + t'w \\ &= d. \end{aligned}$$

2. Nous obtenons directement un algorithme récursif à partir de la preuve précédente:

```python
def pgcd_etendu(a, b):
    if b == 0:
        return (1, 0, a)
    else:
        (s, t, d) = pgcd_etendu(b, a % b)
    return (t, s - (a//b)*t, d)
```

3. Il suffit de calculer $s, t \in \mathbb{Z}$ tels que $sa + tb = pgcd(a, b)$ grâce à l'algorithme précédent. Nous obtenons:

$$\begin{aligned} sa \mod b &= (sa + tb) \mod b & \text{car } tb \mod b = 0 \\ &= pgcd(a, b) \mod b & \text{par déf. de } s, t \\ &= 1 \mod b & \text{par hypothèse} \\ &= 1 & \text{car } b > 1. \end{aligned}$$

2

Question: Donner un algorithme pour la multiplications de large entiers dont le temps est plus rapide que $n^{log_2 3}$.

Solution: Nous avons vu dans *B&B* chapitre 7 un algorithme qui permet de multiplier deux entiers de longueur n dans un temps dans $O(n^{log_2 3})$. L'idée était de remplacer la multiplication à calculer par trois multiplications de taille réduite de moitié.

Suivant la même idée, nous allons donner un algorithme où les entiers à multiplier sont séparés en trois plutôt qu'en deux, et qui permet d'obtenir le produit à calculer à partir de 5 multiplications de taille environ $n/3$ (au lieu de 9). On va montrer que le temps de calcul se réduit alors à $O(n^{log_3 5})$. On illustre la méthode avec un exemple:

Soit $a = 123456$ et $b = 135790$ deux entiers de taille $n = 6$. Nous divisons en 3 chaque entier de façon à obtenir: $s = 12, t = 34, u = 56, v = 13, w = 57, x = 90$. En posant ensuite

$$\begin{aligned} a' &= sv \\ \beta &= (s + t + u)(v + w + x) \\ \gamma &= (s - t + u)(v - w + x) \\ \delta &= (s + 2t + 4u)(v + 2w + 4x) \\ e' &= ux \end{aligned}$$

nous avons défini 5 multiplications d'entiers de taille un tiers de n, donc 2 ici. Il est possible de prouver que pour notre exemple le produit ab s'exprime ensuite comme:

$$ab = 10^8 a' + 10^6 b' + 10^4 c' + 10^2 d' + e'$$

où:

$$\begin{aligned} b' &= (-3a' + 6\beta - 2\gamma - \delta + 12e') \div 6 \\ c' &= (-2a' + \beta + \gamma + 12e') \div 2 \\ d' &= (3a' - 3\beta - \gamma + \delta - 12e') \div 6. \end{aligned}$$

Sur ce principe on peut construire l'algorithme récursif suivant:

```python
def produit(a, b):
  if a < 100 or b < 100:
    return a * b
  else:
    r = int(ceil(log10(max(a,b))/3))
    r4, r3, r2, r1 = 10 ** (4*r), 10 ** (3*r), 10 ** (2*r), 10 ** (r)

    s, t, u = a // r2, (a // r1) % r1, a % r1
    v, w, x = b // r2, (b // r1) % r1, b % r1

    aa = produit(s, u)
    beta = produit(s + t + u, v + w + x)
    gamma = produit(s - t + u, v - w + x)
    delta = produit(s + 2*t + 4*u, v + 2*w + 4*x)
    ee = produit(u, x)

    bb = (-3*aa + 6*beta - 2*gamma - delta + 12*ee) // 6
    cc = (-2*aa + beta +  gamma - 2*ee) // 2
    dd = ( 3*aa - 3*beta - gamma + delta - 12*ee) // 6

  return r4*aa + r3*bb + r2*cc + r1*dd + ee
```

Pour analyser le temps $t(m)$ de l'algorithme en fonction du nombre m de chiffres décimaux de a et b, on peut faire les hypothèses suivantes:

- $t(m)$ non décroissante
- la somme d'un nombre constant d'entiers de m chiffres possède m chiffres.

De plus on peut supposer que les opérations suivantes s'effectuent en un temps dans $O(m)$:

- le calcul de `r,r1,r2,r3,r4`

- l'addition de deux nombres de $O(m)$ chiffres
- la multiplication ou division d'un nombre de $O(m)$ chiffres par une constante ou puissance de 10

On commence par montrer que t, u, w, x ont au plus $\lceil m/3 \rceil$ chiffres et s, v en ont au plus $\lceil m/3 \rceil + 1$.

Si on pose $l = \log_{10} \max(a, b)$, le nombre de chiffres décimaux de $\max(a, b)$ est égal à $m = \lfloor l \rfloor + 1$. L'algorithme définit $r = \lceil l/3 \rceil$. Par définition de l'algorithme t, u, w, x ont au plus r chiffres: en effet ils sont le résultat d'une opération modulo 10^r. Comme s, v comprennent les chiffres restant, ils ont au plus $m - 2r$ chiffres. Or,

Montrons que $r \leq \lceil m/3 \rceil$:

$$\begin{aligned} l \leq \lfloor l \rfloor + 1 &\Rightarrow l/3 \leq (\lfloor l \rfloor + 1)/3 \\ &\Rightarrow \lceil l/3 \rceil \leq \lceil (\lfloor l \rfloor + 1)/3 \rceil \\ &\Rightarrow r \leq \lceil m/3 \rceil \end{aligned}$$

Montrons que $m - 2r \leq r + 1$:

$$\begin{aligned} m - 2r \leq r + 1 &\iff \lfloor l \rfloor + 1 - 2\lceil l/3 \rceil \leq \lceil l/3 \rceil + 1 \\ &\iff \lfloor l \rfloor \leq 3\lceil l/3 \rceil \quad \text{(ce qui est vrai)} \end{aligned}$$

Ainsi chaque nombre passé en argument dans le calcul récursif de `aa,beta,gamma,delta,ee` a aussi au plus $\lceil m/3 \rceil + 1$ chiffres. En effet c'est le résultat d'un nombre constant d'additions de t, u, w, x, s, v, et par hypothèse le nombre de chiffres n'augmente pas. De plus, les autres lignes de la fonction produit prennent un temps dans $O(m)$. Ainsi,

$$t(m) \leq 5t(\lceil m/3 \rceil + 1) + f(m),$$

où $f \in O(m)$.

A cause du $+1$ il impossible d'appliquer directement les théorèmes vus en classe pour déterminer l'order de t. Cependant comme t est non décroissante, nous pouvons procéder de façon similaire à l'exercice 3 de la démonstration 4.

Plus précisément on pose $s(m) = t(m + 3)$, alors:

$$\begin{aligned} s(m) &= t(m + 3) \\ &\leq 5t(\lceil (m + 3)/3 \rceil + 1) + f(m + 3) \\ &= 5t(\lceil m/3 \rceil + 2) + f(m + 3) \\ &\leq 5t(\lceil m/3 \rceil + 3) + f(m + 3) \\ &= 5s(\lceil m/3 \rceil) + f(m + 3). \end{aligned}$$

Nous pouvons appliquer alors le théorème sur les récurrences vu en classe avec $a = 5, b = 3$, et $f \in O(m)$. En choisissant $\epsilon = 0.1$, on obtient $s \in O(m^{log_3 5})$. Puisque $t(m) \leq t(m+3) = s(m)$, nous obtenons $t \in O(m^{log_3 5}) = O(m^{1.4649})$.