Introduction à l'algorithmique **Hiver 2018**

IFT2125-6001 TA: Stéphanie Larocque

Démonstration 7

À partir des corrigés de Maelle Zimmermann

1

Question: Supposons que nous avons accès aux algorithmes suivants:

- `mult_k1`: multiplie un polynôme de degré k avec un polynôme de degré 1 en un temps $O(k)$,
- `mult_kk`: multiplie deux polynômes de degré k en un temps $O(k \log k)$.

Soient $z_1, ..., z_d \in \mathbb{Z}$. Donner un algorithme efficace qui calcule l'unique polynôme

$$p(n) = a_0 + a_1 n + ... + a_d n^d$$

tel que $a_d = 1$ et $p(z_1) = ... = p(z_d) = 0$. Notez que nous représenterons un polynôme $a_0 + a_1 n + ... + a_d n^d$ par le tableau $[a_0, a_1, ..., a_d]$. Analyser l'efficacité de l'algorithme.

Solution: Il suffit de calculer le polynôme

$$p(n) = (n - z_1)(n - z_2)(n - z_3)...(n - z_d)$$

. Ce polynôme a effectivement $z_1, z_2, \dots z_d$ comme racines et son coefficient $a_d = 1$, par construction. Il s'agit donc de l'unique polynôme respectant les conditions demandées. La question est donc de donner un algorithme efficace pour calculer ce produit.

L'idée est de séparer le produit en 2 sous-parties sur lesquelles on effectuera l'appel récursif. Au besoin (si d est impair), on multipliera le polynôme obtenu par $(n - z_d)$. Notons qu'on doit séparer le polynôme en 2 parties de même degré, compte tenu que nous n'avons accès qu'à un algorithme pour multiplier 2 polynômes de même taille.

Voici un tel algorithme:

```
def zeros(Z=[z1,z2,z3,...,zd]):
  if len(Z) == 0:
```

```
    return [1]
elif len(Z) == 1:
    return [-Z[0], 1]      #Retourne le polynome p(n) = n-z1
else:
    m = len(Z) // 2        #Afin d'avoir 2 polynomes de meme degre
    p1 = zeros(Z[:m])      #m premieres racines
    p2 = zeros(Z[m:2*m])   #m racines suivantes
    p = mult_kk(p1,p2)
    if len(Z)%2==1:        #nombre impair de zeros
        r = [-Z[-1], 1]    #p(n) = n - zd
        p = mult_k1(p,r)
    return p
```

Le temps d'exécution de `zeros` est décrit par la récurrence suivante:

$$t(d) = \begin{cases} 1 & \text{si } d \leq 1, \\ 2t(\lfloor \frac{d}{2} \rfloor) + f(\lfloor \frac{d}{2} \rfloor) & \text{si } d > 1 \text{ et est pair}, \\ 2t(\lfloor \frac{d}{2} \rfloor) + t(1) + f(\lfloor \frac{d}{2} \rfloor) + g(d-1) & \text{si } d > 1 \text{ et est impair} \end{cases}$$

où $f(d) \in O(d \log d)$ correspond au temps d'exécution de `mult_kk` et $g(d) \in O(d)$ correspond au temps d'exécution de `mult_k1`.

Ainsi,

$$t(d) \in \begin{cases} 1 & \text{si } d \leq 1, \\ 2t(\lfloor \frac{d}{2} \rfloor) + O(d \log d) & \text{si } d > 1. \end{cases}$$

Appliquons le théorèmes sur les récurrences vu en classe. Nous avons $a = 2, b = 2$ et $f(d) = d \log d$. Posons $\epsilon = 1$. Puisque $f(d) \in O(d \log d) = O(d^{\log_b a}(\log d)^{\epsilon})$, nous concluons que $t(d) \in O(d^{\log_b a}(\log d)^{\epsilon+1}) = O(d(\log d)^2)$.

2

Question: Un circuit $n - tally$ est un circuit qui prend n bits en entrée et produit $1 + \lfloor \log n \rfloor$ bits en sortie. Il compte en binaire le nombre de bits égaux à 1 dans l'entrée. Par exemple, si $n = 9$ et l'entrée est 011001011, alors il y a 5 bits égaux à 1, et la sortie est 0101 (5 en binaire).

Un $(i, j) - adder$ est un circuit qui prend un nombre m de i bits et un nombre n de j bits en entrée. Il calcule $m + n$ en binaire sur $1 + \max(i, j)$ bits de sortie. Par exemple, si l'entrée est $m = 101$ et $n = 10111$ ($i = 3$, $j = 5$), la sortie est la somme des deux nombres, soit 011100.

Il est toujours possible de construire un $(i, j) - adder$ à partir d'exactement $\max(i, j)$ $3 - tallies$. En effet, additionner $m + n$ revient à compter pour chaque position k le

nombre de bits égaux à 1 parmi le $k$ème bit de m, le $k$ème bit de n, et l'éventuel bit de retenue. Comme le calcul doit être fait pour $\max(i,j)$ positions k nous avons besoin de $\max(i,j)$ $3-tallies$.

1. Utiliser des $3-tallies$ et des $(i,j)-adders$ afin de construire un $n-tally$ efficace.
2. Donner une récurrence (avec condition initiales) qui décrit le nombre de $3-tallies$ nécessaires pour construire le $n-tally$, incluant les $3-tallies$ qui font partie des $(i,j)-adders$.
3. Résoudre la récurrence exactement.

Solution:

1. On suppose l'accès aux algorithmes `3_tally` et `ij_adder`. On construit un `n_tally` de façon récursive de la façon suivante:

```
def n_tally(x):
    n = len(x)
    if 1<=x<=3:
        return 3_tally(x)
    else:
        m = n//2                # m = floor(n/2)
        x1 = n_tally(x[:m])     # taille floor(n/2)
        x2 = n_tally(x[m:])     # taille ceil(n/2)
        x3 = ij_adder(x1, x2)
        return x3
```

   Le cas de base est lorsque $1 \leq n \leq 3$, car on peut directement utiliser un `3_tally` dans ce cas. Dans les autres cas, on sépare l'entrée en 2 (respectivement de tailles $\left\lfloor \frac{n}{2} \right\rfloor$ et $\left\lceil \frac{n}{2} \right\rceil$), chacun comptant le nombre de bits "1" dans leur entrée. Le résultat de ces deux $tallies$ est sommé par un $(i,j)-adder$ où $i = 1 + \lfloor \log \lfloor n/2 \rfloor \rfloor$ et $j = 1 + \lfloor \log \lceil n/2 \rceil \rfloor$.

2. Soit $t(n)$ le nombre de $3-tallies$ utilisés afin de construire un $n-tally$ dans la construction donnée en (1). Lorsque $1 \leq n \leq 3$, un seul $3-tally$ est utilisé. Lorsque $n > 3$ le nombre de $3-tallies$ utilisés est $t(\lceil n/2 \rceil) + t(\lfloor n/2 \rfloor)$, plus le nombre de $3-tallies$ utilisés afin de construire le $(i,j)-adder$, c'est-à-dire $\max(i,j)$. Comme $i = 1 + \lfloor \log \lceil n/2 \rceil \rfloor$ et $j = 1 + \lfloor \log \lfloor n/2 \rfloor \rfloor$, nous obtenons

$$t(n) = \begin{cases} 1 & \text{si } 1 \leq n \leq 3, \\ \underbrace{t(\left\lfloor \frac{n}{2} \right\rfloor)}_{\lfloor \frac{n}{2} \rfloor - tally} + \underbrace{t(\left\lceil \frac{n}{2} \right\rceil)}_{\lceil \frac{n}{2} \rceil - tally} + \underbrace{1 + \left\lfloor \log \left\lceil \frac{n}{2} \right\rceil \right\rfloor}_{(i,j)-adder} & \text{si } n > 3 \end{cases} \tag{1}$$

3. Posons $s_i = t(2^i)$, alors nous avons

$$s_i = \begin{cases} 1 & \text{si } 0 \leq i \leq 1, \\ 2s_{i-1} + i & \text{si } i > 1 \end{cases}$$

Le polynôme charactéristique de la récurrence s est $p(x) = (x-2)(x-1)^2$ et ainsi $s_i = c_1 2^i + c_2 + c_3 i$. En résolvant le système (pour $i = 0, 1, 2$)

$$\begin{array}{rrcrcrcl} s_0 = & c_1 & + & c_2 & + & & = & 1 \\ s_1 = & 2c_1 & + & c_2 & + & c_3 & = & 1 \\ s_2 = & 4c_1 & + & c_2 & + & 2c_3 & = & 4 \end{array}$$

nous obtenons $c_1 = 3, c_2 = -2$ et $c_3 = -3$. Ainsi, $s_i = 3 \cdot 2^i - 3i - 2$ et donc $t(n) = s_{\log n} = 3n - 3\log n - 2$ lorsque n est une puissance de 2.

Nous avons donc $t(n) \in \Theta(n : n \text{ est une puissance de 2})$. Puisque $t(n)$ est éventuellement non décroissante (on peut le démontrer), nous concluons par la règle de l'harmonie que $t(n) \in \Theta(n)$.

Alternativement, si on cherche simplement à obtenir l'ordre de t et non sa forme exacte, on peut utiliser le théorème vu en classe (premier cas). Nous avons $a = 2, b = 2$, et $f(n) = \log(n) \in O(n^{\log 2 - \epsilon})$ en prenant n'importe quel ϵ suffisamment petit (par exemple 0.1). On en conclut également que $t(n) \in \Theta(n : n \text{ est une puissance de 2})$.

3

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. On suppose que la propriété d'Euclide

$$pgcd(a, b) = pgcd(b, a \mod b)$$

est vraie pour le moment (preuve plus bas).

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: $b > 0$:

L'hypothèse d'induction est la suivante : Si on a deux nombres a', b' tels que (SPDG) $a' \geq b'$ et que $b' < b$, alors il existe $s', t' \in \mathbb{Z}$ tels que $s'a' + t'b' = pgcd(a', b')$.

On veut montrer que, pour $a \geq b$, il existe $s, t \in \mathbb{Z}$ tels que

$$sa + tb = pgcd(a, b).$$

Par hypothèse d'induction, remarquons que, pour les nombres b et $(a \mod b)$, il existe $s', t' \in \mathbb{Z}$ tels que

$$s'b + t'(a \mod b) = pgcd(b, a \mod b),$$

car $(a \mod b) < b$. Posons maintenant $s = t'$ et $t = s' - (a//b)t'$.

Nous obtenons:

$$\begin{aligned}
sa + tb &= t'a + (s' - (a//b)t')b && \text{par définition de } s \text{ et } t \\
&= s'b + t'(a - (a//b)b) && \text{réarrangement des termes} \\
&= s'b + t'(a \mod b) && a - (a//b)b \text{ est le reste de la division de } a \text{ par } b \\
&= pgcd(b, a \mod b) && \text{par hypothèse d'induction} \\
&= pgcd(a, b) && \text{par propriété d'Euclide} \\
&= d && \text{par définition de } a \text{ et } b.
\end{aligned}$$

Ainsi, on a bel et bien montré que, pour toute paire d'entiers positifs a et b, il existe des entiers s, t tels que

$$sa + tb = d = pgcd(a, b).$$

■

Prouvons maintenant que la **propriété d'Euclide** est vraie :

$$pgcd(a, b) = pgcd(b, a \mod b).$$

Soit $d = pgcd(a, b)$ et $d' = pgcd(b, a \mod b)$. Par contradiction, supposons que $d \neq d'$. Il y a 2 choix:

(a) $d < d'$. Puisque $d' = pgcd(b, a \mod b)$

$$\begin{aligned}
&\Rightarrow d'|b \text{ et } d'|(a \mod b) && \text{par définition du pgcd}\\
&\Rightarrow d'|a && \text{car } a = kb + (a \mod b) \text{ pour un certain } k \in \mathbb{Z}\\
& && \text{et } d' \text{ doit diviser les 2 côtés de l'équation}\\
&\Rightarrow d'|a \text{ et } d'|b
\end{aligned}$$

On a donc que d' est un commun diviseur de a et b, strictement plus grand que le plus grand commun diviseur de a et b, soit d. Contradiction.

(b) $d > d'$. Puisque $d = pgcd(a, b)$

$$\begin{aligned}
&\Rightarrow d|a \text{ et } d|b && \text{par définition du pgcd}\\
&\Rightarrow d|(a \mod b) && \text{car } a - kb = (a \mod b) \text{ pour un certain } k \in \mathbb{Z}\\
& && \text{et } d \text{ doit diviser les 2 côtés de l'équation}\\
&\Rightarrow d|b \text{ et } d|(a \mod b)
\end{aligned}$$

On a donc que d est un commun diviseur de b et $(a \mod b)$, strictement plus grand que le plus grand commun diviseur de b et $(a \mod b)$, soit d'. Contradiction.

Dans les 2 cas, on arrive à une contradiction. De ce fait, on a que $d = d'$ et

$$pgcd(a, b) = pgcd(b, a \mod b)$$

■

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

```
def pgcd_etendu(a, b):
   if b == 0:
      return (1, 0, a) #Cas de base de l'induction
   else:               #Etape d'induction avec b et (a modb)
      (s1, t1, d) = pgcd_etendu(b, a % b)
      s = t1
      t = s1 - (a//b)*t1
   return (s, 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:

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