Introduction à l'algorithmique **Hiver 2018**

IFT2125-6001 TA: Stéphanie Larocque

Démonstration 7

À partir des corrigés de Maelle Zimmermann

1

Question: Suppose we have access to the following algorithms:

- `mult_k1`: multiply a polynomial of degree k with a polynomial of degree 1 in a time $O(k)$,
- `mult_kk`: multiply two polynomials of degree k in a time $O(k \log k)$.

Let $z_1, ..., z_d \in \mathbb{Z}$. Give an efficient algorithm that calculates the unique polynomial

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

such that $a_d = 1$ and $p(z_1) = ... = p(z_d) = 0$. Note that we will represent a polynomial $a_0+a_1n+...+a_dn^d$ by the array $[a_0, a_1, ..., a_d]$. Analyze the effectiveness of the algorithm.

Solution: Just calculate the polynomial

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

. This polynomial actually has $z_1, z_2, \ldots z_d$ as roots and its coefficient $a_d = 1$, by construction. It is therefore the only polynomial respecting the conditions requested. The question is to give an efficient algorithm to calculate this product.

The idea is to separate the product into 2 subparts on which the recursive call will be made. If necessary (if d is odd), multiply the polynomial obtained by $(n - z_d)$. Note that we must separate the polynomial into 2 parts of the same degree, given that we only have access to an algorithm to multiply 2 polynomials of the same size.

Voici un tel algorithme:

```python
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
```

The execution time of |zeros | is described by the following recurrence:

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

where $f(d) \in O(d \log d)$ is the execution time of |mult$_k k|and$ g (d) $\in O(d)$ is the execution time of |mult$_k$1|.

Ainsi,

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

Let's apply the theorems on recurrences seen in class. We have $a = 2, b = 2$ and $f(d) = d \log d$. Let's put $\epsilon = 1$. Since $f(d) \in O(d \log d) = O(d^{\log_b a} (\log d)^{\epsilon})$, we conclude that $t(d) \in O(d^{\log_b a} (\log d)^{\epsilon+1}) = O(d(\log d)^2)$.

2

Question: An $n - tally$ circuit is a circuit that takes n bits as input and produces $1 + \lfloor \log n \rfloor$ bits as output. It counts in binary the number of bits equal to 1 in the input. For example, if $n = 9$ and the entry is 011001011, then there are 5 bits equal to 1, and the output is 0101 (5 in binary).

A $(i, j) - adder$ is a circuit that takes a number m from i bits and a number n from j bits input. It calculates $m + n$ in binary on $1 + \max(i, j)$ output bits. For example, if the entry is $m = 101$ and $n = 10111$ ($i = 3$, $j = 5$), the exit is the sum of the two numbers, 011100.

It is always possible to construct a $(i,j)-adder$ from exactly $\max(i,j)$ $3-tallies$. In fact, adding $m+n$ amounts to counting for each position k the number of bits equal to 1 among the k th bit of m, the k th bit of n, and the possible retaining bit. Since the calculation must be done for $\max(i,j)$ k positions we need $\max(i,j)$ $3-$ tallies.

1. Use $3-tallies$ and $(i,j)-adders$ to build an effective $n-tally$.
2. Give a recurrence (with initial condition) that describes the number of $3-tallies$ needed to build the $n-tally$, including the $3-tallies$ that are part of the $(i,j)-adders$.
3. Solve the recurrence exactly.

Solution:

1. Assume access to algorithms $|3_tally|and|ij_adder|.Webuilda|n_tally|recursivelyasfollows:$

```
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
```

   The basic case is when $1 \leq n \leq 3$, because we can directly use a $|3_tally|inthatcase.Inothercases,theen$ $\left\lfloor \frac{n}{2} \right\rfloor$ and $\left\lceil \frac{n}{2} \right\rceil$), each counting the number of bits "1" in their input. The result of these two $tallies$ is summed by a $(i,j)-adder$ where $i = 1 + \lfloor \log \lfloor n/2 \rfloor \rfloor$ and $j = 1 + \lfloor \log \lceil n/2 \rceil \rfloor$.

2. Let $t(n)$ be the number of $3-tallies$ used to construct a $n-tally$ in the construct given in (1). When $1 \leq n \leq 3$, only one $3-tally$ is used. When $n > 3$ the number of $3-tallies$ used is $t(\lceil n/2 \rceil) + t(\lfloor n/2 \rfloor)$, plus the number of $3-tallies$ used in order to build the $(i,j)-adder$, that is, $\max(i,j)$. Since $i = 1 + \lfloor \log \lceil n/2 \rceil \rfloor$ and $j = 1 + \lfloor \log \lfloor n/2 \rfloor \rfloor$, we get

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

3. Let's put $s_i = t(2^i)$, then we have

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

The characteristic polynomial of the recurrence s is $p(x) = (x-2)(x-1)^2$ and so $s_i = c_1 2^i + c_2 + c_3 i$. Solving the system (for $i = 0, 1, 2$)

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

we get $c_1 = 3, c_2 = -2$ and $c_3 = -3$. So $s_i = 3 \cdot 2^i - 3i - 2$ and so $t(n) = s_{\log n} = 3n - 3\log n - 2$ when n is a power of 2.

So we have $t(n) \in \Theta(n : n \text{is a power of} 2)$. Since $t(n)$ is possibly nondecreasing (we can prove it), we conclude by the rule of harmony that $t(n) \in \Theta(n)$.

Alternatively, if one simply seeks to obtain the order of t and not its exact form, one can use the theorem seen in class (first case). We have $a = 2, b = 2$, and $f(n) = \log(n) \in O(n^{\log 2 - \epsilon})$ taking any ϵ small enough (eg 0.1). We also conclude that $t(n) \in \Theta(n : n \text{is a power of} 2)$.

3

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

1. Show that there are $s, t \in \mathbb{Z}$ such that $sa + tb = d$.
2. Give an efficient algorithm to compute s, t and d from a and b. The algorithm should not calculate d before calculating s and t.
3. Let $a, b \in \mathbb{N}$ such that $b > 1$ and $pgcd(a, b) = 1$. Give an efficient alqorith that calculates $s \in \mathbb{Z}$ such that $sa \bmod b = 1$.

Solution:

1. It is assumed that the property of Euclid

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

is true for the moment (proof below).

Suppose without loss of generality that $a \geq b$ and show the induction proposition on b. <u>Base case: $b = 0$:</u> We have $1 \cdot a + 0 \cdot b = a = pgcd(a, b)$ <u>Induction step: $b > 0$:</u>
The induction hypothesis is: If we have two numbers a', b' such that (SPDG) $a' \geq b'$ and that $b' < b$, then there exists $s', t' \in \mathbb{Z}$ such that $is' + t'b' = pgcd(a', b')$.
We want to show that for $a \geq b$, there exists $s, t \in \mathbb{Z}$ such that

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

By induction hypothesis, note that, for the numbers b and $(a \mod b)$, there exist $s', t' \in \mathbb{Z}$ such that

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

because $(a \mod b) < b$. Now put $s = t'$ and $t = s' - (a//b)t'$.
Nous obtenons:

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

Thus, we have indeed shown that, for any pair of positive integers a and b, there exist integers s, t such that

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

■

Now prove that the **property of Euclid** is true:

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

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

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

We therefore have that is a common divisor of a and b, strictly greater than the greatest common divisor of a and b, that is d. Contradiction.

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

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

We therefore have that d is a common divisor of b and $(a \mathrm{mod} b)$, strictly greater than the greatest common divisor of b and $(a \mathrm{mod} b)$, which is . Contradiction.

In both cases, we come to a contradiction. As a result, we have $d = d'$ and

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

■

2. We directly obtain a recursive algorithm from the previous proof:

```
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. Just calculate $s, t \in \mathbb{Z}$ such that $sa + tb = pgcd(a, b)$ thanks to the previous algorithm. We are getting

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