

FEUILLE D'EXERCICES n° 2

Exercice 1 – [OPÉRATIONS SUR LES POLYNÔMES]

Soit A un anneau commutatif. Soient P et Q deux polynômes de $A[x]$, de degrés respectifs m et n . On note $P = \sum p_i x^i$ et $Q = \sum q_i x^i$, que l'on code par des listes $P = [p_0, \dots, p_m]$ et $Q = [q_0, \dots, q_n]$. On note $\text{cd}(Q) = q_n$ (c'est le coefficient dominant de Q).

On considère les algorithmes d'addition, multiplication et division euclidienne suivants.

Somme

Entrées : P, Q

Sortie : $S = P + Q$

1. $d \leftarrow \max(m, n)$, $k \leftarrow \min(m, n)$
2. Pour i de 0 à d :
3. Si $i \leq k$: $s_i \leftarrow p_i + q_i$
4. Sinon :
5. Si $m < n$: $s_i \leftarrow q_i$
6. Sinon : $s_i \leftarrow p_i$
7. Sortir $S = \sum_{i=0}^d s_i x^i$

Multiplication

Entrées : P, Q

Sortie : $M = PQ$

1. $M \leftarrow 0$
2. Pour i de 0 à m :
3. $M \leftarrow M + p_i x^i Q$
4. Sortir M

Division euclidienne

Entrées : P, Q (où $\text{cd}(Q)$ est inversible)

Sortie : le quotient D et le reste R de la division euclidienne de P par Q

1. $R \leftarrow P$
2. $M \leftarrow \text{cd}(Q)^{-1}$
3. Pour i de 0 à $m - n$: $d_i \leftarrow 0$
4. Tant que $\deg R \geq \deg Q$:
5. $k \leftarrow \deg R - \deg Q$
6. $d_k \leftarrow \text{cd}(R)M$
7. $R \leftarrow R - d_k x^k Q$
8. Sortir $D = \sum_{i=0}^{m-n} d_i x^i$, R

- 1) Exécuter ces algorithmes à la main sur les polynômes $P = x^3 + x + 2$ et $Q = 2x - 1 \in \mathbb{Q}[x]$
- 2) Évaluer le nombre d'opérations nécessaires à l'exécution de chacun de ces algorithmes.
- 3) Démontrer l'algorithme de division euclidienne.
- 4) Soit q une puissance d'un nombre premier. Montrer que l'on peut effectuer la multiplication dans \mathbb{F}_q avec une complexité binaire de $O((\log q)^2)$.

Exercice 2 – [ÉVALUATION D'UN POLYNÔME EN UN POINT]

Soit A un anneau. Soient $P \in A[x]$ et $a \in A$.

- 1) Pour calculer $P(a)$, on peut mettre les puissances successives de a dans une liste l et additionner les $p_i l[i]$ (où les p_i sont les coefficients de P). Écrire cet algorithme et évaluer le nombre d'additions et de multiplications nécessaires à son exécution.
- 2) On peut aussi calculer $p_{m-1} + ap_m$, puis $a(p_{m-1} + ap_m)$, puis $p_{m-2} + a(p_{m-1} + ap_m)$ etc. Autrement dit, on peut calculer

$$P(a) = p_0 + a(p_1 + a(p_2 + a(p_3 + \cdots + a(p_{m-2} + a(p_{m-1} + ap_m)) \dots)))$$

où $m = \deg P$. C'est l'algorithme de Horner. Écrire cet algorithme et évaluer le nombre d'additions et de multiplications nécessaires à son exécution.

Exercice 3 – [DIVISION DANS \mathbb{N}] On rappelle l'algorithme de division des entiers donné en cours.

Division binaire.

Entrée. a, b : deux entiers naturels tels que $b \neq 0$.

Sortie. q, r : quotient et reste de la division euclidienne de a par b .

$a' \leftarrow a, s \leftarrow s(a), t \leftarrow s(b)$.

Pour i de 0 à $s - t$ faire $q_i = 0$.

Tant que $a' \geq b$ faire :

$$b' \leftarrow 2^{s-t} b$$

Si $b' \leq a'$ faire :

$$q_{s-t} \leftarrow 1$$

$$a' \leftarrow a' - b'$$

Sinon faire :

$$q_{s-t-1} \leftarrow 1$$

$$b' = b'/2$$

$$a' \leftarrow a' - b'$$

$$s \leftarrow s(a')$$

Sortir $q = \sum_{i=0}^{s(a)-s(b)} q_i 2^i, r = a'$.

Exécuter cet algorithme sur les entiers $a = 23$ et $b = 4$ puis sur $a = 23$ et $b = 7$.

Exercice 4 – [DIVISION DANS \mathbb{N} : PREUVE DE L'ALGORITHME]

Dans cet exercice, on donne la preuve de l'algorithme de division euclidienne binaire rappelé dans l'exercice 3.

1) Soit k le plus grand entier tel que $a \geq 2^k b$. Montrer que

$$k \in \{s(a) - s(b) - 1, s(a) - s(b)\}.$$

2) Montrer que $s(q) = k + 1$.

On pose $Q_0 = 0$, $a'_0 = a$, $s_0 = s(a'_0) = s(a)$, et on note a'_l , s_l , k_l et Q_l les valeurs respectives de a' , s , k et q après la l -ème exécution de la boucle "Tant que".

3) Montrer que la suite (k_l) est strictement décroissante.

4) En déduire que pour tout l ,

$$Q_l = \sum_{i=1}^l 2^{k_i}$$

5) Montrer que si $a = bQ_{l-1} + a'_{l-1}$, alors $a = bQ_l + a'_l$.

6) Montrer que l'algorithme effectue la division euclidienne de a par b .

Exercice 5 – [FIBONACCI]

Soit `Fib` la procédure définie récursivement par le code Sage suivant.

```
def Fib(n):
    if n<=1:
        return n
    else:
        return Fib(n-1)+Fib(n-2)
```

1) Que calcule `Fib` ?

2) Montrer que pour tout n , on a

$$\text{Fib}(n) = \frac{1}{\sqrt{5}}(\Phi^n - \Phi'^n),$$

où $\Phi = (1 + \sqrt{5})/2$ est le nombre d'or et où $\Phi' = (1 - \sqrt{5})/2$.

3) Soit c_n le nombre d'additions effectuées pour calculer `Fib`(n), montrer que $c_n = \text{Fib}(n+1) - 1$. En déduire que la complexité algébrique de `Fib` est exponentielle.

4) Proposer pour le calcul de `Fib`(n) un algorithme qui demande $O(n)$ opérations dans \mathbb{N} .

Exercice 6 – [REPRODUCTION DES LAPINS]

On s'intéresse à l'évolution d'une population de lapins, suivant le modèle (très schématique) expliqué ci-dessous.

Chaque mois, tout couple de lapins donne naissance à un couple de lapereaux. Chaque lapereau met un mois avant de devenir lapin et pouvoir procréer.

Le premier janvier, un couple de lapins est réuni. On note $N_1 = 1$ le nombre de couples de lapins présents ce jour là. Le premier février, il donne naissance à un couple de lapereaux. Le nombre de couples de lapins est encore $1 = N_2$. Le

premier mars, les lapereaux ont grandi. Le nombre de couples de lapins est donc $N_3 = 2$. Chacun de ces couples donne naissance à un couple de lapereaux ... Et cela continue ainsi tout au long de l'année, et des années qui suivent.

Montrer que les N_i sont les nombres de Fibonacci.

Exercice 7 – [OPÉRATIONS EN BINAIRE SUR MACHINE]

On demande ici d'écrire sur sage les opérations dans \mathbb{N} .

Les entiers sont codés par des liste de 0 et 1. La liste l de longueur s représente l'entier

$$\sum_{i=0}^{s-1} l[i]2^i$$

Pour obtenir une telle liste pour un entier a , on peut utiliser la commande `a.bits()`.

1) Écrire une fonction `PlusBit` d'addition sur un bit. Écrire une fonction `PlusBinaire` d'addition binaire.

2) Écrire une fonction `PlusGrand(a, b)` qui rend `true` si $a \geq b$ et `false` sinon (où a et b sont écrits en binaire).

Écrire une fonction `Decale(a, k)` qui multiplie a par 2^k .

Écrire une fonction `Nettoie` qui enlève les 0 superflus d'une liste.

3) Écrire une fonction `MoinsBit` de soustraction sur un bit.

Écrire une fonction `MoinsBinaire` de soustraction binaire.

4) Écrire une fonction `MultBinaire` de multiplication classique binaire dans \mathbb{N} .

5) Écrire une fonction `DivBinaire` de division classique binaire dans \mathbb{N} .