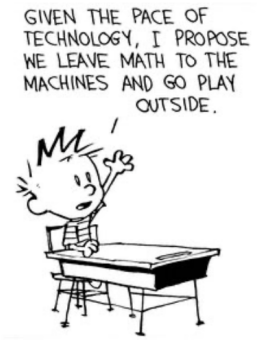
TP INFO 6 | Algorithmes de l'arithmétique entière

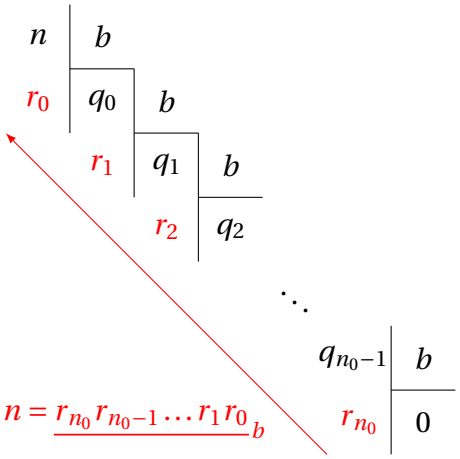
Dans cette séance, nous allons programmer en version itérative et récursive les principaux algorithmes de l'arithmétique des entiers relatifs.



1. Introduction

Nous rappelons ici sans justification les algorithmes étudiés dans le cours d'arithmétique entière :

Algorithme de décomposition en base b



On applique la méthode par divisions successives.

On calcule le couple quotient-reste (q_0, r_0) dans la division de n par b, puis le couple quotient-reste (q_1, r_1) dans la division de q_0 par b, puis le couple quotient-reste dans la division de q_1 par b, etc.

En notant q_{n_0} le premier quotient nul obtenu au cours de l'algorithme, on a

$$n = \underline{r_{n_0} r_{n_0-1} \dots r_1 r_0}_b$$

Calcul du plus grand commun diviseur par l'algorithme d'Euclide

Soit a et b deux entiers naturels tels que $b \neq 0$. On pose $r_0 = a$ et $r_1 = b$.

- Tant que $r_i \neq 0$, on calcule le reste r_{i+1} dans la division euclidienne de r_{i-1} par r_i.
- En notant n_0 le plus petit entier tel que $r_{n_0} = 0$, on a $a \wedge b = r_{n_0-1}$

On retiendra que *le pgcd est le dernier reste non nul dans l'algorithme d'Euclide.*

Cet algorithme admet une variante (algorithme d'Euclide étendu) permettant de trouver une relation de Bezout entre deux entiers. Par exemple, pour $a = 2309$ et $b = 2275$:

$$\begin{cases} 2309 = 2275 \times 1 + 34 \\ 2275 = 34 \times 66 + 31 \\ 34 = 31 \times 1 + 3 \\ 31 = 3 \times 10 + 1 \\ 3 = 1 \times 3 + 0 \end{cases} \xrightarrow[\text{d'Euclide étendu}]{\text{Algorithme}} \begin{cases} 34 = a - b \\ 31 = b - 66 \times (a-b) = -66a + 67b \\ 3 = a - b + 66a - 67b = 67a - 68b \\ 1 = -66a + 67b - 10(67a - 68b) = -736a + 747b \end{cases}$$

2. Exercices

1 — *L'algorithme soustractif de la division euclidienne*

Soit $b \in \mathbb{N}^*$. Pour effectuer la division euclidienne d'un entier naturel a par b, on peut procéder par soustractions successives : on retranche b à a tant que le résultat est supérieur ou égal à b.

1. Écrire une fonction `divE` prenant en argument un couple d'entiers naturels a et b (avec $b \neq 0$) et renvoyant la liste `[q,r]` du quotient et du reste dans la division euclidienne de a par b obtenue selon cet algorithme.
2. Écrire une fonction `divEr` fondée sur une version récursive de l'algorithme soustractif de division euclidienne.

2 — *Décomposition en base deux* f

Dans cet exemple, les chiffres en bases deux d'un entier n sera codé au moyen d'une liste. Par exemple `[1,0,0]` pour l'entier 4.

1. Écrire une fonction non récursive `cbin` prenant en argument un entier naturel et renvoyant sous la forme d'une liste ses chiffres en base deux.
2. Écrire une fonction récursive `cbinr` prenant en argument un entier naturel et renvoyant sous la forme d'une liste ses chiffres en base deux.

3 — *Le crible d'Eratosthène* f

Au III^{e} siècle avant Jesus-Christ, Ératosthène, mathématicien, astronome et philosophe invente une méthode efficace pour énumérer tous les nombres premiers inférieurs à un entier donné n. Cette méthode est connue sous le nom de crible d'Ératosthène, que nous allons décrire dans un langage moderne :

⇛ On initialise une liste `t` de n booléens à `True`. En k-ième case, le `True` s'interprète comme : *jusqu'ici, et jusqu'à preuve du contraire, k semble être premier.*

⇛ Tous les multiples stricts de 2 ne sont pas premiers : on passe à False les `t[2i]` pour $4 \leqslant 2i \leqslant n$.

⇛ Tous les multiples stricts de 3 ne sont pas premiers : on passe à `False` les `t[3i]` pour $9 \leqslant 3i \leqslant n$.

⇛ On constate que 4 n'est pas premier (`t[4]` a été mis à `False` lors du premier passage) : inutile de mettre à jour les multiples de 4 (ils l'ont déjà été).

⇛ Tous les multiples stricts de 5 ne sont pas premier : on passe à `False` les `t[5i]` pour $25 \leqslant 5i \leqslant n$.

⇛ Ainsi de suite.

À la fin de l'algorithme, pour $k \geqslant 2$, la k-ème cellule de `t` contient `True` si et seulement si k est premier.

Écrire la fonction `crible(n)` qui renvoie la liste des nombres premiers appartenant à $[\![1, n]\!]$. On s'attachera à respecter l'esprit de cette technique, et en particulier à éviter divisions et racines carrées.

4 — *L'algorithme naïf de factorisation* f

1. Écrire une fonction récursive `val` prenant en argument un entier naturel n non nul et un nombre premier p renvoyant la p-valuation $\nu_p(n)$ de l'entier n.
2. En déduire, au moyen de la fonction `crible`, une fonction `factorisationPremiere` prenant en argument un entier naturel n non nul et renvoyant sa factorisation première sous la forme d'un dictionnaire au format $p : \nu_p(n)$ avec p premier divisant n.

5 — *Versions itératives et récursives des algorithmes d'Euclide* ff

On revient aux algorithmes d'Euclide et d'Euclide étendu.

1. Écrire une fonction non récursive `pgcd` prenant en argument deux entiers naturels a et b et renvoyant leur pgcd selon l'algorithme d'Euclide.
2. Écrire une fonction récursive `pgcdBis` prenant en argument deux entiers naturels a et b et renvoyant leur pgcd selon l'algorithme d'Euclide.
3. Écrire une fonction non récursive `bezout` prenant en argument deux entiers naturels a et b et renvoyant une liste d'entiers `[u,v]` vérifiant $ua+bv = a\wedge b$ selon l'algorithme d'Euclide étendu.
4. Écrire une fonction récursive `bezoutBis` prenant en argument deux entiers naturels a et b et renvoyant une liste d'entiers `[u,v]` vérifiant $ua+bv = a\wedge b$ selon l'algorithme d'Euclide étendu.

6 — *Algorithme binaire de calcul du pgcd* ff

On considère l'algorithme suivant :

Algorithme 1 : Algorithme binaire de calcul du PGCD

Données : $a \in \mathbb{N}^*$; $b \in \mathbb{N}$ impair tel que $b \geqslant 3$ impair
Initialisation : u ← a, v ← b;
tant que $u \neq v$ **faire**
  si *u est pair* **alors**
    $u \leftarrow u//2$
  sinon
    si $u > v$ **alors**
      $u \leftarrow (u-v)//2$
    sinon
      $u, v \leftarrow (v-u)//2, u$
Renvoyer u

Écrire une fonction `pgcdBin` prenant en argument deux entiers naturels a et b et renvoyant leur pgcd selon cet algorithme.

3. Indications

1

Au 2., effectuer la première itération de l'algorithme sous-tractif puis les autres récursivement.

2

La division euclidienne de n par 2 s'écrit :

$$n = \underline{a_i \cdots a_1 a_0}_2 = 2 \times \underbrace{\underline{a_i \cdots a_1}_2}_{q} + \underbrace{a_0}_{r}$$

On peut en déduire une version récursive du codage en base deux.

3

Utiliser le slicing de Python afin d'alléger le code.

4

Il suffit de parcourir la liste des facteurs premiers potentiels de n et calculer la valuation correspondante.

5

Pour les algorithmes récursifs, on pourra effectuer la première étape de l'algorithme puis les autres récursivement.

6

Il faut se ramener au cas où b est impair au moyen d'une boucle préliminaire.