

TP8 : AUTOUR DES GROUPES ET DE L' ARITHMÉTIQUE

Ce TP aborde le traitement algorithmique de certaines questions dans les groupes et anneaux : dans un premier temps l'anneau des entiers $\mathbb{Z}/n\mathbb{Z}$ puis le groupe des permutations.

1 Arithmétique

L'objet de cette partie est l'implémentation de certains algorithmes vus en maths dans le cours d'algèbre, et plus précisément celui d'arithmétique des entiers.

1.1 Algorithme d'Euclide

Question 1.1 (PGCD) – Écrire une fonction `pgcd(a,b)` qui retourne le PGCD de deux éléments non tous deux nuls de $\mathbb{N}$. Démontrer sa correction (en dégageant l'invariant de boucle).

Écrire ensuite une version récursive `pgcdRec(a,b)`.

Question 1.2 (Algorithme d'Euclide étendu) – Cet algorithme s'appuie sur le précédent mais en donnant également les coefficients de Bézout, c'est à dire u et v tels que $au + bv = PGCD(a,b)$.

On donne la fonction `Bezout(a,b)` qui étant donné deux éléments de $\mathbb{N}^*$ retourne une liste `[u,v]` tel que $au + bv = PGCD(a,b)$:

```
def Bezout(a,b):# Euclide étendu
    r1,r2=a,b
    u1,v1=1,0
    u2,v2=0,1
    #Invariant : ??
    while r2!=0:
        q=r1//r2
        u,v=u2,v2 #temporaire
        u2,v2=u1-q*u2,v1-q*v2
        u1,v1=u,v
        r1,r2=r2,r1%r2
    return [u1,v1]
```

Démontrer la correction de cette fonction en en dégageant un invariant.

1.2 Arithmétique dans $\mathbb{Z}/n\mathbb{Z}$

L'anneau $(\mathbb{Z}/n\mathbb{Z}, +, \times)$ est formé par les classes d'équivalence des entiers modulo n (on notera $\overline{m}$ la classe d'un entier m), muni des opérations induites sur cet ensemble par l'addition et la multiplication dans $\mathbb{Z}$. En pratique, on assimilera un élément de $\mathbb{Z}/n\mathbb{Z}$ à son représentant compris entre 0 et $n - 1$; l'addition et la multiplication seront faites modulo n .

Question 1.3 (Indicatrice d'Euler) – L'indicatrice d'Euler ϕ est l'application qui à un entier $n \geq 1$ associe le nombre d'entiers naturels inférieurs ou égaux à n et premiers avec n .

On peut faire la remarque suivante : Soit m un entier inférieur à m ; alors m est premier avec n ssi il existe u et v tels que $un + vm = 1$; c'est à dire ssi il existe v tel que $vm = 1$ modulo n ; c'est à dire ssi m est inversible dans $\mathbb{Z}/n\mathbb{Z}$. La valeur $\phi(n)$ est donc le nombre d'éléments inversibles de $\mathbb{Z}/n\mathbb{Z}$.

Compléter la fonction `phi(n)` qui retourne la valeur de l'indicatrice d'Euler en un entier n . On peut bien sûr tester tous les entiers entre 0 et $n - 1$ pour voir s'ils sont premiers avec n . On améliorera cela en partant de la liste de ces entiers et, à chaque fois que l'on en repère un non premier avec n , en éliminant ses multiples (cela ressemble un peu au crible d'Ératosthène).

```
def phi(n):
    L=n*[1]# L[i] prendra la valeur 0 si i est multiple
              # d'un entier non premier avec n
    phi=1 # 1 est premier avec n
    i=2
    #A compléter
    return phi
```

Question 1.4 (Inversion modulaire) – Écrire une fonction `Inv_mod(x,n)` qui retourne l'inverse de x modulo n . On testera en début de programme si x est bien premier avec n grâce à une instruction du type `assert`.

1.3 Théorème chinois

1. Rappeler comment on résout un système du type
$$\begin{cases} x \equiv 1 \pmod{117} \\ x \equiv 4 \pmod{25} \end{cases}$$
2. Écrire une fonction `resout(a,n,b,m)` qui, pour m et n premiers entre eux, donne une solution (l'unique solution comprise entre 0 et $mn - 1$) du système
$$\begin{cases} x \equiv a \pmod{n} \\ x \equiv b \pmod{m} \end{cases}$$

1.4 Système RSA

Le système RSA est un système de chiffrement, c'est à dire qu'il sert à transmettre des messages cryptés. On considèrera ici que l'on veut transmettre un nombre m (on ne s'intéresse pas à la façon dont on passe du texte à ce nombre). Le système RSA est dit à clef publique : la partie de la clef permettant le chiffrement peut être publique. Le principe est le suivant :

- On choisit deux nombres premiers p et q , et on pose $n = pq$.
- On choisit un nombre e premier avec $\phi(n)$ (un nombre premier qui ne divise pas $\phi(n)$ par exemple) ; e peut être public. On détermine ensuite d tel que $ed = 1 \pmod{\phi(n)}$; d est la clef de décryptage, secrète.
- L'émetteur envoie au récepteur $m^e \pmod{n}$.
Le système repose alors sur le fait que $(m^e)^d \pmod{n} = m \pmod{n}$.
Il suffit donc au récepteur de calculer $(m^e)^d \pmod{n}$ pour récupérer m (on a supposé $m < n$).

L'intérêt de ce système est que le récepteur peut transmettre à une personne (émetteur) dont il veut recevoir des messages codés la clef (n, e) par un canal public. Il suffit qu'il garde secret d (p , q et donc $\phi(n)$, qui permettent de déterminer d à partir de e , sont également secrets).

La robustesse du code est basée sur la difficulté, lorsque p et q sont assez grands, de les retrouver à partir de n .

1. Si $n = pq$, avec p et q premier, que vaut $\phi(n)$?
2. Montrer l'égalité sur laquelle repose ce système : $(m^e)^d = m \pmod{n}$.
3. On prendra pour tester nos programmes $p = 127$, $q = 131$.
Calculer n , $\phi(n)$; choisir un nombre e ; déterminer d .
4. Écrire une fonction `codeRSA(m,n,e)` qui code le nombre m selon la clef (n, e) ; puis une fonction `decodeRSA(x,n,d)` qui décode le nombre x selon la clef (n, d) .
En pratique, pour éviter les nombres trop grands lors des calculs de puissances modulo n , on utilisera une fonction `puis(x,k,n)` qui calcule $x^k \pmod{n}$ en effectuant le reste modulo n à chaque multiplication par x .
5. Vérifier le bon fonctionnement du système sur notre exemple.

1.5 Complément sur l'indicatrice d'Euler

1. Calculer à la main la somme des $\phi(d)$ lorsque d parcourt l'ensemble des diviseurs de 10.
2. Écrire un programme qui calcule cette somme pour un entier quelconque n . Tester avec quelques entiers. Que constate-t-on ?
3. Démontrer que ce qui a été constaté ci-dessus est vrai pour tout entier naturel $n = pq$, où p et q sont deux nombres premiers (on utilisera l'expression connue de $\phi(n)$, $\phi(p)$, $\phi(q)$).
4. Cas général : soit $n \in \mathbb{N}^*$. On note $A_n = \{0, 1, \dots, n-1\}$ et $\mathcal{D}_n$ l'ensemble des diviseurs de n .
 - (a) Soit n un entier naturel non nul et d un diviseur de n . Soit k un multiple de d . Montrer que d est le pgcd de n et k si et seulement si k/d est premier avec n/d .
 - (b) On note, pour tout $d \in \mathcal{D}_n$, K_d l'ensemble des éléments de A_n qui sont des multiples de d et L_d ceux dont le pgcd avec n est d . Montrer que l'application allant de K_d dans $A_{n/d}$ qui à k associe k/d établit une bijection entre L_d et l'ensemble des éléments de $A_{n/d}$ premiers avec n/d .
 - (c) Conclure.

2 Complément : problème sur l'ordre d'une permutation

Une permutation est une bijection d'un ensemble E dans lui-même. L'ensemble des permutations, muni de la loi de composition, forme un groupe. On s'intéresse ici aux permutations d'un ensemble fini à n éléments : $E_n = \{0, 1, \dots, n-1\}$. Pour représenter en machine une permutation t de cet ensemble, on utilise une liste de longueur n $\mathbf{t}$ telle que $\mathbf{t}[\mathbf{k}] = t(k)$. Prenons un exemple : la permutation de $\{0, 1, 2\}$ qui échange 0 et 1 et qui laisse 2 fixe est représentée par $\mathbf{t}=[1,0,2]$. Remarquons que l'ensemble E_n coïncide avec l'ensemble des indices d'un tel tableau.

On supposera, sans avoir à le vérifier, que les tableaux d'entiers (de taille n) donnés en arguments aux fonctions à écrire représentent bien des permutations (sur E_n). Dans cet esprit, on confond par la suite les permutations et les tableaux d'entiers qui les représentent en machine.

2.1 Ordre d'une permutation

Question 2.1 – Écrire une fonction `composer(t,u)` qui prend (les tableaux représentant) deux permutations sur E_n en arguments et renvoie (le tableau représentant) la composée $\mathbf{t} \circ \mathbf{u}$ de $\mathbf{t}$ et de $\mathbf{u}$.

Rappelons que l'ordre d'une permutation t (et plus généralement celui d'un élément d'un groupe) est le plus petit entier k non nul tel que t^k (t composé k -fois) est l'identité.

Question 2.2 – Écrire une fonction `estIdentite(t)` qui teste si une permutation $\mathbf{t}$ est l'identité (et retourne un booléen).

Question 2.3 – Écrire une fonction `ordre(t)` qui renvoie l'ordre de la permutation $\mathbf{t}$.

2.2 Périodes

Étant donnée une permutation t de E_n , l'orbite d'un élément i de E_n est l'ensemble des $t^k(i)$, $k \geq 0$. On appellera période de i sous t la taille de cette orbite. Notons que la valeur maximale d'une période est n .

Question 2.4 – Pour fixer les idées, déterminer les orbites et périodes dans le cas où la permutation est celle représentée par le tableau $\mathbf{t}=[2,0,1,4,3]$. Déterminer également la décomposition de cette permutation en cycles.

Question 2.5 – Écrire une fonction `periode(t,i)` qui retourne la période de l'élément i de E_n sous la permutation t .

Question 2.6 – Quelle est la complexité (dans le pire des cas), en fonction du cardinal n de E_n (et donc de la longueur n de t), de l'algorithme précédent ?

On souhaite construire le tableau p des périodes de t : $p(i)$ sera égal à la période de i sous t . En appelant la fonction `periode(t,i)` pour chaque i , quelle serait alors la complexité d'un tel algorithme ?

Question 2.7 – Écrire une fonction `periodes(t)` qui renvoie le tableau des périodes décrit ci-dessus. On demande un algorithme de complexité linéaire (on ne demande pas de justifier ce fait). On pourra remarquer que les éléments d'une même orbite ont la même période.

2.3 Détermination efficace de l'ordre

Question 2.8 – Écrire une fonction `ppcm(a,b)` qui prend deux entiers positifs et renvoie leur plus petit multiple commun.

Question 2.9 – On va maintenant calculer l'ordre d'une permutation en remarquant qu'il est égal au ppcm des périodes. Écrire une fonction `ordreEfficace(t)` qui calcule l'ordre de la permutation t selon cette méthode.

Question 2.10 – On montre que l'ordre maximal d'une permutation est de l'ordre de $n^{\sqrt{n}}$. Quelle était la complexité de la fonction `ordre(t)` ? Comparer avec la complexité de `ordreEfficace(t)`.

2.4 Signature d'une permutation

Question 2.11 – Quelle serait la complexité d'un algorithme calculant la signature d'une permutation directement à partir de sa définition ?

Question 2.12 – Écrire une fonction `signature(t)` qui calcule la signature de t à partir du tableau des périodes. Quelle est sa complexité ?