

TP4 : Calculs dans $\mathbb{Z}/n\mathbb{Z}$

N'oubliez pas d'exécuter (valider avec la touche Entrée) les commandes Maple (texte en rouge) avant de les utiliser.

— Les commandes Maple qui peuvent vous être utiles

Beaucoup des très nombreuses fonctions de Maple ne sont pas disponibles au démarrage : elles sont rangées dans des bibliothèques par thème (algèbre linéaire, équations différentielles, arithmétique, graphiques, etc...) et il faut les charger pour les utiliser.

La bibliothèque d'arithmétique s'appelle **numtheory** (théorie des nombres). On commence donc par

```
[ > with(numtheory):  
  Warning, new definition for order
```

Parcourez rapidement les exemples ci-dessous, vous reviendrez faire votre marché au besoin en résolvant les exercices.

```
[ > ifactor(60);  
                                     (2)2 (3) (5)
```

La fonction usuelle de décomposition d'un entier en facteurs premiers, le résultat est joli mais peu utilisable dans un programme.

```
[ > ifactors(60);  
                                     [1, [[2, 2], [3, 1], [5, 1]]]
```

Beaucoup moins lisible mais beaucoup plus utile : le premier élément 1 ou -1 est le signe, le deuxième une liste de couples $[p, k]$ représentant le facteur p^k .

Pour n'avoir que cette liste

```
[ > ifactors(60)[2];  
                                     [[2, 2], [3, 1], [5, 1]]
```

La fonction **phi** de Maple vous permettra de vérifier vos programmes des exercices 1 et 2.

```
[ > phi(60);  
                                     16
```

Les fonctions **divisors** et **factorset** donnent l'ensemble des diviseurs et l'ensemble des facteurs premiers d'un entier.

```
[ > divisors(60);  
                                     {1, 2, 3, 4, 5, 6, 10, 12, 15, 20, 30, 60}  
[ > factorset(60);  
                                     {2, 3, 5}
```

La fonction **order** permet de calculer l'ordre d'un élément modulo n et donc de vérifier vos programmes des exercices 3 et 4.

```
[ > order(49, 60); 49*49;  
                                     2
```

La fonction **primroot** donne le plus petit élément primitif de $\mathbb{Z}/n\mathbb{Z}$ s'il existe.

```
> primroot(11);primroot(60);
2
FAIL
```

Facultatif : pour celles et ceux qui aiment soulever le capot

La plupart des fonctions de Maple sont écrites en Maple, pour les curieux il est possible de faire afficher leur code.

La commande magique qui permet de voir le code que Maple a dans le ventre :

```
> interface(verboseproc=2);
1
> print(numtheory[phi]);
proc(n::Or(integer, And(algebraic, Not(constant))))
local s1, s2, x;
option
remember, system, 'Copyright (c) 1992 by the University of Waterloo. All rights reserved.';
if nargs ≠ 1 then error "invalid arguments"
elif not type(n, 'integer') then 'procname'(n)
elif n < 10 then
    if n < 0 then procname(-n) else op(n + 1, [0, 1, 1, 2, 2, 4, 2, 6, 4, 6]) end if
elif isprime(n) then n - 1
else s1 := numtheory:-factorset(n); s2 := mul(x - 1, x = s1); n*s2 / convert(s1, '**')
end if
end proc
```

La procédure commence par un certain nombre de tests sur le type de l'argument.

Dans les versions plus récentes une partie de ces tests sont dans le typage du paramètre.

Certaines procédures sont de longues listes de tests sur le type des arguments suivis de l'appel aux procédures qui font le boulot et ce sont celles-là qu'il faut aller voir, il peut y avoir un peu de jeu de piste.

J'ai choisi phi parce qu'elle est simple et on peut voir l'algorithme :

- si $n < 10$ le résultat est pris dans une liste
- si n est premier on rend $n - 1$
- sinon on appelle factorset qui rend l'ensemble $\{p_1, \dots, p_k\}$ des facteurs premiers de n sans

leurs exposants et on utilise la formule
$$\frac{n \left(\prod_{i=1}^k (p_i - 1) \right)}{\prod_{i=1}^k p_i} .$$

Dans les versions plus récentes c'est plus lisible car on utilise des variables à la place des

horribles " et "" (expression précédente et préprécédente).

Exercice (facile) : démontrer que la formule utilisée donne le même résultat que la formule usuelle.

— Nombre d'inversibles modulo n

On va écrire nos propres versions de la fonction phi.

Exercice 1 : Ecrire une fonction **NbInv1(n)** qui donne le nombre d'inversibles modulo n en comptant le nombre d'entiers compris entre 1 et n-1 qui sont premiers avec n.

— Solution

```
> NbInv1:=proc(n)
  local c,i;
  c:=0;
  for i from 1 to n-1 do if igcd(i,n)=1 then c:=c+1 fi od;
  c
end:
> NbInv1(60);
16
> NbInv1(1515),phi(1515);
800, 800
```

Exercice 2 : Ecrire une fonction **NbInv2(n)** qui donne le nombre d'inversibles modulo n en utilisant la décomposition de n en facteurs premiers par la fonction **ifactors** et la formule usuelle.

Vous pouvez utiliser la boucle for qui parcourt une liste : **for truc in liste do ... od**.

Comparer la vitesse d'exécution des deux fonctions NbInv sur de grands entiers tirés au hasard.

On trouvera dans la partie facultative une autre formule qui utilise **factorset**.

— Solution

Une première solution assez pépère plutôt lisible :

On récupère le résultat de **ifactors** en oubliant le signe.

C'est une liste de couples $[p, k]$ représentant le facteur p^k .

Pour chaque couple on récupère le p et le k et on accumule dans la variable *phin* le produit des $p^{(k-1)}(p-1)$.

```
> NbInv2:=proc(n)
  local Lfact,Facteur,phin,p,k;
  Lfact:=ifactors(n)[2];phin:=1;
  for Facteur in Lfact do
    p:=Facteur[1];k:=Facteur[2];phin:=phin*p^(k-1)*(p-1)
  od;
  phin
end:
> NbInv2(60),NbInv2(1515);
16, 800
```

Maintenant la soluce hacker fou

```
[ > NbInv2:= proc(n) local fact;  
mul(fact[1]^(fact[2]-1)*(fact[1]-1),fact=ifactors(n)[2])  
end;  
[ > NbInv2(60),NbInv2(1515);  
16,800
```

Les fonctions **add** et **mul** permettent de calculer des sommes ou des produits de termes sans utiliser une boucle **for** lorsque le terme général de cette somme ou produit peut être calculé par une simple expression et ne demande pas l'exécution d'un bloc d'instructions. Ne pas les confondre avec les fonctions **sum** et **product** destinées aux sommes et produits formels.

On va maintenant comparer la vitesse des deux fonctions, ceci sera utile dans d'autres TPs :

On commence par définir une fonction de tirage (pseudo)aléatoire

```
[ > tire:=rand(10^10..10^11):
```

La fonction **rand** permet de définir une fonction de tirage (pseudo)aléatoire dans un intervalle donné.

```
[ > n:=tire();  
n := 77419669077
```

```
[ > t:=time():NbInv1(n);time()-t;
```

```
[ Warning, computation interrupted
```

Calcul interrompu après 20 minutes ! J'ai vu un peu grand, ça dépend aussi de la machine utilisée.

Remarquer comment on mesure le temps d'exécution : on prend le temps avant, après et on fait la différence.

```
[ > t:=time():NbInv2(n);time()-t;  
48096057600  
0  
[ > t:=time():phi(n);time()-t;  
48096057600  
0
```

Il n'y a pas photos, le temps n'est même pas mesurable mais on utilise l'extraordinaire efficacité de Maple à décomposer n.

```
[ > ifactor(n);  
(3)3 (17) (101) (1670003)
```

On va chercher un n pas trop grand pour que **NbInv1** termine en un temps raisonnable.

```
[ > tire:=rand(10^6..10^7):  
[ > n:=tire();  
n := 9657592  
[ > ifactor(n);  
(2)3 (7) (37) (59) (79)  
[ > t:=time():NbInv1(n);time()-t;  
3908736  
44.656
```

45 secondes. On a bien fait d'arrêter le calcul précédent : n était 10000 fois plus grand, 45*10000 secondes = 125 heures de calcul !

```
[ > t:=time():NbInv2(n);time()-t;  
3908736
```

Evidemment

– Ordre d'un élément de $\mathbb{Z}/n\mathbb{Z}$

Exercice 3 : Ecrire une fonction `Ordre1(a,n)` qui rend l'ordre de a modulo n si a est premier avec n , FAIL sinon. On calculera l'ordre en utilisant sa définition : le premier $0 < k$ tel que $a^k \bmod n = 1$.

On pourra utiliser **RETURN** pour s'arrêter dès qu'on a trouvé l'ordre.

– Solution

```
> Ordre1:=proc(a,n)
  local ak,i;
  if igcd(n,a)=1
  then ak:=1;
    for i from 1 do
      ak:=ak*a mod n;if ak=1 then RETURN(i) fi od
    else FAIL fi
  end:
> Ordre1(5,60),order(5,60);
                                FAIL, FAIL
> Ordre1(7,60),order(7,60);
                                4, 4
```

Remarquer le **for** sans test d'arrêt et la sortie de fonction par le **RETURN**.

Exercice 4 : On va essayer d'être plus malin.

Ecrire une fonction `Ordre2(a,n)` qui rend l'ordre de a modulo n si a est premier avec n , FAIL sinon. On calculera l'ordre en utilisant : le premier k diviseur de $\phi(n)$ tel que $a^k \bmod n = 1$.

Attention la fonction **divisors** retourne l'ensemble des diviseurs donc pas forcément dans l'ordre croissant.

```
> divisors(1515);
                                {1, 3, 5, 15, 101, 1515, 303, 505}
```

Pour garantir l'ordre des diviseurs il faut faire

```
> sort(convert(divisors(1515),list));
                                [1, 3, 5, 15, 101, 303, 505, 1515]
```

Comparer l'efficacité des deux fonctions Ordre sur des n grands avec des a dont l'ordre modulo n est grand.

– Solution

```
> Ordre2:=proc(a,n)
  local ListeDiv,Div;
  if igcd(n,a)=1
  then ListeDiv:=sort(convert(divisors(phi(n)),list));
    for Div in ListeDiv do if a^Div mod n=1 then
      RETURN(Div) fi od
    else FAIL fi
```

```

[ end:
[ > Ordre2(5,60),Ordre2(7,60);
                                FAIL, 4

```

On cherche un bon exemple sur lequel Ordre2 est plus rapide

```

[ > tire:=rand(10^7..10^8):
[ > n:=tire();
                                n := 23069879
[ > phi(n);
                                19651896
[ > divisors(phi(n));
{1, 2, 3, 4, 6, 8, 9, 11, 12, 18, 22, 24, 27, 33, 36, 44, 54, 66, 72, 81, 88, 99, 108, 132,
  162, 594, 792, 396, 297, 264, 198, 1782, 1188, 891, 19651896, 8271, 7352, 5514,
  3676, 1838, 21384, 919, 7128, 2376, 2757, 10692, 3564, 5346, 2673, 40436, 33084,
  30327, 22056, 243, 16542, 11028, 10109, 1637658, 181962, 242616, 272943, 363924,
  727848, 545886, 148878, 121308, 99252, 90981, 80872, 74439, 66168, 60654, 49626,
  486, 324, 972, 1091772, 216, 818829, 24813, 20218, 648, 1944, 6550632, 2183544,
  3275316, 4912974, 1786536, 446634, 9825948, 223317, 2456487, 198504, 893268,
  297756, 595512}

```

On cherche un a dont l'ordre soit supérieur à un million.

```

[ > for a from 2 to 30 do if igcd(a,n)=1 and order(a,n)>10^6
  then print(a) fi od;
                                2
                                3
                                11
                                18
                                19
                                20
                                24
                                26
                                29

```

2 convient

```

[ > order(2,n);
                                1637658
[ > t:=time():Ordre1(2,n);time()-t;
                                1637658
                                9.750

```

Environ 10 secondes

```

[ > t:=time():Ordre2(2,n);time()-t;
                                1637658
                                0

```

Le temps n'est pas mesurable.

– Générateurs de $(\mathbb{Z}/n\mathbb{Z})^*$

Exercice 5 : Si l'ordre d'un élément de $(\mathbb{Z}/n\mathbb{Z})^*$ est égal au nombre d'éléments $\phi(n)$ de $(\mathbb{Z}/n\mathbb{Z})^*$, cet élément engendre le groupe $(\mathbb{Z}/n\mathbb{Z})^*$ et le groupe est cyclique.

Ecrire une fonction qui rende un générateur du groupe $(\mathbb{Z}/n\mathbb{Z})^*$ s'il existe, FAIL sinon. Vous pouvez utiliser les fonctions **phi** et **order** de Maple.

Grace à cette fonction construire une suite de couples $[n, \text{un générateur du groupe des inversibles modulo } n \text{ sinon FAIL}]$ pour n allant de 2 à 100.

Pouvez-vous conjecturer pour quels n le groupe $(\mathbb{Z}/n\mathbb{Z})^*$ a un générateur ? Si nécessaire agrandissez l'intervalle.

– Solution

```
> Generateur:=proc(n)
  local a,NbInv;
  NbInv:=phi(n);
  for a from 1 to n-1 do if order(a,n)=NbInv then RETURN(a)
  fi od;
  FAIL
end;
```

On utilise une variable **NbInv** pour ne pas recalculer **phi(n)** à chaque test. De nouveau le **RETURN** permet de sortir dès qu'on trouve ce qu'on cherche.

Le calcul des n pour lesquels il y a un générateur.

```
> seq([n,Generateur(n)],n=2..100);
[2, 1], [3, 2], [4, 3], [5, 2], [6, 5], [7, 3], [8, FAIL], [9, 2], [10, 3], [11, 2], [12, FAIL],
[13, 2], [14, 3], [15, FAIL], [16, FAIL], [17, 3], [18, 5], [19, 2], [20, FAIL], [21, FAIL],
[22, 7], [23, 5], [24, FAIL], [25, 2], [26, 7], [27, 2], [28, FAIL], [29, 2], [30, FAIL],
[31, 3], [32, FAIL], [33, FAIL], [34, 3], [35, FAIL], [36, FAIL], [37, 2], [38, 3],
[39, FAIL], [40, FAIL], [41, 6], [42, FAIL], [43, 3], [44, FAIL], [45, FAIL], [46, 5],
[47, 5], [48, FAIL], [49, 3], [50, 3], [51, FAIL], [52, FAIL], [53, 2], [54, 5], [55, FAIL],
[56, FAIL], [57, FAIL], [58, 3], [59, 2], [60, FAIL], [61, 2], [62, 3], [63, FAIL],
[64, FAIL], [65, FAIL], [66, FAIL], [67, 2], [68, FAIL], [69, FAIL], [70, FAIL], [71, 7],
[72, FAIL], [73, 5], [74, 5], [75, FAIL], [76, FAIL], [77, FAIL], [78, FAIL], [79, 3],
[80, FAIL], [81, 2], [82, 7], [83, 2], [84, FAIL], [85, FAIL], [86, 3], [87, FAIL],
[88, FAIL], [89, 3], [90, FAIL], [91, FAIL], [92, FAIL], [93, FAIL], [94, 5], [95, FAIL],
[96, FAIL], [97, 5], [98, 3], [99, FAIL], [100, FAIL]
```

On compare à Maple

```
> seq([n,primroot(n)],n=2..100);
[2, 1], [3, 2], [4, 3], [5, 2], [6, 5], [7, 3], [8, FAIL], [9, 2], [10, 3], [11, 2], [12, FAIL],
[13, 2], [14, 3], [15, FAIL], [16, FAIL], [17, 3], [18, 5], [19, 2], [20, FAIL], [21, FAIL],
[22, 7], [23, 5], [24, FAIL], [25, 2], [26, 7], [27, 2], [28, FAIL], [29, 2], [30, FAIL],
[31, 3], [32, FAIL], [33, FAIL], [34, 3], [35, FAIL], [36, FAIL], [37, 2], [38, 3],
[39, FAIL], [40, FAIL], [41, 6], [42, FAIL], [43, 3], [44, FAIL], [45, FAIL], [46, 5],
[47, 5], [48, FAIL], [49, 3], [50, 3], [51, FAIL], [52, FAIL], [53, 2], [54, 5], [55, FAIL],
```

[56, FAIL], [57, FAIL], [58, 3], [59, 2], [60, FAIL], [61, 2], [62, 3], [63, FAIL],
 [64, FAIL], [65, FAIL], [66, FAIL], [67, 2], [68, FAIL], [69, FAIL], [70, FAIL], [71, 7],
 [72, FAIL], [73, 5], [74, 5], [75, FAIL], [76, FAIL], [77, FAIL], [78, FAIL], [79, 3],
 [80, FAIL], [81, 2], [82, 7], [83, 2], [84, FAIL], [85, FAIL], [86, 3], [87, FAIL],
 [88, FAIL], [89, 3], [90, FAIL], [91, FAIL], [92, FAIL], [93, FAIL], [94, 5], [95, FAIL],
 [96, FAIL], [97, 5], [98, 3], [99, FAIL], [100, FAIL]

On trouve les n premiers (cf. le cours) mais pas seulement.

Il y a un générateur pour $n = 2, n = 4, n = p^k, n = 2p^k$ où p est un nombre premier différent de 2.

La démonstration n'est pas très compliquée mais un peu longue.

Exercice 6 : Ecrire une fonction qui rende l'ensemble, éventuellement vide, des générateurs du groupe $(\mathbf{Z}/n\mathbf{Z})^*$.

Vous pouvez utiliser la méthode simple ou mieux la remarque du cours qui dit que quand on a un générateur on sait construire les autres.

Vous pouvez aussi constater que quand il y a un générateur il y en a $\phi(\phi(n))$.

– **Solution**

Une solution naïve

```
> Generateurs:=proc(n)
  local a,NbInv,s;
  NbInv:=phi(n);s:=NULL;
  for a from 1 to n-1 do if order(a,n)=NbInv then s:=s,a fi
  od;
  {s}
end;
```

qu'on teste sur l'exemple précédent

```
> seq([n,Generateurs(n),phi(phi(n))],n=2..100);
```

[2, {1}, 1], [3, {2}, 1], [4, {3}, 1], [5, {2, 3}, 2], [6, {5}, 1], [7, {3, 5}, 2], [8, { }, 2],
 [9, {2, 5}, 2], [10, {3, 7}, 2], [11, {2, 6, 7, 8}, 4], [12, { }, 2], [13, {2, 6, 7, 11}, 4],
 [14, {3, 5}, 2], [15, { }, 4], [16, { }, 4], [17, {3, 5, 6, 7, 10, 11, 12, 14}, 8],
 [18, {5, 11}, 2], [19, {2, 3, 10, 13, 14, 15}, 6], [20, { }, 4], [21, { }, 4],
 [22, {7, 13, 17, 19}, 4], [23, {5, 7, 10, 11, 14, 15, 17, 19, 20, 21}, 10], [24, { }, 4],
 [25, {2, 3, 8, 12, 13, 17, 22, 23}, 8], [26, {7, 11, 15, 19}, 4],
 [27, {2, 5, 11, 14, 20, 23}, 6], [28, { }, 4],
 [29, {2, 3, 8, 10, 11, 14, 15, 18, 19, 21, 26, 27}, 12], [30, { }, 4],
 [31, {3, 11, 12, 13, 17, 21, 22, 24}, 8], [32, { }, 8], [33, { }, 8],
 [34, {3, 5, 7, 11, 23, 27, 29, 31}, 8], [35, { }, 8], [36, { }, 4],
 [37, {2, 5, 13, 15, 17, 18, 19, 20, 22, 24, 32, 35}, 12], [38, {3, 13, 15, 21, 29, 33}, 6],
 [39, { }, 8], [40, { }, 8],
 [41, {6, 7, 11, 12, 13, 15, 17, 19, 22, 24, 26, 28, 29, 30, 34, 35}, 16], [42, { }, 4],
 [43, {3, 5, 12, 18, 19, 20, 26, 28, 29, 30, 33, 34}, 12], [44, { }, 8], [45, { }, 8],

[46, {5, 7, 11, 15, 17, 19, 21, 33, 37, 43}, 10], [47, {5, 10, 11, 13, 15, 19, 20, 22, 23, 26, 29, 30, 31, 33, 35, 38, 39, 40, 41, 43, 44, 45}, 22], [48, { }, 8], [49, {3, 5, 10, 12, 17, 24, 26, 33, 38, 40, 45, 47}, 12], [50, {3, 13, 17, 23, 27, 33, 37, 47}, 8], [51, { }, 16], [52, { }, 8], [53, {2, 3, 5, 8, 12, 14, 18, 19, 20, 21, 22, 26, 27, 31, 32, 33, 34, 35, 39, 41, 45, 48, 50, 51}, 24], [54, {5, 11, 23, 29, 41, 47}, 6], [55, { }, 16], [56, { }, 8], [57, { }, 12], [58, {3, 11, 15, 19, 21, 27, 31, 37, 39, 43, 47, 55}, 12], [59, {2, 6, 8, 10, 11, 13, 14, 18, 23, 24, 30, 31, 32, 33, 34, 37, 38, 39, 40, 42, 43, 44, 47, 50, 52, 54, 55, 56}, 28], [60, { }, 8], [61, {2, 6, 7, 10, 17, 18, 26, 30, 31, 35, 43, 44, 51, 54, 55, 59}, 16], [62, {3, 11, 13, 17, 21, 43, 53, 55}, 8], [63, { }, 12], [64, { }, 16], [65, { }, 16], [66, { }, 8], [67, {2, 7, 11, 12, 13, 18, 20, 28, 31, 32, 34, 41, 44, 46, 48, 50, 51, 57, 61, 63}, 20], [68, { }, 16], [69, { }, 20], [70, { }, 8], [71, {7, 11, 13, 21, 22, 28, 31, 33, 35, 42, 44, 47, 52, 53, 55, 56, 59, 61, 62, 63, 65, 67, 68, 69}, 24], [72, { }, 8], [73, {5, 11, 13, 14, 15, 20, 26, 28, 29, 31, 33, 34, 39, 40, 42, 44, 45, 47, 53, 58, 59, 60, 62, 68}, 24], [74, {5, 13, 15, 17, 19, 35, 39, 55, 57, 59, 61, 69}, 12], [75, { }, 16], [76, { }, 12], [77, { }, 16], [78, { }, 8], [79, {3, 6, 7, 28, 29, 30, 34, 35, 37, 39, 43, 47, 48, 53, 54, 59, 60, 63, 66, 68, 70, 74, 75, 77}, 24], [80, { }, 16], [81, {2, 5, 11, 14, 20, 23, 29, 32, 38, 41, 47, 50, 56, 59, 65, 68, 74, 77}, 18], [82, {7, 11, 13, 15, 17, 19, 29, 35, 47, 53, 63, 65, 67, 69, 71, 75}, 16], [83, {2, 5, 6, 8, 13, 14, 15, 18, 19, 20, 22, 24, 32, 34, 35, 39, 42, 43, 45, 46, 47, 50, 52, 53, 54, 55, 56, 57, 58, 60, 62, 66, 67, 71, 72, 73, 74, 76, 79, 80}, 40], [84, { }, 8], [85, { }, 32], [86, {3, 5, 19, 29, 33, 55, 61, 63, 69, 71, 73, 77}, 12], [87, { }, 24], [88, { }, 16], [89, {3, 6, 7, 13, 14, 15, 19, 23, 24, 26, 27, 28, 29, 30, 31, 33, 35, 38, 41, 43, 46, 48, 51, 54, 56, 58, 59, 60, 61, 62, 63, 65, 66, 70, 74, 75, 76, 82, 83, 86}, 40], [90, { }, 8], [91, { }, 24], [92, { }, 20], [93, { }, 16], [94, {5, 11, 13, 15, 19, 23, 29, 31, 33, 35, 39, 41, 43, 45, 57, 67, 69, 73, 77, 85, 87, 91}, 22], [95, { }, 24], [96, { }, 16], [97, {5, 7, 10, 13, 14, 15, 17, 21, 23, 26, 29, 37, 38, 39, 40, 41, 56, 57, 58, 59, 60, 68, 71, 74, 76, 80, 82, 83, 84, 87, 90, 92}, 32], [98, {3, 5, 17, 33, 45, 47, 59, 61, 73, 75, 87, 89}, 12], [99, { }, 16], [100, { }, 16]

Plus malin : on appelle la fonction `Generateur`, si elle rend `FAIL` on rend `{ }`, si elle rend un générateur a on calcule les a^k avec k premier avec $\phi(n)$ qui sont les autres générateurs (cf cours).

```
> Generateurs:=proc(n)
  local a,s,NbInv,k;
  a:=Generateur(n);
  if a=FAIL then {}
  else s:=a;NbInv:=phi(n);
```

```

for k from 2 to NbInv-1 do
  if igcd(k,NbInv)=1 then s:=s,a^k mod n fi od;
{s} fi
end:

```

qu'on teste sur l'exemple précédent

```

> seq([n,Generateurs(n),phi(phi(n))],n=2..100);
[2, {1}, 1], [3, {2}, 1], [4, {3}, 1], [5, {2, 3}, 2], [6, {5}, 1], [7, {3, 5}, 2], [8, { }, 2],
[9, {2, 5}, 2], [10, {3, 7}, 2], [11, {2, 6, 7, 8}, 4], [12, { }, 2], [13, {2, 6, 7, 11}, 4],
[14, {3, 5}, 2], [15, { }, 4], [16, { }, 4], [17, {3, 5, 6, 7, 10, 11, 12, 14}, 8],
[18, {5, 11}, 2], [19, {2, 3, 10, 13, 14, 15}, 6], [20, { }, 4], [21, { }, 4],
[22, {7, 13, 17, 19}, 4], [23, {5, 7, 10, 11, 14, 15, 17, 19, 20, 21}, 10], [24, { }, 4],
[25, {2, 3, 8, 12, 13, 17, 22, 23}, 8], [26, {7, 11, 15, 19}, 4],
[27, {2, 5, 11, 14, 20, 23}, 6], [28, { }, 4],
[29, {2, 3, 8, 10, 11, 14, 15, 18, 19, 21, 26, 27}, 12], [30, { }, 4],
[31, {3, 11, 12, 13, 17, 21, 22, 24}, 8], [32, { }, 8], [33, { }, 8],
[34, {3, 5, 7, 11, 23, 27, 29, 31}, 8], [35, { }, 8], [36, { }, 4],
[37, {2, 5, 13, 15, 17, 18, 19, 20, 22, 24, 32, 35}, 12], [38, {3, 13, 15, 21, 29, 33}, 6],
[39, { }, 8], [40, { }, 8],
[41, {6, 7, 11, 12, 13, 15, 17, 19, 22, 24, 26, 28, 29, 30, 34, 35}, 16], [42, { }, 4],
[43, {3, 5, 12, 18, 19, 20, 26, 28, 29, 30, 33, 34}, 12], [44, { }, 8], [45, { }, 8],
[46, {5, 7, 11, 15, 17, 19, 21, 33, 37, 43}, 10], [47,
{5, 10, 11, 13, 15, 19, 20, 22, 23, 26, 29, 30, 31, 33, 35, 38, 39, 40, 41, 43, 44, 45}, 22
], [48, { }, 8], [49, {3, 5, 10, 12, 17, 24, 26, 33, 38, 40, 45, 47}, 12],
[50, {3, 13, 17, 23, 27, 33, 37, 47}, 8], [51, { }, 16], [52, { }, 8], [53,
{2, 3, 5, 8, 12, 14, 18, 19, 20, 21, 22, 26, 27, 31, 32, 33, 34, 35, 39, 41, 45, 48, 50, 51
}, 24], [54, {5, 11, 23, 29, 41, 47}, 6], [55, { }, 16], [56, { }, 8], [57, { }, 12],
[58, {3, 11, 15, 19, 21, 27, 31, 37, 39, 43, 47, 55}, 12], [59, {2, 6, 8, 10, 11, 13, 14, 18,
23, 24, 30, 31, 32, 33, 34, 37, 38, 39, 40, 42, 43, 44, 47, 50, 52, 54, 55, 56}, 28],
[60, { }, 8], [61, {2, 6, 7, 10, 17, 18, 26, 30, 31, 35, 43, 44, 51, 54, 55, 59}, 16],
[62, {3, 11, 13, 17, 21, 43, 53, 55}, 8], [63, { }, 12], [64, { }, 16], [65, { }, 16],
[66, { }, 8],
[67, {2, 7, 11, 12, 13, 18, 20, 28, 31, 32, 34, 41, 44, 46, 48, 50, 51, 57, 61, 63}, 20],
[68, { }, 16], [69, { }, 20], [70, { }, 8], [71, {7, 11, 13, 21, 22, 28, 31, 33, 35, 42, 44,
47, 52, 53, 55, 56, 59, 61, 62, 63, 65, 67, 68, 69}, 24], [72, { }, 8], [73, {5, 11, 13, 14,
15, 20, 26, 28, 29, 31, 33, 34, 39, 40, 42, 44, 45, 47, 53, 58, 59, 60, 62, 68}, 24],
[74, {5, 13, 15, 17, 19, 35, 39, 55, 57, 59, 61, 69}, 12], [75, { }, 16], [76, { }, 12],
[77, { }, 16], [78, { }, 8], [79, {
3, 6, 7, 28, 29, 30, 34, 35, 37, 39, 43, 47, 48, 53, 54, 59, 60, 63, 66, 68, 70, 74, 75, 77
}, 24], [80, { }, 16],
[81, {2, 5, 11, 14, 20, 23, 29, 32, 38, 41, 47, 50, 56, 59, 65, 68, 74, 77}, 18],

```

[82, {7, 11, 13, 15, 17, 19, 29, 35, 47, 53, 63, 65, 67, 69, 71, 75}, 16], [83, {2, 5, 6, 8, 13, 14, 15, 18, 19, 20, 22, 24, 32, 34, 35, 39, 42, 43, 45, 46, 47, 50, 52, 53, 54, 55, 56, 57, 58, 60, 62, 66, 67, 71, 72, 73, 74, 76, 79, 80}, 40], [84, { }, 8], [85, { }, 32], [86, {3, 5, 19, 29, 33, 55, 61, 63, 69, 71, 73, 77}, 12], [87, { }, 24], [88, { }, 16], [89, {3, 6, 7, 13, 14, 15, 19, 23, 24, 26, 27, 28, 29, 30, 31, 33, 35, 38, 41, 43, 46, 48, 51, 54, 56, 58, 59, 60, 61, 62, 63, 65, 66, 70, 74, 75, 76, 82, 83, 86}, 40], [90, { }, 8], [91, { }, 24], [92, { }, 20], [93, { }, 16], [94, {5, 11, 13, 15, 19, 23, 29, 31, 33, 35, 39, 41, 43, 45, 57, 67, 69, 73, 77, 85, 87, 91}, 22], [95, { }, 24], [96, { }, 16], [97, {5, 7, 10, 13, 14, 15, 17, 21, 23, 26, 29, 37, 38, 39, 40, 41, 56, 57, 58, 59, 60, 68, 71, 74, 76, 80, 82, 83, 84, 87, 90, 92}, 32], [98, {3, 5, 17, 33, 45, 47, 59, 61, 73, 75, 87, 89}, 12], [99, { }, 16], [100, { }, 16]