X Informatique PSI 2009 — Corrigé

Ce corrigé est proposé par Vincent Puyhaubert (Professeur en CPGE) ; il a été relu par Arnaud Borde (École Polytechnique) et Céline Chevalier (ENS Cachan).

Le sujet d'informatique de cette année a pour thème l'arithmétique. Il a pour objectif de donner des méthodes pour calculer la suite des nombres premiers, la décomposition d'un entier en facteurs premiers, et des nombres dits de Carmichael.

- La première partie demande de programmer différentes méthodes pour trouver tous les nombres premiers plus petits qu'un entier n ; les techniques sont très classiques, de la méthode la plus élémentaire à une méthode optimisée (crible d'Érathostène). On programme ensuite la factorisation d'un entier.
- La deuxième partie fait programmer une méthode pour répondre à la question : un entier n donné est-il puissance d'un autre entier ?
- Enfin, la dernière partie introduit les nombres de Carmichael et propose diverses méthodes pour les calculer.

Dans l'ensemble, les programmes à écrire ne sont pas très compliqués. C'est même un aspect décevant du sujet : les algorithmes donnés par l'énoncé sont tellement détaillés qu'il ne reste plus vraiment de place pour la réflexion. Mais si vous n'avez pas l'habitude de programmer, c'est un très bon entraînement.

L'énoncé ne précise pas le contexte historique des nombres de Carmichael, notamment leur rôle dans le petit théorème de Fermat. Cette lacune sera comblée en annexe à ce corrigé.

Indications

1 Tester sans subtilités la divisibilité de n par tous les entiers compris entre 2 et $n-1$.

2 Ne tester cette fois la divisibilité que par les éléments de `premier`. Utiliser un compteur pour mettre à jour le nombre d'éléments de ce tableau.

3 Remplir le tableau `premier` au fur et à mesure, et ne tester les divisibilités que par des éléments de ce tableau.

4 Utiliser un tableau initialisé avec des 1. Rayer une case revient alors à passer sa valeur à 0.

7 Il faut parcourir le tableau `facteur` à l'aide de deux boucles `while` et deux compteurs. Le premier sert à compter le nombre de facteurs distincts de n, le second le nombre d'occurrences de ces facteurs.

8 Utiliser le résultat admis de l'énoncé et les multiplicités stockées dans le tableau `alpha`.

9 Calculer les carrés des entiers naturels jusqu'à tomber sur n ou le dépasser.

10 Il faut simplement effectuer tous les tests intervenant dans la définition des nombres de Carmichael. Pour avoir accès aux facteurs premiers de n, utiliser `facteur` et `alpha`.

11 Utiliser trois boucles `for` pour calculer tous les produits de 3 nombres premiers inférieurs ou égaux à n. Tester ensuite s'il s'agit de nombres de Carmichael.

I. Nombres premiers

Les fonctions de ce corrigé sont toutes rédigées dans le langage Maple. Toutefois, le quotient et le reste de la division euclidienne de deux entiers `a` et `b` s'obtiennent dans ce langage respectivement par les commandes `iquo(a,b)` et `irem(a,b)`. Ces commandes *ne sont pas utilisées* dans les programmes car l'énoncé impose les notations `a/b` et `mod(a,b)`. Pour obtenir des codes qui fonctionnent, il faut donc remplacer ces notations par les vraies fonctions.

De la même manière, une racine carrée s'obtient elle aussi par la commande `sqrt`, mais elle renvoie un nombre réel avec une certaine précision. Pour obtenir un entier, on doit en réalité la combiner avec la fonction partie entière, c'est-à-dire la commande `floor`.

1 Si n est divisible par un entier autre que 1 et lui-même, ce dernier est compris entre 2 et $n-1$. Par suite, il suffit de tester la divisibilité de n par tous les éléments de $[\![\,2\,;\,n-1\,]\!]$ (l'énoncé ne demande pas d'optimisation). Dès qu'un diviseur de n est trouvé, il ne sert plus à rien de tester les entiers suivants, donc on renvoie 0 immédiatement (rappelons au passage qu'en Maple, un `return` termine automatiquement l'exécution du programme). Si tous les tests ont échoué, l'entier est premier et on renvoie 1.

```
estPremier:=proc(n)
local i;
for i from 2 to (n-1) do
  if mod(n,i)=0 then   # n est divisible par i:
    return 0;          # il n'est pas premier
  fi;
od;
return 1;
end;
```

2 La première méthode naïve consiste à prendre successivement tous les entiers compris entre 2 et n et à tester (à l'aide de `estPremier`) s'ils sont premiers ; on range ceux qui le sont dans le tableau `premier`. La seule difficulté est de penser à utiliser une variable pour savoir combien de nombres premiers ont déjà été stockés dans ce tableau, ce qui se fait à l'aide d'un compteur `k`.

```
petitsPremiers:=proc(n)
local i,k;
global premier;
k:=0;
for i from 2 to n do
  if estPremier(i)=1 then
    k:=k+1; premier[k]:=i;
  fi;
od;
return k;
end;
```

> Dans ce programme et dans tous ceux qui suivent, il ne faut pas oublier de créer les variables globales avant de tester les fonctions dans Maple (ce n'est pas indispensable sur une copie). Le tableau `premier` peut par exemple être initialisé par
>
> `premier:=[seq(0,i=1..30)];`
>
> et il peut alors contenir 30 nombres premiers (sachant que le trentième nombre premier est 113).

3 La fonction utilise l'amélioration donnée par l'énoncé. Il s'agit toujours de tester la primalité des entiers de 1 à n, à l'aide d'une boucle `for`. Toutefois, plutôt que de faire appel à la fonction `estPremier`, peu efficace, on teste uniquement la divisibilité par les éléments stockés dans le tableau `premier` (c'est le rôle de la boucle `while`). L'arrêt des tests de divisibilité se fait soit lorsque la liste des nombres premiers est épuisée, soit lorsque le facteur est supérieur à la partie entière de la racine carrée de n.

```
petitsPremiers2:=proc(n)
local i,j,k,test,max;
global premier;
premier[1]:=2;      # 2 est le premier nombre a stocker
k:=1;               # le tableau premier contient 1 nombre premier
max:=sqrt(n);
for i from 3 by 2 to n do
  test:=1;          # test vaut 1 tant que n est supposé premier
  j:=1;
  while j<k and premier[j] <= max do
    if mod(i,premier[j]) = 0 then  # le j-ieme nb premier divise i
      test:=0;                     # i n'est donc pas premier
    fi;
    j:=j+1;
  od;
  if test=1 then    # les tests prouvent que i est premier
    k:=k+1;         # on incrémente le compteur
    premier[k]:=i;  # et on stocke i
  fi;
od;
return k;
end;
```

> Le calcul de `sqrt(n)` est coûteux en temps. Stocker cette valeur dès le départ dans une variable `max` permet d'éviter de l'effectuer à chaque nouvelle boucle.

4 L'algorithme suivant est une traduction directe de la méthode d'Ératosthène, le tableau étant ici donné par la variable `crible`. Les cases sont initalisées à 1, la première case (d'indice 1 en Maple) correspond à l'entier 2, la seconde à 3 et ainsi de suite. Rayer une case revient à passer sa valeur à 0.

La variable `p` simule le premier caillou : elle pointe sur la dernière case examinée (initialement la première case, soit l'entier 2). Elle est incrémentée de 1 jusqu'à trouver une case non rayée. Dès que l'on tombe sur une telle case, son indice augmenté