# X Informatique MP/PC 2004 — Corrigé

Ce corrigé est proposé par Jean-Baptiste Rouquier (ENS Lyon) ; il a été relu par Walter Appel (Professeur en CPGE) et Vincent Puyhaubert (Professeur en CPGE).

Cette épreuve n'est pas nouvelle à l'École Polytechnique, mais c'est la première année qu'elle est obligatoire. Elle est réservée aux candidats de la filière MP n'ayant pas suivi l'option informatique et aux candidats de la filière PC. Elle ne compte que pour l'admission.

Le thème abordé par le sujet est celui de la compression, qui permet de réduire la taille d'une donnée (texte, image, vidéo, etc.). Il existe deux grandes familles d'algorithmes de compression : avec ou sans perte d'information, selon que la transformation est bijective ou non. Ce sujet donnera une idée générale des méthodes non destructives, qui sont par exemple utilisées dans le format zip ou gzip.

Toutes les questions demandent d'écrire du code. Nous avons choisi de le rédiger en Maple car ce langage est le plus connu dans les filières MP et PC. Les fonctions réalisées étant simples, vous pourrez sans peine les réécrire dans votre langage préféré.

- On trouve tout d'abord deux questions faciles de lecture et d'écriture en base trois, pour des nombres strictement inférieurs à 9.
- La deuxième partie demande de manipuler des textes, qui sont représentés par des tableaux d'entiers. Le choix de la base trois n'a aucune incidence.
- La troisième partie utilise les fonctions de la première. La question 6, plutôt facile, demande d'écrire une fonction auxiliaire ; les questions 7 et 8 traitent enfin de compression et décompression. Ces deux dernières questions réutilisent toutes les fonction précédemment écrites.

Soyez rigoureux dans la gestion des indices des tableaux : un décalage d'une case apparaît facilement. Des erreurs dans ce domaine sont sanctionnées par le correcteur. Les notions de programmation sont élémentaires : tableaux, boucles `for` et `while`, variables locales et globales. Certaines fonctions s'écrivent naturellement de façon récursive.

Ce sujet est très progressif, de particulièrement simple au début à difficile dans les deux dernières questions. La question 6, facile, fait exception. Il constitue une bonne préparation au concours de l'École Polytechnique et peut même être utile à des candidats ayant suivi l'option informatique – s'ils le traitent rapidement.

# Indications

## Nombres ternaires

2 $0 \leqslant x \leqslant 8$ donc l'écriture de $x$ a exactement deux chiffres selon les conventions de l'énoncé. `poidsFort(`$x$`)` est le quotient de la division euclidienne de $x$ par 3, `poidsFaible(`$x$`)` en est le reste.

## Textes ternaires

3 On peut écrire une fonction récursive : `longueurMotif(`$t, i, j, m$`)` s'exprime en fonction de `longueurMotif(`$t, i+1, j+1, m-1$`)`. Ne pas oublier la condition d'arrêt.

4 Le sujet demande une complexité quadratique en N ; on peut donc utiliser la fonction précédente un nombre de fois linéaire en N.

Seconde indication : ici aussi on peut écrire une fonction récursive, en notant que `longueurMotifMax(`$t, i, j, m$`)` s'exprime en fonction de `longueurMotif(`$t, i, j, m$`)` et de `longueurMotifMax(`$t, i+1, j, m$`)`.

5 Dans la fonction précédente, remplacer les valeurs renvoyées (par exemple la valeur `max(len, longueurMotifMax(t, i+1, j, m))`) par une affectation des variables globales.

Seconde indication : remplacer l'appel à `max` par les deux cas possibles au moyen d'une structure `if ... then ... else ... fi`.

## Compression

7 Le « régime de croisière » de l'énoncé est une boucle `while`. Utiliser une variable locale pour stocker la position de la fenêtre. « Jusqu'au bout du tableau » peut être codé par « jusqu'à ce qu'on lise un caractère non ternaire ».

8 Utiliser une variable locale `fenetre` stockant les dix-huit éléments de la fenêtre, et la mettre à jour à chaque fois que l'on imprime un caractère. Utiliser une autre variable indiquant la position dans $tc$ des cinq caractères codant le triplet $(\mathrm{A}, \mathrm{L}, \mathrm{C})$ en cours de décodage. Pour chacun de ces triplets, lire caractère par caractère dans `fenetre` la chaîne désignée par ce triplet et l'imprimer.

## Nombres ternaires

**1** L'écriture $\underline{bc}$ représente l'entier $3\,b + c$.

```
entier := (b,c) -> 3*b+c;
```

**2** Comme suggéré par l'énoncé qui suppose dans l'introduction l'existence des deux opérations `div` et `mod`, `poidsFort(`$x$`)` est le quotient dans la division euclidienne de $x$ par 3, `poidsFaible(`$x$`)` en est le reste.

```
poidsFort := x -> x div 3;
poidsFaible := x -> x mod 3;
```

> La fonction `div` proposée par l'énoncé n'existe pas en Maple, face à un ordinateur on écrira à la place :
>
> ```
> poidsFort := x -> iquo(x,3);
> poidsFaible := x -> x mod 3;
> ```

## Textes ternaires

**3** Écrivons une fonction récursive : si $t[i]$ est différent de $t[j]$, la longueur cherchée est nulle. Sinon, c'est 1 plus la longueur de la plus grande chaîne démarrant en $i+1$ et égale à une chaîne démarrant en $j+1$. Enfin si $m = 0$, comme on exige $\ell \leqslant m$, le résultat est 0.

```
longueurMotif := proc(t,i,j,m)
  if m = 0 then 0;
  elif t[i] = t[j] then  1 + longueurMotif(t, i+1, j+1, m-1);
  else 0;
  fi;
end;
```

Cette fonction est bien linéaire en N car il y a au plus N appels récursifs et chaque appel n'effectue qu'un nombre constant d'opérations élémentaires.

**4** Il y a deux cas possibles : soit la chaîne la plus grande est celle commençant en $i$ (pour $k = 0$), soit c'est la chaîne la plus grande parmi celles commençant en $i+1+k$, avec $i + 1 + k < j$. Ceci suggère une fonction récursive dont la condition d'arrêt est la suivante : si $i = j - 1$, alors la seule chaîne possible est celle commençant en $i$.

```
longueurMotifMax := proc(t,i,j,m)
  local len;
  len := longueurMotif(t,i,j,m);
  if i = j-1 then len;
  else max(len, longueurMotifMax(t, i+1, j, m));
  fi;
end;
```

Cette fonction est bien quadratique en N car il y a au plus $j - i \leqslant$ N appels récursifs et chaque appel utilise la fonction linéaire `longueurMotif`.

> On pourrait bien sûr écrire à la place une fonction non récursive qui calcule `longueurMotif`$(t, i+k, j, m)$ pour tous les $k$ autorisés et renvoie le maximum des valeurs trouvées.

**5** Adaptons directement la fonction précédente. Dans le cas où $i = j - 1$, au lieu de renvoyer `len`, on assigne les bonnes valeurs aux variables locales.

Sinon (c'est le cas général, qui correspond au code encadré par `else` ... `fi`), on fait un appel récursif (`motifMax(t, i+1, j, m)`) comme dans la fonction précédente, qui renvoie son résultat dans les variables globales `A`, `L` et `C`. On distingue alors les deux cas du `max` :

- si `len` $<$ L (L est égal au résultat qui aurait été renvoyé par `longueurMotifMax(t, i+1, j, m)`), alors `L` et `C` contiennent déjà la bonne valeur et il suffit donc d'indiquer que la chaîne de longueur maximale démarre en $i + k + 1$, c'est-à-dire d'incrémenter `A` ;
- sinon (`len` $\geqslant$ L), c'est que la chaîne cherchée démarre en $i$ et l'on règle alors les variables globales.

```
motifMax := proc(t,i,j,m)
  global A,L,C;
  local len;
  len := longueurMotif(t,i,j,m);
  if i = j-1
  then A:=0; L:= len; C:= t[j+len];
  else
    motifMax(t, i+1, j, m);
    if len < L
    then A := A+1;
    else A := 0; L:= len; C:= t[j+len]
    fi;
  fi;
end;
```

> Il faut préciser à Maple que les variables `A`, `L` et `C` sont globales par la ligne « `global A,L,C ;` ».
>
> Ici aussi on peut écrire une fonction non récursive, sur le même principe que la remarque de la question 4. La voici :
>
> ```
> motifMax := proc(t,i,j,m)
>   global A,L,C;
>   local len,k;
>   L := -1;
>   for k from 0 to j-i-1 do
>     len := longueurMotif(t,i+k,j,m);
>     if len > L
>     then A:= k; L:= len; C:= t[j+len];
>     fi;
>   od;
> end;
> ```