



10MP04

## CONCOURS ARTS ET MÉTIERS ParisTech - ESTP - ARCHIMEDE

### Épreuve d'Informatique MP

**Durée 3 h**

**Si, au cours de l'épreuve, un candidat repère ce qui lui semble être une erreur d'énoncé, d'une part il le signale au chef de salle, d'autre part il le signale sur sa copie et poursuit sa composition en indiquant les raisons des initiatives qu'il est amené à prendre.**

---

**L'usage de calculatrices est interdit.**

Indiquer en tête de copie ou de chaque exercice le langage utilisé

Quel que soit le langage utilisé, le candidat pourra supposer qu'il dispose d'une fonction `test_liste_vide` qui prend en entrée une liste et renvoie le booléen 1 si la liste est vide et 0 sinon.

Le candidat pourra, s'il le juge utile, supposer que chaque liste se termine par un élément de marquage de fin de liste appelé *NIL*.

Dans tous les exercices, on considérera que les éléments d'un tableau de longueur  $n$  sont indicés de 1 à  $n$ . Le candidat pourra supposer qu'il en est ainsi dans le langage qu'il utilise.

## Exercice 1

a) Ecrire la fonction :

---

```
verification_liste
    données  L : liste d'entiers naturels
    résultat b : booléen
```

---

qui prend en entrée une liste  $L$  d'entiers naturels et renvoie le booléen 0 (ou `true`) lorsqu'au moins un des éléments de la liste n'est pas égal à 0, 1, ou 2 et 1 (ou `false`) sinon.

b) Ecrire la fonction :

---

```
tri_tableau
    données N : entier naturel
           T : tableau d'entiers naturels de longueur N
    résultat U : tableau d'entiers naturels de longueur N
```

---

qui prend en entrée un tableau  $T$  qui ne contient que des 0, 1 ou 2 et renvoie le tableau trié dans l'ordre croissant. On ne créera pas de tableau intermédiaire à l'intérieur du programme et on proposera un algorithme en temps linéaire.

## Exercice 2

Une application  $\varphi$  de l'ensemble  $\{1, 2, \dots, 100\}$  dans lui-même est représentée par un tableau  $T$  à cent cases : Pour tout entier  $i$  dans  $\{1, 2, \dots, 100\}$ , la  $i$ -ème case du tableau  $T$  contient la valeur  $\varphi(i)$ .

Ecrire un programme `inverse` qui prend en entrée le tableau  $T$  et retourne le tableau  $U$  représentant l'application inverse  $\varphi^{-1}$  lorsque  $\varphi$  bijective ; dans le cas contraire, le programme affiche le message "NON BIJECTIF".

## Exercice 3

En plus de la fonction `test_liste_vide` qui prend en entrée une liste et renvoie le booléen 1 si la liste est vide et 0 sinon, on dispose des diverses fonctions et procédures :

$x \text{ MOD } n$  retourne le reste de la division euclidienne de l'entier naturel  $x$  par l'entier naturel non nul  $n$ .

`ajouter( $y, L$ )` ajoute l'élément  $y$  en tête de la liste  $L$ .

`supprimer_tête( $L$ )` supprime la tête de la liste  $L$ .

1. On considère le programme suivant qui prend en entrée une liste contenant des listes de longueur fixée ne contenant que des 0 ou des 1 et retourne une liste de même type. On remarquera qu'une liste de listes peut contenir uniquement la liste vide.

---

```

PROG( $L$  : liste de listes,  $k$  : entier naturel non nul )
variables :  $x$  : entier,  $U$  : liste de listes,  $V0$  : liste de listes,  $V1$  : liste de listes.
 $W0$  : liste de listes,  $W1$  : liste de listes,  $L1$  : liste d'entiers,
 $U \leftarrow L$ 
 $V0 \leftarrow \text{liste\_vide}$ 
 $V1 \leftarrow \text{liste\_vide}$ 
 $W0 \leftarrow \text{liste\_vide}$ 
 $W1 \leftarrow \text{liste\_vide}$ 
tant que test_liste_vide( $U$ ) = 0 faire
     $L1 \leftarrow \text{tête}(U)$ 
     $x \leftarrow \text{tête}(L1)$ 
    si ( $x = 0$ ) alors ajouter(supprimer_tête( $L1$ ),  $V0$ )
    sinon ajouter(supprimer_tête( $L1$ ),  $V1$ )
    fin si
    supprimer_tête( $U$ )
    fin faire
si ( $k > 1$ ) faire  $V1 \leftarrow \text{PROG}(V1, k - 1)$ 
     $V0 \leftarrow \text{PROG}(V0, k - 1)$ 
fin si
tant que test_liste_vide( $V1$ ) = 0 faire
     $L1 \leftarrow \text{tête}(V1)$ 
    ajouter(1,  $L1$ )
    ajouter( $L1$ ,  $W1$ )
    supprimer_tête( $V1$ )
    fin faire
tant que test_liste_vide( $V0$ ) = 0 faire
     $L1 \leftarrow \text{tête}(V0)$ 
    ajouter(0,  $L1$ )
    ajouter( $L1$ ,  $W0$ )
    supprimer_tête( $V0$ )
    fin faire
tant que test_liste_vide( $W0$ ) = 0 faire
    ajouter(tête( $W0$ ),  $U$ )
    supprimer_tête( $W0$ )
    fin faire
tant que test_liste_vide( $W1$ ) = 0 faire
    ajouter(tête( $W1$ ),  $U$ )
    supprimer_tête( $W1$ )
    fin faire
retourner( $U$ )

```

---

- Décrire l'exécution du programme PROG ci-dessous pour l'entrée  
 $L = [[1, 0, 1], [0, 1, 1], [0, 0, 0], [1, 1, 0]], k = 1$  :
- Décrire l'exécution du programme PROG ci-dessous pour l'entrée  
 $L = [[1, 0, 1], [0, 1, 1], [0, 1, 0], [1, 0, 0]], k = 3$  :
- Que calcule le programme PROG ?

d) Démontrer que le programme PROG termine.

2. Qu'effectue la fonction ci-dessous :

---

```
FONCTION1( $n$  : entier naturel,  $k$  : entier naturel)
variables :  $x$  : entier,  $y$  : entier,  $i$  : entier,  $L$  : liste d'entiers.
 $L \leftarrow \text{liste\_vide}$ 
 $x \leftarrow n$ 
pour  $i$  de 1 à  $k$  faire
     $y \leftarrow x \bmod 2$ 
    ajouter( $y, L$ )
     $x \leftarrow (x - y)/2$ 
fin faire

retourner  $L$ 
```

---

3. Qu'effectue la fonction ci-dessous :

---

```
FONCTION2( $L$  : liste d'entiers naturels,  $k$  : entier naturel)
variables :  $x$  : entier,  $y$  : entier,  $i$  : entier,  $L1$  : liste d'entiers,  $L2$  : liste d'entiers,  $U$  : liste de listes.
 $L1 \leftarrow L$ 
 $U \leftarrow \text{liste\_vide}$ 
tant que test_liste_vide( $L1$ ) = 0 faire
     $x \leftarrow \text{tête}(L1)$ 
     $L2 \leftarrow \text{FONCTION1}(x, k)$ 
    ajouter( $L2, U$ )
    supprimer_tête( $L1$ )
fin faire

retourner  $U$ 
```

---

4. a) Décrire l'exécution de la fonction FONCTION2 sur la liste  $L = [6, 0, 3, 5]$  et l'entier  $k = 3$  ? Quel est le résultat de  $\text{PROG}(\text{FONCTION2}(L, 3), 3)$  ?
- b) On souhaite utiliser la fonction FONCTION1, la FONCTION2 et le programme PROG pour trier par ordre croissant des listes d'entiers naturels. Quelle(s) fonction(s) faut-il ajouter? Quel est le domaine de validité de ce programme de tri ?

## Exercice 4

Soit  $\Sigma$  sur un alphabet fini. Soient  $u$  et  $v$  deux mots sur l'alphabet  $\Sigma$ . Soient  $a_1, \dots, a_m, b_1, \dots, b_n$  les lettres de  $\Sigma$  telles que  $u = a_1 \dots a_m$  et  $v = b_1 \dots b_n$ .

On dit que  $u$  est un *sous-mot* de  $v$  si  $u$  est le mot vide  $\epsilon$  ( $m = 0$ ) ou s'il existe une application strictement croissante de  $\{1, \dots, m\}$  dans  $\{1, \dots, n\}$  notée  $\varphi$ , telle que  $a_i = b_{\varphi(i)}$ , pour tout entier  $i$  compris entre 1 et  $m$ .

1. Dans cette question,  $\Sigma$  est l'alphabet à deux lettres  $\{x, y\}$ . Déterminer l'ensemble des sous-mots du mot  $xyxyx$ . Combien y-en a-t'il?
2. Dans le cas général, démontrer que l'ensemble des sous-mots du mot  $v = b_1 \dots b_n$  est fini et proposer une majoration de son cardinal en fonction de  $n$ .
3. On suppose que  $u$  est un sous-mot de  $v$ . Soit  $M(u, v)$  l'ensemble des applications  $\varphi$ ,  $\varphi$  application strictement croissante de  $\{1, \dots, m\}$  dans  $\{1, \dots, n\}$  telle que  $a_i = b_{\varphi(i)}$ , pour tout entier  $i$  compris entre 1 et  $m$ .
  - a) Soit  $j$  le plus petit indice dans  $\{1, \dots, n\}$  tel que  $b_j = a_1$ . Démontrer que toute application  $\varphi$  dans  $M(u, v)$  vérifie  $\varphi(1) \geq j$ .
  - b) Démontrer qu'il existe une application  $\varphi_0$  dans  $M(u, v)$  telle que, pour toute application  $\varphi$  dans  $M(u, v)$  et tout indice  $i$  dans  $\{1, \dots, m\}$ ,  $\varphi(i) \geq \varphi_0(i)$ .
  - c) Ecrire un algorithme qui prend en entrée les listes  $U = [a_1, \dots, a_m]$  et  $V = [b_1, \dots, b_n]$  et qui renvoie la liste  $[\varphi_0(m), \varphi_0(m-1), \dots, \varphi_0(1)]$ .
  - d) Ecrire un algorithme qui prend en entrée les listes  $U = [a_1, \dots, a_m]$  et  $V = [b_1, \dots, b_n]$  et qui renvoie un booléen égal à 1 si  $u$  est un sous-mot de  $v$ , 0 sinon. L'algorithme ne doit parcourir qu'une fois chacune des listes  $U$  et  $V$ . Estimer la complexité de votre algorithme en fonction des entiers  $m$  et  $n$ .
4. Le cardinal de l'ensemble  $M(u, v)$  est noté  $[u, v]$ . Soient  $u'$  et  $v'$  les mots tels que  $u = a_1 u'$  et  $v = b_1 v'$ .
  - a) Lorsque  $a_1 = b_1$ , démontrer l'égalité  $[u, v] = [u', v'] + [u, v']$ .
  - b) Lorsque  $a_1 \neq b_1$ , démontrer l'égalité  $[u, v] = [u, v']$ .
  - c) Proposer un algorithme **récuratif** qui prend en entrée les listes  $U = [a_1, \dots, a_m]$  et  $V = [b_1, \dots, b_n]$  et qui renvoie la valeur de  $[u, v]$ .
5. Dans cette partie, le mot  $u = a_1 \dots a_m$  est fixé. On considère  $\mathcal{L}(u)$  le langage formé par les mots  $v$  de  $\Sigma^*$  tels que  $u$  est un sous-mot de  $v$ .
  - a) Dans le cas particulier où  $\Sigma$  est l'alphabet à deux lettres  $\{x, y\}$  et  $u = xy$ , on remarquera que le langage  $\mathcal{L}(xy)$  est le langage reconnu par l'automate  $\mathcal{A}$  représenté dans la figure 1 ci-dessous. Proposer une expression rationnelle de  $\mathcal{L}(xy)$ .
  - b) Dans le cas général, démontrer que le langage  $\mathcal{L}(u)$  est un langage rationnel et en proposer une expression rationnelle.
  - c) Représenter graphiquement un automate déterministe qui reconnaît le langage  $\mathcal{L}(u)$ . On démontrera précisément que le langage reconnu est  $\mathcal{L}(u)$ .

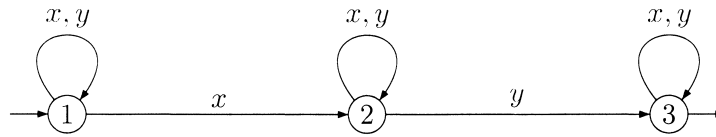


Figure 1: Automate  $\mathcal{A}$

### Exercice 5

La fonction XOR (ou exclusif) est notée  $\oplus$ .

Soit  $f$  la fonction booléenne définie sur  $\{0, 1\}^4$  par  $f(x, y, z, t) = 1$  si et seulement si le nombre de 1 parmi  $x, y, z, t$  est impair, pour tout  $(x, y, z, t)$  dans  $\{0, 1\}^4$ .

1. Ecrire la table de vérité de la fonction  $\oplus$ .
2. Démontrer la propriété d'associativité de la fonction  $\oplus$ , c'est-à-dire :

$$\forall (x, y, z) \in \{0, 1\}^3, (x \oplus y) \oplus z = x \oplus (y \oplus z).$$

3. Ecrire  $f$  avec une formule booléenne.
4. Construire un circuit logique implémentant la fonction  $f$  utilisant uniquement des portes XOR (ou exclusif).
5. Etant donné un entier naturel  $n$  non nul, construire un circuit logique qui prend en entrée  $n$  bits et donne en sortie la parité du nombre de ces bits égaux à 1 en utilisant au plus  $n - 1$  portes XOR.



Figure 2: Porte XOR (ou exclusif)

FIN DE L'EPREUVE