e3a

11MP04

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 le langage de programmation, Caml ou Pascal, choisi pour l'ensemble du sujet.

- L'épreuve est constituée de trois problèmes totalement indépendants.
- Dans toute la suite, on supposera que les éléments d'un tableau de longueur n sont indexés de 1 à n et ce, **quel que soit le langage choisi** en début de sujet.
- Pour les candidats composant avec Pascal, on suppose disposer du type liste d'entiers **Liste**. La liste vide sera représentée par **nil** et on suppose disposer des fonctions suivantes :
  - **cons** prenant en argument un entier x et une liste d'entiers l et renvoyant une liste d'entiers dont la tête est x et la queue l ;
  - **tete** qui renvoie la tête d'une liste d'entiers ;
  - **queue** qui renvoie la queue d'une liste d'entiers ;

TOURNEZ LA PAGE S.V.P.

1 Décomposition de permutations

On note σ_n l'ensemble des permutations de l'ensemble $[\![1, n]\!]$ où $n \in \mathbb{N}^*$. On représente une telle permutation σ sous la forme d'un tableau p de longueur n où $p[i] = \sigma(i)$ pour tout $i \in [\![1, n]\!]$. Les tableaux considérés dans la suite sont supposés contenir des permutations et on ne vérifiera donc pas qu'ils représentent effectivement une permutation. Pour $(i, j) \in [\![1, n]\!]^2$ avec $i \neq j$, la transposition $\tau_{i,j}$ est la permutation définie par :

$$\tau_{i,j}(i) = j \quad \tau_{i,j}(j) = i \text{ et } \tau_{i,j}(k) = k \text{ si } k \notin \{i, j\}$$

Il est rappelé que toute permutation se décompose en produit de transpositions. Dans la suite, on représentera la transposition $\tau_{i,j}$ par le couple d'entiers (i, j).

1.1 Quelques fonctions auxiliaires

1. Écrire une fonction **swap** prenant en argument deux entiers i et j et un tableau p qui échange les éléments d'indice i et j du tableau p.
2. Écrire une fonction **init** prenant en argument un entier n et renvoyant un tableau représentant la permutation $id_{[\![1,n]\!]}$.
3. Écrire une fonction **build** prenant en argument un entier n et une liste de transpositions $[\tau_1, \tau_2, ..., \tau_k]$ et renvoyant un tableau p de longueur n représentant la permutation $\tau_1 \circ \tau_2 \circ ... \circ \tau_k$.

1.2 Un algorithme de décomposition

On considère l'algorithme suivant :

```
decompose p=
  n:=longueur(p)
  l:=[]
  k:=1;
  tant que (k<=n) faire
     si p[k]=k alors k:=k+1 sinon
        temp:=p[k]; swap temp k p; add (k,temp) l
     fin si
  fin faire
  renvoyer(l)
```

où la fonction **add** permet d'ajouter un élément en tête d'une liste et où [ ] désigne la liste vide.

1. Expliciter le déroulement de l'algorithme sur le tableau $p = [|2; 3; 1|]$. On donnera les valeurs de l, k et p à fin de chaque exécution de la boucle. (On pourra vérifier sur cet exemple que si la valeur finale de l est $[\tau_i, ..., \tau_1]$, on a bien : $\sigma = \tau_i \circ ... \circ \tau_1$.)
2. On note l_i, k_i et p_i les valeurs de l, k et p à la fin de la i-ième exécution de la boucle. On convient que $l_0 = [\,]$, $k_0 = 1$ et $p_0 = p$. On note N_i le nombre de points fixes de p_i et L_i le nombre d'éléments présents dans la liste l_i.
   (a) Prouver que la suite $(k_i + N_i)$ est strictement croissante.
   (b) Démontrer que l'algorithme termine toujours et, plus précisément, que la boucle est exécutée au plus $2n - N_0$ fois, en notant n la longueur du tableau p.
   (c) On note L_f le nombre d'éléments contenus dans la liste l lorsque l'exécution s'achève, montrer que :
   $$L_f \leqslant n - N_0$$
   (d) Démontrer que le programme est correct (i.e. si la liste renvoyée est : $[\tau_i, \tau_{i-1}, ..., \tau_1]$, alors $\sigma = \tau_i \circ ... \circ \tau_1$).

3. Si $\sigma \in \sigma_n$ et si $x \in [\![1, n]\!]$, l'orbite de x est : $\mathcal{O}(x) = \{x, \sigma(x), \sigma^2(x), ...\}$. L'ensemble des orbites forme alors une partition de l'ensemble $[\![1, n]\!]$. Par exemple, si $p = [|2; 1; 3|]$, on a :

$$\mathcal{O}(1) = \{1, \sigma(1) = 2, \sigma(2) = 1, ...\} = \{1, 2\} \quad \mathcal{O}(2) = \{2, \sigma(2) = 1, \sigma(1) = 2, ...\} = \mathcal{O}(1) \text{ et}$$

$$\mathcal{O}(3) = \{3, \sigma(3) = 3, ...\} = \{3\}$$

On a donc la partition suivante : $\{\{1, 2\}, \{3\}\}$.
On note D le nombre d'orbites distinctes de σ. Prouver que $L_f = n - D$.

2 Le jeu de Marienbad

Soit $k \in \mathbb{N}^*$. Sur une table sont disposés k tas d'allumettes : le premier tas contient N_1 allumettes, le second N_2, et le dernier tas contient N_k allumettes. On suppose les N_i strictement positifs. Le jeu se joue à deux. Le premier joueur retire un nombre strictement positif d'allumettes d'un tas, le second joueur fait de même, puis le premier, etc. La partie s'arrête lorsque toutes les allumettes ont été retirées ; le joueur gagnant étant celui qui a pris la ou les dernières allumettes présentes sur la table, l'autre étant déclaré perdant.

2.1 Préliminaires

La table du vérité du «ou exclusif», noté $\oplus$ est :

$\oplus$	0	1
0	0	1
1	1	0

On rappelle que $\oplus$ définit une structure de groupe commutatif sur $\{0, 1\}$. Soient n et m des entiers naturels. Leurs décompositions en base 2 s'écrit :

$$n = \sum_{k \geqslant 0} n_k 2^k \text{ et } m = \sum_{k \geqslant 0} m_k 2^k$$

où les (n_k) et les (m_k) sont à valeurs dans $\{0, 1\}$. On définit alors une loi de composition interne sur $\mathbb{N}$, noté abusivement $\oplus$, par :

$$n \oplus m = \sum_{k \geqslant 0} (n_k \oplus m_k) 2^k$$

et on admet que $(\mathbb{N}, \oplus)$ est un groupe commutatif.

1. Déterminer l'élément neutre de $(\mathbb{N}, \oplus)$.
2. Si $x \in \mathbb{N}$, quel est l'inverse x^{-1} de x ?
3. Soient x_1, x_2,..., x_{n-1} et x_n des entiers naturels tels que $x_1 \oplus x_2 \oplus ... \oplus x_n = 0$. Soit $i_0 \in [\![1, n]\!]$ et soit $z \in \mathbb{N}$ avec $z \neq x_{i_0}$. On pose : $x'_i = x_i$ si $i \neq i_0$ et $x'_{i_0} = z$. Prouver que :

$$\bigoplus_{i=1}^{n} x'_i \neq 0$$

4. Soient x_1, x_2,..., x_{n-1} et x_n des entiers naturels tels que $x_1 \oplus x_2 \oplus ... \oplus x_n \neq 0$. Prouver qu'il existe $i_0 \in [\![1, n]\!]$ et $z \in \mathbb{N}$ avec $z < x_{i_0}$ de sorte que si on pose : $x'_i = x_i$ si $i \neq i_0$ et $x'_{i_0} = z$, on a :

$$\bigoplus_{i=1}^{n} x'_i = 0$$

(Si $\sum_{k \geqslant 0} a_k 2^k$ est le développement binaire de $x_1 \oplus x_2 \oplus ... \oplus x_n$, on pourra considérer :

$$k_0 = \max\{k \in \mathbb{N}, a_k \neq 0\}.)$$

Montrer que pour un tel i_0, on a nécessairement $z = \left(\bigoplus_{i=1}^{n} x_i \right) \oplus x_{i_0}$.

TOURNEZ LA PAGE S.V.P.

5. Écrire une fonction **xorb** prenant en argument deux entiers a et b et renvoyant $a \oplus b$ si a et b sont dans $\{0, 1\}$ et renvoyant -1 sinon.
6. Écrire une fonction **tobin** prenant en argument un entier naturel n et renvoyant une liste $[a_0, a_1, ..., a_p]$ telle que :
$$\forall k \in [0, p], a_k \in \{0, 1\} \text{ et } n = \sum_{k=0}^{p} a_k 2^k$$
7. Écrire une fonction **frombin** prenant en argument une liste $[a_0, ..., a_p]$ d'entiers de $\{0, 1\}$ et renvoyant l'entier n défini par :
$$n = \sum_{k=0}^{p} a_k 2^k$$
(On supposera ne pas disposer de fonction calculant 2^k et on veillera à réaliser un minimum de multiplications.)
8. Écrire une fonction **xor** prenant en argument deux entiers naturels n et m et renvoyant $n \oplus m$. (On prendra garde au fait que les listes renvoyées par la fonction **tobin** sur n et m n'ont pas nécessairement la même longueur.)

2.2 Stratégie gagnante

Une configuration du jeu est entièrement caractérisée par la donnée du nombre d'allumettes de chacun des tas. On code donc une configuration par un k-uplet d'entiers naturels $(N_1, N_2, ..., N_k)$. On dit d'une configuration qu'elle est *favorable* lorsque $N_1 \oplus N_2 \oplus ... \oplus N_k \neq 0$; sinon on dit qu'elle est *défavorable*. On note A et B les deux joueurs.

1. (a) Décider si la configuration $(1, 3, 5, 7)$ est favorable. Si c'est le cas, déterminer tous les coups qu'il est possible de jouer et qui placent le jeu dans une configuration défavorable.
   (b) Même question avec $(1, 3, 4, 7)$.
2. Le joueur A récupère la main et le jeu est dans une configuration favorable. Comment doit jouer A pour être certain de gagner la partie ?
3. Le joueur A récupère la main et le jeu est dans une configuration défavorable. Que peut-il faire ?
4. Écrire une fonction **coupSuivant** qui prend en entrée un tableau contenant une configuration non finale du jeu et qui modifie le tableau de sorte que celui-ci contienne la configuration obtenue après avoir joué un des coups correspondant à la stratégie détaillée lors des deux questions précédentes.

3 Résiduels d'un langage

Soit Σ un alphabet fini et non vide. Si $L \subset \Sigma^*$ et si $u \in \Sigma^*$, on pose :
$$u^{-1}L = \{v \in \Sigma^* | uv \in L\}$$
Un tel ensemble est un *résiduel* du langage L. On se propose de montrer qu'un langage est rationnel si et seulement si celui-ci n'a qu'un nombre fini de résiduels et, si c'est le cas, de construire un automate déterministe le reconnaissant ayant un nombre minimal d'états.

1. Soient $(u, v) \in (\Sigma^*)^2$ et $L \subset \Sigma^*$. Comparer $(uv)^{-1}L$ et $v^{-1}(u^{-1}L)$ en justifiant votre réponse.
2. Dans cette question, on suppose que $\Sigma = \{a, b\}$ et on considère le langage $L = a(ba + a)^*$.
   (a) Représenter graphiquement un automate déterministe reconnaissant le langage L.
   (b) Calculer $u^{-1}L$ pour $u = \epsilon$, a, b, aa, ab, ba et bb.
   (c) Décrire sans justification les différents résiduels de L.

3. On revient au cas général et on se donne un langage $L \subset \Sigma^*$. On suppose que l'ensemble des résiduels de L est fini :

$$\{u^{-1}L | u \in \Sigma^*\} \text{ est un ensemble fini.}$$

On considère alors un automate $\mathcal{A} = (Q, \delta, i_o, F)$ où Q est l'ensemble *fini* des résiduels et où δ est définie par :

$$\forall u \in \Sigma^*, \forall a \in \Sigma, \delta(u^{-1}L, a) = (ua)^{-1}L$$

   (a) En choisissant judicieusement l'état initial et l'ensemble des états finals, démontrer que l'automate ainsi construit reconnaît exactement L et conclure.

   (b) Représenter graphiquement l'automate obtenu si L est le langage de la question 2.

4. Réciproquement, on se donne un langage rationnel L et un automate déterministe $\mathcal{A} = (Q, \delta, i_0, F)$ reconnaissant L. Si $q \in Q$, le langage reconnu par q, noté $\mathcal{L}_q$ est le langage reconnu par l'automate (Q, δ, q, F).

   (a) Soit $u \in \Sigma^*$. On note q_u l'état de l'automate obtenu après lecture du mot u. Établir et prouver un lien entre $u^{-1}L$ et $\mathcal{L}_{q_u}$.

   (b) Démontrer finalement que L n'a qu'un nombre fini de résiduels.

5. Si $\mathcal{A} = (Q, \delta, i, F)$ et si $\mathcal{B} = (Q', \delta', i', F')$ sont deux automates déterministes sur l'alphabet Σ, un morphisme d'automates de $\mathcal{A}$ dans $\mathcal{B}$ est une application $f : Q \to Q'$ vérifiant :

$$\forall q \in Q, \forall s \in \Sigma, f(\delta(q, s)) = \delta'(f(q), s)$$

$$f(i) = i' \text{ et } f(F) \subset F'$$

   (a) Si f est un morphisme de $\mathcal{A}$ dans $\mathcal{B}$, comparer les langages reconnus par $\mathcal{A}$ et $\mathcal{B}$.

   (b) Un morphisme d'automate est dit *surjectif* lorsque f est surjective et lorsque $f^{-1}(F') \subset F$. Reprendre la question précédente en supposant le morphisme surjectif.

   (c) Soit L un langage rationnel. On se donne $\mathcal{A}$ un automate déterministe reconnaissant L et on note $\mathcal{B}$ l'automate construit à la question 3.(a). Construire un morphisme d'automates de $\mathcal{A}$ dans $\mathcal{B}$ et en déduire que le nombre d'états de $\mathcal{A}$ est supérieur ou égal au nombre d'états de $\mathcal{B}$.

   (d) Que dire de plus lorsque $\mathcal{A}$ et $\mathcal{B}$ ont le même nombre d'états ?

FIN DE L'EPREUVE