Concours Centrale - Supélec 2009

Épreuve : INFORMATIQUE **Filière** MP

Calculatrices autorisées

Note : les parties I, II.A, II.B et II.C sont indépendantes.

Partie I - La chasse aux fantômes

Les candidats devront indiquer clairement en tête de copie le langage de programmation choisi (Pascal ou Caml). Les fonctions et procédures demandées seront rédigées dans ce langage.

Un groupe de n chasseurs de fantômes combat un groupe de n fantômes. Chaque chasseur est armé d'un canon à ions, capable d'éradiquer un fantôme d'un coup de rayon. Un rayon se propage en ligne droite et termine sa course en touchant le fantôme. Les chasseurs sont confrontés à deux problèmes : il est nécessaire d'éliminer tous les fantômes en même temps, sinon ils se multiplient pour revenir au nombre initial de n fantômes ; il est très dangereux que deux rayons se croisent, cela ne doit donc pas se produire. Le combat se déroule dans une grotte cylindrique creuse, dont la base est un disque. Les déplacements (chasseurs et fantômes) ne sont possibles que sur un rebord confondu avec la circonférence, mais les tirs se font entre deux points de la circonférence, selon des cordes du cercle.

I.A - Les $2n$ protagonistes sont représentés par des points distincts $P_1, P_2, ..., P_{2n}$ répartis dans l'ordre des indices croissants sur la circonférence d'un cercle parcouru dans le sens trigonométrique ; n sont des chasseurs, n des fantômes, mais, dans cette section I.A, on cherche les liens possibles entre les P_i sans s'occuper de qui est chasseur et qui est fantôme. Les points doivent être reliés deux à deux par une corde du cercle selon une stratégie gagnante pour les chasseurs. Pour cela, il faut donc respecter les deux conditions :

i) chaque point est une extrémité d'une et une seule corde,
ii) les différentes cordes ne doivent pas se couper.

Dans toute la suite on fixe un entier $n > 0$ et on appelle *stratégie de taille n* un vecteur $\big(c[1], c[2], \ldots, c[2n]\big)$, tel que pour tout entier i vérifiant $1 \leqslant i \leqslant 2n$ on ait :

$$1 \leqslant c[i] \leqslant 2n, \qquad c\big[c[i]\big] = i, \qquad c[i] \neq i.$$

Cette notation exprime bien le lien entre les points P_i et P_j lorsque $j = c[i]$.

En Pascal, on représente le vecteur par un tableau avec un type `Vecteur` défini par :

```
Const Nmax = ... ; {de taille suffisante pour tout le problème}
Type Vecteur = array [1..Nmax] of integer ;
```

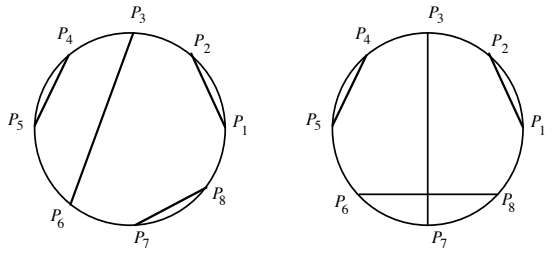
En Caml, on représente le vecteur avec le type « `vecteur` » de Caml. On peut utiliser la fonction `make_vect`. Les éléments des vecteurs en Caml sont numérotés de 0 à $N-1$. On prendra des vecteurs de taille $2n+1$, dont le premier élément ne sera pas utilisé, pour que les points soient bien aux indices 1 à $2n$ dans le vecteur Caml.

On considère des stratégies où les points d'une paire sont reliés par des segments de droites **situés dans un même plan**. Une stratégie est dite « admissible » si les segments ainsi formés ne se coupent pas. On admet que cette propriété ne dépend pas de la position des points sur la circonférence mais seulement de la stratégie.

Par exemple, pour les stratégies de taille 4, c_1 et c_2 ci-contre :

	1	2	3	4	5	6	7	8
c_1	2	1	6	5	4	3	8	7
c_2	2	1	7	5	4	8	3	6

La figure de gauche correspond à c_1 qui est donc admissible, et celle de droite à c_2 qui, elle, ne l'est pas.



I.A.1) Dans le cas $n = 3$, donner le nombre de stratégies possibles, admissibles ou non.

I.A.2) Préciser le nombre de stratégies de taille 3 qui sont admissibles. Les représenter sur une figure.

I.A.3) Déterminer le nombre de stratégies de taille n entier positif (qu'elles soient ou non admissibles).

I.B - Soit i, j, k, l quatre entiers distincts tels que : $1 \leqslant i < j \leqslant 2n$ et $1 \leqslant k < l \leqslant 2n$.

I.B.1) Donner une condition nécessaire et suffisante, portant uniquement sur ces entiers, pour que les segments $[P_iP_j]$ et $[P_kP_l]$ se croisent.

I.B.2) Écrire une fonction « `croise` » prenant les quatre entiers pour paramètres et renvoyant un booléen dont le résultat est `true` si et seulement si les segments $[P_iP_j]$ et $[P_kP_l]$ se croisent. On suppose que les paramètres donnés à la fonction satisfont les conditions données en introduction du I.B.

I.B.3) En utilisant la fonction `croise`, écrire une fonction « `estAdmissible` » qui prend en paramètre le vecteur (et en Pascal, le nombre n de chasseurs) et renvoie un booléen dont le résultat est `true` si la stratégie est admissible.

I.B.4) Donner, en la justifiant, la complexité de cette méthode, en terme de temps de calcul (on se contentera d'une évaluation du type $O(n^\alpha)$, en précisant la valeur de α).

I.C - On souhaite tester la stratégie de façon plus efficace. Pour cela on utilise une fonction « `evalue` » qui prend deux entiers i et j comme paramètres (i et j étant compris entre 1 et $2n$) et dont le résultat est `true` si et seulement si les deux propositions suivantes sont vraies :

- $(i \leqslant k \leqslant j) \Rightarrow (i \leqslant c[k] \leqslant j)$
- les segments ayant leurs extrémités dans l'arc (fermé) $[P_iP_j]$ ne se croisent pas.

I.C.1) Donner la valeur de `evalue(i, j)` dans les trois cas particuliers a, b et c suivants :

a) $j < i$; b) $j \geqslant i$ et $c[i] < i$; c) $j \geqslant i$ et $c[i] > j$.

I.C.2) Pour $i < c[i] < j$, montrer que `evalue(i, j)` se déduit de `evalue(i + 1, c[i] - 1)` et de `evalue(c[i] + 1, j)`.

I.C.3) Écrire une fonction `testStrategie` qui prend en paramètre un vecteur (et en Pascal le nombre n de chasseurs) et renvoie un booléen dont la valeur est `true` si la stratégie est admissible. Cette fonction devra être de complexité $O(n)$.

I.C.4) Démontrer avec soin que la fonction `testStrategie` est bien de complexité $O(n)$.

I.D - La méthode précédente teste si une stratégie est admissible. On souhaite maintenant, le statut (chasseur ou fantôme) des points étant donné, déterminer directement une stratégie admissible, de façon également efficace.

I.D.1) Montrer que si P_i est un chasseur ($1 \leqslant i \leqslant 2n$), alors il existe $j \neq i$, avec $1 \leqslant j \leqslant 2n$ tel que P_j est un fantôme et tel que le nombre de chasseurs soit égal au nombre de fantômes de chaque côté de l'axe P_iP_j.

I.D.2) En déduire un algorithme simple pour construire une stratégie admissible.

I.D.3) On se propose d'évaluer la complexité de cette méthode.

a) Déterminer la complexité dans le pire des cas.

b) Évaluer, sans démonstration, la complexité moyenne.

I.D.4) Un vecteur p contient la nature des points P_i : $p[i] = 1$ pour un chasseur, $p[i] = -1$ pour un fantôme ($1 \leqslant i \leqslant 2n$). On a donc $\sum_{i=1}^{2n} p[i] = 0$ puisqu'il y a autant de fantômes que de chasseurs. Écrire une fonction `cibles`, qui prend en paramètre le vecteur p (et en Pascal, le nombre n de chasseurs) et qui renvoie la liste des indices (i, j) des couples (P_i, P_j) (où P_i est un chasseur et P_j un fantôme), composant une stratégie admissible du problème.

Partie II - Automates finis

On rappelle que les sections II.A, II.B, II.C sont indépendantes et peuvent être traitées dans un ordre quelconque.

Définitions et notations

Un automate (sous entendu fini) déterministe est un quintuplet $\mathscr{A} = (Q, A, i, \delta, F)$ avec :

- Q l'ensemble fini non vide des états ;
- A l'alphabet c'est-à-dire l'ensemble fini non vide des lettres ;
- $i \in Q$ l'état initial ;
- $\delta : Q \times A \to Q$ la fonction de transition (éventuellement partielle) ;
- $F \subseteq Q$ l'ensemble des états terminaux.

Lorsque δ est défini sur l'ensemble $Q \times A$ tout entier, on parle d'automate fini complet. Que l'automate soit complet, ou non, on peut choisir de représenter les transitions non par une fonction de transition δ, mais par un ensemble de transitions $T \subseteq Q \times A \times Q$ (le graphe de δ), avec la condition de déterminisme :

(D) si (q, α, q_1), $(q, \alpha, q_2) \in T$, alors $q_1 = q_2$.

Un automate (fini) non déterministe est un quintuplet $\mathscr{A} = (Q, A, I, \delta, F)$ avec cette fois $I \subseteq Q$ l'ensemble des états initiaux et $\delta : Q \times A \to \mathscr{P}(Q)$ la fonction de transition : si $p \in Q$ et $\alpha \in A$, $\delta(p, \alpha)$ désigne l'ensemble (éventuellement vide) des $q \in Q$ tels qu'il existe une transition étiquetée par α de p vers q. Si on choisit de représenter les transitions par un ensemble inclus dans $Q \times A \times Q$, la condition (D) n'a plus à être vérifiée.

Pour présenter un automate, les candidats pourront, bien entendu, les représenter par un schéma. Ils veilleront alors à bien identifier l'état initial (ou les états initiaux) et les états terminaux par une notation adaptée.

On note A^* l'ensemble des mots écrits dans l'alphabet A (y compris le mot vide). Un langage sur l'alphabet A est un sous-ensemble de A^*. On note en particulier $\mathcal{L}(\mathscr{A})$ le langage de $\mathscr{A}$, c'est-à-dire l'ensemble des mots acceptés par $\mathscr{A}$ (permettant de passer d'un état initial à un état terminal).

Étant donné un mot $u \in A^*$, on note $|u|$ la longueur de u, c'est-à-dire le nombre de ses lettres.

II.A - Mots répétés

II.A.1) Soit A un alphabet. On s'intéresse au langage $L \subseteq A^*$ constitué des mots u qui ne sont de la forme w^2 (c'est-à-dire ww) pour aucun $w \in A^*$:

$$L = \{u \mid \forall w \in A^*,\ u \neq w^2\}.$$

a) On suppose que A est réduit à une lettre.

Montrer alors que L est reconnaissable par automate fini et donner un automate reconnaissant effectivement L, ainsi qu'une expression rationnelle décrivant L.

b) On suppose maintenant que A n'est pas réduit à une lettre. Montrer que L n'est pas reconnaissable.

c) Soit p un entier strictement plus grand que 2. Reprendre les deux questions précédentes en remplaçant L par L_p défini par :

$$L_p = \{u \mid \forall w \in A^*,\ u \neq w^p\}$$

II.B - Logique de Presburger

Soit n un entier strictement positif et $B = \{0,1\}$. On prend comme alphabet l'ensemble B^n. Un mot

$$u = \begin{pmatrix} b_{1,1} \\ \vdots \\ b_{n,1} \end{pmatrix} \cdots \begin{pmatrix} b_{1,m} \\ \vdots \\ b_{n,m} \end{pmatrix} \in (B^n)^*$$

représente le n-uplet d'entiers naturels $(x_1, \cdots, x_n)$ tel que

$$x_i = \sum_{\ell=1}^{m} b_{i,\ell} 2^{\ell-1} \text{ pour tout } 1 \leqslant i \leqslant n.$$

a) Donner une représentation des n-uplets suivants :

(4); $(2,3,0)$; $(2,3,5)$.

On appelle relation dans $\mathbb{N}^n$ tout sous-ensemble de $\mathbb{N}^n$.
On dit qu'une relation $R \subseteq \mathbb{N}^n$ est rationnelle, lorsqu'il existe un automate fini $\mathscr{A}$ qui reconnaît exactement l'ensemble des mots de $(B^n)^*$ représentant les n-uplets de R.

Pour les deux questions b) et c) suivantes, on demande de donner, pour chacune des relations Eg, Inf et Add, un automate fini la reconnaissant effectivement.

b) Montrer que les relations :

$$Eg = \{(x,y) \in \mathbb{N}^2 \mid x = y\} \quad \text{et} \quad Inf = \{(x,y) \in \mathbb{N}^2 \mid x < y\}$$

sont rationnelles.

c) Montrer que la relation :

$$Add = \{(x,y,z) \in \mathbb{N}^3 \mid x + y = z\}$$

est rationnelle.

d) On suppose la relation $R \in \mathbb{N}^n$ rationnelle et on définit les relations $Q \subseteq \mathbb{N}^{n-1}$ et $S \subseteq \mathbb{N}^{n-1}$ par :

$(x_1, \ldots, x_{n-1}) \in Q$ si, et seulement si, il existe $x \in \mathbb{N}$ tel que $(x_1, \ldots, x_{n-1}, x) \in R$.

$(x_1, \ldots, x_{n-1}) \in S$ si, et seulement si, pour tout $x \in \mathbb{N}$, $(x_1, \ldots, x_{n-1}, x) \in R$.

Montrer que les relations Q et S sont rationnelles.

II.C - Automate minimal

Soit $A = \{a,b\}$. Pour chaque entier $N \in \mathbb{N}$, on définit les langages G_N et D_N de la manière suivante :

- $G_N = \{uav \mid u \in A^N,\ v \in A^*\}$.
- $D_N = \{uav \mid u \in A^*,\ v \in A^N\}$.

II.C.1) Donner un automate déterministe complet, à $N+3$ états reconnaissant G_N.

II.C.2) Donner un automate non-déterministe, à $N+2$ états reconnaissant D_N.

II.C.3) Démontrer qu'il n'existe pas d'automate déterministe complet reconnaissant G_N et possédant strictement moins de $N+3$ états.

On pourra raisonner par l'absurde et prouver que les états atteints en lisant a^k depuis l'état initial sont distincts, pour certaines valeurs k.

II.C.4) Déterminiser l'automate proposé dans la question C.2), lorsque $N = 1$.

II.C.5) Démontrer qu'il n'existe pas d'automate déterministe complet reconnaissant D_N et possédant strictement moins de 2^{N+1} états.

II.C.6) Démontrer qu'il n'existe pas d'automate fini (non-déterministe) reconnaissant D_N et possédant strictement moins de $N+2$ états. Même question pour le langage G_N.

• • • FIN • • •