CONCOURS COMMUNS POLYTECHNIQUES

ÉPREUVE SPÉCIFIQUE-FILIÈRE MP

INFORMATIQUE

DURÉE : 2 heures

Les calculatrices programmables et alphanumériques sont autorisées, sous réserve des conditions définies dans la circulaire n° 86-228 du 28 juillet 1986.

PARTIE I. EPREUVE DE LOGIQUE

Résolution d'une énigme par la logique des propositions

Un inspecteur des services de la santé visite un hôpital psychiatrique où des phénomènes étranges lui ont été signalés. Dans cet hôpital, il n'y a que des médecins et des malades, mais ceux-ci peuvent être indifféremment sains d'esprit ou totalement fous. Sa tâche est de mettre fin à cette situation en faisant sortir de l'hôpital les personnes qui n'ont rien à y faire, d'une part les patients sains d'esprit, et d'autre part les médecins fous.

On distingue les individus sains d'esprit des personnes folles de la façon suivante :

- Les premiers n'affirment que des choses vraies.
- Les secondes ne disent que des choses fausses.

Dans une des salles de l'hôpital, l'inspecteur interroge deux personnes que nous nommerons A et B pour préserver leur anonymat. Il constate que :

A affirme que B est fou (1)
B affirme que A est un médecin (2)

Après quelques minutes de réflexion, l'inspecteur décide de faire sortir l'un des deux de l'hôpital.

Si AF (resp. BF) est la proposition "A est fou" (resp. "B est fou"),
si AM (resp. BM) est la proposition "A est un médecin" (resp. "B est un médecin"), alors :

1. Exprimer par une formule de la logique des propositions chacune des deux assertions (1) et (2).

2. En utilisant exclusivement les équations sur les formules logiques (lois de Morgan, associativité, distributivité, ...), déterminer qui de A ou de B l'inspecteur a fait sortir.

3. Peut-on affirmer quelque chose au sujet de celui qui restera dans l'hôpital ?

PARTIE II. EPREUVE D'ALGORITHMIQUE

Le candidat traitera cette deuxième partie en utilisant selon sa préférence soit le langage Caml, soit le langage Pascal.

Version en langage Caml

Partition et union d'arbres binaires de recherche

Un arbre binaire étiqueté par des entiers peut être défini par le type Caml :

```
type int_arbre = Nil | Noeud of int * int_arbre * int_arbre;;
```

Soit A un tel arbre, on notera Etiq(A) l'ensemble des étiquettes de l'arbre A défini récursivement par :

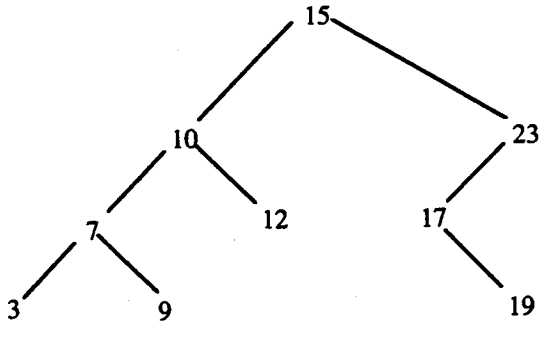
Etiq(*Nil*) = { }
Etiq(*Noeud*(i, g, d)) = $\{i\} \cup$ Etiq(g) $\cup$ Etiq(d)

Un arbre binaire de recherche A (en abrégé ABR), étiqueté par des entiers **distincts**, est un arbre qui vérifie la propriété suivante :

$\forall$ n, noeud de A, de la forme n = Noeud(i, g, d) alors
$\forall\, j \in$ Etiq(g) $\quad j < i$
$\forall\, j \in$ Etiq(d) $\quad j > i$

Exemple : A = Noeud(15,
Noeud(10,
Noeud(7, Noeud(3,Nil,Nil), Noeud(9,Nil,Nil)),
Noeud(12,Nil,Nil)
)
Noeud(23,
Noeud(17,Nil,Noeud(19,Nil,Nil)),
Nil
)
)

est un ABR qui peut être représenté graphiquement par :



Tous les arbres qui seront considérés dans ce sujet sont des arbres binaires de recherche.

On appelle coupure d'un ABR A selon une valeur c une partition de A en un couple de sous-arbres (G_A, D_A) tels que :

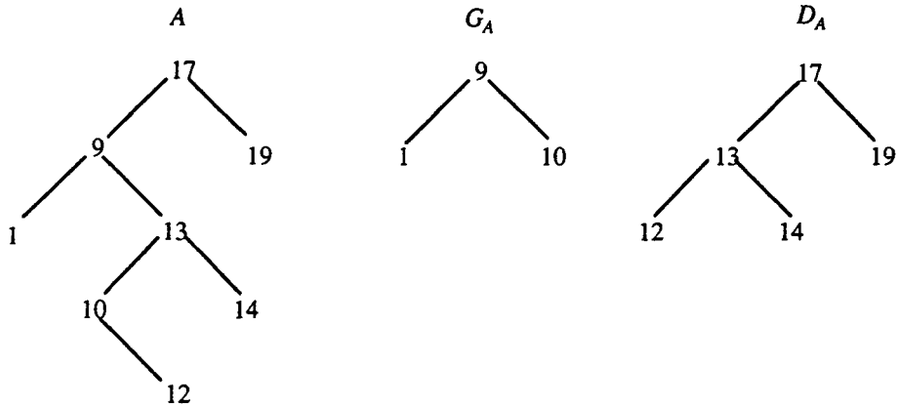
G_A et D_A sont deux ABR

$\text{Etiq}(A) = \text{Etiq}(G_A) \cup \text{Etiq}(D_A)$

$\forall j \in \text{Etiq}(G_A) \quad j \leq c$

$\forall j \in \text{Etiq}(D_A) \quad j > c$

Exemple :



Une coupure de A par rapport à 10

1. On remarque que (Nil, Nil) est une coupure de Nil selon c (pour tout c).

   Soit Noeud(c, g, d) un ABR, montrer que (Noeud(c, g, Nil), d) est une coupure de Noeud(c, g, d) selon c.

2. Soient i un entier, Noeud(i, A, B) un ABR et c un entier. Montrer que si $c < i$ et si (G_A, D_A) est une coupure de A selon c, alors (G_A, Noeud(i, D_A, B)) est une coupure de Noeud(i, A, B) selon c.

3. Par symétrie, construire une coupure de l'ABR Noeud(i, A, B) selon c, dans le cas où $c > i$.

4. Ecrire une fonction récursive Caml **couper**: int-> int_arbre-> (int_arbre*int_arbre)
   telle que (**couper** c a) renvoie une coupure de a selon c.

5. Si $|a|$ désigne la hauteur de l'arbre a, montrer que la complexité de la fonction (**couper** c a), en nombre d'appels récursifs, est inférieure ou égale à $|a|$.

6. En utilisant la fonction **couper**, écrire une fonction non récursive Caml
   inserer_racine: int-> int_arbre-> int_arbre telle que si b = (**inserer_racine** i a) alors
   $\text{Etiq}(b) = \text{Etiq}(a) \cup \{i\}$ et b est un ABR dont la racine est étiquetée par i. On supposera que les arguments i et a de la fonction sont tels que $i \notin \text{Etiq}(a)$.

7. Ecrire une fonction récursive Caml **min**: int_arbre -> int qui renvoie l'étiquette minimale d'un ABR différent de Nil.

8. En utilisant les fonctions **min** et **couper**, écrire une fonction non récursive Caml
union: int_arbre-> int_arbre-> int_arbre telle que
si a et b sont deux ABR tels que les étiquettes de a sont toutes inférieures aux étiquettes de b,
c'est-à-dire, $((\min b) > x, \forall x \in \text{Etiq}(a))$
alors si c = (**union** a b) alors c est un ABR et $\text{Etiq}(c) = \text{Etiq}(a) \cup \text{Etiq}(b)$

9. Donner un majorant de la complexité de (**union** a b) dans le pire cas en fonction de $|a|$ et $|b|$, en comptant uniquement les appels récursifs de fonction.

Version en langage Pascal

Partition et union d'arbres binaires de recherche

On suppose prédéfinis en Pascal les types simples Int_arbre (arbre binaire étiqueté par des entiers) et Couple_arbre (couple d'arbres binaires étiquetés par des entiers).

- **nil** représente l'arbre vide,
- function **noeud** (x:integer; g, d : Int_arbre): Int_arbre
  est une fonction qui renvoie un arbre dont la racine est étiquetée par x et dont les sous-arbres respectivement gauche et droit sont g et d,
  function **etiquette** (a : Int_arbre):integer
  est une fonction qui renvoie l'étiquette de la racine de a
  function **gauche** (a : Int_arbre): Int_arbre
  est une fonction qui renvoie le sous-arbre gauche de a
  function **droit** (a : Int_arbre): Int_arbre
  est une fonction qui renvoie le sous-arbre droit de a
  function **couple** (g, d : Int_arbre) : Couple_arbre
  est une fonction qui renvoie un couple d'arbres dont le premier élément est g et le second d
  function **premier** (c : Couple_arbre) : Int_arbre
  est une fonction qui renvoie le premier arbre du couple
  function **second** (c : Couple_arbre) : Int_arbre
  est une fonction qui renvoie le second arbre du couple

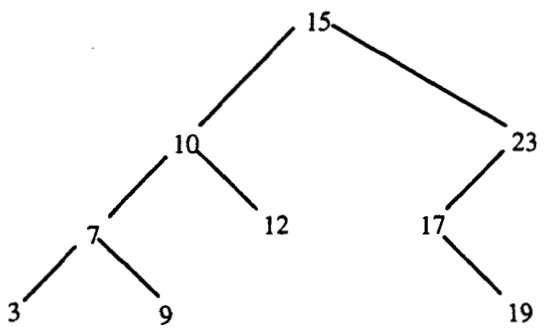
Soit A un Int_arbre, on notera Etiq(A) l'ensemble des étiquettes de l'arbre A défini récursivement par :
Si A = nil alors Etiq (A) = { }
Si A est le résultat de la fonction noeud(i, g, d)
alors $\text{Etiq}(A) = \{i\} \cup \text{Etiq}(g) \cup \text{Etiq}(d)$

Un arbre binaire de recherche A (en abrégé ABR), étiqueté par des entiers **distincts**, est un arbre qui vérifie les propriétés suivantes :
L'arbre vide est un ABR
Si A est le résultat de la fonction noeud(i, g, d) alors
$\forall j \in \text{Etiq}(g) \quad j < i$
$\forall j \in \text{Etiq}(d) \quad j > i$
g et d sont des ABR

Exemple : L'arbre A correspondant à l'expression

```
noeud(15,
        noeud(10,
                noeud(7, noeud(3, nil, nil), noeud(9, nil, nil)),
                noeud(12, nil, nil)
                )
        noeud(23,
                noeud(17, nil, noeud(19, nil, nil)),
                nil
                )
        )
```

est un ABR qui peut être représenté graphiquement par :



Tous les arbres qui sont considérés dans ce sujet sont des arbres binaires de recherche.

On appelle coupure d'un ABR A selon une valeur c une partition de A en un couple de sous-arbres (G_A, D_A) tels que :

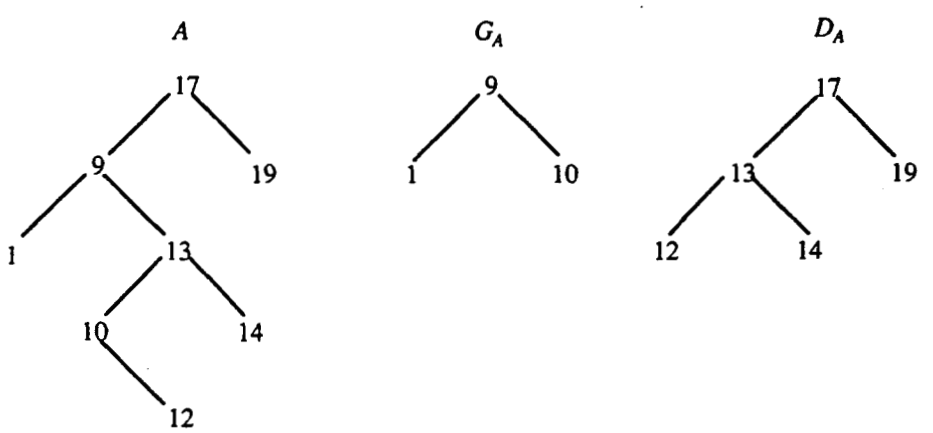
G_A et D_A sont deux ABR

$\text{Etiq}(A) = \text{Etiq}(G_A) \cup \text{Etiq}(D_A)$

$\forall\, j \in \text{Etiq}(G_A) \qquad j \leq c$

$\forall\, j \in \text{Etiq}(D_A) \qquad j > c$

Exemple :



Une coupure de A par rapport à 10

1. On remarque que (nil, nil) est une coupure de nil selon c (pour tout c).

   Soit noeud(c, g, d) un ABR, montrer que (noeud(c, g, nil), d) est une coupure de noeud(c, g, d) selon c.

2. Soient i un entier, Noeud(i, A, B) un ABR et c un entier. Montrer que si $c < i$ et si (G_A, D_A) est une coupure de A selon c, alors (G_A, Noeud(i, D_A, B)) est une coupure de Noeud(i, A, B) selon c.

3. Par symétrie, construire une coupure de l'ABR noeud(i, A, B) selon c, dans le cas où $c > i$.

4. Ecrire une fonction récursive Pascal **couper**(c : integer; a :Int_arbre): Couple_arbre telle que **couper**(c, a) renvoie la coupure de a selon c.

5. Si $|a|$ désigne la hauteur de l'arbre a, montrer que la complexité de la fonction **couper**(c, a), en nombre d'appels récursifs, est inférieure ou égale à $|a|$.

6. En utilisant la fonction **couper**, écrire une fonction non récursive Pascal
inserer_racine(i: integer; a : Int_arbre): Int_arbre telle que si b = **inserer_racine**(i, a) alors
Etiq(b) = Etiq(a) $\cup$ {i} et b est un ABR dont la racine est étiquetée par i. On supposera que les arguments i et a de la fonction sont tels que $i \notin$ Etiq(a).

7. Ecrire une fonction récursive Pascal **min**(a : Int_arbre): integer qui renvoie l'étiquette minimale d'un ABR a différent de nil.

8. En utilisant les fonctions **min** et **couper**, écrire une fonction non récursive Pascal
union(a, b:Int_arbre): Int_arbre telle que
si a et b sont deux ABR tels que les étiquettes de a sont toutes inférieures aux étiquettes de b,
c'est-à-dire, (min(b) > x, $\forall\, x \in$ Etiq(a))
alors si c = **union**(a, b) alors c est un ABR et Etiq(c) = Etiq(a) $\cup$ Etiq(b)

9. Donner un majorant de la complexité de **union** (a, b) dans le pire cas en fonction de $|a|$ et $|b|$, en comptant uniquement les appels récursifs de fonction.

PARTIE III. EXERCICE SUR LES AUTOMATES FINIS

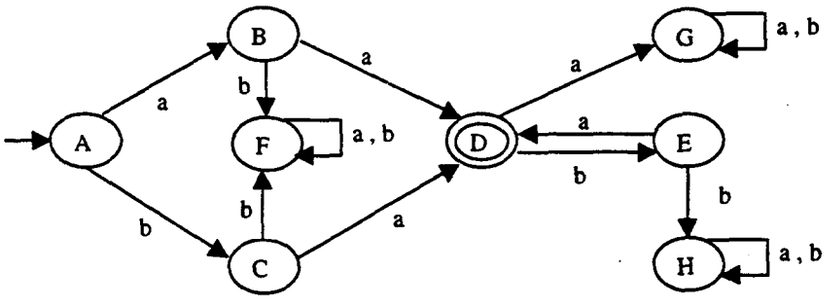
Soit X un alphabet, X^* désigne l'ensemble des mots sur X et ε désigne le mot vide.

Soit E une relation d'équivalence sur X^* ($[x]_E$ désigne la classe d'équivalence de x et X^*/E désigne l'ensemble des classes d'équivalence de X^* selon E) :

- E est d'*index fini* si et seulement si : E engendre un nombre fini de classes d'équivalence sur X^*,
- E est *invariante à droite* si et seulement si : si $x\ E\ y$, alors pour tout $z \in X^*$ on a $xz\ E\ yz$,
- E *sature* un langage $L \subseteq X^*$ si et seulement si : si $x \in L$ alors $[x]_E \subseteq L$.

Soit **N** = (Q, X, δ, A, F) l'automate fini déterministe dont :

- l'ensemble des états est Q = { A, B, C, D, E, F, G, H } ;
- l'état initial est A ;
- l'ensemble des états terminaux est F = { D } ;
- l'alphabet est X = { a, b } ;
- la fonction de transition δ est une application de $Q \times X$ dans Q définie par le graphe suivant :



Le but de l'exercice est de démontrer le théorème suivant dû à NERODE dans le cas du langage $L \subseteq X^*$ reconnu par l'automate **N**.

Théorème de NERODE : Les trois propriétés suivantes sont équivalentes :

a) L est un langage reconnu par un automate fini (déterministe ou indéterministe).

b) Il existe une relation d'équivalence E sur X^*, d'index fini, invariante à droite et qui sature L.

c) La relation d'équivalence $\equiv_L$ définie ci-après est d'index fini :

« $x \equiv_L y$ si et seulement si $\forall z \in X^*$, ($xz \in L$ si et seulement si $yz \in L$) ».

La démonstration suit le schéma a) ⇒ b) ⇒ c) ⇒ a).

Première étape : a) $\Rightarrow$ b)

Le langage L est reconnu par l'automate **N**. Soit Δ la relation d'équivalence sur X^* définie par :

« Pour x et $y \in X^*$, $x \Delta y$ si et seulement si $\delta^*(A, x) = \delta^*(A, y)$ ».

Sachant que δ^* désigne l'application de $Q \times X^*$ dans Q définie par :

a) $\forall q \in Q, \delta^*(q, \varepsilon) = q$,
b) $\forall q \in Q, \forall a \in X, \delta^*(q, a) = \delta(q, a)$,
c) $\forall q \in Q, \forall a_1, \ldots, a_n \in X, \delta^*(q, a_1 \ldots a_n) = \delta(\delta^*(q, a_1 \ldots a_{n-1}), a_n)$.

On admettra la propriété suivante : $\forall u, v \in X^*, \delta^*(q, uv) = \delta^*(\delta^*(q, u), v)$.

Une classe d'équivalence de la relation Δ est composée de l'ensemble des mots de X^* qui, à partir de l'état initial A, permettent d'atteindre un même état q de l'automate. La classe d'équivalence associée à q est le langage noté $\Delta_q = \{ x \mid x \in X^*, \delta^*(A, x) = q \}$ qui caractérise chaque état de l'automate.

1. Soit l'automate fini déterministe **N** :
a) Construire les classes d'équivalence Δ_q de la relation Δ. Chaque classe est un langage dont on donnera une caractérisation sous la forme d'une expression rationnelle ou d'une expression ensembliste. On donnera également un représentant de chaque classe.
b) Montrer que Δ est d'index fini.
c) Montrer que Δ est invariante à droite.
d) Montrer que Δ sature L.

Deuxième étape : b) $\Rightarrow$ c)

2.
a) Montrer que pour tout x et y de X^* tels que $x \Delta y$, on a $x \equiv_L y$.
b) En déduire la relation liant $[x]_\Delta$ et $[x]_{\equiv_L}$.
c) Montrer que $\equiv_L$ est d'index fini.

Troisième étape : c) $\Rightarrow$ a)

3. Soit $\equiv_L$ d'index fini :
a) Construire $X^*/\equiv_L$ l'ensemble de classes d'équivalence de $\equiv_L$ sur X^*.
b) Construire $L/\equiv_L$ l'ensemble de classes d'équivalence de $\equiv_L$ sur L.
c) Construire la relation δ_M telle que : pour tout $y \in X^*$ et $x \in X$, $\delta_M([y]_{\equiv_L}, x) = [yx]_{\equiv_L}$.
d) Montrer que δ_M est bien définie, c'est-à-dire que : si $y_1, y_2 \in [y]_{\equiv_L}$ et $x \in X$ alors $[y_1x]_{\equiv_L} = [y_2x]_{\equiv_L}$.
e) Montrer que le quintuplet $M = (X^*/\equiv_L, X, \delta_M, [\varepsilon]_{\equiv_L}, L/\equiv_L)$ est un automate fini déterministe.
f) Quel est le langage reconnu par cet automate ?
g) Que peut-on dire de **M** ?

Fin de l'énoncé