EPREUVE SPECIFIQUE - FILIERE MP

INFORMATIQUE

Durée : 3 heures

N.B. : Le candidat attachera la plus grande importance à la clarté, à la précision et à la concision de la rédaction. Si un candidat est amené à repérer ce qui peut lui sembler être une erreur d'énoncé, il le signalera sur sa copie et devra poursuivre sa composition en expliquant les raisons des initiatives qu'il a été amené à prendre.

Les calculatrices sont interdites

PRÉAMBULE : Les trois parties qui composent ce sujet sont indépendantes et peuvent être traitées par les candidats dans un ordre quelconque.

Pour les candidats ayant utilisé le langage CaML dans le cadre des enseignements d'informatique, la partie III (Algorithmique et programmation en CaML) se situe en page 6.

Pour les candidats ayant utilisé le langage PASCAL dans le cadre des enseignements d'informatique, la partie III (Algorithmique et programmation en PASCAL) se situe en page 13.

Tournez la page S.V.P.

Partie I : Logique et calcul des propositions

Vous participez à un concours de mathématiques comportant une partie de raisonnement logique. Plusieurs orateurs font des déclarations et vous devez répondre à des questions en vous appuyant sur des informations déduites de ces déclarations. La règle suivante s'applique : « Les orateurs sont de trois natures : les véridiques, les menteurs et les changeants. Les véridiques disent toujours la vérité, les menteurs mentent toujours, et les changeants disent en alternance une vérité et un mensonge (c'est-à-dire, soit une vérité, puis un mensonge, puis une vérité, etc. ; soit un mensonge, puis une vérité, puis un mensonge, etc.). Pendant tout le concours, les orateurs ne peuvent pas changer de nature. »
Les épreuves comportent deux phases :
- Les différents orateurs font plusieurs déclarations dont l'analyse permet de déterminer la nature de chaque orateur (véridique, menteur, changeant commençant par dire la vérité, ou changeant commençant par dire un mensonge).
- Les orateurs font une seconde série de déclarations. Puis, vous devez répondre à des questions en exploitant les informations contenues dans ces déclarations.

Question I.1 *Dans la première phase, quel est le nombre minimum de déclarations que doit faire chaque orateur pour qu'il soit possible de déterminer sa nature ? Justifier votre réponse.*

Question I.2 *Soit un orateur A qui fait une suite de n déclarations A_i. Proposer des formules du calcul des propositions A_V, A_M, A_{CV} et A_{CM} qui permettent de caractériser la nature de A (respectivement véridique, menteur, changeant commençant par dire la vérité, ou changeant commençant par dire un mensonge).*

Vous participez à une première épreuve avec un orateur A qui fait les déclarations suivantes :
- J'aime le rouge mais pas le bleu.
- Soit j'aime le rouge, soit j'aime le vert.
- Si j'aime le rouge et le vert, alors j'aime le bleu.

Nous noterons R, V et B les variables propositionnelles associées au fait que l'orateur aime le rouge, le vert ou le bleu.
Nous noterons A_1, A_2 et A_3 les formules propositionnelles associées aux déclarations de A.

Question I.3 *Représenter les déclarations de l'orateur sous la forme de formules du calcul des propositions A_1, A_2 et A_3 dépendant des variables R, V et B.*

Question I.4 *Appliquer les formules permettant de caractériser la nature des orateurs A_V, A_M, A_{CV} et A_{CM} que vous avez proposées pour la question I.2 pour l'orateur A dépendant des variables A_1, A_2 et A_3.*

Question I.5 *En utilisant le calcul des propositions (résolution avec les tables de vérité), déterminer la nature de l'orateur A. Quelles sont les couleurs qu'aime A ?*

Vous participez à une seconde épreuve avec trois orateurs G, H et I. Vous avez déterminé dans la première phase avec succès que G est un menteur, que H est un véridique et que I est un changeant sans savoir s'il doit dire la vérité ou un mensonge pour sa déclaration suivante. Ceux-ci font les déclarations :

I : Le losange est visible.

G : Le cercle n'est visible que si le losange est visible.

I : Le triangle n'est pas visible.

H : Soit le cercle est visible, soit le triangle est visible.

Nous noterons G_1, H_1, I_1 et I_2 les formules propositionnelles associées aux déclarations des orateurs C, D et E dans cette première épreuve.
Nous noterons C, L et T les variables propositionnelles associées au fait que le cercle, le losange ou le triangle soit visible.

Question I.6 *Représenter les déclarations des orateurs sous la forme de formules du calcul des propositions G_1, H_1, I_1 et I_2 dépendant des variables C, L et T.*

Question I.7 *Représenter les informations sur la nature des orateurs sous la forme d'une formule du calcul des propositions dépendant des variables G_1, H_1, I_1 et I_2.*

Question I.8 *En utilisant le calcul des propositions (résolution avec les formules de De Morgan), déterminer quelle est (ou quelles sont) la (ou les) figure(s) visible(s) ainsi que la nature exacte de I l'orateur changeant.*

Partie II : Automates et langages

Le but de cet exercice est l'étude des propriétés de l'opération $\oplus$ de composition de deux automates.

1 Automate fini complet déterministe

Pour simplifier les preuves, nous nous limiterons au cas des automates finis complets déterministes. Les résultats étudiés s'étendent au cadre des automates finis complets quelconques.

1.1 Représentation d'un automate fini complet déterministe

Déf. II.1 (Automate fini complet déterministe) Soit l'alphabet X (un ensemble de symboles), soit Λ le symbole représentant le mot vide ($\Lambda \notin X$), soit $X^\star$ l'ensemble contenant Λ et les mots composés de séquences de symboles de X (donc $\Lambda \in X^\star$) ; un automate fini complet déterministe sur X est un quintuplet $A = (Q, X, i, T, \delta)$ composé de :
– un ensemble fini d'états : Q ;
– un état initial : $i \in Q$;
– un ensemble d'états terminaux : $T \subseteq Q$;
– une fonction totale de transition confondue avec son graphe : $\delta \subseteq Q \times X \mapsto Q$.

Pour une transition $\delta(o, e) = d$ donnée, nous appelons o l'origine de la transition, e l'étiquette de la transition et d la destination de la transition.

1.2 Représentation graphique d'un automate

Les automates peuvent être représentés par un schéma suivant les conventions :
– les valeurs de la fonction totale de transition δ sont représentées par un graphe orienté dont les nœuds sont les états et les arêtes sont les transitions ;

Tournez la page S.V.P.

– un état initial est entouré d'un cercle (i) ;

– un état terminal est entouré d'un double cercle $((t))$;

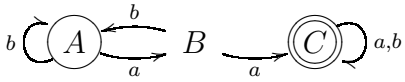
– un état qui est à la fois initial et terminal est entouré d'un triple cercle $(((it)))$;

– une arête étiquetée par le symbole $e \in X$ va de l'état o à l'état d si et seulement si $\delta(o, e) = d$.

Exemple II.1 L'automate $\mathcal{E}_1 = (Q_1, X, i_1, T_1, \delta_1)$ avec :

$$\begin{array}{l} Q_1 = \{A, B, C\} \\ X = \{a, b\} \\ i_1 = A \\ T_1 = \{C\} \\ \quad \delta_1(A, a) = B, \quad \delta_1(A, b) = A, \\ \quad \delta_1(B, a) = C, \quad \delta_1(B, b) = A, \\ \quad \delta_1(C, a) = C, \quad \delta_1(C, b) = C, \end{array}$$

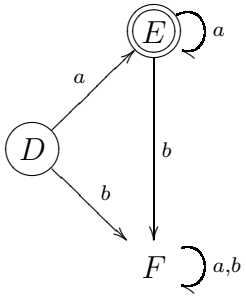
est représenté par le graphe suivant :



Exemple II.2 L'automate $\mathcal{E}_2 = (Q_2, X, i_2, T_2, \delta_2)$ avec :

$$\begin{array}{l} Q_2 = \{D, E, F\} \\ X = \{a, b\} \\ i_2 = D \\ T_2 = \{E\} \\ \quad \delta_2(D, a) = E, \quad \delta_2(D, b) = F, \\ \quad \delta_2(E, a) = E, \quad \delta_2(E, b) = F, \\ \quad \delta_2(F, a) = F, \quad \delta_2(F, b) = F, \end{array}$$

est représenté par le graphe suivant :



1.3 Langage reconnu par un automate fini complet déterministe

Soit $\delta^\star$ l'extension de δ à $Q \times X^\star \mapsto Q$ définie par :

$$\forall q \in Q,\ \delta^\star(q, \Lambda) = q$$

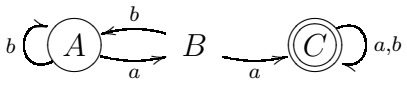
$$\left\{ \begin{array}{l} \forall e \in X, \\ \forall m \in X^\star, \\ \forall o \in Q, \\ \forall d \in Q, \end{array} \right. \quad \delta^\star(o, e.m) = d \Leftrightarrow \exists q \in Q,\ (\delta(o, e) = q) \wedge (\delta^\star(q, m) = d)$$

Soit X un alphabet, un langage sur X est un sous-ensemble de $X^\star$.
Le langage sur X reconnu par un automate fini complet déterministe est :

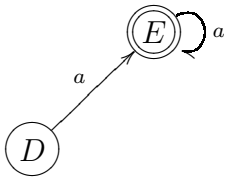
$$L(A) = \{m \in X^\star \mid \exists d \in T,\ \delta^\star(i, m) = d\}$$

Notons que certains états et transitions ne sont pas utiles dans la description d'un langage car ils ne permettent pas d'aller d'un état initial à un état terminal.

Exemple II.3 Le sous-automate de $\mathcal{E}_1$ (exemple II.1) composé des états et transitions utiles est représenté par le graphe suivant :



Exemple II.4 Le sous-automate de $\mathcal{E}_2$ (exemple II.2) composé des états et transitions utiles est représenté par le graphe suivant :



Question II.1 *Donner, sans les justifier, deux expressions régulières ou ensemblistes représentant les langages sur $X = \{a, b\}$ reconnus par les automates $\mathcal{E}_1$ de l'exemple II.1 et $\mathcal{E}_2$ de l'exemple II.2.*

2 Composition d'automates finis complets déterministes

2.1 Définition

Soit l'opération interne $\oplus$ sur les automates finis complets déterministes définie par :

Déf. II.2 (Composition d'automates finis complets déterministes) Soient $\mathcal{A}_1 = (Q_1, X, i_1, T_1, \delta_1)$ et $\mathcal{A}_2 = (Q_2, X, i_2, T_2, \delta_2)$ deux automates finis complets déterministes, l'automate $\mathcal{A} = \mathcal{A}_1 \oplus \mathcal{A}_2$ qui résulte de la composition de $\mathcal{A}_1$ et $\mathcal{A}_2$ est défini par :

$$\mathcal{A} = \mathcal{A}_1 \oplus \mathcal{A}_2 = (Q_1 \times Q_2, X, (i_1, i_2), T_1 \times (Q_2 \setminus T_2) \cup (Q_1 \setminus T_1) \times T_2, \delta_{1\oplus 2})$$

$$\begin{cases} \forall o_1 \in Q_1, \\ \forall o_2 \in Q_2, \\ \forall x \in X, \quad \delta_{1\oplus 2}((o_1, o_2), x) = (d_1, d_2) \Leftrightarrow (\delta_1(o_1, x) = d_1) \wedge (\delta_2(o_2, x) = d_2) \\ \forall d_1 \in Q_1, \\ \forall d_2 \in Q_2, \end{cases}$$

Question II.2 *En considérant les exemples II.1 et II.2, construire le graphe représentant l'automate $\mathcal{E}_1 \oplus \mathcal{E}_2$ (seuls les états et les transitions utiles devront être construits).*

Question II.3 *Caractériser le langage reconnu par $\mathcal{E}_1 \oplus \mathcal{E}_2$ par une expression régulière ou ensembliste.*

Tournez la page S.V.P.

2.2 Propriétés

Question II.4 *Montrer que : si $\mathcal{A}_1$ et $\mathcal{A}_2$ sont des automates finis complets déterministes alors $\mathcal{A}_1 \oplus \mathcal{A}_2$ est un automate fini complet déterministe.*

Question II.5 *Montrer que :*

$$\left\{\begin{array}{l} \forall o_1 \in Q_1, \\ \forall o_2 \in Q_2, \\ \forall m \in X^\star, \quad \delta^\star_{1\oplus 2}((o_1,o_2),m) = (d_1,d_2) \Leftrightarrow (\delta^\star_1(o_1,m) = d_1) \wedge (\delta^\star_2(o_2,m) = d_2) \\ \forall d_1 \in Q_1, \\ \forall d_2 \in Q_2, \end{array}\right.$$

Question II.6 *Soient $\mathcal{A}_1$ et $\mathcal{A}_2$ des automates finis complets déterministes, montrer que :*

$$m \in L(\mathcal{A}_1 \oplus \mathcal{A}_2) \Leftrightarrow (m \in L(\mathcal{A}_1) \wedge m \notin L(\mathcal{A}_2)) \vee (m \in L(\mathcal{A}_2) \wedge m \notin L(\mathcal{A}_1))$$

Question II.7 *Quelle relation liant les langages reconnus par les automates $\mathcal{A}_1$, $\mathcal{A}_2$ et $\mathcal{A}_1 \oplus \mathcal{A}_2$ peut-on en déduire ?*

Partie III : Algorithmique et programmation en CaML

Cette partie doit être traitée par les étudiants qui ont utilisé le langage CaML dans le cadre des enseignements d'informatique. Les fonctions écrites devront être récursives ou faire appel à des fonctions auxiliaires récursives. Elles ne devront pas utiliser d'instructions itératives (c'est-à-dire `for`, `while`, ...) ni de références.

1 Séquence croissante d'entiers

1.1 Définition

Déf. III.1 (séquence d'entiers) Une séquence s de taille n de valeurs entières v_i avec $1 \leq i \leq n$ est notée $\langle v_1, \ldots, v_n \rangle$. Sa taille n est notée $\mid \texttt{s} \mid$.

Déf. III.2 (séquence croissante d'entiers) Une séquence d'entiers $\langle v_1, \ldots, v_n \rangle$ est croissante si et seulement si : $\forall i \in [1, \ldots, n[, v_i \leq v_{i+1}$

Une séquence d'entiers est représentée par le type `sequence` dont la définition est :

```
type sequence == int list;;
```

Nous supposons prédéfinie la fonction `length : sequence -> int` telle que l'appel `(length s)` sur une séquence d'entiers `s` renvoie la valeur entière $\mid \texttt{s} \mid$. Son calcul se termine quelle que soit la valeur de son paramètre.

1.2 Ajout d'une valeur dans une séquence croissante d'entiers

Question III.1 *Écrire en CaML une fonction* `ajoutSequence` *de type :*

```
int -> sequence -> sequence
```

telle que l'appel `(ajoutSequence v s)` *sur une séquence d'entiers* `s` *croissante renvoie une séquence d'entiers croissante contenant les mêmes valeurs que la séquence* `s` *ainsi que l'entier* `v`*. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

Question III.2 *Calculer une estimation de la complexité de la fonction* `ajoutSequence` *en fonction du nombre d'éléments de la séquence d'entiers* `s`*. Cette estimation ne prendra en compte que le nombre d'appels récursifs effectués.*

1.3 Scission d'une séquence croissante d'entiers

Question III.3 *Écrire en CaML une fonction* `scissionSequence` *de type :*

```
sequence -> (sequence * int * sequence)
```

telle que l'appel `(scissionSequence s)` *sur une séquence croissante d'entiers* `s` *de taille impaire strictement positive* $2n+1$ *renvoie un triplet* `(s1,v,s2)` *contenant deux séquences croissantes d'entiers* `s1` *et* `s2` *où* `s1` *est le préfixe de* `s` *de taille* n*,* `s2` *est le suffixe de* `s` *de taille* n *et* `v` *est le* $n+1$*-ième entier de* `s` *(c'est-à-dire que* `s = s1@(v::s2)`*). Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

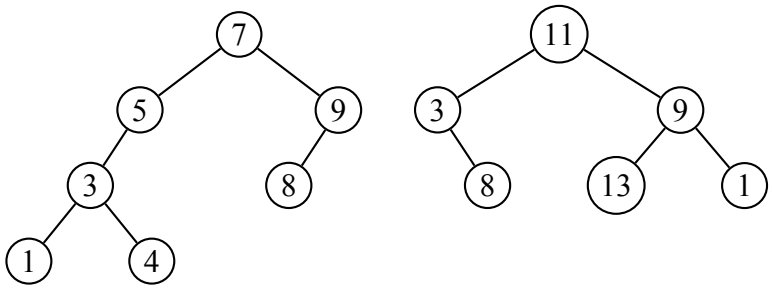
2 Arbre binaire de recherche d'entiers

L'objectif de cet exercice est d'étudier une implantation d'un arbre binaire de recherche et l'opération d'élimination d'une valeur dans un arbre binaire de recherche.

2.1 Arbre binaire d'entiers

Déf. III.3 (arbre binaire d'entiers) Un arbre binaire d'entiers a est une structure qui peut, soit être vide (notée $\emptyset$), soit être un nœud qui contient une étiquette entière (notée $\mathcal{E}(a)$), un sous-arbre gauche (noté $\mathcal{G}(a)$) et un sous-arbre droit (noté $\mathcal{D}(a)$) qui sont tous deux des arbres binaires d'entiers. L'ensemble des étiquettes de tous les nœuds de l'arbre a est noté $\mathcal{C}(a)$.

Exemple III.1 (arbre binaire d'entiers) Voici deux exemples d'arbres binaires étiquetés par des entiers (les sous-arbres vides qui sont les fils gauche ou droit des nœuds ne sont pas représentés) :



Tournez la page S.V.P.

2.2 Profondeur d'un arbre

Déf. III.4 (profondeur d'un arbre) Les branches d'un arbre relient la racine aux sous-arbres vides. La profondeur d'un arbre A est égale au nombre de liaisons entre les nœuds de la branche la plus longue. Nous la noterons $\mid A \mid$. Nous associerons la profondeur -1 à l'arbre vide $\emptyset$.

Exemple III.2 (profondeurs) Les profondeurs des deux arbres binaires de l'exemple III.1 sont respectivement 3 et 2.

2.3 Arbre binaire de recherche d'entiers

Déf. III.5 (arbre binaire de recherche) Un arbre binaire de recherche est un arbre binaire d'entiers dont :
- les fils de la racine sont des arbres binaires de recherche ;
- les étiquettes de tous les nœuds composant le fils gauche de la racine sont inférieures ou égales à l'étiquette de la racine ;
- les étiquettes de tous les nœuds composant le fils droit de la racine sont strictement supérieures à l'étiquette de la racine.

Ces contraintes s'expriment sous la forme :

$$ABR(a) \equiv a \neq \emptyset \Rightarrow \begin{cases} ABR(\mathcal{G}(a)) \wedge ABR(\mathcal{D}(a)) \\ \wedge \forall v \in \mathcal{C}(\mathcal{G}(a)), v \leq \mathcal{E}(a) \\ \wedge \forall v \in \mathcal{C}(\mathcal{D}(a)), \mathcal{E}(a) < v \end{cases}$$

Exemple III.3 (arbres binaires de recherche) Le premier arbre de l'exemple III.1 est un arbre binaire de recherche.

Un arbre binaire d'entiers est représenté par le type `arbre` dont la définition est :

```
type arbre =
  | Vide
  | Noeud of arbre * int * arbre;;
```

Dans l'appel `Noeud( fg, v, fd)`, les paramètres `fg`, `v` et `fd` sont respectivement le fils gauche, l'étiquette et le fils droit de la racine de l'arbre créé.

Exemple III.4 L'expression suivante :

```
Noeud(
   Noeud(
      Noeud(
         Noeud( Vide, 1, Vide),
         3,
         Noeud( Vide, 4, Vide)),
      5,
      Vide),
7,
Noeud(
   Noeud( Vide, 8, Vide),
   9,
   Vide))
```

est alors associée au deuxième arbre binaire représenté graphiquement dans l'exemple III.1.

Soit le programme en langage CaML :

```
let rec eliminer v a =
  let rec aux a =
    match a with
      | Noeud(g,v,Vide) -> (v,g)
      | Noeud(g,v,d) ->
        let (vr,ar) = (aux d) in
        (vr,(Noeud(g,v,ar))) in
  match a with
    | Vide -> Vide
    | Noeud(g,vp,d) ->
      if (vp = v)
      then
        (let rg = (eliminer v g) in
         (if (rg = Vide)
         then d
         else
            let (vm,gp) = aux rg
            in Noeud(gp,vm,d)))
      else
        if (v < vp)
        then
          Noeud((eliminer v g),vp,d)
        else
          Noeud(g,vp,(eliminer v d));;
```

Soit la constante `exemple` définie et initialisée par :

```
let exemple =
   Noeud(Noeud(Noeud(Vide,1,Vide),2,Vide),2,Noeud(Vide,3,Vide));;
```

Question III.4 *Détailler les étapes du calcul de `(eliminer 2 exemple)` en précisant pour chaque appel aux fonctions `eliminer` et `aux`, les valeurs des paramètres et résultats.*

Question III.5 *Soient les arbres binaires d'entiers `a` et `r`, soit l'entier `v`, tels que `r = (eliminer v a)`, montrer que :*

$$ABR(\mathtt{a}) \Rightarrow (ABR(\mathtt{r}) \wedge \mathcal{C}(r) = \mathcal{C}(a) \setminus \{v\})$$

Question III.6 *Montrer que le calcul des fonctions `aux` et `eliminer` se termine quelles que soient les valeurs de leurs paramètres respectant le type des fonctions.*

Question III.7 *Donner des exemples de valeurs du paramètre `a` de la fonction `eliminer` qui correspondent aux meilleur et pire cas en nombre d'appels récursifs des fonctions `aux` et `eliminer` effectués.*

Question III.8 *Calculer une estimation de la complexité dans le meilleur et le pire cas de la fonction `eliminer` en fonction du nombre de valeurs dans l'arbre donné en paramètre. Cette estimation ne prendra en compte que le nombre d'appels récursifs des fonctions `aux` et `eliminer` effectués.*

Tournez la page S.V.P.

3 Problème : Structure de données B-Arbre d'entiers

La création de branches dans un arbre binaire de recherche est effectuée au niveau des feuilles lors de l'ajout de nouvelles valeurs sans possibilité de réorganisation de la structure de l'arbre si les valeurs sont mal réparties. Ceci peut conduire à la construction d'arbres fortement déséquilibrés. De plus, un nœud est nécessaire pour chaque valeur contenue dans l'arbre, ce qui conduit à une structure peu compacte qui demande un grand nombre d'accès en lecture à des positions différentes. Ces accès sont très coûteux lorsque l'arbre est stocké sur un disque pour implanter une base de données par exemple. Pour réduire ces deux défauts, la structure de B-Arbre associe un nombre de valeurs plus important au niveau de chaque nœud, et exploite un algorithme plus complexe lors de l'ajout d'une valeur qui permet d'équilibrer partiellement la structure.
L'objectif de ce problème est l'étude d'une implantation particulière de la structure de B-Arbre en utilisant une liste de couples *(étiquette, sous-arbre)* pour les nœuds de l'arbre et une liste d'étiquettes pour les feuilles de l'arbre.

3.1 Définitions

Un B-Arbre de nombres entiers est une extension d'un arbre binaire de recherche de nombres entiers telle qu'un nœud peut contenir plusieurs étiquettes et plusieurs sous-arbres. Les étiquettes et les sous-arbres sont alternés dans les nœuds de manière à ce que chaque étiquette e ait un sous-arbre à sa gauche et un sous-arbre à sa droite. Les étiquettes contenues dans le sous-arbre gauche sont inférieures ou égales à e et les étiquettes contenues dans le sous-arbre droit sont supérieures à e.

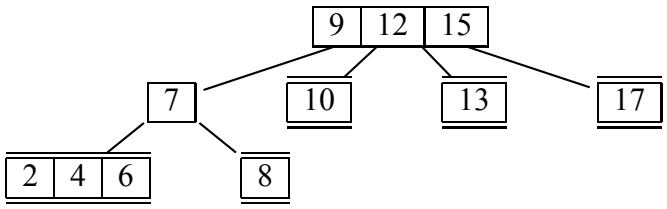
Déf. III.6 (B-Arbre d'ordre n de nombres entiers) Soit n un nombre entier strictement positif, un B-Arbre d'ordre n de nombres entiers a est une structure qui peut être :
- soit une feuille qui contient une séquence croissante notée $\mathcal{E}(a) = \langle e_1, \ldots, e_p \rangle$ de taille p d'étiquettes entières e_i telle que $p + 1 \leq 2n$. Le nombre p est appelé rang de la feuille ;
- soit un nœud qui contient une séquence croissante également notée $\mathcal{E}(a)$ de taille p d'étiquettes entières et une séquence $\mathcal{A}(a) = \langle a_1, \ldots, a_{p+1} \rangle$ de taille $p+1$ de sous-arbres a_i possédant la même structure de B-Arbre d'ordre n de nombres entiers telle que $p + 1 \leq 2n$. Le nombre p est appelé rang du nœud.

Pour toute feuille et pour tout nœud qui ne sont pas à la racine de l'arbre, le rang p vérifie également $n \leq p + 1$. L'ensemble des feuilles de l'arbre a est noté $\mathcal{F}(a)$. L'ensemble des nœuds de l'arbre a est noté $\mathcal{N}(a)$. L'ensemble des étiquettes entières d'un arbre a est noté $\mathcal{C}(a)$. Les étiquettes des nœuds d'un B-Arbre vérifient la contrainte suivante :

$$\forall n \in \mathcal{N}(a), \left\{ \begin{array}{l} \mathcal{E}(a) = \langle e_1, \ldots, e_p \rangle \\ \mathcal{A}(a) = \langle a_1, \ldots, a_{p+1} \rangle \end{array} \right. , \left\{ \begin{array}{l} \forall v \in \mathcal{C}(a_1), v \leq e_1 \\ \forall i \in [1, p[, \forall v \in \mathcal{C}(a_{i+1}), e_i < v \leq e_{i+1} \\ \forall v \in \mathcal{C}(a_{p+1}), e_p < v \end{array} \right.$$

Déf. III.7 (feuille et nœud complet) Un nœud ou une feuille de rang p d'un B-Arbre d'ordre n est complet si et seulement si $p + 1 = 2n$.

Exemple III.5 (B-Arbre d'ordre 2 d'entiers) Voici un exemple de B-Arbre d'ordre 2 d'entiers (les feuilles sont indiquées par une double barre horizontale) :



Un B-Arbre d'entiers est représenté par la constante entière `ordre` et les types `bArbre`, `paire` et `freres` dont la définition est :

```
let ordre = 2;;

type bArbre =
   | Feuille of sequence
   | Noeud of bArbre * freres
and paire == int * bArbre
and freres == paire list;;
```

Dans l'appel `Noeud(p, n)`, les paramètres `p` et `n` sont respectivement le premier fils le plus à gauche a_1, et la séquence croissante des paires d'étiquettes et de fils droit $\langle (e_1, a_2), \ldots, (e_p, a_{p+1}) \rangle$ de la racine de l'arbre composé du nœud ainsi créé. Dans l'appel `Feuille(s)`, le paramètre est la séquence croissante des étiquettes de la racine de l'arbre composé de la feuille ainsi créée. Un arbre vide correspond à une feuille dont la séquence d'étiquettes est vide.

Exemple III.6 L'expression suivante :

```
Noeud(
   Noeud(
      Feuille([ 2; 4; 6 ]),
      [( 7, Feuille([ 8 ]))]
   ),
   [( 9, Feuille([ 10 ]));
     ( 12, Feuille([ 13 ]));
     ( 15, Feuille([ 17 ]))])
```

est alors associée au B-Arbre d'ordre 2 représenté graphiquement dans l'exemple III.5.

3.2 Test de complétude dans un B-Arbre

Question III.9 *Écrire en CaML une fonction `estComplet` de type `bArbre -> boolean` telle que l'appel `(estComplet a)` renvoie la valeur `true` si le nœud, ou la feuille, situé à la racine de `a` est complet et la valeur `false` sinon.*

3.3 Calcul du nombre de valeurs dans un B-Arbre

Question III.10 *Écrire en CaML une fonction `taille` de type `bArbre -> int` telle que l'appel `(taille a)` renvoie le nombre d'étiquettes entières contenues dans le B-Arbre `a`, c'est-à-dire le cardinal de $\mathcal{C}(a)$. L'algorithme utilisé ne devra parcourir qu'une seule fois l'arbre `a`. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

Tournez la page S.V.P.

3.4 Recherche d'une étiquette dans un B-Arbre

Question III.11 *Écrire en CaML une fonction* `recherche` *de type :*

```
int -> bArbre -> boolean
```

telle que l'appel `(recherche v a)` *sur un B-Arbre* `a` *renvoie la valeur* `true` *si* `v` $\in \mathcal{C}(a)$ *et la valeur* `false` *sinon. L'algorithme utilisé ne devra parcourir qu'une seule fois l'arbre* `a`*. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

3.5 Scission d'un B-Arbre

La scission est l'opération qui permet d'augmenter la taille d'une branche dans un B-Arbre lors de l'ajout d'une étiquette.

Question III.12 *Écrire en CaML une fonction* `scissionBArbre` *de type :*

```
bArbre -> (bArbre * int * bArbre)
```

telle que l'appel `(scissionBArbre a)` *sur un B-Arbre d'ordre* n *dont la racine est complète renvoie un triplet* `(a1,v,a2)` *contenant deux B-Arbres d'ordre* n `a1` *et* `a2` *où* `a1` *est le préfixe de* `a` *de rang* $n-1$ *(contient les* $n-1$ *premiers fils et étiquettes de* `a`*),* `a2` *est le suffixe de* `a` *de rang* $n-1$ *(contient les* $n-1$ *derniers fils et étiquettes de* `a`*) et* v *est la* n*-ième étiquette de* `a`*. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

3.6 Ajout d'une valeur dans un B-Arbre d'entiers

L'ajout d'une valeur dans un B-Arbre consiste à parcourir l'arbre de la racine vers la feuille qui devra contenir la valeur en suivant le même algorithme que pour rechercher une valeur. Lors de ce parcours de la racine vers la feuille cible, les nœuds complets traversés, et la feuille destination si elle est complète, doivent être scindés. Lors de la scission d'un élément, celui-ci est remplacé par un nœud contenant une seule étiquette, la valeur obtenue par scission, et les deux sous-arbres produits par la scission. Après chaque scission, le parcours se poursuit dans le sous-arbre conduisant à la feuille cible. L'algorithme se conclut par l'insertion de la valeur dans la feuille destination.

Question III.13 *Détailler les étapes de l'ajout de la valeur* `3` *dans le B-Arbre de l'exemple III.5 en précisant toutes les étapes et les B-Arbres intermédiaires.*

Question III.14 *Montrer que cet algorithme se termine quelles que soient les valeurs de ses paramètres.*

Question III.15 *Montrer que cet algorithme d'ajout d'une valeur dans un B-Arbre préserve la structure de B-Arbre d'ordre* n*, c'est-à-dire que si le paramètre est un B-Arbre d'ordre* n *alors le résultat est un B-Arbre d'ordre* n*.*

Question III.16 *Écrire en CaML une fonction* `ajouter` *de type* `int -> bArbre -> bArbre` *telle que l'appel* `(ajouter v a)` *sur un B-Arbre d'entiers* `a` *renvoie un B-Arbre d'entiers contenant la valeur* `v` *et les mêmes valeurs que l'arbre* `a`*. L'algorithme utilisé ne devra parcourir qu'une seule fois l'arbre* `a`*. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

Partie III : Algorithmique et programmation en PASCAL

Cette partie doit être traitée par les étudiants qui ont utilisé le langage PASCAL dans le cadre des enseignements d'informatique. Les fonctions écrites devront être récursives ou faire appel à des fonctions auxiliaires récursives. Elles ne devront pas utiliser d'instructions itératives (c'est-à-dire `for`, `while`, `repeat`, ...).

1 Séquence croissante d'entiers

1.1 Définition

Déf. III.1 (séquence d'entiers) Une séquence s de taille n de valeurs entières v_i avec $1 \leq i \leq n$ est notée $\langle v_1, \ldots, v_n \rangle$. Sa taille n est notée $|\,\texttt{s}\,|$.

Déf. III.2 (séquence croissante d'entiers) Une séquence d'entiers $\langle v_1, \ldots, v_n \rangle$ est croissante si et seulement si : $\forall i \in [1, \ldots, n[, v_i \leq v_{i+1}$

Une séquence d'entiers est représentée par le type `SEQUENCE`. Nous supposons prédéfinies les constantes et les fonctions suivantes dont le calcul se termine quelles que soient les valeurs de leurs paramètres. Elles pourront éventuellement être utilisées dans les réponses aux questions :

- `Vide` est une constante de valeur `NIL` qui représente une séquence vide ;
- `FUNCTION S_Inserer(e:INTEGER;s:SEQUENCE):SEQUENCE;` renvoie une séquence d'entiers composée d'un premier entier `e` et du reste de la séquence contenu dans `s` ;
- `FUNCTION S_Tete(s:SEQUENCE):INTEGER;` renvoie le premier entier de la séquence `s`. Cette séquence ne doit pas être vide ;
- `FUNCTION S_Queue(s:SEQUENCE):SEQUENCE;` renvoie le reste de la séquence `s` privée de son premier entier. Cette séquence ne doit pas être vide ;
- `FUNCTION S_Juxtaposer(s1,s2:SEQUENCE):SEQUENCE;` renvoie une séquence composée des éléments de la séquence `s1` dans le même ordre que dans `s1` suivis des éléments de la séquence `s2` dans le même ordre que dans `s2` ;
- `FUNCTION S_Taille(s:SEQUENCE):INTEGER;` renvoie la valeur entière $|\,\texttt{s}\,|$.

1.2 Ajout d'une valeur dans une séquence croissante d'entiers

Question III.1 *Écrire en PASCAL une fonction :*

`ajoutSequence(v:INTEGER;s:SEQUENCE):SEQUENCE;`

telle que l'appel `ajoutSequence(v,s)` *sur une séquence d'entiers* `s` *croissante renvoie une séquence d'entiers croissante contenant les mêmes valeurs que la séquence* `s` *ainsi que l'entier* `v`*. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

Question III.2 *Calculer une estimation de la complexité de la fonction* `ajoutSequence` *en fonction du nombre d'éléments de la séquence d'entiers* `s`*. Cette estimation ne prendra en compte que le nombre d'appels récursifs effectués.*

Tournez la page S.V.P.

1.3 Scission d'une séquence croissante d'entiers

Un triplet contenant une valeur entière et deux séquences d'entiers est représenté par le type de base `S_TRIPLET`.
Nous supposons prédéfinies les constantes et les fonctions suivantes dont le calcul se termine quelles que soient les valeurs de leurs paramètres. Elles pourront éventuellement être utilisées dans les réponses aux questions :

- `FUNCTION TS_Creer(s1:SEQUENCE;v:INTEGERS;s2:SEQUENCE):S_TRIPLET;` renvoie un triplet dont les éléments sont les séquences d'entiers `s1` et `s2` et la valeur entière `v`,
- `FUNCTION TS_Préfixe(t:S_TRIPLET):SEQUENCE;` renvoie la première séquence de `t`,
- `FUNCTION TS_Valeur(t:S_TRIPLET):INTEGER;` renvoie la valeur entière de `t`,
- `FUNCTION TS_Suffixe(t:S_TRIPLET):SEQUENCE;` renvoie la seconde séquence de `t`.

Question III.3 *Écrire en PASCAL une fonction :*

`scissionSequence(s:SEQUENCE):S_TRIPLET;`

telle que l'appel `scissionSequence(s)` *sur une séquence croissante d'entiers* `s` *de taille impaire strictement positive* $2n+1$ *renvoie un triplet* `TS_Creer(s1,v,s2)` *contenant deux séquences croissantes d'entiers* `s1` *et* `s2` *où* `s1` *est le préfixe de* `s` *de taille* n*,* `s2` *est le suffixe de* `s` *de taille* n *et* `v` *est le* $n+1$*-ième entier de* `s` *(c'est-à-dire que* `s = S_Juxtaposer(s1,S_Inserer(v,s2))`*). Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

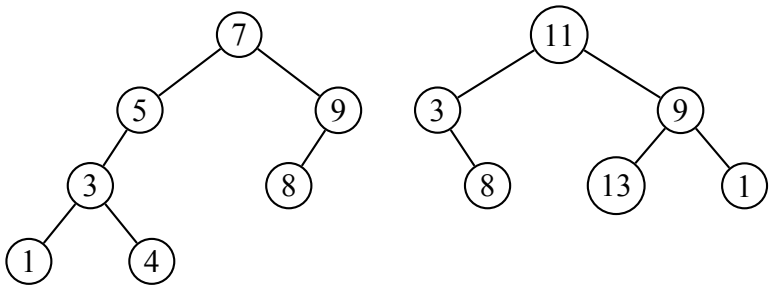
2 Arbre binaire de recherche d'entiers

L'objectif de cet exercice est d'étudier une implantation d'un arbre binaire de recherche et l'opération d'élimination d'une valeur dans un arbre binaire de recherche.

2.1 Arbre binaire d'entiers

Déf. III.3 (arbre binaire d'entiers) Un arbre binaire d'entiers a est une structure qui peut, soit être vide (notée $\emptyset$), soit être un nœud qui contient une étiquette entière (notée $\mathcal{E}(a)$), un sous-arbre gauche (noté $\mathcal{G}(a)$) et un sous-arbre droit (noté $\mathcal{D}(a)$) qui sont tous deux des arbres binaires d'entiers. L'ensemble des étiquettes de tous les nœuds de l'arbre a est noté $\mathcal{C}(a)$.

Exemple III.1 (arbre binaire d'entiers) Voici deux exemples d'arbres binaires étiquetés par des entiers (les sous-arbres vides qui sont les fils gauche ou droit des nœuds ne sont pas représentés) :



2.2 Profondeur d'un arbre

Déf. III.4 (profondeur d'un arbre) Les branches d'un arbre relient la racine aux sous-arbres vides. La profondeur d'un arbre A est égale au nombre de liaisons entre les nœuds de la branche la plus longue. Nous la noterons $\mid A \mid$. Nous associerons la profondeur -1 à l'arbre vide $\emptyset$.

Exemple III.2 (profondeurs) Les profondeurs des deux arbres binaires de l'exemple III.1 sont respectivement 3 et 2.

2.3 Arbre binaire de recherche d'entiers

Déf. III.5 (arbre binaire de recherche) Un arbre binaire de recherche est un arbre binaire d'entiers dont :
- les fils de la racine sont des arbres binaires de recherche ;
- les étiquettes de tous les nœuds composant le fils gauche de la racine sont inférieures ou égales à l'étiquette de la racine ;
- les étiquettes de tous les nœuds composant le fils droit de la racine sont strictement supérieures à l'étiquette de la racine.

Ces contraintes s'expriment sous la forme :

$$ABR(a) \equiv a \neq \emptyset \Rightarrow \begin{cases} ABR(\mathcal{G}(a)) \wedge ABR(\mathcal{D}(a)) \\ \wedge \forall v \in \mathcal{C}(\mathcal{G}(a)), v \leq \mathcal{E}(a) \\ \wedge \forall v \in \mathcal{C}(\mathcal{D}(a)), \mathcal{E}(a) < v \end{cases}$$

Exemple III.3 (arbres binaires de recherche) Le premier arbre de l'exemple III.1 est un arbre binaire de recherche.

Un arbre binaire d'entiers est représenté par le type de base `ARBRE`.
Nous supposons prédéfinies les constantes et les fonctions suivantes dont le calcul se termine quelles que soient les valeurs de leurs paramètres. Elles pourront éventuellement être utilisées dans les réponses aux questions :
- `Vide` est une constante de valeur `NIL` qui représente un arbre vide ;
- `FUNCTION Noeud(fg:ARBRE;v:INTEGER;fd:ARBRE):ARBRE;` est une fonction qui renvoie un arbre dont le fils gauche, l'étiquette et le fils droit de la racine sont respectivement `fg`, `v` et `fd` ;
- `FUNCTION Gauche(a:ARBRE):ARBRE;` est une fonction qui renvoie le fils gauche de la racine de l'arbre `a`. Cet arbre ne doit pas être vide ;
- `FUNCTION Etiquette(a:ARBRE):INTEGER;` est une fonction qui renvoie l'étiquette de la racine de l'arbre `a`. Cet arbre ne doit pas être vide ;
- `FUNCTION Droit(a:ARBRE):ARBRE;` est une fonction qui renvoie le fils droit de la racine de l'arbre `a`. Cet arbre ne doit pas être vide.

Tournez la page S.V.P.

Exemple III.4 L'expression suivante :

```
Noeud(
   Noeud(
      Noeud(
         Noeud( Vide, 1, Vide),
         3,
         Noeud( Vide, 4, Vide)),
      5,
      Vide),
7,
Noeud(
   Noeud( Vide, 8, Vide),
   9,
   Vide))
```

est alors associée au deuxième arbre binaire représenté graphiquement dans l'exemple III.1.

Un couple contenant un entier et un arbre binaire d'entiers est représenté par le type `COUPLE`. Nous supposons prédéfinies les constantes et les fonctions suivantes dont le calcul se termine quelles que soient les valeurs de leurs paramètres. Elles pourront éventuellement être utilisées dans les réponses aux questions :

- `FUNCTION C_Creer(e:INTEGER;a:ARBRE):COUPLE;` renvoie un couple contenant l'entier `e` et l'arbre `a` ;
- `FUNCTION C_Premier(c:COUPLE):INTEGER;` renvoie l'entier contenu dans le couple `c` ;
- `FUNCTION C_Second(c:COUPLE):ARBRE;` renvoie l'arbre contenu dans le couple `c`.

Soit le programme en langage PASCAL :

```
FUNCTION eliminer(v : INTEGER;a:ARBRE):ARBRE;

   FUNCTION aux(a :ARBRE):COUPLE;
   VAR
      rg, g,d : ARBRE;
      e       : INTEGER;
   BEGIN
      IF (a <> Vide) THEN BEGIN
         g := Gauche(a);
         d := Droite(a);
         e := Valeur(a);
         IF (d = Vide) THEN
            aux := Couple(v,g)
         ELSE BEGIN
            r := aux(d);
            vr := Premier(r);
            gr := Second(r);
            aux := Couple(vr,Noeud(gr,v,d))
         END
      END
   END; { aux }
```

```
BEGIN
   IF (a = Vide) THEN
      eliminer := Vide
   ELSE BEGIN
      g := Gauche(a);
      d := Droite(a);
      vp := Valeur(a);
      IF (vp = v) THEN
         rg := eliminer(v,g);
         IF (rg = Vide) THEN
            extraire := d
         ELSE BEGIN
            r := aux(rg);
            vr := Premier(r);
            gr := Second(r);
            eliminer := Noeud(gr,vr,d)
         END
         ELSE
            IF (v < vp) THEN
               eliminer := Noeud(eliminer(v,g),vp,d)
            ELSE
               eliminer := Noeud(g,vp,eliminer(v,d))
   END
END; { eliminer }
```

Soit la constante `exemple` définie et initialisée par :

```
CONST exemple : ARBRE
   = Noeud(
         Noeud( Noeud( Vide, 1, Vide), 2, Vide),
         1,
         Noeud(Vide,3,Vide));
```

Question III.4 *Détailler les étapes du calcul de `eliminer(2,exemple)` en précisant pour chaque appel aux fonctions `eliminer` et `aux`, la valeur du paramètre et du résultat.*

Question III.5 *Soient les arbres binaires d'entiers `a` et `r`, soit l'entier `v`, tels que `r = eliminer(v,a)`, montrer que :*

$$ABR(\mathtt{a}) \Rightarrow (ABR(\mathtt{r}) \wedge \mathcal{C}(r) = \mathcal{C}(a) \setminus \{v\})$$

Question III.6 *Montrer que le calcul des fonctions `aux` et `eliminer` se termine quelles que soient les valeurs de leurs paramètres respectant le type des fonctions.*

Question III.7 *Donner des exemples de valeurs du paramètre `a` de la fonction `eliminer` qui correspondent aux meilleur et pire cas en nombre d'appels récursifs des fonctions `aux` et `eliminer` effectués.*

Question III.8 *Calculer une estimation de la complexité dans le meilleur et le pire cas de la fonction `eliminer` en fonction du nombre de valeurs dans l'arbre donné en paramètre. Cette estimation ne prendra en compte que le nombre d'appels récursifs des fonctions `aux` et `eliminer` effectués.*

Tournez la page S.V.P.

3 Problème : Structure de données B-Arbre d'entiers

La création de branches dans un arbre binaire de recherche est effectuée au niveau des feuilles lors de l'ajout de nouvelles valeurs sans possibilité de réorganisation de la structure de l'arbre si les valeurs sont mal réparties. Ceci peut conduire à la construction d'arbres fortement déséquilibrés. De plus, un nœud est nécessaire pour chaque valeur contenue dans l'arbre, ce qui conduit à une structure peu compacte qui demande un grand nombre d'accès en lecture à des positions différentes. Ces accès sont très coûteux lorsque l'arbre est stocké sur un disque pour implanter une base de données par exemple. Pour réduire ces deux défauts, la structure de B-Arbre associe un nombre de valeurs plus important au niveau de chaque nœud, et exploite un algorithme plus complexe lors de l'ajout d'une valeur qui permet d'équilibrer partiellement la structure.
L'objectif de ce problème est l'étude d'une implantation particulière de la structure de B-Arbre en utilisant une liste de couples *(étiquette, sous-arbre)* pour les nœuds de l'arbre et une liste d'étiquettes pour les feuilles de l'arbre.

3.1 Définitions

Un B-Arbre de nombres entiers est une extension d'un arbre binaire de recherche de nombres entiers telle qu'un nœud peut contenir plusieurs étiquettes et plusieurs sous-arbres. Les étiquettes et les sous-arbres sont alternés dans les nœuds de manière à ce que chaque étiquette e ait un sous-arbre à sa gauche et un sous-arbre à sa droite. Les étiquettes contenues dans le sous-arbre gauche sont inférieures ou égales à e et les étiquettes contenues dans le sous-arbre droit sont supérieures à e.

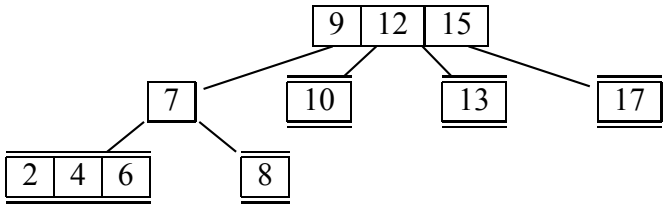
Déf. III.6 (B-Arbre d'ordre n de nombres entiers) Soit n un nombre entier strictement positif, un B-Arbre d'ordre n de nombres entiers a est une structure qui peut être :
- soit une feuille qui contient une séquence croissante notée $\mathcal{E}(a) = \langle e_1, \ldots, e_p \rangle$ de taille p d'étiquettes entières e_i telle que $p + 1 \leq 2n$. Le nombre p est appelé rang de la feuille ;
- soit un nœud qui contient une séquence croissante également notée $\mathcal{E}(a)$ de taille p d'étiquettes entières et une séquence $\mathcal{A}(a) = \langle a_1, \ldots, a_{p+1} \rangle$ de taille $p+1$ de sous-arbres a_i possédant la même structure de B-Arbre d'ordre n de nombres entiers telle que $p + 1 \leq 2n$. Le nombre p est appelé rang du nœud.

Pour toute feuille et pour tout nœud qui ne sont pas à la racine de l'arbre, le rang p vérifie également $n \leq p + 1$. L'ensemble des feuilles de l'arbre a est noté $\mathcal{F}(a)$. L'ensemble des nœuds de l'arbre a est noté $\mathcal{N}(a)$. L'ensemble des étiquettes entières d'un arbre a est noté $\mathcal{C}(a)$. Les étiquettes des nœuds d'un B-Arbre vérifient la contrainte suivante :

$$\forall n \in \mathcal{N}(a), \left\{ \begin{array}{l} \mathcal{E}(a) = \langle e_1, \ldots, e_p \rangle \\ \mathcal{A}(a) = \langle a_1, \ldots, a_{p+1} \rangle \end{array} \right. , \left\{ \begin{array}{l} \forall v \in \mathcal{C}(a_1), v \leq e_1 \\ \forall i \in [1, p[, \forall v \in \mathcal{C}(a_{i+1}), e_i < v \leq e_{i+1} \\ \forall v \in \mathcal{C}(a_{p+1}), e_p < v \end{array} \right.$$

Déf. III.7 (feuille et nœud complet) Un nœud ou une feuille de rang p d'un B-Arbre d'ordre n est complet si et seulement si $p + 1 = 2n$.

Exemple III.5 (B-Arbre d'ordre 2 d'entiers) Voici un exemple de B-Arbre d'ordre 2 d'entiers (les feuilles sont indiquées par une double barre horizontale) :



Un B-Arbre d'entiers est représenté par la constante entière `ordre` et le type `BARBRE`. Une paire d'étiquette entière et de B-Arbre est représentée par le type `PAIRE`. Une séquence de paires d'étiquettes entières et de B-Arbres d'entiers est représentée par le type `FRERES`.

```
CONST ordre : INTEGER = 2;
```

Nous supposons prédéfinies les constantes et les fonctions suivantes dont le calcul se termine quelles que soient les valeurs de leurs paramètres. Elles pourront éventuellement être utilisées dans les réponses aux questions :

- `Vide` est une constante de valeur `NIL` qui représente une séquence d'étiquettes entières et de B-Arbres vide ;
- `FUNCTION Noeud(p:BARBRE;n:FRERES):BARBRE;` est une fonction qui renvoie un nœud d'arbre dont le fils gauche, et la séquence croissante des paires d'étiquettes entières et de fils droits de la racine sont respectivement `p` et `n` ;
- `FUNCTION Feuille(f:SEQUENCE):BARBRE;` est une fonction qui renvoie une feuille d'arbre dont les étiquettes sont contenues dans la séquence croissante d'entiers `f` ;
- `FUNCTION Premier(a:BARBRE):BARBRE;` est une fonction qui renvoie le fils gauche de la racine de l'arbre `a`. La racine de cet arbre ne doit pas être une feuille ;
- `FUNCTION Freres(a:BARBRE):FRERES;` est une fonction qui renvoie l'étiquette de la racine de l'arbre `a`. La racine de cet arbre ne doit pas être une feuille ;
- `FUNCTION Etiquettes(a:ARBRE):SEQUENCE;` est une fonction qui renvoie la séquence d'étiquettes entières de la racine de l'arbre `a`. La racine de cet arbre ne doit pas être un nœud ;
- `FUNCTION F_Inserer(p:PAIRE;s:FRERES):FRERES;` renvoie une séquence d'entiers composée d'un premier entier `e` et du reste de la séquence contenu dans `s` ;
- `FUNCTION F_Tete(s:FRERES):PAIRE;` renvoie la première paire de la séquence `s`. Cette séquence ne doit pas être vide ;
- `FUNCTION F_Queue(s:FRERES):FRERES;` renvoie le reste de la séquence `s` privée de sa première paire. Cette séquence ne doit pas être vide ;
- `FUNCTION F_Juxtaposer(s1,s2:FRERES):FRERES;` renvoie une séquence composée des éléments de la séquence `s1` dans le même ordre que dans `s1` suivis des éléments de la séquence `s2` dans le même ordre que dans `s2` ;
- `FUNCTION P_Creer(e:INTEGER,a:BARBRE):PAIRE;` renvoie une paire contenant l'étiquette entière `e` et le B-Arbre `a` ;
- `FUNCTION P_Etiquette(p:PAIRE):INTEGER;` renvoie l'étiquette contenue dans la paire `p` ;
- `FUNCTION P_Arbre(p:PAIRE):BARBRE;` renvoie le B-Arbre d'entiers contenu dans la paire `p`.

Tournez la page S.V.P.

Exemple III.6 L'expression suivante :

```
Noeud(
   Noeud(
      Feuille(
         S_Inserer( 2, S_Inserer( 4, S_Inserer( 6, Vide )))),
      F_Inserer(
         P_Creer( 7, Feuille( S_Inserer( 8, Vide ))),
         Vide)
   ),
   F_Inserer( P_Creer( 9, Feuille( S_Inserer( 10, Vide))),
     F_Inserer( P_Creer( 12, Feuille( S_Inserer( 13, Vide))),
     F_Inserer(
        P_Creer( 15, Feuille( S_Inserer( 17, Vide))), Vide))))
```

est alors associée au B-Arbre d'ordre 2 représenté graphiquement dans l'exemple III.5.

3.2 Test de complétude dans un B-Arbre

Question III.9 *Écrire en PASCAL une fonction* `estComplet(a:BARBRE):BOOLEAN;` *telle que l'appel* `estComplet(a)` *renvoie la valeur* `true` *si le nœud, ou la feuille, situé à la racine de* `a` *est complet et la valeur* `false` *sinon.*

3.3 Calcul du nombre de valeurs dans un B-Arbre

Question III.10 *Écrire en PASCAL une fonction* `taille(a:BARBRE):INTEGER;` *telle que l'appel* `taille(a)` *renvoie le nombre d'étiquettes entières contenues dans le B-Arbre* `a`*, c'est-à-dire le cardinal de* $\mathcal{C}(a)$*. L'algorithme utilisé ne devra parcourir qu'une seule fois l'arbre* `a`*. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

3.4 Recherche d'une étiquette dans un B-Arbre

Question III.11 *Écrire en PASCAL une fonction :*

`recherche(v:INTEGER;a:bArbre):BOOLEAN`

telle que l'appel `recherche(v, a)` *sur un B-Arbre* `a` *renvoie la valeur* `TRUE` *si* `v` $\in \mathcal{C}(a)$ *et la valeur* `FALSE` *sinon. L'algorithme utilisé ne devra parcourir qu'une seule fois l'arbre* `a`*. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

3.5 Scission d'un B-Arbre

La scission est l'opération qui permet d'augmenter la taille d'une branche dans un B-Arbre lors de l'ajout d'une étiquette.
Un triplet contenant une valeur entière et deux B-Arbre d'entiers est représenté par le type de base `A_TRIPLET`.

Nous supposons prédéfinies les constantes et les fonctions suivantes dont le calcul se termine quelles que soient les valeurs de leurs paramètres. Elles pourront éventuellement être utilisées dans les réponses aux questions :

- `FUNCTION TA_Creer(a1:BARBRE;v:INTEGERS;a2:BARBRE):A_TRIPLET;` renvoie un triplet dont les éléments sont les B-Arbres d'entiers `a1` et `a2` et la valeur entière `v`,
- `FUNCTION TA_Préfixe(t:A_TRIPLET):BARBRE;` renvoie le premier B-Arbre de `t`,
- `FUNCTION TA_Valeur(t:A_TRIPLET):INTEGER;` renvoie la valeur entière de `t`,
- `FUNCTION TA_Suffixe(t:A_TRIPLET):BARBRE;` renvoie le second B-Arbre de `t`.

Question III.12 *Écrire en PASCAL une fonction `scissionBArbre(a:BARBRE):A_TRIPLET;` telle que l'appel `scissionBARBRE(a)` sur un B-Arbre d'ordre n dont la racine est complète renvoie un triplet `TA_Creer(a1,v,a2)` contenant deux B-Arbres d'ordre n `a1` et `a2` telles que `a1` est le préfixe de `a` de taille $n-1$ (contient les $n-1$ premiers fils et étiquettes de `a`), `a2` est le suffixe de `a` de taille $n-1$ (contient les $n-1$ derniers fils et étiquettes de `a`) et v est la n-ième étiquette de `a`. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

3.6 Ajout d'une valeur dans un B-Arbre d'entiers

L'ajout d'une valeur dans un B-Arbre consiste à parcourir l'arbre de la racine vers la feuille qui devra contenir la valeur en suivant le même algorithme que pour rechercher une valeur. Lors de ce parcours de la racine vers la feuille cible, les nœuds complets traversés, et la feuille destination si elle est complète, doivent être scindés. Lors de la scission d'un élément, celui-ci est remplacé par un nœud contenant une seule étiquette, la valeur obtenue par scission, et les deux sous-arbres produits par la scission. Après chaque scission, le parcours se poursuit dans le sous-arbre conduisant à la feuille cible. L'algorithme se conclut par l'insertion de la valeur dans la feuille destination.

Question III.13 *Détailler les étapes de l'ajout de la valeur `3` dans le B-Arbre de l'exemple III.5 en précisant toutes les étapes et les B-Arbres intermédiaires.*

Question III.14 *Montrer que cet algorithme se termine quelles que soient les valeurs de ses paramètres.*

Question III.15 *Montrer que cet algorithme d'ajout d'une valeur dans un B-Arbre préserve la structure de B-Arbre d'ordre n, c'est-à-dire que si le paramètre est un B-Arbre d'ordre n alors le résultat est un B-Arbre d'ordre n.*

Question III.16 *Écrire en PASCAL une fonction `ajouter(v:INTEGER;a:BARBRE):BARBRE;` telle que l'appel `ajouter(v,a)` sur un B-Arbre d'ordre n d'entiers `a` renvoie un B-Arbre d'ordre n d'entiers contenant la valeur `v` et les mêmes valeurs que l'arbre `a`. L'algorithme utilisé ne devra parcourir qu'une seule fois l'arbre `a`. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

Fin de l'énoncé

IMPRIMERIE NATIONALE – 12 1238 – D'après documents fournis