



CONCOURS COMMUNS POLYTECHNIQUES

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 5.

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.

Partie I : Logique et calcul des propositions

Dans un futur lointain, l'espèce humaine a découvert une autre espèce consciente. L'étude de cette espèce a permis de découvrir qu'elle est capable de percevoir si quelqu'un dit la vérité ou un mensonge. Les membres de cette espèce respectent les règles de politesse suivantes lors des discussions au sein d'un groupe : « Les orateurs doivent rester constants au cours d'une discussion : soit ils disent toujours la vérité, soit ils mentent toujours. De plus, si un orateur dit la vérité alors l'orateur suivant doit également dire la vérité. Si le sujet de la discussion change, les orateurs sont libres de changer leurs comportements. ».

Vous assistez à une discussion sur les moyens d'attaque et de défense que peut posséder la faune de cette planète entre trois membres de cette espèce que nous appellerons A , B et C .

A : « Le kjalt peut avoir un dard ou des griffes. »

B : « Non, il n'a pas de dard. »

C : « Il a des pinces et des griffes. »

Nous noterons D , G et P les variables propositionnelles associées au fait qu'un kjalt possède respectivement un dard, des griffes et des pinces.

Nous noterons A_1 , B_1 et C_1 les formules propositionnelles associées aux déclarations de A , B et C dans cette première discussion.

C quitte le groupe et la discussion change de sujet pour parler de la flore de la planète.

A : « Un lyop peut être de couleur mauve mais pas de couleur jaune. »

B : « Il ne peut pas être de couleur verte. »

A : « Il ne peut être de couleur verte que s'il peut être de couleur jaune. »

Nous noterons J , M et V les variables propositionnelles associées au fait qu'un lyop peut être respectivement de couleur jaune, mauve et verte.

Nous noterons A_2 , A_3 et B_2 les formules propositionnelles associées aux déclarations de A et B dans cette seconde discussion.

Question I.1 Représenter les règles de politesse appliquées à la première discussion sous la forme d'une formule du calcul des propositions dépendant des formules A_1 , B_1 et C_1 .

Question I.2 Représenter les informations données par les participants de la première discussion sous la forme de formules du calcul des propositions A_1 , B_1 et C_1 dépendant des variables D , G et P .

Question I.3 En utilisant le calcul des propositions (résolution avec les formules de De Morgan), déterminer le (ou les) moyen(s) d'attaque et de défense que peut posséder un kjalt.

Question I.4 Représenter les règles de politesse appliquées à la seconde discussion sous la forme d'une formule du calcul des propositions dépendant des formules A_2 , A_3 et B_2 .

Question I.5 Représenter les informations données par les participants lors de la seconde discussion sous la forme de trois formules du calcul des propositions A_2 , A_3 et B_2 dépendant des variables J , M et V .

Question I.6 En utilisant le calcul des propositions (résolution avec les tables de vérité), déterminer la (ou les) couleur(s) possible(s) pour un lyop.

Partie II : Automates et langages

Le but de cet exercice est l'étude des propriétés de l'opération d'inversion d'un automate.

1 Automate fini semi-indéterministe

Pour simplifier les preuves, nous nous limiterons au cas des automates finis semi-indéterministes (automates indéterministes sans transitions arbitraires sur ϵ). Les résultats étudiés s'étendent au cadre des automates finis quelconques.

1.1 Représentation d'un automate fini semi-indéterministe

Déf. II.1 (Automate fini semi-indéterministe) Soit l'alphabet X (un ensemble de symboles), soit Λ le symbole représentant le mot vide ($\Lambda \notin X$); un automate fini semi-indéterministe sur X est un quintuplet $A = (Q, X, I, T, \delta)$ composé de :

- un ensemble fini d'états : Q ;
- un ensemble d'états initiaux : $I \subseteq Q$;
- un ensemble d'états terminaux : $T \subseteq Q$;
- une fonction de transition confondue avec son graphe : $\delta \subseteq Q \times X \mapsto \mathcal{P}(Q)$.

Pour une transition $\delta(o, x) = \{d_1, \dots, d_n\}$ donnée, nous appelons o l'origine de la transition, x l'étiquette de la transition et d_1, \dots, d_n les différentes destinations de la transition.

1.2 Représentation graphique d'un automate

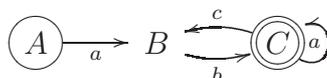
Les automates peuvent être représentés par un schéma suivant les conventions :

- les valeurs de la fonction de transition δ sont représentées par un graphe orienté dont les nœuds sont les états et les arêtes sont les transitions ;
- 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 $x \in X$ va de l'état o à l'état d si et seulement si $d \in \delta(o, x)$.

Exemple II.1 L'automate $\mathcal{E} = (Q, X, I, T, \delta)$ avec :

$$\begin{aligned} Q &= \{A, B, C\} \\ X &= \{a, b, c\} \\ I &= \{A\} \\ T &= \{C\} \\ \delta(A, a) &= \{B\} & \delta(B, b) &= \{C\} \\ \delta(C, a) &= \{C\} & \delta(C, c) &= \{B\} \end{aligned}$$

est représenté par le graphe suivant :



1.3 Langage reconnu par un automate fini semi-indéterministe

Soit X^* l'ensemble contenant Λ et les mots composés de séquences de symboles de X (donc $\Lambda \in X^*$), soit δ^* l'extension de δ à $Q \times X^* \mapsto \mathcal{P}(Q)$ définie par :

$$\forall q \in Q, \delta^*(q, \Lambda) = \{q\}$$

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

Soit X un alphabet, un langage sur X est un sous-ensemble de X^* .

Le langage sur X reconnu par un automate fini semi-indéterministe est :

$$L(A) = \{m \in X^* \mid \exists i \in I, \exists t \in T, t \in \delta^*(i, m)\}$$

Question II.1 Donner sans justification l'expression régulière ou ensembliste représentant le langage sur $X = \{a, b, c\}$ reconnu par l'automate \mathcal{E} de l'exemple II.1.

2 Opération d'inversion d'un automate : \mathcal{A}^{-1}

Soit l'opération interne d'inversion d'un automate fini semi-indéterministe définie par :

Déf. II.2 Soit $\mathcal{A} = (Q, X, I, T, \delta)$ un automate fini semi-indéterministe, l'automate $\mathcal{B} = \mathcal{A}^{-1}$, inverse de l'automate \mathcal{A} , est défini par :

$$\begin{aligned} \mathcal{B} &= (Q, X, I^{-1}, T^{-1}, \delta^{-1}) \\ I^{-1} &= T \\ T^{-1} &= I \\ \forall x \in X, \forall o \in Q, \forall d \in Q, d \in \delta^{-1}(o, x) &\Leftrightarrow o \in \delta(d, x) \end{aligned}$$

Question II.2 Construire l'automate \mathcal{E}^{-1} pour l'automate \mathcal{E} de l'exemple II.1.

Question II.3 Caractériser le langage reconnu par \mathcal{E}^{-1} par une expression régulière ou ensembliste. Comparer ce langage avec le langage reconnu par \mathcal{E} .

3 Étude de l'automate inverse

Question II.4 Montrer que :

$$\forall m \in X^*, \forall x \in X, \forall o \in Q, \forall d \in Q, d \in \delta^*(o, m.x) \Leftrightarrow \exists q \in Q, q \in \delta^*(o, m) \wedge d \in \delta(q, x)$$

Déf. II.3 L'opération d'inversion de mots sur X^* est définie par :

$$\left\{ \begin{array}{l} \Lambda^{-1} = \Lambda \\ \forall x \in X, \forall m \in X^*, (x.m)^{-1} = m^{-1}.x \end{array} \right.$$

Question II.5 Montrer que : $\forall m \in X^*, \forall o \in Q, \forall d \in Q, d \in \delta^*(o, m) \Leftrightarrow o \in \delta^{-1*}(d, m^{-1})$.

Question II.6 Soit un automate fini semi-indéterministe \mathcal{A} , quelle relation liant les langages reconnus par \mathcal{A} et \mathcal{A}^{-1} peut-on déduire des questions précédentes ?

Partie III : Algorithmique et programmation en CaML

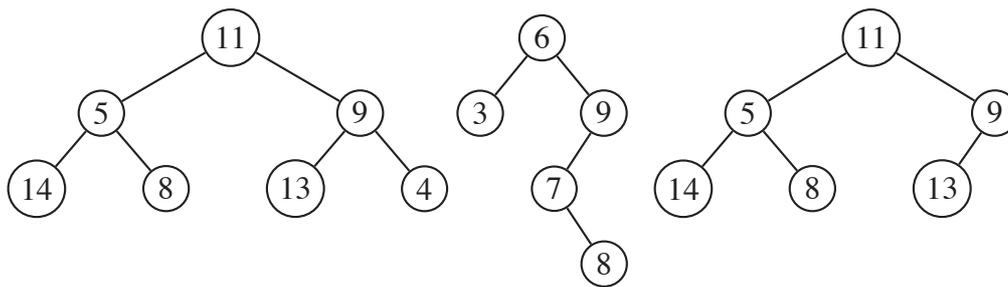
Cette partie doit être traitée par les étudiants qui ont utilisé le langage CaML dans le cadre des enseignements d'informatique. Les fonctions écrites devront être récursives ou faire appel à des fonctions auxiliaires récursives. Elles ne devront pas utiliser d'instructions itératives (c'est-à-dire `for`,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

1 Préliminaire : Arbre binaire d'entiers

1.1 Définition

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



1.2 Profondeur d'un arbre

Déf. III.2 (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 nœuds de la branche la plus longue. Nous la noterons $|A|$. Nous associerons la profondeur -1 à l'arbre vide \emptyset .

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

Question III.1 Donner une définition de la profondeur d'un arbre a en fonction de \emptyset , $\mathcal{G}(a)$ et $\mathcal{D}(a)$.

Question III.2 Soit un ensemble non vide d'étiquettes donné, quelle est la forme de l'arbre contenant ces étiquettes dont la profondeur est maximale ? Quelle est la forme de l'arbre contenant ces étiquettes dont la profondeur est minimale ?

1.3 Arbres binaires complets

Déf. III.3 (Arbre binaire complet) Un arbre binaire complet est un arbre binaire dont tous les niveaux sont complets, c'est-à-dire que tous les nœuds d'un même niveau ont deux fils non vides sauf les nœuds du niveau le plus profond qui n'ont aucun fils (c'est-à-dire deux fils vides).

Exemple III.3 (Arbre binaire complet) Le premier arbre binaire de l'exemple III.1 est complet.

Question III.3 Montrer que, dans un arbre binaire complet non vide, le niveau de profondeur p contient 2^p nœuds.

Question III.4 Calculer le nombre n de nœuds d'un arbre binaire complet non vide de profondeur p .

Question III.5 En déduire la profondeur p d'un arbre binaire complet non vide contenant n éléments.

2 Exercice : arbre binaire de recherche

L'objectif de cet exercice est d'étudier une implantation d'un arbre binaire de recherche et son utilisation dans un algorithme de tri d'une séquence d'entiers.

2.1 Arbre binaire de recherche d'entiers

Déf. III.4 (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 ;
- et 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.4 (Arbres binaires de recherche) Le deuxième arbre de l'exemple III.1 est un arbre binaire de recherche.

2.1.1 Représentation des arbres binaires en CaML

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.5 L'expression suivante :

```
Noeud(  
  Noeud( Vide, 3, Vide)  
  6,  
  Noeud(  
    Noeud(  
      Vide,  
      7,  
      Noeud( Vide, 8, Vide)),  
    9,  
    Vide))
```

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

2.2 Tri d'une séquence d'entiers

2.2.1 Séquence d'entiers

Déf. III.5 (Séquence d'entiers) Une séquence s de taille n de valeurs entières v_i avec $1 \leq i \leq n$ est notée $\langle v_1, \dots, v_n \rangle$. Une même valeur v peut figurer plusieurs fois dans s , nous noterons $\text{card}(v, s)$ le cardinal de v dans s , c'est-à-dire le nombre de fois que v figure dans s avec :

$$\text{card}(v, \langle v_1, \dots, v_n \rangle) = \text{card}\{i \in \mathbb{N} \mid 1 \leq i \leq n, v = v_i\}$$

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

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

Soit le programme en langage CaML :

```
let rec ajouter v a =
  match a with
  | Vide -> Noeud(Vide,v,Vide)
  | Noeud (g,e,d) ->
    if (v = e)
    then Noeud(Noeud(g,e,Vide),v,d)
    else
      (if (v < e)
       then Noeud((ajouter v g),e,d)
       else Noeud(g,e,(ajouter v d))));;

let trier s =
  let rec aux1 l r =
    match l with
    | [] -> r
    | t::q -> (aux1 q (ajouter t r)) in
  let rec aux2 a =
    match a with
    | Vide -> []
    | Noeud(g,v,d) -> (aux2 g) @ (v :: (aux2 d)) in
  (aux2 (aux1 s Vide));;
```

Soit la constante `exemple` définie et initialisée par :

```
let exemple = [ 2; 1; 3];;
```

Question III.6 Détailler les étapes du calcul de `(trier exemple)` en précisant pour chaque appel aux fonctions `ajouter`, `aux1`, `aux2` et `trier`, la valeur du paramètre et du résultat.

Question III.7 Soit l'arbre binaire d'entiers p , soit l'entier e et soit l'arbre binaire d'entiers r tel que $r = (\text{ajouter } e \text{ } p)$, montrer que :

$$ABR(p) \Rightarrow ABR(r)$$

Question III.8 Soit la séquence $p = \langle p_1, \dots, p_m \rangle$ de taille m , soit la séquence $r = \langle r_1, \dots, r_n \rangle$ de taille n telle que $r = (\text{trier } p)$, montrer que :

- $m = n$
- $\forall i, 1 \leq i \leq m, \text{card}(p_i, r) = \text{card}(p_i, p)$
- $\forall i, 1 \leq i < n, r_i \leq r_{i+1}$

Question III.9 Montrer que le calcul des fonctions `ajouter`, `aux1`, `aux2` et `trier` se termine quelles que soient les valeurs de leurs paramètres respectant le type des fonctions.

Question III.10 Donner des exemples de valeurs du paramètre `s` de la fonction `trier` qui correspondent aux meilleur et pire cas en nombre d'appels récursifs effectués.

Calculer une estimation de la complexité dans les meilleur et pire cas de la fonction `trier` en fonction du nombre de valeurs dans les séquences données en paramètre. Cette estimation ne prendra en compte que le nombre d'appels récursifs à `ajouter`, `aux1`, `aux2` et `trier` effectués.

3 Problème : Représentation de systèmes creux

La résolution numérique de problèmes en physique repose souvent sur la recherche de solutions pour un système linéaire qui résulte de la discrétisation du modèle mathématique continu utilisé pour représenter le problème. Lorsque les différentes parties du problème sont faiblement couplées, le système linéaire contient un grand nombre de valeurs nulles. Il est alors qualifié de système creux. Cette notion peut être généralisée à un système contenant un grand nombre de valeurs identiques. Nous appelons cette valeur identique, valeur par défaut des éléments. Pour les systèmes faiblement couplés évoqués précédemment, cette valeur par défaut est 0.0.

L'objectif de ce problème est l'étude d'une représentation d'un vecteur creux par un arbre binaire en numérotant les nœuds par les indices des éléments du vecteur et en étiquetant les nœuds par les valeurs des éléments du vecteur différentes de la valeur par défaut. L'utilisation de cette structure permet de réduire le temps d'accès aux éléments du vecteur creux par rapport à l'utilisation naturelle d'une structure de liste. Par contre, cette structure utilise plus de mémoire que la liste mais moins que la représentation habituelle des vecteurs.

Un vecteur creux est représenté par un arbre binaire dont les nœuds sont soit vides, soit étiquetés par les valeurs des éléments contenus dans le vecteur creux qui sont différentes de la valeur par défaut, et les chemins de la racine de l'arbre aux nœuds sont extraits du codage binaire de l'indice de l'élément dont la valeur est contenue dans le nœud.

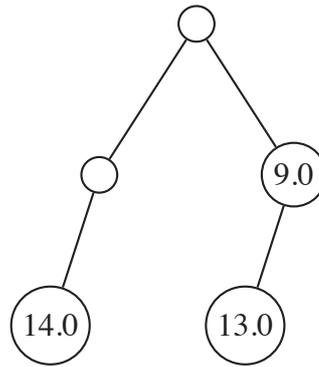
3.1 Arbre binaire partiel de réels

3.1.1 Définition

Un arbre binaire partiel de réels est un arbre binaire de réels dont certains nœuds ne contiennent pas d'étiquettes.

Déf. III.6 (Arbre binaire partiel de réels) Un arbre binaire partiel de réels a est une structure qui peut soit être vide (notée \emptyset), soit être un nœud qui contient une étiquette réelle (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 partiels de réels, soit être une fourche qui est un nœud qui ne contient pas d'étiquette. L'ensemble des fourches de l'arbre a est noté $\mathcal{F}(a)$. L'ensemble des nœuds de l'arbre a qui ne sont pas des fourches est noté $\mathcal{N}(a)$.

Exemple III.6 (Arbre binaire partiel de réels) Voici un exemple d'arbre binaire partiel étiqueté par des réels (les sous-arbres vides des nœuds ne sont pas représentés) :



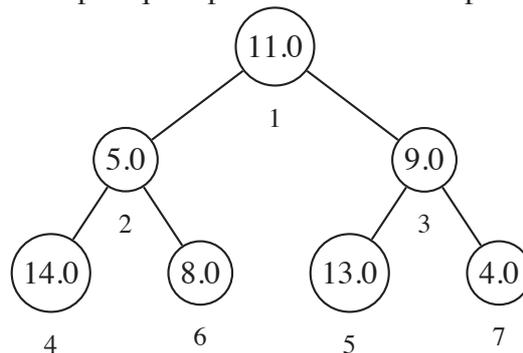
3.1.2 Numérotation hiérarchique des nœuds

Pour exploiter un arbre binaire partiel pour représenter un vecteur creux, il faut associer un indice à chaque nœud de l'arbre contenant une valeur.

Déf. III.7 (Numérotation hiérarchique des nœuds) La numérotation hiérarchique des nœuds d'un arbre binaire consiste à associer le numéro 1 à la racine de l'arbre puis à calculer les numéros des fils dans un nœud à partir du numéro de leur père. Soit un nœud de numéro n à la profondeur p , le numéro de son fils gauche est $n + 2^p$, le numéro de son fils droit est $n + 2^{p+1}$.

Dans l'exemple suivant, le numéro de chaque nœud sera noté en-dessous de son étiquette.

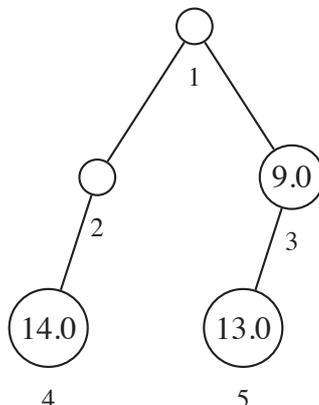
Exemple III.7 Voici un arbre complet qui représente un vecteur plein de 7 éléments.



Le vecteur plein correspondant est :

$\{1 \mapsto 11.0, 2 \mapsto 5.0, 3 \mapsto 9.0, 4 \mapsto 14.0, 5 \mapsto 13.0, 6 \mapsto 8.0, 7 \mapsto 4.0\}$.

Voici un arbre partiel qui représente un vecteur creux de 3 éléments avec une valeur par défaut 0.0.



Le vecteur creux correspondant dont la valeur par défaut 0.0 des éléments est explicitée :

$\{1 \mapsto 0.0, 2 \mapsto 0.0, 3 \mapsto 9.0, 4 \mapsto 14.0, 5 \mapsto 13.0, 6 \mapsto 0.0, 7 \mapsto 0.0\}$.

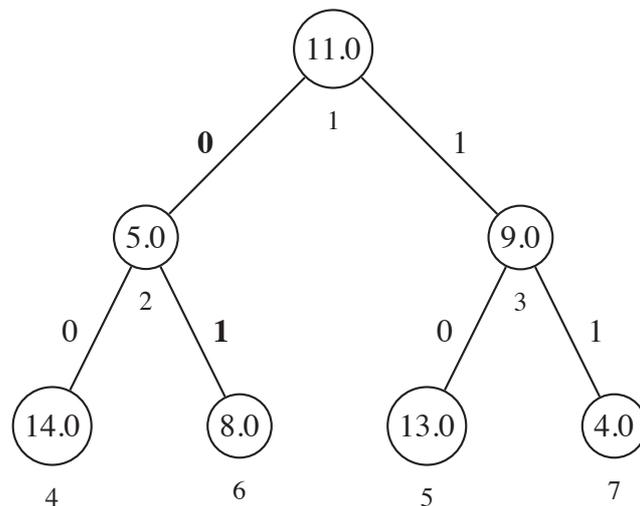
Question III.11 Montrer que l'ensemble des nœuds de profondeur p est égal à l'intervalle $[2^p, 2^{p+1} - 1]$.

Question III.12 Montrer que la numérotation de chaque nœud est unique.

3.1.3 Occurrence d'un nœud dans un arbre

Pour accéder à un nœud, nous devons représenter le chemin dans l'arbre allant de la racine au nœud. Pour cela, nous étiquetons la liaison entre un nœud et son fils gauche par l'entier 0 et la liaison entre un nœud et son fils droit par l'entier 1. Nous appelons alors occurrence, ou chemin, du nœud de numéro n la liste des étiquettes suivies pour aller de la racine à ce nœud. Dans l'exemple suivant, l'occurrence du nœud de numéro 6 étiqueté par la valeur 8.0 est $\langle 0, 1 \rangle$.

Exemple III.8



Si l'occurrence du nœud de numéro n situé à la profondeur p est notée $\langle c_1, \dots, c_p \rangle$ (c_1 étant le lien avec la racine), on remarque que l'occurrence du père de n est $\langle c_1, \dots, c_{p-1} \rangle$.

Question III.13 Soit un nœud de numéro n situé à la profondeur p et d'occurrence $\langle c_1, \dots, c_p \rangle$, montrer que :

$$n = 2^p + \sum_{i=0}^{p-1} c_{i+1} \times 2^i$$

Question III.14 Exprimer la valeur du coefficient c_i en fonction de n et i .

3.1.4 Représentation des arbres binaires partiels en CaML

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

```

type arbre =
| Vide
| Fourche of arbre * arbre
| Noeud of arbre * float * 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. Dans l'appel `Fourche(fg, fd)`, les paramètres `fg` et `fd` sont respectivement le fils gauche et le fils droit de la racine de l'arbre créé.

Exemple III.9 L'expression suivante :

```

Fourche(
  Fourche( Noeud( Vide, 14.0, Vide), Vide),
  Noeud(
  
```

```
Noeud( Vide, 13.0, Vide),  
9.0,  
Vide))
```

est alors associée à l'arbre binaire représenté graphiquement dans l'exemple III.6.

3.1.5 Calcul du nombre de valeurs dans un arbre partiel

Question III.15 *Écrire en CaML une fonction `taille` de type `arbre -> int` telle que l'appel (`taille a`) renvoie le nombre de nœuds étiquetés contenus dans l'arbre binaire partiel `a`, c'est-à-dire le cardinal de $\mathcal{N}(a)$. 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.*

3.1.6 Calcul des bornes d'un arbre partiel

Une paire d'entiers est représentée par le type `paire` défini par :

```
type arbre = int*int;;
```

Question III.16 *Écrire en CaML une fonction `bornes` de type `arbre -> paire` telle que l'appel (`bornes a`) sur un arbre binaire partiel `a` non vide renvoie une paire d'entiers qui correspondent au plus petit, respectivement au plus grand, indice des valeurs contenues dans 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.*

3.1.7 Remplacement d'une valeur dans un arbre partiel

Question III.17 *Écrire en CaML une fonction `remplacer` de type `int -> float -> arbre -> arbre` telle que l'appel (`remplacer i e a`) sur un arbre binaire partiel `a` non vide contenant une valeur pour l'indice `i` renvoie un arbre binaire partiel contenant la valeur `e` à l'indice `i` et les mêmes valeurs que l'arbre `a` pour les autres indices que `i`. 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.*

3.2 Codage d'un vecteur creux par un arbre partiel

L'arbre partiel utilisé dans les questions précédentes correspond à des vecteurs creux dont les valeurs par défaut sont 0.0. Ils ne contiennent donc que les valeurs différentes de 0.0. Cette représentation peut être étendue pour représenter des vecteurs creux avec d'autres valeurs par défaut. Les étiquettes contenues dans l'arbre partiel correspondent alors aux indices et valeurs des éléments du vecteur creux qui sont différentes de la valeur par défaut du vecteur. La valeur par défaut doit faire partie explicitement de la structure du vecteur creux.

Pour améliorer les performances d'accès, la structure de vecteur creux contient en plus de l'arbre partiel et de la valeur par défaut, les minimum et maximum des indices des valeurs contenues dans le vecteur qui sont différentes de la valeur par défaut ainsi que le nombre de valeurs différentes de la valeur par défaut.

Soit v un vecteur creux, nous noterons respectivement $\mathcal{V}_d(v)$, $\mathcal{V}_{min}(v)$, $\mathcal{V}_{max}(v)$, $\mathcal{V}_t(v)$ et $\mathcal{V}_a(v)$, la valeur par défaut, l'indice minimum, l'indice maximum, le nombre de valeurs différentes de la valeur par défaut et l'arbre partiel contenant les valeurs de v .

Déf. III.8 (Vecteur creux bien formé) Un vecteur creux implanté avec un arbre binaire partiel est bien formé si et seulement si :

- les indices minimum et maximum du vecteur creux correspondent aux bornes de l'arbre binaire partiel ;
- l'arbre binaire partiel ne contient pas la valeur par défaut du vecteur creux ;
- le nombre de valeurs contenues dans l'arbre binaire partiel est égal au nombre d'éléments différents de la valeur par défaut du vecteur creux ;
- l'arbre binaire partiel qui contient les valeurs ne doit pas contenir de fourches dont les sous-arbres gauche et droit sont tous deux vides.

Question III.18 Exprimer la définition III.8 sous la forme d'une propriété $VB F(v)$ qui indique que v est un vecteur creux bien formé.

Un vecteur creux est représenté par le type `vecteur` dont la définition est :

```
type vecteur = int * int * int * float * arbre;;
```

Dans l'expression $(imin, imax, t, v, a)$, les paramètres $imin$, $imax$, t , v et a sont respectivement l'indice minimum, l'indice maximum et le nombre des éléments dont la valeur est différente de la valeur par défaut, ainsi que la valeur par défaut et l'arbre binaire partiel contenant les valeurs du vecteur.

Question III.19 Écrire en CaML une fonction `valider` de type `vecteur -> bool` telle que l'appel `(valider v)` renvoie la valeur `true` si le vecteur creux v est bien formé (c'est-à-dire si $VB F(v)$) et la valeur `false` sinon. L'algorithme utilisé ne devra parcourir qu'une seule fois l'arbre associé à v . Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.

Question III.20 Écrire en CaML une fonction `lire` de type `int -> vecteur -> float` telle que l'appel `(lire i v)` renvoie la valeur qui se trouve à l'indice i dans le vecteur creux v . L'algorithme utilisé ne devra parcourir qu'une seule fois l'arbre associé à v . Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.

Question III.21 Écrire en CaML une fonction `ecrire` de type `int -> float -> vecteur -> vecteur` telle que l'appel `(ecrire i e v)`, avec la valeur de e différente de la valeur par défaut de v , renvoie un vecteur creux contenant la valeur e à l'indice i et les mêmes valeurs que le vecteur creux v pour les autres indices que i . L'algorithme utilisé ne devra parcourir qu'une seule fois l'arbre associé à v . Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.

Question III.22 Écrire en CaML une fonction `somme` de type `vecteur -> vecteur -> vecteur` telle que l'appel `(somme v1 v2)` renvoie un vecteur creux dont les éléments ont pour valeur la somme des valeurs des éléments de $v1$ et de $v2$ de même indice. L'algorithme utilisé ne devra parcourir qu'une seule fois les arbres associés à $v1$ et à $v2$. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.

Partie III : Algorithmique et programmation en PASCAL

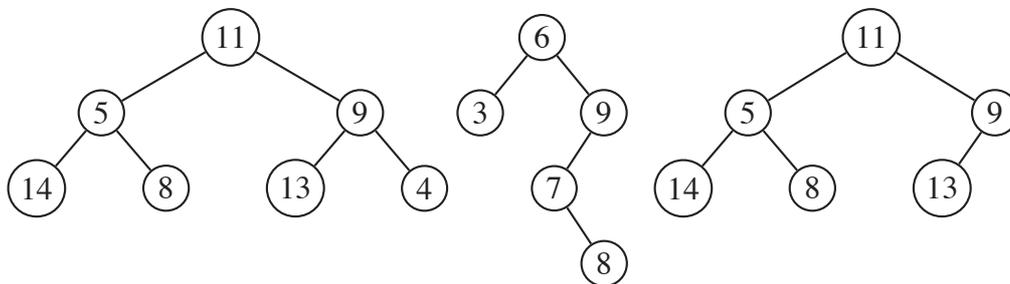
Cette partie doit être traitée par les étudiants qui ont utilisé le langage PASCAL dans le cadre des enseignements d'informatique. Les fonctions écrites devront être récursives ou faire appel à des fonctions auxiliaires récursives. Elles ne devront pas utiliser d'instructions itératives (c'est-à-dire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

1 Préliminaire : Arbre binaire d'entiers

1.1 Définition

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



1.2 Profondeur d'un arbre

Déf. III.2 (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 nœuds de la branche la plus longue. Nous la noterons $|A|$. Nous associerons la profondeur -1 à l'arbre vide \emptyset .

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

Question III.1 Donner une définition de la profondeur d'un arbre a en fonction de \emptyset , $\mathcal{G}(a)$ et $\mathcal{D}(a)$.

Question III.2 Soit un ensemble non vide d'étiquettes donné, quelle est la forme de l'arbre contenant ces étiquettes dont la profondeur est maximale ? Quelle est la forme de l'arbre contenant ces étiquettes dont la profondeur est minimale ?

1.3 Arbres binaires complets

Déf. III.3 (Arbre binaire complet) Un arbre binaire complet est un arbre binaire dont tous les niveaux sont complets, c'est-à-dire que tous les nœuds d'un même niveau ont deux fils non vides sauf les nœuds du niveau le plus profond qui n'ont aucun fils (c'est-à-dire deux fils vides).

Exemple III.3 (Arbre binaire complet) Le premier arbre binaire de l'exemple III.1 est complet.

Question III.3 Montrer que, dans un arbre binaire complet non vide, le niveau de profondeur p contient 2^p nœuds.

Question III.4 Calculer le nombre n de nœuds d'un arbre binaire complet non vide de profondeur p .

Question III.5 En déduire la profondeur p d'un arbre binaire complet non vide contenant n éléments.

2 Exercice : arbre binaire de recherche

L'objectif de cet exercice est d'étudier une implantation d'un arbre binaire de recherche et son utilisation dans un algorithme de tri d'une séquence d'entiers.

2.1 Arbre binaire de recherche d'entiers

Déf. III.4 (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 ;
- et 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.4 (Arbres binaires de recherche) Le deuxième arbre de l'exemple III.1 est un arbre binaire de recherche.

2.1.1 Représentation des arbres binaires en PASCAL

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.

Exemple III.5 L'expression suivante :

```

Noeud(
  Noeud( Vide, 3, Vide)
  6,
  Noeud(
    Noeud(
      Vide,
      7,
      Noeud( Vide, 8, Vide)),
    9,
    Vide))

```

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

2.2 Tri d'une séquence d'entiers

2.2.1 Séquence d'entiers

Déf. III.5 (Séquence d'entiers) Une séquence s de taille n de valeurs entières v_i avec $1 \leq i \leq n$ est notée $\langle v_1, \dots, v_n \rangle$. Une même valeur v peut figurer plusieurs fois dans s , nous noterons $\text{card}(v, s)$ le cardinal de v dans s , c'est-à-dire le nombre de fois que v figure dans s avec :

$$\text{card}(v, \langle v_1, \dots, v_n \rangle) = \text{card}\{i \in \mathbb{N} \mid 1 \leq i \leq n, v = v_i\}$$

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`.

Soit le programme en langage PASCAL :

```

FUNCTION ajouter(v : INTEGER; a : ARBRE) : ARBRE;
VAR
    g, d      : ARBRE;
    e         : INTEGER;
BEGIN
    IF (a = Vide) THEN
        ajouter := Noeud(Vide,v,Vide)
    ELSE BEGIN
        g := Gauche(a);
        d := Droite(a);
        e := Valeur(a);
        IF (v = e) THEN
            ajouter := Noeud(Noeud(g,e,Vide),v,d)
        ELSE
            IF (v < e) THEN
                ajouter := Noeud(ajouter(v,g),e,d)
            ELSE
                ajouter := Noeud(g,e,ajouter(v,d))
            END
        END
    END; { ajouter }

FUNCTION trier(s : SEQUENCE) : SEQUENCE;

    FUNCTION aux1(l : SEQUENCE ; r : ARBRE) : ARBRE;
    VAR
        t      : INTEGER;
        q      : SEQUENCE;
    BEGIN
        IF (l = Vide) THEN
            aux1 := r
        ELSE BEGIN
            t := S_Tete( l );
            q := S_Queue( l );
            aux1 := aux1( q, ajouter( t, r) )
        END
    END; { aux1 }

    FUNCTION aux2(a : ARBRE) : SEQUENCE;
    VAR
        v      : INTEGER;
        g,d    : ARBRE;
    BEGIN
        IF (a = Vide) THEN
            aux2 := Vide
        ELSE BEGIN
            g := A_Gauche(a);
            d := A_Droit(a);
            v := A_Valeur(a);
            aux2 := S_Juxtaposer( aux2(g), S_Inserer( v, aux2(d)))
        END
    END; { aux2 }

```

```

BEGIN
  trier := aux2( aux1( s, Vide))
END; { trier }

```

Soit la constante `exemple` définie et initialisée par :

```

CONST exemple : SEQUENCE
  = S_Inserer( 2, S_Inserer( 1, S_Inserer( 3, Vide)));

```

Question III.6 *Détailler les étapes du calcul de `trier(exemple)` en précisant pour chaque appel aux fonctions `ajouter`, `aux1` et `aux2`, la valeur du paramètre et du résultat.*

Question III.7 *Soit l'arbre binaire d'entiers p , soit l'entier e et soit l'arbre binaire d'entiers r tel que $r = \text{ajouter}(e, p)$, montrer que :*

$$ABR(p) \Rightarrow ABR(r)$$

Question III.8 *Soit la séquence $p = \langle p_1, \dots, p_m \rangle$ de taille m , soit la séquence $r = \langle r_1, \dots, r_n \rangle$ de taille n telle que $r = \text{trier}(p)$, montrer que :*

- a) $m = n$
- b) $\forall i, 1 \leq i \leq m, \text{card}(p_i, r) = \text{card}(p_i, p)$
- c) $\forall i, 1 \leq i < n, r_i \leq r_{i+1}$

Question III.9 *Montrer que le calcul des fonctions `ajouter`, `aux1`, `aux2` et `trier` se termine quelles que soient les valeurs de leurs paramètres respectant le type des fonctions.*

Question III.10 *Donner des exemples de valeurs du paramètre s de la fonction `trier` qui correspondent aux meilleur et pire cas en nombre d'appels récursifs effectués.*

Calculer une estimation de la complexité dans les meilleur et pire cas de la fonction `trier` en fonction du nombre de valeurs dans les séquences données en paramètre. Cette estimation ne prendra en compte que le nombre d'appels récursifs à `ajouter`, `aux1`, `aux2` et `trier` effectués.

3 Problème : Représentation de systèmes creux

La résolution numérique de problèmes en physique repose souvent sur la recherche de solutions pour un système linéaire qui résulte de la discrétisation du modèle mathématique continu utilisé pour représenter le problème. Lorsque les différentes parties du problème sont faiblement couplées, le système linéaire contient un grand nombre de valeurs nulles. Il est alors qualifié de système creux. Cette notion peut être généralisée à un système contenant un grand nombre de valeurs identiques. Nous appelons cette valeur identique, valeur par défaut des éléments. Pour les systèmes faiblement couplés évoqués précédemment, cette valeur par défaut est 0.0.

L'objectif de ce problème est l'étude d'une représentation d'un vecteur creux par un arbre binaire en numérotant les nœuds par les indices des éléments du vecteur et en étiquetant les nœuds par les valeurs des éléments du vecteur différentes de la valeur par défaut. L'utilisation de cette structure permet de réduire le temps d'accès aux éléments du vecteur creux par rapport à l'utilisation naturelle d'une structure de liste. Par contre, cette structure utilise plus de mémoire que la liste mais moins que la représentation habituelle des vecteurs.

Un vecteur creux est représenté par un arbre binaire dont les nœuds sont soit vides, soit étiquetés par les valeurs des éléments contenus dans le vecteur creux qui sont différentes de la valeur par défaut, et les chemins de la racine de l'arbre aux nœuds sont extraits du codage binaire de l'indice de l'élément dont la valeur est contenue dans le nœud.

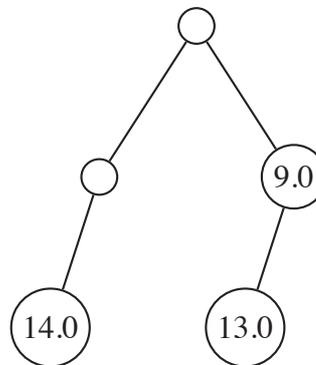
3.1 Arbre binaire partiel de réels

3.1.1 Définition

Un arbre binaire partiel de réels est un arbre binaire de réels dont certains nœuds ne contiennent pas d'étiquettes.

Déf. III.6 (Arbre binaire partiel de réels) Un arbre binaire partiel de réels a est une structure qui peut soit être vide (notée \emptyset), soit être un nœud qui contient une étiquette réelle (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 partiels de réels, soit être une fourche qui est un nœud qui ne contient pas d'étiquette. L'ensemble des fourches de l'arbre a est noté $\mathcal{F}(a)$. L'ensemble des nœuds de l'arbre a qui ne sont pas des fourches est noté $\mathcal{N}(a)$.

Exemple III.6 (Arbre binaire partiel de réels) Voici un exemple d'arbre binaire partiel étiqueté par des réels (les sous-arbres vides des nœuds ne sont pas représentés) :



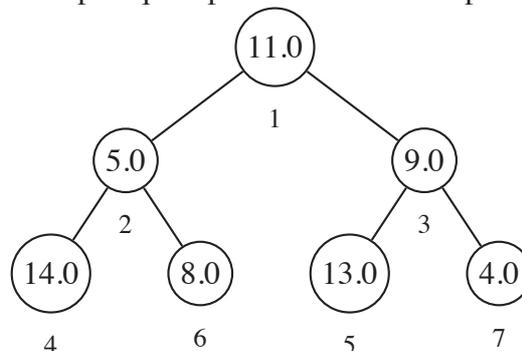
3.1.2 Numérotation hiérarchique des nœuds

Pour exploiter un arbre binaire partiel pour représenter un vecteur creux, il faut associer un indice à chaque nœud de l'arbre contenant une valeur.

Déf. III.7 (Numérotation hiérarchique des nœuds) La numérotation hiérarchique des nœuds d'un arbre binaire consiste à associer le numéro 1 à la racine de l'arbre puis à calculer les numéros des fils dans un nœud à partir du numéro de leur père. Soit un nœud de numéro n à la profondeur p , le numéro de son fils gauche est $n + 2^p$, le numéro de son fils droit est $n + 2^{p+1}$.

Dans l'exemple suivant, le numéro de chaque nœud sera noté en-dessous de son étiquette.

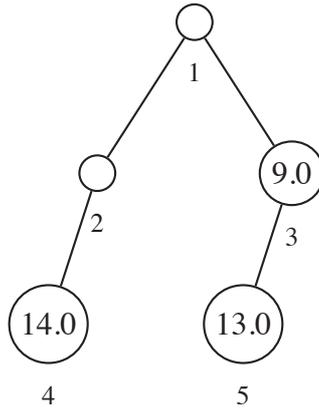
Exemple III.7 Voici un arbre complet qui représente un vecteur plein de 7 éléments.



Le vecteur plein correspondant est :

$\{1 \mapsto 11.0, 2 \mapsto 5.0, 3 \mapsto 9.0, 4 \mapsto 14.0, 5 \mapsto 13.0, 6 \mapsto 8.0, 7 \mapsto 4.0\}$.

Voici un arbre partiel qui représente un vecteur creux de 3 éléments avec une valeur par défaut 0.0.



Le vecteur creux correspondant dont la valeur par défaut 0.0 des éléments est explicitée : $\{1 \mapsto 0.0, 2 \mapsto 0.0, 3 \mapsto 9.0, 4 \mapsto 14.0, 5 \mapsto 13.0, 6 \mapsto 0.0, 7 \mapsto 0.0\}$.

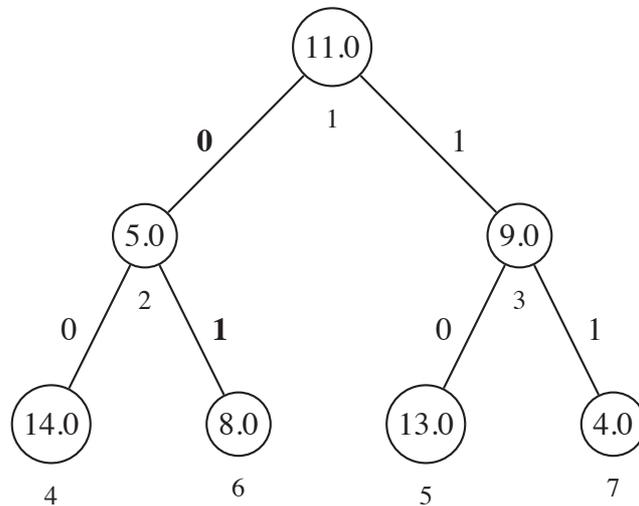
Question III.11 Montrer que l'ensemble des nœuds de profondeur p est égal à l'intervalle $[2^p, 2^{p+1} - 1]$.

Question III.12 Montrer que la numérotation de chaque nœud est unique.

3.1.3 Occurrence d'un nœud dans un arbre

Pour accéder à un nœud, nous devons représenter le chemin dans l'arbre allant de la racine au nœud. Pour cela, nous étiquetons la liaison entre un nœud et son fils gauche par l'entier 0 et la liaison entre un nœud et son fils droit par l'entier 1. Nous appelons alors occurrence, ou chemin, du nœud de numéro n la liste des étiquettes suivies pour aller de la racine à ce nœud. Dans l'exemple suivant, l'occurrence du nœud de numéro 6 étiqueté par la valeur 8.0 est $\langle 0, 1 \rangle$.

Exemple III.8



Si l'occurrence du nœud de numéro n situé à la profondeur p est notée $\langle c_1, \dots, c_p \rangle$ (c_1 étant le lien avec la racine), on remarque que l'occurrence du père de n est $\langle c_1, \dots, c_{p-1} \rangle$.

Question III.13 Soit un nœud de numéro n situé à la profondeur p et d'occurrence $\langle c_1, \dots, c_p \rangle$, montrer que :

$$n = 2^p + \sum_{i=0}^{p-1} c_{i+1} \times 2^i$$

Question III.14 Exprimer la valeur du coefficient c_i en fonction de n et i .

3.1.4 Représentation des arbres binaires partiels en PASCAL

Un arbre binaire partiel 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:REAL;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 Fourche(fg:ARBRE;fd:ARBRE):ARBRE; est une fonction qui renvoie un arbre dont le fils gauche, et le fils droit de la racine sont respectivement fg 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):REAL; est une fonction qui renvoie l'étiquette de la racine de l'arbre a. Cet arbre ne doit pas être vide et sa racine ne doit pas être une fourche ;
- 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.

Exemple III.9 L'expression suivante :

```
Fourche(  
    Fourche( Noeud( Vide, 14.0, Vide), Vide),  
    Noeud(  
        Noeud( Vide, 13.0, Vide),  
        9.0,  
        Vide))
```

est alors associée à l'arbre binaire représenté graphiquement dans l'exemple III.6.

3.1.5 Calcul du nombre de valeurs dans un arbre partiel

Question III.15 Écrire en PASCAL une fonction *taille(a:ARBRE):INTEGER*; telle que l'appel *taille(a)* renvoie le nombre de nœuds étiquetés contenus dans l'arbre binaire partiel a, c'est-à-dire le cardinal de $\mathcal{N}(a)$. 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.

3.1.6 Calcul des bornes d'un arbre partiel

Une paire d'entiers est représentée par le type PAIRE. 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 P_Creer(e1,e2:INTEGER):PAIRE; renvoie une paire d'entiers dont le premier élément est e1 et le second élément e2,
- FUNCTION P_Premier(p:PAIRE):INTEGER; renvoie le premier entier contenu dans la paire d'entiers p,
- FUNCTION P_Second(p:PAIRE):INTEGER; renvoie le second entier contenu dans la paire d'entiers p.

Question III.16 Écrire en PASCAL une fonction *bornes(a:ARBRE):PAIRE*; telle que l'appel *bornes(a)* sur un arbre binaire partiel a non vide renvoie une paire d'entiers qui correspondent au plus petit, respectivement au plus grand, indice des valeurs contenues dans 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.

3.1.7 Remplacement d'une valeur dans un arbre partiel

Question III.17 Écrire en PASCAL une fonction `remplacer(i:INTEGER;e:REAL;a:ARBRE):ARBRE;` telle que l'appel `remplacer(i,e,a)` sur un arbre binaire partiel `a` non vide contenant une valeur pour l'indice `i` renvoie un arbre binaire partiel contenant la valeur `e` à l'indice `i` et les mêmes valeurs que l'arbre `a` pour les autres indices que `i`. 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.

3.2 Codage d'un vecteur creux par un arbre partiel

L'arbre partiel utilisé dans les questions précédentes correspond à des vecteurs creux dont les valeurs par défaut sont 0.0. Ils ne contiennent donc que les valeurs différentes de 0.0. Cette représentation peut être étendue pour représenter des vecteurs creux avec d'autres valeurs par défaut. Les étiquettes contenues dans l'arbre partiel correspondent alors aux indices et valeurs des éléments du vecteur creux qui sont différentes de la valeur par défaut du vecteur. La valeur par défaut doit faire partie explicitement de la structure du vecteur creux.

Pour améliorer les performances d'accès, la structure de vecteur creux contient en plus de l'arbre partiel et de la valeur par défaut, les minimum et maximum des indices des valeurs contenues dans le vecteur qui sont différentes de la valeur par défaut ainsi que le nombre de valeurs différentes de la valeur par défaut.

Soit v un vecteur creux, nous noterons respectivement $\mathcal{V}_d(v)$, $\mathcal{V}_{min}(v)$, $\mathcal{V}_{max}(v)$, $\mathcal{V}_t(v)$ et $\mathcal{V}_a(v)$, la valeur par défaut, l'indice minimum, l'indice maximum, le nombre de valeurs différentes de la valeur par défaut et l'arbre partiel contenant les valeurs de v .

Déf. III.8 (Vecteur creux bien formé) Un vecteur creux implanté avec un arbre binaire partiel est bien formé si et seulement si :

- les indices minimum et maximum du vecteur creux correspondent aux bornes de l'arbre binaire partiel ;
- l'arbre binaire partiel ne contient pas la valeur par défaut du vecteur creux ;
- le nombre de valeurs contenues dans l'arbre binaire partiel est égal au nombre d'éléments différents de la valeur par défaut du vecteur creux ;
- l'arbre binaire partiel qui contient les valeurs ne doit pas contenir de fourches dont les sous-arbres gauche et droit sont tous deux vides.

Question III.18 Exprimer la définition III.8 sous la forme d'une propriété $VBV(v)$ qui indique que v est un vecteur creux bien formé.

Un vecteur creux contenant trois entiers (les valeurs minimale et maximale des indices des valeurs différentes de la valeur par défaut, ainsi que le nombre de valeurs différentes de la valeur par défaut), un réel (la valeur par défaut) et un arbre binaire partiel de réels est représenté par le type `VECTEUR`. 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 V_Creer(imin,imax,t:INTEGER;v:REAL;a:ARBRE):VECTEUR;` renvoie un vecteur creux dont `imin` est l'indice minimum, `imax` est l'indice maximum, `t` est le nombre de valeurs différentes de la valeur par défaut, `v` est la valeur par défaut et l'arbre binaire partiel `a` contient les valeurs dans les nœuds dont le numéro est l'indice de chaque valeur ;
- `FUNCTION V_Minimum(v:VECTEUR):INTEGER;` renvoie l'indice minimum du vecteur creux `v` ;
- `FUNCTION V_Maximum(v:VECTEUR):INTEGER;` renvoie l'indice maximum du vecteur creux `v` ;
- `FUNCTION V_Taille(v:VECTEUR):INTEGER;` renvoie le nombre de valeurs différentes de la valeur par défaut dans le vecteur creux `v` ;
- `FUNCTION V_Defaut(v:VECTEUR):REAL;` renvoie la valeur par défaut du vecteur creux `v` ;

– **FUNCTION** `V_Arbre(v:VECTEUR):ARBRE`; renvoie l'arbre binaire partiel contenant les valeurs du vecteur `v`.

Question III.19 *Écrire en PASCAL une fonction `valider(v:VECTEUR):BOOLEAN`; telle que l'appel `valider(v)` renvoie la valeur `TRUE` si le vecteur creux `v` est bien formé (c'est-à-dire si $VBF(v)$) et la valeur `FALSE` sinon. L'algorithme utilisé ne devra parcourir qu'une seule fois l'arbre associé à `v`. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

Question III.20 *Écrire en PASCAL une fonction `lire(i:INTEGER;v:VECTEUR):REAL`; telle que l'appel `lire(i,v)` renvoie la valeur qui se trouve à l'indice `i` dans le vecteur creux `v`. L'algorithme utilisé ne devra parcourir qu'une seule fois l'arbre associé à `v`. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

Question III.21 *Écrire en PASCAL une fonction `ecrire(i:INTEGER;e:REAL;v:VECTEUR):VECTEUR`; telle que l'appel `ecrire(i,e,v)`, avec la valeur de `e` différente de la valeur par défaut de `v`, renvoie un vecteur creux contenant la valeur `e` à l'indice `i` et les mêmes valeurs que le vecteur creux `v` pour les autres indices que `i`. L'algorithme utilisé ne devra parcourir qu'une seule fois l'arbre associé à `v`. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

Question III.22 *Écrire en PASCAL une fonction `somme(v1,v2:VECTEUR):VECTEUR`; telle que l'appel `somme(v1,v2)` renvoie un vecteur creux dont les éléments ont pour valeur la somme des valeurs des éléments de `v1` et de `v2` de même indice. L'algorithme utilisé ne devra parcourir qu'une seule fois les arbres associés à `v1` et à `v2`. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

Fin de l'énoncé