CONCOURS COMMUNS POLYTECHNIQUES

**EPREUVE SPECIFIQUE - FILIERE MP**

---

# INFORMATIQUE

**Durée : 4 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**

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

# Partie I – Logique et calcul des propositions

De nombreux travaux sont réalisés en Intelligence Artificielle pour construire un programme qui imite le raisonnement humain et soit capable de réussir le test de Turing, c'est-à-dire qu'il ne puisse pas être distingué d'un être humain dans une conversation en aveugle. Vous êtes chargé(e)s de vérifier la correction des réponses données par un tel programme lors des tests de bon fonctionnement. Dans le scénario de test considéré, le comportement attendu est le respect de la règle suivante : pour chaque question, le programme répondra par trois affirmations dont une seule sera correcte.
Nous noterons $A_1$, $A_2$ et $A_3$ les propositions associées aux affirmations effectuées par le programme.

**Question I.1** *Représenter le comportement attendu sous la forme d'une formule du calcul des propositions qui dépend de $A_1$, $A_2$ et $A_3$.*

## 1 Premier cas

Vous demandez au programme : *Quels éléments doivent contenir les aliments que je dois consommer pour préserver ma santé ?*
Il répond les affirmations suivantes :

- $A_1$ Consommez au moins des aliments qui contiennent des glucides, mais pas des lipides !
- $A_2$ Si vous consommez des aliments qui contiennent des glucides alors ne consommez pas d'aliments qui contiennent des lipides !
- $A_3$ Ne consommez aucun aliment qui contient des lipides !

Nous noterons $G$, respectivement $L$, les variables propositionnelles qui correspondent au fait de consommer des aliments qui contiennent des glucides, respectivement des lipides.

**Question I.2** *Exprimer $A_1$, $A_2$ et $A_3$ sous la forme de formules du calcul des propositions. Ces formules peuvent dépendre des variables $G$ et $L$.*

**Question I.3** *En utilisant le calcul des propositions (résolution avec les formules de De Morgan), déterminer ce que doivent contenir les aliments que vous devez consommer pour préserver votre santé.*

## 2 Second cas

Vous demandez au programme : *Quelles activités dois-je pratiquer si je veux préserver ma santé ?*
Suite à une coupure de courant, la dernière affirmation est interrompue.

- $A_1$ Ne faites des activités sportives que si vous prenez également du repos !
- $A_2$ Si vous ne faites pas d'activité intellectuelle alors ne prenez pas de repos !
- $A_3$ Prenez du repos ou faites des activités ...!

Nous noterons $S$, $I$ et $R$ les variables propositionnelles qui correspondent au fait de faire des activités sportives, des activités intellectuelles et de prendre du repos.

**Question I.4** *Exprimer $A_1$, $A_2$ et $A_3$ sous la forme de formules du calcul des propositions. Ces formules peuvent dépendre de $S$, $I$ et $R$.*

**Question I.5** *En utilisant le calcul des propositions (résolution avec les tables de vérité), déterminer quelle(s) activité(s) vous devez pratiquer pour préserver votre santé.*

# Partie II – Automates et langages

Cette partie étudie l'opérateur *racine carrée* $\sqrt{\mathcal{A}}$ pour un automate fini $\mathcal{A}$.

## 1 Automate fini complet déterministe

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

### 1.1 Représentation d'un automate fini complet déterministe

**Déf. II.1 (Alphabet, mot, langage)** L'alphabet $X$ est un ensemble de symboles. $\Lambda \notin X$ est le symbole représentant le mot vide. $X^\star$ est l'ensemble contenant $\Lambda$ et les mots composés de séquences de symboles de $X$. Un langage sur $X$ est un sous-ensemble de $X^\star$.

**Déf. II.2 (Automate fini complet déterministe)** Un automate fini complet déterministe sur $X$ est un quintuplet $\mathcal{A} = (Q, X, i, T, \delta)$ composé :

- d'un ensemble fini d'états : $Q$ ;
- d'un état initial : $i \in Q$ ;
- d'un ensemble d'états terminaux : $T \subseteq Q$ ;
- d'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.

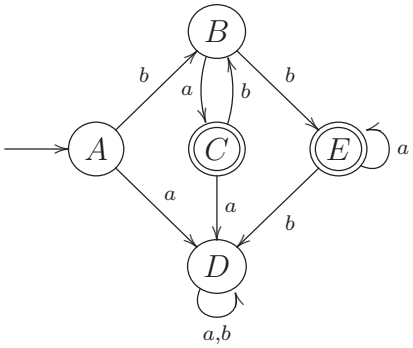
### 1.2 Représentation graphique d'un automate

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

- les valeurs de la fonction totale de transition $\delta$ sont représentées par un graphe orienté dont les nœuds sont les états et les arêtes sont les transitions ;
- l'état initial $i$ est désigné par une flèche $\longrightarrow (i)$ ;
- tout état terminal $t$ est entouré d'un double cercle $((t))$ ;
- 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 $\mathcal{E} = (Q, X, i, T, \delta)$ est représenté par le graphe suivant :

$$Q = \{A, B, C, D, E\} \quad X = \{a, b\} \quad i = A \quad T = \{C, E\}$$
$$\begin{array}{ll} \delta(A, a) = D; & \delta(A, b) = B; \\ \delta(B, a) = C; & \delta(B, b) = E; \\ \delta(C, a) = D; & \delta(C, b) = B; \\ \delta(D, a) = D; & \delta(D, b) = D; \\ \delta(E, a) = E; & \delta(E, b) = D. \end{array}$$



## 1.3 Langage reconnu par un automate fini déterministe

**Déf. II.3 (Transition sur un mot)** L'extension $\delta^\star$ de $\delta$ à $Q \times X^\star \mapsto Q$ est définie par :

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

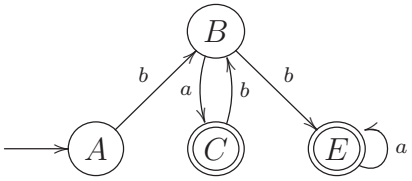
$$\left\{ \begin{array}{l} \forall e \in X, \\ \forall m \in X^\star, \\ \forall o \in Q, \\ \forall d \in Q, \end{array} \right. \delta^\star(o, e.m) = d \Leftrightarrow \exists q \in Q,\ (\delta(o, e) = q) \wedge (\delta^\star(q, m) = d).$$

**Déf. II.4 (Langage reconnu par un automate)** Le langage sur $X$ reconnu par un automate fini déterministe $\mathcal{A}$ est :

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

Notons que certains états et transitions ne sont pas utiles dans la description d'un langage car ils ne permettent pas d'aller de l'état initial à un état terminal. Il s'agit, d'une part, des états qui ne sont pas accessibles ou co-accessibles et, d'autre part, des transitions dont ils sont l'origine ou la destination. Un automate dans lequel ces états et transitions inutiles ont été éliminés est appelé automate émondé.

**Exemple II.2** L'automate émondé extrait de $\mathcal{E}$ (**exemple II.1, page 3**) (composé des états et transitions utiles, c'est-à-dire des états à la fois accessibles et co-accessibles) est représenté par le graphe suivant :




**Question II.1** *Donner, sans la justifier, une expression régulière ou ensembliste représentant le langage $L(\mathcal{E})$ sur $X = \{a, b\}$ reconnu par l'automate $\mathcal{E}$ de* ***l'exemple II.1, page 3****.*

**Question II.2** *Montrer que :*

$$\forall m \in X^\star,\ \forall n \in X^\star,\ \forall o \in Q,\ \forall q \in Q,\ \forall d \in Q,\ (q = \delta^\star(o, m)) \wedge (d = \delta^\star(q, n)) \Leftrightarrow d = \delta^\star(o, m.n).$$

# 2 Racine carrée d'un langage

**Déf. II.5 (Racine carrée d'un langage)** Le langage $\sqrt{L}$ (racine carrée de $L$), est défini par :

$$\sqrt{L} = \{m \in X^\star \mid m.m \in L\}.$$

**Question II.3** *Donner, sans la justifier, une expression régulière ou ensembliste représentant le langage $\sqrt{L(\mathcal{E})}$ sur $X = \{a, b\}$ construit à partir du langage $L(\mathcal{E})$ reconnu par l'automate $\mathcal{E}$ de* ***l'exemple II.1, page 3****.*

**Question II.4** *Soit un langage $L$ sur un alphabet $X$, comparer $L$ avec $\sqrt{L^2}$ puis avec $\left(\sqrt{L}\right)^2$.*

## 3 Racine carrée d'un automate

**Déf. II.6 (Séquence)** Une séquence $s$ de valeurs $v_i$ avec $i \in [d, f]$ est notée $s = \langle v_i \rangle_{i=d}^{f}$. Sa taille est notée $\mid s \mid = f - d + 1$. Si $i \in [d, f]$, sa $i$-ième valeur est notée $s_i = v_i$.
Si les valeurs $v_i$ appartiennent à l'ensemble $V$ alors les séquences de taille $n$ appartiennent à l'ensemble $V^n = \{\langle v_i \rangle_{i=1}^{n}\}$.

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

**Déf. II.7 (Racine carrée)** Soit $\mathcal{A} = (Q_{\mathcal{A}}, X, i_{\mathcal{A}}, T_{\mathcal{A}}, \delta_{\mathcal{A}})$ un automate fini complet déterministe, notons $Q_{\mathcal{A}} = \{q_i\}_{i=0}^{n}$ avec $i_{\mathcal{A}} = q_0$ et $card(Q_{\mathcal{A}}) = n + 1$, alors l'automate $\sqrt{\mathcal{A}} = (Q_{\sqrt{\mathcal{A}}}, X, i_{\sqrt{\mathcal{A}}}, T_{\sqrt{\mathcal{A}}}, \delta_{\sqrt{\mathcal{A}}})$ (racine carrée de $\mathcal{A}$) est défini par :

$$
\begin{aligned}
&Q_{\sqrt{\mathcal{A}}} = {Q_{\mathcal{A}}}^{n+1};\\
&i_{\sqrt{\mathcal{A}}} = \langle q_i \rangle_{i=0}^{n} \in Q_{\sqrt{\mathcal{A}}};\\
&T_{\sqrt{\mathcal{A}}} = \{\langle t_i \rangle_{i=0}^{n} \in Q_{\sqrt{\mathcal{A}}} \mid t_0 = q_j, t_j \in T_{\mathcal{A}}, j \in [0, n]\};\\
&\forall o \in Q_{\sqrt{\mathcal{A}}}, \forall d \in Q_{\sqrt{\mathcal{A}}}, \forall a \in X, d = \delta_{\sqrt{\mathcal{A}}}(o, a) \Leftrightarrow \forall i \in [0, n], d_i = \delta_{\mathcal{A}}(o_i, a).
\end{aligned}
$$

**Question II.5** *En considérant* ***l'exemple II.1, page 3****, construire l'automate* $\sqrt{\mathcal{E}}$ *(seuls les états et les transitions utiles, c'est-à-dire accessibles depuis les états initiaux et co-accessibles depuis les états terminaux, devront être construits).*

**Question II.6** *Caractériser le langage reconnu par* $\sqrt{\mathcal{E}}$ *par une expression régulière ou ensembliste.*

Dans la suite de cette partie, nous considérons que $\mathcal{A}$ est un automate fini complet déterministe.

**Question II.7** *Montrer que* $\sqrt{\mathcal{A}}$ *est un automate fini complet déterministe.*

**Question II.8** *Montrer que :*

$$\forall o \in Q_{\sqrt{\mathcal{A}}}, \forall m \in X^{\star}, {\delta_{\sqrt{\mathcal{A}}}}^{\star}(o, m) = \langle {\delta_{\mathcal{A}}}^{\star}(o_i, m) \rangle_{i=0}^{n}.$$

**Question II.9** *Montrer que :*

$$\forall m \in X^{\star},\ m \in L(\sqrt{\mathcal{A}}) \Leftrightarrow m.m \in L(\mathcal{A}).$$

**Question II.10** *Quelle relation liant les automates* $\mathcal{A}$ *et* $\sqrt{\mathcal{A}}$ *pouvez-vous en déduire ?*

# Partie III – Algorithmique et programmation

Cette partie combine les programmes d'*Informatique pour tous* et d'option *Informatique*. Elle doit être traitée, selon les questions, soit en utilisant le langage Python tel qu'il a été présenté dans le cadre des enseignements d'*Informatique pour tous*, soit en utilisant le langage CaML tel qu'il a été présenté dans le cadre des enseignements d'option *Informatique*.
Les fonctions écrites en langage Python ne devront pas être récursives.
Les fonctions écrites en langage CaML 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, ni d'exceptions.

## 1 Exercice : le tri à bulles

Cet exercice étudie l'algorithme de tri à bulles d'une séquence d'entiers. Pour simplifier les notations et les preuves, les séquences manipulées ne contiennent qu'une seule fois chaque valeur. Les résultats obtenus se généralisent aux séquences quelconques.

**Déf. III.1 (Séquence)** Une séquence $s$ de valeurs $v_i$ avec $i \in [d, f] \subset \mathbb{N}$ est notée $s = \langle v_i \rangle_{i=d}^{f}$.

- Sa taille est notée $\mid s \mid = f - d + 1$ ;
- Si $i \in [d, f]$, sa $i$-ième valeur est notée $s_i = v_i$ ;
- Son domaine, c'est-à-dire l'intervalle des indices de ses valeurs, est noté : $dom(\langle v_i \rangle_{i=d}^{f}) = [d, f]$ ;
- Son co-domaine, c'est-à-dire l'ensemble de ses valeurs, est noté : $codom(\langle v_i \rangle_{i=d}^{f}) = \{v_i\}_{i=d}^{f}$ ;
- La séquence vide de taille 0 est notée $\langle\rangle$ ;
- Si $d \leq d' \leq f' \leq f$ alors $\langle v_i \rangle_{i=d'}^{f'}$ de taille $f' - d' + 1$ désigne la sous-séquence de $\langle v_i \rangle_{i=d}^{f}$ contenant les valeurs de $v_{d'}$ à $v_{f'}$.

Les valeurs contenues dans les séquences $s$ manipulées par la suite sont toutes distinctes, c'est-à-dire que le cardinal du codomaine de $s$ est égal à la taille de $s$ ($card(codom(s)) = \mid s \mid$).

Une séquence sera représentée en CaML et en Python par une liste.

**Question III.1** *Ecrire en CaML une fonction* `lire` *dont le type est* `int -> int list -> int` *telle que l'appel* `(lire i l)` *sur la liste d'entiers* `l` *correspondant à la séquence* $\langle v_i \rangle_{i=1}^{n}$ *de taille* $n$ *avec* `i` $\in [1, n]$ *doit renvoyer l'entier* $v_{\texttt{i}}$*. Cette fonction devra au plus parcourir une seule fois chaque élément de la liste* `l`*. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

Dans la suite de cet exercice, nous considérons la fonction `trier` du programme en langage Python du **listing 1, page 7**.

**Question III.2** *Quelles sont les informations affichées lors de son exécution ? Que contient la variable* `resultat` *après son exécution ? Que contient la variable* `exemple` *après son exécution ? Que se passe-t-il si nous remplaçons l'instruction* `t = copy(p)` *par l'instruction* `t = p` *?*

```
from copy import copy
def trier(p):
    t = copy(p)
    for i in range(len(t)):                # entrée boucle externe
        for j in range(len(t)-i-1):        # entrée boucle interne
            if (t[j+1]<t[j]):
                t[j], t[j+1] = t[j+1], t[j]
            print( i, j, t)
        # sortie boucle interne
    # sortie boucle externe
    return t

exemple = [ 3, 1, 4, 2]
resultat = trier(exemple)
```

Listing 1: tri à bulles en Python

**Question III.3** *Soient les séquences* s *et* r *avec* $dom(\mathtt{s}) = [1, m]$ *et* $dom(\mathtt{r}) = [1, n]$*, telles que* r = trier(s)*, montrer que :*

*(i)* $dom(\mathtt{r}) = dom(\mathtt{s})$*;*

*(ii)* $codom(\mathtt{r}) = codom(\mathtt{s})$*;*

*(iii)* $\forall i \in [1, n[, r_i \leq r_{i+1}$.

*Vous utiliserez pour cela les invariants de boucle :*

— *boucle externe :*
$\forall k \in [n - \mathtt{i}, n[, \mathtt{t}_k \leq \mathtt{t}_{k+1} \wedge dom(\mathtt{t}) = dom(\mathtt{p}) \wedge codom(\mathtt{t}) = codom(\mathtt{p})$*;*

— *boucle interne :*
$\forall k \in [1, j], \mathtt{t}_k \leq \mathtt{t}_{j+1} \wedge dom(\mathtt{t}) = dom(\mathtt{p}) \wedge codom(\mathtt{t}) = codom(\mathtt{p})$.

**Question III.4** *Montrer que le calcul de la fonction* trier *se termine quelles que soient les valeurs de son paramètre* p*.*

**Question III.5** *Donner des exemples de valeurs du paramètre* p *de la fonction* trier *qui correspondent au pire cas en temps d'exécution.*
*Montrer que la complexité en temps d'exécution dans le pire cas de la fonction* trier *en fonction de la taille* $n$ *des séquences données en paramètre est de* $O(n^2)$*.*

**Question III.6** *Ecrire en CaML une fonction* trier *dont le type est* int list -> int list *et qui utilise le même algorithme que la fonction* trier *du programme Python du* ***listing 1 ci-dessus*** *sans afficher les valeurs intermédiaires. L'appel* (trier s) *sur une liste d'entiers* s *doit donc renvoyer une liste d'entiers triée* r *qui contient les mêmes éléments que la liste* s*, c'est-à-dire, satisfaisant les propriétés (i), (ii) et (iii) de la* ***question III.3 ci-dessus****. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

## 2 Problème : le tri par tas

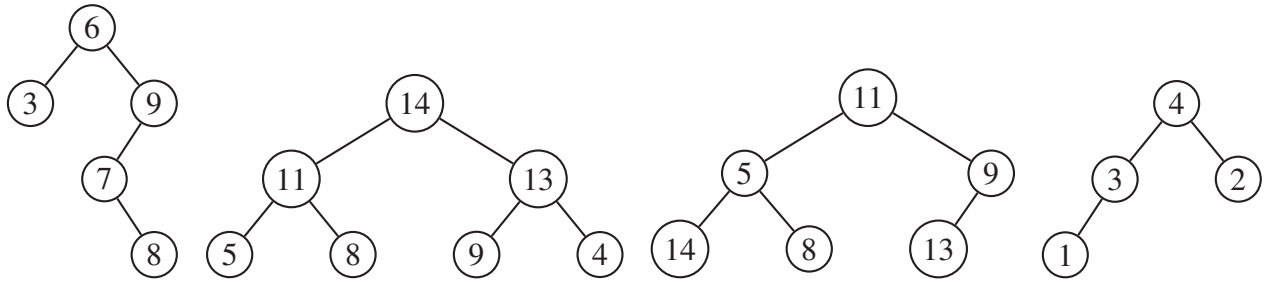
La complexité de l'algorithme de tri à bulles étudié dans l'exercice précédent ne permet pas de l'utiliser pour de gros volumes de données. Ce problème étudie un algorithme de tri plus performant basé sur la structure d'arbre en tas, c'est-à-dire d'arbre binaire parfait partiellement ordonné. Nous considérerons une implantation arborescente en langage CaML et une implantation sous la forme de tableau en langage Python.

### 2.1 Arbre binaire d'entiers

**Déf. III.2 (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. La taille de l'arbre $a$, notée $\mid a \mid$ est le nombre de nœuds de l'arbre $a$.
Cette définition s'exprime sous la forme de la propriété $\mathcal{B}(a)$ qui caractérise les arbres binaires d'entiers $a$ :

$$\mathcal{B}(a) \equiv (a = \emptyset) \vee \big(a \neq \emptyset \wedge \mathcal{B}(\mathcal{G}(a)) \wedge \mathcal{B}(\mathcal{D}(a)) \wedge \mathcal{E}(a) \in \mathbb{N}\big).$$

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




#### 2.1.1 Implantation en langage CaML

Un arbre binaire d'entiers est représenté en langage CaML 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 binaire d'entiers créé.

**Exemple III.2** L'expression suivante :

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

est alors associée au premier arbre binaire représenté graphiquement dans **l'exemple III.1, page 8**.

**Question III.7** *Donner l'expression en langage CaML qui correspond au quatrième arbre binaire de* ***l'exemple III.1, page 8****.*

### 2.1.2 Hauteur dans un arbre binaire

**Déf. III.3 (Hauteur dans un arbre binaire)** Soit $a$ un arbre binaire et $n$ un nœud de $a$. La hauteur de $n$ dans $a$ est égale au nombre de nœuds du chemin sans cycle le plus long reliant $n$ à un sous-arbre vide. Nous la noterons $\eta(n)$. Si $n$ est la racine de l'arbre ($n = a$), il s'agit alors de la hauteur de l'arbre. Nous associerons la hauteur $0$ à l'arbre binaire vide $\emptyset$.

**Exemple III.3 (Hauteurs)** Les hauteurs des quatre arbres binaires de **l'exemple III.1, page 8** sont respectivement 4, 3, 3 et 3.

**Question III.8** *Donner une définition mathématique de la hauteur $\eta(a)$ d'un arbre binaire $a$ en fonction de $\emptyset$, $\mathcal{G}(a)$ et $\mathcal{D}(a)$.*

**Question III.9** *Soit un ensemble non vide d'étiquettes donné, quelle est la structure de l'arbre binaire contenant ces étiquettes dont la hauteur est maximale ? Quelle est la structure de l'arbre binaire contenant ces étiquettes dont la hauteur est minimale ? Justifier vos réponses.*

### 2.1.3 Profondeur d'un nœud et niveau dans un arbre binaire

**Déf. III.4 (Profondeur d'un nœud, niveau dans un arbre binaire)** Soit un arbre binaire $a$ contenant un nœud $n$, la profondeur du nœud $n$ dans $a$ notée $\pi(n)$ est définie par :

$$\pi(n) = \eta(a) - \eta(n).$$

Un niveau dans un arbre binaire $a$ est la séquence de gauche à droite des nœuds de même profondeur dans $a$.

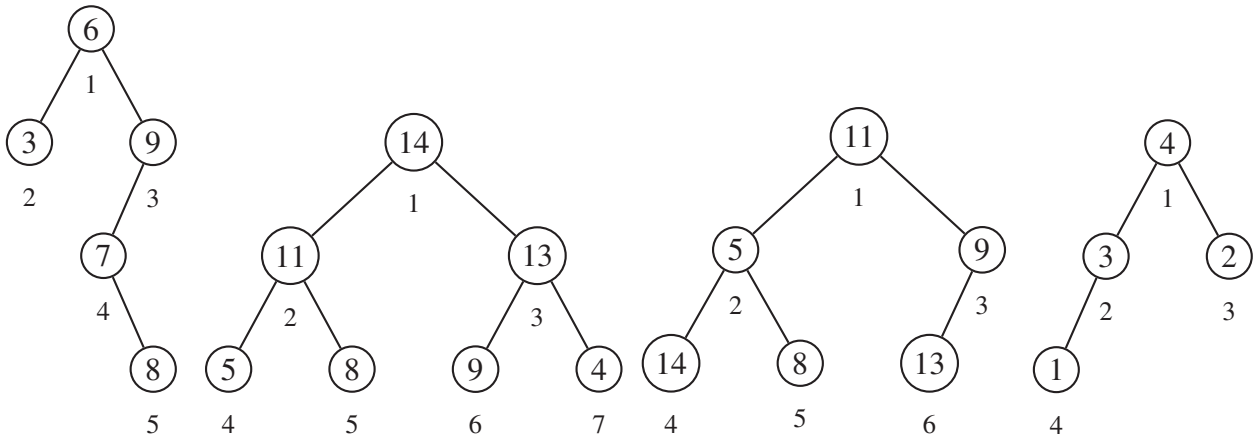
**Exemple III.4 (Profondeurs)** Les étiquettes des nœuds de même profondeur dans les quatre arbres de **l'exemple III.1, page 8** sont :

| Profondeur | Arbre 1 | Arbre 2 | Arbre 3 | Arbre 4 |
|---|---|---|---|---|
| 0 | 6 | 14 | 11 | 4 |
| 1 | 3, 9 | 11, 13 | 5, 9 | 3, 2 |
| 2 | 7 | 5, 8, 9, 4 | 14, 8, 13 | 1 |
| 3 | 8 | | | |

### 2.1.4 Numérotation hiérarchique des nœuds d'un arbre binaire

La numérotation hiérarchique des nœuds d'un arbre binaire $a$ consiste à associer à chaque nœud un numéro compris entre 1 et $\mid a \mid$ par un parcours en largeur partant de la racine (numéro 1) et en parcourant chaque niveau de gauche à droite jusqu'au dernier nœud : le plus profond et le plus à droite (numéro $\mid a \mid$). Nous noterons $\mathcal{N}_i(a)$ le nœud de l'arbre binaire $a$ de numéro $i$ avec $i \in [1, \mid a \mid]$. Dans les exemples suivants, le numéro de chaque nœud sera noté en-dessous de son étiquette.

**Exemple III.5 (Numérotation hiérarchique)** La numérotation hiérarchique des nœuds des quatre arbres de **l'exemple III.1, page 8** produit les arbres numérotés suivants (les sous-arbres vides qui sont fils gauche ou droit des nœuds ne sont pas représentés) :




**Question III.10** *Ecrire en CaML une fonction* `lire` *dont le type est* `int -> arbre -> int` *telle que l'appel* `(lire i a)` *sur l'arbre binaire d'entiers* `a` *avec* `i` $\in [1, \mid$ `a` $\mid]$ *doit renvoyer l'entier* $\mathcal{E}(\mathcal{N}_{\texttt{i}}(\texttt{a}))$. *Cette fonction devra au plus parcourir une seule fois chaque élément de l'arbre* `a`. *Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

## 2.2 Arbre binaire partiellement ordonné d'entiers

**Déf. III.5 (Arbre binaire partiellement ordonné d'entiers)** Un arbre binaire d'entiers $a$ est partiellement ordonné si :

- les fils gauche et droit de $a$ sont des arbres binaires partiellement ordonnés d'entiers ;
- les étiquettes de tous les nœuds composant les fils gauche et droit de $a$ sont inférieures ou égales à l'étiquette de $a$.

Cette définition s'exprime sous la forme de la propriété $\mathcal{O}(a)$ qui caractérise les arbres binaires partiellement ordonnés d'entiers $a$ :

$$\mathcal{O}(a) \equiv (a = \emptyset) \vee (a \neq \emptyset \wedge \mathcal{O}(\mathcal{G}(a)) \wedge \mathcal{O}(\mathcal{D}(a)) \wedge \mathcal{E}(\mathcal{G}(a)) \leq \mathcal{E}(a) \wedge \mathcal{E}(\mathcal{D}(a)) \leq \mathcal{E}(a)).$$

**Exemple III.6 (Arbres binaires partiellement ordonnés d'entiers)** Le deuxième et le quatrième arbres de **l'exemple III.1, page 8** sont des arbres binaires partiellement ordonnés d'entiers.

**Question III.11** *Ecrire en CaML une fonction* `verifier` *dont le type est* `arbre -> bool` *telle que l'appel* `(verifier a)` *sur l'arbre binaire d'entiers* `a` *renvoie la valeur* `true` *si* $\mathcal{O}$(`a`) *et la valeur* `false` *sinon. Cette fonction devra au plus parcourir une seule fois chaque nœud de l'arbre* `a`. *Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

## 2.3 Arbres binaires complets

**Déf. III.6 (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).
Cette définition s'exprime sous la forme de la propriété $\mathcal{C}_n(a)$ qui caractérise les arbres binaires complets $a$ de hauteur $n$ ($\eta(a) = n$) :

$$\mathcal{C}_n(a) \equiv (a = \emptyset \wedge n = 0) \vee (a \neq \emptyset \wedge n \neq 0 \wedge \mathcal{C}_{n-1}(\mathcal{G}(a)) \wedge \mathcal{C}_{n-1}(\mathcal{D}(a))).$$

**Exemple III.7 (Arbre binaire complet)** Le deuxième arbre de **l'exemple III.1, page 8** est complet.

**Question III.12** *Montrer que, dans un arbre binaire complet non vide $a$, le niveau de profondeur $p$ contient $2^p$ nœuds.*

**Question III.13** *Calculer le nombre $n$ de nœuds d'un arbre binaire complet non vide $a$ de hauteur $p = \eta(a)$.*

**Question III.14** *En déduire la hauteur $p = \eta(a)$ d'un arbre binaire complet non vide $a$ contenant $n$ éléments ($n = \mid a \mid$).*

## 2.4 Arbres binaires parfaits

**Déf. III.7 (Arbre binaire parfait)** Un arbre binaire parfait est un arbre binaire dont tous les niveaux sont complets sauf le niveau le plus profond qui peut être incomplet auquel cas ses nœuds sont alignés à gauche de l'arbre.
Cette définition s'exprime sous la forme de la propriété $\mathcal{P}_n(a)$ qui caractérise l'arbre binaire parfait $a$ de hauteur $n$ :

$$\mathcal{P}_n(a) \equiv \mathcal{C}_n(a) \vee a \neq \emptyset \wedge n \neq 0 \wedge \left\{ \begin{array}{ll} & n \neq 1 \wedge \mathcal{P}_{n-1}(\mathcal{G}(a)) \wedge \mathcal{C}_{n-2}(\mathcal{D}(a)) \\ \vee & n \neq 1 \wedge \mathcal{C}_{n-1}(\mathcal{G}(a)) \wedge \mathcal{C}_{n-2}(\mathcal{D}(a)) \\ \vee & \mathcal{C}_{n-1}(\mathcal{G}(a)) \wedge \mathcal{P}_{n-1}(\mathcal{D}(a)) \end{array} \right.$$

**Exemple III.8 (Arbres binaires parfaits)** Le troisième et le quatrième arbres de **l'exemple III.1, page 8** sont parfaits. Le deuxième est également parfait car il est complet.

Soit le type énuméré `categorieArbre` en langage CaML distinguant les arbres binaires complets, parfaits et quelconques :

```
type categorieArbre = Complet| Parfait| Quelconque;;
```

**Question III.15** *Ecrire en CaML une fonction* `analyser` *dont le type est* `arbre -> int * categorieArbre` *telle que l'appel* `(analyser a)` *sur l'arbre binaire* `a` *renvoie une paire contenant la hauteur $\eta$(a) de l'arbre* `a` *ainsi que la valeur* `Complet` *si $\mathcal{C}_{\eta(a)}$(a), sinon la valeur* `Parfait` *si $\mathcal{P}_{\eta(a)}$(a) sinon la valeur* `Quelconque`*. Cette fonction devra au plus parcourir une seule fois chaque nœud de l'arbre* `a`*. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

**Question III.16** *Soit un nœud de numéro $n$ dans le niveau de profondeur $p$ d'un arbre binaire parfait, calculer le nombre de nœuds qui se trouvent à sa gauche dans le niveau de profondeur $p$.*

**Question III.17** *Dans le niveau de profondeur $p+1$ d'un arbre binaire parfait, quel est le nombre de nœuds qui se trouvent à la gauche des fils du nœud de numéro $n$ ($n$ faisant partie du niveau de profondeur $p$).*

**Question III.18** *Soit un nœud de numéro $n$ d'un arbre binaire parfait, calculer les numéros de ses fils gauche et droit.*

**Question III.19** *Déduire de la question précédente, le numéro du père du nœud de numéro $n$ dans un arbre binaire parfait.*

**Question III.20** *Ecrire en CaML une fonction* `lire` *dont le type est* `int -> arbre -> int` *telle que l'appel* `(lire i a)` *sur l'arbre binaire parfait* `a` *avec* `i` $\in [1, |$ `a` $|]$ *doit renvoyer l'entier* $\mathcal{E}(\mathcal{N}_{\texttt{i}}(\texttt{a}))$. *Cette fonction devra au plus parcourir une seule fois chaque élément de la branche qui conduit de la racine de* `a` *au nœud de numéro* `i`. *Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

Soit le programme en langage CaML :

```
let construire l =
  let rec aux1 l a =
    match l,a with
      | ([],_) -> (a,l)
      | (t::q,Vide) -> (Noeud(Vide,t,Vide),q)
      | (_,Noeud(g,v,d)) ->
        match (aux1 l g) with
          | (rga,[]) ->
            ((Noeud(rga,v,d)),[])
          | (rga,rgl) ->
            (match (aux1 rgl d) with
              | (rda,rdl) ->
                ((Noeud(rga,v,rda)),rdl))
  in
  let rec aux2 l a =
    match (aux1 l a) with
      | (ra,[]) -> ra
      | (ra,rl) -> (aux2 rl ra)
  in
  (aux2 l Vide);;

let exemple = [ 1; 2; 3; 4; 5; 6];;
```

**Question III.21** *Détailler les étapes du calcul de* `(construire exemple)` *en précisant pour chaque appel aux fonctions* `construire`, `aux1` *et* `aux2`, *la valeur du paramètre et du résultat.*

**Question III.22** *Soient les séquences d'entiers* `s` *et* `r` *avec* $dom(\texttt{s}) = [1, m]$ *et* $dom(\texttt{r}) = [1, n]$*, les arbres binaires d'entiers* `a` *et* `b` *tels que* $\mathcal{C}_p(\texttt{a})$ *et* `(b,r)` *est la paire renvoyée par* `(aux1 s a)`*, montrer que :*

*(i)* $\forall i, 1 \leq i < 2^p, \mathcal{E}(\mathcal{N}_i(\texttt{b})) = \mathcal{E}(\mathcal{N}_i(\texttt{a}))$ *;*

*(ii)* $m = 0 \implies (\mathcal{C}_p(b) \wedge n = 0)$ *;*

*(iii)* $1 \leq m < 2^p \implies (\mathcal{P}_{p+1}(b) \wedge n = 0 \wedge \forall i \in dom(s), \mathcal{E}(\mathcal{N}_{i+2^p-1}(\texttt{b})) = s_i)$ *;*

*(iv)* $m \geq 2^p \implies (\mathcal{C}_{p+1}(b) \wedge n = m - 2^p \wedge \forall i \in dom(r), r_i = s_{i+2^p-1})$.

**Question III.23** *Soit la séquence* `s` *avec* $dom(\texttt{s}) = [1, m]$*, soit l'arbre binaire d'entiers* `a`*, tels que* `a = (construire s)`*, montrer que :*

*(i)* $\mathcal{P}_p(\texttt{a})$ *;*

*(ii)* $\mid a \mid = m$ *;*

*(iii)* $2^{p-1} \leq m < 2^p$ *;*

*(iv)* $\forall i \in dom(s), \mathcal{E}(\mathcal{N}_i(\texttt{a})) = s_i$.

**Question III.24** *Montrer que le calcul des fonctions* `aux1`*,* `aux2` *et* `construire` *se termine quelles que soient les valeurs de leurs paramètres respectant le type des fonctions.*

**Question III.25** *Donner des exemples de valeurs du paramètre* `s` *de la fonction* `construire` *qui correspondent au pire cas en nombre d'appels récursifs effectués.*
*Montrer que la complexité de la fonction* `construire` *en fonction de la taille* $n$ *des séquences données en paramètre est de* $O(n^2)$*. Cette estimation ne prend en compte que le nombre d'appels récursifs effectués.*

## 2.5 Arbres en tas

**Déf. III.8 (Arbre en tas)** Un arbre en tas est un arbre binaire parfait partiellement ordonné.
Cette définition s'exprime sous la forme de la propriété $\mathcal{T}_n(a)$ qui caractérise les arbres en tas $a$ de hauteur $n$ :

$$\mathcal{T}_n(a) \equiv \mathcal{P}_n(a) \wedge \mathcal{O}(a).$$

**Exemple III.9 (Arbres en tas)** Le deuxième et le quatrième arbres binaires de **l'exemple III.1, page 8** sont en tas.

Lorsqu'un arbre binaire parfait n'est pas partiellement ordonné, les étiquettes des nœuds peuvent être permutées pour obtenir un arbre en tas sans changer la structure d'arbre binaire parfait.

### 2.5.1 Implantation en langage CaML

**Question III.26** *Ecrire en CaML une fonction* `placer` *dont le type est* `arbre -> int -> arbre -> arbre` *telle que l'appel* `(placer g v d)` *sur l'entier* `v` *et les arbres en tas* `g` *et* `d` *tels que* $\eta(\texttt{g}) = \eta(\texttt{d})$ *ou* $\eta(\texttt{g}) = \eta(\texttt{d}) + 1$ *(*$\eta$ *est* ***définie page 9****), renvoie un arbre en tas contenant les même étiquettes que* `g` *et* `d` *ainsi que l'étiquette* `v`*. Cette fonction devra au plus parcourir une branche de* `g` *ou de* `d`*. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

**Question III.27** *Proposer la modification la plus simple possible de la fonction* `construire` *étudiée précédemment* ***page 12*** *pour que l'arbre résultat ait une structure d'arbre en tas.*

### 2.5.2 Implantation par un tableau en langage Python

La structure d'arbre parfait peut être implantée très efficacement par un tableau contenant les étiquettes de l'arbre selon une numérotation hiérarchique des nœuds de l'arbre. Les nœuds de l'arbre seront désignés par leur numéro. L'indice le plus petit de la séquence doit être au moins égal à 1. Vous noterez que, dans le cas où l'indice le plus petit de la séquence est strictement plus grand que 1, toutes les cases du tableau ne correspondent pas à des nœuds de l'arbre. Il est également possible de préciser l'indice de la dernière case utilisée pour représenter le tableau. Alors, les cases suivantes ne correspondent pas à des nœuds.

**Exemple III.10 (Arbres binaires parfaits représentés par un tableau)** Les deuxième, troisième et quatrième arbres de **l'exemple III.1, page 8** sont parfaits. Ils sont représentés par les tableaux suivants (les entiers des lignes supérieures sont les numéros des nœuds et les entiers des lignes inférieures correspondent aux valeurs de leurs étiquettes) :

| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| 14 | 11 | 13 | 5 | 8 | 9 | 4 |

| 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|
| 11 | 5 | 9 | 14 | 8 | 13 |

| 1 | 2 | 3 | 4 |
|---|---|---|---|
| 4 | 3 | 2 | 1 |

Pour le premier tableau, si l'indice correspondant à la racine de l'arbre parfait est le 3, alors seules les cases 3, 6 et 7 correspondent à des nœuds de l'arbre (**exemple III.5, page 10**).

**Question III.28** *Soient :*

- $\mathtt{a}$ *une séquence d'entiers avec* $dom(\mathtt{a}) = [1, n]$ *;*
- $\mathtt{r}$ *et* $\mathtt{f}$ *des entiers avec* $1 \leq \mathtt{r} \leq \mathtt{f} \leq n$ *;*

*tels que :*

- $\mathtt{a}$ *représente un arbre binaire parfait avec* $\mathtt{f}$ *l'indice du dernier nœud ;*
- *si l'étiquette* $\mathcal{E}(\mathcal{N}_{\mathtt{r}}(\mathtt{a}))$ *est remplacée par le maximum de* $\mathcal{E}(\mathcal{G}(\mathcal{N}_{\mathtt{r}}(\mathtt{a})))$ *et* $\mathcal{E}(\mathcal{D}(\mathcal{N}_{\mathtt{r}}(\mathtt{a})))$ *alors* $\mathcal{N}_{\mathtt{r}}(\mathtt{a})$ *est un arbre en tas.*

*Ecrire en Python une procédure* `placer(r,a,f)` *qui permute le contenu de certaines cases du tableau* $\mathtt{a}$ *pour transformer l'arbre binaire parfait contenu dans* $\mathtt{a}$ *avant l'appel dont le dernier nœud est de numéro* $\mathtt{f}$*, en un arbre binaire parfait contenu dans* $\mathtt{a}$ *après l'appel tel que* $\mathcal{N}_{\mathtt{r}}(\mathtt{a})$ *soit un arbre en tas dont le dernier nœud est de numéro* $\mathtt{f}$*. Seules les cases de* $\mathcal{N}_{\mathtt{r}}(\mathtt{a})$ *peuvent être modifiées. Cette procédure est donc telle que :*

- $a$ *est le contenu de* $\mathtt{a}$ *avant l'exécution de l'appel* `placer(r,a,f)` *;*
- $a'$ *est le contenu de* $\mathtt{a}$ *après l'exécution de l'appel* `placer(r,a,f)` *;*
- $\mathcal{P}_m(a) \wedge \mathcal{P}_m(a') \wedge \mathcal{T}_p(\mathcal{N}_{\mathtt{r}}(a'))$ *;*
- $2^{m-1} \leq n < 2^m \wedge p + \mathtt{r} \in \{m-1, m\}$ *;*
- $\langle a_i \rangle_{i=1}^{\mathtt{r}-1} = \langle a'_i \rangle_{i=1}^{\mathtt{r}-1}$ *;*
- $\{a_i\}_{i=\mathtt{r}}^{\mathtt{f}} = \{a'_i\}_{i=\mathtt{r}}^{\mathtt{f}}$ *;*
- $\langle a_i \rangle_{i=\mathtt{f}+1}^{n} = \langle a'_i \rangle_{i=\mathtt{f}+1}^{n}$.

*Cette fonction devra au plus parcourir une branche de l'arbre binaire parfait représenté par* $\mathtt{a}$*. Cette fonction ne devra pas être récursive ni faire appel à des fonctions auxiliaires récursives.*

**Question III.29** *Ecrire en Python une procédure* `entasser(a)` *qui permute le contenu de certaines cases du tableau d'entiers* `a` *pour transformer l'arbre binaire parfait contenu dans* `a` *avant l'appel en un arbre en tas contenu dans* `a` *après l'appel. Cette procédure est donc telle que :*

- `a` *est une séquence d'entiers avec* $dom(\texttt{a}) = [1, n]$ *;*
- $a$ *est le contenu de* `a` *avant l'exécution de l'appel* `entasser(a)` *;*
- $a'$ *est le contenu de* `a` *après l'exécution de l'appel* `entasser(a)` *;*
- $\mathcal{P}_m(a)$ *;*
- $\mathcal{T}_m(a')$ *;*
- $2^{m-1} \leq n < 2^m$ *;*
- $codom(a) = codom(a')$.

*Cette fonction ne devra pas être récursive ni faire appel à des fonctions auxiliaires récursives.*

## 2.6 Tri par tas

La structure d'arbre en tas permet d'implanter un algorithme de tri efficace appelé tri par tas. La structure d'arbre binaire partiellement ordonné assure que l'étiquette de la racine de l'arbre en tas contient la plus grande étiquette du tas. L'algorithme répète l'étape suivante jusqu'à ce que le tas soit vide :

- l'étiquette de la racine est placée à la fin de la séquence triée ;
- le dernier nœud du tas selon la numérotation hiérarchique est enlevé du tas. Son étiquette remplace celle de la racine. L'arbre binaire ainsi obtenu est parfait. Les sous-arbres gauche et droit de sa racine sont des arbres en tas ;
- l'arbre binaire parfait ainsi obtenu est ensuite converti en arbre en tas en utilisant la fonction `entasser`.

**Question III.30** *Détailler les étapes de l'algorithme du tri par tas pour le quatrième arbre de* ***l'exemple III.1, page 8****. Pour chaque étape, donner les représentations du tas sous les formes d'arbre et de tableau.*

### 2.6.1 Implantation en langage Python

L'utilisation d'un tableau pour représenter un tas permet de permuter l'étiquette de la racine et celle du dernier nœud en une seule étape et de construire le tableau trié à partir de la fin.

**Question III.31** *Ecrire en Python une procédure* `trier(s)` *qui permute le contenu de certaines cases du tableau* `s` *pour trier ce tableau. Cette procédure est donc telle que :*

- `s` *est une séquence d'entiers avec* $dom(\texttt{s}) = [1, n]$ *;*
- $s$ *est le contenu de* `s` *avant l'exécution de l'appel* `trier(s)` *;*
- $s'$ *est le contenu de* `s` *après l'exécution de l'appel* `trier(s)` *;*
- $codom(s) = codom(s')$ *;*
- $\forall i \in [1, n[, s'_i \leq s'_{i+1}$.

*Cette fonction ne devra pas être récursive ni faire appel à des fonctions auxiliaires récursives.*

### 2.6.2 Implantation en langage CaML

Le langage CaML ne permet pas de permuter en une seule étape les étiquettes de la racine et du dernier nœud. Cette permutation sera donc combinée avec les opérations `placer` et `entasser` qui réorganisent l'arbre binaire pour préserver la structure de tas.

**Question III.32** *Ecrire en CaML une fonction* `remonter` *dont le type est* `int -> arbre -> arbre` *telle que l'appel* `(remonter i a)` *sur l'entier* `i` *avec* $\texttt{i} = |\texttt{a}|$ *et l'arbre en tas* `a`*, renvoie un arbre en tas de taille* $|\texttt{a}| - 1$ *qui contient les mêmes étiquettes que* `a` *sauf celle de la racine* $\mathcal{E}(\texttt{a})$*. Cette fonction devra au plus parcourir une fois la branche allant de la racine au dernier nœud de* `a`*. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

**Question III.33** *Ecrire en CaML une fonction* `trier` *dont le type est* `int list -> int list` *telle que l'appel* `(trier l)` *sur la liste d'entiers* `l` *renvoie une liste triée d'entiers contenant les mêmes valeurs que* `l`*. Cette fonction exploitera la technique du tri par tas. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

**Fin de l'énoncé**