Les calculatrices sont interdites.

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.

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

Partie I : Automates et langages

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

1 Automate fini complet déterministe

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

1.1 Représentation d'un automate fini complet déterministe

Déf. I.1 (Automate fini complet déterministe) *Soit l'alphabet X (un ensemble de symboles), soit Λ le symbole représentant le mot vide ($\Lambda \notin X$), soit $X^\star$ l'ensemble contenant Λ et les mots composés de séquences de symboles de X (donc $\Lambda \in X^\star$) ; un automate fini complet déterministe sur X est un quintuplet $A = (Q, X, i, T, \delta)$ composé de :*

- *Un ensemble fini d'états : Q ;*
- *Un état initial : $i \in Q$;*
- *Un ensemble d'états terminaux : $T \subseteq Q$;*
- *Une fonction totale de transition confondue avec son graphe : $\delta \subseteq Q \times X \mapsto Q$.*

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

1.2 Représentation graphique d'un automate

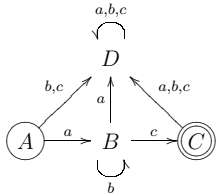
Les automates peuvent être représentés par un schéma suivant les conventions :

- les valeurs de la fonction totale de transition δ sont représentées par un graphe orienté dont les nœuds sont les états et les arêtes sont les transitions ;
- un état initial est entouré d'un cercle (i) ;
- un état terminal est entouré d'un double cercle $((t))$;
- un état qui est à la fois initial et terminal est entouré d'un triple cercle $(((it)))$;
- une arête étiquetée par le symbole $e \in X$ va de l'état o à l'état d si et seulement si $\delta(o, e) = d$.

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

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

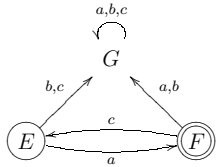
est représenté par le graphe suivant :



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

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

est représenté par le graphe suivant :



1.3 Langage reconnu par un automate fini complet déterministe

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

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

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

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

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

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

2 Composition d'automates finis complets déterministes

2.1 Définition

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

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

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

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

Question I.2 *En considérant les exemples I.1 et I.2, construire le graphe représentant l'automate $\mathcal{E}_1 + \mathcal{E}_2$. (seuls les états et les transitions utiles, c'est-à-dire accessibles depuis les états initiaux, devront être construits).*

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

2.2 Propriétés

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

Question I.5 *Montrer que :*

$$\begin{cases} \forall o_1 \in Q_1, \\ \forall o_2 \in Q_2, \\ \forall m \in X^\star, \ \ \delta^\star_{1+2}((o_1, o_2), m) = (d_1, d_2) \Leftrightarrow (\delta^\star_1(o_1, m) = d_1) \wedge (\delta^\star_2(o_2, m) = d_2) \\ \forall d_1 \in Q_1, \\ \forall d_2 \in Q_2, \end{cases}$$

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

$$m \in L(\mathcal{A}_1 + \mathcal{A}_2) \Leftrightarrow m \in L(\mathcal{A}_1) \vee m \in L(\mathcal{A}_2)$$

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

Partie II : 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 (`for`, `while`, ...) ni de références.

De nombreux algorithmes de recherche reposent sur l'utilisation d'un ensemble de données ordonnées selon une clé. Les opérations essentielles sont l'accès à la donnée dont la clé est la plus petite ainsi que l'élimination de cette donnée minimale. Le but de cet exercice est l'étude d'une réalisation particulière de cette structure à base de tas binomiaux. L'objectif de cette réalisation est de réduire la durée des opérations d'ajout, d'union, de recherche et d'élimination du minimum de l'ensemble.

Pour simplifier le problème, la donnée manipulée est un nombre entier et la clé est la valeur de cet entier. Le problème consiste donc à représenter des ensembles de nombres entiers qui offrent les opérations suivantes :

- Ajout d'un nombre à l'ensemble ;
- Accès au plus petit nombre de l'ensemble ;
- Élimination du plus petit nombre de l'ensemble ;
- Union de deux ensembles.

Nous nous limiterons à l'étude des opérations d'ajout et d'union.

1 Préliminaires

Les algorithmes d'ajout et d'union pour les tas binomiaux sont semblables aux opérations d'incrémentation et d'addition sur des entiers naturels codés en binaire. Nous allons donc étudier ces opérations en préliminaire.

1.1 Opération «ou exclusif»

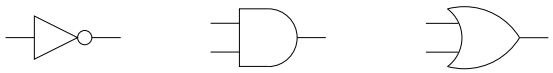
Nous allons définir l'opération booléenne «ou exclusif» dans le cadre du calcul des propositions puis réaliser celle-ci par un circuit électronique et un programme.

Déf. II.1 (Ou exclusif) *Soient deux variables propositionnelles A et B, l'opérateur logique «ou exclusif» noté $A \oplus B$ est tel que cette proposition est fausse si les deux variables ont la même valeur et vraie dans les autres cas.*

Question II.1 *Donner la table de vérité de $A \oplus B$.*

Question II.2 *Donner une formule du calcul des propositions en forme normale disjonctive (disjonction de conjonctions de variables et/ou négation de variables) équivalente à $A \oplus B$.*

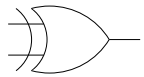
Les portes logiques sont représentées graphiquement par les symboles :



Négation Et Ou

Question II.3 *Proposer un circuit électronique composé de portes logiques comportant deux entrées A et B et une sortie S tel que $S = A \oplus B$.*

Nous noterons par la suite ce circuit sous la forme d'une porte logique, représentée graphiquement par le symbole :



Ou exclusif

Nous allons définir une fonction qui calcule le «ou exclusif».

Question II.4 *Écrire en CaML une fonction `xor` de type `bool -> bool -> bool` telle que l'appel `(xor A B)` renvoie la valeur `true` si la proposition $A \oplus B$ est vraie et `false` sinon.*

1.2 Codage binaire des nombres entiers naturels

Le codage binaire d'un nombre entier naturel est représenté par une liste de valeurs booléennes $b_0, \ldots, b_l$.

Déf. II.2 (Codage binaire d'un entier naturel) *Soit un entier naturel $n \in \mathbb{N}$, le codage binaire de l'entier naturel n est la séquence $b_0, \ldots, b_l$ telle que :*

$$\forall i, 0 \leq i \leq l, b_i \in \{0, 1\}$$
$$n = \sum_{i=0}^{l} b_i \times 2^i$$
$$n = 0 \Rightarrow (l = 0 \wedge b_0 = 0)$$
$$n \neq 0 \Rightarrow b_l = 1$$

Question II.5 *Calculer la valeur de l en fonction de la valeur de n.*

1.2.1 Représentation en CaML

Le codage binaire d'un entier naturel est représenté par le type `entier` équivalent à une liste de `bool`. La valeur `false` correspond à 0, la valeur `true` à 1.

```
type entier == bool list;;
```

Son parcours sera donc effectué de la même manière que celui d'une liste.

1.2.2 Codage d'un entier naturel au format binaire

La première opération réalisée est le codage d'un nombre entier naturel au format binaire.

Question II.6 *Écrire en CaML une fonction `codage` de type `int -> entier` telle que l'appel `(codage n)` renvoie une liste de booléens composée des coefficients $b_0, \ldots, b_l$ correspondant au codage binaire de `n`. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

Question II.7 *Calculer une estimation de la complexité de la fonction `codage` en fonction de la valeur de l'entier naturel `n`. Cette estimation ne prendra en compte que le nombre d'appels récursifs effectués.*

1.2.3 Décodage d'un entier naturel au format binaire

La seconde opération réalisée est le décodage d'un code binaire en nombre entier naturel.

Question II.8 *Écrire en CaML une fonction* `decodage` *de type* `entier -> int` *telle que l'appel* `(decodage E)` *renvoie un entier naturel dont la valeur est calculée à partir des coefficients* $b_0, \ldots, b_l$ *contenus dans la liste de booléens* `E`*. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

1.3 Incrémentation en codage binaire

Nous allons étudier l'opération d'incrémentation d'un nombre entier naturel en codage binaire.

1.3.1 Logique des propositions

Nous allons définir l'opération d'incrémentation par des opérations logiques sur les booléens composant le codage binaire de l'entier naturel.

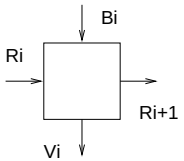
Question II.9 *Soit* $b_0, \ldots, b_l$ *le codage binaire d'un entier naturel* n*, calculer* $v_0, \ldots, v_m$ *le codage de l'entier naturel* $n+1$*. Utiliser pour cela la suite des retenues* $r_0, \ldots, r_m$ *du calcul de l'addition chiffre par chiffre. Exprimer* v_i *et* r_{i+1} *en fonction de* b_i *et* r_i *pour* $0 \leq i \leq l$*. Préciser les valeurs de* m*,* r_0 *et* v_m*.*

1.3.2 Réalisation par un circuit électronique

Nous allons réaliser l'opération d'incrémentation en utilisant des composants électroniques. Celle-ci repose sur une cellule élémentaire d'incrémentation qui est dupliquée $l+1$ fois.

Question II.10 *Proposer un circuit électronique composé de portes logiques qui réalise une cellule comportant deux entrées* b_i *et* r_i *et deux sorties* v_i *et* r_{i+1} *dont le comportement correspond à la réponse proposée à la question II.9.*

Cette cellule sera représentée graphiquement par :



Question II.11 *Proposer un circuit électronique composé de telles cellules pour incrémenter un nombre codé sur* 3 *bits.*

1.3.3 Réalisation par un programme

Nous allons programmer l'opération d'incrémentation d'un nombre entier naturel codé en binaire.

Question II.12 *Écrire en CaML une fonction* `incrementation` *de type* `entier -> entier` *telle que l'appel* `(incrementation E)` *renvoie un entier naturel codé en binaire dont la valeur est égale à la valeur de l'entier naturel codé en binaire* `E` *augmentée de 1. L'algorithme utilisé ne devra parcourir qu'une seule fois la liste* `E`*. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

1.4 Addition en codage binaire

Nous allons étudier l'opération d'addition de deux nombres entiers naturels en codage binaire.

1.4.1 Logique des propositions

Nous allons définir l'opération d'addition par des opérations logiques sur les booléens composant le codage binaire des entiers naturels.

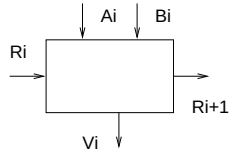
Question II.13 *Soient* $a_0, \ldots, a_m$ *le codage binaire d'un entier naturel* a *et* $b_0, \ldots, b_n$ *le codage binaire d'un entier naturel* b*, calculer* $v_0, \ldots, v_p$ *le codage de l'entier naturel* $a+b$*. Utiliser pour cela la suite des retenues* $r_0, \ldots, r_p$ *du calcul de l'addition chiffre par chiffre. Exprimer* v_i *et* r_{i+1} *en fonction de* a_i*,* b_i *et* r_i *pour* $0 \leq i \leq \min(m,n)$*. Préciser les valeurs de* p*,* r_0*,* v_i *et* r_{i+1} *pour* $\min(m,n) < i \leq \max(m,n)$ *et* v_p*.*

1.4.2 Réalisation par un circuit électronique

Nous allons réaliser l'opération d'addition en utilisant des composants électroniques. Celle-ci repose sur une cellule élémentaire d'addition qui est dupliquée $\min(m,n)+1$ fois ainsi que sur la cellule d'incrémentation.

Question II.14 *Proposer un circuit électronique composé de portes logiques comportant trois entrées* a_i*,* b_i *et* r_i *et deux sorties* v_i *et* r_{i+1} *dont le comportement correspond à la réponse proposée à la question II.13 pour* $0 \leq i \leq \min(m,n)$*.*

Cette cellule sera représentée graphiquement par :



Question II.15 *Proposer un circuit électronique composé de cellules d'addition et d'incrémentation pour additionner deux nombres codés sur* 2 *et* 4 *bits.*

1.4.3 Réalisation par un programme

Nous allons programmer l'opération d'addition de deux nombres entiers naturels codés en binaire.

Question II.16 *Écrire en CaML une fonction* `addition` *de type* `entier -> entier -> entier` *telle que l'appel* `(addition E1 E2)` *renvoie un entier naturel codé en binaire dont la valeur est égale à la somme des valeurs des entiers naturels codés en binaire par* `E1` *et* `E2`*. L'algorithme utilisé ne devra parcourir qu'une seule fois les listes* `E1` *et* `E2`*. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

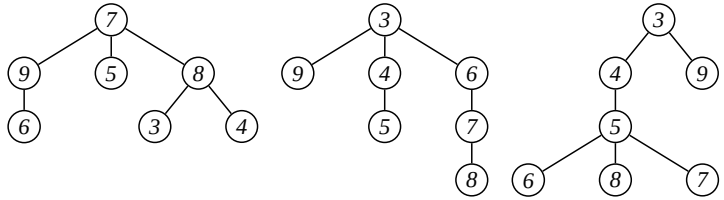
2 Arbres d'entiers en tas

Pour améliorer les performances en temps d'accès aux valeurs, un ensemble d'entiers peut être réalisé par un arbre en utilisant les étiquettes des nœuds pour représenter les valeurs contenues dans l'ensemble.

2.1 Arbres d'entiers

Déf. II.3 (Arbre d'entiers) *Un arbre d'entiers a est une structure qui peut soit être vide (notée $\emptyset$), soit être un nœud qui contient une étiquette entière (notée $\mathcal{E}(a)$) et des fils (notée $\mathcal{S}(a)$), une suite, éventuellement vide, de sous-arbres qui sont tous des arbres d'entiers non vides.*

Exemple II.1 (Arbres d'entiers) *Voici trois exemples d'arbres d'entiers :*



Question II.17 *Exprimer la définition II.3 sous la forme d'une propriété $A(a)$ qui est vraie si et seulement si a est un arbre d'entiers. $A(a)$ sera écrite en exploitant $\emptyset$ et $\mathcal{S}(a)$.*

2.2 Représentation des arbres d'entiers en CaML

Un arbre d'entiers et une liste d'arbres d'entiers sont représentés par les types CaML :

```
type arbre = Vide | Noeud of int * arbres
and arbres == arbre list;;
```

Dans l'appel `Noeud( e, s)`, les paramètres `e` et `s` sont respectivement l'étiquette et la liste des fils de la racine de l'arbre créé.

Exemple II.2 *Le terme suivant*

```
Noeud( 7, [
   Noeud( 9, [
      Noeud( 6, [] ) ] ) ;
   Noeud( 5, [] ) ;
   Noeud( 8, [
      Noeud( 3, [] ) ;
      Noeud( 4, [] ) ] ) ] )
```

est alors associé au premier arbre d'entiers représenté graphiquement dans l'exemple II.1.

2.3 Taille d'un arbre

Déf. II.4 (Taille d'un arbre) *La taille d'un arbre d'entiers est le nombre de nœuds contenus dans l'arbre a. Nous la noterons $\mathcal{T}(a)$.*

Exemple II.3 (Tailles) *La taille des trois arbres de l'exemple II.1 est de 7.*

Question II.18 *Écrire en CaML une fonction `taille` de type `arbre -> int` telle que l'appel `(taille a)` renvoie la taille de l'arbre d'entiers `a`. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

2.4 Hauteur d'un arbre

Déf. II.5 (Hauteur d'un arbre) *Les branches d'un arbre relient la racine aux feuilles. La hauteur d'un arbre a est égale au nombre d'arcs de la branche la plus longue. Nous la noterons $\mathcal{H}(a)$.*

Exemple II.4 (Hauteur d'un arbre) *Les hauteurs des trois arbres de l'exemple II.1 sont respectivement 2, 3 et 3.*

Question II.19 *Donner une définition de la hauteur d'un arbre a en fonction de $\emptyset$ et $\mathcal{S}(a)$.*

Question II.20 *Considérons un arbre d'entiers de taille n. Quelle est la forme de l'arbre dont la hauteur est maximale ? Quelle est la forme de l'arbre dont la hauteur est minimale ? Quelle est la hauteur de l'arbre en fonction de n dans ces deux cas ?*

Question II.21 *Écrire en CaML une fonction `hauteur` de type `arbre -> int` telle que l'appel `(hauteur a)` renvoie la hauteur de l'arbre d'entiers `a`. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

2.5 Profondeur d'un nœud

Déf. II.6 (Profondeur d'un nœud) *La profondeur d'un nœud n dans un arbre a est égale au nombre d'arcs entre la racine et le nœud dans l'arbre. Nous la noterons $\mathcal{P}(n, a)$.*

Exemple II.5 (Profondeur d'un nœud) *Les profondeurs du nœud 7 dans les trois arbres de l'exemple II.1 sont respectivement 0, 2 et 3.*

2.6 Arbres d'entiers en tas

Déf. II.7 (Arbre d'entiers en tas) *Un arbre d'entiers est en tas si les valeurs contenues dans les fils de la racine sont toutes strictement supérieures à la valeur contenue dans la racine et si les fils de la racine sont en tas.*

Exemple II.6 (Arbres d'entiers en tas) *Le premier arbre de l'exemple II.1 n'est pas en tas. Par contre, les deuxième et troisième arbres du même exemple sont en tas.*

Question II.22 *Exprimer la définition précédente sous la forme d'une propriété $AT(a)$ qui est vraie si et seulement si a est un arbre d'entiers en tas. $AT(a)$ est écrite en exploitant $\emptyset$, $\mathcal{E}(a)$ et $\mathcal{S}(a)$.*

Question II.23 *Écrire en CaML une fonction `validerAT` de type `arbre -> bool` telle que l'appel `(validerAT A)` renvoie la valeur `true` si la propriété $AT(A)$ est vraie et la valeur `false` sinon. L'algorithme utilisé ne devra parcourir qu'une seule fois l'arbre. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

3 Arbre binomial d'entiers en tas

3.1 Définitions

Déf. II.8 (Arbre binomial d'entiers en tas) *Un arbre binomial d'ordre k d'entiers en tas est un arbre non vide d'entiers en tas défini récursivement par :*

- *L'arbre d'ordre 0 ne contient qu'un seul nœud ;*

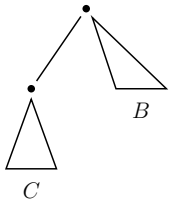
– *La racine d'un arbre d'ordre $k+1$ possède $k+1$ fils qui sont tous des arbres binomiaux d'entiers en tas dont les ordres sont strictement décroissants de gauche (ordre k) à droite (ordre 0).*

Question II.24 *Donner un exemple d'arbre binomial d'entiers en tas pour chacun des ordres 0, 1, 2 et 3.*

3.2 Construction d'un arbre binomial en tas

La structure d'arbre binomial en tas permet de composer deux arbres d'ordre k pour obtenir un arbre d'ordre $k+1$ avec une complexité $O(1)$.

Question II.25 *Montrer que tout arbre binomial A d'ordre $k+1$ d'entiers en tas est composé de deux arbres binomiaux B et C d'ordre k d'entiers en tas tels que l'arbre A est obtenu à partir de B en ajoutant C comme fils le plus à gauche : la racine de A, égale à la racine de B, a comme liste de fils, de gauche à droite, C suivi des fils de la racine de B.*



3.3 Propriétés d'un arbre binomial en tas

Question II.26 *Calculer la taille d'un arbre binomial d'ordre k.*

Question II.27 *Calculer la hauteur d'un arbre binomial d'ordre k.*

Question II.28 *Montrer que le nombre de nœuds de profondeur p dans un arbre binomial d'ordre k, avec $k \geq p$ est égal au coefficient du binôme $\binom{p}{k}$.*

3.4 Validation d'un arbre binomial en tas

Question II.29 *Exprimer la définition équivalente proposée à la question II.25 sous la forme d'une propriété $ABT_k(a)$ qui indique que a est un arbre binomial en tas d'ordre k en exploitant $\emptyset$, $\mathcal{E}(a)$ et $\mathcal{S}(a)$.*

Question II.30 *Écrire en CaML une fonction* `validerABT` *de type* `int -> arbre -> bool` *telle que l'appel* `(validerABT k A)` *renvoie la valeur* `true` *si la propriété $ABT_k(A)$ est vraie et la valeur* `false` *sinon. L'algorithme utilisé ne devra parcourir qu'une seule fois l'arbre. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

4 Tas binomial

La structure de tas binomial consiste à utiliser une suite d'arbres binomiaux en tas, soit vides, soit d'ordre strictement croissant, pour représenter un ensemble de taille quelconque.

4.1 Définitions

Déf. II.9 (Tas binomial) *Un tas binomial est une suite d'arbres binomiaux $a_0, \ldots, a_l$ tels que, soit a_i est vide, soit a_i est un arbre binomial d'ordre i. Si le tas n'est pas vide, alors $a_l \neq \emptyset$.*

Déf. II.10 (Signature d'un tas binomial) *Soit $a_0, \ldots, a_l$ un tas binomial, sa signature est une suite $s_0, \ldots, s_l$ telle que $s_i = 0$ si et seulement si a_i est vide et $s_i = 1$ sinon.*

Déf. II.11 (Taille d'un tas binomial) *La taille d'un tas binomial est la somme du nombre de nœuds contenus dans les arbres qui composent le tas t. Nous la noterons $\mathcal{T}(t)$.*

Question II.31 *Calculer la taille d'un tas binomial $a_0, \ldots, a_l$ à partir de sa signature.*

Question II.32 *Montrer que la signature d'un tas binomial ne dépend que de la taille du tas.*

4.2 Représentation des tas binomiaux en CaML

Un tas binomial est une liste d'arbres d'entiers. Il est donc représenté par le type `arbres` (voir 2.2).

4.3 Validation d'un tas binomial

En préliminaire, nous allons réaliser une opération de validation d'une liste d'arbres qui vérifie qu'il s'agit bien d'un tas binomial.

Question II.33 *Exprimer la définition II.9 sous la forme d'une propriété $TB(t)$ qui indique que t est un tas binomial.*

Question II.34 *Écrire en CaML une fonction* `validerTB` *de type* `arbres -> bool` *telle que l'appel* `(validerTB T)` *renvoie la valeur* `true` *si la propriété $TB($*`T`*$)$ est vraie et la valeur* `false` *sinon. L'algorithme utilisé ne devra parcourir qu'une seule fois le tas. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

4.4 Ajout d'une valeur dans un tas binomial

La première opération consiste à insérer une valeur dans un tas binomial en préservant sa structure.

Nous faisons l'hypothèse que le tas binomial ne contient pas déjà cette valeur.

Question II.35 *Montrer que la signature du tas résultant de l'ajout d'une valeur à un tas est égale au résultat de l'incrémentation binaire de la signature de tas initial. En déduire un algorithme pour l'ajout d'une valeur dans un tas binomial.*

4.5 Fusion de tas binomiaux

La deuxième opération consiste à fusionner deux tas binomiaux, c'est-à-dire construire un tas binomial qui contient les mêmes valeurs que les deux tas binomiaux que l'on souhaite fusionner.

Nous faisons l'hypothèse que les deux tas binomiaux sont disjoints, c'est-à-dire qu'ils ne contiennent pas les mêmes éléments.

Question II.36 *Montrer que la signature du tas résultant de la fusion est égale au résultat de l'addition binaire des signatures des deux tas initiaux. En déduire un algorithme pour la fusion de deux tas binomiaux.*

Partie II : 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 (`for`, `while`, `repeat`, ...).

De nombreux algorithmes de recherche reposent sur l'utilisation d'un ensemble de données ordonnées selon une clé. Les opérations essentielles sont l'accès à la donnée dont la clé est la plus petite ainsi que l'élimination de cette donnée minimale. Le but de cet exercice est l'étude d'une réalisation particulière de cette structure à base de tas binomiaux. L'objectif de cette réalisation est de réduire la durée des opérations d'ajout, d'union, de recherche et d'élimination du minimum de l'ensemble.

Pour simplifier le problème, la donnée manipulée est un nombre entier et la clé est la valeur de cet entier. Le problème consiste donc à représenter des ensembles de nombres entiers qui offrent les opérations suivantes :

- Ajout d'un nombre à l'ensemble ;
- Accès au plus petit nombre de l'ensemble ;
- Élimination du plus petit nombre de l'ensemble ;
- Union de deux ensembles.

Nous nous limiterons à l'étude des opérations d'ajout et d'union.

1 Préliminaires

Les algorithmes d'ajout et d'union pour les tas binomiaux sont semblables aux opérations d'incrémentation et d'addition sur des entiers naturels codés en binaire. Nous allons donc étudier ces opérations en préliminaire.

1.1 Opération «ou exclusif»

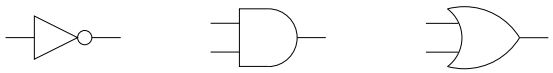
Nous allons définir l'opération booléenne «ou exclusif» dans le cadre du calcul des propositions puis réaliser celle-ci par un circuit électronique et un programme.

Déf. II.1 (Ou exclusif) *Soient deux variables propositionnelles A et B, l'opérateur logique «ou exclusif» noté $A \oplus B$ est tel que cette proposition est fausse si les deux variables ont la même valeur et vraie dans les autres cas.*

Question II.1 *Donner la table de vérité de $A \oplus B$.*

Question II.2 *Donner une formule du calcul des propositions en forme normale disjonctive (disjonction de conjonctions de variables et/ou négation de variables) équivalente à $A \oplus B$.*

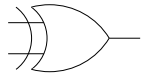
Les portes logiques sont représentées graphiquement par les symboles :



Négation Et Ou

Question II.3 *Proposer un circuit électronique composé de portes logiques comportant deux entrées A et B et une sortie S tel que $S = A \oplus B$.*

Nous noterons par la suite ce circuit sous la forme d'une porte logique, représentée graphiquement par le symbole :



Ou exclusif

Nous allons définir une fonction qui calcule le «ou exclusif».

Question II.4 *Écrire en PASCAL une fonction `xor(A:BOOLEAN;B:BOOLEAN):BOOLEAN;` telle que l'appel `xor(A,B)` renvoie la valeur `true` si la proposition `A` $\oplus$ `B` est vraie et `false` sinon.*

1.2 Codage binaire des nombres entiers naturels

Le codage binaire d'un nombre entier naturel est représenté par une liste de valeurs booléennes $b_0, \ldots, b_l$.

Déf. II.2 (Codage binaire d'un entier naturel) *Soit un entier naturel $n \in \mathbb{N}$, le codage binaire de l'entier naturel n est la séquence $b_0, \ldots, b_l$ telle que :*

$$\forall i, 0 \leq i \leq l, b_i \in \{0, 1\}$$
$$n = \sum_{i=0}^{l} b_i \times 2^i$$
$$n = 0 \Rightarrow (l = 0 \wedge b_0 = 0)$$
$$n \neq 0 \Rightarrow b_l = 1$$

Question II.5 *Calculer la valeur de l en fonction de la valeur de n.*

1.2.1 Représentation en PASCAL

Le code binaire d'un entier naturel est représenté par le type de base `ENTIER` correspondant à une liste de `BOOLEAN`. La valeur `FALSE` correspond à 0, la valeur `TRUE` à 1.

Son parcours sera donc effectué de la même manière que celui d'une liste.

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 :

- `NIL` représente la liste vide de booléens ;
- `FUNCTION E_Insertion(v:BOOLEAN;E:ENTIER):ENTIER;` renvoie une liste de booléens composée d'un premier booléen `v` et du reste de la liste contenu dans `E` ;
- `FUNCTION E_Premier(E:ENTIER):BOOLEAN;` renvoie le premier booléen de la liste `E`. Cette liste ne doit pas être vide ;
- `FUNCTION E_Reste(E:ENTIER):ENTIER;` renvoie le reste de la liste `E` privée de son premier booléen. Cette liste ne doit pas être vide.

1.2.2 Codage d'un entier naturel au format binaire

La première opération réalisée est le codage d'un nombre entier naturel au format binaire.

Question II.6 *Écrire en PASCAL une fonction* `codage(n:INTEGER):ENTIER;` *telle que l'appel* `codage(n)` *renvoie une liste de booléens composée des coefficients* $b_0, \ldots, b_l$ *correspondant au codage binaire de* `n`*. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

Question II.7 *Calculer une estimation de la complexité de la fonction* `codage` *en fonction de la valeur de l'entier naturel* `n`*. Cette estimation ne prendra en compte que le nombre d'appels récursifs effectués.*

1.2.3 Décodage d'un entier naturel au format binaire

La seconde opération réalisée est le décodage d'un code binaire en nombre entier naturel.

Question II.8 *Écrire en PASCAL une fonction* `decodage(E:ENTIER):INTEGER;` *telle que l'appel* `decodage(E)` *renvoie un entier naturel dont la valeur est calculée à partir des coefficients* $b_0, \ldots, b_l$ *contenus dans la liste de booléens* `E`*. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

1.3 Incrémentation en codage binaire

Nous allons étudier l'opération d'incrémentation d'un nombre entier naturel en codage binaire.

1.3.1 Logique des propositions

Nous allons définir l'opération d'incrémentation par des opérations logiques sur les booléens composant le codage binaire de l'entier naturel.

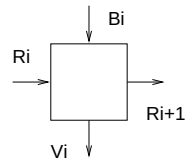
Question II.9 *Soit* $b_0, \ldots, b_l$ *le codage binaire d'un entier naturel* n*, calculer* $v_0, \ldots, v_m$ *le codage de l'entier naturel* $n+1$*. Utiliser pour cela la suite des retenues* $r_0, \ldots, r_m$ *du calcul de l'addition chiffre par chiffre. Exprimer* v_i *et* r_{i+1} *en fonction de* b_i *et* r_i *pour* $0 \leq i \leq l$*. Préciser les valeurs de* m*,* r_0 *et* v_m*.*

1.3.2 Réalisation par un circuit électronique

Nous allons réaliser l'opération d'incrémentation en utilisant des composants électroniques. Celle-ci repose sur une cellule élémentaire d'incrémentation qui est dupliquée $l+1$ fois.

Question II.10 *Proposer un circuit électronique composé de portes logiques qui réalise une cellule comportant deux entrées* b_i *et* r_i *et deux sorties* v_i *et* r_{i+1} *dont le comportement correspond à la réponse proposée à la question II.9.*

Cette cellule sera représentée graphiquement par :



Question II.11 *Proposer un circuit électronique composé de telles cellules pour incrémenter un nombre codé sur* 3 *bits.*

1.3.3 Réalisation par un programme

Nous allons programmer l'opération d'incrémentation d'un nombre entier naturel codé en binaire.

Question II.12 *Écrire en PASCAL une fonction* `incrementation(E:ENTIER):ENTIER;` *telle que l'appel* `incrementation(E)` *renvoie un entier naturel codé en binaire dont la valeur est égale à la valeur de l'entier naturel codé en binaire* `E` *augmentée de 1. L'algorithme utilisé ne devra parcourir qu'une seule fois la liste* `E`*. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

1.4 Addition en codage binaire

Nous allons étudier l'opération d'addition de deux nombres entiers naturels en codage binaire.

1.4.1 Logique des propositions

Nous allons définir l'opération d'addition par des opérations logiques sur les booléens composant le codage binaire des entiers naturels.

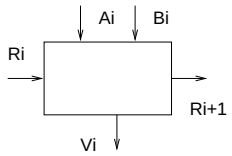
Question II.13 *Soient* $a_0, \ldots, a_m$ *le codage binaire d'un entier naturel* a *et* $b_0, \ldots, b_n$ *le codage binaire d'un entier naturel* b*, calculer* $v_0, \ldots, v_p$ *le codage de l'entier naturel* $a+b$*. Utiliser pour cela la suite des retenues* $r_0, \ldots, r_p$ *du calcul de l'addition chiffre par chiffre. Exprimer* v_i *et* r_{i+1} *en fonction de* a_i*,* b_i *et* r_i *pour* $0 \leq i \leq \min(m,n)$*. Préciser les valeurs de* p*,* r_0*,* v_i *et* r_{i+1} *pour* $\min(m,n) < i \leq \max(m,n)$ *et* v_p*.*

1.4.2 Réalisation par un circuit électronique

Nous allons réaliser l'opération d'addition en utilisant des composants électroniques. Celle-ci repose sur une cellule élémentaire d'addition qui est dupliquée $\min(m,n)+1$ fois ainsi que sur la cellule d'incrémentation.

Question II.14 *Proposer un circuit électronique composé de portes logiques comportant trois entrées* a_i*,* b_i *et* r_i *et deux sorties* v_i *et* r_{i+1} *dont le comportement correspond à la réponse proposée à la question II.13 pour* $0 \leq i \leq \min(m,n)$*.*

Cette cellule sera représentée graphiquement par :



Question II.15 *Proposer un circuit électronique composé de cellules d'addition et d'incrémentation pour additionner deux nombres codés sur 2 et 4 bits.*

1.4.3 Réalisation par un programme

Nous allons programmer l'opération d'addition de deux nombres entiers naturels codés en binaire.

Question II.16 *Écrire en PASCAL une fonction* `addition(E1:ENTIER;E2:ENTIER):ENTIER;` *telle que l'appel* `addition(E1,E2)` *renvoie un entier naturel codé en binaire dont la valeur est égale à la somme des valeurs des entiers naturels codés en binaire* `E1` *et* `E2`*. L'algorithme utilisé ne devra parcourir qu'une seule fois les listes* `E1` *et* `E2`*. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

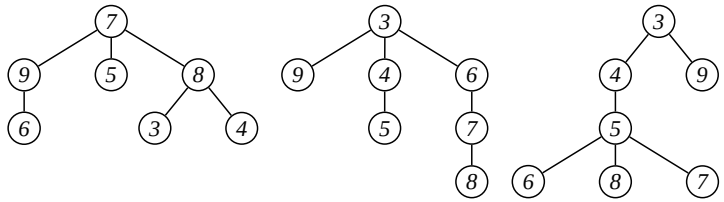
2 Arbres d'entiers en tas

Pour améliorer les performances en temps d'accès aux valeurs, un ensemble d'entiers peut être réalisé par un arbre en utilisant les étiquettes des nœuds pour représenter les valeurs contenues dans l'ensemble.

2.1 Arbres d'entiers

Déf. II.3 (Arbre d'entiers) *Un arbre d'entiers a est une structure qui peut soit être vide (notée $\emptyset$), soit être un nœud qui contient une étiquette entière (notée $\mathcal{E}(a)$) et des fils (notée $\mathcal{S}(a)$), une suite, éventuellement vide, de sous-arbres qui sont tous des arbres d'entiers non vides.*

Exemple II.1 (Arbres d'entiers) *Voici trois exemples d'arbres d'entiers :*



Question II.17 *Exprimer la définition II.3 sous la forme d'une propriété $A(a)$ qui est vraie si et seulement si a est un arbre d'entiers. $A(a)$ sera écrite en exploitant $\emptyset$ et $\mathcal{S}(a)$.*

2.2 Représentation des arbres d'entiers en PASCAL

Un arbre d'entiers est représenté par le type de base `ARBRE`. Une liste d'arbres d'entiers est représentée par le type de base `ARBRES`.

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 ou une liste vide ;
- `FUNCTION Noeud(v:INTEGER;S:ARBRES):ARBRE;` est une fonction qui renvoie un arbre dont l'étiquette et la liste des fils de la racine sont respectivement `v` et `S`,
- `FUNCTION Valeur(a:ARBRE):INTEGER;` est une fonction qui renvoie l'étiquette de la racine de l'arbre `a` ;
- `FUNCTION Fils(a:ARBRE):ARBRES;` est une fonction qui renvoie la liste des fils de la racine de l'arbre `a` ;
- `FUNCTION A_Insertion(a:ARBRE;L:ARBRES):ARBRES;` renvoie une liste d'arbres composée d'un premier arbre `a` et du reste de la liste contenu dans `L` ;
- `FUNCTION A_Premier(L:ARBRES):ARBRE;` renvoie le premier arbre de la liste `L`. Cette liste ne doit pas être vide ;
- `FUNCTION A_Reste(L:ARBRES):ARBRES;` renvoie le reste de la liste `L` privée de son premier arbre. Cette liste ne doit pas être vide.

Exemple II.2 *L'appel suivant*

```
Noeud( 7, A_Insertion(
   Noeud( 9, A_Insertion(
      Noeud( 6, Vide ), Vide ) ), A_Insertion(
   Noeud( 5, Vide ), A_Insertion(
   Noeud( 8, A_Insertion(
      Noeud( 3, Vide ), A_Insertion(
      Noeud( 4, Vide ), Vide ))), Vide ))))
```

renvoie le premier arbre d'entiers représenté graphiquement dans l'exemple II.1.

2.3 Taille d'un arbre

Déf. II.4 (Taille d'un arbre) *La taille d'un arbre d'entiers est le nombre de nœuds contenus dans l'arbre a. Nous la noterons $\mathcal{T}(a)$.*

Exemple II.3 (Tailles) *La taille des trois arbres de l'exemple II.1 est de 7.*

Question II.18 *Écrire en PASCAL une fonction* `taille(a:ARBRE):INTEGER;` *telle que l'appel* `taille(a)` *renvoie la taille de l'arbre d'entiers* `a`*. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

2.4 Hauteur d'un arbre

Déf. II.5 (Hauteur d'un arbre) *Les branches d'un arbre relient la racine aux feuilles. La hauteur d'un arbre a est égale au nombre d'arcs de la branche la plus longue. Nous la noterons $\mathcal{H}(a)$.*

Exemple II.4 (Hauteur d'un arbre) *Les hauteurs des trois arbres de l'exemple II.1 sont respectivement 2, 3 et 3.*

Question II.19 *Donner une définition de la hauteur d'un arbre a en fonction de $\emptyset$ et $\mathcal{S}(a)$.*

Question II.20 *Considérons un arbre d'entiers de taille n. Quelle est la forme de l'arbre dont la hauteur est maximale ? Quelle est la forme de l'arbre dont la hauteur est minimale ? Quelle est la hauteur de l'arbre en fonction de n dans ces deux cas ?*

Question II.21 *Écrire en PASCAL une fonction* `hauteur(a:ARBRE):INTEGER;` *telle que l'appel* `hauteur(a)` *renvoie la hauteur de l'arbre d'entiers* `a`*. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

2.5 Profondeur d'un nœud

Déf. II.6 (Profondeur d'un nœud) *La profondeur d'un nœud n dans un arbre a est égale au nombre d'arcs entre la racine et le nœud dans l'arbre. Nous la noterons $\mathcal{P}(n, a)$.*

Exemple II.5 (Profondeur d'un nœud) *Les profondeurs du nœud 7 dans les trois arbres de l'exemple II.1 sont respectivement 0, 2 et 3.*

2.6 Arbres d'entiers en tas

Déf. II.7 (Arbre d'entiers en tas) *Un arbre d'entiers est en tas si les valeurs contenues dans les fils de la racine sont toutes strictement supérieures à la valeur contenue dans la racine et si les fils de la racine sont en tas.*

Exemple II.6 (Arbres d'entiers en tas) *Le premier arbre de l'exemple II.1 n'est pas en tas. Par contre, les deuxième et troisième arbres du même exemple sont en tas.*

Question II.22 *Exprimer la définition précédente sous la forme d'une propriété $AT(a)$ qui est vraie si et seulement si a est un arbre d'entiers en tas. $AT(a)$ est écrite en exploitant $\emptyset$, $\mathcal{E}(a)$ et $\mathcal{S}(a)$.*

Question II.23 *Écrire en PASCAL une fonction* `validerAT(A:ARBRE):BOOLEAN;` *telle que l'appel* `validerAT(A)` *renvoie la valeur* `TRUE` *si la propriété $AT($*`A`*$)$ est vraie et la valeur* `FALSE` *sinon. L'algorithme utilisé ne devra parcourir qu'une seule fois l'arbre. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

3 Arbre binomial d'entiers en tas

3.1 Définitions

Déf. II.8 (Arbre binomial d'entiers en tas) *Un arbre binomial d'ordre k d'entiers en tas est un arbre non vide d'entiers en tas défini récursivement par :*

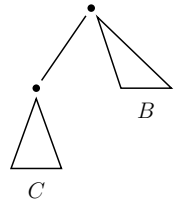
- *L'arbre d'ordre 0 ne contient qu'un seul nœud ;*
- *La racine d'un arbre d'ordre $k+1$ possède $k+1$ fils qui sont tous des arbres binomiaux d'entiers en tas dont les ordres sont strictement décroissants de gauche (ordre k) à droite (ordre 0).*

Question II.24 *Donner un exemple d'arbre binomial d'entiers en tas pour chacun des ordres 0, 1, 2 et 3.*

3.2 Construction d'un arbre binomial en tas

La structure d'arbre binomial en tas permet de composer deux arbres d'ordre k pour obtenir un arbre d'ordre $k+1$ avec une complexité $O(1)$.

Question II.25 *Montrer que tout arbre binomial A d'ordre $k+1$ d'entiers en tas est composé de deux arbres binomiaux B et C d'ordre k d'entiers en tas tels que l'arbre A est obtenu à partir de B en ajoutant C comme fils le plus à gauche : la racine de A, égale à la racine de B, a comme liste de fils, de gauche à droite, C suivi des fils de la racine de B.*



3.3 Propriétés d'un arbre binomial en tas

Question II.26 *Calculer la taille d'un arbre binomial d'ordre k.*

Question II.27 *Calculer la hauteur d'un arbre binomial d'ordre k.*

Question II.28 *Montrer que le nombre de nœuds de profondeur p dans un arbre binomial d'ordre k, avec $k \geq p$ est égal au coefficient du binôme* $\begin{pmatrix} p \\ k \end{pmatrix}$.

3.4 Validation d'un arbre binomial en tas

Question II.29 *Exprimer la définition équivalente proposée à la question II.25 sous la forme d'une propriété $ABT_k(a)$ qui indique que a est un arbre binomial en tas d'ordre k en exploitant $\emptyset$, $\mathcal{E}(a)$ et $\mathcal{S}(a)$.*

Question II.30 *Écrire en PASCAL une fonction* `validerABT(k:INTEGER;A:ARBRE):BOOLEAN;` *telle que l'appel* `validerABT(k,A)` *renvoie la valeur* `TRUE` *si la propriété $ABT_k($*`A`*$)$ est vraie et la valeur* `FALSE` *sinon. L'algorithme utilisé ne devra parcourir qu'une seule fois l'arbre. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

4 Tas binomial

La structure de tas binomial consiste à utiliser une suite d'arbres binomiaux en tas, soit vides, soit d'ordre strictement croissant, pour représenter un ensemble de taille quelconque.

4.1 Définitions

Déf. II.9 (Tas binomial) *Un tas binomial est une suite d'arbres binomiaux $a_0, \ldots, a_l$ tels que, soit a_i est vide, soit a_i est un arbre binomial d'ordre i. Si le tas n'est pas vide, alors $a_l \neq \emptyset$.*

Déf. II.10 (Signature d'un tas binomial) *Soit $a_0, \ldots, a_l$ un tas binomial, sa signature est une suite $s_0, \ldots, s_l$ telle que $s_i = 0$ si et seulement si a_i est vide et $s_i = 1$ sinon.*

Déf. II.11 (Taille d'un tas binomial) *La taille d'un tas binomial est la somme du nombre de nœuds contenus dans les arbres qui composent le tas t. Nous la noterons $\mathcal{T}(t)$.*

Question II.31 *Calculer la taille d'un tas binomial $a_0, \ldots, a_l$ à partir de sa signature.*

Question II.32 *Montrer que la signature d'un tas binomial ne dépend que de la taille du tas.*

4.2 Représentation des tas binomiaux en PASCAL

Un tas binomial est une liste d'arbres d'entiers. Il est donc représenté par le type de base `ARBRES` (voir 2.2).

4.3 Validation d'un tas binomial

En préliminaire, nous allons réaliser une opération de validation d'une liste d'arbres qui vérifie qu'il s'agit bien d'un tas binomial.

Question II.33 *Exprimer la définition II.9 sous la forme d'une propriété $TB(t)$ qui indique que t est un tas binomial.*

Question II.34 *Écrire en PASCAL une fonction* `validerTB(T:ARBRES):BOOLEAN;` *telle que l'appel* `validerTB(T)` *renvoie la valeur* `TRUE` *si la propriété $TB(T)$ est vraie et la valeur* `FALSE` *sinon. L'algorithme utilisé ne devra parcourir qu'une seule fois le tas. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

4.4 Ajout d'une valeur dans un tas binomial

La première opération consiste à insérer une valeur dans un tas binomial en préservant sa structure.

Nous faisons l'hypothèse que le tas binomial ne contient pas déjà cette valeur.

Question II.35 *Montrer que la signature du tas résultant de l'ajout d'une valeur à un tas est égale au résultat de l'incrémentation binaire de la signature de tas initial. En déduire un algorithme pour l'ajout d'une valeur dans un tas binomial.*

4.5 Fusion de tas binomiaux

La deuxième opération consiste à fusionner deux tas binomiaux, c'est-à-dire construire un tas binomial qui contient les mêmes valeurs que les deux tas binomiaux que l'on souhaite fusionner.

Nous faisons l'hypothèse que les deux tas binomiaux sont disjoints, c'est-à-dire qu'ils ne contiennent pas les mêmes éléments.

Question II.36 *Montrer que la signature du tas résultant de la fusion est égale au résultat de l'addition binaire des signatures des deux tas initiaux. En déduire un algorithme pour la fusion de deux tas binomiaux.*

Fin de l'énoncé