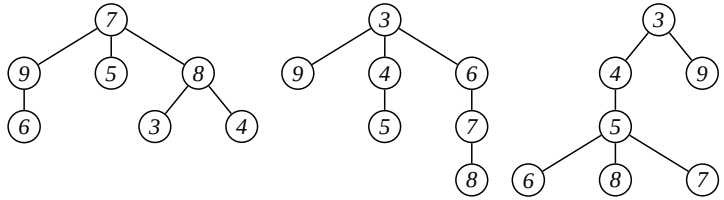
2.1 Arbres d'entiers

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 ;*

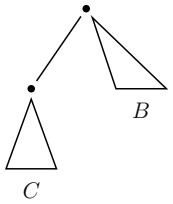
– *La racine d'un arbre d'ordre $k+1$ possède $k+1$ fils qui sont tous des arbres binomiaux d'entiers en tas dont les ordres sont strictement décroissants de gauche (ordre k) à droite (ordre 0).*

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.*