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 d'entrelacement || de deux automates. Cette opération permet de composer deux automates qui doivent lire la même entrée. Elle est utilisée pour la modélisation d'activités concurrentes qui partagent une même ressource.

1 Automate fini

Pour simplifier les preuves, nous nous limiterons au cas des automates finis semi-indéterministes, c'est-à-dire les automates finis non déterministes qui ne contiennent pas de transitions instantanées (ou ϵ-transitions). Les résultats étudiés s'étendent au cadre des automates finis quelconques.

1.1 Représentation d'un automate fini

Déf. I.1 (Automate fini semi-indéterministe) *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 semi-indéterministe sur X est un quintupiet $A = (Q, X, I, T, \gamma)$ composé de :*

- *Un ensemble fini d'états : Q ;*
- *Un ensemble d'états initiaux : $I \subseteq Q$;*
- *Un ensemble d'états terminaux : $T \subseteq Q$;*
- *Une relation de transition confondue avec son graphe : $\gamma \subseteq Q \times X \times Q$.*

Pour une transition (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.

Remarquons que γ est le graphe d'une application de transition $\delta : Q \times X \to \mathcal{P}(Q)$ dont les valeurs sont définies par :

$$\forall o \in Q,\ \forall e \in X,\ \delta(o, e) = \{d \in Q \mid (o, e, d) \in \gamma\}$$

La notation γ est plus adaptée que δ à la formalisation et la construction des preuves dans le cadre des automates indéterministes.

1.2 Représentation graphique d'un automate

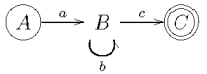
Les automates peuvent être représentés par un schéma suivant les conventions :

- les valeurs de la relation de transition γ 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 $(o, e, d) \in \gamma$.

Exemple I.1 *L'automate $\mathcal{E}_1 = (Q_1, X, I_1, T_1, \gamma_1)$ avec :*

$$\begin{aligned} Q_1 &= \{A, B, C\} \\ X &= \{a, b, c, d, e\} \\ I_1 &= \{A\} \\ T_1 &= \{C\} \\ \gamma_1 &= \{(A, a, B), (B, b, B), (B, c, C)\} \end{aligned}$$

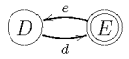
est représenté par le graphe suivant :



Exemple I.2 *L'automate $\mathcal{E}_2 = (Q_2, X, I_2, T_2, \gamma_2)$ avec :*

$$\begin{aligned} Q_2 &= \{D, E\} \\ X &= \{a, b, c, d, e\} \\ I_2 &= \{D\} \\ T_2 &= \{E\} \\ \gamma_2 &= \{(D, d, E), (E, e, D)\} \end{aligned}$$

est représenté par le graphe suivant :



1.3 Langage reconnu par un automate fini

Soit $\gamma^\star$ l'extension de γ à $Q \times X^\star \times Q$ définie par :

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

$$\begin{cases} \forall e \in X, \\ \forall m \in X^\star, \\ \forall o \in Q, \\ \forall d \in Q, \end{cases} \quad (o, e.m, d) \in \gamma^\star \Leftrightarrow \exists q \in Q,\ ((o, e, q) \in \gamma) \wedge ((q, m, d) \in \gamma^\star)$$

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

$$L(A) = \{m \in X^\star \mid \exists o \in I,\ \exists d \in T,\ (o, m, d) \in \gamma^\star\}$$

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

2 Entrelacement de mots et de langages

Soit l'opération sur les mots définie par :

Déf. I.2 (Entrelacement de mots) *Soient m_1 et m_2 deux mots de $X^\star$, le langage $m_1 \mid\mid m_2$ résultant de l'entrelacement de m_1 et m_2 est défini par :*

$$\forall m \in X^\star,\ \Lambda \mid\mid m = \{m\}$$
$$\forall m \in X^\star,\ m \mid\mid \Lambda = \{m\}$$
$$\begin{cases} \forall x_1 \in X, \\ \forall m_1 \in X^\star, \\ \forall x_2 \in X, \\ \forall m_2 \in X^\star, \end{cases} (x_1.m_1) \mid\mid (x_2.m_2) = \{x_1.m \mid m \in m_1 \mid\mid (x_2.m_2)\} \cup \{x_2.m \mid m \in (x_1.m_1) \mid\mid m_2\}$$

Question I.2 *Soit l'alphabet $X = \{a, b, c, d\}$, construire explicitement le langage $(a.b) \mid\mid (c.d)$.*

3 Entrelacement de langages

Soit l'opération interne sur les langages définie par :

Déf. I.3 (Entrelacement de langages) *Soient L_1 et L_2 deux langages sur X, le langage $L_1 \mid\mid L_2$ résultant de l'entrelacement des mots de L_1 et L_2 est défini par :*

$$L_1 \mid\mid L_2 = \bigcup_{m_1 \in L_1, m_2 \in L_2} m_1 \mid\mid m_2$$

4 Entrelacement d'automates

4.1 Définition

Soit l'opération interne sur les automates finis semi-indéterministes définie par :

Déf. I.4 (Entrelacement d'automates) *Soient $\mathcal{A}_1 = (Q_1, X, I_1, T_1, \gamma_1)$ et $\mathcal{A}_2 = (Q_2, X, I_2, T_2, \gamma_2)$ deux automates finis semi-indéterministes, l'automate $\mathcal{A}$ qui résulte de l'entrelacement de $\mathcal{A}_1$ et $\mathcal{A}_2$ est défini par :*

$$\mathcal{A} = (Q_1 \times Q_2, X, I_1 \times I_2, T_1 \times T_2, \gamma_{1||2})$$

$$\begin{cases} \forall o_1 \in Q_1, \\ \forall o_2 \in Q_2, \\ \forall x \in X, \\ \forall d_1 \in Q_1, \\ \forall d_2 \in Q_2, \end{cases} ((o_1, o_2), x, (d_1, d_2)) \in \gamma_{1||2} \Leftrightarrow ((o_1, x, d_1) \in \gamma_1 \wedge o_2 = d_2) \vee (o_1 = d_1 \wedge (o_2, x, d_2) \in \gamma_2)$$

Question I.3 *En considérant les exemples I.1 et I.2, construire l'automate $\mathcal{E}_1 \mid\mid \mathcal{E}_2$.*

Question I.4 *Caractériser le langage reconnu par $\mathcal{E}_1 \mid\mid \mathcal{E}_2$, par une expression régulière ou ensembliste.*

4.2 Propriétés

Question I.5 *Montrer que : si $\mathcal{A}_1$ et $\mathcal{A}_2$ sont des automates finis semi-indéterministes alors $\mathcal{A}_1 \mid\mid \mathcal{A}_2$ est un automate fini semi-indéterministe.*

Question I.6 *Montrer que :*

$$\forall m_1 \in X^\star, \forall m_2 \in X^\star, \Lambda \in m_1 \mid\mid m_2 \Leftrightarrow ((m_1 = \Lambda) \wedge (m_2 = \Lambda))$$

Question I.7 *Montrer que :*

$$\begin{cases} \forall x \in X, \\ \forall m \in X^\star, \\ \forall m_1 \in X^\star, \\ \forall m_2 \in X^\star, \end{cases} x.m \in m_1 \mid\mid m_2 \Leftrightarrow ((m_1 = x.m_1') \wedge (m \in m_1' \mid\mid m_2)) \vee ((m_2 = x.m_2') \wedge (m \in m_1 \mid\mid m_2'))$$

Question I.8 *Montrer que :*

$$\begin{cases} \forall m \in X^\star, \\ \forall m_1 \in X^\star, \\ \forall m_2 \in X^\star, \\ \forall o_1 \in Q_1, \\ \forall o_2 \in Q_2, \\ \forall d_1 \in Q_1, \\ \forall d_2 \in Q_2, \end{cases} ((o_1, o_2), m, (d_1, d_2)) \in \gamma_{1||2}^\star \Leftrightarrow \begin{cases} m \in m_1 \mid\mid m_2, \\ (o_1, m_1, d_1) \in \gamma_1^\star, \\ (o_2, m_2, d_2) \in \gamma_2^\star \end{cases}$$

Question I.9 *Soient $\mathcal{A}_1$ et $\mathcal{A}_2$ des automates finis semi-indéterministes, montrer que :*

$$L(\mathcal{A}_1 \mid\mid \mathcal{A}_2) = L(\mathcal{A}_1) \mid\mid L(\mathcal{A}_2)$$

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.

Format de description d'une fonction : La description d'une fonction, lorsque celle-ci est demandée, c'est-à-dire à la question II.25, doit au moins contenir :

1. L'objectif général ;
2. Le rôle des paramètres de la fonction ;
3. Les contraintes sur les valeurs des paramètres ;
4. Les caractéristiques du résultat renvoyé par la fonction ;
5. Le rôle des variables locales à la fonction ;
6. Le principe de l'algorithme ;
7. Des arguments de terminaison du calcul pour toutes les valeurs des paramètres qui vérifient les contraintes présentées au point 3.

La structure d'ensemble d'entiers est utilisée dans de nombreux algorithmes (par exemple, pour la déterminisation ou la minimisation d'automates). L'objectif de ce problème est l'étude d'une réalisation particulière de cette structure à base d'arbres binaires de recherche équilibrés nommés arbres AVL (du nom de leurs créateurs Adelson-Velskii et Landis). L'objectif de cette réalisation est d'optimiser à la fois l'occupation mémoire et la durée de recherche d'un élément dans un ensemble.

1 Réalisation à base de listes

La réalisation la plus simple d'un ensemble d'entiers repose sur l'utilisation d'une liste d'entiers qui contient exactement un exemplaire de chaque élément contenu dans l'ensemble.

1.1 Représentation en CaML

Nous ferons l'hypothèse qu'un élément appartenant à un ensemble n'apparaît qu'une seule fois dans la liste représentant cet ensemble.

Un ensemble d'entiers est représenté par le type `ensemble` équivalent à une liste de `int`.

```
type ensemble == int list;;
```

Le parcours d'un ensemble sera donc effectué de la même manière que celui d'une liste.

Nous allons maintenant définir plusieurs opérations sur les ensembles d'entiers et estimer leur complexité.

1.2 Opérations sur la structure d'ensemble

1.2.1 Insertion dans un ensemble

La première opération est l'insertion d'un entier à un ensemble.

Question II.1 *Écrire en CaML une fonction* `inserer_ens` *de type* `int -> ensemble -> ensemble` *telle que l'appel* `(inserer_ens v E)` *renvoie un ensemble contenant les mêmes éléments que l'ensemble* `E` *ainsi que l'élément* `v` *s'il ne figurait pas déjà dans* `E`*. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

Question II.2 *Calculer une estimation de la complexité de la fonction* `inserer_ens` *en fonction du nombre d'éléments de l'ensemble* `E`*. Cette estimation ne prendra en compte que le nombre d'appels récursifs effectués.*

1.2.2 Recherche dans un ensemble

La seconde opération consiste à rechercher un entier dans un ensemble.

Question II.3 *Écrire en CaML une fonction* `rechercher_ens` *de type* `int -> ensemble -> bool` *telle que l'appel* `(rechercher_ens v E)` *renvoie la valeur* `true` *si l'ensemble* `E` *contient l'entier* `v` *et la valeur* `false` *sinon. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

Question II.4 *Donner des exemples de valeurs des paramètres* `v` *et* `E` *de la fonction* `rechercher_ens` *qui correspondent au meilleur et pire cas en nombre d'appels récursifs effectués.*

Calculer une estimation de la complexité dans les meilleur et pire cas de la fonction `rechercher_ens` *en fonction du nombre d'éléments de l'ensemble* `E`*. Cette estimation ne prendra en compte que le nombre d'appels récursifs effectués.*

2 Réalisation à base d'arbres binaires de recherche

L'utilisation de la structure d'arbre binaire de recherche permet de réduire la complexité en temps de calcul pour les opérations d'insertion et de recherche.

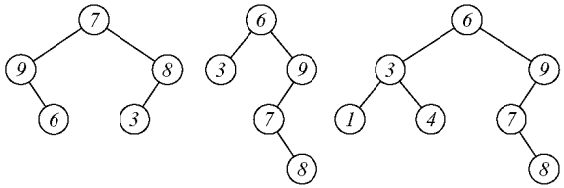
2.1 Arbres binaires d'entiers

Un ensemble d'entiers peut être réalisé par un arbre binaire en utilisant les étiquettes des nœuds pour représenter les éléments contenus dans l'ensemble.

2.1.1 Définition

Déf. II.1 (Arbre binaire d'entiers) *Un arbre binaire d'entiers a est une structure qui peut soit être vide (notée $\emptyset$), soit être un nœud qui contient une étiquette entière (notée $\mathcal{E}(a)$), un sous-arbre gauche (noté $\mathcal{G}(a)$) et un sous-arbre droit (noté $\mathcal{D}(a)$) qui sont tous deux des arbres binaires d'entiers. L'ensemble des étiquettes de tous les nœuds de l'arbre a est noté $\mathcal{C}(a)$.*

Exemple II.1 (Arbres binaires d'entiers) *Voici trois exemples d'arbres binaires étiquetés par des entiers (les sous-arbres vides des nœuds ne sont pas représentés) :*



2.1.2 Profondeur d'un arbre

Déf. II.2 (Profondeur d'un arbre) *Les branches d'un arbre relient la racine aux feuilles. La profondeur d'un arbre A est égale au nombre de nœuds de la branche la plus longue. Nous la noterons $\mid A \mid$.*

Exemple II.2 (Profondeurs) *Les profondeurs des trois arbres binaires de l'exemple II.1 sont respectivement 3, 4 et 4.*

Question II.5 *Donner une définition de la profondeur d'un arbre a en fonction de $\emptyset$, $\mathcal{G}(a)$ et $\mathcal{D}(a)$.*

Question II.6 *Considérons un arbre binaire d'entiers représentant un ensemble contenant n éléments. Quelle est la forme de l'arbre dont la profondeur est maximale ? Quelle est la forme de l'arbre dont la profondeur est minimale ? Calculer la profondeur de l'arbre en fonction de n dans ces deux cas. Vous justifierez vos réponses.*

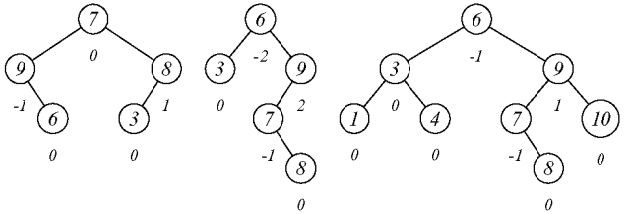
2.1.3 Déséquilibre d'un arbre

Déf. II.3 (Déséquilibre d'un nœud dans un arbre binaire) *Le déséquilibre $\Delta(a)$ d'un nœud a dans un arbre binaire est égal à la différence entre la profondeur de son fils gauche et la profondeur de son fils droit, c'est-à-dire :*

$$\Delta(\emptyset) = 0$$
$$a \neq \emptyset \Rightarrow \Delta(a) = \mid \mathcal{G}(a) \mid - \mid \mathcal{D}(a) \mid$$

Nous considérerons par la suite des arbres binaires dont chaque nœud est décoré par son déséquilibre.

Exemple II.3 (Arbres binaires d'entiers décorés) *Voici les arbres de l'exemple II.1 décorés par le déséquilibre de chaque nœud :*



2.1.4 Représentation des arbres binaires en CaML

Un arbre binaire d'entiers dont les nœuds sont décorés par leurs déséquilibres est représenté par le type CaML :

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

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

Exemple II.4 *Le terme suivant*

```
Noeud( 0,
   Noeud( -1,
      Vide,
      9,
      Noeud( 0, Vide, 6, Vide)),
   7,
   Noeud( 1,
      Noeud( 0, Vide, 3, Vide),
      8,
      Vide))
```

est alors associé au premier arbre binaire représenté graphiquement dans l'exemple II.3.

2.1.5 Profondeur d'un arbre binaire d'entiers

Question II.7 *Écrire en CaML une fonction `profondeur` de type `arbre -> int` telle que l'appel `(profondeur a)` renvoie la profondeur de l'arbre binaire d'entiers `a`. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

2.1.6 Validation d'un arbre binaire d'entiers décoré

Une première opération consiste à déterminer si les valeurs des déséquilibres des nœuds d'un arbre binaire sont correctes.

Question II.8 *Écrire en CaML une fonction `valider_decore` de type `arbre -> bool` telle que l'appel `(valider_decore A)` renvoie la valeur `true` si l'arbre binaire `A` est vide ou si l'arbre binaire `A` n'est pas vide et si les valeurs des déséquilibres des nœuds de l'arbre binaire `A` sont correctes 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.*

Question II.9 *Calculer une estimation de la complexité de la fonction `valider_decore` en fonction du nombre de nœuds de l'arbre `A`. Cette estimation ne prendra en compte que le nombre d'appels récursifs effectués.*

2.2 Arbres binaires de recherche

Déf. II.4 (Arbre binaire de recherche) *Un arbre binaire de recherche est un arbre binaire d'entiers dont les étiquettes de tous les nœuds composant le fils gauche de la racine sont strictement inférieures à l'étiquette de la racine et les étiquettes de tous les nœuds composant le fils droit de la racine sont strictement supérieures à l'étiquette de la racine. Cette contrainte s'exprime sous la forme :*

$$a \neq \emptyset \Rightarrow (\forall v \in \mathcal{C}(\mathcal{G}(a)), v < \mathcal{E}(a)) \wedge (\forall v \in \mathcal{C}(\mathcal{D}(a)), \mathcal{E}(a) < v)$$

Exemple II.5 (Arbres binaires de recherche) *Le deuxième et le troisième arbre de l'exemple II.3 sont des arbres binaires de recherche.*

Notons qu'un arbre binaire de recherche ne peut contenir qu'un seul exemplaire d'une valeur donnée.

Nous considérerons par la suite des arbres de recherche dont chaque nœud est décoré par son déséquilibre.

2.2.1 Validation d'un arbre binaire de recherche

Une première opération consiste à déterminer si un arbre binaire d'entiers est un arbre binaire de recherche.

Question II.10 *Écrire en CaML une fonction* `valider_abr` *de type* `arbre -> bool` *telle que l'appel* `(valider_abr A)` *renvoie la valeur* `true` *si l'arbre binaire d'entiers* `A` *est un arbre binaire de recherche 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.*

2.2.2 Recherche dans un arbre binaire de recherche

Une deuxième opération consiste à déterminer si un arbre binaire contient une étiquette particulière.

Question II.11 *Écrire en CaML une fonction* `rechercher_abr` *de type* `int -> arbre -> bool` *telle que l'appel* `(rechercher_abr v A)` *renvoie la valeur* `true` *si l'un des nœuds de l'arbre binaire de recherche* `A` *contient l'étiquette* `v` *et la valeur* `false` *sinon. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

Question II.12 *Donner des exemples de valeurs des paramètres* `v` *et* `A` *de la fonction* `rechercher_abr` *qui correspondent au meilleur et pire cas en nombre d'appels récursifs effectués.*

Calculer une estimation de la complexité dans les meilleur et pire cas de la fonction `rechercher_abr` *en fonction de la profondeur de l'arbre* `A`*. Cette estimation ne prendra en compte que le nombre d'appels récursifs effectués.*

2.2.3 Insertion dans un arbre binaire de recherche

Une deuxième opération consiste à insérer un élément dans un arbre binaire de recherche en préservant cette structure. Pour cela, l'insertion se fait au niveau des feuilles de l'arbre (ou des nœuds ne contenant qu'un seul fils en lieu et place du fils absent).

Question II.13 *Calculer $\delta(v, a)$ l'accroissement de la profondeur d'un arbre binaire de recherche a dans lequel l'étiquette v est insérée en fonction de $\emptyset$, de la relation entre v et $\mathcal{E}(a)$, de $\Delta(a)$, de $\delta(v, \mathcal{G}(a))$ ou de $\delta(v, \mathcal{D}(a))$.*

Question II.14 *Écrire en CaML une fonction* `inserer_abr` *de type* `int -> arbre -> arbre` *telle que l'appel* `(inserer_abr v A)` *renvoie un arbre binaire de recherche contenant les mêmes étiquettes que l'arbre binaire de recherche* `A` *ainsi que l'étiquette* `v` *s'il ne la contenait pas déjà. Cette fonction doit également calculer les valeurs des déséquilibres de chaque nœud de l'arbre résultat. 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.*

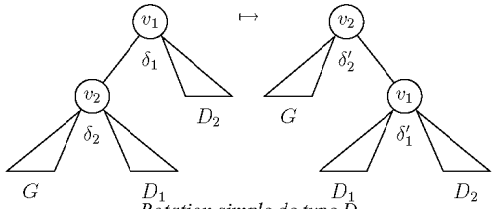
Question II.15 *Donner une estimation de la complexité dans les meilleur et pire cas de la fonction* `inserer_abr` *en fonction de la profondeur de l'arbre* `A`*. Cette estimation ne prendra en compte que le nombre d'appels récursifs effectués.*

2.3 Rotations d'un arbre binaire de recherche

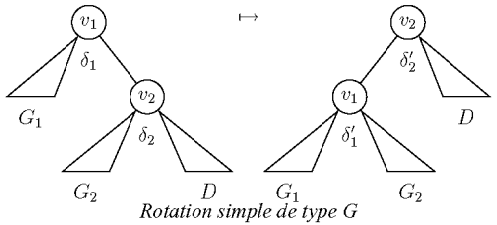
Les rotations sont des transformations d'un arbre binaire de recherche qui sont utilisées pour équilibrer la structure de l'arbre, c'est-à-dire donner une valeur proche à la longueur de chaque branche. Il s'agit donc de réduire le déséquilibre entre les différentes branches qui sont de longueurs différentes.

Déf. II.5 *Rotation simple*

Une rotation simple concerne 2 nœuds imbriqués de l'arbre. Un arbre qui ne possède pas la structure adéquate (partie gauche de la règle) ne sera pas transformé.



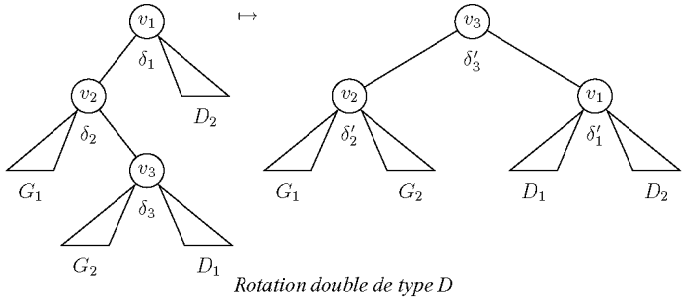
Rotation simple de type D



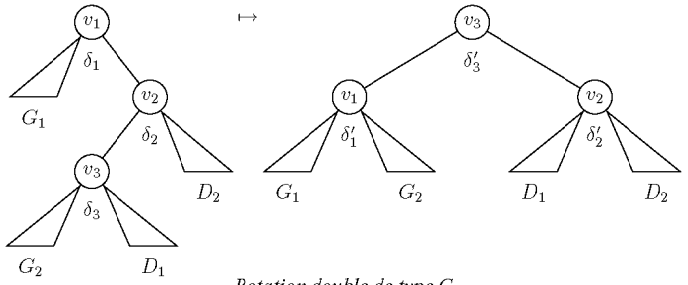
Rotation simple de type G

Déf. II.6 *Rotation double*

Une rotation double concerne 3 nœuds imbriqués de l'arbre. Un arbre qui ne possède pas la structure adéquate (partie gauche de la règle) ne sera pas transformé.



Rotation double de type D



Rotation double de type G

Question II.16 *Exprimer les rotations doubles comme des combinaisons de rotations simples.*

Question II.17 *Montrer que les rotations ne perturbent pas la structure d'arbre binaire de recherche.*

Question II.18 *Pour chaque rotation possible, calculer les nouvelles valeurs δ'_i des déséquilibres des 2 ou 3 nœuds concernés de l'arbre après la rotation en fonction des valeurs δ_i avant la rotation.*

2.3.1 Rotation simple de type D

Une première opération consiste à effectuer une rotation simple de type D dans un arbre binaire de recherche pour équilibrer celui-ci.

Question II.19 *Écrire en CaML une fonction* `rotationSD` *de type* `arbre -> arbre` *telle que l'appel* `(rotationSD A)` *sur un arbre* `A` *dont la structure permet l'application d'une rotation simple de type D sur la racine renvoie l'arbre binaire de recherche* `A` *sur lequel une rotation simple de type D a été appliquée à la racine.*

2.3.2 Rotation simple de type G

Une deuxième opération consiste à effectuer une rotation simple de type G dans un arbre binaire de recherche pour équilibrer celui-ci.

Nous supposons prédéfinie la fonction `rotationSG` de type `arbre -> arbre` telle que l'appel `(rotationSG A)` sur un arbre `A` dont la structure permet l'application d'une rotation simple de type G à la racine renvoie l'arbre binaire de recherche `A` sur lequel une rotation simple de type G a été appliquée à la racine. Son calcul se termine quelles que soient les valeurs de son paramètre. Elle pourra éventuellement être utilisée dans les réponses aux questions.

2.3.3 Rotation double de type D

Une troisième opération consiste à effectuer une rotation double de type D dans un arbre binaire de recherche pour équilibrer celui-ci.

Question II.20 *Écrire en CaML une fonction* `rotationDD` *de type* `arbre -> arbre` *telle que l'appel* `(rotationDD A)` *sur un arbre* `A` *dont la structure permet l'application d'une rotation double de type D à la racine renvoie l'arbre binaire de recherche* `A` *sur lequel une rotation double de type D a été appliquée à la racine.*

2.3.4 Rotation double de type G

Une quatrième opération consiste à effectuer une rotation double de type G dans un arbre binaire de recherche pour équilibrer celui-ci.

Nous supposons prédéfinie la fonction `rotationDG` de type `arbre -> arbre` telle que l'appel `(rotationDG A)` sur un arbre `A` dont la structure permet l'application d'une rotation double de type G à la racine renvoie l'arbre binaire de recherche `A` sur lequel une rotation double de type G a été appliquée à la racine. Son calcul se termine quelles que soient les valeurs de son paramètre. Elle pourra éventuellement être utilisée dans les réponses aux questions.

3 Arbres équilibrés

Pour réduire la complexité des opérations d'insertion et de recherche, il est nécessaire que les longueurs de toutes les branches de l'arbre aient des valeurs semblables. L'arbre a alors une structure dite équilibrée.

3.1 Définitions

Déf. II.7 (Arbres binaires équilibrés) *Un arbre binaire est dit équilibré si le déséquilibre de chacun de ses nœuds appartient à* $\{-1, 0, 1\}$. *Nous appellerons nœuds déséquilibrés les nœuds dont le déséquilibre n'appartient pas à* $\{-1, 0, 1\}$.

Exemple II.6 (Arbres binaires équilibrés) *Le premier et le troisième arbres de l'exemple II.3 sont équilibrés. Le deuxième arbre est déséquilibré. Il possède 2 nœuds déséquilibrés qui contiennent les étiquettes* 6 *et* 9.

3.2 Principe de l'équilibrage

Question II.21 *L'insertion d'une nouvelle étiquette dans un arbre binaire de recherche équilibré peut produire des nœuds déséquilibrés. Donner les différentes formes possibles pour le nœud déséquilibré le plus profond produit lors de l'insertion d'une nouvelle étiquette dans un arbre binaire équilibré en précisant la valeur du déséquilibre de ce nœud.*

Question II.22 *Montrer qu'une rotation suffit à rééquilibrer le nœud déséquilibré le plus profond qui est apparu lors de l'insertion d'une nouvelle étiquette dans un arbre binaire équilibré. Donner la rotation qui doit être utilisée pour rééquilibrer ce nœud pour chacune des formes étudiées dans la question précédente.*

3.3 Équilibrage d'un nœud

Une première opération consiste à équilibrer les nœuds qui viennent d'être déséquilibrés par l'opération d'insertion.

Question II.23 *Écrire en CaML une fonction* `equilibrage` *de type* `arbre -> arbre` *telle que l'appel* `(equilibrage A)` *avec* `A` *un arbre binaire de recherche dont la racine est déséquilibrée et dont les fils gauche et droit sont équilibrés renvoie un arbre binaire de recherche équilibré contenant les mêmes étiquettes que* `A`.

3.4 Insertion équilibrée

Une seconde opération consiste à insérer une nouvelle étiquette dans un arbre de recherche équilibré en préservant la structure équilibrée de l'arbre.

Question II.24 *Modifier la fonction* `inserer_abr` *de type* `int -> arbre -> arbre` *écrite en CaML à la question II.14 telle que si* `A` *est un arbre binaire de recherche équilibré alors l'appel* `(inserer_abr v A)` *renverra un arbre binaire de recherche équilibré.*

Question II.25 *Expliquer la fonction* `inserer_abr` *définie à la question précédente.*

3.5 Complexité des opérations dans un arbre équilibré

La complexité calculée aux questions II.15 et II.12 pour les opérations d'insertion et de recherche dépend de la profondeur de l'arbre. Pour comparer celle-ci avec celle d'une réalisation à base de liste, il faut estimer la profondeur de l'arbre en fonction du nombre d'éléments contenus dans l'ensemble. Nous avons calculé celle-ci dans le meilleur des cas pour un arbre équilibré à la question II.6. Il faut maintenant évaluer celle-ci dans le pire des cas.

Question II.26 *Quelles sont les formes de l'arbre équilibré a dans le meilleur cas et le pire cas pour l'insertion et la recherche d'une étiquette v dans a ?*

Question II.27 *Soit $\mathcal{N}(h)$ le nombre de nœuds dans un arbre dans le meilleur des cas, exprimer $\mathcal{N}(h)$ en fonction de h.*

Question II.28 *Soit $\mathcal{N}(h)$ le nombre de nœuds dans un arbre dans le pire des cas, exprimer $\mathcal{N}(h)$ sous la forme d'une relation de récurrence dépendant de $h-1$ et $h-2$.*

Question II.29 *Soit $\mathcal{F}(h) = \mathcal{N}(h) - 1$, résoudre l'équation linéaire à coefficients constants dérivée de la relation de récurrence précédente pour obtenir la valeur de $\mathcal{F}(h)$ en fonction de h.*

Question II.30 *Déduire des questions II.27 et II.29 un encadrement de la profondeur d'un arbre en fonction du nombre de nœuds qu'il contient. En déduire la complexité des opérations de recherche et d'insertion dans un arbre binaire de recherche équilibré.*

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`, ...).

Format de description d'une fonction : La description d'une fonction, lorsque celle-ci est demandée, c'est-à-dire à la question II.25, doit au moins contenir :

1. L'objectif général ;
2. Le rôle des paramètres de la fonction ;
3. Les contraintes sur les valeurs des paramètres ;
4. Les caractéristiques du résultat renvoyé par la fonction ;
5. Le rôle des variables locales à la fonction ;
6. Le principe de l'algorithme ;
7. Des arguments de terminaison du calcul pour toutes les valeurs des paramètres qui vérifient les contraintes présentées au point 3.

La structure d'ensemble d'entiers est utilisée dans de nombreux algorithmes (par exemple, pour la déterminisation ou la minimisation d'automates). L'objectif de ce problème est l'étude d'une réalisation particulière de cette structure à base d'arbres binaires de recherche équilibrés nommés arbres AVL (du nom de leurs créateurs Adelson-Velskii et Landis). L'objectif de cette réalisation est d'optimiser à la fois l'occupation mémoire et la durée de recherche d'un élément dans un ensemble.

1 Réalisation à base de listes

La réalisation la plus simple d'un ensemble d'entiers repose sur l'utilisation d'une liste d'entiers qui contient exactement un exemplaire de chaque élément contenu dans l'ensemble.

1.1 Représentation en PASCAL

Nous ferons l'hypothèse qu'un élément appartenant à un ensemble n'apparaît qu'une seule fois dans la liste représentant cet ensemble.

Un ensemble d'entiers est représenté par le type de base `ENSEMBLE` correspondant à une liste d'entiers.

Le parcours d'un ensemble 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 d'entiers ;
- `FUNCTION LE_Inserer(V:INTEGER;E:ENSEMBLE):ENSEMBLE;` renvoie une liste d'entiers composée d'un premier entier `V` et du reste de la liste contenu dans `E` ;

- `FUNCTION LE_Premier(E:ENSEMBLE):INTEGER;` renvoie le premier entier de la liste `E`. Cette liste ne doit pas être vide ;
- `FUNCTION LE_Reste(E:ENSEMBLE):ENSEMBLE;` renvoie le reste de la liste `E` privée de son premier entier. Cette liste ne doit pas être vide.

Nous allons maintenant définir plusieurs opérations sur les ensembles d'entiers et estimer leur complexité.

1.2 Opérations sur la structure d'ensemble

1.2.1 Insertion dans un ensemble

La première opération est l'insertion d'un entier à un ensemble.

Question II.1 *Écrire en PASCAL une fonction* `inserer_ens(v:INTEGER;E:ENSEMBLE):ENSEMBLE;` *telle que l'appel* `inserer_ens(v,E)` *renvoie un ensemble contenant les mêmes entiers que l'ensemble* `E` *ainsi que l'entier* `v` *s'il ne figurait pas déjà dans* `E`*. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

Question II.2 *Calculer une estimation de la complexité de la fonction* `inserer_ens` *en fonction du nombre d'éléments de l'ensemble* `E`*. Cette estimation ne prendra en compte que le nombre d'appels récursifs effectués.*

1.2.2 Recherche dans un ensemble

La seconde opération consiste à rechercher un entier dans un ensemble.

Question II.3 *Écrire en PASCAL une fonction* `rechercher_ens(v:INTEGER;E:ENSEMBLE):BOOLEAN;` *telle que l'appel* `rechercher_ens(v,E)` *renvoie la valeur* `TRUE` *si l'ensemble* `E` *contient l'entier* `v` *et la valeur* `FALSE` *sinon. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

Question II.4 *Donner des exemples de valeurs des paramètres* `v` *et* `E` *de la fonction* `rechercher_ens` *qui correspondent au meilleur et pire cas en nombre d'appels récursifs effectués.*

Calculer une estimation de la complexité dans les meilleur et pire cas de la fonction `rechercher_ens` *en fonction du nombre d'éléments de l'ensemble* `E`*. Cette estimation ne prendra en compte que le nombre d'appels récursifs effectués.*

2 Réalisation à base d'arbres binaires de recherche

L'utilisation de la structure d'arbre binaire de recherche permet de réduire la complexité en temps de calcul pour les opérations d'insertion et de recherche.

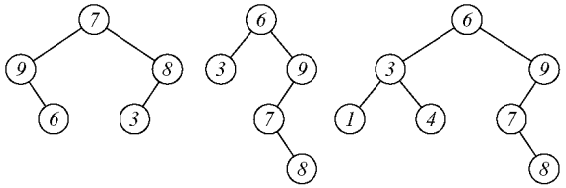
2.1 Arbres binaires d'entiers

Un ensemble d'entiers peut être réalisé par un arbre binaire en utilisant les étiquettes des nœuds pour représenter les éléments contenus dans l'ensemble.

2.1.1 Définition

Déf. II.1 (Arbre binaire d'entiers) *Un arbre binaire d'entiers a est une structure qui peut soit être vide (notée $\emptyset$), soit être un nœud qui contient une étiquette entière (notée $\mathcal{E}(a)$), un sous-arbre gauche (noté $\mathcal{G}(a)$) et un sous-arbre droit (noté $\mathcal{D}(a)$) qui sont tous deux des arbres binaires d'entiers. L'ensemble des étiquettes de tous les nœuds de l'arbre a est noté $\mathcal{C}(a)$.*

Exemple II.1 (Arbres binaires d'entiers) *Voici trois exemples d'arbres binaires étiquetés par des entiers (les sous-arbres vides des nœuds ne sont pas représentés) :*



2.1.2 Profondeur d'un arbre

Déf. II.2 (Profondeur d'un arbre) *Les branches d'un arbre relient la racine aux feuilles. La profondeur d'un arbre A est égale au nombre de nœuds de la branche la plus longue. Nous la noterons $\mid A \mid$.*

Exemple II.2 (Profondeurs) *Les profondeurs des trois arbres binaires de l'exemple II.1 sont respectivement 3, 4 et 4.*

Question II.5 *Donner une définition de la profondeur d'un arbre a en fonction de $\emptyset$, $\mathcal{G}(a)$ et $\mathcal{D}(a)$.*

Question II.6 *Considérons un arbre binaire d'entiers représentant un ensemble contenant n éléments. Quelle est la forme de l'arbre dont la profondeur est maximale ? Quelle est la forme de l'arbre dont la profondeur est minimale ? Calculer la profondeur de l'arbre en fonction de n dans ces deux cas. Vous justifierez vos réponses.*

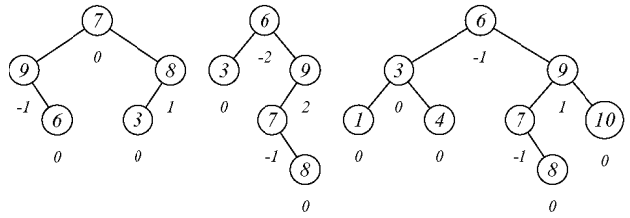
2.1.3 Déséquilibre d'un arbre

Déf. II.3 (Déséquilibre d'un nœud dans un arbre binaire) *Le déséquilibre $\Delta(a)$ d'un nœud a dans un arbre binaire est égal à la différence entre la profondeur de son fils gauche et la profondeur de son fils droit, c'est-à-dire :*

$$\Delta(\emptyset) = 0$$
$$a \neq \emptyset \Rightarrow \Delta(a) = \mid \mathcal{G}(a) \mid - \mid \mathcal{D}(a) \mid$$

Nous considérerons par la suite des arbres binaires dont chaque nœud est décoré par son déséquilibre.

Exemple II.3 (Arbres binaires d'entiers décorés) *Voici les arbres de l'exemple II.1 décorés par le déséquilibre de chaque nœud :*



2.1.4 Représentation des arbres binaires en PASCAL

Un arbre binaire d'entiers dont les nœuds sont décorés par leurs déséquilibres est représenté par le type de base `ARBRE`.

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

- `Vide` est une constante de valeur `NIL` qui représente un arbre ou une liste vide ;
- `FUNCTION Noeud(d:INTEGER;fg:ARBRE;v:INTEGER;fd:ARBRE):ARBRE;` est une fonction qui renvoie un arbre dont le déséquilibre, le fils gauche, l'étiquette et le fils droit de la racine sont respectivement `d`, `fg`, `v` et `fd`,
- `FUNCTION Desequilibre(a:ARBRE):INTEGER;` est une fonction qui renvoie le déséquilibre de la racine de l'arbre `a` ;
- `FUNCTION Gauche(a:ARBRE):ARBRE;` est une fonction qui renvoie le fils gauche de la racine de l'arbre `a` ;
- `FUNCTION Etiquette(a:ARBRE):INTEGER;` est une fonction qui renvoie l'étiquette de la racine de l'arbre `a` ;
- `FUNCTION Droit(a:ARBRE):ARBRE;` est une fonction qui renvoie le fils droit de la racine de l'arbre `a`.

Exemple II.4 *L'appel suivant*

```
Noeud( 0,
   Noeud( -1,
      Vide,
      9,
      Noeud( 0, Vide, 6, Vide)),
   7,
   Noeud( 1,
      Noeud( 0, Vide, 3, Vide),
      8,
      Vide))
```

renvoie le premier arbre binaire d'entiers représenté graphiquement dans l'exemple II.3.

2.1.5 Profondeur d'un arbre binaire d'entiers

Question II.7 *Écrire en PASCAL une fonction `profondeur(a:ARBRE):INTEGER;` telle que l'appel `profondeur(a)` renvoie la profondeur de l'arbre binaire d'entiers `a`. Cette fonction devra*

être récursive ou faire appel à des fonctions auxiliaires récursives.

2.1.6 Validation d'un arbre binaire d'entiers décoré

Une première opération consiste à déterminer si les valeurs des déséquilibres des nœuds d'un arbre binaire sont correctes.

Question II.8 *Écrire en PASCAL une fonction* `valider_decore(A:ARBRE):BOOLEAN;` *telle que l'appel* `valider_decore(A)` *renvoie la valeur* `TRUE` *si l'arbre binaire* `A` *est vide ou si l'arbre binaire* `A` *n'est pas vide et si les valeurs des déséquilibres des nœuds de l'arbre binaire* `A` *sont correctes 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.*

Question II.9 *Calculer une estimation de la complexité de la fonction* `valider_decore` *en fonction du nombre de nœuds de l'arbre* `A`*. Cette estimation ne prendra en compte que le nombre d'appels récursifs effectués.*

2.2 Arbres binaires de recherche

Déf. II.4 (Arbre binaire de recherche) *Un arbre binaire de recherche est un arbre binaire d'entiers dont les étiquettes de tous les nœuds composant le fils gauche de la racine sont strictement inférieures à l'étiquette de la racine et les étiquettes de tous les nœuds composant le fils droit de la racine sont strictement supérieures à l'étiquette de la racine. Cette contrainte s'exprime sous la forme :*

$$a \neq \emptyset \Rightarrow (\forall v \in \mathcal{C}(\mathcal{G}(a)), v < \mathcal{E}(a)) \wedge (\forall v \in \mathcal{C}(\mathcal{D}(a)), \mathcal{E}(a) < v)$$

Exemple II.5 (Arbres binaires de recherche) *Le deuxième et le troisième arbre de l'exemple II.3 sont des arbres binaires de recherche.*

Notons qu'un arbre binaire de recherche ne peut contenir qu'un seul exemplaire d'une valeur donnée.

Nous considérerons par la suite des arbres de recherche dont chaque nœud est décoré par son déséquilibre.

2.2.1 Validation d'un arbre binaire de recherche

Une première opération consiste à déterminer si un arbre binaire d'entiers est un arbre binaire de recherche.

Question II.10 *Écrire en PASCAL une fonction* `valider_abr(A:ARBRE):BOOLEAN;` *telle que l'appel* `valider_abr(A)` *renvoie la valeur* `TRUE` *si l'arbre binaire d'entiers* `A` *est un arbre binaire de recherche 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.*

2.2.2 Recherche dans un arbre binaire de recherche

Une deuxième opération consiste à déterminer si un arbre binaire contient une étiquette particulière.

Question II.11 *Écrire en PASCAL une fonction* `rechercher_abr(v:INTEGER;A:ARBRE):BOOLEAN;` *telle que l'appel* `rechercher_abr(v,A)` *renvoie la valeur* `TRUE` *si l'un des nœuds de l'arbre binaire de recherche* `A` *contient l'étiquette* `v` *et la valeur* `FALSE` *sinon. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

Question II.12 *Donner des exemples de valeurs des paramètres* `v` *et* `A` *de la fonction* `rechercher_abr` *qui correspondent au meilleur et pire cas en nombre d'appels récursifs effectués.*

Calculer une estimation de la complexité dans les meilleur et pire cas de la fonction `rechercher_abr` *en fonction de la profondeur de l'arbre* `A`*. Cette estimation ne prendra en compte que le nombre d'appels récursifs effectués.*

2.2.3 Insertion dans un arbre binaire de recherche

Une deuxième opération consiste à insérer un élément dans un arbre binaire de recherche en préservant cette structure. Pour cela, l'insertion se fait au niveau des feuilles de l'arbre (ou des nœuds ne contenant qu'un seul fils en lieu et place du fils absent).

Question II.13 *Calculer $\delta(v,a)$ l'accroissement de la profondeur d'un arbre binaire de recherche a dans lequel l'étiquette v est insérée en fonction de $\emptyset$, de la relation entre v et $\mathcal{E}(a)$, de $\Delta(a)$, de $\delta(v,\mathcal{G}(a))$ ou de $\delta(v,\mathcal{D}(a))$.*

Question II.14 *Écrire en PASCAL une fonction* `inserer_abr(v:INTEGER;A:ARBRE):ARBRE;` *telle que l'appel* `inserer_abr(v,A)` *renvoie un arbre binaire de recherche contenant les mêmes étiquettes que l'arbre binaire de recherche* `A` *ainsi que l'étiquette* `v` *s'il ne la contenait pas déjà. Cette fonction doit également calculer les valeurs des déséquilibres de chaque nœud de l'arbre résultat. 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.*

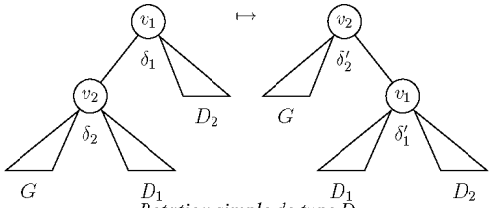
Question II.15 *Donner une estimation de la complexité dans les meilleur et pire cas de la fonction* `inserer_abr` *en fonction de la profondeur de l'arbre* `A`*. Cette estimation ne prendra en compte que le nombre d'appels récursifs effectués.*

2.3 Rotations d'un arbre binaire de recherche

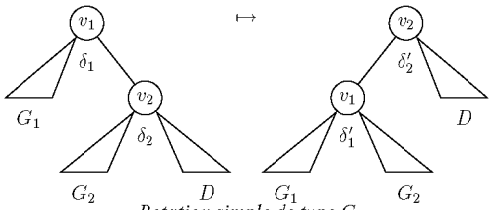
Les rotations sont des transformations d'un arbre binaire de recherche qui sont utilisées pour équilibrer la structure de l'arbre, c'est-à-dire donner une valeur proche à la longueur de chaque branche. Il s'agit donc de réduire le déséquilibre entre les différentes branches qui sont de longueurs différentes.

Déf. II.5 *Rotation simple*

Une rotation simple concerne 2 nœuds imbriqués de l'arbre. Un arbre qui ne possède pas la structure adéquate (partie gauche de la règle) ne sera pas transformé.



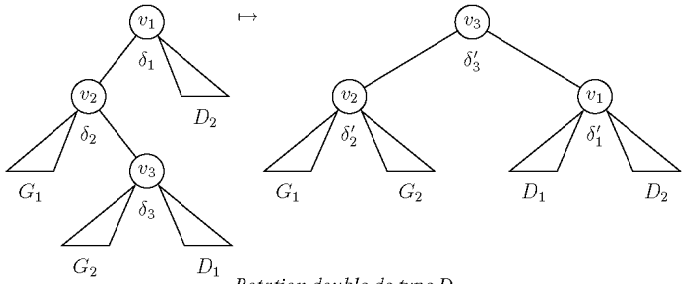
Rotation simple de type D



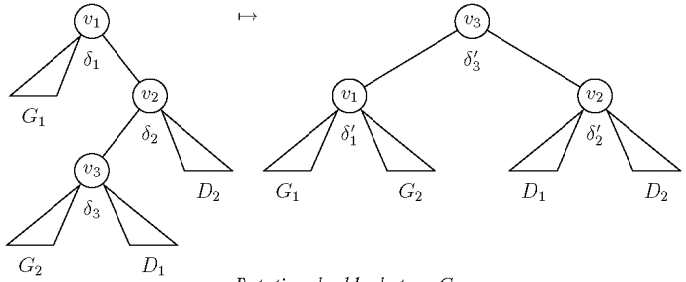
Rotation simple de type G

Déf. II.6 *Rotation double*

Une rotation double concerne 3 nœuds imbriqués de l'arbre. Un arbre qui ne possède pas la structure adéquate (partie gauche de la règle) ne sera pas transformé.



Rotation double de type D



Rotation double de type G

Question II.16 *Exprimer les rotations doubles comme des combinaisons de rotations simples.*

Question II.17 *Montrer que les rotations ne perturbent pas la structure d'arbre binaire de recherche.*

Question II.18 *Pour chaque rotation possible, calculer les nouvelles valeurs δ'_i des déséquilibres des 2 ou 3 nœuds concernés de l'arbre après la rotation en fonction des valeurs δ_i avant la rotation.*

2.3.1 Rotation simple de type D

Une première opération consiste à effectuer une rotation simple de type D dans un arbre binaire de recherche pour équilibrer celui-ci.

Question II.19 *Écrire en PASCAL une fonction* `rotationSD(A:ARBRE):ARBRE;` *telle que l'appel* `(rotationSD A)` *sur un arbre* `A` *dont la structure permet l'application d'une rotation simple de type D sur la racine renvoie l'arbre binaire de recherche* `A` *sur lequel une rotation simple de type D a été appliquée à la racine.*

2.3.2 Rotation simple de type G

Une deuxième opération consiste à effectuer une rotation simple de type G dans un arbre binaire de recherche pour équilibrer celui-ci.

Nous supposons prédéfinie la fonction `rotationSG(A:ARBRE):ARBRE;` telle que l'appel `(rotationSG A)` sur un arbre `A` dont la structure permet l'application d'une rotation simple de type G à la racine renvoie l'arbre binaire de recherche `A` sur lequel une rotation simple de type G a été appliquée à la racine. Son calcul se termine quelles que soient les valeurs de son paramètre. Elle pourra éventuellement être utilisée dans les réponses aux questions.

2.3.3 Rotation double de type D

Une troisième opération consiste à effectuer une rotation double de type D dans un arbre binaire de recherche pour équilibrer celui-ci.

Question II.20 *Écrire en PASCAL une fonction* `rotationDD(A:ARBRE):ARBRE;` *telle que l'appel* `(rotationDD A)` *sur un arbre* `A` *dont la structure permet l'application d'une rotation double de type D à la racine renvoie l'arbre binaire de recherche* `A` *sur lequel une rotation double de type D a été appliquée à la racine.*

2.3.4 Rotation double de type G

Une quatrième opération consiste à effectuer une rotation double de type G dans un arbre binaire de recherche pour équilibrer celui-ci.

Nous supposons prédéfinie la fonction `rotationDG(A:ARBRE):ARBRE;` telle que l'appel `(rotationDG A)` sur un arbre `A` dont la structure permet l'application d'une rotation double de type G à la racine renvoie l'arbre binaire de recherche `A` sur lequel une rotation double de type G a été appliquée à la racine. Son calcul se termine quelles que soient les valeurs de son paramètre. Elle pourra éventuellement être utilisée dans les réponses aux questions.

3 Arbres équilibrés

Pour réduire la complexité des opérations d'insertion et de recherche, il est nécessaire que les longueurs de toutes les branches de l'arbre aient des valeurs semblables. L'arbre a alors une structure dite équilibrée.

3.1 Définitions

Déf. II.7 (Arbres binaires équilibrés) *Un arbre binaire est dit équilibré si le déséquilibre de chacun de ses nœuds appartient à* $\{-1, 0, 1\}$. *Nous appellerons nœuds déséquilibrés les nœuds dont le déséquilibre n'appartient pas à* $\{-1, 0, 1\}$.

Exemple II.6 (Arbres binaires équilibrés) *Le premier et le troisième arbres de l'exemple II.3 sont équilibrés. Le deuxième arbre est déséquilibré. Il possède 2 nœuds déséquilibrés qui contiennent les étiquettes* 6 *et* 9.

3.2 Principe de l'équilibrage

Question II.21 *L'insertion d'une nouvelle étiquette dans un arbre binaire de recherche équilibré peut produire des nœuds déséquilibrés. Donner les différentes formes possibles pour le nœud déséquilibré le plus profond produit lors de l'insertion d'une nouvelle étiquette dans un arbre binaire équilibré en précisant la valeur du déséquilibre de ce nœud.*

Question II.22 *Montrer qu'une rotation suffit à rééquilibrer le nœud déséquilibré le plus profond qui est apparu lors de l'insertion d'une nouvelle étiquette dans un arbre binaire équilibré. Donner la rotation qui doit être utilisée pour rééquilibrer ce nœud pour chacune des formes étudiées dans la question précédente.*

3.3 Équilibrage d'un nœud

Une première opération consiste à équilibrer les nœuds qui viennent d'être déséquilibrés par l'opération d'insertion.

Question II.23 *Écrire en PASCAL une fonction* `equilibrage(A:ARBRE):ARBRE;` *avec* `A` *un arbre binaire de recherche dont la racine est déséquilibrée et dont tous les nœuds contenus dans les fils gauche et droit de la racine sont équilibrés renvoie un arbre binaire de recherche équilibré contenant les mêmes étiquettes que* `A`.

3.4 Insertion équilibrée

Une seconde opération consiste à insérer une nouvelle étiquette dans un arbre de recherche équilibré en préservant la structure équilibrée de l'arbre.

Question II.24 *Modifier la fonction* `inserer_abr(v:INTEGER;A:ARBRE):ARBRE` *écrite en PASCAL à la question II.14 telle que si* `A` *est un arbre binaire de recherche équilibré alors l'appel* `inserer_abr(v,A)` *renverra un arbre binaire de recherche équilibré.*

Question II.25 *Expliquer la fonction* `inserer_abr` *définie à la question précédente.*

3.5 Complexité des opérations dans un arbre équilibré

La complexité calculée aux questions II.15 et II.12 pour les opérations d'insertion et de recherche dépend de la profondeur de l'arbre. Pour comparer celle-ci avec celle d'une réalisation à base de liste, il faut estimer la profondeur de l'arbre en fonction du nombre d'éléments contenus dans l'ensemble. Nous avons calculé celle-ci dans le meilleur des cas pour un arbre équilibré à la question II.6. Il faut maintenant évaluer celle-ci dans le pire des cas.

Question II.26 *Quelles sont les formes de l'arbre équilibré* a *dans le meilleur cas et le pire cas pour l'insertion et la recherche d'une étiquette* v *dans* a *?*

Question II.27 *Soit* $\mathcal{N}(h)$ *le nombre de nœuds dans un arbre dans le meilleur des cas, exprimer* $\mathcal{N}(h)$ *en fonction de* h.

Question II.28 *Soit* $\mathcal{N}(h)$ *le nombre de nœuds dans un arbre dans le pire des cas, exprimer* $\mathcal{N}(h)$ *sous la forme d'une relation de récurrence dépendant de* $h-1$ *et* $h-2$.

Question II.29 *Soit* $\mathcal{F}(h) = \mathcal{N}(h) - 1$, *résoudre l'équation linéaire à coefficients constants dérivée de la relation de récurrence précédente pour obtenir la valeur de* $\mathcal{F}(h)$ *en fonction de* h.

Question II.30 *Déduire des questions II.27 et II.29 un encadrement de la profondeur d'un arbre en fonction du nombre de nœuds qu'il contient. En déduire la complexité des opérations de recherche et d'insertion dans un arbre binaire de recherche équilibré.*

Fin de l'énoncé