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

Partie I : Logique et calcul des propositions

Les jeux virtuels sur ordinateur font souvent appel à des énigmes logiques régies par le calcul des propositions. Vous participez actuellement à une partie dont les règles sont les suivantes :

Les propositions composant une énigme sont alternativement vraies et fausses, c'est-à-dire que :
- *soit les propositions de numéro pair sont vraies et les propositions de numéro impair fausses,*
- *soit les propositions de numéro pair sont fausses et les propositions de numéro impair vraies.*

Dans un labyrinthe, vous vous retrouvez bloqué dans une salle face à une porte sur laquelle se trouvent deux interrupteurs étiquetés A et B en position ouverte. Sur son seuil figure l'inscription suivante :

Pour ouvrir la porte :

P1 Il faut fermer l'interrupteur A.

P2 Il faut fermer simultanément les interrupteurs A et B.

P3 Il ne faut pas fermer l'interrupteur B.

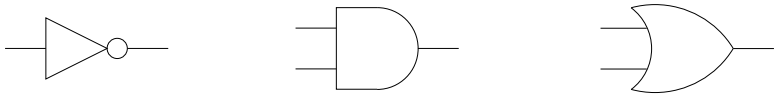
Attention, en cas d'erreur la salle s'auto-détruira...

Question I.1 *Exprimer P_1, P_2 et P_3 sous la forme de formules du calcul des propositions dépendant de A et de B.*

Question I.2 *Exprimer la règle du jeu dans le contexte des propositions P_1, P_2 et P_3.*

Question I.3 *En utilisant le calcul des propositions (résolution avec les formules de De Morgan ou table de vérité), déterminer l'action à effectuer pour ouvrir la porte.*

Les portes logiques sont représentées graphiquement par les symboles :



Question I.4 *Les interrupteurs A et B laissent passer un courant (valeur logique vraie) quand ils sont fermés. Proposer un circuit électronique composé de portes logiques comportant deux sorties O et D. La sortie O laisse passer le courant pour ouvrir la porte. La sortie D laisse passer le courant pour détruire la salle.*

Partie II : Automates et langages

Le but de cet exercice est l'étude des propriétés de l'opération $\times$ 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. II.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

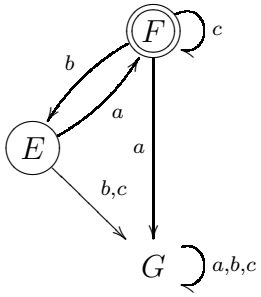
Les automates peuvent être représentés par un schéma suivant les conventions :

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

$$
\begin{array}{l}
Q_1 = \{E, F, G\} \\
X = \{a, b, c\} \\
i_1 = E \\
T_1 = \{F\} \\
\begin{array}{lll}
\delta_1(E,a) = F, & \delta_1(E,b) = G, & \delta_1(E,c) = G, \\
\delta_1(F,a) = G, & \delta_1(F,b) = E, & \delta_1(F,c) = F, \\
\delta_1(G,a) = G, & \delta_1(G,b) = G, & \delta_1(G,c) = G
\end{array}
\end{array}
$$

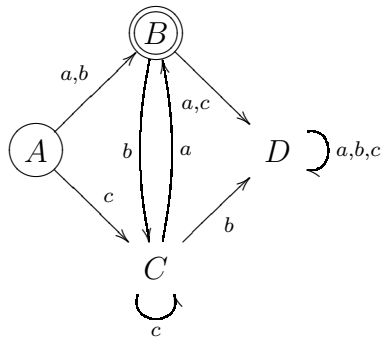
est représenté par le graphe suivant :



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

$$
\begin{array}{l}
Q_2 = \{A, B, C, D\} \\
X = \{a, b, c\} \\
i_2 = A \\
T_2 = \{B\} \\
\begin{array}{lll}
\delta_2(A,a) = B, & \delta_2(A,b) = B, & \delta_2(A,c) = C, \\
\delta_2(B,a) = D, & \delta_2(B,b) = C, & \delta_2(B,c) = D, \\
\delta_2(C,a) = B, & \delta_2(C,b) = D, & \delta_2(C,c) = C, \\
\delta_2(D,a) = D, & \delta_2(D,b) = D, & \delta_2(D,c) = D
\end{array}
\end{array}
$$

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 :

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

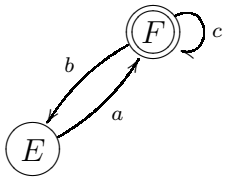
$$
\left\{
\begin{array}{l}
\forall e \in X, \\
\forall m \in X^\star, \\
\forall o \in Q, \\
\forall d \in Q,
\end{array}
\right.
\quad \delta^\star(o, e.m) = d \Leftrightarrow \exists q \in Q,\ (\delta(o,e) = q) \wedge (\delta^\star(q,m) = d)
$$

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 :

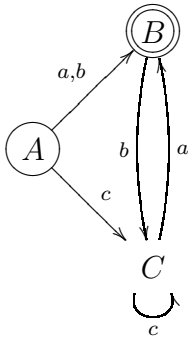
$$L(A) = \{m \in X^\star \mid \exists d \in T,\ \delta^\star(i, m) = d\}$$

Notons que certains états et transitions ne sont pas utiles dans la description d'un langage car ils ne permettent pas d'aller d'un état initial à un état terminal.

Exemple II.3 *Le sous-automate de $\mathcal{E}_1$ (exemple II.1) composé des états et transitions utiles est représenté par le graphe suivant :*



Exemple II.4 *Le sous-automate de $\mathcal{E}_2$ (exemple II.2) composé des états et transitions utiles est représenté par le graphe suivant :*



Question II.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 II.1 et $\mathcal{E}_2$ de l'exemple II.2.*

2 Composition d'automates finis complets déterministes

2.1 Définition

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

Déf. II.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 $\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2$ qui résulte de la composition de $\mathcal{A}_1$ et $\mathcal{A}_2$ est défini par :*

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

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

Question II.2 *En considérant les exemples II.1 et II.2, construire le graphe représentant l'automate $\mathcal{E}_1 \times \mathcal{E}_2$ (seuls les états et les transitions utiles devront être construits).*

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

2.2 Propriétés

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

Question II.5 *Montrer que :*

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

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

$$m \in L(\mathcal{A}_1 \times \mathcal{A}_2) \Leftrightarrow m \in L(\mathcal{A}_1) \wedge m \in L(\mathcal{A}_2)$$

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

Partie III : Algorithmique et programmation en CaML

Cette partie doit être traitée par les étudiants qui ont utilisé le langage CaML dans le cadre des enseignements d'informatique. Les fonctions écrites devront être récursives ou faire appel à des fonctions auxiliaires récursives. Elles ne devront pas utiliser d'instructions itératives (`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 III.22, 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. L'objectif de ce problème est l'étude d'une réalisation particulière due à Guibas et Sedgewick de cette structure à base d'arbres binaires de recherche quasi-équilibrés nommés arbres bicolores (car les noeuds sont colorés avec deux couleurs différentes). L'objectif de cette réalisation est d'optimiser à la fois l'occupation mémoire et la durée des opérations d'insertion et d'élimination 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 dans un ensemble.

Question III.1 *Écrire en CaML une fonction* `insertion_ens` *de type* `int -> ensemble -> ensemble` *telle que l'appel* `(insertion_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 III.2 *Calculer une estimation de la complexité de la fonction* `insertion_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 Élimination dans un ensemble

La seconde opération consiste à éliminer un entier d'un ensemble.

Question III.3 *Écrire en CaML une fonction* `elimination_ens` *de type* `int -> ensemble -> ensemble` *telle que l'appel* `(elimination_ens v E)` *renvoie un ensemble contenant les mêmes entiers que l'ensemble* `E` *sauf l'entier* `v` *s'il figurait dans* `E`*. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

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

Calculer une estimation de la complexité dans les meilleur et pire cas de la fonction `elimination_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 bicolore

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

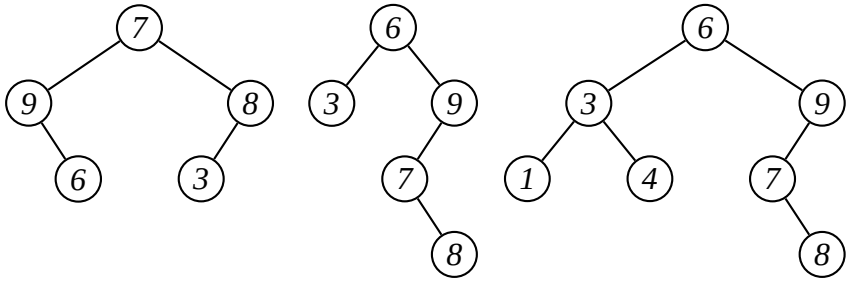
2.1 Arbres binaires d'entiers

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

Exemple III.1 (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. III.2 (Profondeur d'un arbre) *Les branches d'un arbre relient la racine aux sous-arbres vides. La profondeur d'un arbre A est égale au nombre de nœuds de la branche la plus longue. Nous la noterons $\mid A \mid$.*

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

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

Question III.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.*

2.2 Arbres bicolores

Nous considérerons par la suite des arbres binaires dont chaque nœud est décoré par une couleur blanche ou grise.

L'utilisation de contraintes sur les couleurs des nœuds permet de réduire le déséquilibre possible entre les profondeurs des différentes branches d'un arbre.

Déf. III.3 (Arbres bicolores) *Un arbre coloré est un arbre dont les nœuds sont également décorés par une couleur. Un arbre bicolore est un arbre coloré qui respecte les contraintes suivantes :*

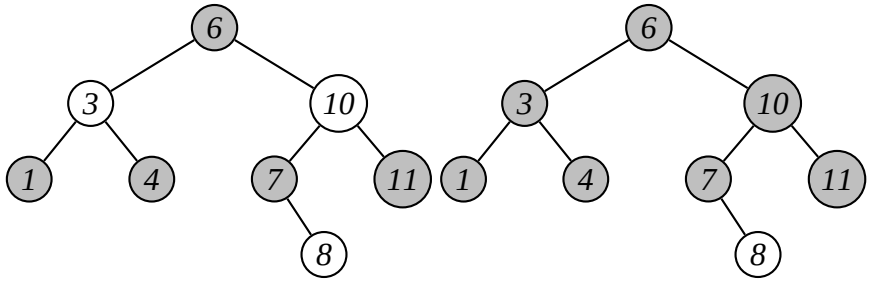
P1 *Il existe deux couleurs différentes. Nous choisirons Blanc et Gris.*

P2 *Tous les nœuds fils directs d'un nœud coloré en Blanc doivent être colorés en Gris.*

P3 *Toutes les branches doivent contenir le même nombre de nœuds colorés en Gris.*

Notons qu'un même arbre binaire peut être coloré de différentes manières qui respectent les contraintes des arbres bicolores.

Exemple III.3 (Arbres binaires bicolores) *Voici deux colorations du troisième arbre de l'exemple III.1 qui respectent les contraintes des arbres bicolores :*



2.2.1 Rang d'un arbre bicolore

Déf. III.4 (Rang d'un arbre bicolore) *Le rang d'un arbre bicolore A est égal au nombre de nœuds colorés en gris dans une des branches de l'arbre (par définition de l'arbre bicolore, ce nombre est le même pour toutes les branches). Nous le noterons $\mathcal{R}(A)$.*

Exemple III.4 (Rang d'un arbre bicolore) *Les rangs des arbres bicolores de l'exemple III.3 sont respectivement 2 et 3.*

Question III.7 *Soit A un arbre bicolore composé de n nœuds avec $n > 0$. Montrer que $\mathcal{R}(A) \leq \mid A \mid \leq 2 \times \mathcal{R}(A) + 1$ et que $2^{\mathcal{R}(A)} - 1 \leq n$.*

Question III.8 *Soit A un arbre bicolore composé de n nœuds avec $n > 0$. Montrer que $E(\log_2(n)) \leq \mid A \mid \leq 2 \times E(\log_2(n))$ (*$\log_2$ *est le logarithme en base 2 et $E()$ est la partie entière).*

2.2.2 Représentation des arbres binaires bicolores en CaML

Un arbre binaire d'entiers dont les nœuds sont décorés par une couleur est représenté par les types CaML :

```
type couleur = Blanc | Gris;;

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

Notons que l'arbre vide n'a pas de couleur associée.

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

Exemple III.5 *Le terme suivant*

```
Noeud( Gris,
   Noeud( Blanc,
      Noeud( Gris, Vide, 1, Vide),
      3,
      Noeud( Gris, Vide, 4, Vide)),
   6,
   Noeud( Blanc,
      Noeud( Gris,
         Vide,
         7,
         Noeud( Blanc, Vide, 8, Vide)),
      10,
      Noeud( Gris, Vide, 11, Vide)))
```

est alors associé à l'arbre binaire bicolore représenté graphiquement dans l'exemple III.3.

2.2.3 Rang d'un arbre bicolore d'entiers

Question III.9 *Écrire en CaML une fonction `rang` de type `arbre -> int` telle que l'appel `(rang a)` renvoie le rang de l'arbre bicolore d'entiers `a`. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

2.2.4 Validation d'un arbre bicolore d'entiers

Une première opération consiste à déterminer si la structure de l'arbre et les valeurs des couleurs des nœuds d'un arbre binaire sont compatibles avec la définition d'un arbre bicolore.

Question III.10 *Écrire en CaML une fonction `validation_bicolore` de type `arbre -> bool` telle que l'appel `(validation_bicolore 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 couleurs des éléments de l'arbre binaire `A` respectent les contraintes pour que `A` soit un arbre bicolore 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 III.11 *Calculer une estimation de la complexité de la fonction* `validation_bicolore` *en fonction du nombre de nœuds de l'arbre* `A`*. Cette estimation ne prendra en compte que le nombre d'appels récursifs effectué.*

2.3 Arbres binaires de recherche

Déf. III.5 (Arbre binaire de recherche) *Un arbre binaire de recherche est un arbre binaire d'entiers dont les fils de la racine sont des arbres binaires de recherche et 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 :*

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

Exemple III.6 (Arbres binaires de recherche) *Le deuxième et le troisième arbre de l'exemple III.1 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 bicolores.

2.3.1 Validation d'un arbre binaire de recherche

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

Question III.12 *Écrire en CaML une fonction* `validation_abr` *de type* `arbre -> bool` *telle que l'appel* `(validation_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.*

2.3.2 Insertion dans un arbre binaire de recherche

Une seconde 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 sous-arbres vides contenus dans l'arbre.

Question III.13 *Écrire en CaML une fonction* `insertion_abr` *de type* `int -> arbre -> arbre` *telle que l'appel* `(insertion_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 associer la couleur blanche au nœud de la valeur insérée. 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.*

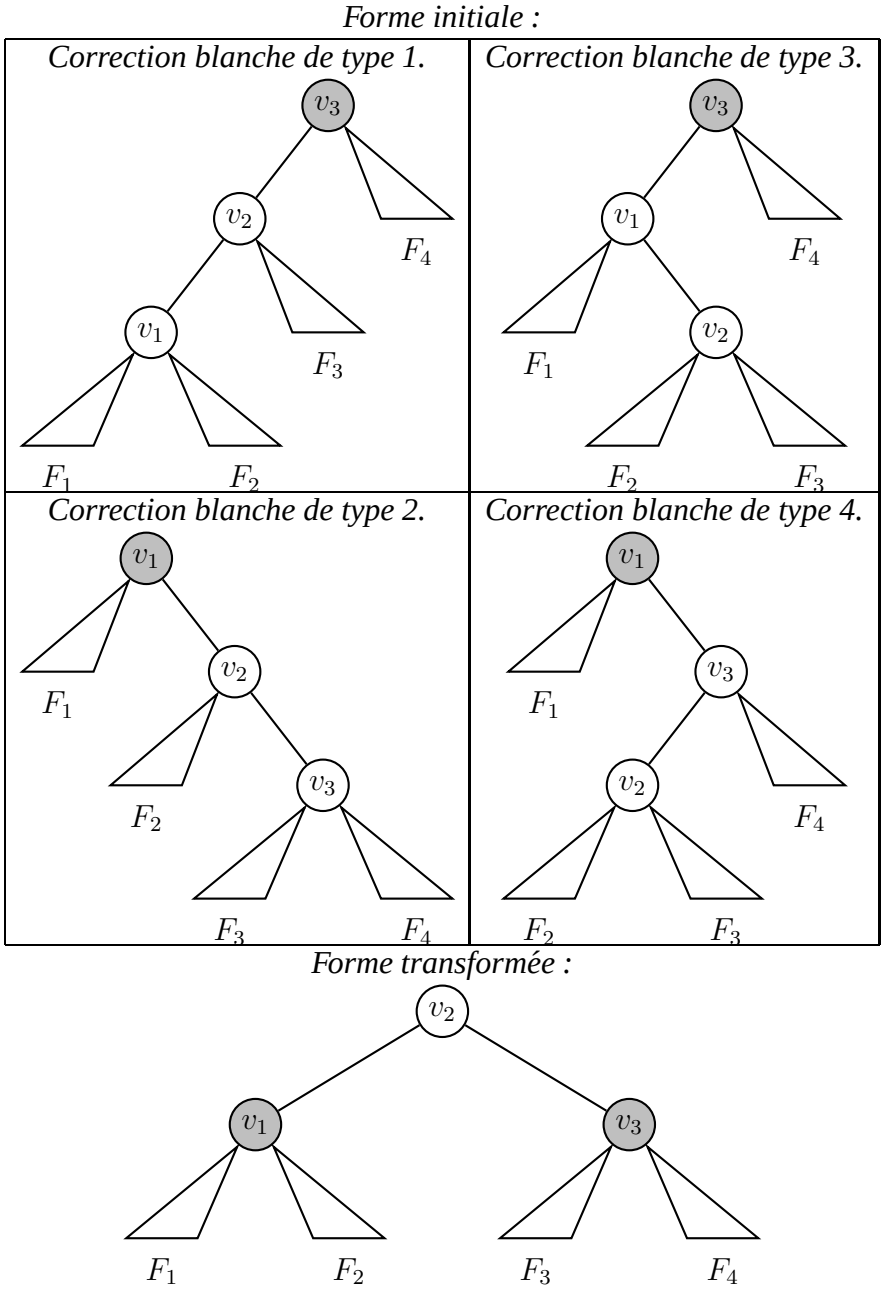
Question III.14 *Donner une estimation de la complexité dans les meilleur et pire cas de la fonction* `insertion_abr` *en fonction de la profondeur de l'arbre* `A`*. Cette estimation ne prendra en compte que le nombre d'appels récursifs effectués. Donner également une estimation de la complexité si* `A` *n'est pas un arbre bicolore.*

2.4 Préservation de la structure bicolore

Lors de l'insertion d'une valeur dans un arbre bicolore, le nouveau nœud est coloré en blanc. L'insertion d'un élément dans un arbre binaire de recherche bicolore peut invalider les contraintes et produire un arbre binaire de recherche coloré qui n'est pas bicolore. Les transformations suivantes visent à restaurer la structure d'arbre bicolore à partir de l'arbre coloré obtenu par insertion.

Question III.15 *Quelles sont les propriétés d'un arbre bicolore (parmi **P1**, **P2** et **P3**) qui peuvent être invalidées par l'insertion d'une valeur sachant que l'on associe la couleur blanche au nœud associé à cette valeur ?*

Déf. III.6 (Correction blanche) *Une correction blanche concerne 3 nœuds imbriqués de l'arbre. Elle s'applique uniquement si l'arbre possède une des quatre formes suivantes. Un arbre qui ne possède pas la forme adéquate ne sera pas transformé.*



Question III.16 *Montrer que si F_1, F_2, F_3 et F_4 sont des arbres bicolores de rang n alors le résultat d'une correction blanche est un arbre bicolore de rang $n+1$.*

Question III.17 *Insérer la valeur* 9 *dans le premier arbre de recherche bicolore de l'exemple III.3, puis appliquer une correction blanche autant de fois que nécessaire pour obtenir un arbre bicolore. Représenter tous les arbres colorés intermédiaires.*

Question III.18 *Montrer que l'insertion d'une valeur dans un arbre de recherche bicolore suivie de l'application de corrections blanches sur la branche dans laquelle la valeur a été insérée tant que ces corrections sont applicables permet d'obtenir un arbre de recherche bicolore.*

Question III.19 *Soit un arbre de recherche bicolore* A*, montrer que le nombre de corrections blanches nécessaires pour obtenir un arbre bicolore après l'insertion d'une valeur dans* A *est strictement inférieur à la différence entre la profondeur et le rang de l'arbre avant insertion (*$\mid A \mid - \mathcal{R}(A)$*).*

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

2.5 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 III.21 *Modifier la fonction* `insertion_abr` *de type* `int -> arbre -> arbre` *écrite en CaML à la question III.13 telle que si* `A` *est un arbre binaire de recherche bicolore alors l'appel* `(insertion_abr v A)` *renverra un arbre binaire de recherche bicolore.*

Question III.22 *Expliquer la fonction* `insertion_abr` *définie à la question précédente.*

Partie III : Algorithmique et programmation en PASCAL

Cette partie doit être traitée par les étudiants qui ont utilisé le langage PASCAL dans le cadre des enseignements d'informatique. Les fonctions écrites devront être récursives ou faire appel à des fonctions auxiliaires récursives. Elles ne devront pas utiliser d'instructions itératives (`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 III.22, 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. L'objectif de ce problème est l'étude d'une réalisation particulière due à Guibas et Sedgewick de cette structure à base d'arbres binaires de recherche quasi-équilibrés nommés arbres bicolores (car les noeuds sont colorés avec deux couleurs différentes). L'objectif de cette réalisation est d'optimiser à la fois l'occupation mémoire et la durée des opérations d'insertion et d'élimination 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 dans un ensemble.

Question III.1 *Écrire en PASCAL une fonction* `insertion_ens(v:INTEGER;E:ENSEMBLE):ENSEMBLE;` *telle que l'appel* `insertion_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 III.2 *Calculer une estimation de la complexité de la fonction* `insertion_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 Élimination dans un ensemble

La seconde opération consiste à éliminer un entier d'un ensemble.

Question III.3 *Écrire en PASCAL une fonction* `elimination_ens(v:INTEGER;E:ENSEMBLE):ENSEMBLE;` *telle que l'appel* `elimination_ens(v,E)` *renvoie un ensemble contenant les mêmes entiers que l'ensemble* `E` *sauf l'entier* `v` *s'il figurait dans* `E`*. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

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

Calculer une estimation de la complexité dans les meilleur et pire cas de la fonction `elimination_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 bicolore

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

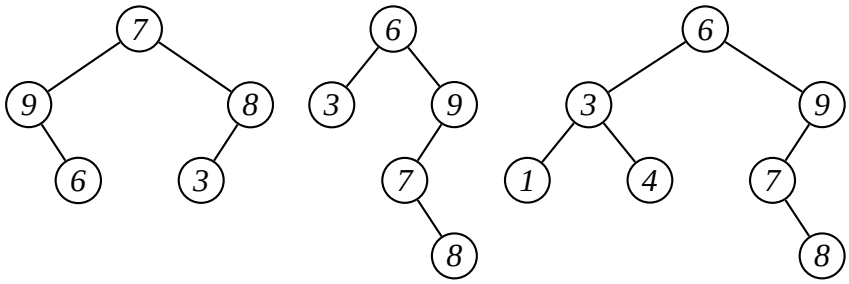
2.1 Arbres binaires d'entiers

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

Exemple III.1 (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. III.2 (Profondeur d'un arbre) *Les branches d'un arbre relient la racine aux sous-arbres vides. La profondeur d'un arbre A est égale au nombre de nœuds de la branche la plus longue. Nous la noterons $\mid A \mid$.*

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

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

Question III.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.*

2.2 Arbres bicolores

Nous considérerons par la suite des arbres binaires dont chaque nœud est décoré par une couleur blanche ou grise.

L'utilisation de contraintes sur les couleurs des nœuds permet de réduire le déséquilibre possible entre les profondeurs des différentes branches d'un arbre.

Déf. III.3 (Arbres bicolores) *Un arbre coloré est un arbre dont les nœuds sont également décorés par une couleur. Un arbre bicolore est un arbre coloré qui respecte les contraintes suivantes :*

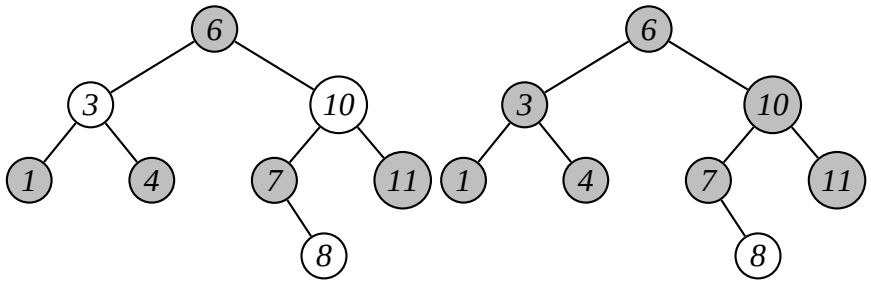
P1 *Il existe deux couleurs différentes. Nous choisirons Blanc et Gris.*

P2 *Tous les nœuds fils directs d'un nœud coloré en Blanc doivent être colorés en Gris.*

P3 *Toutes les branches doivent contenir le même nombre de nœuds colorés en Gris.*

Notons qu'un même arbre binaire peut être coloré de différentes manières qui respectent les contraintes des arbres bicolores.

Exemple III.3 (Arbres binaires bicolores) *Voici deux colorations du troisième arbre de l'exemple III.1 qui respectent les contraintes des arbres bicolores :*



2.2.1 Rang d'un arbre bicolore

Déf. III.4 (Rang d'un arbre bicolore) *Le rang d'un arbre bicolore A est égal au nombre de nœuds colorés en gris dans une des branches de l'arbre (par définition de l'arbre bicolore, ce nombre est le même pour toutes les branches). Nous le noterons $\mathcal{R}(A)$.*

Exemple III.4 (Rang d'un arbre bicolore) *Les rangs des arbres bicolores de l'exemple III.3 sont respectivement 2 et 3.*

Question III.7 *Soit A un arbre bicolore composé de n nœuds avec $n > 0$. Montrer que $\mathcal{R}(A) \leq \mid A \mid \leq 2 \times \mathcal{R}(A) + 1$ et que $2^{\mathcal{R}(A)} - 1 \leq n$.*

Question III.8 *Soit A un arbre bicolore composé de n nœuds avec $n > 0$. Montrer que $E(\log_2(n)) \leq \mid A \mid \leq 2 \times E(\log_2(n))$ ($\log_2$ est le logarithme en base 2 et $E()$ est la partie entière).*

2.2.2 Représentation des arbres binaires bicolores en PASCAL

Une couleur est représentée par un type énuméré `COULEUR` dont les deux valeurs possibles sont `GRIS` et `BLANC`.

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

Notons que l'arbre vide n'a pas de couleur associée.

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(c:COULEUR;fg:ARBRE;v:INTEGER;fd:ARBRE):ARBRE;` est une fonction qui renvoie un arbre dont la couleur, le fils gauche, l'étiquette et le fils droit de la racine sont respectivement `c`, `fg`, `v` et `fd`,
- `FUNCTION Couleur(a:ARBRE):COULEUR;` est une fonction qui renvoie la couleur 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 III.5 *L'appel suivant*

```
Noeud( GRIS,
   Noeud( BLANC,
      Noeud( GRIS, Vide, 1, Vide),
      3,
      Noeud( GRIS, Vide, 4, Vide)),
   6,
   Noeud( BLANC,
      Noeud( GRIS,
         Vide,
         7,
         Noeud( BLANC, Vide, 8, Vide)),
      10,
      Noeud( GRIS, Vide, 11, Vide)))
```

renvoie l'arbre binaire d'entiers bicolore représenté graphiquement dans l'exemple III.3.

2.2.3 Rang d'un arbre bicolore d'entiers

Question III.9 *Écrire en PASCAL une fonction* `rang(a:ARBRE):INTEGER;` *telle que l'appel* `rang(a)` *renvoie le rang de l'arbre bicolore d'entiers* `a`*. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

2.2.4 Validation d'un arbre bicolore d'entiers

Une première opération consiste à déterminer si la structure de l'arbre et les valeurs des couleurs des nœuds d'un arbre binaire sont compatibles avec la définition d'un arbre bicolore.

Question III.10 *Écrire en PASCAL une fonction* `validation_decore(A:ARBRE):BOOLEAN;` *telle que l'appel* `validation_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 couleurs des éléments de l'arbre binaire* `A` *respectent les contraintes pour que* `A` *soit un arbre bicolore 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 III.11 *Calculer une estimation de la complexité de la fonction* `validation_bicolore` *en fonction du nombre de nœuds de l'arbre* `A`*. Cette estimation ne prendra en compte que le nombre d'appels récursifs effectué.*

2.3 Arbres binaires de recherche

Déf. III.5 (Arbre binaire de recherche) *Un arbre binaire de recherche est un arbre binaire d'entiers dont les fils de la racine sont des arbres binaires de recherche et 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 :*

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

Exemple III.6 (Arbres binaires de recherche) *Le deuxième et le troisième arbre de l'exemple III.1 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 bicolores.

2.3.1 Validation d'un arbre binaire de recherche

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

Question III.12 *Écrire en PASCAL une fonction* `validation_abr(A:ARBRE):BOOLEAN;` *telle que l'appel* `validation_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.*

2.3.2 Insertion dans un arbre binaire de recherche

Une seconde 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 sous-arbres vides contenus dans l'arbre.

Question III.13 *Écrire en PASCAL une fonction* `insertion_abr(v:INTEGER;A:ARBRE):ARBRE;` *telle que l'appel* `insertion_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 associer la couleur blanche au nœud de la valeur insérée. 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 III.14 *Donner une estimation de la complexité dans les meilleur et pire cas de la fonction* `insertion_abr` *en fonction de la profondeur de l'arbre* `A`*. Cette estimation ne prendra en compte que le nombre d'appels récursifs effectués. Donner également une estimation de la complexité si* `A` *n'est pas un arbre bicolore.*

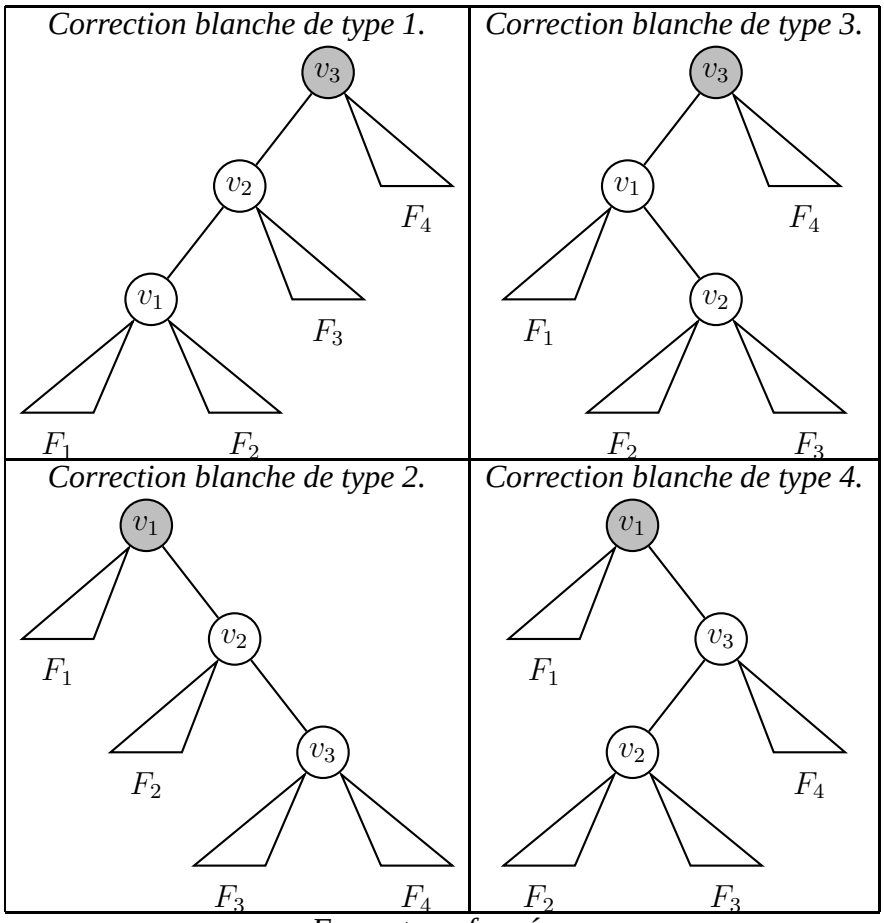
2.4 Préservation de la structure bicolore

Lors de l'insertion d'une valeur dans un arbre bicolore, le nouveau nœud est coloré en blanc. L'insertion d'un élément dans un arbre binaire de recherche bicolore peut invalider les contraintes et produire un arbre binaire de recherche coloré qui n'est pas bicolore. Les transformations suivantes visent à restaurer la structure d'arbre bicolore à partir de l'arbre coloré obtenu par insertion.

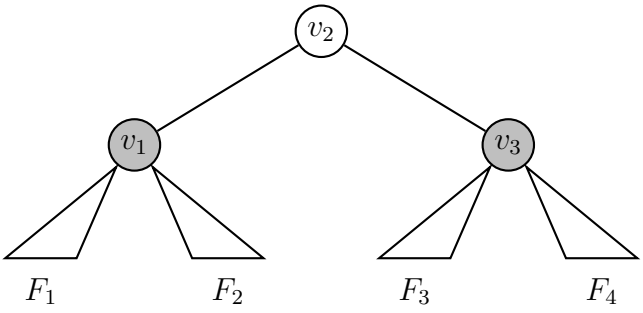
Question III.15 *Quelles sont les propriétés d'un arbre bicolore (parmi* ***P1****,* ***P2*** *et* ***P3****) qui peuvent être invalidées par l'insertion d'une valeur sachant que l'on associe la couleur blanche au nœud associé à cette valeur ?*

Déf. III.6 (Correction blanche) *Une correction blanche concerne 3 nœuds imbriqués de l'arbre. Elle s'applique uniquement si l'arbre possède une des quatre formes suivantes. Un arbre qui ne possède pas la forme adéquate ne sera pas transformé.*

Forme initiale :



Forme transformée :



Question III.16 *Montrer que si F_1, F_2, F_3 et F_4 sont des arbres bicolores de rang n alors le résultat d'une correction blanche est un arbre bicolore de rang $n+1$.*

Question III.17 *Insérer la valeur 9 dans le premier arbre de recherche bicolore de l'exemple III.3, puis appliquer une correction blanche autant de fois que nécessaire pour obtenir un arbre bicolore. Représenter tous les arbres colorés intermédiaires.*

Question III.18 *Montrer que l'insertion d'une valeur dans un arbre de recherche bicolore suivie de l'application de corrections blanches sur la branche dans laquelle la valeur a été insérée tant que ces corrections sont applicables permet d'obtenir un arbre de recherche bicolore.*

Question III.19 *Soit un arbre de recherche bicolore A, montrer que le nombre de corrections blanches nécessaires pour obtenir un arbre bicolore après l'insertion d'une valeur dans A est strictement inférieur à la différence entre la profondeur et le rang de l'arbre avant insertion ($\mid A \mid - \mathcal{R}(A)$).*

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

2.5 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 III.21 *Modifier la fonction* `insertion_abr(v:INTEGER;A:ARBRE):ARBRE` *écrite en PASCAL à la question III.13 telle que si* `A` *est un arbre binaire de recherche bicolore alors l'appel* `insertion_abr(v,A)` *renverra un arbre binaire de recherche bicolore.*

Question III.22 *Expliquer la fonction* `insertion_abr` *définie à la question précédente.*

Fin de l'énoncé