Les fonctions devront être récursives ou faire appel à des fonctions auxiliaires récursives. Elles ne devront pas utiliser d'instructions itératives (`for, while, ...`) ni de références.
Il est rappelé que toute fonction doit être expliquée et qu'il est demandé de donner la signature de toutes les fonctions auxiliaires.
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 l'aide d'arbres binaires de recherches 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 cet ensemble.

Partie I - 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.
Un ensemble d'entiers est représenté par le type `ensemble` équivalent à une liste de type `int` :
```
type ensemble = int list;;
```
Nous ferons l'hypothèse qu'un élément appartenant à un ensemble n'apparaît qu'une seule fois dans la liste représentant cet ensemble.

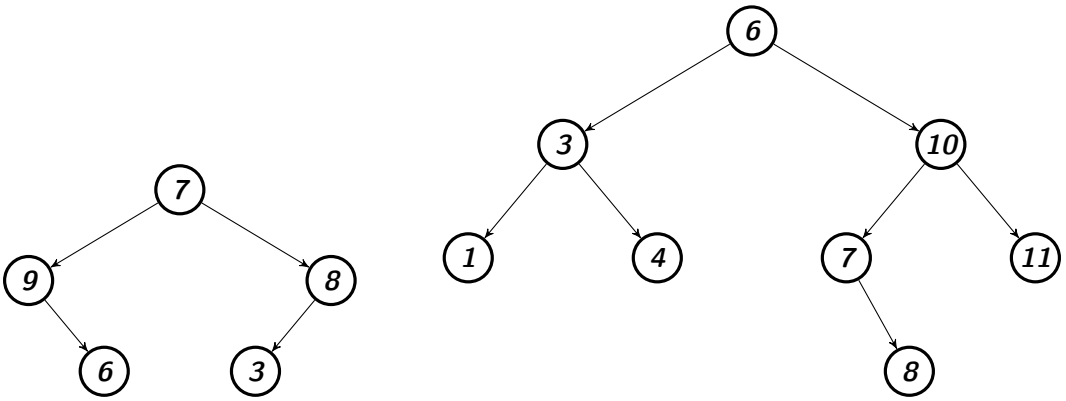
1. L'insertion
   (a) Écrire une fonction `insertionEns : int -> ensemble -> ensemble` telle que l'appel `insertionEns v e` renvoie un ensemble contenant les mêmes éléments que l'ensemble `e` ainsi que l'élément `v` s'il ne figurait pas déjà dans `e`.
   (b) Donner (en justifiant rapidement) la complexité de la fonction `insertionEns` en fonction du nombre d'éléments de l'ensemble `e`.
2. L'élimination
   (a) Écrire une fonction `eliminationEns : int -> ensemble -> ensemble` telle que l'appel `eliminationEns v e` renvoie un ensemble contenant les mêmes éléments que l'ensemble `e` sauf l'élément `v` s'il figurait dans `e`.
   (b) Donner (en justifiant rapidement) la complexité de la fonction `eliminationEns` en fonction du nombre d'éléments de l'ensemble `e`. On donnera en particulier des exemples qui correspondent aux meilleur et pire cas.

Partie II - Arbres binaires bicolores

L'utilisation d'une structure d'arbre binaire bicolore permet de réduire la complexité en temps de calcul pour les opérations d'insertion et d'élimination.
Un ensemble d'entiers peut être réalisé par un arbre binaire en utilisant les étiquettes des noeuds pour représenter les éléments contenus dans l'ensemble.

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

Exemple 1 (Arbres binaires d'entiers) *Voici deux exemples d'arbres binaires étiquetés par des entiers (les sous arbres vides des noeuds ne sont pas représentés) :*



Definition 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 noeuds de la branche la plus longue. Nous la noterons* $|a|$.

Exemple 2 (Profondeur d'un arbre) *Les profondeurs des deux arbres de l'exemple 1 sont respectivement 3 et 4.*

3. Donner une définition de la profondeur d'un arbre a en fonction de $\emptyset$, de $G(a)$ et de $D(a)$.
4. 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.

Nous considérons par la suite des arbres binaires dont chaque noeud est décoré par une couleur blanche ou grise. L'utilisation de contraintes sur les couleurs des noeuds permet de réduire le déséquilibre possible entre les profondeurs des différentes branches d'un arbre.

Definition 3 (Arbres bicolores) *Un arbre coloré est un arbre dont les noeuds sont décorés par une couleur. Un arbre bicolore est un arbre coloré qui respecte les conditions suivantes :*

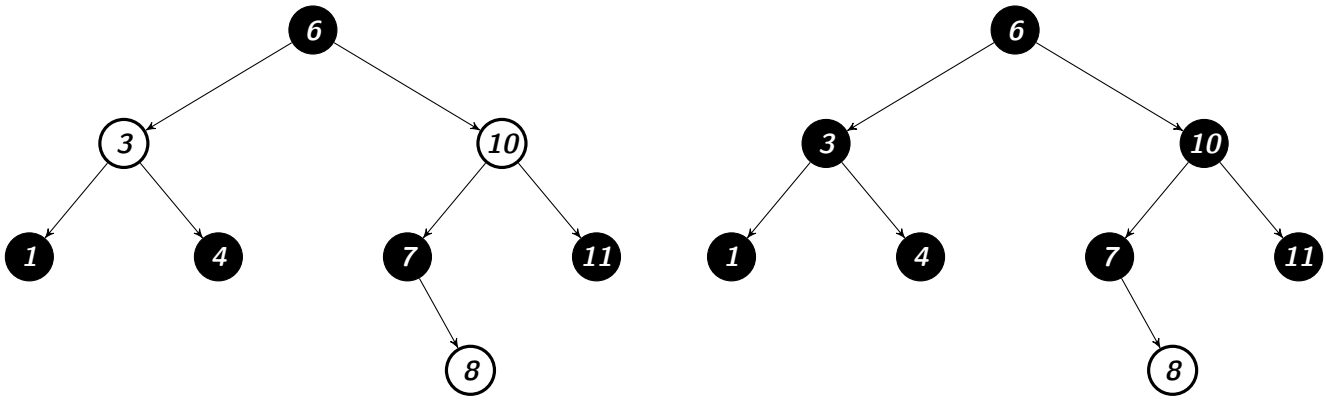
P1 *Il existe au plus deux couleurs différentes. Nous choisirons blanc et gris.*

P2 *Tous les noeuds fils directs d'un noeud coloré en blanc doivent être colorés en gris.*

P3 *Toutes les branches doivent contenir le même nombre de noeuds colorés en gris.*

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

Exemple 3 (Arbres binaires bicolores) *Voici deux colorations du deuxième arbre de l'exemple précédent respectant les contraintes des arbres bicolores.*



Definition 4 (Rang d'un arbre bicolore) *Le rang d'un arbre bicolore a est égal au nombre de noeuds colorés en gris dans une branche de l'arbre (par définition de l'arbre bicolore, ce nombre est le même dans toutes les branches). Nous le noterons* $\mathscr{R}(a)$.

Exemple 4 *Les rangs des arbres bicolores ci-dessus sont respectivement* 2 *et* 3.

5. (a) Soit a un arbre bicolore. Montrer que

$$\mathscr{R}(a) \leq |a| \leq 2 \times \mathscr{R}(a) + 1$$

   (b) Soit a un arbre bicolore composé de n noeuds avec $n > 0$. Montrer que $2^{\mathscr{R}(a)} - 1 \leq n$.
   On attend un raisonnement précis. On pourra rédiger une récurrence.

   (c) Soit a un arbre bicolore composé de n noeuds avec $n > 0$. Montrer que

$$E(\log_2(n)) \leq |a| \leq 2 \times E(\log_2(n+1)) + 1$$

   où $\log_2$ est le logarithme en base 2 et E la fonction partie entière.

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

```
type couleur = Blanc | Gris;;
type arbre = Vide | N of couleur * arbre * int * arbre;;
```

Notons que l'arbre vide n'a pas de couleur associée.
Dans l'appel `N(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.

6. Écrire une fonction `rang : arbre -> int` telle que l'appel `rang a` renvoie le rang de l'arbre bicolore d'entiers `a`.
7. Écrire une fonction `validationBicolore : arbre -> bool` telle que l'appel `validationBicolore a` renvoie la valeur `true` si l'arbre binaire `a` est vide ou si les valeurs des couleurs de ses éléments respectent les contraintes pour que ce soit un arbre bicolore et la valeur `false` sinon. L'algorithme utilisé ne devra parcourir qu'une seule fois l'arbre.
8. Calculer une estimation de la complexité de la fonction `validationBicolore` en fonction du nombre de noeuds de l'arbre. Cette estimation ne prendra en compte que le nombre d'appels récursifs effectué.

Partie III - Arbres binaires de recherche colorés

Definition 5 (Arbre 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 noeuds composant le fils gauche de la racine sont strictement inférieures à l'étiquette de la racine et les étiquettes de tous les noeuds composant le fils droit de la racine sont strictement supérieures à l'étiquette de la racine. Cela signifie que*

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

Exemple 5 *Le deuxième arbre de l'exemple* 1 *est un arbre binaire de recherche mais pas le premier arbre.*

Notons qu'un arbre binaire de recherche ne peut contenir qu'un seul exemplaire d'une valeur donnée. Nous considérons par la suite des arbres de recherche bicolores.
Les premières opérations consistent à déterminer si un arbre bicolore d'entiers est un arbre binaire de recherche et à insérer un élément dans un arbre binaire d'entiers en conservant la structure.

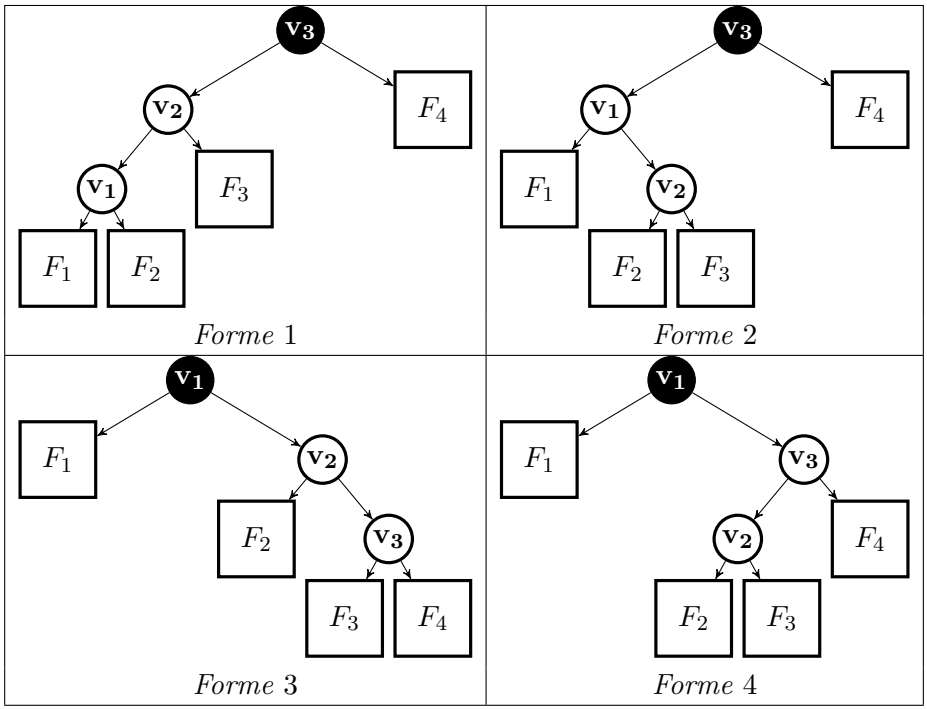
9. Écrire une fonction `validationABR : arbre -> bool` telle que `validationABR a` renvoie la valeur `true` si l'arbre binaire d'entiers `a` est un arbre binaire de recherche et la valeur `false` sinon. L'algorithme utilisé ne devra parcourir qu'une seule fois l'arbre.
10. (a) Écrire une fonction `insertionABR : int -> arbre -> arbre` telle que `insertionABR 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 noeud de la valeur insérée. L'algorithme utilisé ne devra parcourir qu'une seule fois l'arbre.

    (b) Donner une estimation de la complexité dans les meilleur et pire cas de la fonction `insertionABR` en fonction de la profondeur de l'arbre. Cette estimation prendra en compte que le nombre d'appels récursifs effectués.

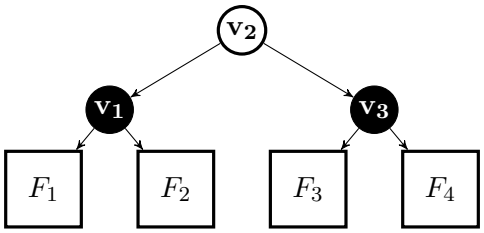
Lors de l'insertion d'une valeur dans un arbre bicolore, le nouveau noeud est coloré en blanc. L'insertion d'un élément dans un arbre binaire de recherche bicolore peut invalider les contraintes et produire un arbre binaire de recherche coloré qui n'est pas bicolore. Les transformations suivantes visent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

11. 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 noeud de cette valeur ?

Definition 6 (Correction blanche) *Une correction blanche concerne* 3 *noeuds 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.*



Dans ces quatre cas, on transforme l'arbre en l'arbre suivant



12. 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 $n+1$.
13. Insérer la valeur 9 dans le premier arbre de recherche bicolore de lexemple 3, puis appliquer une correction blanche autant de fois que nécessaire pour obtenir un arbre bicolore. Représenter tous les arbres colorés intermédiaires.
14. Expliquer comment, à l'aide de corrections blanches, on peut modifier la procédure d'insertion dans un arbre binaire de recherche bicolore pour préserver les propriétés **P1**, **P2** et **P3**.
15. Soit un arbre de recherche bicolore a, montrer que le nombre de corrections blanches nécessaires pour obtenir un arbre bicolore après linsertion dune valeur dans a est strictement inférieur à $|a| - \mathscr{R}(a)$ la différence entre la profondeur et le rang de larbre avant insertion.
16. Écrire une fonction `correctionBlanche : arbre -> arbre` telle que l appel `correctionBlanche a` sur un arbre `a` dont la structure permet l application dune correction blanche sur la racine renvoie larbre binaire de recherche sur lequel une correction blanche a été appliquée à la racine.
17. Écrire une fonction `insertionABRC : int -> arbre -> arbre` telle que si `a` est un arbre binaire de recherche coloré, l'appel `insertionABRC v a` renvoie un arbre binaire de recherche coloré contenant les mêmes étiquettes que l'arbre binaire de recherche `a` ainsi que l'étiquette `v` s'il ne la contenait pas déjà.