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.

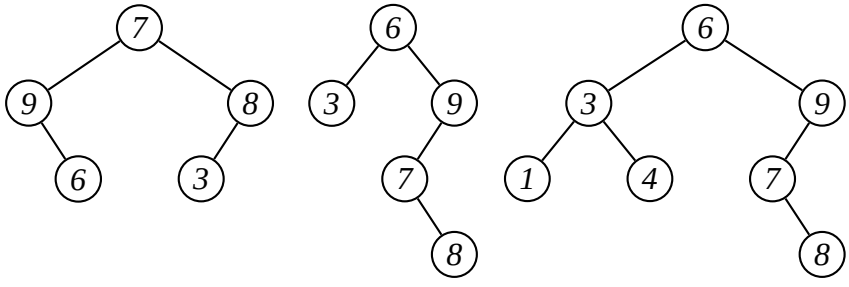
2.1 Arbres binaires d'entiers

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 :*

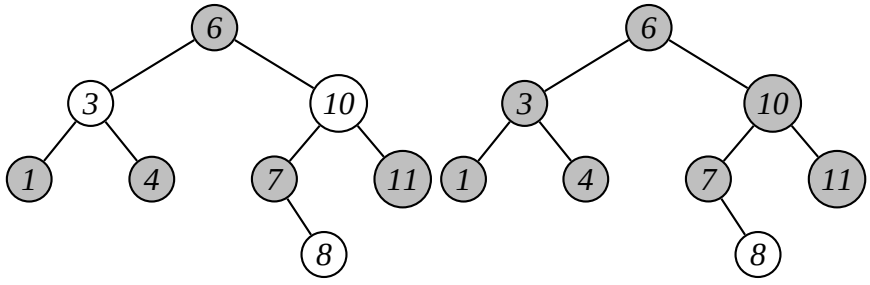
P1 *Il existe deux couleurs différentes. Nous choisirons Blanc et Gris.*

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.*

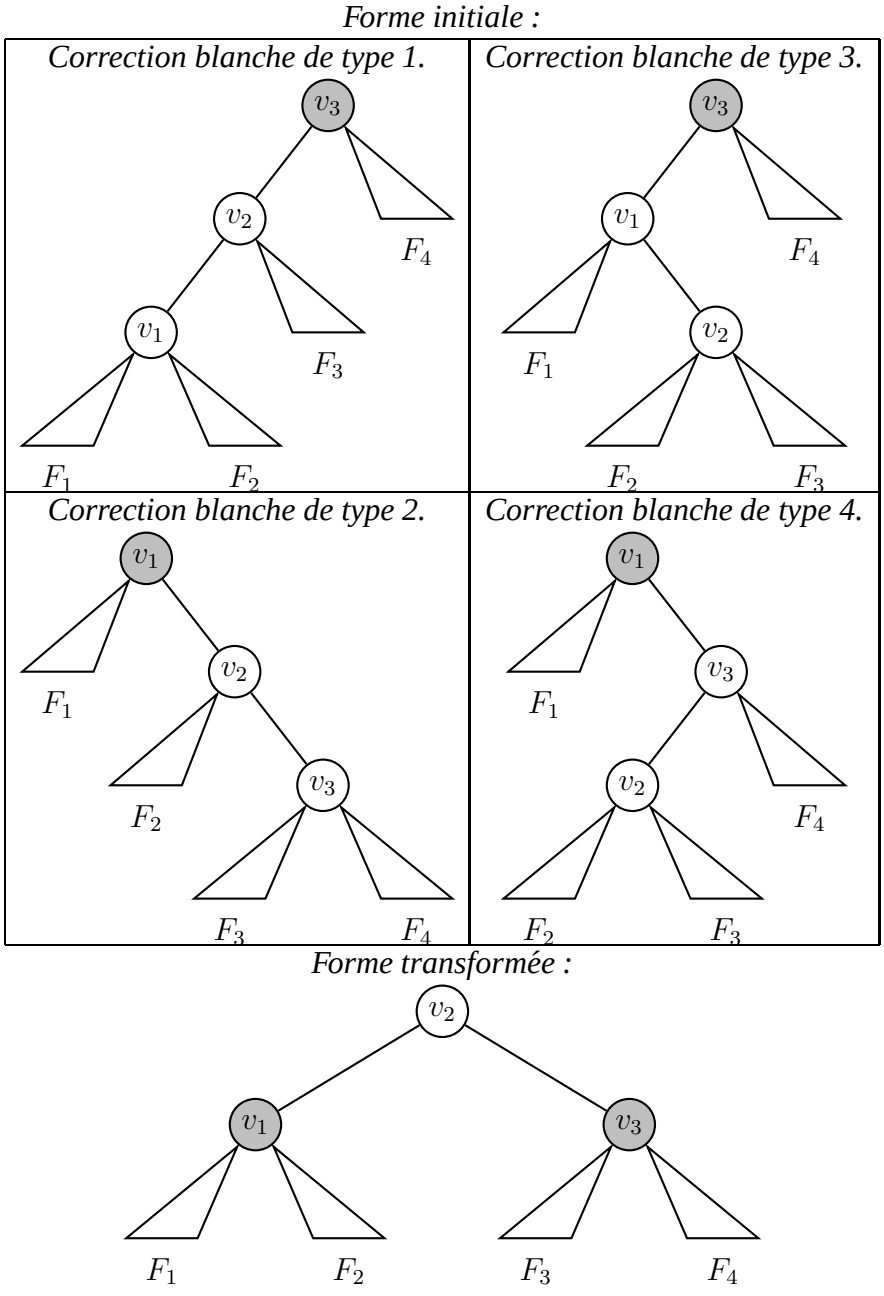
Question III.14 *Donner une estimation de la complexité dans les meilleur et pire cas de la fonction* `insertion_abr` *en fonction de la profondeur de l'arbre* `A`*. Cette estimation ne prendra en compte que le nombre d'appels récursifs effectués. Donner également une estimation de la complexité si*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

ÉCOLE POLYTECHNIQUE – ÉCOLES NORMALES SUPÉRIEURES

CONCOURS D'ADMISSION 2014 **FILIÈRE MP SPÉCIALITÉ INFO**

COMPOSITION D'INFORMATIQUE – A – (XULCR)

(Durée : 4 heures)

L'utilisation des calculatrices **n'est pas autorisée** pour cette épreuve.
Le langage de programmation choisi par le candidat doit être spécifié en tête de la copie.

⋆ ⋆ ⋆

Arbres croissants

On étudie dans ce problème la structure d'*arbre croissant*, une structure de données pour réaliser des files de priorité.

La partie I introduit la notion d'arbre croissant et la partie II les opérations élémentaires sur les arbres croissants. L'objet de la partie III est l'analyse des performances de ces opérations. Enfin la partie IV applique la structure d'arbre croissant, notamment au problème du tri.

Les parties peuvent être traitées indépendamment. Néanmoins, chaque partie utilise des notations et des fonctions introduites dans les parties précédentes. Les tableaux sont indexés à partir de 0, indépendamment du langage de programmation choisi. On note $\log(n)$ le logarithme à base 2 de n.

Arbres binaires. Dans ce problème, on considère des arbres binaires. Un arbre est soit l'arbre vide, noté $\mathtt{E}$, soit un nœud constitué d'un sous-arbre gauche g, d'un entier x et d'un sous-arbre droit d, noté $\mathtt{N}(g,x,d)$. La *taille* d'un arbre t, notée $|t|$, est définie récursivement de la manière suivante :

$$\begin{aligned} |\mathtt{E}| &= 1 \\ |\mathtt{N}(g,x,d)| &= 1+|g|+|d|. \end{aligned}$$

La *hauteur* d'un arbre t, notée $h(t)$, est définie récursivement de la manière suivante :

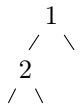
$$\begin{aligned} h(\mathtt{E}) &= 0 \\ h(\mathtt{N}(g,x,d)) &= 1+\max(h(g),\, h(d)). \end{aligned}$$

Le nombre d'occurrences d'un entier y dans un arbre t, noté $occ(y,t)$, est défini récursivement de la manière suivante :

$$\begin{aligned} occ(y,\mathtt{E}) &= 0 \\ occ(y,\mathtt{N}(g,x,d)) &= 1+occ(y,g)+occ(y,d) \quad \text{si } y=x \\ occ(y,\mathtt{N}(g,x,d)) &= occ(y,g)+occ(y,d) \quad \text{sinon.} \end{aligned}$$

L'ensemble des éléments d'un arbre t est l'ensemble des entiers y pour lesquels $occ(y,t) > 0$.

Par la suite, on s'autorisera à dessiner les arbres de la manière usuelle. Ainsi l'arbre $\mathtt{N}(\mathtt{N}(\mathtt{E},2,\mathtt{E}),1,\mathtt{E})$ pourra être dessiné sous la forme



On se donne le type `arbre` suivant pour représenter les arbres binaires.

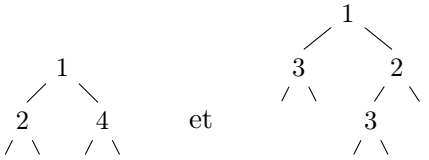
```
          (* Caml *)
type arbre =
E | N of arbre * int * arbre;;
```

```
                    { Pascal }
type arbre = ^noeud;
noeud = record gauche: arbre;
element: integer; droit: arbre; end;
```

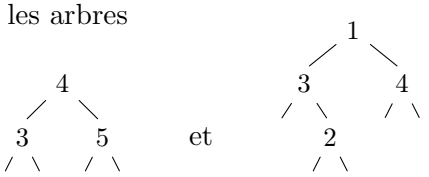
En Pascal, l'arbre vide est la constante `const E: arbre = nil` et on suppose donnée une fonction `function N(g: arbre; x: integer; d: arbre) : arbre;` pour construire le nœud $\mathtt{N}(\mathtt{g},\mathtt{x},\mathtt{d})$.

Partie I. Structure d'arbre croissant

On dit qu'un arbre t est un *arbre croissant* si, soit $t = \mathtt{E}$, soit $t = \mathtt{N}(g,x,d)$ où g et d sont eux-mêmes deux arbres croissants et x est inférieur ou égal à tous les éléments de g et d. Ainsi les arbres



sont des arbres croissants mais les arbres



n'en sont pas.

Question 1 Écrire une fonction `minimum` qui prend en argument un arbre croissant t, en supposant $t \neq \mathtt{E}$, et renvoie son plus petit élément.

```
(* Caml *)  minimum: arbre -> int
{ Pascal }  function minimum(t: arbre) : integer;
```

Question 2 Écrire une fonction `est_un_arbre_croissant` qui prend en argument un arbre t et détermine s'il a la structure d'arbre croissant. On garantira une complexité $O(|t|)$.

```
(* Caml *)  est_un_arbre_croissant: arbre -> bool
{ Pascal }  function est_un_arbre_croissant(t: arbre) : boolean;
```

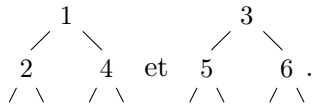
Question 3 Montrer qu'il y a exactement $n!$ arbres croissants possédant n nœuds étiquetés par les entiers $1, \ldots, n$ (chaque nœud étant étiqueté par un entier distinct).

Partie II. Opérations sur les arbres croissants

L'opération de *fusion* de deux arbres croissants t_1 et t_2, notée $\texttt{fusion}(t_1, t_2)$, est définie par récurrence de la manière suivante :

$$\begin{array}{rcll}
\texttt{fusion}(t_1, \texttt{E}) & = & t_1 & \\
\texttt{fusion}(\texttt{E}, t_2) & = & t_2 & \\
\texttt{fusion}(\texttt{N}(g_1, x_1, d_1), \texttt{N}(g_2, x_2, d_2)) & = & \texttt{N}(\texttt{fusion}(d_1, \texttt{N}(g_2, x_2, d_2)), x_1, g_1) & \text{si } x_1 \leq x_2 \\
\texttt{fusion}(\texttt{N}(g_1, x_1, d_1), \texttt{N}(g_2, x_2, d_2)) & = & \texttt{N}(\texttt{fusion}(d_2, \texttt{N}(g_1, x_1, d_1)), x_2, g_2) & \text{sinon.}
\end{array}$$

Note importante : dans la troisième (resp. la quatrième) ligne de cette définition, on a sciemment échangé les sous-arbres g_1 et d_1 (resp. g_2 et d_2). Dans les parties III et IV de ce problème apparaîtront les avantages de la fusion telle que réalisée ci-dessus (d'autres façons de réaliser la fusion n'auraient pas nécessairement de telles propriétés).

Question 4 Donner le résultat de la fusion des arbres croissants  et .

Question 5 Soit t le résultat de la fusion de deux arbres croissants t_1 et t_2. Montrer que t possède la structure d'arbre croissant et que, pour tout entier x, $occ(x, t) = occ(x, t_1) + occ(x, t_2)$.

Question 6 Déduire de l'opération `fusion` une fonction `ajoute` qui prend en arguments un entier x et un arbre croissant t et renvoie un arbre croissant t' tel que $occ(x, t') = 1 + occ(x, t)$ et $occ(y, t') = occ(y, t)$ pour tout $y \neq x$.

```
(* Caml *)  ajoute: int -> arbre -> arbre
{ Pascal }  function ajoute(x: integer; t: arbre) : arbre;
```

Question 7 Déduire de l'opération `fusion` une fonction `supprime_minimum` qui prend en argument un arbre croissant t, en supposant $t \neq \texttt{E}$, et renvoie un arbre croissant t' tel que, si m désigne le plus petit élément de t, on a $occ(m, t') = occ(m, t) - 1$ et $occ(y, t') = occ(y, t)$ pour tout $y \neq m$.

```
(* Caml *)  supprime_minimum: arbre -> arbre
{ Pascal }  function supprime_minimum(t: arbre) : arbre;
```

Question 8 Soient $x_0, \ldots, x_{n-1}$ des entiers et $t_0, \ldots, t_n$ les $n+1$ arbres croissants définis par $t_0 = \texttt{E}$ et $t_{i+1} = \texttt{fusion}(t_i, \texttt{N}(\texttt{E}, x_i, \texttt{E}))$ pour $0 \leq i < n$. Écrire une fonction `ajouts_successifs` qui

prend en argument un tableau contenant les entiers $x_0, \ldots, x_{n-1}$ et qui renvoie l'arbre croissant t_n.

```
(* Caml *)  ajouts_successifs: int vect -> arbre
{ Pascal }  function ajouts_successifs(x: array[0..n-1] of integer) : arbre;
```

Question 9 Avec les notations de la question précédente, donner, pour tout n, des valeurs $x_0, \ldots, x_{n-1}$ qui conduisent à un arbre croissant t_n de hauteur au moins égale à $n/2$.

Question 10 Toujours avec les notations de la question 8, donner la hauteur de l'arbre t_n obtenu à partir de la séquence d'entiers $1, 2, \ldots, n$, c'est-à-dire $x_i = i + 1$. On justifiera soigneusement la réponse.

Partie III. Analyse

On dit qu'un nœud $\mathtt{N}(g, x, d)$ est *lourd* si $|g| < |d|$ et qu'il est *léger* sinon. On définit le *potentiel* d'un arbre t, noté $\Phi(t)$, comme le nombre total de nœuds lourds qu'il contient.

Question 11 Écrire une fonction `potentiel` qui prend en argument un arbre t et renvoie $\Phi(t)$, tout en garantissant une complexité $O(|t|)$.

```
(* Caml *)  potentiel: arbre -> int
{ Pascal }  function potentiel(t: arbre) : integer;
```

On définit le *coût* de la fusion des arbres croissants t_1 et t_2, noté $C(t_1, t_2)$, comme le nombre d'appels récursifs à la fonction `fusion` effectués pendant le calcul de $\mathtt{fusion}(t_1, t_2)$. En particulier, on a $C(t, \mathtt{E}) = C(\mathtt{E}, t) = 0$.

Question 12 Soient t_1 et t_2 deux arbres croissants et t le résultat de $\mathtt{fusion}(t_1, t_2)$. Montrer que

$$C(t_1, t_2) \leq \Phi(t_1) + \Phi(t_2) - \Phi(t) + 2(\log |t_1| + \log |t_2|). \tag{1}$$

Question 13 Soient $x_0, \ldots, x_{n-1}$ des entiers et $t_0, \ldots, t_n$ les $n + 1$ arbres croissants définis par $t_0 = \mathtt{E}$ et $t_{i+1} = \mathtt{fusion}(t_i, \mathtt{N}(\mathtt{E}, x_i, \mathtt{E}))$ pour $0 \leq i < n$. Montrer que le coût total de cette construction est en $O(n \log(n))$.

Question 14 Montrer que, dans la construction de la question précédente, une des opérations `fusion` peut avoir un coût au moins égal à $n/2$ (on exhibera des valeurs $x_0, \ldots, x_{n-1}$ le mettant en évidence). Justifier alors la notion de complexité *amortie* logarithmique pour la fusion de deux arbres croissants.

Question 15 Soit t_0 un arbre croissant contenant n nœuds, c'est-à-dire de taille $2n + 1$. On construit alors les n arbres croissants $t_1, \ldots, t_n$ par $t_{i+1} = \mathtt{fusion}(g_i, d_i)$ où $t_i = \mathtt{N}(g_i, x_i, d_i)$, pour $0 \leq i < n$. En particulier, on a $t_n = \mathtt{E}$. Montrer que le coût total de cette construction est en $O(n \log(n))$.