Deuxième devoir non surveillé d'option informatique

Exercice

Exercice 1. * Piles et programmation impérative

On suppose données des fonctions `nouvell_pile`, `empile`, `depile` et `est_vide` permettant de manipuler eds piles mutables.

1. Écrire une procédure prenant une pile mutable `p` et un élément `x` et faisant en sorte que `p` soit privée de tous ses `x`, el autres éléments étant toujours dans leur ordre initial.
2. Modifier cette procédure pour en outre renvoyer le nombre de `x` qui ont ainsi été supprimés.

Problème : arbres bicolores

J'ai volontairement laissé (au moins) une erreur d'énoncé présente dans l'énoncé originel. Je vous laisse le plaisir de la découvrir et de la corriger, ce n'est pas si difficile ici. Ne tombez cependant pas dans l'excès de croire que dès que vous avez du mal avec une question, c'est dû à une erreur d'énoncé !

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 nœuds 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 Réalisation Caml

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

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

```
1 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'ensembles

1.2.1 Insertion dans un ensemble

1. Écrire en Caml une fonction `insertion_ens` de type `int -> ensemble -> ensemble` telle que l'appel `(insertion_ens v e)` sur un entier `v` et un ensemble `e` renvoie un ensemble contenant les mêmes éléments que `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.
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

1. 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.
2. 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.
3. Calculer une estimation de la complexité dans les meilleurs et pires 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 partir d'arbre 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

Définition 2.1. *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 $\mathcal{C}(a)$.*

L'ensemble des arbres binaires d'entiers est noté $\mathcal{B}$.

Pour tout $a \in \mathcal{B}$, on notera N_a son nombre de nœuds.

2.1.2 Profondeur d'un arbre

Définition 2.2. *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 h_a .*

Ainsi, `Vide` est de profondeur nulle, un nœud dont les deux fils sont vides est de profondeur 1. L'arbre représenté figure 1 a pour profondeur 4.

1. Donner une définition (récursive) de la profondeur d'un arbre a en fonction de $\emptyset$, $\mathcal{G}(a)$ et $\mathcal{D}(a)$.
2. Selon cette définition, quelle est la hauteur d'une feuille ?
3. Soit a un arbre binaire d'entiers de profondeur k. Montrer que $N_a \leqslant 2^k - 1$.
   On attend ici une démonstration complète par récurrence.
4. Soit $k \in \mathbb{N}$. Quel est le nombre minimal de nœuds pour un arbre de hauteur k ?
5. On fixe $n \in \mathbb{N}$.
   Montrer que la hauteur minimale d'un arbre de n nœuds est $\left\lfloor \log_2(n) + 1 \right\rfloor$.
   Indication : Soit a de hauteur minimale à n nœuds. Soit k sa hauteur. Encadrer n entre deux entiers dépendant de k (selon que le dernier étage est plein ou pas...)

2.2 Arbres bicolores

Nous considérerons par la suite des arbres binaires dont chaque nœud est décoré par une couleur blanche ou grise. *Remarque :* Historiquement, dans l'article de Guibas et Sedgwik, ces arbres étaient rouge et noirs, car le rouge était la couleur qui ressortait le mieux sur l'imprimante laser couleur (pointe de la technologie en 1978) du labo de recherche où ils travaillaient. Comme nous n'avons pas les mêmes moyens, nous remplaçons dans ce sujet le rouge par du blanc et le noir par du gris.

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

Définition 2.3. *Un arbre « coloré » est un arbre dont les nœuds sont décorés par une couleur. Un arbre « bicolore » est un arbre coloré qui respecte les contraintes suivantes :*

- ***(P1)*** *Chaque nœud a une couleur, blanc ou gris.*
- ***(P2)*** *Les deux fils d'un nœud blanc doivent être gris.*
- ***(P3)*** *Toutes les branches doivent contenir le même nombre de nœuds gris.*

Afin de respecter la condition (P2) lorsque les deux fils d'un nœud blanc sont vide, nous conviendrons que `Vide` est gris.

Attirons votre attention, pour l'étude de (P3), sur le fait qu'une branche part de la racine et termine à un $\emptyset$.

Notons qu'un même arbre binaire peut être coloré de différentes manières qui respectent les contraintes des arbres bicolores (voir la figure 1).

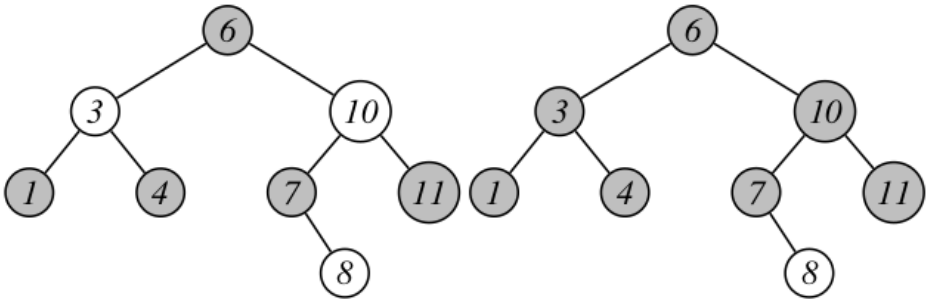

FIGURE 1 – Deux colorations valides d'un même arbre binaire d'entiers

2.2.1 Hauteur grise d'un arbre bicolore

Définition 2.4. *La hauteur grise 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 $HG(a)$.*

1. Soit a un arbre bicolore entièrement gris, de profondeur p. Montrer que son nombre de nœuds est $2^p - 1$.
2. Soit a un arbre bicolore composé de n nœuds, tel que $n > 0$. Montrer que
$$HG(a) \leqslant |a| \leqslant 2HG(a) + 1$$
3. Montrer que $2^{HG(a)} - 1 \leqslant n$.
Indication : Utiliser une récurrence sur la hauteur
4. Soit a un arbre bicolore composé de n nœuds, tel que $n > 0$. Montrer que
$$\log_2(n+1) \leqslant |a| \leqslant 2\log_2(n+1) + 1.$$

Ainsi la hauteur d'un arbre bicolore est logarithmique en son nombre de nœuds. Reste encore à voir comment on peut insérer des éléments dans un tel arbre en maintenant son caractère bicolore.

2.2.2 Représentation d'un arbre bicolore en Caml

Un arbre binaire d'entiers coloré est représenté par les types Caml :

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

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 : le terme suivant représente le premier arbre de la figure 1 :

```
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)))
```

2.2.3 Hauteur grise d'un arbre bicolore

Écrire en Caml une fonction `hauteur_grise` de type `arbre -> int` telle que l'appel `(hauteur_grise a)` sur un arbre bicolore a renvoie la hauteur grise de a. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives. Cette fonction devra avoir une complexité logarithmique en le nombre de nœuds de l'arbre.

2.2.4 Validation d'un arbre bicolore

1. Écrire en Caml une fonction `validation_bicolore` de type `arbre -> bool` telle que l'appel `(validation_bicolore a)` sur un arbre bicolore a renvoie la valeur `true` si `a` vérifie les conditions (P1), (P2) et (P3) 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. 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 Arbre binaire de recherche

Définition 2.5. *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 :*

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

Rappel : le symbole $\wedge$ signifie «et».

Notons que selon cette définition, 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 Recherche dans un arbre binaire de recherche

Écrire en Caml une fonction `appartient_abr` de type `arbre -> int -> bool` telle que l'appel `(appartient_abr a x)` sur un arbre binaire de recherche `a` et un entier `x` renvoie la valeur `true` si $x \in \mathcal{C}(a)$ et la valeur `false` sinon.

2.3.2 Validation d'un arbre binaire de recherche

Écrire en Caml une fonction `validation_abr` de type `arbre -> bool` telle que l'appel `(validation_abr a)` sur un arbre binaire d'entiers `a` renvoie la valeur `true` si `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.

2.3.3 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.

1. Écrire en Caml une fonction `insertion_abr` de type `int -> arbre -> arbre` telle que l'appel `(insertion abr v a)` sur un entier `v` et un arbre binaire de recherche `a` renvoie un arbre binaire de recherche contenant les mêmes étiquettes que `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, lorsqu'insertion il y a. 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. 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. Puis donner une estimation de la complexité en fonction de nombre de nœuds de l'arbre, distinguer alors le cas où `a` est un arbre bicolore.

2.4 Préservation de la structure d'arbre 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.

1. 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 ?

Définition 2.6. *Une correction blanche concerne 3 nœuds imbriqués de l'arbre. Elle s'applique uniquement si l'arbre possède une des quatre formes de la figure 2, son résultat est alors l'arbre de la figure 3. Un arbre qui ne possède pas la forme adéquate ne sera pas transformé.*

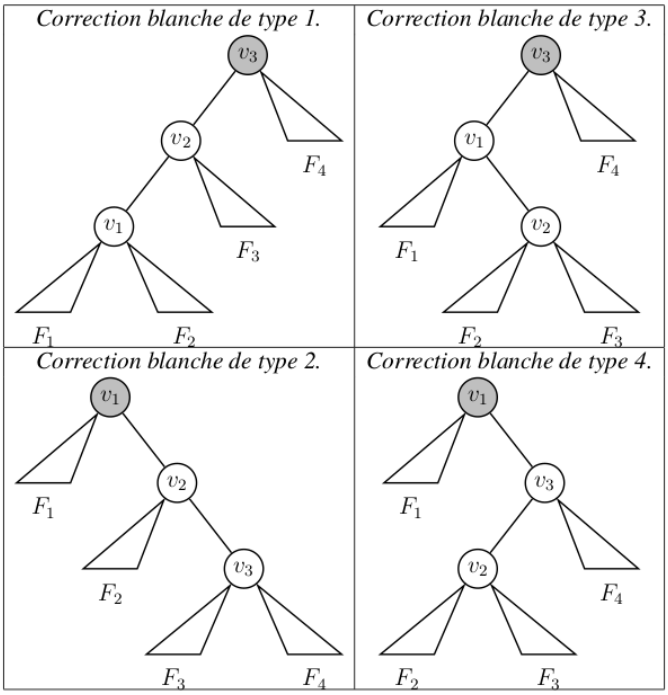

FIGURE 2 – Les quatre situations où appliquer une correction blanche

2. Montrer que si F_1 , F_2 , F_3 et F_4 sont des arbres bicolores de hauteur grise n alors le résultat d'une correction blanche est un arbre bicolore de hauteur grise $n + 1$.
3. Insérer la valeur 9 dans le premier arbre de recherche bicolore de l'exemple, puis appliquer une correction blanche autant de fois que nécessaire pour obtenir un arbre bicolore. Représenter tous les arbres colorés intermédiaires.

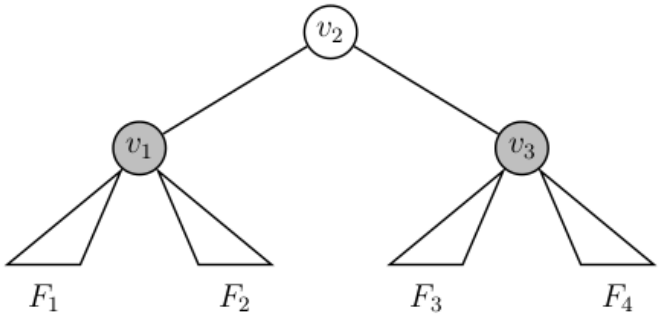

FIGURE 3 – Résultat d'une correction blanche

4. 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.
5. 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 la hauteur grise de l'arbre avant insertion $\big(|a| - HG(A)\big)$.
6. 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

1. Modifier la fonction `insertion_abr` de type `int -> arbre -> arbre` écrite en CaML à la partie 2.3.3 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.
2. Donner les spécifications de la fonction `insertion_abr` définie à la question précédente. On donnera en particulier :
   (a) le rôle des paramètres ;
   (b) les contraintes sur la valeur des paramètres ;
   (c) les caractéristiques du résultat renvoyé par la fonction ;
   (d) le principe de l'algorithme ;
   (e) des arguments de terminaison pour toutes les valeurs des paramètres vérifiant les conditions énoncées au point 2.