TP 7 : arbres binaires

Ce TP est à déposer à la fin de cette séance sur Moodle. Vous pouvez également déposer une version améliorée de votre TP jusqu'à la fin de cette semaine.

Fichier à récupérer: `tp7.mlw`

Soit le type des arbres et le type option vus en cours et en TD:

```
type tree elt = Empty | Node (tree elt) elt (tree elt)

type option 'a = None | Some 'a
```

Dans ce TP, on s'intéresse uniquement aux arbres binaires de recherche d'entiers dont tous les éléments sont distincts, vus en TD.

Pour rappel, un arbre binaire de recherche est un arbre binaire tel que, à chaque niveau de l'arbre, l'entier à la racine est supérieur à tous les entiers du sous-arbre gauche, et inférieur à tous les entiers du sous-arbre droit.

1. Compléter le prédicat `estABR` vu en TD.

```
predicate estABR (a: tree int) = (* à compléter *)
```

2. Écrire le sous-programme `rechercher_min` recherchant dans un arbre binaire de recherche son élément le plus petit (s'il existe). La spécification de ce sous-programme utilisera les prédicats `mem (x : elt) (a : tree elt)` et `occ` vus en cours et présents dans la bibliothèque des arbres.

```
val rechercher_min (a: tree int): option int (* à compléter *)
```

3. Écrire le sous-programme `rechercher_max` recherchant dans un arbre binaire de recherche son élément le plus grand (s'il existe). La spécification de ce sous-programme utilisera le prédicat `mem (x : elt) (a : tree elt)` vu en cours et présent dans la bibliothèque des arbres.

```
val rechercher_max (a: tree int): option int (* à compléter *)
```

4. Écrire le sous-programme `inserer` spécifié en TD et insérant un élément dans un arbre binaire de recherche. La spécification de ce sous-programme utilisera également le prédicat `mem`, et non pas le prédicat `contient` vu en TD.

```
val inserer (a: tree int) (k : int) : tree int (* à compléter *)
```

5. Écrire le sous-programme `supprimer_max` supprimant le plus grand élément d'un arbre binaire de recherche. Ce programme renverra la paire constituée du nouvel arbre et de l'élément ayant été supprimé.

```
val supprimer_max (a: tree int): (tree int, int) (* à compléter *)
```

6. Écrire le sous-programme `supprimer` supprimant un entier d'un arbre binaire de recherche. Ce sous-programme utilisera le sous-programme `supprimer_max`.

```
val supprimer (a: tree int) (k : int) : tree int (* à compléter *)
```