TP 16 : Implémentation des arbres binaires de recherche

Le but de ce TP est de manipuler les arbres binaires de recherche que l'on a vu en cours, et de les utiliser pour représenter des dictionnaires et des ensembles. Leur implémentation par les ABR est explicitement au programme de la MP2I : ce que le programme attend que vous connaissiez sans avoir beaucoup de contexte correspond grosso modo à l'exercice 1.

Exercice 1 – Manipulation des arbres binaires de recherche en OCaml

Dans ce TP, on utilise le type suivant :

```OCaml
type 'a abr =
  | V
  | N of 'a abr * 'a * 'a abr
```

Un fichier `tp16_eleve.ml` est disponible sur le cahier de prépas et sur les ordinateurs de la H210. Il contient ce type et l'exemple d'ABR du cours.

▶ **Question 1** Écrire deux fonctions `max_abr : 'a abr -> 'a` et `min_abr : 'a abr -> 'a` calculant la plus grande et la plus petite étiquette d'un arbre binaire de recherche. On lèvera une exception si l'arbre en entrée est vide.

▶ **Question 2** En déduire une fonction `est_abr : 'a abr -> bool` vérifiant si un arbre non vide est un arbre binaire de recherche. Quel est le problème de cette fonction ?

L'Exercice 3 propose d'implémenter efficacement cette fonction.

▶ **Question 3** Écrire une fonction `recherche : 'a -> 'a abr -> bool` déterminant si une étiquette est présente dans l'ABR. *On impose qu'un appel à `recherche x a` entraîne au plus $h(a) + 1$ appels récursifs, ce qui interdit notamment de chercher l'étiquette dans l'arbre entier.*

★ **Question 4** Montrer par induction sur les ABR que votre fonction est correcte : elle renvoie bien `true` si et seulement si l'étiquette est présente dans l'ABR.

On considérera que si l'on insère une étiquette déjà présente dans un ABR, on renvoie l'ABR non modifié.

▶ **Question 5** Implémenter une fonction `insere : 'a -> 'a abr -> 'a abr` insérant un élément dans un ABR.

★ **Question 6** Écrire une fonction `abr_of_list : 'a list -> 'a abr` qui, depuis une liste d'éléments `l`, renvoie l'arbre dans lequel on a successivement inséré les éléments de `l`.

★ **Question 7** En déduire une fonction `tri : 'a list -> 'a list` qui renvoie la liste en entrée triée en calculant l'arbre binaire de recherche associé à l'entrée par `abr_of_list`, puis en parcourant cet arbre binaire de recherche par un parcours infixe.

★ **Question 8** Déterminer sa complexité dans le meilleur et dans le pire cas.

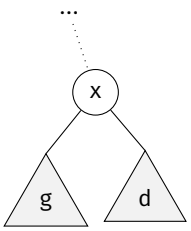
Exercice 2 – Suppression d'une étiquette dans un arbre binaire de recherche

Une opération importante dans les ABR est la suppression d'étiquette : étant donné une étiquette x et un ABR A, renvoyer un nouvel ABR contenant les mêmes étiquettes que A sauf x.
Pour cet exercice, je vous conseille **vivement**, si ce n'est pas déjà fait, de sortir un papier et un crayon pour réfléchir à une façon d'implémenter la suppression d'une étiquette d'un arbre binaire de recherche. On va se baser sur le squelette de code suivant :

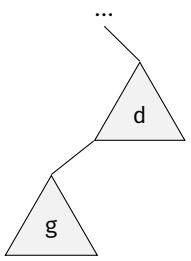
OCaml
```
1 let rec supprime a x = match a with
2   | V -> ...
3   | N (V, e, d) when e = x -> ...
4   | N (g, e, V) when e = x -> ...
5   | N (g, e, d) when e < x -> ...
6   | N (g, e, d) when e > x -> ...
7   | N (g, e, d)            -> ... (* e = x *)
```

▶ **Question 1** Compléter les lignes 2 à 6.

Dans le cas de la ligne 7, où l'on a trouvé le nœud contenant `e = x` avec deux fils non vides, c'est plus compliqué. On cherche à créer un nouvel arbre qui ne contient que les étiquettes de `g` et `d`.



▶ **Question 2** Implémenter la stratégie qui consiste à mettre `g` tout à gauche de `d`, de manière à obtenir un arbre de la forme ci-contre. Justifier que l'on obtient bien un ABR. *On complètera la ligne 7 et on prendra soin de tester cette fonction pour vérifier qu'elle fait bien ce que l'on attend.*



★ **Question 3** Est-ce que vous voyez un désavantage à cette stratégie ?

Une stratégie similaire ajouterait `g` tout à droite de `d`. On propose une autre stratégie :

- d'abord, on remarque que l'étiquette minimale `m` de l'arbre `d` qui ne peut pas avoir de fils gauche,
- supprimer `m` de l'arbre `d` tombe donc dans le cas ligne 3, on calcule alors `d'` l'ABR obtenu après suppression de `m` dans `d`,
- on se rend compte que l'arbre `N (g, m, d')` est bien un arbre binaire de recherche qui convient !

▶ **Question 4** Implémenter cette stratégie dans une nouvelle fonction `supprimer2 : 'a abr -> 'a -> 'a abr`. *On pourra écrire une fonction auxiliaire* `supprime_min : 'a abr -> 'a * 'a abr` *tel que* `supprime_min a` *est le couple contenant l'étiquette minimale de* `a` *et un ABR contenant le reste des étiquettes.*

▶ **Question 5** En général, a-t-on `insere (supprime2 a x) x = a` ?

Exercice 3 – ★ D'autres fonctions à implémenter de manière la plus efficace possible

▶ **Question 1** Écrire une fonction `separe : 'a abr -> 'a -> 'a abr * 'a abr` tel que `separe a x` renvoie un couple d'ABR contenant toutes les étiquettes strictement inférieures à x d'un côté, et toutes les étiquettes supérieures ou égales à x de l'autre. *Un nombre d'appels récursifs linéaire en la hauteur est attendu.*

▶ **Question 2** Écrire une fonction linéaire en le nombre de nœuds d'un arbre vérifiant qu'il est bien un arbre binaire de recherche : `est_abr : 'a abr -> bool`.

Les arbres binaires de recherche permettent de représenter un *ensemble* d'éléments comparables. Il est également possible de représenter un *multiensemble*, c'est-à-dire un ensemble dans lequel un élément peut apparaître plusieurs fois. On utilise les mêmes opérations que les ensembles, sauf qu'un élément peut y être présent plusieurs fois : alors, supprimer un élément de l'ensemble n'en supprime qu'une occurrence.

▶ **Question 3** Écrire un type `'a multiset` représentant un multiensemble d'éléments de type `'a` comme un arbre binaire de recherche où à chaque nœud est associé une étiquette supplémentaire entière représentant le nombre d'occurrences de l'élément.

▶ **Question 4** Écrire la fonction `nb_occurences : 'a multiset -> 'a -> int` renvoyant le nombre d'occurrences d'un élément dans le multiensemble.

▶ **Question 5** Écrire une fonction `'a multiset -> 'a -> int` donnant la taille du multiensemble (c'est-à-dire son nombre d'éléments en comptant leur multiplicité).

▶ **Question 6** Écrire une fonction `nth_plus_petit : 'a multiset -> int -> 'a` tel que `nth_plus_petit d k` renvoie la k^e plus petite valeur de d en comptant les multiplicités.