Exercice 1 : Tableaux associatifs immuables avec des ABR

On définit le type `('a, 'b) abr`, correspondant aux arbres binaires de recherche étiquettés par des clés de type `'a`, avec une valeur de type `'b` associée à chaque clé, ainsi :

```
type ('a, 'b) abr = Vide | Noeud of ('a, 'b) abr * 'a * 'b * ('a, 'b) abr
```

On suppose que le type `'a` est totalement ordonné (et on utilise les opérateurs usuels pour les comparaisons) et immuable.

1. On remarque que la définition ne garantit pas que la propriété des ABR est vérifiée.
   (a) Écrire deux fonction `min : ('a, 'b) abr -> 'a` et `max : ('a, 'b) abr -> 'a` qui, étant donné A ABR non vide qui vérifie bien la définition vue en cours, calculent respectivement la plus petite et la plus grande clé de A.
   Ces fonctions déclencheront une erreur si A est vide. On ne définit pas le comportement si A ne vérifie pas la définition d'un ABR.
   (b) Écrire une fonction `est_abr : ('a, 'b) abr -> bool` qui vérifie qu'un arbre de tyep `abr` correspond bien à la définition d'un arbre binaire de recherche.
   (c) Quelle est l'ordre de grandeur de la complexité de votre fonction dans le pire cas ?

   Dans toutes les questions suivantes, on suppose de tout valeur de type `abr` passée en argument respecte la définition d'un ABR. Notamment, on n'appellera **pas** la fonction `est_abr` pour le vérifier.
2. Écrire les fonctions de spécification suivante :
   (a) `mem : ('a, 'b) abr -> 'a -> bool` qui, étant donné un ABR A et une clé k, indique si k est une clé de A.
   (b) `find : ('a, 'b) abr -> 'a -> 'b` qui, étant donné un ABR A et une clé k présente dans A, renvoie la valeur associée à k dans A.
   Si k n'est pas dans A, cette fonction déclenchera une erreur.
   (c) `replace : ('a, 'b) abr -> 'a -> 'b -> ('a, 'b) abr` qui, étant donné un ABR A, une clé k et une valeur v, renvoie un ABR A' identique à A sauf que k y est associée à v.
   Si k n'est pas dans A, A' contient donc un nœud de plus que A, alors que si k est déjà dans A, A' contient le même nombre de nœuds mais l'étiquette du nœud de clé k est différente.
   (d) `remove : ('a, 'b) abr -> 'a -> ('a, 'b) abr` qui, étant donné un ABR A et une clé k, renvoie un ABR A' identique à A sauf que la clé k n'y est pas.
   Si k n'est pas dans A, alors A' est égal à A.
3. Tester sur des exemples simples les fonctions précédentes, en utilisant des chaînes de caractères comme clés et des entiers comme valeurs.
4. On veut montrer que la complexité de `remove` est linéaire en la hauteur dans le pire cas :
   (a) Montrer que si k est une clé de A et que le nœud de clé k dans A a au plus un enfant, alors dans ce cas la complexité de la fonction est au plus linéaire en la hauteur de A.
   (b) En déduire que la complexité de la fonction est au plus linéaire en la hauteur de A dans le pire cas.

Exercice 2 : Comparaison avec les tables de hachage

Le module `Hashtbl` en OCaml permet d'implémenter, à l'aide d'une table de hachage, les tableaux associatifs mutables :

- `('a, 'b) Hashtbl.t` désigne le type d'un tableau associatif de clés de type `'a` et de valeurs de type `'b`.

  Ocaml autorise un type mutable pour `'a` **mais ceci est une très mauvaise idée.**

  Le type `'a` doit permettre les comparaisons, ce ne peut donc pas être un type fonctionnel.
- `Hashtbl.create : int -> ('a, 'b) Hashtbl.t` qui crée une nouvelle table de hachage, vide, dont on précise en argument la capacité initiale (c'est à dire la place initialement occupée en mémoire). On peut toujours choisir une petite capacité initiale car la capacité sera augmentée selon le besoin.
- `Hashtbl.mem : ('a, 'b) Hashtbl.t -> 'a -> bool` indique si une clé est présente.
- `Hashtbl.find : ('a, 'b) Hashtbl.t -> 'a -> 'b` renvoie la valeur associée à une clé si elle est présente, et déclenche une erreur sinon.
- `Hashtbl.replace : ('a, 'b) Hashtbl.t -> 'a -> 'b -> unit` :
  `replace d k v` modifie d pour contenir l'association (k, v). Si k était déjà une clé de d, alors la valeur associée est remplacée par v, sinon l'association (k, v) est ajoutée.

  Il est déconseillé d'utiliser `Hashtbl.add` qui a un comportement ne correspondant pas à la définition vu en cours (associations multiples pour une seule clé).
- `Hashtbl.remove : ('a, 'b) Hashtbl.t -> 'a -> unit` :
  `remove d k` modifie d en retirant (si elle existe) l'association de k dans d.

1. Écrire une fonction `abr_to_hashtbl : ('a, 'b) abr -> ('a, 'b) Hashtbl.t` qui prend en argument un ABR et met son contenu dans une table de hachage.
2. Tester votre fonction de conversion, et vérifiez que vous trouvez les mêmes résultats avec les fonction `find` et `mem` que sur l'implémentation avec les ABR.
3. Quels sont les avantages et les inconvénients de ces deux implémentations ?