TD 4 - Arbres rouge-noir

Retrouvez tous les énoncés et les corrections des TP sur ma page personnelle :

`http://perso.ens-lyon.fr/hadrien.croubois/`

Étude de complexité des différentes structures de données

Question 1

Quels structures de donnés connaissez vous ? Quels sont leurs points forts et leurs points faibles ?

Question 2

Remplissez, autant que possible, le tableaux suivant en utilisant la notation $O(\dots)$. Si possible faite la distinction entre complexité moyenne et complexité dans le pire cas.

	Recherche	Insertion	Suppression	Minimum	Extraction du min	Fusion
Liste						
Liste trié						
ABR						
AVL						
Red-Black Tree						
Tas *	/	$O(\log n)$	$O(\log n)$	$O(1)$	$O(\log n)$	$O(m\ log(n+m))$
Tas de Fibonnaci *	/	$O(1)$	$O(\log n)$	$O(1)$	$O(\log n)$	$O(1)$

* Hors programme.

Les arbres rouges noirs

Un arbre rouge et noir est un arbre binaire de recherche donc chaque nœud contient une information supplémentaire, sa couleur, qui peut valoir soit rouge soit noir.

En contrôlant la manière dont les nœuds sont colores on peut contenir la différence de longueur entre les branches (chemins de la racines aux feuilles) dans un facteur 2 et ainsi garantir un certain équilibre dans l'arbre.

Nous représenterons les arbres rouge et noir grâce aux types suivants :

```
type color =
    Red
  | Black
;;
type 'a tree =
    Empty
  | N of color * 'a tree * 'a * 'a tree
;;
```

On supposera que l'on dispose d'une relation d'ordre sur le type `'a`. Un arbre est dit de recherche si et seulement si pour tout nœud étiqueté par l'élément x, tous les éléments présents dans le sous-arbre gauche sont inférieurs à x et tous ceux du sous-arbre droit sont supérieurs a x. Un arbre binaire de recherche est un arbre rouge et noir s'il satisfait les deux propriétés supplémentaires suivantes :

1. Si un nœud est rouge alors ses fils sont noirs.
2. Chaque chemin simple reliant un nœud à une feuille contient le même nombre de nœuds noirs.

Question 3 :

Ecrivez une fonction `mem` qui cherche si un élément est présent dans un arbre rouge et noir.

```
val mem :  'a → 'a tree → bool
```

Vérification de la structure

Question 4 :

Ecrivez une fonction `check_1` qui vérifie qu'un objet de type `'a tree` vérifie la propriété (1), i.e. que tout fils

d'un nœud rouge est noir.

```
val check_1 :  'a tree → bool
```

La hauteur noire $hn(t)$ d'un arbre rouge et noir t est le nombre de nœuds noirs présents dans un chemin quelconque descendant de sa racine à n'importe quelle feuille.

Question 5

Écrivez une fonction `black_height` qui calcule la hauteur noire d'un objet de type `'a tree`. Dans le cas ou cette hauteur n'est pas définie (parce que l'arbre ne vérifie pas la propriété (2)), vous lèverez une exception.

```
val black_height :  'a tree → int
```

Question 6

Déduisez-en une fonction `check_2` qui vérifie qu'un objet de type `'a tree` vérifie bien la propriété (2), i.e. que tout chemin simple reliant un nœud à une feuille descendante contient le même nombre de nœuds noirs.

```
val check_2 :  'a tree → bool
```

Insertion

Le principe de l'insertion d'un élément x dans un arbre rouge et noir est le suivant. On effectue tout d'abord une insertion en procédant comme pour un arbre binaire de recherche simple. Un nœud est donc crée à la place d'une feuille, à une position choisie de manière à respecter l'ordre entre éléments. On colorie ce nouveau nœud en rouge, ce qui permet de préserver la propriété (2).

Ainsi, après cette première étape, seule la propriété (1) peut être violée : il se peut en effet que le père du nœud introduit soit lui aussi rouge ! L'idée est alors de remonter la structure de l'arbre en réarrangeant astucieusement les nœuds et leurs couleurs de manière à remonter ce conflit et à le faire disparaître, tout en maintenant la propriété (2).

On procède de la manière suivante (si nécessaire) :

- Si l'oncle du nœud courant est rouge, marquer le père et l'oncle du nœud courant comme noir, marquer le grand père comme rouge, et répéter l'opération sur le grand père.
- Sinon, si le nœud courant est un fils droit, faire une rotation gauche de son père.
- Sinon, marquer son père comme noir et son grand père comme rouge et procéder a une rotation a droite du grand père.

Question 7

Écrivez une fonction `conflict` qui prend pour argument un objet de type `'a tree` . S'il s'agit d'un conflit, votre fonction réarrangera l'arbre comme indiqué ci-dessus. Dans tous les autres cas, elle rendra l'arbre inchangé.

```
val conflict :  'a tree → 'a tree
```

Suppression

Question 8 (Difficile)

Rappelez la méthode de suppression d'un nœud dans un ABR. Comment pourrait-on l'adapter au cas des arbres rouge-noir ?

Question 9

Implémentez tous les outils nécessaire a la suppression d'un nœud dans un arbre rouge-noir.