Algorithmique des structures arborescentes
Contrôle continu, TP noté
Jeudi 13 avril 2017 — Durée 1h20

Mode d'emploi

Créez un fichier `nom-prenom.ml` contenant les définitions et explications demandées, et envoyez-le en fin de séance par courrier à vos enseignants (lien) avec comme sujet du message : **TP ASDA 13/4/17**.

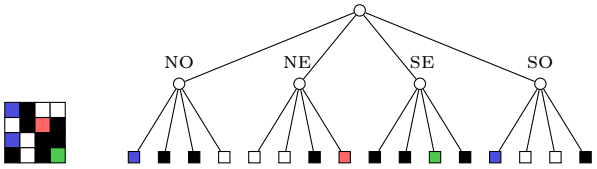
Attention ! Les **noms des fonctions**, l'**ordre et le type de leurs arguments** doivent être respectés, et le fichier ne doit **PAS** contenir d'erreur de syntaxe ou de typage. Dans le cas contraire, votre réponse sera comptée comme nulle.

Exercice 1 : Arbres quadrants

On utilise les arbres pour représenter des images **carrées** de taille 2^n (la taille est le nombre de pixels de l'image).

- Si $n = 0$, l'image contient 1 pixel dont on représente la couleur par un triplet d'entiers (r, g, b).
- Sinon, on divise l'image en 4 quarts : Nord-Ouest, Nord-Est, Sud-Est et Sud-Ouest, et on représente chaque sous-image de taille 2^{n-1} récursivement.

Ce découpage conduit à une représentation des images par des arbres dont les nœuds internes ont 4 fils, représentant les sous-images Nord-Ouest, Nord-Est, Sud-Est et Sud-Ouest. Par exemple, l'image carrée à gauche de la figure suivante est représentée par l'arbre à droite de l'image.



1. Définir un type `color` représentant une couleur composée de 3 entiers.
2. Définir un type `qtree` permettant de représenter ces arbres.
   Rappel :
   - les feuilles contiennent une couleur de type `color`,
   - les nœuds internes ont 4 fils.
3. Écrire une fonction `color_count : qtree -> color -> int` telle que `color_count t c` calcule le nombre de feuilles de l'arbre `t` qui ont la couleur `c`.
4. Écrire une fonction `clockwise : qtree -> qtree` telle que `clockwise t` calcule l'arbre représentant la rotation de 90 degrés dans le sens des aiguilles d'une montre de l'image représentée par `t`.
   Conseil. Réfléchissez à une écriture ***récursive*** de cette rotation.
5. Écrire une fonction `lr_sym : qtree -> qtree` telle que `lr_sym t` calcule l'arbre qui représente la symétrie par rapport à l'axe vertical de l'image représentée par `t`.

Exercice 2 sur l'autre page.

Exercice 2 : k-AVL

On utilise le type suivant pour les arbres binaires :

```
type 'a btree = Leaf of 'a | Node of 'a btree * 'a * 'a btree
```

On adopte les définitions suivantes :

— une *branche* d'un arbre binaire est un chemin reliant une feuille à la racine,

— la *longueur* d'une branche est le nombre d'arêtes de la branche,

— la *hauteur* d'un arbre est la longueur maximale d'une branche.

La hauteur de l'arbre `Leaf(7)` est donc 0. Celle de l'arbre `Node(Leaf 0,4,Node(Leaf 1,7,Leaf 2))` est 2.

Pour un entier $k \geqslant 0$, un ***k-AVL*** est un arbre binaire ayant la propriété suivante :

> Pour ***tout*** nœud x de l'arbre, les hauteurs des deux sous-arbres de x ***diffèrent au plus de k***.

1. Définir un arbre `t1` de type `btree` qui ***n'est pas*** un 1-AVL.

Pour faciliter l'écriture des fonctions, on veut ajouter dans chaque nœud d'un arbre binaire la *hauteur* du sous-arbre en ce nœud. On définit un type OCaml `htree` pour prendre en compte cette information :

```
type 'a htree = HLeaf of 'a | HNode of 'a htree * 'a * int * 'a htree
```

On appelle les arbres de type `'a htree` des *arbres enrichis*.

2. Écrire une fonction `height : 'a htree -> int` telle que `height t` renvoie la $3^{\text{ème}}$ composante (de type `int`) de la racine de `t`, c'est-à-dire la composante supposée représenter la hauteur de `t`.
3. Écrire une fonction `make_htree : 'a htree -> 'a -> 'a htree -> 'a htree` telle que l'appel `make_htree left x right` renvoie l'arbre composé du sous-arbre gauche `left`, du sous-arbre droit `right` et de la clé `x` à la racine, en supposant la composante *hauteur* correctement calculée pour `left` et `right`.
4. Écrire une fonction `enrich : 'a btree -> 'a htree` telle que `enrich t` renvoie l'arbre enrichi obtenu à partir de `t` en ajoutant en chaque nœud de l'arbre la hauteur du sous-arbre en ce nœud.
5. Écrire une fonction `is_k_avl : int -> 'a btree -> bool` telle que `is_k_avl k t` renvoie `true` si `t` est un `k`-AVL, et `false` sinon.
6. On rappelle qu'un arbre parfait est un arbre dont toutes les feuilles sont à la même profondeur. En utilisant la question 5, écrire une fonction `is_perfect : 'a btree -> bool` telle que `is_perfect t` renvoie `true` si `t` est parfait et `false` sinon.

La question suivante n'est pas une question de programmation. Donnez votre réponse en commentaire dans le fichier (la notation pour les commentaires en OCaml est `(*...*)`).

7. Pour $h \geqslant 0$, on note $m(h)$ le nombre ***minimal*** de nœuds que peut contenir un 1-AVL de hauteur h.
   (a) Que vaut $m(0)$?
   (b) Trouver une relation de récurrence liant $m(h)$, $m(h-1)$ et $m(h-2)$.
   (c) En déduire que $m(h)+1 \geqslant 2(m(h-2)+1)$.
   (d) En déduire par récurrence sur h que $m(h) \geqslant 2^{h/2}$, où $h/2$ désigne le quotient de h par 2.
   (e) Montrer que la hauteur d'un 1-AVL est $O(\log n)$, où n est le nombre de nœuds du 1-AVL.
   (f) Généraliser le résultat précédent : pour k fixé, un k-AVL à n nœuds a une hauteur $O(\log n)$.

Remarque culturelle. Lorsqu'on utilise les AVL comme arbres binaires de recherche, on peut utiliser des rotations après une insertion ou une suppression pour rétablir la propriété d'être un AVL.