# BE de programmation fonctionnelle
# Session 2

1h30 : avec documents

Année 2019-2020

## Préambule

— Le code rendu doit impérativement compiler. Pour cela, les fonctions non implémentées peuvent être remplacées par un code quelconque, par exemple **let** evalue **=** **fun** _ −> **assert** false .
— Vous devez tout écrire dans le fichier `be.ml`.
— Les noms et types des fonctions doivent être respectés (tests automatiques).
— Pour tester dans `utop` vous devez ouvrir le module Be (**open** Be;;).
— La non utilisation d'itérateur sera pénalisée ainsi que l'utilisation inutile d'accumulateurs.
— Les exercices sont indépendants.

## 1 Suite de Syracuse

La suite de Syracuse d'un nombre entier $N > 0$ est définie par récurrence, de la manière suivante :

$u_0 = N$
$\forall n \in \mathbb{N} : u_{n+1} =$
$u_n/2$ si $u_n$ est pair
$3 * u_n + 1$ si $u_n$ est impair

La conjecture de Syracuse est l'hypothèse mathématique selon laquelle la suite de Syracuse de n'importe quel entier strictement positif atteint 1.

On définit alors :

— le temps de vol en altitude : plus petit indice $n$ tel que $u_{n+1} < u_0$.
— l'altitude maximale : valeur maximale de la suite.

▷ **Exercice 1**

— *Écrire la fonction* syracuse*, qui pour un entier* $n$ *renvoie sa suite de Syracuse. On supposera la conjecture vraie et on arrêtera le calcul au premier 1.*

— *Écrire la fonction* tempsVolsAltitude*, qui calcule le temps de vol en altitude de la suite de Syracuse d'un entier.*

— *Écrire la fonction* altitudeMax*, qui calcule l'altitude maximale de la suite de Syracuse d'un entier.*

## 2 Arbres bicolores

Issu de Wikipédia :

Un arbre bicolore, ou arbre rouge-noir est un type particulier d'arbre binaire de recherche équilibré. Chaque nœud de l'arbre possède en plus de ses données propres un attribut binaire qui est souvent interprété comme sa "couleur" (rouge ou noir). Cet attribut permet de garantir l'équilibre de l'arbre : lors de l'insertion ou de la suppression d'éléments, certaines propriétés sur les relations entre les nœuds et leurs couleurs doivent être maintenues, ce qui empêche l'arbre de devenir trop déséquilibré, y compris dans le pire des cas. Durant une insertion ou une suppression, les nœuds sont parfois réarrangés ou changent leur couleur afin que ces propriétés soient conservées.

Le principal intérêt des arbres bicolores réside dans le fait que malgré les potentiels réarrangements ou coloriages des nœuds, la complexité (en le nombre d'éléments) des opérations d'insertion, de recherche et de suppression est logarithmique.

Un arbre bicolore est un arbre binaire de recherche dans lequel chaque nœud a un attribut supplémentaire : sa couleur, qui est soit rouge soit noire. En plus des restrictions imposées aux arbres binaires de recherche :

1. chaque nœud possède une valeur, telle que chaque nœud du sous-arbre gauche ait une valeur inférieure ou égale à celle du nœud considéré
2. chaque nœud du sous-arbre droit possède une valeur supérieure ou égale à celle-ci

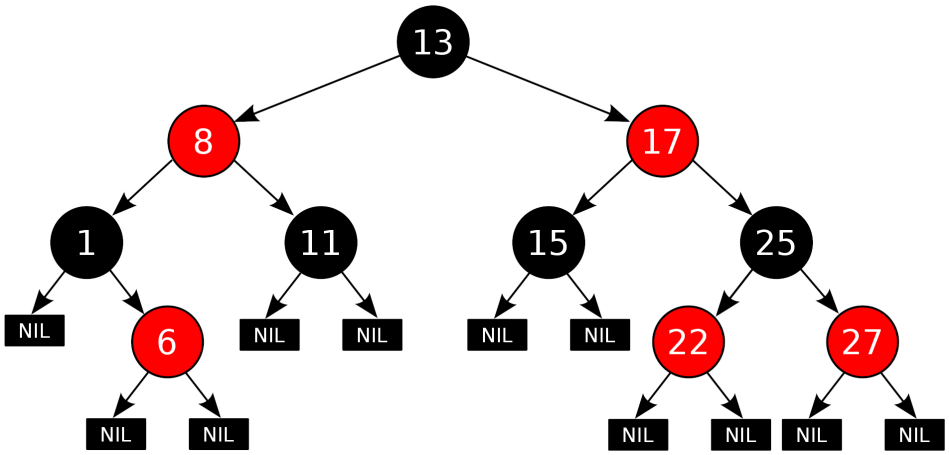
Les règles suivantes sont utilisées :

1. Un nœud est soit rouge soit noir ;
2. La racine est noire ;
3. Les enfants d'un nœud rouge sont noirs ;
4. Tous les nœuds ont 2 enfants. Ce sont d'autres nœuds ou des feuilles NIL, qui ne possèdent pas de valeur et qui sont les seuls nœuds sans enfants.

Leur couleur est toujours noire et rentre donc en compte lors du calcul de la hauteur noire.

5. Le chemin de la racine à n'importe quelle feuille (NIL) contient le même nombre de nœuds noirs. On peut appeler ce nombre de nœuds noirs la hauteur noire.

Voici un exemple d'arbre bicolore :




Le type 'a arbreBicolore vous est donné ainsi que quelques instances d'arbres bicolores, dont celui du sujet ( arbreSujet ).

▷ **Exercice 2** *Nous souhaitons vérifier qu'une instance de type* 'a arbreBicolore *est bien un arbre bicolore.*

- *Écrire le contrat, les tests unitaires et le code de la fonction* estABR *qui vérifie si une instance de* 'a arbreBicolore *valident les propriétés d'un arbre binaire de recherche.*
- *Écrire une fonction* filsRougeNoirs *qui vérifie que les enfants d'un nœud rouge sont noirs.*
- *Écrire une fonction* hauteurNoire *qui renvoie la haute noire de l'arbre, et qui lève l'exception* HauteurNoireException *si tous les chemins de la racine à n'importe quelle feuille (NIL) ne contiennent pas le même nombre de nœuds noirs.*
- *Écrire une fonction* estArbreBicolore *qui vérifie si une instance de* 'a arbreBicolore *valident les propriétés d'un arbre bicolore. Vous mettrez en commentaire la complexité en fonction du nombre de valeur dans l'arbre de votre fonction et commenterez (peut-elle être améliorée ?) cette complexité.*