TD numéro 5 : arbres binaires

À préparer avant la séance

• Écrire une fonction booléenne qui teste si une valeur appartient à un arbre.

• Écrire une fonction qui retourne la liste résultant du parcours infixe d'un arbre : en chaque nœud, on parcourt le fils gauche, puis on note la valeur du nœud, puis on parcourt le fils droit.

• Écrire une fonction qui, étant donné un arbre de nombres, remplace les valeurs négatives par 0.

À faire pendant la séance

• Écrire une fonction qui calcule la hauteur d'un arbre, définie ainsi : la hauteur d'un arbre est 1 + le maximum des hauteurs des sous-arbres (gauche, droit), la hauteur d'une feuille étant zéro.

• Écrire une fonction qui calcule le minimum d'un arbre binaire contenant des nombres.

• Écrire une fonction qui, étant donné un arbre de nombres A, ajoute des feuilles à A. La valeur de chacune de ces nouvelles feuilles contient la somme des valeurs sur le chemin de la racine aux (anciennes) feuilles.

• Écrire une fonction qui, étant donné un arbre de nombres, renvoie un arbre dont la valeur des nœuds est une liste de deux nombres représentant le nombre d'éléments pairs et impairs présents dans les sous-arbres et lui-même.

TD numéro 6 : arbres binaires de recherche

On considère maintenant des arbres dont les valeurs sont des nombres, et pour lesquels les nœuds sont ordonnés : en tout nœud, le sous-arbre gauche contient des nombres plus petits que celui du nœud, et le sous-arbre droit des nombres plus grands.

À préparer avant la séance

• Écrire une fonction booléenne qui teste si une valeur appartient à un arbre binaire de recherche.

• Écrire une fonction qui insère un nombre dans un arbre binaire de recherche.

À faire pendant la séance

• Écrire une fonction booléenne qui teste qu'un arbre binaire est un arbre binaire de recherche.

• Écrire une fonction qui trie une liste de nombres en passant par la construction d'un arbre binaire de recherche.

• Donner la spécification de la fonction mystère ci-dessous :

```
(define mystere
  (lambda (a x) ; a ABR, x nb
    (cond ((vide? a)'())
          ((= (valeur a) x) (list (valeur a)))
          ((< x (valeur a)) (cons (valeur a) (mystere (fils-g a) x)))
          (else (cons (valeur a) (mystere (fils-d a) x))))))
```