AVL

Dans ce TD, on va s'intéresser des arbres binaires de recherche spécifiques, que sont les arbres AVL, découverts par G. Adelson-Velsky et E. Landis.

Dans cet exercice, on notera $h(T)$ la hauteur de l'arbre T. Si T_g est le fils gauche de T et T_d le fils droit de T, on notera $balance(T) = h(T_d) - h(T_g)$ la balance de l'arbre T.

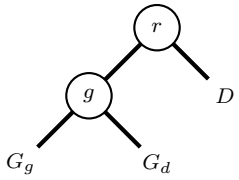

Figure 1: Quelques nœuds d'un arbre binaire

1. **Manipulations**

   a. Effectuer respectivement une rotation droite sur la racine de l'arbre donné en Figure 1, puis une rotation gauche sur la racine de l'arbre résultant.

   b. On suppose dans cette question que les sous-arbres D, G_g et G_d de l'arbre donné en Figure 1 sont équilibrés.

      (i) Si $|balance(g)| \leqslant 1$, que dire du sous-arbre enraciné en g ?

      On suppose désormais que $|balance(g)| \leqslant 1$ et que $balance(r) = -2$. Donner les relations entre hauteurs des sous-arbres, dessiner les différents cas possibles et effectuer les rotations nécessaires afin d'équilibrer l'arbre donné en Figure 1 dans le cas où :

      (ii) $balance(g)$ vaut 0
      (iii) $balance(g)$ vaut -1.
      (iv) $balance(g)$ vaut 1. On pourra séparer G_d en une racine g_d et deux sous-arbres G_{dg} et G_{dd}.
      (v) Calculer la balance de chacun des sous-arbres de l'arbre obtenu après la (ou les) rotations(s) effectuée(s).

   c. Construire l'AVL obtenu en insérant successivement les entiers suivants : 7, 12, 25, 16, 21, 13, 2, 5, 4, 9, 30, 27, 6, 11. On dessinera l'arbre obtenu après chaque ajout.

   d. À partir de l'AVL construit précédemment, on supprime successivement les entiers suivants, tout en respectant la structure d'AVL: 30, 27, 16. Quel arbre obtient-on ? On dessinera l'arbre obtenu après chaque suppression.

2. **Programmation** — Dans cet exercice, on utilisera les types C suivants :

```
typedef struct noeud{
  int valeur;         /* étiquette */
  int balance;        /* balance */
  struct noeud *fg;   /* fils gauche */
  struct noeud *fd:   /* fils droit */
} Noeud, *AVL;
```

a. Écrire une fonction `int EstAVL(AVL a)` qui renvoie `1` si l'arbre binaire de recherche a est un arbre binaire de recherche équilibré (*on suppose que* `a` *a été construit indépendemment; le champs* `balance` *n'est pas rempli*), et renvoie `0` sinon.

b. Écrire une fonction `void RotationD(AVL *a)` qui effectue une rotation droite de l'arbre `*a`. On n'oubliera pas de rectifier la balance des nœuds concernés.

c. Écrire une fonction `void RotationG(AVL *a)` qui effectue une rotation gauche de l'arbre `*a`.

d. Écrire une fonction `void RotationDoubleG(AVL *a)` qui effectue sur l'arbre `*a` une rotation gauche sous le fils gauche, suivie d'une rotation droite sous la racine. C'est la rotation effectuée dans l'exercice précédent si la `balance` du nœud `g` vaut `1`.

e. Écrire une fonction `void Equilibre(AVL *a)` qui choisit et appelle les fonctions de rotation adéquates afin d'équilibrer l'arbre `*a`. On pourra supposer que les sous-arbres de `*a` sont des arbres AVL.

f. Écrire une fonction `void Insertion(AVL *a, int x)` qui effectue l'insertion de l'entier `x` dans l'AVL `*a`. On n'oubliera pas d'effectuer les changements de balance et les rotations nécessaires afin de conserver la struture d'arbre AVL.

g. Écrire une fonction `AVL Extraction(AVL *a, int x)` qui effectue la suppression de l'entier `x` dans l'AVL `*a` et renvoie l'adresse du nœud extrait, ou bien `NULL` en cas d'échec.