TD7 : Arbres binaires équilibrés

Exercice 1 – Arbres binaires de recherche équilibrés

Dans le support de cours, toutes les fonctions C correspondant à l'insertion dans un ABR ne sont pas données. Cet exercice a pour but de fournir cette implémentation. Afin de ne pas reprendre à l'identique les fonctions fournies dans le cours, on choisit ici de passer les arbres en paramètres et non en retour des fonctions. On considère, dans la suite de l'énoncé, des arbres binaires de recherche équilibrés contenant des entiers. On commence avec la structure de données suivante :

```
1 typedef struct noeud {
2     int valeur;
3     struct noeud * fg;
4     struct noeud * fd;
5 } Noeud;
6
7 typedef Noeud* ABR;
```

Q 1.1 Écrire une fonction `Noeud* rechercherValeur(ABR ab, int val);` qui recherche si un élément `val` est dans l'arbre et retourne un pointeur sur l'élément, ou NULL sinon.

Q 1.2 Écrire une fonction `void insereElem_ss_eq(ABR* ab, int val);` qui insère un élément dans l'ABR sans effectuer d'équilibrage.

Q 1.3 Donner l'arbre binaire de recherche obtenu en ajoutant itérativement les entiers de 1 à 6 dans un arbre vide (sans équilibrage). Rappeler pourquoi il est important d'avoir des arbres équilibrés.

Q 1.4 Pour équilibrer un arbre binaire efficacement, il faut parvenir à connaître rapidement la hauteur d'un noeud. Expliquer comment modifier la structure pour pouvoir faire cela. Écrire une fonction `AB_hauteur` qui étant donné un arbre binaire, retourne sa hauteur. Écrire une fonction `void majHauteur(ABR ab);` qui met à jour la hauteur d'un ABR si au moins l'un de ses fils a été modifié.

Q 1.5 Écrire une fonction `void rotationDroite(ABR* ab);` qui permet de réaliser une rotation droite d'un arbre binaire (la figure 1 présente une telle rotation).

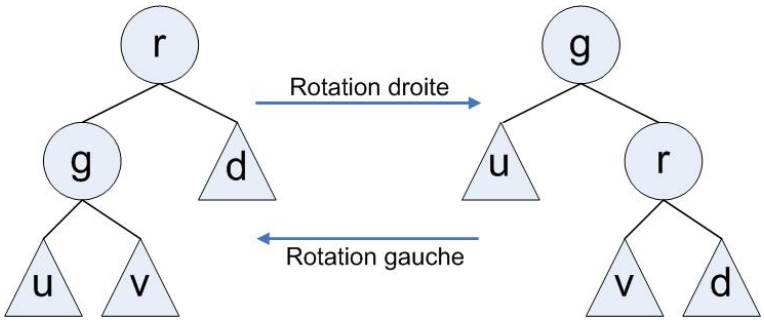


FIGURE 1 – Rotation droite et rotation gauche dans un arbre binaire

Q 1.6 Écrire une fonction `AB_rotationGauche` qui permet de réaliser une rotation gauche d'un arbre binaire.

Q 1.7 Écrire une fonction `void insererElem_avec_eq(ABR * ab, int v);` qui étant donnés un ABR équilibré et une valeur, retourne l'arbre équilibré obtenu en ajoutant la valeur à l'arbre. Notons A l'arbre obtenu après insertion sans équilibrage, et G et D ses sous-arbres gauche et droite respectivement. On rééquilibre A de la manière suivante :

- Si $|h(G) - h(D)| < 2$, aucune réorganisation n'est nécessaire.
- Si $h(G) - h(D) = 2$, alors notons g et d les sous-arbres gauche et droite de G respectivement. Si $h(g) < h(d)$, alors on effectue une rotation gauche de G. Dans tous les cas, on effectue ensuite une rotation droite de A.
- Si $h(G) - h(D) = -2$ alors notons g et d les sous-arbres gauche et droite de D respectivement. Si $h(d) < h(g)$, alors on effectue une rotation droite de D. Dans tous les cas, on effectue ensuite une rotation gauche de A.

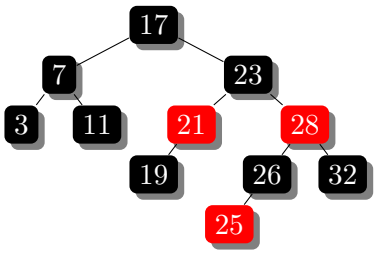
Q 1.8 Donner l'arbre binaire de recherche obtenu quand on ajoute les nombres entiers de 1 à 6 dans un arbre binaire vide, avec équilibrage.

Exercice 2 – Arbre Rouge-Noir

Les Arbres Binaires de Recherche (ABR) dans leur version équilibrée (AVL) ont des propriétés très intéressantes mais certaines opérations peuvent demander un nombre important de rotations (notamment pendant une suppression). Pour pallier à ces inconvénients, d'autres structures de données ont été proposées, comme par exemple les arbres rouge-noir. Un arbre rouge-noir (ARN) :

- est un arbre binaire de recherche,
- dont les nœuds sont colorés en noir ou en rouge,
- la racine étant colorée en noir,
- les nœuds fils d'un nœud rouge (s'ils existent) étant colorés en noir,
- et le nombre de nœuds noirs sur tous les chemins de la racine à une feuille est constant.

Dans cet exercice, on appellera "chemin" dans l'arbre tout chemin de la racine à une feuille de l'arbre. La figure ci-dessous représente un ARN (les seuls sommets rouges sont les sommets 21, 25 et 28).



Q 2.1 Montrer que dans un ARN, on ne peut avoir deux nœuds rouges successifs le long d'un chemin. En quoi cette propriété aide-t-elle à interdire des ARN trop déséquilibrés ?

Q 2.2 Définir une structure permettant de représenter un nœud d'ARN.

Q 2.3 Écrire une fonction C permettant de calculer le nombre de nœuds rouges dans un ARN.

Q 2.4 Écrire une ou des fonctions C permettant de vérifier qu'un `ARNtree` vérifie bien les propriétés d'un arbre rouge-noir.

Q 2.5 Proposer un algorithme d'insertion dans un ARN. Insérer 18 puis 24 dans l'ARN de la figure.

Q 2.6 Quelle est la complexité de cette fonction ? Voyez-vous une modification nécessaire dans la structure de données de ARNtree ?