POLYTECH SORBONNE

Algorithmique générale
MAIN-3, année 2022
Séance TP N°4
Avril 2022
VOYLES Evan

Objectif

☞ Arbres binaires équilibrés et balisés.

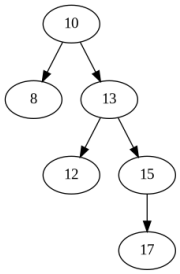
Problème [Arbres équilibrés balisés]

La documentation générale pour ce travail se trouve ici, la documentation technique se trouve là, et voici le répertoire git.

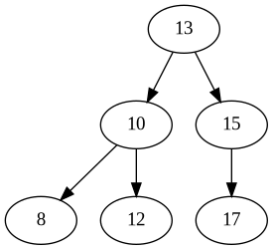
Préliminaires : Structure Arbre AVL

J'ai choisi d'implémenté un arbre AVL pour avoir une ABR qui s'équilibre après chaque insertion. La définition de cette structure est nommé `Node` dans le code écrit pour ce TP.

Afin d'implémenter un arbe qui s'équilibre, on aura besoins de quelques fonctions qui transforment un arbre binaire de recherche. Les deux qu'on va en servir sont des rotations dites "à gauche" et "à droite". Dans le code on parle des fonctions `rotLeft` et `rotRight`. Examinons ce qui se passe après une rotation.



(a) un arbre binaire de recherche non équilibré



(b) Après rotation à gauche autour du noeud 13

FIGURE 1 – Un ABR qui se transforment en ARBE. On remarque qu'on a changé les facteurs d'équilibre de les noeuds pour que chaque noeurd vérifié l'invariant $\epsilon(T) \in \{-1, 0, 1\}$, ou $\epsilon(T)$ est la facteur de déséquilibre d'un sousarbre.

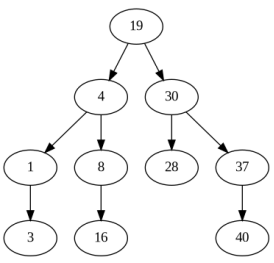
Vérifiant cet invariant, on dit que l'arbre dans Figure 1b est *équilibré*. On rajoute la transformation de rotation à gauche ce qui est en effet l'inverse de l'opération de rotation à droite. Pour aborder l'insertion d'un arbre AVL, on s'en sert de plusiers fonctions d'utilité comme `addKeyBST`, une fonction pour insérer un élément dans un ABR ordinaire. On aura des autres fonctions telles que `updateAVL` qui calcule la facteur de déséquilibre de chaque noeud, en mettant à jours leurs hauteur. Donc, une insertion à un arbre AVL consiste simplement d'une insertion dans un ABR et ensuite une étape de réequilibration. Étudions la création d'un arbre AVL à partir des éléments dans l'ensemble : $\{1, 4, 8, 3, 19, 30, 28, 37, 16, 40\}$.

```
Node *tex2 = newNode(1);

tex2 = addKeyAVL(tex2, 4);
tex2 = addKeyAVL(tex2, 8);
tex2 = addKeyAVL(tex2, 3);
tex2 = addKeyAVL(tex2, 19);
tex2 = addKeyAVL(tex2, 30);
tex2 = addKeyAVL(tex2, 28);
tex2 = addKeyAVL(tex2, 37);
tex2 = addKeyAVL(tex2, 16);
tex2 = addKeyAVL(tex2, 40);

createDotBST(tex2, "tex_2.dot");
```

(a) Code pour insérer des éléments dans un arbre AVL



(b) ABRE résultant

Partie A : Structure Arbre Balisé

1. implémenter une procédure `versAbreBalise` permettant de transformer une ABRE, donnée en argument, en un ARBE-balisé. Cette procédure ne retourne aucune valeur ! Quelle est la complexité de votre procédure ?

On note que n'importe quel ABR a n noeuds et $n+1$ feuilles externes. Pour convertir un ABRE à un ABRE-balisé, il s'agit de stocker tous les clés dans les feuilles externes, dans l'ordre croissant. Alors pour définir la fonction `toBalise`, on commence par écrire un mécanisme pour

- ☞ avoir une liste des clés dans l'ordre croissant
- ☞ avoir une liste des noeuds avec des feuilles externes
- ☞ mettre les clés dans l'ordre croissant dans les feuilles externes, aussi dans l'ordre croissant.

Toute cette fonctionalité est implémentée dans la fonction `toBalise`.

2. Implémenter la procédure `inserer` qui permet d'insérer un nouveau noeud dans un ARBRE-balisé.

Ce comportement est implémenté par `insertBalise`. On considère le fait que tous les noeuds sont contenu dans leus feuilles internes d'un ARBE-balisé, ce qui simplifie énormement le logique du code pour le traiter.

3. Implémenter la procédure `supprimer` qui permet de supprimer un noeud.

La fonction `deleteBalise` implémente cette fonctionalité.

Partie B : Structure Arbre Balisé