TP n°21 - Équilibrage des ABR - Part 1 - Rotations

Définition 1 (Arbre binaire équilibré)

Soit $\mathcal{S} \in \mathcal{T}$ un ensemble d'arbres binaires. On dit que les arbres de S sont équilibrés s'il existe une constante c telle que, pour n'importe quel arbre $t \in \mathcal{S}$, on a :

$$h(t) \leq c \log_2(n(t)+1) - 1$$

avec notre convention sur la hauteur $h(E) = -1$.

Remarque. On peut prendre n'importe quel logarithme dans la définition ci-dessus.
c est la même constante pour TOUS les arbres de $\mathcal{S}$.
Comme on a toujours $h(t) \geq \lceil \log_2(n(t)+1) \rceil - 1$, cela revient à dire que les arbres de $\mathcal{S}$ ont une hauteur qui évolue de manière logarithmique avec leur nombre de nœuds : si j'insère des nœuds, je le ferai de manière suffisamment compacte, en évitant tout phénomène de peigne.

Soit $\mathcal{S}$ un ensemble d'arbres binaires construits selon un certain procédé. Garantir que le arbres de $\mathcal{S}$ sont équilibrés permet d'être certain que tous les arbres générés selon ce procédé auront des hauteurs contrôlées, évoluant de manière logarithmique avec le nombre de nœuds.

Attention. Pour les ABR, tous les accesseurs et transformateurs ont une complexité temporelle directement en lien avec la hauteur de l'arbre. Si les ABR générés sont équilibrés, on sera alors assurés que toutes ces opérations s'effectueront avec une complexité au pire logarithmique par rapport au nombre de nœuds de l'arbre binaire.

L'analyse de l'équilibre des ABR est donc essentielle pour la maîtrise de la complexité temporelle des algorithmes utilisant cette structure de données.

Commençons par étudier l'équilibre des ABR générés avec notre méthode de construction actuelle.

Exercice 1 (Analyse de l'équilibre d'ABR générés aléatoirement sans rééquilibrage).

1. Récupérer le code `bst-balance-analysis-init.c` sur Moodle, ranger le correctement dans votre arborescence de fichiers dans un nouveau dossier `TP21`. Renommez-le `NOM_bst-balance-analysis.c`
2. Écrire une fonction `bst *bst_create_random(int n_nodes)` qui génère un ABR en insérant successivement `n_nodes` valeurs aléatoires choisies entre 0 et `RAND_MAX`.
3. Écrire une fonction `void write_barchart(char *path, int n_samples, int n_nodes)` qui calcule le diagramme en bâtons des hauteurs pour `n_samples` arbres de `n_nodes` nœuds générés aléatoirement avec la fonction précédente. Cette fonction écrit les valeurs de ce diagramme en bâtons dans un fichier de chemin `path` au format CSV.
   On utilisera une virgule comme séparateur pour le format CSV. On testera que l'écriture du fichier est correcte avec de petites valeurs de `n_samples` et `n_nodes`. **Consigne : c'est le code C qui doit calculer les valeurs du diagramme en bâtons, pas le script Python qui va suivre ! !. Indication pour le calcul du diagramme : on a vu quelle était la pire (la plus grande) hauteur possible pour un arbre binaire...**
4. Écrire un script python `NOM_plot_barchart.py` qui permet de représenter le diagramme en bâtons des valeurs à partir du fichier CSV généré par votre code C. *Indication : on pourra utiliser la fonction `bar` du package Python `matplotlib.pyplot`*
5. Adapter le code C et le script Python pour que le diagramme en bâtons généré indique également la hauteur optimale (la plus faible) que l'on pourrait obtenir pour un arbre à `n_nodes` nœuds.
6. Faire des tests, analyser les diagrammes en bâtons obtenus et conclure.

Définition 2 (Arbre AVL (Addison-Velsky and Landis))

On définit inductivement la notion d'arbre binaire AVL :

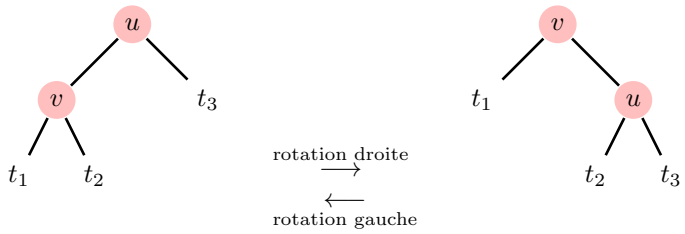
- L'arbre vide est AVL ;
- Un arbre $N(\ell, x, r)$ est AVL si :
  - ℓ est AVL
  - r est AVL
  - les hauteurs des sous-arbres ℓ et r diffèrent d'au plus 1 :

$$|h(r) - h(\ell)| \leq 1$$

Exercice 2 (Les arbres AVL sont équilibrés).

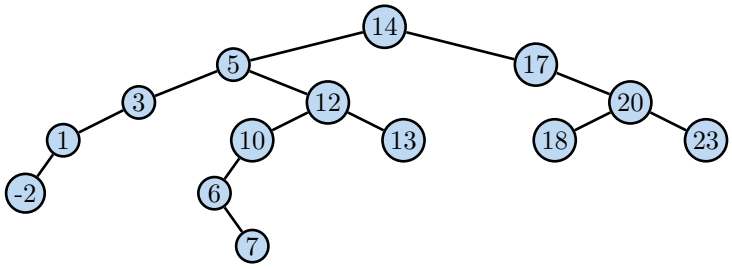
1. Soit $\mathcal{A}$ l'ensemble des arbres AVL. Montrer qu'il existe une constante $c > 1$ telle que, pour tout $t \in \mathcal{A}$, on a :
$$c^{h(t)+1} - 1 \leq n(t) \leq 2^{h(t)+1} - 1$$
Que vaut c ?
2. En déduire que les arbres AVL sont équilibrés au sens de la première définition.
3. Donner la plage de valeurs de hauteur pour des arbres AVL à 1000 nœuds puis à 100000 nœuds. Quelle aurait été la borne supérieure de cette plage de valeurs sans la propriété AVL ?

On peut donc essayer d'équilibrer nos ABR en tentant des modifications qui permettent de les rendre AVL.
Pour cela, nous allons utiliser deux opérations de modification (transformateurs) : la rotation gauche autour d'un nœud, et la rotation droite autour d'un nœud :

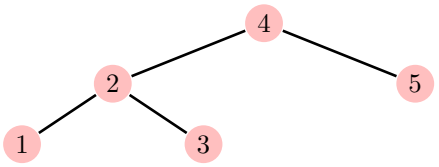


Exercice 3 (Opérations de rotation (gauche ou droite) autour d'un nœud).

1. Au considère l'ABR de la figure ci-dessous (c'est le même arbre que celui du premier TP sur les ABR !). Dessiner l'arbre obtenu après une rotation droite autour du nœud d'étiquette 10.



2. A partir de l'arbre modifié, faire à présent une rotation gauche autour du nœud d'étiquette 17 de l'arbre donné ci-dessus.
3. Faire à présent une rotation droite autour du nœud d'étiquette 14.
4. Montrer que les opérations de rotation gauche et droite conservent la propriété d'ABR (et vérifiez le sur vos manipulations précédentes !)
5. Copier le code `NOM_bst-balance-analysis.c` dans un nouveau fichier `NOM_bst-avl.c`. Commenter le `main` et créer un nouveau `main` vide.
6. Dans le fichier `NOM_bst-avl.c`, implémenter la fonction `void rotate_right(bst *n)` effectuant une rotation du nœud d'adresse `n`. **On procédera en échangeant les étiquettes des nœuds** u **et** v et en mettant à jour les liens entre les nœuds. Pensez à mettre à jour les parents ! Pensez à bien tester tous les cas possibles pour chaque rotation : pratiquez une programmation défensive et sure !
7. Tester cette fonction dans votre nouveau `main`, sur le petit cas ci-dessous en tournant autour de la racine :



8. Même chose pour `void rotate_left(bst *n)`.
9. Vérifier, en codant des tests en dur dans le `main`, que les deux transformateurs implémentés sont bien réciproques l'un de l'autre.
10. Tester sur l'exemple travaillé sur les 3 premières questions de cet exercice.