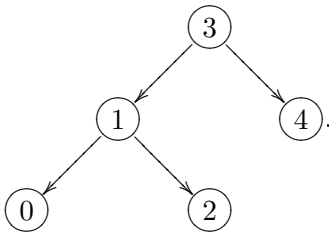
Licence Informatique **TD Algorithmique 3**
Université Bordeaux 1

Arbres binaires de recherche par position

Rappels La *taille* d'un arbre est le nombre de ses nœuds. La *position* d'un nœud dans un arbre de taille t est le rang de visite de ce nœud, compris entre 0 et $t-1$, relativement au *parcours infixe* de l'arbre. Ce parcours consiste à :
– parcourir récursivement le sous-arbre gauche issu de la racine ;
– visiter la racine ;
– parcourir récursivement le sous-arbre droit issu de la racine.
En particulier, la position de *la racine d'un arbre* est la taille du sous-arbre gauche qui en est issu. Exemple :



Un *arbre binaire de recherche* (ABR) est un arbre binaire dont tout nœud n possède une *clé* vérifiant la propriété suivante : la clé de n est strictement supérieure à la clé de tout nœud du sous-arbre gauche issu de n, et strictement inférieure à la clé de tout nœud du sous-arbre droit issu de n.

Nous considérons ici des arbres binaires de recherche *par position* : la clé de tout nœud y est définie par sa position. Le but de cet exercice est de montrer qu'une séquence éditable peut être implémentée par un tel arbre.

Exercice 1. - Séquences, arbres et motifs Pour chaque "motif proposé" (en dernière page), donnez une suite d'entiers insérés dans un arbre qui le produit. Cette suite est-elle unique ? Peut-on la (les) caractériser ?

(Lorsque les sous-arbres gauches et droits ne sont pas dessinés, ils sont vides.)

Exercice 2. - Signatures des méthodes d'insertion, de recherche et de supression Donner la signature des fonctions récupérer, insérer et supprimer.

Exercice 3. - Algorithmes des méthodes d'insertion, de recherche et de supression Nous supposons l'arbre vide représenté par nil, et un arbre non vide par un nœud (sa racine) à quatre champs : données, taille, gauche et droite.

– Proposer un algorithme réalisant la fonction récupérer. Une erreur doit être signalée si la position p donnée ne respecte pas la contrainte $0 \leq p < t$ où t désigne la taille de l'arbre.

– Proposer un algorithme réalisant la fonction insérer. Une erreur doit être signalée si la position p donnée ne respecte pas la contrainte $0 \leq p \leq t$ où t désigne la taille de l'arbre. Quel intérêt présente l'utilisation d'un champ taille plutôt que d'un champ position ?
– Proposer un algorithme réalisant la fonction supprimer. Une erreur doit être signalée si la position p donnée ne respecte pas la contrainte $0 \leq p < t$ où t désigne la taille de l'arbre.

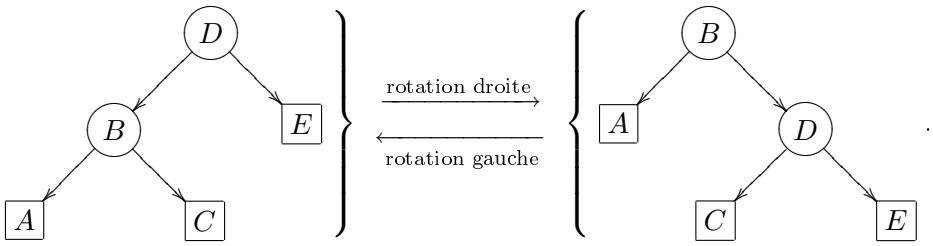
Rappels La *distance* d'un sommet à la racine de l'arbre est la longueur de l'unique chemin qui va de la racine à ce sommet.

La *hauteur* d'un arbre non vide est la longueur (en nombre d'arêtes) du plus long chemin de sa racine vers l'une des ses feuilles. En particulier, la hauteur d'un arbre réduit à un nœud (sa racine) est 0. Par convention, nous supposerons que l'arbre vide a pour hauteur -1. La hauteur de l'arbre illustré en page précédente est 2.

Exercice 4. - Coût en moyenne des opérations d'insertion et de recherche On supposera que la somme des distances de tous les sommets à la racine de l'arbre est $\sim 2n \log n$. (La démonstration de ces propriétés est difficile.)

En déduire la complexité en moyenne des algorithmes d'insertion et de recherche dans un arbre binaire de recherche.

Exercice 5. - Arbres d'*Adelson-Velsky & Landis* (AVL) Un arbre (AVL) est un ABR dont tout nœud est tel que les hauteurs des sous-arbres gauche et droit qui en sont issus diffèrent au plus de 1. Le maintien de cette condition d'*équilibre* par insertion et suppression est assuré par l'utilisation d'opérations de *rotation* gauche et droite :



Nous supposons qu'un nœud possède désormais un cinquième champ : hauteur.

Proposer un algorithme réalisant la fonction rotation-droite (on supposera la fonction rotation-gauche définie « symétriquement »). Modifier les algorithmes d'insertion et de suppression afin de maintenir l'équilibre des arbres par rotation.

Exercice 6. - Tests logiques Analyser les algorithmes ci-dessus et indiquer pour chaque test (`si` / `sinon`) s'il est :

– presque toujours vrai ;
– presque toujours faux ;
– vrai à chaque itération sauf la dernière ;
– faux à chaque itération sauf la dernière ;
– presque aléatoirement vrai ou faux.