TP n° 4

TAD ArbreBinaire

L'objectif de cette séance de travaux pratiques est d'implémenter en Java le TAD ArbreBinaire vu en travaux dirigés, ainsi que certaines fonctions usuelles.

Exercice 1 – Implémentation de l'Arbre Binaire

✎ Dans un nouveau projet Eclipse, ou dans un nouveau paquetage d'un projet existant, créer une nouvelle classe `Arbrebinaire`.

✎ Proposer une implémentation en Java du type `ArbreBinaire` et de ses opérateurs : 'arbre_vide', 'assembler', 'est_vide', 'racine', 'sag' et 'sad'. Comme pour les structures précédentes, le type Élément sera implémenté par la classe Java `Object`.
Note importante : On considérera qu'un arbre vide a une racine nulle.

Exercice 2 – Opérations usuelles sur les arbres binaires

✎ Dans une nouvelle classe `Expression` écrire le programme permettant d'obtenir l'arbre décrit dans la figure ci-dessous.

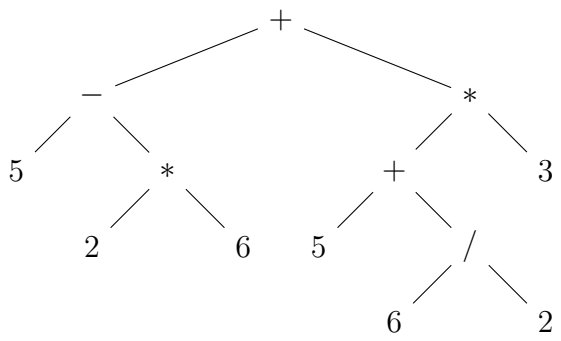


FIGURE 1 – Arbre binaire décrivant une expression algébrique.

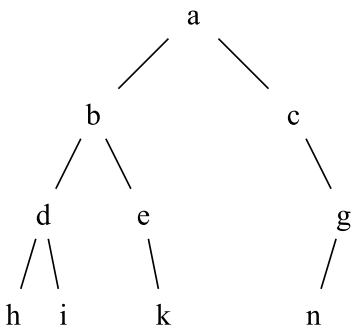
✎ Écrire la méthode `taille` de la classe `ArbreBinaire` qui retourne sous forme d'entier le nombre de noeuds de l'arbre binaire. Vérifier que la taille de l'arbre de l'exercice 1 est 13.

✎ Écrire la méthode `hauteur` de la classe `ArbreBinaire` qui retourne le nombre de noeuds du plus long chemin de la racine vers la feuille la plus profonde (ou la plus haute) de l'arbre binaire. Vérifier que la hauteur de l'arbre de l'exercice 1 est 5.

✎ Dans la classe `Expression`, écrire la fonction `évaluer` qui évalue de manière récursive l'expression codée par l'arbre binaire. Vérifier que l'expression vaut 17.

Exercice 3 – Implémentation contiguë

Dans cet exercice, nous proposons une implémentation contigüe (i.e. non récursive) de l'arbre binaire. Le principe est illustré dans la figure ci-dessous. Les noeuds de l'arbre sont rangés dans un tableau, la racine étant stockée à l'indice 0. La relation entre en père et son fils gauche (et respectivement entre un père et son fils droit) sont exprimés par la relation mathématique de leurs indices.



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
a	b	c	d	e		g	h	i		k			n	

✎ Déterminer la relation mathématique entre l'indice de chaque noeud et ceux de son fils gauche, de son fils droit, et de son père (s'ils existent).

✎ Proposer une implémentation du type `ArbreBinaire` à partir d'un tableau de caractères. L'initialisation de l'arbre se fera dans le constructeur en passant un tableau de caractères.

✎ Proposer une implémentation des opérateurs 'racine', 'sag' et 'sad'. Vu que la représentation n'est plus récursive, on devra pour ces opérateurs passer en paramètre l'indice dans la tableau de la racine de l'arbre (exemples : 0 pour l'arbre de racine 'a', 1 pour l'arbre de racine 'b', 10 pour l'arbre de racine 'k').

✎ Proposer une implémentation des fonctions `taille` et `hauteur`. Calculer la taille et la hauteur de l'arbre 'a' (taille=10, hauteur=4), ainsi que des sous-arbres 'b' (taille=6, hauteur=3) et 'k' (taille=1, hauteur=1).

✎ Implémenter la méthode `toString` qui permet de parcourir l'arbre en largeur et qui renvoie une chaîne de caractères composée des nœuds (caractères) de l'arbre. Dans l'exemple fourni ci-dessus, le parcours en largeur renvoie la chaîne `"abcdeghikn"`.