Arbres binaires (Binary Trees)

1 Mesures

Exercice 1.1 (Taille (Size))

1. Donner les axiomes définissant l'opération *taille* sur le type abstrait `arbre binaire`.
2. Écrire une fonction qui calcule la taille d'un arbre binaire.

Exercice 1.2 (Hauteur (Height))

1. Donner les axiomes définissant l'opération *hauteur* sur le type abstrait `arbre binaire`.
2. Écrire une fonction qui calcule la hauteur d'un arbre binaire.

Exercice 1.3 (Longueurs de cheminement et Profondeurs moyennes)

1. Soit les fonctions suivantes :

```
fonction frec(arbrebinaire B, entier h,  ref entier n) : entier
debut
    si B = arbre-vide alors
        retourne 0
    sinon
         n ← n + 1
         retourne h + frec(g(B), h+1, n) + frec(d(B), h+1, n)
    fin si
fin

fonction fun(arbrebinaire B) : reel
variables
   entier  nb
debut
    si B = arbre-vide alors
         /* Exception */
    sinon
        nb ← 0
        retourne (frec (B, 0, nb) / nb)
    fin si
fin
```

   (a) Quel sera le résultat de la fonction `fun` appliquée à l'arbre de la figure 1 ?

   (b) Quelle mesure calcule cette fonction ?

2. Écrire une fonction qui calcule la longueur de cheminement externe d'un arbre binaire.
3. Que faut-il modifier à cette fonction pour qu'elle calcule la profondeur moyenne externe ?

2 Parcours

Exercice 2.1 (Parcours profondeur (Depth-first Traversal))

1. En considérant un parcours en profondeur main gauche de l'arbre de la figure 1 donner la liste des nœuds pour chacun des trois ordres induits.

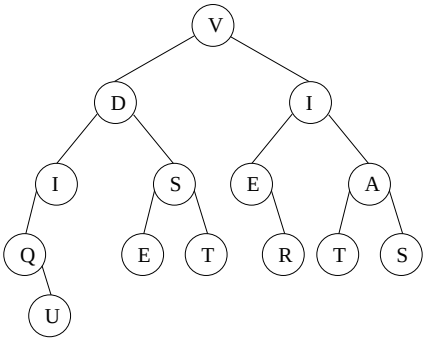

FIGURE 1 – Arbre binaire pour parcours

2. Donner l'arbre 1 sous la forme $<r, G, D>$ avec _ pour représenter l'arbre vide.
3. Écrire une fonction qui affiche un arbre binaire sous la forme $<r, G, D>$, avec _ pour représenter l'arbre vide.

Exercice 2.2 (Parcours largeur (Breadth-first Traversal))

1. Dérouler l'algorithme du parcours largeur sur l'arbre de la figure 1.
2. Écrire une fonction qui calcule la largeur d'un arbre binaire.

---

3 Applications

Exercice 3.1 (Cherche le chemin)

1. Écrire une fonction qui recherche une valeur x dans un arbre binaire. La fonction retournera l'arbre contenant x en racine si la recherche est positive, la valeur `None` sinon.
2. La fonction précédente ne permet pas de connaître le chemin pour accéder à x depuis la racine. Écrire une version de la recherche qui retourne ce chemin.

Autre solution ?

Exercice 3.2 (Père et fils)

La classe `BinTreeParent` ci-dessous permet de représenter les arbres binaires avec en chaque nœud les liens vers les fils, mais également le lien vers le père.

```
1    class BinTreeParent:
2        def __init__(self, key, parent, left, right):
3            self.key = key
4            self.parent = parent
5            self.left = left
6            self.right = right
```

Écrire une fonction qui construit à partir d'un arbre binaire "classique" (`BinTree`) un arbre binaire équivalent (contenant les mêmes valeurs aux même places) mais avec le père renseigné en chaque nœud (`BinTreeParent`).

Exercice 3.3 (Dégénéré, parfait ou complet)

1. **Arbre dégénéré (*degenerate*) :**
   (a) Rappeler ce qu'est un arbre dégénéré.
   (b) Comment tester si un arbre est dégénéré si on connaît sa taille et sa hauteur ?
   (c) Écrire une fonction qui détermine si un arbre binaire est dégénéré (sans utiliser `hauteur` et `taille`).
2. **Arbre complet (*perfect*) :**
   (a) Donner plusieurs définitions d'un arbre complet.
   (b) Comment tester si un arbre est complet en utilisant sa taille et sa hauteur ?
   (c) Écrire une fonction qui teste si un arbre est complet (vous pouvez utiliser une seule fois la fonction qui calcule la `hauteur`).
   (d) Écrire à nouveau l'algorithme de test sans utiliser `hauteur`.
3. **Arbre parfait (*complete*) :**
   (a) Rappeler ce qu'est un arbre parfait.
   (b) Comment modifier les algorithmes qui testent si un arbre est complet pour qu'ils testent si l'arbre est parfait ?

Exercice 3.4 (Tree serialization)

Voici la liste des valeurs en ordre préfixe de rencontre lors du parcours profondeur de l'arbre du sujet précédent :

```
1  L_tutoPref = ['V', 'D', 'I', 'Q', 'U', 'S', 'E', 'T', 'I', 'E', 'R',
2                'A', 'T', 'S']
```

Il n'est pas possible de reconstruire l'arbre à partir de cette seule liste : il y a plusieurs arbres qui donnent cet ordre préfixe.

Par contre, c'est possible à partir de cette liste :

```
1  L_tuto = ['V', 'D', 'I', 'Q', '#', 'U', '#', '#', '#', 'S', 'E', '#', '#',
2            'T', '#', '#', 'I', 'E', '#', 'R', '#', '#', 'A', 'T', '#', '#',
3            'S', '#', '#']
```

On parle de "sérialisation". (Preorder traversal serialization of a binary tree.)

1. Écrire une fonction qui retourne un arbre binaire sous sa forme "sérialisée" : une liste comme celle présentée ci-dessus.
2. Écrire une fonction qui reconstruit l'arbre à partir de sa forme "sérialisée".

Bonus : Écrire une fonction qui vérifie si une liste est une "sérialisation" correcte d'un arbre binaire, sans reconstruire l'arbre.

4 Des expressions et des arbres - midTerm 2016

On peut représenter une expression par un arbre : les nœuds internes contiennent les opérateurs, les opérandes, elles, sont contenues dans les nœuds externes. Les expressions dont il est question ici sont des expressions arithmétiques utilisant les opérateurs binaires + - * et /.

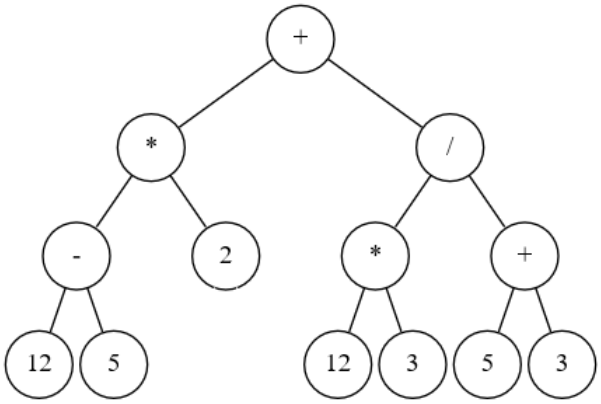

FIGURE 2 – Arbre de l'expression $(12-5)*2+(12*3)/(5+3)$

Exercice 4.1 (Dessine moi)

Les arbres binaires B_1, B_2, B_3 et B_4 représentent des expressions.

Lors du parcours profondeur main gauche

— les valeurs de B_1 sont rencontrées en préfixe dans cet ordre : + - / 8 2 5 + 4 * 2 3

— les valeurs de B_2 sont rencontrées en suffixe dans cet ordre : 8 20 - 2 2 * 2 + /

— les parcours de B_3 et B_4 donnent en infixe le même ordre : 8 + 7 / 5 - 4 * 2

— B_3 et B_4 sont différents mais les deux expressions qu'ils représentent valent toutes les deux -5.

Dessiner les arbres B_1, B_2, B_3 et B_4 et donner les valeurs des expressions représentées par B_1 et B_2.

Exercice 4.2 (Affiche moi)

Écrire la fonction `exp2str` qui retourne une chaîne contenant l'expression complètement parenthésée représentée par un arbre.

Exemple d'application sur l'arbre de la figure 2 :

```
>>> exp2str(B)
'(((12-5)*2)+((12*3)/(5+3)))'
```

Exercice 4.3 (Compte moi)

Écrire la fonction `nodes` qui retourne le nombre d'opérateurs, ainsi que le nombre d'opérandes d'une expression représentée par un arbre binaire.

Exemple d'application sur l'arbre de la figure 2 :

```
>>> nodes(B)
(6, 7)
```

Arbres binaires (Binary Trees)
Occurrences - Hierarchical numbering

5 Occurrences

Exercice 5.1 (Affichage par occurrences)

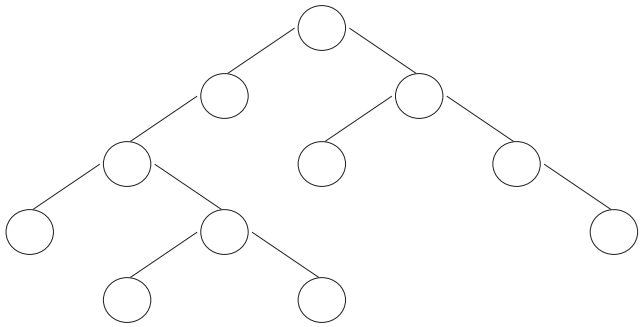

FIGURE 3 – Arbre binaire pour occurrences

1. Donner la représentation sous forme de liste d'occurrences de l'arbre de la figure 3.
2. Écrire la fonction permettant d'afficher la représentation par occurrences d'un arbre binaire.

Exercice 5.2 (Arbre binaire et code préfixe – *Partiel 2015*)

Une méthode pour compresser des fichiers textes consiste à encoder les caractères par des mots binaires. On considère ici un encodage de taille variable : chaque caractère peut être représenté par un nombre différent de bits. Il est évidemment conseillé d'utiliser des codes courts pour les caractères fréquents, et de réserver les codes longs pour les caractères rares.
Il faut d'autre part qu'aucun code ne soit préfixe d'un autre. Par exemple si on choisit d'encoder 'a' par 11 et 'b' par 111, on ne saura plus si 11111 désigne 'ab' ou 'ba'.

1. Considérons le code suivant :

lettre	a	b	c	d	e	f
code	0	101	100	111	1101	1100

   Décoder 110111001100010011101.

2. L'encodage est représenté par un arbre dont les **feuilles** sont les lettres à encoder (le champ `cle` contient la lettre) : le code d'une lettre se lit en suivant la branche de cette lettre à partir de la racine (fils gauche = 0, fils droit = 1).
   À quoi correspond le code d'une lettre ?
3. Dessiner l'arbre correspondant au code de la question 1.
4. L'arbre représentant l'encodage est un arbre localement complet dont toutes les feuilles sont utilisées. Écrire une fonction qui affiche le code d'une lettre donnée si elle existe dans l'arbre (type donné en annexe). Par exemple, avec le code de la question 1, si la lettre donnée est 'e', la fonction affichera "1101".
   Deux versions possibles pour cette fonction :
   — La "simple" mais pas très optimale : qui parcourt tout l'arbre.
   — La plus optimale (mais...) : qui s'arrête dès que possible. Si cette version est choisie, elle devra de plus afficher "no code found" si la lettre n'est pas trouvée dans l'arbre.
   Dans les deux cas, vous devez écrire une fonction récursive et la fonction d'appel.

   Choisissez la version qui vous convient sachant que c'est bien entendu la deuxième version qui rapportera le plus de points. [1]

1. Des fois, il vaut mieux peu de points que pas de point.

6 Numérotation hiérarchique (Hierarchical Numbering)

Exercice 6.1 (Occurrences ↔ Numérotation hiérarchique)

1. Écrire une fonction qui à partir de l'occurrence d'un nœud (une chaîne) calcule son numéro hiérarchique.
2. Écrire la fonction inverse : qui construit l'occurrence d'un nœud à partir de son numéro hiérarchique.

Exercice 6.2 (Implémentation hiérarchique)

Rappel :

On peut utiliser un simple tableau (une liste en Python) pour représenter un arbre binaire. Il suffit de stocker chaque valeur à la position correspondant au numéro en ordre hiérarchique du noeud la contenant. Ici, toutes les cases non "utilisées" ont la valeur $\emptyset$ (`None` en Python).

1. Donner le tableau représentant l'arbre de la figure 4.
2. Que faut-il modifier aux parcours (profondeur et largeur) si l'arbre en paramètre est donné sous la forme d'une liste (implémentation "hiérarchique") ?

Exercice 6.3 (Test implémentation hiérarchique – *Partiel 2014*)

Écrire la fonction qui vérifie si les deux arbres en paramètre, un en représentation classique ("objet") l'autre en implémentation par liste, sont identiques.

Exercice 6.4 (Object ↔ List)

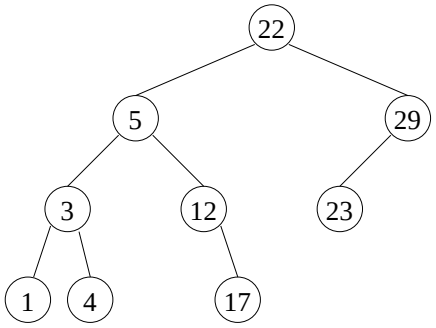

FIGURE 4 – Arbre pour implémentation hiérarchique

1. Écrire une fonction qui construit la liste contenant la représentation hiérarchique d'un arbre binaire (implémentation "classique"). La valeur particulière `None` sera utilisée pour indiquer un arbre vide.

   L'arbre ici est considéré comme quelconque. Que peut-on modifier si l'arbre est parfait ?

2. Écrire la fonction qui construit la représentation "objet" de l'arbre donné sous forme hiérarchique.