VII - TP 2

Arbres binaires : interface

1 Présentation

1.1 Définition

Les structures arborescentes permettent de représenter efficacement et synthétiquement des données issues de problèmes divers : probabilités, arbres de généalogie, organisation des répertoires sur un disque dur, liste des appels effectuées par une fonction récursive complexe, etc.

Les arbres sont une structure de données constituée d'éléments appelés nœud, auxquels on peut associer un certain nombre d'enfants. On s'intéresse dans ce TP aux arbres binaires, c'est à dire les arbres pour lesquels chaque nœud possède exactement deux enfants.

L'arbre vide un arbre qui ne contient aucun nœud.

Les enfants chaque arbre possède deux enfants, l'enfant gauche et l'enfant droit. Chaque enfant est lui même un arbre (il peut être vide).

La racine le seul nœud de l'arbre qui ne possède pas de parents.

Les feuilles il s'agit de nœuds dont les deux enfants sont des arbres vides.

Les étiquettes chaque nœud peut être étiquetté par une valeur.

On prendra comme convention de représenter l'arbre vide par la valeur spéciale `None`. On assimilera également les arbres à leur nœud racine.

On donne ci-dessous l'interface du type `Arbre`.

Fonction	Description
`creer_vide()`	Renvoie un arbre vide
`est_vide(a)`	Détermine si l'arbre `a` est vide
`gauche(a)`	Renvoie l'enfant gauche de l'arbre `a`
`droit(a)`	Renvoie l'enfant droit de l'arbre `a`
`etiquette(a)`	Renvoie l'étiquette de la racine de l'arbre `a`
`Arbre(e, ag, ad)`	Renvoie un arbre dont la racine a pour étiquette `e`, pour enfant gauche `ag` et pour enfant droit `ad`

Ces fonctions seront importées depuis le fichier `ds.py` à l'aide de la commande

Code

```
from ds import Arbre, creer_vide, est_vide, gauche, droit, etiquette
```

1.2 Première utilisation

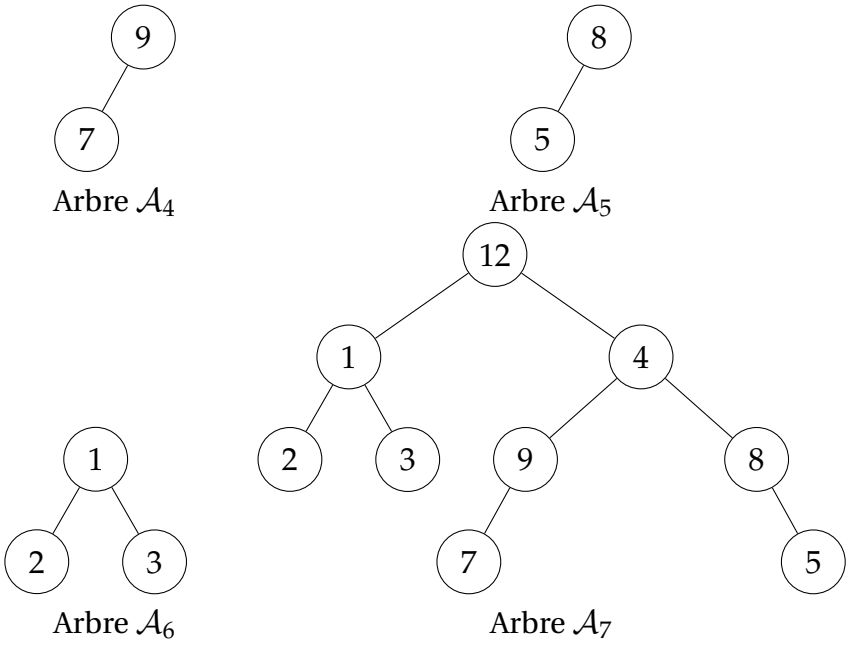
Question 1. On peut représenter un arbre de la manière suivante :

Code

```
a0 = Arbre(5, None, None)
a1 = Arbre(-2, None, None)
a2 = Arbre(1, a0, a1)
a3 = Arbre(1,
           Arbre(4,
                 None,
                 Arbre(3, None, None)),
```

```
            Arbre(5,
                  Arbre(2,
                        Arbre(12, None, None),
                        Arbre(9, None, None)),
                  Arbre(1,
                        Arbre(7, None, None),
                        None)))
```

1. Représenter graphiquement les arbres `a0`, `a1`, `a2`, `a3`.
2. Quelle est l'étiquette de la racine de l'arbre $\mathcal{A}_0$?
3. Quel est le sous-arbre droit du nœud étiqueté 1 de l'arbre $\mathcal{A}_3$?
4. Quel est le sous-arbre gauche du nœud étiqueté 5 de l'arbre $\mathcal{A}_3$?
5. Quelles sont les étiquettes des feuilles de l'arbre de l'arbre $\mathcal{A}_3$?
6. Donner le code python correspondant aux arbres suivants :



2 Exercices

2.1 Feuille d'un arbre

Écrire une fonction `est_feuille` qui étant donné un arbre `a` détermine si celui-ci est constitué d'un seul élément (sa racine est donc également une feuille).

Code

```
def est_feuille(a):
    """ Arbre -> bool
    Détermine si a est constitué d'un seul élément """
    pass
```

Code

```
print(est_feuille(a1))
print(est_feuille(a2))
```

Résultat

```
True
False
```

2.2 Taille d'un arbre

Écrire une fonction récursive `taille` qui étant donné un arbre `a` détermine le nombre de nœuds que contient l'arbre `a`.

Code

```
def taille(a):
    """ Arbre -> int
    Renvoie le nombre de nœuds de a """
    pass
```

Code

```
for a in [a0, a1, a2, a3, a4, a5, a6, a7]:
    print(taille(a), end=" ")
```

Résultat

```
1 1 3 9 2 2 3 9
```

2.3 Somme des éléments d'un arbre

Écrire une fonction récursive `somme` qui étant donné un arbre `a` non vide dont les étiquettes de tous les nœuds sont des entiers, détermine la somme des étiquettes de l'arbre `a`.

Code

```
def somme(a):
    """ Arbre -> int
    Renvoie la somme des éléments de l'arbre a """
    pass
```

Code

```
for a in [a0, a1, a2, a3, a4, a5, a6, a7]:
    print(somme(a), end=" ")
```

Résultat

```
5 -2 4 44 16 13 6 51
```

2.4 Hauteur d'un arbre

Écrire une fonction récursive `hauteur` qui étant donné un arbre `a` non vide détermine le nombre de "niveaux" dont est constitué l'arbre.

Code

```
def hauteur(a):
    """ Arbre -> int
    Renvoie la hauteur de l'arbre """
    pass
```

Code

```
for a in [a0, a1, a2, a3, a4, a5, a6, a7]:
    print(hauteur(a), end=" ")
```

Résultat

```
1 1 2 4 2 2 2 4
```

2.5 Affichage d'un arbre en mode texte

Écrire une fonction `affiche_infixe` qui étant donné un arbre `a` en réalise l'affichage à l'aide de l'algorithme récursif suivant :

1. Si l'arbre est vide on n'affiche rien.
2. Sinon :
   a. on affiche une parenthèse ouvrante ;
   b. on affiche récursivement le sous-arbre gauche ;
   c. on affiche l'étiquette du nœud racine de l'arbre ;
   d. on affiche récursivement le sous-arbre droit ;
   e. on affiche une parenthèse fermante.

On utilisera l'argument optionnel `end` de la fonction `print` afin que tous les affichages soient réalisés sur la même ligne.

Code

```
def affiche_infixe(a):
    """ Arbre -> Nonetype
    Affiche l'arbre a de manière infixe """
    pass
```

Code

```
affiche_infixe(a7)
```

Résultat

```
(((2)1(3))12(((7)9)4(8(5))))
```

Question 2.

1. Représenter graphiquement les arbres dont les affichages par la fonction `affiche_infixe` est :
   a. `((3)4(5))` **b.** `(3(4(5)))` **c.** `(((3)4)5)` **d.** `(3((4)5))` **e.** `((3(4))5)`
2. Inverser les étapes c. et b. de l'algorithme d'affichage d'un arbre binaire. Tester et commenter.
3. Inverser les étapes c. et d. de l'algorithme d'affichage d'un arbre binaire. Tester et commenter.

2.6 Recherche d'un élément dans un arbre

Écrire une fonction récursive `recherche` qui étant donné un arbre `a` dont les étiquettes de tous les nœuds sont des entiers, et un entier `e` détermine si l'entier `e` correspond à étiquette d'un des nœuds de l'arbre.

Code

```
def rechercher(a, e):
    """ Arbre, int -> bool
    Renvoie True ssi e est une des étiquettes de a """
    pass
```

Code

```
print(rechercher(a7, 3))
print(rechercher(a7, 8))
print(rechercher(a7, 42))
```

Résultat

```
True
True
False
```

2.7 Maximum des éléments d'un arbre

Écrire une fonction récursive `maximum` qui étant donné un arbre `a` (supposé non vide) dont les étiquettes de tous les nœuds sont des entiers, détermine la plus grande valeur d'une étiquette de l'arbre `a`.

Code

```
def maximum(a):
    """ Arbre -> int
    Renvoie la plus grande étiquette de a """
    pass
```

Code

```
print(maximum(a6))
```

Résultat

```
3
```

3 Égalité d'arbres

3.1 Égalité d'arbres

On dit que deux arbres sont égaux lorsqu'ils ont la même structure et que leurs étiquettes sont les mêmes (aux mêmes endroits). Deux arbres égaux ont la même représentation graphique.

Écrire une fonction `est_egal` qui étant donné deux arbres `a1` et `a2` détermine si les deux arbres sont égaux.

Code

```
def est_egal(a1, a2):
    """ Arbre, Arbre -> bool
    Détermine si les arbres a1 et a2 sont identiques """
    pass
```

Code

```
print(est_egal(a1, a1))
print(est_egal(a1, a2))
```

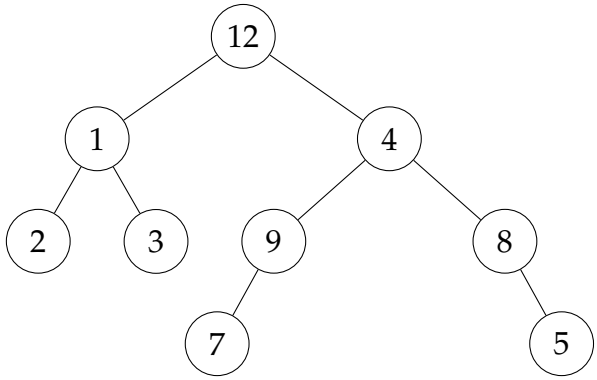
Résultat

```
True
False
```

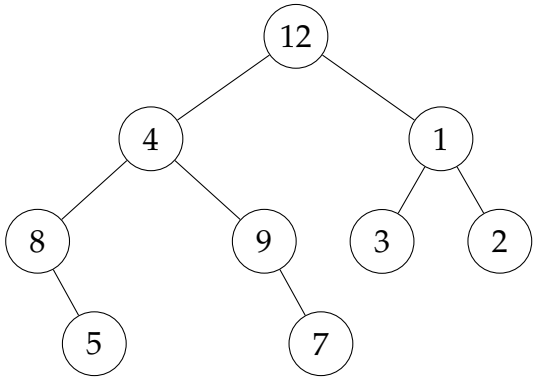
3.2 Égalité faible d'arbres

On dit que deux arbres sont *faiblement* égaux si on peut obtenir l'un en échangeant un nombre arbitaire d'enfants gauche et droits. Les étiquettes doivent cependant se trouver au même endroit après transformation.

Par exemple, les deux arbres ci-dessous ne sont pas égaux mais sont faiblement égaux.



Arbre $\mathcal{A}_7$



Arbre $\mathcal{A}_8$

Écrire une fonction `est_egalf` qui étant donné deux arbres `a1` et `a2` détermine si les deux arbres sont faiblement égaux.

Code

```
def est_egalf(a1, a2):
    """ Arbre, Arbre -> bool
    Renvoie True si et seulement si les arbres a1 et a2 sont faiblement
    ↪ égaux """
    pass
```

Code

```
a8 = Arbre(12,
           Arbre(4,
                 Arbre(8, None, Arbre(5, None, None)),
                 Arbre(9, None, Arbre(7, None, None))),
           Arbre(1, Arbre(3, None, None), Arbre(2, None, None)))
print(est_egalf(a4, a5))
print(est_egalf(a7, a8))
```

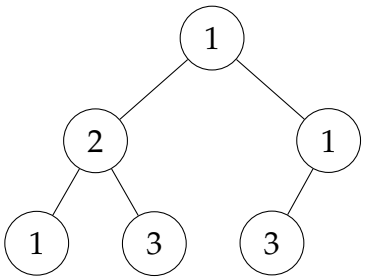
Résultat

```
False
True
```

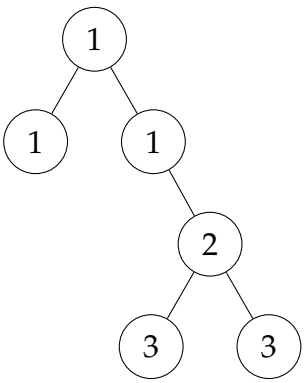
3.3 Égalité de contenu d'arbres

On dit que deux arbres ont le même contenu si ils possèdent les mêmes étiquettes (qui possèdent le même nombre d'occurrence), sans se soucier de la structure de l'arbre.

Par exemple, les arbres ci-dessous ont le même contenu, mais ne sont pas égaux ni faiblement égaux :



Arbre $\mathcal{A}_9$



Arbre $\mathcal{A}_{10}$

3.3.1 Contenu d'un arbre

Écrire une fonction `contenu` qui étant donné un arbre `a` et un dictionnaire `d` *ajoute* au dictionnaire le contenu de `a` de la manière suivante :

- les clés du dictionnaire correspondront aux étiquettes de l'arbre ;
- les valeurs du dictionnaires correspondront aux occurrences de la clé correspondante.

On rappelle que si `d` est un dictionnaire `k in d` vaut `True` si et seulement si `k` est une des clés du dictionnaire `d`. On rappelle également que l'on peut accéder à la valeur associée à la clé `k` du dictionnaire `d` à l'aide de l'instruction `d[k]`.

Code

```
d = {3:6, 1:2, 2:4}
print(3 in d)
print(6 in d)
print(d[3])
d[3] += 1
print(d[3])
```

Résultat

```
True
False
6
7
```

Code

```
def contenu(a, d):
    """ Arbre, dict -> Nonetype
    Ajoute à d le contenu de a """
    pass
```

Code

```
d = dict()
a9 = Arbre(1,
           Arbre(2, Arbre(1, None, None), Arbre(3, None, None)),
           Arbre(1, Arbre(3, None, None), None))
contenu(a9, d) # ne renvoie rien. Exécuté pour son effet de bord.
print(d)
```

Résultat

```
{1: 3, 2: 1, 3: 2}
```

3.3.2 Égalité de dictionnaires

Écrire une fonction `est_dico_egal` qui étant donné deux dictionnaires détermine si ceux-ci sont égaux, c'est à dire :

- `d1` et `d2` possèdent le même ensemble de clés ;
- pour chaque clé `k` de leur ensemble de clé commun, les valeurs associées sont identiques : `d1[k]` est égal à `d2[k]`.

Code

```
def est_dico_egal(d1, d2):
    """ dict, dict -> bool
    Détermine si les dictionnaires d1 et d2 sont égaux """
    pass
```

Code

```
d1 = {1:2, 2:4}
d2 = {1:2, 2:4, 3:5}
print(est_dico_egal(d1, d2))
print(est_dico_egal(d2, d2))
```

Résultat

```
False
True
```

3.3.3 Égalité de contenu d'arbres

Déduire des questions précédentes une fonction `est_egalc` qui étant donné deux arbres `a1` et `a2` détermine si les arbres `a1` et `a2` possèdent le même contenu.

Code

```
def est_egalc(a1, a2):
    """ Arbre, Arbre -> bool
    Renvoie True ssi les arbres a1 et a2 ont le même contenu """
    pass
```

Code

```
a10 = Arbre(1,
            Arbre(1, None, None),
            Arbre(1,
                  None,
                  Arbre(2, Arbre(3, None, None), Arbre(3, None, None))))
print(est_egalc(a8, a9))
print(est_egalc(a9, a10))
```

Résultat

```
False
True
```