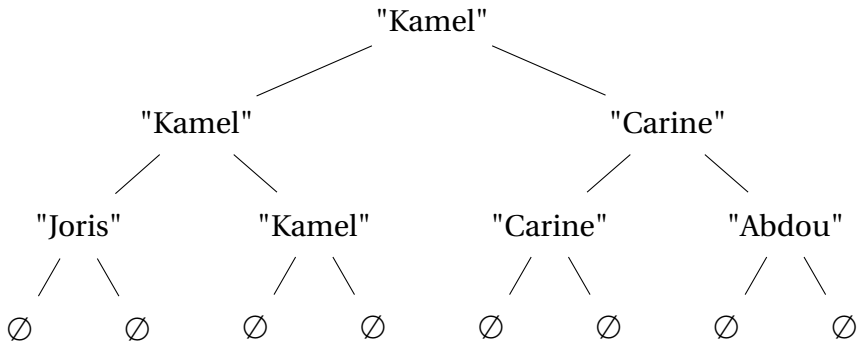
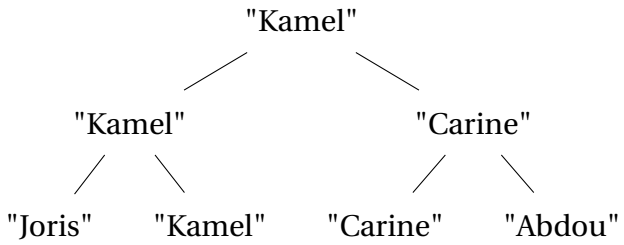
IV - TP 3 Parcours d'arbres - Tournoi

La fédération de badminton souhaite gérer ses compétitions à l'aide d'un logiciel. Pour ce faire, une structure arbre de compétition a été définie récursivement de la façon suivante : un arbre de compétition est soit l'arbre vide, noté ∅, soit un triplet composé d'une chaîne de caractères appelée étiquette, d'un arbre de compétition appelé sous-arbre gauche et d'un arbre de compétition appelé sous-arbre droit. On représente graphiquement un arbre de compétition de la façon suivante :



Comme souvent pour alléger la présentation, on ne note pas les arbres vides. On représentera l'arbre `A` précédent par :



Cet arbre se lit de la façon suivante :

- 4 participants se sont affrontés : Joris, Kamel, Carine, et Abdou. Leurs noms apparaissent en bas de l'arbre, se sont les étiquettes des feuilles de l'arbre.
- Au premier tour, Kamel a battu Joris, et Carine a battu Abdou.
- En finale, Kamel a battu Carine.
- Le vainqueur de la compétition est Kamel.

Pour s'assurer que chaque finaliste ait joué le même nombre de matchs, un arbre de compétition a toutes ses feuilles à la même hauteur : il s'agit donc d'arbres parfaits. On suppose également que tous les joueurs de la compétition ont des noms différents.

On utilisera les fonctions usuelles pour représenter et manipuler les arbres : `Arbre`, `est_vide`, `est_feuille`, `etiquette`, `gauche`, et `droit`. L'arbre vide est représenté par la valeur `None`. Ces fonctions seront importées depuis le fichier `ds.py` à l'aide de la commande :

Code

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

L'arbre de l'énoncé est alors :

Code

```
a = Arbre("Kamel",
          Arbre("Kamel",
                Arbre("Joris", None, None),
                Arbre("Kamel", None, None)),
          Arbre("Carine",
                Arbre("Carine", None, None),
                Arbre("Abdou", None, None)))
```

1 Informations sur le tournoi

1.1 Nombre de joueurs du tournoi

Écrire une fonction `nb_joueurs` qui étant donné un arbre de compétition `a` renvoie le nombre de joueurs ayant participé à cette compétition.

Code

```
def nb_joueurs(a):
    """ Arbre -> int
    Renvoie le nombre de joueurs ayant participé à la compétition d'arbre
    a """
    pass
```

Code

```
print(nb_joueurs(a))
```

Résultat

```
4
```

1.2 Nombre de rounds d'un tournoi

Écrire une fonction `nb_rounds` qui étant donné un arbre de compétition `a` renvoie le nombre de rounds dont a été composée la compétition.
Remarque. Un round correspond à un "niveau" de la compétition : la finale, les demi-finales, les quarts de finale, etc. sont des rounds.

Code

```
def nb_rounds(a):
    """ Arbre -> int
    Renvoie le nombre de rounds de la compétition a """
    pass
```

Code

```
print(nb_rounds(a))
```

Résultat

```
2
```

2 Informations sur un joueur

2.1 Nombre d'occurrences du joueur dans l'arbre

Écrire une fonction `occurrences` qui étant donné un arbre `a` et le nom `nom` d'un joueur renvoie le nombre d'occurences de l'étiquette `nom` parmi les nœuds de l'arbre `a`.

Code

```
def occurrences(a, nom):
    """ Arbre, str -> int
    Renvoie le nombre d'occurrences de nom dans a """
    pass
```

Code

```
print(occurrences(a, "Kamel"))
```

Résultat

```
3
```

2.2 Nombre de matchs joués par le joueur

Déduire de la fonction `occurrences`, une fonction `nombre_matchs` qui étant donné un arbre de compétition `a` et un nom `nom` détermine le nombre de matchs joués par `nom` pendant à la compétition.

Attention aux cas particuliers

Code

```
def nombre_matchs(a, nom):
    """ Arbre, str -> int
    Renvoie le nombre de matchs joués par le joueur nom dans la
 ↪  compétition d'arbre a """
    pass
```

Code

```
print(nombre_matchs(a, "Kamel"))
print(nombre_matchs(a, "Carine"))
```

Résultat

```
2
2
```

3 Informations sur les joueurs

3.1 Liste des joueurs du tournoi

Compléter le code de la fonction `liste_joueurs` qui étant donné un arbre de compétition `a` renvoie la liste des noms de tous les joueurs ayant participé à la compétition. On remarquera qu'il s'agit des étiquettes des feuilles de l'arbre `a`. Si jamais l'arbre de compétition est vide, on renverra la liste vide.

On rappelle que si `l1` et `l2` sont deux listes python, alors `l1 + l2` renvoie la concaténation de `l1` et de `l2` : c'est la liste constituée de tous les éléments de `l1` et de `l2` mis bout à bout.

Code

```
def liste_joueurs(a):
    """ Arbre -> [str]
    Renvoie la liste des joueurs ayant participé à la compétition d'arbre
    a """
    # if est_vide(a):
    #     return ....................................
    # elif est_feuille(a):
    #     return [..................................]
    # else:
    #     return ....................................
```

Code

```
print(liste_joueurs(a))
```

Résultat

```
['Joris', 'Kamel', 'Carine', 'Abdou']
```

3.2 Liste des joueurs par niveau

En vous inspirant de la fonction `liste_joueurs`, écrire une fonction `niveau` qui étant donné un entier `i` et un arbre de compétition `a` renvoie la liste des étiquettes des nœuds de profondeur `i` de l'arbre `a`.

On remarquera que :

- Si un arbre de compétition `a` est vide alors pour tout entier `i`, `niveau(i, a)` doit renvoyer la liste vide.
- le niveau 0 correspond à la racine de l'arbre.
- le niveau `i` de l'arbre `a` correspond au niveau `i - 1` du sous-arbre gauche de `a` et au niveau `i - 1` du sous-arbre droit de `a`.

Code

```
def niveau(a, i):
    """ Arbre, int -> [str]
    Renvoie la liste des étiquettes des nœuds de profondeur i de l'arbre a
    """
    pass
```

Code

```
print(niveau(a, 0))
print(niveau(a, 1))
print(niveau(a, 2))
```

Résultat

```
['Kamel']
['Kamel', 'Carine']
['Joris', 'Kamel', 'Carine', 'Abdou']
```

3.3 Liste des joueurs vaincus I

Écrire une fonction `vaincus_gagnant` qui étant donné un arbre de compétition a renvoie la liste des joueurs ayant été vaincus par le gagnant de la compétition.

Code

```
def vaincus_gagnant(a):
    """ Arbre -> [str]
    Renvoie la liste des joueurs ayant joué un match contre le gagnant de
    ↪ la compétition d'arbre a """
    pass
```

Code

```
print(vaincus_gagnant(a))
```

Résultat

```
['Joris', 'Carine']
```

4 Parcours en largeur

4.1 Algorithme de cours

En vous inspirant de l'algorithme vu en cours, écrire une fonction `parcours_largeur` qui étant donné un arbre `a` en renvoie la liste des étiquettes, telles qu'obtenues à l'aide d'un parcours en largeur de l'arbre.

On utilisera un objet `file_noeuds` de type `list` pour représenter la file d'attente des nœuds à traiter :

- la file est vide lorsque `file_noeud` est la liste `[]` ;
- on enfile la racine de l'arbre `a` à l'aide de l'instruction `file_noeuds.append(a)` ;
- on défile le premier élément de la file à l'aide de l'instruction `file_noeuds.pop(0)`
- on accède sans défiler au premier élément de la file à l'aide de l'instruction `file_noeud[0]`

On fera attention à **ne pas enfiler** le sous-arbre gauche ou le sous-arbre droit dans le cas où ceux-ci sont vides.

On utilisera un objet `liste_etiquettes` de type `list` pour stocker la liste des étiquettes de l'arbre `a` dans l'ordre du parcours. Initialement celle-ci sera vide. L'opération de traitement sur le nœud parcouru sera l'ajout de son étiquette à la liste `liste_etiquettes`.

Code

```
def parcours_largeur(a):
    """ Arbre -> [str]
    Renvoie la liste des étiquettes des nœuds de l'arbre a, telle
    ↪ qu'obtenue à l'aide d'un parcours en largeur """
    pass
```

Code

```
print(parcours_largeur(a))
```

Résultat

```
['Kamel', 'Kamel', 'Carine', 'Joris', 'Kamel', 'Carine', 'Abdou']
```

4.2 Liste des joueurs vaincus II

On souhaite établir la liste des joueurs ayant été vaincus par un joueur donné dans la compétition d'arbre `a`. Pour cela, on utilise l'algorithme suivant :

- on détermine le nœud `n` de profondeur *minimale* de l'arbre de compétition `a` dont l'étiquette est égale au nom du joueur recherché ;
- la liste des joueurs vaincus par `nom` dans la compétition d'arbre `a` est la liste des joueurs vaincus par le gagnant de la compétition dont l'arbre est le sous-arbre de `a` de racine `n`.

Par exemple, dans l'arbre de compétition de l'énoncé, si on souhaite établir la liste des joueurs ayant été vaincus par Carine :

- le nœud de profondeur minimal `n` est alors le nœud à la profondeur 1 et d'étiquette `"Carine"` ;
- le sous-arbre de racine `n` est constitué de trois éléments. Dans cette sous-compétition, Carine n'a vaincu qu'un seul joueur, Abdou.

Ainsi la liste des joueurs vaincus par Carine est donc `["Abdou"]`.

4.2.1 Recherche d'un nœud de profondeur minimale

Écrire une fonction `recherche_noeud` qui étant donné un arbre `a` et un `nom` renvoie le nœud de profondeur minimal de `a` dont l'étiquette est `nom`. On utilisera pour cela un parcours en largeur d'abord et on s'arrêtera dès que l'étiquette du nœud courant est le `nom` recherché.

Dans le cas où `nom` n'est pas une étiquette de l'arbre `a`, on renverra `None`.

Code

```
def recherche_noeud(a, nom):
    """ Arbre, str -> Arbre
    Renvoie le sous-arbre de a dont la racine a pour étiquette nom et est
    de profondeur minimale dans a """
    pass
```

Code

```
print(recherche_noeud(a, "Carine"))
```

Résultat

```
<ds.Noeud object at 0x7fee413cf580>
```

4.2.2 Liste des joueurs vaincus

Écrire une fonction `joueurs_vaincus` qui étant donné un arbre de compétition `a` et le nom d'un joueur `nom` renvoie la liste des joueurs vaincus par `nom` dans la compétition. On utilisera pour cela l'algorithme décrit en section 4.2 ainsi que les fonctions `vaincus_gagnant` et `recherche_noeud` précédemment écrites.

Code

```
def joueurs_vaincus(a, nom):
    """ Arbre, str -> [str]
    Renvoie la liste des joueurs vaincus par nom dans la compétition
    d'arbre a """
    pass
```

Code

```
print(joueurs_vaincus(a, "Carine"))
print(joueurs_vaincus(a, "Kamel"))
print(joueurs_vaincus(a, "Joris"))
```

Résultat

```
['Abdou']
['Joris', 'Carine']
[]
```