Séance 7: MODULES, ARBRES

L1 – Université Nice Sophia Antipolis

Dans les exercices suivants, on manipule des arbres binaires d'expression arithmétiques (non vides), comme ils ont été définis dans le cours.

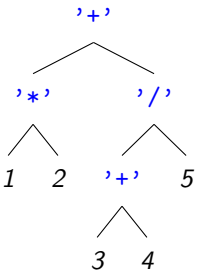
Les expressions seront des nombres entiers (sur les feuilles de l'arbre) ou des chaînes de caractères contenant les opérations (sur les nœuds de l'arbre).

On ne s'intéresse pas à la façon dont les arbres sont représentés, on va utiliser uniquement les fonctions suivantes :

- `arbre(r, Ag, Ad)` qui à partir d'une racine `r`, et des arbres `Ag` et `Ad` renvoie l'arbre de racine `r`, de fils gauche `Ag` et de fils droit `Ad` ;
- `est_operateur(obj)` qui renvoie `True` si `obj` est un des opérateurs `'+'`, `'*'`, `'-'`, `'/'` ;
- `est_feuille(obj)` qui renvoie `True` si `obj` est une feuille, et `False` sinon ;
- `racine(A)` qui renvoie la racine de l'arbre `A`
- `fg(A)` qui renvoie le fils gauche de l'arbre `A`
- `fd(A)` qui renvoie le fils droit de l'arbre `A`.

Exercice 1 (Mise en route, ⋆)

1. *Téléchargez le fichier* `arbres.py` *sur la page du cours. Placez-le dans le répertoire où vous écrivez vos TP. Créez un fichier* `tp8-arbres.py`. *Quelle ligne faut-il écrire dans ce fichier pour pouvoir utiliser les fonctions définies dans* `arbres.py` *?*
2. *Sans utiliser de liste, mais uniquement la fonction* `arbre(r, Ag, Ad)`*, créez l'arbre* `A1` *défini ci-contre.*



□

Exercice 2 (Manipulation d'arbres, ⋆⋆)

1. *Écrire une fonction* `contient42(A)` *qui renvoie* `True` *si une des feuilles de l'arbre* `A` *est 42, et* `False` *sinon.*
2. *Écrire une fonction* `compte_pairs(A)` *qui renvoie le nombre de feuilles de* `A` *qui sont des nombres pairs.*
3. *Écrire une fonction* `feuilles_paires(A)` *qui renvoie la liste des feuilles de* `A` *qui sont des nombres pairs.*

On appelle profondeur *d'un nœud la distance entre ce nœud et la racine.*

4. *Écrire une fonction* `liste_profondeur(A,n)` *qui renvoie la liste des nœuds de* `A` *de profondeur* `n`.
5. *Écrire une fonction* `profondeur_max_pairs(A)` *qui renvoie la profondeur maximale des feuilles paires de* `A` *(et* `-1` *si* `A` *n'a pas de feuille paire).*

□

Exercice 3 (Comptage et transformation, ⋆⋆)

1. *Programmez une fonction* `compter(A, op)` *renvoyant le nombre d'apparitions de l'opérateur* `op` *dans l'arbre* `A`.
   Exemple : `compter(A1, '+')` *renvoie* 2.
2. *Programmez une fonction* `remplacer(A, op1, op2)` *qui renvoie une copie de l'arbre* `A` *où chaque apparition de l'opérateur* `op1` *aura été remplacée par l'opérateur* `op2`.
   Exemple : `remplacer(A1, '+', '-')` *renvoie* `['-', ['*', 1, 2], ['/', ['-', 3, 4], 5]]`.

□

Exercice 4 (Mirroir, ⋆⋆)
Programmez la fonction `miroir(A)` *retournant l'image inversée (à tous les niveaux) de l'arbre* `A`.

Exemple : `miroir(A1)` *renvoie* `['+', ['/', 5, ['+', 4, 3]], ['*', 2, 1]]`. □

Exercice 5 (Parcours suffixe et évaluation avec une pile, ⋆⋆⋆)
Le but de cet exercice est d'écrire une fonction non récursive qui évalue un arbre binaire d'expression arithmétique décrit par son parcours suffixe. *Le parcours profondeur suffixe d'un arbre est obtenu en visitant d'abord le fils gauche, puis le fils droit, puis la racine. On l'appelle souvent* notation polonaise inversée *ou* notation calculatrice HP *(les très vieilles calculatrices fonctionnaient comme ça).*

Par exemple, le parcours profondeur suffixe de l'expression `(1-2)*(3/4)` *est* `[1, 2, '-', 3, 4, '/', '*']`, *et on veut écrire la fonction* `eval_suffixe(L)` *telle que* `eval_suffixe([1, 2, '-', 3, 4, '/', '*'])`, *renvoie* `-0.75` *($-\frac{3}{4}$). On ajoute la contrainte que cette fonction ne doit pas être récursive !*

Indication : Pour évaluer une expression suffixe, on peut la lire de gauche à droite et utiliser une pile pour stocker les valeurs déjà calculées. Pour en savoir davantage, demandez à votre chargé de TP ou consultez la page wikipedia[1].

□

1. https://fr.wikipedia.org/wiki/Notation_polonaise_inverse