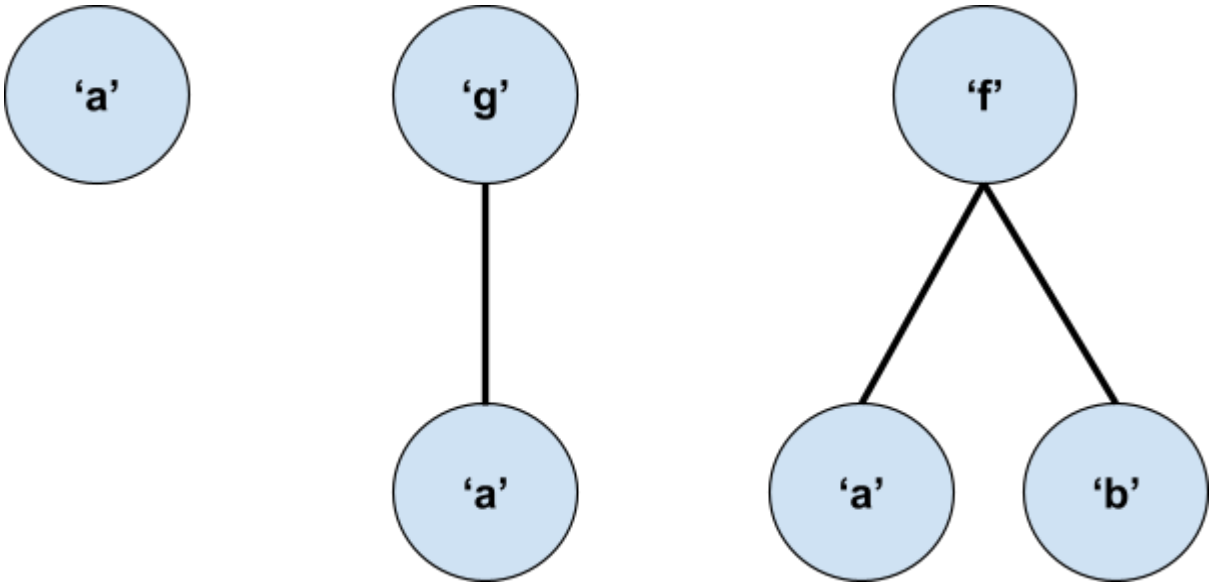
TD - Programmation et structures de données
Pierre-Etienne Moreau

MINESnancy

Arbres et Expressions

L'objectif de ce TD est de concevoir une classe permettant de représenter des arbres n-aires. Nous allons considérer une structure récursive où les nœuds et les arbres sont confondus : chaque nœud composant l'arbre peut avoir 0, 1, 2 ou n fils. Les fils étant eux-mêmes des nœuds.

Dans cette représentation tout nœud est un arbre. Ainsi, les illustrations suivantes sont des arbres :



Exercice 1

Réfléchir aux méthodes que doit proposer une classe **Tree**.

Exercice 2

Définissez une classe **Tree** ayant pour constructeur `Tree(label, *children)` et offrant les méthodes :

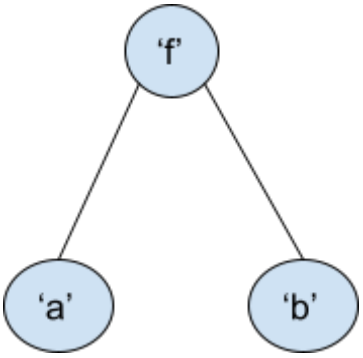
- `label(self) -> str`
- `children(self) -> tuple[Tree]`
- `nb_children(self) -> int`
- `child(self, i: int) -> Tree`
- `is_leaf(self) -> bool`

La notation `*children` signifie que `children` est une liste. Cela permet de voir `Tree` comme un constructeur ayant un nombre variable de paramètres : le label, suivi de 0 à n arbres.

La méthode `label()` renvoie le label stocké au sein du nœud receveur. Ce dernier est une chaîne de caractères.

La méthode `children()` renvoie un tuple d'arbres, éventuellement vide.
La méthode `child(i)` permet de récupérer le i-ème sous-arbre
La méthode `is_leaf()` renvoie `True` lorsque l'arbre est une feuille

Vérifiez que les méthodes fonctionnent correctement en créant par exemple les arbres qui figurent en tête de ce document ; notamment l'arbre suivant :



Utilisez `unittest` pour écrire des jeux de tests

Exercice 3

Complétez la classe **Tree** en y ajoutant la méthode :

- `depth(self) -> int`

La profondeur d'un nœud correspond à la profondeur la plus élevée entre les fils à laquelle est ajouté 1. La profondeur d'une feuille est 0.

Exercice 4

Complétez la classe **Tree** en y ajoutant les méthodes :

- `__str__(self) -> str`
- `__eq__(self, __value: object) -> bool`

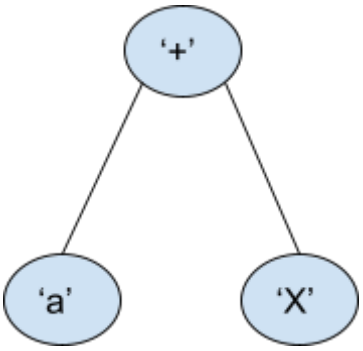
La méthode `__str__` renvoie une chaîne correspondant à la notation préfixée de l'arbre. Ainsi, les 3 arbres illustrés ci-dessus seront représentés par les chaînes `a`, `g(a)` et `f(a,b)`

La méthode `__eq__` permet de comparer deux arbres. Elle renvoie `True` lorsque les deux arbres sont égaux, et `False` sinon

Utilisez `unittest` pour écrire des jeux de tests
Vérifiez par exemple que la méthode `__eq__` fonctionne correctement en créant deux arbres égaux mais pas identiques (leurs adresses mémoires sont distinctes).
Utiliser le fichier test_expression.py pour vérifier que votre code passe les premiers tests et corriger votre code si ce n'est pas le cas.

Exercice 5

On utilise à présent la classe **Tree** pour représenter des expressions arithmétiques.
Par exemple, l'expression a + X peut être représentée par l'arbre suivant (les opérateurs, constantes et variables sont représentées par des labels de type chaîne) :



Définissez une méthode `deriv(self, var: str) -> Tree` qui calcule la dérivée du receveur par rapport à un nom de variable passé en argument.
Ainsi, `Tree('+', Tree('a'), Tree('X')).deriv('X')` doit renvoyer l'arbre `Tree('+', Tree('0'), Tree('1'))`, qui s'affichera sous la forme `+(0,1)`

Notez que la variable de dérivation (`'X'` dans l'exemple ci-dessus) n'est pas un arbre. Mais vous pouvez construire un arbre (`Tree('X')`) pour le rechercher plus facilement dans l'expression à dériver.

Pour le calcul de la dérivée vous devez prendre en compte la dérivée d'une addition (noeud ayant pour label `'+'`), la dérivée d'une multiplication (`'*'`) et la dérivée d'une constante (une feuille dont le label n'est pas la variable de dérivation).
Testez la fonction en dérivant le polynôme $3X^2 + 5X + 7$

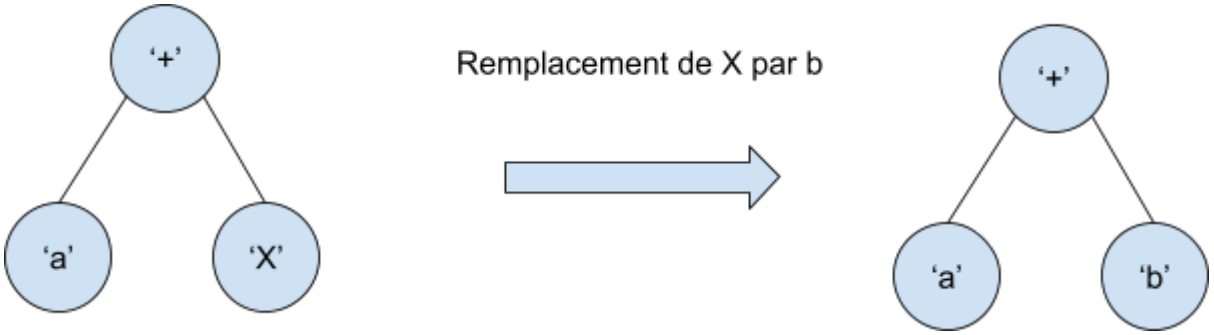
Avant de vous lancer, avez vous bien compris comment représenter le polynôme $3X^2 + 5X + 7$? (il faut le représenter avec un arbre)

Pour les plus rapides :

Exercice 6

Ajoutez une méthode `substitute(self, t1: Tree, t2: Tree) -> Tree` à la classe **Tree**.
Cette méthode remplace toutes les occurrences du sous-arbre `t1` par le sous-arbre `t2`, dans l'arbre passé en premier argument (le *receveur*).

Ainsi, la substitution de X par b dans l'arbre +(a,X) renvoie l'arbre +(a,b)
Autrement dit : `Tree('+', Tree('a'), Tree('X')).substitute(Tree('X'), Tree('b'))` renvoie un arbre dont l'affichage est `+(a,b`



Utilisez `unittest` pour écrire des jeux de tests
Par exemple, vous pouvez vérifier que `e1.substitution(e1, e2) == e2` pour quelques expressions `e1` et `e2` bien choisies.

Exercice 7 (assez difficile)

Ecrivez une fonction `simplify(self) -> Tree` qui effectue les simplifications suivantes :

- X + 0 est remplacé par X
- 0 + X est remplacé par X
- X * 0 est remplacé par 0
- 0 * X est remplacé par 0
- X * 1 est remplacé par X
- 1 * X est remplacé par X
- a + b, avec a et b des arbres représentant des entiers, est remplacé par l'arbre c où c correspond à la somme des entiers représentés par a et b
- a * b, avec a et b des arbres représentant des entiers, est remplacé par l'arbre c où c correspond au produit des entiers représentés par a et b

Appliquez cette fonction sur la dérivée de $3X^2 + 5X + 7$ et vérifiez que vous obtenez une forme simplifiée 6X+5 ou encore 3(X+X)+5

Exercice 8

Utilisez les méthodes `substitute` et `simplify` pour calculer la valeur en un point du polynôme $3X^2 + 5X + 7$

Exercice 9

Améliorez l'affichage pour avoir une notation plus usuelle (on parle de notation infixe)

Pour aller plus loin...

Proposez puis coder un algorithme permettant de construire un arbre à partir d'une chaîne de caractères

Par exemple `'+(X,3)'`
Ou encore `'X+3'`