MINESnancy

Séance 3

Arbres

Pierre-Etienne Moreau
Guillaume Bonfante

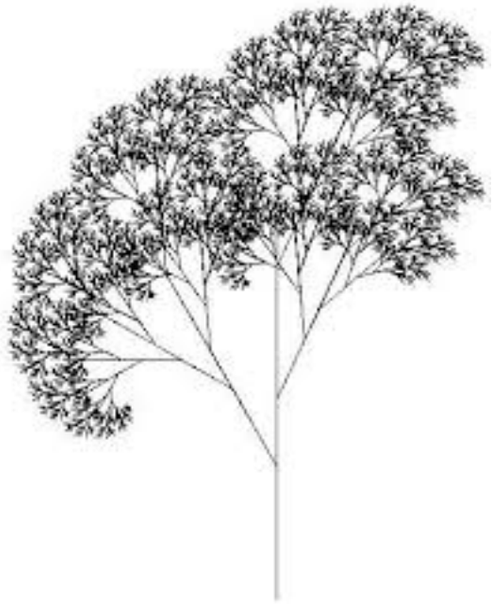
La plupart des bons algorithmes fonctionnent grâce à une méthode astucieuse pour organiser les données.

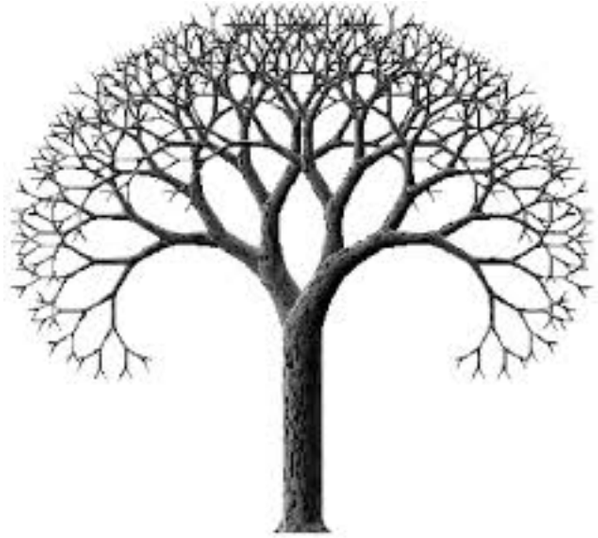
Exemples

- Une **pile** pour vérifier qu'une expression est bien parenthèsée
- Un **dictionnaire** pour associer un score à un mot
- Une **liste** de villes pour trouver un plus court chemin
- Un **ensemble** pour représenter les valeurs atteignables au compte-est-bon

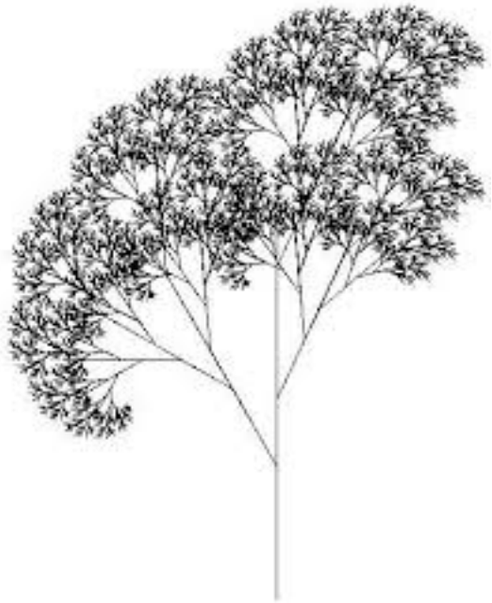
La structure d’arbre a de nombreuses applications

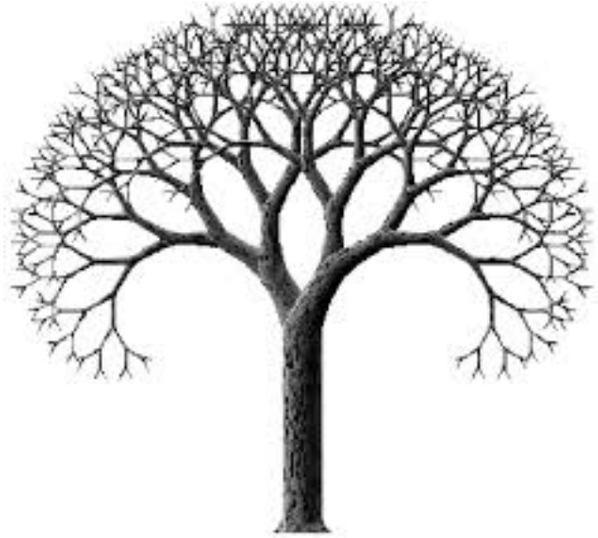
Un arbre est une structure récursive

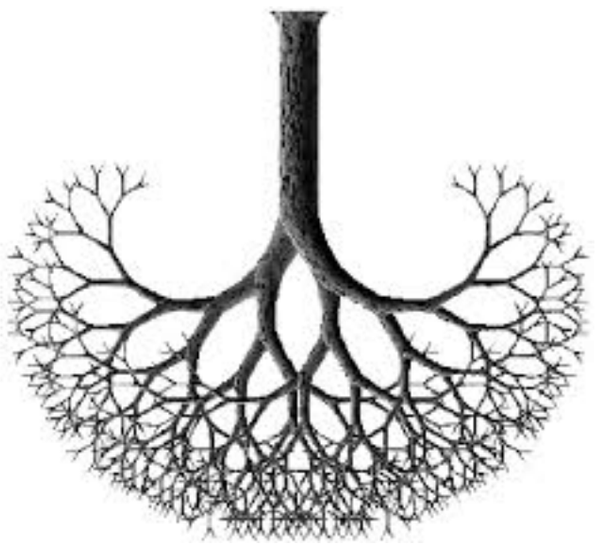




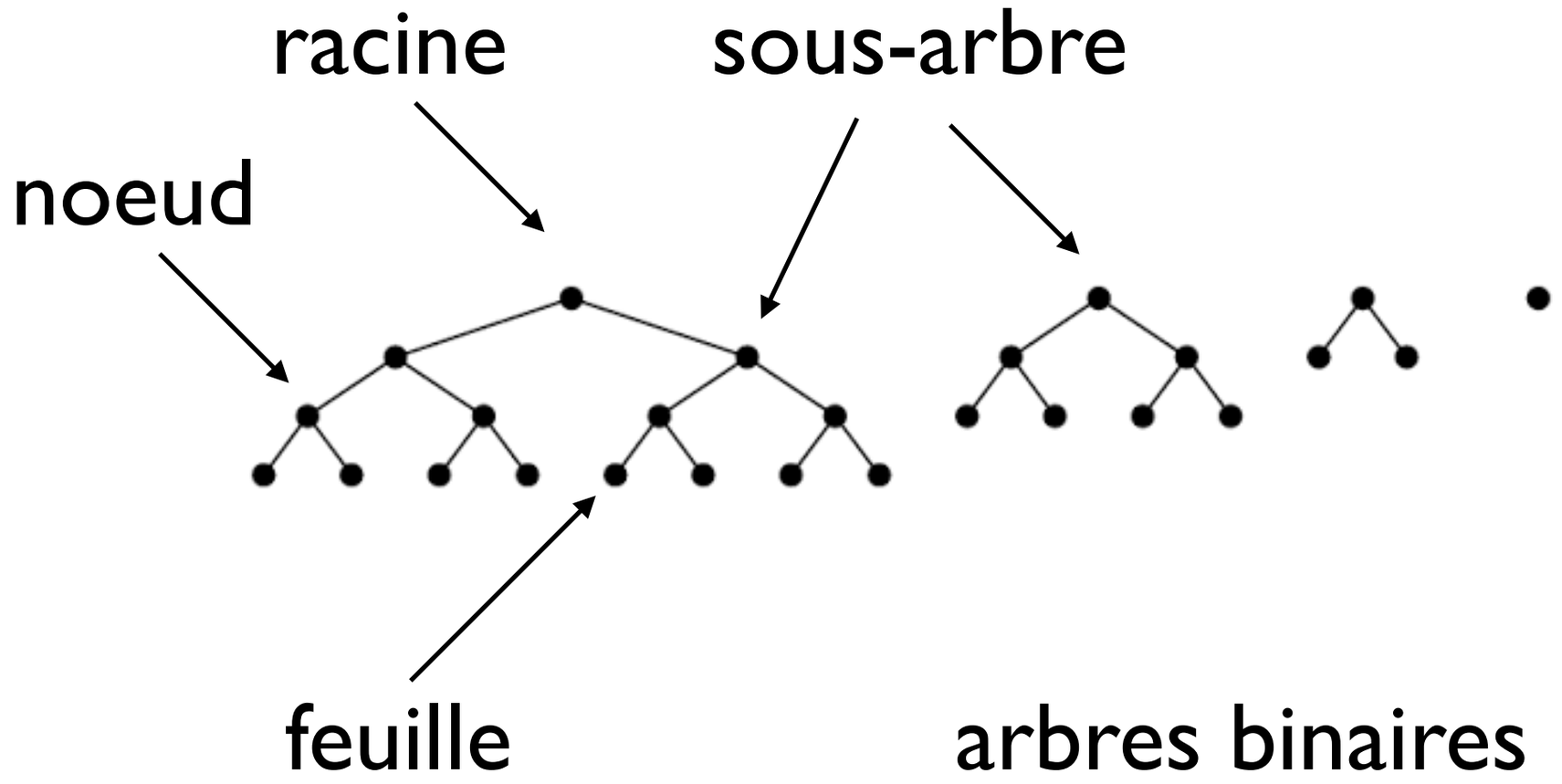
Un arbre est une structure récursive



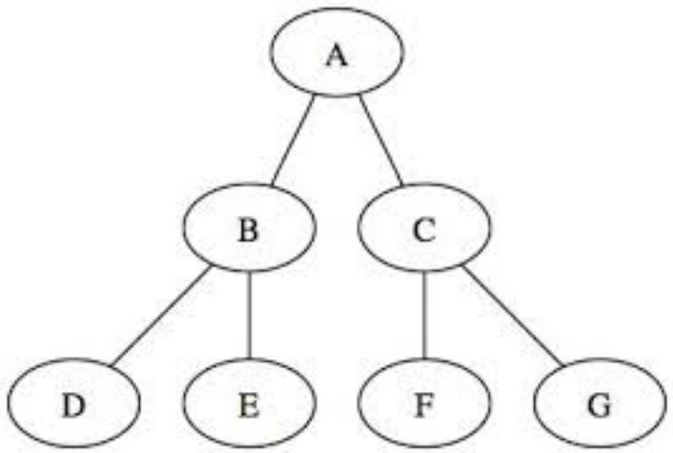


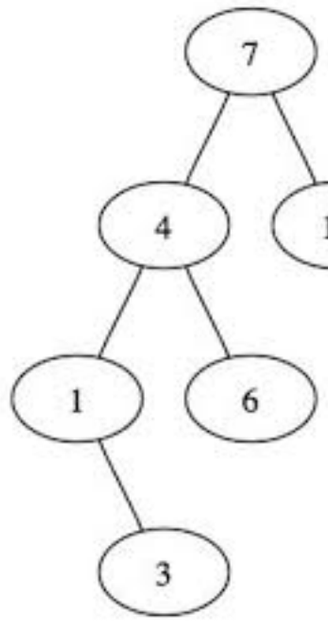


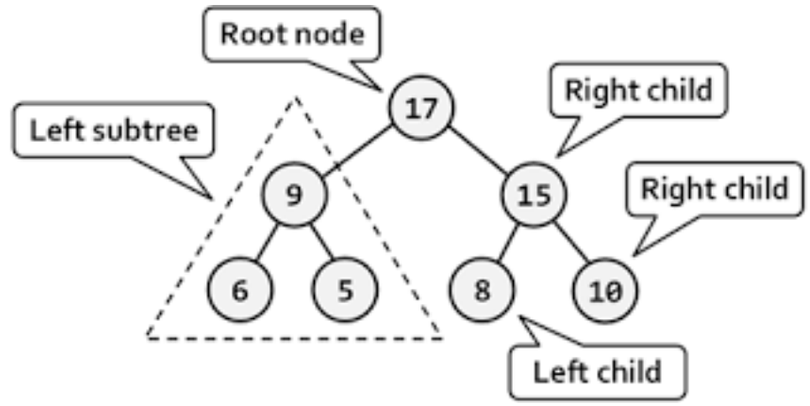
Un peu de terminologie



Arbres binaires avec labels





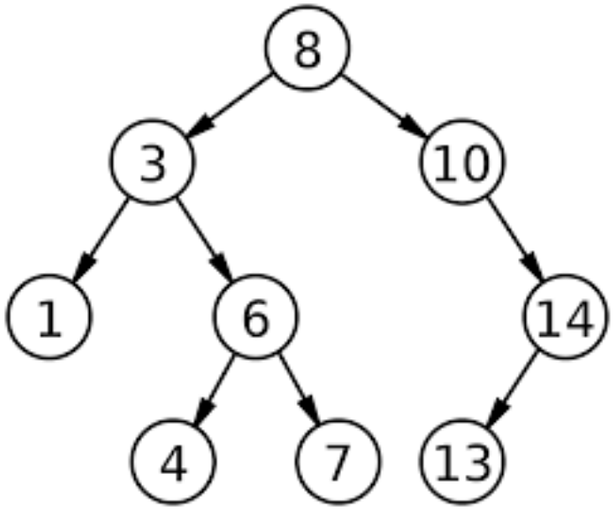


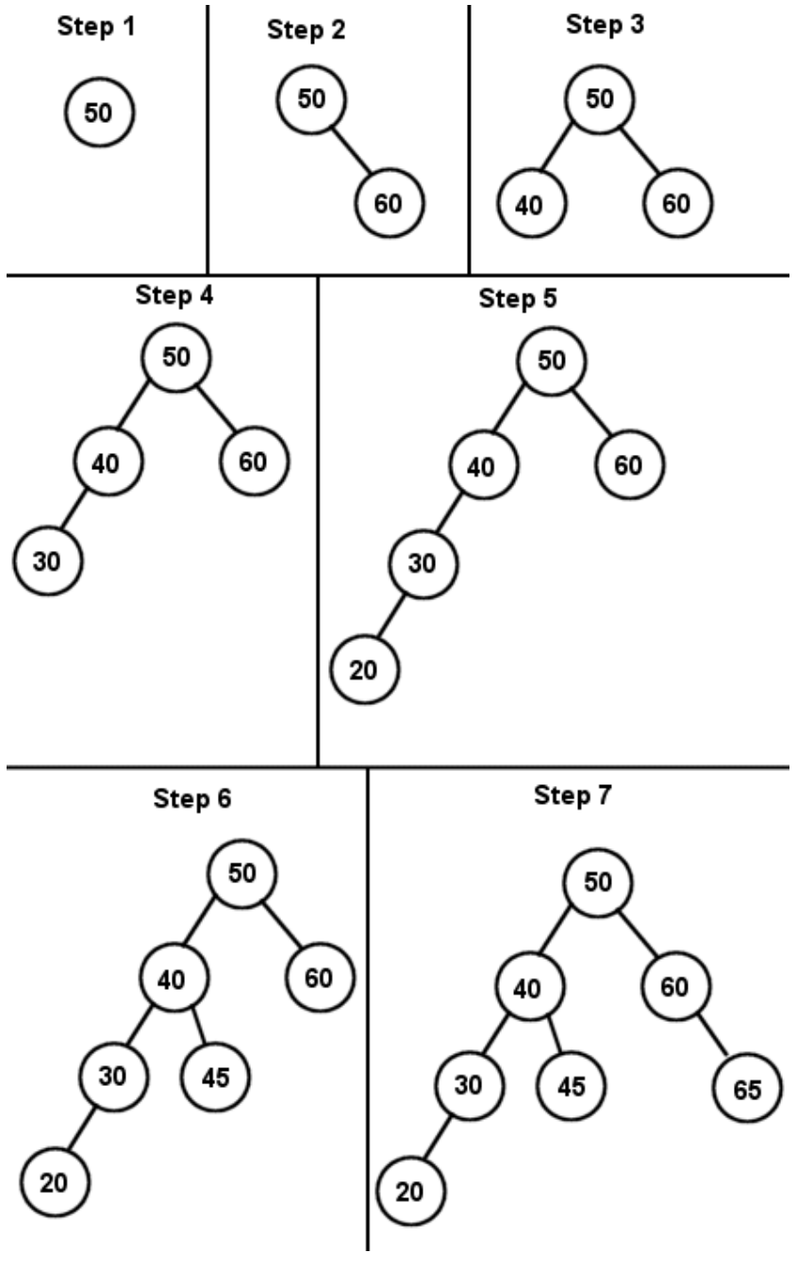
complet

Permet de représenter des ensembles

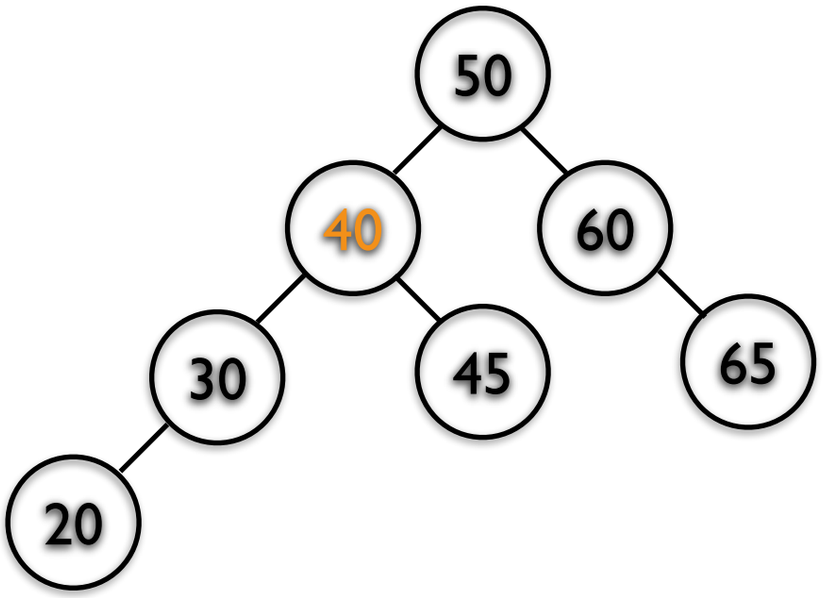
lorsque les éléments sont comparables

Arbre Binaire de Recherche

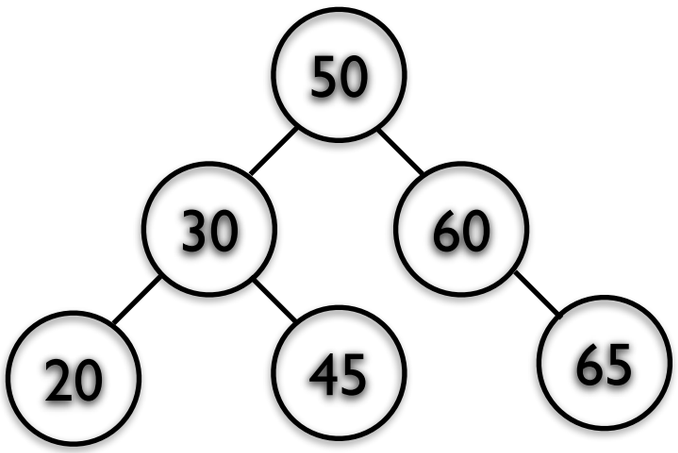




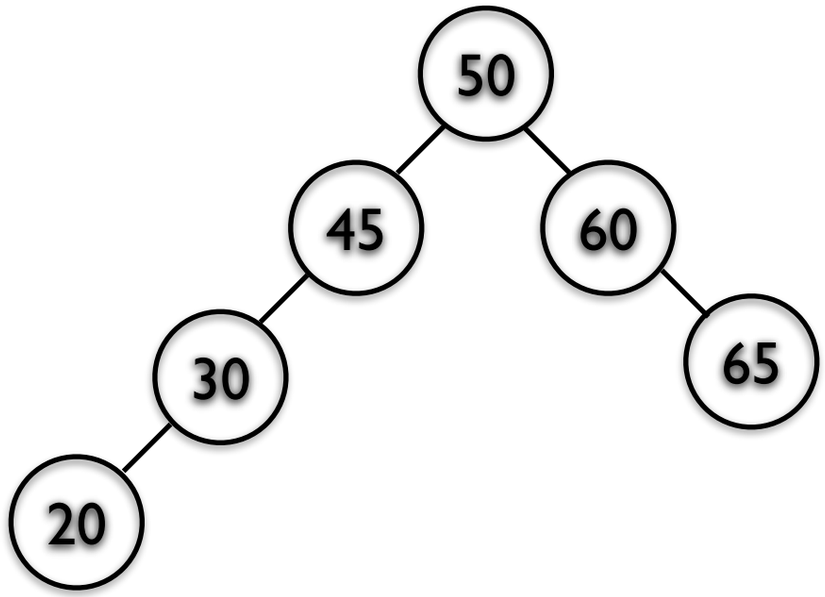
50, 60, 40, 30, 20, 45, 65



Arbre 1



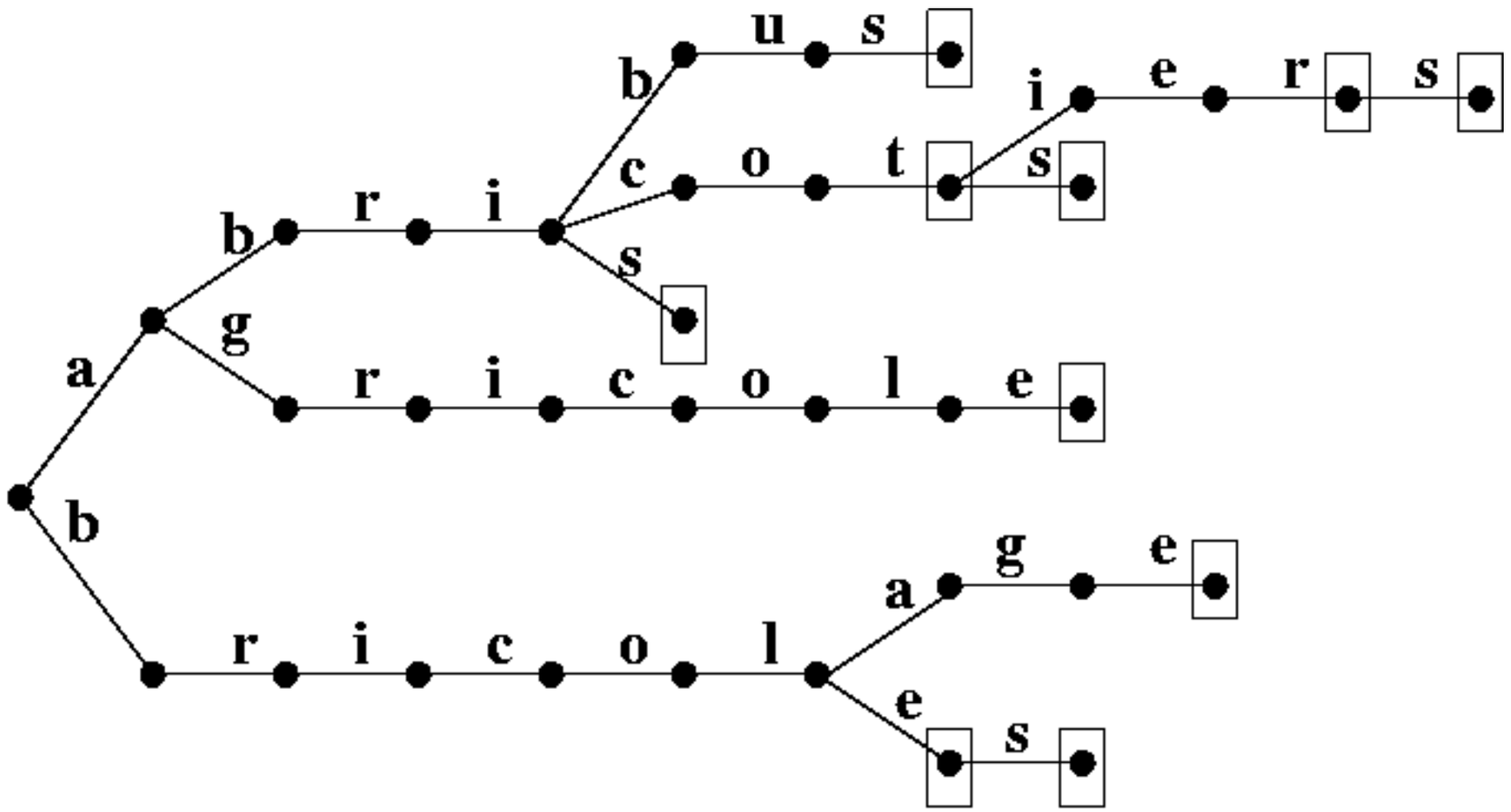
Arbre 2



Permet de représenter un automate

Arbre de discrimination

Les arbres ne sont pas forcément binaires



Application à l'analyse syntaxique

Arbre de syntaxe

Grammaire

- EPA ::= Identificateur
- EPA ::= Constante
- EPA ::= (EPA Op EPA)

Expression complètement Parenthésée

- Op ::= + | - | * | /
- x est une EPA
- 3 est une EPA
- x * 3 n'est pas une EPA
- (x * 3) est une EPA
- (3) n'est pas une EPA

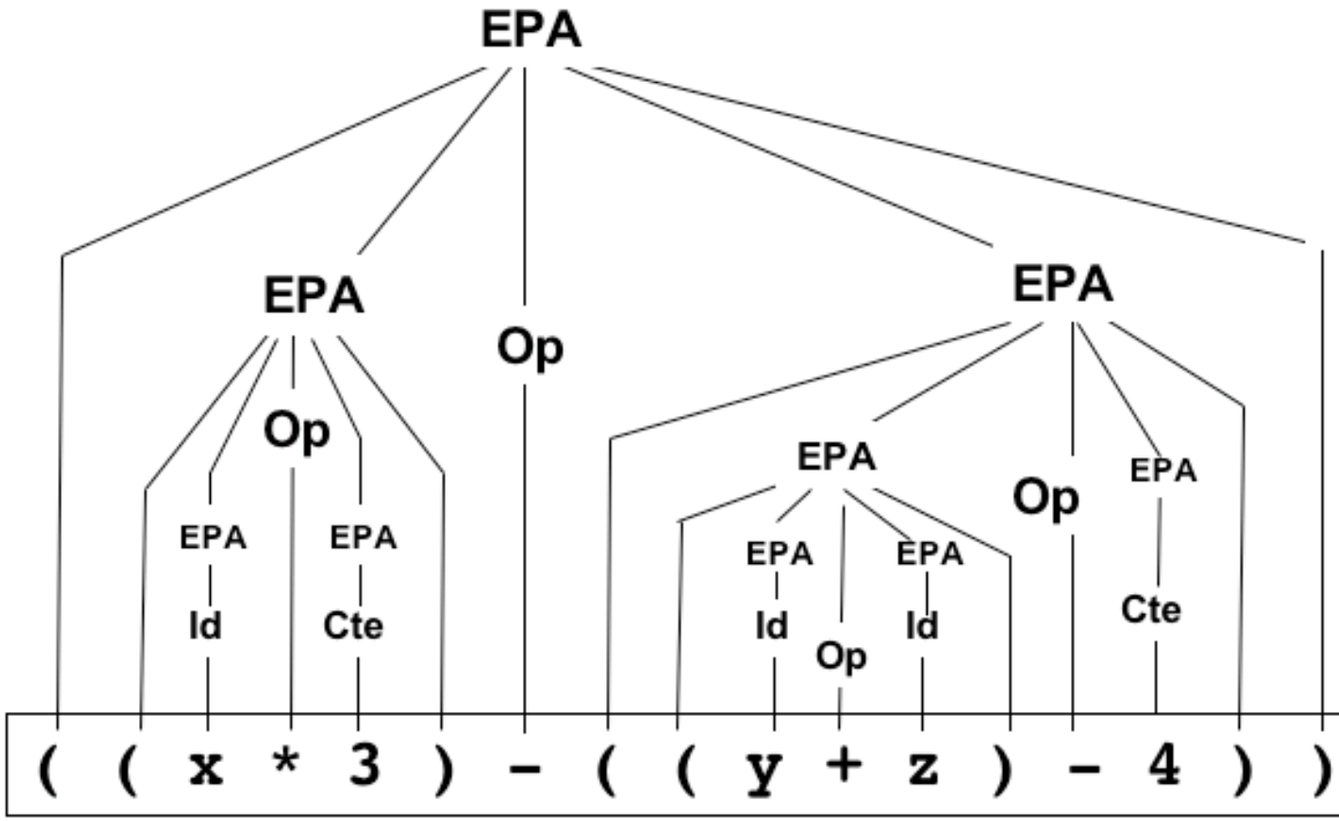
((x * 3) - ((y + z) - 4)) ?

Grammaire

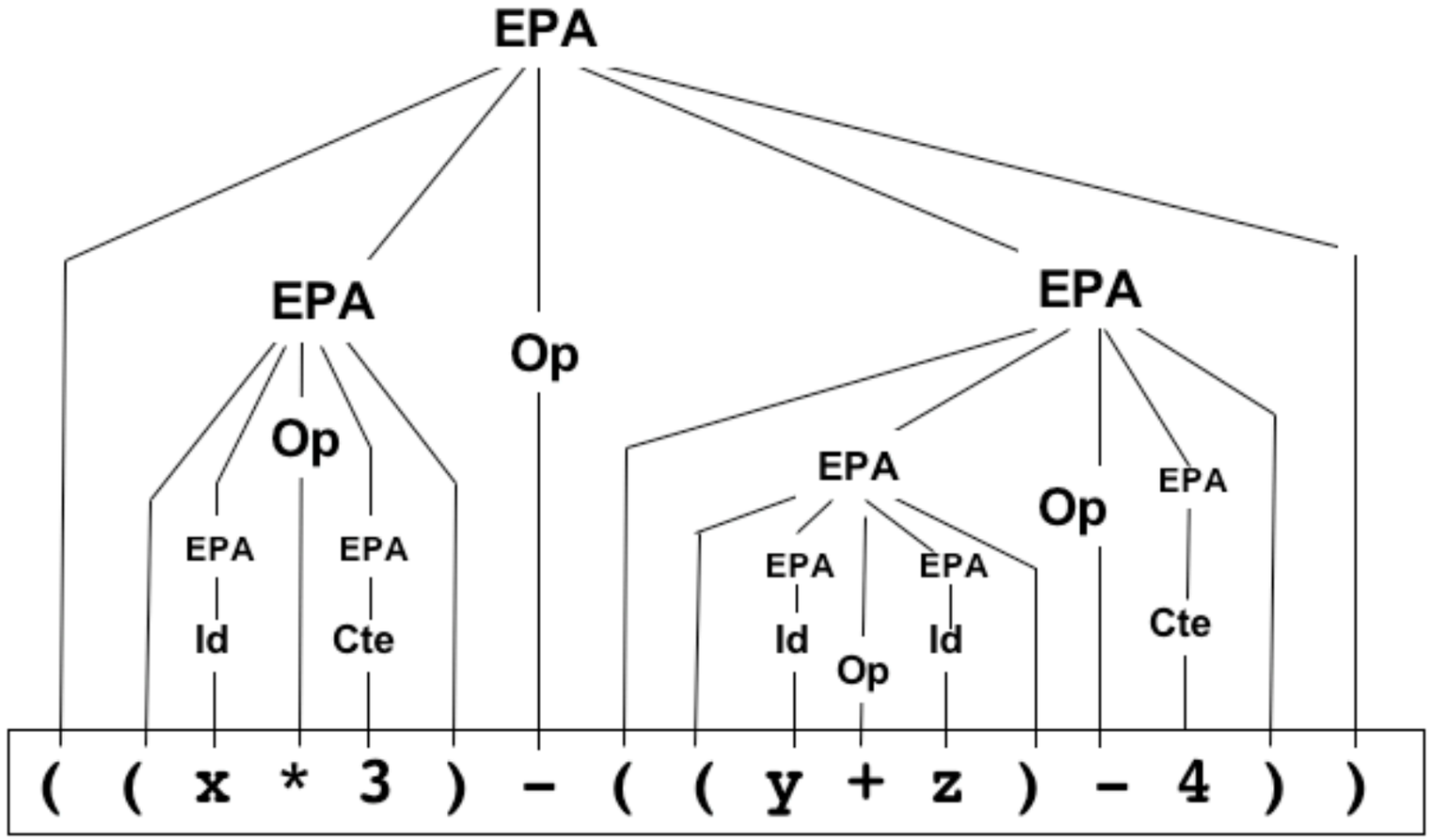
- EPA ::= Id
- EPA ::= Cte
- EPA ::= (EPA Op EPA)
- Id ::= a | b | aa | ba | ...
- Cte ::= 0 | 1 | 2 | ... | 9 | 10 | ...
- Op ::= + | - | * | /

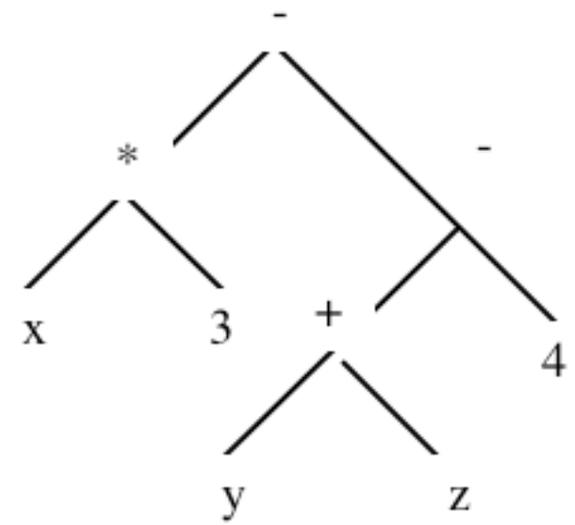
Grammaire
Arbre de syntaxe

- EPA ::= Id
- EPA ::= Cte
- EPA ::= (EPA Op EPA)
- Id ::= a | b | aa | ba | ...
- Cte ::= 0 | 1 | 2 | ... | 9 | 10 | ...
- Op ::= + | - | * | /



Arbre de syntaxe abstraite





AST
(Abstract Syntax Tree)

Quelques Pythoneries

Variables "privées"

- Dans une classe :
  - `_champ` : **convention** pour indiquer que la variable est privée et ne doit pas être utilisée à l'extérieur de la classe
  - `__champ` : **une transformation** est effectuée pour empêcher d'utiliser la variable à l'extérieur de la classe

getters, setters

```
class Point:
  def __init__(self,x):
    self.__x = x

def get_x(self):
  return self.__x

def set_x(self, new_x):
  self.__x = new_x
```

Tests unitaires avec assert

```
class Point:
    ...

if __name__ == '__main__':
    p = Point.origin()
    assert(p.x == 0)
```

Tests unitaires avec unittest

point.py

```
class Point:

  ...
```

test_point.py

```
import unittest
from point import Point
class TestPoint(unittest.TestCase):

  def test_create_origin(self):
    p = Point.origin()
    self.assertEqual(p.x, 0)

  def test_move(self):
    p = Point.origin()
    p.move(5)
    self.assertEqual(p.x, 5)
    self.assertTrue(p.x == 5)

if __name__ == '__main__':
  unittest.main()
```

Intérêts de `unittest`

- Tous les tests sont exécutés
- On voit rapidement quel test échoue
- Les messages d'erreur sont plus clairs (on peut voir le résultat attendu)

Comment programmer un arbre en Python ?

Définition d'une classe

```
class Tree:

    def __init__(self, symbol, *children):

        self.__symbol = symbol

        self.__children = children
```

- Utilisation : `Tree('a')` ou `Tree('a',Tree('b'))`
- `children` est un tuple de `Tree`

*children

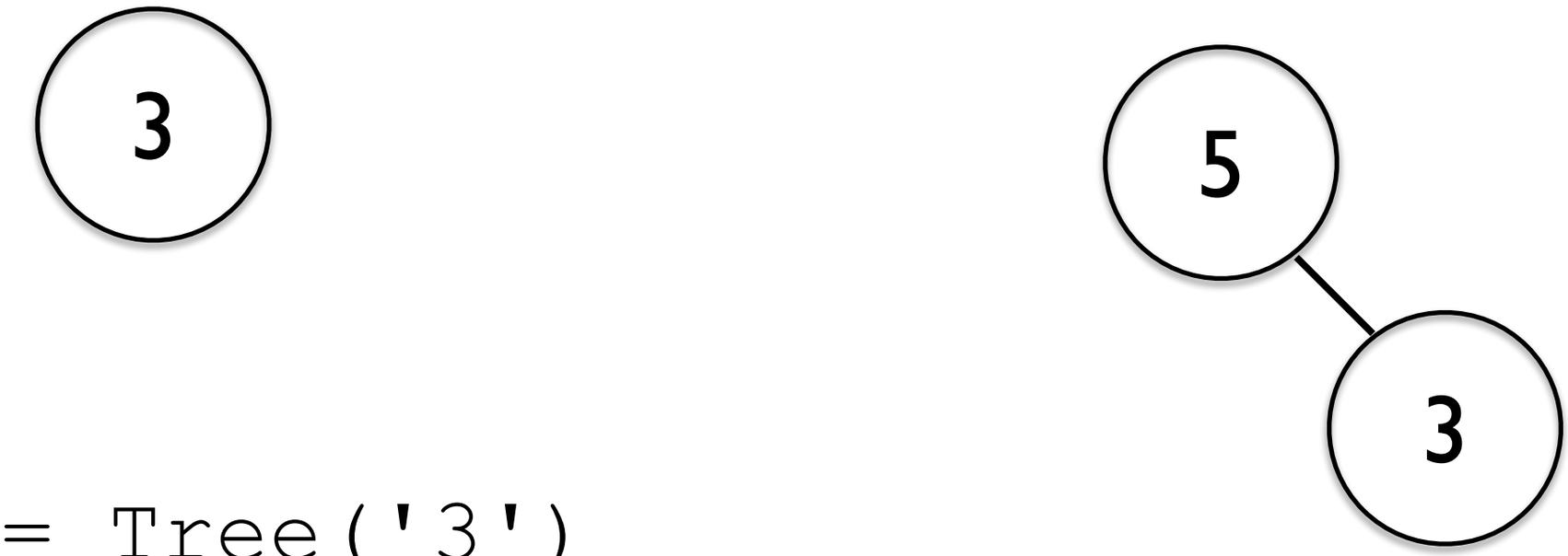
```
class Tree:

    def __init__(self, symbol, *children):
```

- Si L est une liste : `[Tree('a'),Tree('b')]`
- Comment construire l'arbre `f(a,b)` ?

Comment créer un arbre


```
t1 = Tree('3')
```

Comment créer un arbre




```
t1 = Tree('3')
```

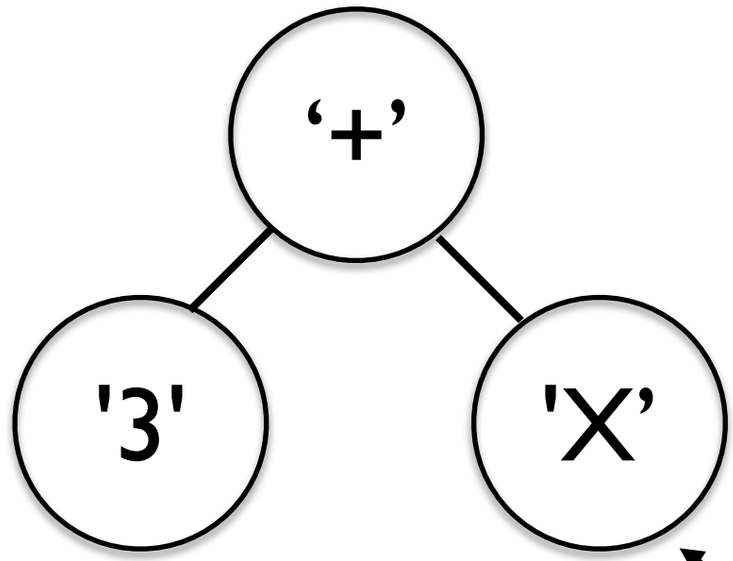
```
t2 = Tree('5', t1)
```

Le label peut être un entier ou une chaine

Comment représenter une expression avec un arbre

3 + x

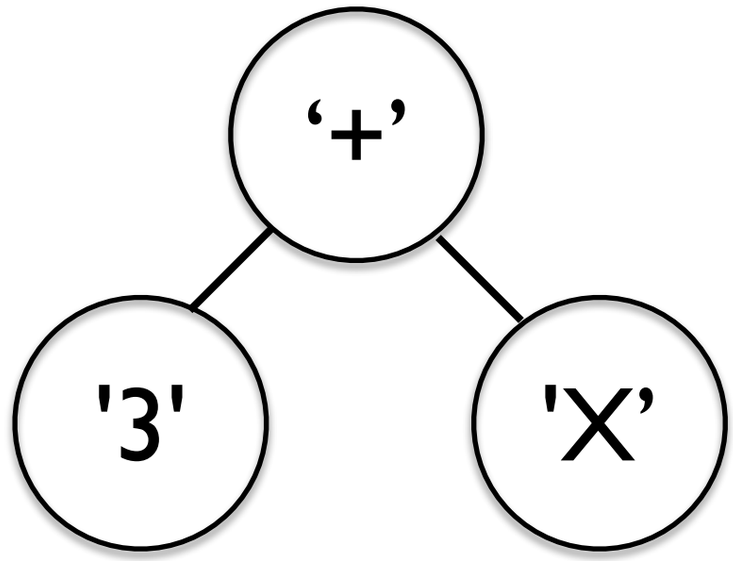
X est une variable mathématique



noeud dont le label est 'X'

Comment représenter une expression avec un arbre

3 + x




```
expression =
 Tree(‘+’,
   Tree('3'), Tree('X’))
```