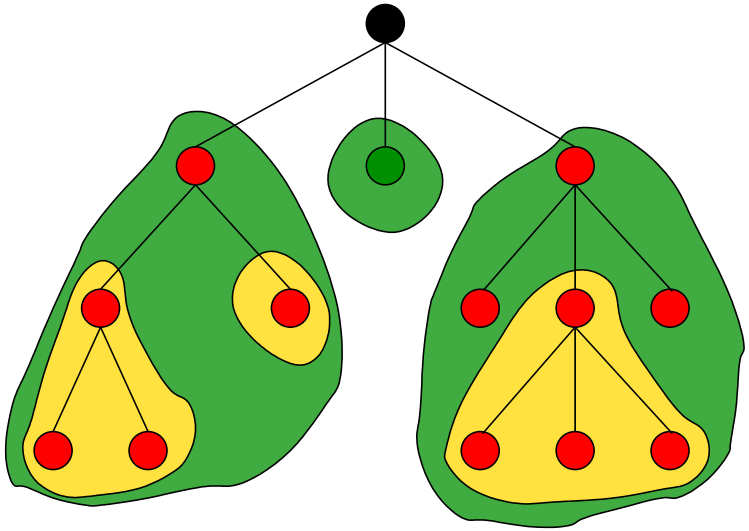
Cours : Structures de données arborescentes partie 1

Jean-Stéphane Varré

Université Lille 1

jean-stephane.varre@univ-lille1.fr

Qu'est-ce qu'un arbre ?



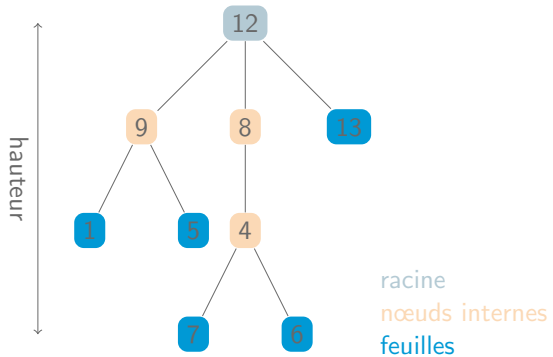
En résumé

- structure de données non linéaire, naturellement récursive,
- organisation hiérarchique entre les données stockées,
- utilisée pour structurer des données :
  - système de fichiers,
  - base de données,
  - sites web,
  - fichiers XML.

Vocabulaire

- nœud : caractérisé par une donnée associée + un nombre fini de fils, possède un unique père
- feuille : nœud sans fils
- nœud interne : un nœud qui n'est pas une feuille
- arité d'un nœud n : nombre de fils du nœud n
- arité d'un arbre a : nombre maximal de fils d'un nœud de a
- racine d'un arbre a : c'est le seul nœud sans père
- profondeur d'un nœud n : nombre de nœuds sur la branche entre la racine et le nœud n exclu
- hauteur d'un arbre a : c'est le nombre de nœuds sur la branche qui va de la racine de a à la feuille de profondeur maximale

Exemple



Hauteur, profondeur, arité

Soit A un arbre d'arité $a \geq 2$, de taille n et de hauteur h.

- le nombre n_p de nœuds de A à profondeur $0 \leq p \leq h$ vérifie

$$1 \leq n_p \leq a^p$$

- la taille n vérifie l'encadrement

$$h+1 \leq n \leq \frac{a^{h+1}-1}{a-1}$$

- le hauteur h vérifie l'encadrement

$$\log_a(n(a-1)+1)-1 \leq h \leq n-1$$

Si $a=2$, $h+1 \leq n \leq 2^{h+1}-1$ et $\log_2(n+1)-1 \leq h \leq n-1$.

Arbre binaire

Definition

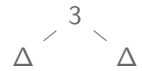
Un arbre binaire est

- soit l'arbre vide, noté Δ ;
- soit un triplet (e, g, d), appelé nœud, dans lequel
  - e est l'élément, ou encore étiquette, de la racine de l'arbre ;
  - g est le sous-arbre gauche de l'arbre ;
  - et d est le sous-arbre droit de l'arbre.

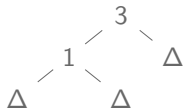
Les sous-arbres gauche et droit d'un arbre binaire non vide sont eux-mêmes des arbres binaires. La structure d'arbre binaire est donc une structure récursive.

Exemples de représentation des arbres

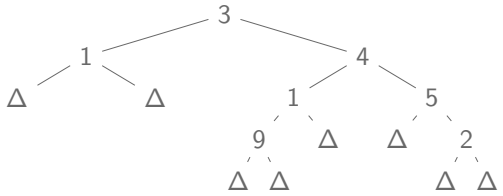
1. $(3, \Delta, \Delta)$ pour l'arbre



2. $(3, (1, \Delta, \Delta), \Delta)$ pour l'arbre



3. $(3, (1, \Delta, \Delta), (4, (1, (9, \Delta, \Delta), \Delta), (5, \Delta, (2, \Delta, \Delta))))$



Implantation

En Python :

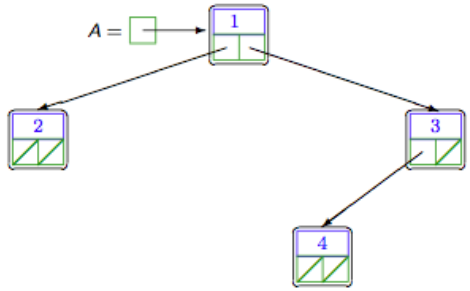
- une implantation non mutable, qui suit la définition précédente

```
1  from collections import namedtuple
2  __Tree = namedtuple ('Tree', ['tag', 'left', 'right'])
3
4  def empty_tree ():
5    return None
6
7  """
8      :type e: any
9      :type l: __Tree
10     :type r: __Tree
11 """
12 def cons (e,l,r):
13   return __Tree (tag=e, left=l, right=r)
```

- une implantation mutable, c'est-à-dire qui permet de modifier les sous-arbres d'un nœud, sera faite comme classiquement avec des dictionnaires

Une implantation en C sera vue en PdC.

Représentation chaînée



Crédits : E.W.

Opérations primitives

```
1  exception EmptyTree
```

Les constructeurs :

```
1  empty_tree ()
2
3  cons (t,l,r)
```

Les sélecteurs :

```
1  """ Return the tag of the root of t"""
2  def root (t)
3
4  """ Return the left subtree of t"""
5  def left (t)
6
7  """ Return the right subtree of t"""
8  def right (t)
```

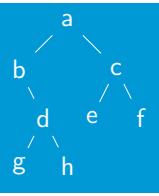
déclenchent l'exception `:EmptyTree`

Un prédicat :

```
1  def is_empty (t)
```


Parcours en profondeur

- On traite récursivement les nœuds de l'arbre.
- Trois sens de parcours :
  - préfixé : traiter la racine, puis le sous-arbre gauche, puis le sous-arbre droit ;
  - postfixé : traiter le sous-arbre gauche, puis le sous-arbre droit, puis la racine ;
  - infixé : traiter le sous-arbre gauche, puis la racine, puis le sous-arbre droit.



infixé	b, g, d, h, a, e, c, f
postfixé	g, h, d, b, e, f, c, a
préfixé	a, b, d, g, h, c, e, f

Parcours en profondeur - récursif

```
procedure AffichagePrefixe(a)
    if a n'est pas vide then
        (* traitement de la racine *)
        afficher l'étiquette de la racine
        (* traitement des fils gauche et droit *)
        AffichagePrefixe(Gauche(a))
        AffichagePrefixe(Droit(a))
    end if
end procedure
```

Note : pour changer le type de parcours il suffit d'échanger les trois instructions du `if`.

Comment dérécursiver le parcours ?

Parcours en profondeur - itératif avec pile

```
procedure AffichagePrefixe(a)
    soit p une pile d'arbres
    empiler a dans p
    while p n'est pas vide do
        (* on traite l'arbre au sommet de la pile *)
        s ← sommet de p
        dépiler p
        if s n'est pas vide then
            afficher l'étiquette de la racine de s
            empiler le Droit(s) dans p
            empiler le Gauche(s) dans p
        end if
    end while
end procedure
```

Il ne faut pas se tromper et bien empiler le sous-arbre droit avant le sous-arbre gauche car la pile est une structure LIFO.

Parcours en largeur

On traite les nœuds, des moins profonds aux plus profonds, par strates.

Comment réaliser un tel parcours ?

Parcours en largeur - itératif avec file

```
procedure AffichageEnLargeur(a)
    soit f une file d'arbres
    enfile a dans f
    while f n'est pas vide do
        s ← tête de f
        défiler f
        if s n'est pas vide then
            afficher la racine de s
            enfiler le Gauche(s) dans f
            enfiler le Droit(s) dans f
        end if
    end while
end procedure
```

Calcul de la taille d'un arbre binaire

```
procedure Taille(a)
    if a est vide then
        return 0
    else
        t_g ← Taille(Gauche(a))
        t_d ← Taille(Droit(a))
        return 1 + t_g + t_d
    end if
end procedure
```


Calcul de la hauteur d'un arbre binaire

Par convention, on aura $h(\Delta) = -1$.

```
procedure Hauteur(a)
    if a est vide then
        return −1
    else
        h_g ← Hauteur(Gauche(a))
        h_d ← Hauteur(Droit(a))
        return 1 + max(h_g, h_d)
    end if
end procedure
```


Construction d'arbres binaires

A partir d'une liste d'étiquettes, comment construire un arbre ?

Structures possibles pour 3 étiquettes :

Pouvant être étiquetés de 3! = 6 manières différentes chacun, pour 3 étiquettes différentes.

Un arbre est équilibré s'il est vide, ou bien si

1. ses deux sous-arbres sont équilibrés ;
2. et les tailles de ses deux sous-arbres ne diffèrent que d'1 au plus

En corollaire, dans un arbre équilibré, la hauteur des sous-arbres gauche et droit diffère au plus de 1.

Construction d'un arbre équilibré à partir d'une liste

```
procedure CreerArbreEquilibre(l)
    if l est vide then
        return Δ
    else
        soient x la tête de l, et l′ son reste
        soit (l_1, l_2) le couple de listes obtenu en séparant l en deux listes de
longueurs égales (à 1 près)
        g ← CreerArbreEquilibre(l_1)
        d ← CreerArbreEquilibre(l_2)
        return creer(x, g, d)
    end if
end procedure
```

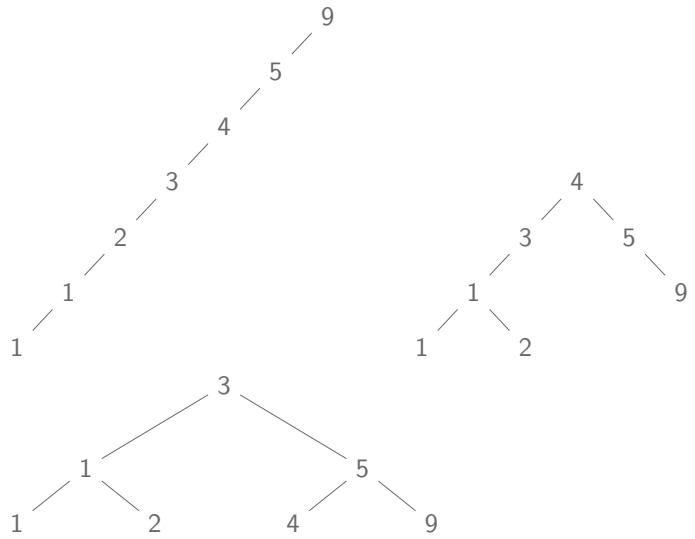
Définition des arbres binaires de recherche

Definition

Un arbre binaire de recherche (ou ordonné) est un arbre vide, ou un arbre binaire satisfaisant les trois conditions suivantes

1. l'élément étiquetant la racine de l'arbre est supérieur ou égal à tous les éléments étiquetant les nœuds du sous-arbre gauche ;
2. l'élément étiquetant la racine de l'arbre est inférieur à tous les éléments étiquetant les nœuds du sous-arbre droit ;
3. les deux sous-arbres sont eux-mêmes des arbres binaires de recherche.

Exemple



Est-ce un arbre binaire de recherche ?

Require: un arbre binaire a
Ensure: Vrai si a est un ABR, Faux sinon
  if a est vide **then**
    return Vrai
  end if
  if le sous-arbre gauche de a n'est pas un ABR **then**
    return Faux
  end if
  if le sous-arbre droit de a n'est pas un ABR **then**
    return Faux
  end if
  if la racine de a n'est pas supérieure à tous les éléments du sous-arbre gauche **then**
    return Faux
  end if
  if la racine de a n'est pas inférieure à tous les éléments du sous-arbre droit **then**
    return Faux
  end if
  return Vrai

Propriétés d'un ABR

- la plus petite étiquette se trouve dans le nœud le plus à gauche de l'arbre
- la plus grande étiquette se trouve dans le nœud le plus à droite de l'arbre

La recherche de la plus grande ou de la plus petite étiquette se fait en $\mathcal{O}(h)$. Si l'arbre est équilibré alors c'est en $\Theta(\log(n))$.

- le parcours infixé d'un ABR aboutit à une liste croissante des valeurs stockées aux nœuds de l'ABR.

? Question

Où se trouve la valeur médiane dans un ABR ?

Recherche dans un ABR

```
procedure RECHERCHER(e, a)
Ensure: un sous-arbre de a dont la racine est étiquetée par e s'il en existe,
  l'arbre vide sinon
    if a est vide then
        return Δ
    else
        r ← RACINE(a)
        if e = r then
            return a
        else
            if e ≤ r then
                return RECHERCHER(e,GAUCHE(a))
            else
                return RECHERCHER(e,DROIT(a))
            end if
        end if
    end if
end procedure
```


Recherche des valeurs extrêmes

[Rappel] Soit a un ABR non vide.

1. la plus petite valeur des nœuds de a est dans le nœud le plus à gauche de a.
2. la plus grande valeur des nœuds de a est dans le nœud le plus à droite de a.

```
procedure ETIQUETTEMAX(a)
    if le sous-arbre droit de a est vide then
        return RACINE(a)
    else
        return ETIQUETTEMAX(FILSDROIT(a))
    end if
end procedure
```


Insertion dans un ABR

Pour insérer dans un ABR il suffit de faire une recherche et d'insérer à l'endroit où la recherche infructueuse nous a amené.

```
procedure INSERER(e, a)
    if a est vide then
        return CREER(e, Δ, Δ)
    else
        soient r = racine(a), g = FILSGAUCHE(a) et d = FILSDROIT(a)
        if e ≤ r then
            a_g ← INSERER(e,g)
            remplacer le fils gauche de a par a_g
            return a
        else
            a_d ← INSERER(e,d)
            remplacer le fils droit de a par a_d
            return a
        end if
    end if
end procedure
```

Construction d'un ABR

```
procedure CONSTRUIRE(l)
    if l est vide then
        return Δ
    else
        soit x la tête de l et l' le reste de l
        a_r ← CONSTRUIRE(l')
        return INSERER(x,a_r)
    end if
end procedure
```