Arbres enracinés [tn] - Algorithmique

Karine Zampieri, Stéphane Rivière

Unisciel algoprog Version 21 mai 2018

Table des matières

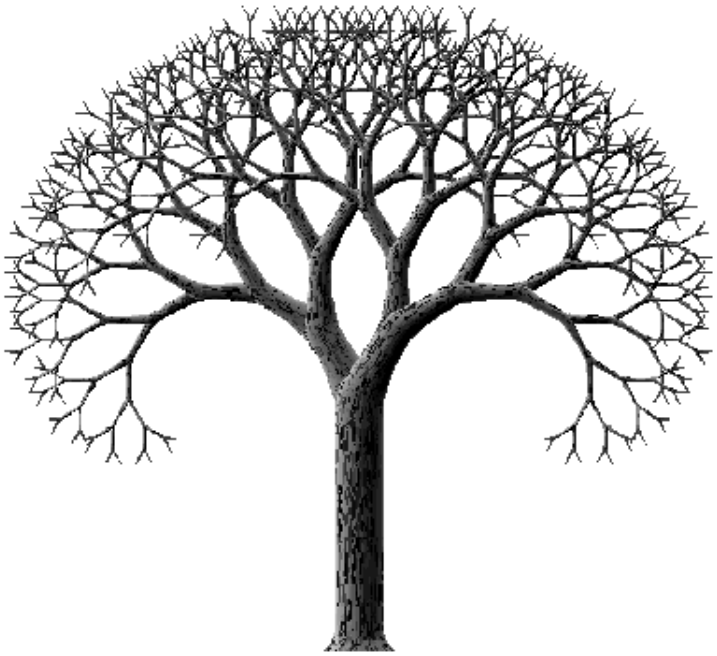
Arbres enracinés

Mots-Clés Arbres enracinés, Graphes, Algorithmique ■
Requis Axiomatique impérative, Récursivité des actions, Complexité des algorithmes ■
Difficulté ● ● ○

Introduction

La structure d'arbre trouve de nombreuses utilités en informatique : évaluation d'expressions algébriques, modélisation de l'inclusion d'ensembles, d'organisations hiérarchiques, de schémas en analyse, d'arbre généalogique, arborescence des dossiers dans un système d'exploitation, variables structurées C et en Cobol...

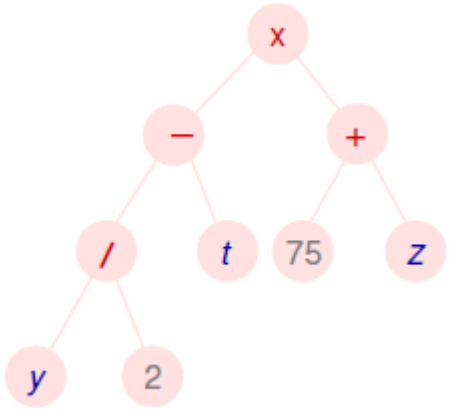
Ce module présente quelques exemples de données arborescentes, les définitions et la terminologie concernant les **arbres enracinés**, le cas particulier des **arbres binaires ordonnés** puis les **parcours des arbres**.



1 Exemples de données arborescentes

Expressions arithmétiques

Elles peuvent être représentées par des arbres étiquetés par des *opérateurs*, des *constantes* et des *variables*. La structure de l'arbre rend compte de la priorité des opérateurs et rend inutile tout parenthésage. Par exemple, l'arbre suivant correspond à l'expression arithmétique $(y/2 - t)(75 + z)$.

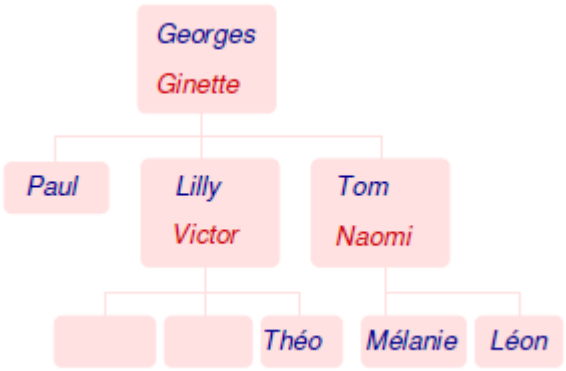


Arbre syntaxique

Il représente l'analyse d'une phrase à partir d'un ensemble de règles qui constitue la *grammaire* : une *phrase* est composée d'un *groupe nominal* suivi d'un *groupe verbal*, un groupe nominal peut être constitué d'un *article* et d'un *nom commun*, ...

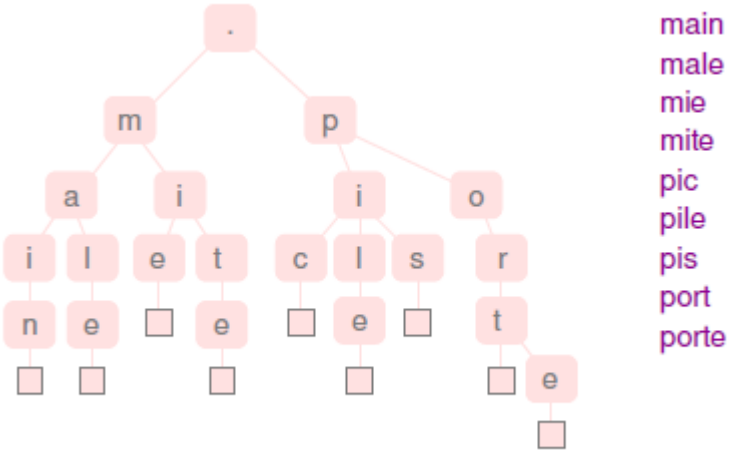
Arbre généalogique

Il représente la descendance (dans le cas présent) d'une personne ou d'un couple. Les noeuds de l'arbre sont étiquetés par les membres de la famille et leurs conjoints. L'arborescence est construite à partir des liens de parenté (les enfants du couple).



Arbre lexicographique (ou dictionnaire)

Il représente un ensemble de mots. Les préfixes communs à plusieurs mots apparaissent une seule fois dans l'arbre, ce qui se traduit par un gain d'espace mémoire. De plus la recherche d'un mot est assez efficace, puisqu'il suffit de parcourir une branche de l'arbre en partant de la racine, en cherchant à chaque niveau parmi les fils du noeud courant la lettre du mot de rang correspondant.



2 Définitions et terminologie

2.1 Définitions

Arbre enraciné, racine

Un **arbre enraciné**, ou **arborescence**, est une structure de données constituée de sommets assemblés par **niveaux** dans lequel l'un des sommets se distingue des autres : on appelle ce sommet la **racine**.

Attention

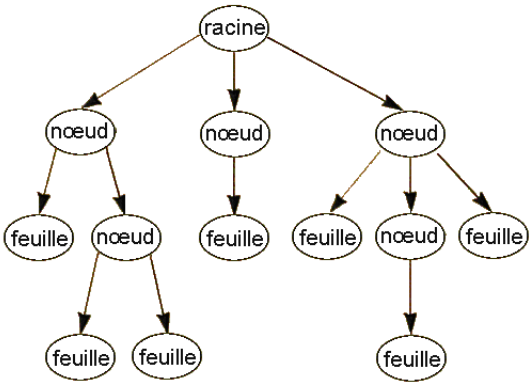
Dans la suite de ce module et sauf avis contraire, tous les arbres que nous manipulerons seront des arbres enracinés et nous omettrons de le préciser. En outre, on appellera souvent **noeuds** les sommets des arbres (enracinés).

Feuille, Noeud interne

Un noeud sans fils est appelé **noeud externe**, **noeud terminal** ou plus simplement une **feuille**. Un noeud qui n'est pas une feuille est un **noeud interne**.

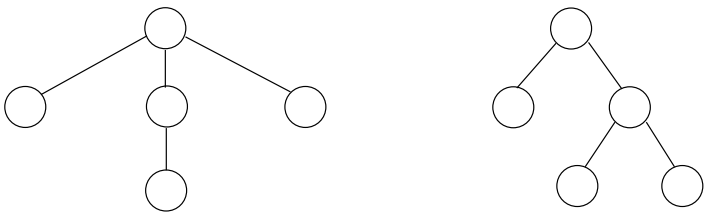
Représentation visuelle

Lorsqu'on dessine un arbre, on a l'habitude de le représenter avec la tête en bas, c.-à-d. que la racine est tout en haut et les noeuds fils sont représentés en-dessous du noeud père.



Arbre enraciné = Arbre orienté

En réalité, la racine impose un sens de parcours de l'arbre qui se retrouve orienté par l'utilisation qui en est faite. Ainsi, la figure suivante présente deux **arbres qui ne diffèrent que s'ils sont** considérés comme des arbres **enracinés**.

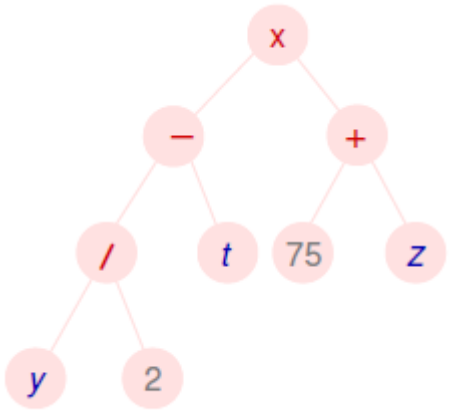


Représentation informatique

Chaque noeud porte une **étiquette** ou **valeur** ou **clé**. Un noeud est ainsi défini par son **étiquette** et ses **sous-arbres**. On peut donc représenter un arbre par un n-uplet $\langle e, a_1, ..., a_k \rangle$ dans lequel e est l'étiquette portée par le noeud, et a_1, ... , a_k sont ses sous-arbres.

Exemple

L'arbre correspondant à l'expression arithmétique $(y/2 - t)(75 + z)$:

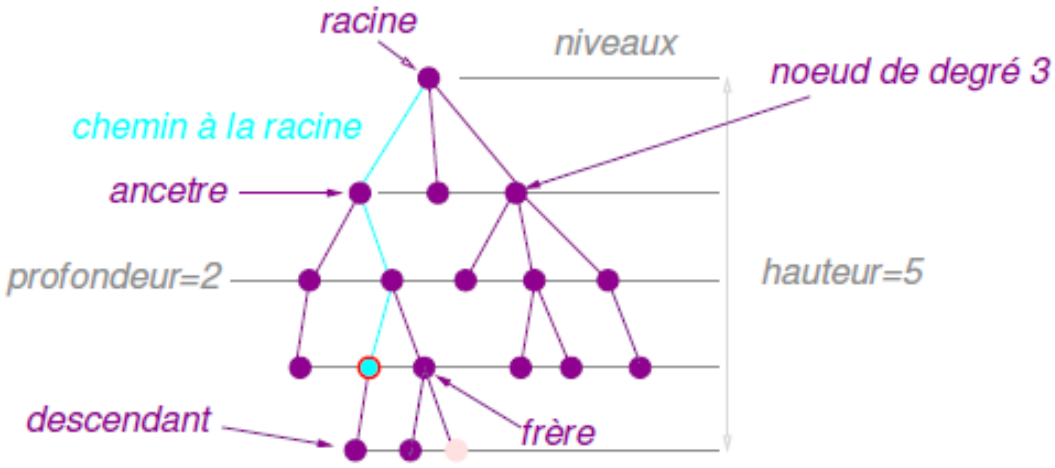


sera représenté par :

$$\langle \times, \langle -, \langle /, \langle y \rangle, \langle 2 \rangle \rangle, \langle t \rangle \rangle, \langle +, \langle 75 \rangle, \langle z \rangle \rangle \rangle$$

2.2 Terminologie

La terminologie est inspirée des liens de parenté.



Ancêtre, Chemin
Soit x un noeud d'un arbre T de racine r. Un noeud quelconque y sur **l'unique chemin** qui relie x à la racine r est appelé **ancêtre** de x.

Descendant, Sous-arbre
Si y est un ancêtre de x, alors x est un **descendant** de y. Le **sous-arbre de racine** x est l'arbre composé des descendants de x enraciné en x.

Père, Fils, Frère d'un noeud
Soit T un arbre de racine r. Si T contient l'arête (y, x) alors y est le **père** de x et x est le **fils** de y. La racine est le seul noeud qui n'ait pas de père. Un **frère** d'un noeud x est un fils du père de x et qui n'est pas x.

Degré d'un noeud
Le nombre de fils du noeud x est appelé le **degré** de x. Donc, selon qu'un arbre (enraciné) est vu comme un arbre (enraciné) ou un graphe, le degré de ses sommets n'a pas la même valeur !

Profondeur (ou niveau) d'un noeud
La longueur du chemin entre la racine r et le noeud x est la **profondeur** de x : le premier niveau (par convention, profondeur 0) contient la racine seulement, le deuxième niveau contient les deux fils de la racine, ..., les noeuds du niveau k sont les fils des noeuds du niveau $k-1$...

Hauteur de l'arbre

La plus grande profondeur que puisse avoir un noeud quelconque de l'arbre T est la **hauteur** de T. C'est donc aussi le nombre de noeuds qui jalonnent la branche la plus longue.

Définition de la hauteur

Elle varie en fonction des auteurs. Pour certains la hauteur d'un arbre contenant un seul noeud est -1.

Les arbres en théorie des graphes

En théorie des graphes, un arbre est un graphe connexe et sans cycles, c.-à-d. qu'entre deux sommets quelconques du graphe il existe un chemin et un seul. On parle dans ce cas d'arbre non planté, puisqu'il n'y a pas de sommet particulier qui joue le rôle de racine. Les sommets sont voisins les uns des autres, il n'y a pas de hiérarchie qui permet de dire qu'un sommet est le père (ou le fils) d'un autre sommet.

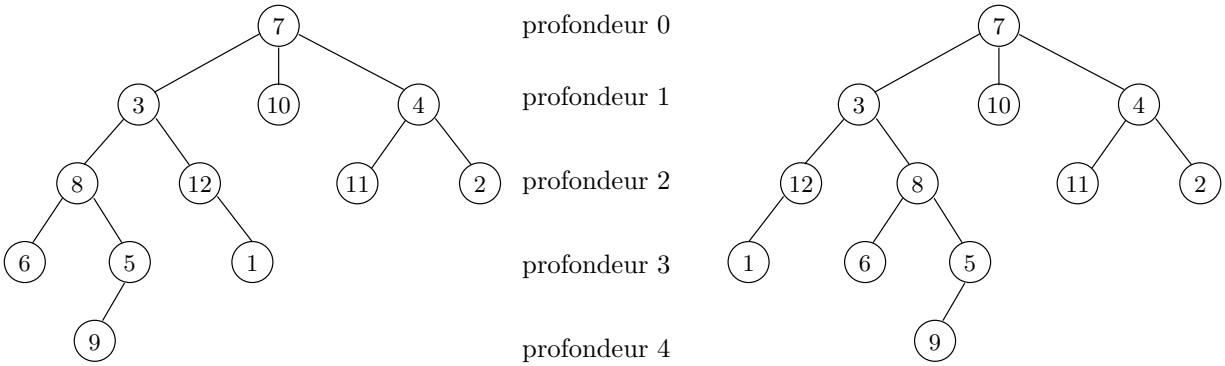
On démontre qu'un graphe de n sommets qui a cette propriété contient exactement $n-1$ arêtes, et que l'ajout d'une nouvelle arête crée un cycle, tandis que le retrait d'une arête le rend non connexe.

2.3 Arbres particuliers

Arbre ordonné

Un **arbre ordonné** est un arbre enraciné dans lequel les fils de chaque noeud sont ordonnés entre eux.

- Le sous-arbre enraciné en 8 contient les noeuds 8, 6, 5 et 9.
- Les deux arbres **ne diffèrent que s'ils sont ordonnés**. Ils sont identiques si on les regarde comme de simples arbres (enracinés).



Arbre dégénéré

Un arbre dont tout noeud possède au plus un descendant est appelé **arbre dégénéré**.

Arbre n-aire

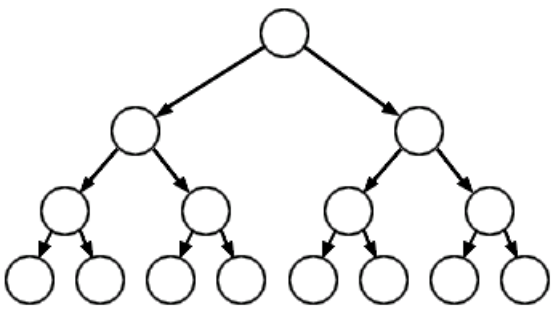
Un arbre dont tout noeud possède au plus un nombre déterminé n de descendants est appelé **arbre n-aire**.

Arbre complet

Si dans un arbre n-aire, toutes les feuilles ont le même niveau, et si la racine et tous les noeuds internes ont le même nombre de fils, l'arbre est alors **complet**.

Exemple

La figure montre un arbre « 2-aire » complet : chaque noeud non terminal à 2 fils, et toutes les feuilles sont au même niveau.



3 Arbres binaires

3.1 Définitions

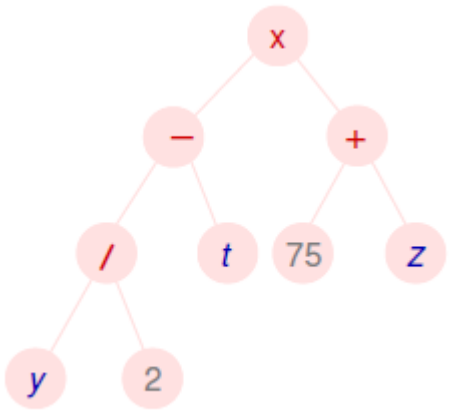
On distingue les **arbres binaires** (AB) des arbres généraux. Leur particularité est que les fils sont singularisés : chaque noeud a un **fils gauche** et un **fils droit**. Ils se décrivent plus aisément de manière récursive :

Définition récursive
Un **arbre binaire** est une structure définie sur un ensemble **fini** de noeuds qui est :

- Soit vide.
- Soit formé de trois ensembles disjoints de noeuds : une racine, un arbre binaire appelé son **sous-arbre gauche** et un arbre binaire appelé son **sous-arbre droit**.

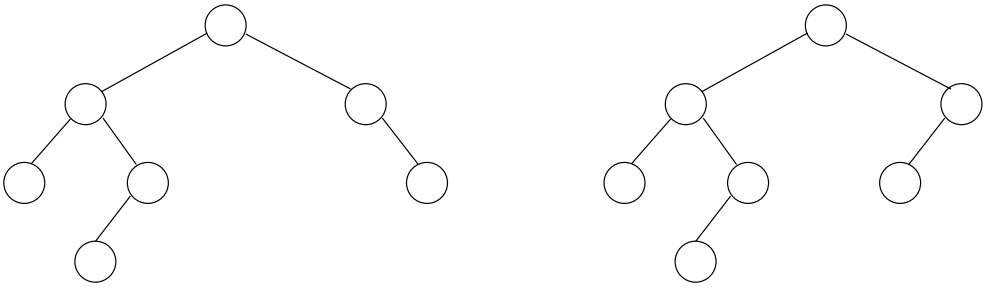
Attention
L'écriture d'un arbre s'en trouve modifiée puisqu'**un noeud a toujours deux fils**. En reprenant l'expression arithmétique $(y/2 - t)(75 + z)$, l'arbre binaire qui la représente s'écrit :

$$\langle \times, \langle -, \langle /, \langle y, \langle\rangle, \langle\rangle\rangle, \langle 2, \langle\rangle, \langle\rangle\rangle\rangle, \langle t, \langle\rangle, \langle\rangle\rangle\rangle, \langle +, \langle 75, \langle\rangle, \langle\rangle\rangle, \langle z, \langle\rangle, \langle\rangle\rangle\rangle\rangle$$



Attention
Un arbre binaire **est plus** qu'un arbre ordonné dont chaque noeud serait de degré au plus deux : dans un arbre binaire, si un noeud n'a qu'un seul fils, la position de ce fils — qu'il soit **fils gauche** ou **fils droit** — est importante.

Exemple
La figure présente deux arbres enracinés **qui ne diffèrent** que quand ils sont vus comme des **arbres binaires**.



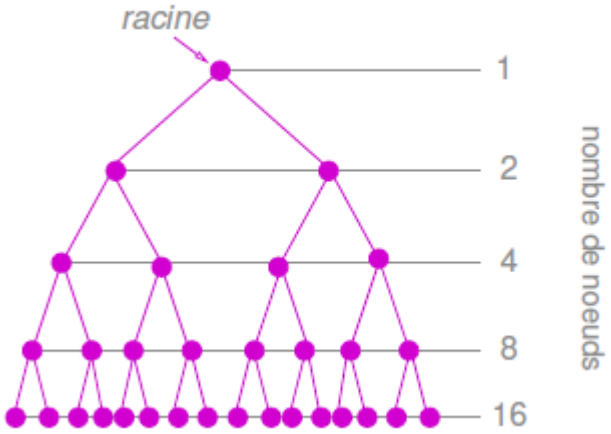
Arbre n-aire

Un **arbre** n**-aire** est une généralisation de la notion d'arbre binaire où chaque noeud est de degré au plus n et non plus simplement de degré au plus 2.

3.2 Quelques caractéristiques

Propriété
Le nombre de feuilles au niveau h d'un arbre binaire complet est 2^{h-1}.

Preuve
Soit A un arbre binaire complet.



Le nombre de noeuds de A au niveau 0 est 1, au niveau 1 est 2^1, au niveau 2 est 2^2, ..., et au niveau p est donc 2^p. ■

Propriété
Le nombre total de noeuds d'un arbre binaire complet de h niveaux est :

$$\sum_{i=0}^{h-1} 2^i = 2^h - 1$$

Corollaire
La hauteur $h(A)$ (le nombre de niveaux) d'un arbre binaire A contenant n noeuds est au moins égale à :

$$\lfloor \lg n \rfloor + 1$$

Preuve
Soit A est un arbre de hauteur h comportant n noeuds. Pour tout entier m, on a :

$$h \leq m - 1 \Rightarrow n < 2^{m-1}$$

Par contraposée, on a :

$$n \geq 2^{m-1} \Rightarrow h \geq m$$

Le plus grand entier m vérifiant $n \geq 2^{m-1}$ est $\lfloor \lg n \rfloor + 1$. On a donc :

$$h \geq \lfloor \lg n \rfloor + 1$$

■

Montrez que pour tout entier $n \geq 1$:

$$\lfloor \lg n \rfloor + 1 = \lceil \lg (n+1) \rceil$$

Conséquence

La hauteur minimale $\lfloor \lg n \rfloor + 1$ est atteinte lorsque l'arbre est complet. Pour être efficaces, les algorithmes qui utilisent des arbres binaires font en sorte que ceux ci soient équilibrés (voir les tas ou les AVL-arbres par exemple).

3.3 Fonctionnalités et Implémentation

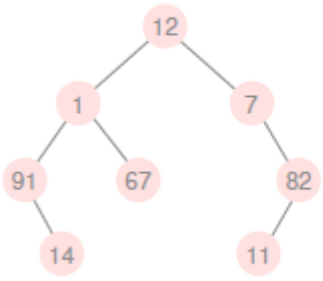
Fonctionnalités

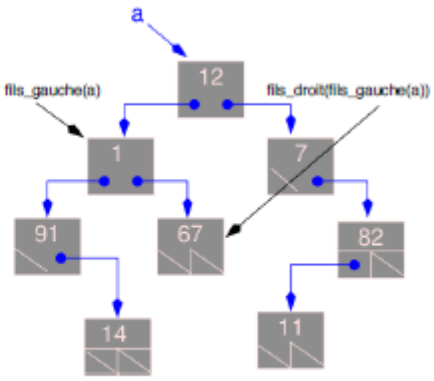
Pour pouvoir manipuler des arbres binaires, on doit disposer des fonctionnalités suivantes :

- `racine(A)`
- `filsGauche(A)`
- `filsDroit(A)`
- `val(A)`

Implémentation

C/C++ On utilise en général des pointeurs pour représenter les liens entre un noeud et ses fils.





Python On peut représenter un arbre vide par une liste vide `[]` et un arbre non vide par une liste comprenant 3 éléments `[clé,filsgauche,filsdroit]`.

4 Parcours des arbres

4.1 Parcours en profondeur

Les parcours permettent d'effectuer tout un ensemble de traitement sur les arbres. Le parcours le plus simple à programmer est celui dit en **profondeur d'abord**.

Parcours en profondeur d'abord

On descend le plus profondément possible dans l'arbre puis, une fois qu'une feuille a été atteinte, on remonte pour explorer les autres branches en commençant par la branche « la plus basse » parmi celles non encore parcourues ; les fils d'un noeud sont bien évidemment parcourus suivant l'ordre sur l'arbre.

Algorithme PP

```
PP(A)
Début
  Si A n'est pas réduit à une feuille Faire
    Pour tous les fils u de racine(A) Faire
      Dans l'ordre PP(u)
    FinPour
  FinSi
Fin
```

Cas d'un arbre binaire non vide

On parcourt récursivement son sous-arbre gauche, puis son sous-arbre droit, la racine de l'arbre pouvant être traitée au début, entre les deux parcours ou à la fin. Dans le premier cas, on dit que les noeuds sont traités dans un ordre **préfixe**, dans le second cas dans un ordre **infixe** et dans le troisième cas dans un ordre **postfixe**.

Algorithme Parcours

```
Parcours(AB a)
Début
  Si Non estvide(A)
    TraitementPrefixe(val(A))
    Parcours(filsGauche(A))
    TraitementInfixe(val(A))
    Parcours(filsDroit(A))
    TraitementPostfixe(val(A))
  FinSi
Fin
```

Affichage des valeurs d'un arbre binaire

Les trois algorithmes suivants affichent les valeurs contenues dans les noeuds d'un arbre binaire, selon des parcours en profondeur préfixe, infixe et postfixe respectivement.

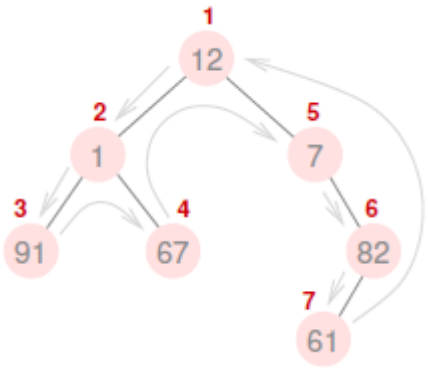
Parcours préfixe d'un arbre binaire

```
Préfixe(A)
Début
  Si A <> Nil Faire // Non estvide(A)
    Afficher(racine(A))
    Préfixe(filsGauche(A))
    Préfixe(filsDroit(A))
  FinSi
Fin
```

Exemple

Pour l'arbre ci-dessous, un affichage préfixe donnerait :

```
12 1 91 67 7 82 61
```



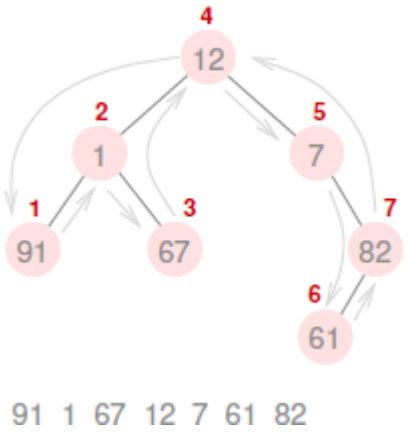
Parcours infixe d'un arbre binaire

```
Infixe(A)
Début
  Si A <> Nil Faire
    Infixe(filsGauche(A))
    Afficher(racine(A))
    Infixe(filsDroit(A))
  FinSi
Fin
```

Exemple

Pour l'arbre exemple :

```
91 1 67 12 7 61 82
```



91 1 67 12 7 61 82

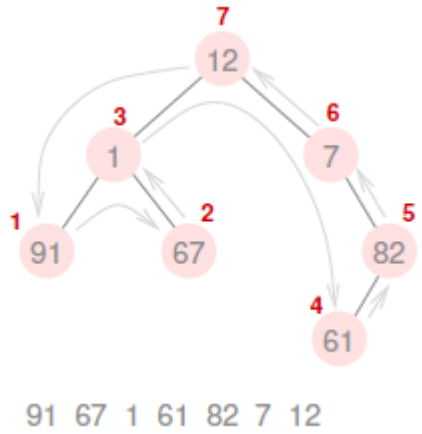
Parcours postfixe d'un arbre binaire

```
Postfixe(A)
Début
  Si A <> Nil Faire
    Postfixe(filsGauche(A))
    Postfixe(filsDroit(A))
    Afficher(racine(A))
  FinSi
Fin
```

Exemple

Pour l'arbre exemple :

```
91 67 1 61 82 7 12
```



91 67 1 61 82 7 12

4.2 Parcours en largeur

Dans un **parcours en largeur d'abord**, tous les noeuds à une profondeur i doivent avoir été visités avant que le premier noeud à la profondeur $i+1$ ne soit visité. Un tel parcours nécessite que l'on se souvienne de l'ensemble des branches qu'il reste à visiter. Pour ce faire, on utilise une *file* (ici notée F).

Parcours en largeur d'un arbre

```
PL(A)
Début
  F <- enfiler(racine(A))
  TantQue non FileVide(F) Faire
    u <- defiler(F)
    Pour tous les fils v de u faire
      Dans l'ordre enfiler(F, v)
    FinPour
Fin
```