**Modèle** : Une représentation du monde réel. En physique et en maths, il prend ses valeurs dans des espaces denses, en informatique dans des espaces discrets finis.

**Problème/Algorithme/Programme** Ne pas confondre les 3 notions.

**Problème** : Du grec problema (jeté devant ie obstacle). Un problème est constitué d'une situation initiale et d'un but à atteindre pour la situation finale.
**Algorithme** : (Al Khwarizmi : 820 ap JC) Une façon systématique de procéder pour faire quelque chose en un nombre fini d'étapes élémentaires.

**Programme** : Mise en oeuvre concrète d'un algorithme dans un langage de programmation.

**Preuve de programme** : On peut prouver qu'il y a conformité entre le modèle mathématique et le programme, mais pas qu'il répond à la sémantique naturelle du problème (ie qu'il marche). Il faudrait pour cela être capable de tester TOUS les cas possibles.

**Syntaxe/Sémantique** : La syntaxe est maîtrisable, mais pas la sémantique.

**Ambiguité** : Les langages naturels sont ambigus ("Rogrigue, as tu du coeur?"), tandis que les langages de programmation ne le sont pas ⇒ déterminisme.

**Compilateur** :

1. **parsing** ou **analyse lexicale** : Assemble les symboles en groupes formant des expressions.
2. **analyse syntaxique** : Vérifie que les expressions sont grammaticalement correctes.
3. **table des symboles** : Visibilité des symboles, renommage, contrôle de type.
4. **génération de code** : syntaxe abstraite.

**Abstrait** : En informatique, abstrait signifie : on oublie certaines informations.
**Grammaire regex** : Fait elle partie des grammaires hors contexte ? (oui ?)

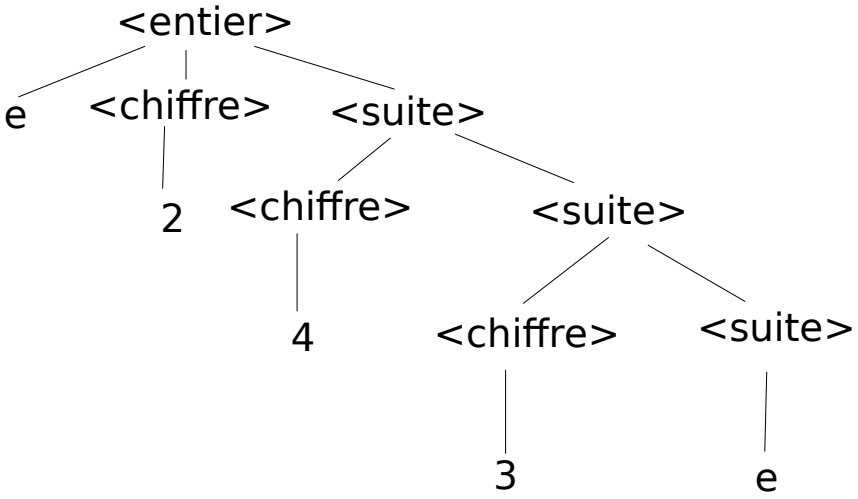
**Grammaire hors contexte** : Ne dépend pas du contexte. Elle est constituée de :

- Un ensemble $T$ de symboles terminaux (alphabet des symboles)
- Un ensemble $N$ de symboles non-terminaux ($T \cap N = \varnothing$) (alphabet des variables -> à voir comme une catégorie.
- Un ensemble $R$ de règles (couples $(x, y)$ notés $x \to y$, de mots sur $V \cup N$)
- Un symbole de départ $S \in N$

**Exemple** : Un entier

- $T = \{-, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- $N = \{< chiffre >, < entier >, < suite >\}$
- Règles :
  - $< entier >::= (-|\varepsilon) < origine >< suite >$
  - $< suite >::= (0| < chiffre >) < suite > |\varepsilon$
  - $< chiffre >::= 1|2|3|4|5|6|7|8|9$
- $S =< entier >?$

Construction de l'arbre de 243 :


<entier>
e
<chiffre>
<suite>
2
<chiffre>
<suite>
4
<chiffre>
<suite>
3
e

**Théorie de la calculabilité** : Cactérisation des fonctions calculables sous forme d'un algorithme.

**Exemple** : Savoir à partir d'un couple (algorithme, données initiales) si l'algorithme se termine ou pas n'est pas calculable.

**Preuve** : L'ensemble des algorithmes est dénombrable (car un algorithme est un mot fini sur un alphabet fini), tandis que les fonctions ne sont pas dénombrable (théorème de Cantor).

**Théorème de Cantor** : Il n'existe pas de bijection entre $E$ et $P(E)$.

**Preuve** :

1. Il existe une injection triviale de $E$ dans $P(E)$
2. Supposons qu'il existe une bijection $f$ de $E$ dans $P(E)$ et considérons $D = \{x \in E \mid x \notin P(E)\}$.
   Comme $f$ est bijective, $\exists\, y \in E \mid f(y) = D$
   Si $y \in D$ alors $y \notin f(y) = D$, Si $y \notin D = f(y)$ alors $y \in D$
   Contradiction

**Deux calculs possibles** :

- Complexité en temps
- Complexité en espace (obsolète)

**Méthode** :

- On choisit des **opérations de référence** (celles qu'on va comptabiliser). Si on les choisit toutes, on dit qu'on réalise une **analyse fine**.
- On calcule le nombre d'opérations de référence -> fonction de $n$ (nombre de données à l'entrée)
- On donne un **ordre de grandeur asymptotique** en $O(f(n))$ où $f \in \{\ln(n), n, n^2\}$

**Arbre binaire** : Structure de donnée soit $\varnothing$ soit $< o, A_g, A_d >$ avec :

- $o$ est un élément (noeud) de l'arbre appelé **racine**
- $A_g$ et $A_d$ sont des arbres binaires disjoints appelés sous arbre gauche et sous arbre droit.

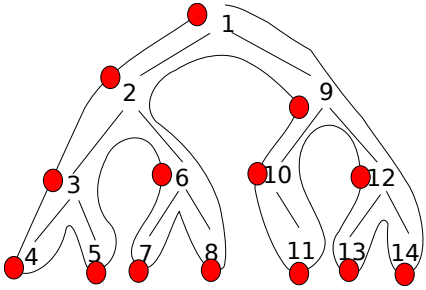
**Arbre étiqueté** : On crée une fonction de l'ensemble des noeuds dans un ensemble $A$ donné.

**Vocabulaire** : **Fils droit** ,**Fils gauche** ,**Point simple** ,**Point double** ,**Noeaud interne** ,**Feuille**

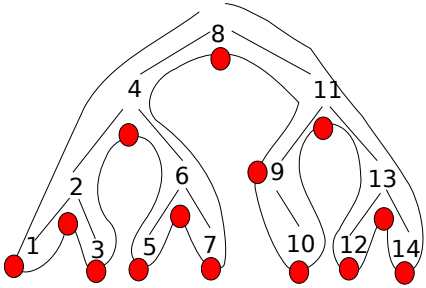
**Cas particuliers**

- **Arbre parfait**
- **Arbre complet**
- **Arbre dégénéré ou filiforme**
- **Localement complet** (pas de point simple)
- **Arbre trié**

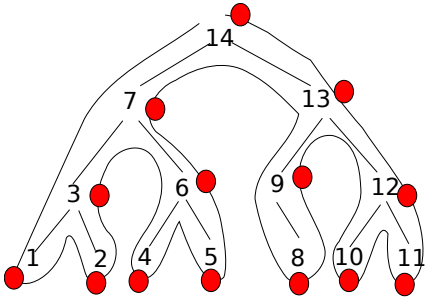
**Ordre préfixe** (pré-ordre) :



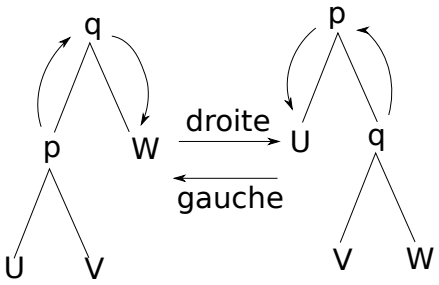

**Ordre infixe** :



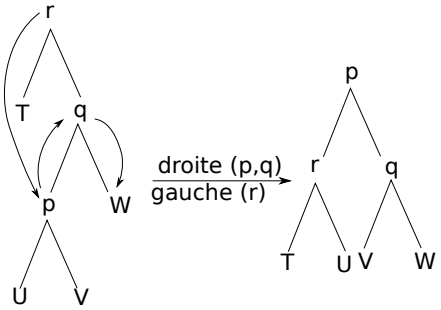

**Ordre suffixe** :



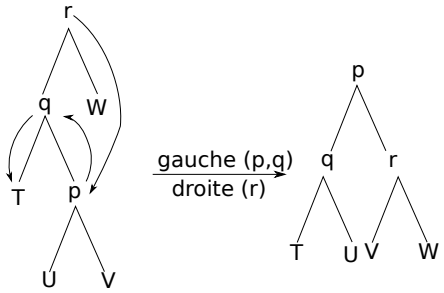
**Rotations simples** :




**Rotation droite-gauche** :




**Rotation gauche-droite** :



**Arbre AVL** : C'est un arbre trié dans lequel en tout noeud les hauteurs du sous arbre droit et du sous arbre gauche diffèrent d'au plus 1.

**Construction de l'arbre** : On place les éléments dans l'arbre trié.

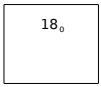
1. Si on crée un déséquilibre à l'extrême gauche on fait une rotation droite.
2. Si on crée un déséquilibre dans la partie gauche on fait une rotation gauche-droite.
3. Si on crée un déséquilibre à l'extrême droite on fait une rotation gauche.
4. Si on crée un déséquilibre dans la partie droite on fait une rotation droite-gauche.

**Choix du noeud** : On fait la rotation sur le noeud le plus bas parmis les noeuds en déséquilibre.

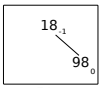
**Suppression d'un élément** On part du père de l'élément supprimé et on remonte jusqu'à la racine en faisant des rotations à chaque fois qu'on a un déséquilibre.

**Exemple** : $\{18, 98, 51, 70, 62, 13, 29, 33, 57, 61, 35, 15, 3\}$

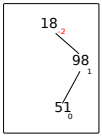
18



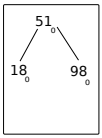

98



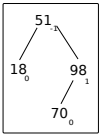

51



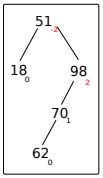

dg sur 18



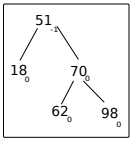

70



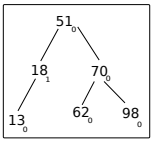

62



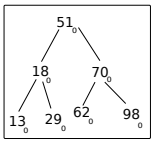

droite sur 98



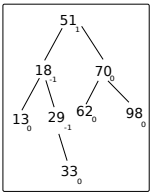

13



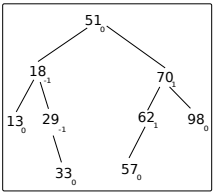

29



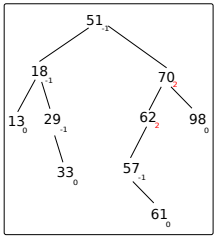

33



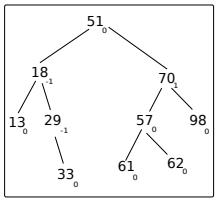

57



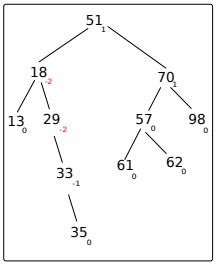

61



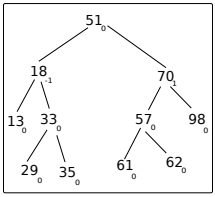

gd sur 62



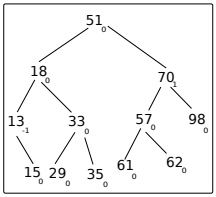

35




gauche sur 29




15




3