Licence académique Ingénierie des Systèmes Informatique (ISI)

Systèmes formels

http://www.larbiguezouli.com
ou
http://fac-sciences.univ-batna.dz/cs/enseignants/guezouli_larbi_site/index.html

Présenté par Dr Larbi GUEZOULI

Systèmes formels

- Chapitre I: Introduction [2 séances]
  1. Notion de système formel
  2. Exemples introductifs
  3. Définitions d'un système formel
  4. Définitions d'une preuve
  5. Définitions d'un théorème
- Chapitre II: Récursion et induction [2 séances]
  1. Récursion
     1. Utilisation des fonction récursives
     2. Formulation en système formel
  2. Induction
     1. Raisonnement pas récurrence sur N
     2. Définitions inductives
     3. Principe général d'une définition inductive

Systèmes formels

- Chapitre III: Système de réécriture [2 séances]
  1. Définitions
  2. Réécritures des mots et termes
- Chapitre IV: Type de données abstraits [2 séances]
  1. Définitions
  2. Implantation d'un TDA en C++
  3. Implantation d'un TDA en Java
- Chapitre V: Graphes et arbres [2 séances]
  1. Graphes
     - Représentation informatique des graphes
     - Définitions
     - Connexité
  2. Arbres
     - Propriétés
     - Forêt
     - Arborescence
     - Propriétés des arbres binaires
     - Représentation informatique des arbres

Mode d'évaluation :

Examen final : coef 2
Contrôle Continu : coef 1

Chapitre I
Introduction

Plan

1. Notion de système formel
2. Exemples introductifs
3. Définitions d'un système formel
4. Définitions d'une preuve
5. Définitions d'un théorème

CHAPITRE I
INTRODUCTION

1. Notion de système formel

- Garder le même raisonnement syntaxique sans prendre en considération la sémantique.

CHAPITRE I
INTRODUCTION

2. Exemples introductifs

Exp. 1

« Les hommes sont mortels. Abdallah est un homme. Donc Abdallah est mortel. »

Les mots « homme », « mortel » et « Abdallah » sont substituables par des symboles et le raisonnement reste valide.

Le modèle abstrait devient: « **Si tout X est Y. Si Z est X. Alors Z est Y** »

Tel que: X, Y et Z sont des variables (symboles)

"Si", "tout" et "Alors" sont des opérateurs

CHAPITRE I
INTRODUCTION

2. Exemples introductifs (suite)

Exp. 2

Soit un jeux simple de dominos. On pose les pièces de gauche à droite sur une seule ligne.

1	2	2	3	3	4

L'état du jeux est égale au dernier nombre à droite. (dans ce cas : 4).

Le jeux contient 5 pièces: <1,2>, <2,3>, <3,4>, <4,5>, <5,1>.

CHAPITRE I
INTRODUCTION

2. Exemples introductifs (suite)

Exp. 2 (suite)

Le joueur dépose une pièce quelconque au début. Puis le dépôt des pièces sera à droite tel que la partie gauche de la pièce déposé est égale à l'état du jeux.

La fin de la partie aura lieu dans les 3 cas:

- Il ne reste plus de pièces à déposer;
- On ne peux plus déposer de pièces;
- L'état du jeux = 1.

2. Exemples introductifs (suite)

Exp. 2 (suite)

Une partie de ce jeux sera formalisée mathématiquement pas une séquence d'états, par exemple:

2 -> 3 -> 4 -> 5 -> 1

3. Définitions d'un système formel

Un **système formel** est un quadruplet

SF=<A, γ, Ax, R>

tel que:

A: est l'ensemble fini de symboles;

γ: procédé de construction de mots (ensemble de formules);

Ax: Ensemble fini de mots particuliers appelés Axiomes;

3. Définitions (suite)

R: Ensemble fini de règles de déduction (dérivation) de la forme:

$w_1,\ldots,w_k \rightarrow m_1,\ldots,m_n$ tel que w_i et m_i sont des mots.

Il existe 2 types de règles de dérivation:

- Règles de production: qui produisent de nouveaux mots à partir des mots initiaux;
- Règles de réécriture: qui donnent une nouvelle forme pour le même mot.

Exp.1

Le système axiomatique de Peano:

- Alphabet {x, y,*}
- Mots S_1*S_2 tel que S_1 et S_2 sont des suites quelconques de x ou y
- Axiome: x*x
- Une règle: w_1*$w_2 \rightarrow yw_1$*yw_2 avec w_1 et w_2 sont des mots.

Cette règle est une règle de production.

Exp.2

Le système de Douglas Hoffstadter:

- Alphabet {M,I,U}
- Mots: Toute suite de l'alphabet
- Axiome: MI
- Règles: soit "w" un mot

$wI \rightarrow wIU$	(Production)
$Mw \rightarrow Mww$	(Production)
$III \rightarrow U$	(Réécriture)
$UU \rightarrow$	(Réécriture)

4. Définitions d'une preuve

Dans un système formel une **preuve** (démonstration) est une suite finie de mots $w_1w_2...w_k$ où chaque w_i:

- Soit est un axiome
- Soit se déduit des w_j ($j<i$) par l'une des règles de production

5. Définitions d'un théorème

Dans un système formel un théorème est un mot T tel que: il existe une preuve $w_1w_2...w_k$
avec $T=w_k$ et on note: $\vdash T$

Exp.1

Dans le système axiomatique de Peano,
yyx*yyx est un théorème dont la preuve est:
x*x, yx*yx, yyx*yyx

Exp.1

Dans le système de Douglas Hoffstadter,
MIIUIIU est un théorème, et la preuve:
MI, MII (R2), MIIU (R1), MIIUIIU (R2)

CHAPITRE II
RÉCURSION ET INDUCTION

PLAN

1. Récursion
   1. Utilisation des fonction récursives
   2. Formulation en système formel
2. Induction
   1. Raisonnement pas récurrence sur N
   2. Définitions inductives
   3. Principe général d'une définition inductive

1. Récursion

Une fonction récursive est une fonction dont la définition contient un (ou plusieurs) appel à elle-même

Exp. La fonction de Fibonnacci

n>0

fb(1) = 1

fb(n) = fb(n-1) + fb(n-2)

Exp. (suite)

Fonction fb(n)

si n = 1 alors retourner 1

sinon retourner fb(n-1) + fb(n-2)

fsi

Notons:

- **Cas terminal**: le cas où on peut calculer le résultat immédiatement.
- **Appels récursifs**: Ce sont des appels à la fonction elle-même dans le corps de la fonction.
- **Appel récursif terminal**: Si un appel récursif n'est pas suivi d'un calcul dans le corps de la fonction cet appel est terminal

Dans l'exemple précédent, le cas terminal est le cas où n=1.

Les appels récursifs fb(n-1) et fb(n-2).

Les appels récursifs fb(n-1) et fb(n-2) sont terminaux.

1.1. Utilisation des fonction récursives

Soit l'exemple des tours de Hanoi.

Principe du jeux:

- n disques de taille croissante numérotés de 1 à n.
- 3 piquets: départ, auxiliaire, arrivée.
- Au départ, les disques sont empilés autour du piquet départ par ordre décroissant (le grand en bas).
- Fin du jeux: les disques sont empilés autour du piquet arrivée par ordre décroissant (le grand en bas).
- Pour jouer: déplacer un disque du sommet d'un piquet vers un autre avec interdiction d'empiler un disque sur un disque plus petit.

1.2. Formulation en système formel

Soit le système formel formalisant le problème du jeux de Hanoi suivant: SFH<A, γ, Ax, R>, avec:

Les symboles: A = {n, a, b, c,[,]}

Les mots: [S, x, y, z]; avec S ∈ {1,2,3} et (x,y,z) ∈ $\{a,b,c\}^3$.

L'axiome: Ax = {[3,a,b,c]}

Les règles: R = {[n, x, y, z] → [n-1, x, z, y][n, x, y, z][n-1, y, x, z]

[0, x, y, z] →

}

Solution

Le cas terminal est pour un nombre de piquets nul. Dans ce cas on ne fait rien.

```
Hanoi (n, dep, aux, arr) {
  Si (n > 0) alors {
    Hanoi (n-1, dep, arr, aux);
    Déplacer (n, dep, arr);
    Hanoi (n-1, aux, dep, arr);
  }
}
```

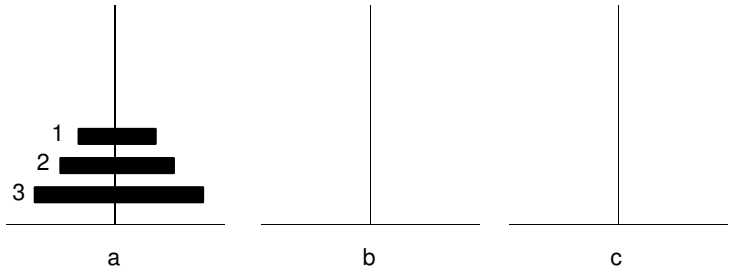
Solution (suite)

La fonction Déplacer(n, dep, arr) permet de déplacer le disque 'n' du piquet 'dep' vers le piquet 'arr'.

```
Déplacer(n, a, b) {
  Si (n est sommet de a) alors {
    Dépiler n de a;
    Empiler n dans b;
  }
}
```


Solution (suite)

Dérouler l'algorithme pour n=3, et les 3 piquets a,b,c.



2. Induction

L'induction structurelle est une généralisation de la preuve par récurrence.

2.1. Raisonnement pas récurrence sur N

Théorème: Soit P(n) un prédicat (propriété) dépendant de l'entier n. Si les deux conditions suivantes sont vérifiées:

- P(0) est vrai;
- P(n) implique P(n+1) pour tout n;

alors pour tout entier n, P(n) est vrai.

Exp.

On considère u_n avec $n \in N$.

$$\begin{cases} u_0 = 2 \\ u_{n+1} = 1 + \dfrac{1}{1+u_n} \end{cases}$$

démontrer par récurrence que pour tout naturel n on a $1 \le u_n \le 2$

Pour u_0 on a $1 \le u_0 \le 2$

Supposant que $1 \le u_n \le 2$ est vrai, et vérifiant que $1 \le u_{n+1} \le 2$

Exp. (suite)

$1 \leq u_n \leq 2$

$2 \leq 1+u_n \leq 3$

$1/2 \geq 1/(1+u_n) \geq 1/3$

$1 + 1/2 \geq 1 + 1/(1+u_n) \geq 1 + 1/3$

$3/2 \geq u_{n+1} \geq 4/3$

$2 \geq 3/2 \geq u_{n+1} \geq 4/3 \geq 1$

$2 \geq u_{n+1} \geq 1$

2.2. Définitions inductives

Pour définir des parties d'un ensemble E on se base sur d'autres parties.

Exp. Les entiers pairs correspondent aussi au plus petit ensemble qui contient 0 et tel que si n est pair, alors n+2 est pair.

2.3. Principe général d'une définition inductive

Dans une définition inductive,

- Certains éléments de l'ensemble E sont donnés explicitement (c'est-à-dire on se donne un ensemble B d'éléments de E, qui constitue l'ensemble de base de la définition inductive) ;
- Les autres éléments de l'ensemble E sont définis en fonction d'éléments appartenant déjà à l'ensemble E selon certaines règles.

Exp. 1

Soit A = { '(', ')' } l'alphabet constitué de la parenthèse ouvrante et de la parenthèse fermante.

L'ensemble $D \subset A^*$ des parenthèses bien formés, appelé langage de Dyck, est défini inductivement par:

$\varepsilon \in D$

$x \in D \Rightarrow (x) \in D$

$(x,y) \in D^2 \Rightarrow xy \in D$

CHAPITRE II
RÉCURSION ET INDUCTION

Exp. 2

Le langage L sur l'alphabet A = {a,b} des mots de la forme $a^n bc^n$, $n \in N$, se définit inductivement par:

$b \in L$;

$w \in L \Rightarrow awc \in L$

CHAPITRE III
SYSTÈME DE RÉÉCRITURE

PLAN

1. Définitions
2. Réécritures des mots et termes

CHAPITRE III
SYSTÈME DE RÉÉCRITURE

1. Définitions

- **Le terme**: Soit F un ensemble de fonctions (symboles fonctionnels) et X un ensemble de variables.

  T(F) un ensemble de **termes clos** (termes sans variables)

  T(F,X) un ensemble de **termes ouverts** (termes avec variables)

  Un symbole fonctionnel $f \in F$ est **d'arité** n veut dire qu'il accepte n opérandes, et on écrit $f(t_1, \dots, t_n)$ avec les t_i sont des termes.

CHAPITRE III
SYSTÈME DE RÉÉCRITURE

1. Définitions

Exp:

Soit X = {x} l'ensemble de variables ouvertes, et soit F={+,*,a} l'ensemble de symboles clos, avec arité(+)=2, arité(*)=2, arité(a)=0

a+a+a est un terme clos;

a*x+a*a est un terme ouvert.

1. Définitions (suite)

- **Une position** p d'un terme t représente le symbole de la position p dans le terme t et note t(p).
- On note Pos(t) l'ensemble des positions du terme t, ou bien la taille de t.
- t(ε) appelé symbole de tête de t.

Exp.1

ε.1 (= 1) Position du premier fils

ε.1.2 (= 12) Position du second fils du premier fils

1. Définitions (suite)

Exp.2

Soit F = {f:2, a:0, b:0} et X = {x, y}

Quelles sont les positions de x, a, f(a,y) dans le terme:

t = f(x, f(a, y)) $\in$ T(F,X)

t(x) = 1

t(a) = 21

t(f(a,y)) = 2

1. Définitions (suite)

On note:

t(w) représente le symbole fonctionnel de t à la position w;

$t|_w$ représente le sous-terme de t à la position w;

$t[t']_w$ remplace le sous-terme de t à la position w par t';

Var(t) représente l'ensemble de variable du terme t.

1. Définitions (suite)

Exp.

Soit t=f(a, f(b, g(x))), t' = f(b, b) et w = 22.

Calculer: t(w), $t|_w$, $t[t']_w$

t(w) = g

$t|_w$ = g(x)

$t[t']_w$ = f(a, f(b, f(b, b)))

1. Définitions (suite)

- Une **substitution** est une application notée comme suit $\sigma = \{x_1 \rightarrow t_1, x_2 \rightarrow t_2 , \dots , x_n \rightarrow t_n\}$.

  L'application de cette substitution sur un terme t se fait comme suit:

  Si $t = x_i$, alors $\sigma(x_i) = t_i$;

  $\sigma(x_k) = x_k$; avec $x_k \notin \{x_1, \dots, x_n\}$;

  $\sigma(f(u_1, \dots , u_m)) = f(\sigma(u_1), \dots , \sigma(u_m))$;

1. Définitions (suite)

Exp.

Soit l'ensemble de symboles fonctionnels F = {+:2, *:2, a:0, b:0} et l'ensemble de variables X = {x, y}.

Une substitution est définie par $\sigma = \{x \rightarrow a + a, y \rightarrow b\}$.

Appliquer la substitution σ au terme t = a*x+a*y

$\sigma(t)$ = a*a+a+a*b

2. Réécritures des mots et termes

- Une **règle de réécriture** est une paire de termes orientée, noté $l \rightarrow r$

tel que: l est le membre gauche de la règle

r est le membre droit de la règle.

Exp.

Pour des mots: ab $\rightarrow$ ba; ax $\rightarrow$ x avec x variable

Pour des termes: g(x) $\rightarrow$ f(a, f(b,x)) avec x variable

2. Réécritures des mots et termes (suite)

- Les **conditions de construction** des règles de réécriture:
  - Le membre gauche d'une règle de réécriture n'est pas une variable.
  - L'ensemble des variables du membre droit est inclus dans l'ensemble des variables du membre gauche $Var(r) \subseteq Var(l)$

- Un **système de réécriture** sur les termes est un ensemble de règles de réécriture.

2. Réécritures des mots et termes (suite)

Exp.

Soit la règle de réécriture :

f(f(a,x),h(x)) $\rightarrow$ g(b, x)

Appliquer cette règle à:

h(f(f(a,b),h(b))) : h(g(b, b))

f(f(a,c),h(c)) : g(b, c)

2. Réécritures des mots et termes (suite)

- Propriétés des règles de réécriture:
  - L'ensemble des variables d'une règle noté Var($l \rightarrow r$) est définie par Var(l) $\cup$ Var(r)
  - Comme Var(r) $\subseteq$ Var(l) donc Var($l \rightarrow r$) = Var(l)
  - Une règle de réécriture est **régulière** si Var(r) = Var(l). Un système de réécriture est **régulier** si toutes ses règles sont régulières.

2. Réécritures des mots et termes (suite)

- Une **relation de réécriture** $\rightarrow_R$ associée à un système de réécriture R est définie par **t** $\rightarrow_R$ **t'** s'il existe une position p dans t, une règle (l $\rightarrow$ r) dans R et une substitution σ tel que $\mathbf{t|_p = \sigma(l)}$ et $\mathbf{t' = t[\sigma(r)]_p}$

- Dans un système de réécriture R, on dit qu'un terme t est en **R-forme normale** s'il n'existe pas de terme s tel que t $\rightarrow_R$ s.

PLAN

1. Définitions
2. Implantation d'un TDA en C++
3. Implantation d'un TDA en Java

CHAPITRE IV
TYPE DE DONNÉES ABSTRAIT

1. Définitions

- Un type de données abstrait est un ensemble de données et d'opérations.
- Il est caractérisé par :
  - Son identité;
  - Type de données manipulées;
  - Les opérations sur les données;
  - Les propriétés de ces opérations.

CHAPITRE IV
TYPE DE DONNÉES ABSTRAIT

2. Implantation d'un TDA en C++

Nous nous intéressons dans cette section au type de données Pile.

Le langage C++ les mécanismes nécessaires pour la réalisation d'un TDA Pile (**classes** et **templates**)

Pour que notre pile soit abstraite nous utilisons les templates pour définir les éléments de la pile comme suit:

template <typename Element>

CHAPITRE IV
TYPE DE DONNÉES ABSTRAIT

2. Implantation d'un TDA en C++ (suite)

Nous allons réaliser la pile en utilisant un tableau de 10 éléments. La classe PILE doit contenir les attributs et les méthodes qui manipulent ces attributs:

CHAPITRE IV
TYPE DE DONNÉES ABSTRAIT

2. Implantation d'un TDA en C++ (suite)

```
const short dimPile = 10;
template <typename Element>
class PILE {
 private:
   short sommet;
   Element tableau [dimPile];
 public:
   PILE();
   void Empiler(Element elem);
   void Depiler(Element &elem);
   bool EstVide();
};
```

3. Implantation d'un TDA en Java

Le langage Java offre la possibilité d'utiliser les interfaces qui représentent la notion d'abstraction ou les classes abstraites.

3.1. Implantation d'un TDA avec classes abstraites

Le mot clé "abstract" permet de définir une classe abstraite comme suit:

3.1. Implantation d'un TDA avec classes abstraites (suite)

```
public abstract class Pile {
  abstract void Empiler(Object obj);
  abstract Object Despiler();
  abstract boolean EstVide();
  abstract Object sommet();
}
```


3.1. Implantation d'un TDA avec classes abstraites (suite)

```
public class PileParTableau extends Pile {
     private int tailleMax;
     private int sommet;
     private Object[] tableau;
     public PileParTableau(int max) {
          tailleMax = max;
          sommet = -1;
          tableau = new Object[tailleMax];
     }
```


3.1. Implantation d'un TDA avec classes abstraites (suite)

```
     Object Despiler() {
          if (!EstVide()) {
               Object obj=tableau[sommet--];
               System.out.println("Dépiler"+obj);
               return obj;
          }
          return null;
     }
     boolean EstVide() {return (sommet == -1);}
     Object sommet() {return tableau[sommet];}
}
```

CHAPITRE IV
TYPE DE DONNÉES ABSTRAIT

3.1. Implantation d'un TDA avec classes abstraites (suite)

```
public class Test {
  public static void main(String[] args) {
    PileParTableau p = new PileParTableau(10);
    for (int i = 0; i < 10; i++)
      p.Empiler(new Integer(i));
    while (!p.EstVide())
      p.Despiler();
  }
}
```


CHAPITRE IV
TYPE DE DONNÉES ABSTRAIT

3.2. Implantation d'un TDA avec Interface

```
public interface Pile {
  void Empiler(Object obj);
  Object Despiler();
  boolean EstVide();
  Object sommet();
}
```


CHAPITRE IV
TYPE DE DONNÉES ABSTRAIT

3.2. Implantation d'un TDA avec Interface

```
public class PileParTableau implements Pile
{
     private int tailleMax;
     private int sommet;
     private Object[] tableau;
     ...
}
```

La classe Test reste la même.

CHAPITRE V
GRAPHES ET ARBRES

PLAN

1. Graphes
   1. Représentation informatique des graphes
   2. Définitions
   3. Connexité
2. Arbres
   1. Propriétés
   2. Forêt
   3. Arborescence
   4. Propriétés des arbres binaires
   5. Représentation informatique des arbres

1. Graphes

Un graphe **non orienté** noté **G=<S,A>** tel que:

S: ensemble de sommets

A: ensemble d'arêtes (s,t) ⇔ (t,s)

Un tel graphe modélise une relation symétrique.

1. Graphes (suite)

Un graphe **orienté** noté **G=<S,A>** tel que:

S: ensemble de sommets

A: ensemble d'arcs (s,t)

Une arête ou un arc (s,s) est appelé **boucle**.

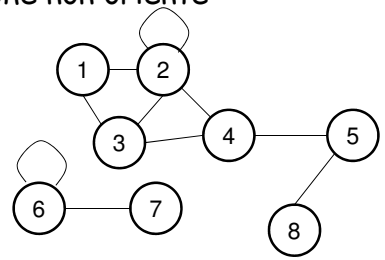
On peut associer une **valuation** à un graphe: càd une valeur associée à chaque arête ou arc.

1. Graphes (suite)

Exemple

Soit G=<S,A> tel que S = {1,2,3,4,5,6,7,8} et A = {(1,2),(2,2),(2,3),(3,1),(3,4),(4,2),(4,5),(5,8),(6,6),(6,7)}

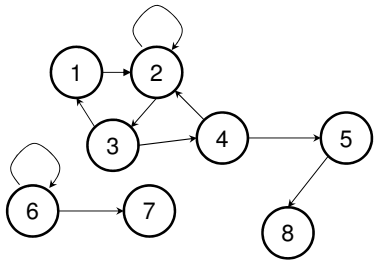
Graphe non orienté



1. Graphes (suite)

Exemple (suite)

Graphe orienté



1.1. Représentation informatique des graphes

Un graphe est représenté par une matrice M_G tel que :

$M_{ij} = 1$ si $(i,j) \in A$

$M_{ij} = 0$ sinon.

Exp:

0	1	0	0	0	0	0	0
0	1	1	0	0	0	0	0
1	0	0	1	0	0	0	0
0	1	0	0	1	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	1	1	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

1.1. Représentation informatique des graphes (suite)

Pour un graphe valué on utilise des valeurs à la place des 1.

- **Matrice d'incidence** N_G

tel que:

$N_{ij} = 1$ si l'arête (arc) j admet le sommet i comme origine.

$N_{ij} = -1$ si l'arête (arc) j admet le sommet i comme arrivée.

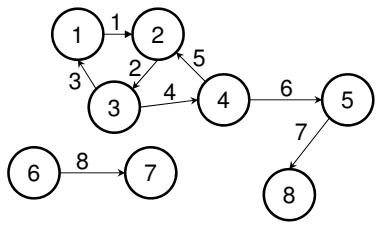
$N_{ij} = 0$ sinon.

1.1. Représentation informatique des graphes (suite)

Inconvénient: La matrice d'incidence n'est pas utilisé dans le cas d'un graphe avec boucles.

1.1. Représentation informatique des graphes (suite)

Exp:



1	0	-1	0	0	0	0	0
-1	1	0	0	-1	0	0	0
0	-1	1	1	0	0	0	0
0	0	0	-1	1	1	0	0
0	0	0	0	0	-1	1	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	-1
0	0	0	0	0	0	-1	0

1.1. Représentation informatique des graphes (suite)

• **Liste d'adjacence**: liste des sommets et pour chacun liste des successeurs.

((1, (2)), (2, (2, 3)), (3, (1, 4)), (4, (2, 5)), (5, (8)), (6, (6, 7)), (7, ()), (8, ()))

1.2. Définitions

• **Ordre** du graphe: Nombre de sommets |S|

• **Taille** du graphe: Nombre d'arêtes ou d'arcs |A|

• Un **sous-graphe** d'un graphe G: Quand on retire des arêtes ou arcs à G.

• Un **graphe partiel** du graphe G: Quand on retire des sommets avec leurs arêtes ou arcs incidents.

• Dans un graphe non-orienté une **chaine** est une suite non vide d'arêtes adjacentes.

• Dans un graphe non-orienté un **cycle** est une chaine tel que les arêtes ne se répètent pas, et les sommet de début et de fin sont les mêmes.

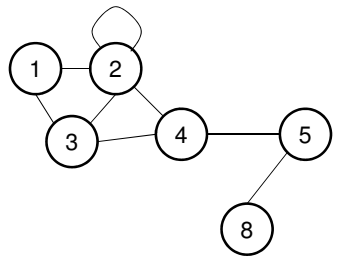
1.2. Définitions (suite)

• Dans un graphe orienté un **chemin** est une suite non vide d'arcs consécutifs.

• Dans un graphe orienté un **circuit** est un chemin fermé ne passant pas 2 fois par le même arc.

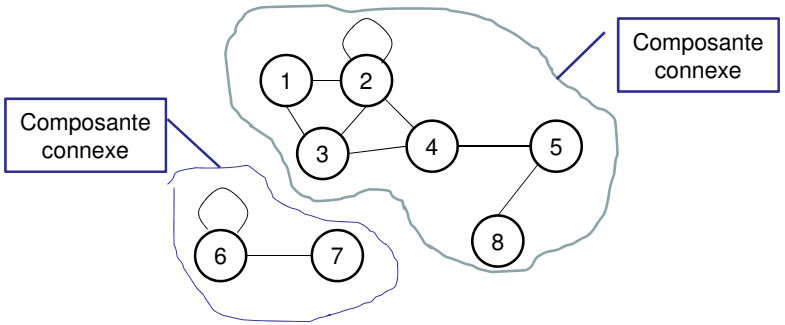
1.3. Connexité

• Un graphe non orienté est **connexe** s'il existe une chaine entre toute paire de sommets distincts.



1.3. Connexité (suite)

• Une **composante connexe** d'un graphe G est une partie du graphe G qui forme un petit graphe connexe.



1.3. Connexité (suite)

• **Théorème 1:** Tout graphe connexe d'ordre n>0 a au moins n-1 arêtes.

• **Théorème 2:** Tout graphe sans cycle d'ordre n>0 a au plus n-1 arêtes.

2. Arbres

Un arbre est un graphe connexe sans cycle.

2.1. Propriétés

➢Soit un graphe G=<S,A> d'ordre n ≥ 2, si G est un arbre alors:

• Tout couple de sommets est relié par une chaine unique

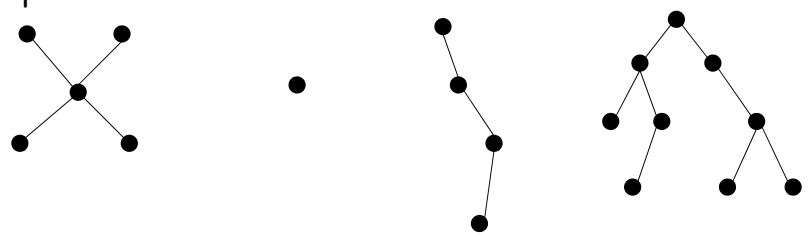
• On ajoutant une arête à G on crée un cycle

• On supprimant une arêtes G ne devient plus connexe

• G admet n-1 arêtes

2.1. Propriétés (suite)

Exp:



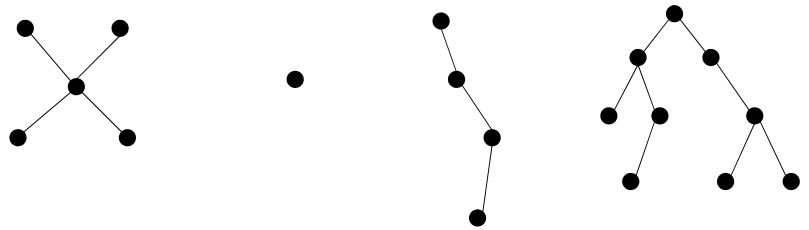
2.1. Propriétés (suite)

- Un arbre d'ordre n avec n>1 a au moins deux feuilles
- Un arbre d'ordre n avec n>0 a n-1 arêtes

2.2. Forêt

Une forêt est un graphe dont chaque composante connexe est un arbre.

Ou bien, une forêt est un graphe sans cycle.



2.2. Forêt (suite)

Un sommet d'un arbre est appelé **nœud**.

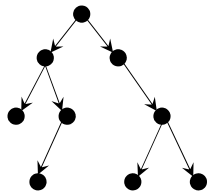
Un nœud adjacent à une seule arête est appelé **feuille**.

On peut distinguer un nœud particulier pour l'appeler **racine**.

Une **branche** est une chaine reliant une feuille à la racine.

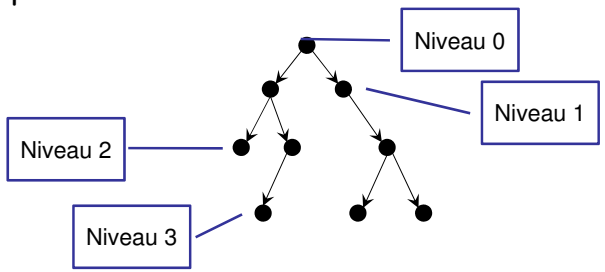
2.3. Arborescence

Une **arborescence k-aire** est un arbre enraciné **orienté** dont chaque sommet à **au plus** k successeurs appelés **fils**.



2.3. Arborescence (suite)

Un **niveau** est un ensemble de nœuds qui ont une distance égale par rapport à la racine. Les niveaux sont numérotés à partir de 0.

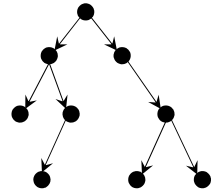


2.3. Arborescence (suite)

La **hauteur (profondeur)** d'un arbre binaire est le nombre d'arcs sur un chemin de longueur maximale.

Exp:

L'arbre suivant à une hauteur ou profondeur = 3



2.3. Arborescence (suite)

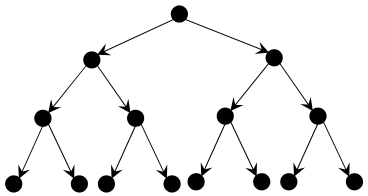
Un arbre k-aire est **complet** si tous les nœuds ont 0 ou k fils

Un arbre complet est **saturé** si tous les chemins ont un nombre maximal de nœuds.

Exp:

L'arbre suivant est 2-aire avec une profondeur = 3

Il est complet et saturé



2.4. Propriétés des arbres binaires

Soit un arbre binaire saturé à **n** nœuds et de hauteur **h**:

- Le niveau **i**, $i \geq 0$, contient $\mathbf{2^i}$ nœuds;
- La hauteur $\mathbf{h = \log_2(n+1)-1}$;
- Le nombre de feuilles est $\mathbf{2^h = (n+1)/2}$;
- Le nombre de nœuds est $\mathbf{2^{h+1}-1}$

2.5. Représentation informatique des arbres

2.5.1. Représentation séquentielle d'un arbre binaire

On utilise un tableau **T** de **n**(=2^{h+1}-1) cases.

T[n-1] : Contient la racine.

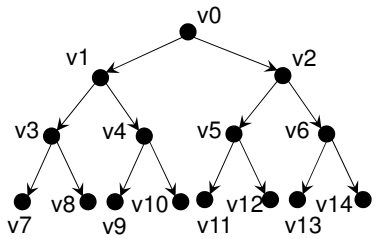
T[2i] : Contient le fils gauche du nœud i

T[2i+1] : Contient le fils droit du nœud i

2.5.1. Représentation séquentielle d'un arbre binaire

Exp: h = 3; n = 2^{3+1}-1 = 15

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
v1	v2	v3	v4	v5	v6	v7	v8	v9	v10	v11	v12	v13	v14	v0



2.5.2. Représentation récursive d'arbre binaire

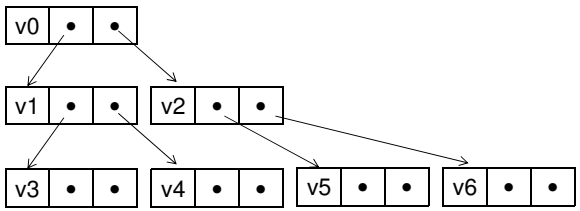
On utilise un pointeur vers l'arbre

Un nœud est représenté par une valeur et deux pointeurs:

- un pointeur vers le fils gauche
- un autre vers le fils droit.

2.5.2. Représentation récursive d'arbre binaire

Exp: h = 2; n = 2^{2+1}-1 = 7



2.5.3. Représentation récursive d'un arbre k-aire

On utilise un pointeur vers l'arbre

Un nœud est représenté par une valeur et deux pointeurs:

- un pointeur vers le premier fils
- un autre vers la liste des frères.

CHAPITRE V
GRAPHES ET ARBRES

2.5.3. Représentation récursive d'un arbre k-aire

Exp: k = 4