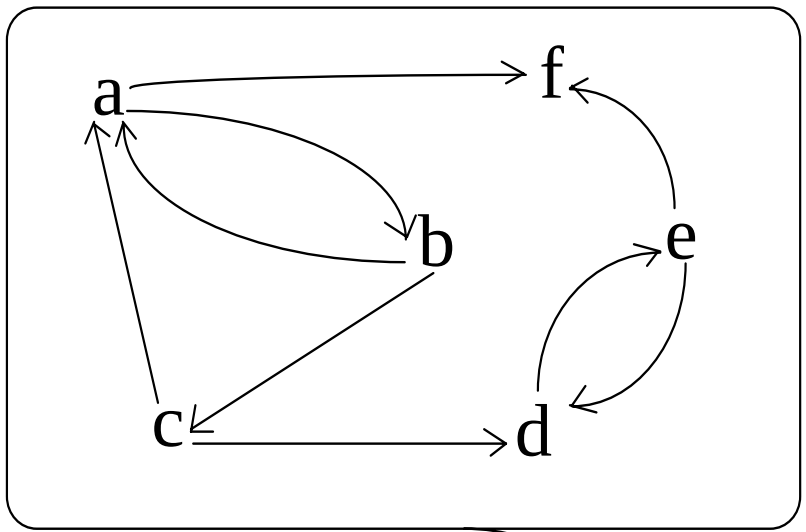
5
THEORIE DES GRAPHES

PLAN

- **Généralités et définitions**
- **Représentation d'un graphe**
- **Arbres et arborescences**
- **Exploration d'un graphe**
- **Connexité et forte connexité**

5.1 GENERALITES ET DEFINITIONS

5-1-1 GRAPHES ORIENTES



EXEMPLE: **G1 = (X1 , U1)**

X1 = {<u>sommets</u>} = {a,b,c,d,e,f}

U1 = {<u>arcs</u>} = {(a,b),(b,a),(b,c),(c,a),(c,d),(a,f),
(e,f),(d,e),(e,d)}

Γ : $X \rightarrow \mathcal{P}(X)$

$x \rightarrow \Gamma(x) =$ {successeurs de x}

EXEMPLE(suite) $\Gamma 1(b)=\{a,c\}$ $\Gamma 1(f)=\{\emptyset\}$ $\Gamma 1(d)=\{e\}$

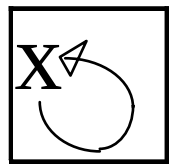
Chemin : suite d'arcs telle que l'extrémité terminale d'un arc coïncide avec l'extrémité initiale de l'arc suivant

---

EXEMPLE (suite) ((a,b),(b,c),(c,d)) ou (a,b,c,d)

---

boucle : arc du type (x,x)



circuit : chemin dont le premier sommet coïncide avec le dernier

---

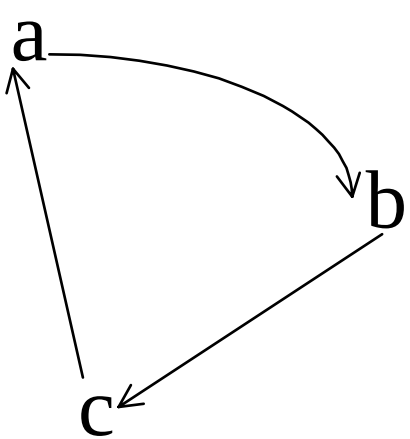
EXEMPLE (suite) **(a,b,c,a) circuit de G1**

---

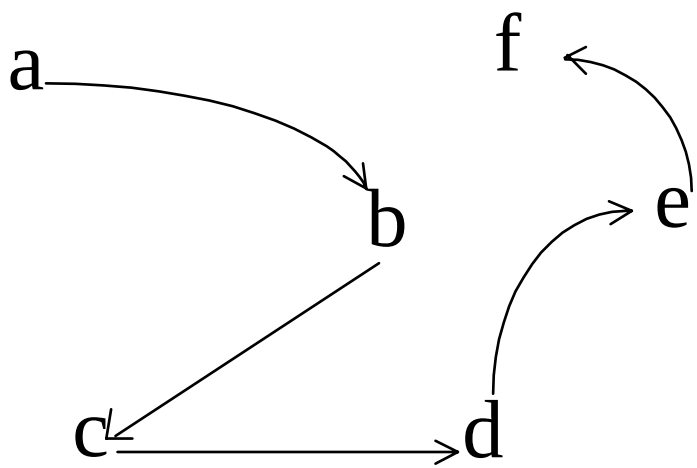
chemin hamiltonien : chemin qui passe une fois et une seule par chaque sommet

---

EXEMPLE (suite) **(a,b,c,d,e,f)**



un circuit



un chemin hamiltonien

5-1-2 UTILISATION DES GRAPHES

- **Modélisation, représentation de problèmes**

Exemple: plan de ville, arbre généalogique, états d'un système..

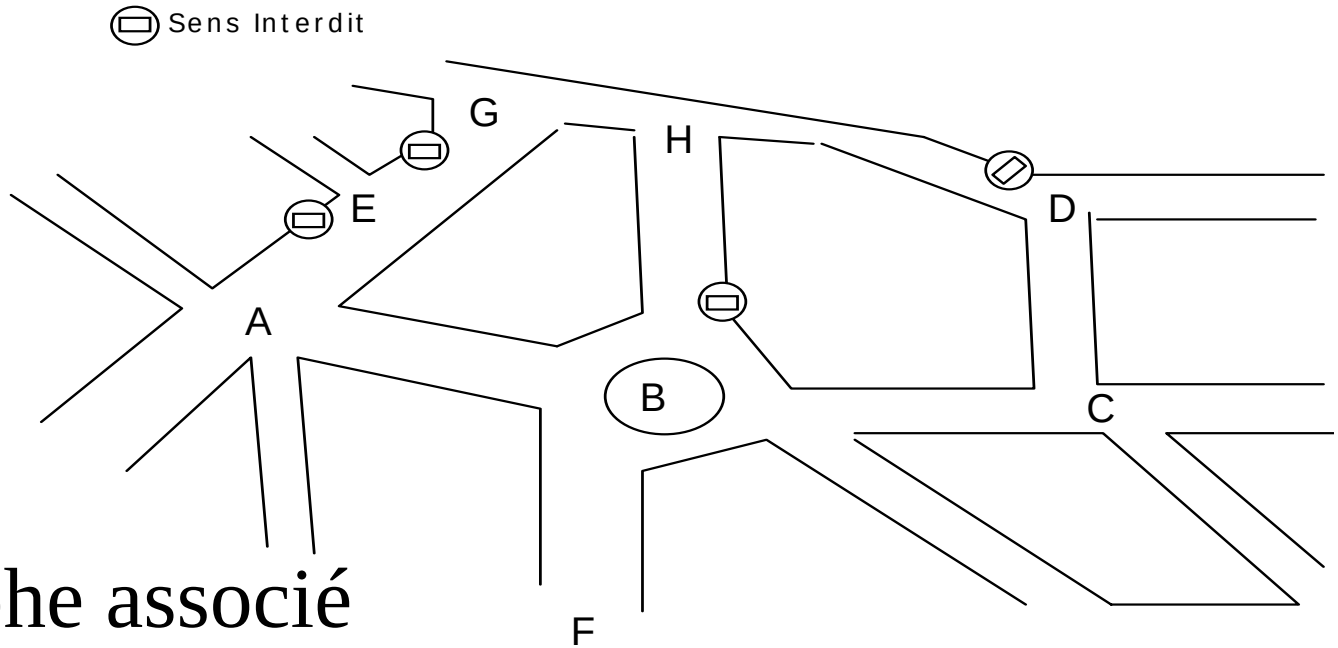
- **Résolution de problèmes**

Exemple: plus court chemin, ordonnancement, flots, ...

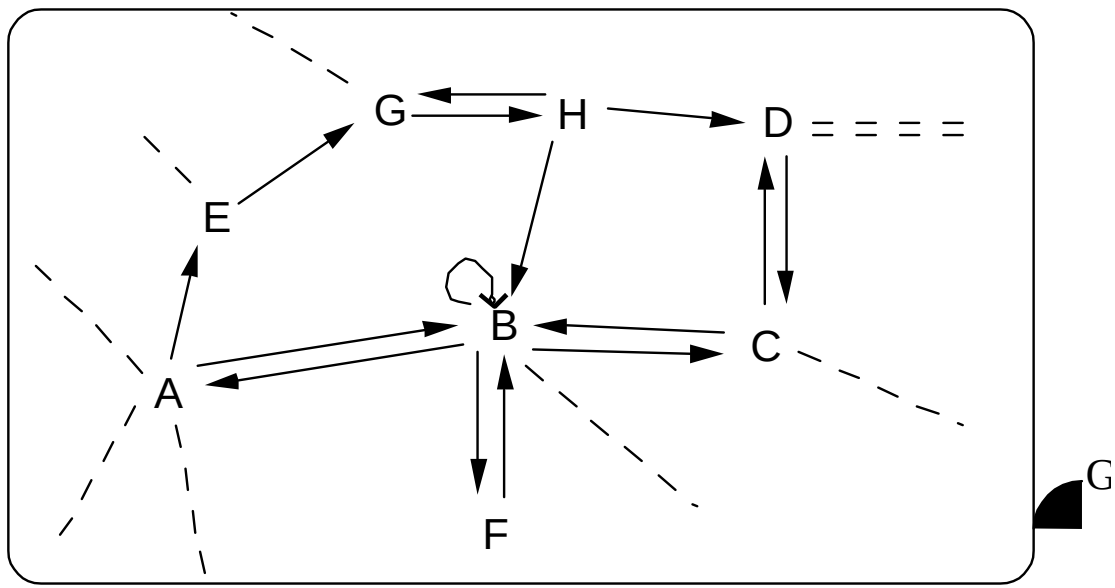
- **Outils**

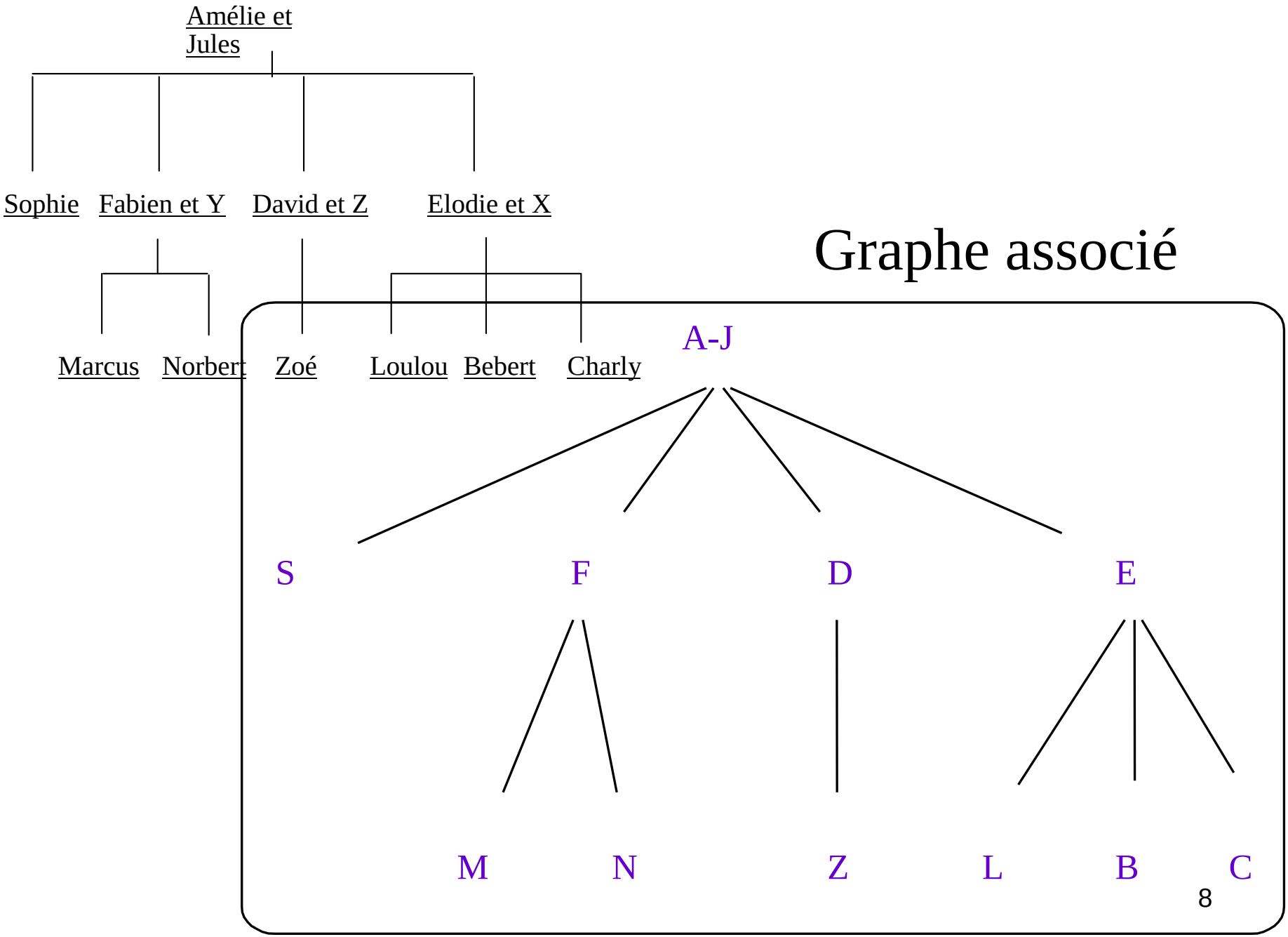
Exemple: structures de données,.

- **...**



Graphe associé

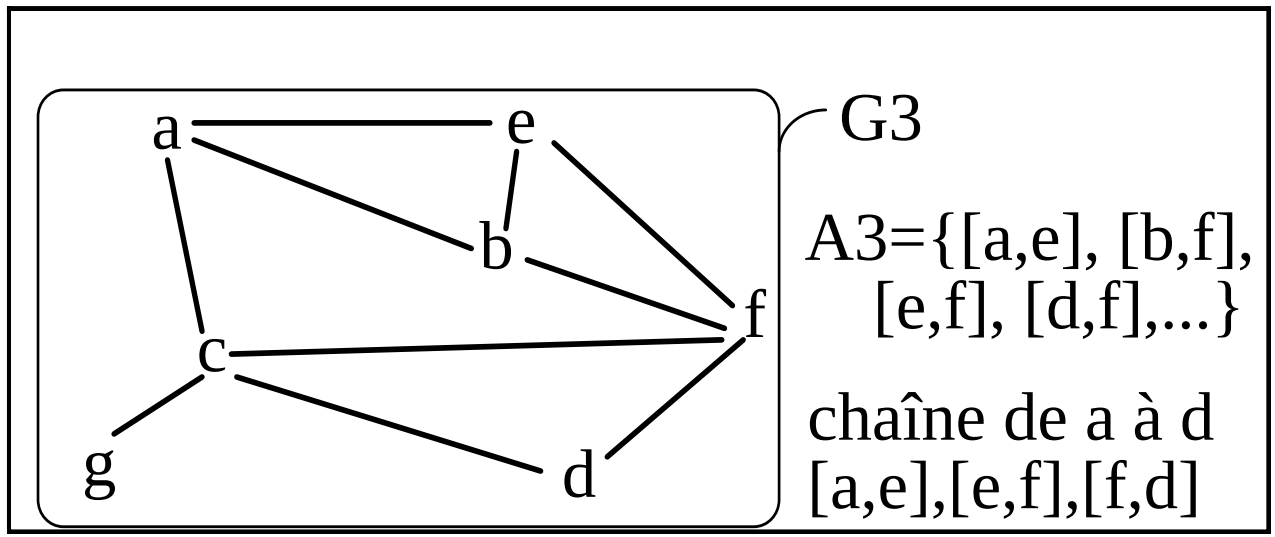



Amélie et Jules
Sophie
Fabien et Y
David et Z
Elodie et X
Marcus
Norbert
Zoé
Loulou
Bebert
Charly
Graphe associé
A-J
S
F
D
E
M
N
Z
L
B
C

5-1-3 GRAPHES NON ORIENTES

G= (X,A) A est un ensemble d'arêtes

arête : arc "sans orientation"



Chaîne : suite d'arêtes telle que toute arête a une extrémité commune avec l'arête précédente (sauf la première) et l'autre avec l'arête suivante (sauf la dernière)

cycle : chaîne dont les deux extrémités coïncident

EXEMPLE (suite) **cycle de G3: [aefba]**

---

chaîne et cycle sont définis aussi dans un graphe orienté (on ne tient plus compte de l'orientation)

EXEMPLE (suite) **cycle de G1: (bacb)**

---

connexité : un graphe est connexe si toute paire de sommet est reliée par une chaîne

---

EXEMPLE (suite) **G1 et G3 connexes, G3' non connexe**

---

degré x $\in$ X

d(x) = nombre de voisins de x

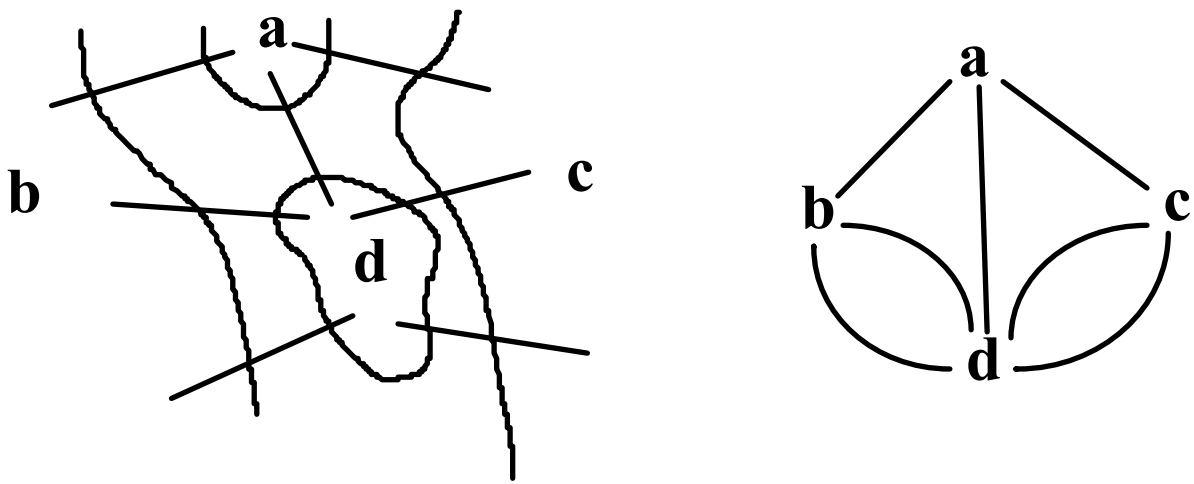
EXEMPLE (suite) **dans G3 d(c) = 4 d(b) = 3**

---

chaîne eulérienne : chaîne qui passe une fois et une seule par chaque arête

Théorème d'Euler

Un multigraphe connexe admet une chaîne eulérienne si et seulement si le nombre de sommets de degré impair est 0 ou 2

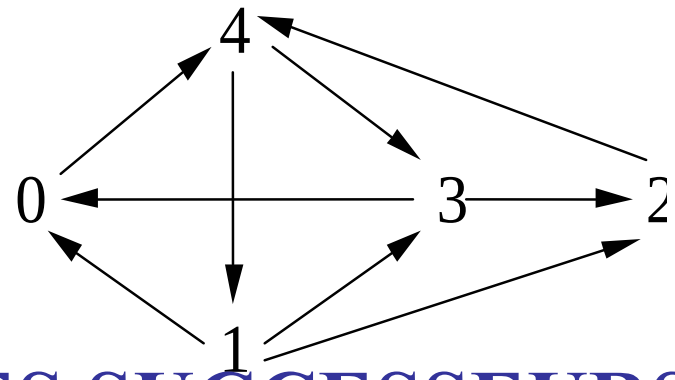


1766 La Pregel à Koenigsberg

5.2 REPRESENTATION D 'UN GRAPHE

G2 = (X,A)

|X2| = n = 5
|A2| = m = 9

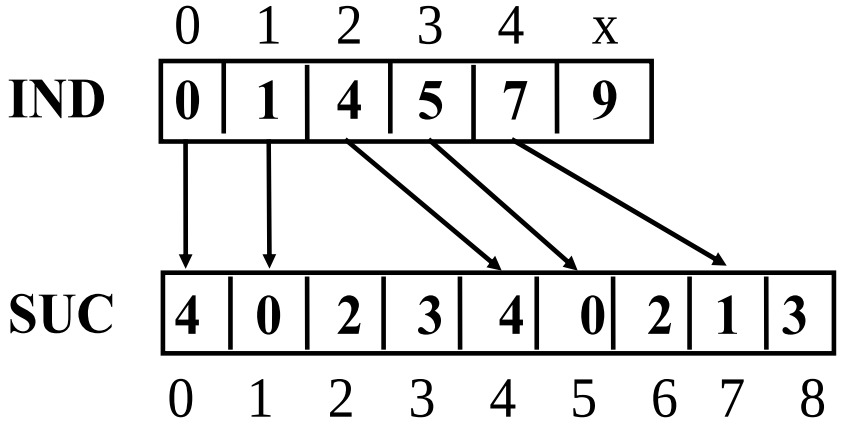


5-2-1 TABLEAU DES SUCCESSEURS

X	Nb suc	liste des successeurs		
0	1	4		
1	3	2	3	0
2	1	4		
3	2	2	0	
4	2	1	3	

- accès facile aux successeurs
- assez souple
- assez peu de place mémoire

5-2-2 DEUX TABLEAUX



IND[i] = indice du premier successeur de i dans SUC

SUC : tableau des successeurs

Γ(i) = {SUC[IND[i]], SUC[IND[i]+1],... , SUC[IND[i+1]-1] }

- accès facile aux successeurs

- très peu souple

- très peu de place mémoire

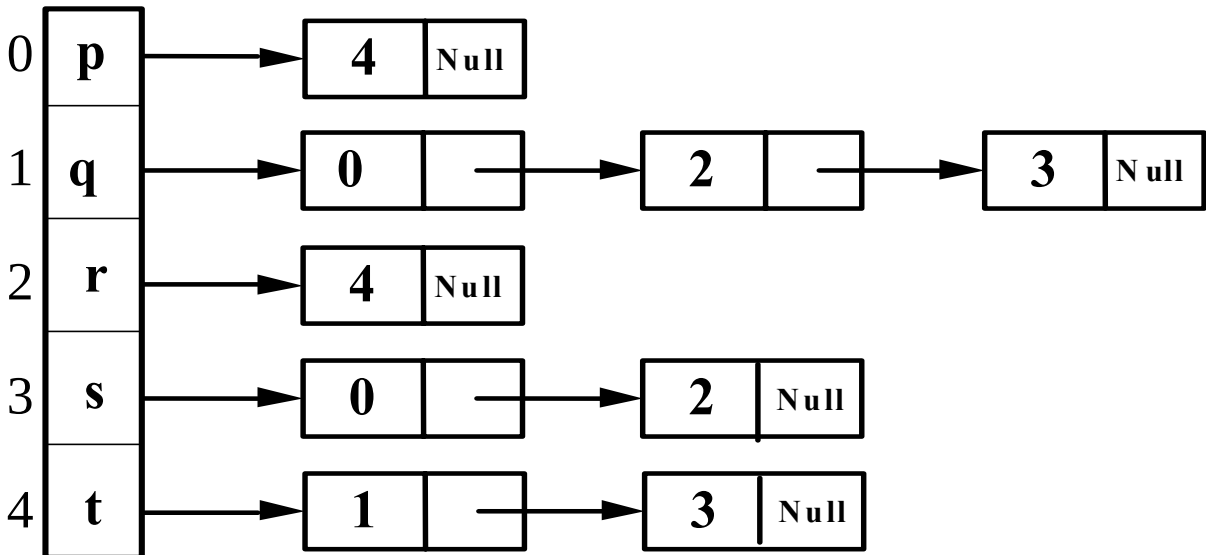
Exemple: i=3,
IND[3]=5 IND[4]=7
Γ(3) =
{SUC[5],SUC[6]}={0,2}

5-2-3 UNE MATRICE

	0	1	2	3	4
0	0	0	0	0	1
1	1	0	1	1	0
2	0	0	0	0	1
3	1	0	1	0	0
4	0	1	0	1	0

- balayages
- place mémoire importante
- souple (évolution du graphe)
- calculs matriciels possibles

5-2-4 TABLEAU ADRESSANT DES LISTES CHAÎNEES DE SUCCESSEURS



- **place mémoire assez faible**
- **souple (évolution du graphe)**
- **peu facile à manipuler**

  (ex: nombre de successeurs ?)

5-2-5 La Classe CGraphe

classe CGraphe

{

Nous supposerons qu'un graphe $G=(X,U)=(X,\Gamma)$ est défini par l'ensemble X de ses n sommets de la classe CSommet et soit par l'ensemble U de ses m arcs de la classe CArcs ou arêtes de la classe CArêtes soit par l'application Γ de la classe CSuc qui à chaque sommet x de X associe $\Gamma(x)$={successeurs ou voisins de x}

}

Dans la suite nous donnons une présentation algorithmique des problèmes de graphes

5.3 EXPLORATION D 'UN GRAPHE

G=(X,A) donné avec |X|=n |A|=m

5.3.1 Descendants d'un sommet

Définition

$D(x_0)$= {sommets $x \in X$ tels qu'il existe un chemin de x_0 à x}

Détermination de $D(x_0)$

Principe

marquer x_0

tant que **$\exists (x,y) \in A$ t.q. x marqué et y non marqué**

faire

marquer y;

fait; D(x)={sommets marqués}

void **Aveugle** **(CSommet x_0;**

CTab1 marque *//tableau associant un booléen à*
//chaque sommet **initialisé à "faux"**;

CTab2 père*//tableau associant un sommet père à*
//chaque sommet **initialisé à "∅"):**

booléen modif = vrai;

…

début

marque[x_0]=vrai;

tant que **modif** faire *//jusqu'à ne plus pouvoir marquer de sommets*

modif = faux ;

pour tout **u ∈ A** faire

si **marque[ext_init(u)]** et **non marque[ext_term(u)]** alors

//ext_init(u) (resp. ext_term) =extrémité initiale (resp. terminale) de l'arc u
//l'arc u a seulement son extrémité initiale marquée

modif = vrai ;

marque[ext_term(u)]=vrai;

père[ext_term(u)]= ext_init(u);

finsi;

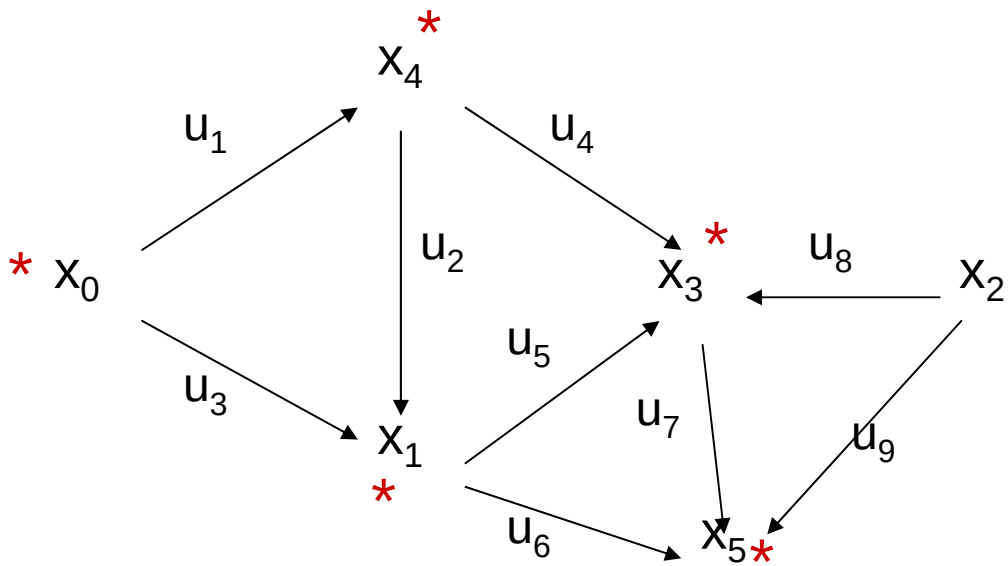
fait;

fait;

fin ;

père: tableau permettant la reconstitution des chemins de marquage de x_0 vers ses descendants.

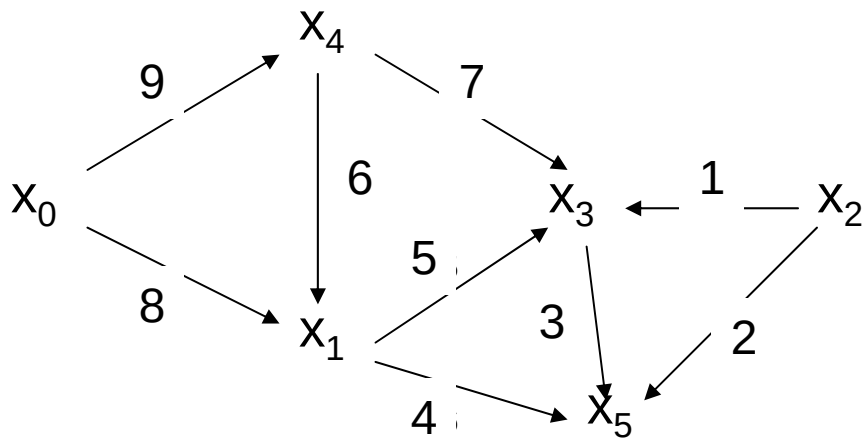
EXEMPLE



	x_1	x_2	x_3	x_4	x_5
père	x_4	$\varnothing$	x_4	x_0	x_1

Ordre de marquage des sommets: 0, 4, 1, 3, 5 au premier passage

EXEMPLE



Ordre de marquage des sommets: 0, 1, 4 puis 3, 5 en plusieurs passages

La complexité dépend de la numérotation des arcs

Pire des cas

a —5→ **b** —4→ **c** —3→ **d** —2→ **e** —1→ **f**

Complexité *(au pire)*: O(m.n)

La procédure suivante décrit un parcours en largeur depuis x_0.

void **Largeur** **(CSommet x0**;
CTab2 père *//tableau de n sommets initialisé à "∅"***;**
CTab3 L *//tableau de n entiers initialisé à "∞"***)**
//L[x]longueur (nombre d'arcs) d'un chemin le plus court de x_0 à x.

File F(n) *//file de sommets*
CTab1 marque *//tableau de n booléens initialisé à "faux";*

CSommet x, y

début*//initialisation*

marque[x_0]=vrai;

L[x_0]=0;

F=filevide(n);

F.enfiler(x_0);

tant que **non F.estvide** faire

x = F.entête ;

F.defiler;

pour tout **y ∈ Γ(x)** faire

si **non marque[y]** alors

marque[y]=vrai;

L[y]= L[x]+1;

père[y]= x ;

F.enfiler(y);

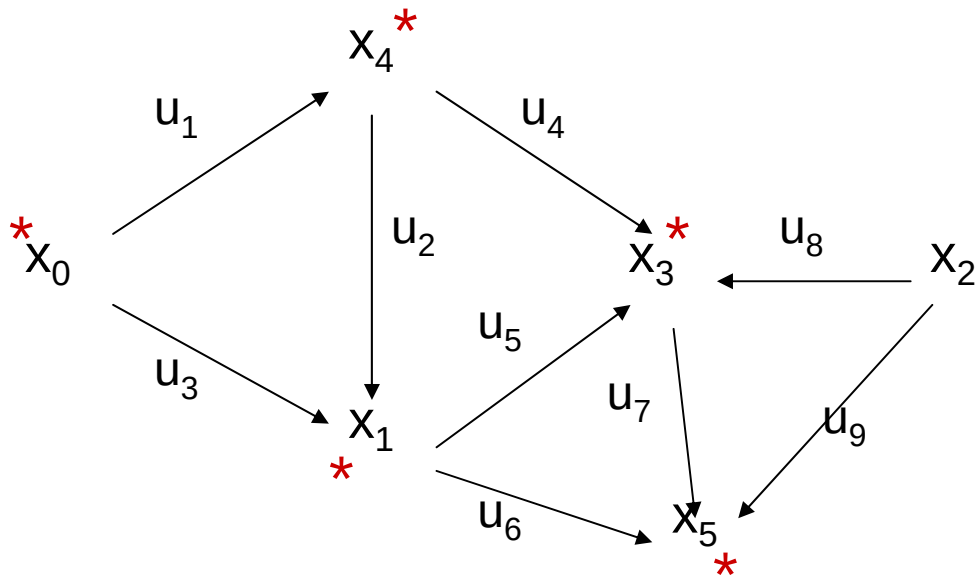
finsi;

fait;

fait;

fin ;

EXEMPLE



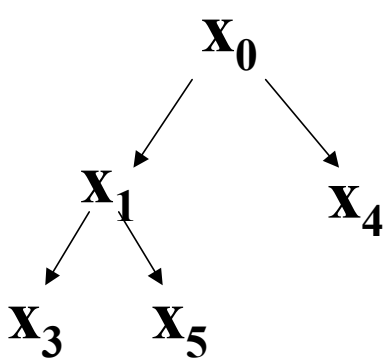
	x_1	x_2	x_3	x_4	x_5
père	x_0		x_1	x_0	x_1
L(x)	**1**		**2**	**1**	**2**

complexité: O(m) (hypothèse: m≥n)

(pour tout sommet faire / pour tout successeur faire)

EXEMPLE

arborescence "en largeur" parcourue:



```
void Imprimer_chemin (CSommet s,v)
//procédure appelée après Largeur(s,père,L)
//Imprime le chemin le plus court de s à v
début
   si   v == s   alors imprimer s ;
   sinon
        si père[v]==∅  alors
                imprimer "pas de chemin de" s "à" v
        sinon
                Imprimer_chemin(s,père[v]);
                imprimer v;
        finsi;
   finsi;
fin;
```

Premier appel

G.Imprimer_chemin(x_0 ,x_n) avec $x_0 \neq x_n$

Les descendants de x_0 ont été déterminés par un parcours "à l'aveugle" puis en largeur à partir de x_0;

la procédure suivante décrit un parcours en profondeur depuis x_0.

La procédure Profondeur utilise la procédure récursive suivante:

void **Visiter** **(CSommet x; CTab1marque** *//tableau de booléen*
CTab2 père *//tableau sommet de sommet***)**
procédure récursive (utilisation d'une pile);
début

```
    pour tout   y ∈ Γ(x)   faire
            si non marque[y] alors
                    marque[y]=vrai;
                    père[y]= x ;
                    Visiter (y,marque,père);
            finsi;
    fait;
```

fin

complexité: O(m)

procedure **Profondeur** **(CSommet x_0; CTab1 marque; CTab2 père)**;

début

pour tout **$x \in X$** faire

marque[x]=faux;

père[x]= ∅ ;

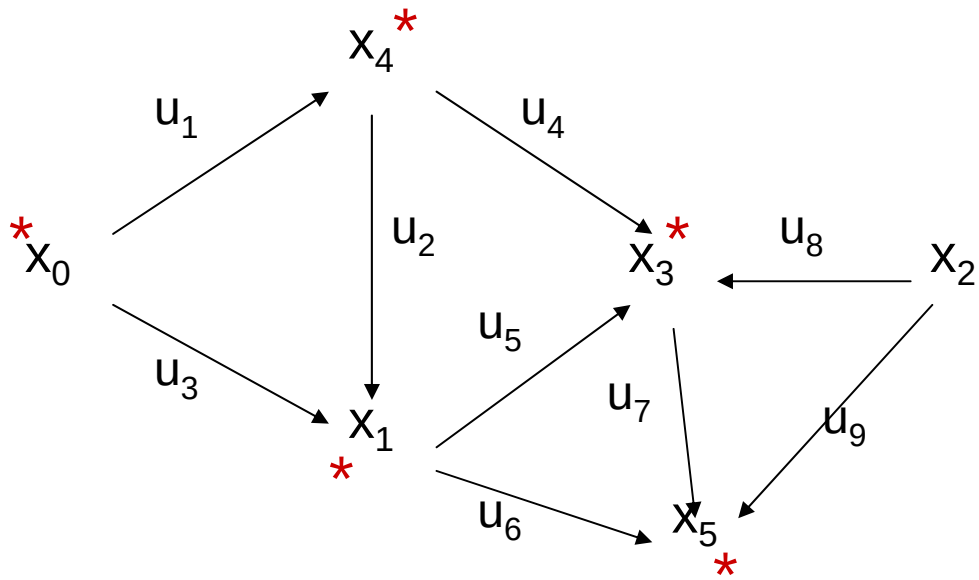
fait;

marque[x_0]=vrai;

Visiter(x_0,marque,père);

fin ; **complexité: O(m)**

EXEMPLE

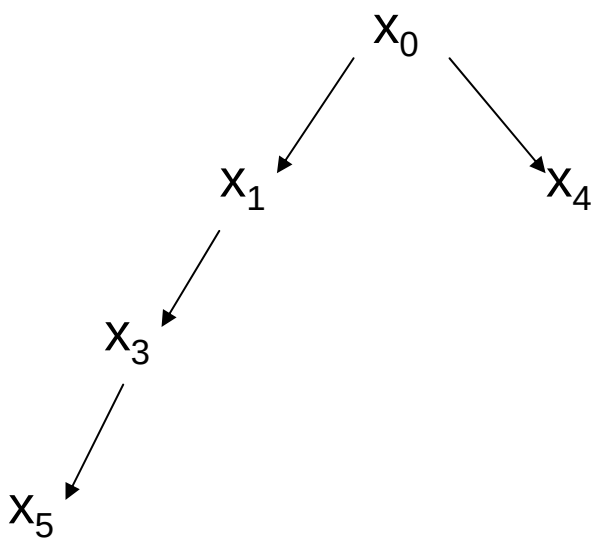


	x_1	x_2	x_3	x_4	x_5
père	x_0		x_1	x_0	x_3

EXEMPLE

	x_1	x_2	x_3	x_4	x_5
père	x_0	$\varnothing$	x_1	x_0	x_3

arborescence "en profondeur" parcourue:



5.3.2 PARCOURS D'UN GRAPHE NON ORIENTE

Utilisation des deux procédures précédentes légèrement modifiées:

remplacer	**Γ(x)={successeurs de x}**
par	**V(x)={voisins de x}**

5.3.3 PARCOURS COMPLET D 'UN GRAPHE

but: traitement en chaque sommet du graphe (une seule fois)**.**

A partir de Profondeur et en utilisant Visiter on obtient:

```
void Parcours_Graphe (CTab1 marque; CTab2 père);
début
    pour tout  x∈X  faire  marque[x]=faux; père[x]= ∅ ; fait;
    pour tout x∈X faire
          si non marque[x] alors
                marque[x]=vrai;
                Visiter (x,marque,père);
          finsi;
    fait;
fin ;
```

complexité: O(m)

On prendra soin d'ajouter dans Visiter

"traiter(y);"

entre "marque[y]=vrai;" et "père[y]= x;"

5.4 CONNEXITE

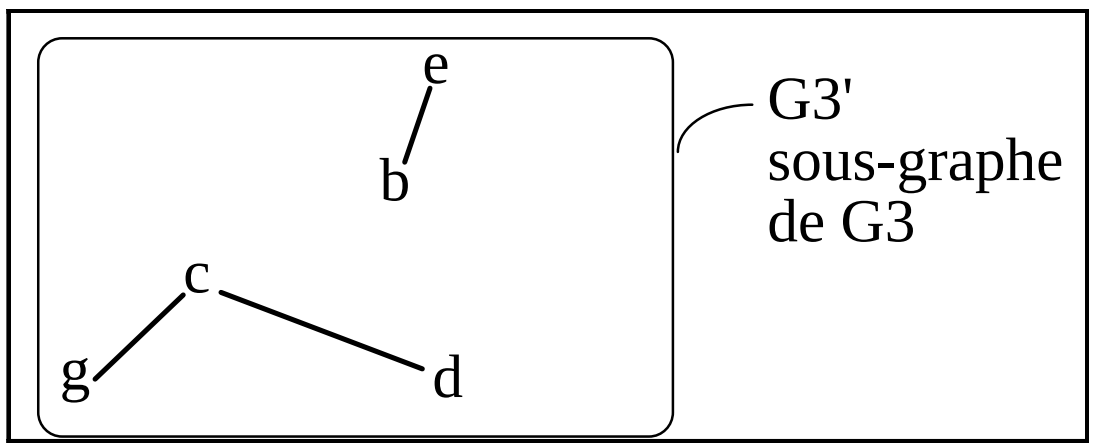
5.4.1 DEFINITIONS

sous-graphe : G'=(X',A') de G (orienté ou non orienté):

$$X' \subseteq X \quad A' \subseteq A \qquad \text{et}$$

$$\forall x,y \in X',\ [x,y] \in A \Leftrightarrow [x,y] \in A'$$

EXEMPLE (suite)



---

sous-ensemble maximal pour une propriété $\wp$:

sous-ensemble tel que l'ajout d'un élément lui fait perdre la propriété $\wp$

Composante connexe de G: sous-graphe de G connexe maximal

EXEMPLE (suite)

G3' a 2 composantes connexes

5.4.2 DETERMINATION DES COMPOSANTES CONNEXES D'UN GRAPHE

entrée: graphe G=(X,U)

(G est non orienté ou on ne tient pas compte de l'orientation)

sortie: liste des composantes

principe

-déterminer une composante connexe C
en partant d'un sommet quelconque
- retirer C du graphe et recommencer

Procédure "aveugle" de recherche des composantes connexes:

E = X;

Tant que **E non vide** faire

marquer + un sommet x de E;

tant que **c'est possible** faire

marquer + tout voisin (non encore marqué +)

d'un sommet déjà marqué + ;

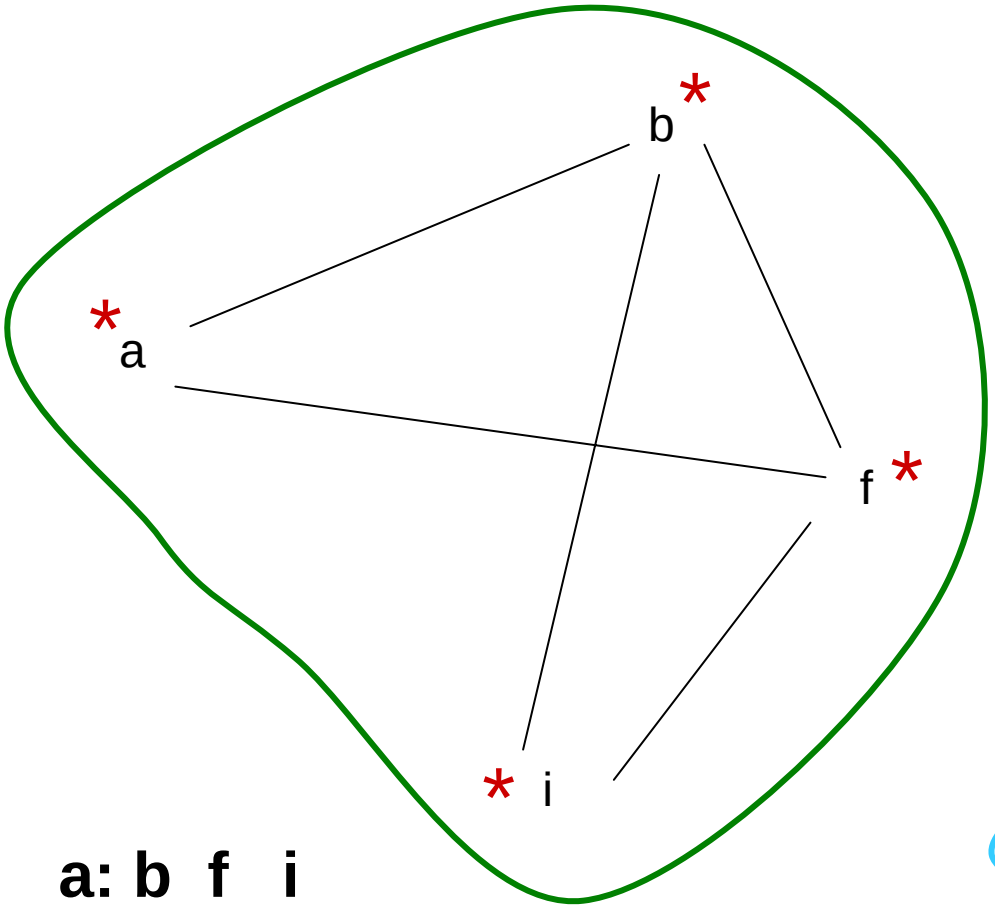
fait;

Écrire C l'ensemble des sommets marqués +; le sous graphe de G dont les sommets sont ceux de C est une composante f-connexe de G;

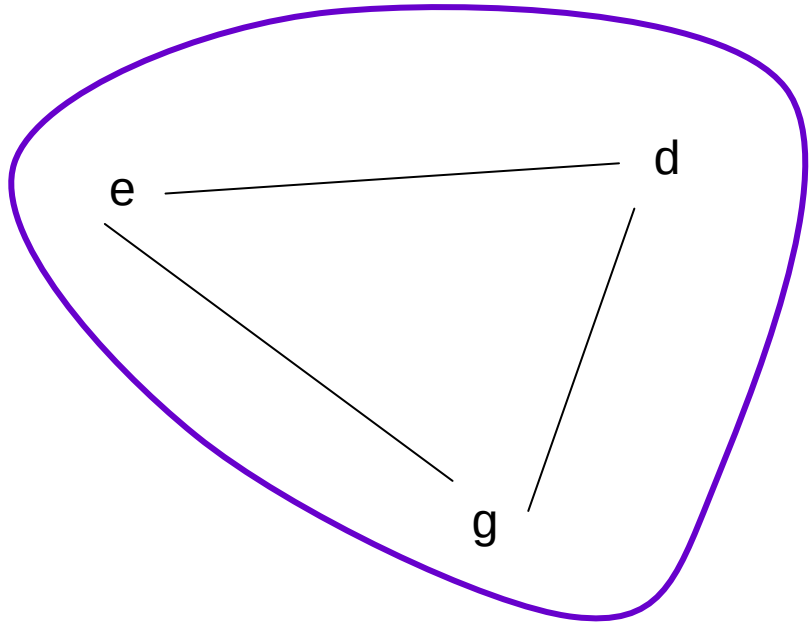
E = E - C; C = Ø;

fait; fin;

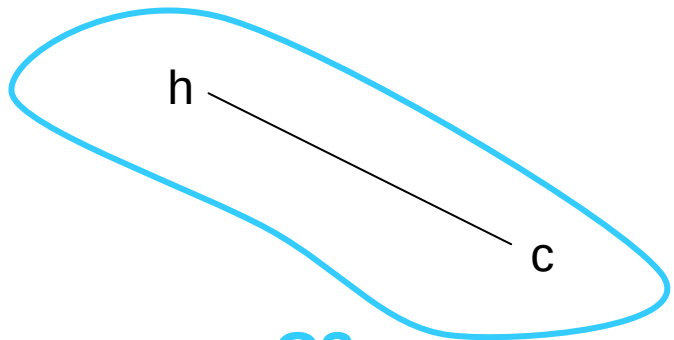
complexité: $O(n^2)$



C1

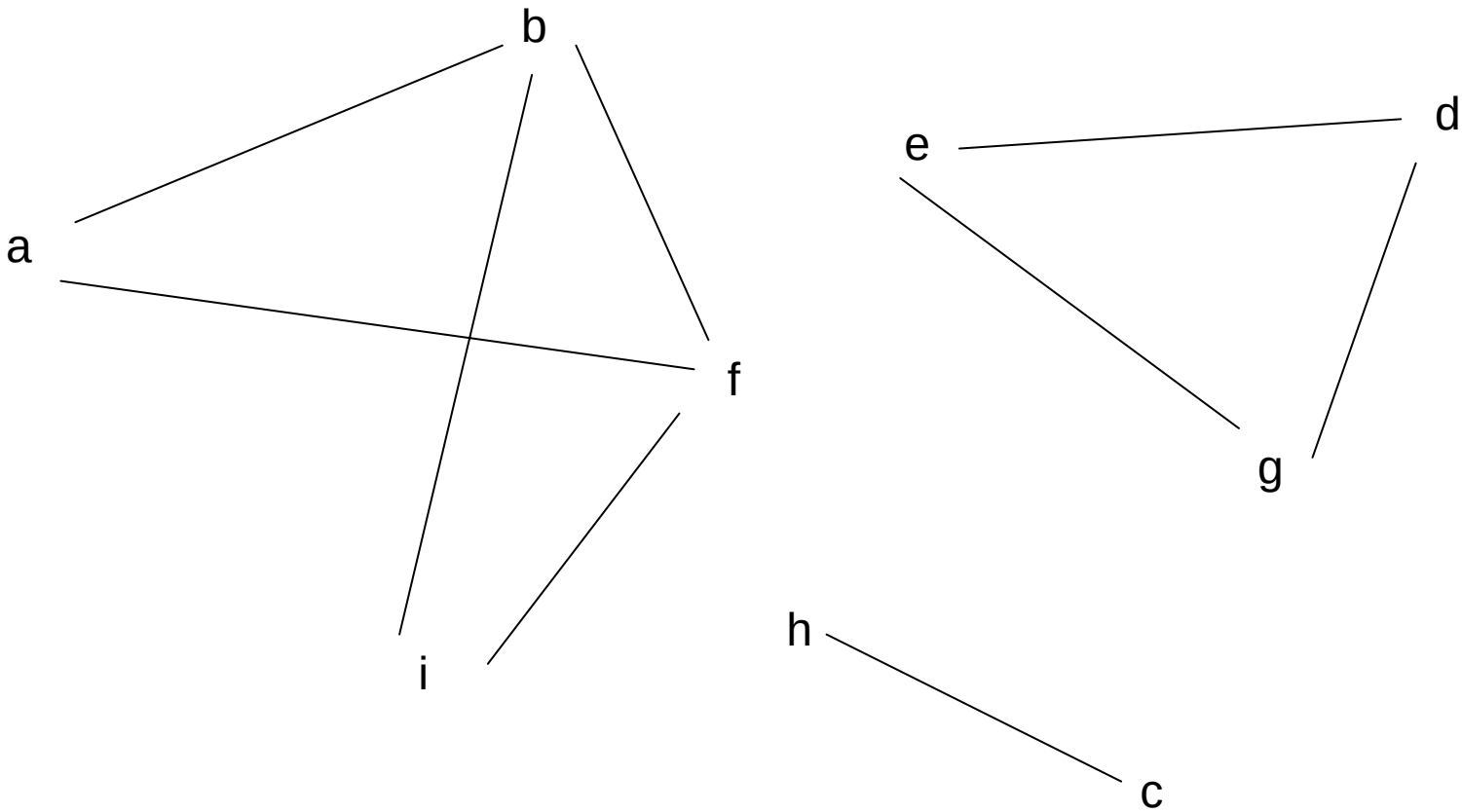


C2



C3

a: b f i
b: a f i
f: a b i
i: a b f


b
a
f
i
e
d
g
h
c

Recherche par un parcours en profondeur:

void **Comp_connexes** **(CComp comp** *//liste des composantes, comp[I]={sommets de la Ième composantes}*)

CTab1 marque; **CTab2 père;**

CGraphe G'=(X',A'); **entier nc**;

début

X'=X; A'=A;

nc=0; *(nombre de composantes)*

pour tout **x∈X'** faire

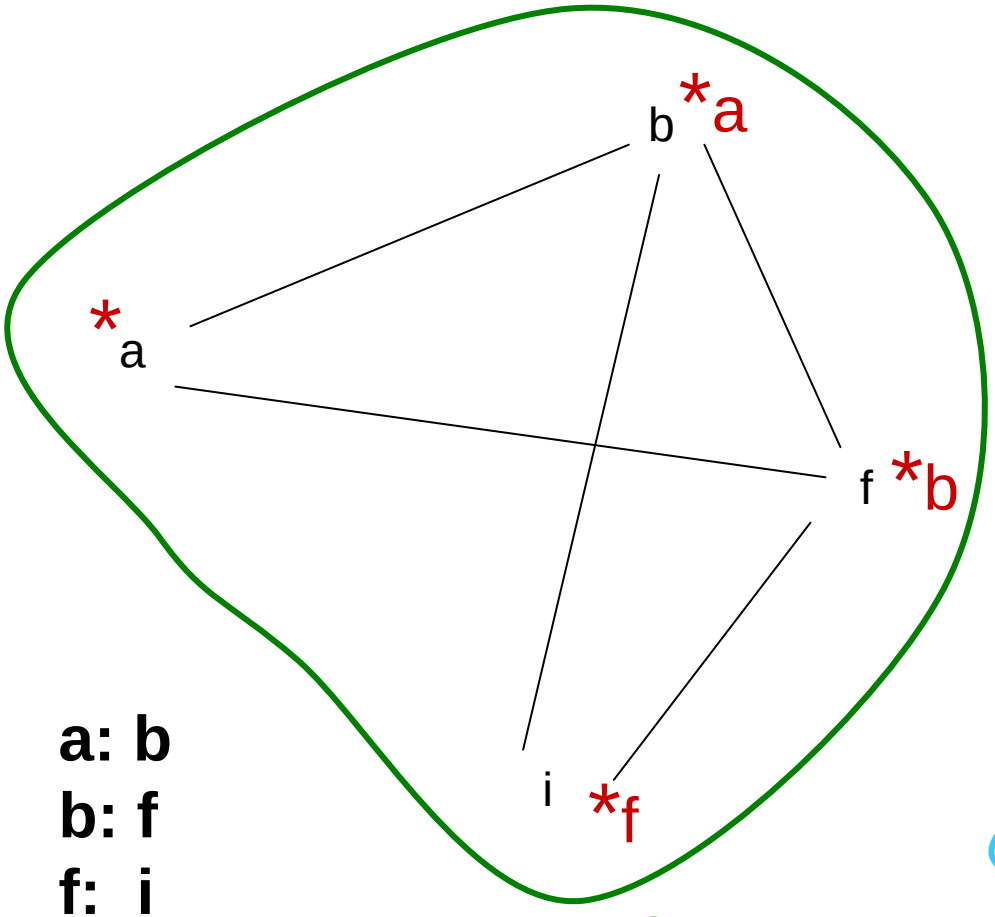
marque[x]=faux;

père[x]= ∅ ;

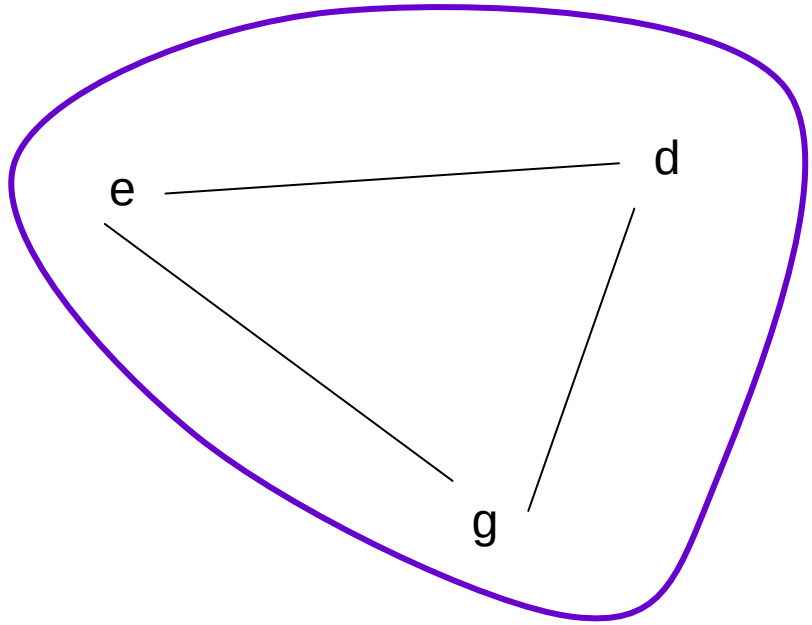
fait;

```
pour tout x∈X' faire
        nc=nc+1;
        si non marque[x] alors
                marque[x]=vrai;
                Visiter(x,marque,père);
        finsi;
        Comp(nc)={x∈X'|marque[x]=vrai};
        X'=X'-Comp(nc) ;
        A'=sous-graphe de G défini par X';
fait;
fin;
```

complexité: O(m)



C1



C2

C3

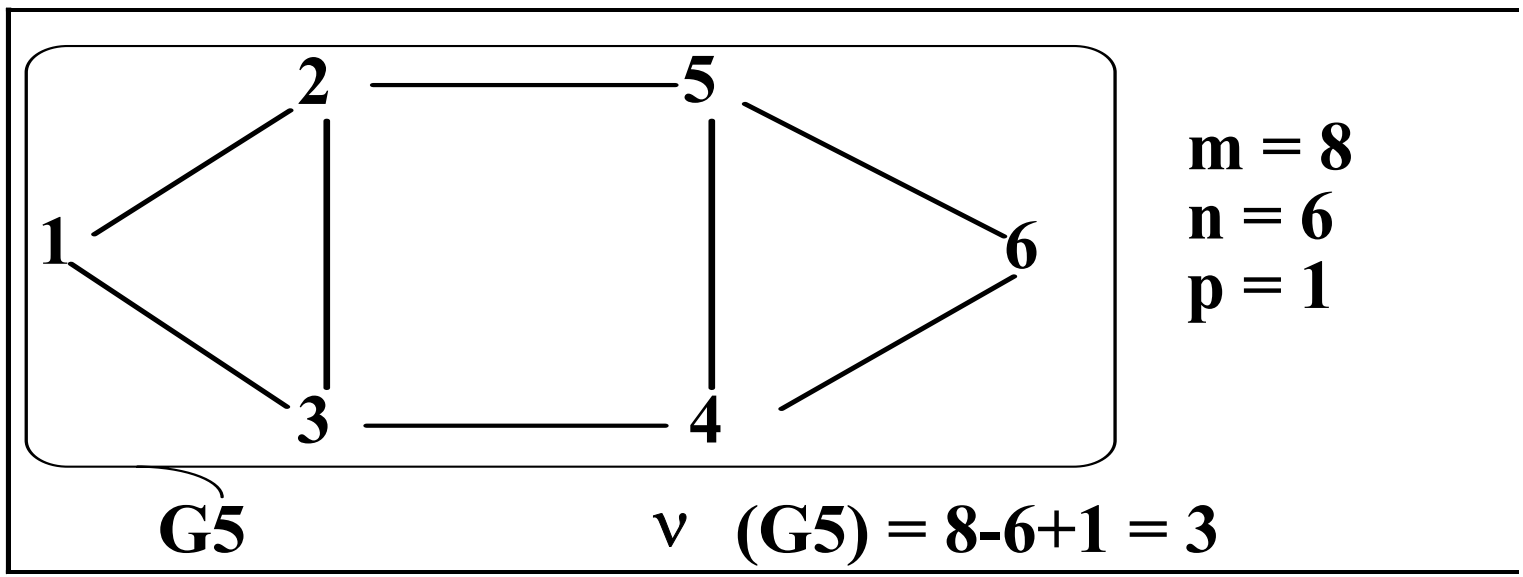
a: b
b: f
f: i
i: -
f: -
b: -
a: -

5.4 ARBRES ET ARBORESCENCES

nombre cyclomatique :

G graphe de n sommets, m arêtes et p composantes connexes

$$\nu(G) = m - n + p$$



ν(G) = nombre d'éléments d'une base de cycles

arbre: définitions équivalentes

H=(X,U), n sommets ($n \geq 2$)

(1) H connexe et sans cycle

(2) H sans cycle et n-1 arêtes

(3) H connexe et n-1 arêtes

(4) H sans cycle et en ajoutant une arête on crée un et un seul cycle

(5) H connexe et si on supprime une arête , il n'est plus connexe

(6) une chaîne et une seule entre toute paire de sommets

démonstration

(1) H connexe et sans cycle

(2) H sans cycle et n-1 arêtes

(3) H connexe et n-1 arêtes

1⇒2 H sans cycle $\Rightarrow$ $\nu(H)=0$, et H connexe

$\Rightarrow p=1 \Rightarrow m-n+1=0 \Rightarrow m=n-1$

2⇒3 H sans cycle $\Rightarrow$ $\nu(H)=0$, et $m=n-1$

$\Rightarrow (n-1)-n+p=0 \Rightarrow p=1 \Rightarrow$ H connexe

démonstration

(3) H connexe et n-1 arêtes

(4) H sans cycle et en ajoutant une arête on crée un et un seul cycle

3⇒4 H connexe ⇒ p=1, et m=n-1 ⇒

$\nu(H)=(n-1)-n+p=0 \Rightarrow$ H est sans cycle.

Si on ajoute une arête, on obtient H' t.q. m'=m+1=n, n'=n et p'=1

⇒ $\nu(H')=n-n+1=1 \Rightarrow$ 1 cycle

Démonstration (suite)

(4) H sans cycle et en ajoutant une arête on crée un et un seul cycle

(5) H connexe et si on supprime une arête , il n'est plus connexe

4⇒5 si H n'est pas connexe, ∃ {x,y} non reliés par une chaîne ⇒ en ajoutant l'arête [x,y] on ne crée pas de cycle et (4) n'est pas vérifié ⇒ H est connexe. p=1⇒m=n-1. Si on ôte une arête, on obtient H" t.q. m"=n-2 et n"=n ⇒ ν(H")=m"-n"+p"=0 (car sans cycle)

⇒ n-2-n+p"=0 ⇒ p"=2 ⇒ H" est non connexe

Démonstration (suite)

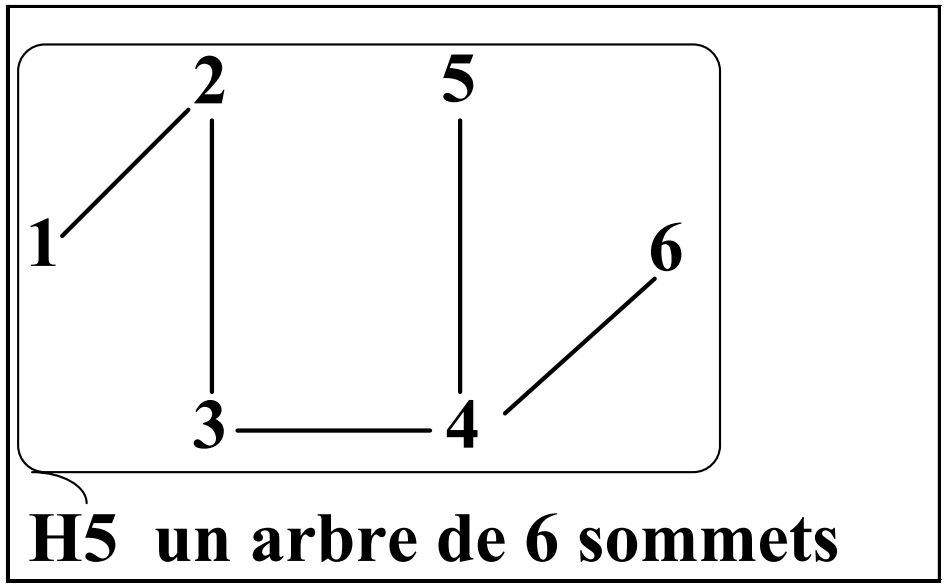
(5) H connexe et si on supprime une arête , il n'est plus connexe

(6) une chaîne et une seule entre toute paire de sommets

(1) H connexe et sans cycle

5⇒6 H connexe ⇒∃ au moins une chaîne entre 2 sommets; la suppression d'une arête rend H non connexe ⇒ cette chaîne est unique

6⇒1 ∃ une chaîne entre 2 sommets ⇒ H est connexe; elle est unique ⇒pas de cycle



H5 un arbre de 6 sommets

graphes orientés

racine : sommet r tel qu'il existe un chemin de r à tout autre sommet du graphe

degré intérieur(resp.extérieur): d'un sommet x: nombre d'arcs d'extrémité terminale (resp. initiale) x notés $d^-(x)$ et $d^+(x)$

arborescence: définitions équivalentes

H=(X,U) n sommets ($n \geq 2$)

(1) **H arbre avec une racine r**

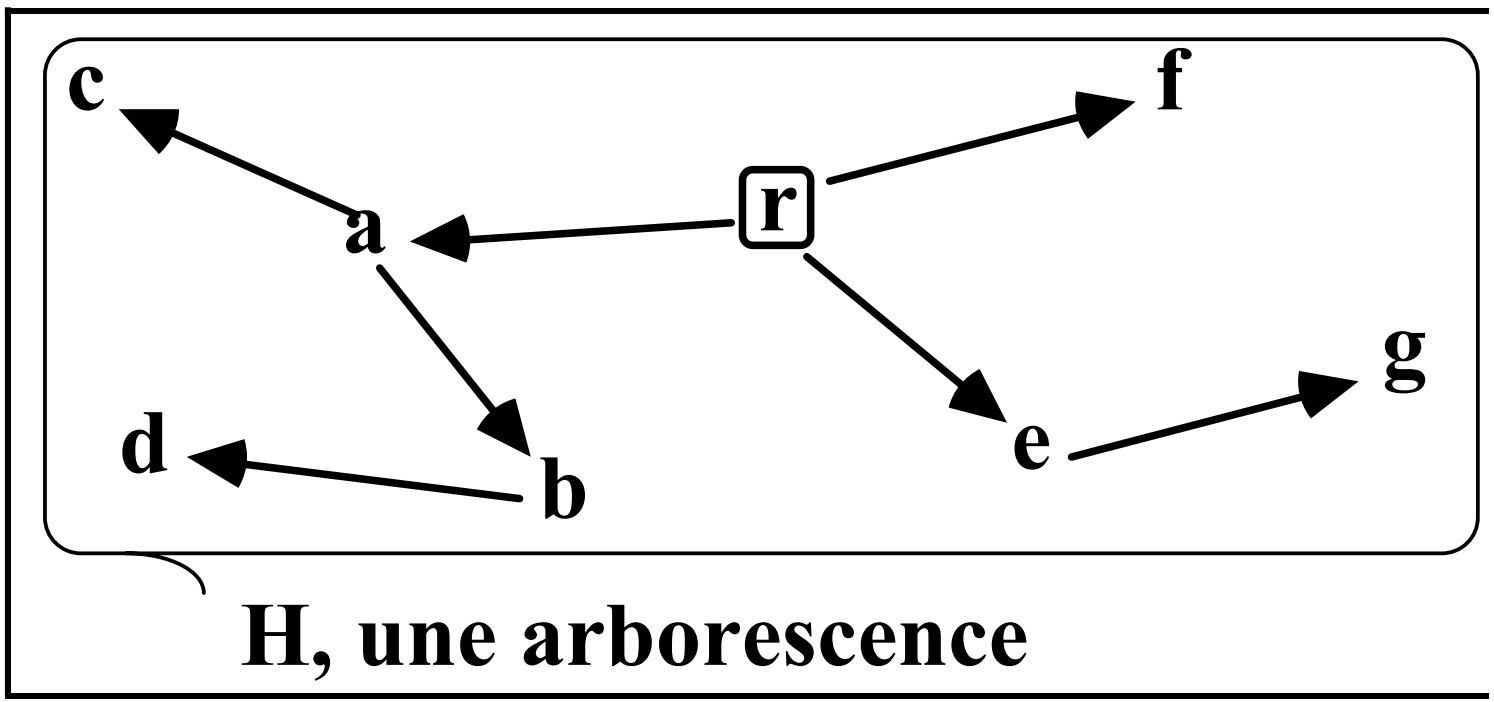
(2) **$\exists$ r $\in$ X, relié à tout x $\in$ X par un chemin unique**

(3) **H connexe et**

$\exists$ r $\in$ X t.q. $d^-(r)=0$ et

$d^-(x)=1$ pour tout $x \neq r$

(4) **H sans cycle et $\exists$ un sommet $r \in X$ t. q. $d^-(r)=0$ et $d^-(x)=1$ pour tout $x \neq r$**



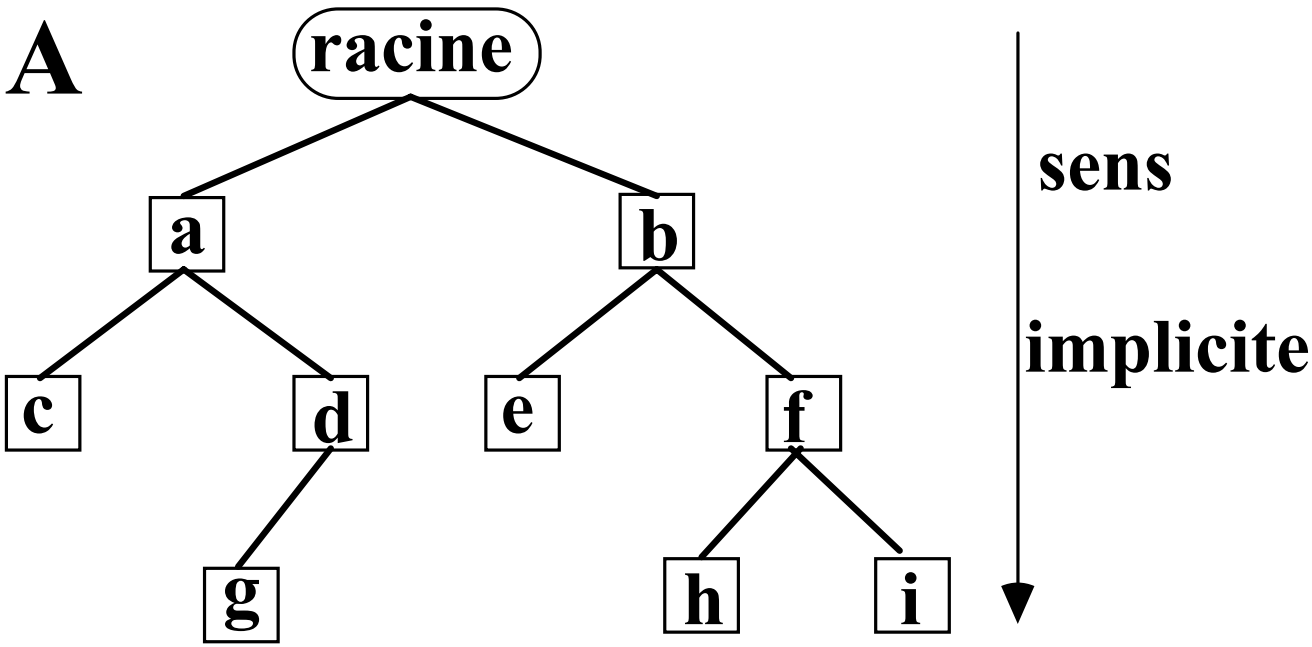
arborescence =

"arbre enraciné" (rooted tree)

= "arbre" en informatique

Ex: arbre généalogique, tournois, arbre des espèces animales,...

arborescence binaire:



(voir cours N° 3)