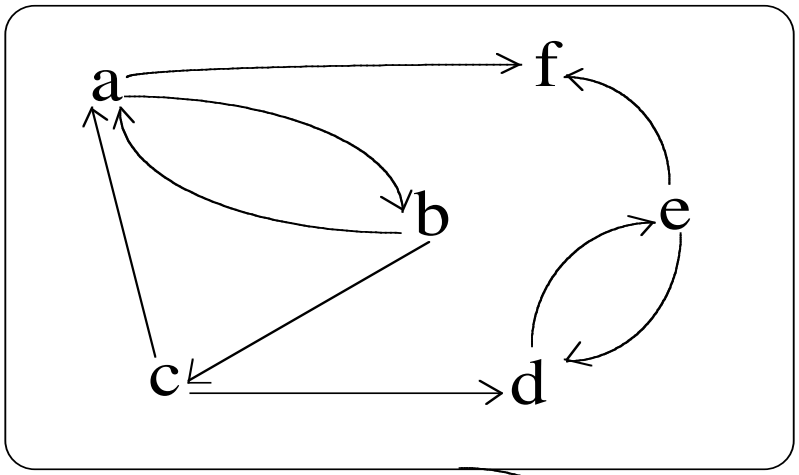
NFP136- Cours5
THEORIE DES GRAPHES

PLAN

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

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)}

G : **X** → *P* **(X)**

x → G(x) = {successeurs de x}

EXEMPLE(suite) **G1(b)={a,c} G1(f)={Ø} G1(d)={e}**

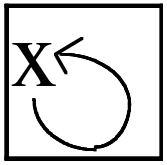
Chemin : suite d'arcs telle que l'extrémité terminale d'un arc coïncide avec l'extrémité initiale de l'arc suivant

---

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

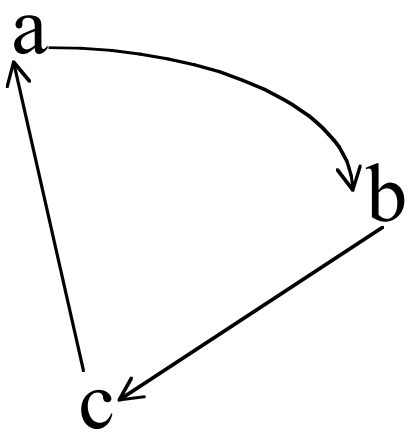
---

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

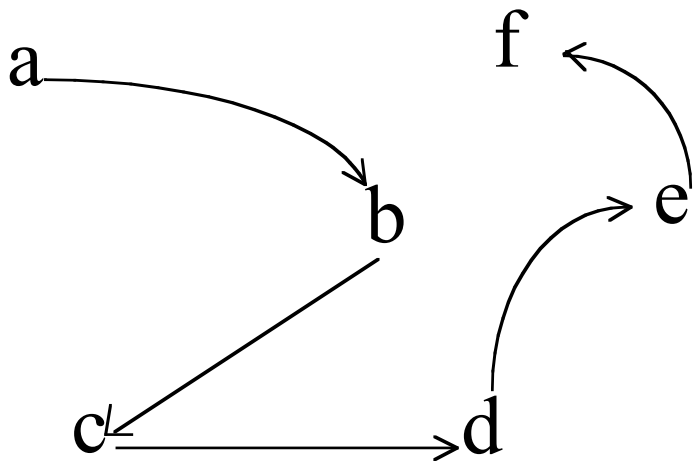
---

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..

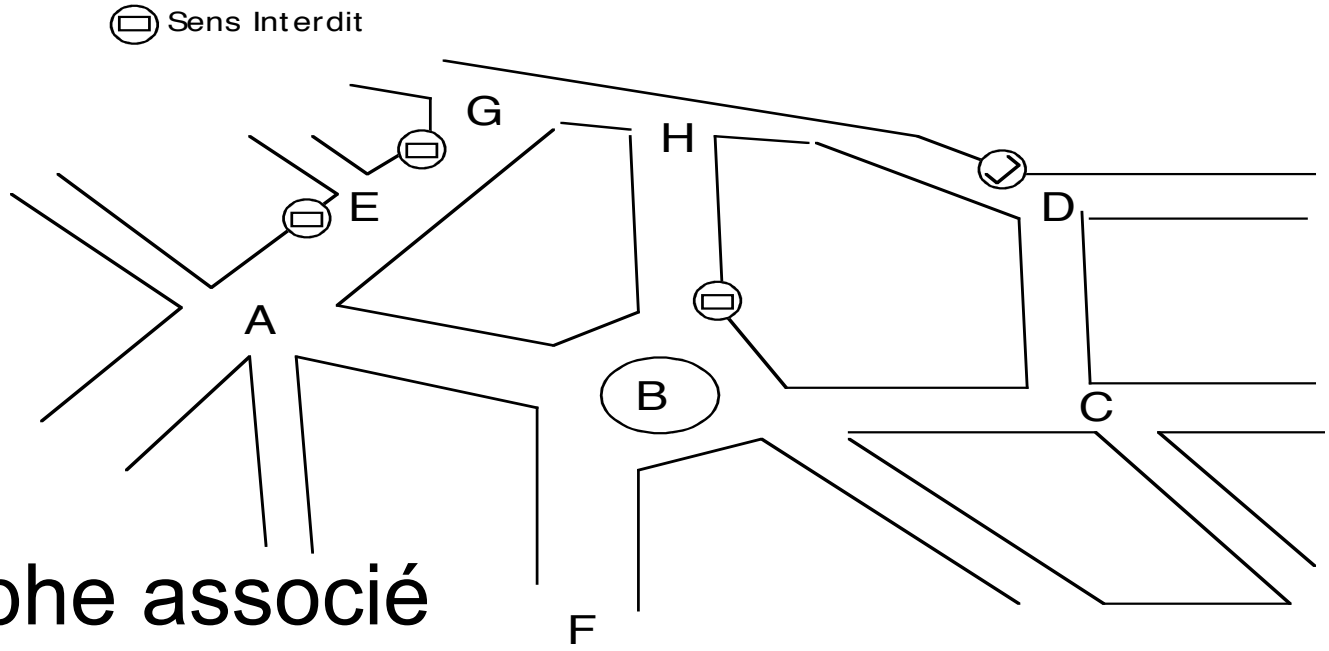
- **Résolution de problèmes**

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

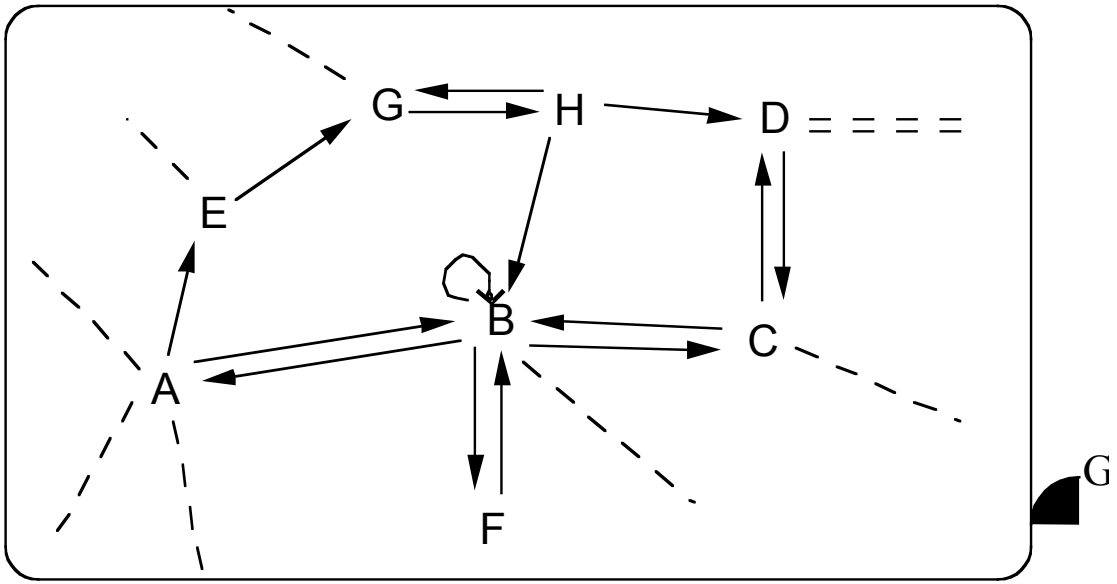
- **Outils**

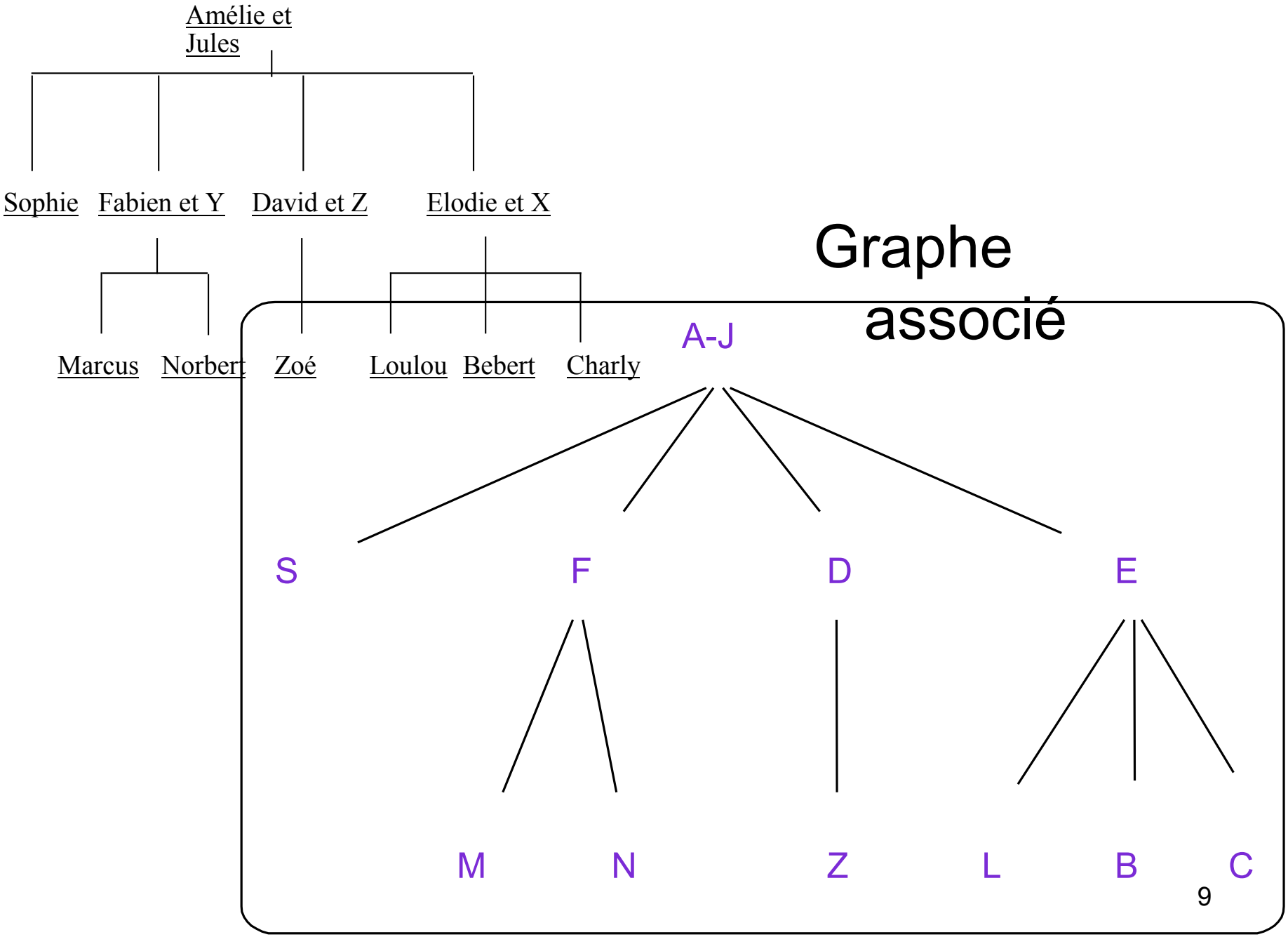
Exemple: structures de données,.

- **...**



Graphe associé

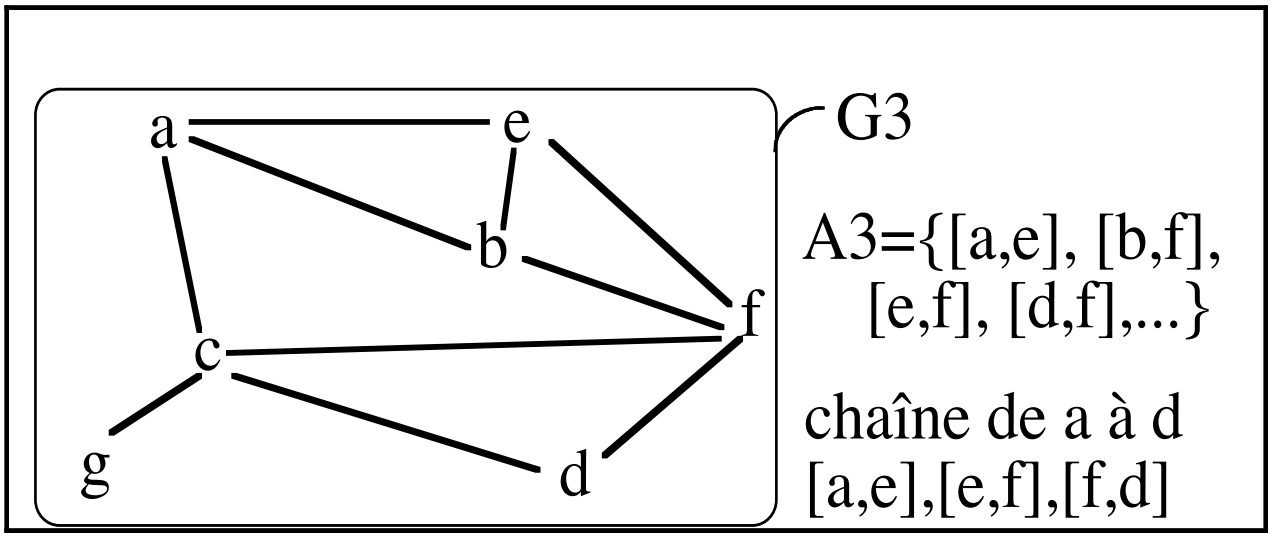



Amélie et Jules
Sophie
Fabien et Y
David et Z
Elodie et X
Marcus
Norbert
Zoé
Loulou
Bebert
Charly
Graphe associé
A-J
S
F
D
E
M
N
Z
L
B
C

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,**

---

degré $x \in X$

d(x) = nombre de voisins de x

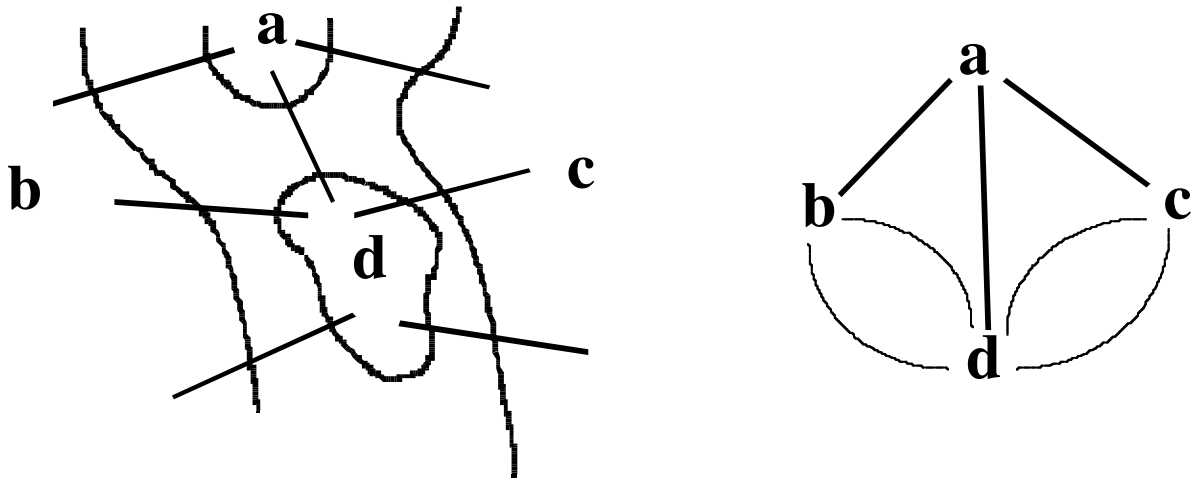
EXEMPLE (suite) **dans G3 d(c) = 4 d(b) = 3**

---

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

Théorème d'Euler

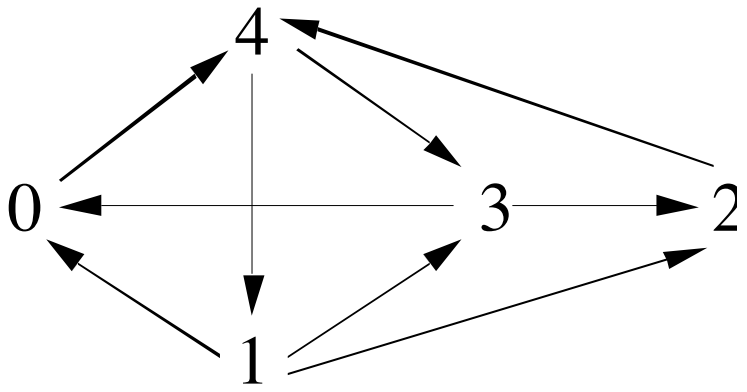
Un multigraphe connexe admet une chaîne eulérienne si et seulement si le nombre de sommets de degré impair est 0 ou 2



1766 La Pregel à Koenigsberg

5.2 REPRESENTATION D 'UN GRAPHE

Exemple

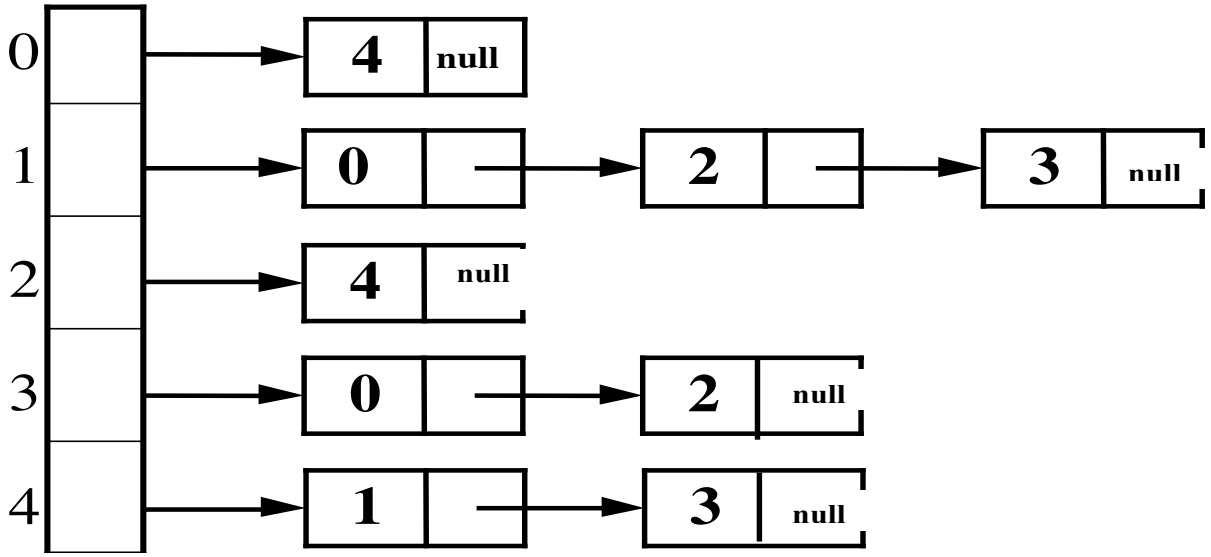


5-2-1 UNE MATRICE

	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

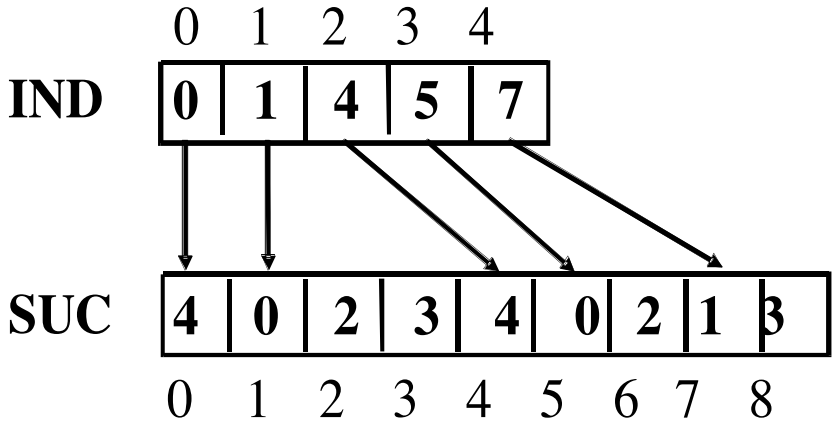
5-2-2 TABLEAU ADRESSANT DES LISTES CHAÎNEES DE SUCCESSEURS



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

  (ex: nombre de successeurs ?)

5-2-3 DEUX TABLEAUX



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

SUC : tableau des successeurs

G(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
G(3) =
{SUC[5],SUC[6]}={0,2}

5-2-4 Une implémentation Java utilisant un tableau de listes de successeurs

D'abord une classe Sommet :

```
public class Sommet{
        private int num;
        private Liste succListe;
            //pointe la liste des numeros des successeurs du sommet

public Sommet( int a ){
        num=a;
        succListe = null;
}
public int numSommet(){
        return num;
}
public Liste lesSucc(){
        return succListe;
}
```

Utilise la classe Liste d'entiers

D’abord une classe Sommet - suite:

```
public void ajouteSucc (int b){
        succListe = new Liste(b, succListe);
}

public void affichage(){
        System.out.println("Sommet "+ num);
        System.out.print("Successeurs : ");
        Liste L=succListe;
        while (L != null){
                System.out.print (L.tete()+ " , ");
                L=L.queue();
        }
        System.out.println("");
}
}//fin classe sommet
```

Voici la classe Graphe :

```
public class Graphe{
	private Sommet[] tabG;

public Graphe( int nbSom ){
	//cree un graphe etant donne un nombre de sommets
	//les sommets sont alors numerotes 0,1,..., nbSom-1
	tabG = new Sommet[nbSom];
	//creation des Sommets
	for (int i=0; i< tabG.length; i++){
		tabG[i]= new Sommet(i);    //sommet numero i
	}
}
```

Voici la classe Graphe – un deuxième constructeur:

```
public Graphe( int[] tabNumSom ){
//cree un graphe etant donne le tableau des numeros de sommets
       tabG = new Sommet[tabNumSom.length];
       //creation des Sommets
       for (int i=0; i< tabG.length; i++){
              tabG[i]= new Sommet(tabNumSom[i]);
                            //numero du i_eme sommet
       }
}
```

Voici la classe Graphe – suite :

```
public int trouveIndiceSom (int num) {
        //trouve dans tabG l'indice dont le numero du sommet est num
        for (int i=0; i< tabG.length; i++){
                if (tabG[i].numSommet()==num) return i;
        }
        return (-1); //echec
}
public void ajouteArc(int x, int y){
        //d'abord trouver l'indice de x
        int ind = trouveIndiceSom(x);
        //ajouter y comme successeur du sommet tabG[ind];
        tabG[ind].ajouteSucc(y);
}
public void affichage(){
        System.out.println("Debut graphe");
        for (int i=0; i< tabG.length; i++){
                tabG[i].affichage();
        }
        System.out.println("Fin graphe");
}
```

Mise en commun sur un exemple :

```
import java.io.*;
public class testGraphe{
    static BufferedReader in =
        new BufferedReader(new InputStreamReader(System.in));
    public static void main (String[] args) {
        int[] tabNum= {5, 10, 15};
        Graphe G=new Graphe(tabNum);
        G.ajouteArc (5,10);
        G.ajouteArc (5,15);
        G.affichage();

}//fin main
}
```

Mise en commun sur un exemple :

```
>java testGraphe
Debut graphe
Sommet 5
Successeurs : 15 , 10 ,
Sommet 10
Successeurs :
Sommet 15
Successeurs :
Fin graphe
```

Mise en commun sur un autre exemple :

```
import java.io.*;
public class testGraphe1{
    static BufferedReader in =
        new BufferedReader(new InputStreamReader(System.in));
    public static void main (String[] args) {
        Graphe G=new Graphe(6);
        G.ajouteArc (0,4);
        G.ajouteArc (0,1);
        G.ajouteArc (4,1);
        G.ajouteArc (4,3);
        G.ajouteArc (1,3);
        G.ajouteArc (1,5);
        G.ajouteArc (3,5);
        G.ajouteArc (2,3);
        G.ajouteArc (2,5);
        G.affichage();
}
}
```

Mise en commun sur un autre exemple :

```
>java testGraphe1
Debut graphe
Sommet 0
Successeurs : 1 , 4 ,
Sommet 1
Successeurs : 5 , 3 ,
Sommet 2
Successeurs : 5 , 3 ,
Sommet 3
Successeurs : 5 ,
Sommet 4
Successeurs : 3 , 1 ,
Sommet 5
Successeurs :
Fin graphe
```

5.3 EXPLORATION D ’UN GRAPHE

G=(X,A) donné avec |X|=n |A|=m

Descendants d'un sommet

Définition

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

Détermination de $D(x_0)$

Principe

marquer x_0

tant que **∃ (x,y)∈A t.q. x marqué et y non marqué**

faire

marquer y;

fait; D(x)={sommets marqués}

Marquage des descendants d'un sommet- procédure aveugle

Marquage des descendants d'un sommet- **procedure aveugle** :

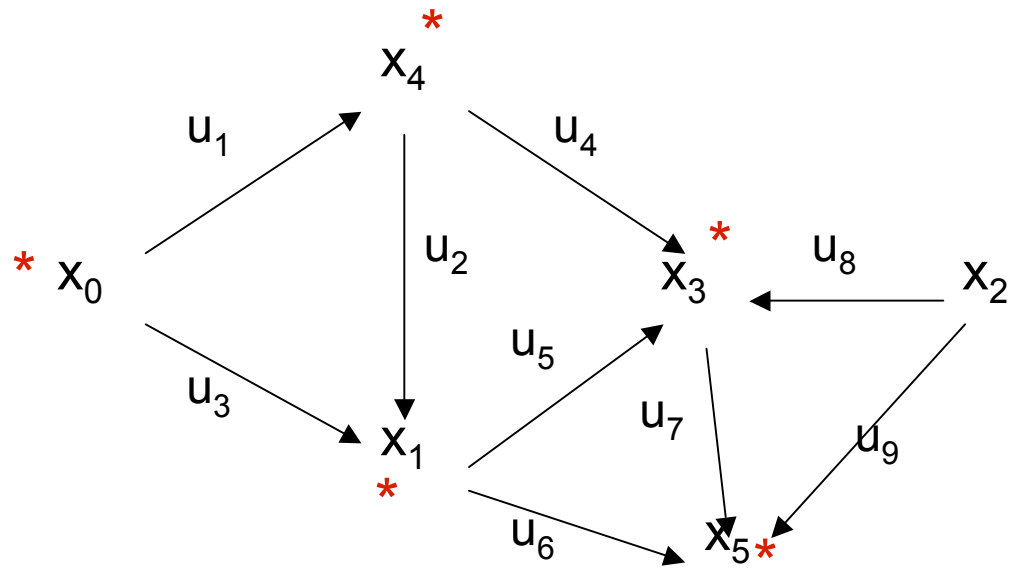
```
public void aveugle (int x, boolean[] marque, int[] pere){
  int indx= trouveIndiceSom(x);
  marque[indx]=true;
  boolean modif = true;

  while (modif) {
    modif=false;
    //on regarde tous les arcs
    for (int i=0; i< tabG.length; i++) if (marque[i]) {
      Liste L= tabG[i].lesSucc();
      while (L != null) {
        int indy= trouveIndiceSom(L.tete());
        if (!marque[indy]) {
          marque[indy]=true;
          pere[indy]=i;
          modif=true;
        }
        L=L.queue();
      }
    }
  }//fin while
}//fin aveugle
```

Complexité *(au pire)*: **O(m.n)**

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

EXEMPLE




```
Sommets marques depuis 0 :
0 a partir de -1
1 a partir de 0
3 a partir de 1
4 a partir de 0
5 a partir de 1
```

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

Marquage des descendants d'un sommet- parcours en largeur

Même principe que pour les arbres binaires : on marque tous les successeurs d'un sommet avant de traiter les autres sommets. On utilise une file.

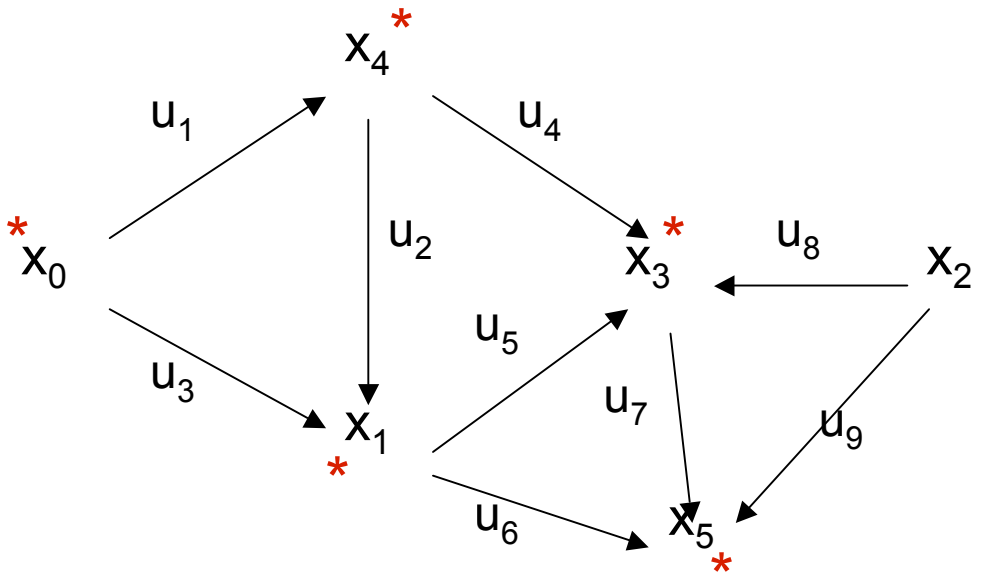
Parcours en largeur :

```
public void parcoursLargeur (int x, boolean[] marque, int[] pere, int[]
Long){
  File F= new File(tabG.length);
  int indx= trouveIndiceSom(x);
  Long[indx]=0;
  marque[indx]=true;
  F.enfiler(indx);

  while (! F.estVide()) {
    indx = F.defiler();
    Liste L= tabG[indx].lesSucc();
    while (L != null) {
      int indy= trouveIndiceSom(L.tete());
      if (!marque[indy]) {
        marque[indy]=true;
        pere[indy]=indx;
        Long[indy]=Long[indx]+1;
        F.enfiler(indy);
      }
      L=L.queue();
    }
  }//fin while
}//fin parcoursLargeur
```

Complexité *(au pire)*: **O(m)**

EXEMPLE

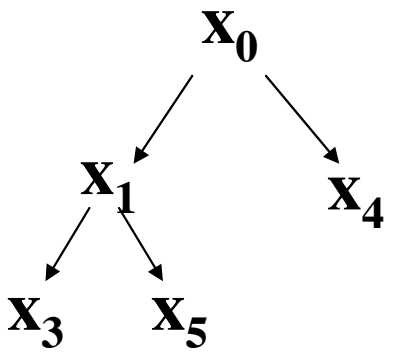


	x_1	x_2	x_3	x_4	x_5
père	x_0		x_1	x_0	x_1
L(x)	**1**		**2**	**1**	**2**

complexité: O(m) (hypothèse: m≥n)

EXEMPLE

arborescence "en largeur" parcourue:



Pour retrouver le chemin d'un sommet s à un sommet v, on appelle d'abord parcoursLargeur(s, marque, pere, Long)

```
//Imprime le chemin le plus court de s à v
début
   si   (v = s)   alors imprimer s ;
   sinon
        si (pere[v] = -1)  alors
                imprimer "pas de chemin de" s "à" v
        sinon
                Imprimer_chemin(s,pere[v]);
                imprimer v;
        finsi;
   finsi;
fin
```

Marquage des descendants d’un sommet- parcours en profondeur

On avance tant qu‘on peut dans le graphe, et le parcours des chemins laissés de coté se fait lorsqu'on ne peut plus avancer.

Parcours en profondeur- version récursive :

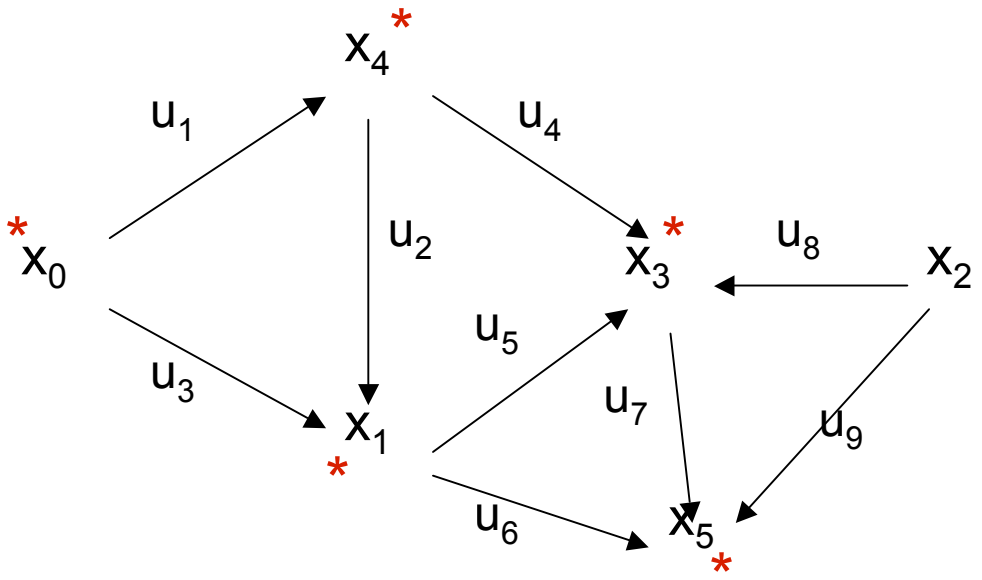
```
public void parcoursProfondeur (int x, boolean[] marque, int[] pere){
        int indx= trouveIndiceSom(x);
        marque[indx]=true;
        Liste L= tabG[indx].lesSucc();
        while (L != null) {
                int y= L.tete();
                int indy= trouveIndiceSom(y);
                if (!marque[indy]) {
                        pere[indy]=indx;
                        parcoursProfondeur (y,  marque, pere);
                }
                L=L.queue();
        }//fin while
}//fin parcoursProfondeur
```

Complexité *(au pire)*: **O(m)**

Parcours en profondeur en utilisant une pile:

```
public void parcoursProfondeurPile (int x, boolean[] marque, int[] pere){
        int indx= trouveIndiceSom(x);
        marque[indx]=true;
        Pile P = new Pile();
        P.empiler(indx);
        while (! P.estVide()) {
                int y= P.sommet();
                P.depiler();
                int indy= trouveIndiceSom(y);
                marque[indy]=true;
                Liste L= tabG[indy].lesSucc();
                while (L != null) {
                        int indz= trouveIndiceSom(L.tete());
                        if (!marque[indz]) {
                                pere[indz]=indy;
                                P.empiler(indz);
                        }
                        L=L.queue();
                }//fin while
        }
}//fin parcoursProfondeurPile
```

EXEMPLE

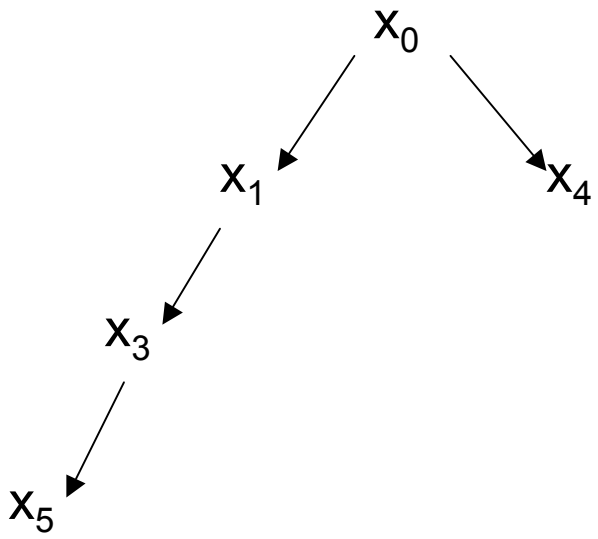


	x_1	x_2	x_3	x_4	x_5
père	x_0		x_1	x_0	x_3

EXEMPLE

	x_1	x_2	x_3	x_4	x_5
père	x_0	**-1**	x_1	x_0	x_3

arborescence "en profondeur" parcourue:



5.4 CONNEXITE

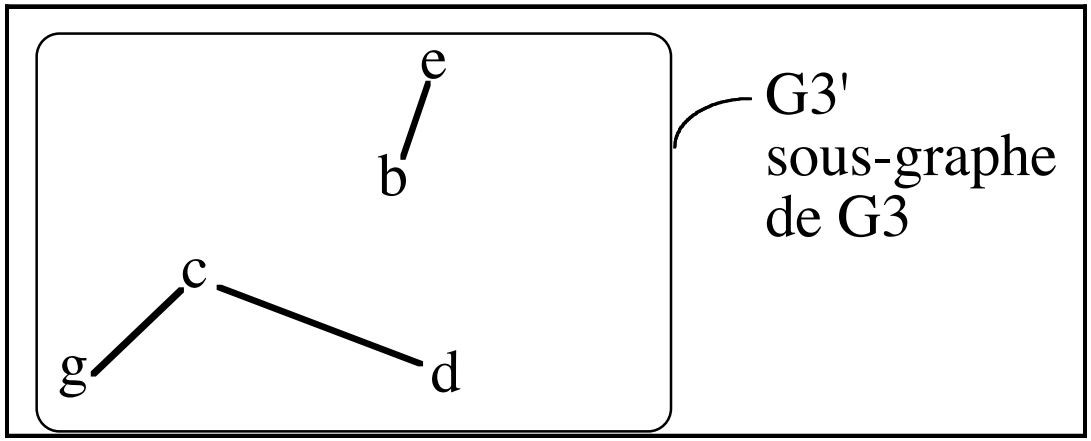
5.4.1DEFINITIONS

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 + ;

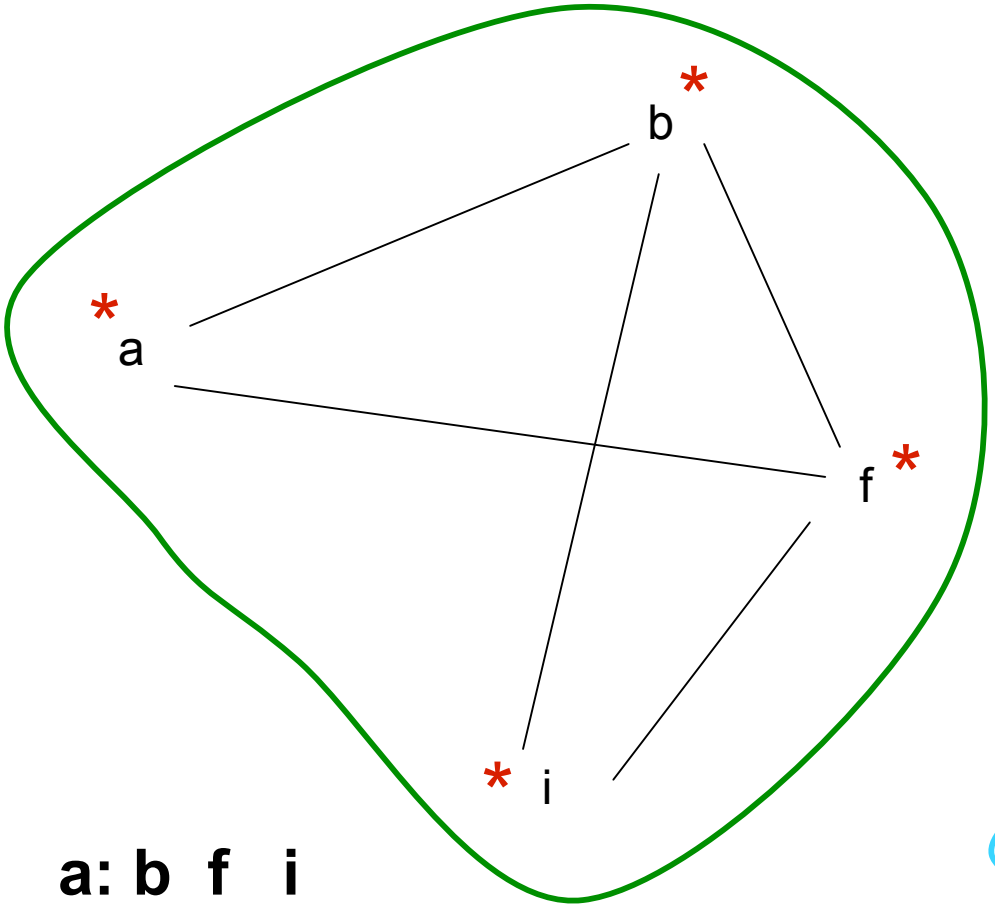
fait**;**

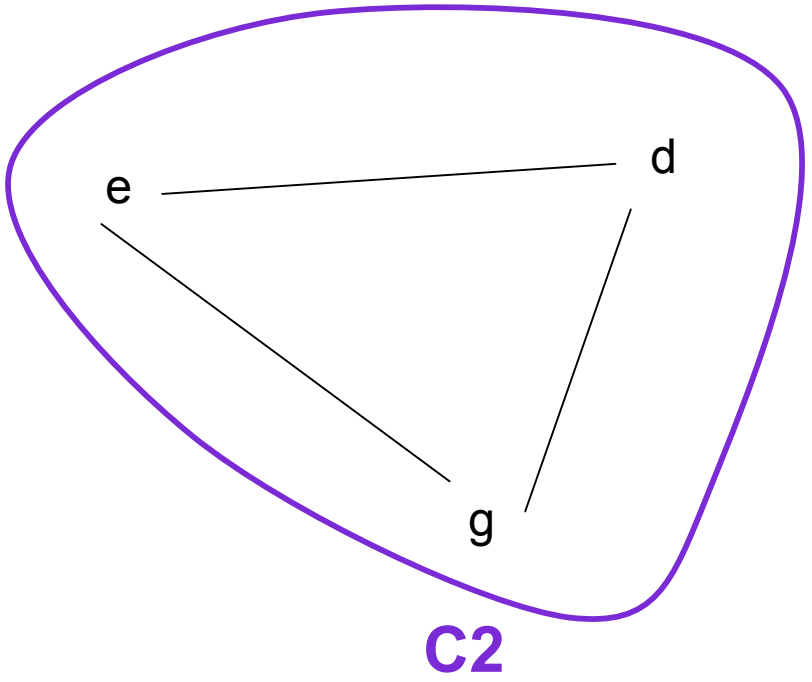
Écrire C l'ensemble des sommets marqués +; le sous graphe de G dont les sommets sont ceux de C est une composante f-connexe de G;

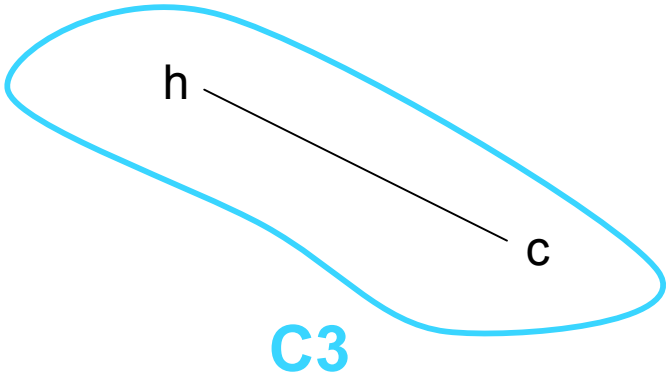
E = E - C; C = Ø;

fait**;** fin;

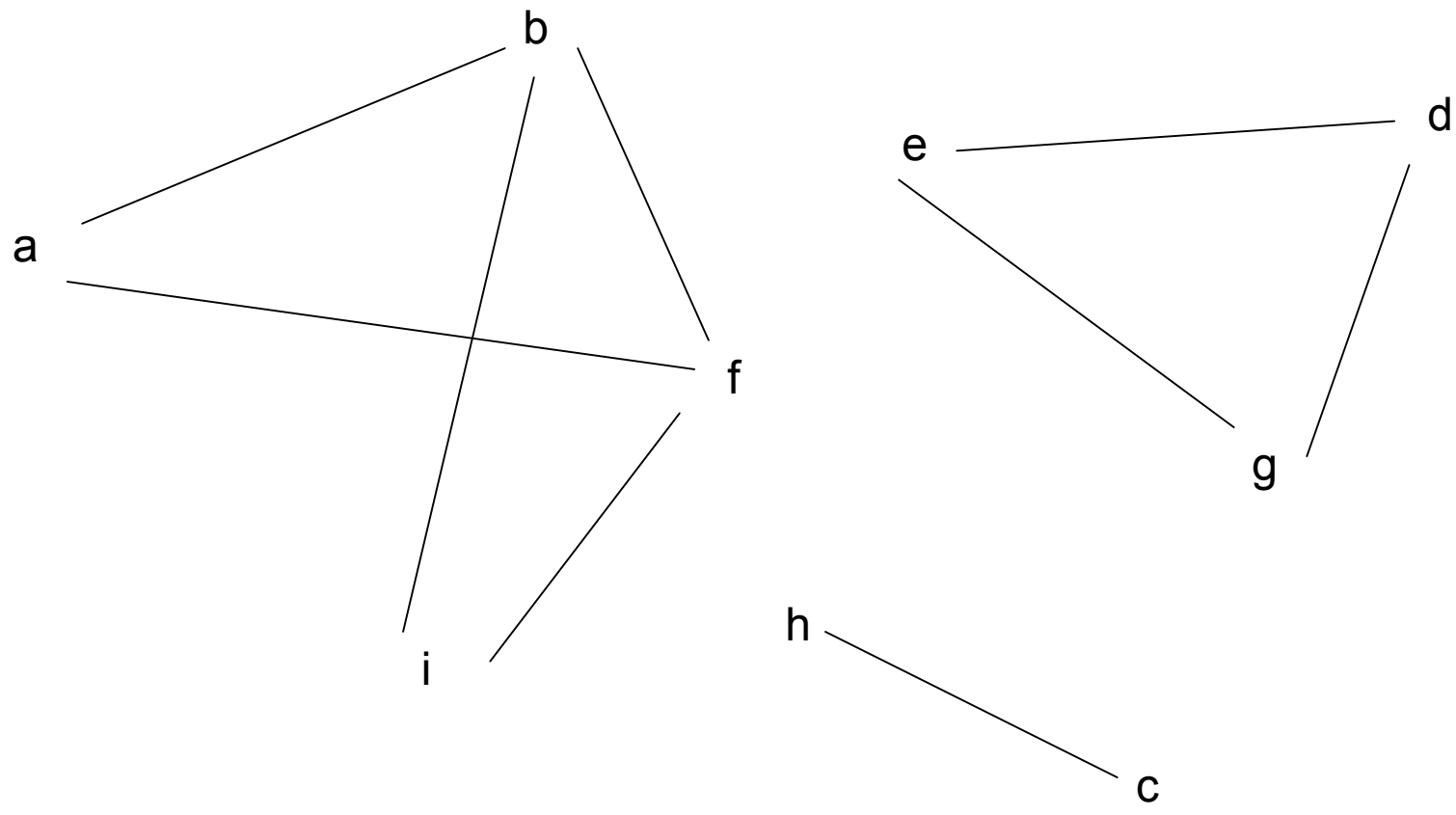
complexité: $O(n^2)$







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


b
a
f
e
d
g
i
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

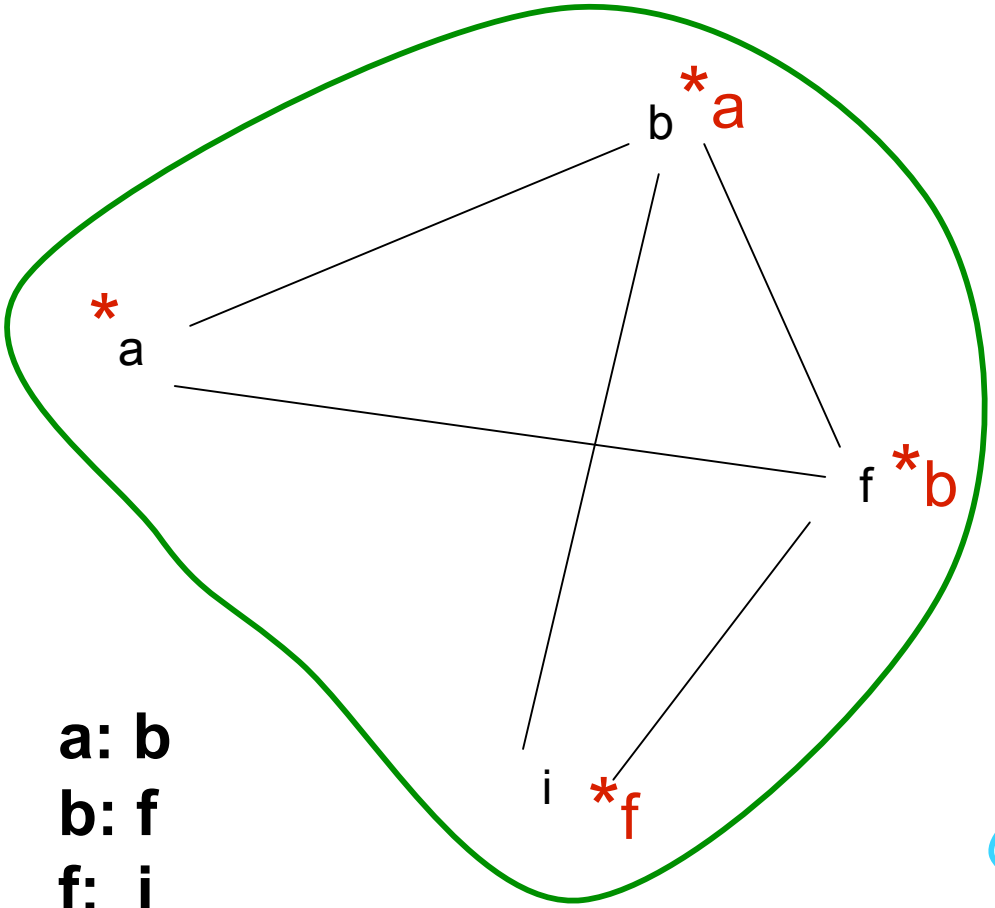
marque[x]=faux;

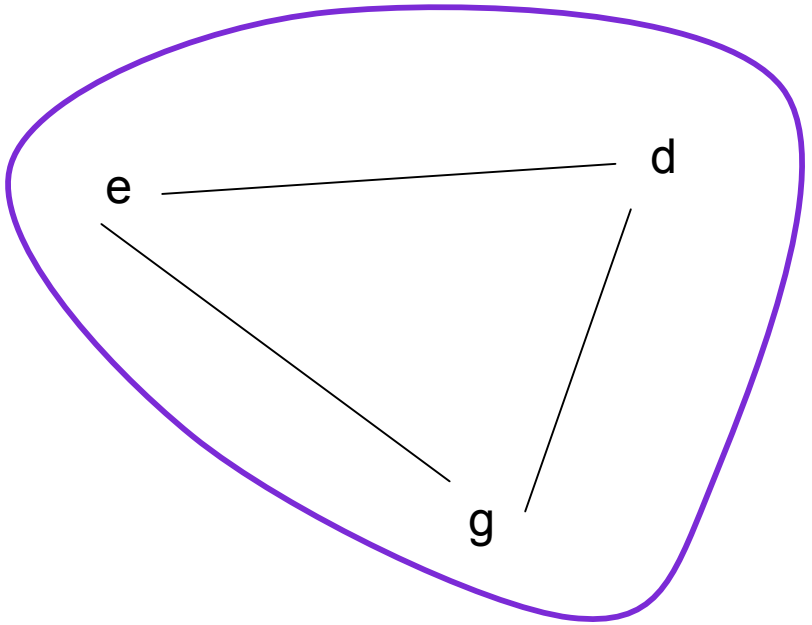
père[x]= -1 ;

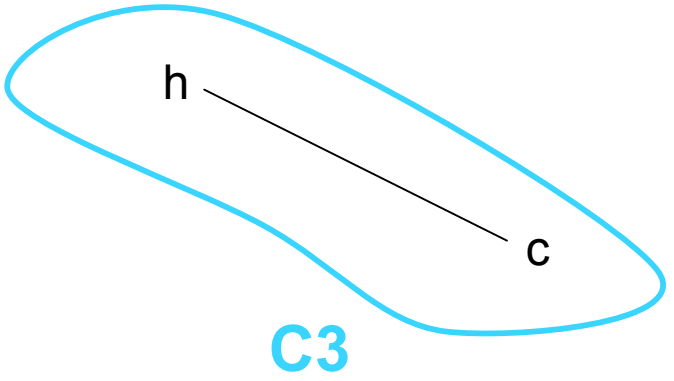
fait;

```
pour tout x∈X' faire
        nc=nc+1;
        si non marque[x] alors
                marque[x]=vrai;
                parcoursProfondeur(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)







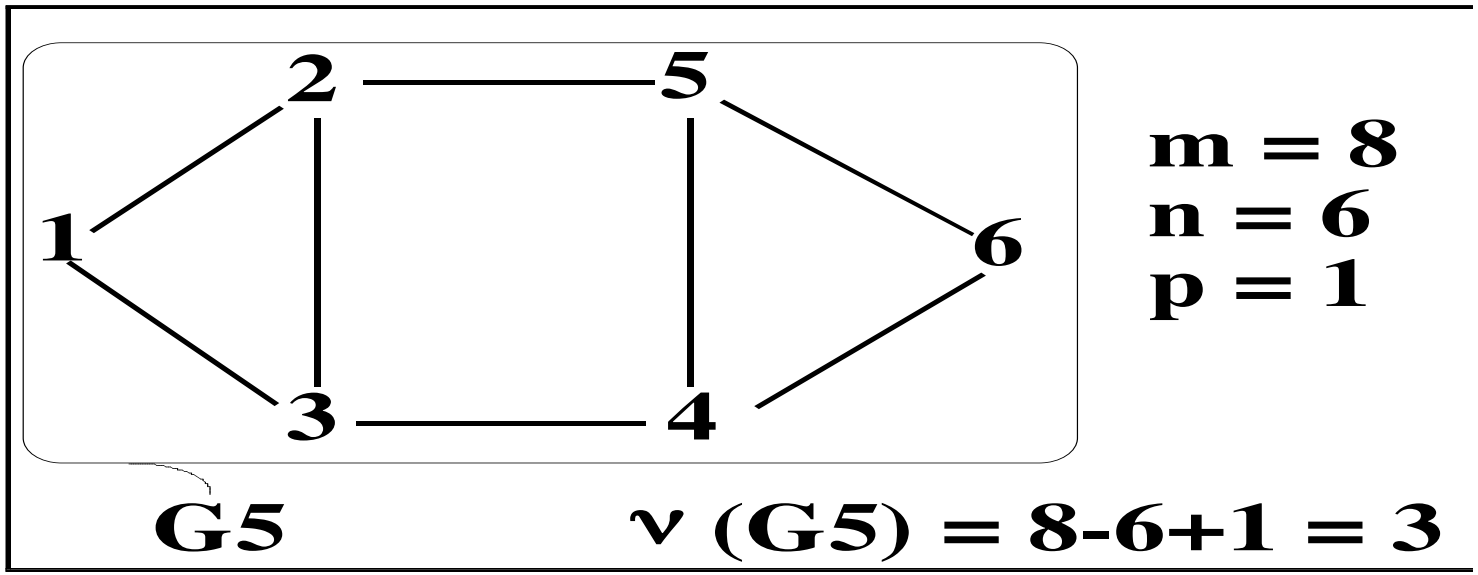
a: b
b: f
f: i
i: -
f: -
b: -
a: -

5.5 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 ⇒ $\nu(H)=0$, et H connexe

⇒ p=1 ⇒ m-n+1 = 0 ⇒ m=n-1

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

⇒ (n-1)-n+p=0 ⇒ p=1 ⇒ 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 ⇒

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

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

⇒ $\nu(H')=n-n+1=1$ ⇒ 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)

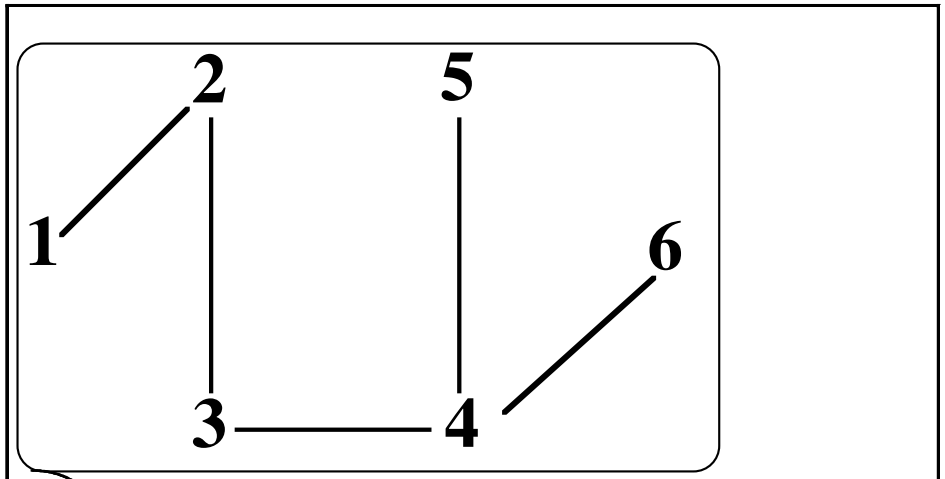
(5) H connexe et si on supprime une arête , il n'est plus connexe

(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 ≥ 2)

(1) H arbre avec une racine r

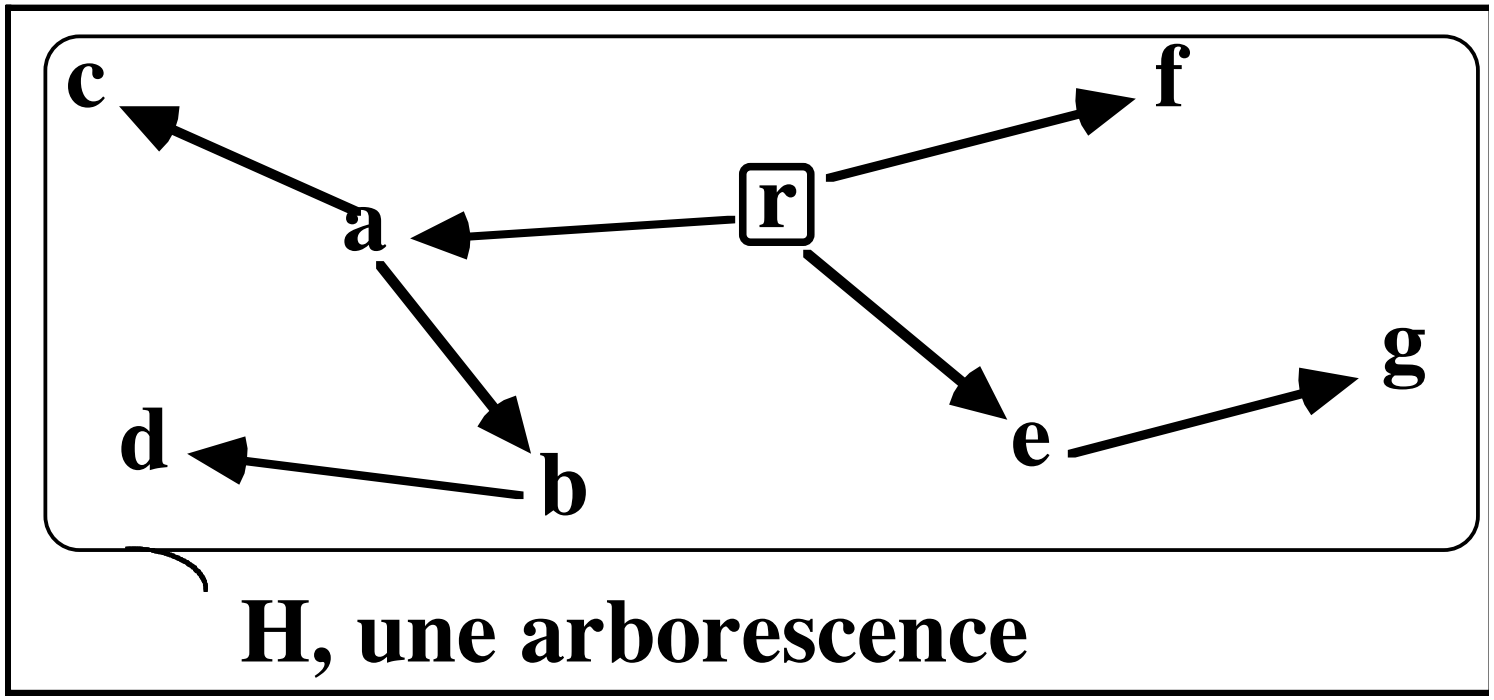
(2) ∃ r ∈ X, relié à tout x ∈ X par un chemin unique

(3) H connexe et

∃ r ∈ X t.q. $d^-(r)=0$ et

$d^-(x)=1$ pour tout x ≠ r

(4) **H sans cycle et ∃ un sommet r ∈X t. q. d^-(r)=0 et d^-(x)=1 pour tout x ≠ r**



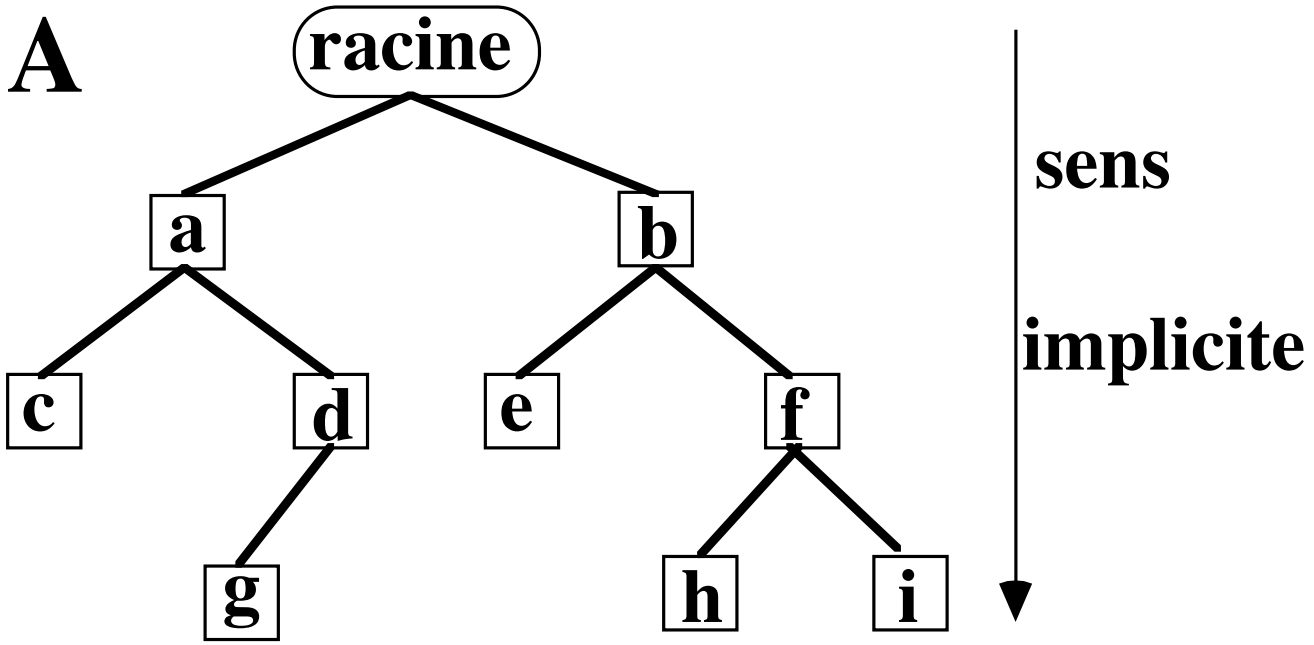
arborescence =

"arbre enraciné" (rooted tree)

= "arbre" en informatique

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

arborescence binaire:



(voir cours N° 3)