

INF 431



B. WERNER



Graphes I

3 mars 2010

La page des Projets est ouverte !

Fabrice LE FESSANT, responsable des projets

Ce n'est pas : premiers arrivés premiers servis !

À propos du Projet

Calendrier : Voir le web !

Conseils :

- bien **réfléchir** au choix de son binôme ;
- bien **réfléchir** avant de choisir le sujet ;
- **réfléchir** à la répartition du travail ; **réfléchir** avant de programmer ;
- **ne pas s'y prendre au dernier moment** ;
- écrire un **rapport clair et précis**, avec explication des choix de programmation, performances du programme, améliorations éventuelles, etc.
- Ne pas hésiter à contacter l'auteur du sujet **par email** en cas de problème **lié au sujet**.

Plan

- 0. À propos du projet.
- I. À quoi servent les graphes.
- II. Représentations des graphes.
- III. Le package `grapheX`.
- IV. Accessibilité.

I. À quoi servent les graphes ?

Jusqu'à présent :

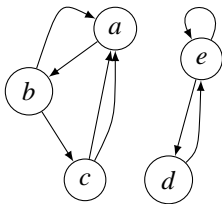
- **tableau** : informations sans corrélation ;
- **liste, arbre** : une adresse mémoire
- deux adresses mémoire : **files de priorité** ;
- que faire si les structures sont plus complexes ?

Quelques exemples :

- modélisation de réseaux de toutes sortes : plan du métro, train, guidage GPS, liaisons électriques, gazoducs, spatiales, INTERNET, le web ;
- représentations de liens logiques : écosystèmes ; graphe de dépendance entre fichiers source, diagramme d'héritage, etc. ;
- modélisation de l'état d'un système ;
- allocations de ressources, chemin de coûts minimaux, etc.

Abstraction !

Définition



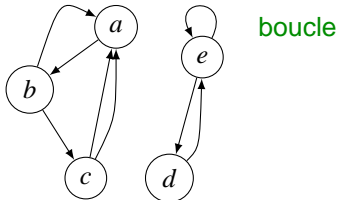
Un **graphe** $\mathcal{G} = (\mathcal{S}, \mathcal{A})$ est donné par un ensemble \mathcal{S} de **sommets** et un ensemble \mathcal{A} d'**arcs**, $\mathcal{A} \subset \mathcal{S} \times \mathcal{S}$.

Ex. $\mathcal{S} = \{a, b, c, d, e\}$, $\mathcal{A} = \{(a, b), (b, a), (b, c), (c, a), (d, e), (e, d)\}$.

Rem. La complexité des algorithmes sera fonction de $n = |\mathcal{S}|$ et souvent également de $m = |\mathcal{A}|$ (avec $m \leq n^2$).

Nous parlerons essentiellement de graphes finis

Terminologie



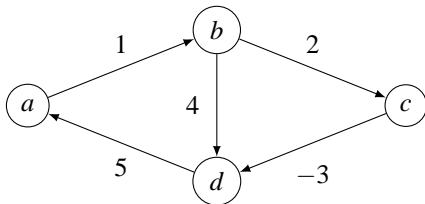
- $\alpha = (a, b)$ est **orienté** de a vers b ;
- α a pour **origine** a et **destination** b ;
- α est **incident** à a ainsi qu'à b ;
- a est un **prédécesseur** de b , et b un **successeur** de a ;
- a et b sont **adjacents**.

Graphe **simple** s'il ne comporte pas de boucles ou d'arcs multiples ;
multigraphe s'il comporte des arcs multiples (mais pas de boucles).

L'**arité** (ou **degré**) de $x \in \mathcal{S}$ est le nombre d'arcs partant ou arrivant en x .

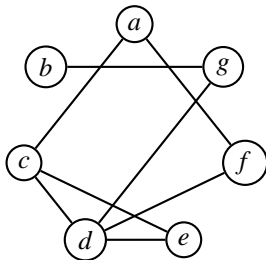
Graphe valué

Ex. Carte des distances, etc.

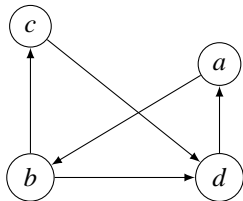
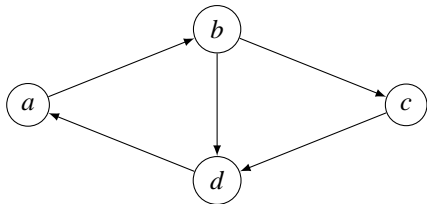


Graphe non orienté (ou symétrique)

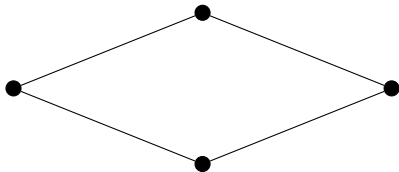
Cas particulier (fréquent) où $(i,j) \in \mathcal{A} \Leftrightarrow (j,i) \in \mathcal{A}$; on parle alors plutôt d'ensemble d'**arêtes**.



Graphes et dessins

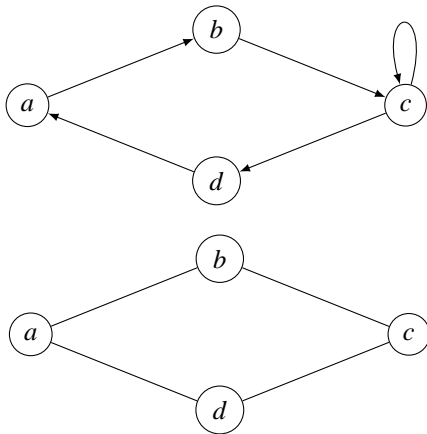


Graphe abstrait :



Graphe sous-jacent

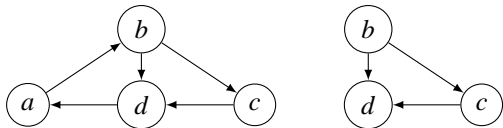
Graphe non-orienté obtenu en enlevant les flèches :



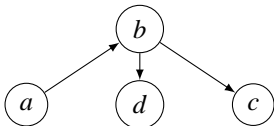
Sous-graphe

Si $\mathcal{G} = (\mathcal{S}, \mathcal{A})$, $\mathcal{S}' \subset \mathcal{S}$, on note $\mathcal{A}|\mathcal{S}'$ l'ensemble des arêtes de \mathcal{A} qui ont leurs deux extrémités dans \mathcal{S}' .

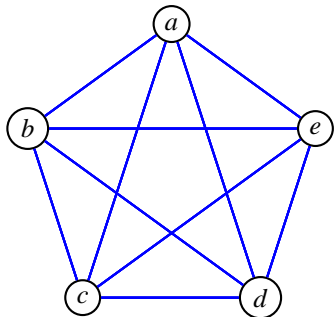
Le graphe $\mathcal{G}' = \mathcal{G}|\mathcal{S}' = (\mathcal{S}', \mathcal{A}|\mathcal{S}')$ est appelé le **graphe induit** de \mathcal{G} par \mathcal{S}' .



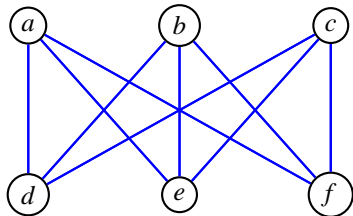
Tout graphe $(\mathcal{S}', \mathcal{A}')$ avec $\mathcal{S}' \subset \mathcal{S}$ et $\mathcal{A}' \subset \mathcal{A}$ est appelé **sous-graphe** de \mathcal{G} ; si $\mathcal{S}' = \mathcal{S}$, on parle de **graphe couvrant**.



Quelques graphes classiques



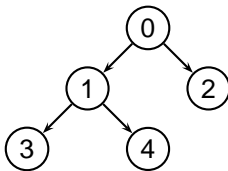
K_5
graphe complet



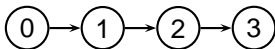
$K_{3,3}$
graphe complet biparti

Graphe/arbre/liste

Un arbre est un graphe particulier :



Une liste peut être vue comme un arbre filiforme, et donc un graphe :



Quelques problèmes fondamentaux

- Accessibilité, connexité, plus courts chemins.
- Planarité.
- Coloration (th. des 4 couleurs ; attribution des fréquences).
- Dessins : comment dessiner un graphe plan (circuit électronique) ?
- Visualisation : comment comprendre un graphe avec 10^6 sommets ?
- Optimisation combinatoire : voyageur de commerce, etc.

II. Représentations des graphes

Comment manipuler un graphe ?

- On doit d'abord modéliser le problème et trouver un codage adéquat. Cela peut se faire par acquisition, etc.
- **Attention** : il n'y a pas de représentation canonique (dessin) d'un graphe...
- Une fois cela fait, on peut passer au codage de l'information.

Quelles structures ?

Abstraction : $\mathcal{G} = (\mathcal{S}, \mathcal{A})$.

Codage de \mathcal{S} : ensemble de sommets, par exemple $[0..n - 1]$; en Java, n'importe quel objet de **Collection**.

Codage de \mathcal{A} :

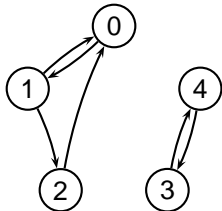
- **graphe simple** (pas de boucle ni arête multiple) : matrice, tableau de listes (ou autre structure garantissant l'unicité).
- **multigraphe** : tableau de listes (ou multi-ensembles).

Matrice d'adjacence

Pour \mathcal{G} simple, matrice $n \times n$ telle que :

$$M_{i,j} = \begin{cases} 1 & \text{si } (i,j) \in \mathcal{A}, \\ 0 & \text{sinon.} \end{cases}$$

Ex.



$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Graphe valué : $M_{i,j} = v(i,j)$.

Représentation par listes de successeurs

On associe à chaque sommet i la liste des sommets j tels que $(i,j) \in \mathcal{A}$.

Ex.

$$L[0] = (1)$$

$$L[1] = (0, 2)$$

$$L[2] = (0)$$

$$L[3] = (4)$$

$$L[4] = (3)$$

Comparaisons des deux représentations

On manipulera les deux représentations en fonction des contextes.
On pourra aussi abstraire une partie des propriétés.

type	mémoire	utilisation
matrice	$O(n^2)$	graphe dense
listes	$O(\mathcal{A} + n)$	graphe creux

- Listes vs. matrices : on gagne quand $|\mathcal{A}| \ll n^2$.
- Matrices mal adaptées aux arcs multiples ; très adaptées aux petits graphes.

III. Le package `grapheX`

Forcément un compromis

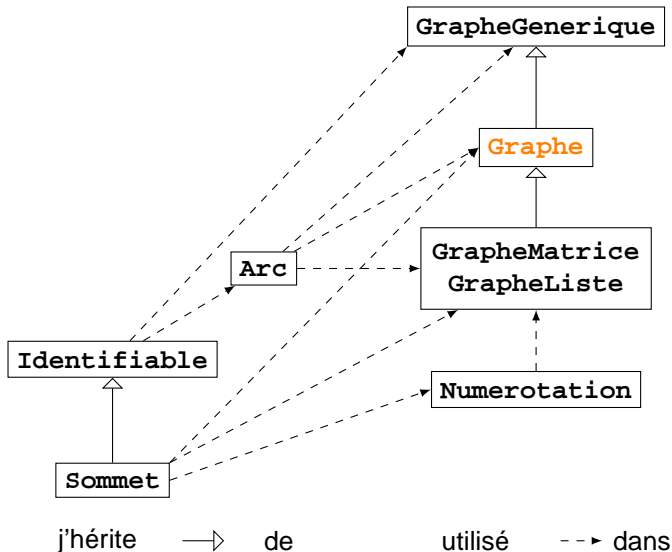
Remarque préliminaire : il est impossible d'écrire une classe de graphe qui soit universelle.

Il n'est pas dit que ce soit l'implantation la plus efficace.

Si les performances sont cruciales, il vaut peut-être mieux réécrire les structures adaptées à la main.

Toutefois, on utilisera souvent la classe abstraite pour les exemples.

Le schéma



La classe abstraite GrapheGenerique

```
public abstract class
  GrapheGenerique <S extends Identifiable>{

  public abstract int taille();

  public abstract void ajouterSommet(S s);

  public abstract Collection<S> sommets();

  public abstract boolean existeArc(S s, S t);

  public abstract void
    ajouterArc(S s, S t, int val);

  public abstract int valeurArc(S s, S t);

  public abstract void enleverArc(S s, S t);

  public abstract
    Collection<Arc<S>> voisins(S s);
}
```

Identifiable et Arc

```
public class Identifiable
    implements Comparable<Identifiable>{
    ...
    public Identifiable(String identifiant) {
        ident = identifiant;
    }
    ...
}
```

```
// L'arc o -> d avec valeur val
public class Arc<S extends Identifiable>
    implements Comparable<Arc<S>>{
    ...
    public Arc(S o0, S d0, int val0) {...}
    ...
}
```

Un exemple d'identifiable : Sommet

```
package grapheX;

public class Sommet extends Identifiable {

    public Sommet(String nn) {
        super(nn);
    }

    public String toString() {
        return identifiant();
    }

}
```

La classe abstraite Graphe

Type de base : graphe orienté.

```
public abstract class Graphe
  extends GrapheGenerique <Sommet>{

  public abstract int taille();
  public abstract Graphe copie();

  public abstract void ajouterSommet(Sommet s);
  public abstract boolean
    existeArc(Sommet s, Sommet t);
  public abstract void
    ajouterArc(Sommet s, Sommet t, int val);
  public abstract int
    valeurArc(Sommet s, Sommet t);
  public abstract void
    enleverArc(Sommet s, Sommet t);

  public abstract Collection<Sommet> sommets();
  public abstract
    Collection<Arc<Sommet>> voisins(Sommet s);

  public boolean oriente;
```

Première classe concrète : GrapheMatrice

Implantation avec des "matrices".

```
package grapheX;
```

```
public class GrapheMatrice extends Graphe{
    private Vector<Vector<Arc<Sommet>>> M;
    private Numerotation numerotation; // pratique...!

    public int taille(){
        return M.size();
    }

    public GrapheMatrice(int n){
        numerotation = new Numerotation(n);
        M = new Vector<Vector<Arc<Sommet>>>(n);
        M.setSize(n);
    }
    ...
}
```

Seconde classe concrète : GrapheListe

```
package grapheX;

public class GrapheListe extends Graphe{
    private Vector<LinkedList<Arc<Sommet>>> L;
    private Numerotation numerotation;

    public int taille(){
        return L.size();
    }
    ...
}
```

Un programme de test

```
import java.io.*;
import java.util.*;
import grapheX.*;

class Test1{
    public static void main(String[] args){
        Sommet a = new Sommet("a");
        Sommet b = new Sommet("b");
        Sommet c = new Sommet("c");
        GrapheListe G = new GrapheListe(3);
        G.ajouterArc(a, b); // a -> b
        G.ajouterArc(b, c); // b -> c
        G.ajouterArc(c, a); // c -> a
        // affichage des sommets
        for(Sommet s : G.sommets())
            System.out.println(s);
        // affichage des voisins de a
        for(Arc<Sommet> alpha : G.voisins(a))
            System.out.println(alpha);
    } }
```

Lecture à partir d'un fichier

```
#prim1
4 vs
u v w x
u 2 v 16 x 29
v 3 u 16 x 20 w 25
w 2 v 25 x 10
x 3 u 29 v 20 w 10
```

```
public
static GrapheListe deFichier(String nomfic){
    try{
        Scanner scan =
            new Scanner(
                new BufferedReader(new FileReader(nomfic)));
        GrapheListe G = new GrapheListe();
        remplirAvecScanner(G, scan);
        return G;
    }
    catch(Exception e) { }
    return null;
}
```

```

public static void
remplirAvecScanner(GrapheListe G, Scanner scan){
    System.out.println(scan.next());
    int n = scan.nextInt();
    remplir(G, n);
    String str = scan.next();
    boolean estValue = option(str, 'v');
    boolean estSym = option(str, 's');
    boolean avecCouples = option(str, 'c');

    if(estSym)
        G.orienté = false;
    System.out.println("n = "+n);
    for(int i = 0; i < n; i++){
        Sommet s = new Sommet(scan.next());
        G.ajouterSommet(s);
    }
}

```

```

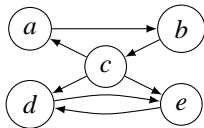
if(avecCouples){
    ...
}
else{
    // format "s ns t0 t1 ... t{ns-1}"
    // ou      "s ns t0 v0 t1 v1 ... t{ns-1} v{ns-1}"
    System.out.println("Avec listes, estvalue="+estValue);
    for(int r = 0; r < n; r++){
        Sommet s = new Sommet(scan.next());
        int si = G.numerotation.numero(s);
        int nj = (int)scan.nextInt();
        for(int k = 0; k < nj; k++){
            Sommet t = new Sommet(scan.next());
            int ti = G.numerotation.numero(t);
            if(estValue)
                G.ajouterArc(si, ti,
                    (int)scan.nextInt());
            else
                G.ajouterArc(si, ti, 1);
        }
    }
}
...
}

```

IV. Accessibilité

Un **chemin** (s_0, s_1, \dots, s_p) dans \mathcal{G} est une suite finie non vide de sommets telle que $(s_k, s_{k+1}) \in \mathcal{A}$. La **longueur** du chemin est p ($=$ nombre d'arcs empruntés).

Ex. (c, a, b, c, e) est un chemin dans :



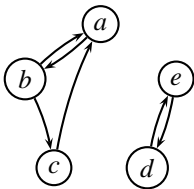
Un chemin est

- **simple** si les arcs (s_{i-1}, s_i) pour $i = 1, \dots, p$ sont deux à deux \neq .
- **élémentaire** si les sommets sont deux à deux distincts.
- un **circuit** si $p \geq 1$ et $s_p = s_0$; il est **élémentaire** si (s_0, \dots, s_{p-1}) est élémentaire.

Rem. Graphe non orienté : chemin \rightarrow chaîne (arêtes distinctes 2 à 2); circuit \rightarrow cycle.

A) Propriétés fondamentales

Lemme. L'ensemble des chemins d'un graphe est infini ssi le graphe possède des circuits.



Déf. c' est un **chemin extrait** de c si c' est chemin, c et c' ont même origine et extrêmité et la suite des arcs de c' est une sous-suite de la suite des arcs de c .

Ex. (a, b, c) est un chemin extrait de (a, b, c, a, b, c) .

Lemme de König

Lemme. (König) De tout chemin, on peut extraire un chemin élémentaire.

Dém. Récurrence sur la longueur p d'un chemin :

- Si $p = 0$, le chemin est élémentaire.
- Soit $c = (s_0, s_1, \dots, s_p)$ un chemin de longueur $p > 0$.

S'il n'est pas élémentaire, il existe deux sommets égaux $s_i = s_j$ avec $0 \leq i < j \leq p$.

$c' = (s_0, \dots, s_i, s_{j+1}, \dots, s_p)$ est strictement plus petit que c et on peut en extraire un chemin élémentaire par hypothèse de récurrence, qui sera lui-même extrait de c . \square

Premières propriétés de connexité

Déf. Soit \mathcal{G} un graphe non orienté. Il est **connexe** ssi deux sommets sont toujours joignables par un chemin.

Lemme 1. Un graphe connexe à n sommets possède au moins $n - 1$ arêtes.

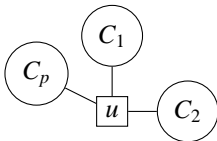
Lemme 2. Un graphe à n sommets ayant au moins n arêtes possède un cycle.

Dém. du Lemme 1 (récurrence sur n)

- vraie pour $n = 1$.

• Supposons $n \geq 2$.

Soit $u \in \mathcal{S}$ et considérons le graphe \mathcal{G}_u induit de \mathcal{G} par $\mathcal{S} - \{u\}$, C_1 , C_2, \dots, C_p ses composantes connexes :



Pour tout $k \in \{1, \dots, p\}$, il existe au moins une arête liant u à un sommet de C_k (sinon \mathcal{G} ne serait pas connexe).

Soit $\mathcal{G}^{(k)}$ le sous-graphe induit par C_k . Par hypothèse de récurrence : $m_k \geq n_k - 1$.

$$m \geq p + \left(\sum_{k=1}^p m_k \right) \geq \sum_{k=1}^p n_k = n - 1. \square$$

Caractérisation des arbres

Prop. soit \mathcal{G} un graphe simple à n sommets. Les propriétés suivantes sont équivalentes et caractérisent le fait que \mathcal{G} est un arbre.

- (i) \mathcal{G} est un graphe connexe sans cycle ;
- (ii) \mathcal{G} est un graphe connexe ayant $n - 1$ arêtes ;
- (iii) \mathcal{G} est un graphe sans cycle possédant $n - 1$ arêtes ;
- (iv) \mathcal{G} est un graphe dans lequel deux sommets quelconques sont liés par une seule chaîne ;
- (v) \mathcal{G} est un graphe connexe dont la connexité disparaît dès qu'on enlève une arête quelconque ;
- (vi) \mathcal{G} est un graphe sans cycle tel que l'ajout d'une arête quelconque crée un cycle et un seul.

B) Fermeture transitive

Pb. existe-il un chemin (de longueur ≥ 0) entre deux sommets donnés de \mathcal{G} ?

Déf. On appelle **fermeture transitive** du graphe \mathcal{G} le graphe $\mathcal{G}^* = (\mathcal{S}, \mathcal{A}^*)$ avec $(s, t) \in \mathcal{A}^*$ ssi il existe un chemin entre s et t dans \mathcal{G} .

Deux solutions : matrice d'adjacence ; parcours.

Pour simplifier : $\mathcal{S} = \{0, 1, 2, \dots, n - 1\}$.

Premier algorithme de calcul

Première idée : on construit une suite de graphes $(\mathcal{G}^{(r)})_{1 \leq r \leq n}$,

- $\mathcal{G}^{(1)}$ est le graphe initial ;
- pour $r \geq 2$, $\mathcal{G}^{(r)} = (\mathcal{S}, \mathcal{A}^{(r)}, \mathcal{M}^{(r)})$ est défini par $(i, j) \in \mathcal{A}^{(r)}$ ssi il existe un chemin de i à j de longueur exactement r .

Un chemin de longueur $r > 1$ entre i et j s'écrit (i, k, \dots, j) avec k voisin de i et (k, \dots, j) un chemin de longueur $r - 1$.

En booléens :

$$\mathcal{M}_{i,j}^{(r)} = \bigvee_{k=0}^{n-1} \mathcal{M}_{i,k}^{(1)} \wedge \mathcal{M}_{k,j}^{(r-1)}.$$

\leftrightarrow produit des matrices $\mathcal{M}^{(1)}$ et $\mathcal{M}^{(r-1)}$ où on a remplacé l'addition par le OU logique, et la multiplication par le ET logique.

	0	1
0	0	1
1	1	1

OU logique \vee

	0	1
0	0	0
1	0	1

ET logique \wedge

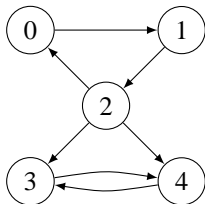
Plus généralement, si $(i, j) \in \mathcal{G}^*$, c'est qu'il existe un chemin de longueur $< n$ entre les deux (par le lemme de König).

D'où :

$$\mathcal{M}^* = I_n + \mathcal{M}^{(1)} + \mathcal{M}^{(2)} + \dots + \mathcal{M}^{(n-1)}.$$

La matrice identité d'ordre n , notée I_n , a été ajoutée pour tenir compte de l'accessibilité de tout sommet avec lui-même.

Exemple



$$\mathcal{M} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}.$$

$$\mathcal{M}^{(2)} = \begin{pmatrix} & & & 1 & \\ 1 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & 1 & \\ & & & & 1 \end{pmatrix}, \quad \mathcal{M}^{(3)} = \begin{pmatrix} 1 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & 1 & \\ & & & & 1 \end{pmatrix},$$

$$\mathcal{M}^{(4)} = \begin{pmatrix} & & & 1 & \\ & & & & 1 \\ 1 & & & & \\ & & 1 & & \\ & & & 1 & \\ & & & & 1 \end{pmatrix}, \quad \mathcal{M}^* = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ & & & 1 & 1 \\ & & & 1 & 1 \end{pmatrix}.$$

Thm. Le produit de deux matrices se fait en temps $O(n^\omega)$ (où ω est un exposant compris entre 2 et 3).

Coro. $O(n^{1+\omega})$ opérations, soit généralement $O(n^4)$ en pratique.

Prop. En booléens :

$$\mathcal{M}^* = (I + \mathcal{M})^{n-1}$$

$\Rightarrow O((\log n)n^\omega)$ (exponentiation binaire).

Prop. Le nombre de chemins de longueur ℓ entre i et j est $\mathcal{M}_{i,j}^{(\ell)}$, \mathcal{M} étant considérée comme une matrice entière.

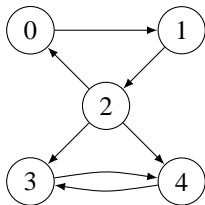
L'algorithme de Roy et Warshall

Idée : construire de proche en proche des chemins (élémentaires) de plus en plus longs.

On construit $\mathcal{G}^{(r)} = (\mathcal{S}, \mathcal{A}^{(r)})$ pour $r \geq -1$:

- $(i, j) \in \mathcal{A}^{(-1)}$ ssi \exists chemin $i \rightarrow j$ ne passant par aucun sommet intermédiaire.
- $(i, j) \in \mathcal{A}^{(0)}$ ssi \exists chemin $i \rightarrow j$ passant **au plus** par le sommet 0.

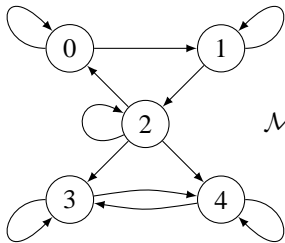
Ex.



$$\mathcal{M} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}.$$

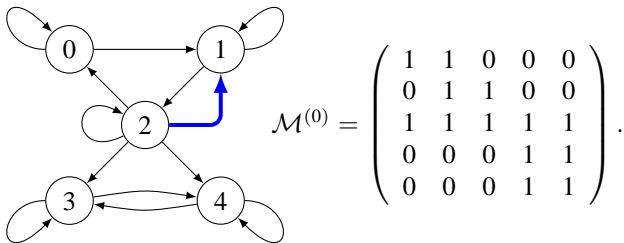
Calcul de $\mathcal{G}^{(-1)}$

On doit rajouter les boucles.

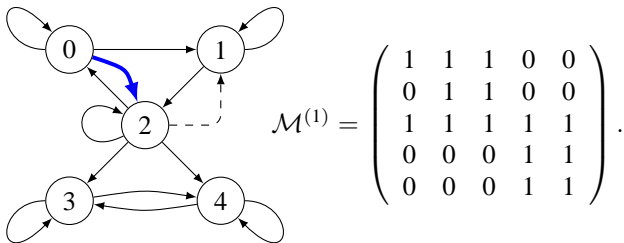


$$\mathcal{M}^{(-1)} = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix}.$$

Calcul de $\mathcal{G}^{(0)}$: on rajoute les chemins $(i, 0, j)$, ici $\{(2, 0, 1)\}$.

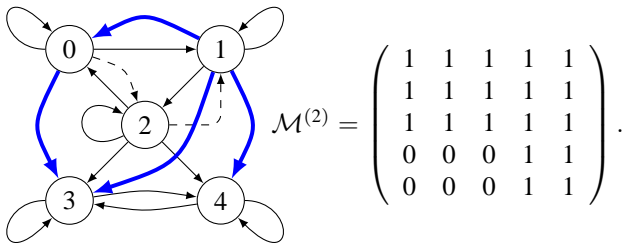


Calcul de $\mathcal{G}^{(1)}$: tous les chemins passant dans $\{0, 1\}$, i.e., $\{(0, 1, 2), (2, 0, 1, 2)\}$.



Calcul de $\mathcal{G}^{(2)}$: on rajoute les chemins

$\{(0, 1, 2, 0), (0, 1, 2, 3), (0, 1, 2, 4), (1, 2, 0), (1, 2, 0, 1), (1, 2, 3), (1, 2, 4)\}$



Et c'est tout.

Algorithme

$(i, j) \in \mathcal{A}^{(r)}$ ssi \exists chemin $i \rightarrow j$ ne passant par aucun sommet intermédiaire d'indice $> r$.

$\iff (i, j) \in \mathcal{A}^{(r-1)}$ ou $((i, r) \in \mathcal{A}^{(r-1)} \text{ et } (r, j) \in \mathcal{A}^{(r-1)})$. En booléens :

$$\mathcal{M}_{i,j}^{(r)} = \mathcal{M}_{i,j}^{(r-1)} + \mathcal{M}_{i,r}^{(r-1)} \mathcal{M}_{r,j}^{(r-1)}.$$

RoyWarshall(M)

// M est la matrice d'adjacence booléenne n x n
// d'un graphe G

1. `N <- copie(M); P <- matrice_identite(n, n);`
2. **pour** `r <- 0 à n-1 faire`
 pour `i <- 0 à n-1 faire`
 pour `j <- 0 à n-1 faire`
 `P[i][j] <- N[i][j] OU (N[i][r] ET N[r][j]);`
 `N := P; // copie de P dans N`
3. **retourner** `N; // matrice d'adjacence de G *`

Prop. La complexité de cet algorithme est en $O(n^3)$.

Amélioration

Prop. la matrice \mathbf{P} est inutile et on peut effectuer les calculs *en place*. (cf. poly pour la preuve)

RoyWarshall(M)

// M est la matrice d'adjacence booléenne n x n

// d'un graphe G

1. N <- copie(M);

 pour i <- 0 à n-1 faire N[i][i] <- true;

2. pour r <- 0 à n-1 faire

 pour i <- 0 à n-1 faire

 pour j <- 0 à n-1 faire

 N[i][j] <- N[i][j] OU (N[i][r] ET N[r][j]);

3. retourner N; // matrice d'adjacence de G *

Le code en Java

```
static Graphe RoyWarshall(Graphe G){
    Graphe F = G.copie();

    for(Sommet s : G.sommets())
        F.ajouterArc(s, s);
    for(Sommet r : G.sommets()){
        for(Sommet s : G.sommets())
            for(Sommet t : G.sommets())
                if(!F.existeArc(s, t)
                    && (F.existeArc(s, r)
                        && F.existeArc(r, t)))
                    F.ajouterArc(s, t);
    }
    return F;
}
```

Résumé du cours

- Introduction aux graphes.
- Classe abstraite.
- Accessibilité (Roy-Warshall, Floyd).

Prochains rendez-vous : TD cet après-midi ; amphi 06 mercredi prochain 10/03.

Un occasion d'en savoir plus : François BACCELLI (INRIA, ENS, Académie des Sciences) : [Routage Espace-Temps](#) (avec des graphes aléatoires).

Jeudi 4 Mars, 17h30, CINE, Salle de Réunion du LIX (Aîle 0)