

1. Introduction

1.1 Définitions

1.2 Représentation des graphes

2. Parcours de graphes

2.1 Parcours en largeur

2.2 Parcours en profondeur

3. Fermeture transitive des graphes

3.1 Algorithme de WARSHALL

4. Recherche du plus court chemin

4.1 Algorithme de FORD

Quelques définitions

Définition (Successeurs)

$\mathcal{P}(X)$ ensemble des parties de X

$\Gamma : X \rightarrow \mathcal{P}(X)$

$x \mapsto \Gamma(x) = \{y \in X; (x, y) \in U\}$

Associe à chaque sommet l'ensemble de ses successeurs. Un graphe peut être noté (X, Γ) .

Exemple

$\Gamma(1) = \{2\}$, $\Gamma(2) = \{1\}$, $\Gamma(3) = \{2, 3, 4\}$, $\Gamma(4) = \{2, 3\}$, $\Gamma(5) = \{2\}$.

Quelques définitions

Définition (Graphe)

Soit X un ensemble fini.

Un graphe est un couple (X, U) où X est un ensemble et U est l'ensemble de couples (x, y) de X .

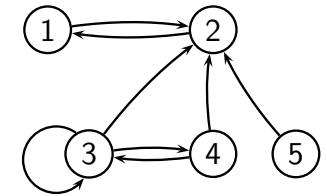
- Les éléments de X : sommets du graphe
- Les éléments de U : arcs du graphe

$NS = \text{card}(X)$ nombre de sommets, $NA = \text{card}(U)$ nombre d'arcs.

Exemple

$X = \{1, 2, 3, 4, 5\}$

$U = \{(1, 2), (2, 1), (3, 2), (4, 2), (5, 2), (3, 3), (3, 4), (4, 3)\}$



Orienté

Définition (Orienté/Non orienté)

Les arcs peuvent avoir un sens ou non, on dit alors que les graphes sont **orientés** ou **non orientés**

Dans un graphe orienté, soit un arc (i, j) :

- i : origine
- j : extrémité

Dans un graphe non orienté on ne distingue pas l'origine de l'extrémité d'un arc. On parle **d'arête**.

Dans la suite du cours les graphes sont orientés.

Graphe partiel

On peut supprimer des arcs d'un graphes

Définition (Graphe partiel)

$G_1 = [X_1, U_1]$ est un *graphe partiel* de $G = [X, U]$ ssi

$$X_1 = X$$

$$U_1 \subset U.$$

Exemple



On a supprimé les arcs $(5, 2)$, $(4, 2)$, $(3, 4)$, $(4, 3)$ et $(3, 3)$ **mais** on a conservé les sommets 4 et 5.

Exemple de sous graphe induit

Exemple Si on supprime le sommet 2 il faut au moins supprimer les arcs $(1, 2)$, $(2, 1)$, $(3, 2)$, $(4, 2)$ et $(5, 2)$.



Sous-graphe

On peut aussi supprimer les sommets

Définition (Sous graphe)

$G_2 = [X_2, U_2]$ est un **sous-graphe** de $G = [X, U]$ ssi

- $X_2 \subset X$ on enlève des sommets ;
- $U_2 \subset U \cap (X_2 \times X_2)$ on enlève des arrêtes dont toutes les arrêtes impossibles.

Définition (Sous graphe induit)

On dit que le sous graphe est **induit** lorsque seules les arrêtes impossibles sont supprimées

$G_2 = [X_2, U_2]$ est un *sous-graphe induit* de $G = [X, U]$ ssi

- $X_2 \subset X$;
- $U_2 = U \cap (X_2 \times X_2)$.

Chemin

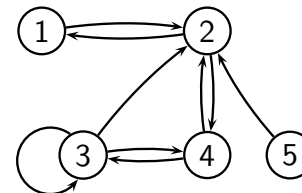
Définition (Chemin)

Un *chemin* est une suite finie de sommets $(x_1, \dots, x_p)$ et telle que $\forall i \in \{1, 2, \dots, p\} : (x_i, x_{i+1}) \in U$.

Dans un chemin,

- x_1 est le sommet de départ,
- x_p est le sommet d'arrivée.

Exemple

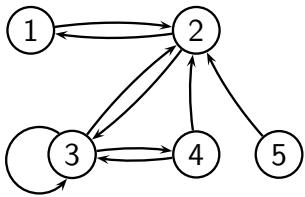


- $(1, 2, 4)$ est un chemin de 1 vers 4 ;
- $(2, 4, 5)$ n'est pas un chemin car $(4, 5)$ n'existe pas ;
- $(3, 4, 2)$ est un chemin de 3 vers 2 ;
- $(3, 4, 2, 1, 2)$ est aussi un chemin de 3 vers 2.

Chemin (suite)

Définition

- Un circuit est un chemin pour lequel le sommet de départ est égal au sommet d'arrivée.
- Un chemin est simple s'il passe une seule fois par les arcs qui le composent.
- Un chemin est élémentaire s'il passe une seule fois par les sommets qui les composent.



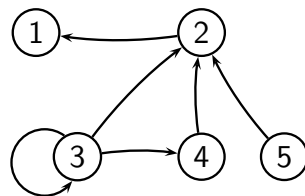
- $(1, 2, 1)$ et $(3, 4, 3, 3)$ sont des circuits ;
- $(1, 2, 1, 2, 3)$ est un chemin non simple (composé) ;
- $(1, 2, 3)$ chemin simple et élémentaire ;
- $(1, 2, 1, 2, 3)$ n'est pas élémentaire ;
- $(1, 2, 1)$ est un circuit non élémentaire.

Définition (anti-symétrie)

Un graphe est anti-symétrique si pour tout couple (x, y) , $(x, y) \in U$ et $(y, x) \in U \Rightarrow x = y$. C'est à dire lorsque $x \neq y$:

- ou bien $(x, y) \notin U$
- ou bien $(y, x) \notin U$

Exemple



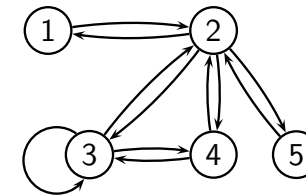
Symétrie

Définition (Symétrie)

Un graphe $[X, U]$ est symétrique si pour tout couple (x, y) de sommets, $(x, y) \in U \Rightarrow (y, x) \in U$. C'est à dire :

- ou bien $(x, y) \notin U$
- ou bien $(x, y) \in U$ et $(y, x) \in U$

Exemple



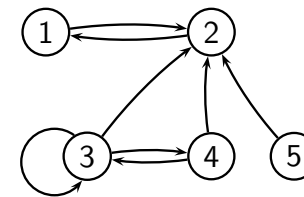
Complétude

Définition

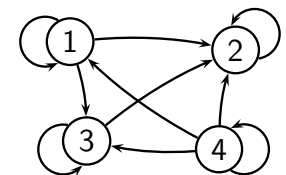
Un graphe est complet si pour tout couple de sommets (x, y) soit $(x, y) \in U$ soit $(y, x) \in U$.

Exemple

Un graphe non complet :



Un graphe complet :



Le graphe n'est pas complet car $(4, 5) \notin U$ et $(5, 4) \notin U$.

1. Introduction

1.1 Définitions

1.2 Représentation des graphes

2. Parcours de graphes

2.1 Parcours en largeur

2.2 Parcours en profondeur

3. Fermeture transitive des graphes

3.1 Algorithme de WARSHALL

4. Recherche du plus court chemin

4.1 Algorithme de FORD

Introduction

Le choix de la représentation d'un graphe

- est importante car elle agit sur la complexité ;
- dépend des opérations qui seront effectuées sur le graphe :
 - il n'y a pas de meilleur représentation ;
 - il n'y a pas de représentation passe partout.

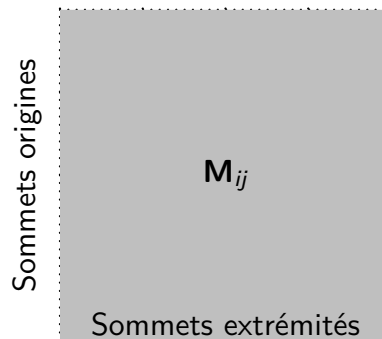
Il faut donc se demander :

- Le nombre de sommets est-il important ?
- Devra-t-on chercher souvent les successeurs d'un sommet ?
- Devra-t-on souvent chercher les prédécesseurs d'un sommet ?
- Devra-t-on souvent ajouter ou supprimer des arcs ?
- Devra-t-on souvent ajouter ou supprimer des sommets ?

La **représentation graphique** n'est utilisable qu'en cours pour tester les exemples.

Représentation matricielle

On représente le graphe par une matrice d'incidence sommet-sommet.

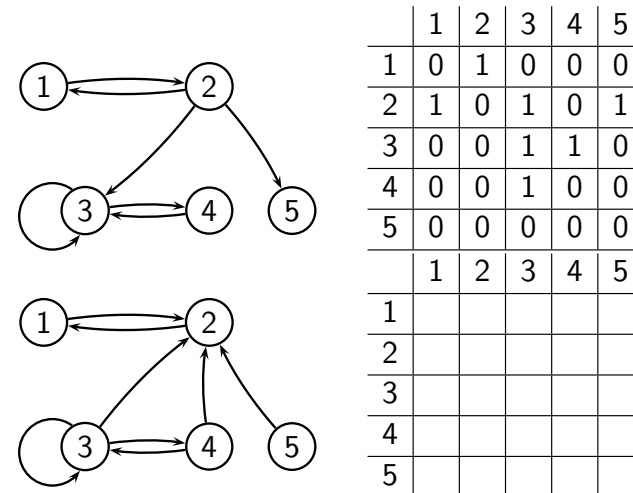


On utilise une matrice de dimension (NS, NS) où NS est le nombre de sommets $M = (M_{ij})_{1 \leq i, j \leq NS}$.

$$M_{i,j} = \begin{cases} 0 & \text{si } (i, j) \notin U \\ 1 & \text{si } (i, j) \in U \end{cases}$$

On peut aussi utiliser une matrice de booléens.

Exemple



Performance de la représentation matricielle

En temps

- Si on cherche les successeurs d'un sommet i on doit parcourir la ligne du sommet i et chercher les cases contenant un 1
 $\Rightarrow O(NS)$.
- Si on cherche les prédécesseurs d'un sommet j on doit parcourir la colonne du sommet j et chercher les cases contenant un 1
 $\Rightarrow O(NS)$.
- Si on cherche l'existence d'un arc (i, j) On se place directement en M_{ij} . $\Rightarrow O(1)$: une constante, ne dépend pas de la taille du tableau.

Performances du dictionnaire

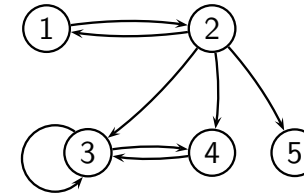
Temps

- On a directement accès aux successeurs d'un sommet i : si le tableau s'appelle t , $t[i]$ est la liste des successeurs du i^e sommet.
 $\Rightarrow O(1)$.
- Si on cherche les prédécesseurs d'un sommet j , on doit utiliser le dictionnaire du graphe inverse ou parcourir tous les arcs.
 $\Rightarrow O(NA)$ (NA est le nombre d'arêtes).
- Si on veut savoir si un arc (i, j) existe on doit parcourir le tableau $t[j]$. Dans le pire des cas tous les sommets sont successeurs de i
 $\Rightarrow O(NS)$.

Représentation sous forme d'un dictionnaire

On utilise un tableau : la i^e ligne contient les successeurs du i^e sommet.

Exemple



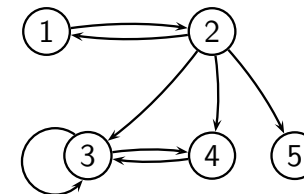
x	$\Gamma(x)$
1	2
2	1 3 4 5
3	3 4
4	3
5	

Représentation par vecteurs d'extrémités

On utilise deux tableaux un pour les origines et l'autre pour les extrémités. C'est à dire qu'on stocke une liste des arcs.

On utilise 2 tableaux de taille NA le nombre d'arc.

Exemple



<i>orig</i>	2	2	1	2	2	4	3	3
<i>extr</i>	1	3	2	4	5	3	4	3

Performances des vecteurs d'extrémités

En temps :

- Si on veut connaître les successeurs d'un sommet i , il faut parcourir le tableau origines et récupérer les extrémités correspondantes.
 $\Rightarrow O(NA)$
- Si on veut connaître les prédécesseurs d'un sommet i , il faut parcourir le tableau extrémités et récupérer les origines correspondantes.
 $\Rightarrow O(NA)$
- Pour savoir si un arc (i, j) existe, il faut parcourir le tableau origine et récupérer l'extrémité correspondante.
 $\Rightarrow O(NA)$

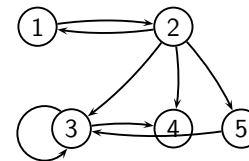
Représentation par origine

Pour améliorer la version précédente, il est possible d'ordonner l'une des listes par exemple celle des origines. Ainsi, il devient possible de maintenir un tableau qui signale le début et la fin de la liste des successeurs de chaque origine.

Il faut 3 tableaux :

- un tableau (TOrig) de taille NA des origines de chaque arc (classé par origine croissante);
- un tableau (TExr) de taille NA des extrémités de chaque arc (correspondant au précédent);
- Un tableau (TDeb) de taille $NS + 1$ qui signale le début et la fin de chaque série dans le premier.

Exemple



Sommets	1	2	3	4	5	F	
TDeb	1	2	6	8	8	9	
TOrig	1	2	2	2	3	3	5
TExr	2	1	3	4	5	3	4

Tous les arcs ayant pour sommets origine i sont entre les cases $TDeb[i]$ et $TDeb[i + 1]$.

Performance de la représentation par origine

En temps :

- On a directement accès aux successeurs d'un sommet i .
 $\Rightarrow O(1)$.
- Si on cherche les prédécesseurs d'un sommet j , on doit parcourir tous les arcs.
 $\Rightarrow O(NA)$.
- Si on veut savoir si un arc (i, j) existe on doit parcourir le tableau $t[i]$. Dans le pire des cas tous les sommets sont successeurs de i
 $\Rightarrow O(NS)$.

1. Introduction

1.1 Définitions

1.2 Représentation des graphes

2. Parcours de graphes

2.1 Parcours en largeur

2.2 Parcours en profondeur

3. Fermeture transitive des graphes

3.1 Algorithme de WARSHALL

4. Recherche du plus court chemin

4.1 Algorithme de FORD

Parcours de graphes

Soit un graphe $G = [X, U]$

on veut parcourir tous les sommets du graphe pour leur appliquer un traitement.

- **Exploration en profondeur :**
Cette méthode consiste à suivre un chemin le plus loin possible et à revenir en arrière pour explorer les sommets précédemment ignorés.
- **Exploration en largeur :**
On explore en priorité les successeurs d'un sommet

Si le graphe contient des cycles, que peut-il se passer ?

Marquage des sommets explorés

Tableau de marque

Pour se souvenir des sommets déjà visités, il est possible d'utiliser un tableau de booléens marque :

$$marque[i] = \begin{cases} \text{true} & \text{si on a déjà visité le sommet} \\ \text{false} & \text{si on n'a pas encore visité le sommet} \end{cases}$$

En python une liste sera utilisée.

Méthode

Quelle que soit la façon de parcourir le graphe il faut 2 structures de données :

- Une structure permettant de connaître les sommets déjà explorés :
 - pour ne pas visiter 2 fois un sommet (p. ex si on compte les sommets) ;
 - pour faire des opérations différentes (p. ex si on cherche un plus court chemin).
- Une structure stockant les sommets à explorer :
 - à chaque fois qu'on passe par un sommet il faut ajouter ses successeurs dans cette structure
 - il est important de conserver l'ordre voulu.

Stockage des sommets à explorer

Sommets à explorer

Pour stocker les sommets à explorer, il faut une structure dépendant de l'ordre d'exploration à visiter :

- lorsqu'on explore en profondeur, le prochain sommet doit être l'un des successeurs du sommet courant **donc l'un de ceux qu'on vient d'ajouter** ;
- lorsqu'on explore en largeur, le prochain sommet doit être à même distance du départ que le sommet courant **donc c'est le premier sommet ajouté parmi ceux qu'on n'a pas encore visité**.

Il faut de plus les méthodes suivantes :

- ajouter : pour ajouter un sommet à la structure ;
- retirer : pour retirer le prochain sommet à explorer ;
- est_vide : pour vérifier si la structure est vide.

Implémentation en Python

En Python, on peut utiliser les structures suivantes :

- **Liste (list)** : Pour une exploration en profondeur, on peut utiliser une liste comme une pile (stack) en ajoutant et retirant les éléments à la fin de la liste avec `append()` et `pop()`.
- **File double (deque)** : Pour une exploration en largeur, on peut utiliser une file (queue) en ajoutant les éléments à la fin avec `append()` et en retirant les éléments du début avec `popleft()`.

Algorithme

Données : deb : le point de départ,
marque : le tableau des marques

début

```

avisiter ← initAVisiter()
avisiter.ajoute(deb)
tant que avisiter.estVide() = false faire
    s = avisiter.prochain()           // prochain sommet à visiter
    avisiter.enleveProchain()         // on le retire de la liste
    si marque.estMarque(s) alors
        // on est déjà passé là
        ... on fait les opérations prévues
    sinon
        // on n'est jamais passé là
        ... on fait les opérations prévues
        marque.marquer(s)             // il faut marquer s
        pour chaque t successeur de s faire
            avisiter.ajoute(t)         // il faut visiter ses successeurs

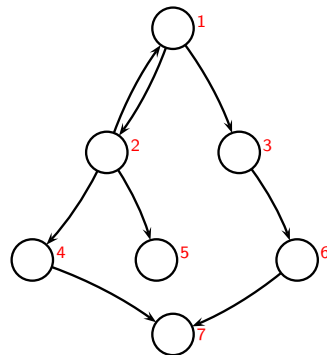
```

Algorithme 1 : ParcoursGeneral(Sommet deb, TableauBooleen marque)

Parcours en largeur

Pour le parcours en largeur, il faut visiter

- les successeurs du point de départ les uns après les autres;
- ensuite les successeurs de ces points;
- ...



⇒ Il faut visiter les points dans l'ordre dans lequel on les a ajoutés à la liste des points à visiter.

⇒ On utilise une file (premier arrivé premier servi)

Parcours de l'ensemble du graphe

Cet algorithme ne permet de visiter que les noeuds accessibles depuis le point de départ. Pour parcourir le graphe il faut appeler ParcoursGeneral tant qu'il reste au moins un sommet s non marqué.

Données : $G = [X, U]$: le graphe

début

```

marque ← initMarques()
pour s ∈ X faire
    si marque.estMarque(s) = FAUX alors
        ParcoursGeneral(s, marque)

```

Algorithme 2 : ParcoursTotal(Graphe G)

Algorithme

Données : deb : le point de départ,
marque : le tableau des marques

début

```

file ← initFile(). [deque()]
file.ajouteALaFin(deb) [file.append(deb)]
tant que file.estVide() = false faire
    s = file.premier()           // prochain sommet à visiter
    file.enlevePremier() [s = file.popleft()] // on le retire de la liste
    si marque.estMarque(s) alors
        // on est déjà passé là
        ... on fait les opérations prévues
    sinon
        // on n'est jamais passé là
        ... on fait les opérations prévues
        marque.marquer(s) [marque[s] = True] // il faut marquer s
        pour chaque t successeur de s faire
            file.ajouteALaFin(t) [file.append(t)] // il faut visiter ses successeurs

```

Algorithme 3 : ParcoursLargeur(Sommet deb, TableauBooleen marque)

Parcours de l'ensemble du graphe (Rappel)

Pour visiter l'ensemble du graphe

Données : $G = [X, U]$: le graphe

début

```
marque ← initMarques()
pour s ∈ X faire
    si marque.estMarque(s) = FAUX alors
        Parcours...(s, marque)
```

Algorithme 4 : ParcoursTotal(Graphe G)

Étude de la complexité

Si on ne considère que les actions nécessaires pour le parcours :

- On passe par chaque sommet une fois par arc dont il est l'origine
 $\Rightarrow O(NA)$ passages.
- Une seule fois par sommet, on parcourt les successeurs qu'on ajoute à la liste $\Rightarrow O(NS)$ parcours des successeurs.

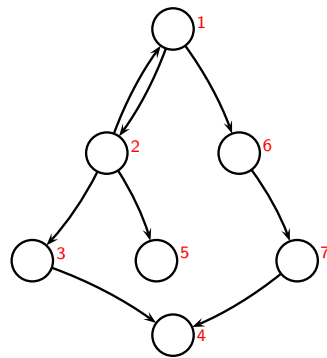
La complexité du parcours des successeurs dépend de la représentation :

- représentation matrice incidence
 - explorer les successeurs $O(NS) \Rightarrow NS \times NS \Rightarrow O(NS^2)$.
 - en tout : $O(NA) + O(NS^2)$ opérations.
- Représentation dictionnaire
 - pour chaque sommet on parcourt ses successeurs uniquement donc en tout on parcourt $\sum_{s \in X} \# \Gamma(s) = O(NA)$ successeurs.
 - en totalité, $O(NA)$ opérations.

Parcours en profondeur

Pour le parcours en profondeur, il faut visiter

- un des successeurs du points de départ ;
 - puis un successeurs de ce dernier ;
 - ...
 - on ne revient en arrière que lorsqu'il n'y a pas de successeur ;
 - ...
- $\Rightarrow$ Il faut visiter en premier les derniers points ajoutés.
 $\Rightarrow$ On utilise une pile (dernier arrivé premier servi)



Algorithme

Données : deb : le point de départ

marque : le tableau des marques

début

```
pile ← initPile() [list()]
pile.ajouteAuSommet(deb) [list.append(dev)]
tant que pile.estVide() = false faire
    s = pile.sommet()
    pile.enleveSommet() [s = list.pop()] // prochain sommet à visiter
    // on le retire de la liste
    si marque.estMarque(s) alors
        // on est déjà passé là
        ... on fait les opérations prévues
    sinon
        // on n'est jamais passé là
        ... on fait les opérations prévues
        marque.marquer(s) [marque[s]=True] // il faut marquer s
        pour chaque t successeur de s faire
            pile.ajouteAuSommet(t) [pile.append(t)] // il faut visiter ses
            successeurs
```

Algorithme 5 : ParcoursProfondeur(Sommet deb, TableauBooleen marque)

Parcours de l'ensemble du graphe (Rappel)

Pour visiter l'ensemble du graphe

Données : $G = [X, U]$: le graphe

début

```

marque ← initMarques()
pour  $s \in X$  faire
    si  $marque.estMarque(s) = FAUX$  alors
        Parcours...(s, marque)

```

Algorithme 6 : ParcoursTotal(Graphe G)

Parcours de l'ensemble du graphe (Rappel)

Pour visiter l'ensemble du graphe

Données : $G = [X, U]$: le graphe

début

```

marque ← initMarques()
pour  $s \in X$  faire
    si  $marque.estMarque(s) = FAUX$  alors
        Parcours...(s, marque)

```

Algorithme 8 : ParcoursTotal(Graphe G)

Parcours en profondeur version récursive

La méthode s'appelle elle-même sur les successeurs qu'elle rencontre.

Données : s : le point de départ

marque : le tableau des marques

début

```

si  $marque.estMarque(s)$  alors
    // on est déjà passé là
    ... on fait les opérations prévues
sinon
    // on n'est jamais passé là
    ... on fait les opérations prévues
    marque.marquer(s);
    pour chaque  $t$  successeur de  $s$  faire
        ParcoursRecuratif(t, marque);

```

Algorithme 7 : ParcoursRecuratif(Sommet s , TableauBooleen marque)

Où est passé la pile ? (Pile des appels récursifs)

1. Introduction

1.1 Définitions

1.2 Représentation des graphes

2. Parcours de graphes

2.1 Parcours en largeur

2.2 Parcours en profondeur

3. Fermeture transitive des graphes

3.1 Algorithme de WARSHALL

4. Recherche du plus court chemin

4.1 Algorithme de FORD

Fermeture transitive

Soit $G = [X, U]$ un graphe.

Définition (Fermeture transitive)

La fermeture transitive de G est le graphe $\overline{G} = [\overline{X}, \overline{U}]$ tq $(i, j) \in \overline{G}$ ssi il existe un chemin d'origine i et d'extrémité j .

Le calcul de la fermeture transitive passe par l'évaluation du nombre de chemins entre deux sommets.

Nombre de chemins entre deux sommets (suite)

- $HR_k \Rightarrow HR_{k+1}$:
 - M_{ij}^k indique le nombre de chemins de longueur k de i vers j .
 - $M_{ij}^{k+1} = \sum_{l=1}^{NS} M_{il}^k M_{lj}$.
 - Or M_{il}^k nombre de chemins de longueur k de i à l .
 - et M_{lj} nbre d'arcs de l à j (0 ou 1).
 - Donc $M_{il}^k M_{lj}$ est le nombre de chemins de longueur $k+1$ de i à j passant par l en dernier.
 - Ainsi $\sum_{l=1}^{NS} M_{il}^k M_{lj}$ est la somme des chemins passant par l lorsque l balaie l'ensemble des sommets.
- C.Q.F.D.

Nombre de chemins entre deux sommets

Soient :

$M = (M_{i,j})_{1 \leq i,j \leq n}$ matrice incidence sommet sommet.

$M^k = M \times \cdots \times M$ le produit de cette matrice k fois.

Lemme

Les coefficients de M^k correspondent aux nombres de chemins ayant k arcs (de longueur k) entre tous les couples de sommets.

preuve par récurrence sur k :

- pour $k = 1$, M_{ij} est le nombre de chemins de longueur 1 de départ i et d'arrivée j .

Existence d'un chemin entre deux sommets

Pour savoir s'il existe un chemin entre i et j il suffit de calculer $M^1, M^2, \dots, M^\infty$.

Cela pose un problème ! Mais :

- S'il existe un chemin entre i et j il existe un chemin élémentaire entre i et j .
- Un chemin élémentaire a au plus NS sommets.
- Pour prouver l'existence ou non d'un chemin il suffit de s'arrêter à M^{NS} .

Lemme

Il existe un chemin de i à j si et seulement si :

$$M_{ij} + M_{ij}^2 + \cdots + M_{ij}^{NS} \neq 0$$

Remarque : on peut utiliser une matrice booléenne, comment faire ?

Algorithme

Données : M matrice incidence du graphe G , NS : nbre sommets

début

```

 $FT \leftarrow M$ 
 $A \leftarrow M$ 
pour  $i=2$  à  $NS$  faire
     $A \leftarrow A * M$ 
     $FT \leftarrow FT + A$ 

```

Résultat : FT fermeture transitive du graphe

Algorithme 9 : FermetureTransitive

Remarque :

- FT contiendra le nombre de chemins de longueur $\leq NS$ pour tous les couples de sommets.
- (i, j) appartient à $\overline{U}$ ssi $F_{ij} \neq 0$.
- Si on remplace toutes les valeurs non nulles de FT par 1 on a la matrice incidence de $\overline{G}$ fermeture transitive de G .

Fermeture transitive : Algorithme de WARSHALL

Principe

- On compte le nombre de chemins élémentaires ne passant par aucun sommet intermédiaire (les arcs).
- On compte le nombre de chemins élémentaires ne passant par aucun sommet intermédiaire (cas précédant) ou passant uniquement par le sommet 1.
- On compte le nombre de chemins élémentaires ne passant par aucun sommet intermédiaire ou passant uniquement par les sommets appartenant à $\{1, 2\}$.
- ...
- On compte le nombre de chemins élémentaires ne passant par aucun sommet intermédiaire ou passant par les sommets appartenant à $\{1, \dots, NS\}$.

Complexité

- Quel est la complexité ?
- Comment faire mieux ?

Définitions

- Soit M matrice incidence du graphe.
- On définit une suite de graphes $G^{(k)}$ pour $k \in \{0, \dots, NS\}$
 - $G^{(0)} = G$ graphe de départ
 - $G^{(k)}$ est défini par :
 (i, j) est un arc de $G^{(k)}$ ssi il existe dans G un chemin de départ i et d'arrivée j dont les sommets intermédiaires appartiennent à $\{1, \dots, k\}$.

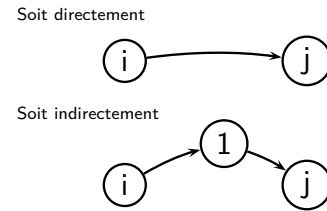
Remarque :

- $\emptyset \subseteq \{1, \dots, k\}$ donc les chemins ne sont pas obligés de passer par un autre sommet.
- On ne considère que les chemins élémentaires donc ils ne passent pas 2 fois par le même point.
- On note $M^{(k)}$ la matrice de $G^{(k)}$.

Construction $M^{(k)}$ par récurrence

- $M^{(0)} = M$ par définition
- Pour construire $M^{(1)}$, on cherche les couples (i, j) tels qu'il existe un chemin élémentaire dont le sommet intermédiaire appartient à $\{1\}$.

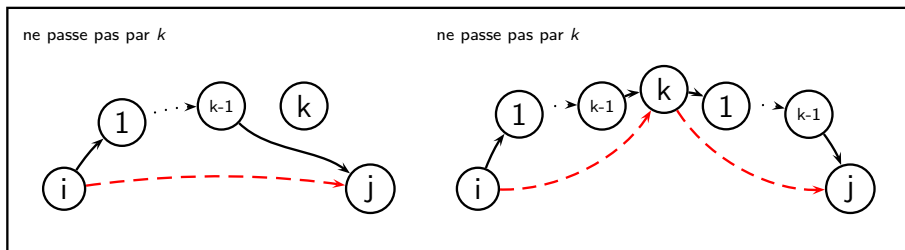
Il y a un chemin entre i et j $\Leftrightarrow$ passant éventuellement par 1



- Donc M_{ij}^1 est vrai si M_{ij}^0 est vrai ou (M_{i1}^0 est vrai et M_{1j}^0 est vrai).

Construction $M^{(k)}$ par récurrence (suite)

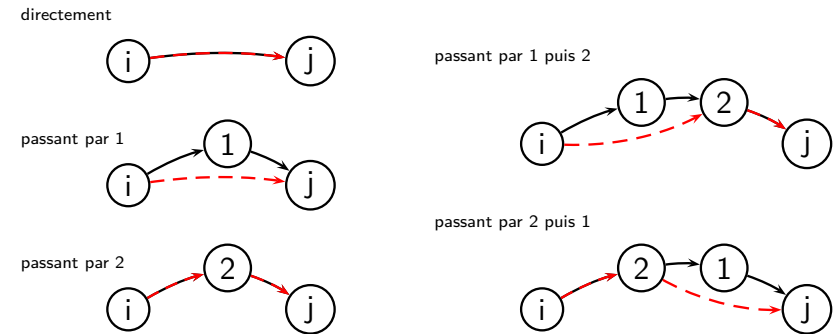
- Pour construire $M^{(k)}$, on cherche les couples (i, j) tels qu'il existe un chemin élémentaire dont les sommets intermédiaires appartiennent à $\{1, \dots, k\}$ sachant qu'on a déjà $M^{(k-1)}$.



- Soit le chemin ne passe pas par k alors (i, j) appartient à G^{k-1} .
- Soit le chemin passe une fois par k alors (i, k) appartient à G^{k-1} et (k, j) appartient à G^{k-1} .
- Donc M_{ij}^k est vrai si M_{ij}^{k-1} est vrai ou (M_{ik}^{k-1} est vrai et M_{kj}^{k-1} est vrai).

Construction $M^{(k)}$ par récurrence (suite)

- Pour construire $M^{(2)}$, on cherche les couples (i, j) tels qu'il existe un chemin élémentaire dont les sommets intermédiaires appartiennent à $\{1, 2\}$.
- Il y a un chemin entre i et j passant éventuellement par 1 ou 2 $\Leftrightarrow$



- M_{ij}^2 est vrai si M_{ij}^1 est vrai ou (M_{i2}^1 est vrai et M_{2j}^1 est vrai).

Algorithme

Données : M la matrice booléenne du graphe, NS le nombre de sommets

début

```

 $M^0 \leftarrow M$ 
pour  $k = 1$  à  $NS$  faire
  pour  $i = 1$  à  $NS$  faire
    pour  $j = 1$  à  $NS$  faire
       $M_{ij}^k \leftarrow M_{ij}^{k-1}$  ou ( $M_{ik}^{k-1}$  et  $M_{kj}^{k-1}$ )

```

Algorithme 10 : WARSHALL(M , NS)

1. Introduction

1.1 Définitions

1.2 Représentation des graphes

2. Parcours de graphes

2.1 Parcours en largeur

2.2 Parcours en profondeur

3. Fermeture transitive des graphes

3.1 Algorithme de WARSHALL

4. Recherche du plus court chemin

4.1 Algorithme de FORD

Plus court chemin

Pour aller d'un point à un autre, le coût est celui du chemin de poids minimal.

Remarque

Pourquoi ne pas tester tous les chemins possibles ?

Si le graphe est complet le nombre de chemins possibles est $NS!$, même si le graphe est incomplet, ce nombre peut être combinatoire.

Le problème de la recherche d'un chemin de valeur minimale peut se présenter sous 3 formes :

- recherche entre 2 sommets donnés ;
- recherche entre 1 sommet et tous les autres ;
- recherche sur tous les couples de sommets.

Nous ne verrons que la recherche entre un point et tous les autres.

Nous verrons ce calcul en 2 étapes :

- calcul de la valeur du chemin minimal ;
- calcul du chemin lui même.

Présentation du problème

On considère un graphe valué :

Définition (Graphe valué)

Un graphe valué est un graphe orienté muni d'une application $w : U \rightarrow \mathbb{R}$. Pour tout arc $u = (s, s')$ $w(s, s')$ est la valeur, le poids ou le coût de l'arc u .
Notation : $G[X, U, w]$.

- Dans ce cours les graphes ont toujours des valuations positives.
- La valeur d'un chemin entre 2 sommets est la somme des valeurs des arcs qui le composent.
- Comment calculer la distance entre 2 points sachant que plusieurs chemins sont possibles ?

Principe

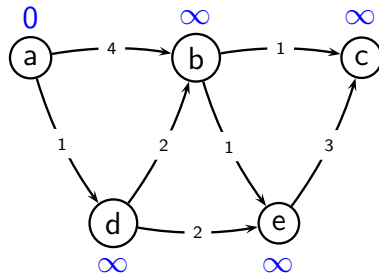
On calcule progressivement la longueur du plus court chemin à partir du sommet source s .

- Pour cela on maintient $d_s(v)$ les distances entre s le sommet initial et v les autres sommets.
- Au départ :

$$d_s(v) = \begin{cases} 0 & \text{si } s = v \\ \infty & \text{si } s \neq v \end{cases}$$

- On parcourt les arcs dans un ordre quelconque.
- Pour chaque arc (v, v') :
 - si $d_s(v) + w(v, v') < d_s(v')$ le chemin de s vers v' passant par v est plus court que celui trouvé précédemment
 - sinon ce chemin est plus long.
- Si on parcourt plusieurs fois les arcs, on finit par trouver l'ensemble des chemins les plus courts au départ de s .

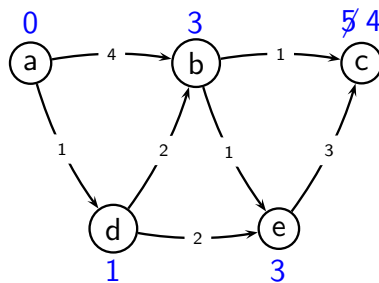
Exemple départ



On cherche le chemin le plus court au départ de a.

On parcourt par exemple les arcs dans l'ordre (a,b), (b,c), (a,d), (d,b), (b,e), (e,c) et (d,e).

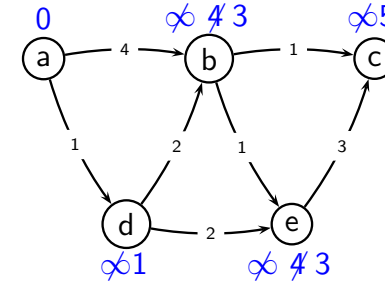
Exemple second parcours



On parcourt par exemple les arcs dans l'ordre (a,b), (b,c), (a,d), (d,b), (b,e), (e,c) et (d,e).

Lors du troisième parcours il n'y a aucun changement de valeur ce qui permet de savoir qu'on peut s'arrêter. Tous les chemins les plus courts ont été trouvés.

Exemple premier parcours



On cherche le chemin le plus court au départ de a.

On parcourt par exemple les arcs dans l'ordre (a,b), (b,c), (a,d), (d,b), (b,e), (e,c) et (d,e).

Est-ce suffisant ? Quel chemin plus court a-t-on trouvé ?

Description de l'algorithme

On cherche le plus court chemin entre s sommet de départ et les autres sommets.

Remarque :

- L'algorithme de FORD indique aussi si il existe ou non un chemin entre s et les autres sommets : il y a un chemin de s à v si $d_s(v) < \infty$.
- Calculer la distance d'un point s à un autre point v demande le calcul de toutes les distances de s à tous les autres points du graphe.

Il reste 2 questions :

- Pourquoi faut-il calculer toutes les distances ?
- Combien faut-il d'étapes (de parcours de la liste des arcs) ?

Détermination du plus court chemin

Comment calcule-t-on les chemins ?

- Comme les chemins sont emboîtés, pour chaque sommet il suffit de connaître son prédécesseur.
- Il suffit de mémoriser dans un tableau que le père du sommet v est v' .
- Il faut mettre à jour ce tableau à chaque fois qu'on met à jour la distance.

Algorithme

Données : s sommet de départ, $G = (S, U, w)$ le graphe, w est positif
début

```

pour chaque sommet  $v \in S$  faire
     $d_s(v) \leftarrow \infty$ 
     $pred(v) \leftarrow \emptyset$  // a priori pas de prédécesseur
    continue  $\leftarrow$  vrai
tant que continue faire
    continue  $\leftarrow$  faux
    pour chaque arrête  $(v, v') \in U$  faire
        si  $d_s(v) + w(v, v') < d_s(v')$  alors
             $d_s(v') \leftarrow d_s(v) + w(v, v')$ 
             $pred(v') \leftarrow v$  // le prédécesseur de  $v'$  est maintenant  $v$ 
            continue  $\leftarrow$  vrai

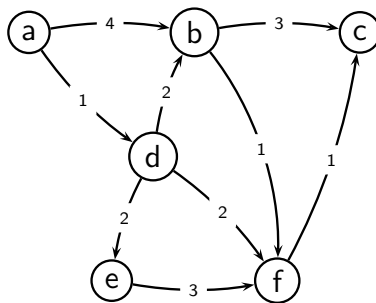
```

Résultat : d_s le tableau des distances depuis s et $pred$ le tableau des prédécesseurs

Algorithme 12 : FORD

Remarques pour avoir le chemin de s à t , on part de t et on remonte les prédécesseurs.

Exemple



Calculer les plus courts chemins au départ de a , en utilisant l'ordre de parcourt (a, b) , (a, d) , (b, c) , (b, f) , (d, e) , (d, f) , (e, f) .

- Calculez les distances et les prédécesseurs.
- Donnez les chemins

Conclusion

- L'algorithme de FORD fonctionne aussi s'il existe des chemins de valeurs négatives
 - il faut vérifier s'il existe des cycles de valeur négative ;
 - l'algorithme permet de le faire.
- On peut améliorer la complexité en choisissant correctement l'ordre de parcourt des arcs. C'est le principe de l'algorithme de DIJKSTRA.
- L'algorithme est utilisé dans les cas où le calcul est distribué dans le graphe, chaque sommet du graphe n'a besoin de savoir que :
 - sa distance avec ses voisins directes ;
 - leur distance avec les voisins du graphe connus ;
 - on parle de vecteur de distance.