



Recherche Opérationnelle R.O.

Partie 3: Optimisation Combinatoire

Pr. Abdessamad Kamouss

Cycle Ingénieur
ENSAM Casablanca

Contenu du module

1 Introduction à la recherche opérationnelle

2 Programmation linéaire

- Modélisation en Programmation linéaire
- Résolution des PL
- Dualité
- Post-Optimisation

3 Optimisation combinatoire

- Généralités sur les graphes
- Parcours eulériens et hamiltoniens des graphes
- Problème de plus court chemin
- Problèmes de transport et de flots

Contenu du module

1 Introduction à la recherche opérationnelle

2 Programmation linéaire

- Modélisation en Programmation linéaire
- Résolution des PL
- Dualité
- Post-Optimisation

3 Optimisation combinatoire

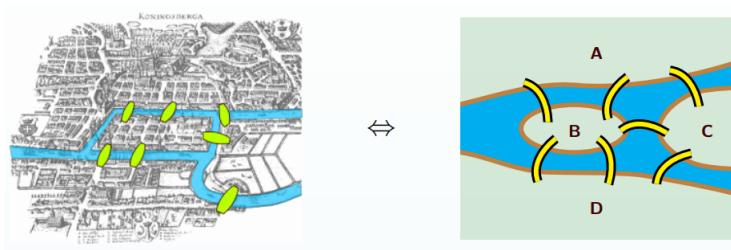
- Généralités sur les graphes
- Parcours eulériens et hamiltoniens des graphes
- Problème de plus court chemin
- Problèmes de transport et de flots

Définition des graphes

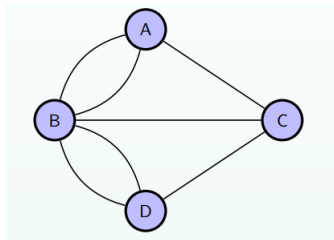
- Un graphe est tout d'abord :
 - un ensemble d'**éléments** ;
 - un ensemble de **relations** entre ces éléments.
 - Point de vue mathématique : une relation binaire
 - Point de vue pratique : représentation abstraite d'un réseau.
- Deux grandes familles :
 - les graphes **non-orientés** ;
 - les graphes **orientés**.
- Nombreuses conceptualisations et modélisations possibles.
- Utilisés dans des domaines très variés : économie, informatique, industrie, chimie, sociologie,...

Définition des graphes

Problème des sept ponts de Königsberg (Euler, 1735)



Représentation sous forme de graphe :



Définition des graphes

La théorie des graphes étudie des structures discrètes en forme de «réseaux»

Définition d'un graphe

Un graphe $(\mathcal{S}, \mathcal{A})$ est la donnée de deux ensembles d'objets :

- \mathcal{S} : l'ensemble des sommets ou noeuds.
- \mathcal{A} : un sous ensemble de $\mathcal{S} \times \mathcal{S}$ représentant l'ensemble des arêtes ou arcs.
 - **arêtes** symbolisant une relation symétrique,
 - **arcs** symbolisant une relation orientée.

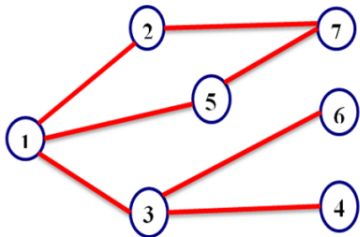
Types de graphes

Nous pouvons alors distinguer deux familles de graphes :

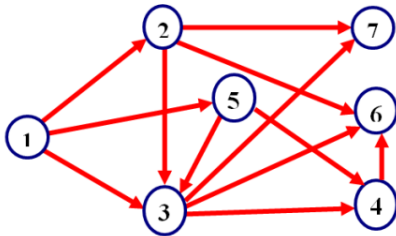
- les graphes dirigés (en anglais, directed graphs ou digraph) ;
- les graphes symétriques (en anglais, undirected graphs).

Types de graphes

Graphe Symétrique



Graphe Orienté



Modélisation

- réseaux de transports routier, d'eau, d'électricité
- réseaux informatiques
- réseaux sociaux : (Facebook : graphe non orienté, twitter : graphe orienté)
- graphe du web
- réseau de transports de données (téléphonie, wifi, réseaux informatique...)
- représentation d'un algorithme, du déroulement d'un jeu ;
- réseaux de régulation génétique
- organisation logistique
- ordonnancement de projet
- ...

Définition (Graphe symétrique)

Un **graphe non-orienté ou symétrique** est une relation binaire \mathcal{G} sur un ensemble \mathcal{A} , notée $\mathcal{G} = (\mathcal{S}, \mathcal{A})$ où \mathcal{S} est l'ensemble des sommets et \mathcal{A} l'ensemble des arêtes qui est formé des $\{\mathbf{x}, \mathbf{y}\} \in \mathcal{A}$ qui sont en relation.

Vocabulaire des graphes symétriques

- L'**ordre** d'un graphe est son nombre de sommets.
- La **taille** d'un graphe est son nombre d'arêtes.
- Si $\{x, y\}$ est une arête, x et y sont voisins et sont appelés les extrémités de cet arête.

Définition (Graphe orienté)

Un **graphe dirigé ou orienté** est une relation binaire \mathcal{G} sur un ensemble \mathcal{A} , notée $\mathcal{G} = (\mathcal{S}, \mathcal{A})$ où \mathcal{S} est l'ensemble des sommets et \mathcal{A} est l'ensemble des arcs qui sont formé des couples $(\mathbf{x}, \mathbf{y}) \in \mathcal{S} \times \mathcal{S}$ qui sont en relation.

Vocabulaire des graphes orientés

- L'**ordre** d'un graphe est son nombre de sommets.
- La **taille** d'un graphe est son nombre d'arcs.
- Pour l'arc (x, y) , on dit que :
 - x et y sont adjacents.
 - x est son **origine** et y son **extrémité**.
 - y est un **successeur** (ou suivant) de x
 - x est un **prédécesseur** (ou précédent) de y .

Définitions (Degré)

- Un **graphe simple** est un graphe qui ne contient pas des multi-arêtes ou multi-arcs, c'est-à-dire qu'entre deux sommets il n'existe qu'une seule arête ou un seul arc dans chaque sens.
- Soit x un sommet d'un graphe symétrique. Le **degré** de x , noté $d(x)$, est le nombre d'arêtes incidentes à x , c'est-à-dire contenant x .
- Soit x un sommet d'un graphe orienté. On note $d^+(x)$ le nombre d'arcs ayant x comme extrémité initiale, et $d^-(x)$ le nombre d'arcs ayant x comme extrémité finale. Ainsi, on a le **degré de** x , noté $d(x)$ vérifie : $d(x) = d^+(x) + d^-(x)$.

On a :

- $\sum_{x \in \mathcal{S}} d(x) = 2|A|.$
- $\sum_{x \in \mathcal{S}} d^+(x) = \sum_{x \in \mathcal{S}} d^-(x) = |A|.$

représentation de Graphe - Matrice d'adjacence

Afin de représenter un graphe, on a plusieurs possibilités :

- Matrice d'adjacence
- Matrice d'incidence
- Listes d'adjacence
- ...

Matrice d'adjacence d'un graphe simple

Soit $\mathcal{G} = (\mathcal{S}, \mathcal{A})$ un graphe simple de n sommets numérotés de 1 à n .

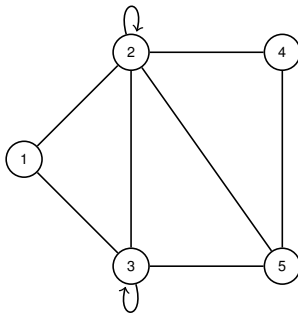
La **matrice d'adjacence** $A = (a_{ij})$ de \mathcal{G} est la matrice carrée d'ordre n où chaque ligne et chaque colonne représentent un sommet et qui est définie par :

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1j} & \cdots & a_{1n} \\ a_{21} & \cdots & a_{2j} & \cdots & a_{2n} \\ \vdots & & \vdots & & \vdots \\ a_{n1} & \cdots & a_{nj} & \cdots & a_{nn} \end{pmatrix}$$

Graphe - Matrice d'adjacence

Graphe symétrique non valué

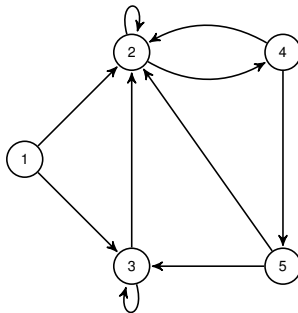
$$a_{ij} = \begin{cases} 1 & \text{si } \{i, j\} \in \mathcal{A} \\ 0 & \text{sinon.} \end{cases}$$



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

Graphe orienté non valué

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

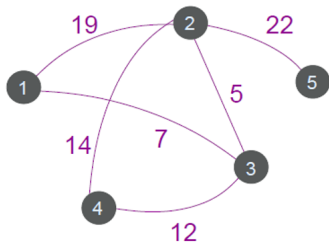


$$A = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 \end{pmatrix}$$

Graphe - Matrice d'adjacence

Graphe symétrique valué

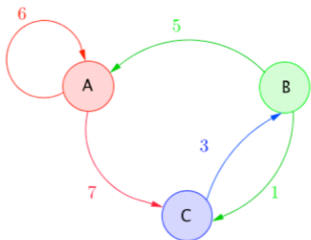
$$a_{ij} = \begin{cases} p & \text{si } \{i, j\} \in \mathcal{A} \\ 0 & \text{si } i = j \\ +\infty & \text{sinon.} \end{cases}$$



$$A = \begin{pmatrix} 0 & 19 & 7 & +\infty & +\infty \\ 19 & 0 & 5 & 14 & 22 \\ 7 & 5 & 0 & 12 & +\infty \\ +\infty & 14 & 12 & 0 & +\infty \\ +\infty & 22 & +\infty & +\infty & 0 \end{pmatrix}$$

Graphe orienté valué

$$a_{ij} = \begin{cases} p & \text{si } (i, j) \in \mathcal{A} \\ +\infty & \text{sinon} \end{cases}$$



$$A = \begin{pmatrix} 6 & +\infty & 7 \\ +\infty & +\infty & 1 \\ +\infty & 3 & +\infty \end{pmatrix}$$

Contenu du module

1 Introduction à la recherche opérationnelle

2 Programmation linéaire

- Modélisation en Programmation linéaire
- Résolution des PL
- Dualité
- Post-Optimisation

3 Optimisation combinatoire

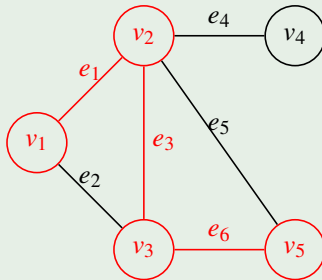
- Généralités sur les graphes
- **Parcours eulériens et hamiltoniens des graphes**
- Problème de plus court chemin
- Problèmes de transport et de flots

Chaîne dans un graphe symétrique

- Une **chaîne** dans un graphe G est une suite ayant pour éléments alternativement des sommets et des arêtes, commençant et se terminant par un sommet et telle que chaque arête est encadré par ses extrémités.
- On dit que la chaîne relie le premier sommet et le dernier sommet.
- $l(C)$ est la **longueur d'une chaîne** C qui représente son nombre d'arêtes.
- $d(x, y)$ est la **distance** entre deux sommets x et y qui représente la longueur de la plus petite chaîne les reliant.
- $\delta(G)$ est le **diamètre d'un graphe** G qui représente la plus grande distance entre deux sommets.

Exemple

Dans le graphe G suivant :



- On considère la chaîne : $C = (v_1, e_1, v_2, e_3, v_3, e_6, v_5)$. $l(C) = 3$.
- La distance entre v_1 et v_5 est $d(v_1, v_5) = 2$.
- Le diamètre de G est $\delta(G) = 2$

Les graphes de diamètre 0 sont les graphes stables (sans arête)

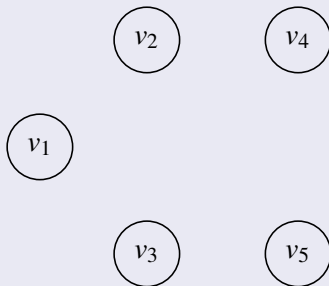
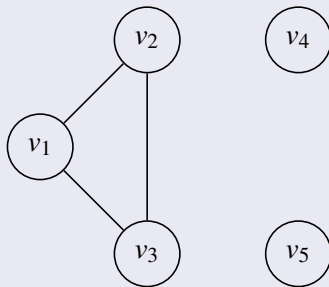
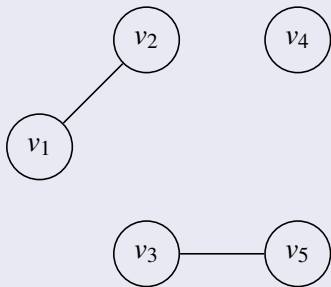


Figure – Graphe stable

Graphes de diamètre 1



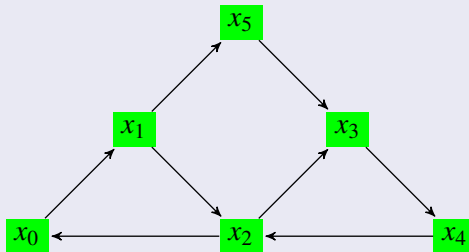
Chaîne élémentaire et chaîne simple

- **Chaîne élémentaire :**
si chaque **sommet** apparaît au plus une fois.
- **Chaîne simple :**
si chaque **arête** apparaît au plus une fois.
- **Chaîne fermée :**
Une chaîne dont les extrémités sont les mêmes.
- **Cycle**
Une chaîne simple et fermée.

Chemin dans un digraphe

- Un **chemin de x à y** est une suite finie d'arcs consécutifs, reliant x à y .
La **longueur d'un chemin** est le nombre d'arcs du chemin.
- **Chemin élémentaire :**
un chemin ne passant pas deux fois par un même **sommet** du graphe
- **Chemin simple :**
un chemin ne passant pas deux fois par un même **arc** du graphe.
- **Chemin fermée :**
un chemin dont l'origine et l'extrémité coïncident.
- **Circuit :**
un chemin simple dont les deux extrémités sont identiques.

Exemple



- $(x_0, x_1, x_2, x_3, x_4)$ est un chemin.
- $(x_0, x_1, x_2, x_3, x_4, x_2, x_0)$ est un circuit.

Graphe eulérien et graphe semi-eulérien

- **Graphe eulérien**

Un graphe symétrique est dit **eulérien** s'il abrite un cycle (chaîne simple fermée) passant une et une seule fois par toutes les arêtes.

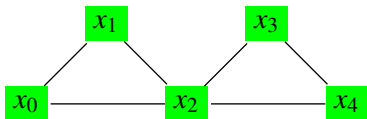
- **Graphe semi-eulérien**

Un graphe symétrique est dit **semi-eulérien** s'il abrite une chaîne simple passant une et une seule fois par toutes les arêtes.

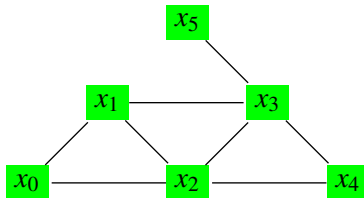
Intuitivement :

Un graphe est eulérien (ou semi-eulérien) s'il est possible de le dessiner sans lever le crayon et sans passer deux fois par le même trait.

Graphe eulérien



Graphe eulérien



Graphe semi-eulérien

Graphe hamiltonien et graphe semi-hamiltonien

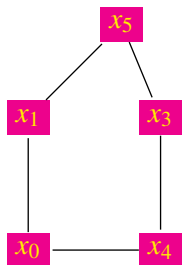
- **Graphe hamiltonien**

Un graphe symétrique est dit **hamiltonien** s'il abrite un cycle (chaîne élémentaire fermée) passant une et une seule fois par tous les sommets.

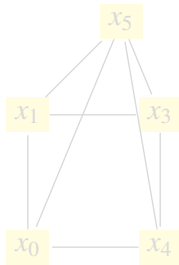
- **Graphe semi-hamiltonien**

Un graphe symétrique est dit **semi-hamiltonien** s'il abrite une chaîne (chaîne élémentaire) passant une et une seule fois par tous les sommets.

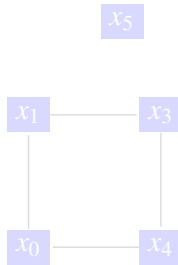
Graphe eulérien/hamiltonien - Exemples



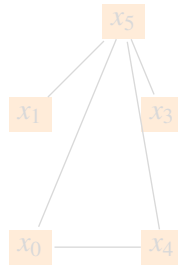
H et E



H et \bar{E}

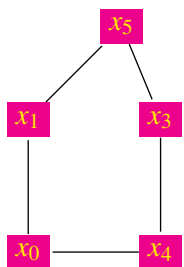


\bar{H} et E

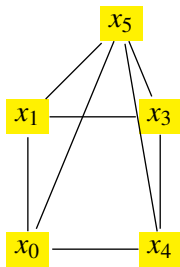


\bar{H} et \bar{E}

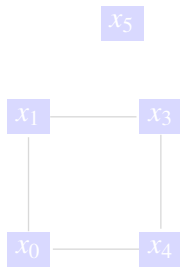
Graphe eulérien/hamiltonien - Exemples



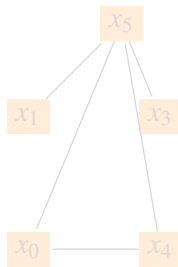
H et E



H et \bar{E}

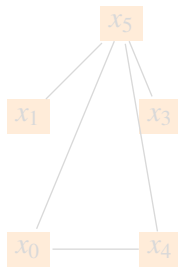
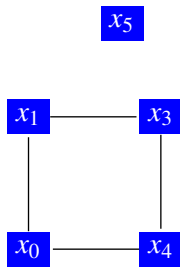
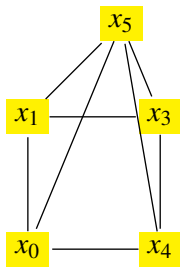
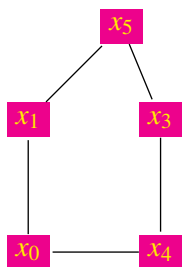


\bar{H} et E

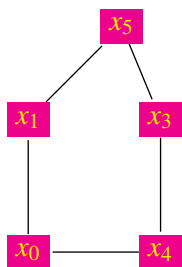


\bar{H} et \bar{E}

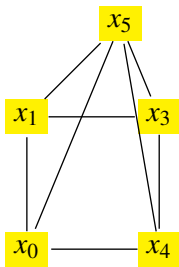
Graphe eulérien/hamiltonien - Exemples



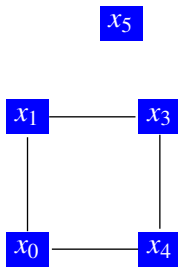
Graphe eulérien/hamiltonien - Exemples



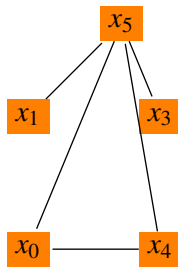
H et E



H et \bar{E}



\bar{H} et E



\bar{H} et \bar{E}

Caractérisation des graphes eulériens et semi-eulériens

Théorème d'Euler

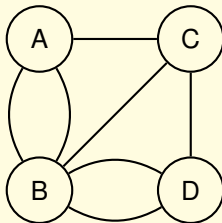
Soit $\mathcal{G} = (\mathcal{S}, \mathcal{A})$ un graphe symétrique. Notons par $d(x)$ le degré de x .

- (i) Si pour tout $x \in \mathcal{S}$ on a $d(x)$ est pair, alors \mathcal{G} admet **un cycle eulérien**.
- (ii) Si il existe uniquement deux sommets distincts x_1 et x_2 qui ont **des degrés impaires** pour tout $x \in \mathcal{S} \setminus \{x_1, x_2\}$ $d(x)$ est pair, alors \mathcal{G} abrite **une chaîne eulérienne** d'extrémités x_1 et x_2 .
- (iii) Dans tous les autres cas \mathcal{G} **n'abrite pas de chaîne eulérienne**.

Réponse négative au problème d'Euler

Euler souhaitait se promener dans sa ville natale en passant une et une seule fois par chaque pont.

Donc le graphe doit être eulérien.



Mais, d'après le résultat précédent, ce n'est pas possible puisque les degrés des sommets ne sont pas tous pairs.

Graphe hamiltonien

Caractérisation des graphes hamiltoniens

Théorème de Dirac

On considère un graphe simple à n sommets ($n \geq 3$).

Si chaque sommet a un degré au moins égale à $\frac{n}{2}$ alors **le graphe est hamiltonien**.

Théorème de Ore

On considère un graphe simple à n sommets ($n \geq 3$).

Si la somme des degrés de toute paire de sommets non adjacents vaut au moins n alors **le graphe est hamiltonien**.

Attention. Les deux résultats précédents représentent des conditions suffisantes mais qui ne sont pas nécessaires.

Contenu du module

1 Introduction à la recherche opérationnelle

2 Programmation linéaire

- Modélisation en Programmation linéaire
- Résolution des PL
- Dualité
- Post-Optimisation

3 Optimisation combinatoire

- Généralités sur les graphes
- Parcours eulériens et hamiltoniens des graphes
- **Problème de plus court chemin**
- Problèmes de transport et de flots

Problème de plus court chemin

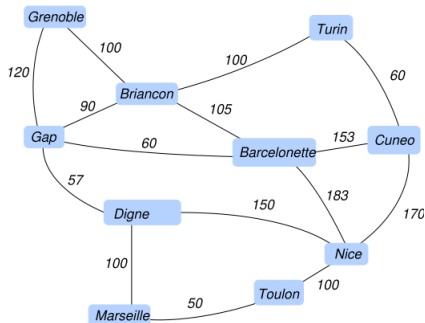
Les problèmes de cheminement dans les graphes comptent parmi les problèmes les plus anciens de la théorie des graphes.

En particulier la recherche d'un plus court chemin **PCC** constitue un problème amplement investi et fait partie des plus importants problèmes étudiés par leurs applications.

Plusieurs applications :

- Recherche d'un trajet le plus court.
- Recherche de la route la plus rapide.
- Recherche de la stratégie optimale.
- ...

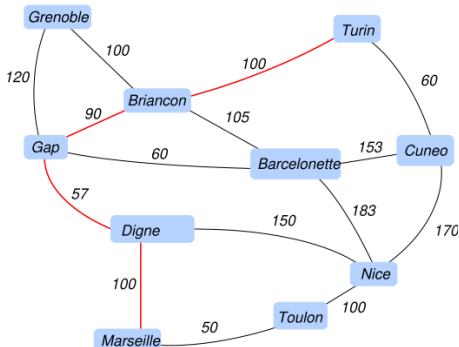
Trajet le plus court entre deux villes : On considère la carte suivantes permettant de relier Marseille et Turin :



On cherche à déterminer sur cette carte le plus court chemin entre ces deux villes.

Trajet le plus court entre deux villes :

Le plus court chemin entre Marseille et Turin :



Définitions

- **Une racine** dans un graphe orienté (resp. non-orienté) est un sommet r tel qu'il existe un chemin (resp. une chaîne) de r à u pour tout sommet $u \neq r$.
- Un circuit (resp. cycle) de longueur négative est appelé un **circuit (resp. cycle) absorbant**.

Théorème

Une condition nécessaire et suffisante pour trouver un plus court chemin d'un sommet **s** à tous les sommets du graphe est que :

- **s** soit racine du graphe,
- le graphe ne contient pas de circuit/cycle absorbant.

Théorème de Sous-optimalité

Soit C un plus court chemin de u à v et u', v' deux sommets de C .

Alors le sous-chemin de C de u' à v' est aussi un plus court chemin de u' à v' .

Preuve :

On considère le sous-chemin chemin C_1 de u' à v' contenu dans le chemin C . Supposons par absurde qu'il existe un autre chemin C_2 de u' à v' qui est plus court que C_1 .

Donc le chemin C' de u à v constitué des sous-chemins suivants : sous-chemin de u à u' , C_2 et sous-chemin de v' à v sera plus court que le chemin C .

Ce qui représente une contradiction.

Nous allons étudier deux algorithmes qui permettent de résoudre des problèmes de recherche de plus courts chemins à racine unique :

- un algorithme **(Dijkstra)** qui peut être utilisé pour des graphes dont les pondérations de tous les arêtes ou les arcs sont positives ou nulles ;
- un algorithme **(Bellman-Ford)** qui peut être utilisé pour n'importe quel graphe avec éventuellement des pondérations négatives mais qui ne comportant pas de circuit absorbant.

Algorithme de Dijkstra



Algorithme de Dijkstra

Algorithme de Dijkstra

Dijkstra est un algorithme de recherche de distance et de plus court chemin entre **un sommet racine fixé s** et **tous les autres sommets** d'un graphe.

Il est composée de deux étapes :

Initialisation :

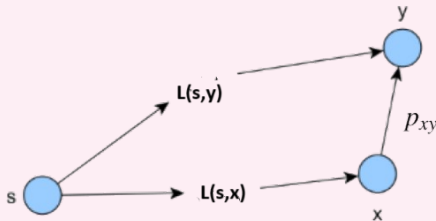
- Attribuer à chaque sommet successeur de **s** une distance égale au poids entre **s** et ce successeur.
- Les autres sommets non reliés directement à **s** , la valeur $+\infty$.

On commence les itérations :

- Au début de chaque itération, on choisit un sommet **x** parmi ceux qui n'ont pas encore été traités, dont la distance à **s** est minimale.
- Pour chaque successeur **y** de **x** , on compare la valeur actuelle de la longueur du chemin déjà trouvé **$L(s, y)$** avec la longueur du nouveau chemin **$L(s, x) + p_{xy}$** .

Algorithme de Dijkstra

Algorithme de Dijkstra



- On met alors éventuellement à jour la distance $L(s,y)$ si la nouvelle valeur trouvée est meilleure.
- Quand tous les successeurs du sommet x ont été examinés, on rajoute x à la liste des sommets traités, et l'on recommence par une nouvelle itération, avec le choix d'un nouveau sommet.
- Les itérations se terminent lorsque la totalité des sommets sont traités.

Notations

Soit $G = (\mathcal{S}, \mathcal{A})$ un graphe orienté ou symétrique à valuation **positives ou nulles**.

Soient **s** le sommet racine à partir duquel on va chercher les plus courts chemins.

Afin d'exécuter l'algorithme Dijkstra on va définir les tableaux **L**, **P** et **M** suivants :

- **L** un tableau de n cases destiné à contenir les distances de **s** aux autres sommets.
- **P** un tableau de n cases destiné à contenir les prédécesseurs de chacun des sommets dans un plus court chemin d'origine **s**.
- **M** la liste de tous les sommets déjà traités (et donc $\mathcal{S} \setminus M$ sera la liste de tous les sommets à traiter).

Algorithme de Dijkstra

Initialisation

```
1:  $L[s] \leftarrow 0$ 
2:  $P[s] \leftarrow s$  ( $s$  est la racine)
3: for ( $x \neq s$ ) do
4:   if ( $x$  est un successeur de  $s$ ) then
5:      $L[x] \leftarrow p_{sx}$ 
6:      $P[x] \leftarrow s$ 
7:   else
8:      $L[x] \leftarrow +\infty$ 
9:      $P[x] \leftarrow \emptyset$ 
10:  end if
11: end for
12:  $M \leftarrow \{s\}$ 
```

Algorithme de Dijkstra

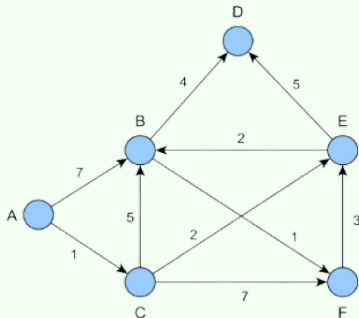
Traitement

```
1: while ( $M \neq S$ ) do  
2:   choisir  $x \in S \setminus M$  tq : pour tout  $y \in S \setminus M$ ,  $L[x] \leq L[y]$   
3:   for ( $y \in S \setminus M$  tel que  $y$  soit un successeur de  $x$ ) do  
4:     if ( $L[x] + p_{xy} < L[y]$ ) then  
5:        $L[y] \leftarrow L[x] + p_{xy}$   
6:        $P[y] \leftarrow x$   
7:     end if  
8:   end for  
9:    $M \leftarrow M \cup \{x\}$   
10: end while
```

Algorithme de Dijkstra

Exemple

Considérons le graphe G orienté valué dont la représentation sagittale est la suivante :

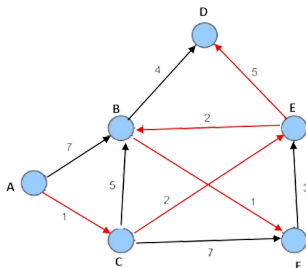


Question : Appliquer l'algorithme de Dijkstra à ce graphe en partant du sommet A

Algorithme de Dijkstra

Initialisation et Traitement :

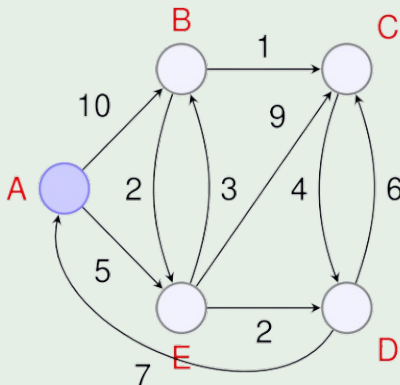
	L						P					
	A	B	C	D	E	F	A	B	C	D	E	F
A	0	7	1	$+\infty$	$+\infty$	$+\infty$	A	A	A	-	-	-
A, C	0	6	1	$+\infty$	3	8	A	C	A	-	C	C
A, C, E	0	5	1	8	3	8	A	E	A	E	C	C
A, C, E, B	0	5	1	8	3	6	A	E	A	E	C	B
A, C, E, B, F	0	5	1	8	3	6	A	E	A	E	C	B
A, C, E, B, F, D	0	5	1	8	3	6	A	E	A	E	C	B



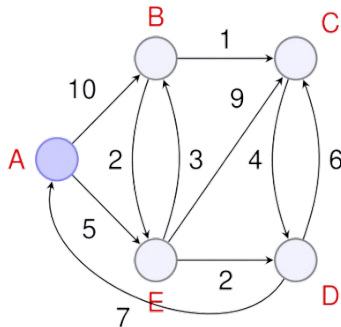
Algorithme de Dijkstra

Exemple (Application de l'algorithme de Dijkstra)

Donner les plus courts chemin d'origine le sommet A dans le graphe suivant :



Algorithme de Dijkstra

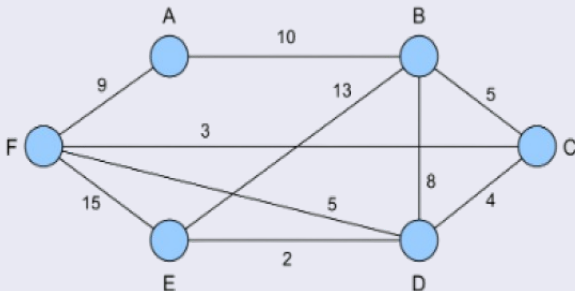


	L					P				
	A	B	C	D	E	A	B	C	D	E
A	0	10	$+\infty$	$+\infty$	5	A	A	-	-	A
AE	0	8	14	7	5	A	E	E	E	A
AED	0	8	13	7	5	A	E	D	E	A
AEDB	0	8	9	7	5	A	E	B	E	A
AEDBC	0	8	9	7	5	A	E	B	E	A

Algorithme de Dijkstra

Exercice

Considérons le graphe non orienté valué dont la représentation sagittale est la suivante :



Question : Appliquer l'algorithme de Dijkstra à ce graphe en partant du sommet A

Algorithme de Bellman-Ford



Algorithme de Bellman-Ford

Bellman-Ford est un algorithme de recherche de distance et de plus court chemin entre **un sommet racine fixé s** et **tous les autres sommets** d'un graphe.

Notations

- **s** le sommet du graphe à partir duquel on va rechercher tous les plus courts chemins vers les autres sommets du graphe.
- **L** le tableau de dimension n qui contient les distances de **s** aux autres sommets.
- **P** le tableau de dimension n qui contient le prédécesseur de chacun des sommets dans un plus court chemin d'origine **s** .
- **Continue** un booléen indiquant la fin ou non de l'algorithme.

Algorithme de Bellman-Ford

Initialisation

```
1:  $L[s] = 0$   
2:  $P[s] = s$   
3: for  $(x \neq s)$  do  
4:    $L[x] \leftarrow +\infty$   
5:    $P[x] \leftarrow \emptyset$   
6: end for  
7: Continue  $\leftarrow$  Vrai
```

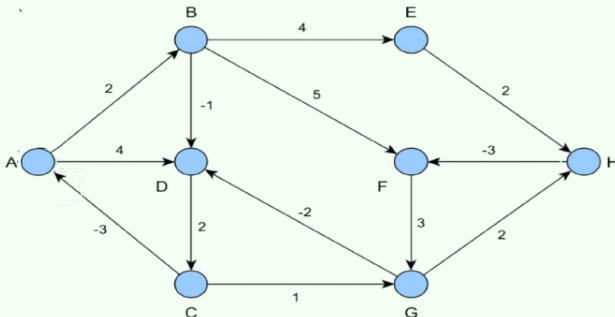
Traitement

```
1: while (Continue) do  
2:   Continue  $\leftarrow$  Faux  
3:   for ( $x \in \mathcal{S}$ ) do  
4:     for ( $y \in \mathcal{S}$  tq  $y$  soit un successeur de  $x$ ) do  
5:       if  $L[x] + p_{xy} < L[y]$  then  
6:          $L[y] = L[x] + p_{xy}$   
7:          $P[y] = x$   
8:         Continue  $\leftarrow$  Vrai  
9:       end if  
10:    end for  
11:  end for  
12: end while
```

Algorithme de Bellman-Ford

Exemple

Considérons le graphe G orienté valué dont la représentation sagittale est la suivante :

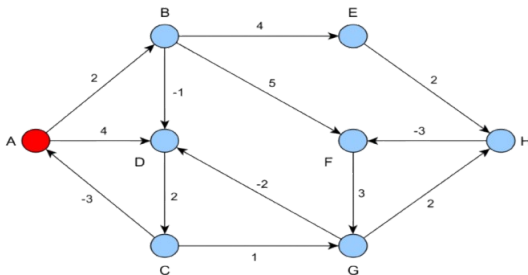


Question : Appliquer l'algorithme de Bellman-Ford à ce graphe en partant du sommet A

Algorithme de Bellman-Ford

Initialisation

Durant cette phase d'initialisation on commence par le sommet racine **A** et on va attribuer la valeur $+\infty$ à la table L et la valeur \emptyset à la table P pour tous les sommets à l'exception de l'origine **A**



	L								P							
	A	B	C	D	E	F	G	H	A	B	C	D	E	F	G	H
A	0	∞	∞	∞	∞	∞	∞	∞	A	-	-	-	-	-	-	-

Algorithme de Bellman-Ford

Traitement : 1^{ère} itération

Dans cette itération on procède au traitement de la totalité des sommets dans l'ordre alphabétique.

	L								P							
	A	B	C	D	E	F	G	H	A	B	C	D	E	F	G	H
Init	0	∞	∞	∞	∞	∞	∞	∞	A	-	-	-	-	-	-	-
A	0	2	∞	4	∞	∞	∞	∞	A	A	-	A	-	-	-	-
B	0	2	∞	1	6	7	∞	∞	A	A	-	B	B	B	-	-
C	0	2	∞	1	6	7	∞	∞	A	A	-	B	B	B	-	-
D	0	2	3	1	6	7	∞	∞	A	A	D	B	B	B	-	-
E	0	2	3	1	6	7	∞	8	A	A	D	B	B	B	-	E
F	0	2	3	1	6	7	10	8	A	A	D	B	B	B	F	E
G	0	2	3	1	6	7	10	8	A	A	D	B	B	B	F	E
H	0	2	3	1	6	5	10	8	A	A	D	B	B	H	F	E

Algorithme de Bellman-Ford

Traitement : 2^{ème} itération

Dans cette itération on procède au traitement de la totalité des sommets dans l'ordre alphabétique.

	L								P							
	A	B	C	D	E	F	G	H	A	B	C	D	E	F	G	H
pré	0	2	3	1	6	5	10	8	A	A	D	B	B	H	F	E
A	0	2	3	1	6	5	10	8	A	A	D	B	B	H	F	E
B	0	2	3	1	6	5	10	8	A	A	D	B	B	H	F	E
C	0	2	3	1	6	5	4	8	A	A	D	B	B	H	C	E
D	0	2	3	1	6	5	4	8	A	A	D	B	B	H	C	E
E	0	2	3	1	6	5	4	8	A	A	D	B	B	H	C	E
F	0	2	3	1	6	5	4	8	A	A	D	B	B	H	C	E
G	0	2	3	1	6	5	4	6	A	A	D	B	B	H	C	G
H	0	2	3	1	6	3	4	6	A	A	D	B	B	H	C	G

Algorithme de Bellman-Ford

Traitement : 3^{ème} itération

Dans cette itération on procède au traitement de la totalité des sommets dans l'ordre alphabétique.

	L								P							
	A	B	C	D	E	F	G	H	A	B	C	D	E	F	G	H
pré	0	2	3	1	6	3	4	6	A	A	D	B	B	H	C	G
A	0	2	3	1	6	3	4	6	A	A	D	B	B	H	C	G
B	0	2	3	1	6	3	4	6	A	A	D	B	B	H	C	G
C	0	2	3	1	6	3	4	6	A	A	D	B	B	H	C	G
D	0	2	3	1	6	3	4	6	A	A	D	B	B	H	C	G
E	0	2	3	1	6	3	4	6	A	A	D	B	B	H	C	G
F	0	2	3	1	6	3	4	6	A	A	D	B	B	H	C	G
G	0	2	3	1	6	3	4	6	A	A	D	B	B	H	C	G
H	0	2	3	1	6	3	4	6	A	A	D	B	B	H	C	G

Algorithme de Bellman-Ford

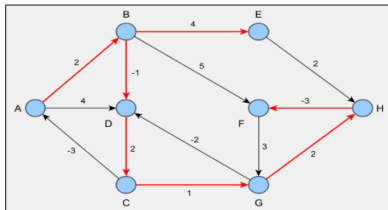
Traitement : Convergence de l'algorithme

Après le passage d'une itération sans aucun changement la variable

Continue reste sur la valeur **Faux** donc l'algorithme converge.

On obtient donc le résultat suivant :

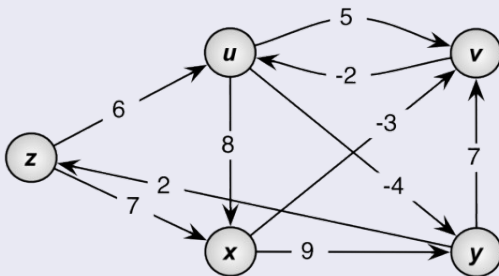
	L								P							
	A	B	C	D	E	F	G	H	A	B	C	D	E	F	G	H
Fin	0	2	3	1	6	3	4	6	A	A	D	B	B	H	C	G



Algorithme de Bellman-Ford

Exercice

Considérons le graphe non orienté valué dont la représentation sagittale est la suivante :



Question : Appliquer l'algorithme de Bellman-Ford à ce graphe en partant du sommet z

Contenu du module

1 Introduction à la recherche opérationnelle

2 Programmation linéaire

- Modélisation en Programmation linéaire
- Résolution des PL
- Dualité
- Post-Optimisation

3 Optimisation combinatoire

- Généralités sur les graphes
- Parcours eulériens et hamiltoniens des graphes
- Problème de plus court chemin
- Problèmes de transport et de flots

Introduction : Problème de transport

Problème de transport - Flot maximal

- Le problème du **flot maximal** sur un **réseau de transport** est une problématique centrale dans la recherche opérationnelle et l'optimisation combinatoire.
- Il consiste à maximiser la quantité de flot qui peut être acheminée d'une ou plusieurs **sources** vers une ou plusieurs **destinations** à travers un **réseau de transport** (ou un graphe orienté), tout en respectant certaines contraintes.
- Le réseau est représenté par un graphe où les noeuds sont des points (représentant des villes, des serveurs, ou tout autre point de connexion), et les arcs sont des liens entre ces noeuds, ayant chacun une capacité maximale de transport.
- L'objectif est de déterminer le **flot maximal** que l'on peut envoyer des sources (noeuds de départ) aux puits (noeuds d'arrivée), en tenant compte des capacités des arcs.

Applications

Le problème du flot maximal a de nombreuses applications pratiques, notamment dans :

- Gestion des réseaux de transport
 - Trafic routier et transport public
 - Aéroports et ports
- Réseaux de communication et transmission de données
 - Internet et réseaux informatiques
 - Télécommunications
- Distribution des ressources
 - Réseaux d'eau, d'électricité ou de gaz
 - Logistique et approvisionnement
- Planification et gestion des projets
 - Gestion des flux de production
 - Planification de projets

Applications

- Assignment de tâches ou de ressources
 - Affectation des travailleurs
 - Gestion des matchs sportifs
- Biologie et médecine
 - Réseaux sanguins ou lymphatiques :
 - Recherche génomique et biologie computationnelle
- Problèmes financiers
 - Flux de capitaux en investissement
 - Flux des transferts financiers
- Matching bipartite et problèmes d'affectation
- ...

Réseau de transport

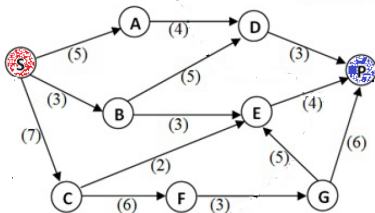
Définition (Réseau de transport)

On appelle **réseau de transport** un graphe orienté simple antisymétrique $G = (\mathcal{S}, \mathcal{A})$ valué positivement sans boucle, ayant :

- une racine **s**,
- un puits **p**,

Les valuations des arcs sont appelées **capacités** c_{ij} .

On dénote $C = \{c_{ij}, (i, j) \in \mathcal{A}\}$ et on dit que $G = [\mathcal{S}, \mathcal{A}, C]$ est un **réseau de transport avec capacités**.



Définition (Flot)

On appelle **flot réalisable** sur un **réseau de transport avec capacités** $G = [\mathcal{S}, \mathcal{A}, C]$ une application $\phi : \mathcal{A} \rightarrow \mathbb{R}$ qui vérifie :

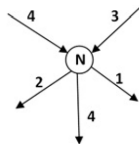
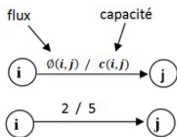
❶ **Les contraintes de capacité :**

Pour tout arc (i, j) de \mathcal{A} on a $0 \leq \phi_{ij} \leq c_{ij}$;

❷ **Les contraintes de conservation (Kirchoff) :**

Pour tout sommet i de $\mathcal{S} \setminus \{s, p\}$, on a : $\sum_{k \in \mathcal{A}^-(i)} \phi_{ki} = \sum_{j \in \mathcal{A}^+(i)} \phi_{ij}$

$$\sum_{k \in \mathcal{A}^+(s)} \phi_{sk} = \sum_{j \in \mathcal{A}^-(p)} \phi_{jp}$$



Définitions (Flot)

Dans le contexte de la définition précédente :

- ① $V(\phi) = \sum_{k \in \mathcal{A}^+(s)} \phi_{sk} = \sum_{j \in \mathcal{A}^-(p)} \phi_{jp}$ est appelée la **valeur du flot**.

Elle représente le volume total que met en circulation le flot réalisable ϕ dans le réseau de transport.

- ② La quantité $\mathbf{Cr}_{ij} = \mathbf{c}_{ij} - \phi_{ij}$ est la **capacité résiduelle** de l'arc (i,j) .

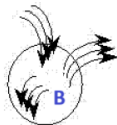
- ③ Si $\phi_{ij} = \mathbf{c}_{ij}$, on dit dans ce cas que l'arc (i,j) est **saturé**.

Lemme (généralisation des contraintes de Kirchoff)

Si B est un sous-ensemble de S alors :
le **flot sortant** de B est égal au **flot entrant** dans B .

Preuve :

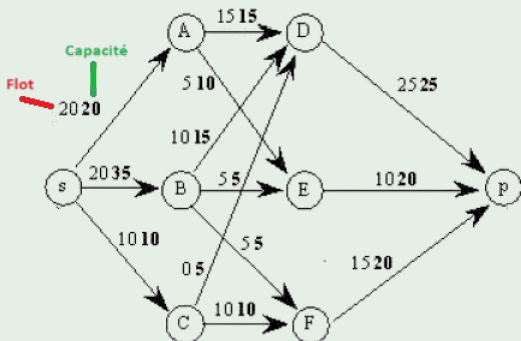
On somme l'égalité de Kirchoff sur l'ensemble des sommets de B .
On a les flots des arcs qui ont les 2 extrémités dans B vont apparaître de chaque côté de l'égalité donc vont se simplifier.



Il reste alors d'un côté de l'égalité la somme des flux des arcs entrants dans B et de l'autre côté de l'égalité, la somme des flux des arcs sortants de B .

Exemple

Le schéma suivant représente un exemple de flot sur un réseau de transport :



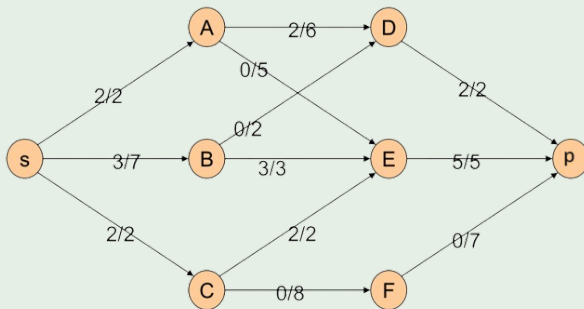
- **Pour le sommet B** : flot entrant = 20 et flot sortant = $10+5+5 = 20$.
- Si on considère le sous ensemble $X = \{B, C, D, E\}$:
 Flot entrant = $15+5+20+10 = 50$ et flot sortant = $25+10+5+10 = 50$

Flot Complet

Définition (Flot Complet)

On dit qu'un **flot est complet** si tout chemin du réseau de transport allant de s à p contient au moins un **arc saturé**.

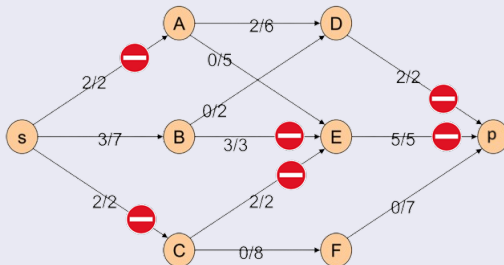
Exemple



Flot maximal

Amélioration du flot

On considère le flot suivant :



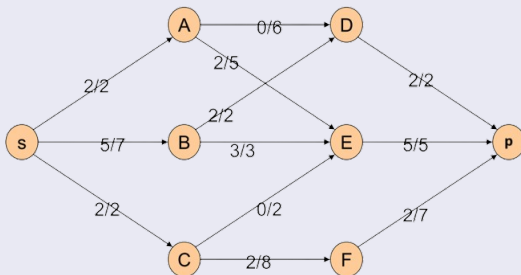
On constate que la valeur de ce flot est $V(\phi) = 7$.

Question

Peut-on améliorer la valeur de ce flot ?

Flot maximal

Sur le réseau de transport précédent, on peut obtenir un flot meilleur :



On constate que la valeur de ce flot est $V(\phi) = 9$.

Question

Est-ce que ce flot est maximal ?

Flot maximal

Définition (Flot maximal)

Soit $G = [\mathcal{S}, \mathcal{A}, C]$ un réseau de transport avec capacités.

Le flot de valeur maximale ϕ^* qui est, parmi l'ensemble des flots réalisables, celui qui maximise la quantité $V(\phi)$ est appelé **flot maximal** :

$$(\forall \phi \text{ un flot réalisable}) : V(\phi^*) \geq V(\phi)$$

Remarques

- Un flot maximal n'est pas nécessairement unique.
- Le problème de flot maximal peut-être modélisé par programmation linéaire.
- Mais les caractéristiques de celui-ci permettent d'élaborer des algorithmes de résolutions exactes plus efficaces dans la recherche du flot maximal : **algorithme de Ford-Fulkerson**.

Flot maximal - Algorithme de Ford-Fulkerson

Algorithme de Ford-Fulkerson

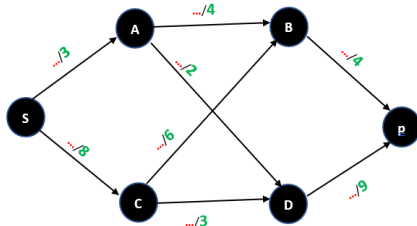
Soit $G = [\mathcal{S}, \mathcal{A}, C]$ un réseau de transport avec capacités.

On commence par un flot réalisable initial et on suit les étapes suivantes :

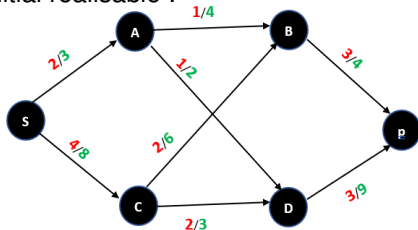
- 1 Marquer l'entrée **s** par +.
- 2 Soit i un sommet marqué non examiné.
Étudier tous les successeurs j de i :
Marquer j par **+i** : s'il est non marqué et si $\phi_{ij} < c_{ij}$.
Étudier tous les prédécesseurs k de i :
Marquer k par **-i** : s'il est non marqué et si $\phi_{ki} > 0$.
- 3 Si le sommet **p** est marqué, aller en (4)
Sinon : S'il reste des sommets marqués non examinés, aller en (2).
Sinon le flot est maximal, **FIN**.
- 4 Améliorer le flot à l'aide de la chaîne améliorante ayant permis de marquer **p** et effacer les marques, sauf celle de **s**, et aller en (1).

Exemple : Algorithme de Ford-Fulkerson

On considère le réseau de transport suivant :



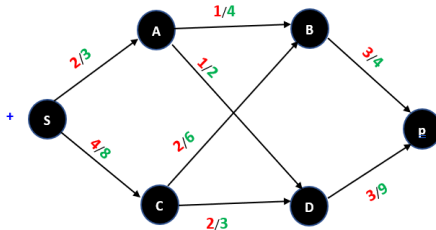
On cherche un flot initial réalisable :



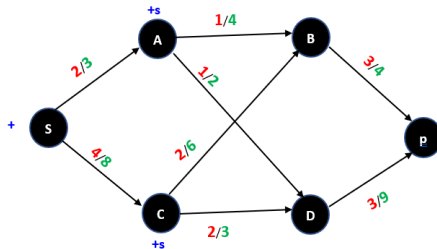
On constate que la valeur de ce flot est $V(\phi) = 6$.

Exemple : Algorithme de Ford-Fulkerson

On commence le marquage par le sommet **S** :

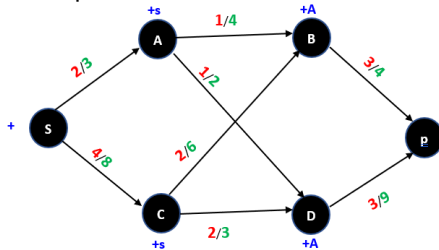


On examine le sommet **S** :

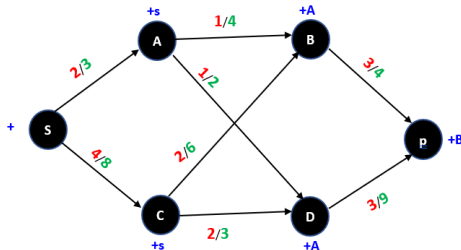


Exemple : Algorithme de Ford-Fulkerson

On examine le sommet **A** puis le sommet **C** :

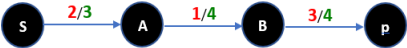


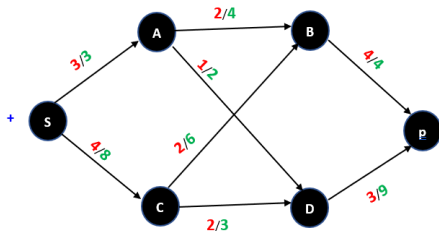
On examine le sommet **B** puis le sommet **D** :



Exemple : Algorithme de Ford-Fulkerson

- Puisque le sommet **p** a été marqué, on va améliorer le flot à l'aide de la chaîne améliorante ayant permis de marquer **p**.

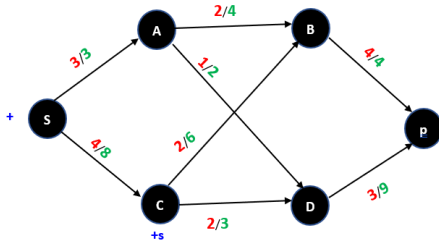
- La chaîne améliorante est 
- La valeur possible d'amélioration est $\alpha = \min\{3 - 2; 4 - 1; 4 - 3\} = 1$.
- On obtient donc le flux amélioré suivant :



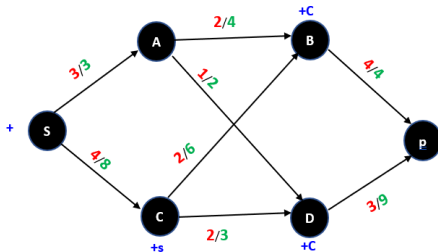
- La valeur de ce nouveau flot est $V(\phi) = 7$.

Exemple : Algorithme de Ford-Fulkerson

On examine le sommet **S** :

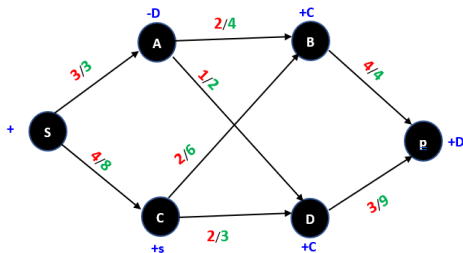


On examine le sommet **C** :



Exemple : Algorithme de Ford-Fulkerson

On examine le sommet **D** puis le sommet **B** :



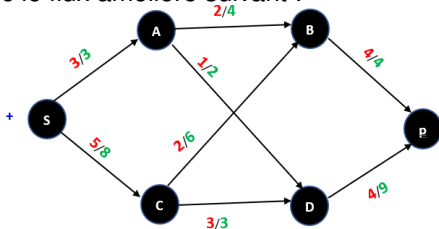
- Puisque le sommet **p** a été marqué, on va améliorer le flot à l'aide de la chaîne améliorante ayant permis de marquer **p**.

- La chaîne améliorante est

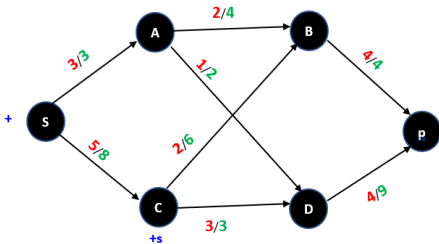
- La valeur possible d'amélioration est $\alpha = \min\{8 - 4; 3 - 2; 9 - 3\} = 1$.

Exemple : Algorithme de Ford-Fulkerson

- On obtient donc le flux amélioré suivant :

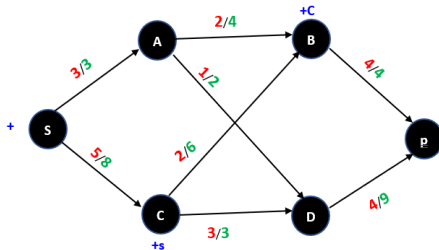


- La valeur de ce nouveau flot est $V(\phi) = 8$.
- On examine le sommet **S** :

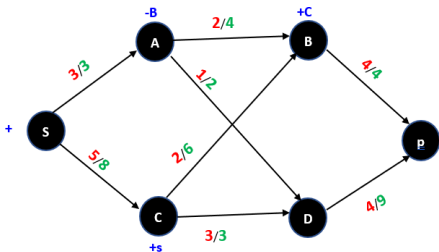


Exemple : Algorithme de Ford-Fulkerson

On examine le sommet **C** :

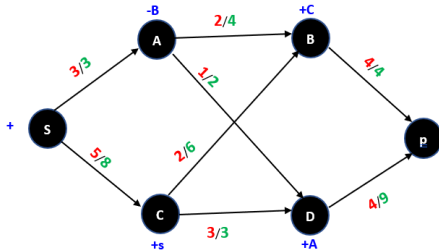


On examine le sommet **B** :

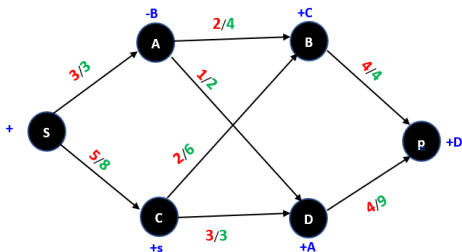


Exemple : Algorithme de Ford-Fulkerson

On examine le sommet **A** :



On examine le sommet **D** :



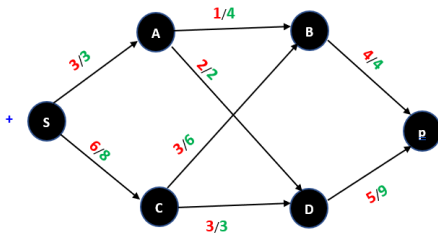
Exemple : Algorithme de Ford-Fulkerson

- Puisque le sommet **p** a été marqué, on va améliorer le flot à l'aide de la chaîne améliorante ayant permis de marquer **p**.

- La chaîne améliorante est



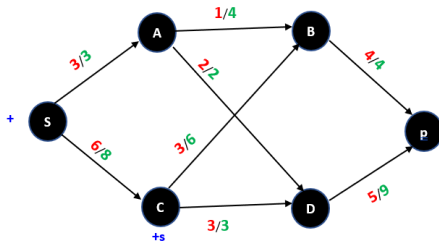
- La valeur possible d'amélioration est
 $\alpha = \min\{8 - 5; 6 - 2; 2; 2 - 1; 9 - 4\} = 1.$
- On obtient donc le flux amélioré suivant :



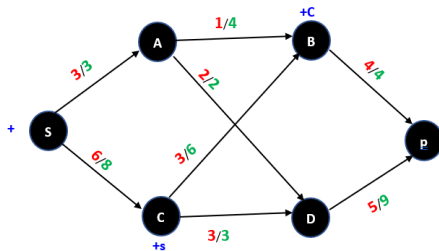
- La valeur de ce nouveau flot est $V(\phi) = 9.$

Exemple : Algorithme de Ford-Fulkerson

On examine le sommet **S** :

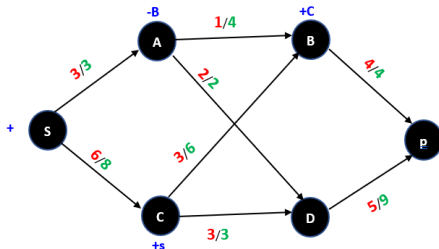


On examine le sommet **C** :



Exemple : Algorithme de Ford-Fulkerson

On examine le sommet **B** :



On examine le sommet **A**, mais on ne peut pas marquer un autre sommet. Puisqu'il ne reste aucun sommet marqué non examiné et on n'arrive pas à marquer le sommet puits **p** alors :

- Le dernier flot obtenu est **maximal**
- La valeur maximale du flot est $V(\phi^*) = 9$.

- Pourquoi l'algorithme de Ford-Fulkerson permet d'obtenir une solution exacte au problème de flot maximal ?
- Quelques définitions et propriétés permettent de justifier ceci

Définition (Coupe)

Une **coupe** K dans le réseau de transport $G = [\mathcal{S}, \mathcal{A}, C]$, est une partition de \mathcal{S} en deux sous-ensembles U et \bar{U} tels que $s \in U$ et $p \in \bar{U}$.

La **capacité d'une coupe** est définie par :

$$c(K) = \sum_{\substack{(i,j) \in \mathcal{A} \\ i \in U \text{ et } j \in \bar{U}}} c_{ij}$$

Donc $c(K)$ désigne la somme des capacités des arcs sortants de U et allant vers \bar{U} .

Proposition (Coupe)

Pour toute coupe $K = \{U, \bar{U}\}$ la valeur de tout flot ϕ est la valeur du flot circulant entre U et \bar{U} est $\phi_{U \rightarrow \bar{U}} - \phi_{\bar{U} \rightarrow U}$, i.e :

$$V(\phi) = \sum_{i \in U, j \in \bar{U}} \phi_{ij} - \sum_{j \in \bar{U}, i \in U} \phi_{ji}$$

Preuve : On a $V(\phi) = \sum_{i \in U} \left(\sum_j \phi_{ij} - \sum_j \phi_{ji} \right)$

$$V(\phi) = \sum_{i \in U} \left(\sum_{\substack{j \\ j \in U}} \phi_{ij} + \sum_{\substack{j \\ j \in \bar{U}}} \phi_{ij} - \sum_{\substack{j \\ j \in U}} \phi_{ji} - \sum_{\substack{j \\ j \in \bar{U}}} \phi_{ji} \right)$$

En simplifiant : $V(\phi) = \sum_{i \in U, j \in \bar{U}} \phi_{ij} - \sum_{j \in \bar{U}, i \in U} \phi_{ji}$

Proposition (Coupe)

Pour toute coupe $K = \{U, \overline{U}\}$ et un flot réalisable ϕ , on a :

$$c(K) \geq V(\phi)$$

Preuve :

Pour tout arc $(i, j) \in \mathcal{A}$, nous avons : $0 \leq \phi_{ij} \leq c_{ij}$ d'où :

$$\begin{aligned} c(K) &= \sum_{i \in U, j \in \overline{U}} c_{ij} \\ &\geq \sum_{i \in U, j \in \overline{U}} \phi_{ij} \\ &\geq \sum_{i \in U, j \in \overline{U}} \phi_{ij} - \sum_{j \in \overline{U}, i \in U} \phi_{ji} \\ &\geq V(\phi) \end{aligned}$$

Bien fondé de l'algorithme de Ford-Fulkerson

Le résultat théorique sur lequel repose la recherche d'un flot de valeur maximale dans le théorème de Ford-Fulkerson :

Théorème : Coupe minimale - Flot maximal

Soit $G = [\mathcal{S}, \mathcal{A}, C]$ un réseau de transport avec capacités.

On note par \mathbb{F} la famille des flots réalisables de G et par \mathbb{K} la famille des coupes dans G .

On considère les deux quantités suivantes :

$$V_M = V(\phi^*) = \max_{\phi \in \mathbb{F}} \{V(\phi)\}$$

$$c_m = \min_{K \in \mathbb{K}} \{c(K)\}$$

Alors

$$V_M = c_m.$$

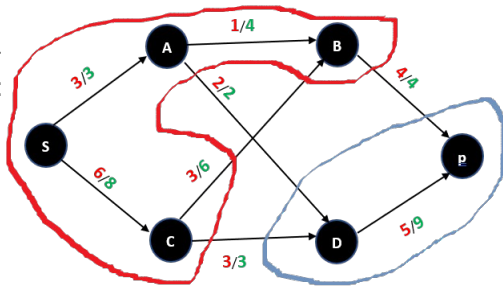
Bien fondé de l'algorithme de Ford-Fulkerson

On reprend l'exemple précédent dans la dernière itération :

On considère la coupe

$$K = \{U, \overline{U}\} \text{ avec}$$
$$U = \{s; A; B; C\}$$

et $\overline{U} = \{D; p\}$.



- Tout $(i, j) \in \mathcal{A}$ tel que $i \in U$ et $j \in \overline{U}$ est saturé.
- Tout $(j, i) \in \mathcal{A}$ tel que $j \in \overline{U}$ et $i \in U$ vérifie $\phi_{ji} = 0$.
- Alors

$$c(K) = \min_{i \in U, j \in \bar{U}} \sum c_{ij} = \max_{i \in U, j \in \bar{U}} \sum \phi_{ij} - \min_{j \in \bar{U}, i \in U} \sum \phi_{ji} = \max V(\phi^*)$$

Optimisation combinatoire

- L'optimisation combinatoire constitue une branche très importante de la recherche opérationnelle.
- Elle permet de résoudre plusieurs problèmes réels et de réaliser des gains considérables et d'assurer des optimisations dans les problèmes de la recherche de chemins et de cycles optimaux ainsi que l'optimisation de flots dans les réseaux.
- Il reste très important d'aborder :
 - **Les problèmes de couverture et d'affectation** : Problème de couverture minimale, Problème d'affectation, Problème de couplage maximal.
 - **Les problèmes de partitionnement et de découpe** : Partitionnement de graphes et Problème du sac à dos.
 - **Les problèmes de tournées et de chemins** : Problème du voyageur de commerce, Problème de tournées de véhicules.