1 - Notions Fondamentales de la Théorie des Graphes

Université Alger 1, Dept Maths & Informatique

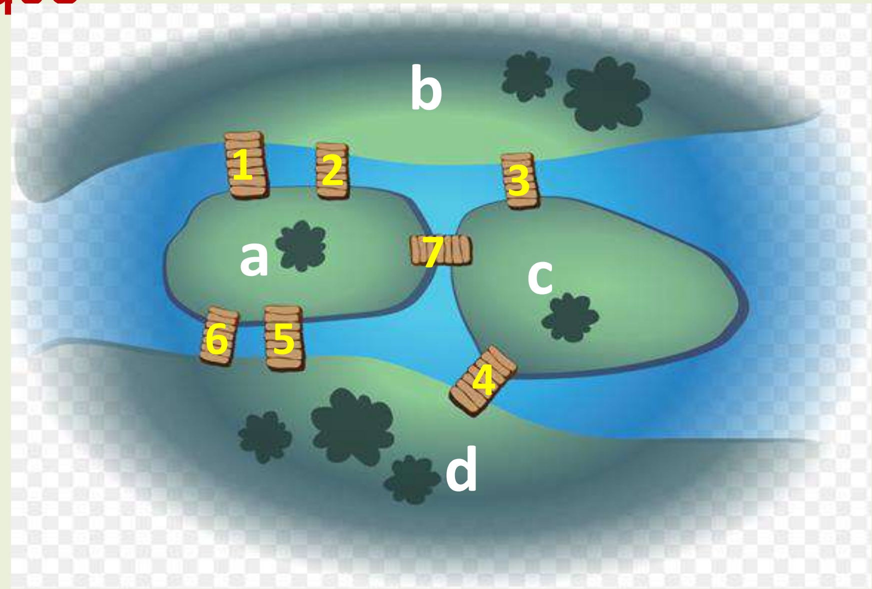
Dr. Fodil LAIB

Février 2017

Principe et Origines

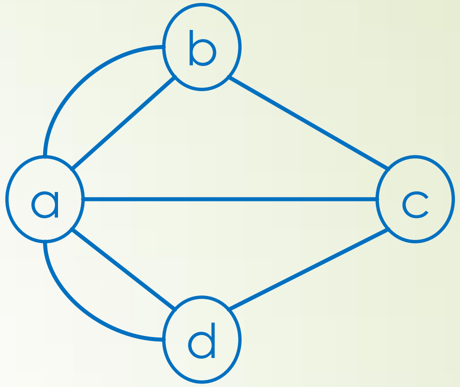
- La théorie des graphes visualise une problématique par un graphe synoptique.
- Elle propose des algorithmes de résolution.

Historique



Les 7 ponts de Königsberg

- Travaux d'Euler (1735) : Comment traverser les 7 ponts de la ville de Königsberg (Russie) une seule fois et revenir au point de départ ?



Représentation de Königsberg par un graphe

- Le théorème d'Euler affirme que ce problème n'admet pas de solution.
- Kirchhoff (1847) : analyse des circuits électriques
- Hamilton (1857) : Trouver un chemin passant une seule fois par les 18 villes du jeu icosien.

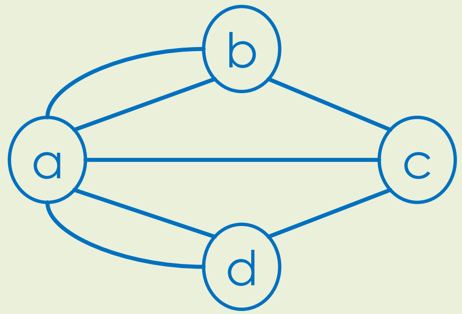
Définitions de Base

Un graphe G=(X,U) est composé de :

X : ensemble des sommets

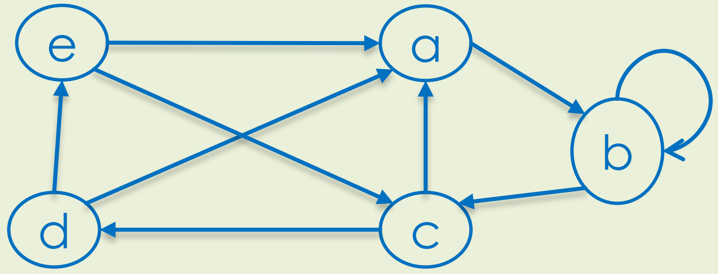
U : ensemble des liens reliant les sommets.

- Graphe non orienté : les liens sont des arêtes



Graphe non orienté

- Graphe orienté : les liens sont des arcs

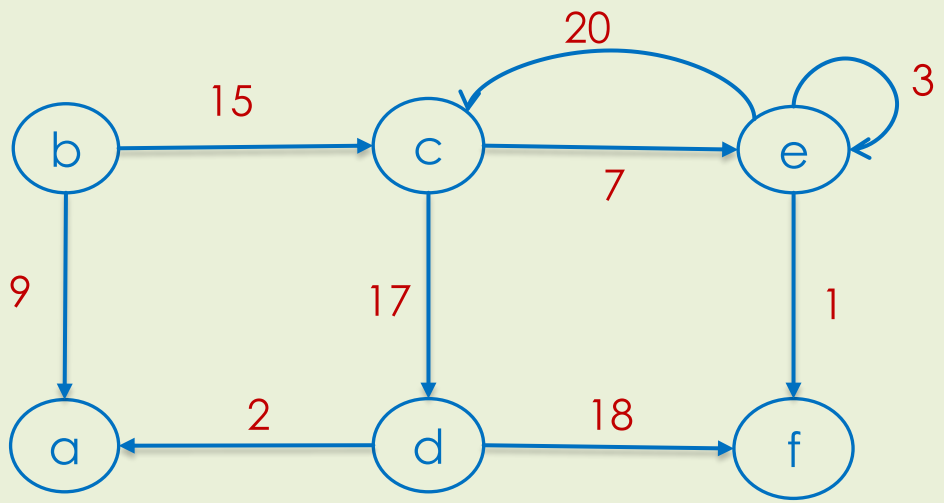


Graphe Orienté

- X = {a, b, c, d, e}
- U={ (a,b), (b,b),(b,c), (c,a), (c,d), (d,a), (d,e), (e,a), (e,c) }

- Un arc *u=(x, x)* dont les deux extrémités coïncident est une boucle. L'arc *u=(b,b)* est une boucle.
- Ordre d'un graphe : c'est le nombre de sommets du graphe, $n = |X|$.
- Taille d'un graphe : c'est le nombre d'arcs du graphe, $m = |U|$.

Graphe valué : tout arc (ou arête) porte une valeur numérique. Ces valeurs peuvent être des quantités transportées, des débits, des coûts, etc.

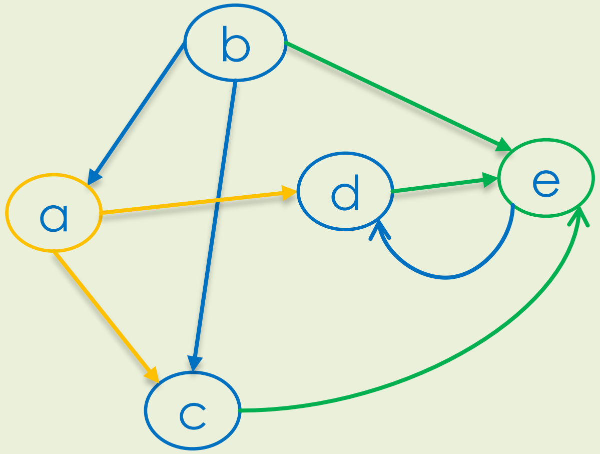


Graphe orienté et valué

Prédécesseurs et Successeurs

Soit x un sommet d'un graphe orienté :

- $U^-(x) = \{y \in U, (y,x) \in U\}$: ensemble des prédécesseurs de x
- $U^+(x) = \{y \in U, (x,y) \in U\}$: ensemble des successeurs de x
- $U(x) = U^-(x) \cup U^+(x)$: ensemble des voisins (ou sommets adjacents) de x



- Les successeurs de a sont $U^+(a) = \{c, d\}$,
- Les prédécesseurs de e sont : $U^-(e) = \{b, c, d\}$
- Les voisins de a sont $U(a) = \{b, c, d\}$

Degrè d'un sommet

- $d^-(x) = |U^-(x)|$: demi-degré intérieur de x
- $d^+(x) = |U^+(x)|$: demi-degré extérieur de x
- $d(x) = d^-(x) + d^+(x)$: degré de x
- Les boucles ne sont pas prises en compte.

Eg.

- $d^+(a) = 2$
- $d^-(a) = 1$

$\Rightarrow d(a) = 2 + 1 = 3$

- $d^+(e) = 1$
- $d^-(e) = 3$

$\Rightarrow d(e) = 1 + 3 = 4$

Modélisation par un Graphe

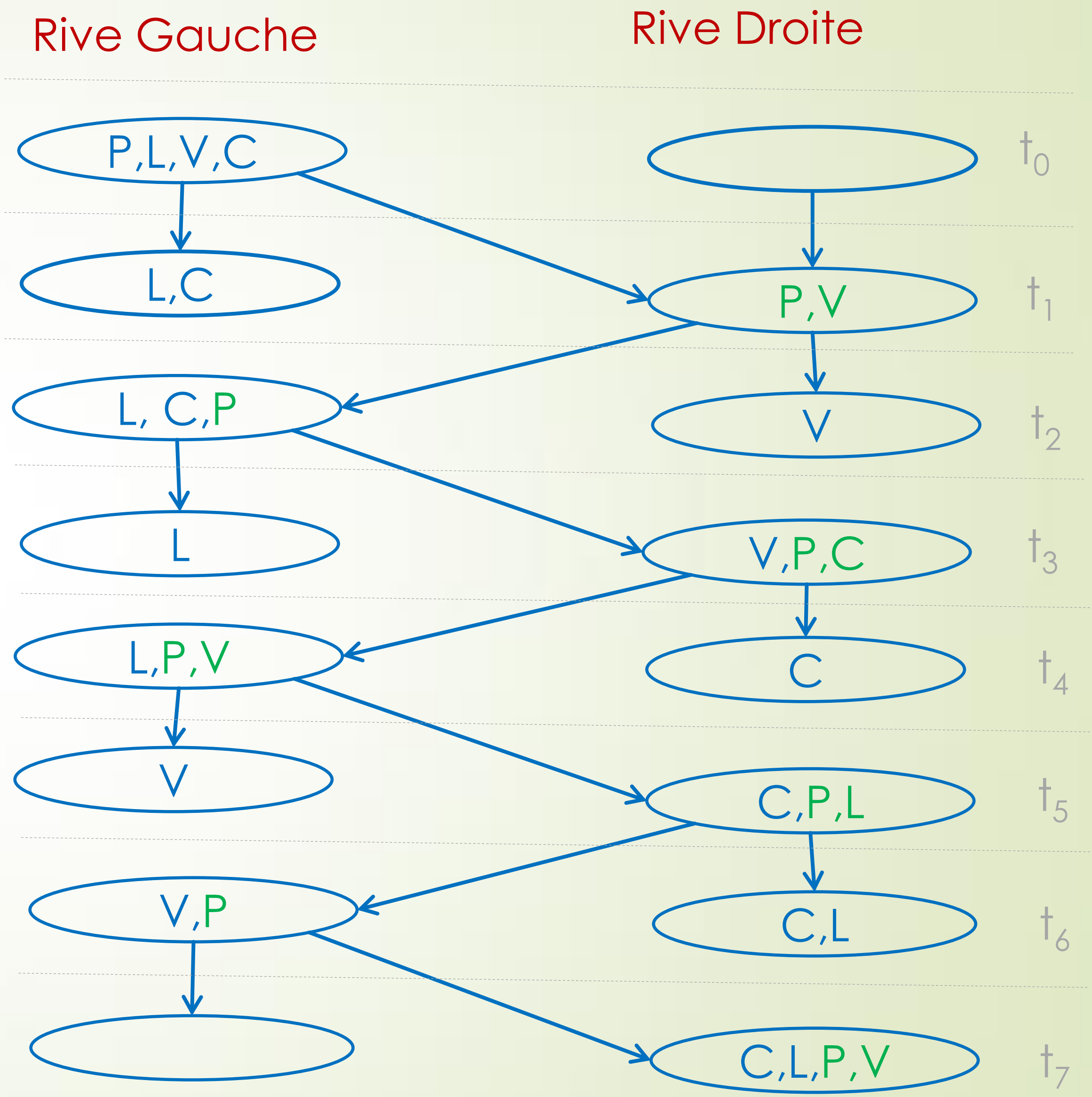
Exemple 1 (problème du passeur)

Un passeur (P) doit faire traverser une rivière à un loup (L), une chèvre (V) et un chou (C) dans une petite barque à deux places.

Pour des raisons évidentes, on ne peut laisser seules sur une rive le loup et la chèvre ou la chèvre et le chou.

Solution

- Un sommet représente l'état d'une rive à un instant donné.
- Un arc représente le passage d'une rive d'un état à un autre.

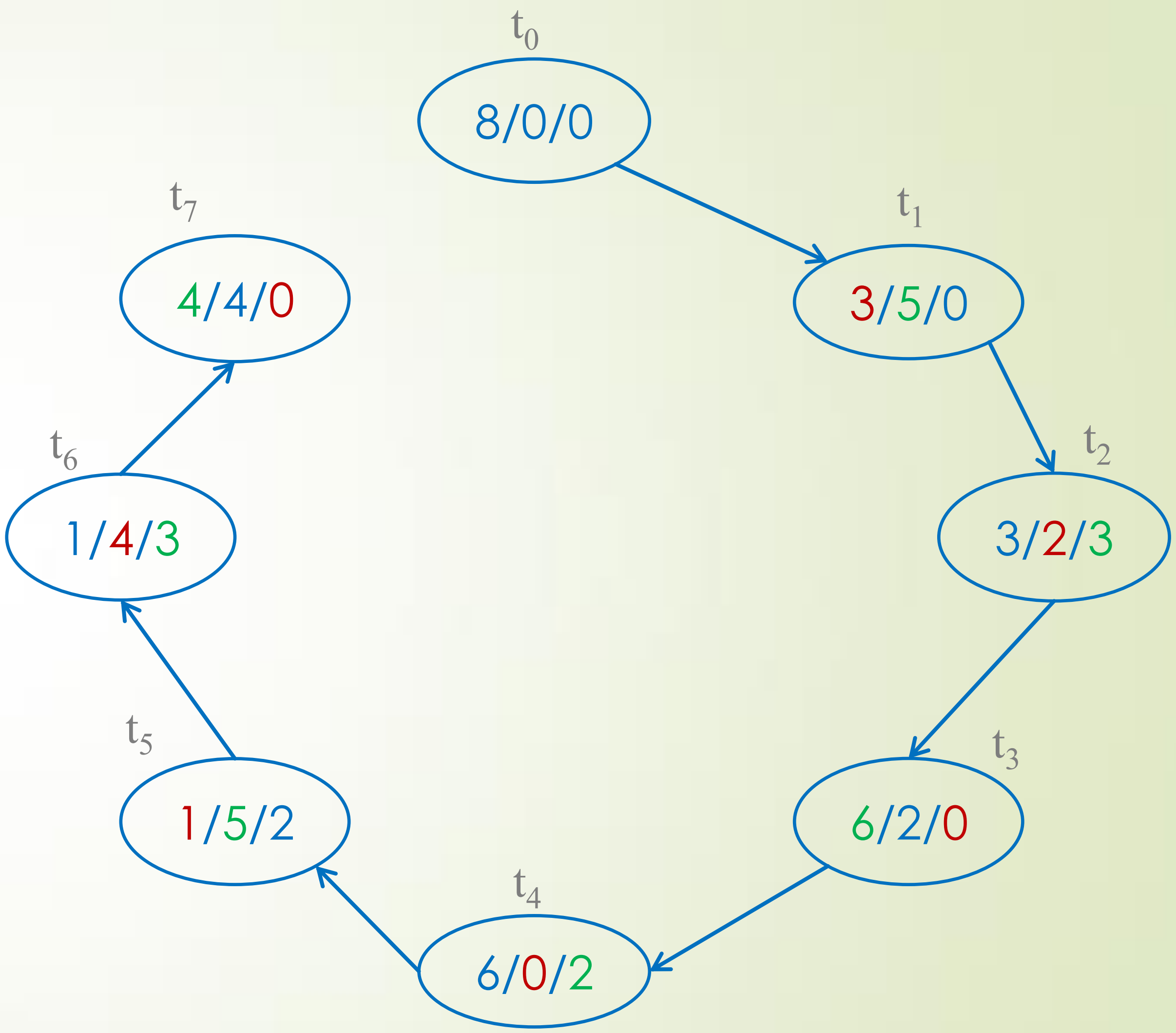


Exemple 2 (transvaser 3 récipients)

- Soient 3 récipients A, B et C de capacités 8, 5 et 3 litres respectivement. Le récipient A est rempli d'un liquide, les deux autres (B et C) sont vides.
- Comment utiliser les récipients B et C pour répartir ce liquide en deux quantités égales de 4 litres ? Utiliser un graphe pour représenter la solution de ce problème.

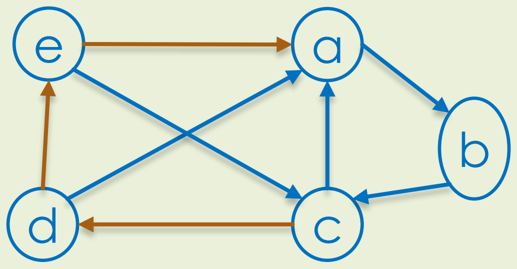
Solution

- Chaque sommet du graphe est un triplet (q_1,q_2,q_3) où q_i représente l'état du récipient i (i=A,B,C).
- A l'instant t_0, A est plein, B et C sont vide, on a donc le sommet (8/0/0)
- A l'instant t_1, on a versé 5 litres dans B, il reste 3 litres dans A, C est toujours vide, on a donc le sommet (3/5/0).
- A l'instant final t_7, on aura 4 litres dans A, 4 litres dans B et 0 litres dans C, d'où le sommet (4/4/0).



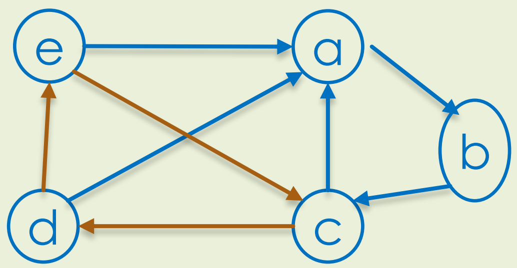
Les Chemins et les Circuits

- **Chemin :** c'est une séquence d'arcs qui se suivent



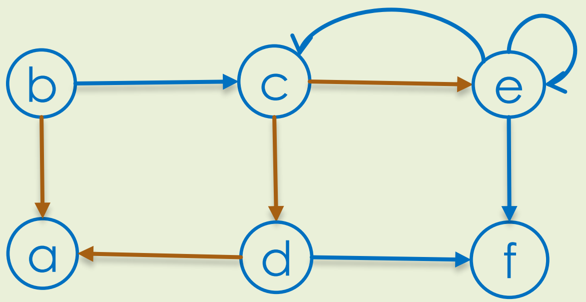
(c, d, e, a) est un chemin

- **Circuit :** c'est un chemin fermé



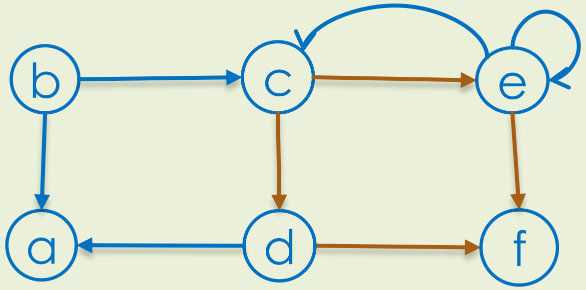
(e, c, d, e) est un circuit

- **Chaine :** c'est une séquence d'arcs, tel que 2 arcs consécutifs ont un sommet en commun ; l'orientation des arcs n'a pas d'importance.



(b,a,d,c,e) est une chaine

- **Cycle :** c'est une chaine fermée

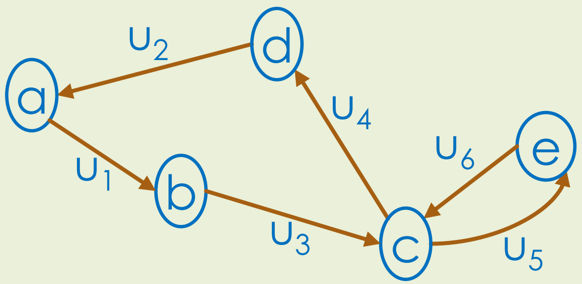


(c,e,f,d) est un cycle

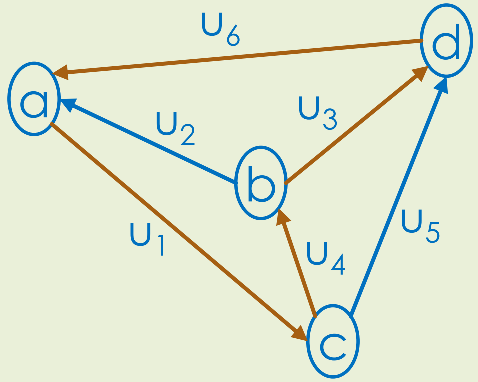
Graphe Heulerien et Hamiltonien

- Un **chemin simple** (resp. un circuit) ne passe qu'une seule fois par chacun de ses arcs.
- Un **chemin élémentaire** (resp. un circuit) ne passe qu'une seule fois par chacun de ses sommets.

- Un **graphe eulérien** possède un circuit simple de longueur $m = |U|$.
- Un **graphe hamiltonien** possède un circuit élémentaire de longueur $n = |X|$.

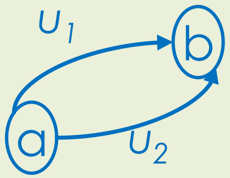


$(u_1,u_3,u_5,u_6,u_4,u_2)$ circuit eulérien

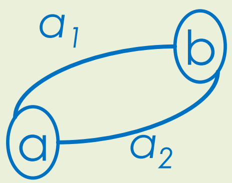


(u_1,u_4,u_3,u_6) circuit hamiltonien

- Deux arcs $u_1(x_1,y_1)$ et $u_2(x_2,y_2)$ sont parallèles si $x_1=x_2$ et $y_1=y_2$.

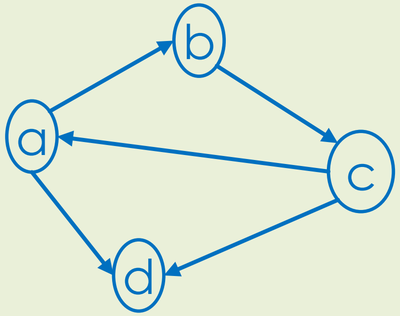


Les arcs u_1 et u_2 sont parallèles

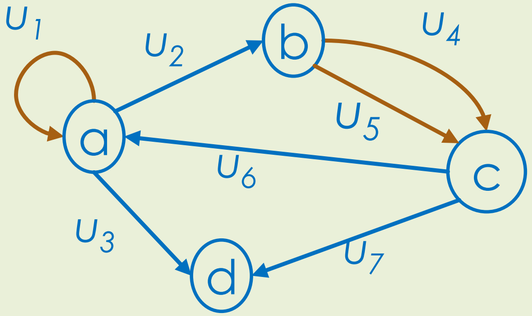


Les arêtes a_1 et a_2 sont parallèles

- Un **graphe simple** n'a pas de boucle, et n'a pas d'arcs (resp. arêtes) parallèles.



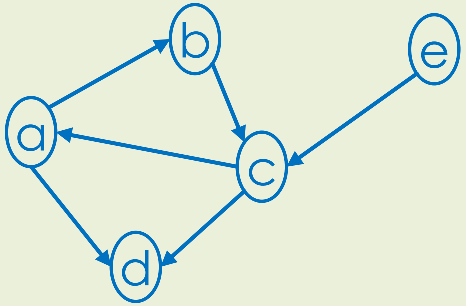
Graphe Simple



Graphe Non Simple
U_1 est une boucle
U_4 et u_5 sont parallèles

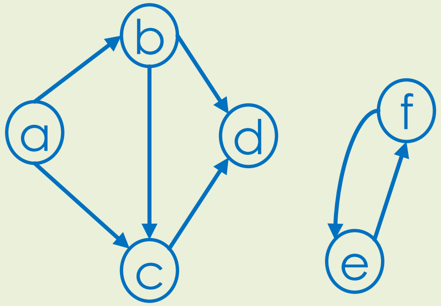
La Connexité

- Soit $G = (X, U)$ un graphe. Si $\forall\, x, y \in X$, il existe au moins une chaine reliant x à y, alors G est connexe, sinon il est non connexe.



Graphe connexe

Il existe une chaine entre tout couple de sommets

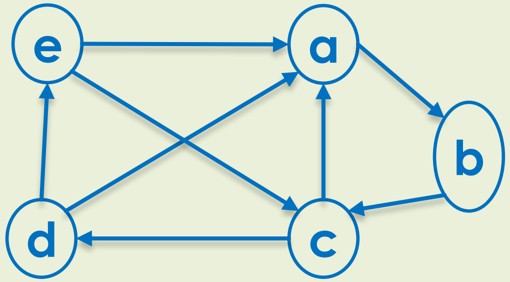


Graphe non connexe

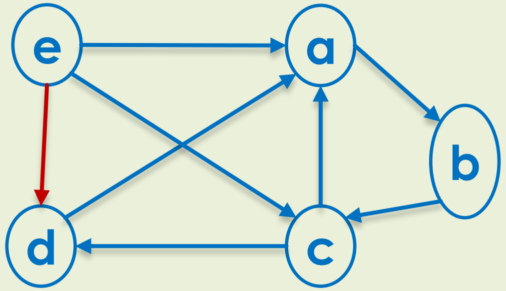
Par exemple, il y a pas de chaine entre b et f

- Ce graphe admet 2 composantes connexes {a,b,c,d} et {e,f}

- Un graphe est **fortement connexe** si et seulement si $\forall\, x, y \in X$, il existe un chemin les reliant.



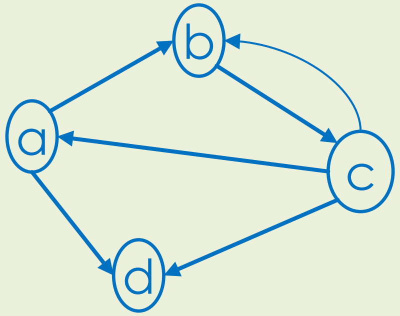
Graphe fortement Connexe



Graphe non fortement connexe

Par exemple, pas de chemin entre b et e

Représentation Informatique d'un Graphe



Cas de graphe non valué

Liste des arcs : c'est un tableau à 2 lignes et $m = |U|$ colonnes qui contient tous les arcs $u_i \in U$

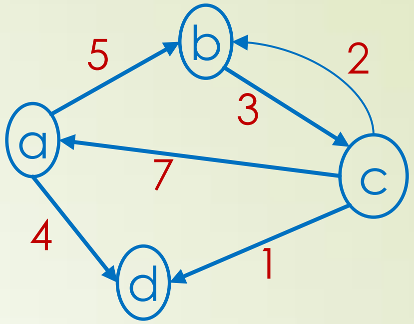
a	a	b	c	c	c
b	d	c	a	b	d

Matrice d'adjacence : c'est une matrice carré $M = [m_{ij}]$ où

$$m_{ij} = \begin{cases} 1 & \text{si} \quad (i,j) \in U \\ 0 & \quad \text{sinon} \end{cases}$$

- Eg.

$$M = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$



Cas de graphe valué

Liste des arcs : On ajoute une 3° ligne au tableau pour contenir les valeurs des arcs

a	a	b	c	c	c
b	d	c	a	b	d
5	4	3	7	2	1

Matrice d'adjacence : Soit v_{ij} la valeur de l'arc (i,j):

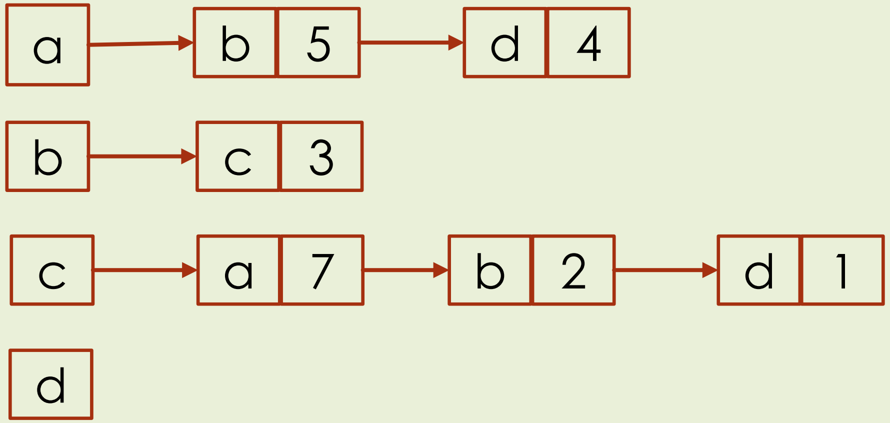
$$m_{ij} = \begin{cases} v_{ij} & \text{si} \quad (i,j) \in U \\ \infty & \quad \text{sinon} \end{cases}$$

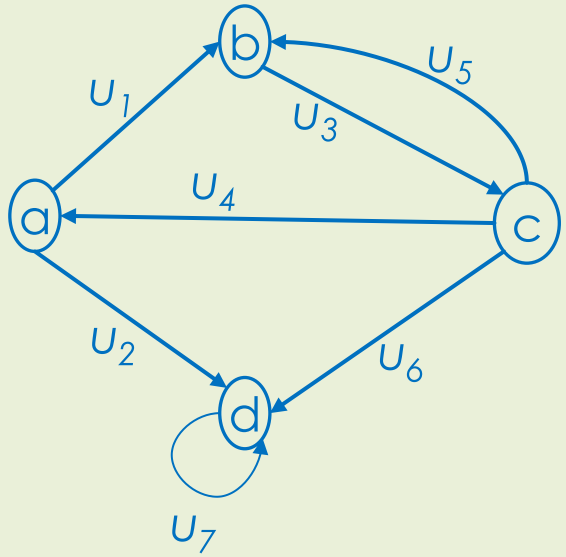
- Eg

$$M = \begin{bmatrix} \infty & 5 & \infty & 4 \\ \infty & \infty & 3 & \infty \\ 7 & 2 & \infty & 1 \\ \infty & \infty & \infty & \infty \end{bmatrix}$$

...

Liste chainée : pour chaque sommet $x \in X$, on lui associe la liste de ses successeurs et la valeur de chaque arc :





Matrice d'incidence sommet-arc $N = [n_{ij}]$:

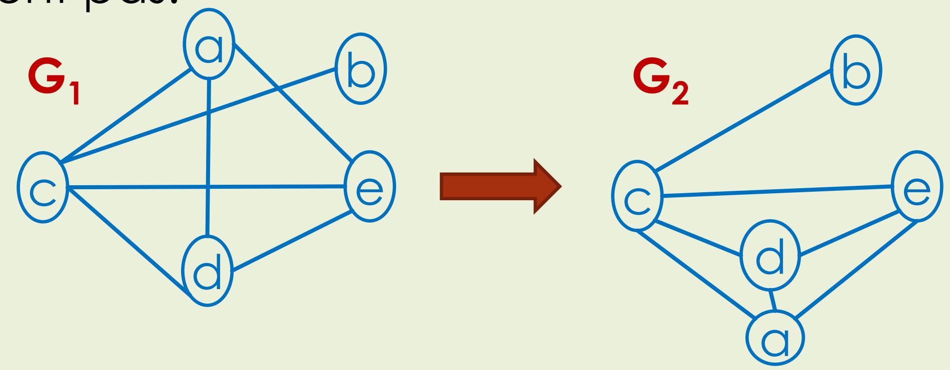
- Chaque ligne de la matrice représente un sommet *i*
- Chaque colonne de la matrice représente un arc $j = (x_j, y_j)$

$$n_{ij} = \begin{cases} 1 & \text{si} \quad i = x_j \\ -1 & \text{si} \quad i = y_j \\ 0 & \text{si} \quad i = x_j = y_j \quad (\text{cas d'une boucle}) \\ \infty & \text{sinon} \end{cases}$$

$$N = \begin{array}{c} \begin{matrix} U_1 & U_2 & U_3 & U_4 & U_5 & U_6 & U_7 \end{matrix} \\ \begin{bmatrix} 1 & 1 & \infty & -1 & \infty & \infty & \infty \\ -1 & \infty & 1 & \infty & -1 & \infty & \infty \\ \infty & \infty & -1 & 1 & 1 & 1 & \infty \\ \infty & -1 & \infty & \infty & \infty & -1 & 0 \end{bmatrix} \end{array} \begin{matrix} a \\ b \\ c \\ d \end{matrix}$$

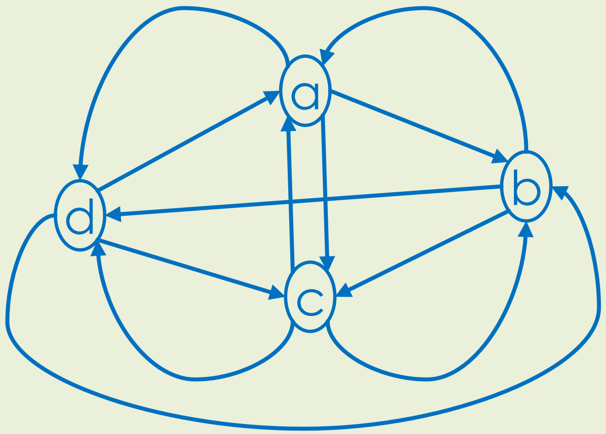
Types de Graphes

- G est un **graphe planaire** si ses arcs ne se croisent pas.



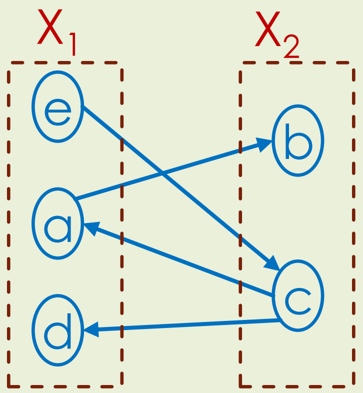
Graphe non planaire G1 converti en graphe planaire G2

- G est un **graphe complet** s'il y'a un arc entre tout couple de sommets du graphe



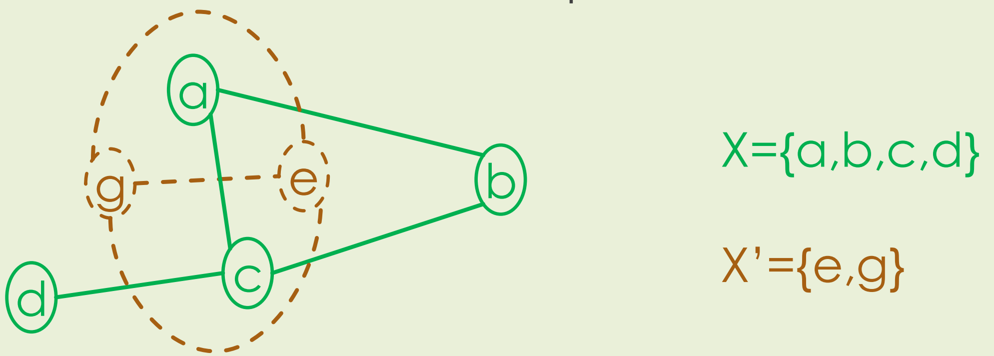
Graphe complet

- G=(X,U) est un **graphe biparti** si X est réparti en 2 sous-ensembles X_1 et X_2 disjoints et non vides, tel que tout arc de U a une extrimité dans X_1 et l'autre dans X_2



Graphe biparti

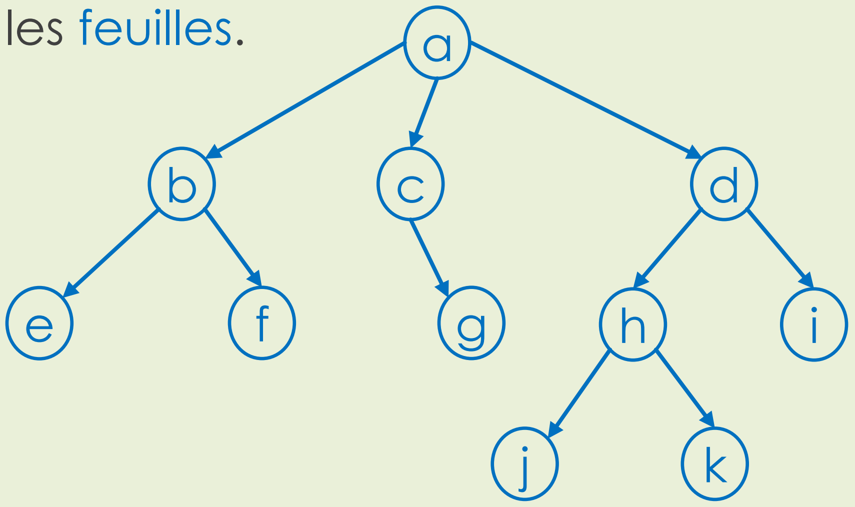
- Soit G=(X,U) un graphe. Le graphe G'=(X',U') est **le dual** de G si pour toute arête $u \in U$, il existe une seule arête $u' \in U$ tel que u' croise u.



Graphe G et son dual G'

- **Un arbre** est un graphe où chaque sommet ne possède qu'un seul prédécesseur, sauf le sommet racine qui n'a aucun prédécesseur.

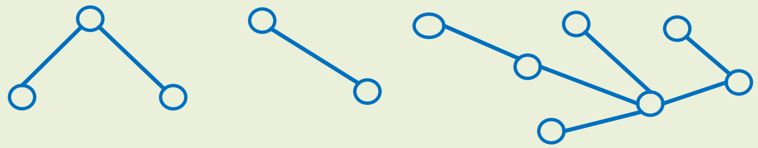
Les sommets qui n'ont pas de successeurs sont appelé les feuilles.



Un Arbre

{a} est la racine, {e,f,g,j,k,i} sont les feuilles

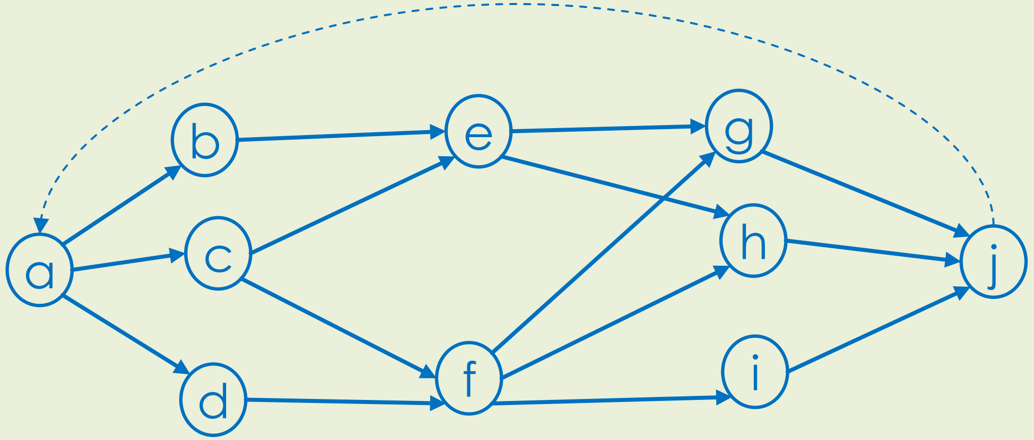
- **Une foret** est un graphe non connexe, dont chaque composante connexe est un arbre.



Une Foret

- **Un réseau** est un graphe fortement connexe, sans boucles.

Un réseau possède 2 sommets particuliers : sommet entrée et sommet sortie, liés par un arc fictif garantissant la forte connexité du graphe.

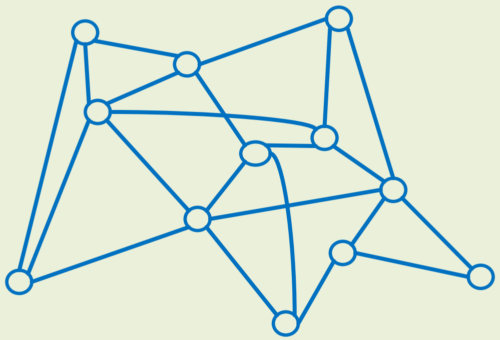


Un Réseau

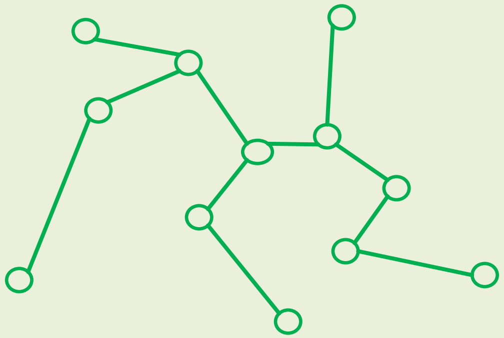
{a} : entrée du réseau, { j } : sortie du réseau

Applications des Graphes

- **L'arbre couvrant**, eg. optimiser la connexion des quartiers d'une ville par la fibre optique

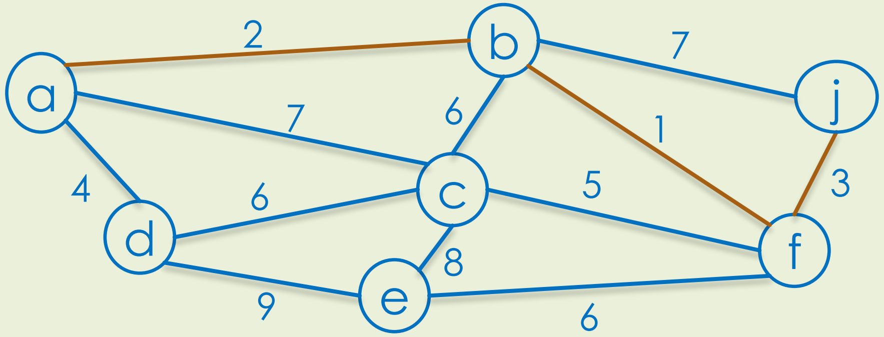


Les rues d'une ville



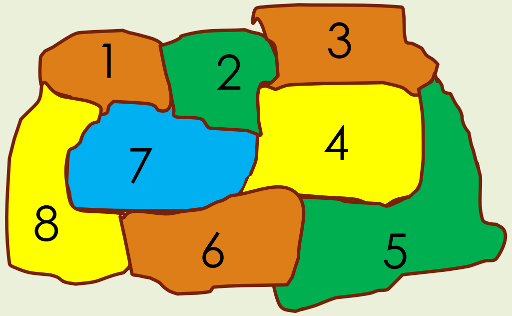
Passages de la fibre optique (arbre couvrant)

- **Plus court chemin** : routage de paquets de données transitant par un réseau (internet)

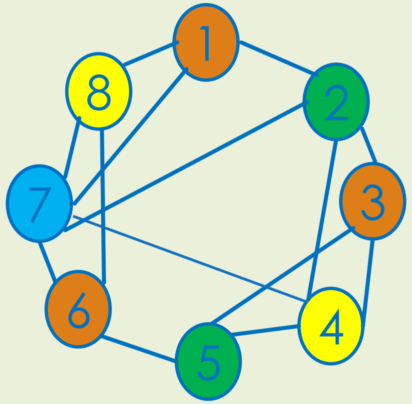


Le plus court chemin entre a et j est (a,b,f,j)

- **Problème de coloriage :** utiliser un nombre minimum de couleurs

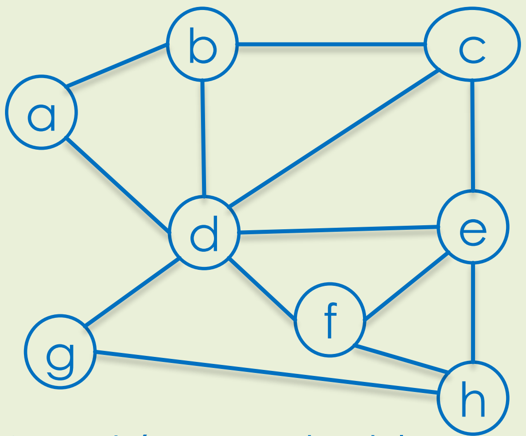


Coloriage d'une carte géographique

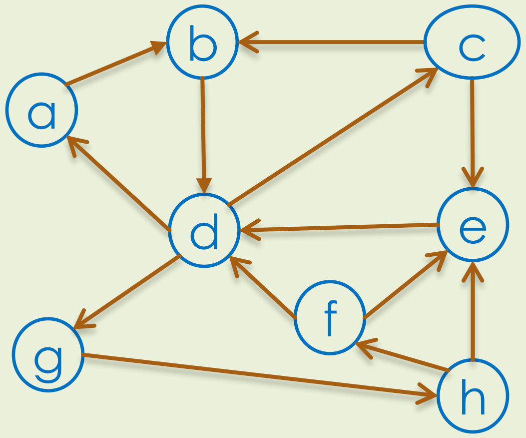


Modélisation par un graphe

- **Forte Connexité** : placer des sens uniques dans une ville en garantissant un chemin entre tous ses quartiers :

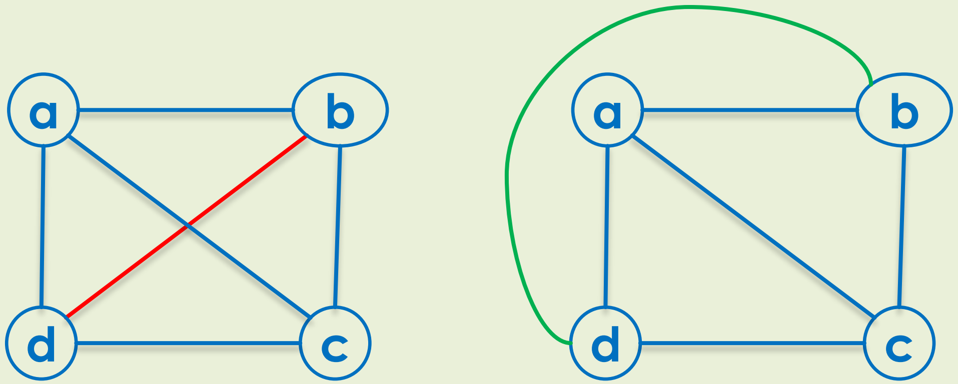


Avant (rues a double sens)



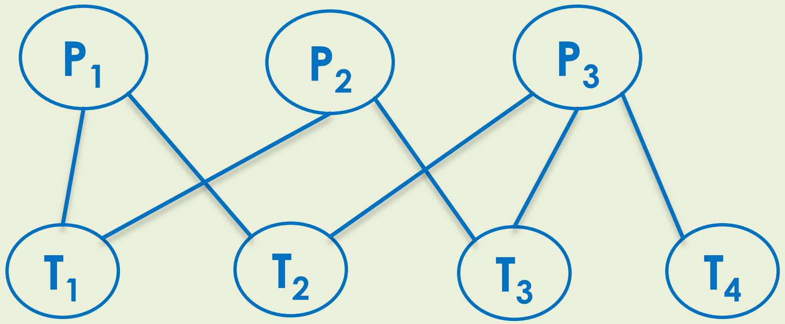
Après (rues à sens uniques)

- **Graphe planaire :** conception de circuit électronique intégré, les liens entre les composants ne doivent pas se croiser.

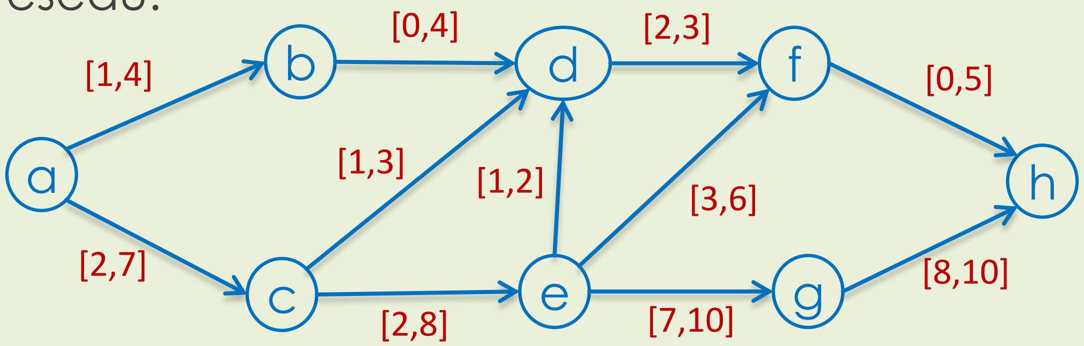


Conception erronée Conception correcte

- **Graphe biparti** : affectations des taches $T_1 \ldots T_m$ à des processus $P_1 \ldots P_k$



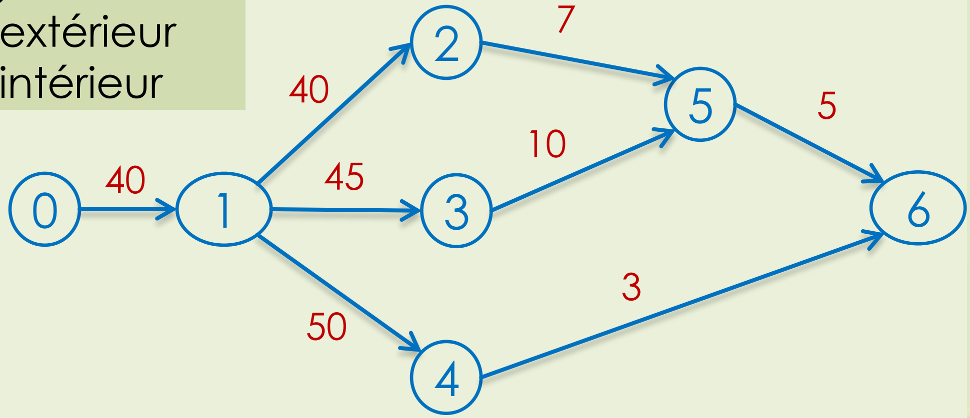
- **Réseau (problème de flot maximum)** : acheminer une quantité maximale entre A et B en respectant les contraintes [min,max] du réseau.



- **Ordonnancement des tâches** : Chemin critique

(1) fondations
(2) gros œuvres
(3) électricité
(4) chauffage central
(5) peinture extérieur
(6) peinture intérieur

- Sommet : fin d'une tâche
- Valeur d'un arc : durée d'une tâche



2 – Les Arbres et Arborescence en Théorie des Graphes

Université Alger 1, Dept Maths & Informatique

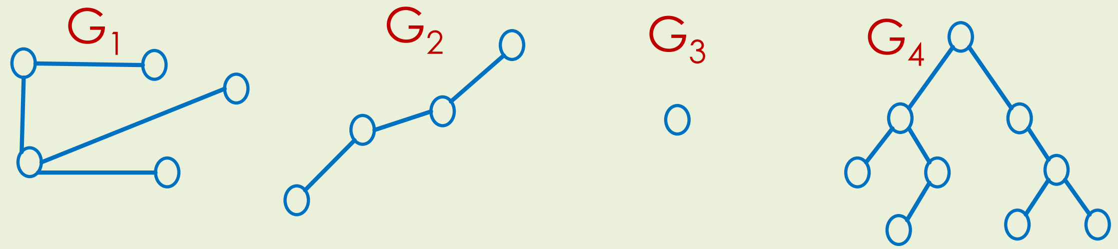
Dr. Fodil LAIB

Février 2017

Un Arbre

- Les arbres sont des graphes particuliers, très utilisés en algorithmique et en informatique.
- Soit G=(X,U) un graphe d'ordre $n = |X|$ et de taille $m = |U|$.
- G est arbre s'il est connexe et acyclique (c.-à-d. sans cycle).
- Un arbre vérifie les propriétés suivantes :

1. Le nombre d'arcs $m = n - 1$
2. Si on supprime un arc, le graphe G sera déconnecté (ne sera plus connexe).
3. Si on ajoute un arc, G devient cyclique.



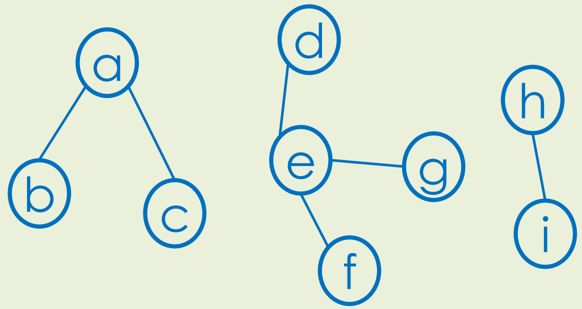
Exemples d'arbres

- On appelle **feuille** d'un arbre un sommet adjacent à une seule arête.
- **Une forêt** est un graphe, dont chaque composant connexe est un arbre. Autrement dit une forêt est un graphe acyclique.

Sur ordinateur, une forêt peut être représentée à l'aide de 2 tableaux :

- NA : nombre d'arêtes de chaque arbre de la forêt
- AR : les arêtes des arbres

Exemple : Soit la foret suivante :



On peut la sauvegarder sur ordinateur sous forme de :

NA

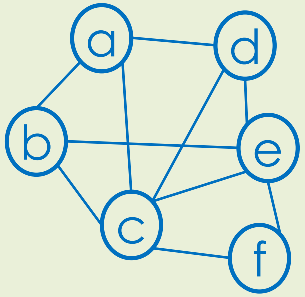
2	3	1

AR

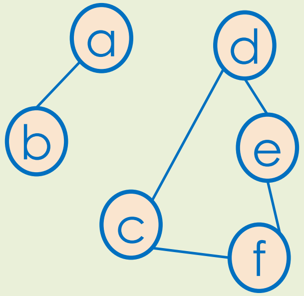
a	a	e	e	e	h
b	c	d	f	g	I

Arbre Couvrant

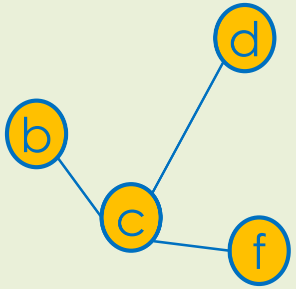
- Lorsque on élimine quelques arêtes d'un graphe G, on obtient un **graphe partiel** de G.
- Lorsque on élimine quelques sommets de G et les arêtes associées, on obtient un **sous graphe** de G.



Graphe Initial

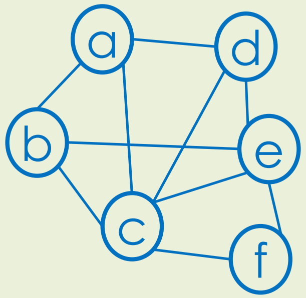


Graphe Partiel

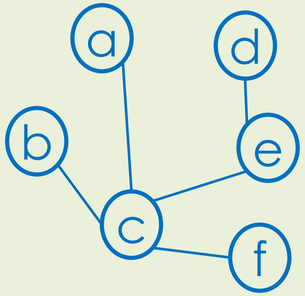


Sous Graphe

- Un arbre $T=(X_T,U_T)$ est un **arbre couvrant** du graphe $G=(X,U)$ si T est un graphe partiel de G c-à-d $X_T = X$ et $U_T \subset U$

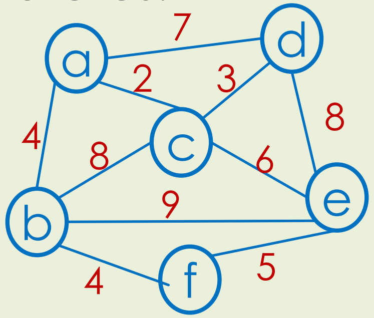


Graphe G

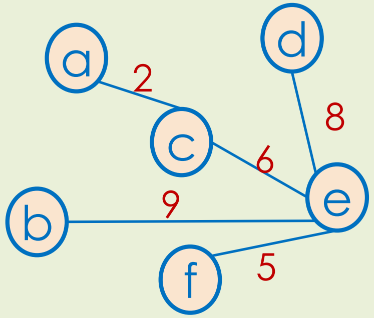


Arbre Couvrant de G

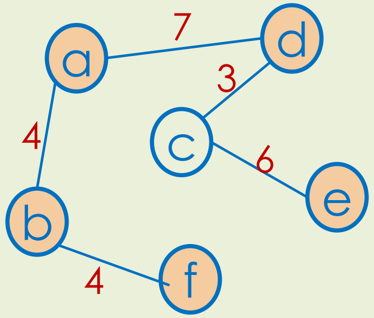
- Soit $G=(X,U,W)$ un graphe valué où $W=\{w_1,\ldots w_m\}$ sont les poids des arêtes.



Graphe Valué



Arbre couvrant de poids w=30

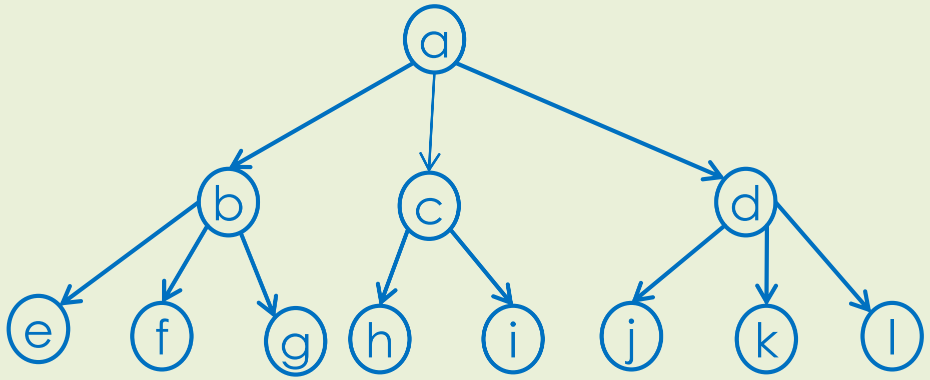


Arbre couvrant de poids w=24

- Un arbre couvrant de poids minimum n'est pas forcement unique.
- Un arbre couvrant de poids min est unique si pour chaque sommet du graphe G, ses arêtes ont des poids distincts.

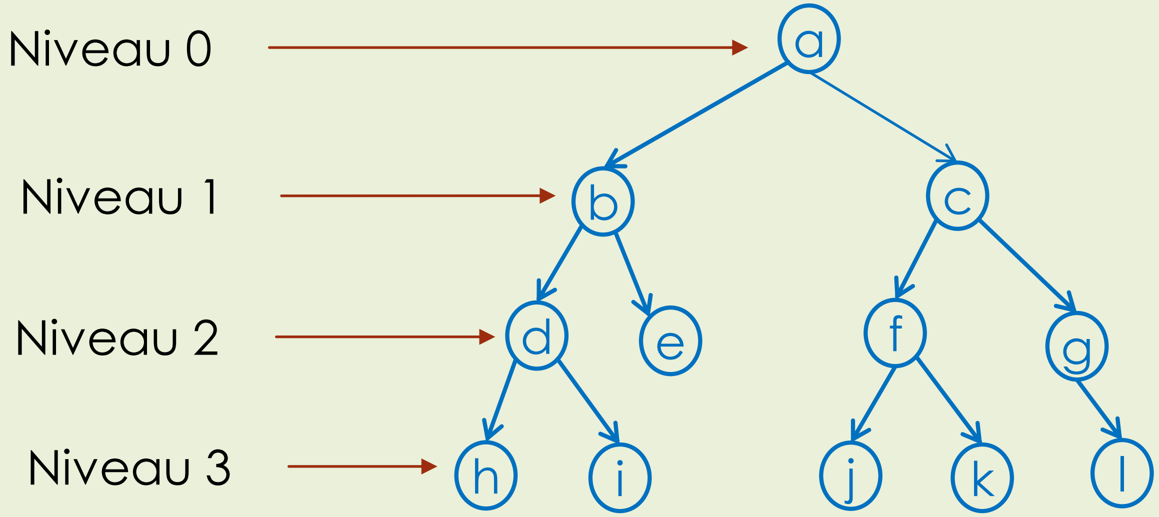
Une Arborescence

- Dans le cas d'un **graphe orienté**, si un arbre possède un seul sommet sans prédécesseurs, ce sommet est appelé **racine**, et l'arbre est dit **enraciné**, ou bien une **arborescence**.



- Une chaine reliant la racine d'un arbre à une feuille est appelé une **branche**.
- Dans une arborescence, il existe un chemin unique de la racine vers tous les autres sommets.

- Une **arborescence k-aire** est un arbre orienté dont chaque sommet a au plus k successeurs : ses **fils**.
- Quand k=2, l'arbre est **binaire**.



- Dans une arborescence, un **niveau** est un ensemble de nœuds qui sont équidistants de la racine. La racine se trouve au niveau 0.
- La **hauteur** (ou **profondeur**) d'un arbre est le nombre d'arcs sur un chemin de longueur maximale.

Représentation Informatique des Arbres :

Sur ordinateur, un arbre peut être représenté de différentes manières selon le contexte :

- Représentation générale d'un graphe (matrice d'adjacence, liste d'arcs, etc.)
- Représentation récursive des arbres binaires:
    1. Un arbre est un pointeur sur un sommet
    2. Un sommet est une valeur et 2 pointeurs :
        1. Un sur l'arbre fils de gauche
        2. Un sur l'arbre fils de droite

1. Représentation récursive des arbres k-aires:
    1. Un arbre est un pointeur sur un sommet
    2. Un sommet a une valeur et 2 pointeurs :
        1. Un sur l'alrbre appelé premier fils
        2. Un sur la liste des freres

Quelques Applications des Arbres Couvrants :

- Connecter économiquement un ensemble de stations : terminaux, téléphones, usines, etc.
- Produire un graphe partiel permettant d'analyser rapidement le grand graphe original
- Conception des algorithmes de routage
- Trouver des solutions rapides à des problèmes complexes tels que le traveling salesmen et Steiner tree.
- Conception de réseaux

Théorème de Cayley (1889) : Dans un graphe complet à n sommets, il existe n^{n-2} arbres couvrants.

Algorithme de Kruskal (1956)

L'algorithme de Kruskal retrouve l'arbre, où la foret d'arbres, de poids minimum dans un graphe G=(X,U) valué.

Procedure Kruskal (Arbre couvrant de poids minimum)
Input : G = (X,U,W) // le graphe
Output : A // l'arbre couvrant

- Ordonner la liste des arêtes dans l'ordre croissant, soit L la liste obtenue : L={u_1,u_2,…,u_m}
- Poser A = ∅, k = 1

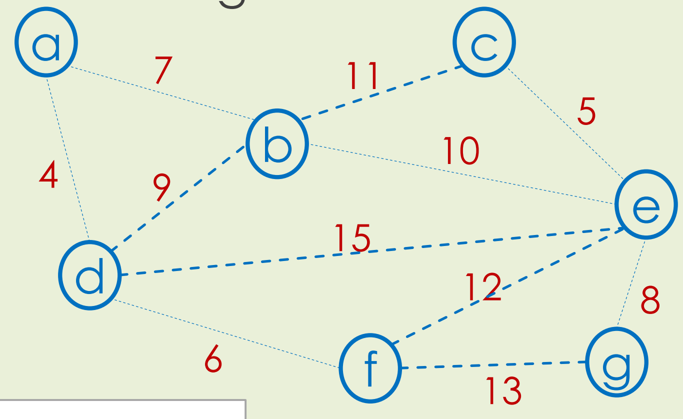
Tant que $|A| < |X|$ et $k \leq m$ faire
    Si {A,u_k} ne forme pas de cycle alors
        ajouter u_k à A
    Fin Si
    K = k +1
Fin tant que

Procédure : Tester si l'ajout de l'arc u à l'arbre T forme un circuit

- Introduire l'arbre T=(X_T,U_T) et l'arc u=(x_u,y_u)

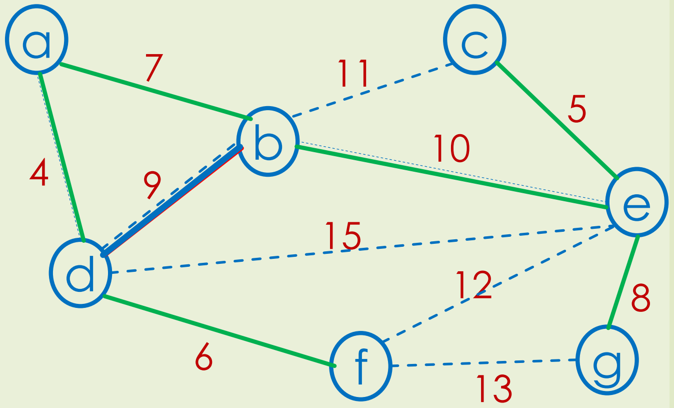
Si $x_u, y_u \in X_T$ alors
    l'ajout de u à T formera un circuit
Sinon
    l'ajout de u à T ne formera pas de circuit.

Exemple 1 : Retrouver l'arbre de poids min de ce graphe à l'aide de l'algorithme de Kruskal :



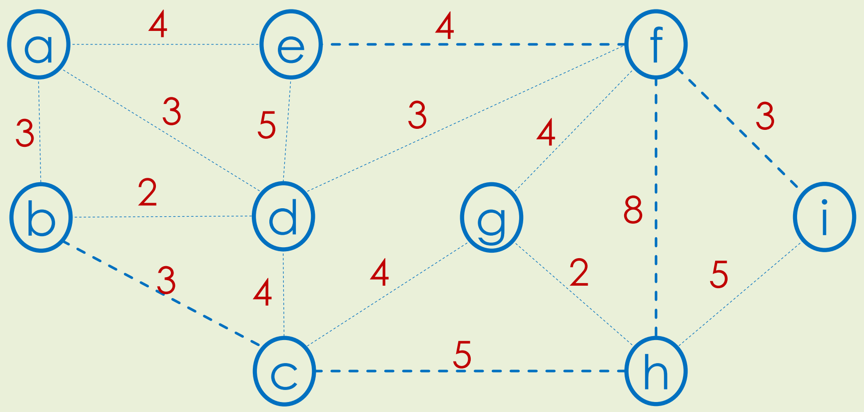
Solution

a	d	4	Pas de circuit
c	e	5	Pas de circuit
d	f	6	Pas de circuit
a	b	7	Pas de circuit
e	g	8	Pas de circuit
b	d	9	Circuit
b	e	10	Pas de circuit
b	c	11	Nombre d'arêtes = 6, soit |X|-1, donc on arête ici.
e	f	12	
f	g	13	
d	e	15	

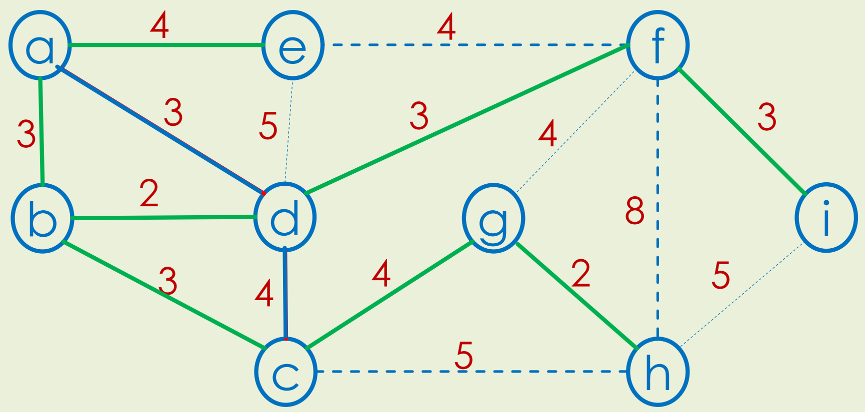


L'arbre couvrant de poids minimal est A={ab, ad, be, ce, df, eg}, son poids est w=40.

Exemple 2 : Trouver l'arbre de poids min



Solution



b	d	2	Pas de circuit
g	h	2	Pas de circuit
a	b	3	Pas de circuit
a	d	3	Circuit
b	c	3	Pas de circuit
d	f	3	Pas de circuit
f	i	3	Pas de circuit
a	e	4	Pas de circuit
c	d	4	Circuit
c	g	4	Pas de circuit
e	f	4	Nombre d'arêtes = 8, soit |X|-1, donc on arête ici.
f	g	4	
c	h	5	
d	e	5	
h	i	5	
f	h	8	

L'arbre couvrant de poids minimal est A={bd, gh, ab, bc, df, fi, ae, cg}, son poids est w=2+2+3+3+3+3+4+4 =24.

Algorithme de Boruvka (1926)

- Appelé aussi algorithme de Sollin
- Cet algorithme permet de retrouver un arbre couvrant de poids minimal.

Principe : Commencer par une forêt dont chaque composante est un sommet du graphe. Lier les composantes par des arêtes de poids minimums jusqu'à ce que la forêt se transforme en arbre couvrant de poids minimum.

Procedure Boruvka (Arbre couvrant de poids minimum)
Input : G = (X,U,W) // le graphe
Output : F // la foret (arbre) couvrante

- Poser F = X

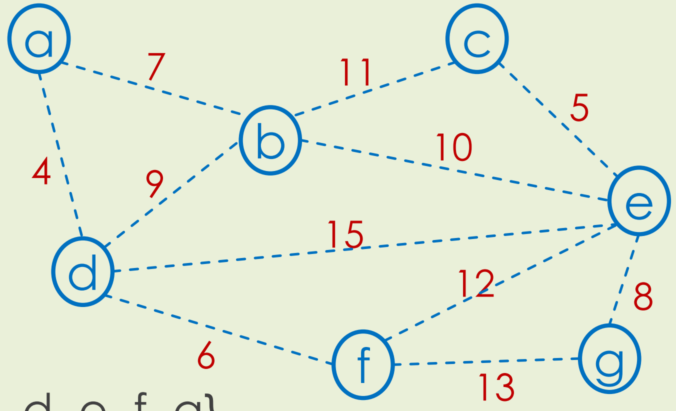
Tant que |F| > 1 faire
- L = ∅

Pour tout composant C∈F faire
- Trouver une arête u_p∈U de poids minimum liant C vers une autre composante C'∈F
- Ajouter u_p à L.

Fin Pour tout
- Ajouter L à F

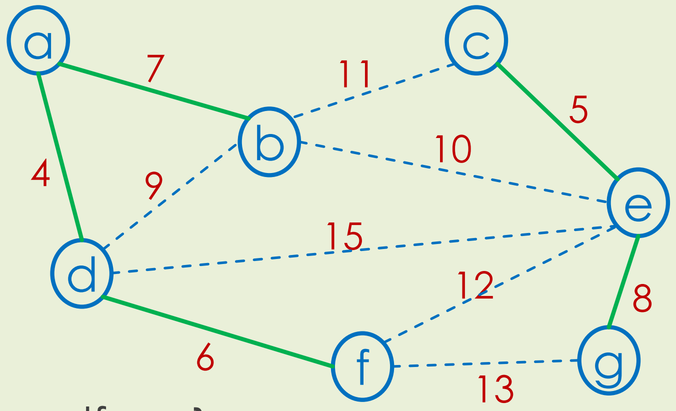
Fin tant que

Exemple



Initialisation : F = {a, b, c, d, e, f, g}

Itération 1: |F| = 7 > 1

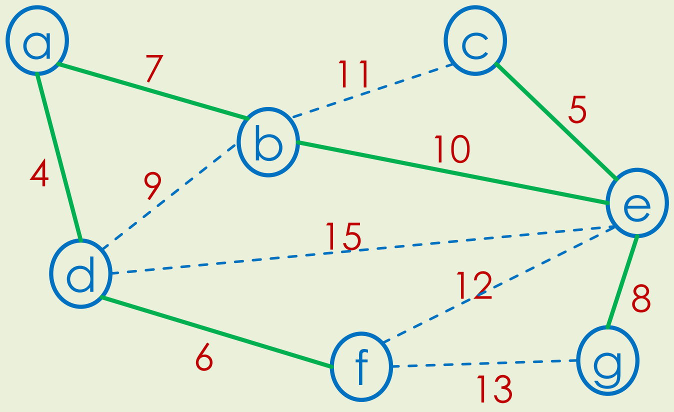


L={ab, ab, ce, df, eg} .

Ajoutons L à F, on obtient

F = { {ad, ab, df}, {ce, eg} }

[Itération 2](#) : | F | = 2 > 1



L = {be}.

Ajoutons L à F, on obtient

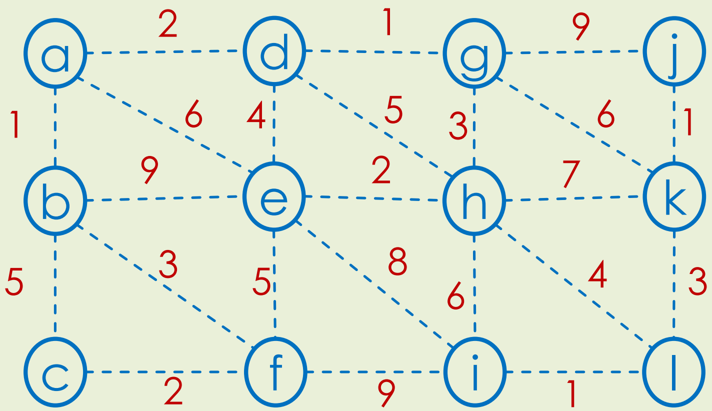
F = { {ad, ab, df,ce, eg, be} }

[Itération 3](#) : | F | = 1. Fin

F est un arbre couvrant de poids minimum.

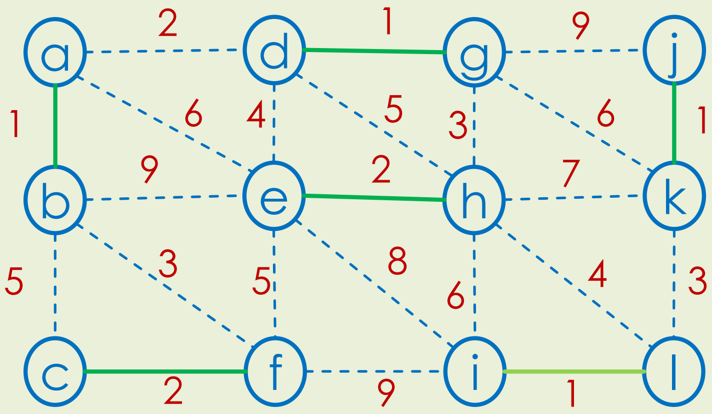
Son poids est = 7+4+10+5+6+8 = 40.

Example 2 : Retrouver l'arbre de poids minimum à l'aide de l'agorithme de Boruvka



[Initialisation](#) : F = { a, b, c, d, e, f, g, h, i, j, k, l }

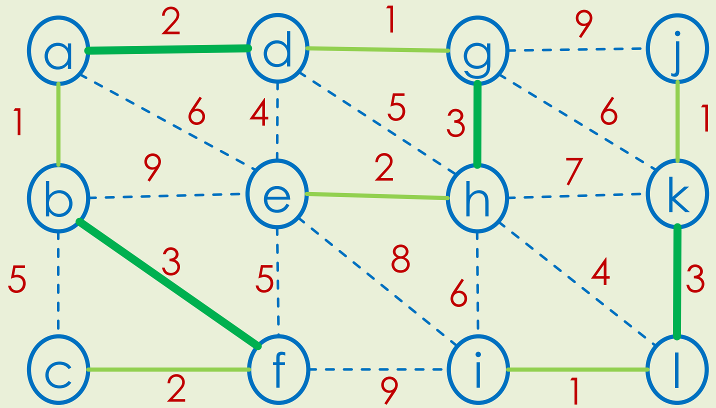
[Itération 1](#) : | F | = 12 > 1



L = {ab, cf, dg, eh, il, jk}.

Ajoutons L à F, on obtient

[Itération 2](#): |F| = 6 > 1

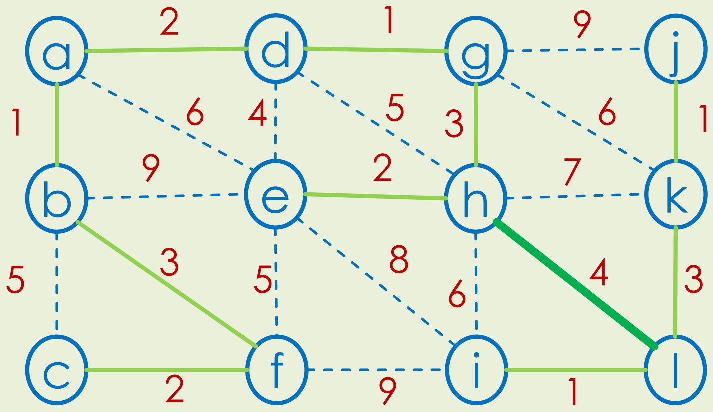


L = {ad, bf, gh, kl}.

Ajoutons L à F, on obtient

F = { {ab, ad, bf, cf, dg, gh, eh}, { jk, kl, il} }

[Itération 3](#) : |F| = 2

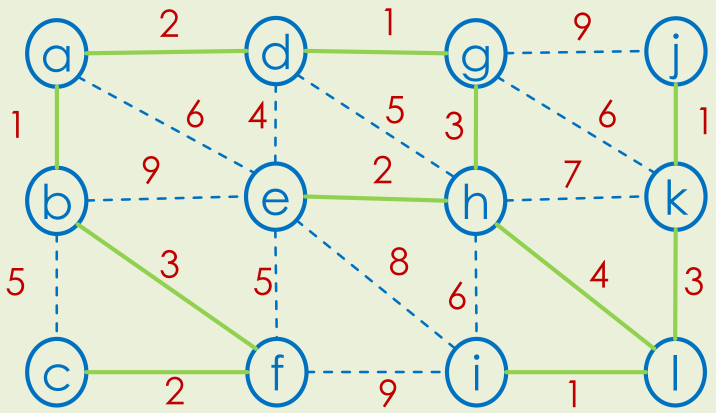


L = {hl}.

Ajoutons L à F, on obtient :

F={{ab, ad, bf, cf, dg, gh, eh, hl, jk, kl, il} }

[Itération 4](#) : |F| = 1 ==> arbre de poids minimum



Poids min = (2+1+2+2+1)+(1+3+1+3)+(3+4)

=8+8+7 =23

La Complexité d'un Algorithme

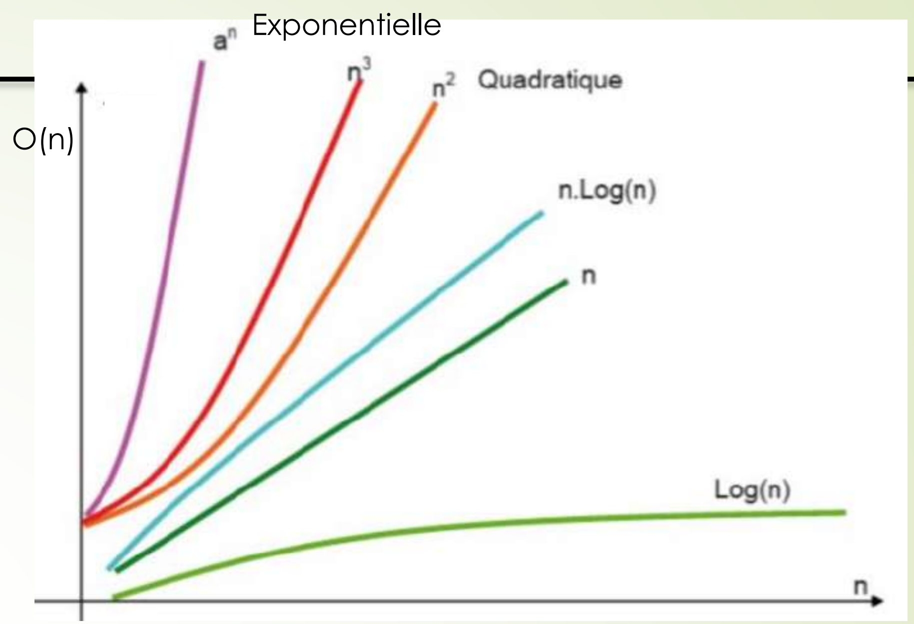
- **Complexité temporelle** : c'est le nombre d'instructions nécessaires à l'exécution d'un algorithme. On note c(n)
- **Complexité spatiale** : c'est l'espace mémoire nécessaire au stockage des données de l'algorithme.

Exemple : Rechercher le minimum d'un tableau de n entiers.

- Dans le pire des cas, il faut n affectations et $n-1$ tests, soit $c(n) = n + (n-1) = 2n - 1$.
- Cet algorithme est de complexité linéaire, on note cette complexité par O(n).

Quelques classes de Complexités :

c(n)	Notation	Complexité
$c(n) = c$	O(1)	Constante
$c(n) = 2n + 1$	O(n)	Linéaire
$c(n) = \log n$	O(log n)	Logarithmique
$c(n) = n^3 + 2n + 4$	O(n^3)	Polynomiale
$c(n) = a^n + 3n^2 - 5$	O(a^n)	Exponentielle



Exemple : Considérons un PC pour lequel durée exécution d'une instruction élémentaire = 0,001 s

Complexité de l'Algorithme	Temps D'Exécution sur Machine	
	n =10	n = 1000
$\log n$	0,001s	0,003 s
$\sqrt{n}$	0,003 s	0,032 s
n	0,01s	1 s
n^2	0,1s	17 min
n^3	1 s	12 jours
n^4	10 s	32 ans
2^n	1,024 s	3,4 *10^{287} millénaires

Complexité de l'algorithme Kruskal :

- Le tri des arêtes est la partie qui consomme le plus de temps dans cet algorithme.
- Avec un algorithme de tri rapide, le tri des arêtes peut se faire en m log m opérations pour les m arêtes du graphe.
- Donc, la complexité de l'algorithme de Kruskal est **O(m log m)**.

Complexité de l'algorithme Bovurka :

- La boucle externe de l'algorithme prend O(log n) itérations,
- En incluant la boucle interne , l'algorithme prend O(m log n).

Algorithme de Prim (1957)

Procédure Prim (Arbre couvrant de poids minimum)

Input : G=(X,U,W) // le graphe valué
Output : A // l'arbre couvrant

On pose $A = \emptyset$, $S = \{x_1\}$ // x_1 premier sommet de X

Tant que $|S| < |X|$ faire

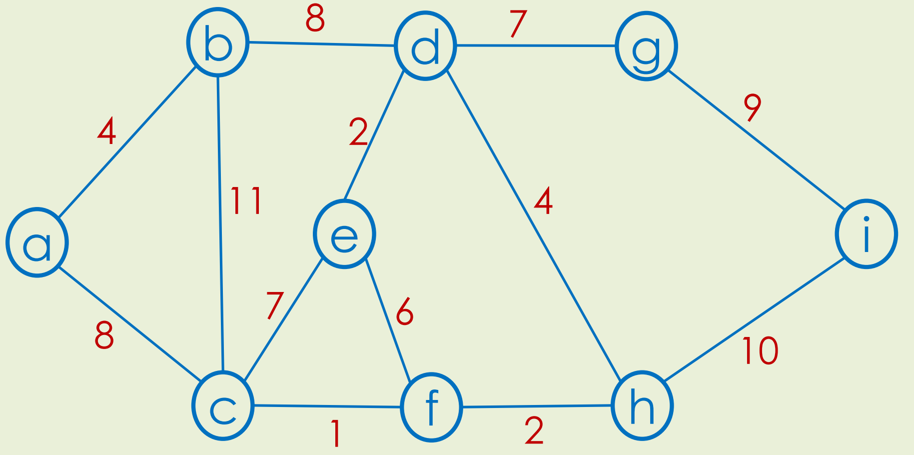
- Sélectionner l'arête $u = (x, y) \in U$ de poids minimum telle que $x \in S$ et $y \in X/S$
- $S = S \cup \{y\}$
- $A = A \cup \{u\}$

Fin tant que

Compléxité : O(| X | ²)

Mais on peut améliorer cette complexité avec certaines techniques algorithmiques.

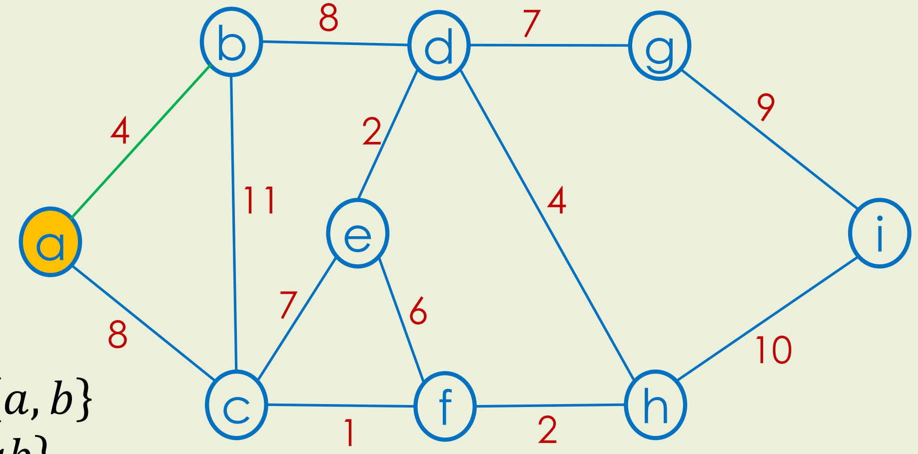
Exemple : Utiliser l'algorithme de Prim pour retrouver un arbre couvrant de poids minimum du graphe suivant :



Solution

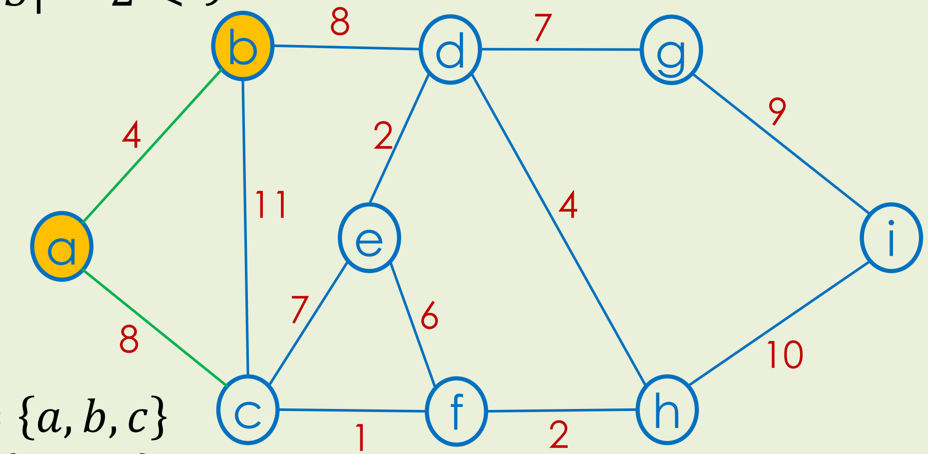
Initialisation : $A = \emptyset$, $S = \{a\}$

Itération 1 : $|S| = 1 < 9 = |X|$



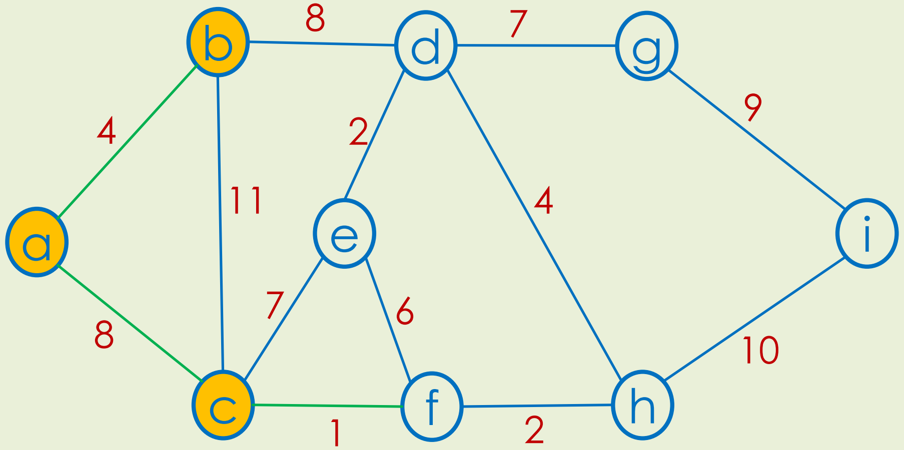
$u = (a, b)$
$S = S \cup \{b\} = \{a, b\}$
$A = A \cup \{u\} = \{ab\}$

[Itération 2 :](#) $|S| = 2 < 9$



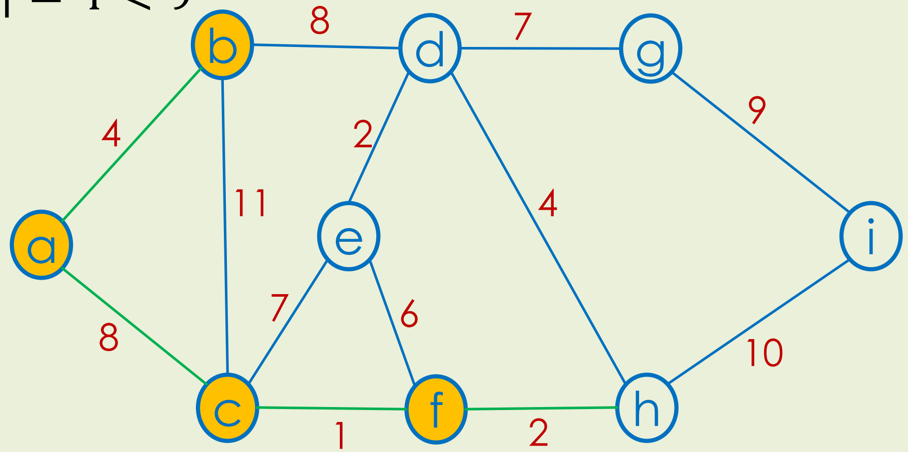
$u = (a, c)$
$S = S \cup \{c\} = \{a, b, c\}$
$A = A \cup \{u\}$=$\{ab, ac\}$

[Itération 3 :](#) $|S| = 3 < 9$



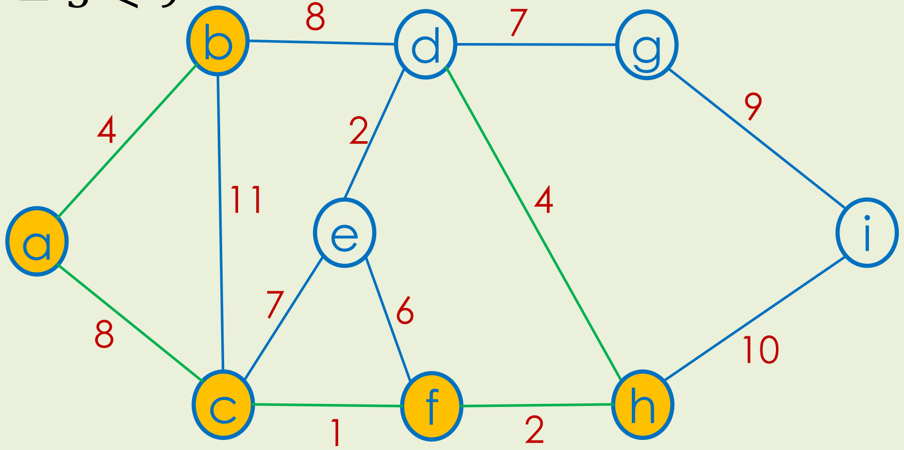
$u = (c, f)$
$S = S \cup \{c\} = \{a, b, c, f\}$
$A = A \cup \{u\}$=$\{ab, ac, cf\}$

[Itération 4 :](#) $|S| = 4 < 9$



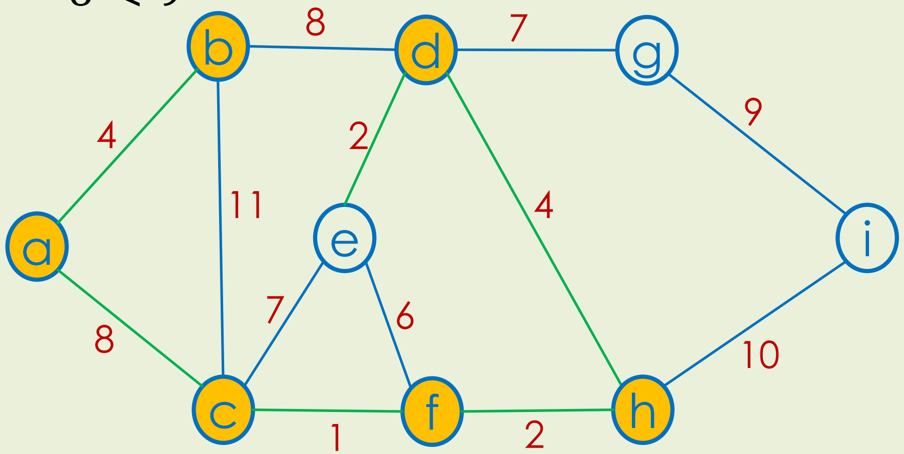
$u = (f, h)$
$S = S \cup \{c\} = \{a, b, c, f, h\}$
$A = A \cup \{u\}$=$\{ab, ac, cf, fh\}$

[Itération 5 :](#) $|S| = 5 < 9$



$u = (h, d)$
$S = S \cup \{c\} = \{a, b, c, f, h, d\}$
$A = A \cup \{u\}$=$\{ab, ac, cf, fh, dh\}$

Itération 6 : $|S| = 6 < 9$

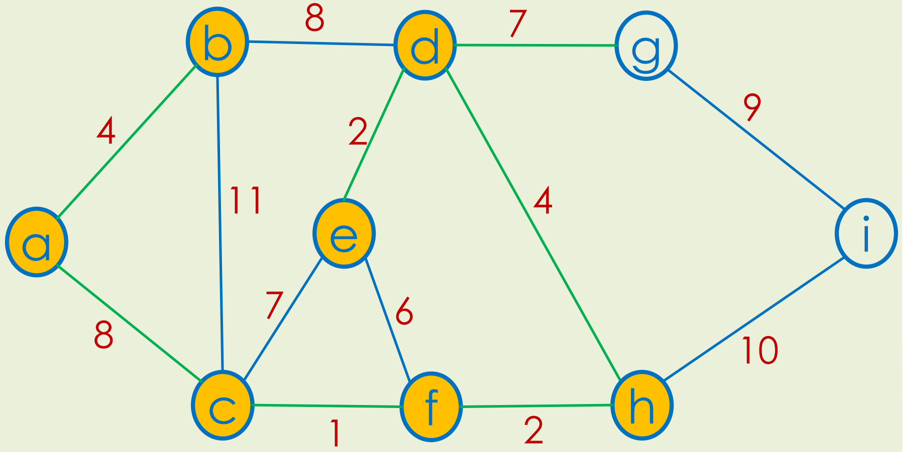


$u = (d, e)$

$S = S \cup \{c\} = \{a, b, c, f, h, d, e\}$

$A = A \cup \{u\}=\{ab, ac, cf, fh, dh, de\}$

Itération 7 : $|S| = 7 < 9$

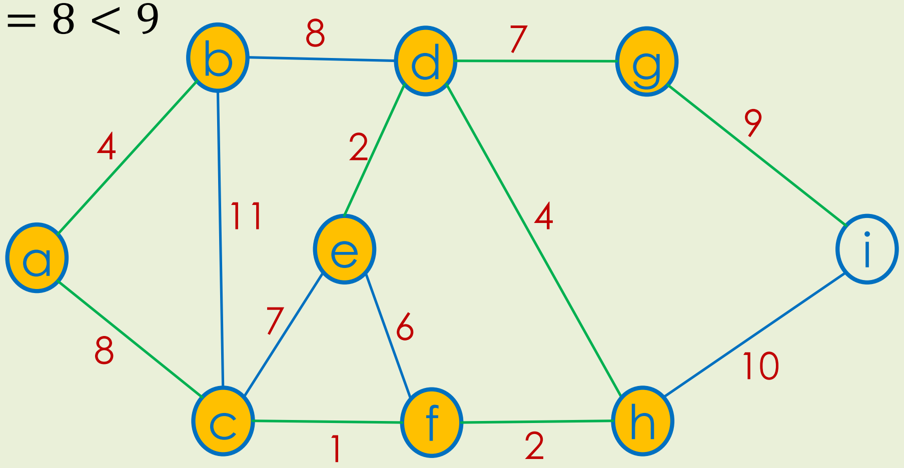


$u = (d, g)$

$S = S \cup \{c\} = \{a, b, c, f, h, d, e, g\}$

$A = A \cup \{u\}=\{ab, ac, cf, fh, dh, de, dg\}$

Itération 8 : $|S| = 8 < 9$

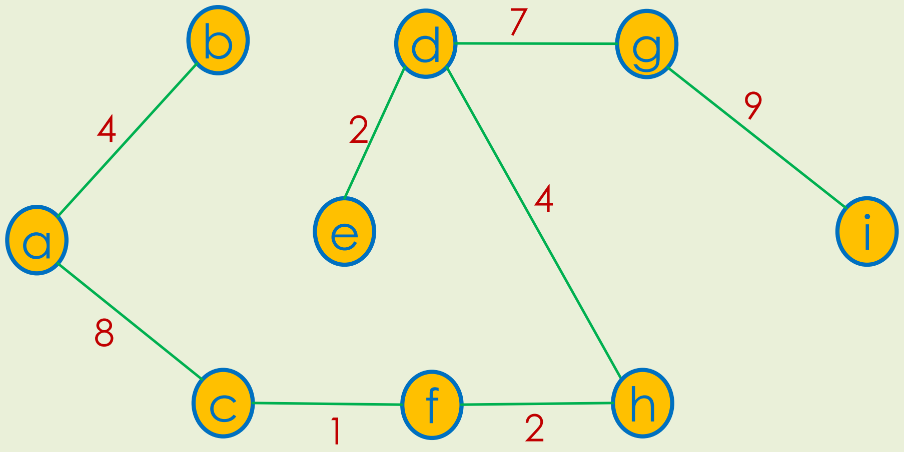


$u = (g, i)$

$S = S \cup \{c\} = \{a, b, c, f, h, d, e, g, i\}$

$A = A \cup \{u\}=\{ab, ac, cf, fh, dh, de, dg, gi\}$

Itération 9 : $|S| = 9 \nless 9 \;\Rightarrow fin$



Le poids de cet arbre couvrant est

w = 4 + 8 + 1 + 2 + 4 + 2 + 7 + 9 = 37

Code C pour l’algorithme de Prim


```c
#include <stdio.h>
#include <stdlib.h>

#define Max 20
#define Inf 99999

double G[Max][Max], poids;
int A[Max][2], S[Max], T[Max];
int n_G, n_A, n_S, n_T;
int a_min, b_min, indice;


int charger_fichier_donnees();
void afficher_matrice_adjacence();
void initialiser_variables();
int detecte_arete_minimale();
int prim();
void afficher_arbre_couvrant();






int main()
{
	if (! charger_fichier_donnees() )  exit(0);
	afficher_matrice_adjacence();
	prim();
	afficher_arbre_couvrant();
	return 1;
}
```

```c
int charger_fichier_donnees()
{	FILE *f;
	char nom_fichier[80];
	int i,j;
	printf("\n\nAlgorithme de PRIM (arbre couvrant de poids 
minimum)");
	printf("\n****************************************
********");
	printf("\n\n\tFichier de donnees : ");
	scanf("%s",nom_fichier);
	if (  (  f=fopen(nom_fichier, "r")  ) == NULL)
	{	printf("\n\n\tFichier de donnees incorrect.");
		return 0;
	}
	else
	{	fscanf(f,"%d",&n_G);
		for(i=1; i<=n_G; i++)
			for(j=1; j<=n_G; j++) fscanf(f, "%d", 
&G[i][j]);
	}
	return 1;
}

void afficher_matrice_adjacence()
{	int i,j;
	printf("\n\n\tNombre de sommets = %d",n_G);
	printf("\n\n\tMatrice d'adjacence :");

	for(i=1; i<=n_G; i++)
	{	printf("\n");
		for(j=1; j<=n_G; j++) printf("\t%d",G[i][j]);
	}
}
```

```c
void initialiser_variables()
{	int i,j;
	for(i=1; i <= n_G; i++)
		for(j=1; j <= n_G; j++)
			if(G[i][j] == 0) G[i][j] = Inf;

	S[1] = 1;
	n_S = 1;

	for(i=1; i <= n_G - 1 ; i++) T[i] = i+1;
	n_T = n_G - 1;

	n_A = 0;
	poids= 0;
}

int detecte_arete_minimale()
{	int a, b, i,j;
	double v_min = Inf;

  for(i=1; i <= n_S; i++)
  {	a = S[i];
	for(j=1; j <= n_T; j++)
	{	b = T[j];
		if( G[a][b] < v_min)
		{	v_min = G[a][b];
			a_min = a;
			b_min = b;
			indice = j;
		}
	}
  }
  return 1;
}
```

```c
int prim()
{
	initialiser_variables();
	while(n_S < n_G)
	{
		detecte_arete_minimale();

		A[++n_A][1] = a_min;
		A[n_A][2] = b_min;
		S[++n_S] = b_min;
		for(int j=indice; j <= n_T - 1; j++) T[j] = T[j+1];
		n_T--;
		poids = poids + G[a_min][b_min];
	}
	return 1;
}

void afficher_arbre_couvrant()
{
	printf("\n\n\tArbre couvrant de poids minimum : ");
	for(int i=1; i <= n_A; i++) printf("\n\t%d,
%d",A[i][1],A[i][2]);
	printf("\n\n\tPoids de l'arbre : %d",poids);
}
```

Exécution du programme C

- Les données (matrice d'adjacence) sont sauvegardées sur un fichier texte, puis on lance le programme C précèdent en lui indiquant le fichier de données sur lequel il doit travailler.

Exemple : Reprenons le graphe à 9 sommets de la page Algorithme de Prim

Etape 1: on crée un fichier texte nommé **graphe1.txt**, contenant :

- sur la première ligne : on saisit le nombre de sommets du graphe (9 dans ce cas)
- juste après, on saisit la matrice d'adjacence du graphe comme suit :

graphe1.txt - Bloc-notes

Fichier Edition Format Affichage ?

```
9
0   4   8   0   0   0   0   0   0
4   0   11  8   0   0   0   0   0
8   11  0   0   7   1   0   0   0
0   8   0   0   2   0   7   4   0
0   0   7   2   0   6   0   0   0
0   0   1   0   6   0   0   2   0
0   0   0   7   0   0   0   0   9
0   0   0   4   0   2   0   0   10
0   0   0   0   0   0   9   10  0
```

Etape 2 : on exécute le programme C. Apres le lancement, on saisit le nom du fichier de données :

D:\PRIM4.EXE

```
Algorithme de PRIM (arbre couvrant de poids minimum)
*****************************************************

      Fichier de donnees : graphe1.txt_
```

Le résultat suivant sera affiché :

```
      Arbre couvrant de poids minimum :
      1, 2
      1, 3
      3, 6
      6, 8
      8, 4
      4, 5
      4, 7
      7, 9

      Poids de l'arbre : 37
```

3 – Problème du Plus Court Chemin en Théorie des Graphes

Université Alger 1, Dept Maths & Informatique

Dr. Fodil LAIB

Avril 2017

Introduction

Problématique :

- Soit un graphe orienté et valué G=(X,U,W).
- Il peut y avoir plusieurs chemins entre deux sommets x et y.
- Trouver le chemin le plus court entre x et y.

Applications:

- Optimiser le trafic urbain
- Navigation des rebots
- Traitement de texte (Latex)
- Conception des cartes électroniques
- Appels de fonctions dans les algorithmes avancés
- Routage des messages dans les télécommunications

Il y a 2 classes d'algorithmes :

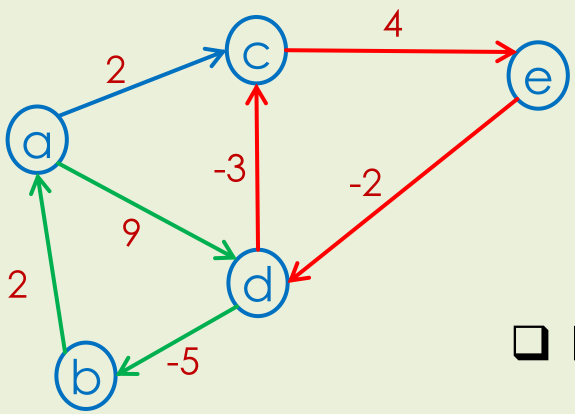
- Classe **P1** : Trouver les plus courts chemins entre un sommet source s et les autres sommets du graphe :
  - Algorithme de Dijkstra
  - Algorithme de Bellman
- Classe **P2** : Trouver les plus courts chemins entre tous les couples de sommets du graphe :
  - Algorithme de Dantzig
  - Algorithme de Floyd

Notation : Selon le besoin, on note

- Les valeurs des arcs $u, \cdots, u_m \in U$ par $w_1, \cdots, w_m$ avec $W = (w_1, \cdots, w_m)$.
- La valeur de l'arc $(x, y) \in U$ est notée par $w(x, y)$

Quelques Résultats

Un **circuit absorbant** est un circuit dont la longueur totale est inférieure à 0.



- La longueur du circuit (c,e,d,c) est w = 4-2-3 = -1
  ⇒ circuit absorbant

- La longueur du circuit (a,d,b,a) est w = 9-5+2 = 6
  ⇒ circuit non absorbant

Théorème 1: Le problème P1 admet une solution dans le graphe G=(X,U,W) à partir d'un sommet $s \in X$ si et seulement si

1. Il existe un chemin entre s et tous les autres sommets, et
2. G est sans circuit absorbant.

Lemme : Si on a

1. G est un graphe simple non orienté connexe, et
2. $w_i > 0$ pour toutes les arêtes,

alors P1 et P2 ont une solution à partir de tout sommet de G.

Remarque : Cette solution n'est pas nécessairement unique.

Théorème : Tout sous-chemin $x_i \cdots x_j$ d'un chemin optimal $x_1 \cdots x_i \cdots x_j \cdots x_k$ est un chemin optimal entre x_i et x_j.

Théorème : La suite $D(1), \cdots, D(n)$ représente les plus courtes distances de la source s vers les autres sommets si et seulement si

1. $D(1) = 0$, et
2. $D(y) \leq D(x) + w(x,y), \forall (x,y) \in U$.

Algorithme de Dijkstra (1959)

Il permet de résoudre P1 pour $w_i > 0$ pour tous les i.

Procedure Dijkstra : Recherche des Plus Courts Chemins

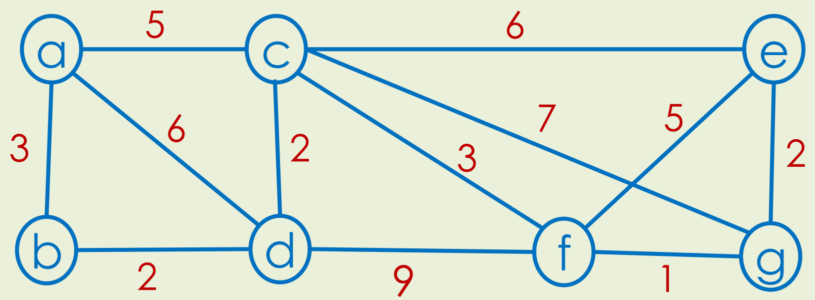
Input: G(X,U,W) et s
Output: D et P

- $D(s) = 0$ et $D(x) = \infty$ pour $x \in X/x$
- $\forall\, x \in X,\ \ P(x) = \emptyset$
- $F = X$

Tant que $F \neq \emptyset$ **Faire**
  - Trouver x tel que $D(x) = \min\limits_{z \in F} D(z)$
  - $F = F - \{x\}$

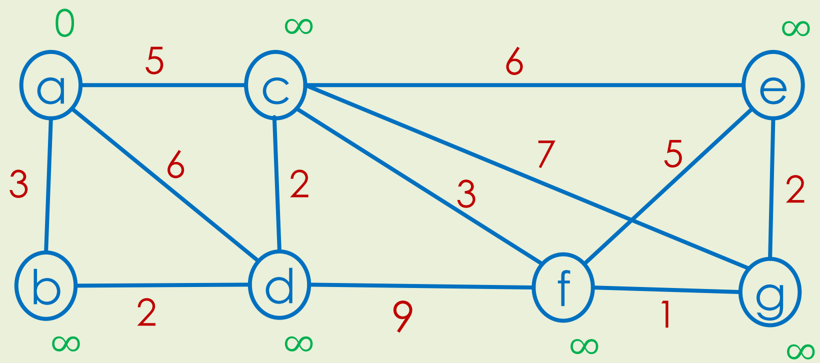
  Pour tout $y \in Successeurs(x)$ **Faire**
    Si $D(y) > d(x) + w(x,y)$ **Alors**
      $D(y) = d(x) + w(x,y)$
      $P(y) = x$
    Fin SI
  Fin Pour
Fin Tant que

Exemple 1 : Appliquer l'algorithme de Dijkstra pour retrouver les plus courts chemins entre le sommet a et les autres sommets du graphe suivant :

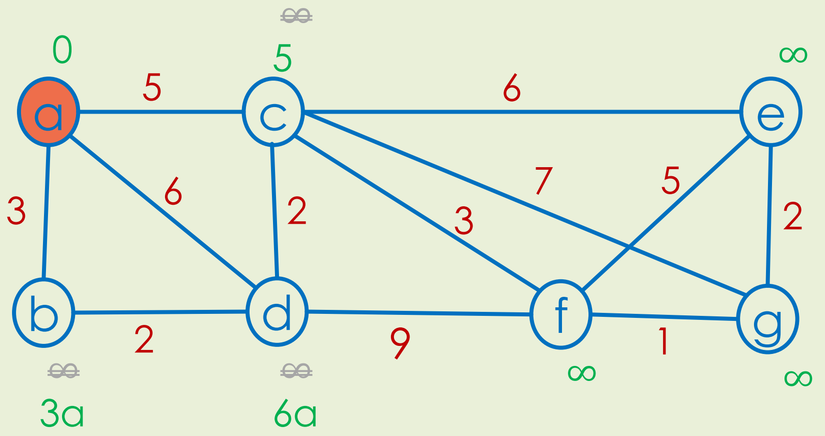


1iere approche : Graphique

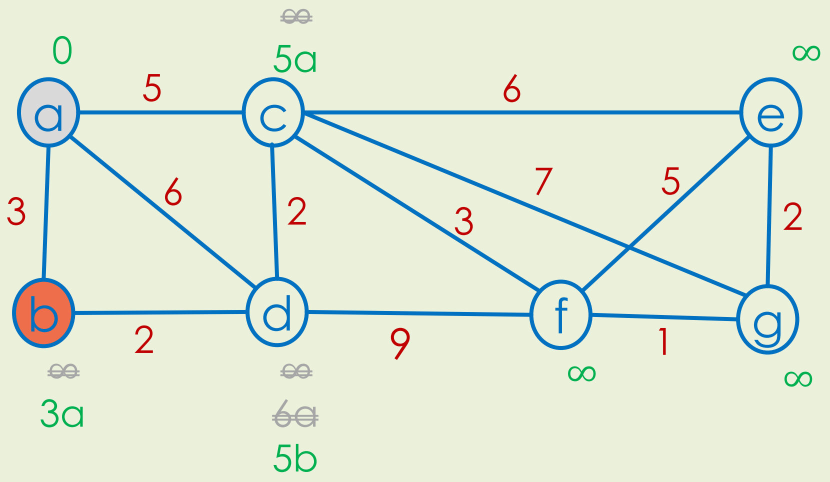
Initialisation :



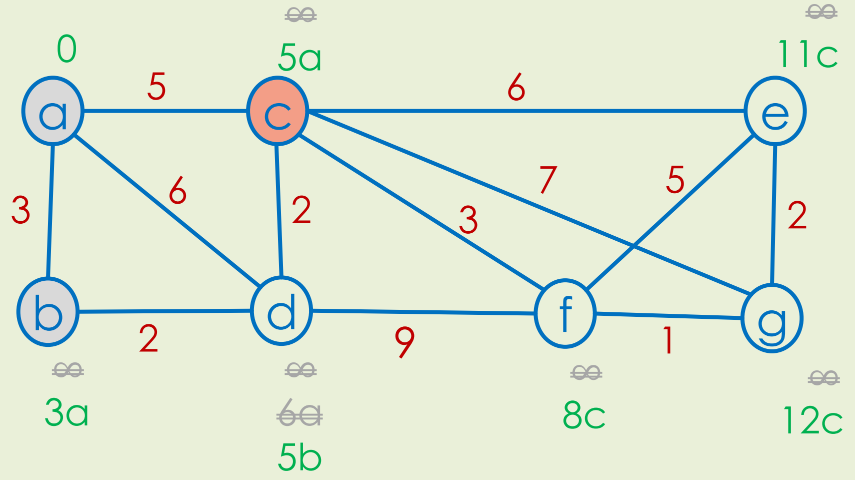
Itération 1:



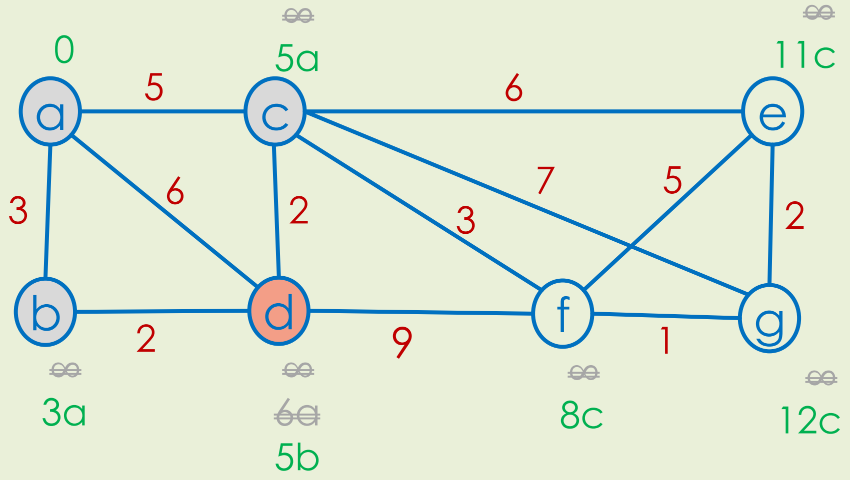
Itération 2 :



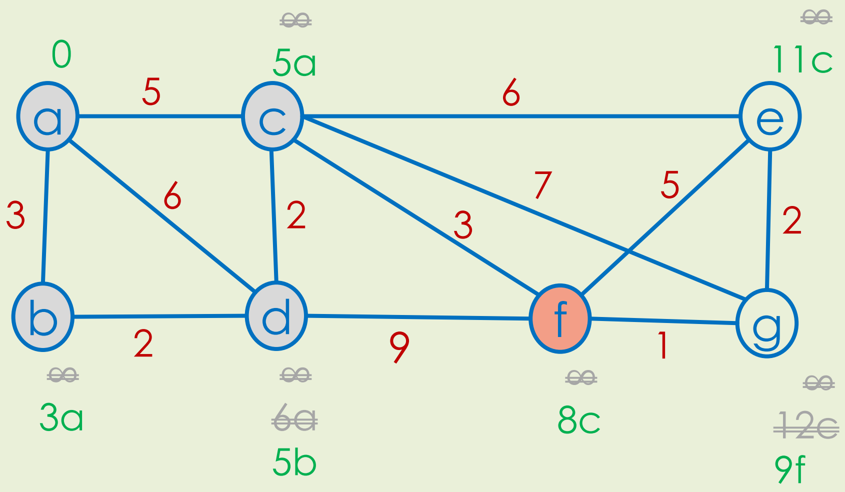
Itération 3 :



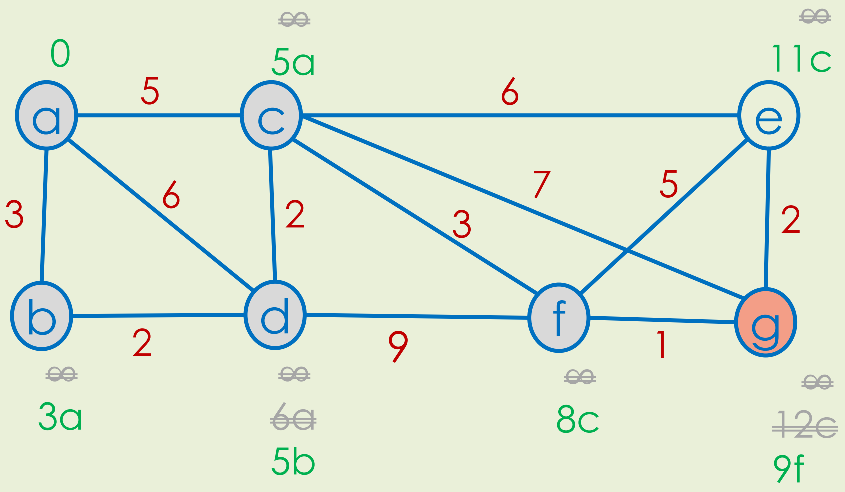
Itération 4 :



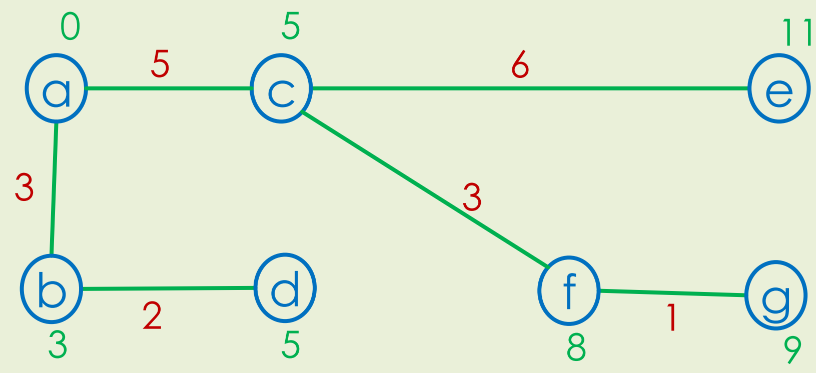
Itération 5 :



Itération 6 :

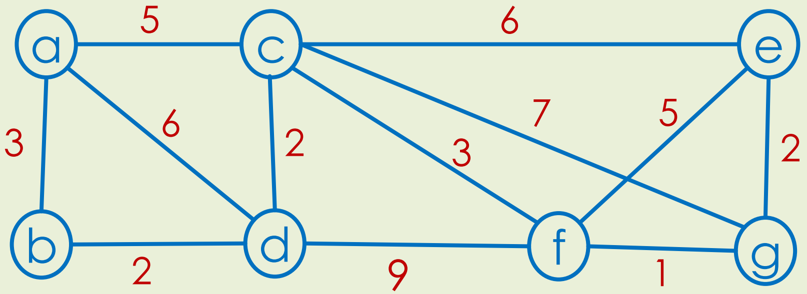


Résultat de l'algorithme



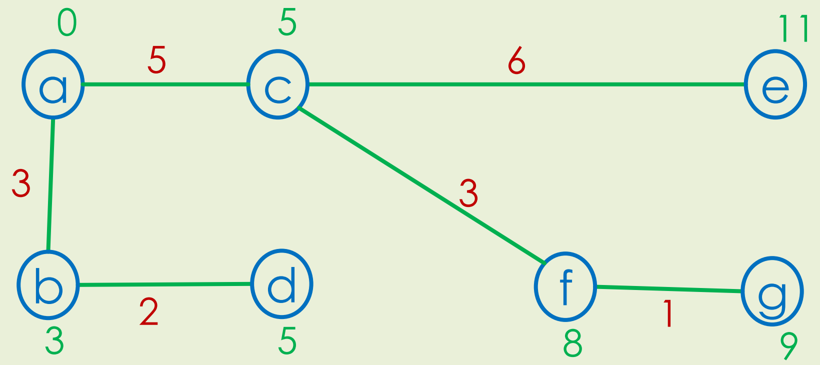
	a	b	c	d	e	f	g
D	0	3	5	5	11	8	9
P	∅	a	a	b	c	c	f

2ieme approche : Tableau

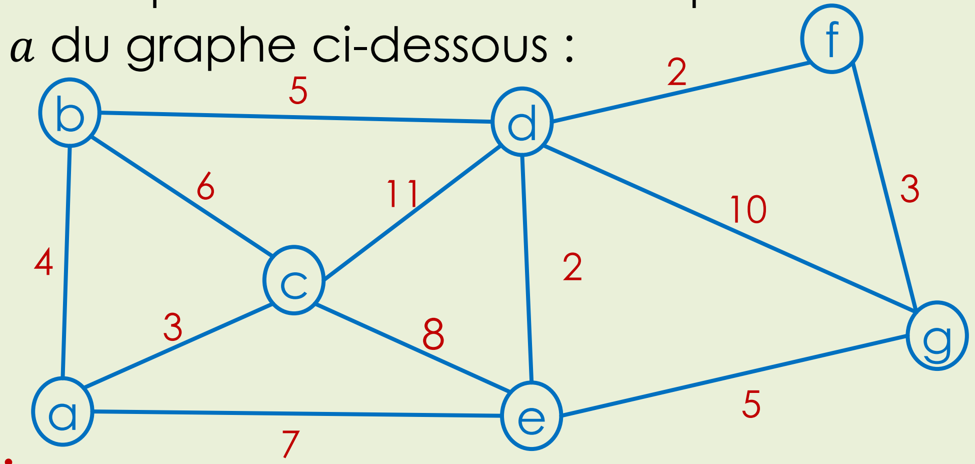


	a	b	c	d	e	f	g
	0	∞	∞	∞	∞	∞	∞
a	0	**3*a***	5*a*	6*a*	∞	∞	∞
b	0	3*a*	**5*a***	5*b*	∞	∞	∞
c	0	3*a*	5*a*	**5*b***	11*c*	8*c*	12*c*
d	0	3*a*	5*a*	5*b*	11*c*	**8*c***	12*c*
f	0	3*a*	5*a*	5*b*	11*c*	8*c*	**9*f***
g	0	3*a*	5*a*	5*b*	**11*c***	8*c*	9*f*

	a	*b*	*c*	*d*	*e*	*f*	*g*
D	0	3	5	5	11	8	9
P	∅	a	a	b	c	c	f



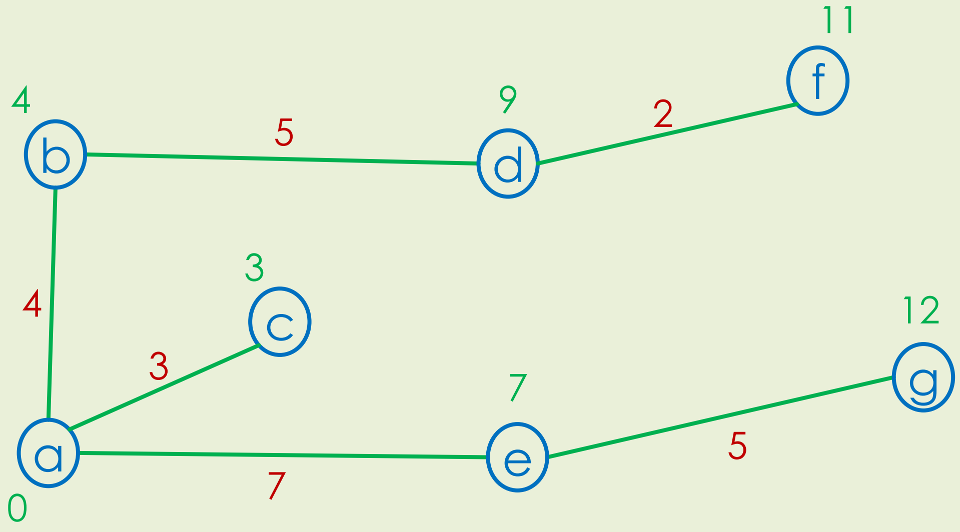
Exemple 2 : Appliquer l'algorithme de Dijkstra pour déterminer les plus courts chemins en partant du sommet a du graphe ci-dessous :



Solution :

	a	b	c	d	e	f	g
	0	∞	∞	∞	∞	∞	∞
a	0	$4a$	$3a$	∞	$7a$	∞	∞
c	0	$4a$	$3a$	$14c$	$7a$	∞	∞
b	0	$4a$	$3a$	$9b$	$7a$	∞	∞
e	0	$4a$	$3a$	$9b$	$7a$	∞	$12e$
d	0	$4a$	$3a$	$9b$	$7a$	$11d$	$12e$
f	0	$4a$	$3a$	$9b$	$7a$	$11d$	$12e$

	a	b	c	d	e	f	g
D	0	4	3	9	7	11	12
P	∅	a	a	b	a	d	e



Complexité : La complexité temporelle de l'algorithme de Dijkstra est $O(|X|^2)$.

Algorithme de Bellman (1956)

Procedure Bellman : Recherche des Plus Courts Chemins

Input: G(X,U,W) et s

Output: D et P

- $D(s) = 0$ et $D(x) = \infty$ pour $x \in X/x$
- $\forall\, x \in X,\ \ P(x) = \emptyset$

// Rechercher les plus courts chemins

Pour i allant de 1 à |X|-1 Faire

  Pour tout arc $(u, v) \in U$ Faire

    Si $D(v) > D(u) + w(u, v)$ Alors

      $D(v) = D(u) + w(u, v)$

      $P(v) = u$

    Fin Si

  Fin Pour

Fin Pour

// Détecter circuit absorbant

Pour tout arc $(u, v) \in U$ Faire

  Si $D(u) + w(u, v) < D(v)$ Alors

    Retourner « Il existe un circuit absorbant »

  Fin SI

Fin Pour

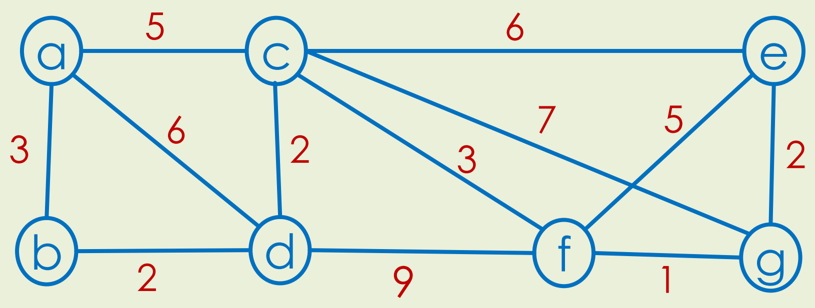
Retourner « Pas de circuit absorbant »

- L'algorithme de Bellman (appelé aussi algorithme de Bellman-Ford) retrouve les plus courts chemins à partir d'un sommet de départ même si certains arcs portent des valeurs négatives.
- Son principe : pour chaque sommet du graphe, on parcourt séquentiellement tous les arcs pour voir si on peut améliorer les distances.
- La deuxième partie de l'algorithme vérifie s'il n'existe pas de circuit absorbant dans le graphe.

Remarque : Cet algorithme peut s'arrêter avant la (|X|-1)ieme itération si à l'issue d'une itération $k < |X| - 1$, on n'arrive pas à améliorer la distance d'aucun des sommets.

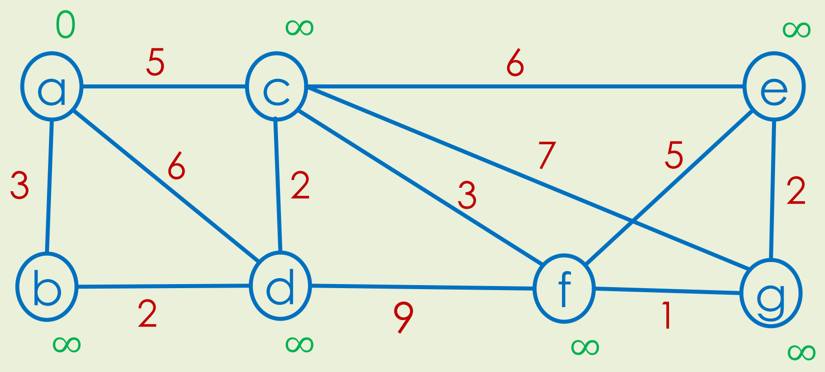
Complexité : La complexité de l'algorithme de Bellman est $O(|X|^3)$.

Exemple 1 : Appliquer l'algorithme de Bellman pour retrouver les plus courts chemins entre le sommet a et les autres sommets du graphe suivant :



Solution

Initialisation : On initialise $D(a) = 0$ et $D(x) = \infty$ pour les autres sommets



- A chaque itération, le parcourt séquentiel des arcs de ce graphe doit se faire dans l'ordre suivant :

ab, ac, ad,

ba, bd,

ca, cd, ce, cf, cg,

da, db, dc, df,

ec, ef,eg

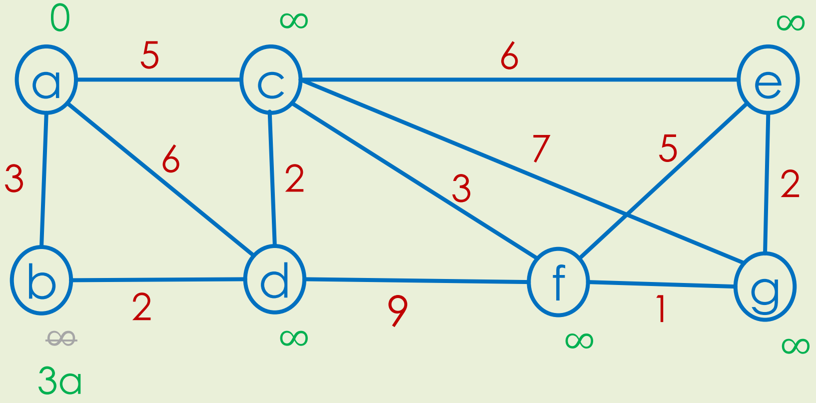
fc, fd, fe, fg

gc, ge, gf

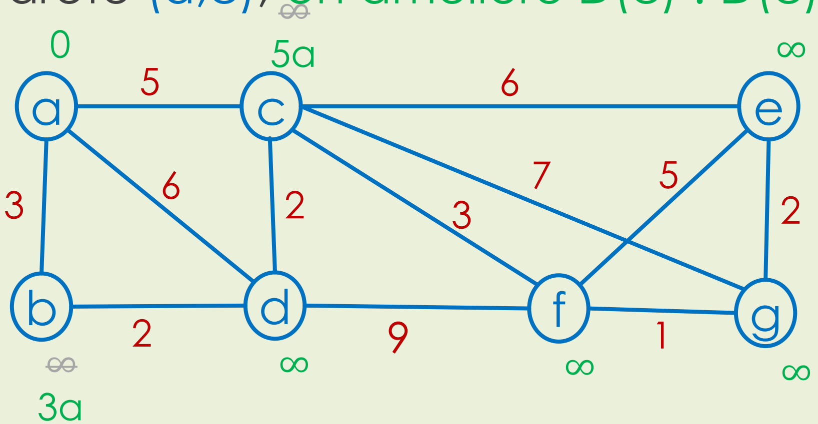
1iere approche : Graphique

Itération 1 : On parcourt toutes les arêtes du graphe pour voir si on peut améliorer certaines distances :

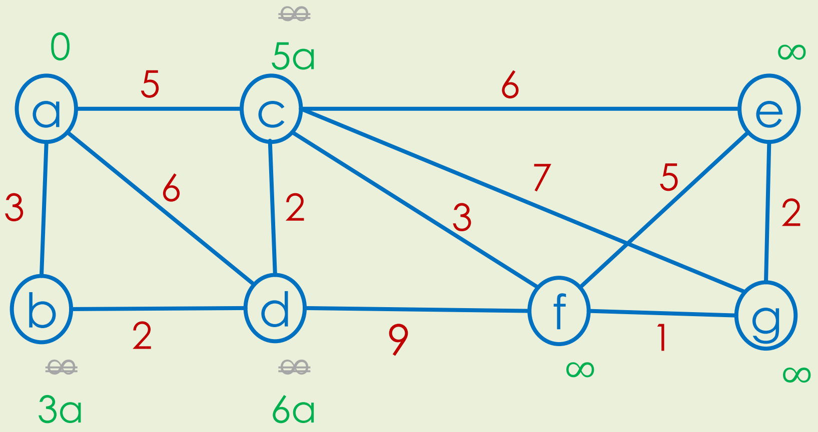
- Avec l’arête (a,b), on améliore D(b) : D(b) = 3



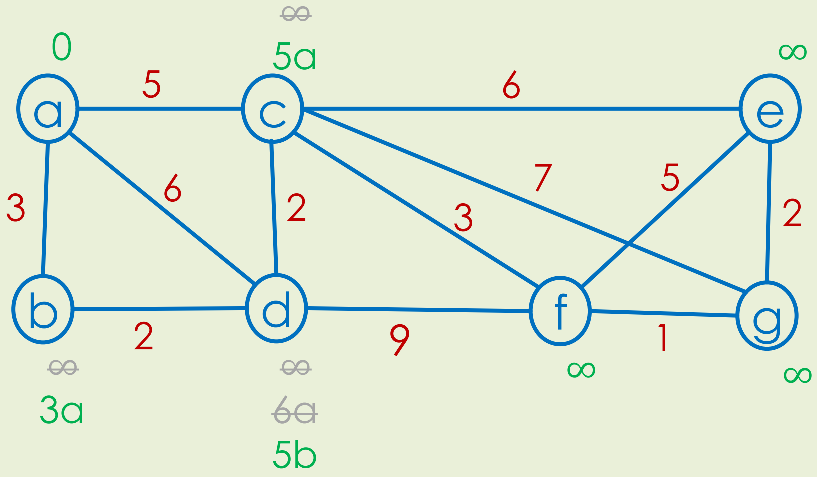
- Avec l’arête (a,c), on améliore D(c) : D(c) = 5



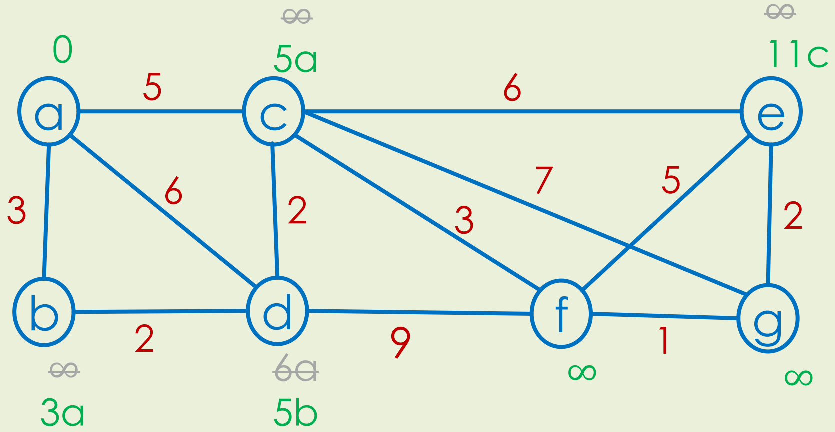
- l’arête (a,d), on améliore D(d) : D(d) = 6



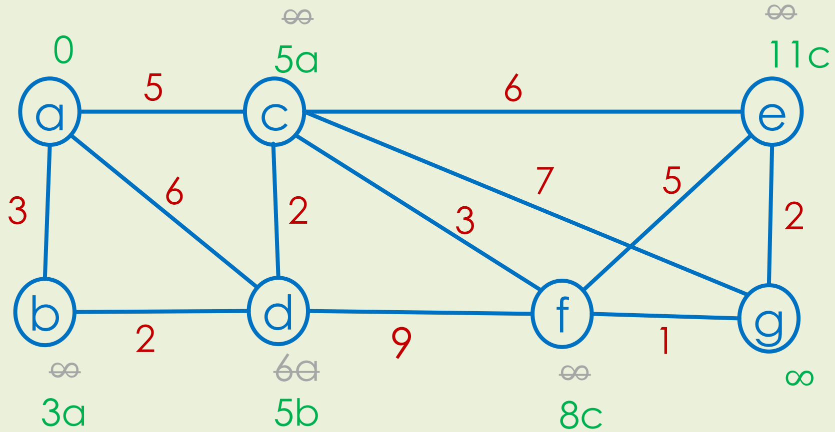
- l’arête (b,a), aucune amélioration
- l’arête (b,d), on améliore D(d) : D(d) = 3+2=5



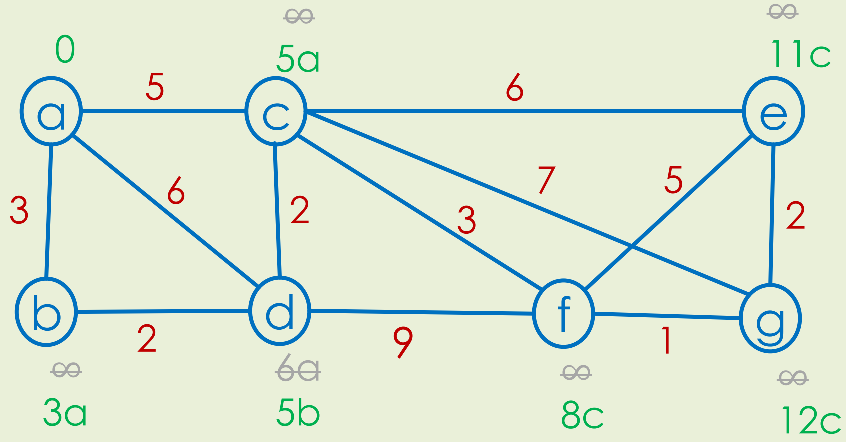
- Arête (c,a), aucune amélioration
- Arête (c,d), aucune amélioration
- Arête (c,e), on améliore D(e): D(e)=5+6=11



- Arête (c,f), on améliore D(f) : D(f) = 5+3=8

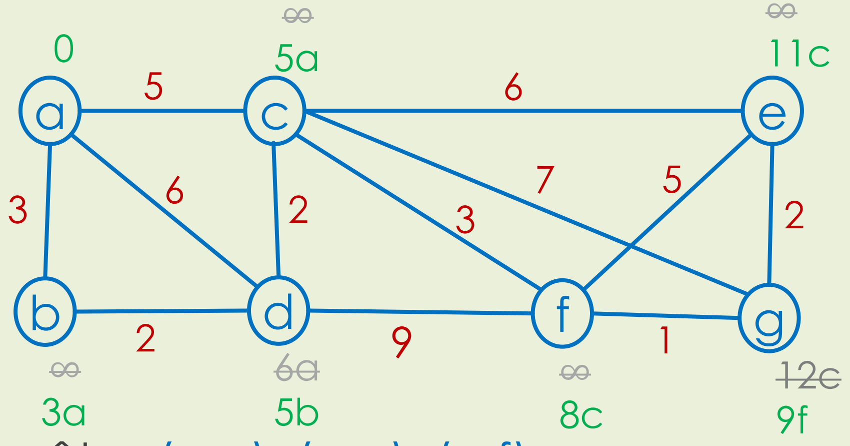


- Arête (c,g), on améliore D(g) : D(g) = 5+7=12



- Les arêtes (d,a), (d,b), (d,c), (d,f) : aucune amélioration
- Les arêtes (e,c), (e,f), (e,g) : aucune amélioration
- Les arêtes (f,c), (f,d), (f,e), aucune amélioration

- Arête (f,g), on améliore D(g) : D(g)=8+1=9



- Les arêtes (g,c), (g,e), (g,f) : aucune amélioration

Itération 2 : On parcourt à nouveau toutes les arêtes du graphe pour voir si on peut améliorer certaines distances :

- Les arêtes (a,b), (a,c), (a,d) : aucune amélioration
- Les arêtes (b,a), (b,d) : aucune amélioration
- Les arêtes (c,a), (c,d), (c,e), (c,f), (c,g) : aucune amélioration
- Les arêtes (d,a), (d,b), (d,c), (d,f) : aucune amélioration
- Les arêtes (e,c), (e,f), (e,g) : aucune amélioration
- Les arêtes (f,c), (f,d), (f,e), (f,g) : aucune amélioration
- Les arêtes (g,c), (g,e), (g,f) : aucune amélioration

L'itération 2 n'améliore aucune distances D(x), on arrête donc l'algorithme à ce niveau. Et comme le nombre d'itération $i = 2 < |X| - 1 = 7 - 1 = 6$, la solution obtenue est optimale.

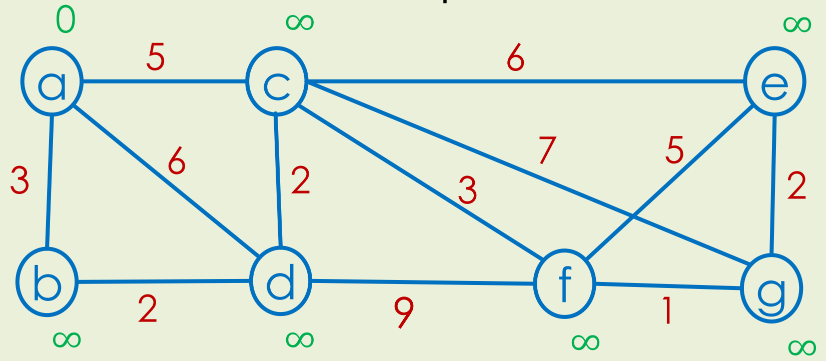
Les chemins obtenus sont :

	a	*b*	*c*	*d*	*e*	*f*	*g*
D	0	3	5	5	11	8	9
P	Ø	a	a	b	c	c	f

...

2° approche : Tableau

On affiche seulement les arcs qui améliorent les distances.



Initialisation :

	a	b	c	d	e	f	g
	0	∞	∞	∞	∞	∞	∞

Itération 1 :

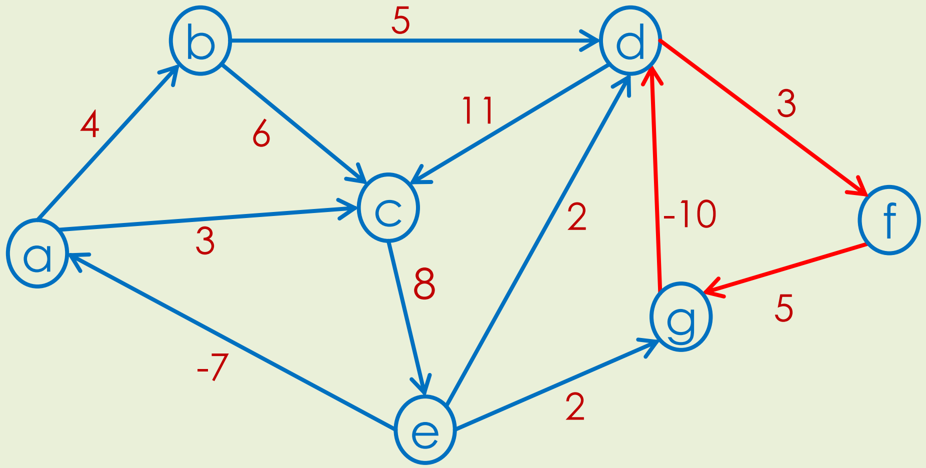
	a	b	c	d	e	f	g
ab		3					
ac			5				
ad				6			
bd				5			
ce					11		
cf						8	
cg							12
fg							9

Itération 2 :

On parcourt à nouveaux tous les 24 arêtes du graphe. On constate qu'aucun d'eux n'améliore la distance d'un sommet. On arrête donc à cette itération. Cette solution est optimale.

	a	*b*	*c*	*d*	*e*	*f*	*g*
D	0	3	5	5	11	8	9
P	∅	a	a	b	c	c	f

Exemple 2 (avec circuit absorbant) : Utiliser l'algorithme de Bellman pour retrouver les plus courts chemins partant du sommet a pour le graphe ci-dessous :



Solution

Initialisation :

	a	b	c	d	e	f	g
	0	∞	∞	∞	∞	∞	∞

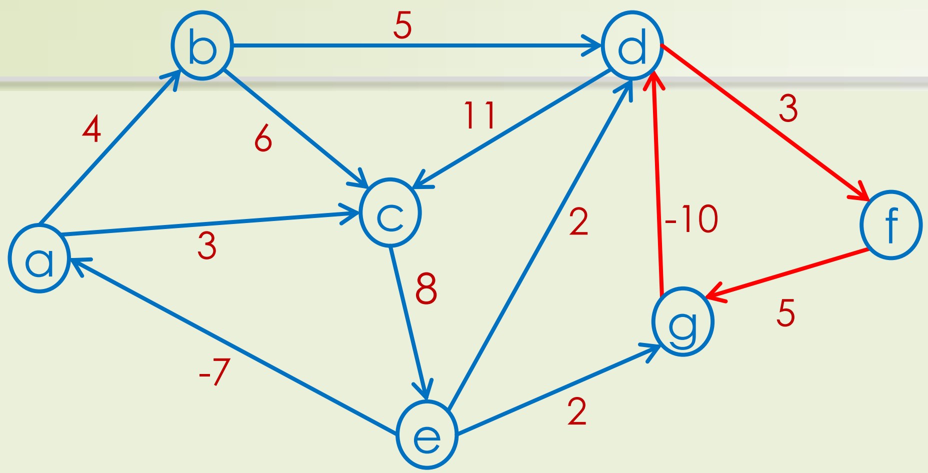
Itération 1 :

	a	b	c	d	e	f	g
	0	∞	∞	∞	∞	∞	∞
ab		4					
ac			3				
bd				9			
ce					11		
df						12	
eg							13
gd				3			

Itération 2 :

	a	b	c	d	e	f	g
	0	4	3	3	11	12	13
df						6	
fg							11
gd				1			

…



Itération 3 :

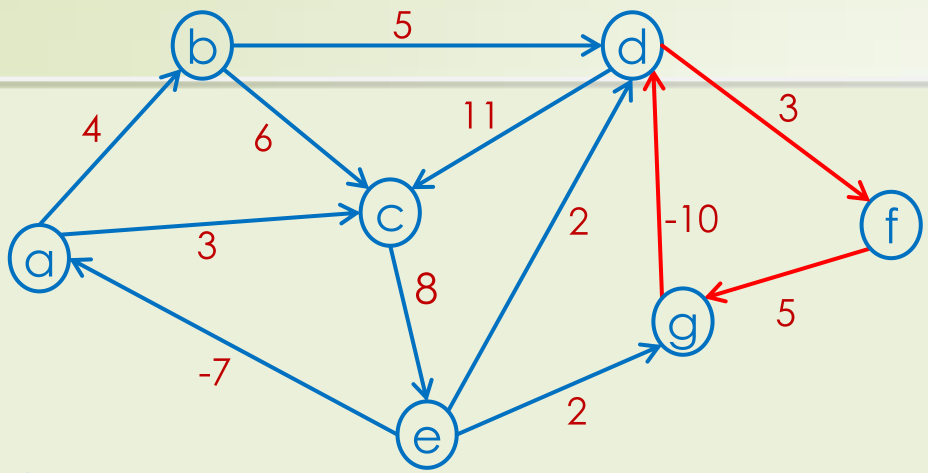
	a	b	c	d	e	f	g
	0	4	3	1	11	6	11
df						4	
fg							9
gd				-1			

Itération 4 :

	a	b	c	d	e	f	g
	0	4	3	-1	11	4	9
df						2	
fg							7
gd				-3			

Itération 5 :

	a	b	c	d	e	f	g
	0	4	3	-3	11	2	7
df						0	
fg							5
gd				-5			



Itération 6 :

	a	b	c	d	e	f	g
	0	4	3	-5	11	0	5
df						-2	
fg							3
gd				-7			

L'itération 6 doit être la dernière (|X|-1 = 7-1=6). Faisant une itération de plus pour voir s'il y aurait un circuit absorbant.

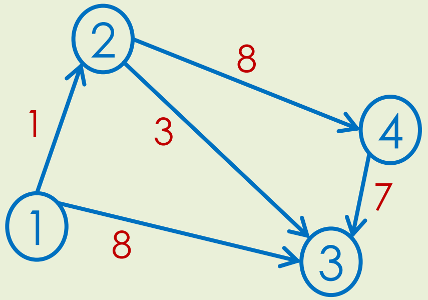
Itération 7 :

	a	b	c	d	E	f	g
	0	4	3	-7	11	-2	3
df						-4	
fg							1
gd				-9			

Cette 7° itération améliore les distances, donc il y a un circuit absorbant dans ce graphe.

Algorithme de Dantzig

- Il détermine les plus courts chemins entre tous les couples de sommets (x,y) du graphe (problème P2).
- Les résultats sont donnés dans
  - Matrice des distances $D = [d_{ij}]$ où d_{ij} est la plus courte distance entre les sommets i et j.
  - Matrice des prédécesseurs $P = [p_{ij}]$ où p_{ij} est le prédécesseur de j dans le plus court chemin entre i et j.
- Soit le graphe à 4 sommets suivant :



$$D = \begin{bmatrix} d_{11} & d_{12} & d_{13} & d_{14} \\ d_{21} & d_{22} & d_{23} & d_{24} \\ d_{31} & d_{32} & d_{33} & d_{34} \\ d_{41} & d_{42} & d_{43} & d_{44} \end{bmatrix}$$

$$P = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \\ p_{41} & p_{42} & p_{43} & p_{44} \end{bmatrix}$$

où

- $[d_{i1} \quad d_{i2} \quad d_{i3} \quad d_{i4}]$ est le vecteur des plus courtes distances entre le sommet i et les autres sommets du graphe, et
- $[p_{i1} \quad p_{i2} \quad p_{i3} \quad p_{i4}]$ sont les prédécesseurs des sommets 1, 2, 3 et 4 sur les plus court chemins à partir du sommet i, avec $i = 1 \cdots 4$.

Principe de l'algorithme : L'idée est de:

- commencer par le graphe G_0 possédant un seul sommet x_1 et construire les matrices $D^{(0)} = [0]$ et $P^{(0)} = [1]$
- Construire le graphe G_1 en ajoutant le sommet x_2 à G_0, puis construire les matrices

  $D^{(1)} = \begin{bmatrix} 0 & d_{21} \\ d_{21} & 0 \end{bmatrix}$ et $P^{(1)} = \begin{bmatrix} 1 & d_{21} \\ d_{21} & 2 \end{bmatrix}$
- Ajouter graduellement d'autres sommets $x_3 \cdots x_n$ jusqu'à l'obtention du graphe $G_n = G$ en construisant au passage les matrices $D^{(2)} \dots D^{(n-1)}$ et $P^{(2)} \dots P^{(n-1)}$ avec $D^{(n-1)} = D$ et$P^{(n-1)} = P$

...

Théorème : Si le graphe G est sans circuit absorbant, l'algorithme de Dantzig résout le problème P2 en un temps $O(n^3)$.

<u>Procedure Dantzig</u> : Recherche des Plus Courts Chemins entre tous les couples de sommets

Input: M //matrice d'adjacence

Output : $D = [d_{ij}]$ et $P = [p_{ij}]$

//Initialisation , n = dimension(M)

pour $i, j = 1 \cdots n$, $d_{ij} = \infty, d_{ii} = 0$, $p_{ij} = 0, p_{ii} = i$

```
Pour k allant de 1à  n-1 Faire
    Pour i allant de 1 à  k Faire
        Pour j allant de 1 à k Faire
            //insérer ici Bloc A
        Fin Pour j
    Fin Pour i
    Pour 1 allant de 1 à k Faire
        Pour j allant de 1 à k Faire
            //insérer ici Bloc B
        Fin Pour j
    Fin Pour i
Fin Pour k
```

//Bloc A

$d = D(i, j) + M(j, k+1)$

Si $d < D(i, k+1)$ Alors

    D(i,k+1)=d

    P(i,k+1)=j

Fin Si

$d = M(k+1, j) + D(j, i)$

Si $d < D(k+1, i)$ Alors

    $D(k+1, i) = d$

    Si $i = j$ Alors

        $P(k+1, i) = k+1$

    Sinon

        $P(k+1, i) = P(j, i)$

    Fin Si

Fin Si

//Bloc B

$d = D(i, k+1) + D(k+1, j)$

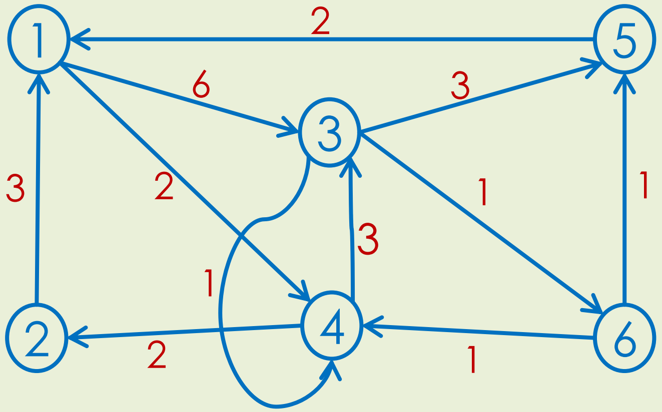
Si d<D(i,j) Alors

    D(i,j)=d

    P(i,j)=P(k+1,j)

Fin Si

Exemple : Utiliser l'algorithme de Dantzig pour retrouver tous les plus courts chemins entre les sommets du graphe suivants :



Solution :

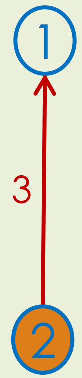
$$M = \begin{bmatrix} 0 & \infty & 6 & 2 & \infty & \infty \\ 3 & 0 & \infty & \infty & \infty & \infty \\ \infty & \infty & 0 & 1 & 3 & 1 \\ \infty & 2 & 3 & 0 & \infty & \infty \\ 2 & \infty & \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & 1 & 1 & 0 \end{bmatrix}$$

K = 0 : On commence avec le sommet {1}

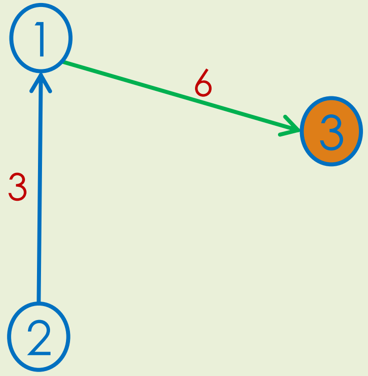
$$D^{(0)} = [0], \quad P^{(0)} = [1]$$

K = 1 : On ajoute le sommet {2}

$$D^{(1)} = \begin{bmatrix} 0 & \infty \\ 3 & 0 \end{bmatrix}, \quad P^{(1)} = \begin{bmatrix} 1 & 0 \\ 2 & 2 \end{bmatrix}$$



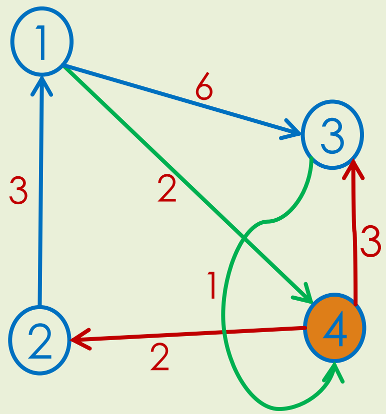
K = 2 : On ajoute le sommet {3}



$$D^{(2)} = \begin{bmatrix} 0 & \infty & 6 \\ 3 & 0 & 9 \\ \infty & \infty & 0 \end{bmatrix}, \quad P^{(2)} = \begin{bmatrix} 1 & 0 & 1 \\ 2 & 2 & 1 \\ 0 & 0 & 3 \end{bmatrix}$$

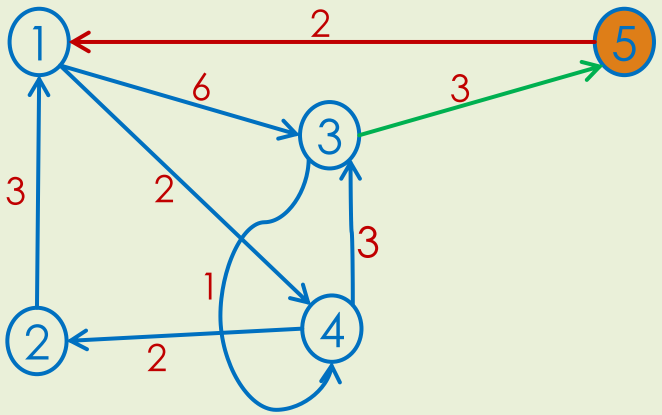
...

K = 3 : On ajoute le sommet 4



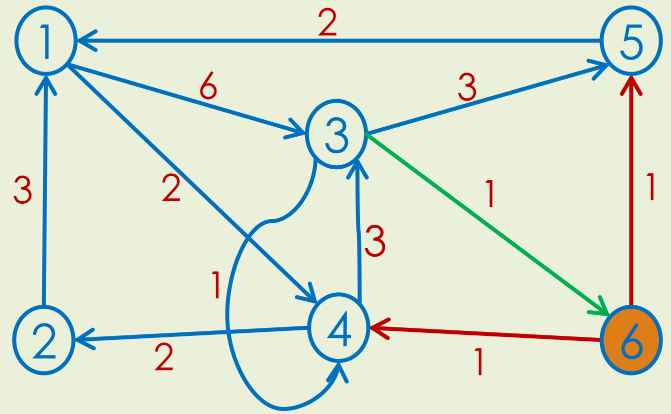
$$D^{(3)} = \begin{bmatrix} 0 & 4 & 5 & 2 \\ 3 & 0 & 8 & 5 \\ 6 & 3 & 0 & 1 \\ 5 & 2 & 3 & 0 \end{bmatrix}, \quad P^{(3)} = \begin{bmatrix} 1 & 4 & 4 & 1 \\ 2 & 2 & 4 & 1 \\ 2 & 4 & 3 & 3 \\ 2 & 4 & 4 & 4 \end{bmatrix}$$

K = 4 : On ajoute le sommet {5}



$$D^{(4)} = \begin{bmatrix} 0 & 4 & 5 & 2 & 8 \\ 3 & 0 & 8 & 5 & 11 \\ 5 & 3 & 0 & 1 & 3 \\ 5 & 2 & 3 & 0 & 6 \\ 2 & 6 & 7 & 4 & 0 \end{bmatrix}, \quad P^{(4)} = \begin{bmatrix} 1 & 4 & 4 & 1 & 3 \\ 2 & 2 & 4 & 1 & 3 \\ 5 & 4 & 3 & 3 & 3 \\ 2 & 4 & 4 & 4 & 3 \\ 5 & 4 & 4 & 1 & 5 \end{bmatrix}$$

K = 5 : On ajoute le sommet {6}



$$D^{(5)} = \begin{bmatrix} 0 & 4 & 5 & 2 & 7 & 6 \\ 3 & 0 & 8 & 5 & 10 & 9 \\ 4 & 3 & 0 & 1 & 2 & 1 \\ 5 & 2 & 3 & 0 & 5 & 4 \\ 2 & 6 & 7 & 4 & 0 & 8 \\ 3 & 3 & 4 & 1 & 1 & 0 \end{bmatrix}, \quad P^{(5)} = \begin{bmatrix} 1 & 4 & 4 & 1 & 6 & 3 \\ 2 & 2 & 4 & 1 & 6 & 3 \\ 5 & 4 & 3 & 3 & 6 & 3 \\ 2 & 4 & 4 & 4 & 6 & 3 \\ 5 & 4 & 4 & 1 & 5 & 3 \\ 5 & 4 & 4 & 6 & 6 & 6 \end{bmatrix}$$

4 – Problème de Flots en Théorie des Graphes

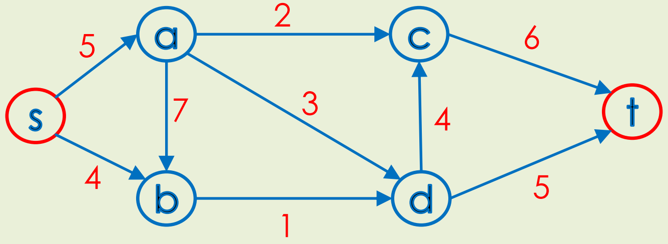
Université Alger 1, Dept Maths & Informatique

Dr. Fodil LAIB

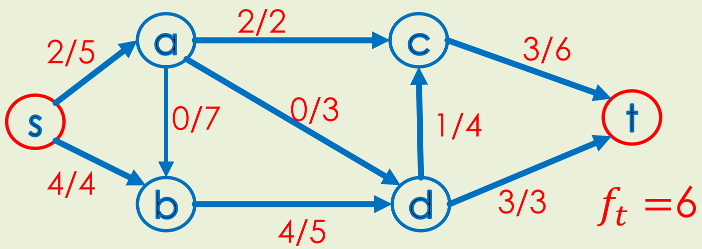
Mai 2017

Flot dans un Réseau

- Un réseau est un graphe orienté et valué G=(X,U,W) ayant deux sommets particuliers : la source s et le puit t.



- Un flot est une quantité de matière (marchandise, eau, électricité, information, etc.) envoyée par le sommet s, elle traverse certains arcs du graphe, pour atteindre le puits t.



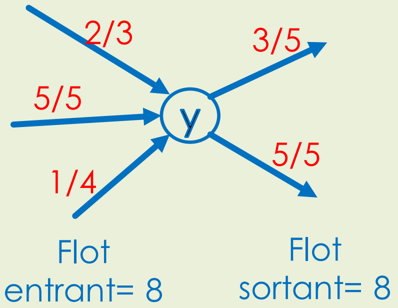
Un flot réalisable vérifie les 3 contraintes suivantes :

1) Le flot $f(u)$ qui traverse l'arc $u \in U$ vérifie la contrainte de capacité :

$$0 \leq f(u) \leq w(u)$$

2) Le flot qui traverse un sommet x est conservé c-à-d :

$$\sum_{y \in d^-(x)} f(y,x) = \sum_{y \in d^+(x)} f(x,y)$$



3) Le flot total f envoyé de la source s arrive au puits t c-à-d

$$f = \sum_{y \in d^+(s)} f(s,y) = \sum_{y \in d^-(t)} f(y,t)$$

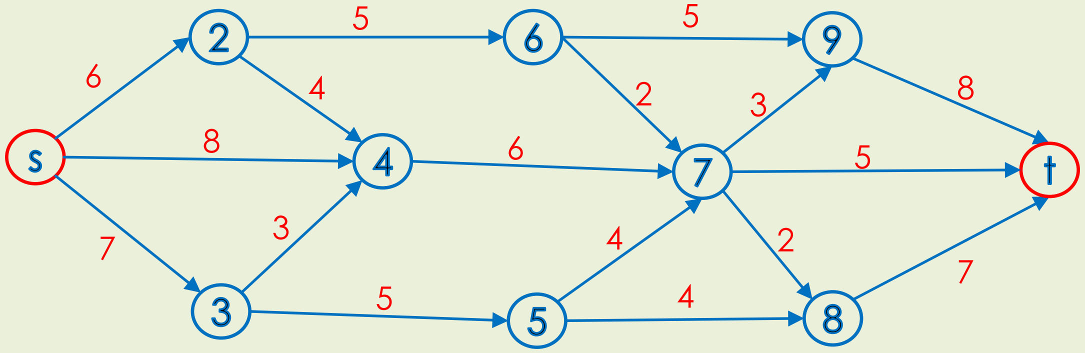
- Le problème du flot maximum consiste à trouver les valeurs des flots élémentaires $f(u)$, $\forall\, u \in U$ qui maximisent le flot total f.

Notation : La matrice des flots élémentaire est $F = [f_{ij}]$ où f_{ij} est le flot de l'arc $(i,j) \in U$. On a bien

$$f = \sum_{i=1}^{n} \sum_{j=1}^{n} f_{ij}$$

Exemple de Flot

Soit le réseau de distribution d'eau suivant :



- Il est composé des conduites (s,2), (s,4), …:
- La valeur de chaque arc représente la quantité maximale que la conduite peut transporter
- La source de l'eau est le sommet s
- La destination de l'eau est le sommet t

Question : Trouver la quantité maximale d'eau qu'on peut transporter de la source vers la destination en respectant les capacités des conduites.

Réseau avec plusieurs Sources/puits :

- Si le graphe possède plusieurs sources $\{s_1 \cdots s_k\}$, on ajoute une source fictive s_0 avec

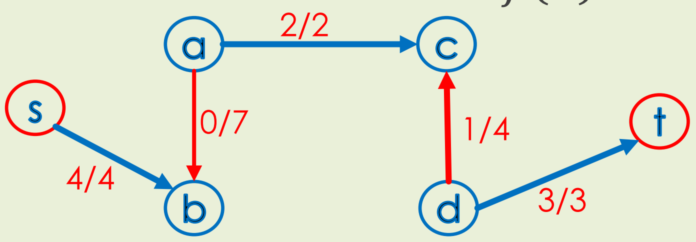
$$w(s_0, s_i) = \sum_{y \in d^+(s_i)} w(s_i, y)$$

- Si le graphe possède plusieurs puits $\{t_1 \cdots t_k\}$, on ajoute un puits fictif t_0 avec

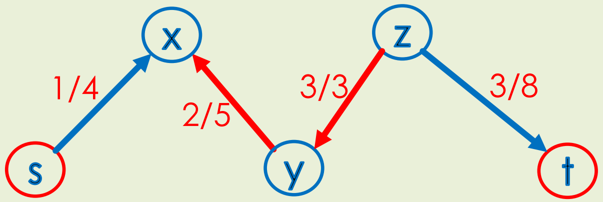
$$c(t_i, t_0) = \sum_{y \in d^-(t_i)} c(y, t_i)$$

Chaine Améliorante

- Une chaine C est dite améliorante si elle lie la source s au puits t et satisfait :
  - les arcs u directs vérifient $f(u) < w(u)$
  - les arcs u inversés vérifient $f(u) > 0$



- (s,b), (a,c), (d,t) sont des arcs directs
- (a,b), (d,c) sont des arcs inversés
- Cette chaine n'est pas améliorante car $f(a,b) \ngtr 0$



Chaine améliorante car

- Les arcs directs vérifient $f(s,x) < w(s,x)$, $f(z,t) < w(z,t)$
- Les arcs indirects vérifient $f(y,x) > 0$, $f(z,y) > 0$

Sur cette chaine, on peut augmenter le flot de $\varepsilon = \min(\varepsilon_1, \varepsilon_2)$

- $\varepsilon_1 = \min\limits_{u \epsilon C^+}(w(u) - f(u))$ où C^+ l'ensemble des arcs directs de la chaine C
- $\varepsilon_1 = \min\limits_{u \epsilon C^-} f(u)$ où C^-est l'ensemble des arcs indirects de la chaine C.

Les nouveaux flots élémentaires $f'(u)$ sur cette chaine seront

$$f'(u) = \begin{cases} f(u) + \varepsilon, & u \in C^+ \\ f(u) - \varepsilon, & u \in C^- \end{cases}$$

On vérifie facilement que le nouveau flot est réalisable.

<u>Calcul :</u>

$$\varepsilon_1 = min\{4-1, 8-3\} = min\{3,5\} = 3$$

$$\varepsilon_2 = min\{2,3\} = min\{3,5\} = 2$$

$$\Rightarrow \ \varepsilon = \min(3,2) = 2$$

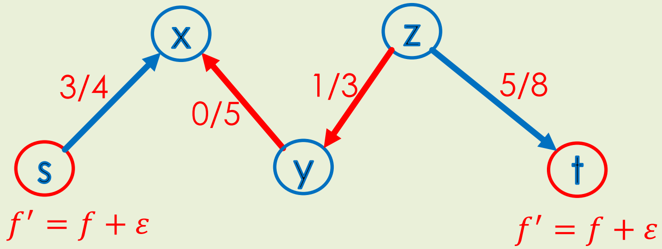
Les nouveaux flots élémentaires sur cette chaine sont

$f(s,x) = 1 + 2 = 3$

$f(y,x) = 2 - 2 = 0$

$f(z,y) = 3 - 2 = 1$

$f(z,t) = 3 + 2 = 5$



Coupe Minimale et Flot Maximum

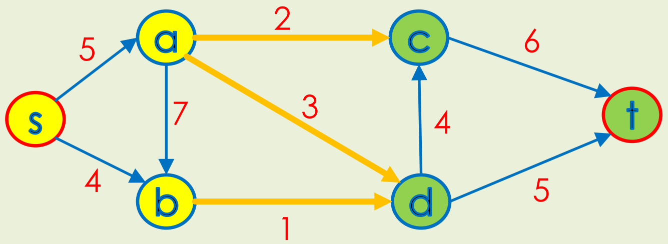
- Une coupe $(Y, \bar{Y})$ est définie par l'ensemble $Y \subset X$ et son compliment $\bar{Y}$ tel que
  - $Y \cup \bar{Y} = X$
  - $Y \cap \bar{Y} = \emptyset$
  - $s \in Y$ et $t \in \bar{Y}$
- La capacité d'une coupe est définie par

$$w(Y, \bar{Y}) = \sum_{x \in Y, y \in \bar{Y}} w(x, y) - \sum_{x \in Y, y \in \bar{Y}} w(y, x)$$

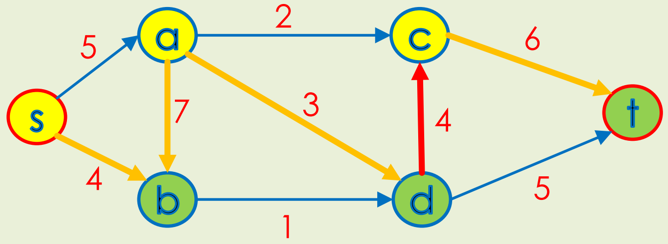
Théorème de Ford-Fulkerson : Soit $G = (X, U, W)$ un graphe valué. Pour tout flot réalisable F, de valeur f, et toute coupe $(Y, \bar{Y})$, on a

$$f \leq w(Y, \bar{Y}).$$

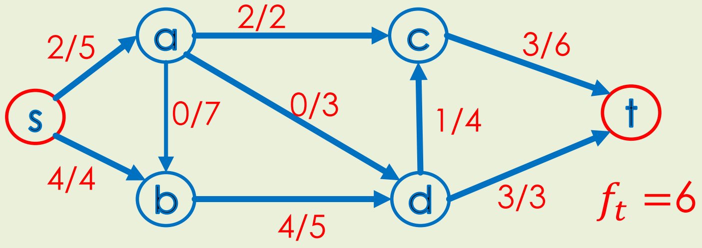
Corolaire : S'il existe un flot F^* et une coupe Y^* tel que $f^* = w(Y^*, \overline{Y^*})$ alors le flot F est maximal. Y^* est appelée coupe minimale.



- Soit la coupe $Y_1 = \{s, a, b\}$ et $\bar{Y}_1 = \{c, d, t\}$. La capacité de cette coupe est $w(Y_1, \bar{Y}_1) = w(a, c) + w(a, d) + w(b, d) = 2 + 3 + 1 = 6$



- Soit la capacité $Y_2 = \{s, a, c\}$ et $\bar{Y}_2 = \{b, d, t\}$. La capacité de cette coupe est $w(Y_2, \bar{Y}_2) = w(s, b) + w(a, b) + w(a, d) - w(d, t) + w(c, t) = 4 + 7 + 3 - 4 + 6 = 16$



- On peut verifier que la coupe $(Y_1, \bar{Y}_1)$ ci-dessous est minimale, donc le flot ci-dessous est maximal.

Parcours d'un Graphe

On parcourt un graphe pour de nombreuses raisons :

- Enumérer tous les sommets du graphe
- Enumérer tous les arcs du graphe
- Trouver un chemin, ou tous les chemins, entre 2 sommets x et y
- Détecter les composantes connexes d'un graphe.
- Etc.

<u>Algorithme DFS</u> : Parcours en profondeur d'un graphe

Input: $G(X,U), x$ // x sommet de départ

Output: C // une chaine des sommets parcourus

- Marquer sommet x comme visité

Pour tout arc (x,y) **Faire**

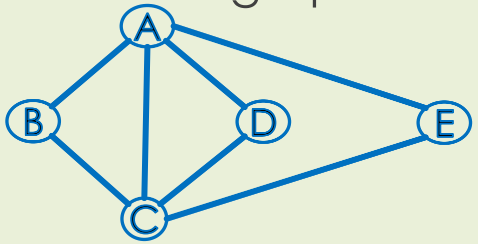
    Si le sommet y n'est pas encore visité **Alors**

        - marquer y
        - ajouter y à C comme poursuivant de x
        - DFS(y) // appel récursif de DFS

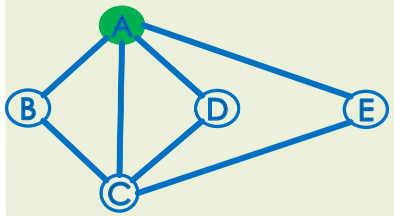
    Fin SI

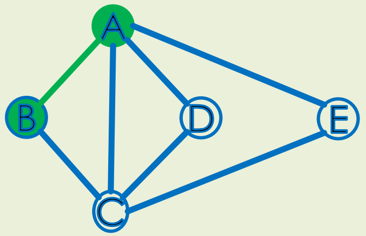
Fin Pour

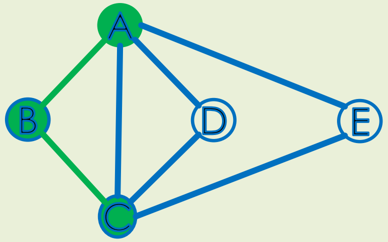
Exemple : Utiliser DFS pour trouver toutes les chaines liant le sommet A au sommet E du graphe suivant :

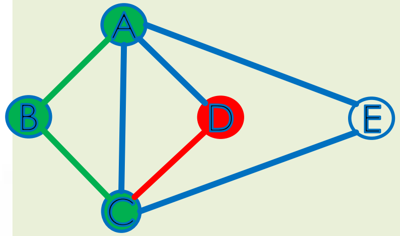


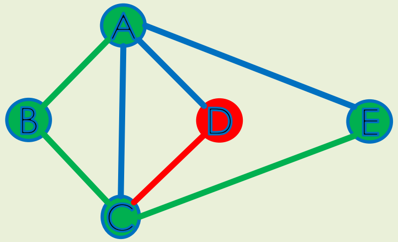
Solution



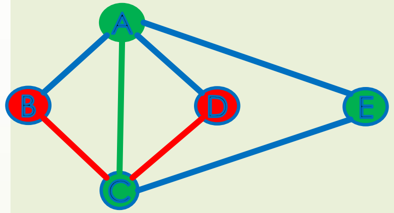




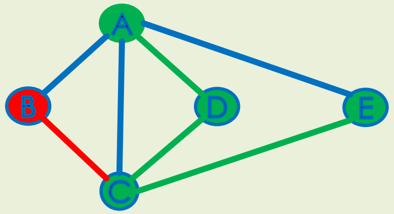




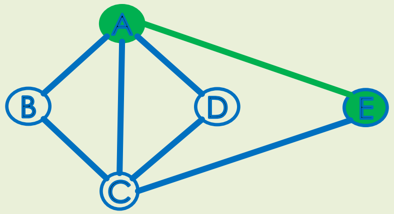
La 1iere chaine obtenue est (A,B,C, E)



La 2ime chaine obtenue est (A, C, E)



La 3ieme chaine obtenue est (A, D, C, E)



La 4ieme chaine obtenue est (A, E)

Algorithme du Flot Maximum

Algorithme Ford-Fulkerson : Flot Maximum

Input: G(X,U,W), s, t
Output: $F = [f(u),\ u \in U]$ et f // F matrice des flux passant par chacun des arcs

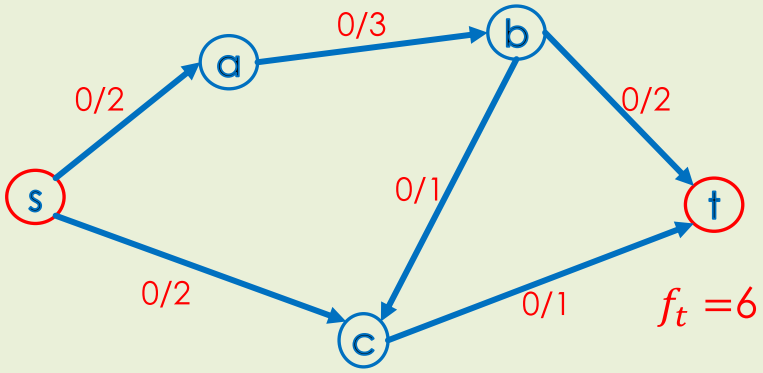
•$f(u) = 0,\ \forall\, u \in U,\quad f = 0$
Tant que il existe une chaine améliorante C **Faire**
// calculer l'augmentation du flot
$\varepsilon_1 = \min(w(u) - f(u)), u\epsilon C^+$
$\varepsilon_1 = \min f(u),\quad u\epsilon C^-$
$\varepsilon = \min(\varepsilon_1, \varepsilon_2)$
// améliorer le flot
$f = f + \varepsilon$
$f(u) = f(u) + \varepsilon,\ \ u\ \in C^+$
$f(u) = f(u) - \varepsilon,\ \ u\ \in C^-$
Fin Tant que

Remarque : La recherche des chaines améliorantes peut se faire à l'aide d'une méthode de parcours de graphe comme DFS.

- La complexité temporelle de l 'algorithme de Ford-Fulkerson est $O(mf)$ où $m = |U|$ et f est la valeur du flot maximum.
- Autres algorithme de flot maximum :
  - Algorithme de d'Edmonds-Karp : complexité $O(m^2n)$ (le chemin augmentant est choisi le plus court possible à chaque fois)
  - Algorithme de Golberg-Tarjan : complexité $O(n^3)$

...

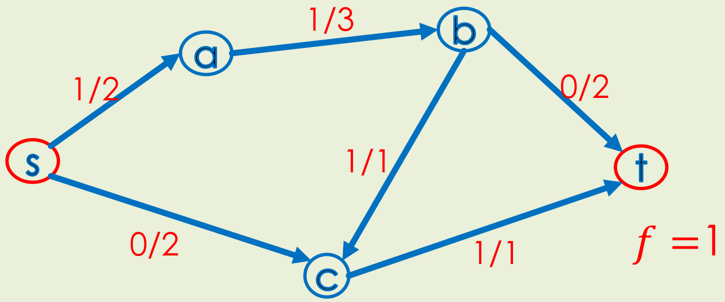
Exemple 1 : Maximiser le flot dans le graphe suivant



Solution

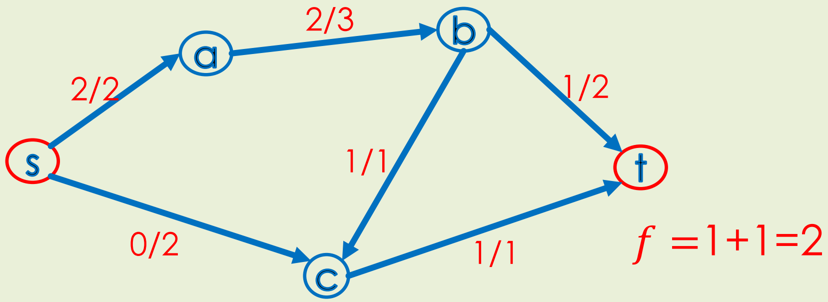
Itération 1 : On démarre par le sommet s et marque en profondeur (DFS) les sommets pour construire une chaine améliorante

Sommet	s	a	b	c	t
Origine	-	s	a	b	c
ε	∞	2	2	1	ε=1



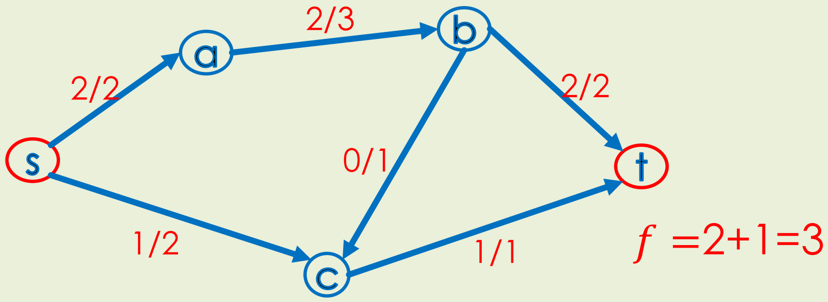
Itération 2 : Sur la dernière chaine parcourue, le plus proche sommet ayant un successeur non visité est b

Sommet	s	a	b	t
Origine	-	s	a	b
ε	∞	1	1	ε=1



Itération 3 : Sur la dernière chaine parcourue, le plus proche sommet ayant un successeur non visité est s:

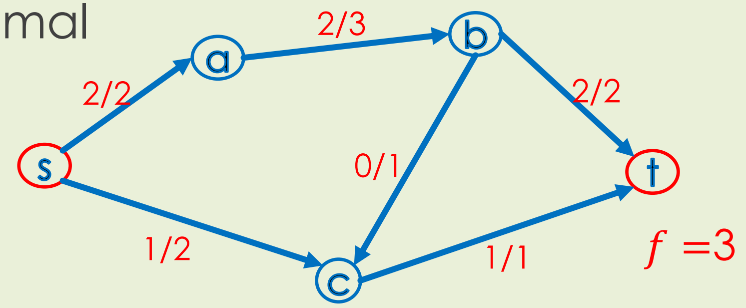
Sommet	s	c	b	a	t
Origine	-	s	-c	-b	b
ε	∞	2	1	1	ε=1



Itération 4 : Sur la dernière chaine parcourue, le plus proche sommet ayant un successeur non visité est c :

Sommet	s	c
Origine	-	s
ε	∞	2

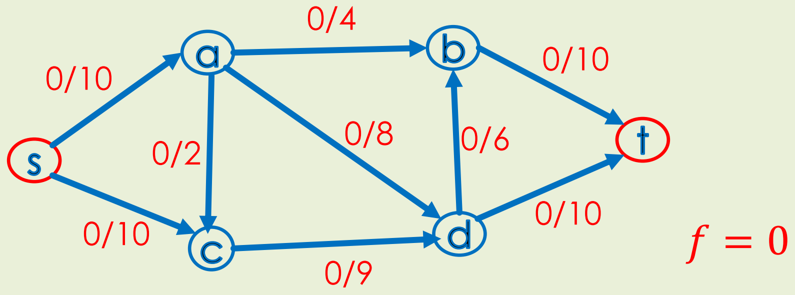
Il n y a plus de chaine améliorante, le flot obtenu est maximal



Remarque : On a la coupe $Y = \{s, a, b, c\}$, et $\bar{Y} = \{t\}$ avec $w(Y, \bar{Y}) = w(b, t) + w(c, t) = 2 + 1 = 3$.

Puisque $f = w(Y, \bar{Y})$, ceci confirme encore une fois que ce flot est maximal.

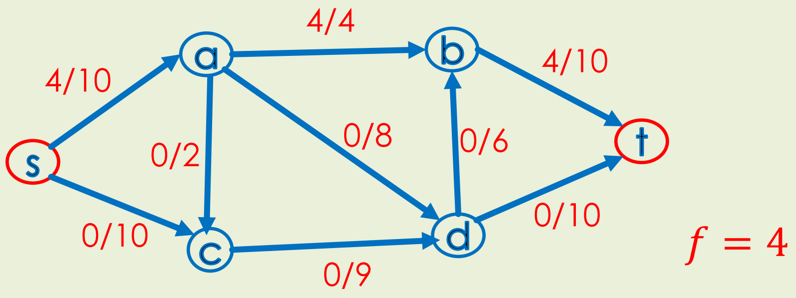
Exemple 2 : Calculer le flot max dans le réseau suivant :



Solution

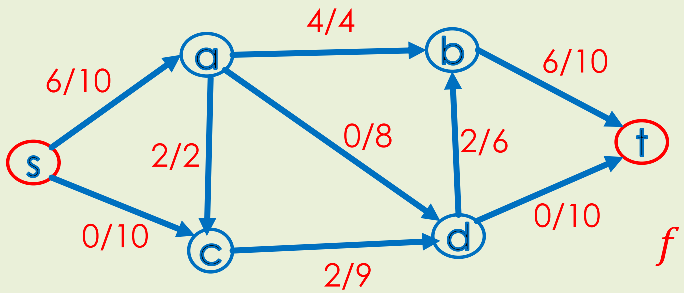
Itération 1 :

Sommet	s	a	b	t
Origine	-	s	a	b
ε	∞	10	4	$\varepsilon = 4$



Itération 2 :

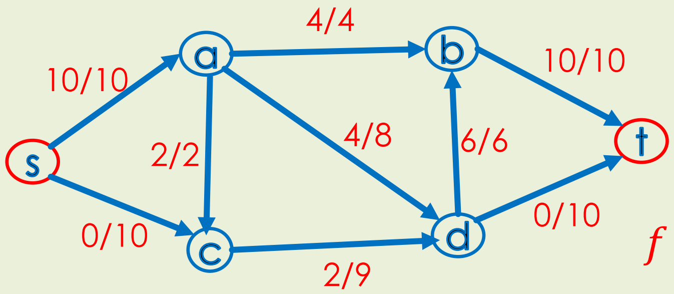
Sommet	s	a	c	d	b	t
Origine	-	s	a	c	d	b
ε	∞	6	2	2	2	$\varepsilon = 2$



$f = 4 + 2 = 6$

Itération 3 :

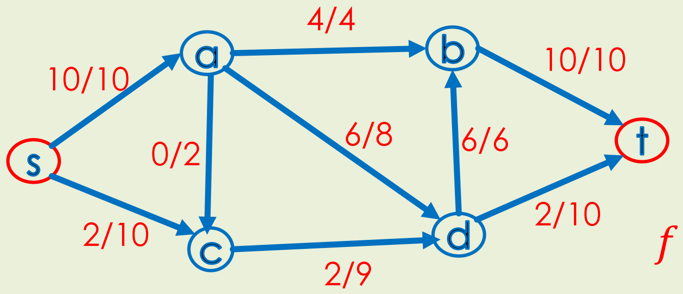
Sommet	s	a	d	b	t
Origine	-	s	a	d	b
ε	∞	4	4	4	$\varepsilon = 4$



$f = 6 + 4 = 10$

Itération 4 :

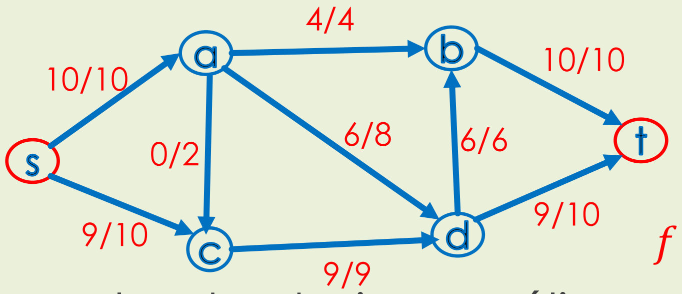
Sommet	s	c	a	d	t
Origine	-	s	-c	a	d
ε	∞	10	2	2	$\varepsilon = 2$



$f = 10 + 2 = 12$

Itération 5 :

Sommet	s	c	d	t
Origine	-	s	c	d
ε	∞	8	7	$\varepsilon = 7$



$f = 12 + 7 = 19$

Il n'y a plus de chaine améliorante, ce flot est max.

D'ailleurs la coupe $Y = \{s, a, c\}$ est de capacité 19.

5 – Méthodes d'Ordonnancement en Théorie des Graphes

Université Alger 1, Dept Maths & Informatique

Dr. Fodil LAIB

Mai 2017

Méthode PERT

- PERT = Program Evaluation and Review Technique.
- Méthode développée en 1958 aux USA.
- Elle permet d'optimiser la gestion des tâches d'un grand projet impliquant des milliers de personnes et de taches.
- La méthode PERT implique :
  - un découpage du projet en tâches précises;
  - l'estimation des durées de chaque tache;
  - détermination des prédécesseurs de chaque tache.
- La 1iere application a permis de réduire le délai de réalisation d'un projet de 7 à 4 ans.

Le graphe PERT :

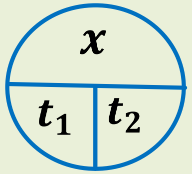
- représente les taches et les dépendances entre elles;
- affiche une date de début au plutôt et une date au plus tard pour chaque tâche;
- détermine un chemin critique qui conditionne la durée minimale du projet;
- montre les tâches critiques qu'il faut réaliser absolument dans leurs temps, sinon tout le projet sera retardé.

Une tâche non critique peut être retardée dans la limite de la marge calculée sans impacter la durée totale du projet.

Conventions de la Méthode PERT

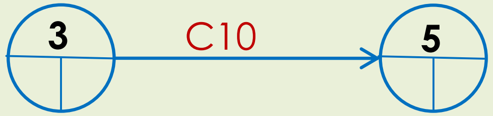
- Une tâche est représentée par une étiquette sur une flèche. L'étiquette indique le code de la tâche et sa durée.

- La tâche B ci-dessous dure 30 jours. On note $w(B) = 30$
- Une étape : représente le début où la fin d'une tâche. Elle est visualisée par un cercle ayant 3 portions :

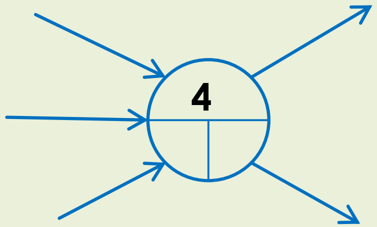


  - x : numéro de l'étape (facultatif)
  - t_1 : date au plutôt
  - t_2: date au plus tard

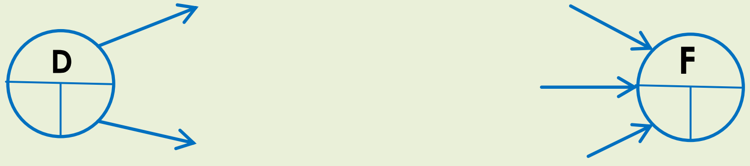
- La tâche C10 démarre à l'étape 3 et se termine à l'étape 5.



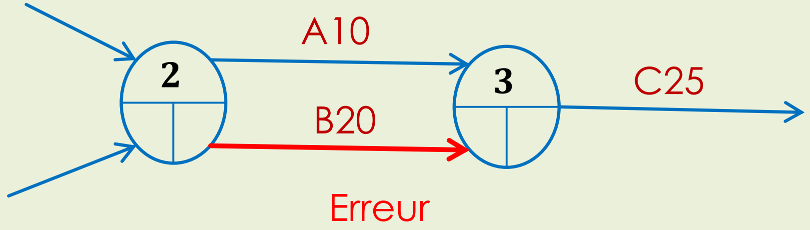
- Une Etape peut recevoir et émettre plusieurs tâches :



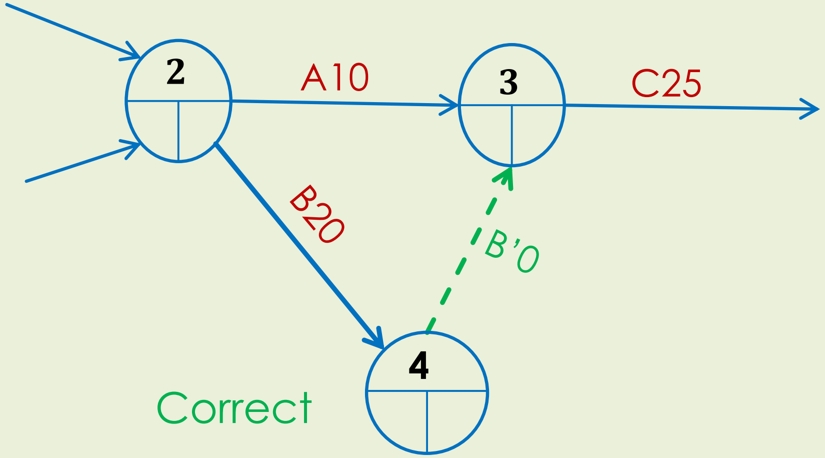
- Le graphe PERT démarre toujours par l'étape D (ou Début) et se termine par l'étape F (Fin).



- Deux tâches, ou leurs successeurs, ne peuvent pas être raccordées aux mêmes étapes de départ et de fin.



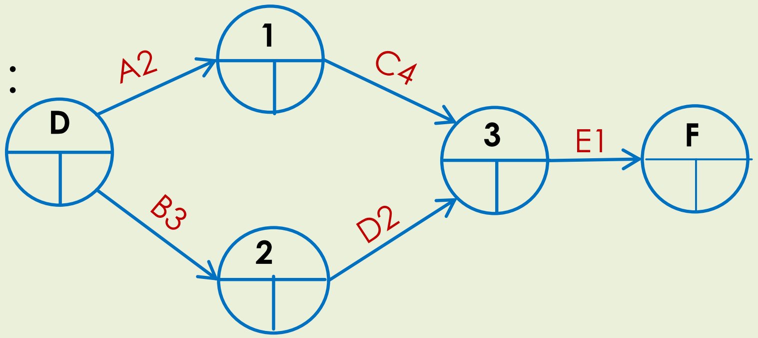
- Pour y remédier, il faut ajouter une tâche fictive en pointillés de durée 0.



Exemple 1 : Soit les taches suivantes d'un projet

Tâches	Prédécesseurs	Durées
A	-	2
B	-	3
C	A	4
D	B	2
E	C, D	1

Le graphe PERT correspondant est :

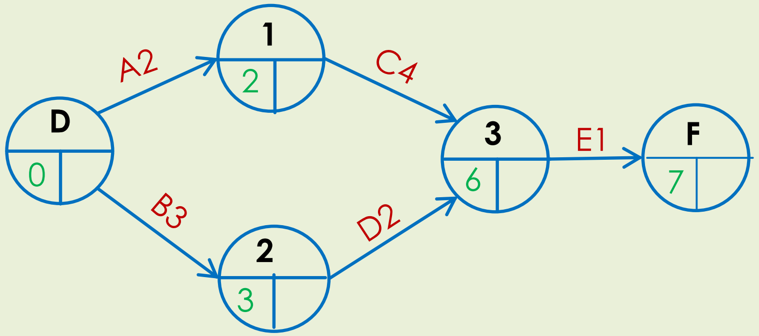


Sur le graphe, on constate bien que :

- Chaque tâche (arc) est précédée et suivie par une étape (sommet)
- les tâches A et B n'ont pas de prédécesseurs,
- la tâche C est précédée par A,
- la tâche D est précédée par B,
- la tâche E est précédée par C et D.

Calcul des dates au plutôt :

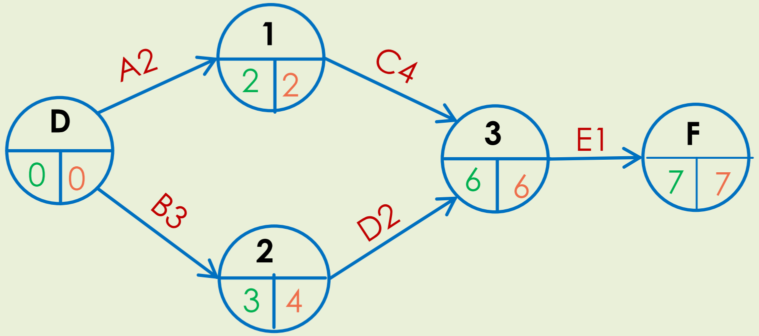
- Etape D, on pose $t_1(D) = 0$.
- Etape y : $t_1(y) = \max_{x \in d^-(y)} \{t_1(x) + w(x,y)\}$.



Etape 1 : 0 +2 = 2
Etape 2: 0 + 3 = 3
Etape 3 : max{2+4; 3+2} = 6
Etape F : 6+1=7

Calcul des date au plus tard :

- Etape F : $t_2(F) = t_1(F)$ (ici, $t_2(F) = 7$).
- Etape x : $t_2(x) = \min_{y \in d^+(x)} \{t_2(y) - w(x,y)\}$



Etape 3 : 7 - 1 = 6
Etape 2 : 6 - 4 = 2
Etape 1 : 6 - 4= 2
Etape D : min{2-2; 4-3} = 0

Construction d’un Réseau PERT

Exemple 2 :

Tableau des tâches : Exemple d’un projet de 7 tâches :

Tâches	Prédécesseurs	Durées
A	-	5
B	D	3
C	D	6
D	-	2
E	B, H	3
F	C	1
G	A	2
H	C	2

Remarque : A priori, ce projet peut durer 24 jours (la somme des durées.

Solution : D’abord, on détermine les niveaux et les successeurs de chaque tâche.

Les Niveaux du Graphe :

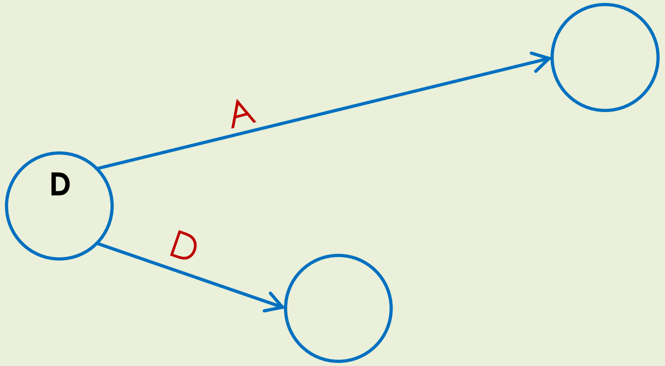
- Les tâches de **Niveau 1** sont celles qui n’ont pas de prédécesseurs.
- Les tâches de **Niveau 2** sont celles dont les prédécesseurs sont ceux de **Niveau 1**.
- Les tâches de **Niveau 3** sont celles dont les prédécesseurs sont ceux de **Niveau 2**. Etc.

Tâches	Prédécesseurs	Niveaux	Successeurs
A	-	Niveau 1	G
B	D^{1}	Niveau 2	E
C	D^{1}	Niveau 2	F, H
D	-	Niveau 1	B, C
E	B^{2}, H^{3}	Niveau 4	-
F	C^{2}	Niveau 3	-
G	A^{1}	Niveau 2	-
H	C^{2}	Niveau 3	E

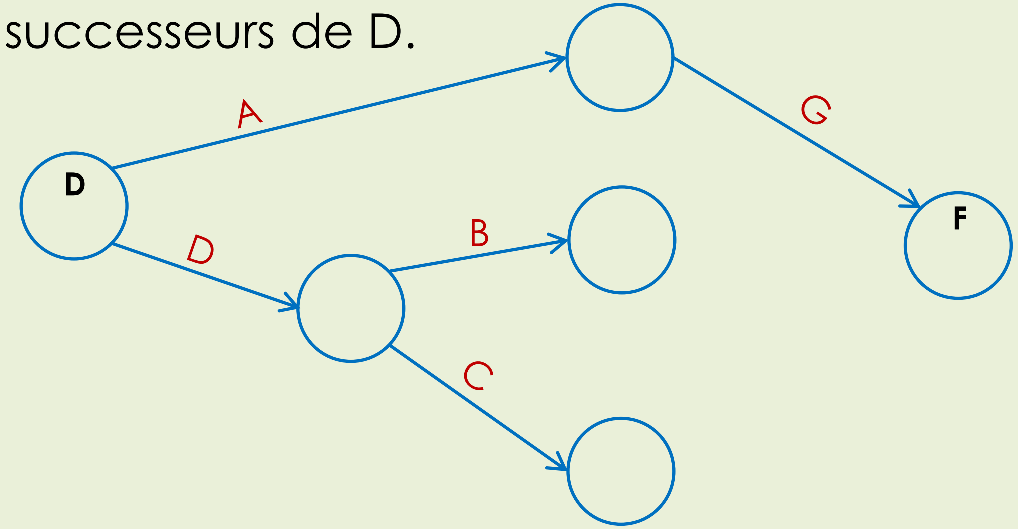
…

Construction progressive du graphe PERT :

1) On commence par les tâches de niveau1, soient A et D, qu'on raccorde au sommet Départ (D).

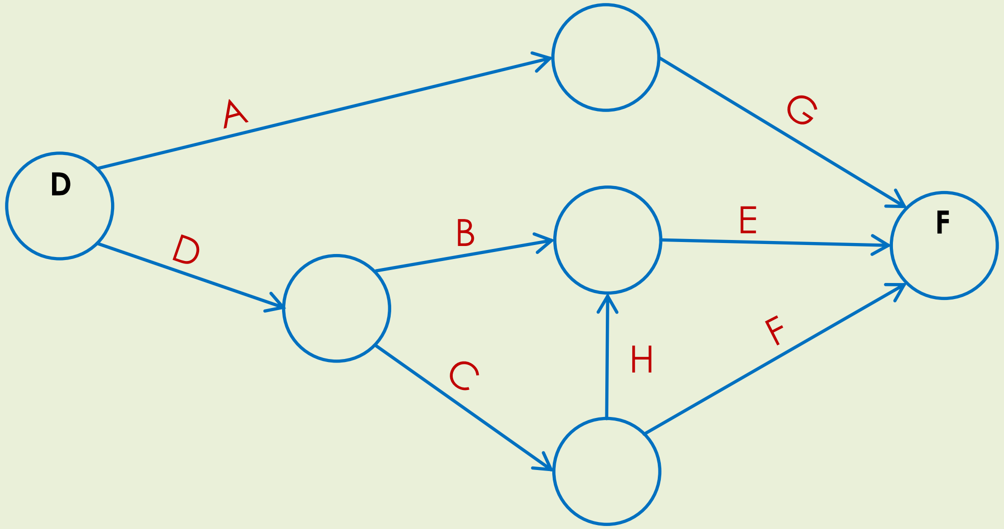


2) On ajoute les tâches de niveau 2, soient B, C et G, avec G étant le successeur de A; et B et C successeurs de D.

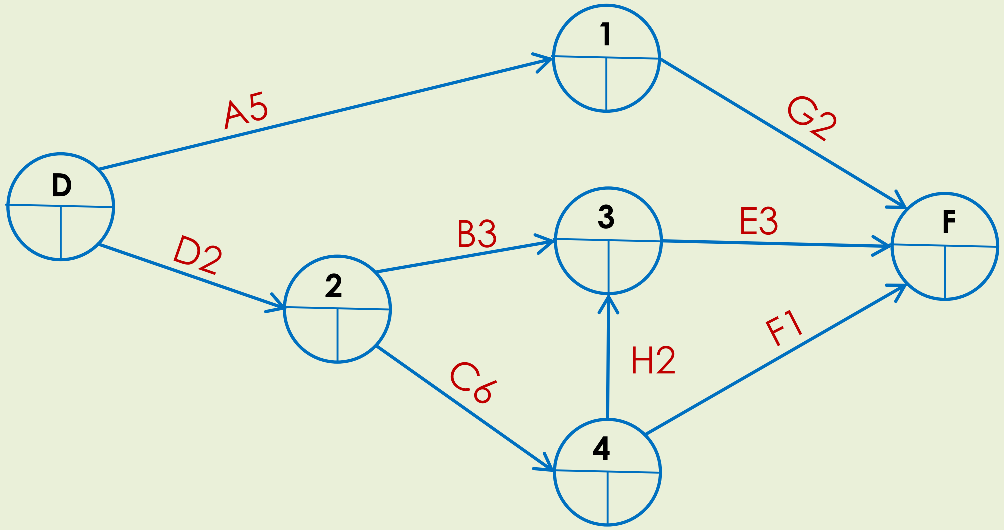


Attention : A chaque fois, on doit s'assurer que les tâches de niveau k-1 soient connectées à leurs successeurs de niveau k.

3) On ajoute les tâches de niveau 3, soient F et H comme successeurs de C. Puis on ajoute E (niveau 4) comme successeur de B et H.



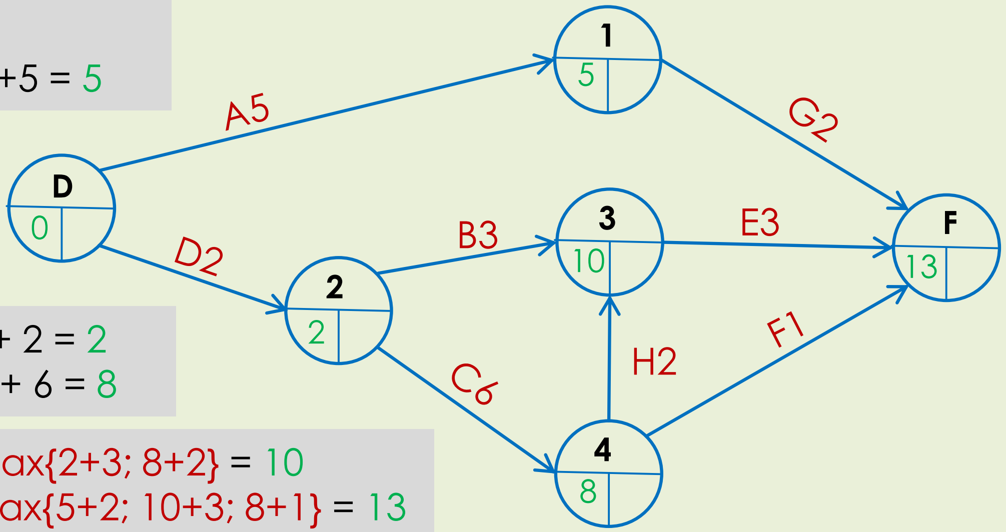
4) On affiche les durées de chaque tâche, et on numérote les étapes.



...

Calcul des dates au plutôt (t_1) :

Etape D : 0
Etape 1 : 0 +5 = 5

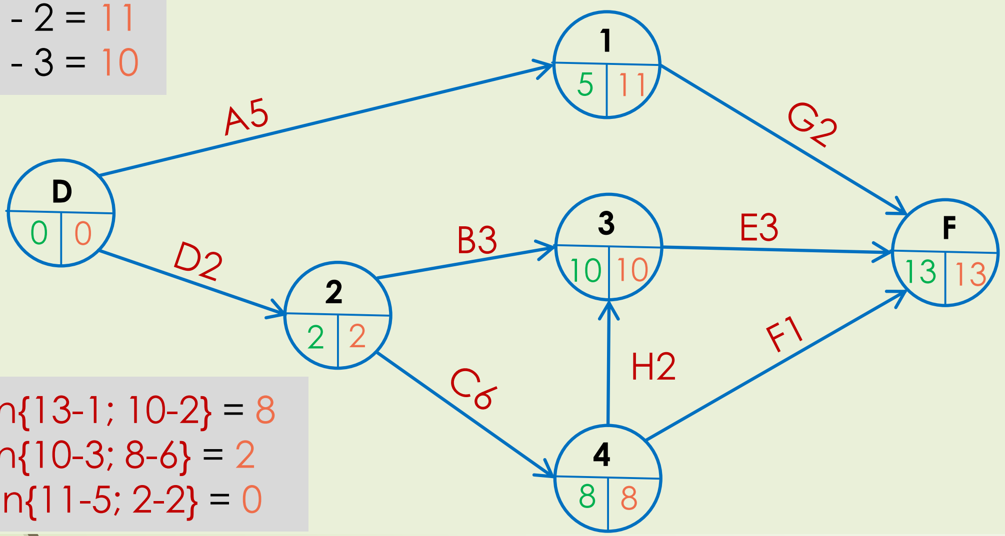


Etape 2: 0 + 2 = 2
Etape 4 : 2 + 6 = 8

Etape 3 : max{2+3; 8+2} = 10
Etape F : max{5+2; 10+3; 8+1} = 13

Calcul des dates au plus tard (t_2) :

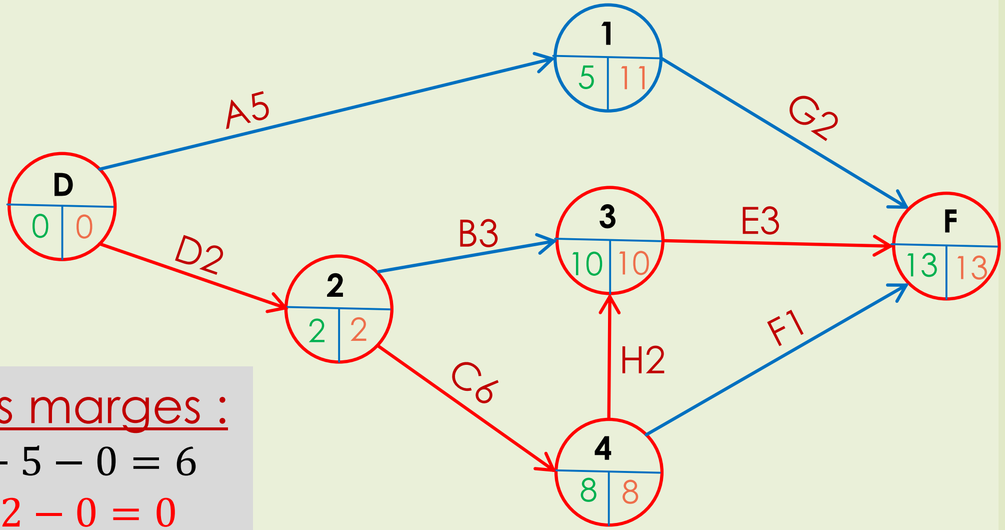
Etape F : 13
Etape 1 : 13 - 2 = 11
Etape 3 : 13 - 3 = 10



Etape 4 : min{13-1; 10-2} = 8
Etape 2 : min{10-3; 8-6} = 2
Etape D : min{11-5; 2-2} = 0

- Une **étape critique** x vérifie $t_1(x) = t_2(x)$
- La **marge** $m(u)$ d'une tâche u comprise entre les étapes x et y est : $m(u) = t_2(y) - w(u) - t_1(x)$

La tâche u commence au plutôt à $t_1\,(x)$, et au plus tard à $t_2\,(x)$. Elle se termine au plutôt à $t_1\,(y)$, et au plus tard à $t_2\,(y)$.



Calcul des marges :

$m(A) = 11 - 5 - 0 = 6$
$m(D) = 2 - 2 - 0 = 0$
$m(G) = 13 - 2 - 5 = 6$
$m(B) = 10 - 3 - 2 = 5$
$m(C) = 8 - 6 - 2 = 0$

$m(H) = 10 - 2 - 8 = 0$
$m(E) = 13 - 3 - 10 = 0$
$m(F) = 13 - 1 - 8 = 4$

Interprétation :

- La durée minimale du projet est 13 jours.
- Le chemin critique est composé des tâches D, C, H, E.
- Un retard sur ces tâches va retarder tout le projet.
- La tâche non critique A (resp G, B, F) peut être retarder de 6 jours (resp 6, 5, 4 jours) sans impacter la durée du projet.

Exemple 3 : Utiliser la méthode PERT pour optimiser la gestion du projet suivant:

Tâches	Prédécesseurs	Durées
A	-	6
B	-	2
C	A	3
D	C	4
E	A	1
F	C, E	7
G	A, B	2
H	G	3
I	D, F	4

Somme des durée = 32 jours

Solution :

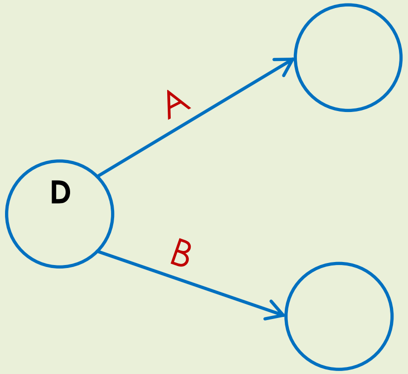
Tableau des niveaux et des successeurs :

Tâches	Prédécesseurs	Niveaux	Successeurs
A	-	Niveau 1	C, E, G
B	-	Niveau 1	G
C	A^1	Niveau 2	D, F
D	C^2	Niveau 3	I
E	A^1	Niveau 2	F
F	C^2, E^2	Niveau 3	I
G	A^1, B^1	Niveau 2	H
H	G^2	Niveau 3	-
I	D^3, F^3	Niveau 4	-

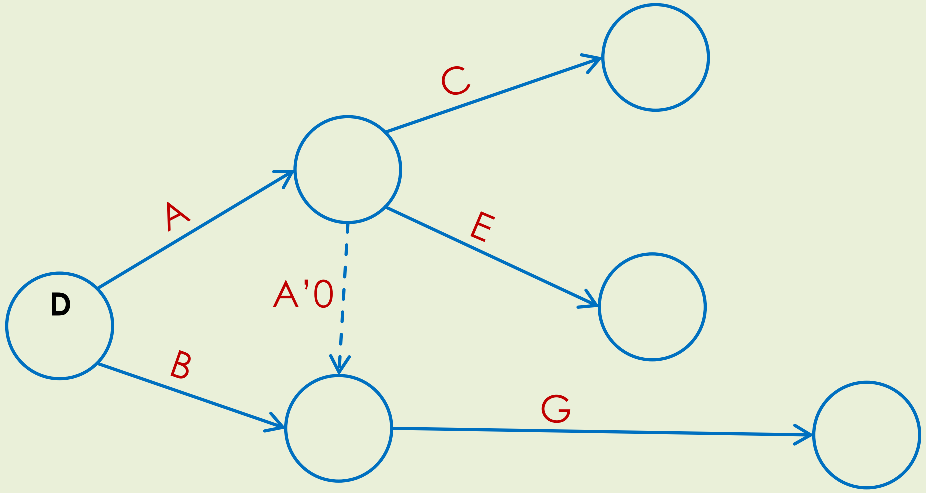
…

Graphe PERT :

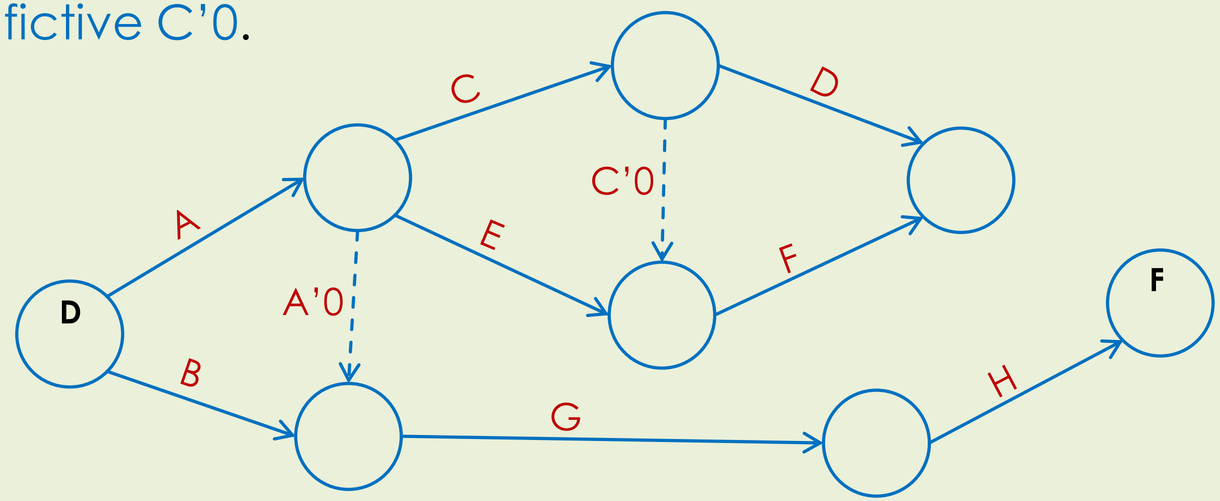
1) On commence par les tâches de niveau 1, c-à-d A et B.



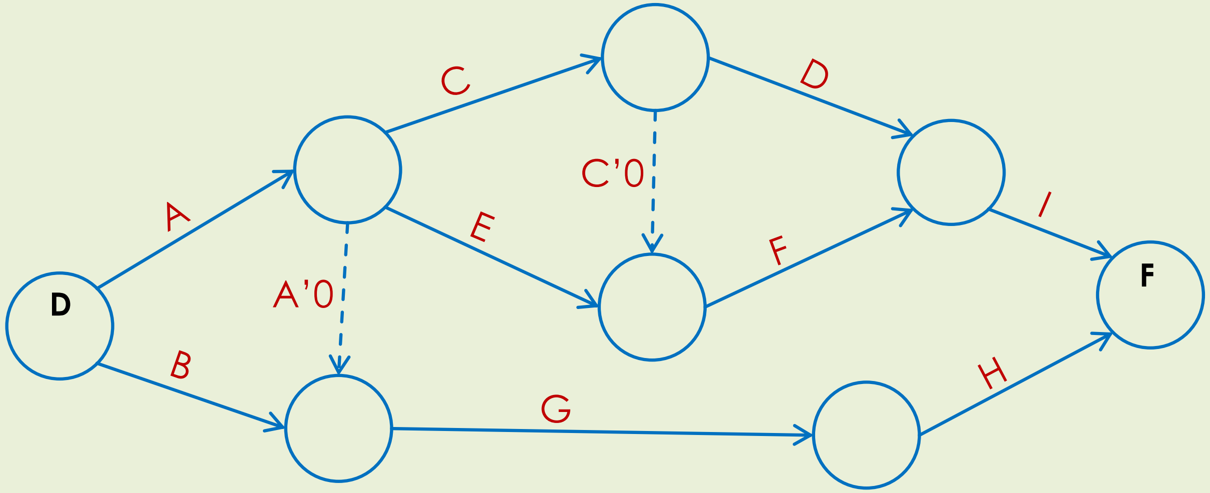
2) On passe aux tâches de niveau 2, c-à-d C, E et G. On a G successeur de A, on les lie par une tâche fictive A'0.



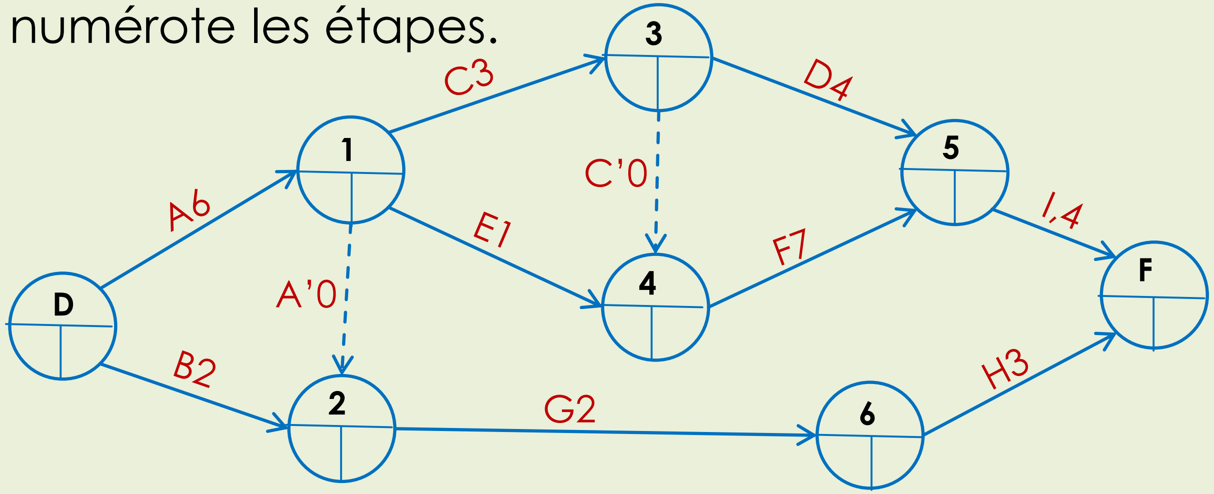
3) On passe aux tâches de niveau 3, c-à-d D, F et H. On a F successeur de C, on les lie par une tâche fictive C'0.



4) On passe à la tâche I de niveau 4, qui est le successeur de D et F.



5) On affiche les durées de chaque tâche, et on numérote les étapes.



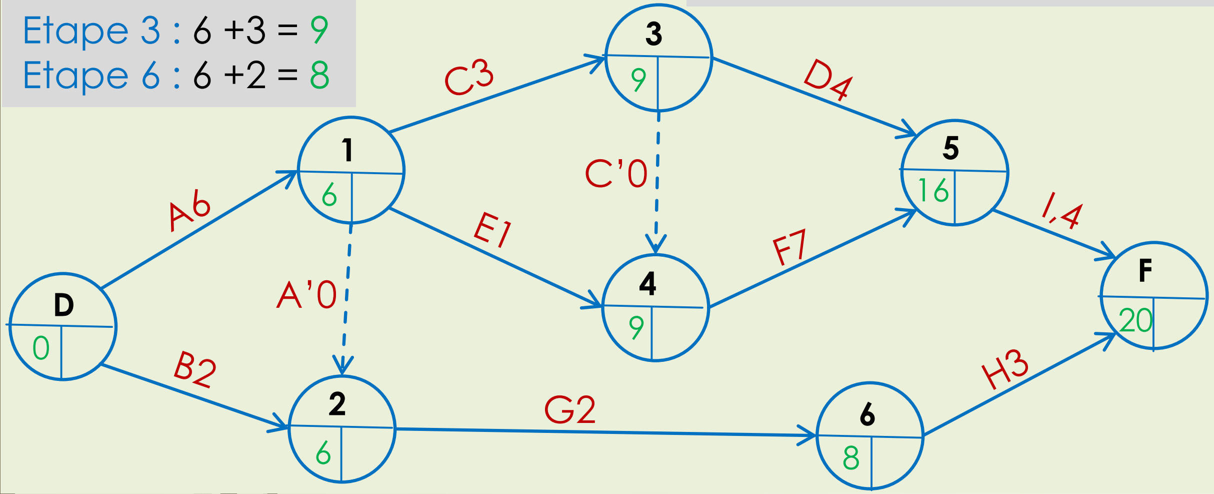
Calcul des dates au plutôt (t_1) :

Etape D : 0
Etape 1 : 0 +6 = 6
Etape 2 : max{6+0; 0+2} = 6

Etape 3 : 6 +3 = 9
Etape 6 : 6 +2 = 8

Etape 4 : max{6+1; 9+0} = 9
Etape 5 : max{9+4; 9+7} = 16

Etape F : max{16+4; 8+3} = 20

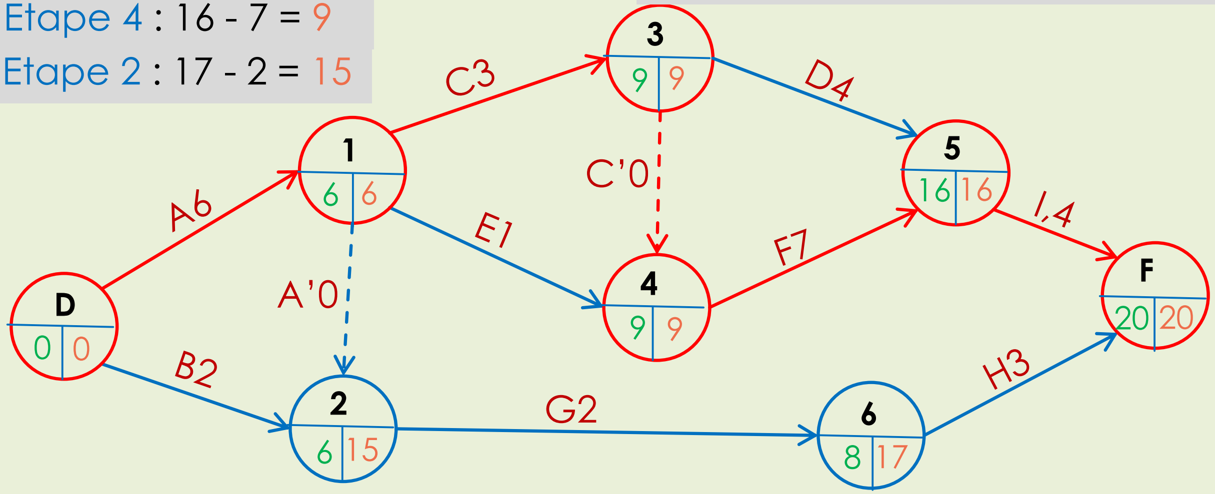


Calcul des dates au plus tard (t_1) :

Etape F : 20
Etape 6 : 20 - 3 = 17
Etape 5 : 20 - 4 = 16
Etape 4 : 16 - 7 = 9

Etape 2 : 17 - 2 = 15

Etape 3 : min{16-4; 9-0} = 9
Etape 1 : min{9-3; 9-1; 15-0} = 6
Etape D : min{6-6; 15-2} = 0



Calcul des marges :

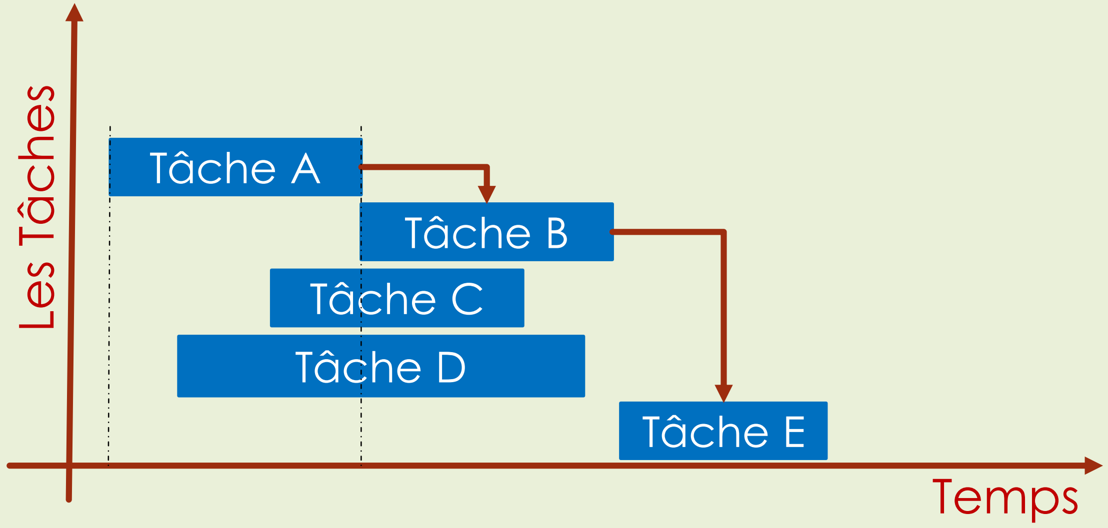
$m(A) = 6 - 6 - 0 = 0$
$m(B) = 15 - 2 - 0 = 13$
$m(C) = 9 - 3 - 6 = 0$
$m(D) = 16 - 4 - 9 = 3$
$m(E) = 9 - 1 - 6 = 2$
$m(F) = 16 - 7 - 9 = 0$
$m(G) = 17 - 2 - 6 = 9$
$m(H) = 20 - 3 - 8 = 9$
$m(I) = 20 - 4 - 16 = 0$

Interprétation :

- La durée minimale du projet est 20 jours.
- Le chemin critique est composé des tâches A, C, F, I.
- Les tâches non critiques sont B, D, E, G et H. Elles disposent respectivement des marges 13, 3, 2, 9 et 9 jours.

Diagramme de Gantt (1910)

- Le diagramme de Gantt permet de visualiser sur un plan à deux dimensions le déroulement des tâches d'un projet.
- Cette méthode suppose connus :
  - les dates de début de chaque tâche,
  - les durées de chaque tâche.
- Parfois, on peut ajouter les dépendances entre les tâches.

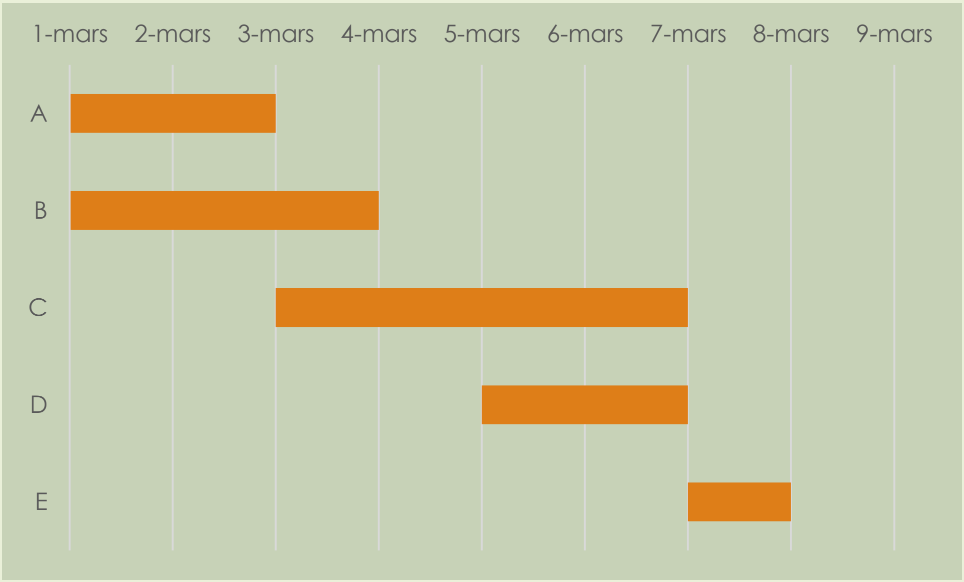


Sur ce diagramme, la tâche B ne peut commencer avant la fin de A, et E commence après la terminaison de B.

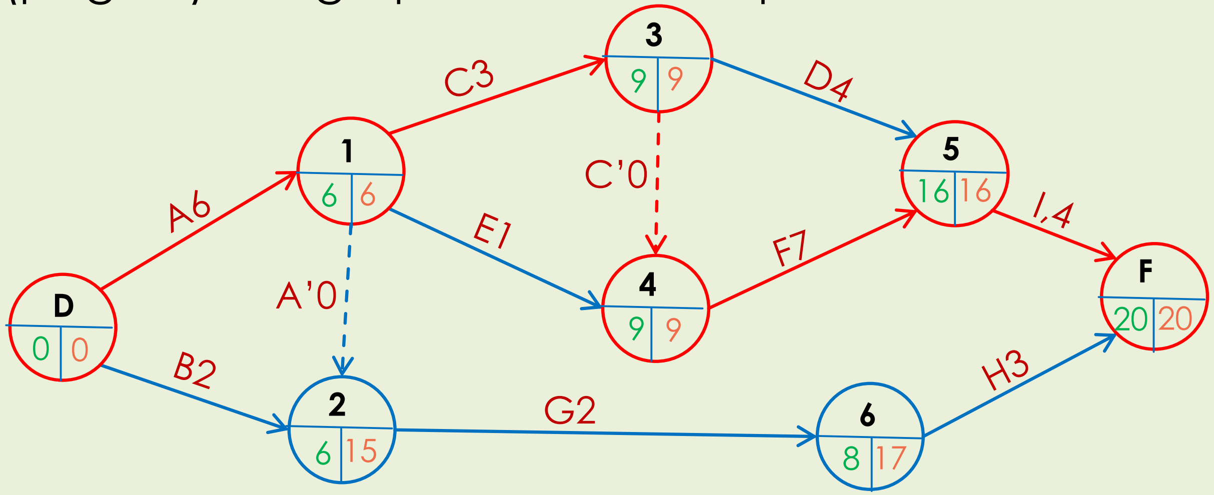
Exemple 4 : Soit un projet dont les dates de débuts et les durées sont comme suit :

Tâches	Débuts	Durées	Fin
A	01/03/2017	2	03/03/2017
B	01/03/2017	3	04/03/2017
C	03/03/2017	4	07/03/2017
D	05/03/2017	2	07/03/2017
E	07/03/2017	1	08/03/2017

Le diagramme de Gantt correspondant est :



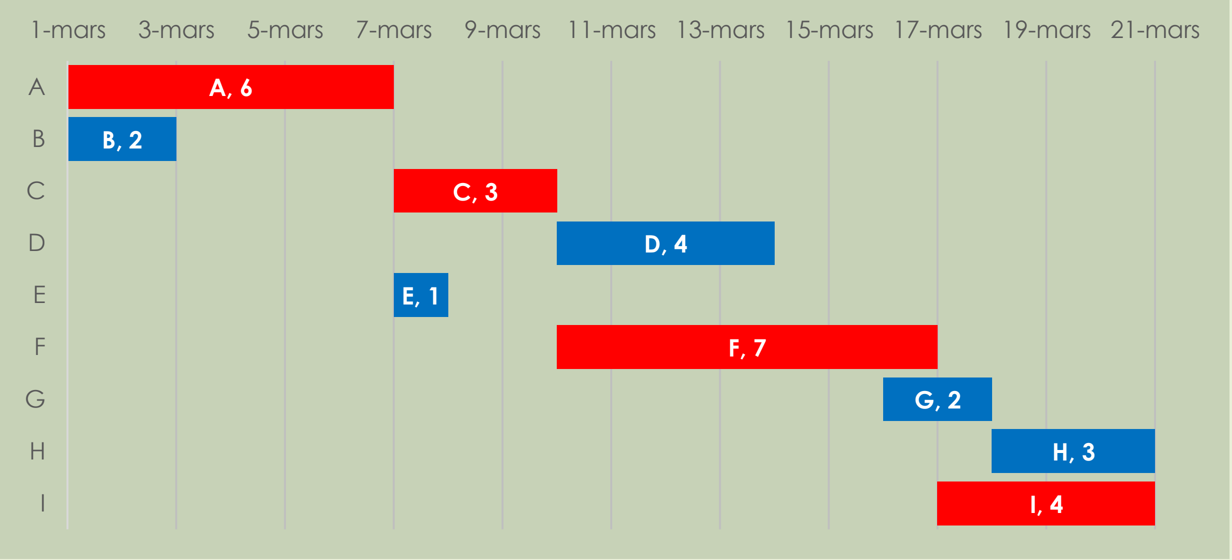
Exemple 5 : Reprenons les données de l'exemple 3 (page 8). Le graphe PERT correspondant est :



Considérons que le projet démarre le 01/03/2017.
La date de Début de chaque tâche correspond à t_2 de l'étape qui la précède.

Tâches	Prédécesseurs	Durées	t_2	Débuts
A	-	6	0	01/03/2017
B	-	2	0	01/03/2017
C	A	3	6	07/03/2017
D	C	4	9	10/03/2017
E	A	1	6	07/03/2017
F	C, E	7	9	10/03/2017
G	A, B	2	15	16/03/2017
H	G	3	17	18/03/2017
I	D, F	4	16	17/03/2017

Le diagramme de Gantt correspondant (réalisé avec Excel) est le suivant :



- La tâche critique A doit absolument démarrer le 01/03/2017 pour éviter un retard au projet.
- La tâche non critique B peut commencer à n'importe quel jour, l'essentiel est qu'elle se termine au plus tard le 07/03/2017 afin d'éviter tout retard au projet.