Théorie des graphes

Université de Jijel

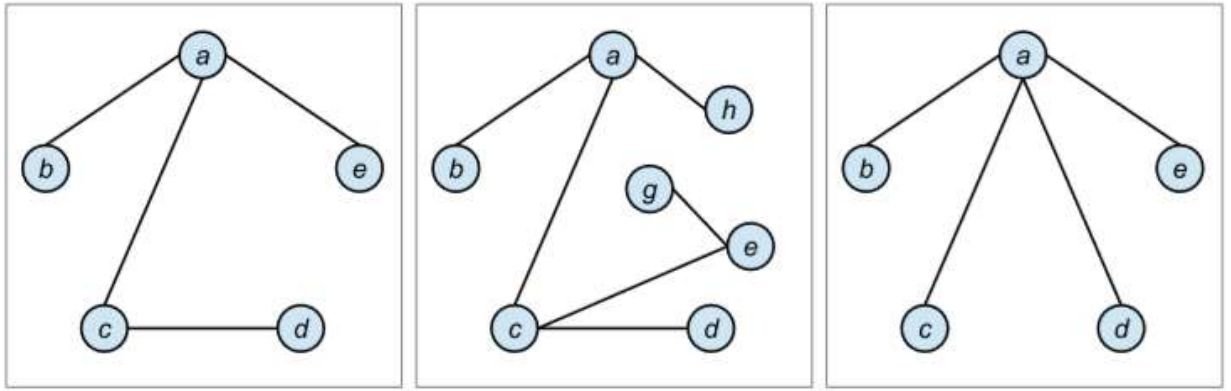
Mr.HEMIOUD

- Introduction aux théorie des graphes
- Notions de base
- Problème de coloration d'un graphe
- **Arbre Couvrant de poids Minimal : ACM**
  - Algorithme de Kruskal
  - Algorithme de Prim
- **Problème du plus court chemin**
  - algorithme de Dijkstra (source unique)
  - algorithme de Bellman-Ford (source unique)
  - algorithme de Floyd-Warshall (tous les PCC)
- **Le Problème du Flot Maximal**
  - Les réseaux de transport
  - Le flot maximum et la coupe minimum
  - L'algorithme de Ford et Fulkerson
- **le Réseau PERT**

Problèmes d'ACM et de cheminement

Arbre.

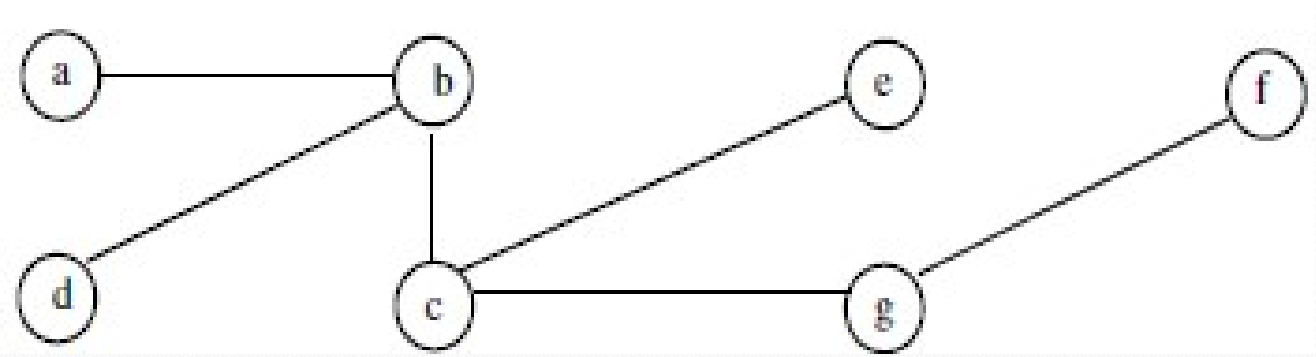
- Un **arbre** est un graphe qui peut être défini par l'une de propositions suivantes :
  - un arbre est un graphe **connexe** et **acyclique**;
  - un arbre est un graphe **connexe** avec **n–1 arêtes**;
  - un arbre est un graphe **acyclique** avec **n–1 arêtes**;
  - un arbre est un graphe dans lequel il existe une **chaine unique entre chaque paire de sommets**.



(a) Un premier arbre (b) Une deuxième arbre (c) Un troisième arbre

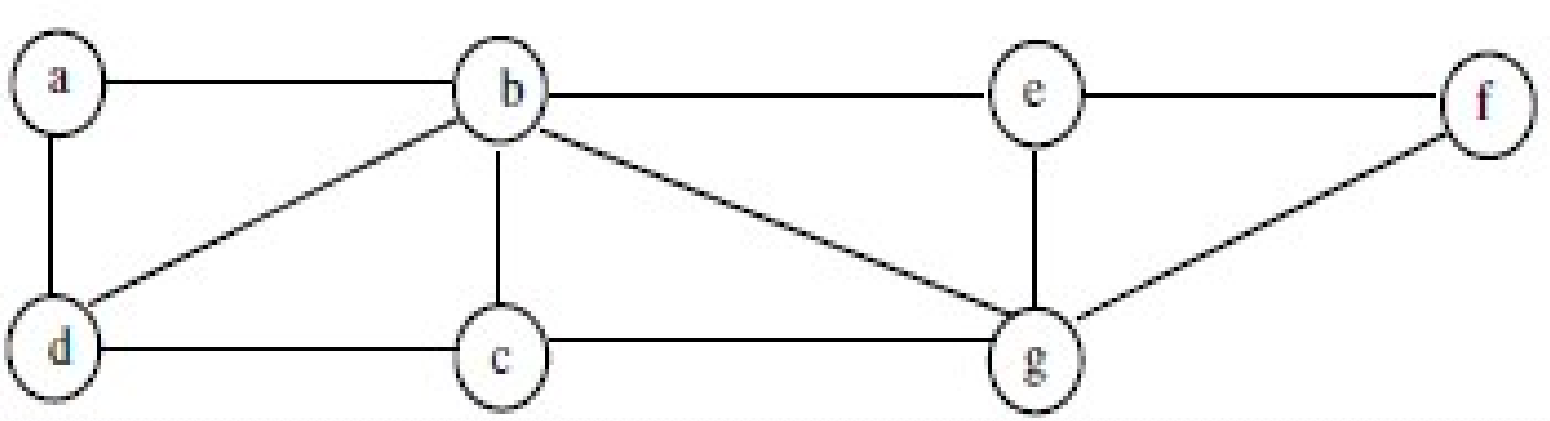
Arbre.

- Par exemple, le graphe suivant est un arbre :



- On appelle forêt un graphe dont chaque composante connexe est un arbre.

Exercice : On considère le graphe non orienté suivant :



- Combien faut-il enlever d'arêtes à ce graphe pour le transformer en arbre ? Donnez un graphe partiel de ce graphe qui soit un arbre.

Arbre couvrant de poids minimal : ACM

Le problème de l'arbre couvrant minimal

Soit $G = (E, U, V)$ un graphe non orienté, connexe et valué.

$$V = \{v(i,j) \,/\, v(i,j) = \text{ coût de l'arête } (i,j)\}$$

Le problème de l'arbre couvrant minimal de G consiste à trouver un arbre couvrant de G dont le coût total des arêtes est minimal.
Si G n'est pas connexe, on peut calculer une forêt couvrante minimale.

- Algorithme de Kruskal;
- Algorithme de Prim

Exemple d'application

Construction de lignes électriques : Soit une ensemble de n sites qui doivent être reliés par des lignes à haute tension. Le coût d'une ligne joignant deux sites est proportionnel à la distance entre ces deux sites.

Objectif : Construire à coût minimal un réseau connectant tous les sites.

Modélisation à l'aide d'un graphe : $G = (E, U, V)$

- E : ensemble des sites
- U : l'ensemble de toutes les connections possibles entre les différents sites
- V : le coût des différentes connections

Le réseau optimal est l'arbre couvrant minimal de G.

Algorithme de Kruskal
Algorithme de Prim

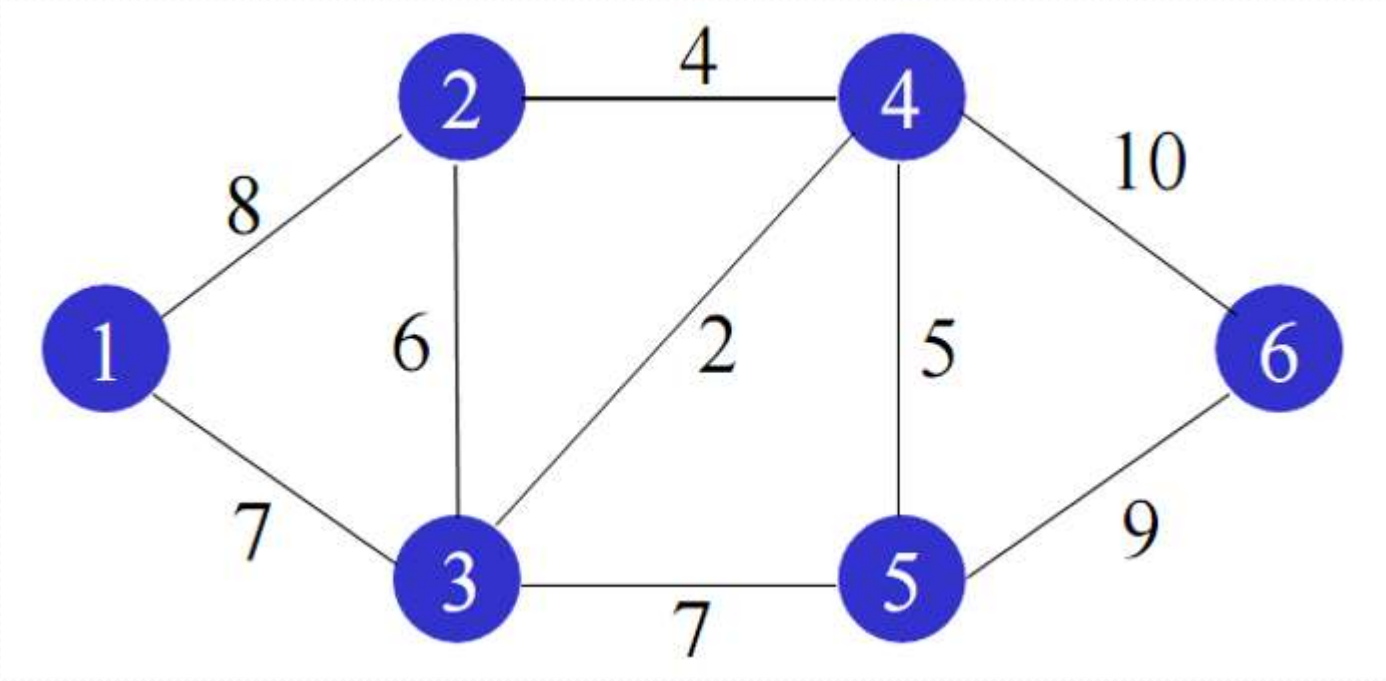
Algorithme de Kruskal

Algorithme

```
1:  procedure KRUSKAL
2:      ∀i ∈ E, cc(i) ← i
3:      Trier les arêtes
4:      T ← ∅
5:      for k = 1 à n − 1 do
6:          Choisir une arête (x, y)
7:          if cc(x) ≠ cc(y) then
8:              T = T ∪ {(x, y)}
9:              for all sommet i do
10:                 if cc(i) = cc(x) then
11:                     cc(i) ← cc(y)
12:                 end if
13:             end for
14:         end if
15:     end for
16: end procedure
```

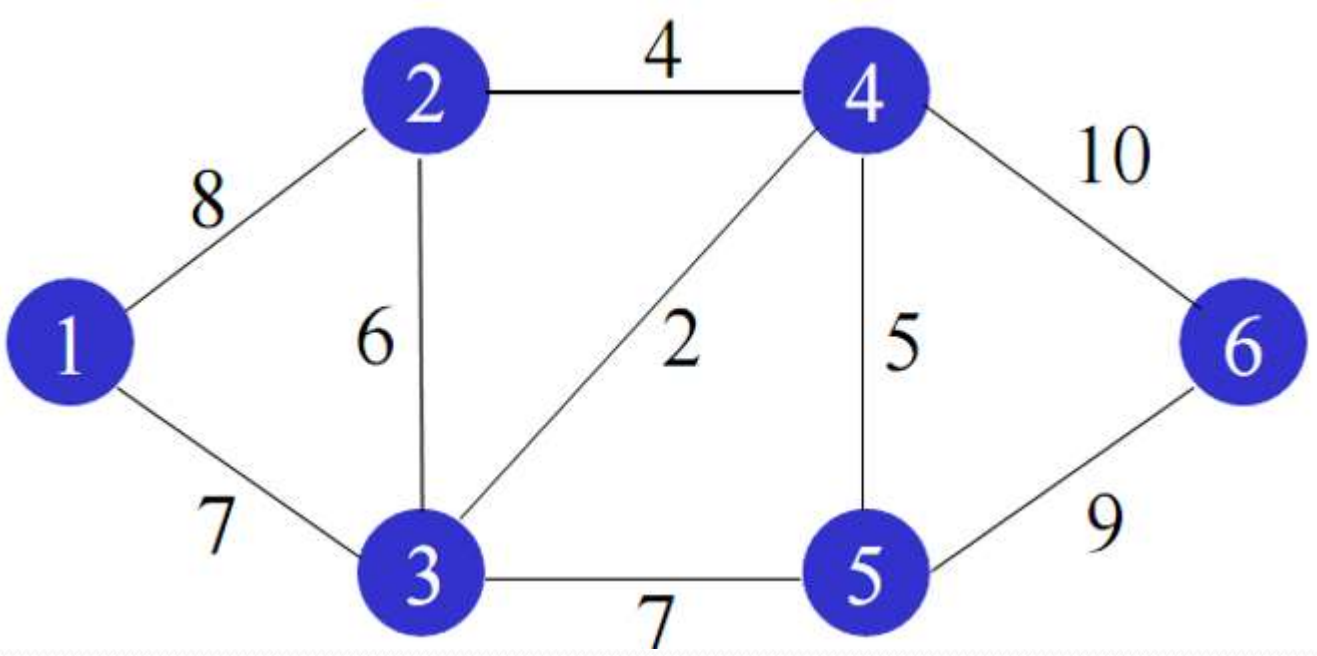
Algorithme de Kruskal

1. Trier les arcs en ordre croissant de poids ;
2. Construire un arbre en sélectionnant les arcs selon l'ordre établi à l'étape 1.



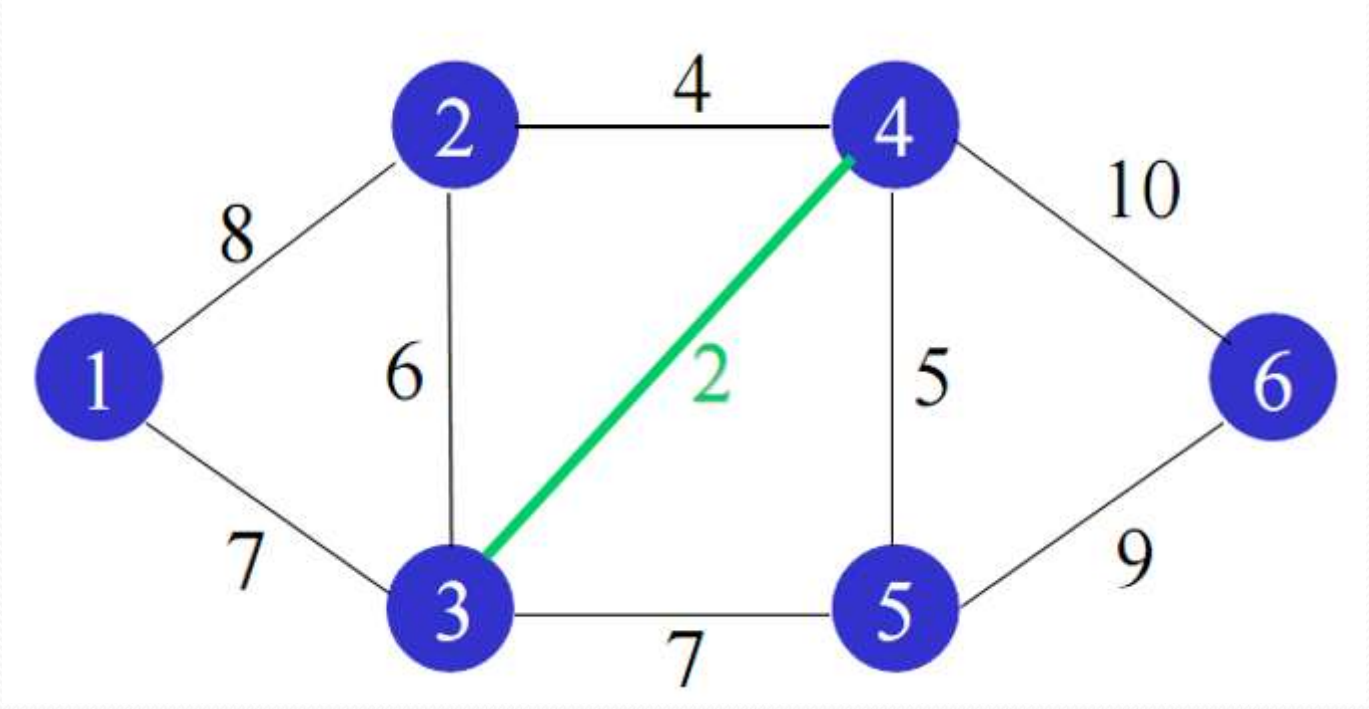
Algorithme de Kruskal

1. Trier les arcs { (3,4), (2,4), (4,5), (2,3), (1,3), (3,5), (1,2), (5,6),(4,6) }
2. Construire un arbre T = { }

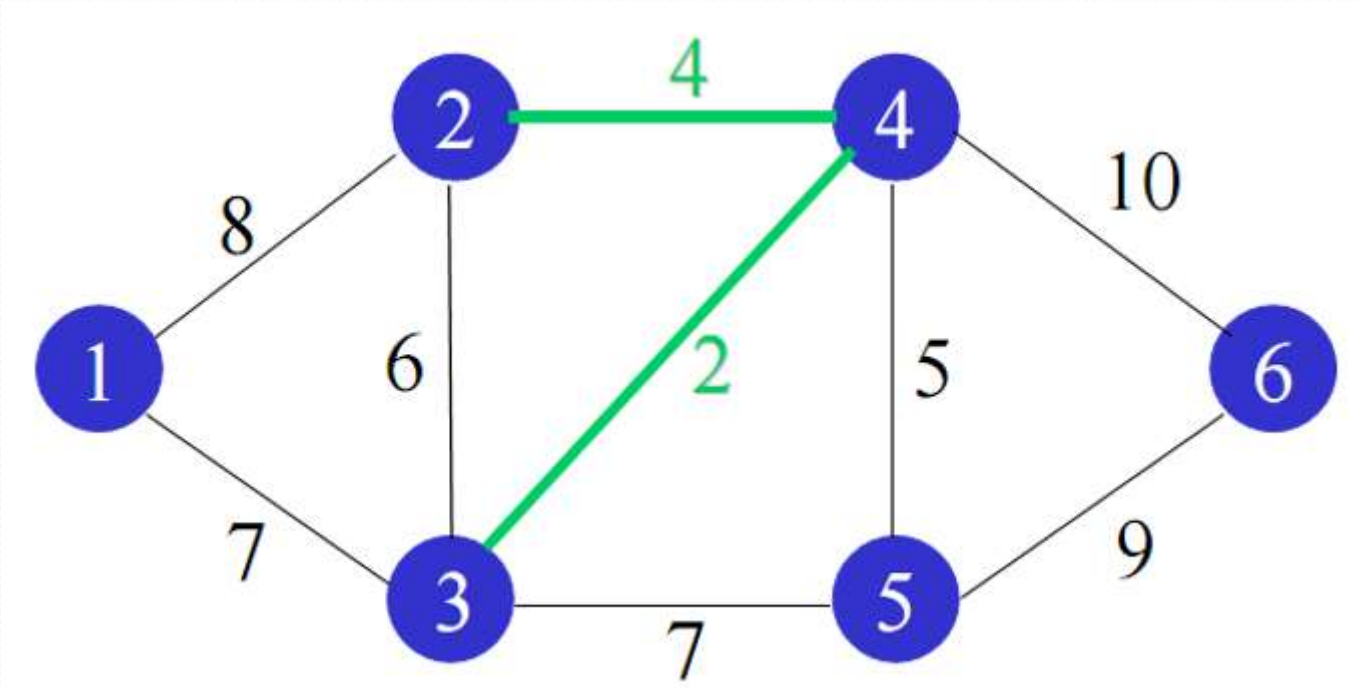


- On évalue (3,4) :

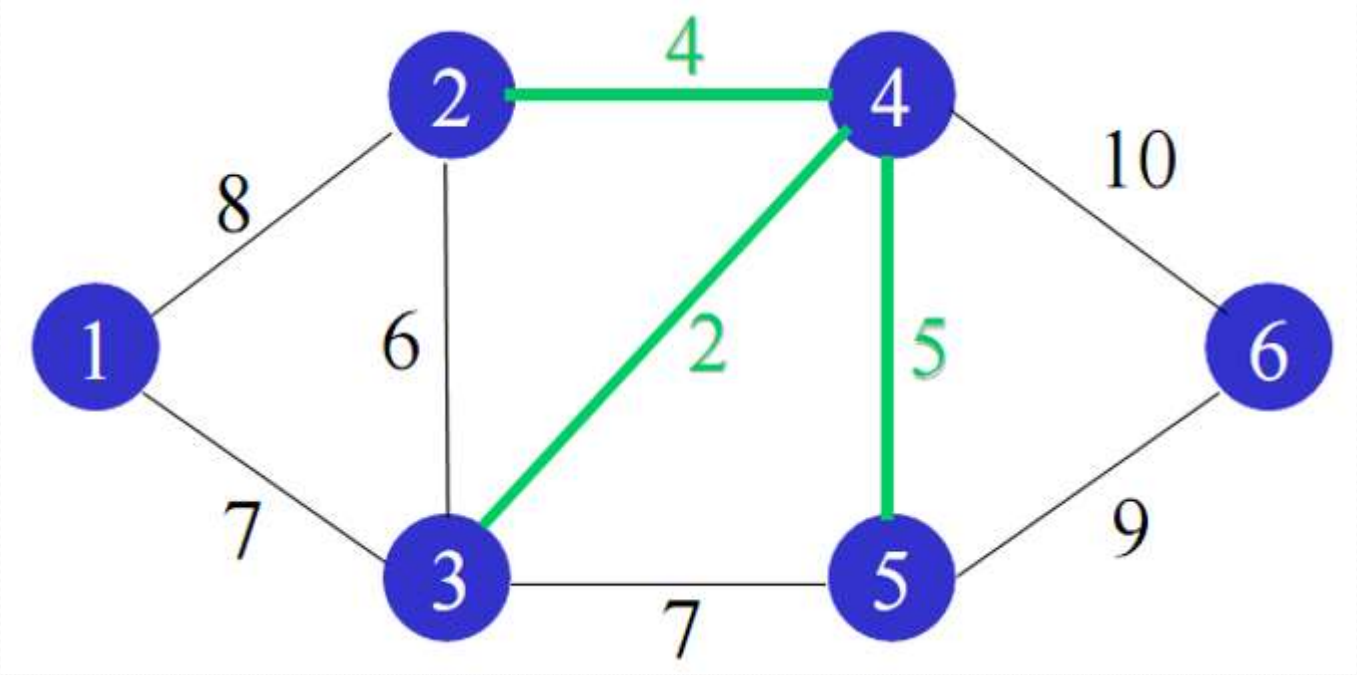
{ ~~(3,4)~~, (2,4), (4,5), (2,3), (1,3), (3,5), (1,2), (5,6),(4,6) }
T= { (3,4) }



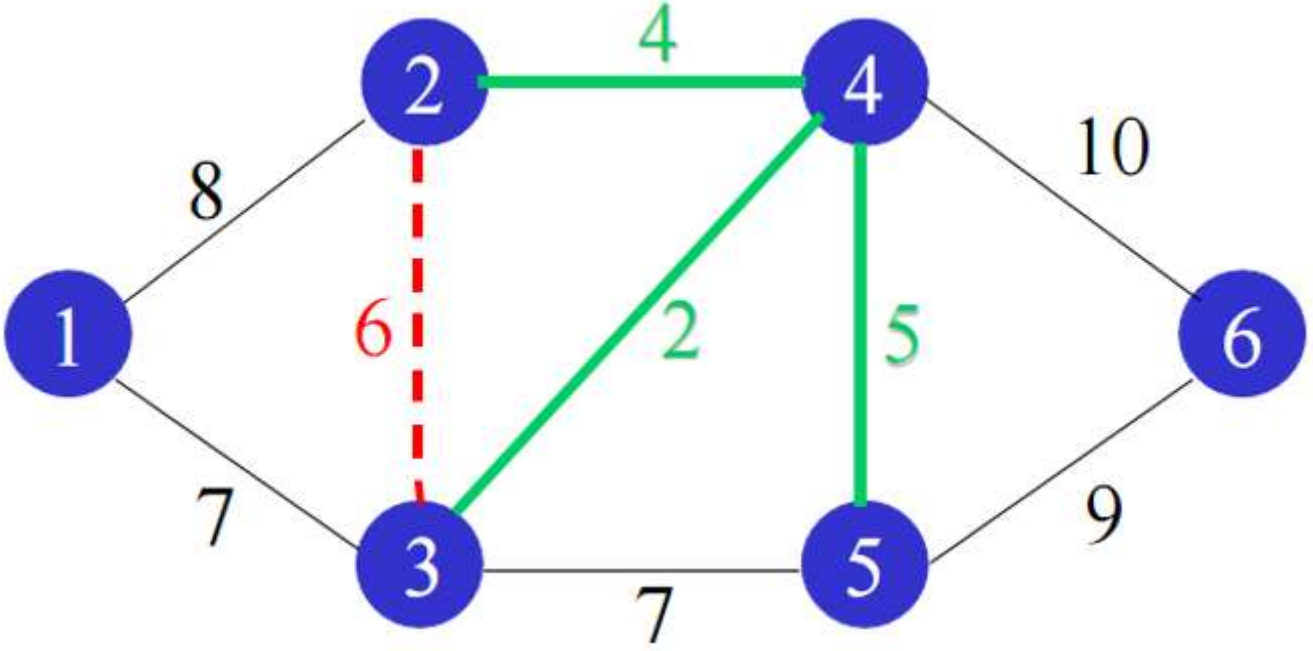
- On évalue (2,4) :
- {~~(3,4)~~, ~~(2,4)~~, (4,5), (2,3), (1,3), (3,5), (1,2), (5,6),(4,6) }
  T= {(3,4), (2,4)}



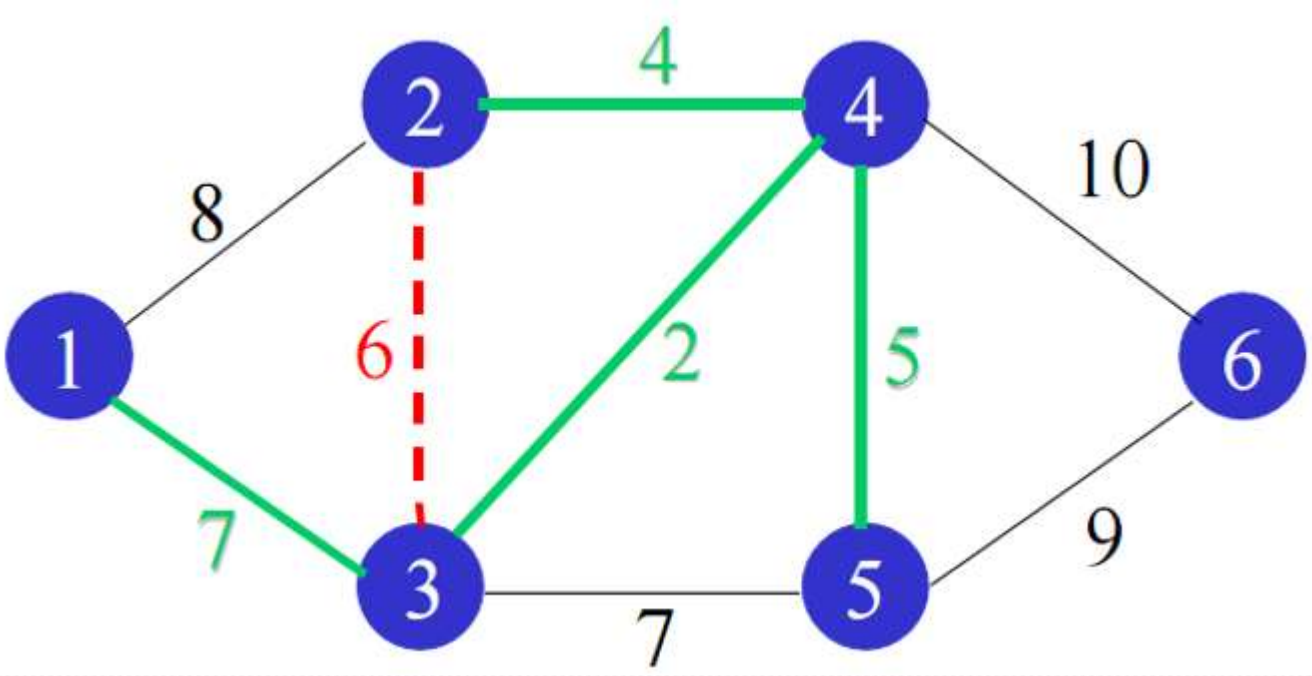
- On évalue (4,5):
- {~~(3,4)~~, ~~(2,4)~~, ~~(4,5)~~, (2,3), (1,3), (3,5), (1,2), (5,6),(4,6) }
  T= { (3,4), (2,4), (4,5) }



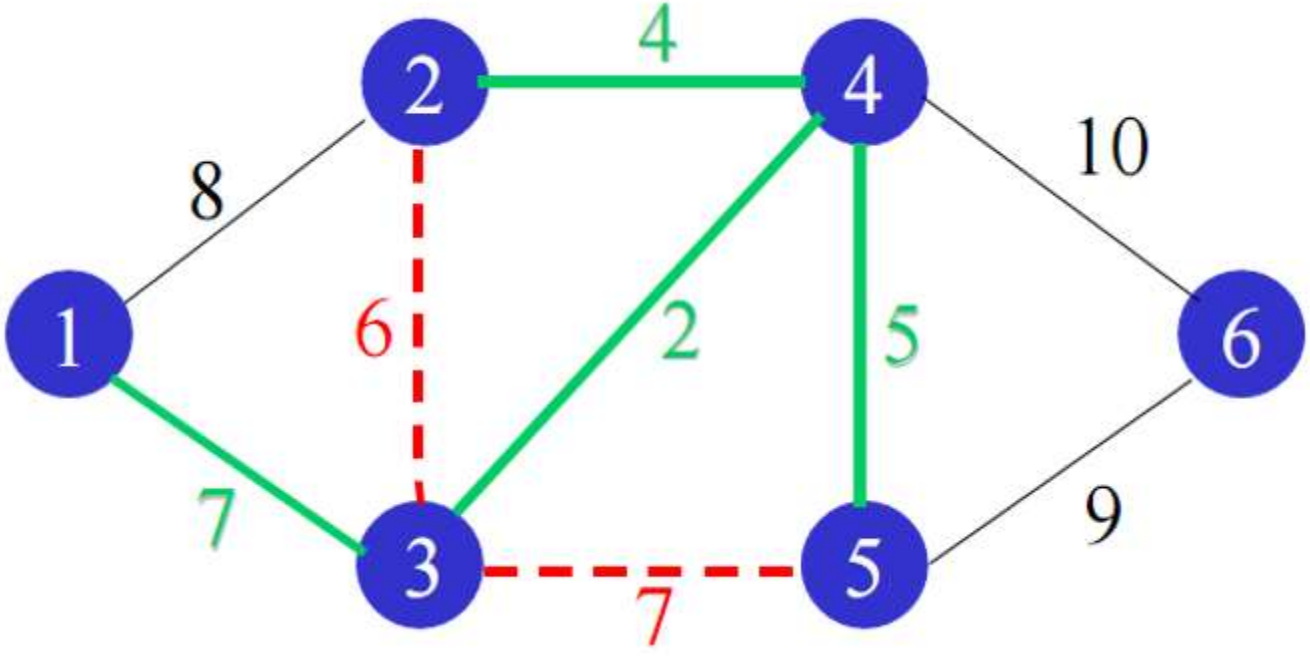
- On évalue (2,3):
- {~~(3,4)~~, ~~(2,4)~~, ~~(4,5)~~, ~~(2,3)~~, (1,3), (3,5), (1,2), (5,6),(4,6) }
  T= { (3,4), (2,4), (4,5) }



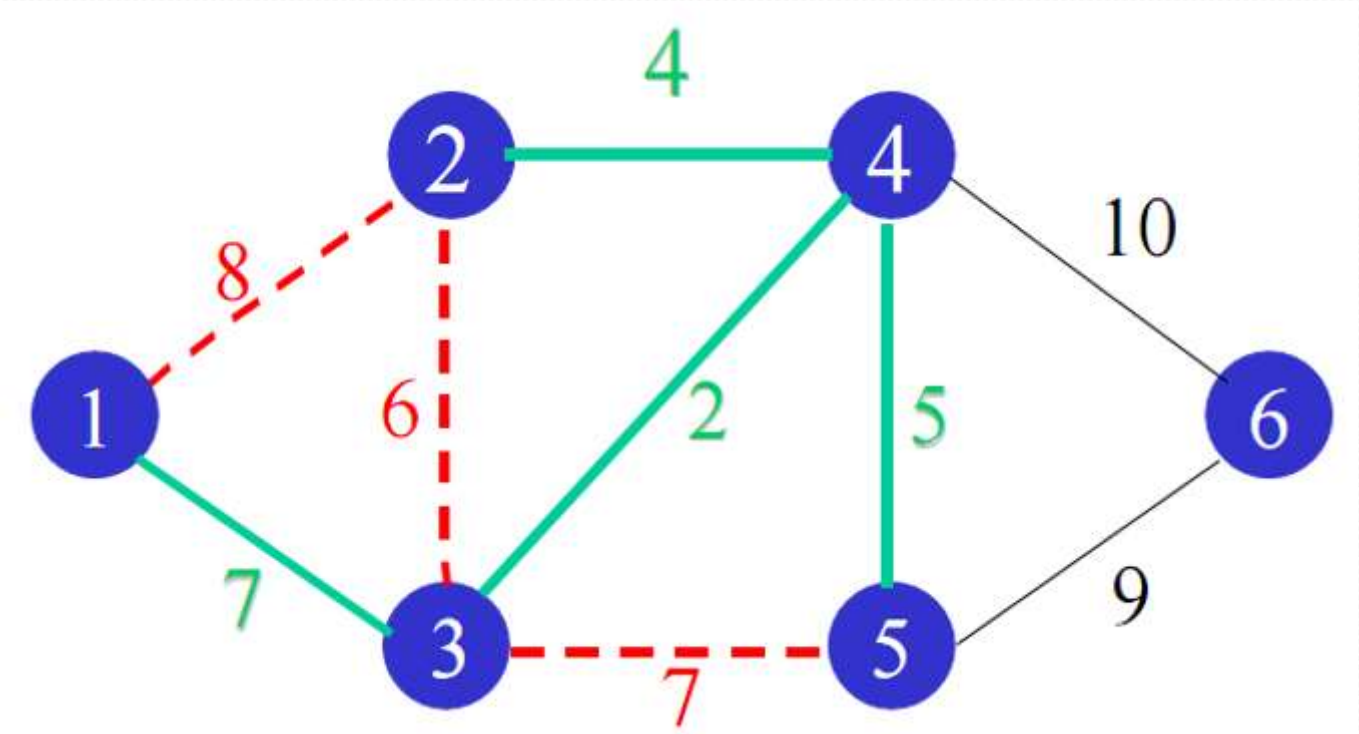
- On évalue (1,3):
- {~~(3,4)~~, ~~(2,4)~~, ~~(4,5)~~, ~~(2,3)~~, ~~(1,3)~~, (3,5), (1,2), (5,6),(4,6) }
  T= { (3,4), (2,4), (4,5), (1,3) }



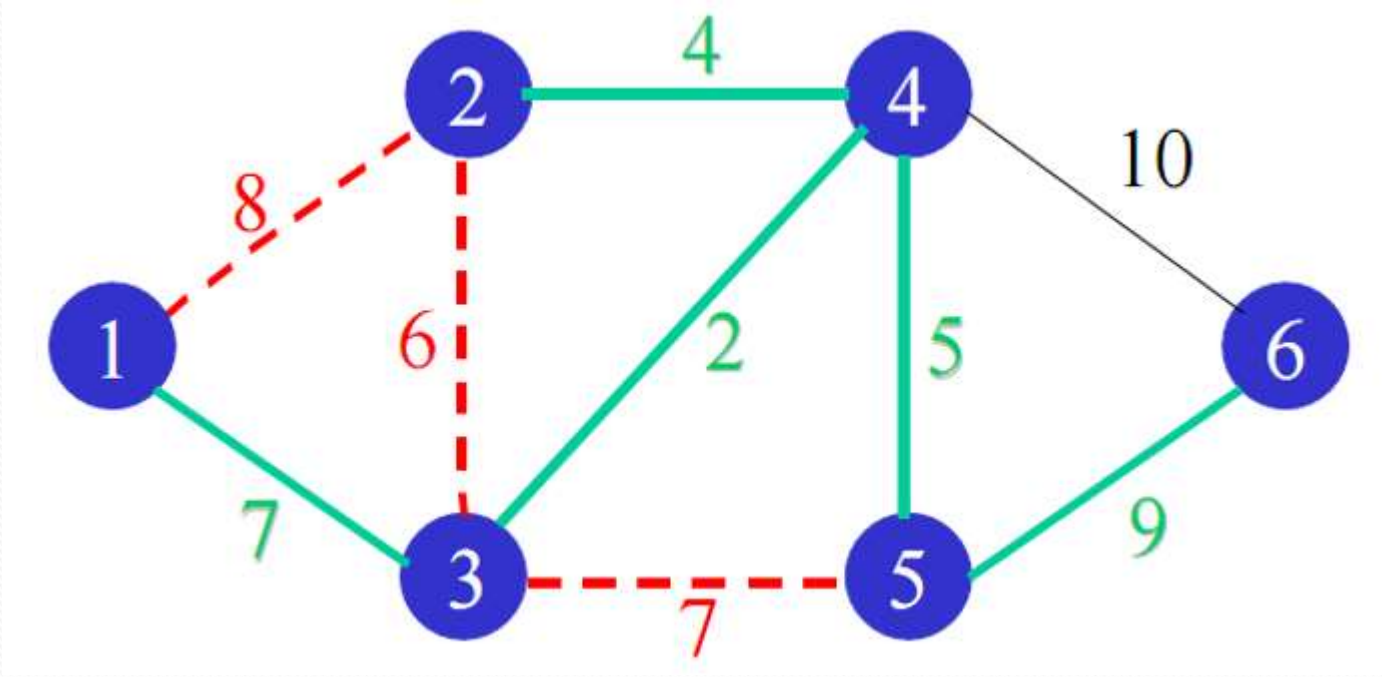
- On évalue (3,5):
- {~~(3,4)~~, ~~(2,4)~~, ~~(4,5)~~, ~~(2,3)~~, ~~(1,3)~~, ~~(3,5)~~, (1,2), (5,6),(4,6) }
  T= { (3,4), (2,4), (4,5), (1,3) }



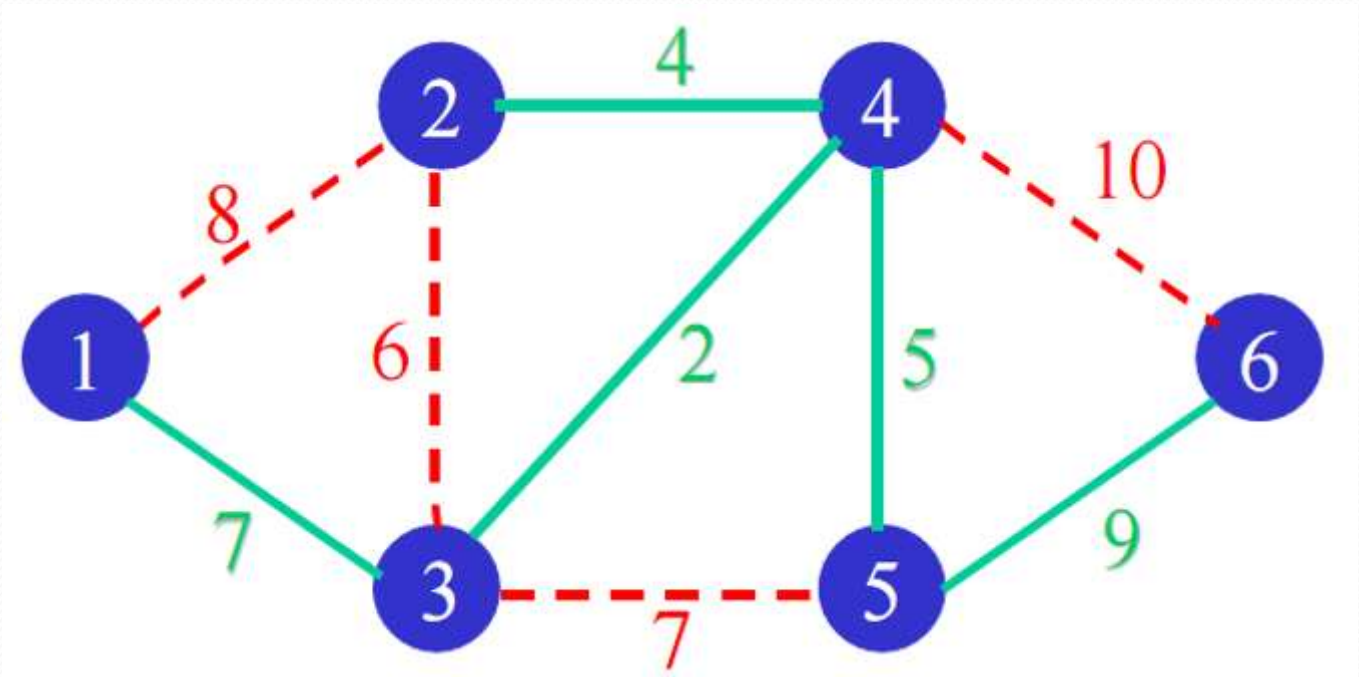
- On évalue (1,2):
- {~~(3,4)~~, ~~(2,4)~~, ~~(4,5)~~, ~~(2,3)~~, ~~(1,3)~~, ~~(3,5)~~, ~~(1,2)~~, (5,6) ,(4,6) }
  T= { (3,4), (2,4), (4,5), (1,3) }

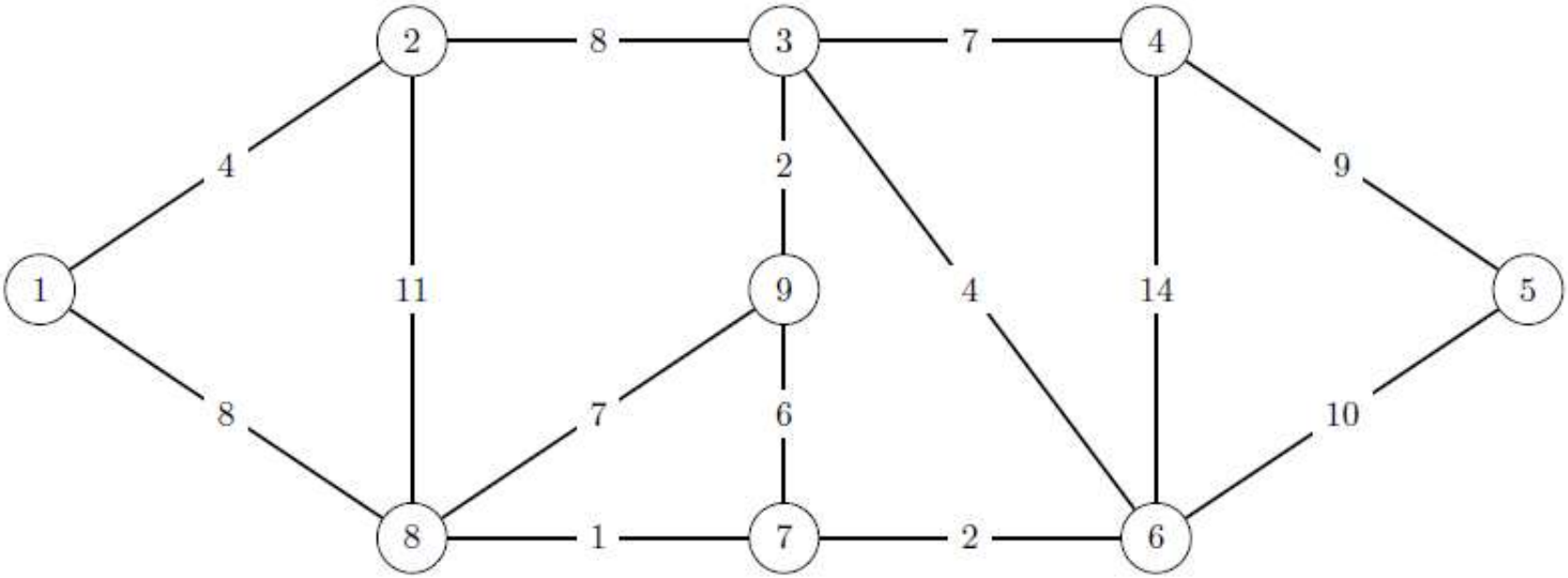


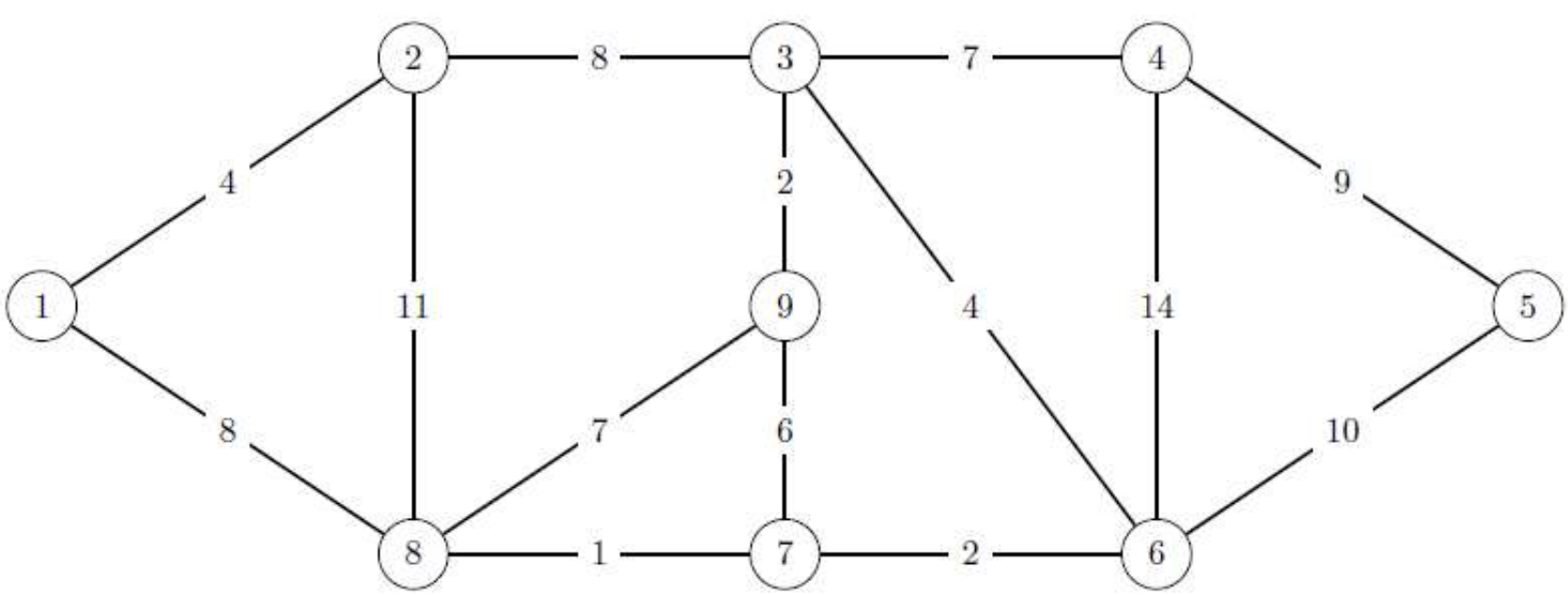
- On évalue (5,6):
- {~~(3,4)~~, ~~(2,4)~~, ~~(4,5)~~, ~~(2,3)~~, ~~(1,3)~~, ~~(3,5)~~, ~~(1,2)~~, ~~(5,6)~~, (4,6) }
  T= { (3,4), (2,4), (4,5), (1,3), (5,6) }



- On évalue (4,6):
- {~~(3,4)~~, ~~(2,4)~~, ~~(4,5)~~, ~~(2,3)~~, ~~(1,3)~~, ~~(3,5)~~, ~~(1,2)~~, ~~(5,6)~~, ~~(4,6)~~}
  T= { (3,4), (2,4), (4,5), (1,3), (5,6) }




2
8
3
7
4
4
2
9
1
11
9
4
14
5
8
7
6
10
8
1
7
2
6



On trie les arêtes du graphe. On obtient l'ordre suivant : {7; 8} < {3; 9} = {6; 7}< {1; 2} = {3; 6} < {7; 9} < {8; 9} = {3; 4} < {2; 3} = {1; 8} < {4; 5} < {5; 6} < {2; 8} < {4; 6}.

Algorithme de Prim

0. Initialiser T={}, S={} et W= V (sommets)

1. Choisir un nœud i $\in$ W, poser S = { i } et W = W \ { i }

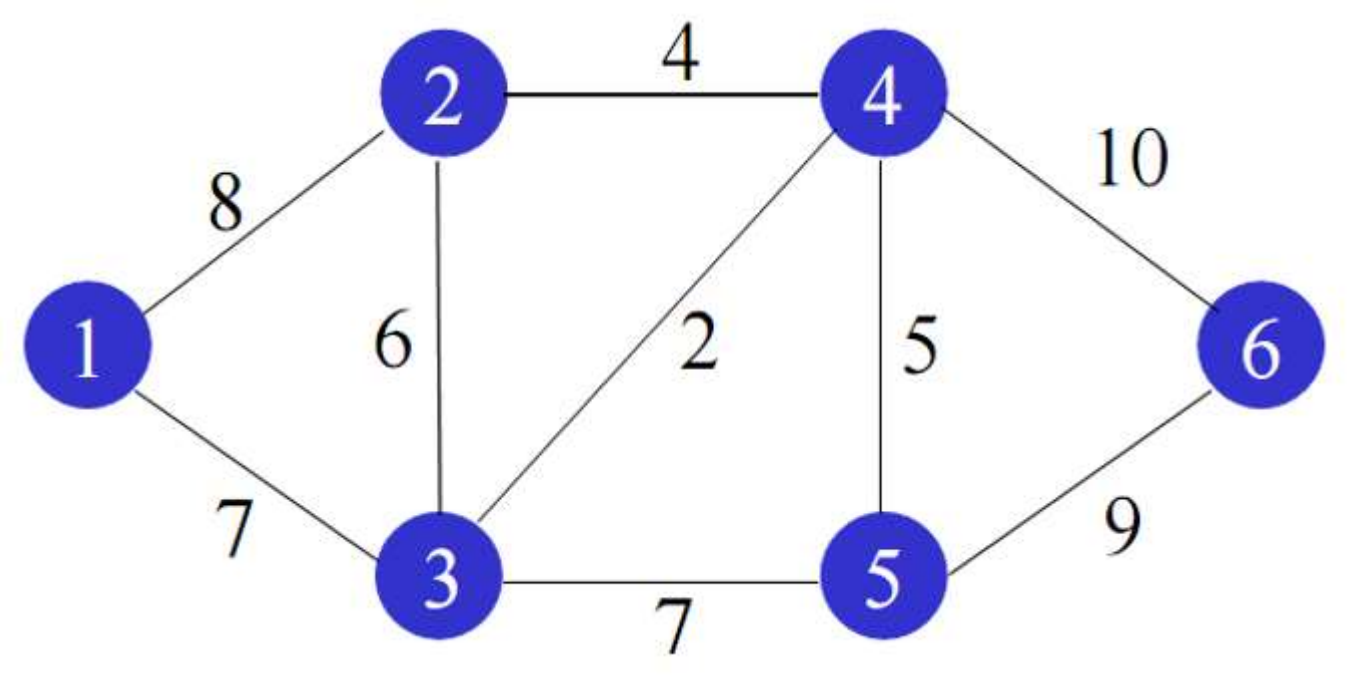
2. Choisir un nœud j $\in$ W tel que l'arc (i, j), où i $\in$ S, ait le $c_{i,j}$ le plus petit;

3. Si (i,j) ne forme pas de cycle alors inclure cet arc dans l'arbre, i.e. T = T $\cup$ (i,j), j $\in$ S et W = W \ { j } et aller à l'étape 4; sinon exclure l'arc et retourner à l'étape 2;

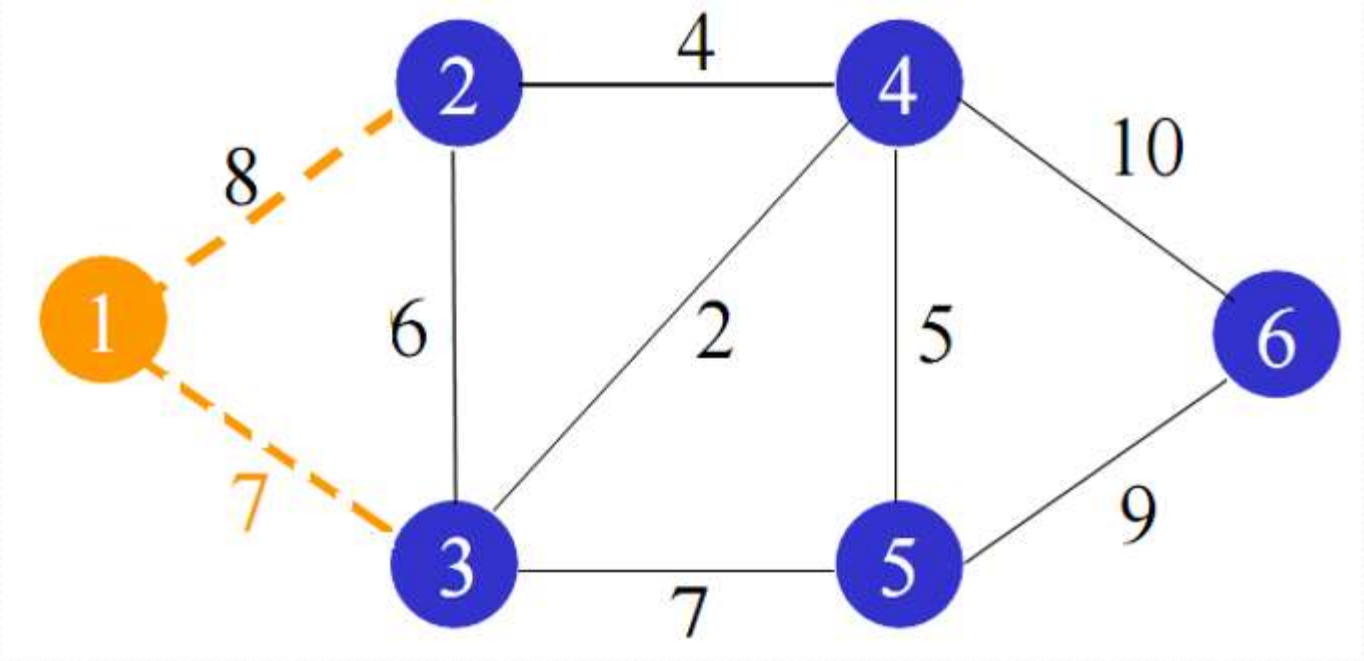
4. Si S = V alors l'arbre de poids minimal est trouvé; sinon retourner à l 'étape 2.

Algorithme de Prim

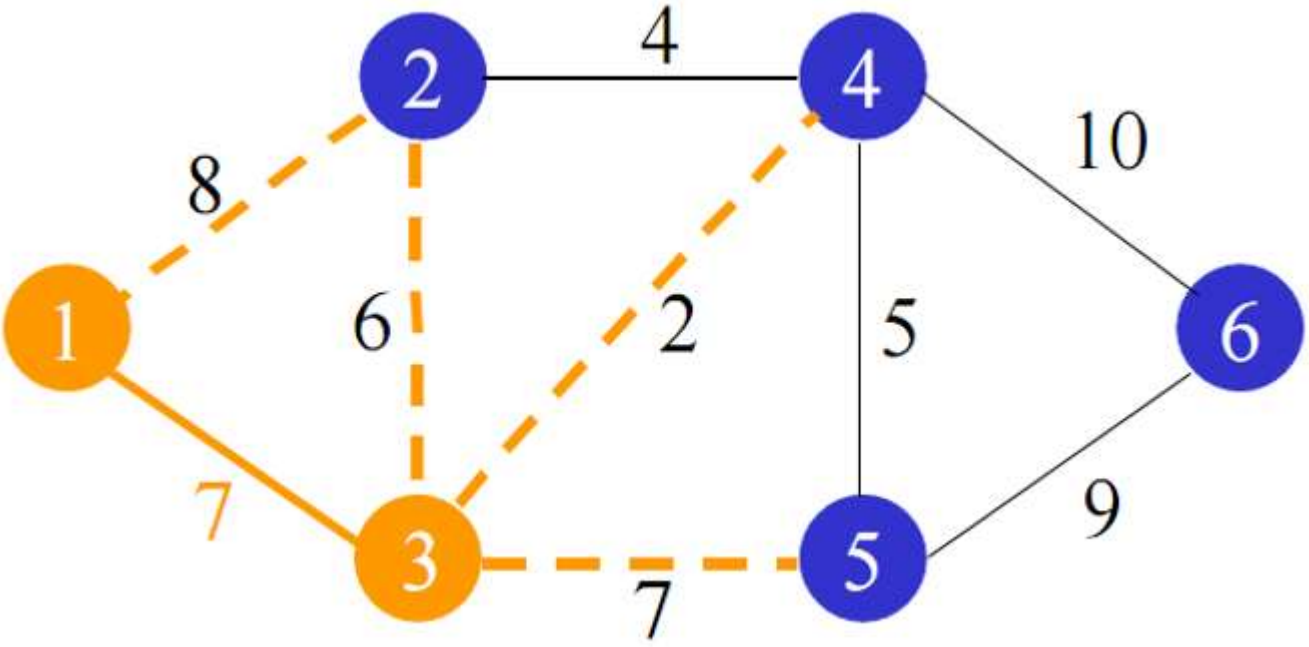
- S = { }
- W = { 1, 2, 3, 4, 5, 6 }
- T = { }



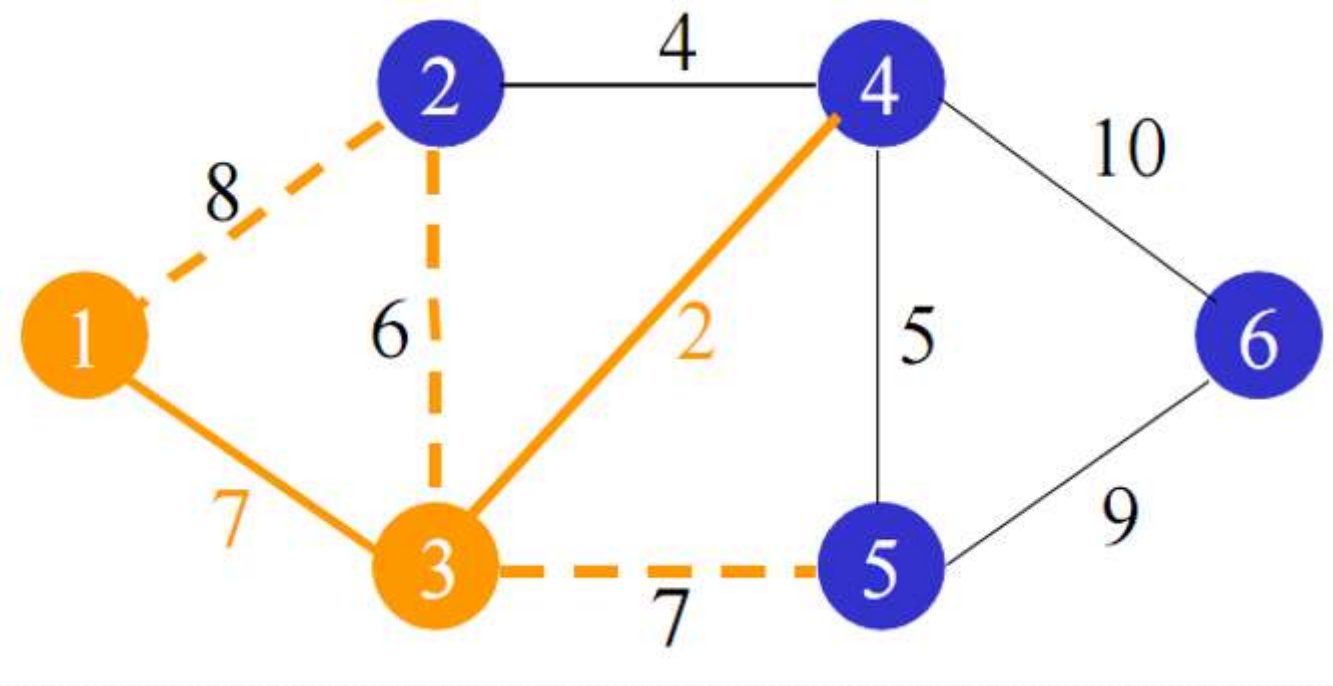
- S = { 1 }
- W = { 2, 3, 4, 5, 6 }
- T = { }



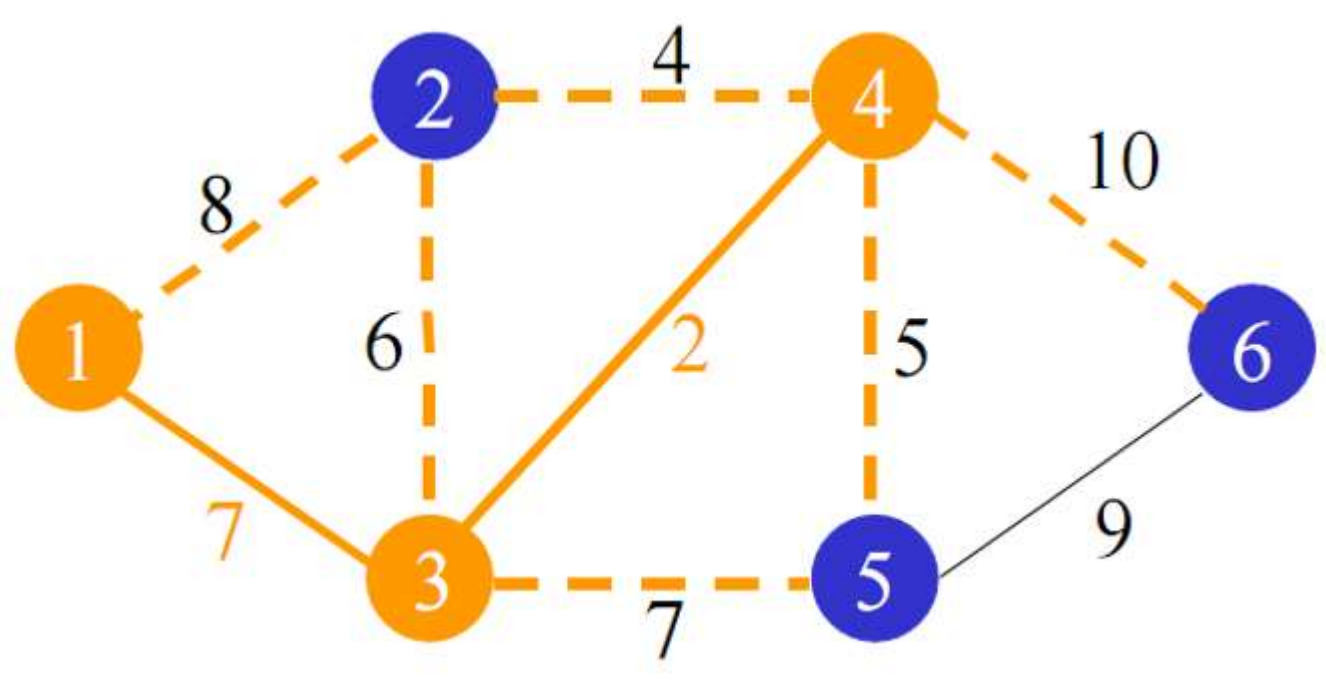
- S = { 1,3 }
- W = { 2, 4, 5, 6 }
- T = { (1,3) }



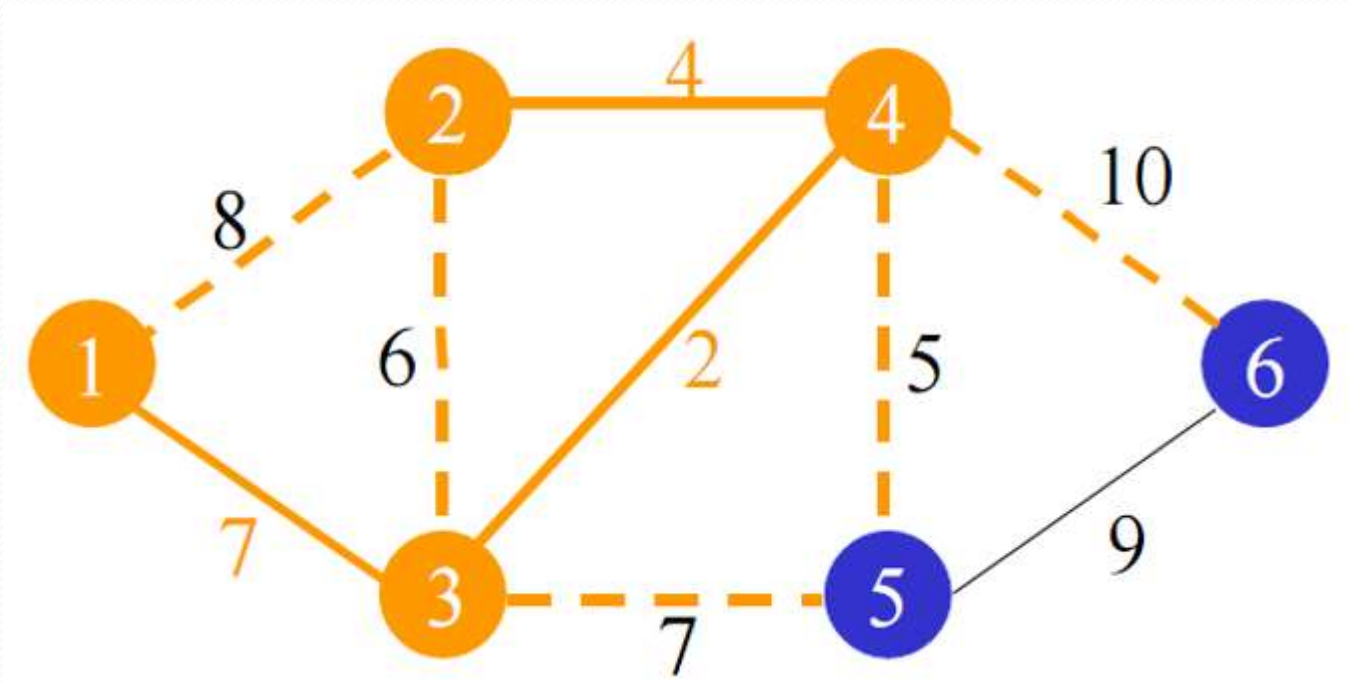
- S = { 1,3 }
- W = { 2, 4, 5, 6 }
- T = { (1,3), (3,4) }



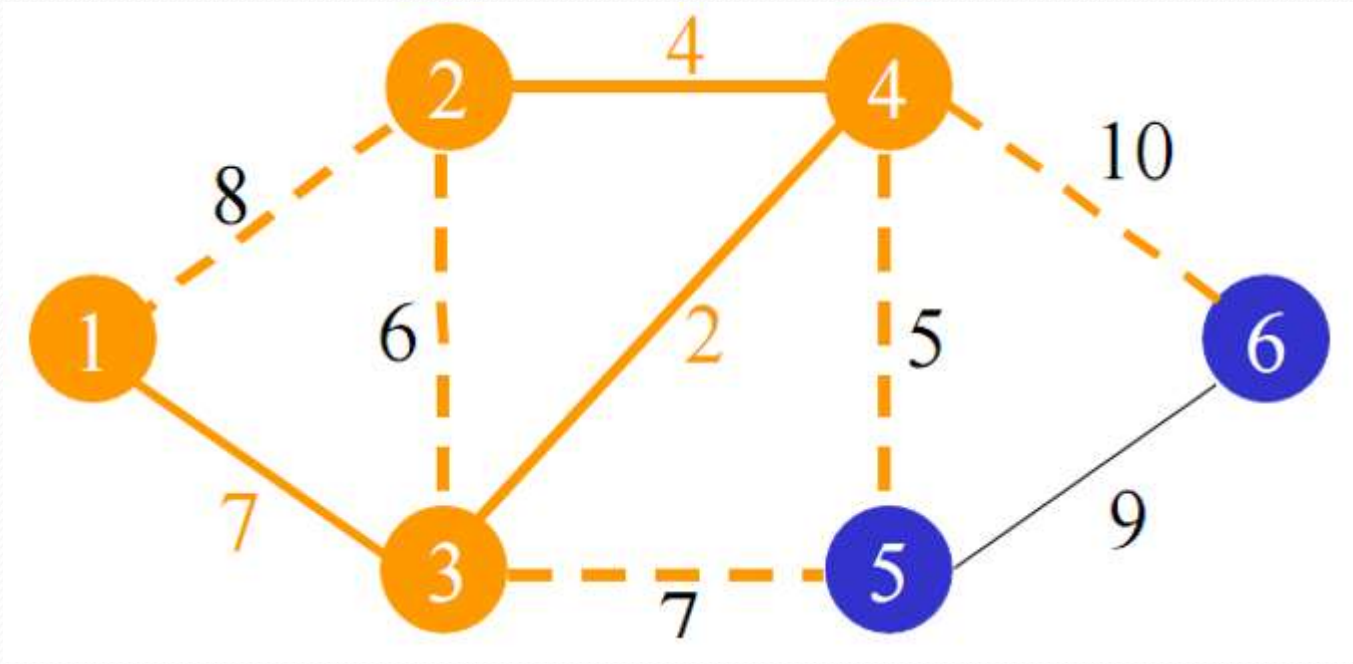
- S = { 1,3,4 }
- W = { 2, 5, 6 }
- T = { (1,3), (3,4) }



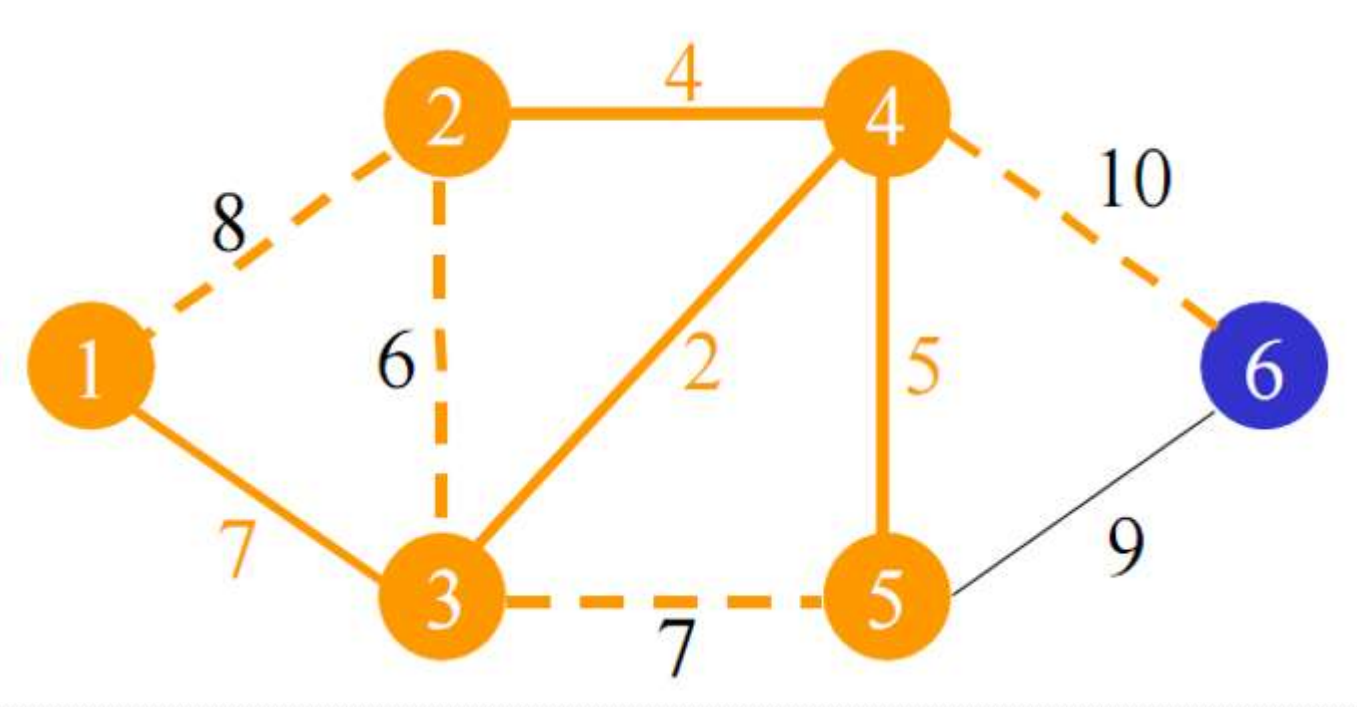
- S = { 1,3,4 }
- W = { 5, 6 }
- T = { (1,3), (3,4), (2,4) }



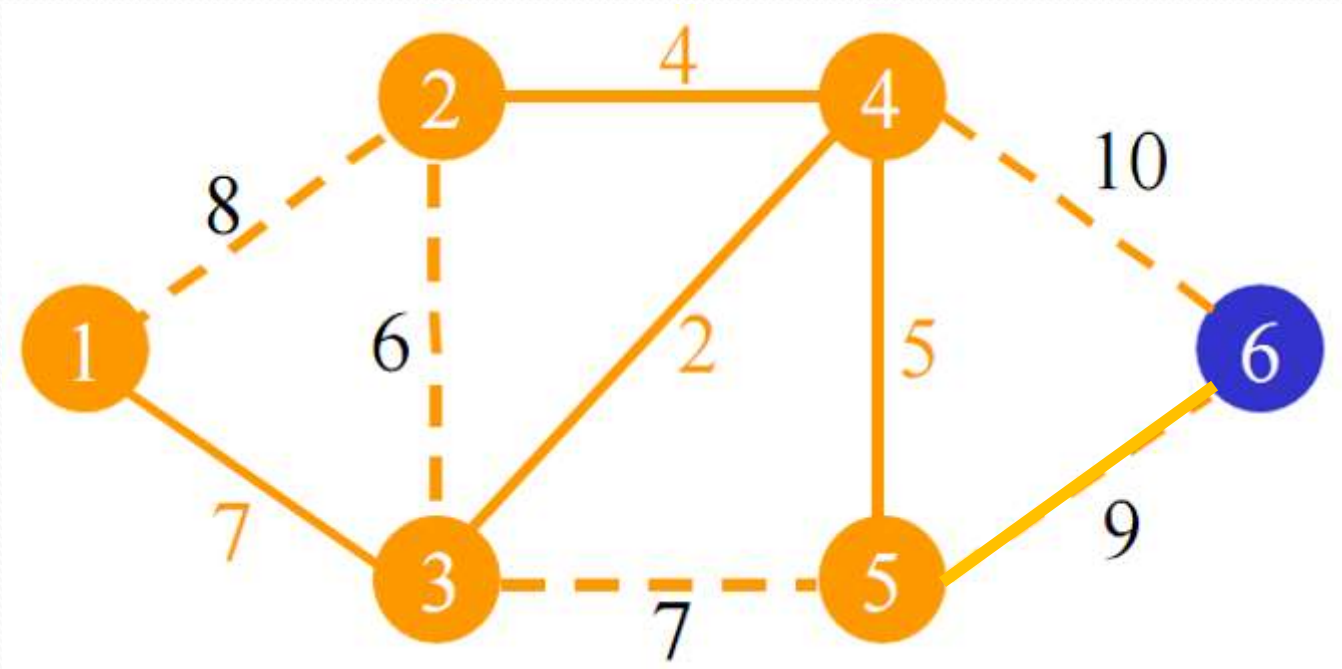
- S = { 1,3,4,2 }
- W = { 5, 6 }
- T = { (1,3), (3,4), (2,4) }

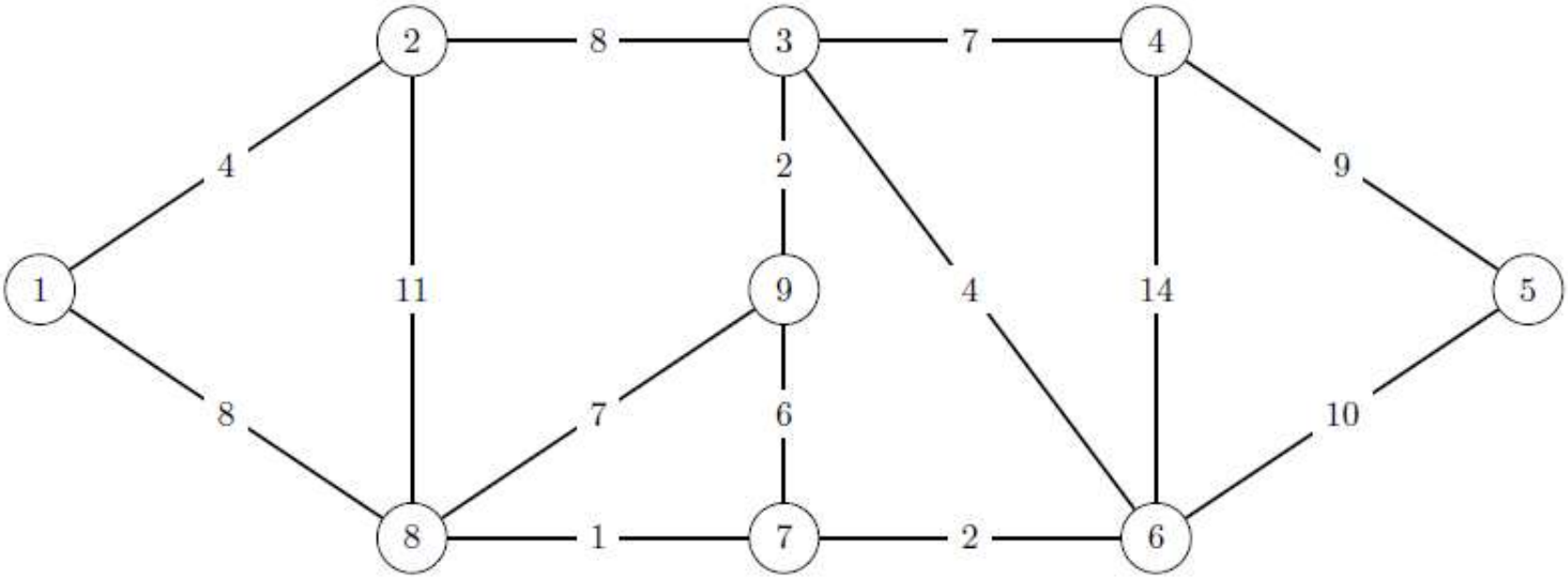


- S = { 1,3,4,2 }
- W = { 6 }
- T = { (1,3), (3,4), (2,4), (4,5), (5,6)}



- S = { 1, 3, 4, 2 ,5}
- W = { }
- T = { (1,3), (3,4), (2,4), (4,5), (5,6)}




2
8
3
7
4
4
2
9
1
11
9
4
14
5
8
7
6
10
8
1
7
2
6

- Introduction aux théorie des graphes
- Notions de base
- Problème de coloration d'un graphe
- Arbre Couvrant de poids Minimal : ACM
  - Algorithme de Kruskal
  - Algorithme de Prim
- **Problème du plus court chemin**
  - algorithme de Dijkstra (source unique)
  - algorithme de Bellman-Ford (source unique)
  - algorithme de Floyd-Warshall (tous les PCC)
- **Le Problème du Flot Maximal**
  - Les réseaux de transport
  - Le flot maximum et la coupe minimum
  - L'algorithme de Ford et Fulkerson
- **le Réseau PERT**

Problème du plus court chemin

Problème du plus court chemin

Le **problème du plus court chemin** est un problème classique en algorithmique et en théorie des graphes, où l'objectif est de trouver le chemin le plus court entre deux nœuds dans un graphe pondéré.

Exemple d'application : Réseau de transport urbain

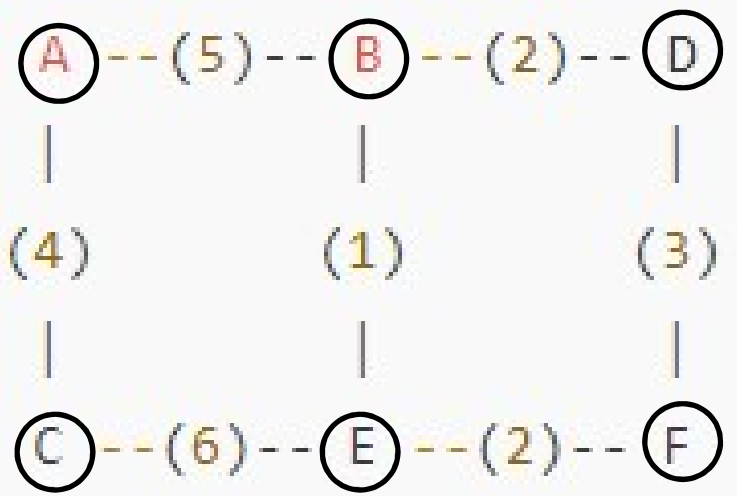
- Imaginons une ville avec un réseau de transports en commun (bus, métro, tramway) qui relie différents points d'intérêt de la ville. Chaque station est un nœud, et chaque ligne de transport (par exemple, un trajet en métro ou un bus) est une arête reliant deux nœuds, avec un poids représentant le temps ou la distance entre les stations.

Objectif

- L'objectif est de trouver le plus court chemin entre deux stations, c'est-à-dire le trajet qui minimise le temps ou la distance entre ces deux stations.

Modélisation:

- Les stations sont représentées par des nœuds
- et les connexions entre elles par des arêtes avec des poids qui représentent le temps de trajet entre les stations.
- Voici un petit graphe représentant le réseau :



Problème du plus court chemin

- Ce problème du **Plus Court Chemin** (PCC) peut être posé de la façon suivante:

- Etant donné un graphe orienté valué **G (S,A,*v*)**, on associe à chaque arc **u(i,j)** un nombre réel **l(u)** ou $\mathbf{l_{ij}}$, appelé la **longueur** ou le **poids** de l'arc,

- le PCC entre deux sommets **s** (source) et **d** (destination) de **G** consiste à déterminer, parmi tous les chemins allant de **s** à **d**, un chemin, noté **u***, dont la longueur totale **l(u*)** soit **minimale**

Plus courts chemins

Plus courts chemins dans un graphe

- G = (S, A, w) un graphe pondéré.

  La longueur du chemin (u_1, … , u_k) est

$$\sum_{i=1}^{k-1} w(u_i, u_{i+1})$$

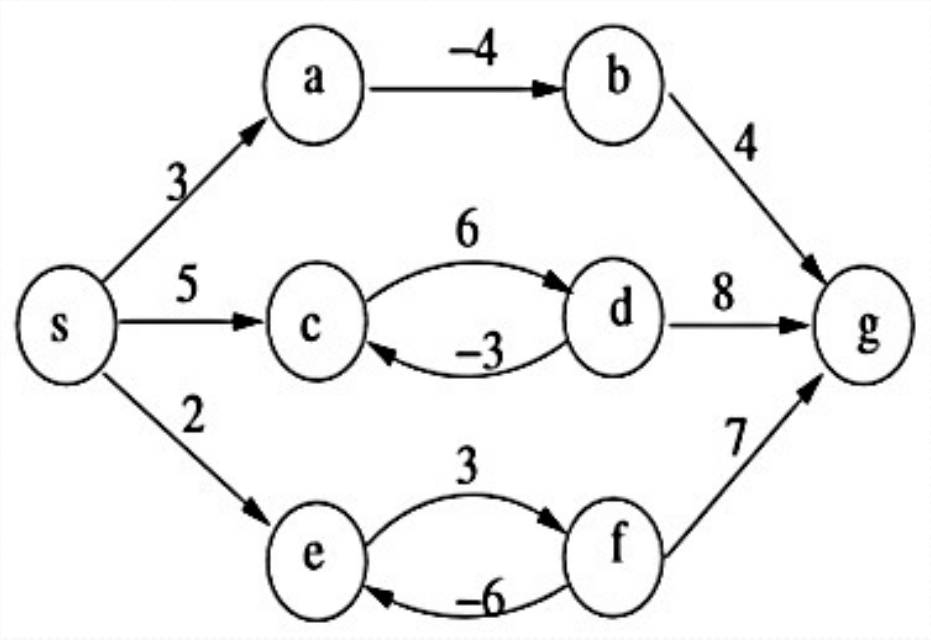
- Notation :

$$\delta(u, v)$$

la longueur d'un plus court chemin de u à v.

Condition Nécessaire:

- Le problème du plus court chemin a une solution si et seulement s'il n'existe pas dans le graphe de circuit de longueur strictement négative pouvant être atteint à partir de l'origine (s).
- Un circuit négatif est appelé **circuit absorbant**



Algorithme de résolution

Algorithmes	Type du PCC	Propriétés du graphe	
		Type de graphe	Longueur
Dijkstra	**D'un sommet à tous les autres sommets**	**Graphe orienté (et non orienté)**	**Longueur positives**
Bellman		**Graphe orienté sans circuit (sommet d'origine doit être sans prédécesseur)**	**Longueur quelconque (nombre réel)**
Bellman-Ford		**Graphe orienté**	
Floyd	**Entre tous les couples de sommets**	**Graphe orienté sans circuit absorbant**	

8

- **Circuits de longueur négative** :
  - pas de plus courts chemins,
- **Arcs de poids positifs :**
  - **algorithme de Dijkstra** (source unique)
- **Arcs de poids quelconques** :
  - **algorithme de Bellman-Ford** (source unique)
  - algorithme de Floyd-Warshall (tous les PCC)

Algorithme de Dijkstra

Algorithme de Dijkstra

- L'idée est de déterminer pas à pas la plus courte distance depuis le sommet source **s** pour atteindre chacun des sommets. Il faut donc garantir à chaque itération que la distance pour arriver à un sommet **d** est bien minimum et que cela ne peut pas être remis en cause par la suite.

- Cet algorithme permet de calculer le **PCC** d'un sommet « s » à un sommet « d » ou d'un sommet « s » à tous les autres sommets dans un graphe de longueur positive.

Algorithme de Dijkstra

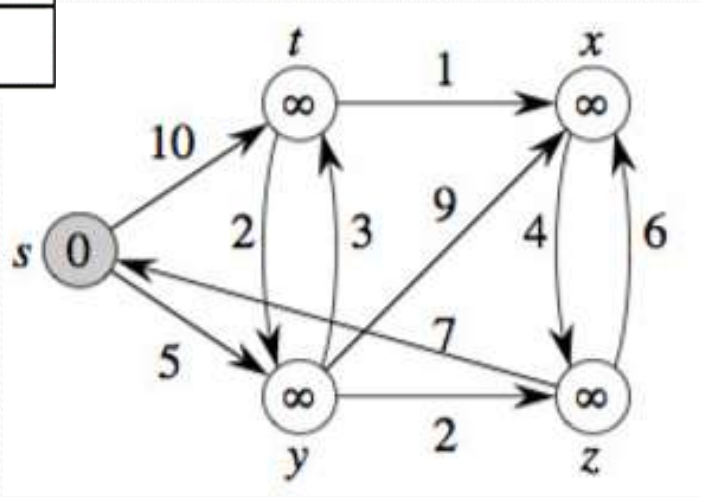
Algorithme de DIJKSTRA :

Données : Un graphe $G(V, E)$, une fonction w **positive** et un sommet s.

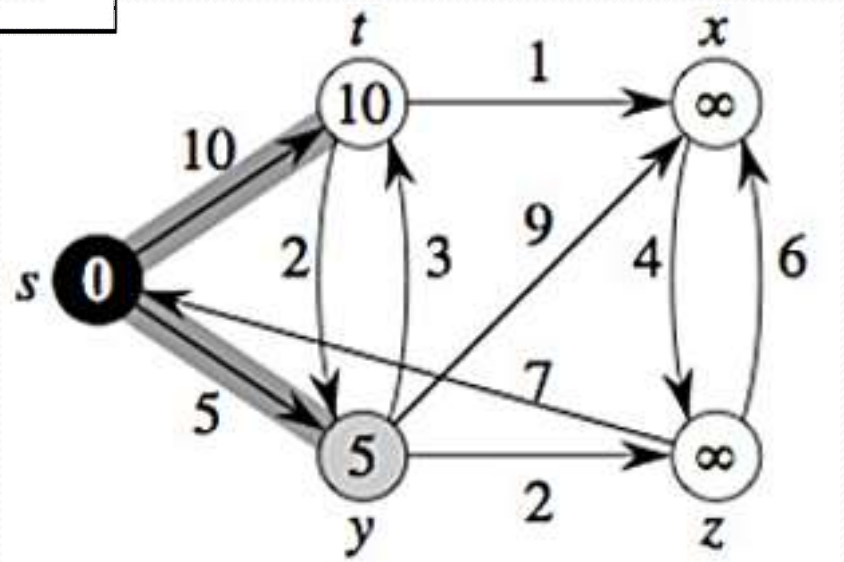
Instructions :

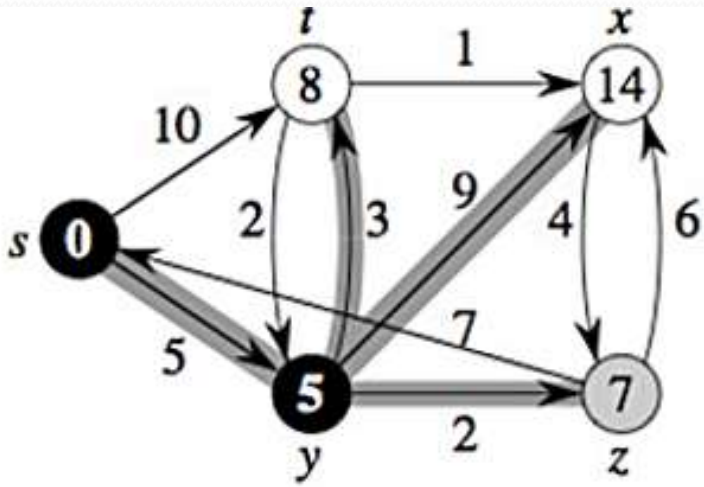
1 - $\forall$ v $\in V$, $\phi[v] = +\infty$ et $\pi[v] = -1$; // Initialisation
2 - $\phi[s] = 0$;
3 - Soit Q un ensemble de sommets initialisé à V ;
4 - **TANT QUE** Q est non vide :
5 - Soit u le sommet de Q avec $\phi[u]$ minimum
6 - Retirer u de Q
7 - **POUR TOUT** $v \in \Gamma^+(u)$: //successeur de u
8 - **SI** $(\phi[v] > \phi[u] + w(u, v))$ **ALORS** // Mise à jour
9 - $\phi[v] = \phi[u] + w(u, v)$;
10- $\pi[v] = u$;

	s	t	y	x	z
	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$

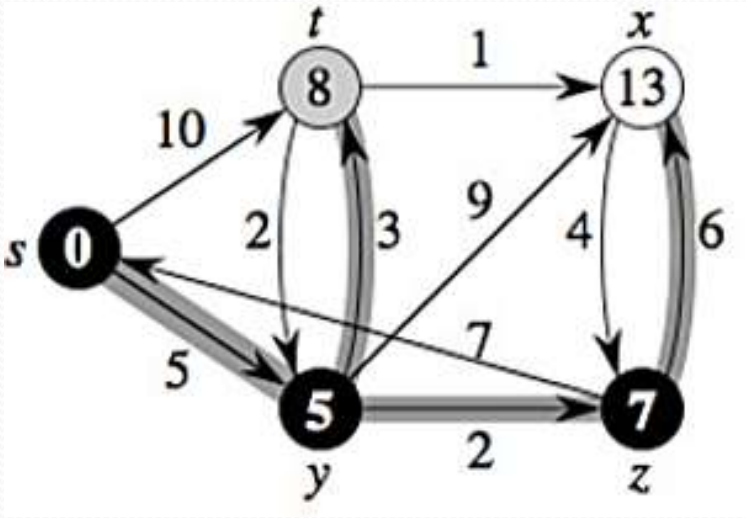


	s	t	y	x	z
	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$
s	0	10	5	$+\infty$	$+\infty$

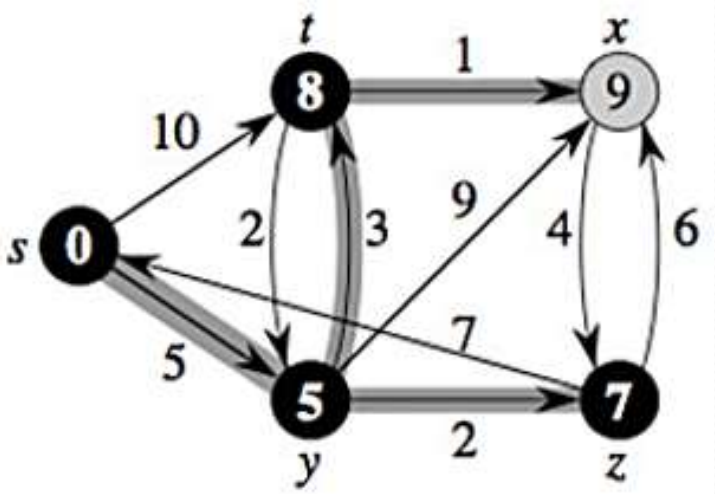




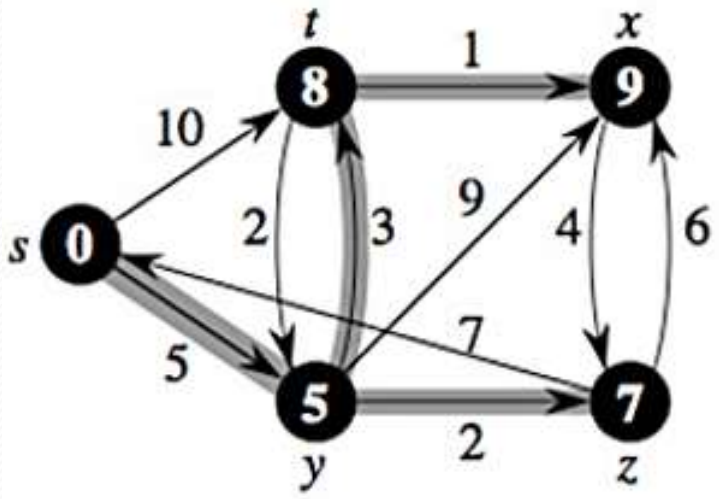
	s	t	y	x	z
	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$
s	0	10	5	$+\infty$	$+\infty$
y	0	8	5	14	7



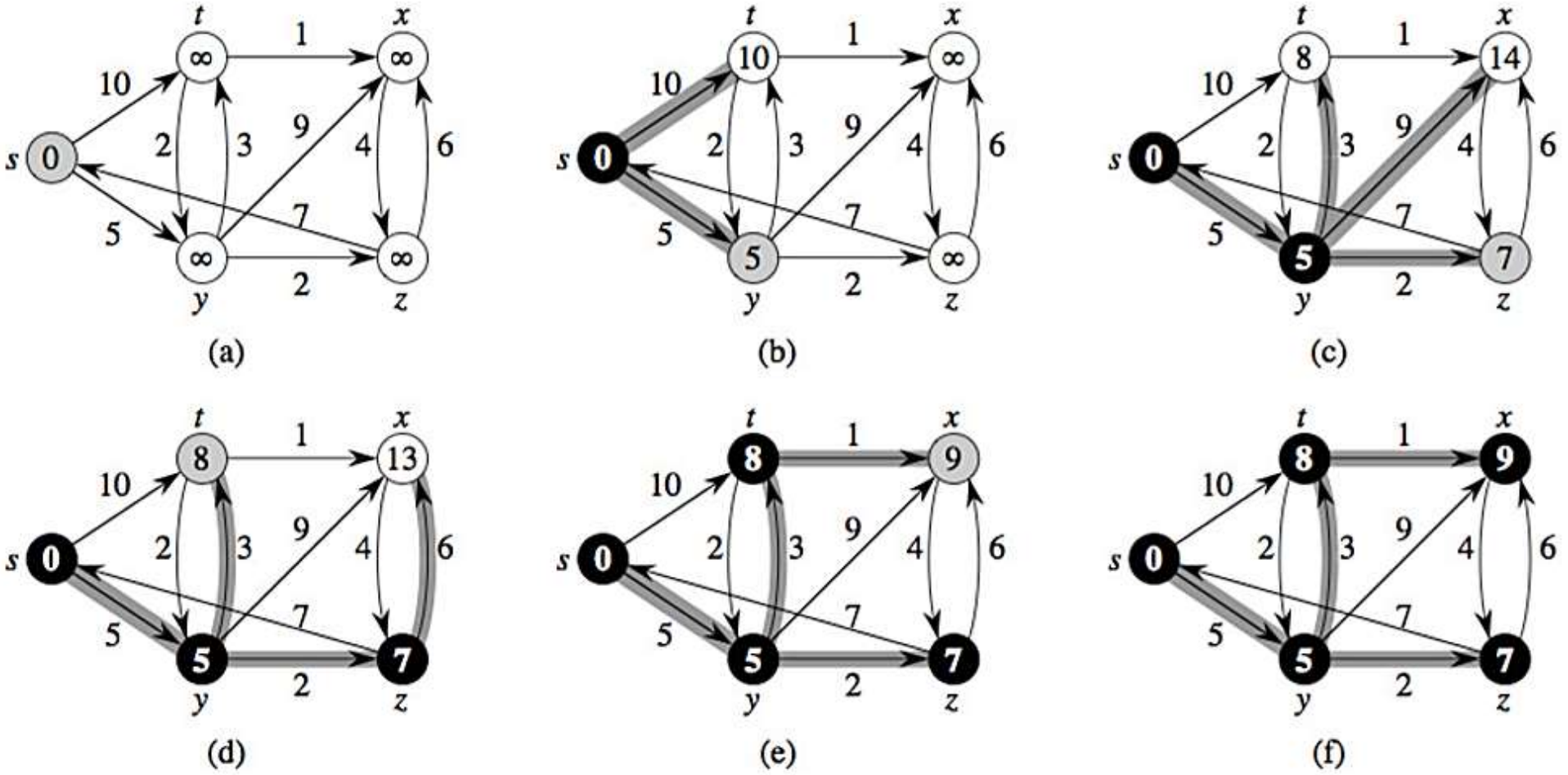
	s	t	y	x	z
	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$
s	0	10	5	$+\infty$	$+\infty$
y	0	8	5	14	7
z	0	8	5	13	7



	s	t	y	x	z
	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$
s	0	10	5	$+\infty$	$+\infty$
y	0	8	5	14	7
z	0	8	5	13	7
t	0	8	5	9	7



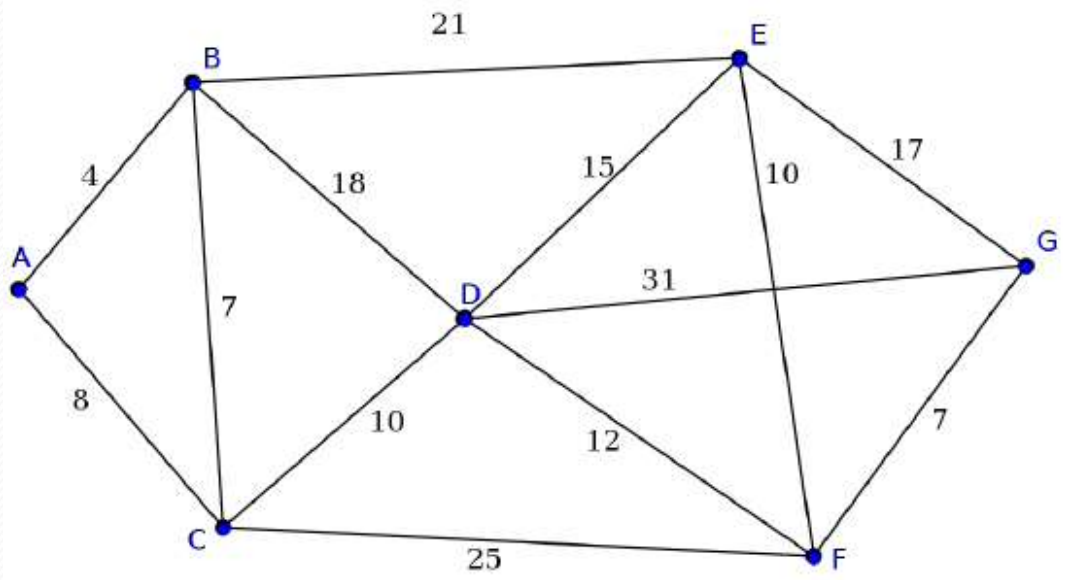
	s	t	y	x	z
	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$
s	0	10	5	$+\infty$	$+\infty$
y	0	8	5	14	7
z	0	8	5	13	7
t	0	8	5	9	7
x	0	8	5	9	7

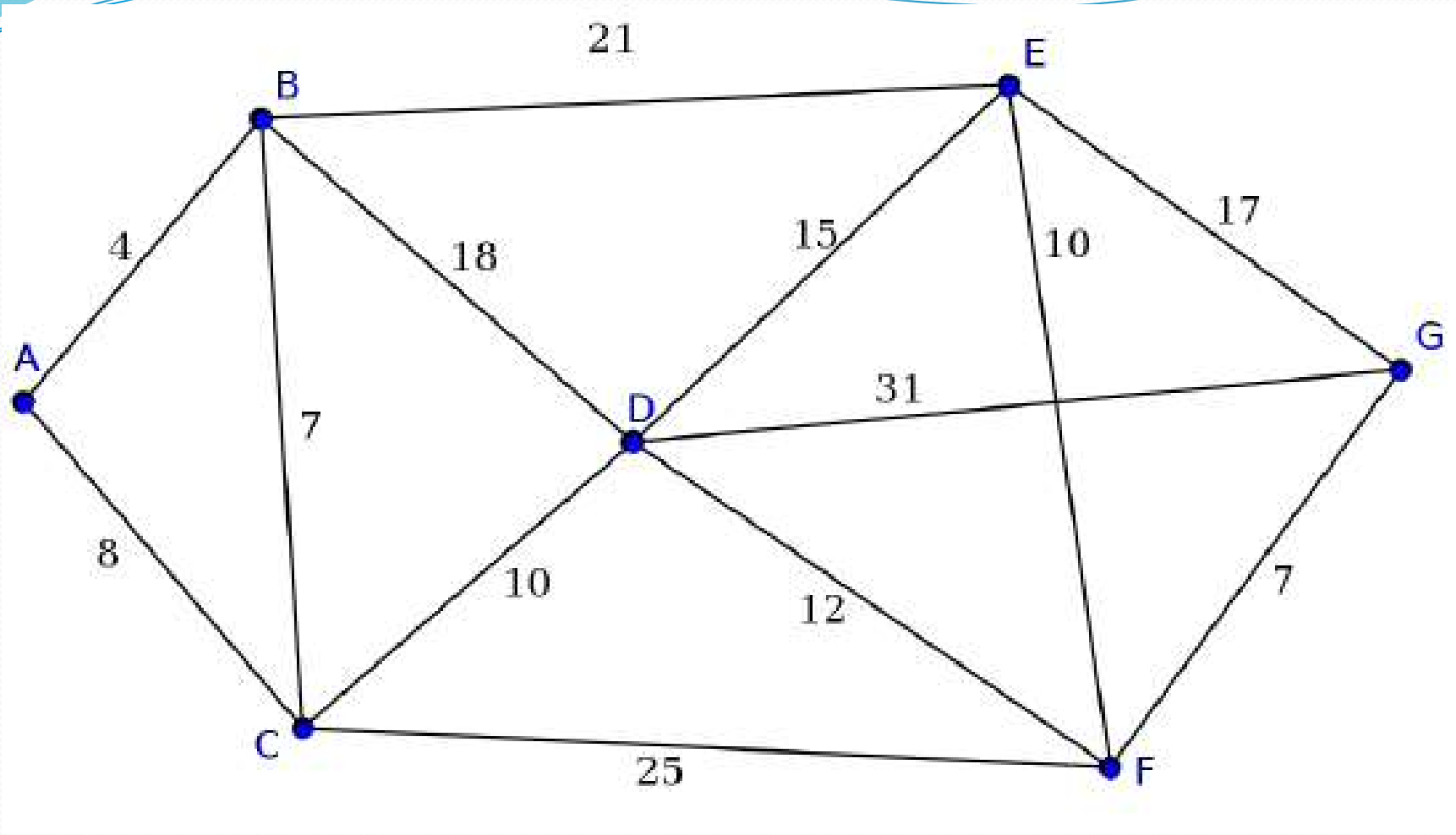


Exécution de l'algorithme de Dijkstra sur le graphe.

Exercice 1: Une région est munie d'un réseau de trains, représenté par le graphe G ci-contre. Les stations sont symbolisées par les sommets A, B, C, D, E, F et G. Chaque arête représente une ligne reliant deux gares. Les temps de parcours (correspondance comprise) en minutes entre chaque sommet ont été rajoutés sur le graphe.

- Déterminer le plus court chemin en minutes, reliant la gare **B** à la gare **G**.



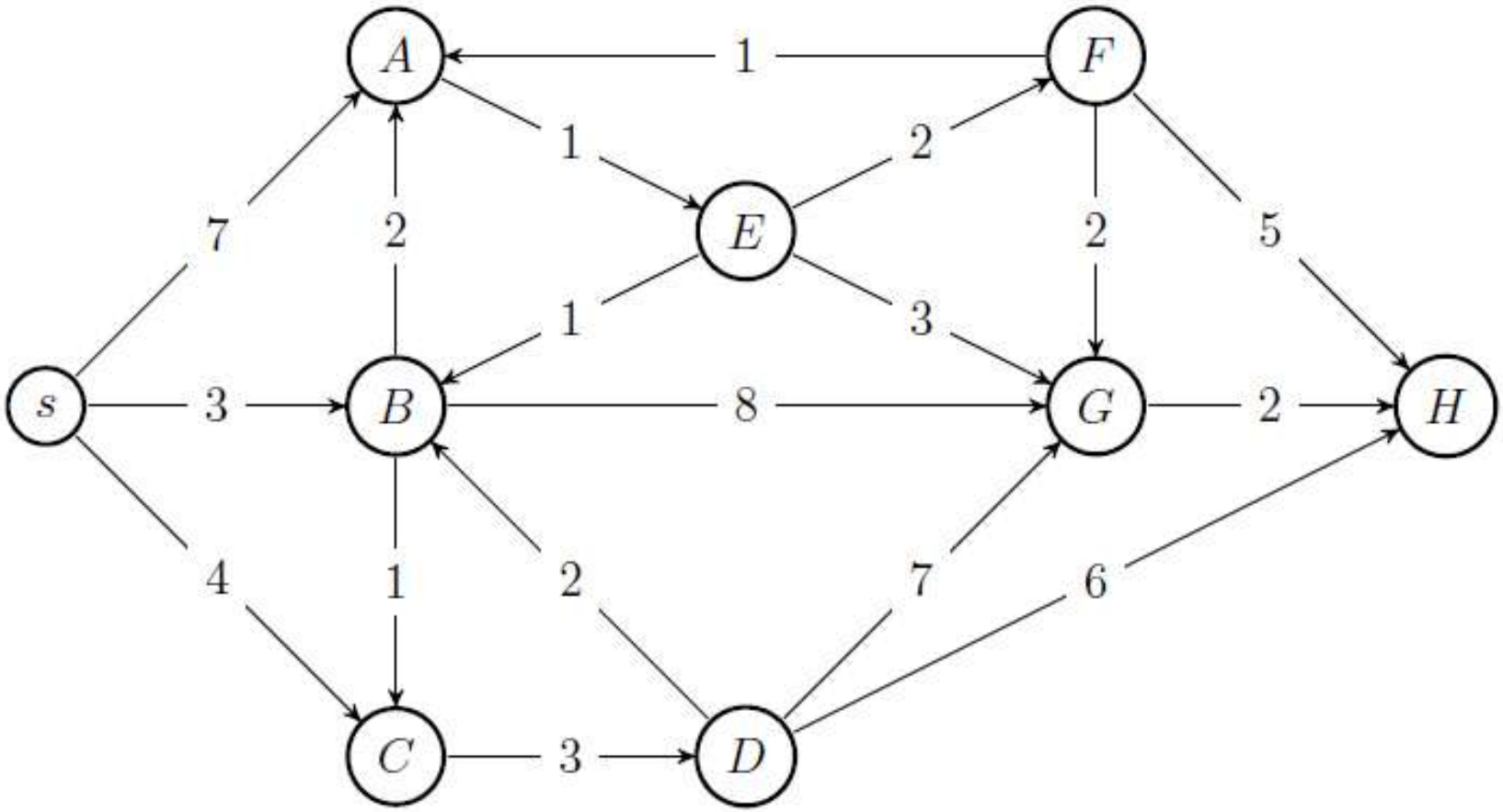


Le plus court chemin en minutes, reliant la gare **B** à la gare **G** est B – C – D – F – G, et la durée est de 36 minutes

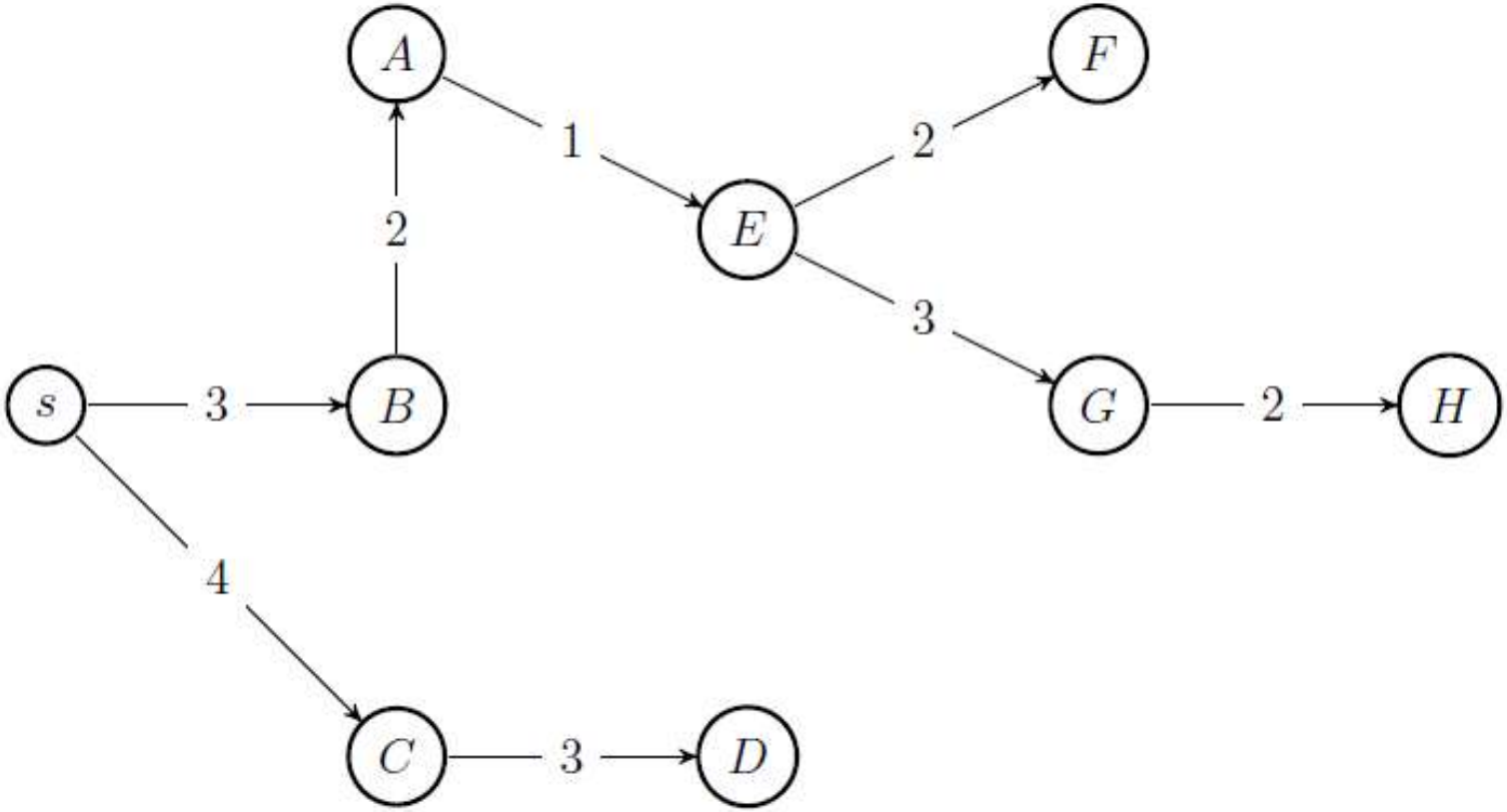
Exercice 2:

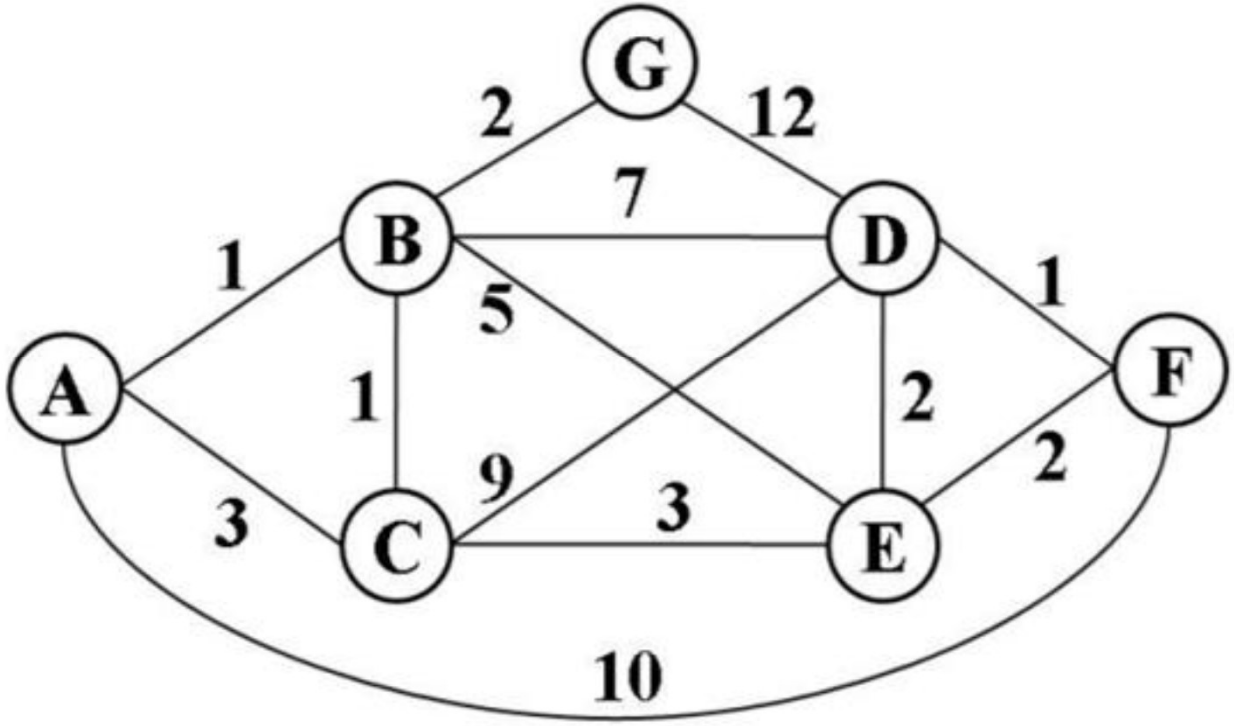
1. En utilisant l'algorithme de Dijkstra, trouver les plus courts chemins de **s** aux autres sommets du graphe suivant G

2. Dessiner l'arborescence des plus courts chemins d'origine **s**.


A
F
E
s
B
G
H
C
D
1
1
2
7
2
2
5
1
3
3
8
2
4
1
2
7
6
3

pivot	s	A	B	C	D	E	F	G	H
	0	∞	∞	∞	∞	∞	∞	∞	∞
s	X	7	3	4					
B		5	X					11	
C				X	7				
A		X				6			
E						X	8	9	
D					X				13
F							X		
G								X	11
H									X


A
F
1
2
E
2
3
s
3
B
G
2
H
4
C
3
D


G
2
12
7
B
D
1
1
5
A
1
F
2
2
9
3
3
C
E
10

Algorithme de Bellman-Ford

Algorithme de Bellman-Ford

- **Entrée** : un graphe G = (S, A), un sommet source **s**,
- **Sortie** : les longueurs des plus courts chemins issus de **s**, (constitués de au plus $n - 1$ arcs)

- **Sous-problèmes** $\phi(i, v)$: longueur minimale d'un chemin de **s** à ***v*** constitué de au plus ***i*** arcs,

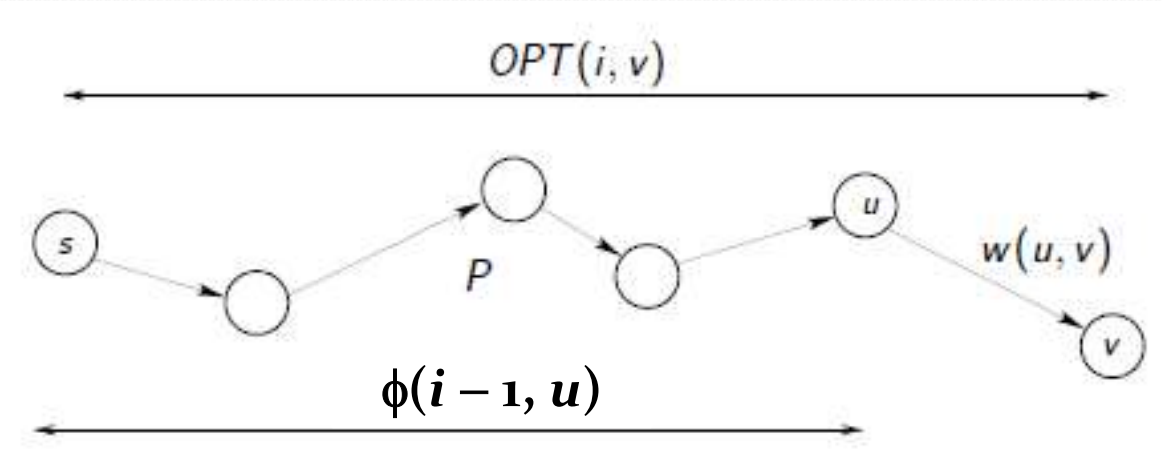
- **Objectif final** : $\phi(n - 1, v)$

(plus courts chemins issus de **s** constitués de au plus $n - 1$ arcs : $\phi(n - 1, v) = \delta(v)$)

- **Algorithme** :
  - initialiser ϕ (0, v) pour tout v,
  - calculer ϕ (1, v) pour tout v,
  - . . .
  - calculer $\phi(n-1, v)$ pour tout v.

- Soit P un chemin optimal pour le sous-problème $\phi(i, v)$
  - si P contient au plus $\boldsymbol{i-1}$ arcs alors $\phi(i, v) = \phi(i-1, v)$,
  - si P contient $\boldsymbol{i}$ arcs, alors il existe un arc $(u, v) \in$ A tel que

$$\boldsymbol{\phi(i\ , v) = \phi(i-1, u) + w(u, v)}$$



Définition récursive de $\phi(i, v)$:

- $\phi(0, s) = 0$
- $\phi(0, v) = +\infty$ si $v \neq s$,
- $\phi(i, v) = \mathbf{min}[\phi(i-1, v)\ ,\ \phi(i-1, u) + w(u, v)]$ si $i \neq 0$
  (tel que $u \in \Gamma^{-}(v)$ c.-à-d. $(u, v) \in A$)

Algorithme de BELLMAN-FORD :

Données : Un graphe $G(V, E)$, une fonction w et un sommet s.

Sortie : Vrai si le graphe ne contient pas de circuit absorbant, faux sinon

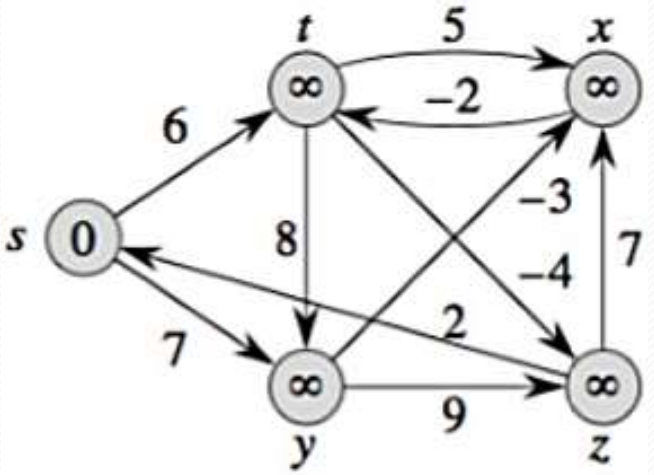
Instructions :

1 - Pour i de 1 à $|V| - 1$ et $\forall v \in V$, $\phi[i, v] = +\infty$ et $\pi[i, v] = -1$; // Initialisation
2 - $\phi[0, s] = 0$;
3 - **POUR** i de 1 à $|V| - 1$:
4 - **POUR TOUT** sommet v de V :
4 - **POUR TOUT** $u \in \Gamma^-(v)$: // pour tous les prédécesseurs de v
5 - **SI** $(\phi[i, v] > \phi[i - 1, u] + w(u, v))$ **ALORS** // Mise à jour
6 - $\phi[i, v] = \phi[i - 1, u] + w(u, v)$;
7 - $\pi[i, v] = u$;
8 - **POUR TOUT** arc uv de E :
9 - **SI** $(\phi[|V| - 1, v] > \phi[|V| - 1, u] + w(u, v))$ **ALORS**
10- **RETOURNER** Faux ;
11- **RETOURNER** Vrai

Initialisation

$\phi(o, s) = o$

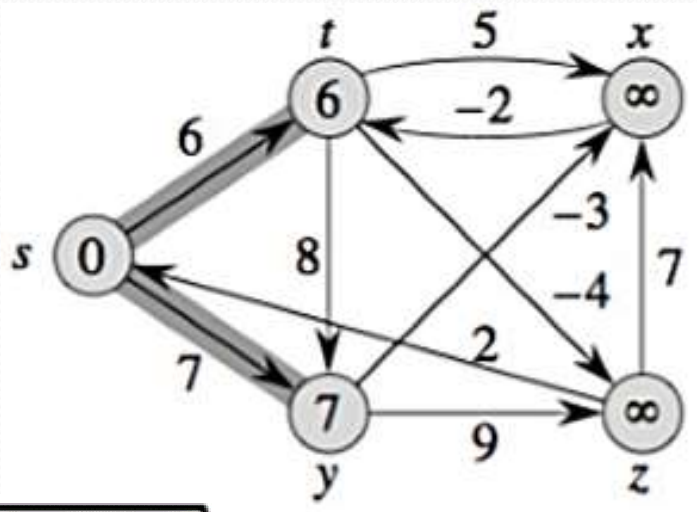
$\phi(o, v) = +\infty$ si $v \neq s$,



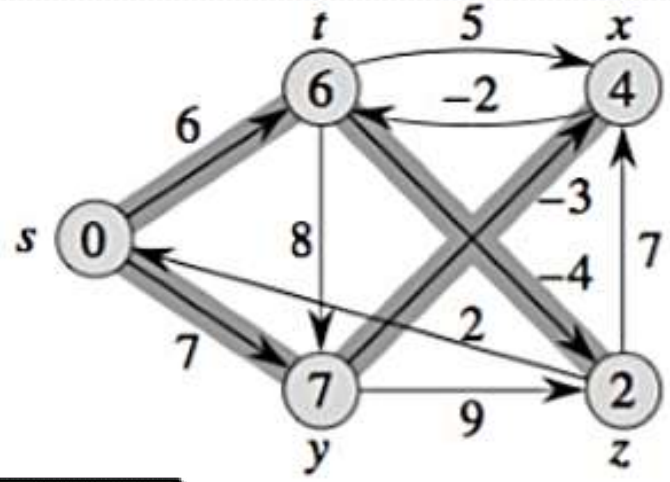
Iter	s	t	y	x	z
0	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$
1					
2					
3					
4					

$$\phi(1, t) = \min[\phi(o, t) , \phi(o, s) + w(s, t), \phi(o, x) + w(x, t)]$$

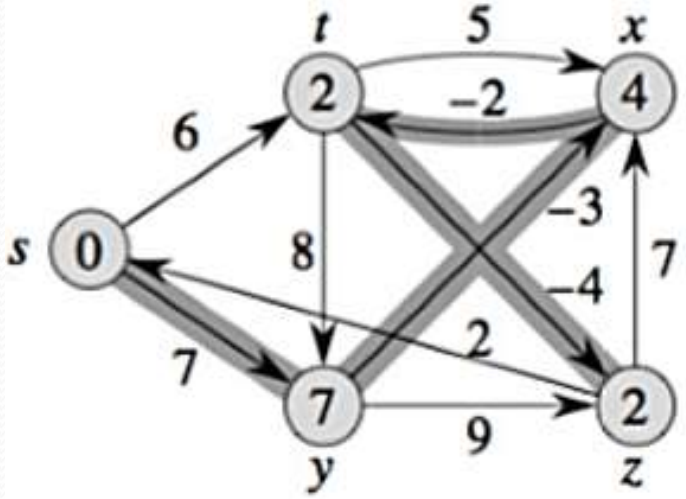
$$\phi(1, y) = \min[\phi(o, y) , \phi(o, s) + w(s, y), \phi(o, t) + w(t, y)]$$



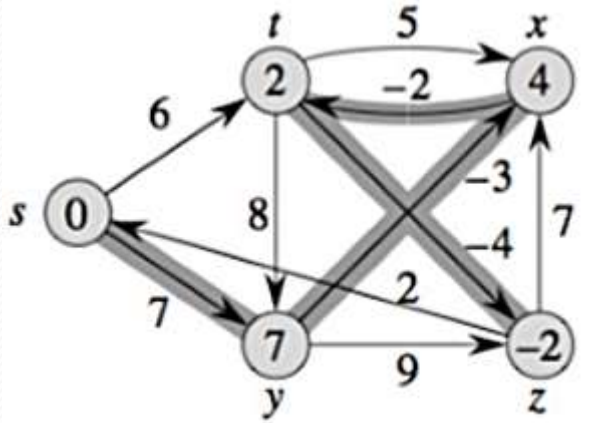
Iter	s	t	y	x	z
0	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$
1	0	6	7	$+\infty$	$+\infty$
2					
3					
4					



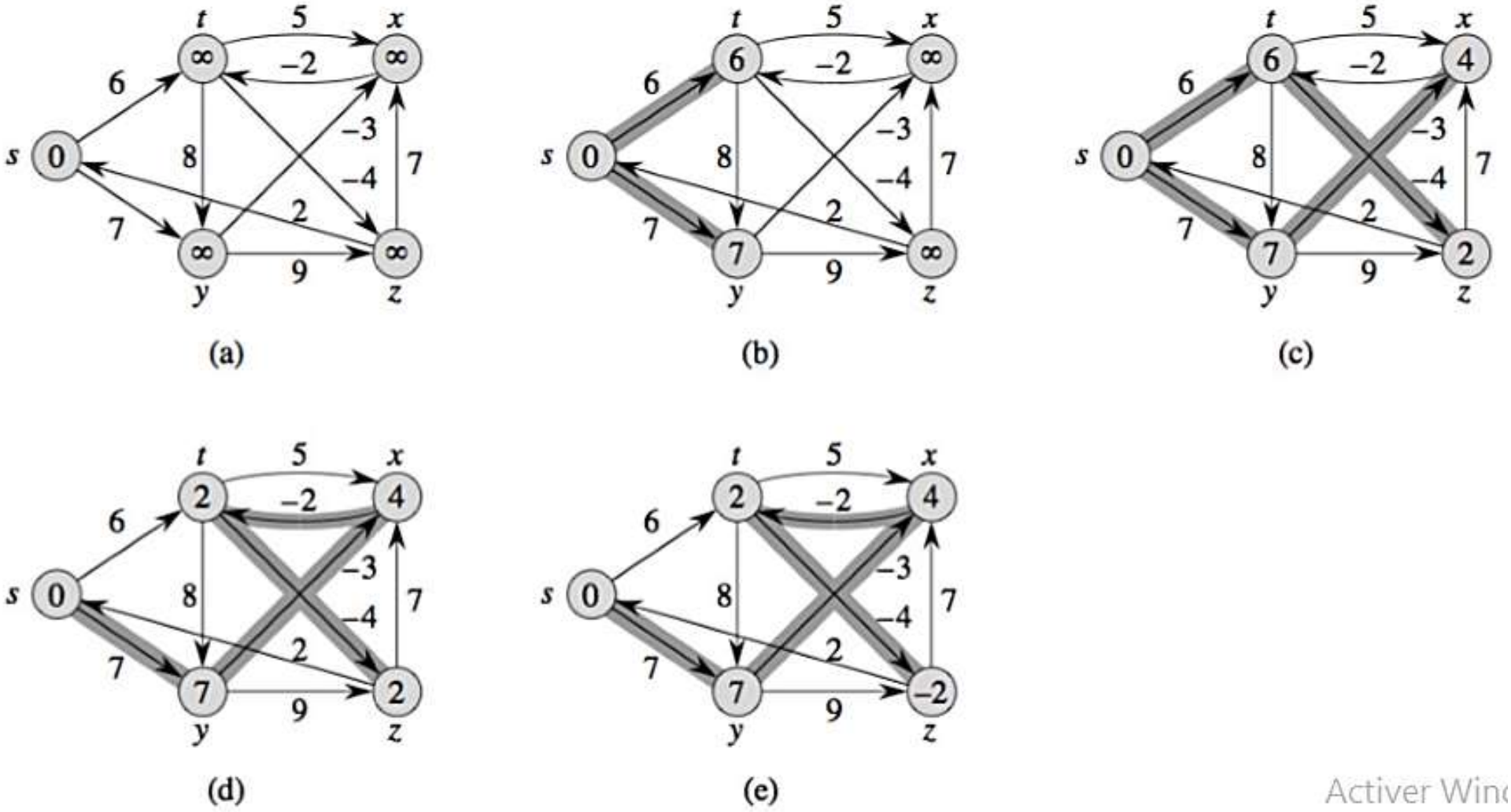
Iter	s	t	y	x	z
0	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$
1	0	6	7	$+\infty$	$+\infty$
2	0	6	7	4	2
3					
4					



Iter	s	t	y	x	z
0	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$
1	0	6	7	$+\infty$	$+\infty$
2	0	6	7	4	2
3	0	2	7	4	2
4					

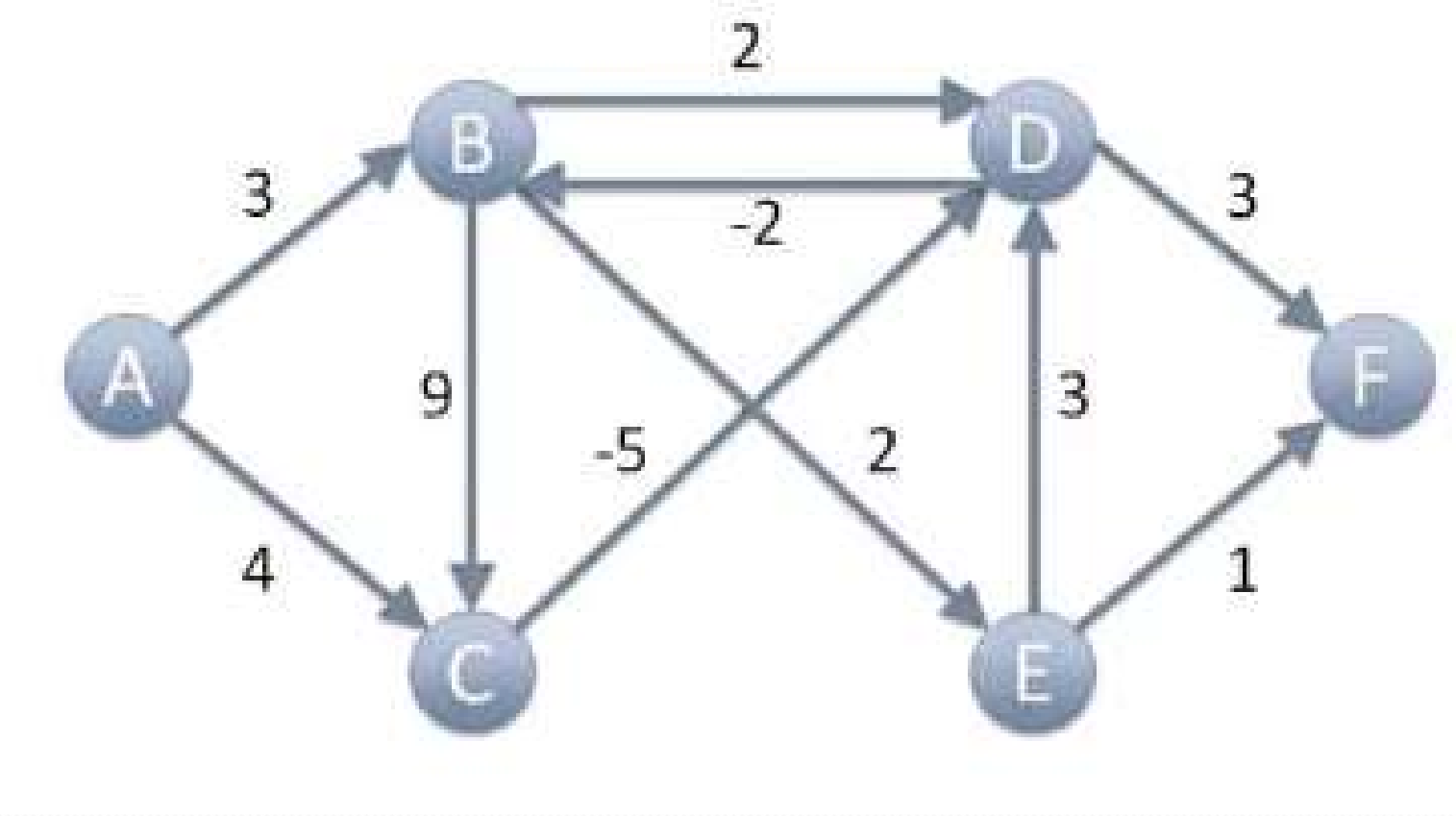


Iter	s	t	y	x	z
0	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$
1	0	6	7	$+\infty$	$+\infty$
2	0	6	7	4	2
3	0	2	7	4	2
4	0	2	7	4	-2



Execution de l 'algorithme de Bellman-Ford avec l 'ordre suivant pour les sommets z,t,x,y,s.

Exemple



Algorithme de Floyd-Warshall

Algorithme de Floyd-Warshall (1959)

Chemins les plus courts entre toutes les paires de sommets

- Si l'on veut trouver pour toutes les paires de sommets les chemins les plus courts, l'algorithme décrit précédemment peut être appliqué en considérant successivement chaque sommet comme sommet de départ

- L'algorithme de Floyd-Warshall est cependant plus efficace pour le **problème de plus courts chemins entre tous les sommets**

- Les résultats retournés par l'algorithme sont deux matrices :
  - Une **matrice D** qui contient dans chaque cellule ***ij***, la distance du plus court chemin entre les sommets ***i*** et ***j*** partant de ***i***
  - Une **matrice P**, qui contient dans chaque cellule ***ij***, le prédécesseur de ***j*** dans le plus court chemin entre ***i*** et ***j***

- Lors de l'exécution de l'algorithme, il est préférable de remplir les matrice **D** et **P** simultanément.

- Le principe est le suivant :
  - Initialiser la matrice $\mathbf{D^0}$**= W** est la matrice d'adjacence du graphe
  - Initialiser la matrice $\mathbf{P^0}$ avec la formule :

$$P^0 = \begin{cases} NIL, & si\ i = j\ ou\ w_{ij} = \infty \\ i & sinon \end{cases}$$

- Une fois ces matrices initialisées, on va faire le calcul suivant, pour **K** allant de **1** à ***n***, ou ***n*** est la taille du graphe :

- Pour la matrice $\mathbf{D^K}$:

$$d_{ij}^{K} = \begin{cases} w_{ij} & si\ K = 0 \\ \min\left(d_{ij}^{K-1}, d_{ik}^{K-1} + d_{kj}^{K-1}\right) & si\ k \geq 1 \end{cases}$$

- Pour la matrice $\mathbf{P^K}$:

$$\boldsymbol{P_{ij}^{K}} = \begin{cases} P_{ij}^{K-1}\ si\ d_{ij}^{K-1} \leq d_{ik}^{K-1} + d_{kj}^{K-1} \\ P_{kj}^{K-1}\ si\ d_{ij}^{K-1} > d_{ik}^{K-1} + d_{kj}^{K-1} \end{cases}$$

Floyd-Warshall-Algorithm(G, w)

```
Floyd-Warshall-Algorithm(G, w)
// Initialization  Initialisation de D avec W, P avec NIL
pour(i = 1; i <= n; i++)
    pour(j = 1; j <= n; j++)
        si ((i,j)ϵ E) alors
            d_ij ← w_ij // w_ij = w(i,j)
            p_ij ← i
        fsi
    fpour
Fpour
// Determination des chemins sans sommet intermédiaire

Pour (k = 1; k <= n; k++)
    Pour (i = 1; i <= n; i++)
        Pour (j = 1; j <= n; j++)
            si (d_ij> d_ik+ d_kj) alors
                d_ij ← d_ik+ d_kj
                p_ij ← p_kj
            fsi
        fpour
    fpour
fpour
```

Floyd-Warshall-Algorithm(G, w) -1

```
Floyd-Warshall-Algorithm(G, w)
// Initialization
   Initialisation de D avec W, P avec NIL
// Determination des chemins sans sommet intermédiaire

pour(i = 1; i <= n; i++)
   pour(j = 1; j <= n; j++)
      si ((i,j)∈ E) alors
         d_ij ← w_ij // w_ij = w(i,j)
         p_ij ← i
      fsi
   fpour
fpour
```

Floyd-Warshall-Algorithm(G, w) -2

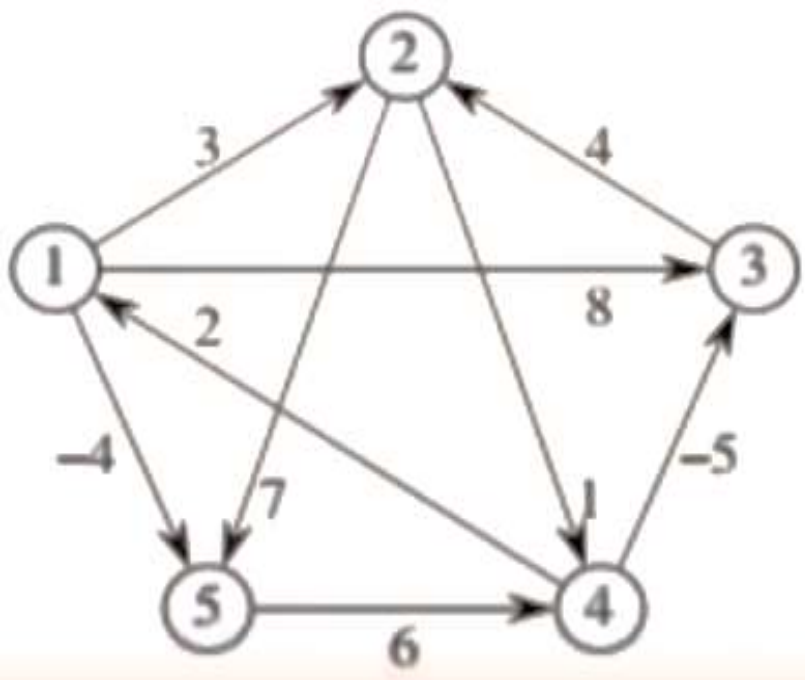
```
// On tente de raccourcir les chemins par le
//sommet intermédiaire k

Pour (k = 1; k <= n; k++)
   Pour (i = 1; i <= n; i++)
      Pour (j = 1; j <= n; j++)
         si (dij> dik+ dkj) alors
            dij ← dik+ dkj
            pij ← pkj
         fsi
      fpour
   fpour
fpour
```

- Si pendant l'exécution de l'algorithme on a $d_{ii} < 0$ l'algorithme peut ***s'arrêter*** car il y a un circuit ***absorbant***, et donc il n'y a pas de solution.

- **Remarque:**

La formule de calcul de l'Algorithme de Floyd-Warshall correspond à la formule de calcul de la multiplication de matrices

Exemple



$$D^{O} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

$$P^{O} = \begin{pmatrix} NIL & 1 & 1 & NIL & 1 \\ NIL & NIL & NIL & 2 & 2 \\ NIL & 3 & NIL & NIL & NIL \\ 4 & NIL & 4 & NIL & NIL \\ NIL & NIL & NIL & 5 & NIL \end{pmatrix}$$

$$D^0 = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad P^0 = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & \text{NIL} & 4 & \text{NIL} & \text{NIL} \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^1 = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad P^1 = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^1 = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad P^1 = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

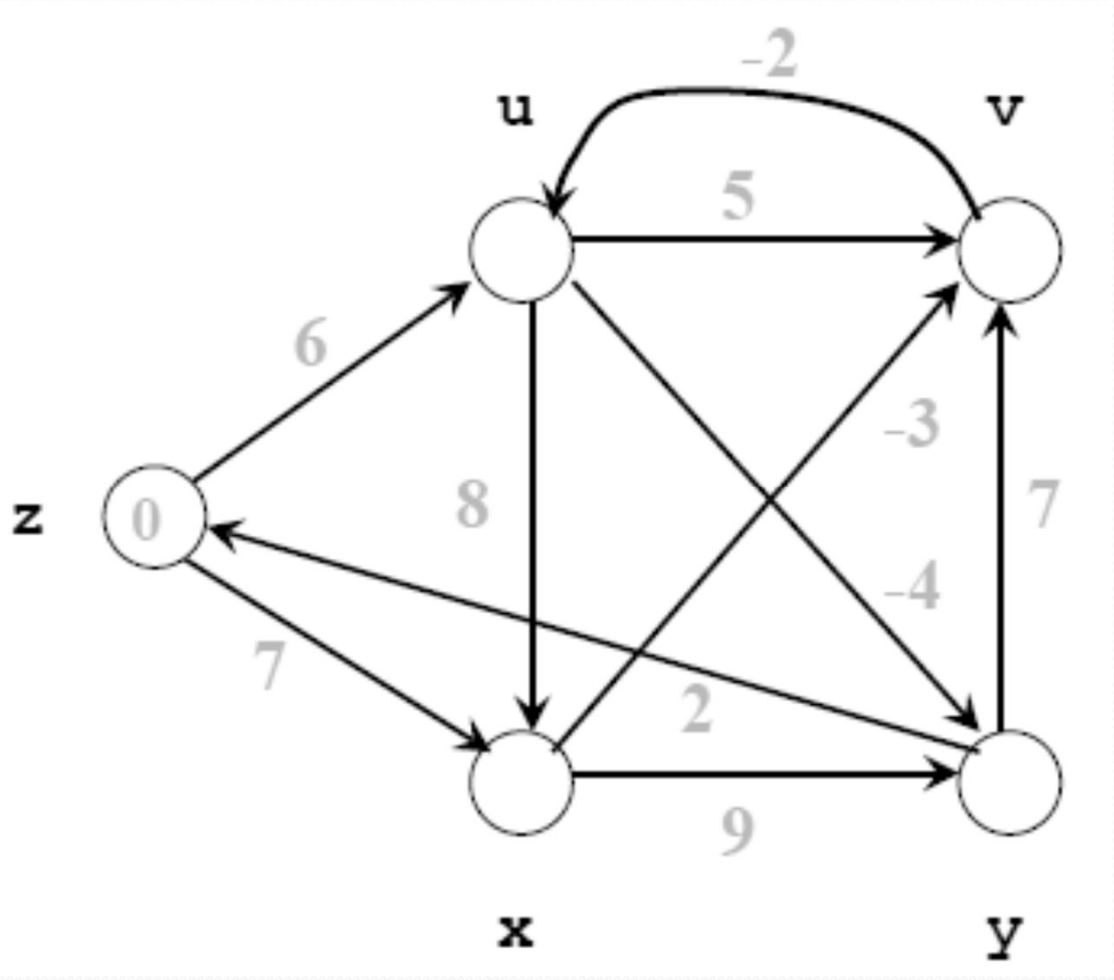
$$D^2 = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad P^2 = \begin{pmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^3 = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad P^3 = \begin{pmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^4 = \begin{pmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad P^4 = \begin{pmatrix} \text{NIL} & 1 & 4 & 2 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$

$$D^4 = \begin{pmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \qquad P^4 = \begin{pmatrix} \text{NIL} & 1 & 4 & 2 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$

$$D^5 = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \qquad P^5 = \begin{pmatrix} \text{NIL} & 3 & 4 & 5 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$


-2
u
v
5
6
-3
z
0
8
7
-4
7
2
9
x
y