Chapitre2

Problèmes d’ACM et de cheminement

Arbre couvrant de poid 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.

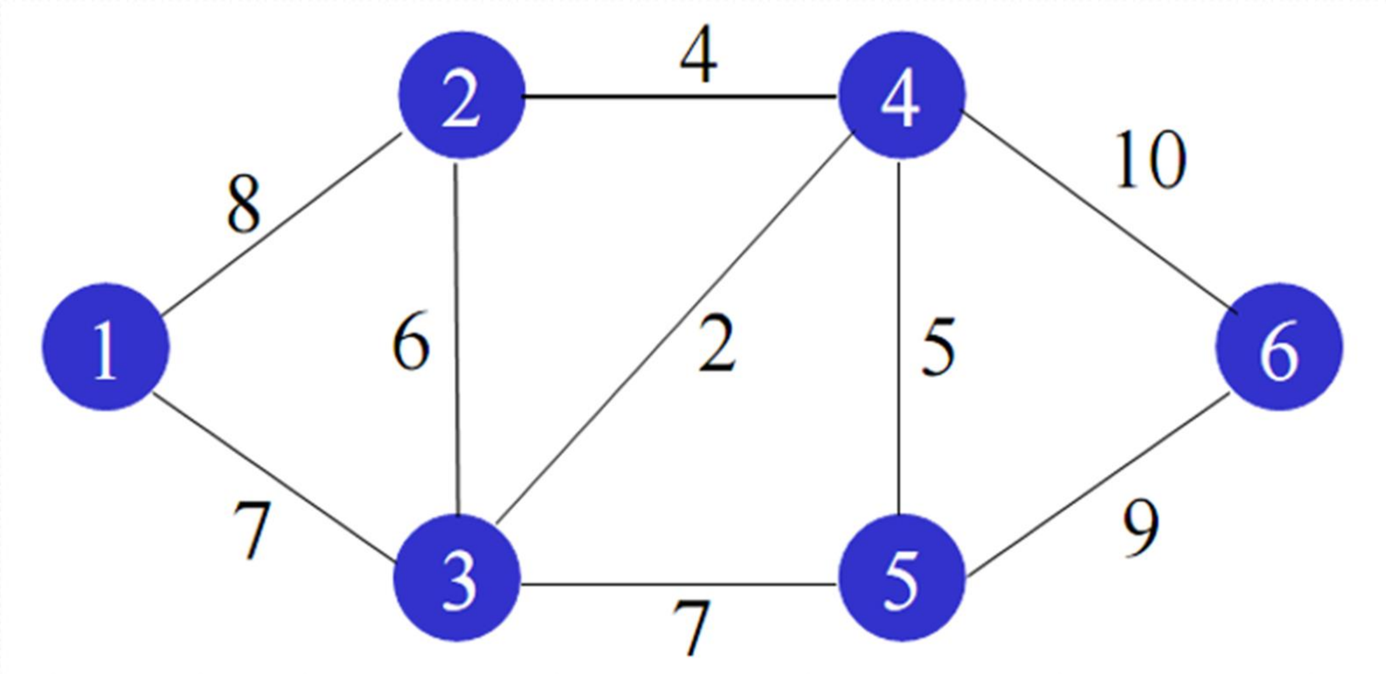
Algorithme de Kruskal

Algorithme

procedure KRUSKAL
    $\forall i \in E,\ cc(i) \leftarrow i$
    Trier les arêtes
    $T \leftarrow \varnothing$
    for $k = 1$ à $n - 1$ **do**
        Choisir une arête (x, y)
        if $cc(x) \neq cc(y)$ **then**
            $T = T \cup \{(x, y)\}$
            for all sommet i **do**
                if $cc(i) = cc(x)$ **then**
                    $cc(i) \leftarrow cc(y)$
                end if
            end for
        end if
    end for
end procedure

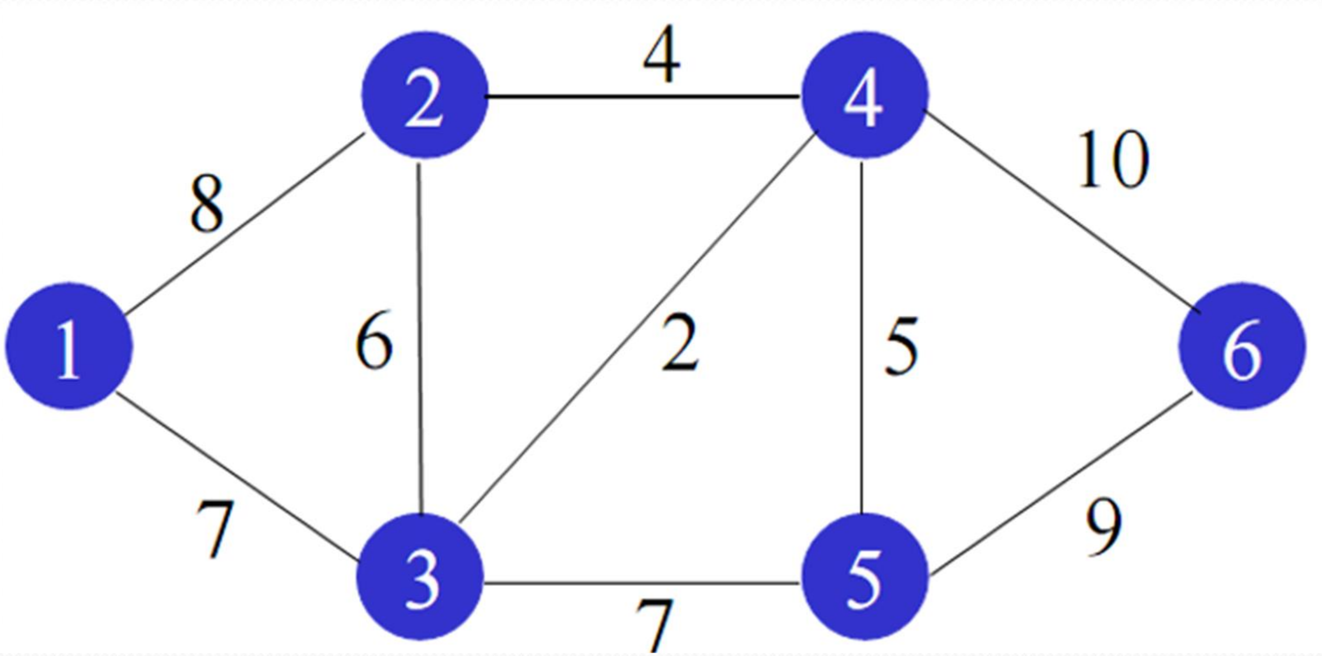
Algorithme de Kruskal

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

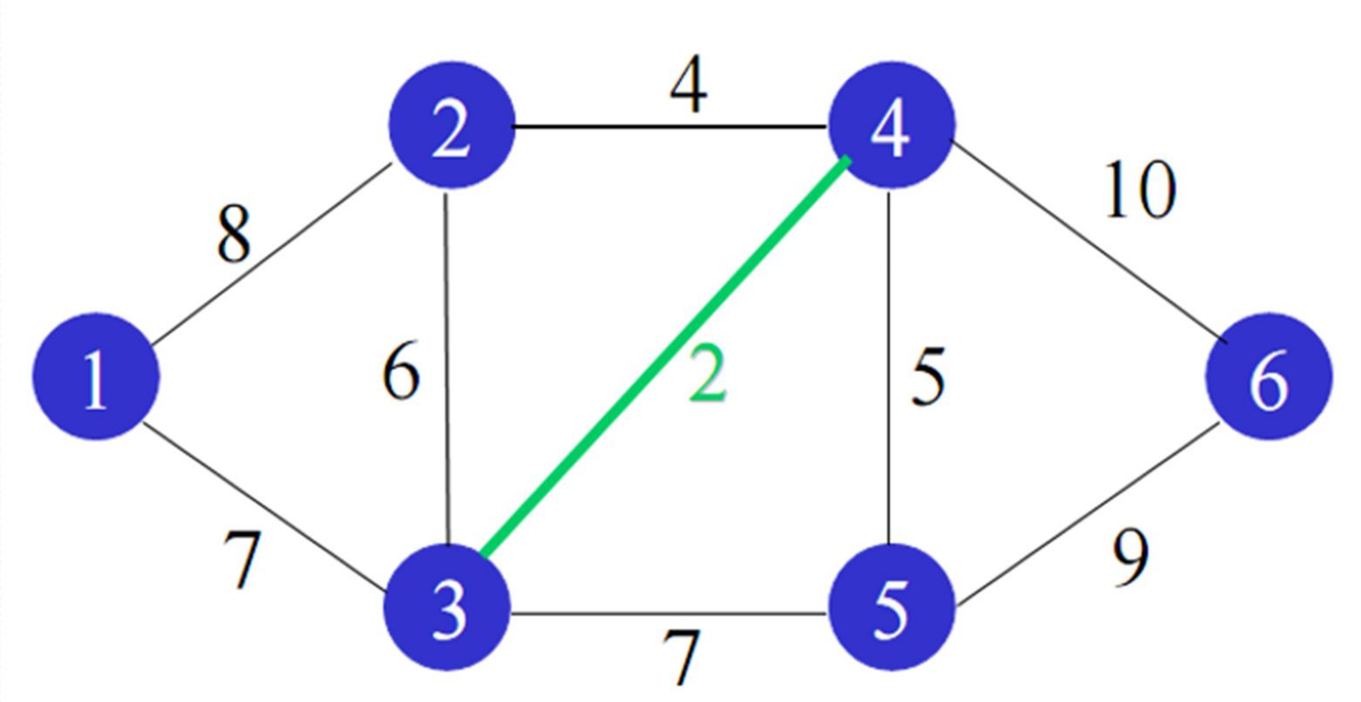
1. Trier les arcs { (3,4), (2,4), (4,5), (2,3), (1,3), (3,5), (1,2), (5,6),(4,6) }
2. Construire un arbre T = { }



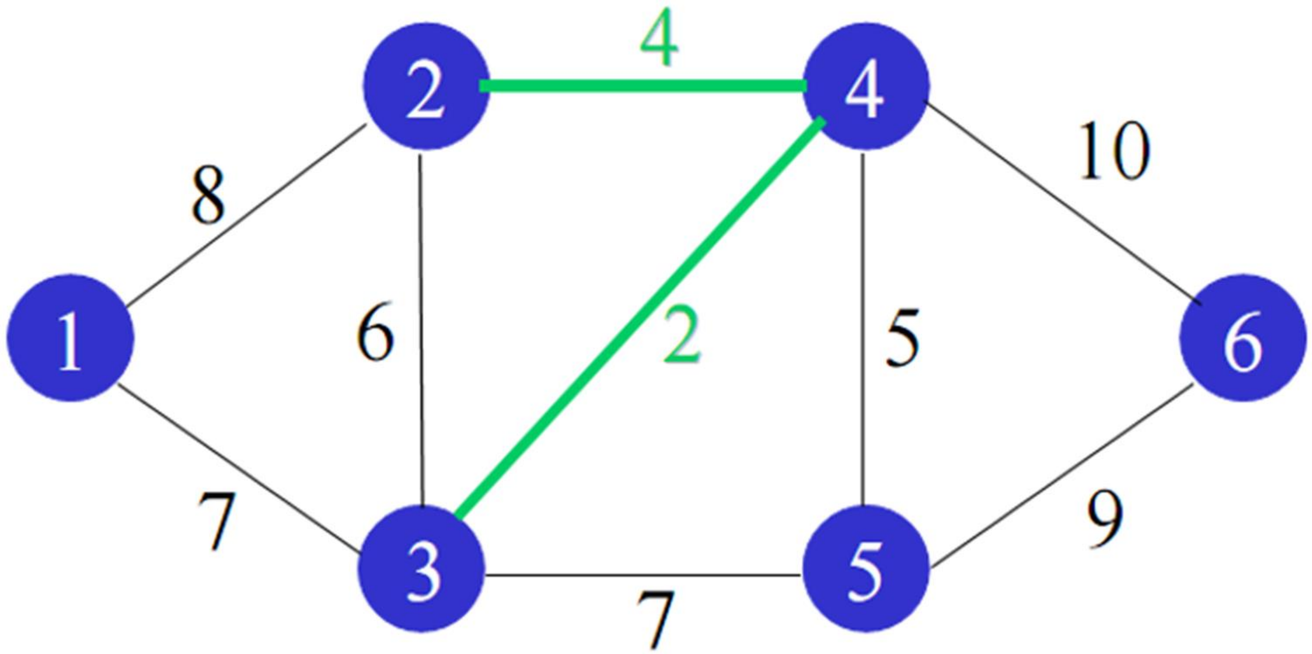
- On évalue (3,4) :

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

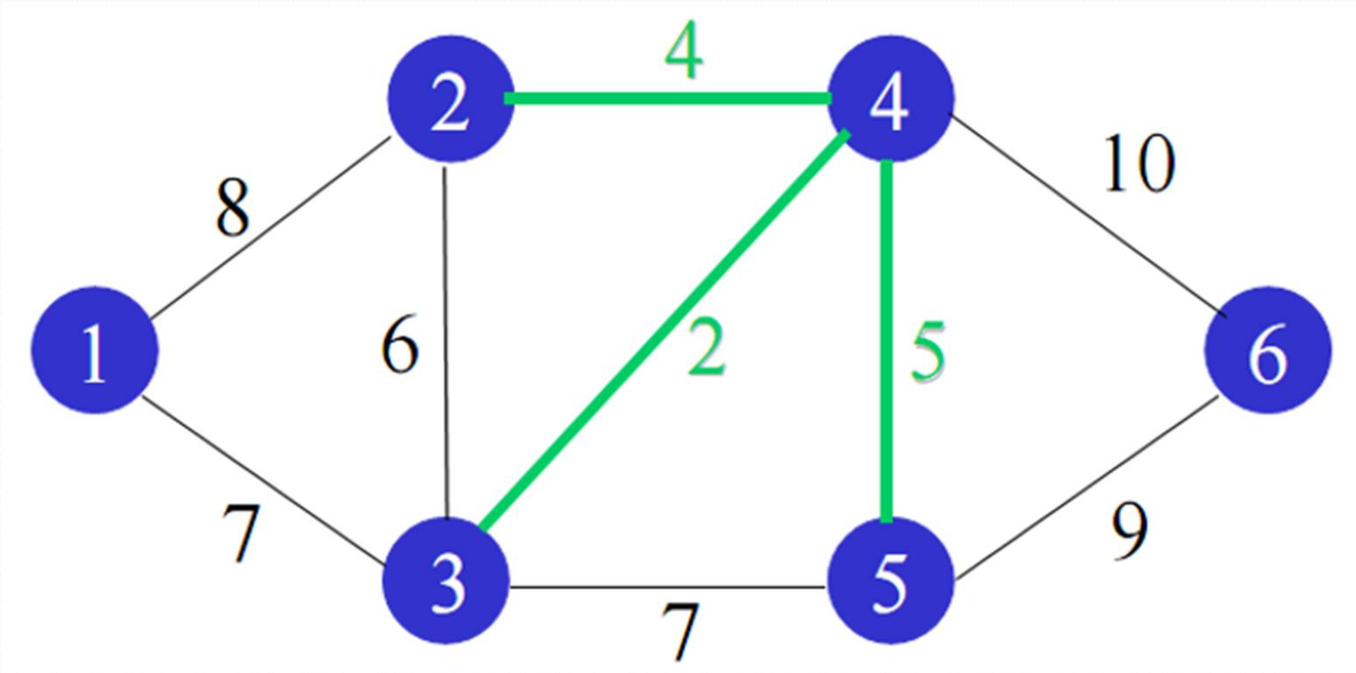
T= { (3,4) }



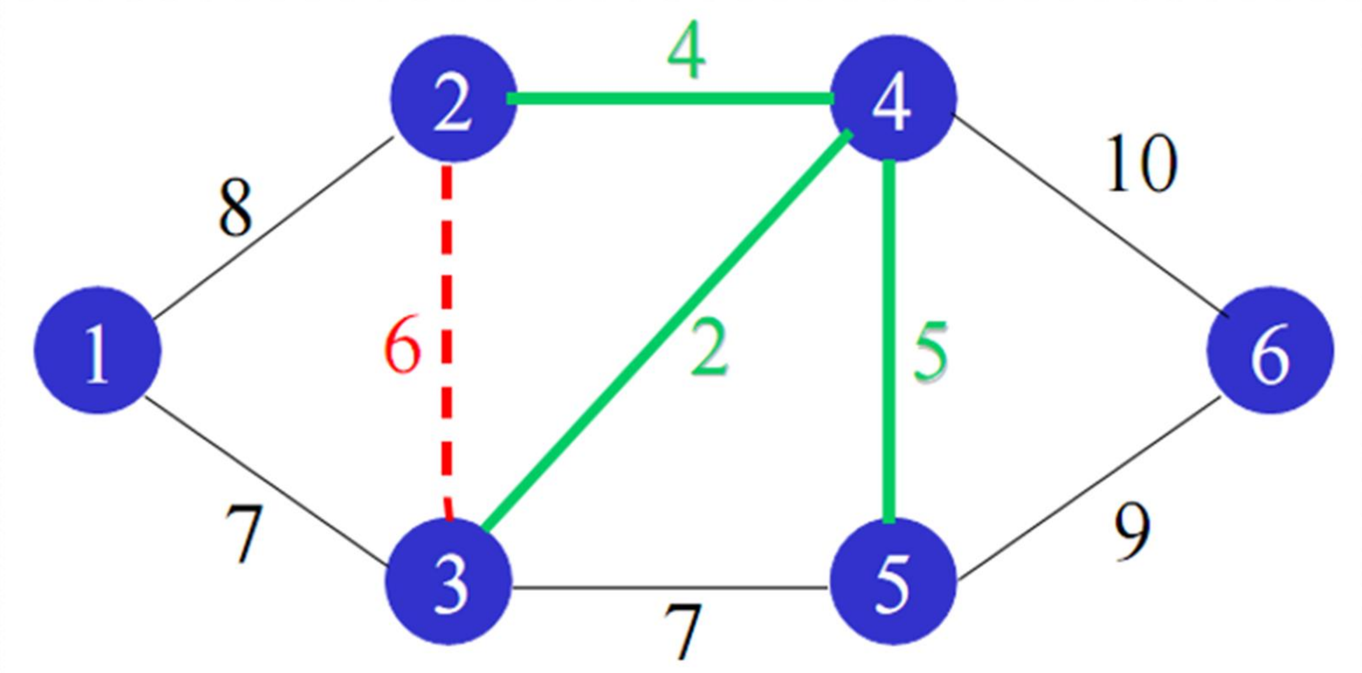
- On évalue (2,4) :
- {~~(3,4)~~, ~~(2,4)~~, (4,5), (2,3), (1,3), (3,5), (1,2), (5,6),(4,6) }
  T= {(3,4), (2,4)}



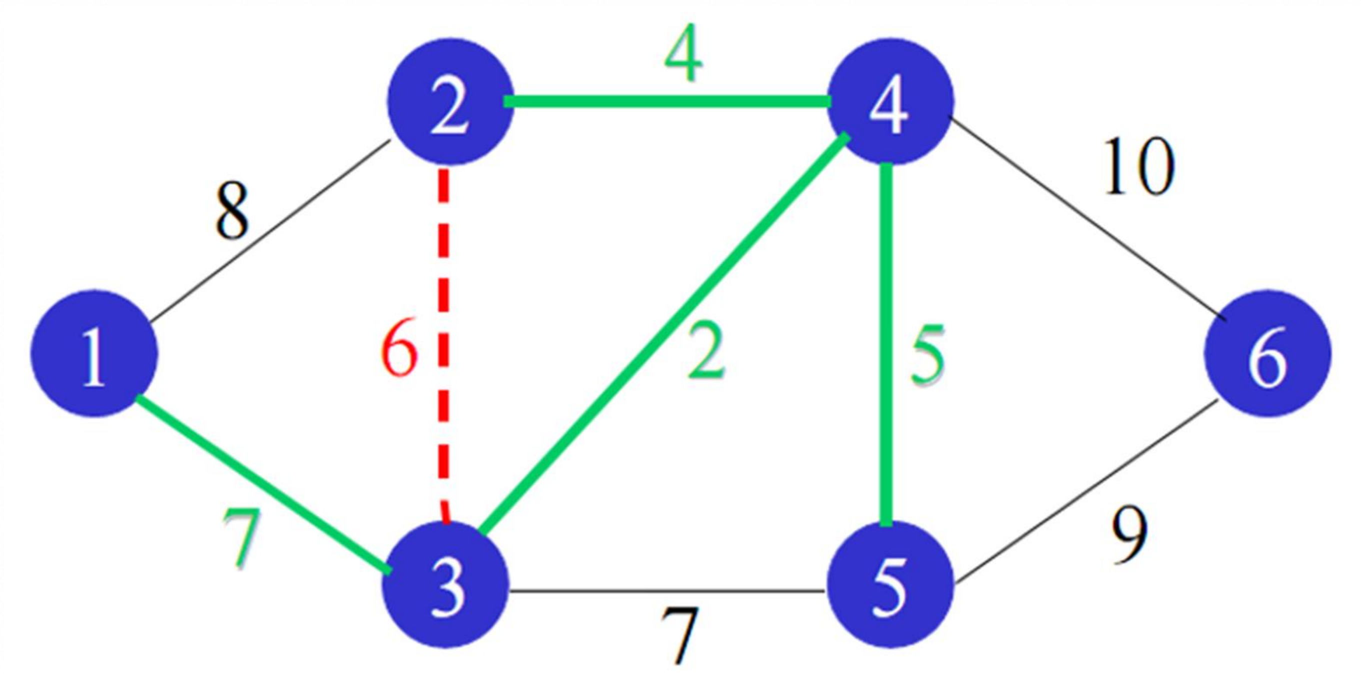
- On évalue (4,5):
- {~~(3,4)~~, ~~(2,4)~~, ~~(4,5)~~, (2,3), (1,3), (3,5), (1,2), (5,6),(4,6) }
  T= { (3,4), (2,4), (4,5) }



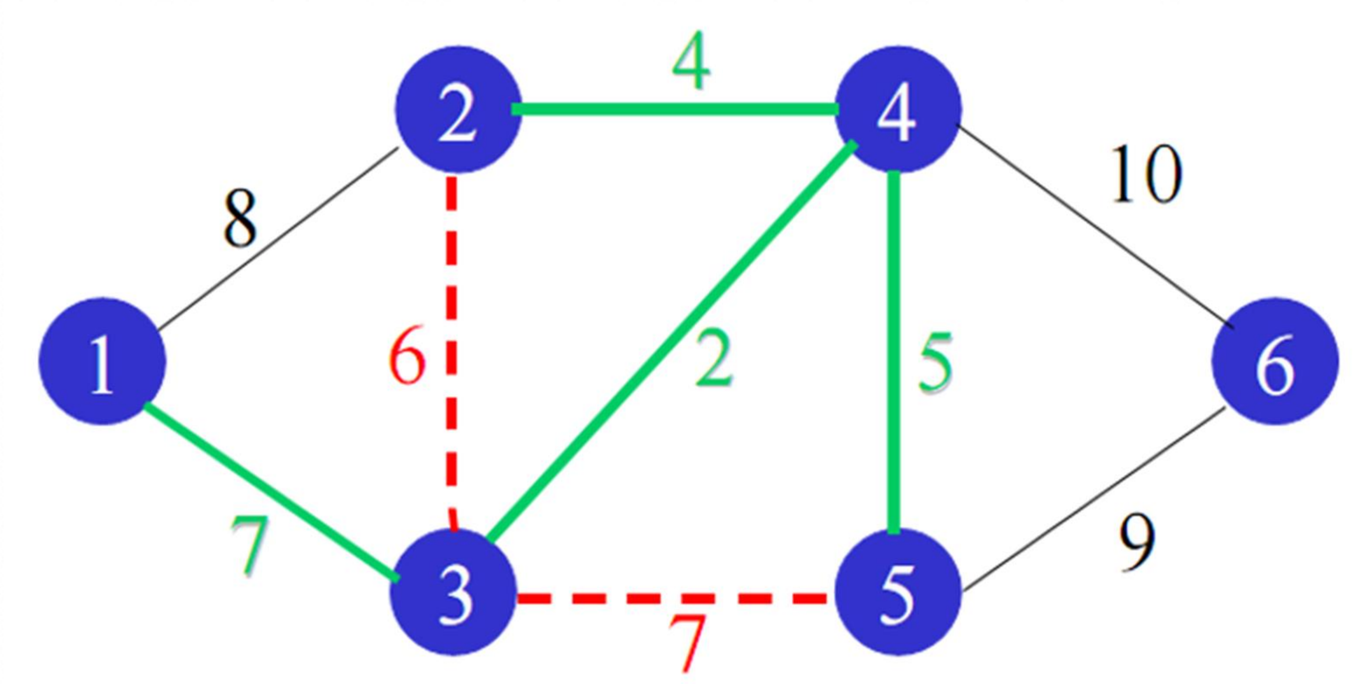
- On évalue (2,3):
- {~~(3,4)~~, ~~(2,4)~~, ~~(4,5)~~, ~~(2,3)~~, (1,3), (3,5), (1,2), (5,6),(4,6) }
  T= { (3,4), (2,4), (4,5) }



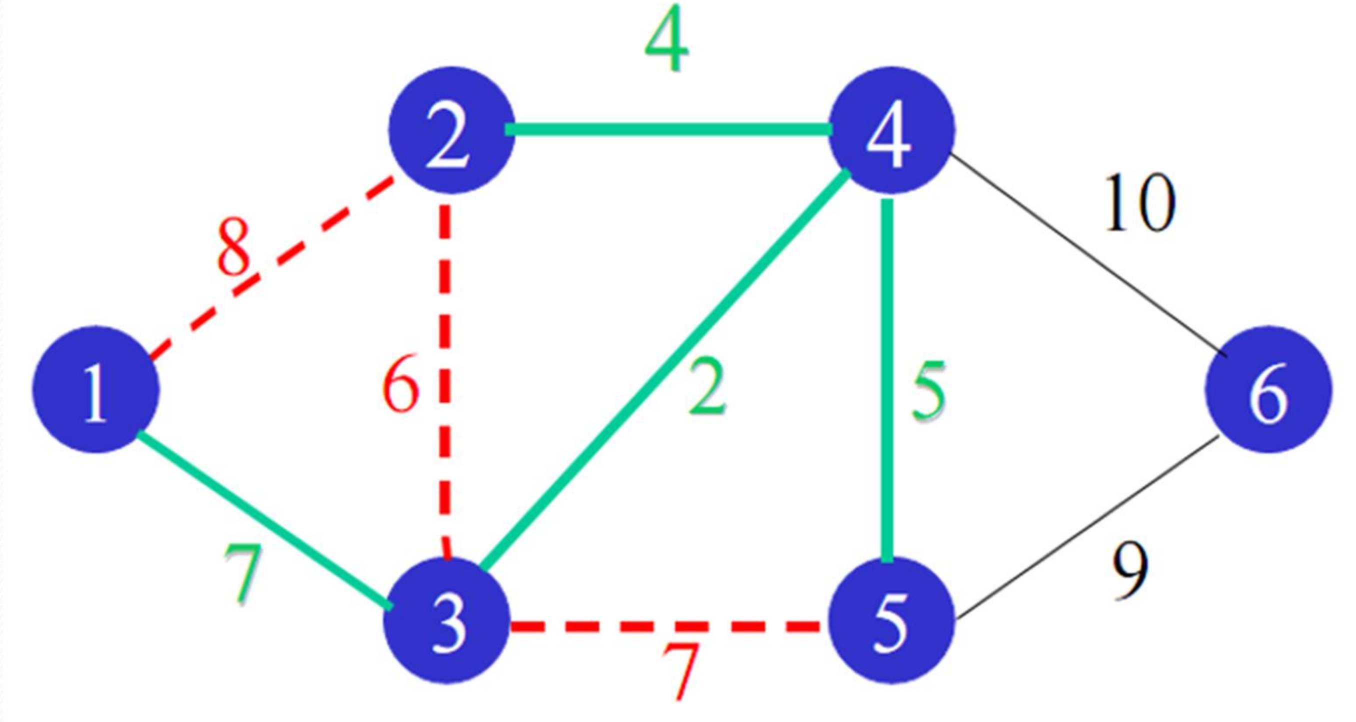
- On évalue (1,3):
- {~~(3,4)~~, ~~(2,4)~~, ~~(4,5)~~, ~~(2,3)~~, ~~(1,3)~~, (3,5), (1,2), (5,6),(4,6) }
  T= { (3,4), (2,4), (4,5), (1,3) }



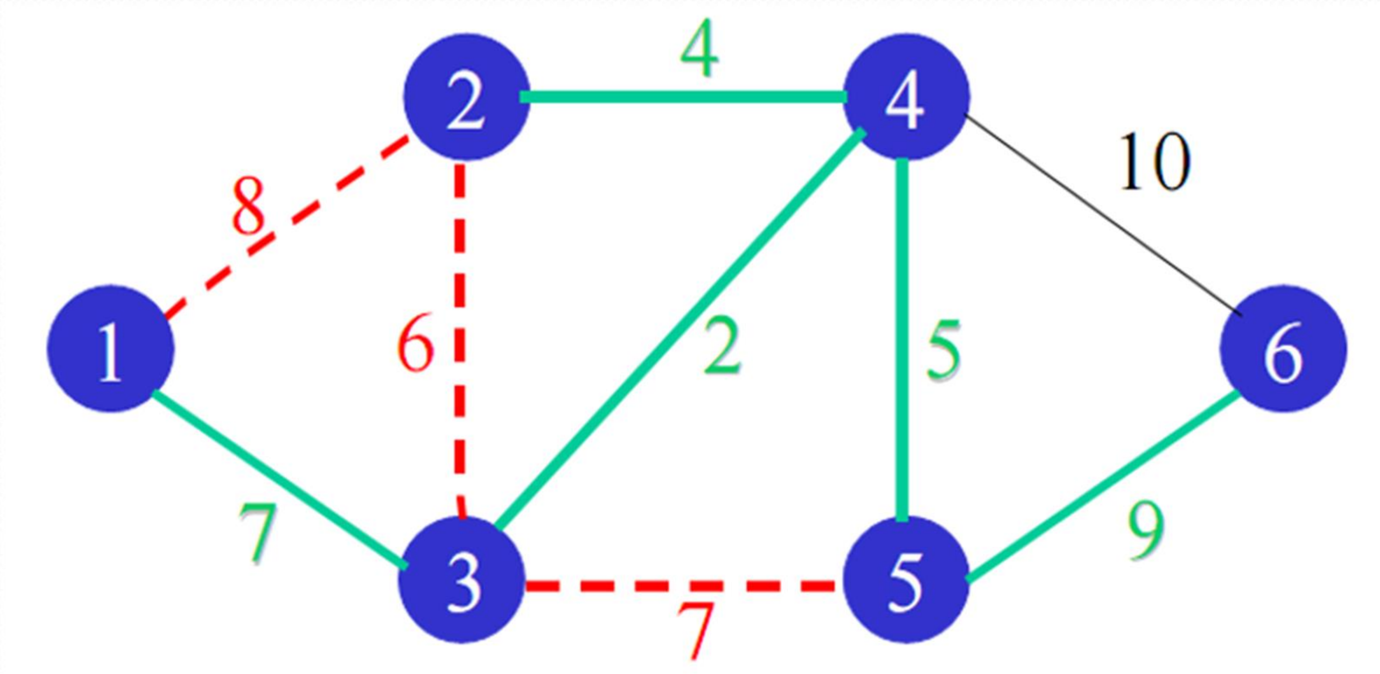
- On évalue (3,5):
- {~~(3,4)~~, ~~(2,4)~~, ~~(4,5)~~, ~~(2,3)~~, ~~(1,3)~~, ~~(3,5)~~, (1,2), (5,6),(4,6) }
  T= { (3,4), (2,4), (4,5), (1,3) }



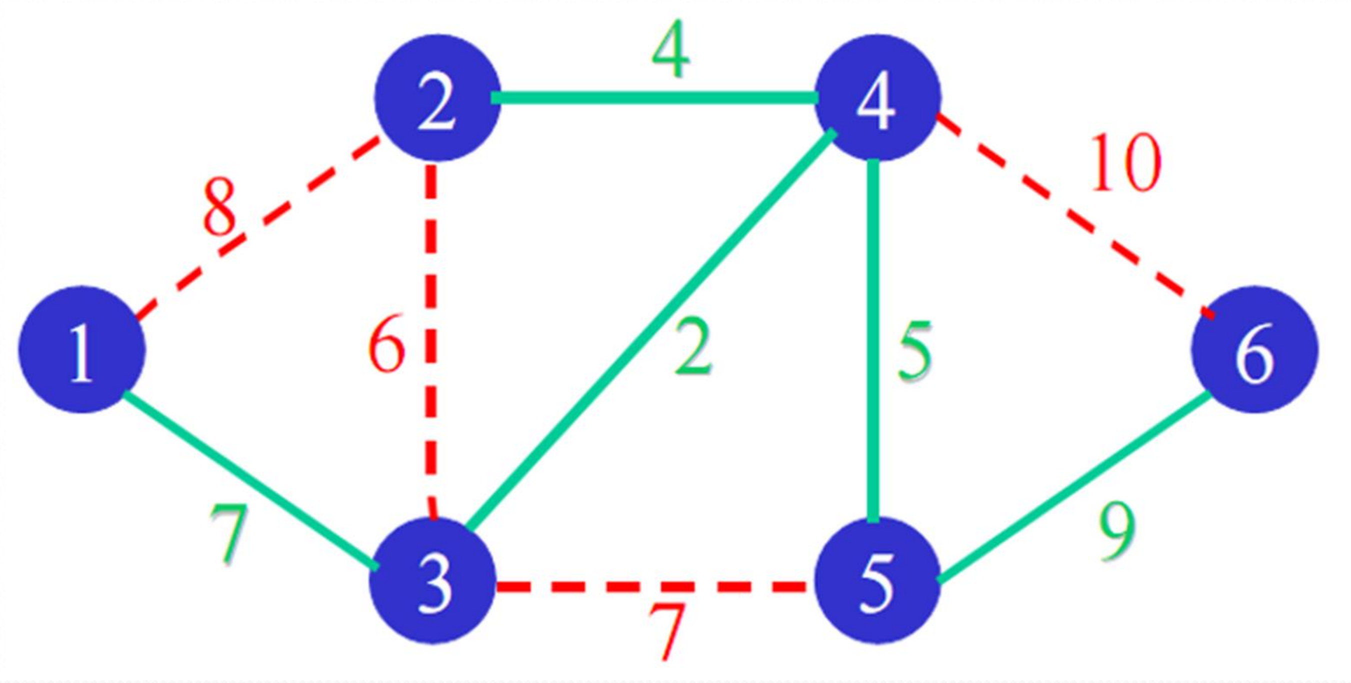
- On évalue (1,2):
- {~~(3,4)~~, ~~(2,4)~~, ~~(4,5)~~, ~~(2,3)~~, ~~(1,3)~~, ~~(3,5)~~, ~~(1,2)~~, (5,6) ,(4,6) }
  T= { (3,4), (2,4), (4,5), (1,3) }

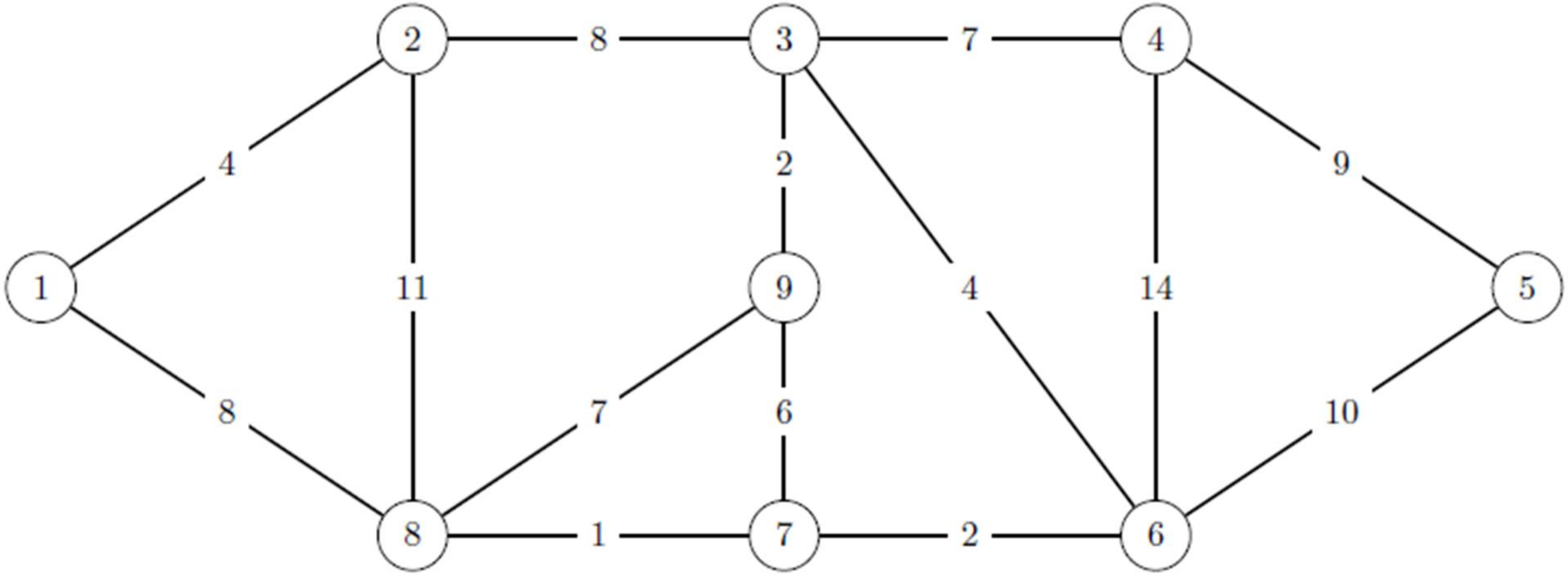


- On évalue (5,6):
- {~~(3,4)~~, ~~(2,4)~~, ~~(4,5)~~, ~~(2,3)~~, ~~(1,3)~~, ~~(3,5)~~, ~~(1,2)~~, ~~(5,6)~~, (4,6) }
  T= { (3,4), (2,4), (4,5), (1,3), (5,6) }



- 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) }





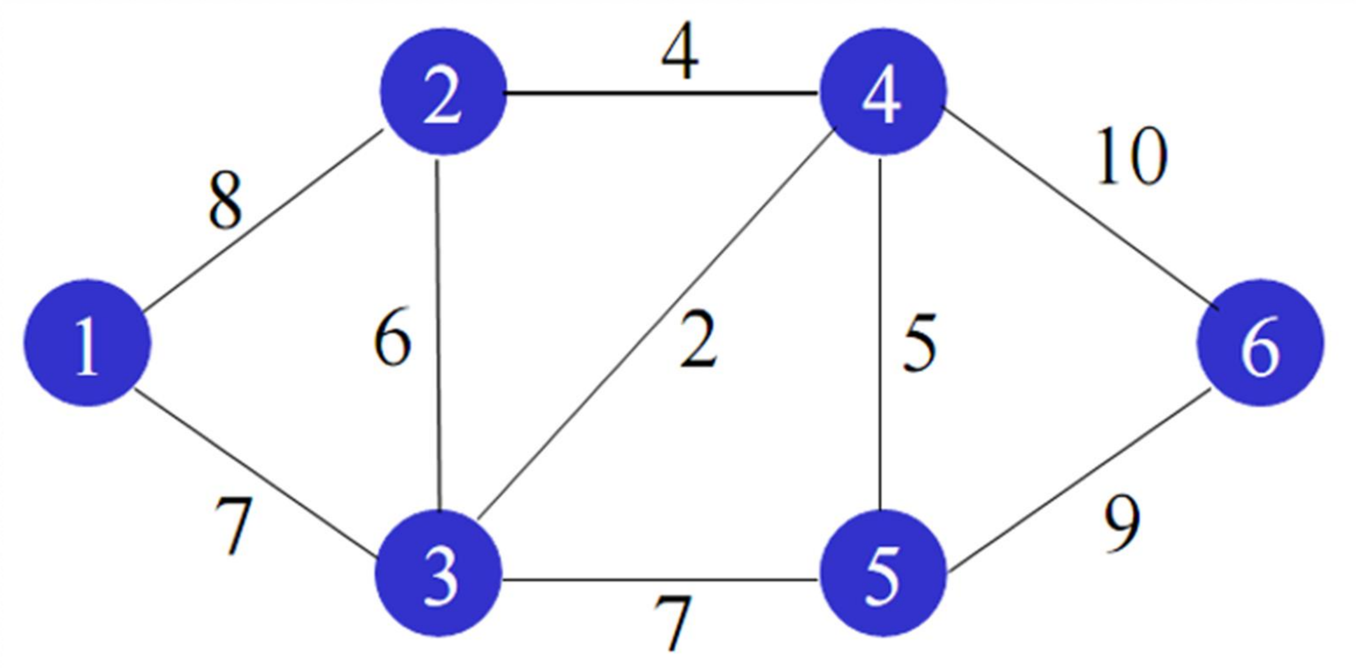
On trie les arêtes du graphe. On obtient l'ordre suivant : {7; 8} < {3; 9} = {6; 7}< {1; 2} = {3; 6} < {7; 9} < {8; 9} = {3; 4} < {2; 3} = {1; 8} < {4; 5} < {5; 6} < {2; 8} < {4; 6}.

Algorithme de Prim

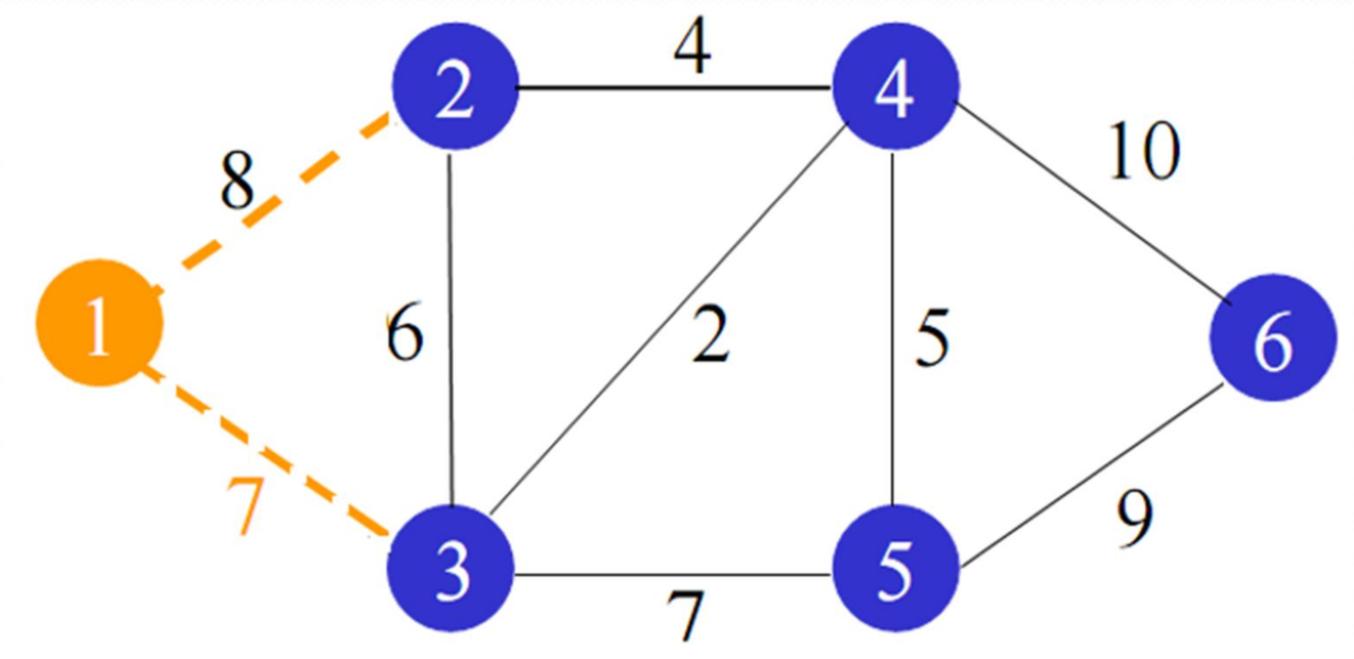
0. Initialiser T={}, S={} et W= V (sommets)
1. Choisir un nœud $i \in W$, poser $S = \{ i \}$ et $W = W \setminus \{ i \}$
2. Choisir un nœud $j \in W$ tel que l'arc (i, j), où $i \in S$, ait le $c_{i,j}$ le plus petit;
3. Si (i,j) ne forme pas de cycle alors inclure cet arc dans l'arbre, i.e. $T = T \cup (i,j)$, $j \in S$ et $W = W \setminus \{ j \}$ 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

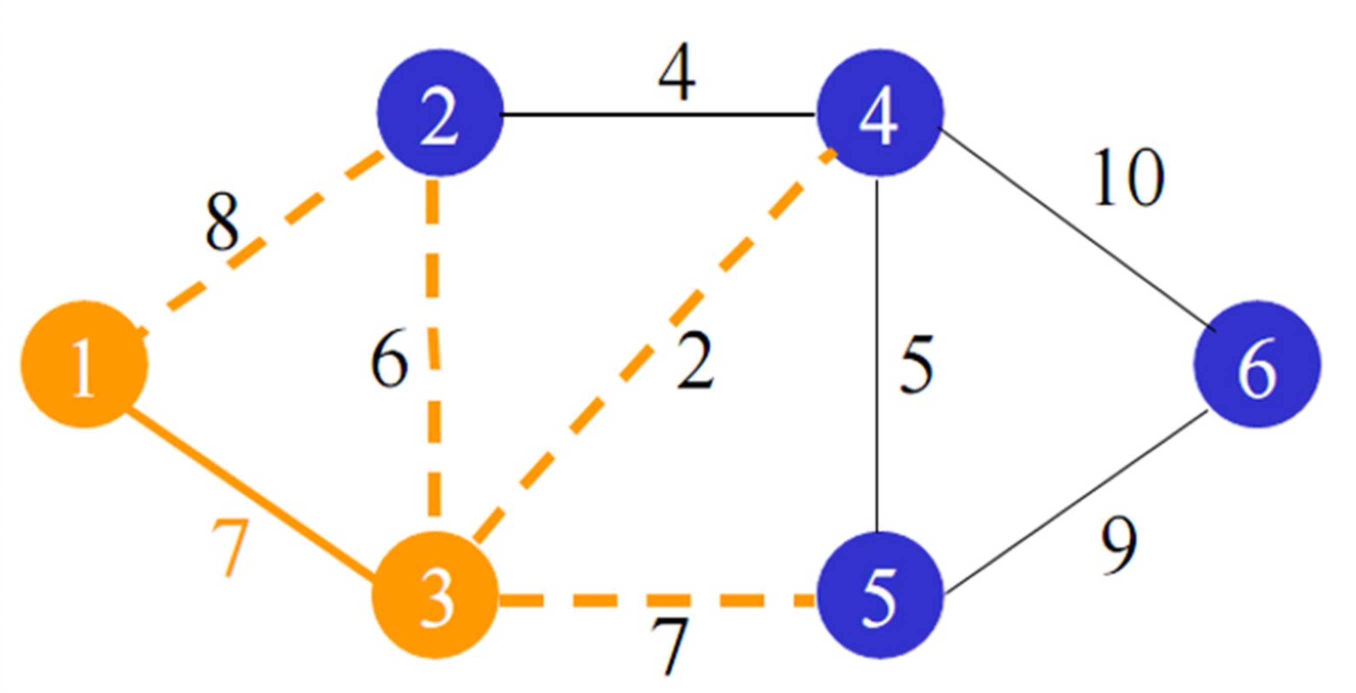
- S = { }
- W = { 1, 2, 3, 4, 5, 6 }
- T = { }



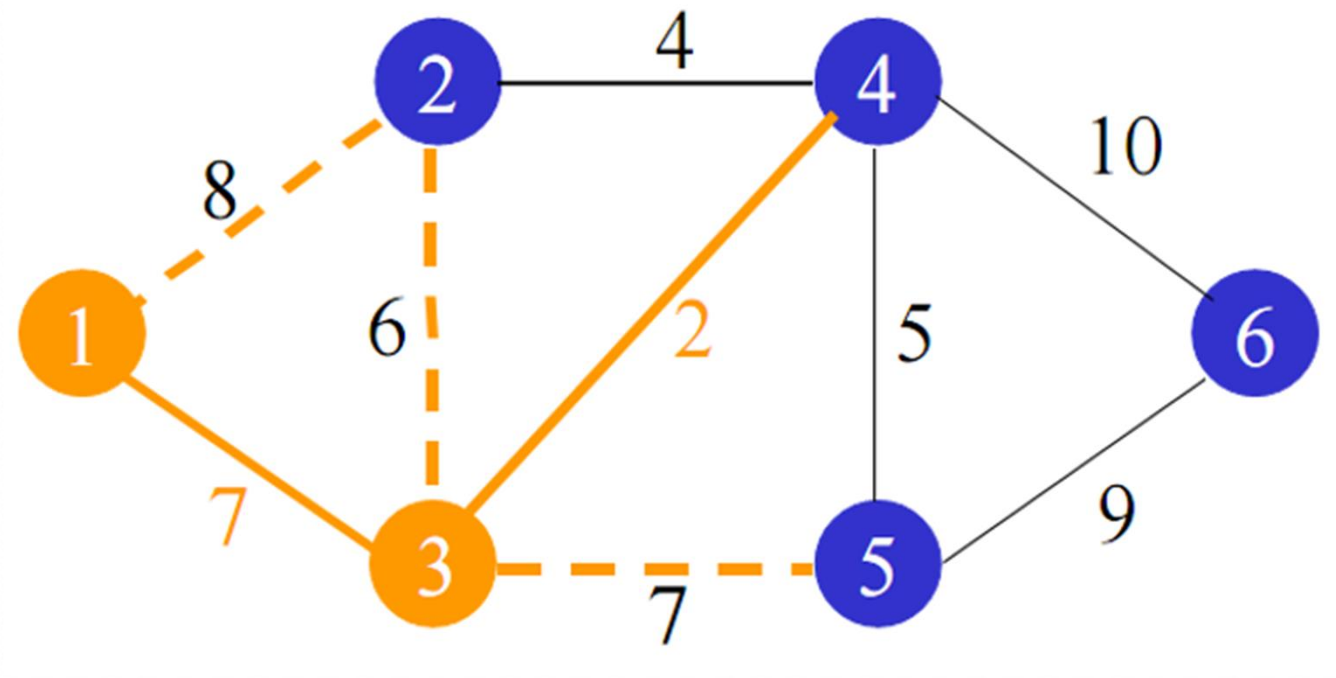
- S = { 1 }
- W = { 2, 3, 4, 5, 6 }
- T = { }



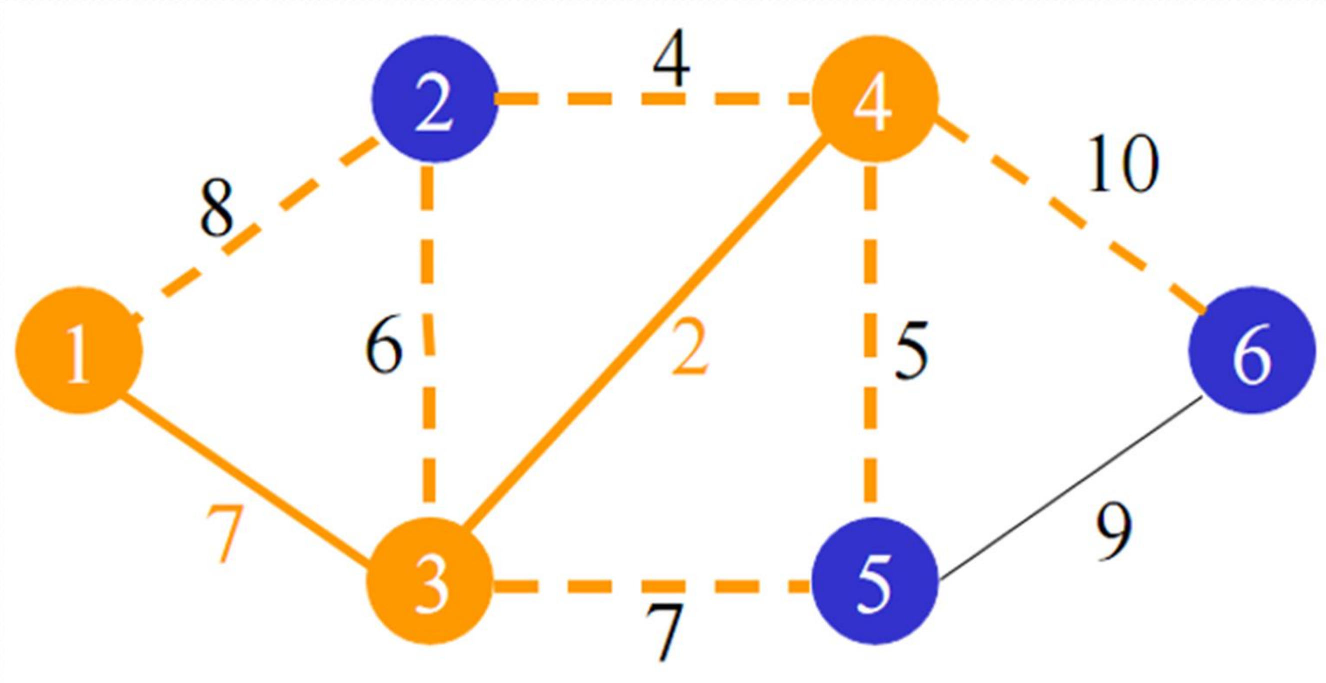
- S = { 1,3 }
- W = { 2, 4, 5, 6 }
- T = { (1,3) }



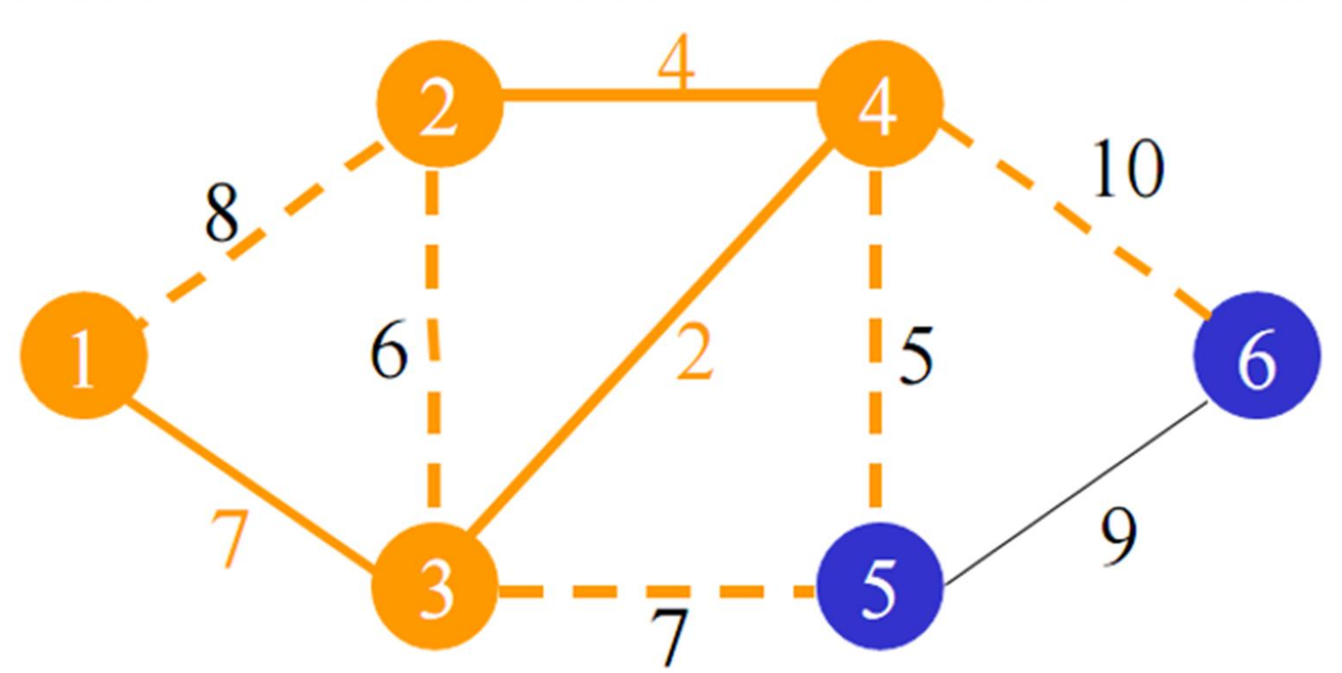
- S = { 1,3 }
- W = { 2, 4, 5, 6 }
- T = { (1,3), (3,4) }



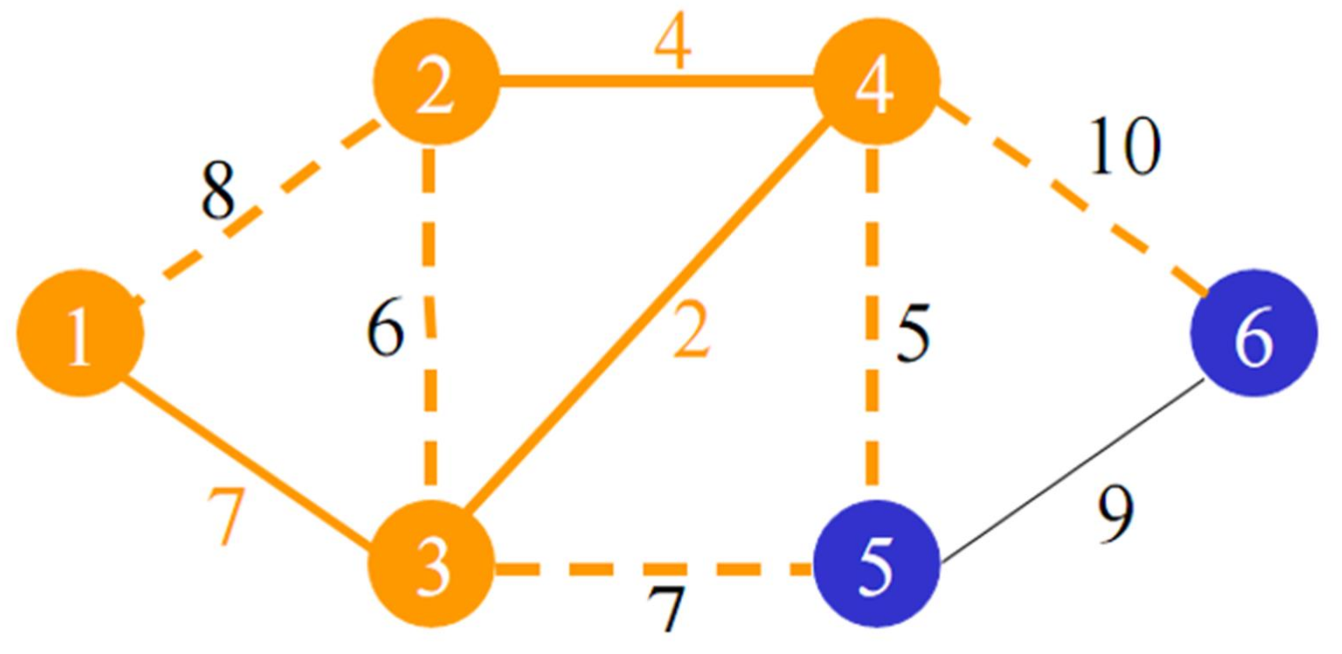
- S = { 1,3,4 }
- W = { 2, 5, 6 }
- T = { (1,3), (3,4) }



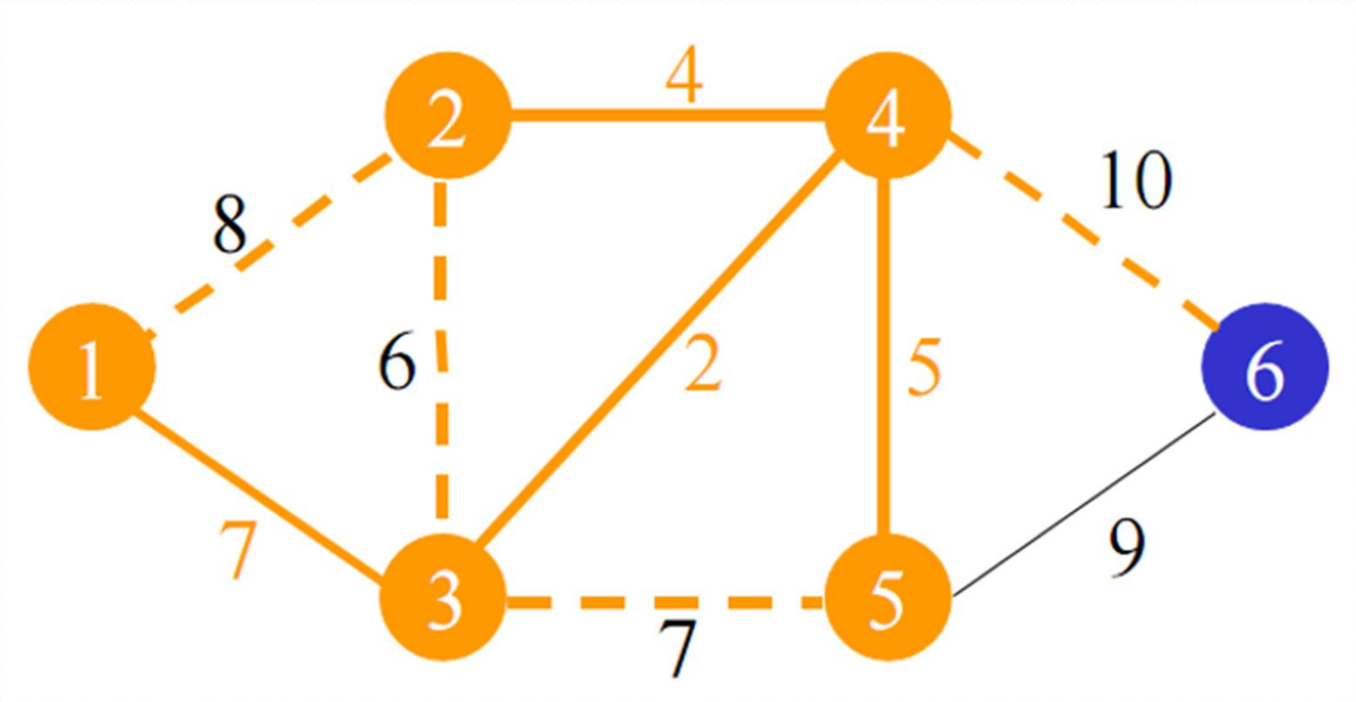
- S = { 1,3,4 }
- W = { 5, 6 }
- T = { (1,3), (3,4), (2,4) }



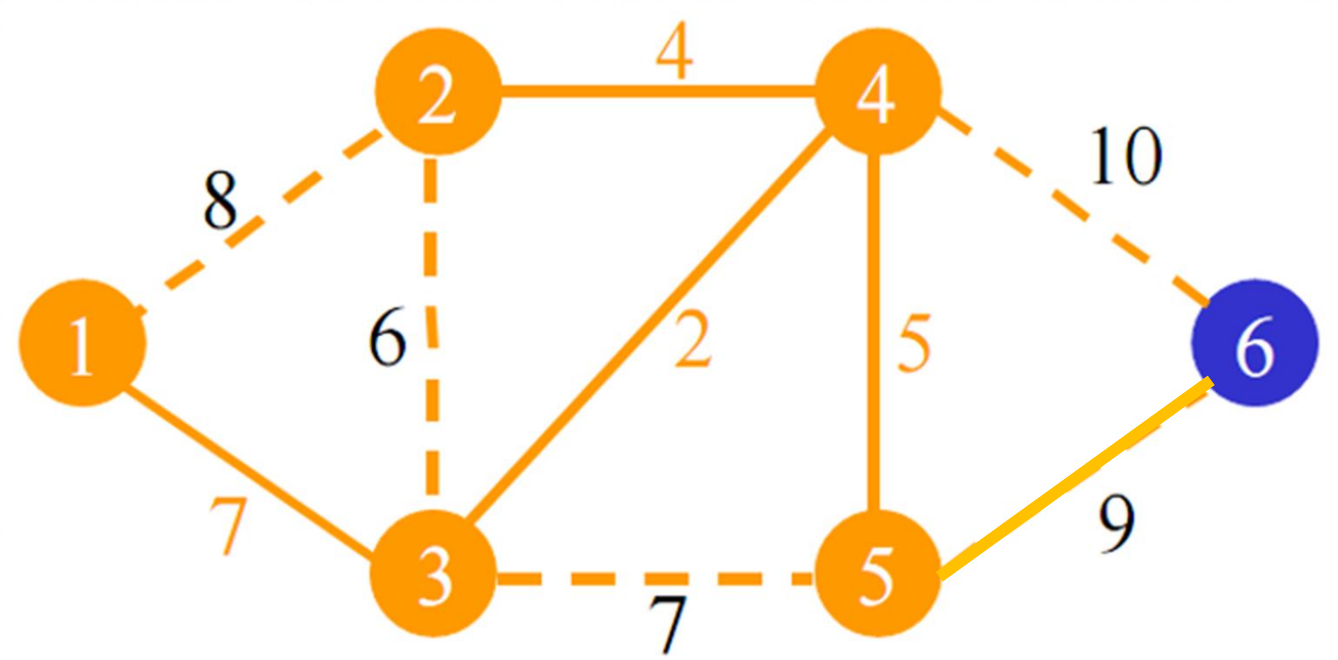
- S = { 1,3,4,2 }
- W = { 5, 6 }
- T = { (1,3), (3,4), (2,4) }

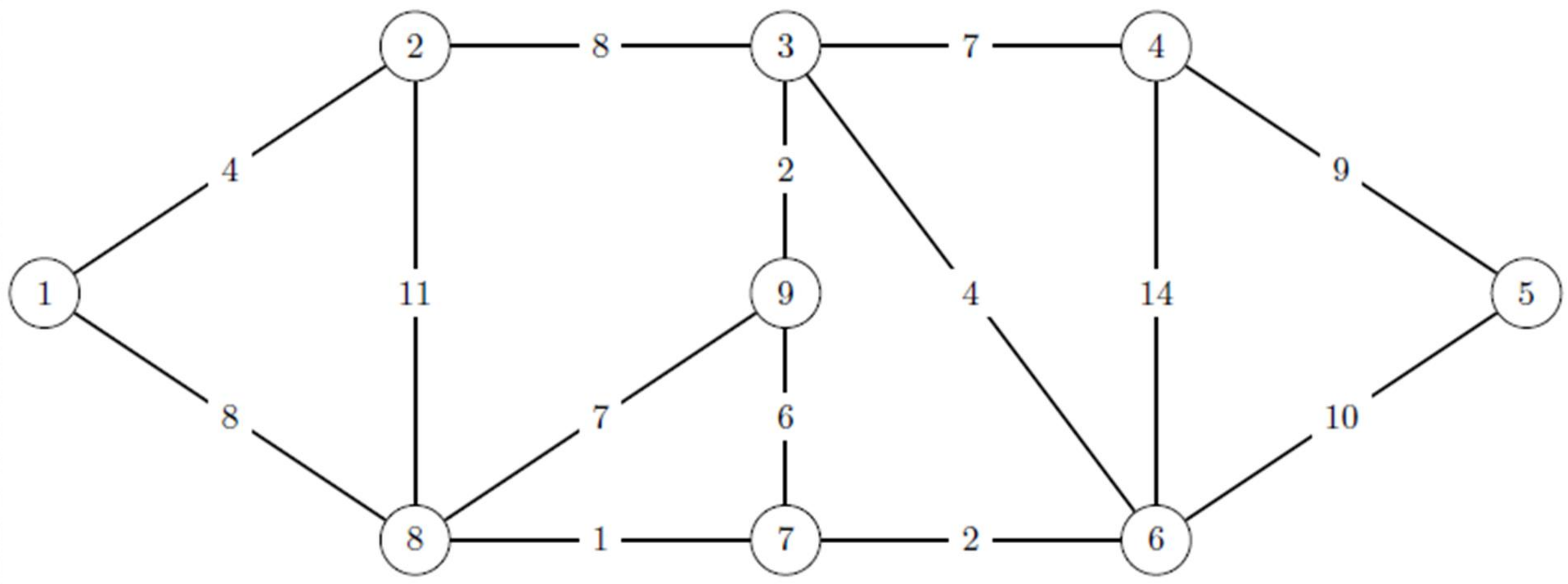


- S = { 1,3,4,2 }
- W = { 6 }
- T = { (1,3), (3,4), (2,4), (4,5), (5,6)}



- 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

Problème du plus court chemin

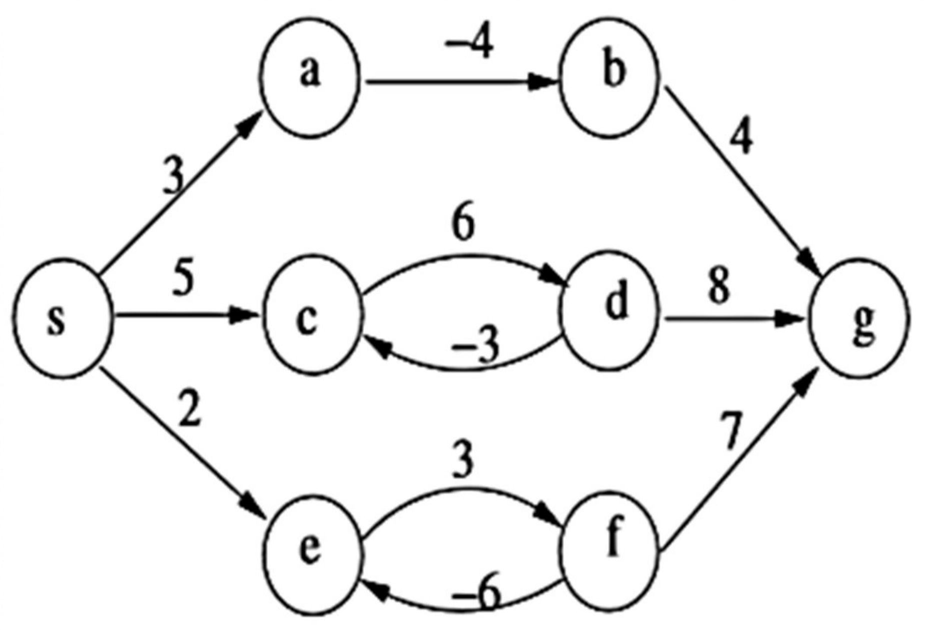
- Ce problème du **Plus Court Chemin** (PCC) peut être posé de la façon suivante:
- Etant donné un graphe orienté valué **G (S,A,*v*)**, on associe à chaque arc **u(i,j)** un nombre réel **l(u)** ou $\mathbf{l_{ij}}$, appelé la **longueur** ou le poids de l'arc, le PCC entre deux sommets **s** (source) 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

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**	

Condition Nécessaire:

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



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 :

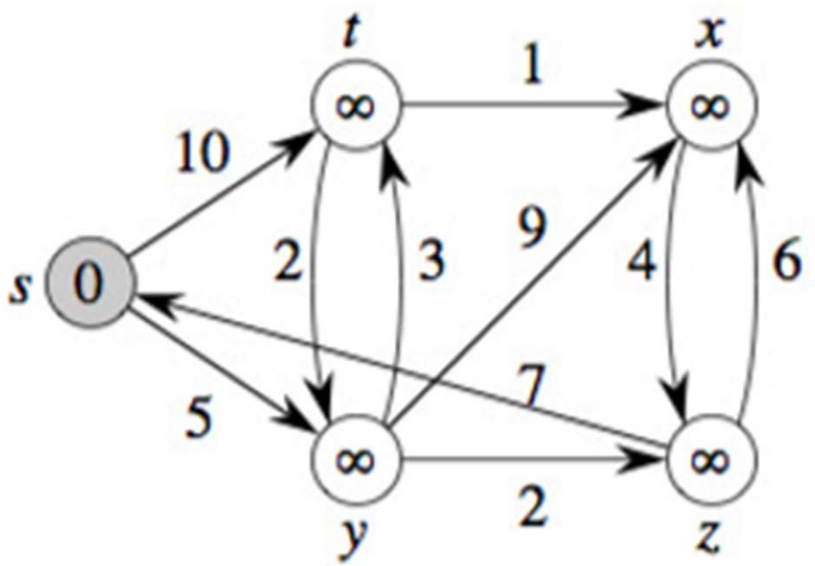
Données : Un graphe $G(V, E)$, une fonction w **positive** et un sommet s.

Instructions :

```
∀ v ∈ V, φ[v] = +∞ et π[v] = −1 ; // Initialisation
φ[s] = 0 ;
Soit Q un ensemble de sommets initialisé à V ;
TANT QUE Q est non vide :
    Soit u le sommet de Q avec φ[u] minimum
    Retirer u de Q
    POUR TOUT v ∈ Γ⁺(u) : //successeur de u
        SI (φ[v] > φ[u] + w(u, v)) ALORS // Mise à jour
            φ[v] = φ[u] + w(u, v) ;
            π[v] = u ;
```

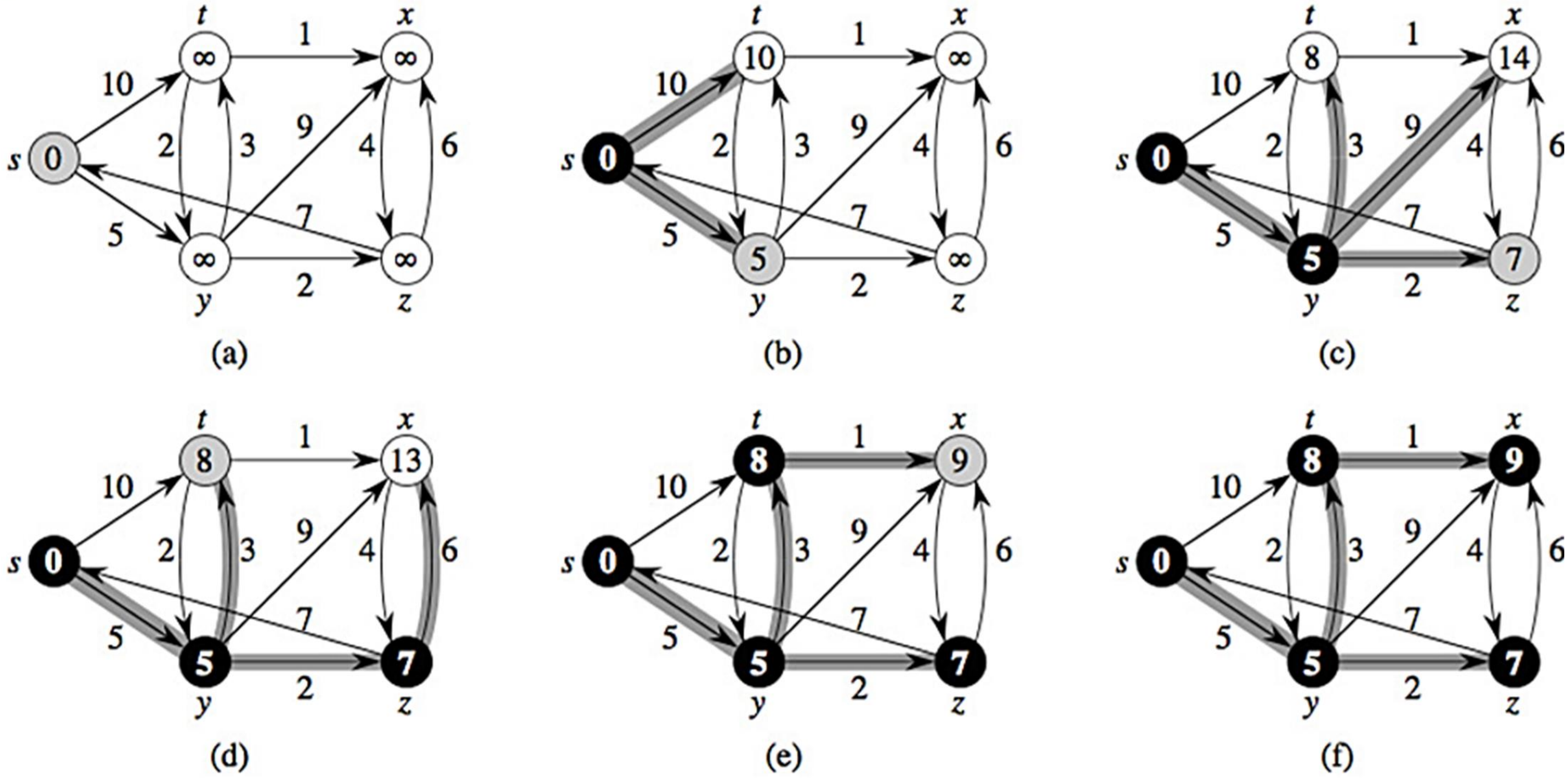
Exemple

Iter	s	t	y	x	z
0	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$
1	0	10	5	$+\infty$	$+\infty$
2	0	8	5	14	7
3	0	8	5	13	7
4	0	8	5	9	7
5	0	8	5	9	7



s	t	y	x	z
-1	y	s	t	y

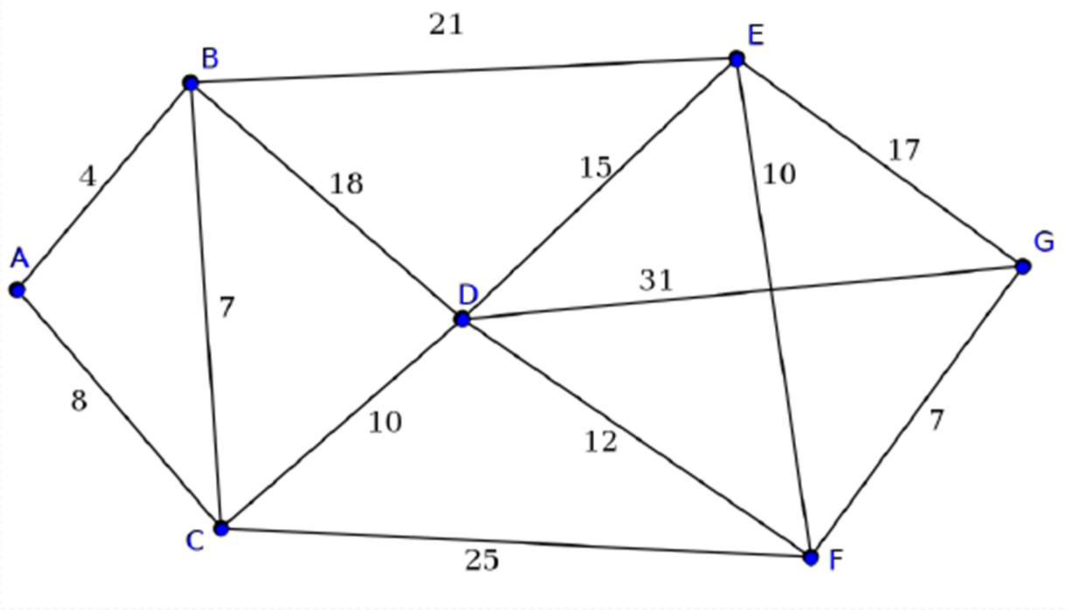
Arborescence des plus courts chemins (π)

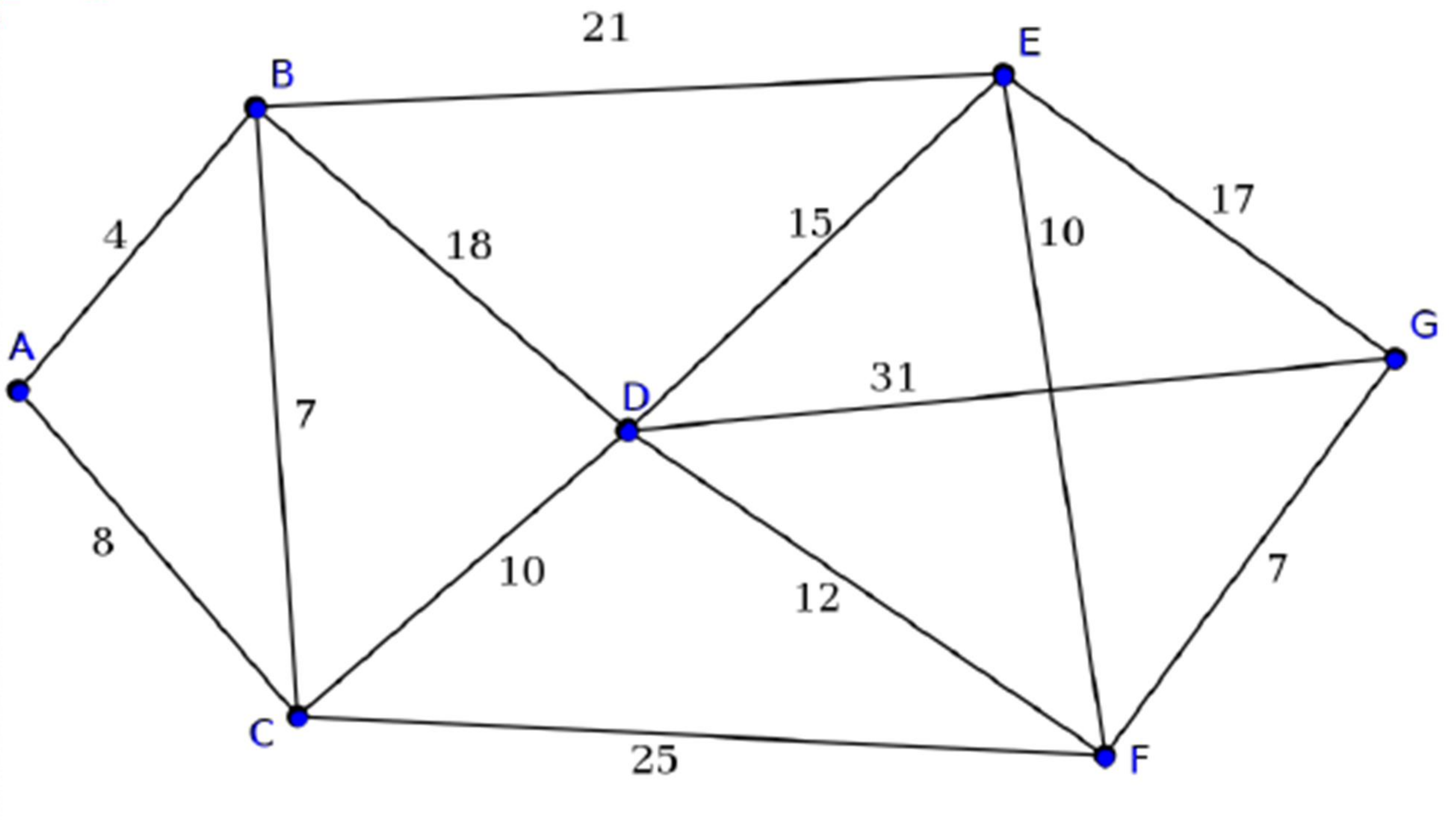


Exécution de l'algorithme de Dijkstra sur le graphe.

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**.



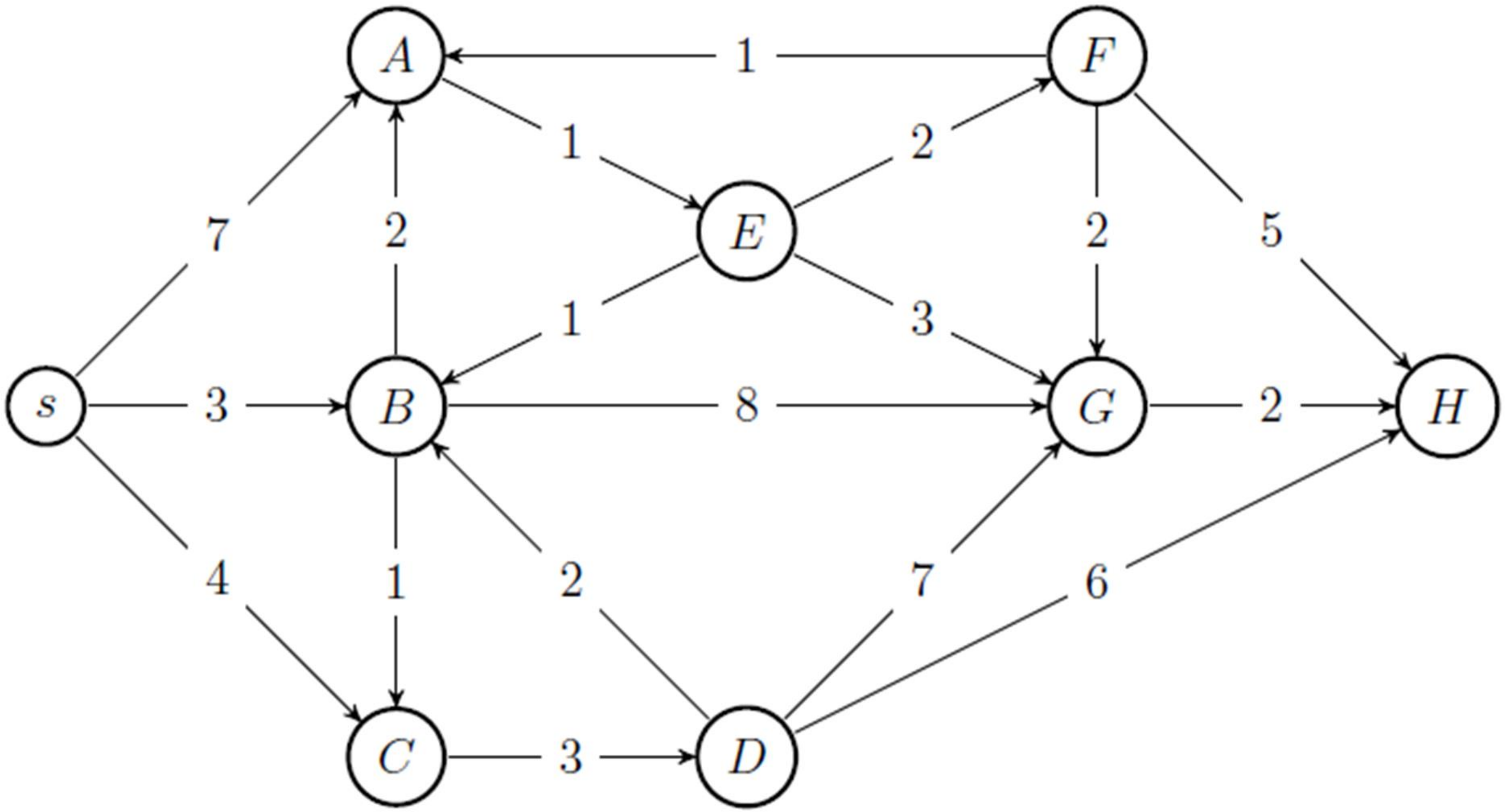


Le plus court chemin en minutes, reliant la gare **B** à la gare **G** est B – C – D – F – G, et la durée est de 36 minutes

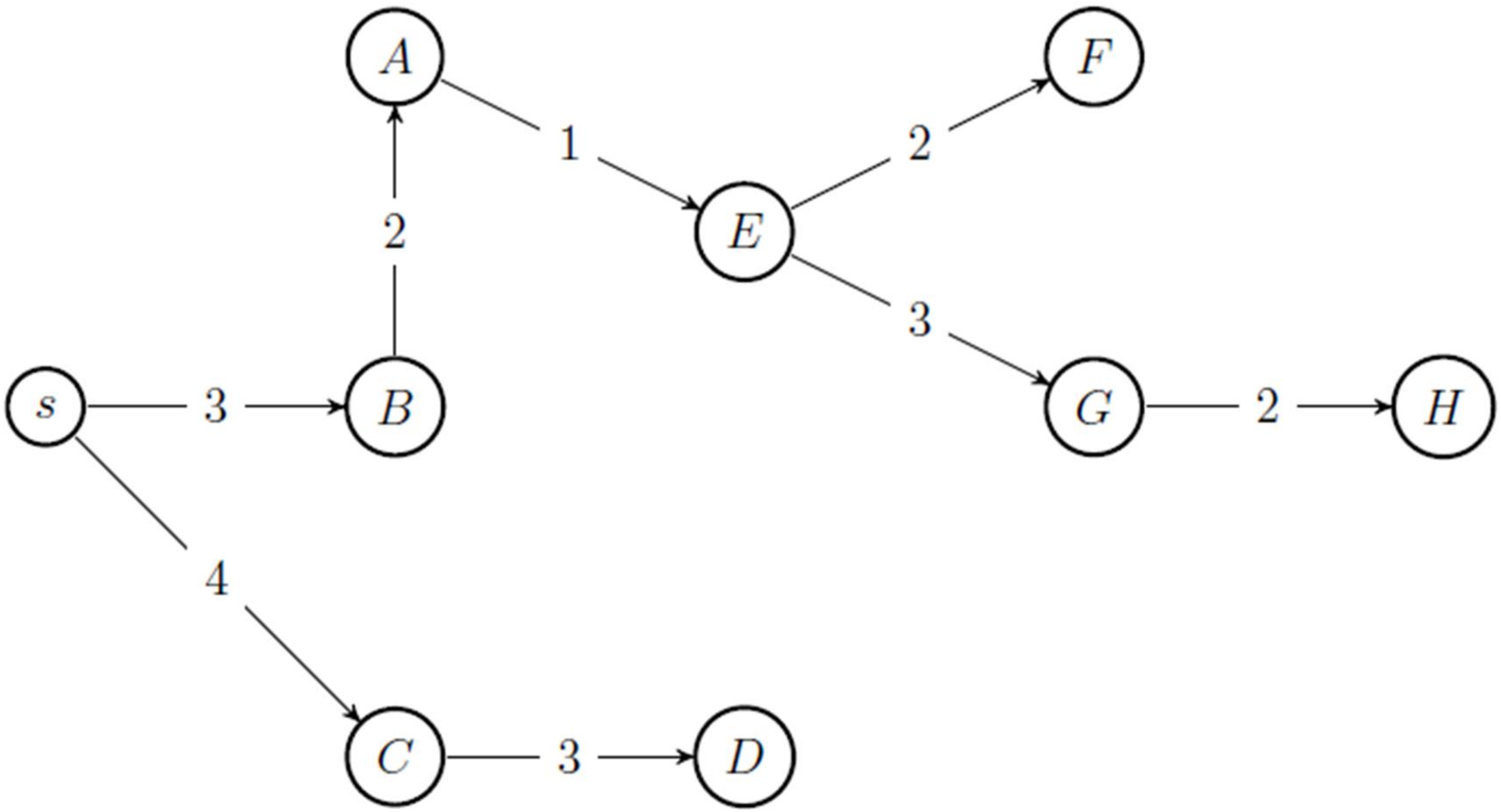
Exercice 2:

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

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
2
E
3
s
3
B
G
2
H
4
C
3
D

Plus courts chemins

Plus courts chemins dans un graphe

- G = (S, A, w) un graphe pondéré.

  La longueur du chemin ($u_1, \ldots , u_k$) 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.

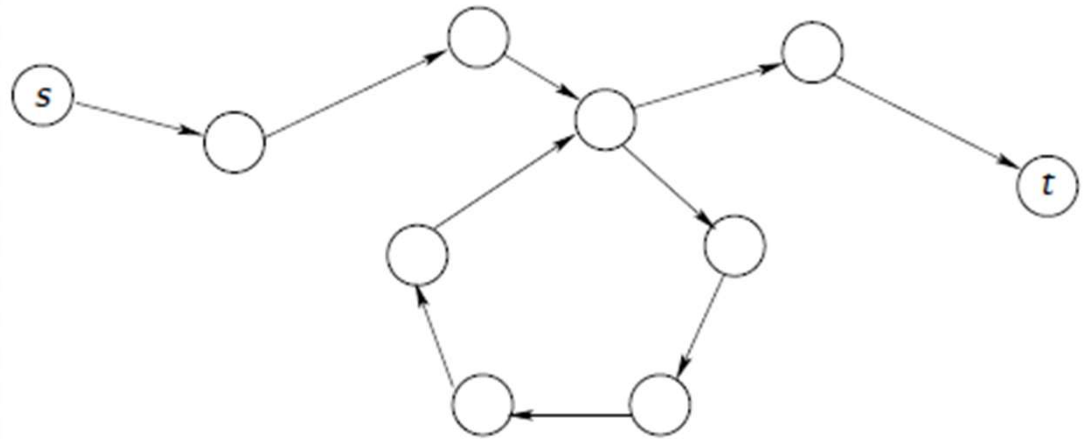
- **Arcs de poids positifs :**
  - **algorithme de Dijkstra** (source unique)

- **Circuits de longueur négative** :
  - pas de plus courts chemins,

- **Arcs de poids quelconques** :
  - **algorithme de Bellman-Ford** (source unique)
  - algorithme de Floyd-Warshall (tous les PCC)

Plus courts chemins élémentaires

Si le graphe G ne contient aucun cycle de longueur négative, alors il existe un plus court chemin élémentaire entre s et t.

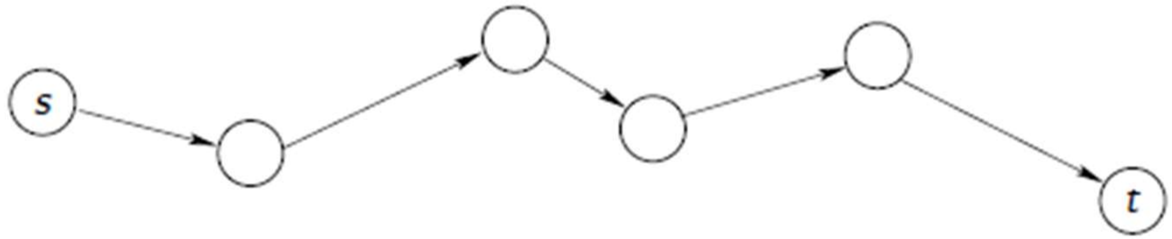


Chemin élémentaire :

chaque sommet apparaît au plus une fois (pas de cycle dans le chemin)

Plus courts chemins élémentaires

Si le graphe G ne contient aucun cycle de longueur négative, alors il existe un plus court chemin élémentaire entre s et t.

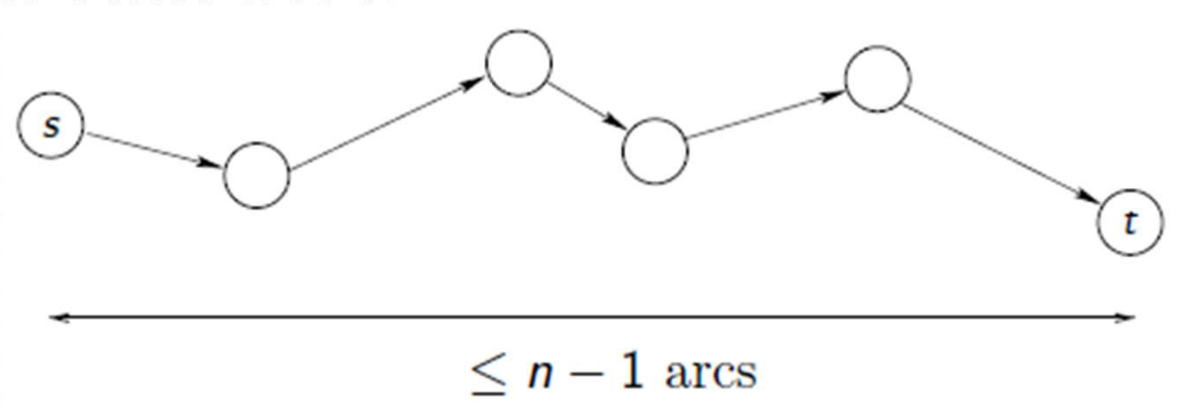


Chemin élémentaire :

chaque sommet apparaît au plus une fois (pas de cycle dans le chemin)

Plus courts chemins élémentaires

Si le graphe G ne contient aucun cycle de longueur négative, alors il existe un plus court chemin élémentaire entre s et t.



Chemin élémentaire :

- au plus n sommets $\Rightarrow$ au plus $n - 1$ arcs

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

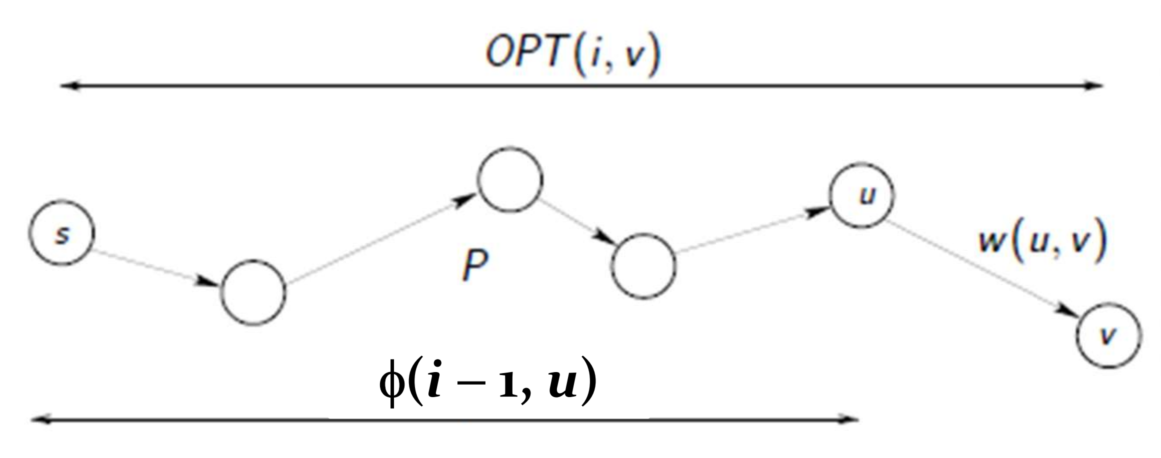
(plus courts chemins issus de s constitués de au plus $n - 1$ arcs : $\phi(n - 1, \mathrm{v}) = \delta(\mathrm{v})$)

- **Algorithme** :

  - initialiser $\phi\ (0, v)$ pour tout v,
  - calculer $\phi\ (1, v)$ 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

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



Définition récursive de $\phi(i, v)$:

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

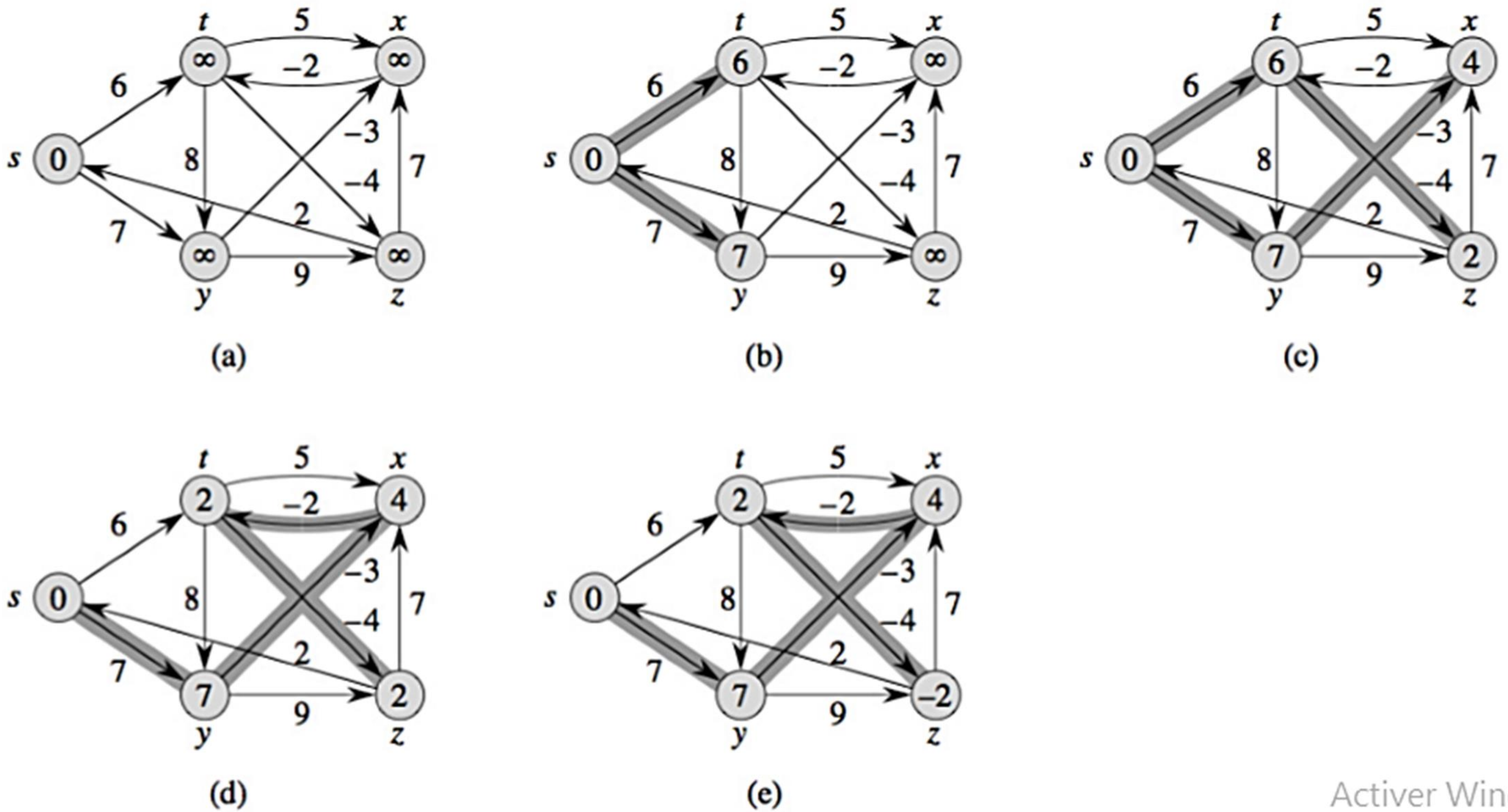
Algorithme de BELLMAN-FORD :

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 :

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

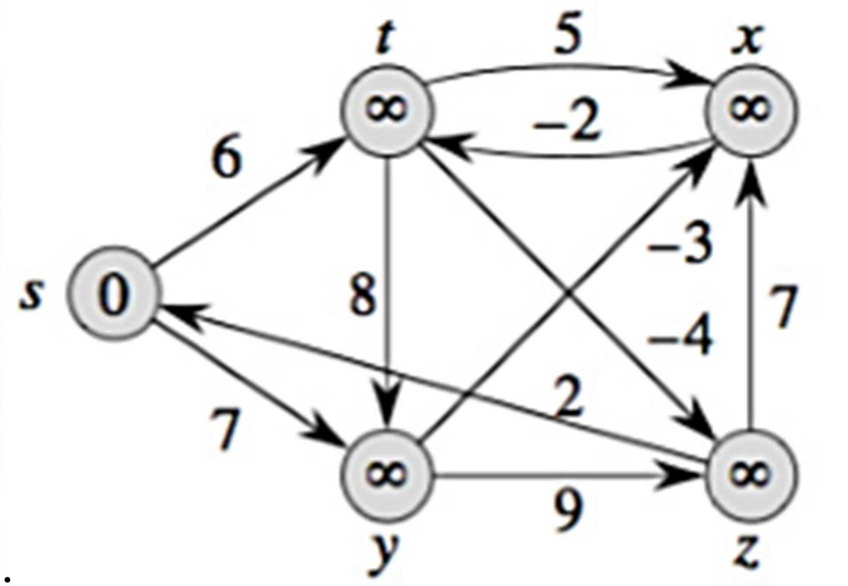


Execution de l 'algorithme de Bellman-Ford avec l 'ordre suivant pour les sommets z,t,x,y,s.

Exemple

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

Itérations de l'algorithme de Bellman-Ford (ϕ).



s	t	y	x	z
-1	x	s	y	t

Arborescence des plus courts chemins (π).

Exemple