Chapitre2

Problèmes d'ACM et de cheminement

1

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

2

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.

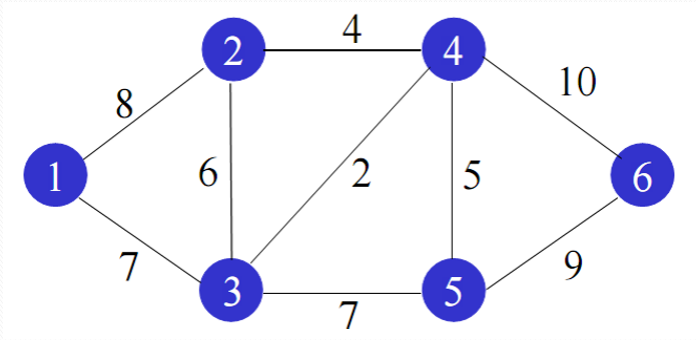
Algorithme de Kruskal

Algorithme

```
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

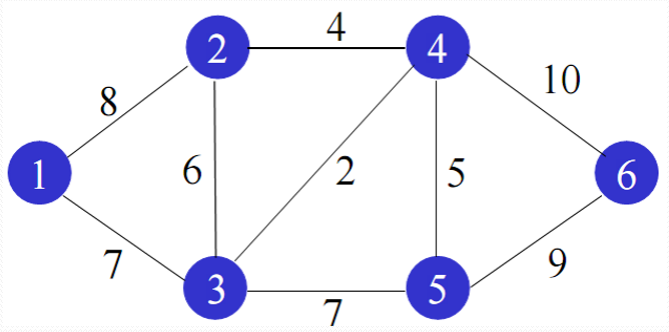
1. Trier les arcs en ordre croissant de poids ;
2. Construire un arbre en sélectionnant les arcs selon l'ordre établi à l'étape 1.



5

Algorithme de Kruskal

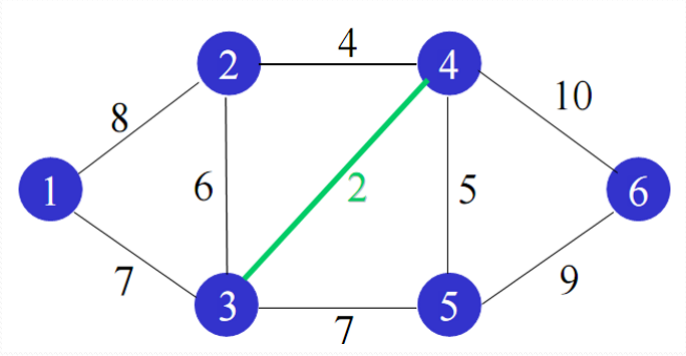
1. Trier les arcs { (3,4), (2,4), (4,5), (2,3), (1,3), (3,5), (1,2), (5,6),(4,6) }
2. Construire un arbre T = { }



6

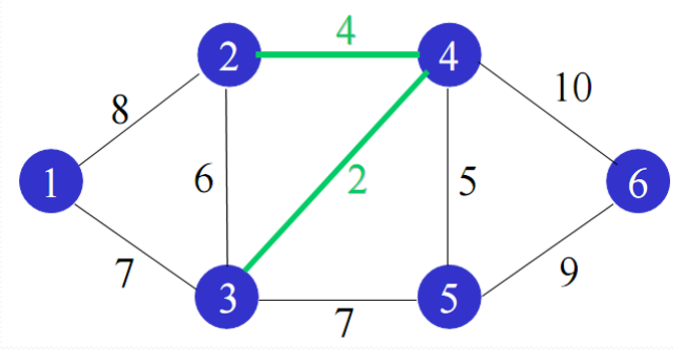
- On évalue (3,4) :

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



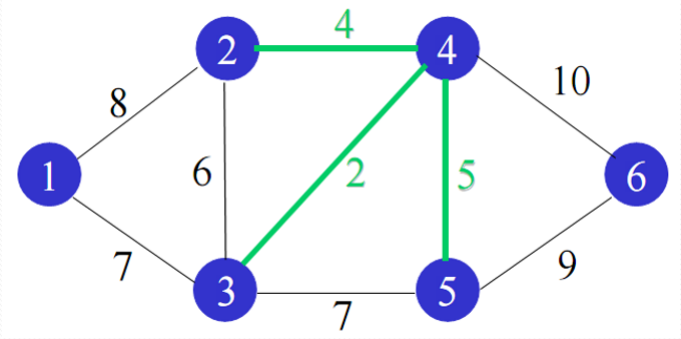
7

- 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)}



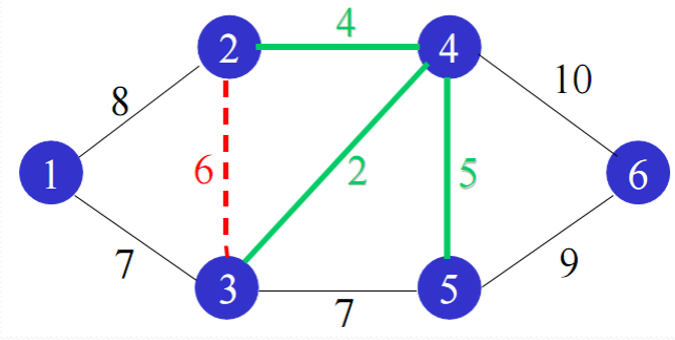
8

- 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) }



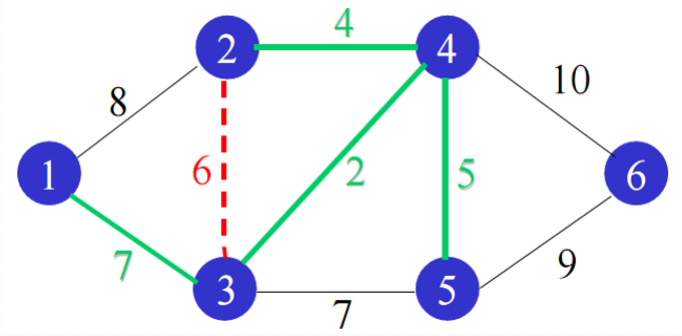
9

- 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) }



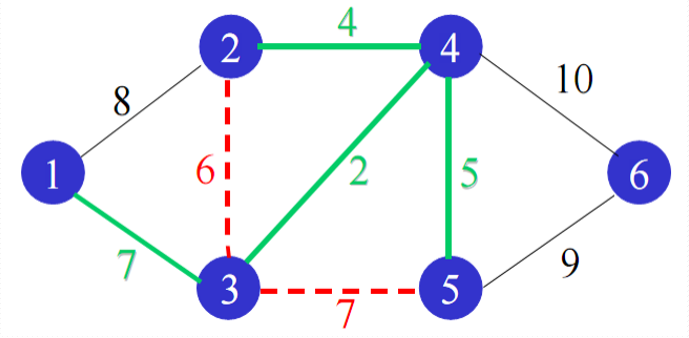
10

- 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) }



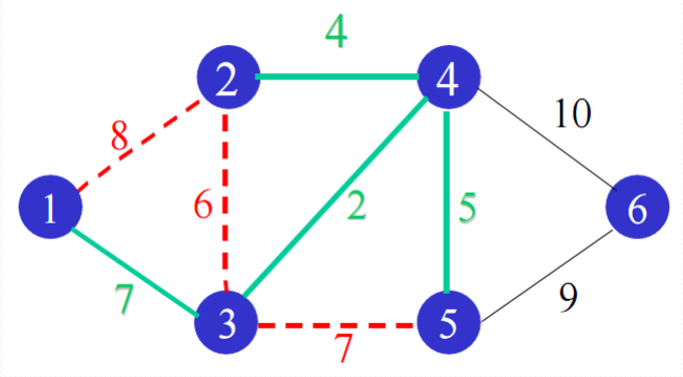
11

- 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) }



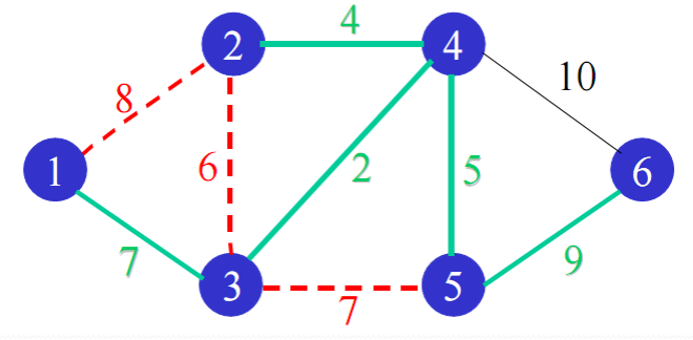
12

- 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) }



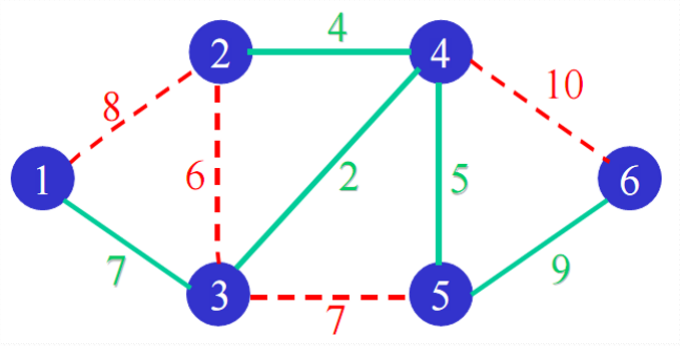
13

- 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) }



14

- 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) }



15

Algorithme de Prim

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

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

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

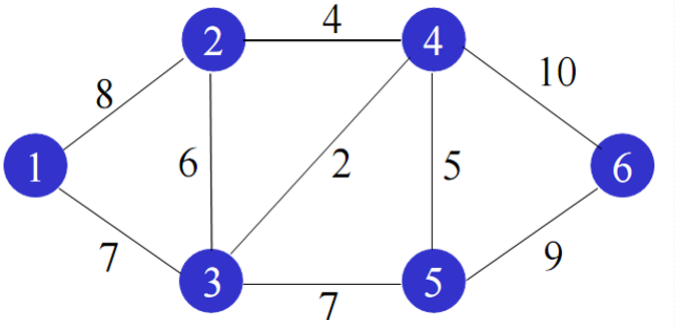
3. Si (i,j) ne forme pas de cycle alors inclure cet arc dans l'arbre, i.e. T = T ∪ (i,j), j ∈ S et W = W \ { j } et aller à l'étape 4; sinon exclure l'arc et retourner à l'étape 2;

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

16

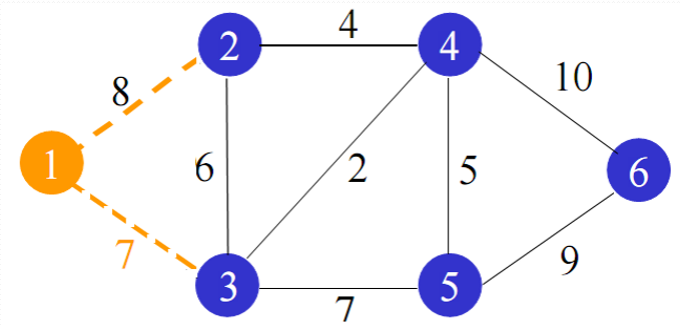
Algorithme de Prim

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



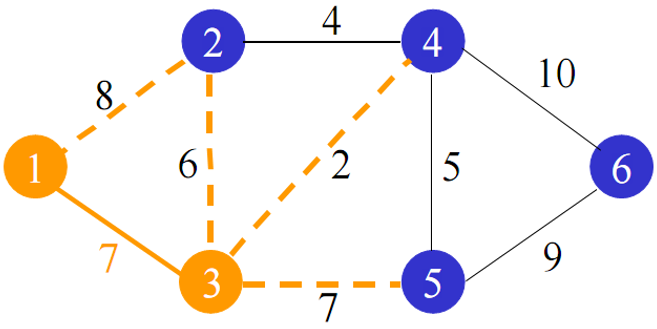
17

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



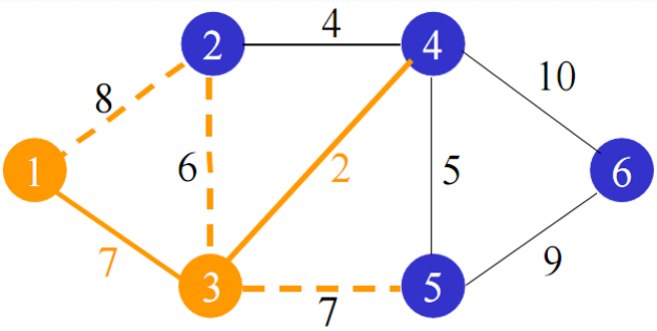
18

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



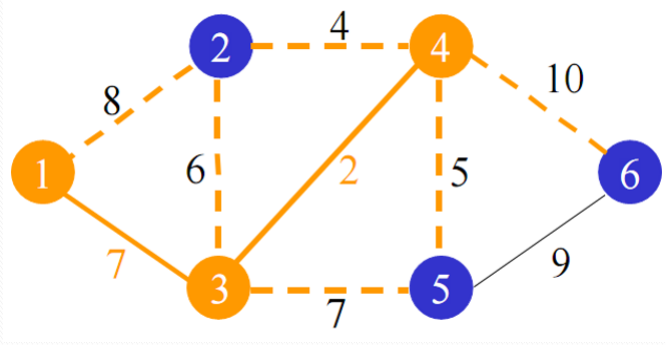
19

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



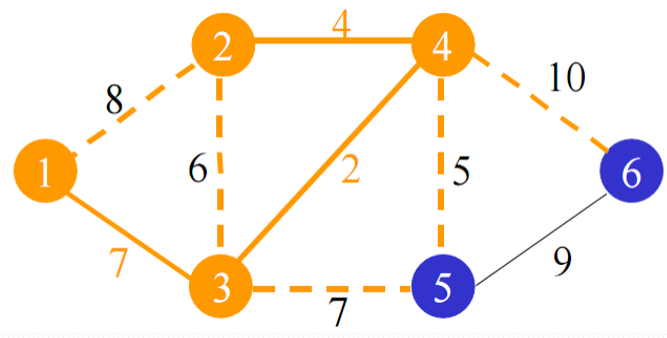
20

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



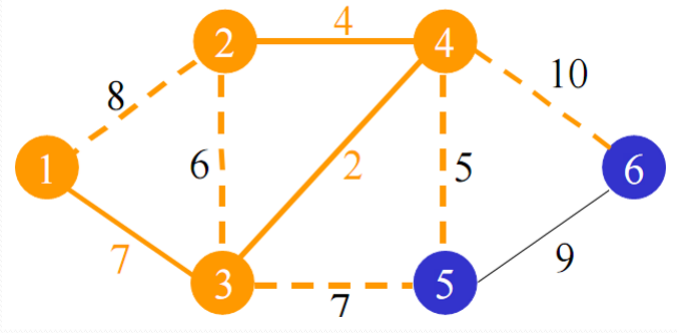
21

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



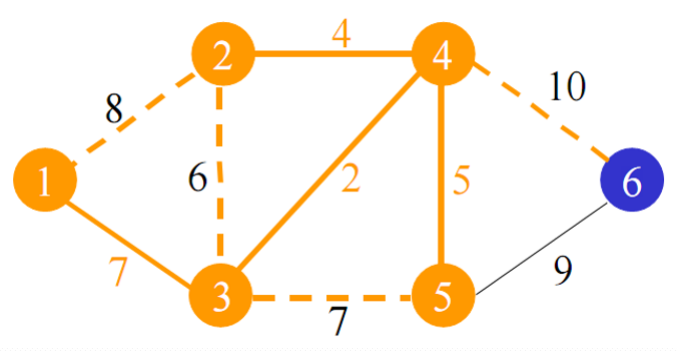
22

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



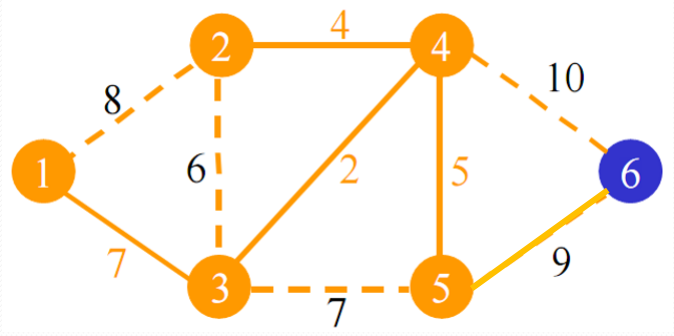
23

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



24

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



25

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

26

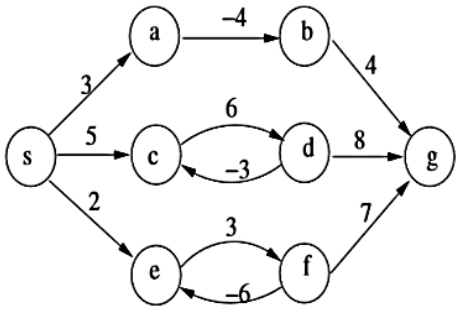
Algorithme de résolution

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

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



28

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.

29

Algorithme de Dijkstra

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

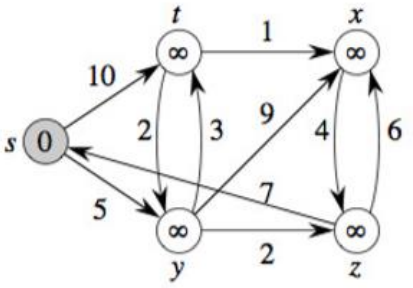
1 - $\forall$ v $\in V$, $\phi[v] = +\infty$ et $\pi[v] = -1$; // Initialisation
2 - $\phi[s] = 0$;
3 - Soit Q un ensemble de sommets initialisé à V ;
4 - **TANT QUE** Q est non vide :
5 - Soit u le sommet de Q avec $\phi[u]$ minimum
6 - Retirer u de Q
7 - **POUR TOUT** $v \in \Gamma^+(u)$: //successeur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

30

Exemple

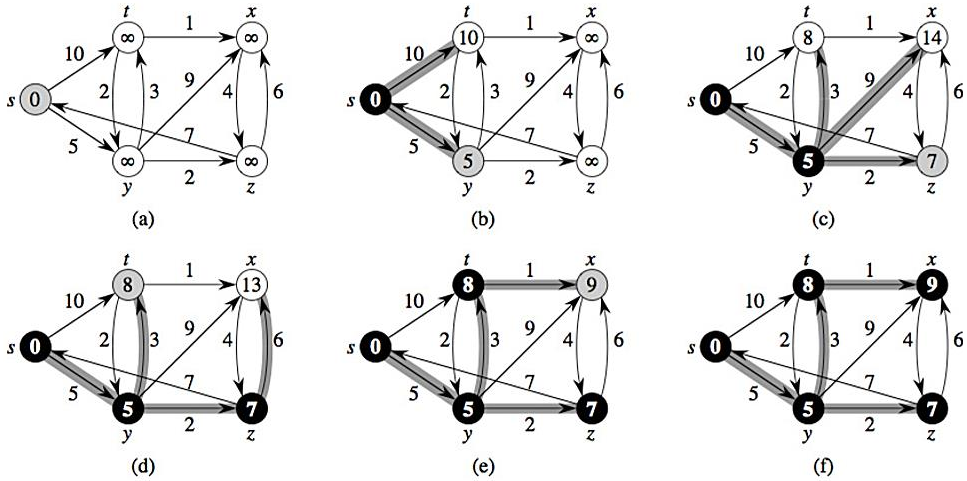
Iter	s	t	y	x	z
0	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$
1	0	10	5	$+\infty$	$+\infty$
2	0	8	5	14	7
3	0	8	5	13	7
4	0	8	5	9	7
5	0	8	5	9	7

s	t	y	x	z
-1	y	s	t	y



Arborescence des plus courts chemins (π)

31



Exécution de l'algorithme de Dijkstra sur le graphe.

32

Algorithme de Bellman-Ford

- L'algorithme de Bellman-Ford permet de trouver les plus courts chemins à origine unique dans le cas où le graphe contient des arcs dont le coût est négatif, sous réserve que le graphe ne contien pas de circuit absorbant (dans ce cas, l'agorithme de Bellman-Ford va détecter l'existance de circuits absorbants).

33

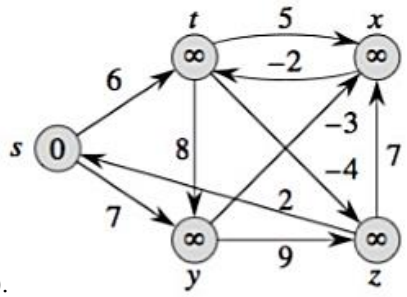
Algorithme de BELLMAN-FORD :
Données : Un graphe $G(V, E)$, une fonction w et un sommet s.
Sortie : Vrai si le graphe ne contient pas de circuit absorbant, faux sinon
Instructions :

```
1 -   Pour i de 1 à |V| − 1 et ∀v ∈ V, φ[i, v] = +∞ et π[i, v] = −1 ; // Initialisation
2 -   φ[0, s] = 0 ;
3 -   POUR i de 1 à |V| − 1 :
4 -       POUR TOUT sommet v de V :
4 -           POUR TOUT u ∈ Γ⁻(v) : // pour tous les prédécesseurs de v
5 -               SI (φ[i, v] > φ[i − 1, u] + w(u, v)) ALORS // Mise à jour
6 -                   φ[i, v] = φ[i − 1, u] + w(u, v) ;
7 -                   π[i, v] = u ;
8 -   POUR TOUT arc uv de E :
9 -           SI (φ[|V| − 1, v] > φ[|V| − 1, u] + w(u, v)) ALORS
10-               RETOURNER Faux ;
11-   RETOURNER Vrai
```

34

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

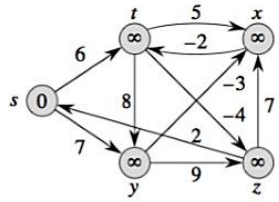


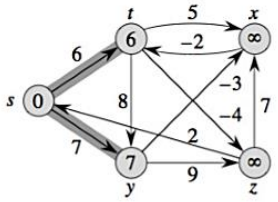
Itérations de l'algorithme de Bellman-Ford (ϕ).

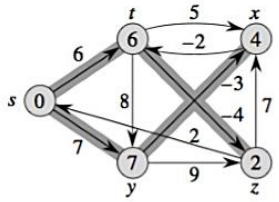
s	t	y	x	z
-1	x	s	y	t

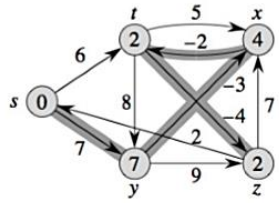
Arborescence des plus courts chemins (π).

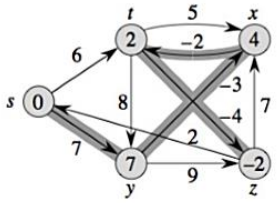
35





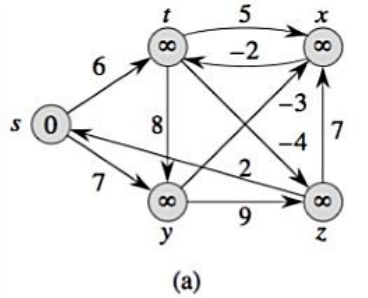






Execution de l 'algorithme de Bellman-Ford avec l 'ordre suivant pour les sommets z,t,x,y,s.

Iter	s	t	y	x	z
0	0	+∞	+∞	+∞	+∞
1	0	4	3	+∞	+∞
2	0	4	3	7	5
3	0	1	3	7	5
4	0	1	2	4	5



Itérations de l'algorithme de Bellman-Ford dans un graphe avec circuit absorbant.

37

Synthèse

- En résumé, en fonction des caractéristiques du problème à résoudre il faudra choisir le bon algorithme :
- Si le graphe ne comporte pas de circuit alors, que l'on recherche un plus court chemin ou un plus long chemin, il suffit de trier les sommets topologiquement avec un parcours en profondeur d'abord, puis de considérer chaque sommet dans l'ordre ainsi défini et relâcher à chaque fois tous les arcs partant de ce sommet;
- Si le graphe comporte des circuits, alors
  - Si la fonction coût est telle que tout sous chemin d'un chemin optimal est également optimal, alors on pourra appliquer Dijkstra;
  - Sinon, on appliquera Bellman-Ford, et on vérifiera en même temps que le graphe ne comporte pas de circuits absorbants;

38