Algorithmique des graphes

4 — Plus court chemin

Marie-Pierre Béal (Cours d'Anthony Labarre)

Université
Gustave Eiffel

Graphes pondérés orientés

Définition 1

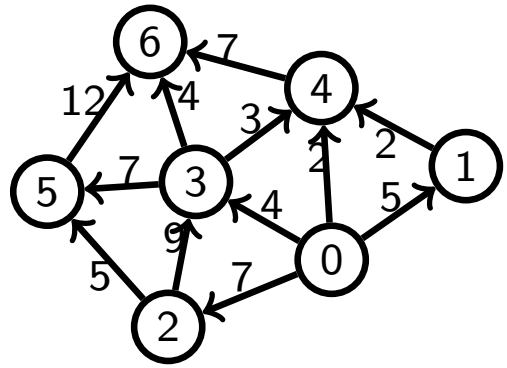
Un **graphe orienté pondéré** est un graphe orienté $G = (V, A, w)$, où $w : A \to \mathbb{R} : (u, v) \mapsto w(u, v)$ est une fonction affectant à chaque arc un poids réel.

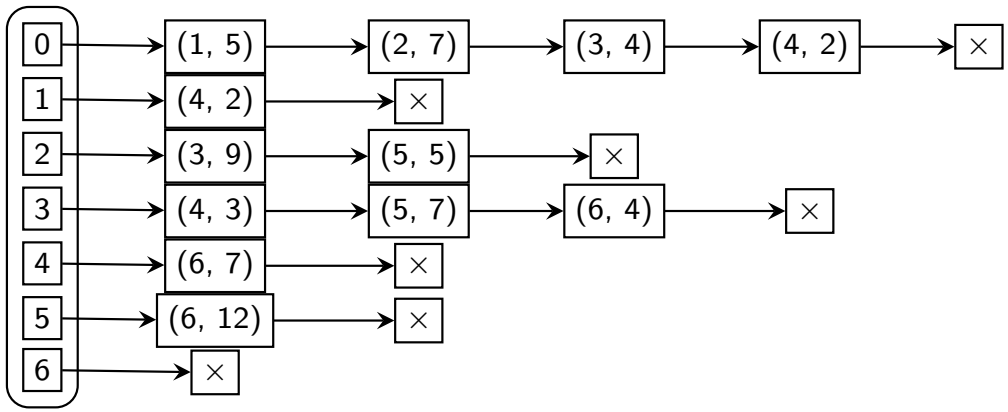
Implémentation des graphes pondérés orientés

- On peut facilement modifier ce qu'on a déjà vu pour implémenter les graphes pondérés :
  - matrice d'adjacence : poids au lieu de booléens ;
  - listes d'adjacence : couples (successeur, poids) au lieu de successeur.

Exemple 1

	0	1	2	3	4	5	6
0		5	7	4	2		
1					2		
2				9		5	
3					3	7	4
4							7
5							12
6							





Implémentation des graphes pondérés

- On suppose l'existence d'une classe `GrapheOrientéPondéré` très similaire à la classe `GrapheOrienté`, avec quelques modifications :
  - ajouter_arc(u, v, poids) ;
  - ajouter_arcs(séquence) ;
  - arc() ;
  - boucles() ;
  - sous_graphe_induit(séquence).

Implémentation des graphes pondérés

- On suppose l'existence d'une classe `GrapheOrientéPondéré` très similaire à la classe `GrapheOrienté`, avec quelques modifications :
  - ajouter_arc(u, v, poids) ;
  - ajouter_arcs(séquence) ;
  - arc() ;
  - boucles() ;
  - sous_graphe_induit(séquence).
- ... et quelques ajouts ou modifications :
  - successeur(sommet) ;
  - poids_arc(u, v).

Plus courts chemins dans un graphe

- Les graphes orientés pondérés modélisent fréquemment des réseaux routiers.
- Comment faire pour trouver un chemin de moindre coût (ou poids) allant d'un sommet à un autre ?
- Le **poids** d'un chemin est la somme des poids de ses arcs.
- Si le graphe n'était pas pondéré, comment calculerait-on un plus court chemin allant d'un sommet à un autre ?

L'algorithme de Dijkstra

- L'algorithme de Dijkstra construit un **arbre des plus courts chemins au départ d'un sommet** :

L'algorithme de Dijkstra

- L'algorithme de Dijkstra construit un **arbre des plus courts chemins au départ d'un sommet** :
  - la racine de cet arbre est le sommet de départ (la *source*) ;

L'algorithme de Dijkstra

- L'algorithme de Dijkstra construit un **arbre des plus courts chemins au départ d'un sommet** :
  - la racine de cet arbre est le sommet de départ (la *source*) ;
  - l'arbre ne contient qu'un chemin de poids minimum entre la source et chaque sommet du graphe.

L'algorithme de Dijkstra

- L'algorithme de Dijkstra construit un **arbre des plus courts chemins au départ d'un sommet** :
  - la racine de cet arbre est le sommet de départ (la *source*) ;
  - l'arbre ne contient qu'un chemin de poids minimum entre la source et chaque sommet du graphe.
- Hypothèses : **le graphe est sans arc de poids négatif**.

Description de l'algorithme de Dijkstra

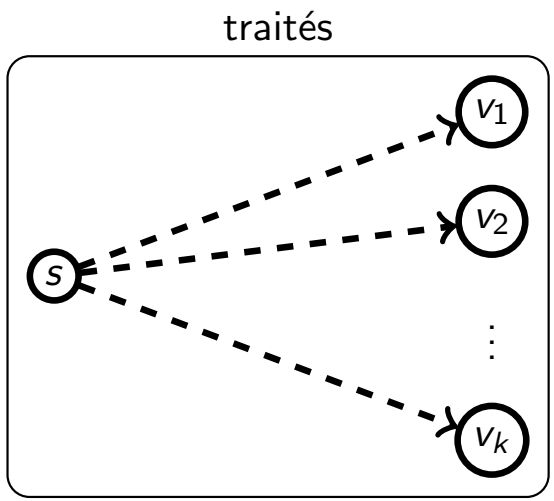
- L'algorithme maintient en permanence un ensemble S de sommets déjà traités ;

Description de l'algorithme de Dijkstra

- L'algorithme maintient en permanence un ensemble S de sommets déjà traités ;
- À chaque étape de l'algorithme, on examine le sommet non traité le plus proche, et on l'utilise pour découvrir de nouveaux raccourcis ;

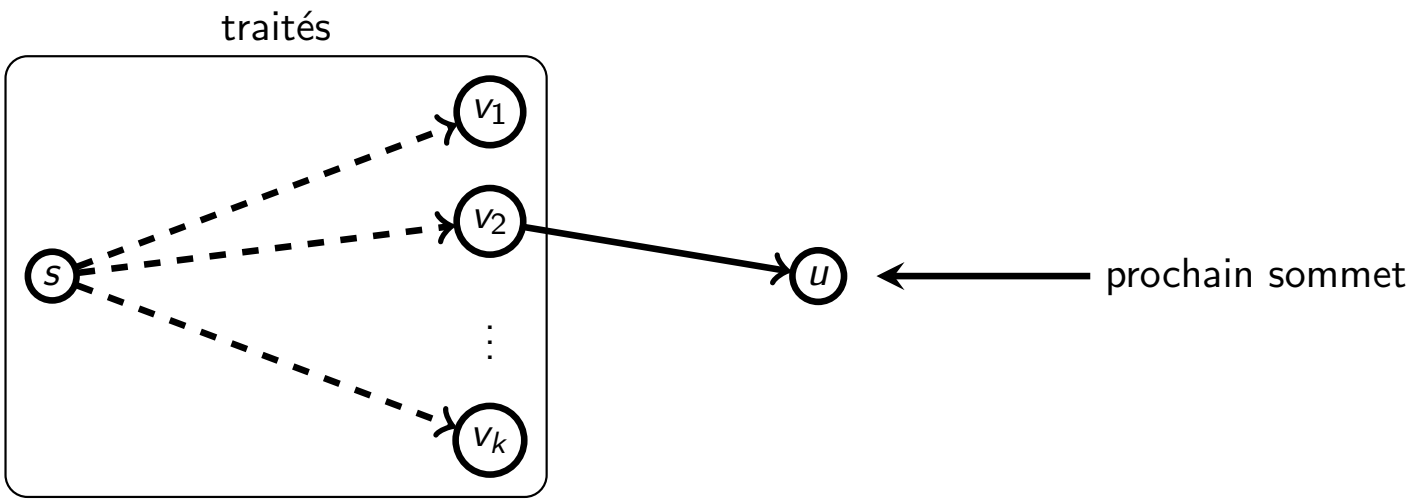
Description de l'algorithme de Dijkstra

- L'algorithme maintient en permanence un ensemble S de sommets déjà traités ;
- À chaque étape de l'algorithme, on examine le sommet non traité le plus proche, et on l'utilise pour découvrir de nouveaux raccourcis ;



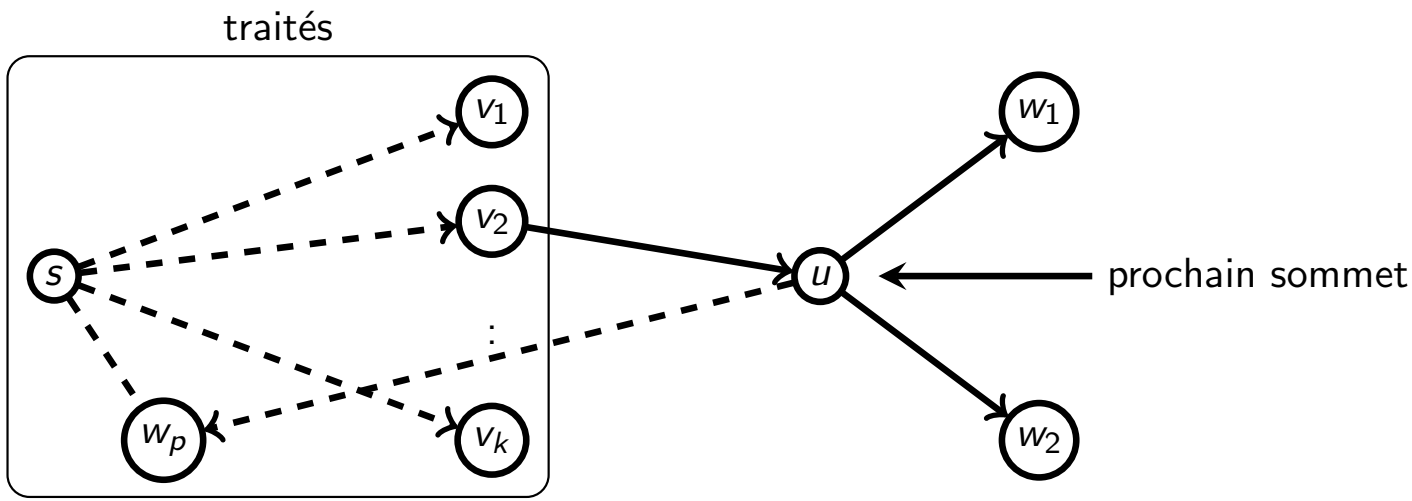
Description de l'algorithme de Dijkstra

- L'algorithme maintient en permanence un ensemble S de sommets déjà traités ;
- À chaque étape de l'algorithme, on examine le sommet non traité le plus proche, et on l'utilise pour découvrir de nouveaux raccourcis ;



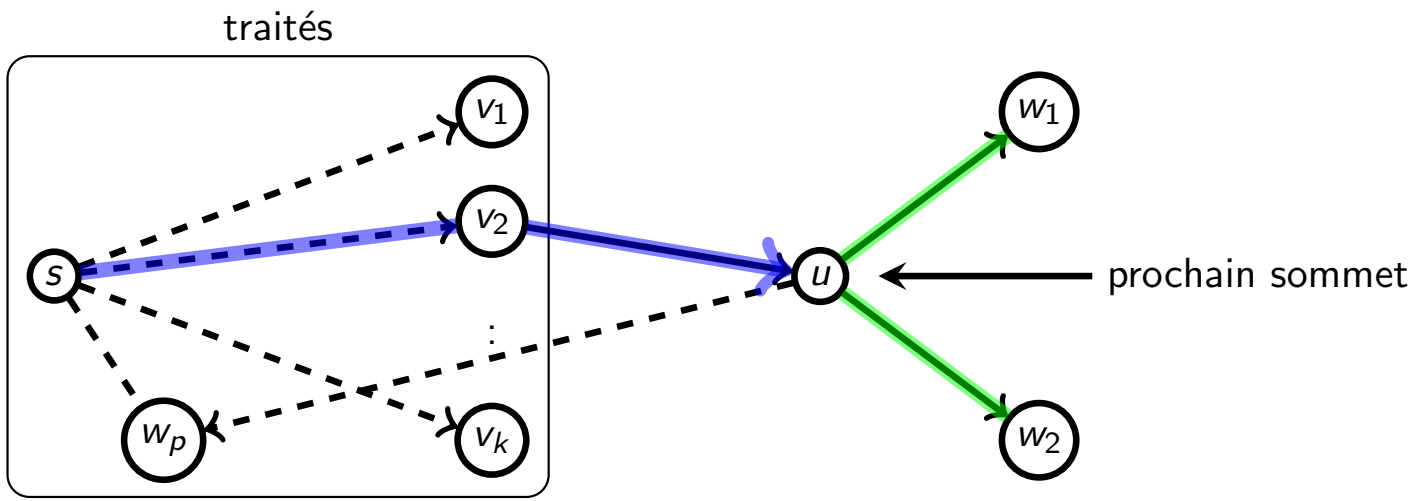
Description de l'algorithme de Dijkstra

- L'algorithme maintient en permanence un ensemble S de sommets déjà traités ;
- À chaque étape de l'algorithme, on examine le sommet non traité le plus proche, et on l'utilise pour découvrir de nouveaux raccourcis ;



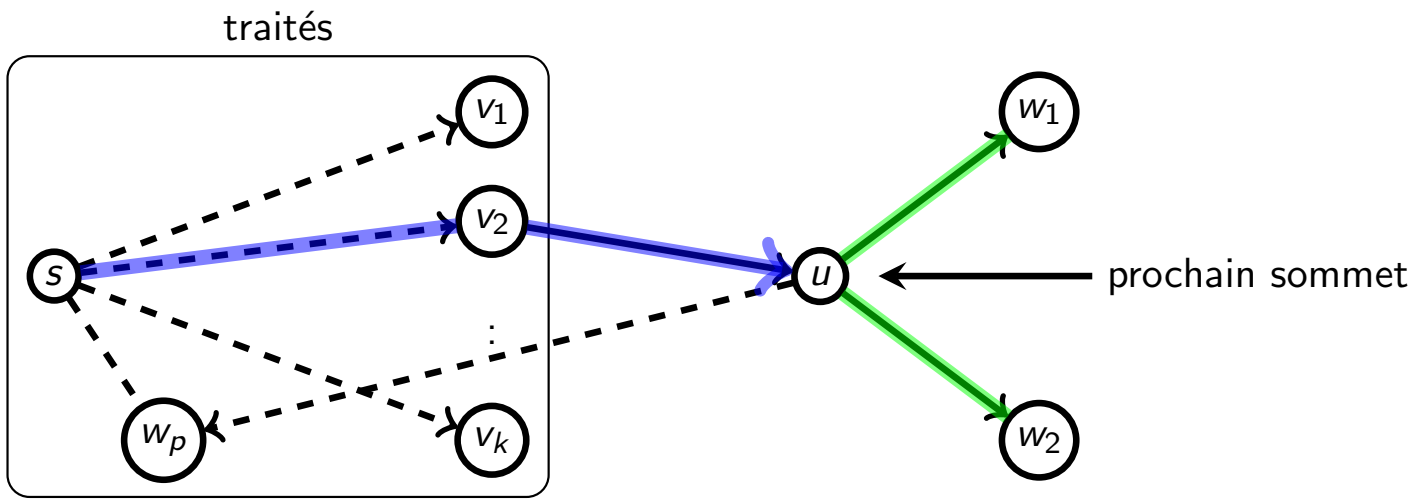
Description de l'algorithme de Dijkstra

- L'algorithme maintient en permanence un ensemble S de sommets déjà traités ;
- À chaque étape de l'algorithme, on examine le sommet non traité le plus proche, et on l'utilise pour découvrir de nouveaux raccourcis ;



Description de l'algorithme de Dijkstra

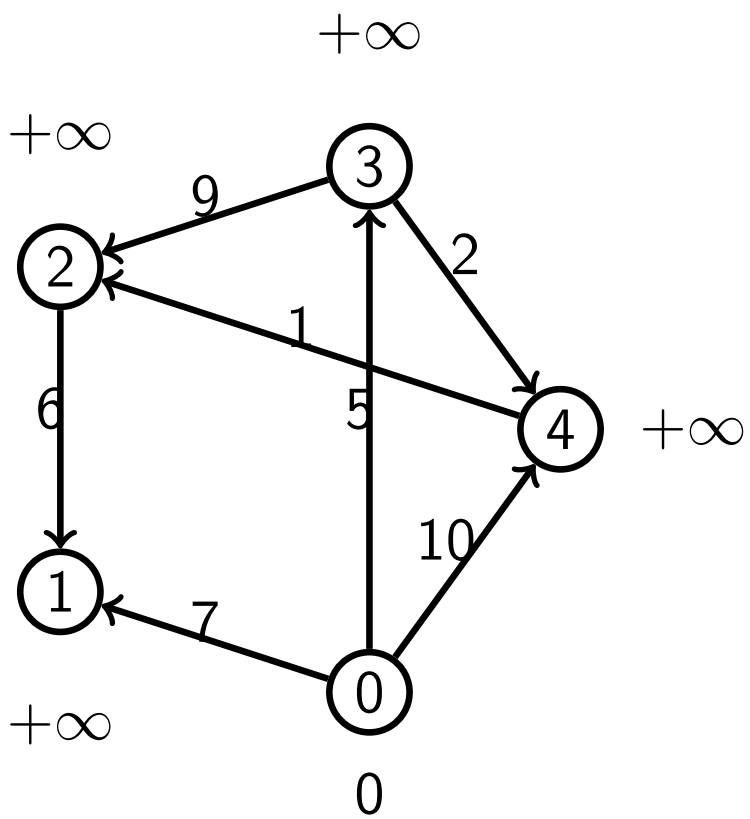
- L'algorithme maintient en permanence un ensemble S de sommets déjà traités ;
- À chaque étape de l'algorithme, on examine le sommet non traité le plus proche, et on l'utilise pour découvrir de nouveaux raccourcis ;



- **Attention :** on ne traite chaque sommet qu'une fois et la distance d'un sommet déjà traité ne change plus jamais.

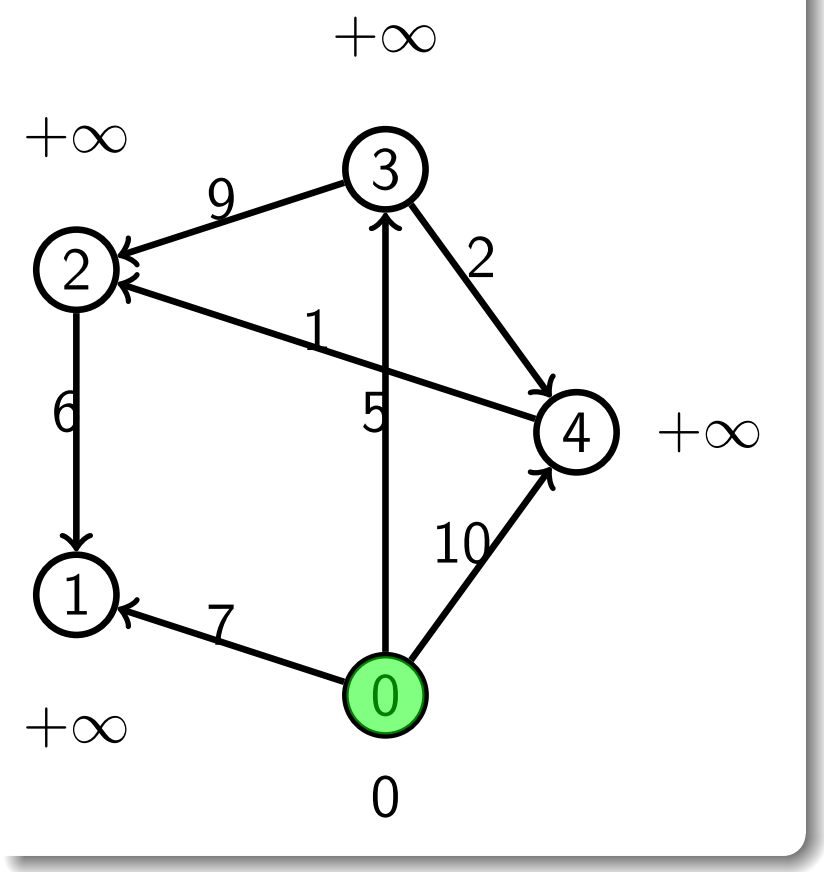
Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



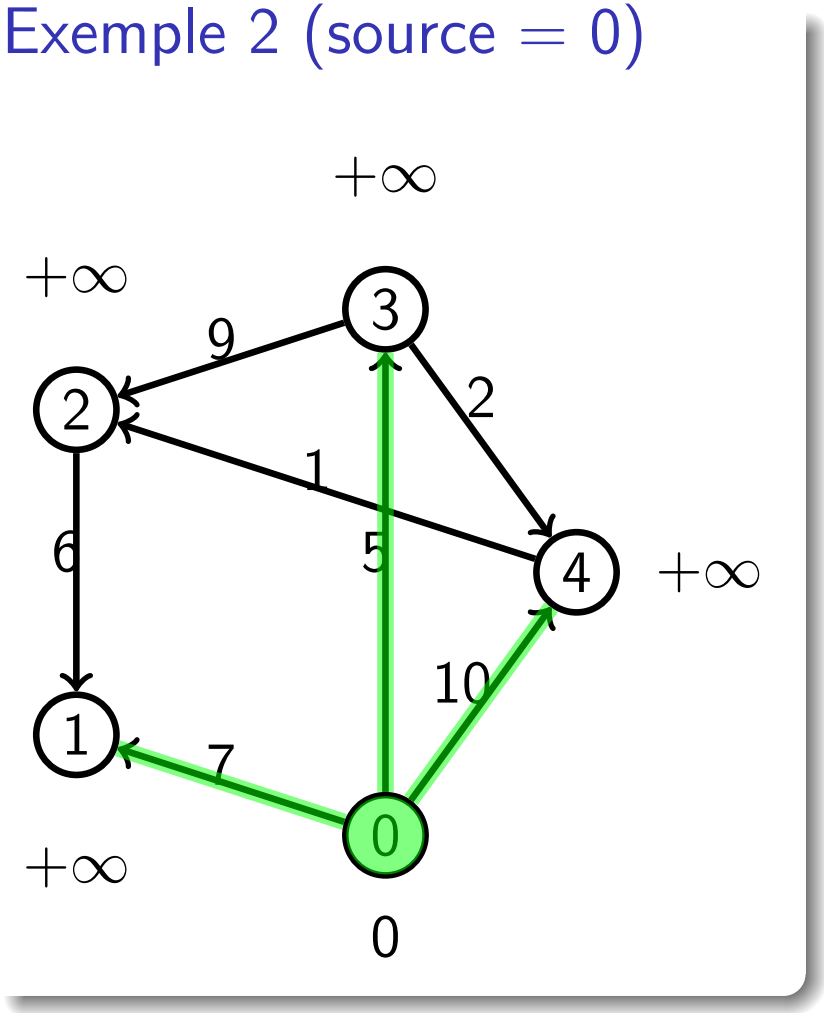
Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



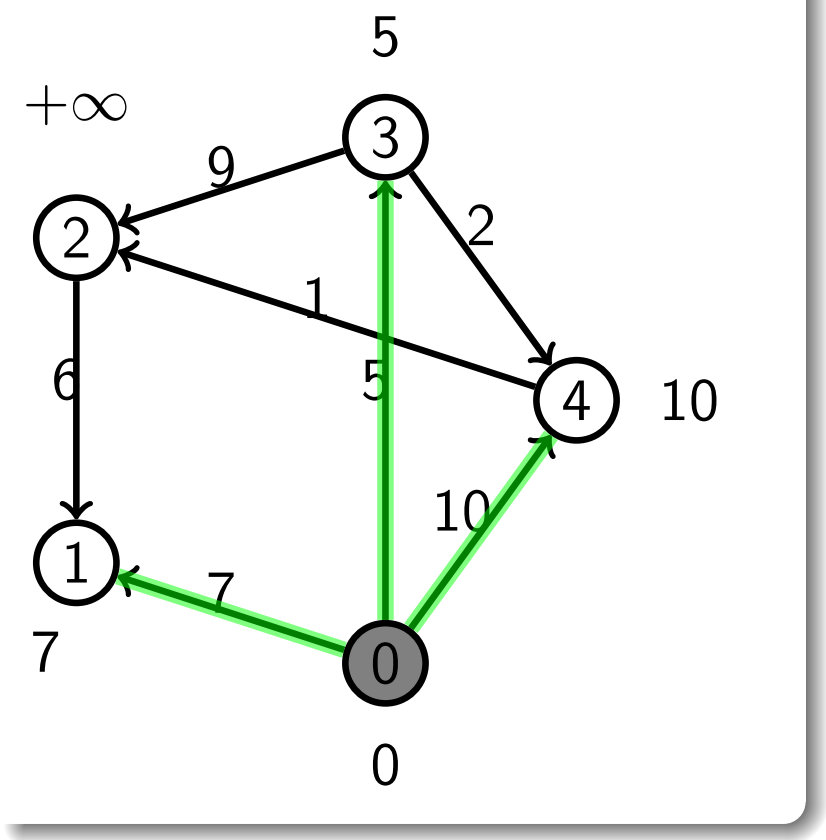
Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



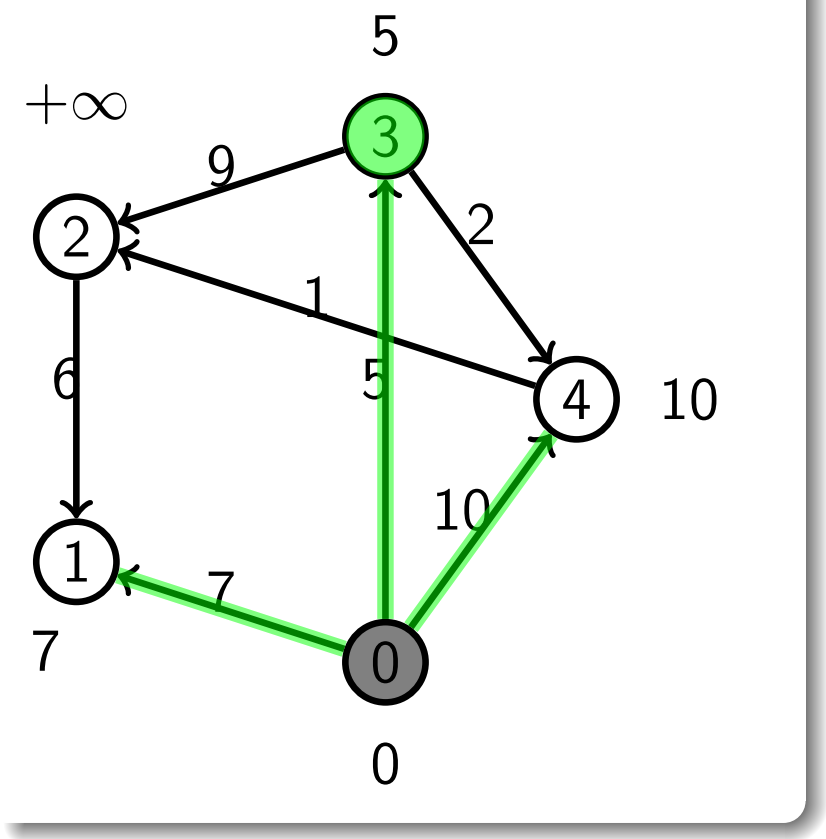
Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



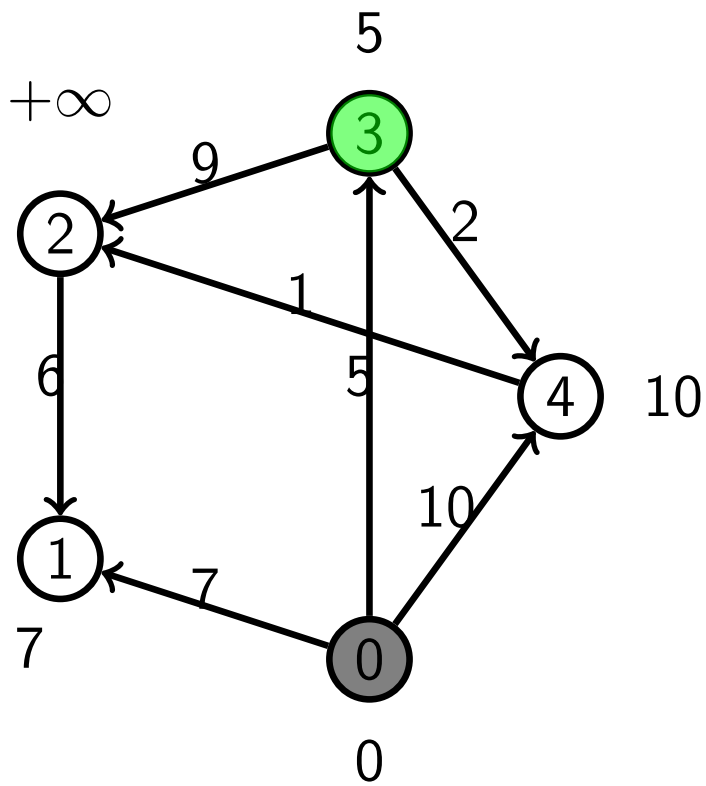
Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



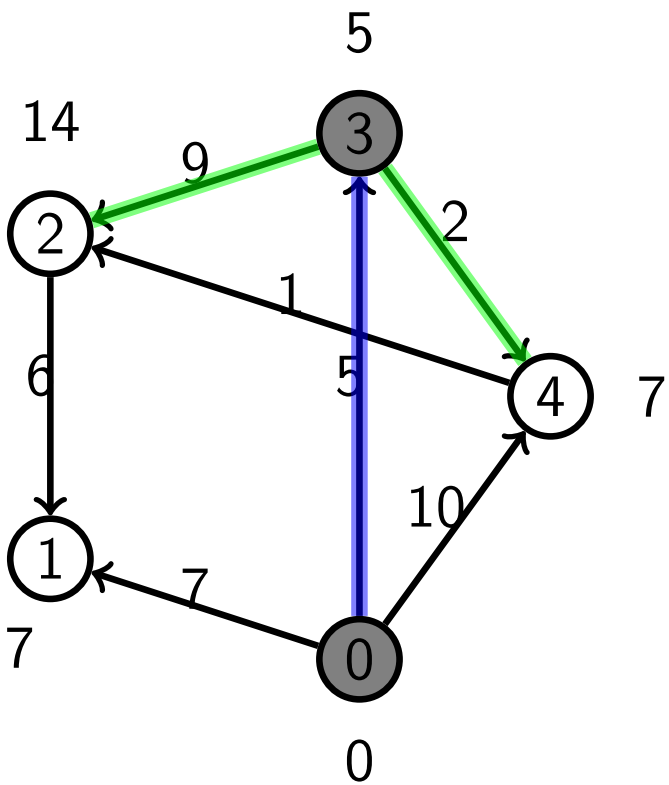
Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



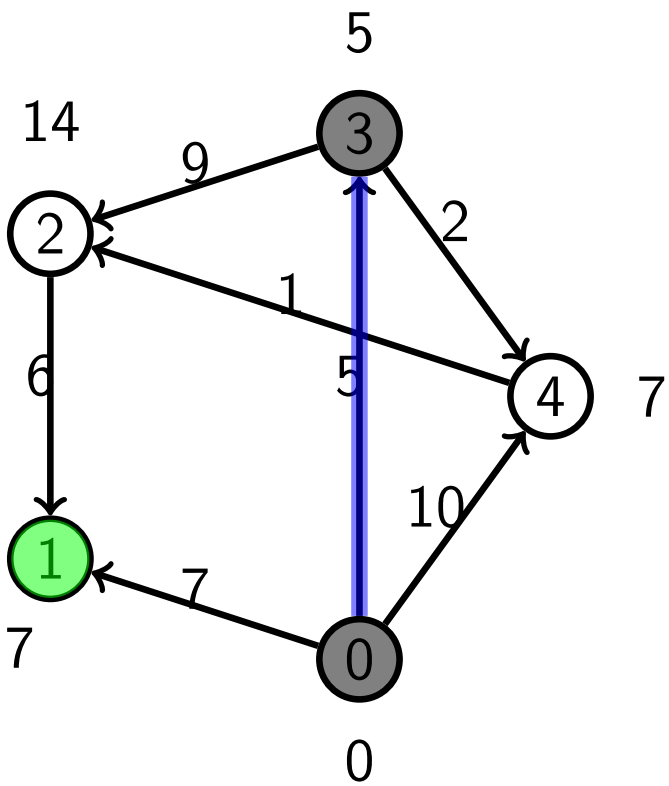
Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



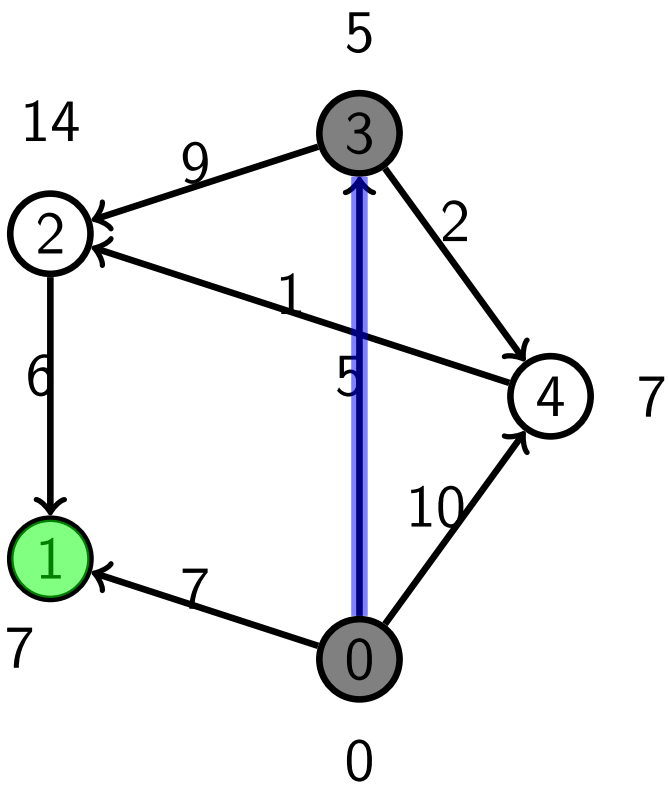
Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



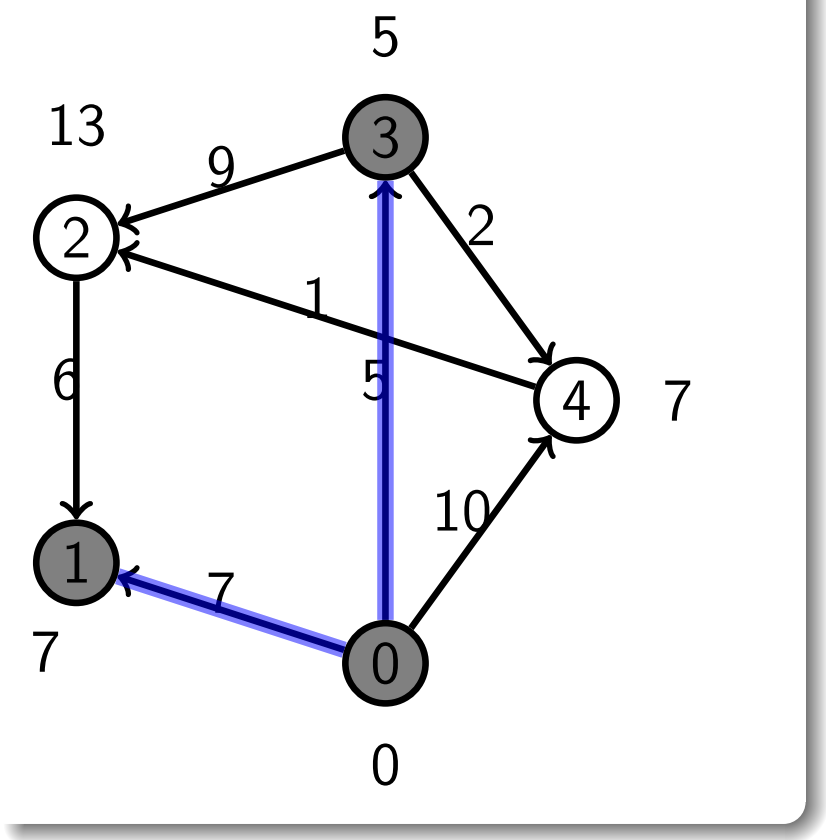
Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



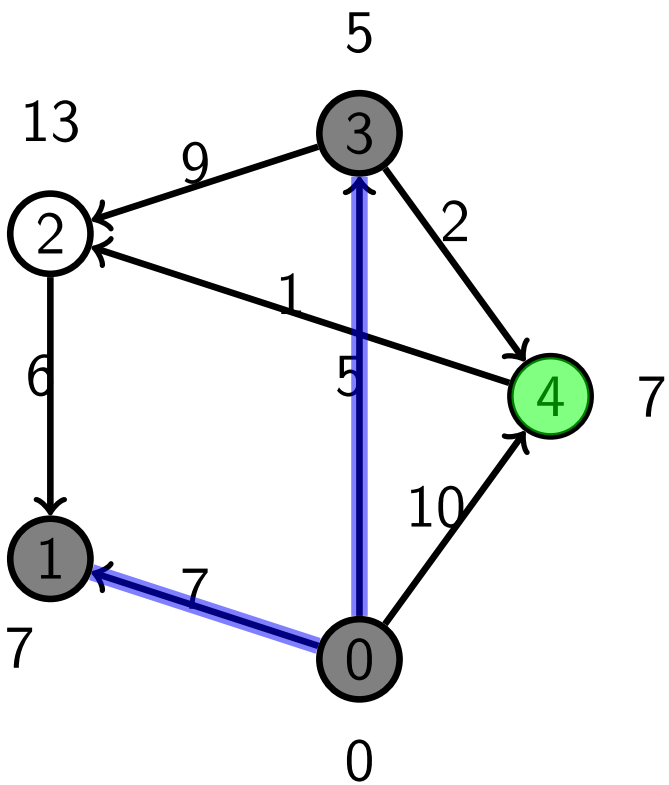
Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



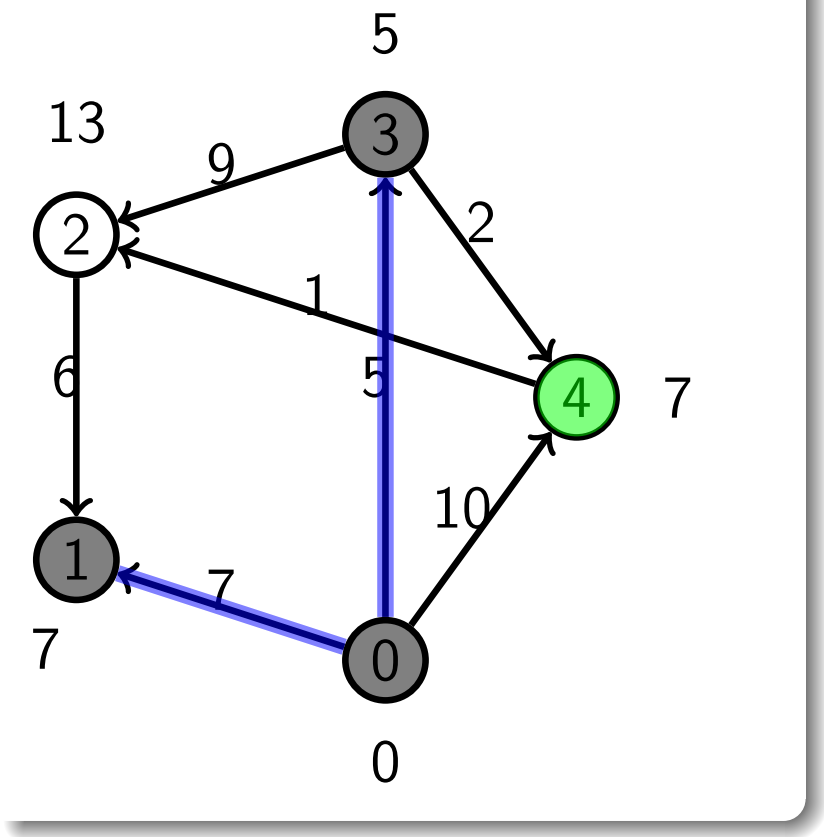
Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



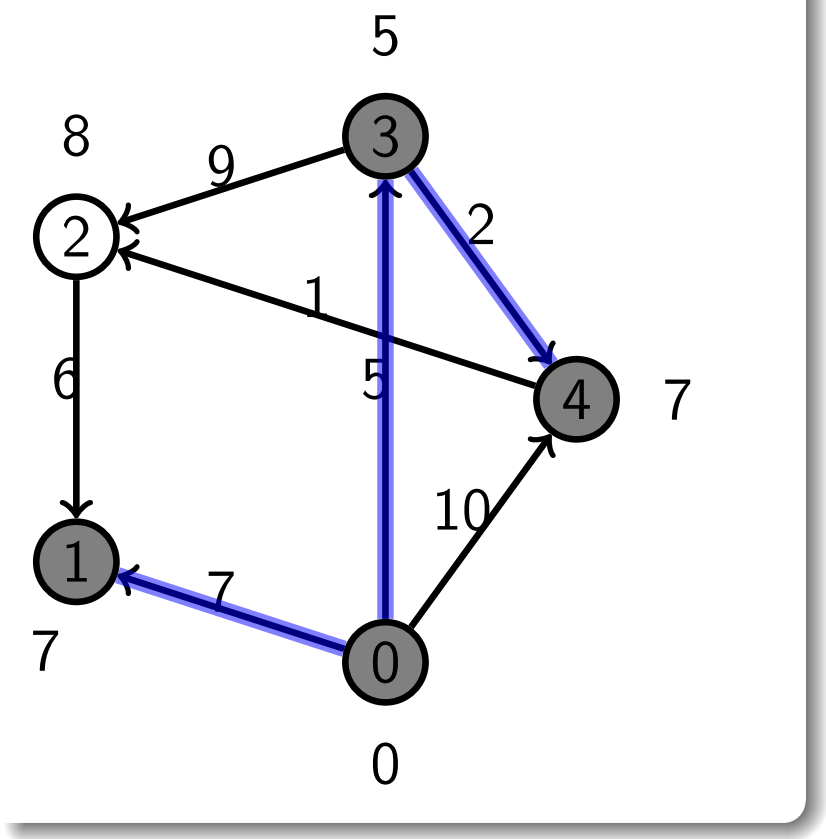
Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



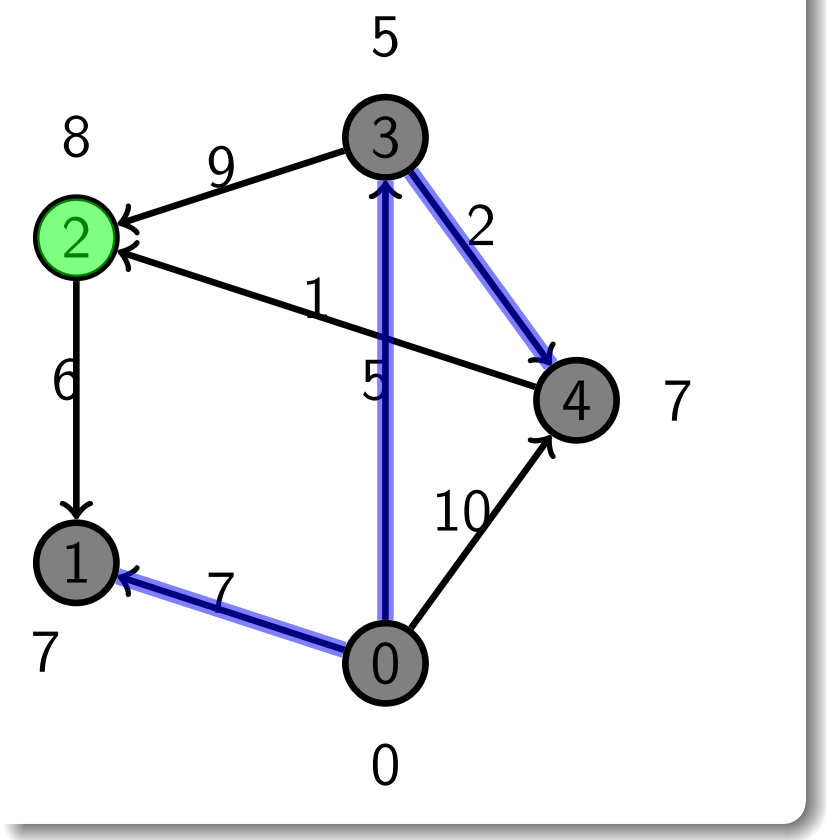
Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



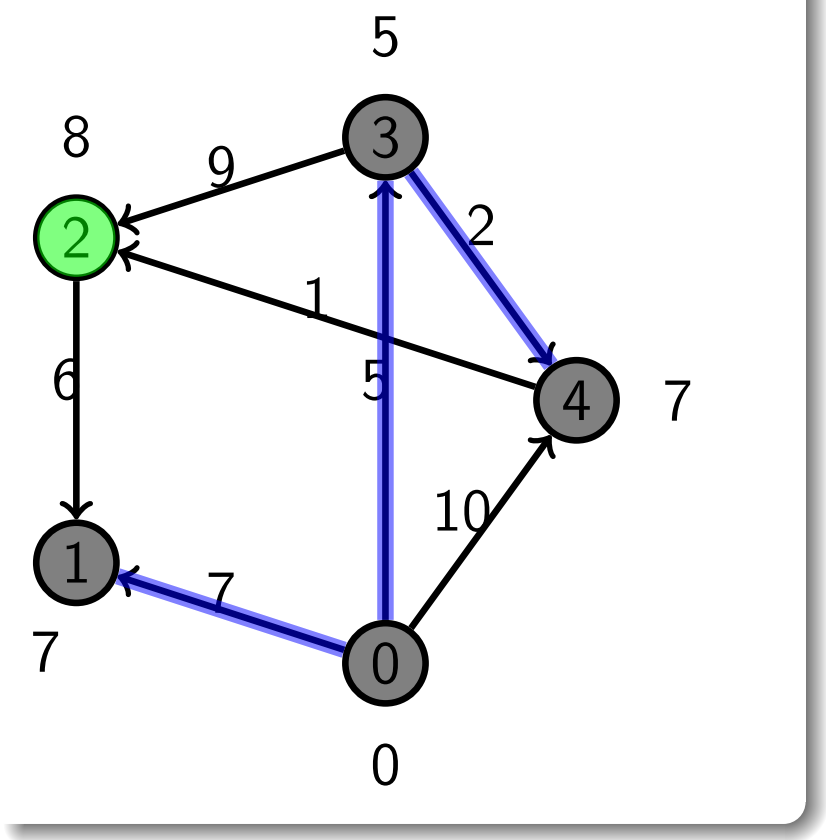
Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



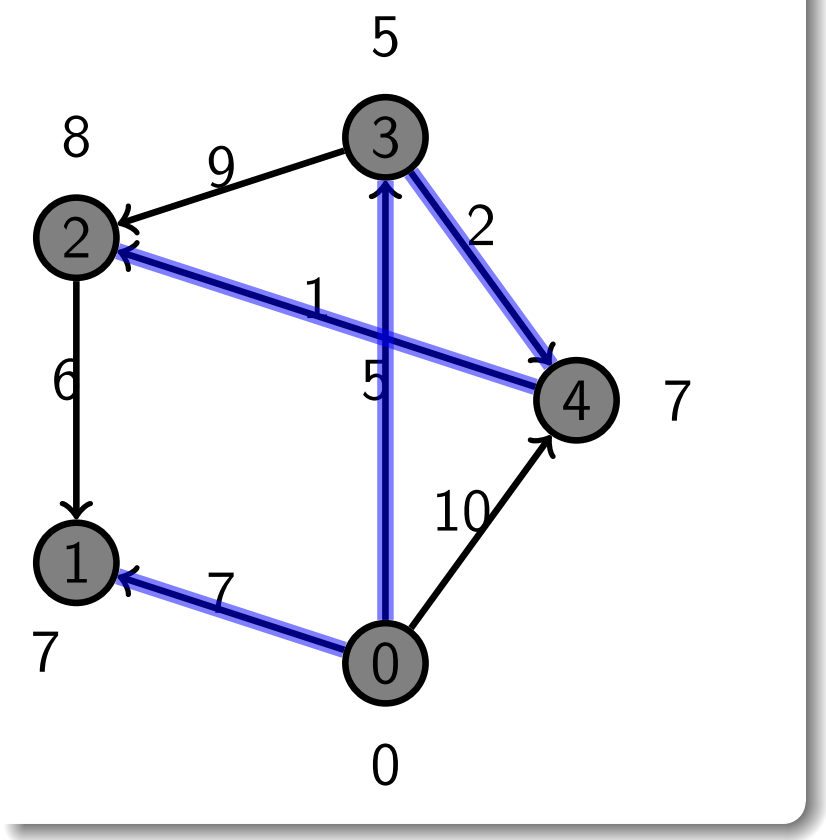
Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



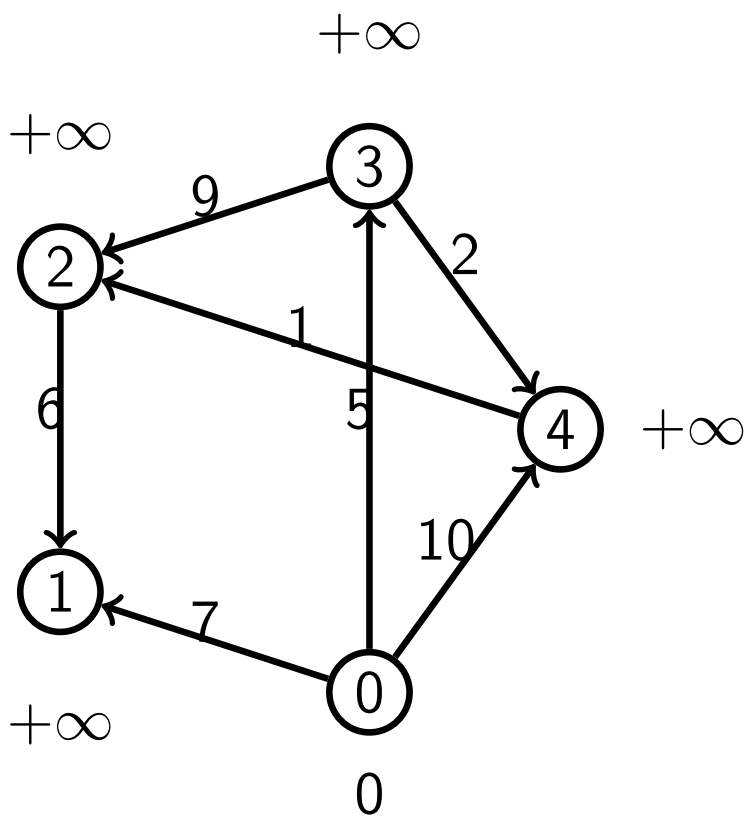
Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



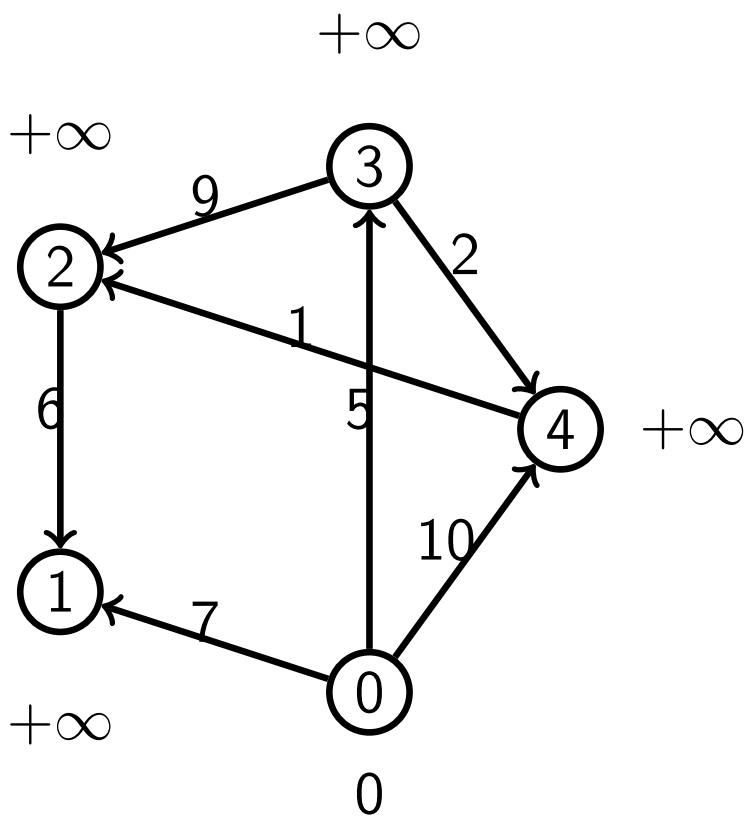
Les coulisses

distances :

étape	0	1	2	3	4
(a)	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$

Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



Les coulisses

distances :

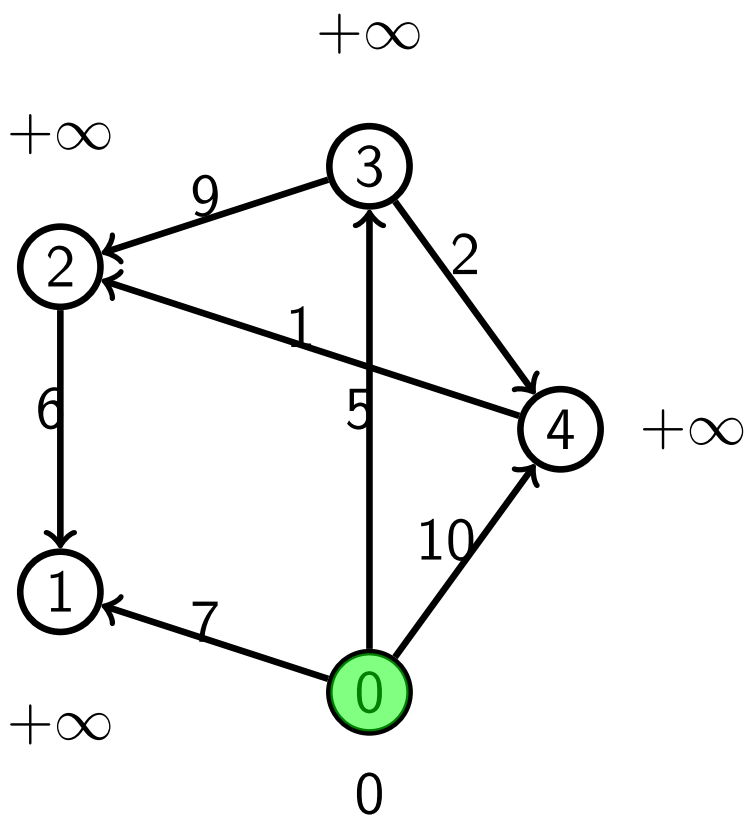
étape	0	1	2	3	4
(a)	0	+∞	+∞	+∞	+∞

parents :

0	1	2	3	4

Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



Les coulisses

distances :

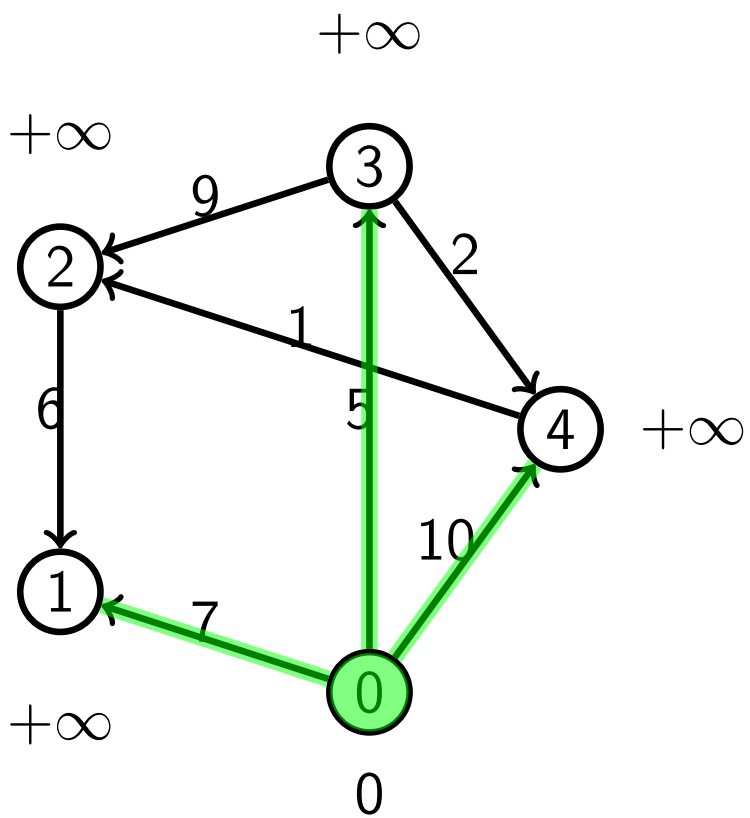
étape	0	1	2	3	4
(a)	0	+∞	+∞	+∞	+∞

parents :

0	1	2	3	4

Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



Les coulisses

distances :

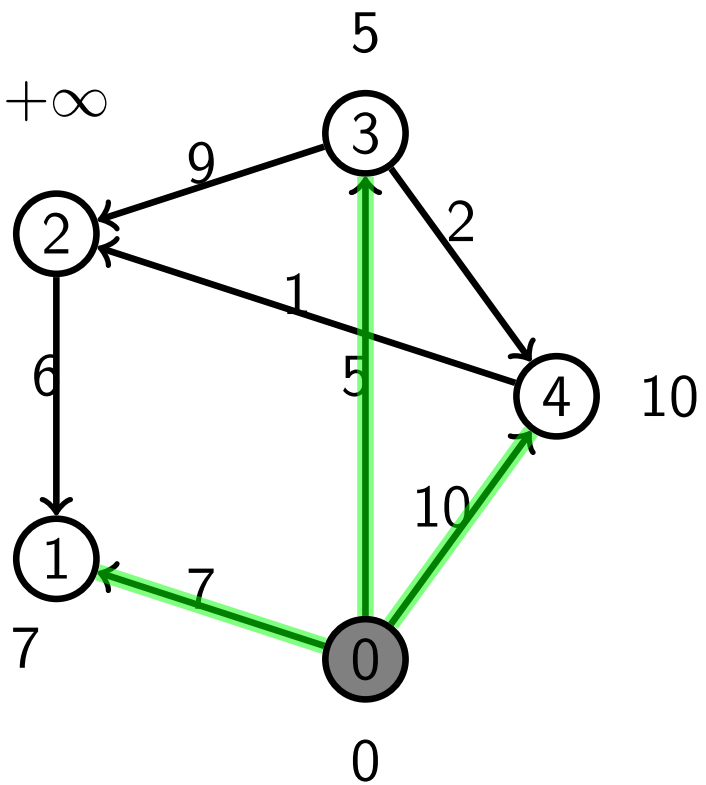
étape	0	1	2	3	4
(a)	0	+∞	+∞	+∞	+∞

parents :

0	1	2	3	4

Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



Les coulisses

distances :

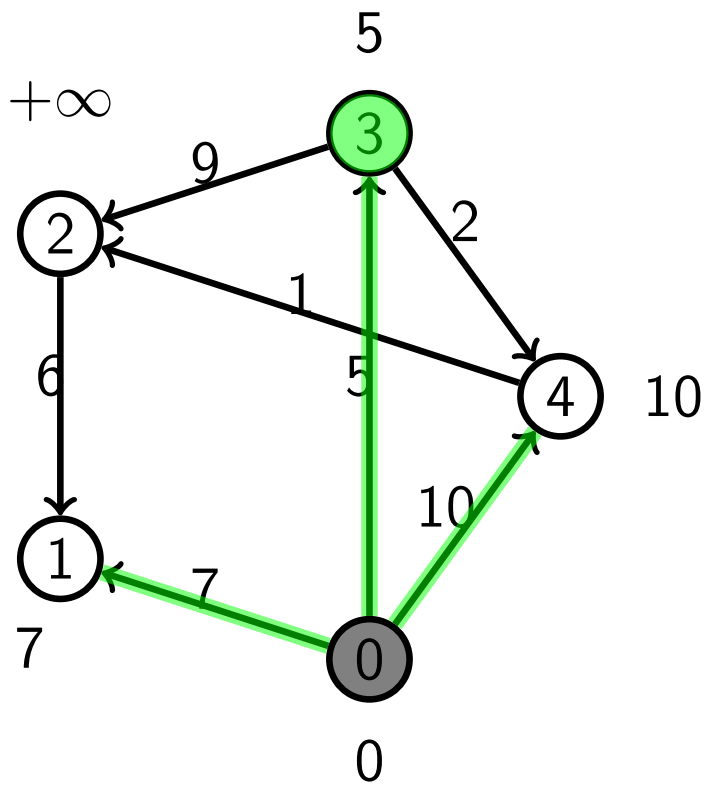
étape	0	1	2	3	4
(a)	0	+∞	+∞	+∞	+∞
(b)	0	7	+∞	5	10

parents :

0	1	2	3	4
	0		0	0

Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



Les coulisses

distances :

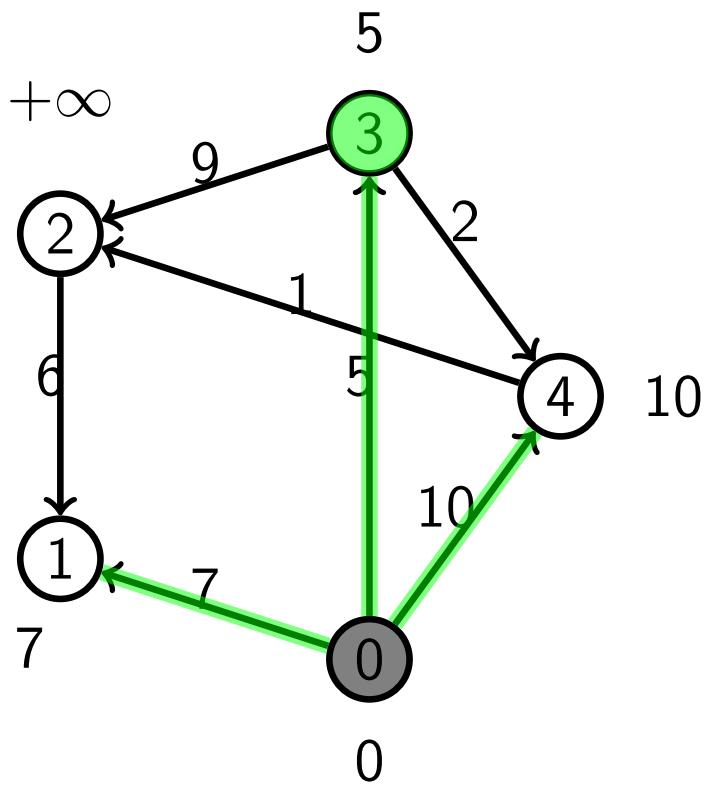
étape	0	1	2	3	4
(a)	0	+∞	+∞	+∞	+∞
(b)	0	7	+∞	5	10

parents :

0	1	2	3	4
	0		0	0

Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



Les coulisses

distances :

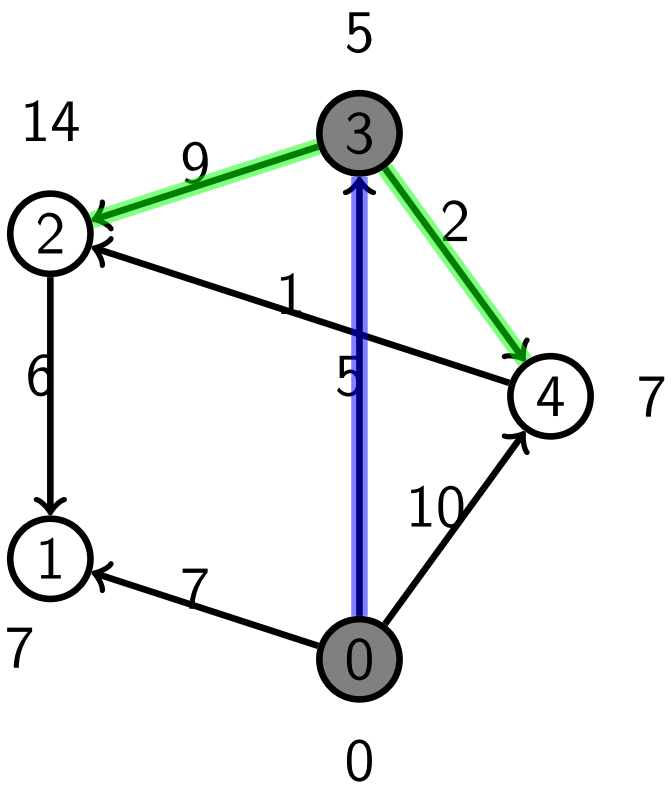
étape	0	1	2	3	4
(a)	0	+∞	+∞	+∞	+∞
(b)	0	7	+∞	5	10

parents :

0	1	2	3	4
	0		0	0

Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



Les coulisses

distances :

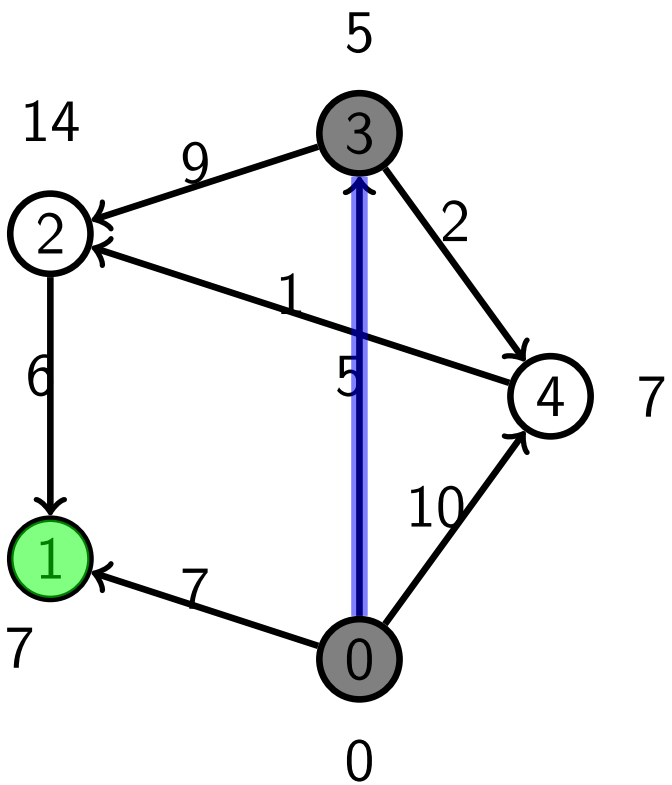
étape	0	1	2	3	4
(a)	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$
(b)	0	7	$+\infty$	5	10
(c)	0	7	14	5	7

parents :

0	1	2	3	4
	0	3	0	3

Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



Les coulisses

distances :

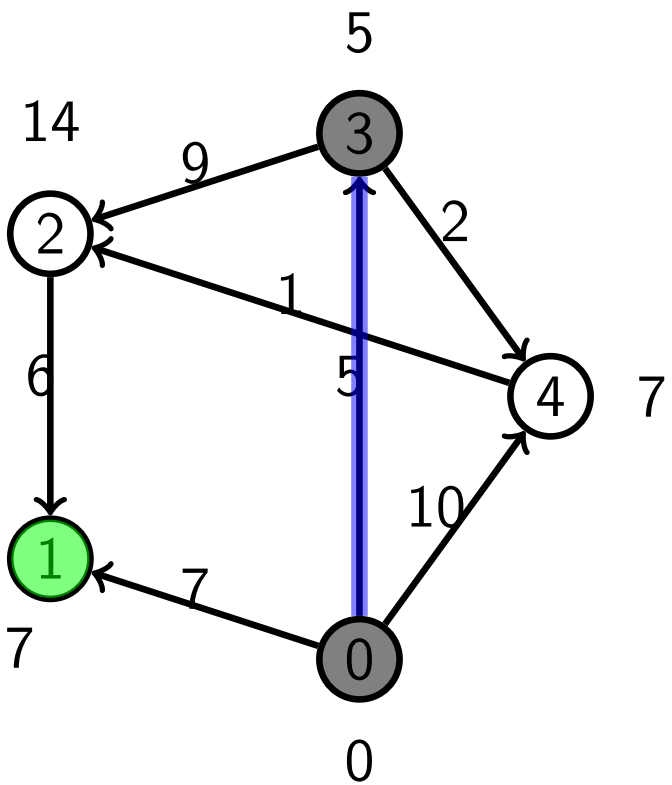
étape	0	1	2	3	4
(a)	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$
(b)	0	7	$+\infty$	5	10
(c)	0	7	14	5	7

parents :

0	1	2	3	4
	0	3	0	3

Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



Les coulisses

distances :

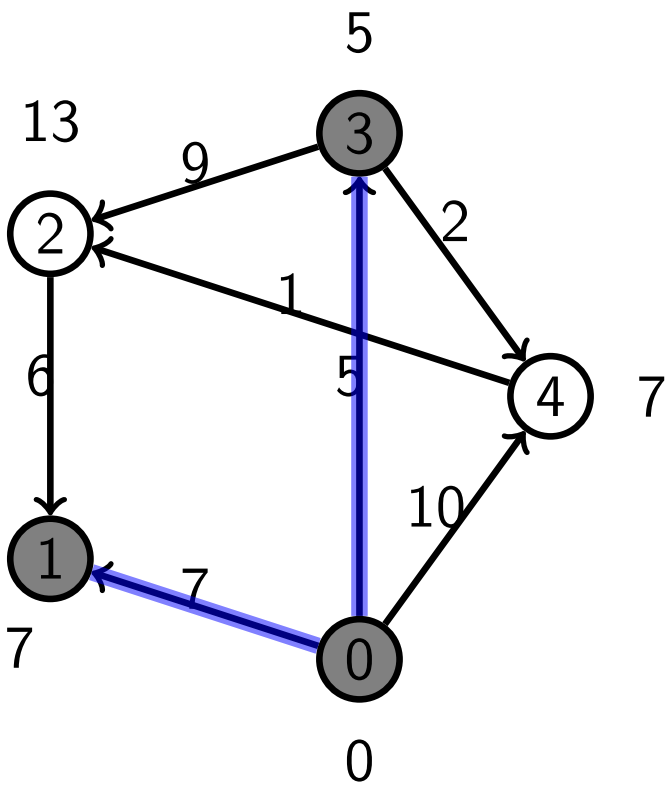
étape	0	1	2	3	4
(a)	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$
(b)	0	7	$+\infty$	5	10
(c)	0	7	14	5	7

parents :

0	1	2	3	4
	0	3	0	3

Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



Les coulisses

distances :

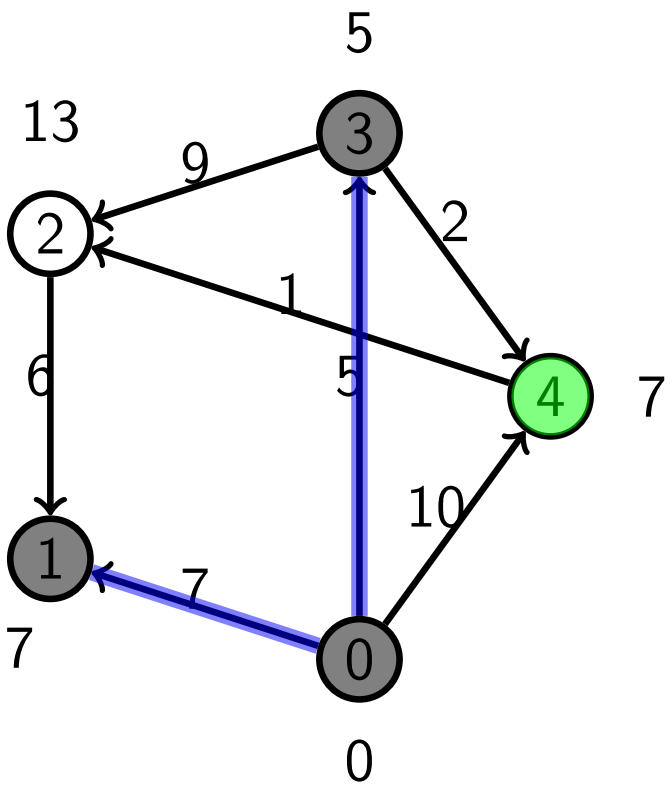
étape	0	1	2	3	4
(a)	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$
(b)	0	7	$+\infty$	5	10
(c)	0	7	14	5	7
(d)	0	7	14	5	7

parents :

0	1	2	3	4
	0	3	0	3

Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



Les coulisses

distances :

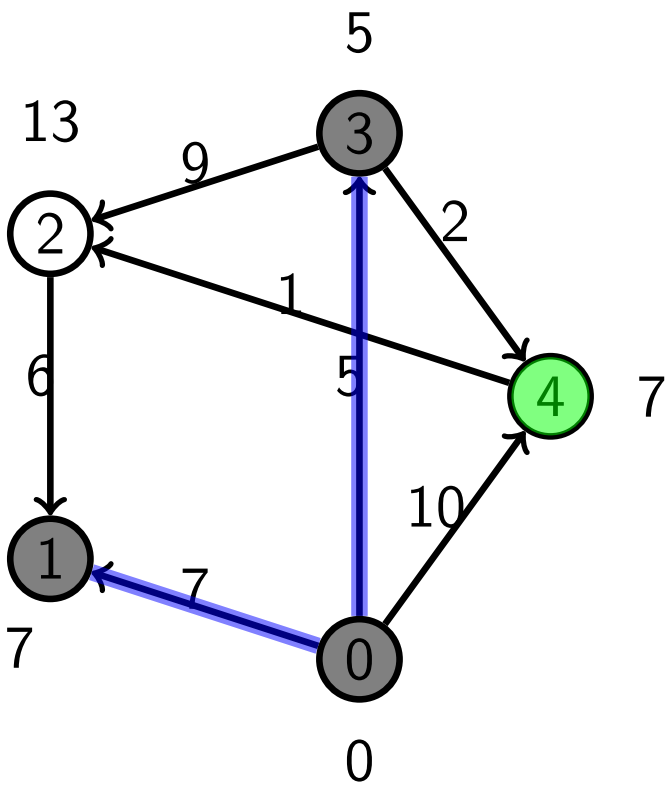
étape	0	1	2	3	4
(a)	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$
(b)	0	7	$+\infty$	5	10
(c)	0	7	14	5	7
(d)	0	7	14	5	7

parents :

0	1	2	3	4
	0	3	0	3

Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



Les coulisses

distances :

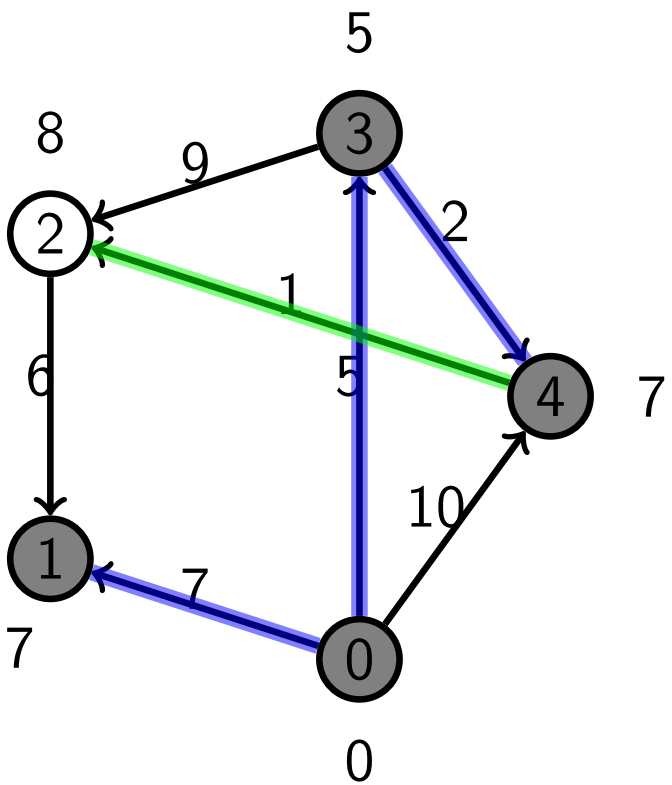
étape	0	1	2	3	4
(a)	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$
(b)	0	7	$+\infty$	5	10
(c)	0	7	14	5	7
(d)	0	7	14	5	7

parents :

0	1	2	3	4
	0	3	0	3

Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



Les coulisses

distances :

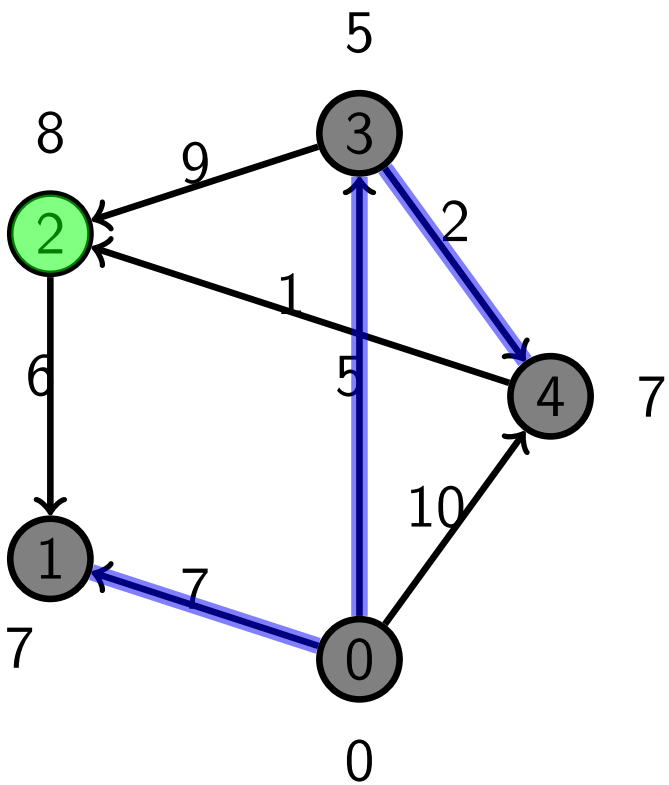
étape	0	1	2	3	4
(a)	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$
(b)	0	7	$+\infty$	5	10
(c)	0	7	14	5	7
(d)	0	7	14	5	7
(e)	0	7	8	5	7

parents :

0	1	2	3	4
	0	4	0	3

Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



Les coulisses

distances :

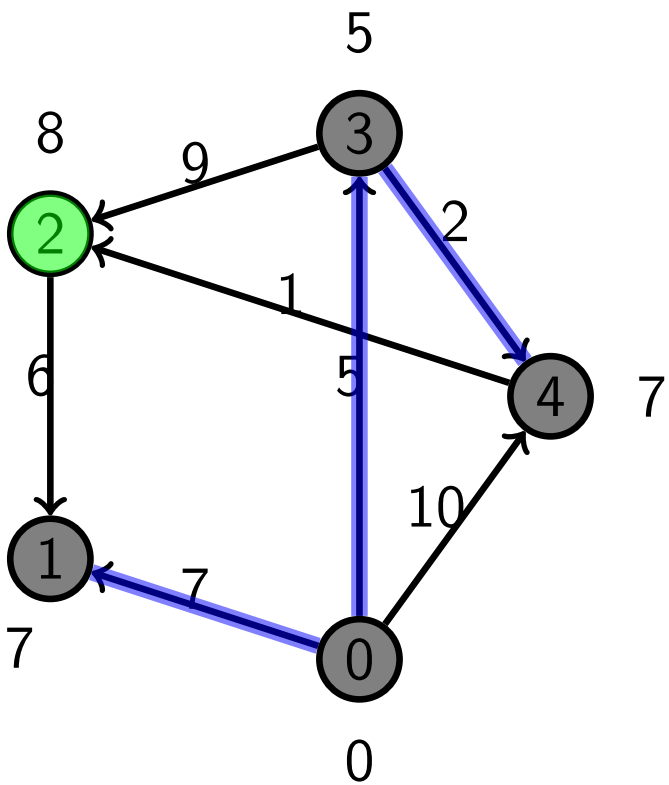
étape	0	1	2	3	4
(a)	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$
(b)	0	7	$+\infty$	5	10
(c)	0	7	14	5	7
(d)	0	7	14	5	7
(e)	0	7	8	5	7

parents :

0	1	2	3	4
	0	4	0	3

Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



Les coulisses

distances :

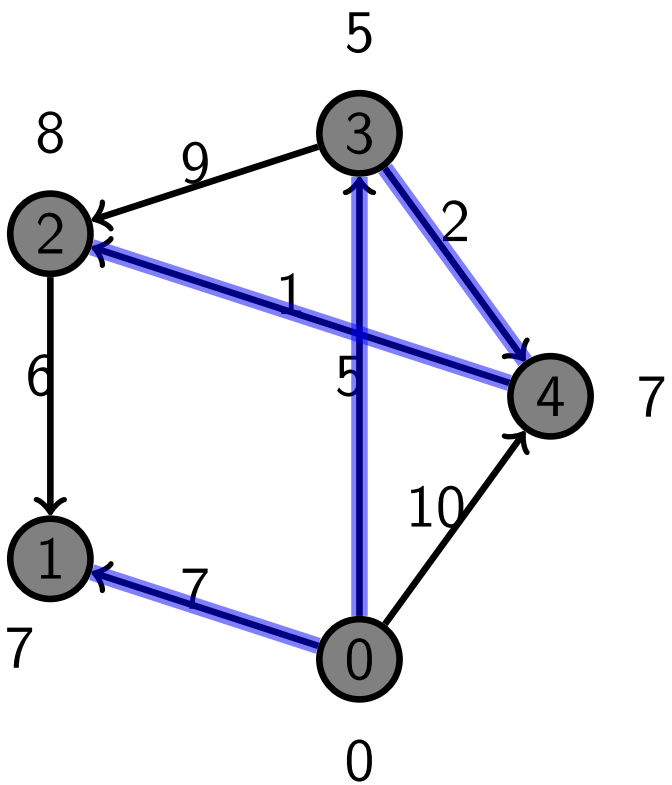
étape	0	1	2	3	4
(a)	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$
(b)	0	7	$+\infty$	5	10
(c)	0	7	14	5	7
(d)	0	7	14	5	7
(e)	0	7	8	5	7

parents :

0	1	2	3	4
	0	4	0	3

Déroulement de l'algorithme de Dijkstra

Exemple 2 (source = 0)



Les coulisses

distances :

étape	0	1	2	3	4
(a)	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$
(b)	0	7	$+\infty$	5	10
(c)	0	7	14	5	7
(d)	0	7	14	5	7
(e)	0	7	8	5	7
(f)	0	7	8	5	7

parents :

0	1	2	3	4
	0	4	0	3

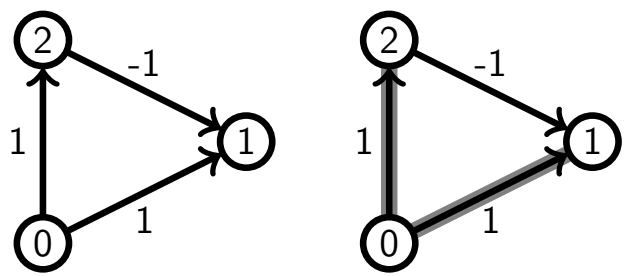
Poids négatifs

L'algorithme de Dijkstra ne fonctionne (en général) pas s'il y a des arcs de poids négatif.

S'il y a des cycles de coût négatif accessibles à partir de la source, il n'y a pas de plus courts chemins.

Même sans cycle de coût négatif, s'il y a des arcs de poids négatif, Dijsktra est faux.

Exemple 3 (plus courts chemins depuis 0)



1. la solution "rate" le plus court chemin $0 \to 2 \to 1$.

Extraction du sommet le plus proche

L'extraction du minimum peut se faire à l'aide d'un algorithme naïf :

Algorithme 1 : ExtraireSommetLePlusProche(S, distances)

Entrées : un ensemble S de sommets, la distance de chaque sommet de S.

Résultat : le sommet de S le plus proche est extrait et renvoyé.

```
1 sommet ← NIL;
2 distance_min ← +∞;
3 pour chaque candidat ∈ S faire
4 |   si distances[candidat] < distance_min alors
5 |   |   sommet ← candidat;
6 |   |   distance_min ← distances[candidat];
7 si sommet ≠ NIL alors S ← S\ sommet ;
8 renvoyer sommet;
```

Extraction du sommet le plus proche

L'extraction du minimum peut se faire à l'aide d'un algorithme naïf :

Algorithme 1 : ExtraireSommetLePlusProche(S, distances)

Entrées : un ensemble S de sommets, la distance de chaque sommet de S.

Résultat : le sommet de S le plus proche est extrait et renvoyé.

```
1 sommet ← NIL;
2 distance_min ← +∞;
3 pour chaque candidat ∈ S faire
4 |   si distances[candidat] < distance_min alors
5 |   |   sommet ← candidat;
6 |   |   distance_min ← distances[candidat];
7 si sommet ≠ NIL alors S ← S\ sommet ;
8 renvoyer sommet;
```

Un tas comme serait plus efficace. Mais attention, ici, les poids des éléments changent en cours d'exécution !

L'algorithme de Dijkstra proprement dit

Algorithme 2 : DIJKSTRA(G, source)

Entrées : un graphe pondéré orienté G, un sommet source.
Sortie : la longueur d'un plus court chemin de la source à chacun des sommets du graphe ($+\infty$ pour les sommets non accessibles).

```
1 a_traiter ← G.sommets();
2 distances ← tableau(G.nombre_sommets(), +∞);
3 distances[source] ← 0;
4 tant que a_traiter.pas_vide() faire
5 |   u ← ExtraireSommetLePlusProche(a_traiter, distances);
6 |   si u = NIL alors renvoyer distances ;
7 |   pour chaque v ∈ G.successeurs(u) faire
8 |   |   distances[v] ← min(distances[v], distances[u] + G.poids_arc(u,
  |   |     v));
9 renvoyer distances;
```

Complexité

- On passe $O(|V|)$ fois dans la boucle principale ;

Complexité

- On passe $O(|V|)$ fois dans la boucle principale ;
- Extraire chaque sommet coûte $O(|S|) = O(|V|)$;

Complexité

- On passe $O(|V|)$ fois dans la boucle principale ;
- Extraire chaque sommet coûte $O(|S|) = O(|V|)$;
- On examine chaque arc une fois ;
  $\Rightarrow O(|A| + |V|^2) = O(|V|^2)$.

Complexité

- On passe $O(|V|)$ fois dans la boucle principale ;
- Extraire chaque sommet coûte $O(|S|) = O(|V|)$;
- On examine chaque arc une fois ;
  $\Rightarrow O(|A| + |V|^2) = O(|V|^2)$.
- Une structure de tas adaptée permet de rabaisser la complexité à $O((|V| + |A|) \log |V|)$.

Correction de l'algorithme de Dijkstra

Soit T l'ensemble des sommets déjà traités, et :

- $\delta(s,t) =$ la distance **réelle** de s à t ;
- distance$(s,t) =$ la distance de s à t **calculée à chaque étape de l'algorithme.**

On a toujours distance$(s,t) \geq \delta(s,t)$.

Correction de l'algorithme de Dijkstra

Montrons par **induction** sur $|T|$ que

- P_1 : pour tout sommet t de T (en particulier quand t entre dans T) on a distance$(s, t) = \delta(s, t)$.
- P_2 : pour toute sommet $t \notin T$, distance(s, t) est le coût d'un plus court chemin de s à t qui ne passe que par des sommets intermédiaires dans T.

Correction de l'algorithme de Dijkstra

Montrons par **induction** sur $|T|$ que

- P_1 : pour tout sommet t de T (en particulier quand t entre dans T) on a distance$(s,t) = \delta(s,t)$.
- P_2 : pour toute sommet $t \notin T$, distance(s,t) est le coût d'un plus court chemin de s à t qui ne passe que par des sommets intermédiaires dans T.

1. **cas de base :** $T = \{s\}$, et distance$(s,s) = 0 = \delta(s,s)$. On a aussi P_2 avec la relaxation faite quand s rentre dans T.

Correction de l'algorithme de Dijkstra

Montrons par **induction** sur $|T|$ que

- P_1 : pour tout sommet t de T (en particulier quand t entre dans T) on a distance$(s,t) = \delta(s,t)$.
- P_2 : pour toute sommet $t \notin T$, distance(s,t) est le coût d'un plus court chemin de s à t qui ne passe que par des sommets intermédiaires dans T.

1. **cas de base :** $T = \{s\}$, et distance$(s,s) = 0 = \delta(s,s)$. On a aussi P_2 avec la relaxation faite quand s rentre dans T.
2. **induction :**

Correction de l'algorithme de Dijkstra

Montrons par **induction** sur $|T|$ que

- P_1 : pour tout sommet t de T (en particulier quand t entre dans T) on a distance$(s,t) = \delta(s,t)$.
- P_2 : pour toute sommet $t \notin T$, distance(s,t) est le coût d'un plus court chemin de s à t qui ne passe que par des sommets intermédiaires dans T.

1. **cas de base :** $T = \{s\}$, et distance$(s,s) = 0 = \delta(s,s)$. On a aussi P_2 avec la relaxation faite quand s rentre dans T.
2. **induction :**
   - hypothèses d'induction : P_1 et P_2

Correction de l'algorithme de Dijkstra

Montrons par **induction** sur $|T|$ que

- P_1 : pour tout sommet t de T (en particulier quand t entre dans T) on a distance$(s,t) = \delta(s,t)$.
- P_2 : pour toute sommet $t \notin T$, distance(s,t) est le coût d'un plus court chemin de s à t qui ne passe que par des sommets intermédiaires dans T.

1. **cas de base :** $T = \{s\}$, et distance$(s,s) = 0 = \delta(s,s)$. On a aussi P_2 avec la relaxation faite quand s rentre dans T.
2. **induction :**
   - hypothèses d'induction : P_1 et P_2
   - à prouver : Soit $t \notin T$ le prochain sommet à traiter. On doit montrer que P_1 et P_2 sont vraies après traitement de t avec $T \cup \{t\}$.

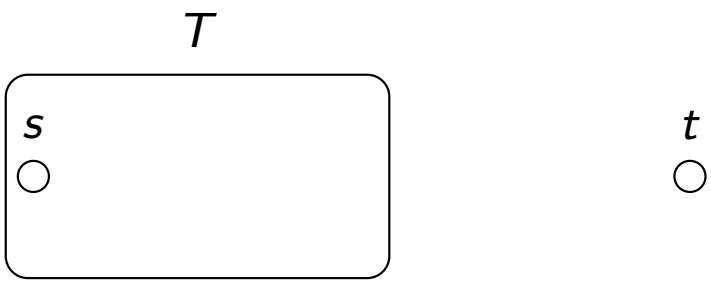
Correction de l'algorithme de Dijkstra

Montrons par **induction** sur $|T|$ que

- P_1 : pour tout sommet t de T (en particulier quand t entre dans T) on a distance$(s,t) = \delta(s,t)$.
- P_2 : pour toute sommet $t \notin T$, distance(s,t) est le coût d'un plus court chemin de s à t qui ne passe que par des sommets intermédiaires dans T.

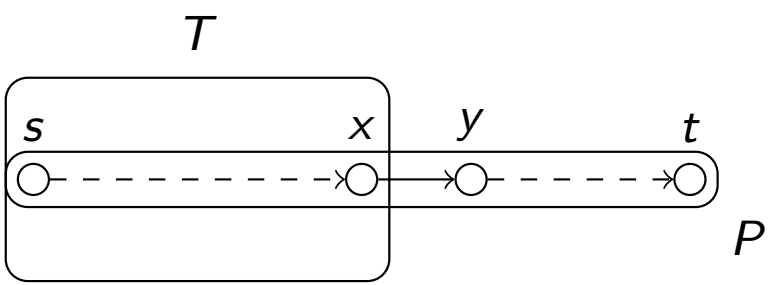
1. **cas de base :** $T = \{s\}$, et distance$(s,s) = 0 = \delta(s,s)$. On a aussi P_2 avec la relaxation faite quand s rentre dans T.
2. **induction :**
   - hypothèses d'induction : P_1 et P_2
   - à prouver : Soit $t \notin T$ le prochain sommet à traiter. On doit montrer que P_1 et P_2 sont vraies après traitement de t avec $T \cup \{t\}$.
   - On prend un plus court chemin P de s à t de poids $\delta(s,t)$. Le premier sommet qui sort de T dans ce chemin est noté y.

Correction de l'algorithme de Dijkstra



□

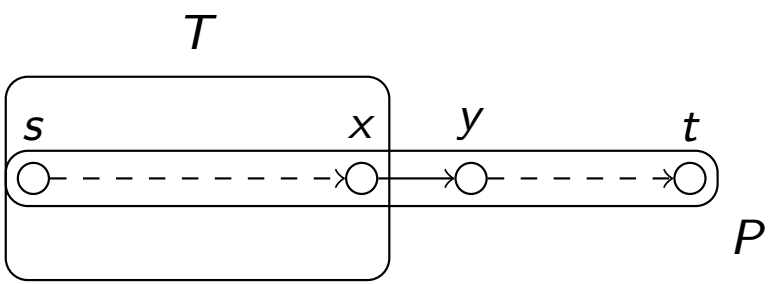
Correction de l'algorithme de Dijkstra



$\delta(s,t) \geq \delta(s,y)$ car les poids sont ≥ 0

□

Correction de l'algorithme de Dijkstra

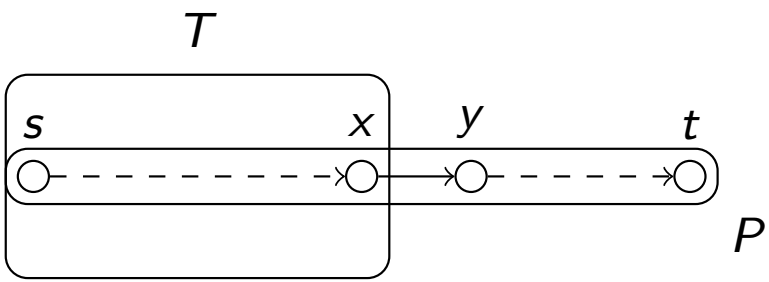


$\delta(s,t) \geq \delta(s,y)$ car les poids sont ≥ 0

$\geq$ distance(s,y) car on a P_2

□

Correction de l'algorithme de Dijkstra



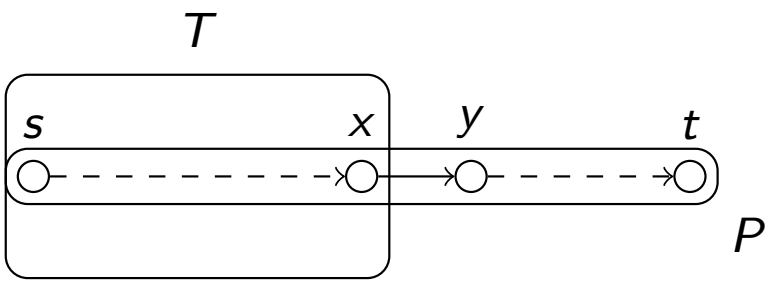
$\delta(s,t) \geq \delta(s,y)$ car les poids sont ≥ 0

$\geq$ distance(s,y) car on a P_2

$\geq$ distance(s,t) car t va rentrer dans T.

□

Correction de l'algorithme de Dijkstra



$\delta(s,t) \geq \delta(s,y)$ car les poids sont ≥ 0

$\geq$ distance(s,y) car on a P_2

$\geq$ distance(s,t) car t va rentrer dans T.

Donc $\delta(s,t) =$ distance(s,t) et P_1 est vraie pour $T \cup \{t\}$. Avec la relaxation faite lorsque t entre dans T, P_2 va rester vraie pour $T \cup \{t\}$. □

Plus courts chemins à partir d'une source pour les graphes acycliques

Algorithme 3 : DijkstraAcyclique(G, source)

Entrées : un graphe pondéré orienté G acyclique, même avec des arcs de poids négatifs, un sommet source.

Sortie : la longueur d'un plus court chemin de la source à chacun des sommets du graphe ($+\infty$ pour les sommets non accessibles).

```
1 distances ← tableau(G.nombre_sommets(), +∞);
2 distances[source] ← 0;
3 pour chaque u dans un ordre topologique faire
4 |   pour chaque v ∈ G.successeurs(u) faire
5 |   |   distances[v] ← min(distances[v], distances[u] + G.poids_arc(u,
  |   |     v));
6 renvoyer distances;
```

Correction des plus courts chemins à partir d'une source pour les graphes acycliques

La preuve et la même que pour Dijkstra. On utilise cette fois le fait que les sommets sont en ordre topologique.

Complexité des plus courts chemins à partir d'une source pour les graphes acycliques

- Un ordre topologique s'obtient en $O(|V| + |A|)$ avec des listes d'adjacence ;

Complexité des plus courts chemins à partir d'une source pour les graphes acycliques

- Un ordre topologique s'obtient en $O(|V| + |A|)$ avec des listes d'adjacence ;
- On examine chaque arc une fois ;
  $\Rightarrow O(|A| + |V|)$.

Problèmes de Dijkstra

- En l'absence de poids négatifs, Dijkstra s'en sortait puisqu'il n'était pas nécessaire de revenir sur ses décisions ;

Problèmes de Dijkstra

- En l'absence de poids négatifs, Dijkstra s'en sortait puisqu'il n'était pas nécessaire de revenir sur ses décisions ;
- L'algorithme de Bellman-Ford règlera ce problème en examinant les arcs plusieurs fois.

L'algorithme de Bellman-Ford

- L'algorithme de Bellman-Ford recherche également des raccourcis entre les sommets ;

L'algorithme de Bellman-Ford

- L'algorithme de Bellman-Ford recherche également des raccourcis entre les sommets ;
- Le fonctionnement est différent de celui de Dijkstra, qui cherchait des chemins moins coûteux de la source s à un sommet u en passant par un autre sommet v ;

L'algorithme de Bellman-Ford

- L'algorithme de Bellman-Ford recherche également des raccourcis entre les sommets ;
- Le fonctionnement est différent de celui de Dijkstra, qui cherchait des chemins moins coûteux de la source s à un sommet u en passant par un autre sommet v ;
- Ici, on vérifie pour chaque arc (u, v) s'il existe un chemin moins coûteux de u à v ;

L'algorithme de Bellman-Ford

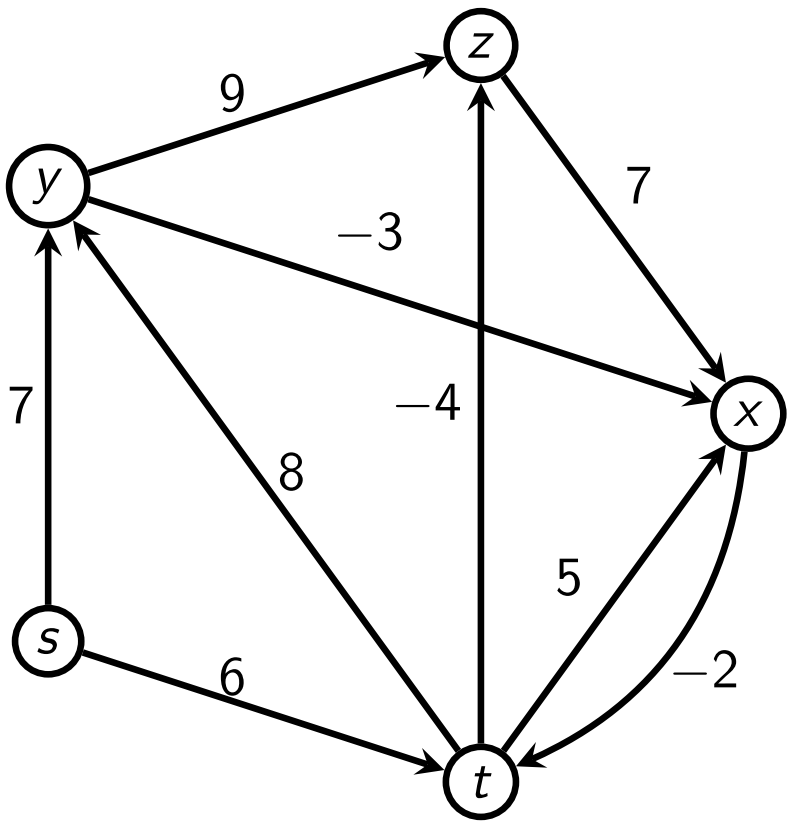
- L'algorithme de Bellman-Ford recherche également des raccourcis entre les sommets ;
- Le fonctionnement est différent de celui de Dijkstra, qui cherchait des chemins moins coûteux de la source s à un sommet u en passant par un autre sommet v ;
- Ici, on vérifie pour chaque arc (u, v) s'il existe un chemin moins coûteux de u à v ;
- On examine l'ensemble de **tous** les arcs $|V|$ fois.

Déroulement de l'algorithme de Bellman-Ford

On examine les arcs dans l'ordre lexicographique :

$$(s,t),(s,y),(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x).$$

Exemple 4 (source = s)



Les coulisses

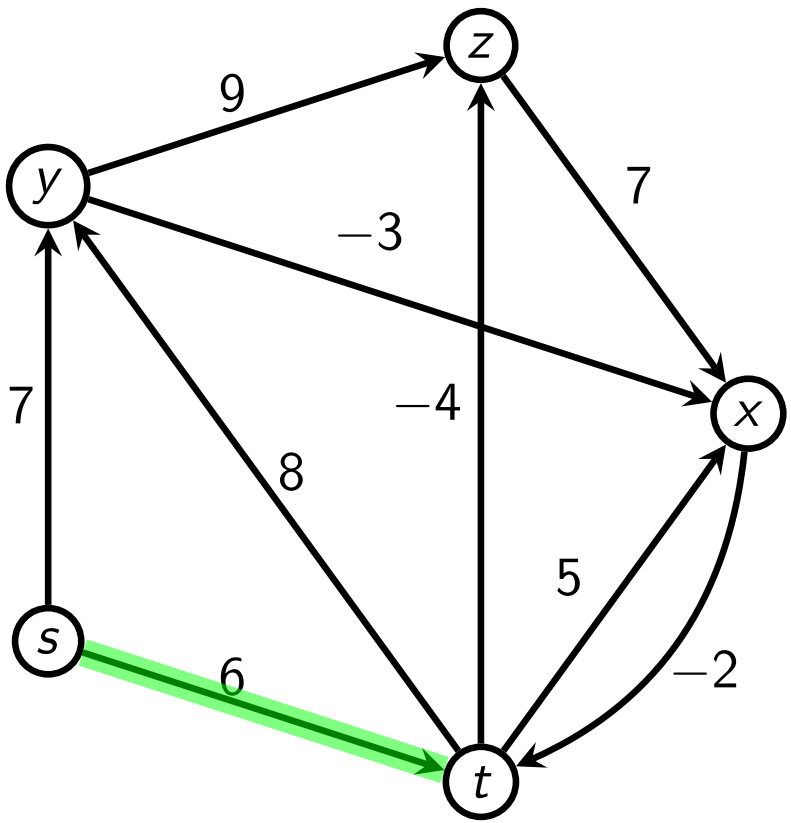
étape	s	t	x	y	z
(a)	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$

Déroulement de l'algorithme de Bellman-Ford

On examine les arcs dans l'ordre lexicographique :

$$(s,t),(s,y),(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x).$$

Exemple 4 (source = s)



Les coulisses

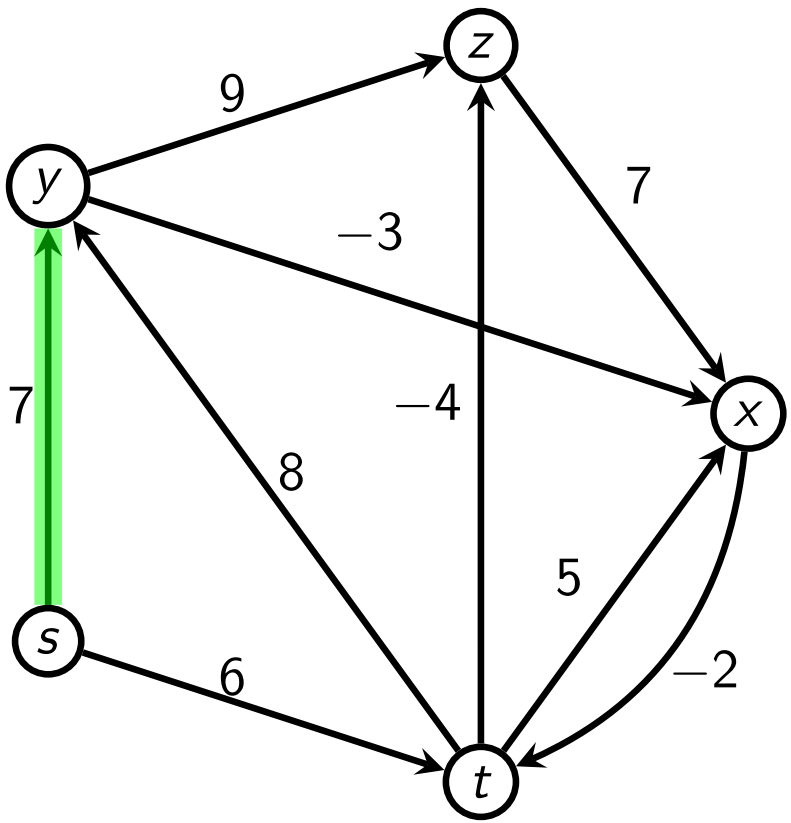
étape	s	t	x	y	z
(a)	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$

Déroulement de l'algorithme de Bellman-Ford

On examine les arcs dans l'ordre lexicographique :

$$(s,t),(s,y),(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x).$$

Exemple 4 (source = s)



Les coulisses

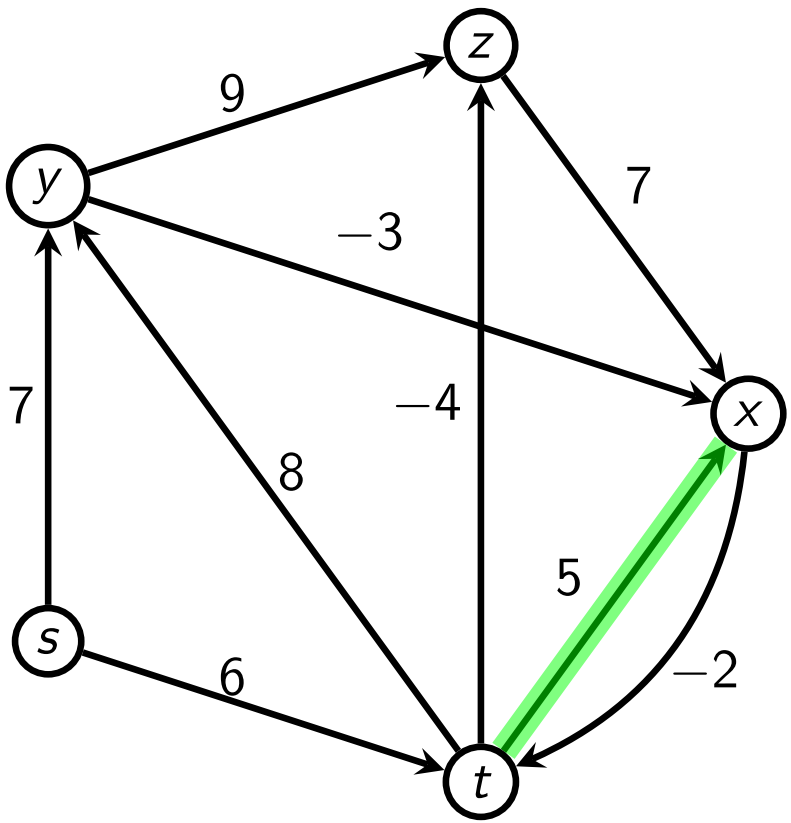
étape	s	t	x	y	z
(a)	0	6	$+\infty$	$+\infty$	$+\infty$

Déroulement de l'algorithme de Bellman-Ford

On examine les arcs dans l'ordre lexicographique :

$$(s,t),(s,y),(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x).$$

Exemple 4 (source = s)



Les coulisses

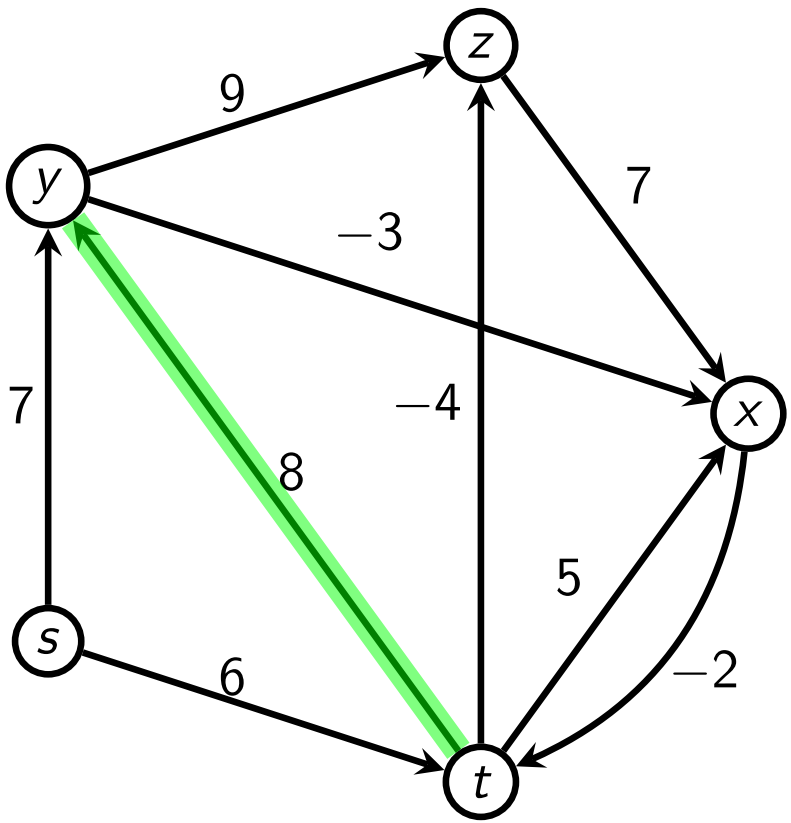
étape	s	t	x	y	z
(a)	0	6	$+\infty$	7	$+\infty$

Déroulement de l'algorithme de Bellman-Ford

On examine les arcs dans l'ordre lexicographique :

$$(s,t),(s,y),(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x).$$

Exemple 4 (source = s)



Les coulisses

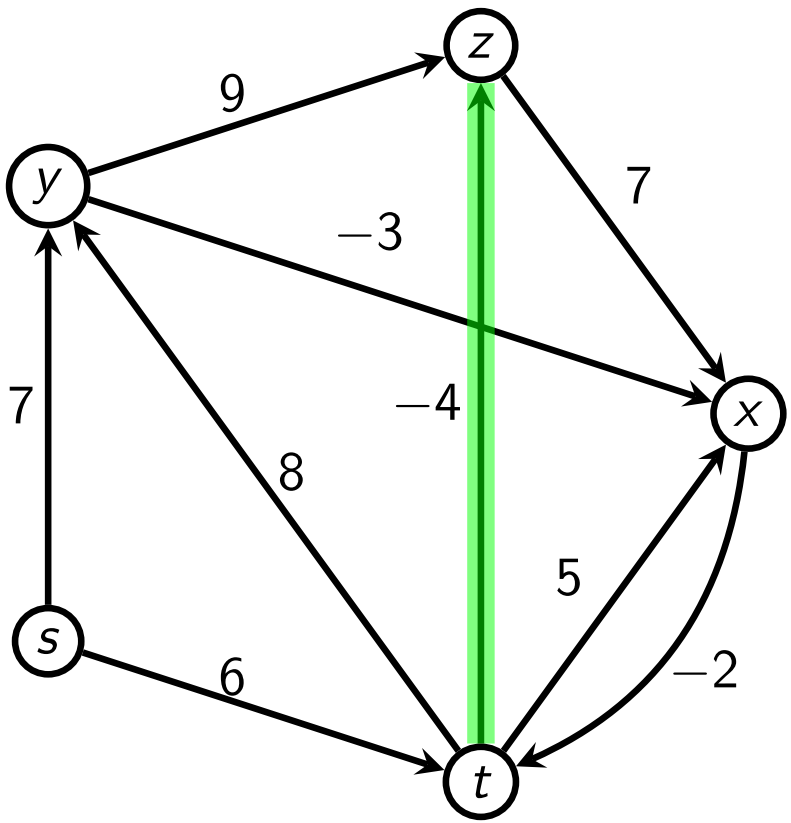
étape	s	t	x	y	z
(a)	0	6	11	7	$+\infty$

Déroulement de l'algorithme de Bellman-Ford

On examine les arcs dans l'ordre lexicographique :

$$(s,t),(s,y),(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x).$$

Exemple 4 (source = s)



Les coulisses

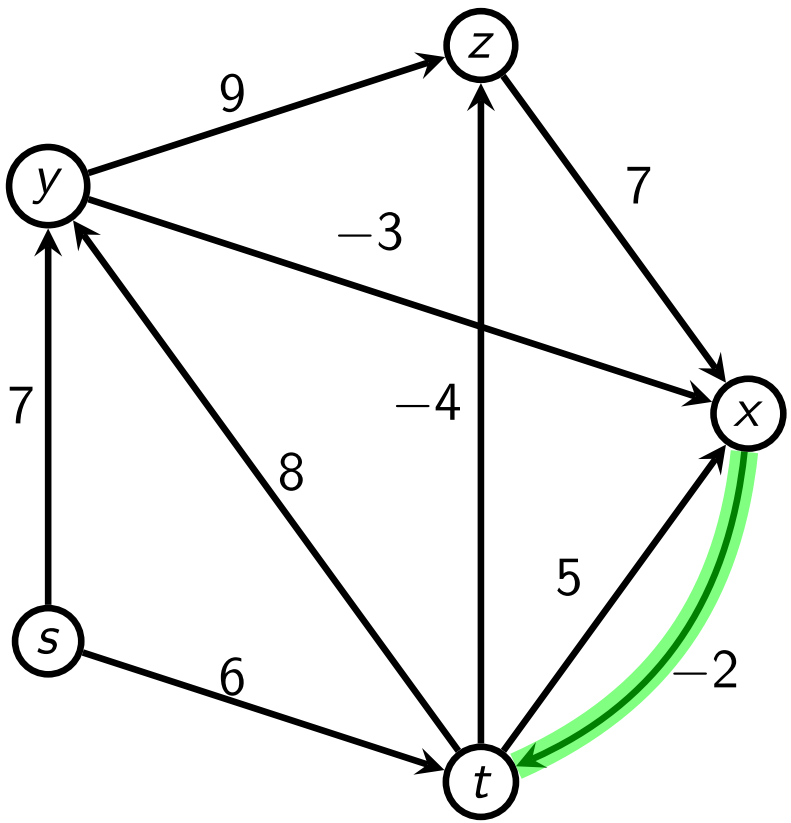
étape	s	t	x	y	z
(a)	0	6	11	7	$+\infty$

Déroulement de l'algorithme de Bellman-Ford

On examine les arcs dans l'ordre lexicographique :

$$(s,t),(s,y),(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x).$$

Exemple 4 (source = s)



Les coulisses

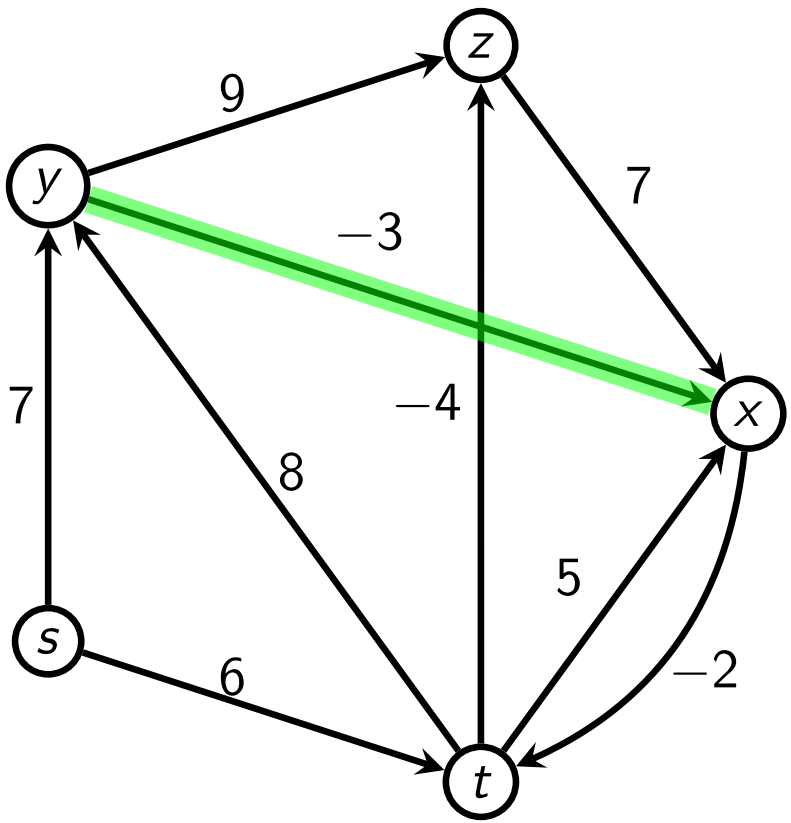
étape	s	t	x	y	z
(a)	0	6	11	7	2

Déroulement de l'algorithme de Bellman-Ford

On examine les arcs dans l'ordre lexicographique :

$$(s,t),(s,y),(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x).$$

Exemple 4 (source = s)



Les coulisses

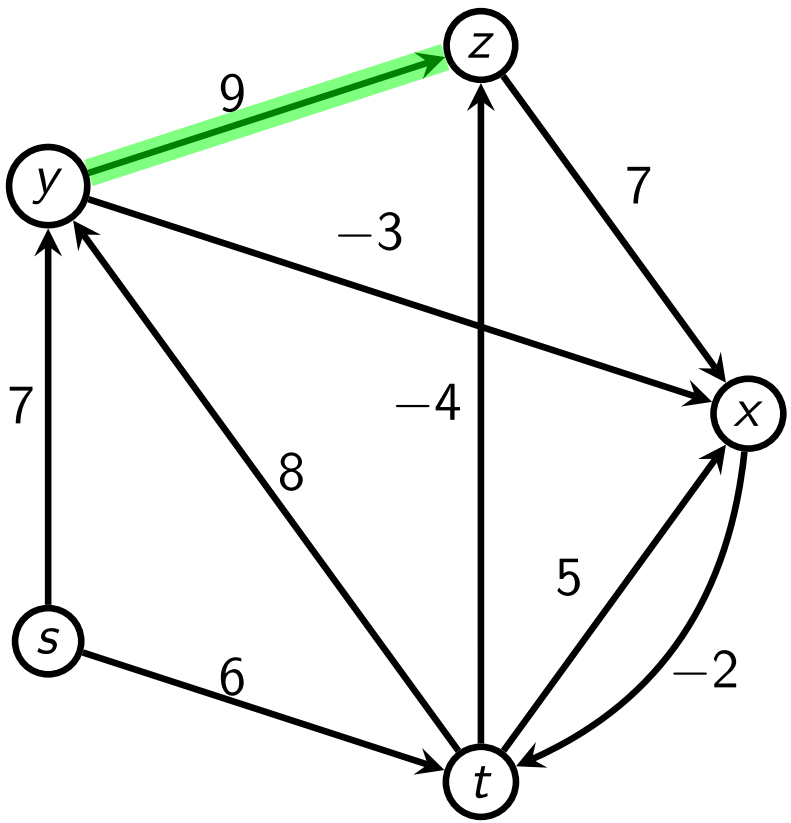
étape	s	t	x	y	z
(a)	0	6	11	7	2

Déroulement de l'algorithme de Bellman-Ford

On examine les arcs dans l'ordre lexicographique :

$$(s,t),(s,y),(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x).$$

Exemple 4 (source = s)



Les coulisses

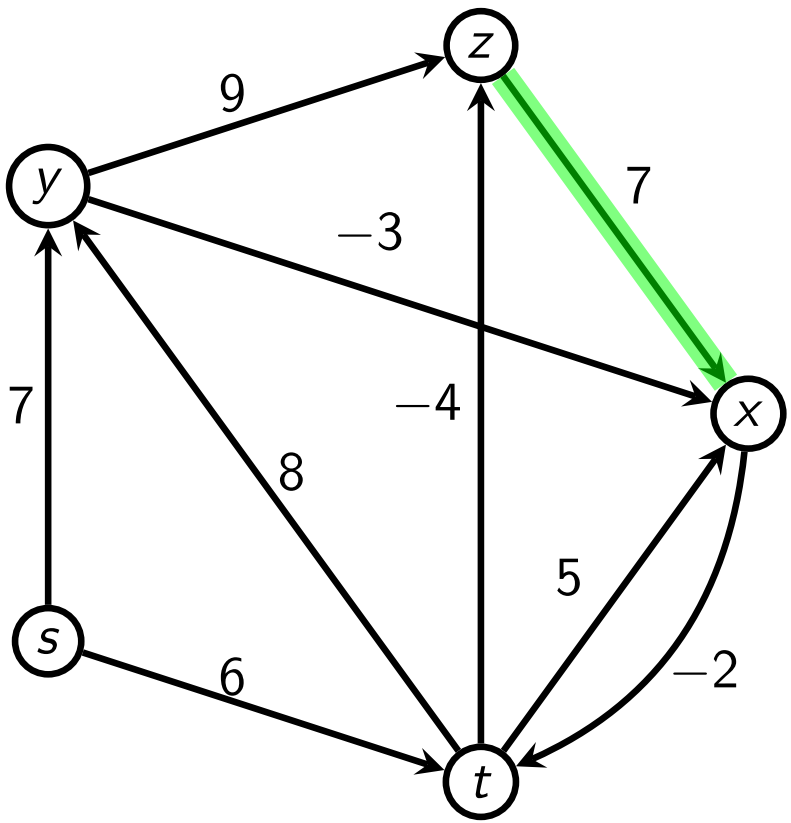
étape	s	t	x	y	z
(a)	0	6	4	7	2

Déroulement de l'algorithme de Bellman-Ford

On examine les arcs dans l'ordre lexicographique :

$$(s,t),(s,y),(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x).$$

Exemple 4 (source = s)



Les coulisses

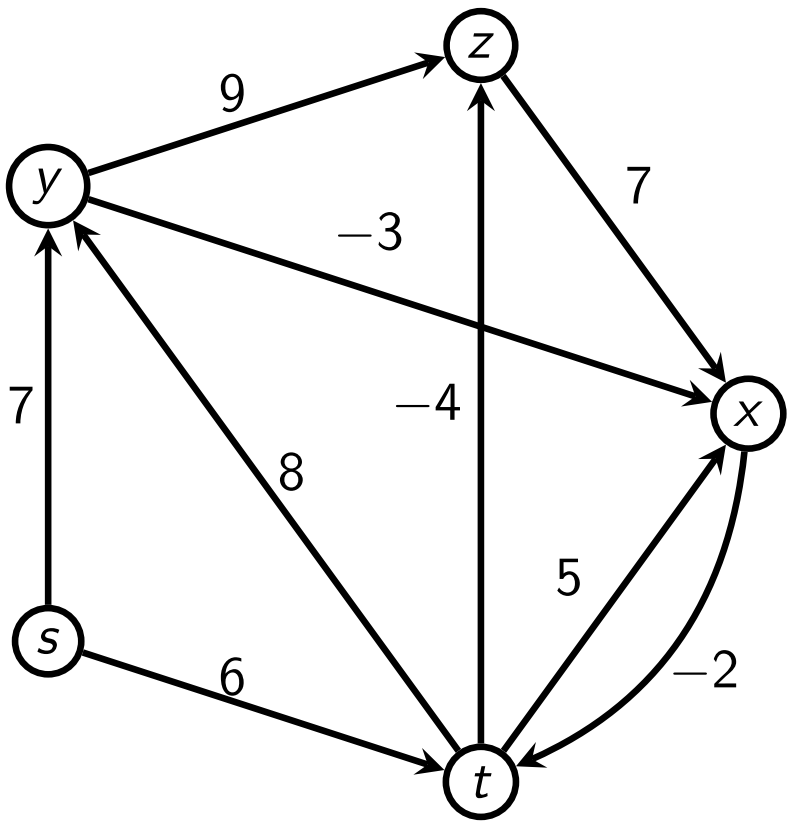
étape	s	t	x	y	z
(a)	0	6	4	7	2

Déroulement de l'algorithme de Bellman-Ford

On examine les arcs dans l'ordre lexicographique :

$$(s,t),(s,y),(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x).$$

Exemple 4 (source = s)



Les coulisses

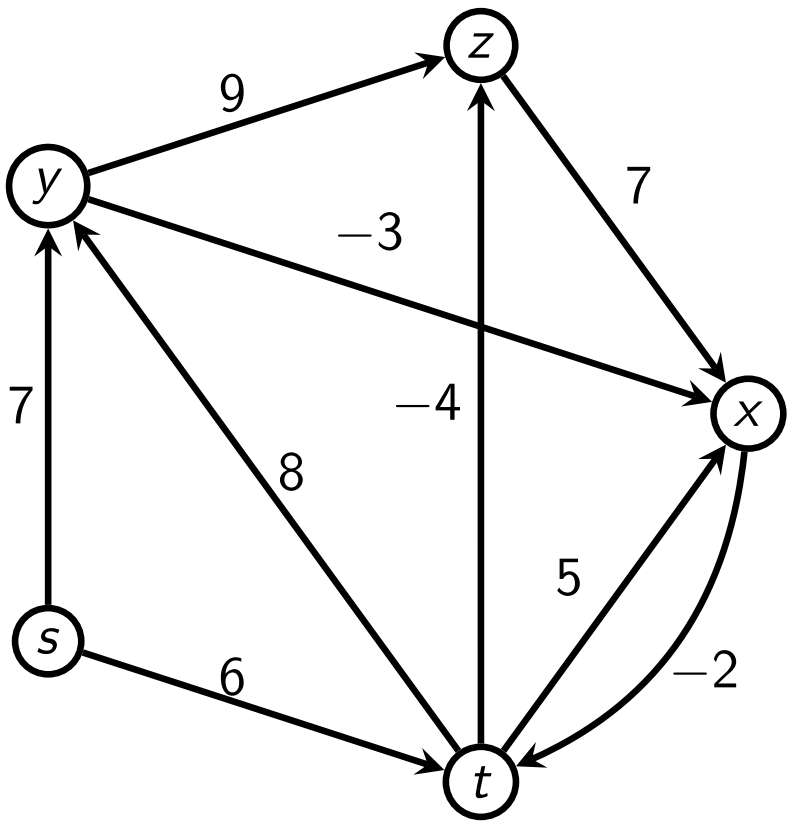
étape	s	t	x	y	z
(a)	0	6	4	7	2

Déroulement de l'algorithme de Bellman-Ford

On examine les arcs dans l'ordre lexicographique :

$$(s,t),(s,y),(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x).$$

Exemple 4 (source = s)



Les coulisses

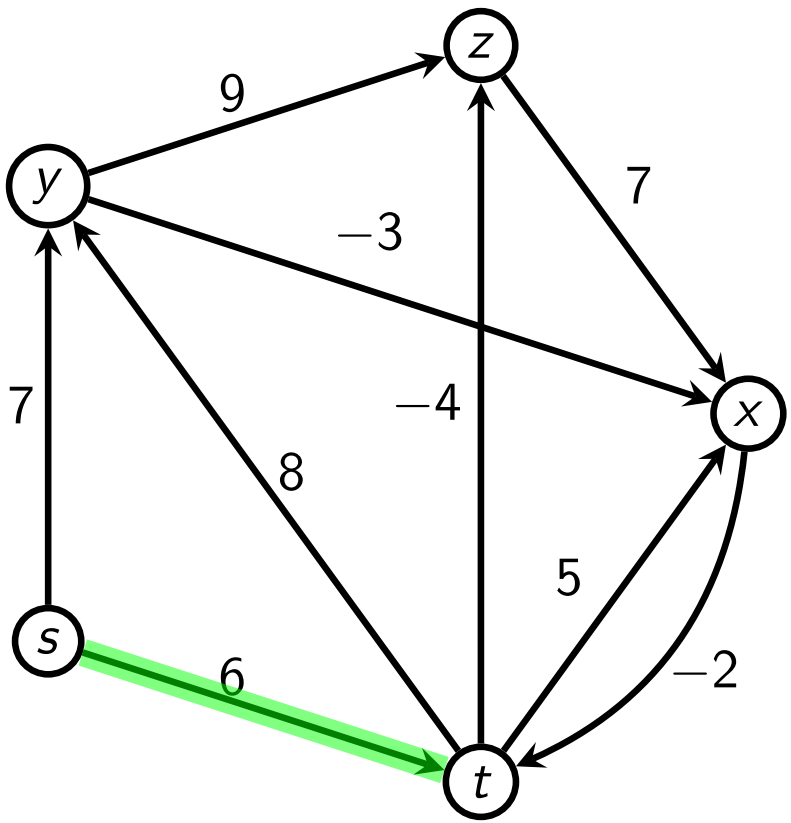
étape	s	t	x	y	z
(a)	0	6	4	7	2
(b)	0	6	4	7	2

Déroulement de l'algorithme de Bellman-Ford

On examine les arcs dans l'ordre lexicographique :

$$(s,t),(s,y),(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x).$$

Exemple 4 (source = s)



Les coulisses

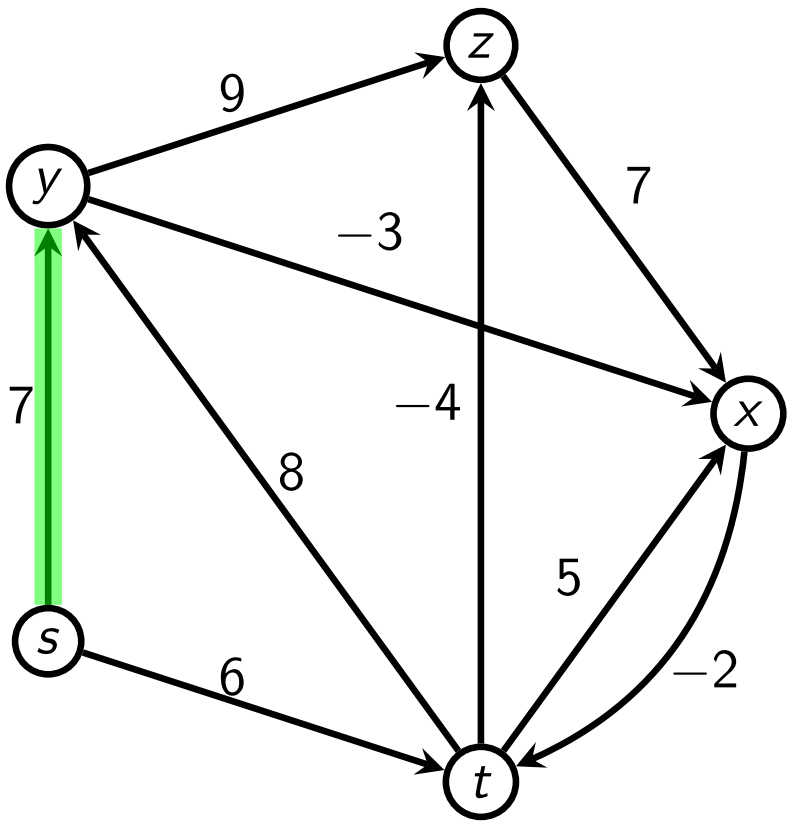
étape	s	t	x	y	z
(a)	0	6	4	7	2
(b)	0	6	4	7	2

Déroulement de l'algorithme de Bellman-Ford

On examine les arcs dans l'ordre lexicographique :

$$(s,t),(s,y),(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x).$$

Exemple 4 (source = s)



Les coulisses

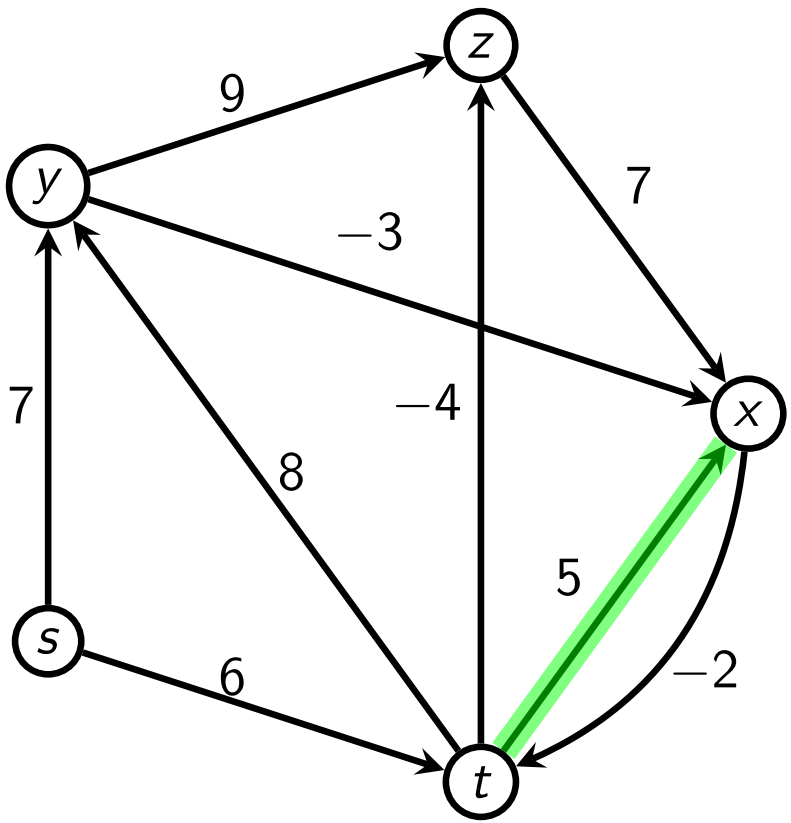
étape	s	t	x	y	z
(a)	0	6	4	7	2
(b)	0	6	4	7	2

Déroulement de l'algorithme de Bellman-Ford

On examine les arcs dans l'ordre lexicographique :

$$(s,t),(s,y),(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x).$$

Exemple 4 (source = s)



Les coulisses

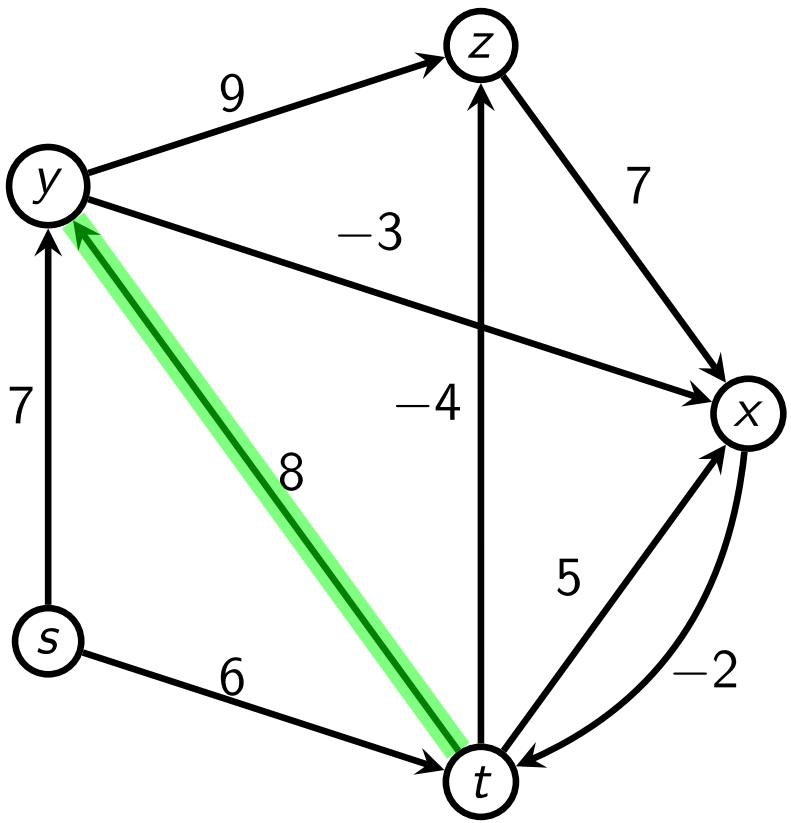
étape	s	t	x	y	z
(a)	0	6	4	7	2
(b)	0	6	4	7	2

Déroulement de l'algorithme de Bellman-Ford

On examine les arcs dans l'ordre lexicographique :

$$(s,t),(s,y),(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x).$$

Exemple 4 (source = s)



Les coulisses

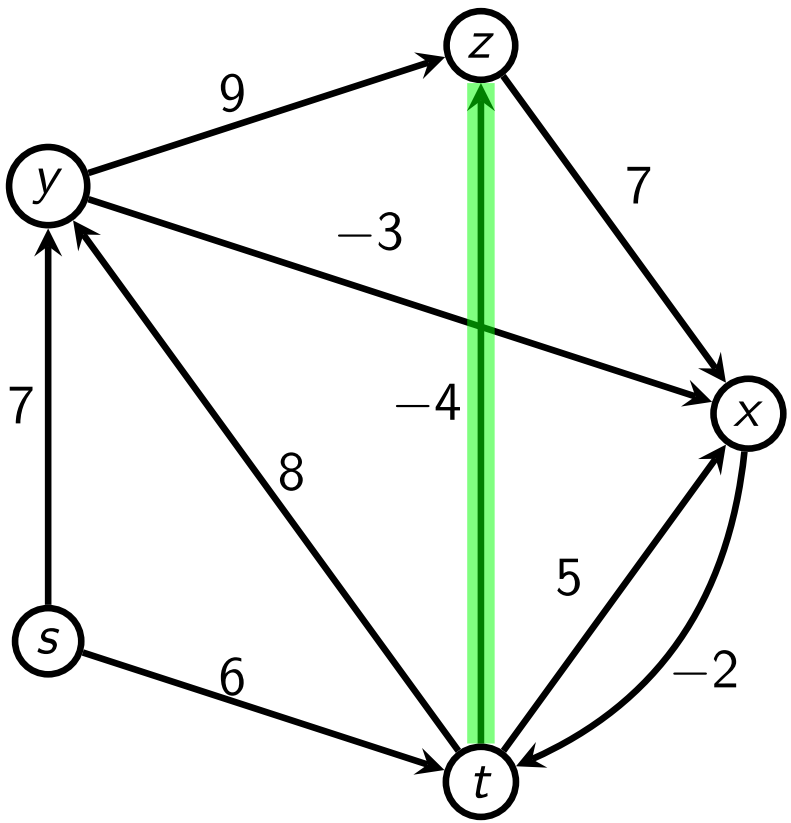
étape	s	t	x	y	z
(a)	0	6	4	7	2
(b)	0	6	4	7	2

Déroulement de l'algorithme de Bellman-Ford

On examine les arcs dans l'ordre lexicographique :

$$(s,t),(s,y),(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x).$$

Exemple 4 (source = s)



Les coulisses

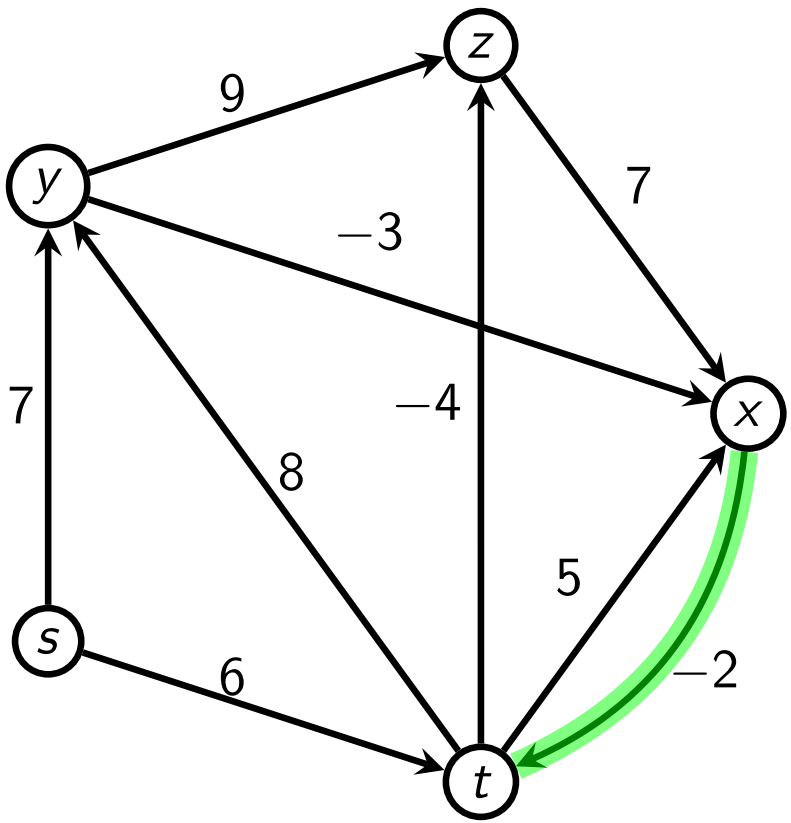
étape	s	t	x	y	z
(a)	0	6	4	7	2
(b)	0	6	4	7	2

Déroulement de l'algorithme de Bellman-Ford

On examine les arcs dans l'ordre lexicographique :

$$(s,t),(s,y),(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x).$$

Exemple 4 (source = s)



Les coulisses

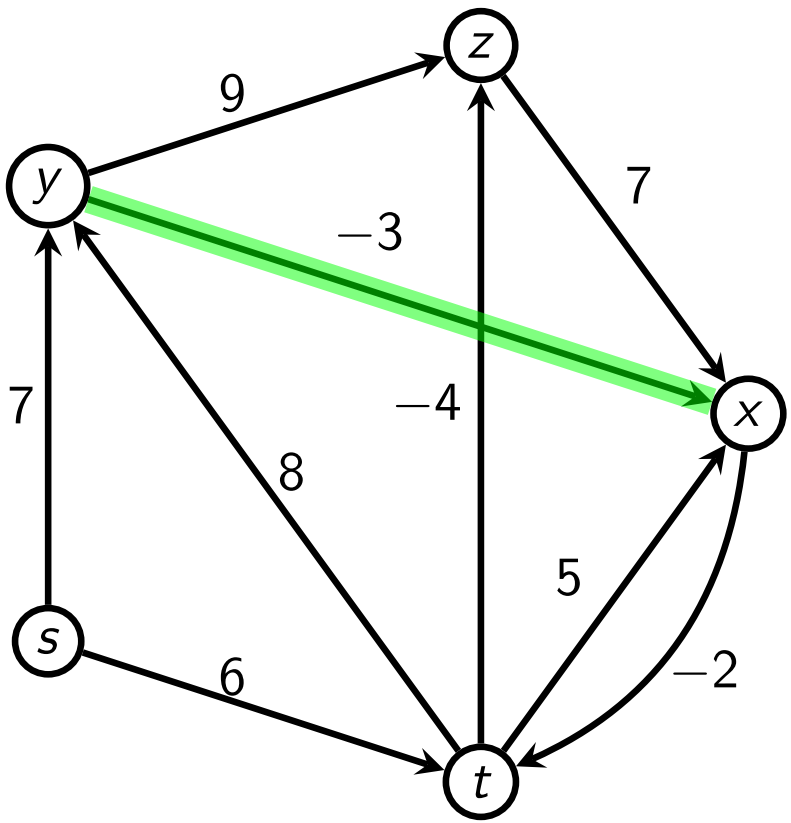
étape	s	t	x	y	z
(a)	0	6	4	7	2
(b)	0	6	4	7	2

Déroulement de l'algorithme de Bellman-Ford

On examine les arcs dans l'ordre lexicographique :

$$(s,t),(s,y),(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x).$$

Exemple 4 (source = s)



Les coulisses

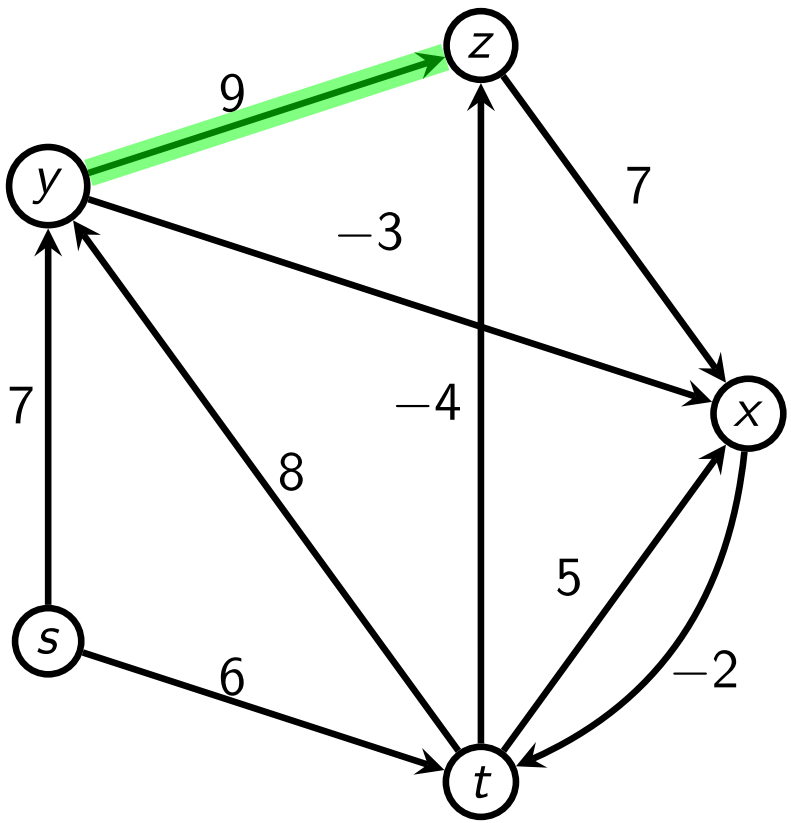
étape	s	t	x	y	z
(a)	0	6	4	7	2
(b)	0	2	4	7	2

Déroulement de l'algorithme de Bellman-Ford

On examine les arcs dans l'ordre lexicographique :

$$(s,t),(s,y),(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x).$$

Exemple 4 (source = s)



Les coulisses

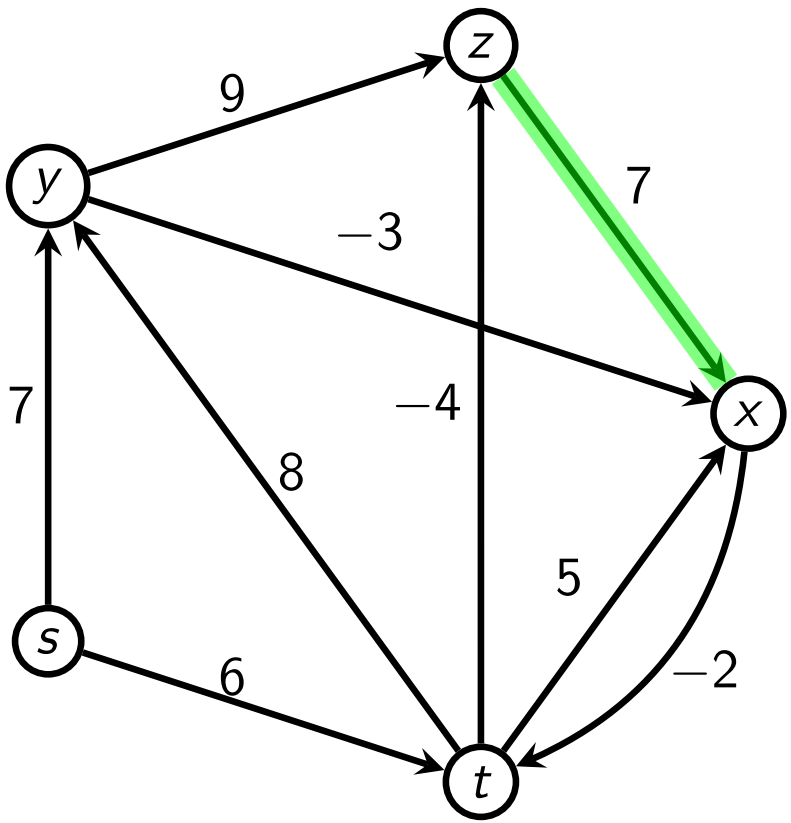
étape	s	t	x	y	z
(a)	0	6	4	7	2
(b)	0	2	4	7	2

Déroulement de l'algorithme de Bellman-Ford

On examine les arcs dans l'ordre lexicographique :

$$(s,t),(s,y),(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x).$$

Exemple 4 (source = s)



Les coulisses

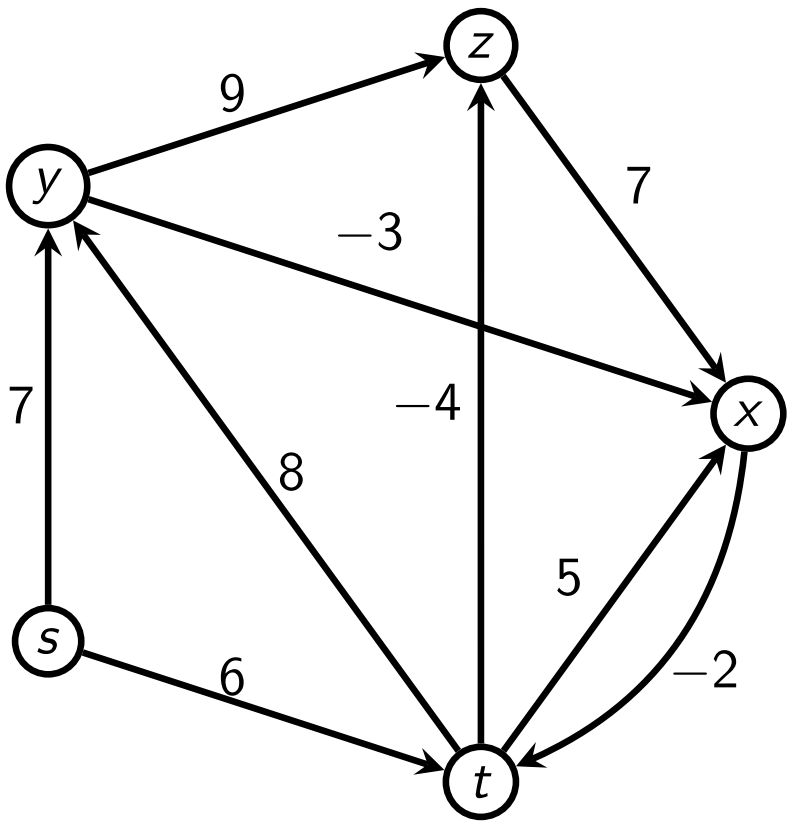
étape	s	t	x	y	z
(a)	0	6	4	7	2
(b)	0	2	4	7	2

Déroulement de l'algorithme de Bellman-Ford

On examine les arcs dans l'ordre lexicographique :

$$(s,t),(s,y),(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x).$$

Exemple 4 (source = s)



Les coulisses

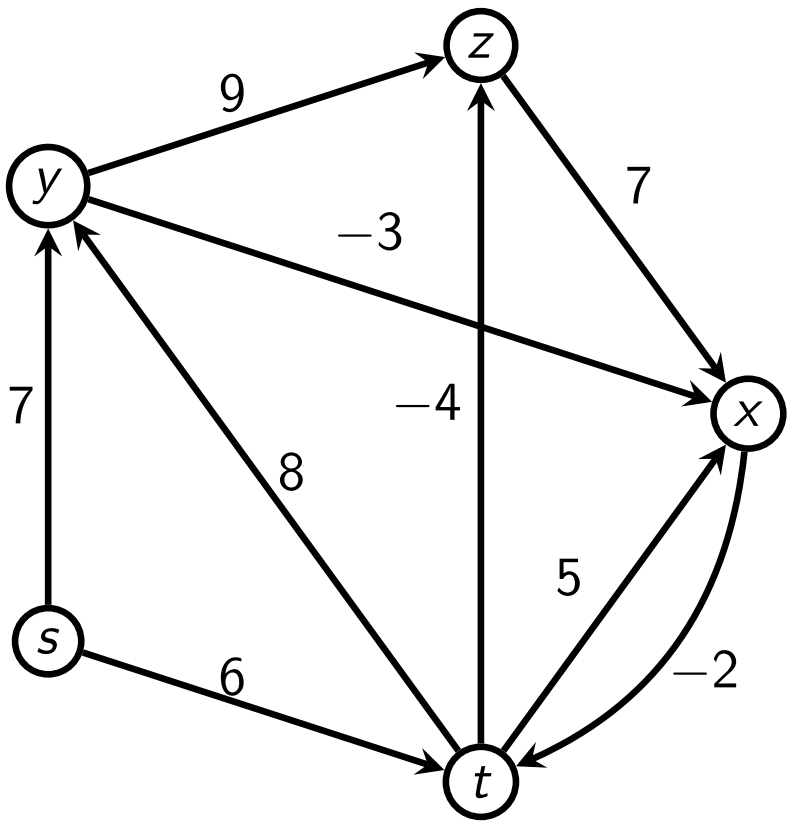
étape	s	t	x	y	z
(a)	0	6	4	7	2
(b)	0	2	4	7	2

Déroulement de l'algorithme de Bellman-Ford

On examine les arcs dans l'ordre lexicographique :

$$(s,t),(s,y),(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x).$$

Exemple 4 (source = s)



Les coulisses

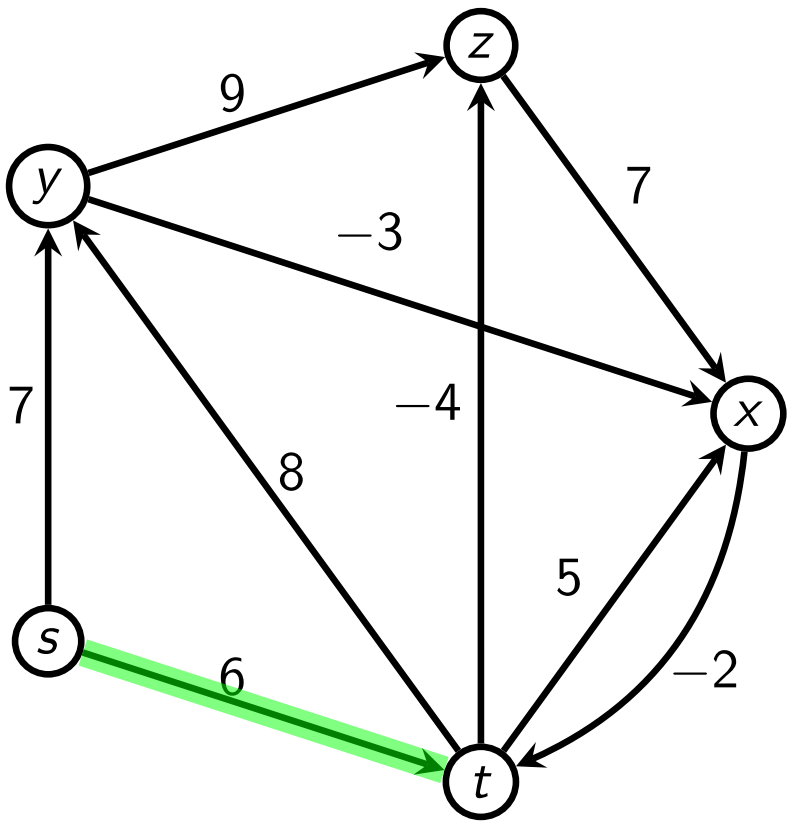
étape	s	t	x	y	z
(a)	0	6	4	7	2
(b)	0	2	4	7	2
(c)	0	2	4	7	2

Déroulement de l'algorithme de Bellman-Ford

On examine les arcs dans l'ordre lexicographique :

$$(s,t),(s,y),(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x).$$

Exemple 4 (source = s)



Les coulisses

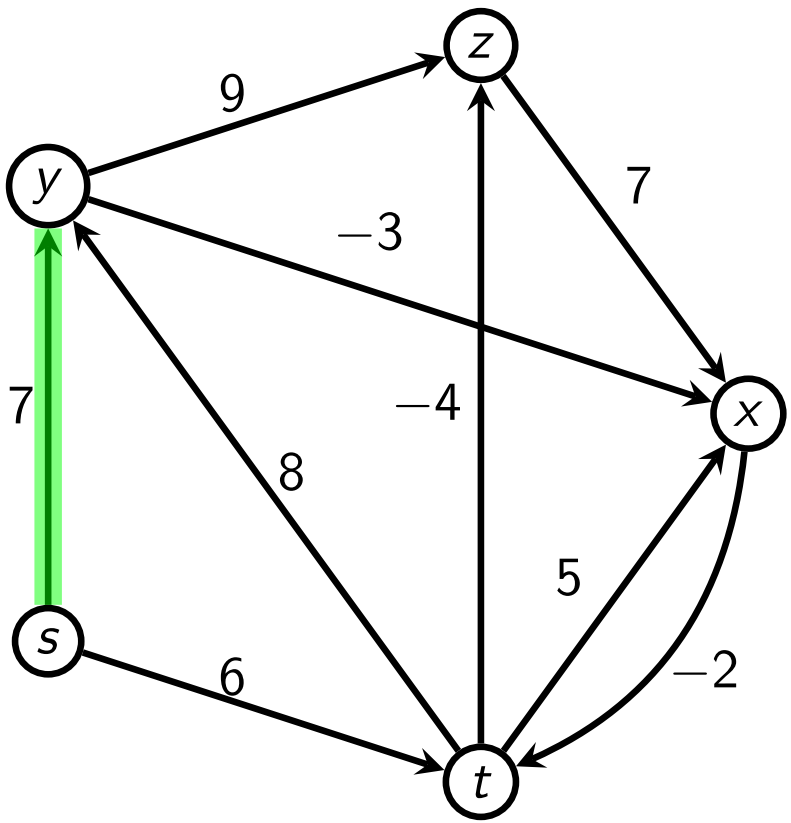
étape	s	t	x	y	z
(a)	0	6	4	7	2
(b)	0	2	4	7	2
(c)	0	2	4	7	2

Déroulement de l'algorithme de Bellman-Ford

On examine les arcs dans l'ordre lexicographique :

$$(s,t),(s,y),(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x).$$

Exemple 4 (source = s)



Les coulisses

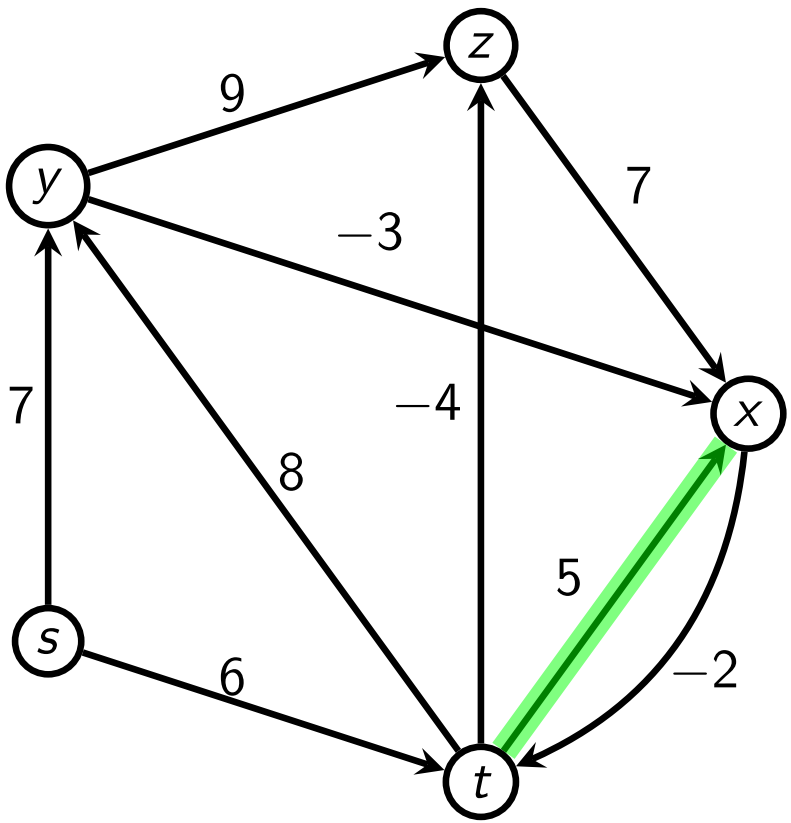
étape	s	t	x	y	z
(a)	0	6	4	7	2
(b)	0	2	4	7	2
(c)	0	2	4	7	2

Déroulement de l'algorithme de Bellman-Ford

On examine les arcs dans l'ordre lexicographique :

$$(s,t),(s,y),(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x).$$

Exemple 4 (source = s)



Les coulisses

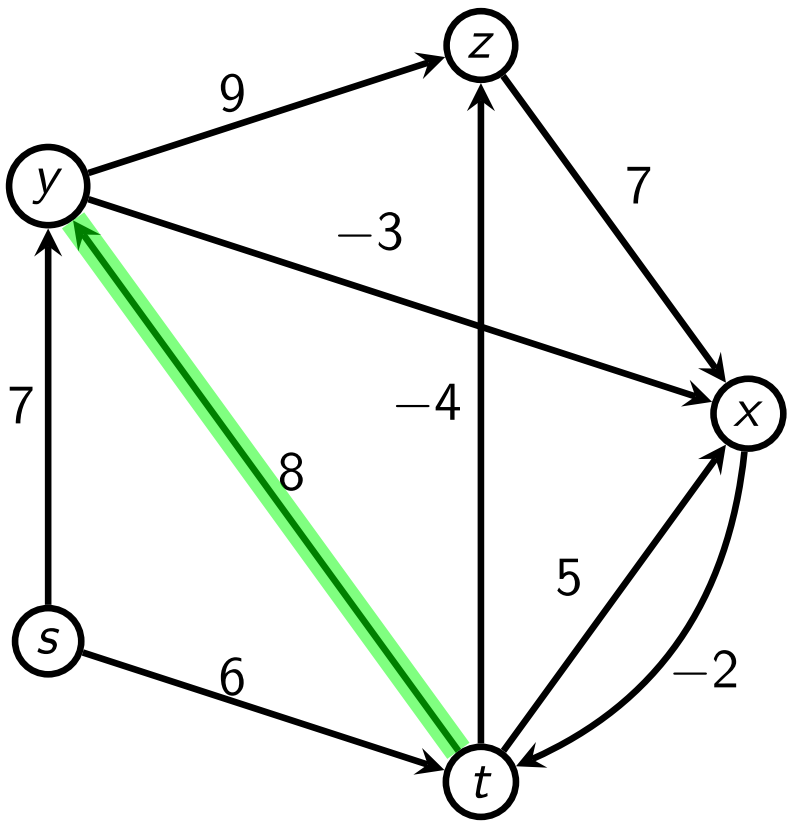
étape	s	t	x	y	z
(a)	0	6	4	7	2
(b)	0	2	4	7	2
(c)	0	2	4	7	2

Déroulement de l'algorithme de Bellman-Ford

On examine les arcs dans l'ordre lexicographique :

$$(s,t),(s,y),(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x).$$

Exemple 4 (source = s)



Les coulisses

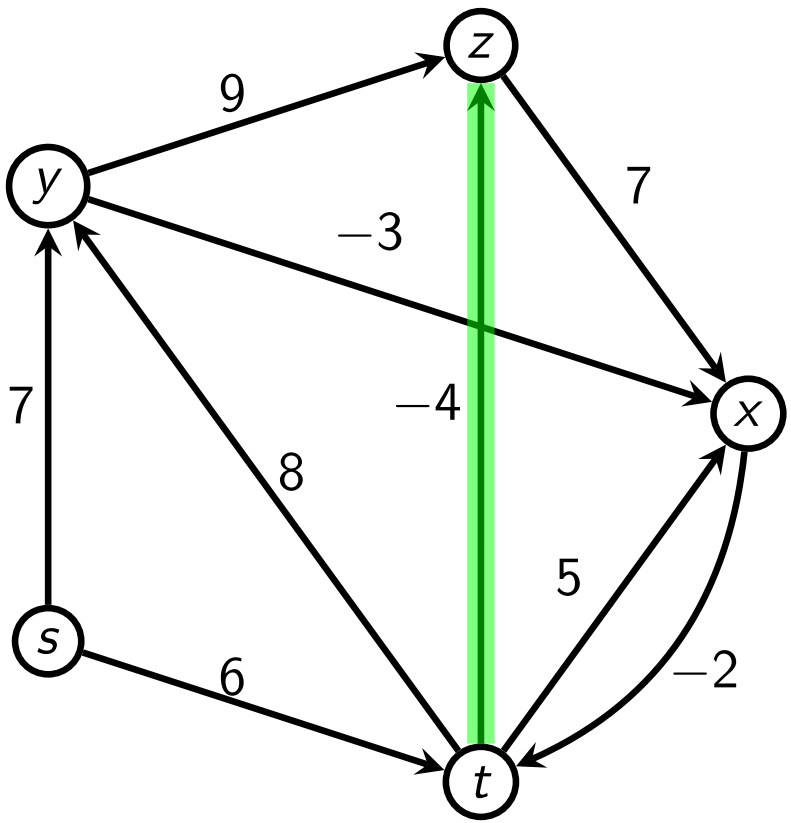
étape	s	t	x	y	z
(a)	0	6	4	7	2
(b)	0	2	4	7	2
(c)	0	2	4	7	2

Déroulement de l'algorithme de Bellman-Ford

On examine les arcs dans l'ordre lexicographique :

$$(s,t),(s,y),(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x).$$

Exemple 4 (source = s)



Les coulisses

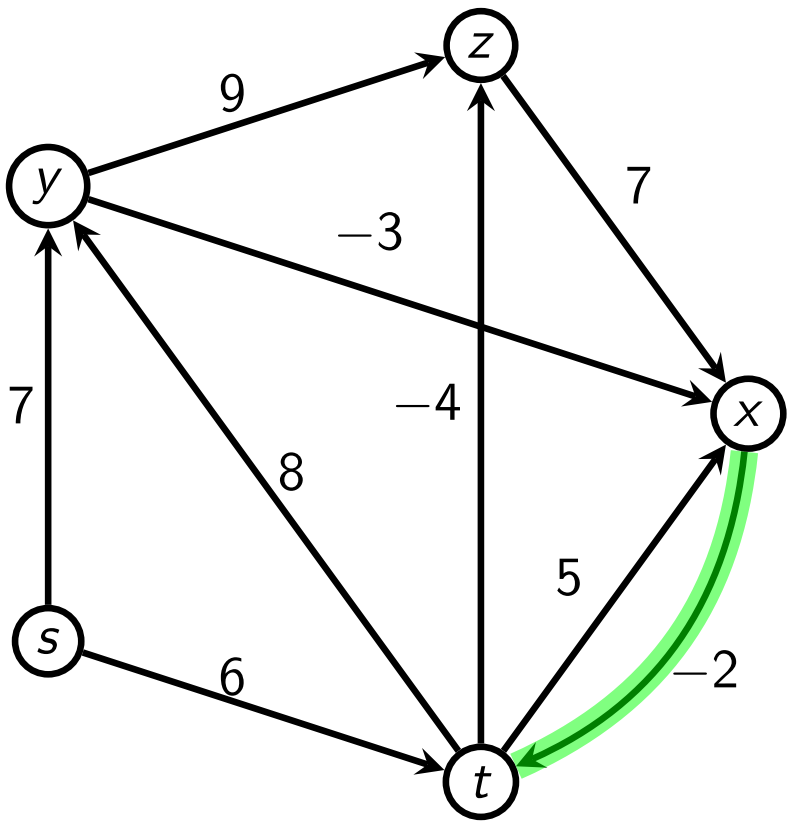
étape	s	t	x	y	z
(a)	0	6	4	7	2
(b)	0	2	4	7	2
(c)	0	2	4	7	2

Déroulement de l'algorithme de Bellman-Ford

On examine les arcs dans l'ordre lexicographique :

$$(s,t),(s,y),(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x).$$

Exemple 4 (source = s)



Les coulisses

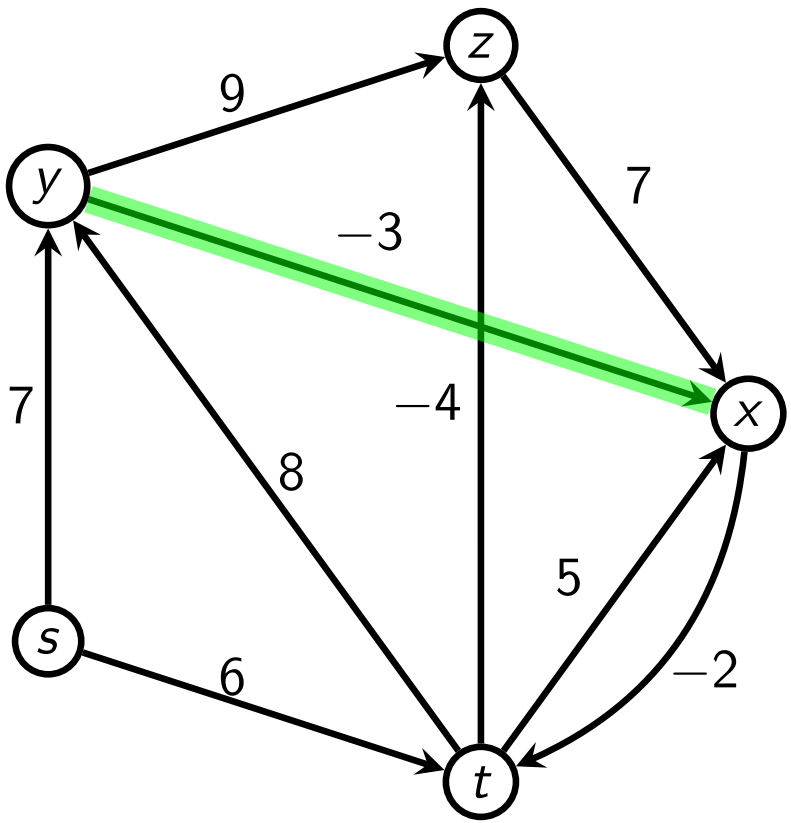
étape	s	t	x	y	z
(a)	0	6	4	7	2
(b)	0	2	4	7	2
(c)	0	2	4	7	-2

Déroulement de l'algorithme de Bellman-Ford

On examine les arcs dans l'ordre lexicographique :

$$(s,t),(s,y),(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x).$$

Exemple 4 (source = s)



Les coulisses

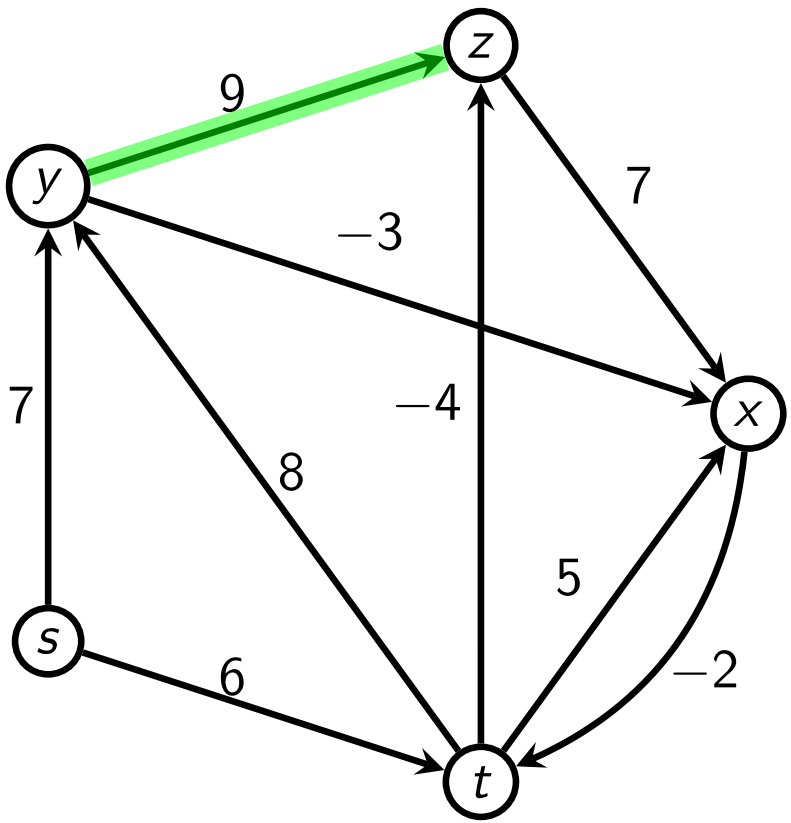
étape	s	t	x	y	z
(a)	0	6	4	7	2
(b)	0	2	4	7	2
(c)	0	2	4	7	-2

Déroulement de l'algorithme de Bellman-Ford

On examine les arcs dans l'ordre lexicographique :

$$(s,t),(s,y),(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x).$$

Exemple 4 (source = s)



Les coulisses

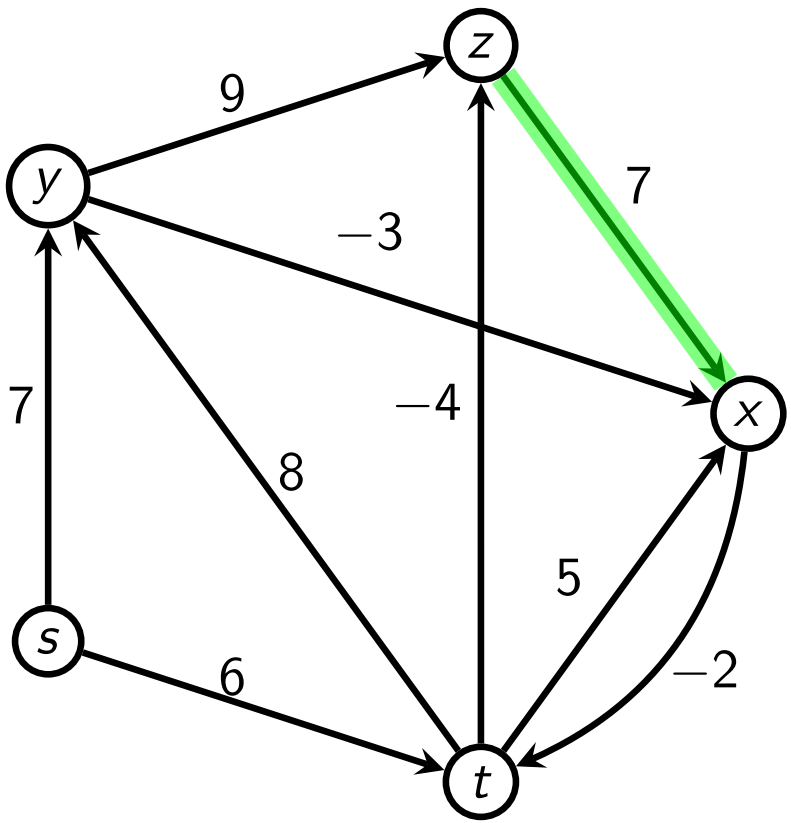
étape	s	t	x	y	z
(a)	0	6	4	7	2
(b)	0	2	4	7	2
(c)	0	2	4	7	-2

Déroulement de l'algorithme de Bellman-Ford

On examine les arcs dans l'ordre lexicographique :

$$(s,t),(s,y),(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x).$$

Exemple 4 (source = s)



Les coulisses

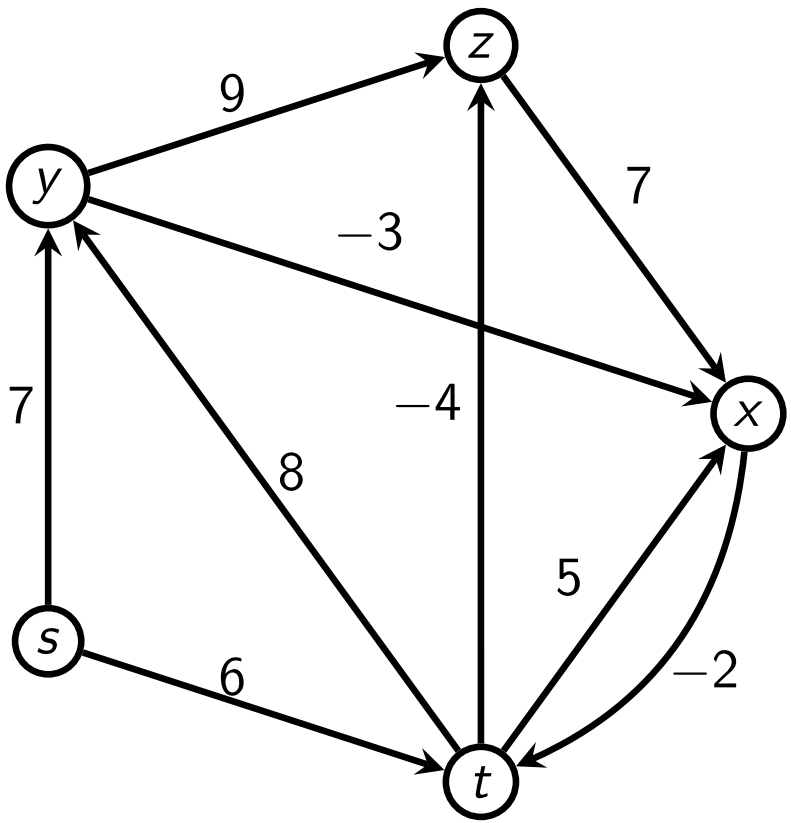
étape	s	t	x	y	z
(a)	0	6	4	7	2
(b)	0	2	4	7	2
(c)	0	2	4	7	-2

Déroulement de l'algorithme de Bellman-Ford

On examine les arcs dans l'ordre lexicographique :

$$(s,t),(s,y),(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x).$$

Exemple 4 (source = s)



Les coulisses

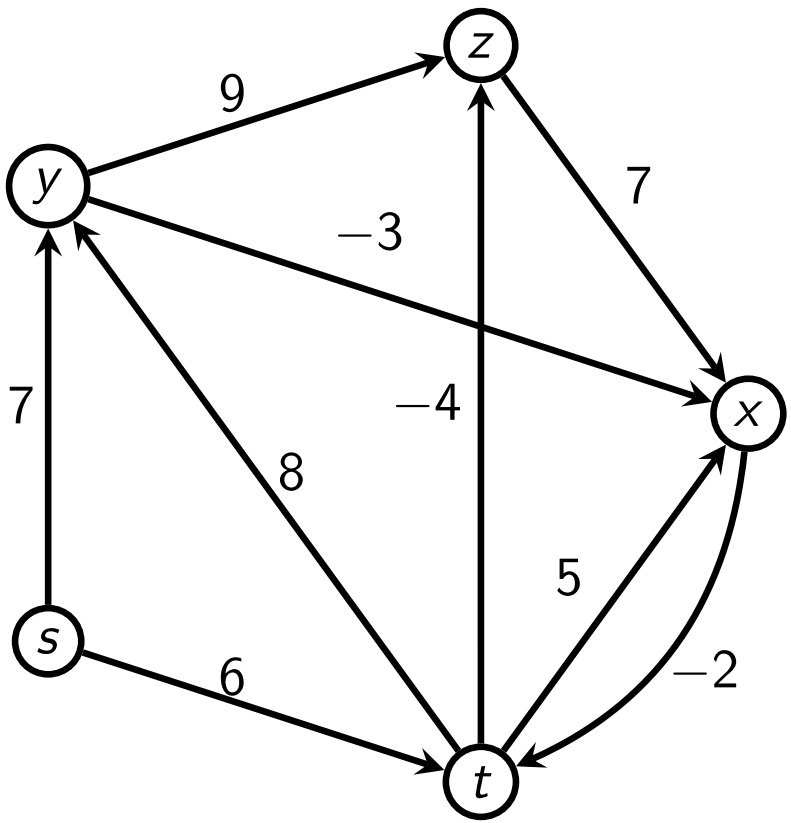
étape	s	t	x	y	z
(a)	0	6	4	7	2
(b)	0	2	4	7	2
(c)	0	2	4	7	-2

Déroulement de l'algorithme de Bellman-Ford

On examine les arcs dans l'ordre lexicographique :

$$(s,t),(s,y),(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x).$$

Exemple 4 (source = s)



Les coulisses

étape	s	t	x	y	z
(a)	0	6	4	7	2
(b)	0	2	4	7	2
(c)	0	2	4	7	-2

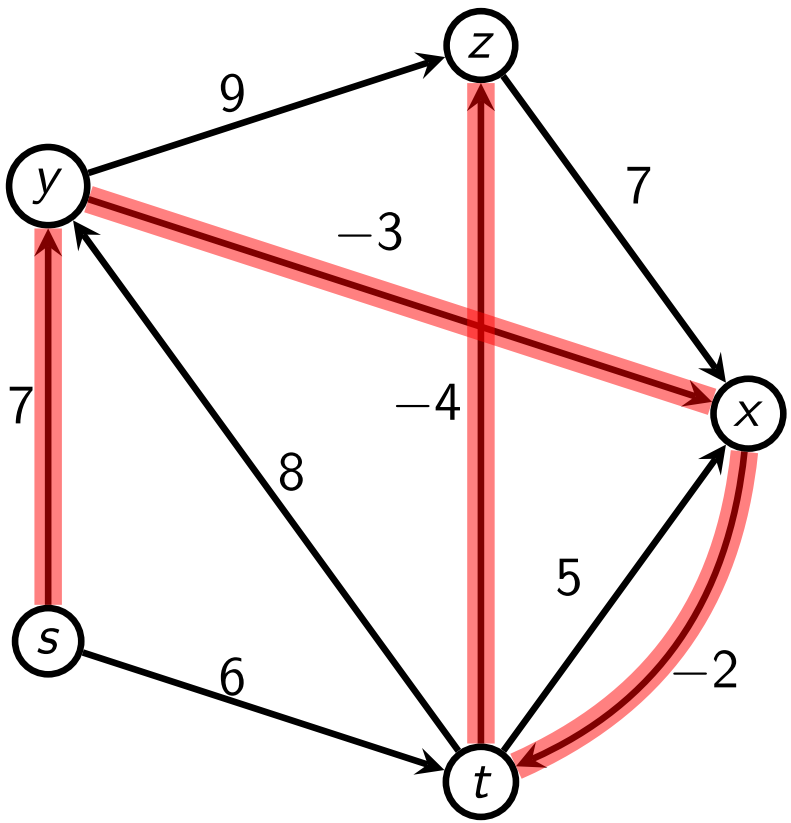
(les étapes suivantes ne changent plus rien)

Déroulement de l'algorithme de Bellman-Ford

On examine les arcs dans l'ordre lexicographique :

$$(s,t),(s,y),(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x).$$

Exemple 4 (source = s)



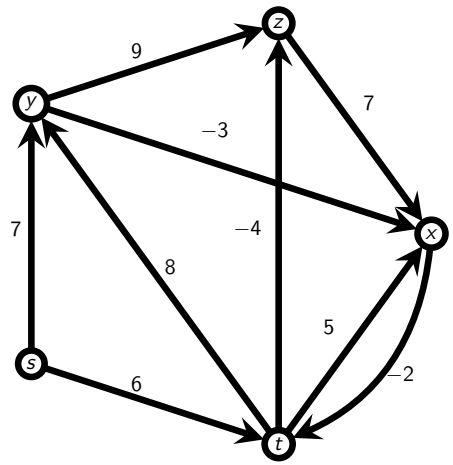
Les coulisses

étape	s	t	x	y	z
(a)	0	6	4	7	2
(b)	0	2	4	7	2
(c)	0	2	4	7	-2

(les étapes suivantes ne changent plus rien)

En "pratique" (enfin, dans les exercices . . .)

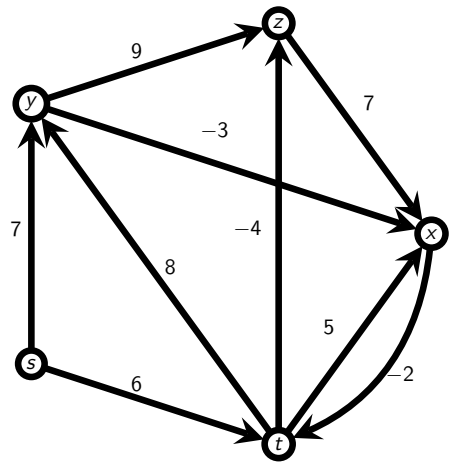
Le déroulement complet étant long, on ne vous demandera généralement les détails que pour une passe sur tous les arcs.



étape	s	t	x	y	z	après traitement des arcs issus de ...
0	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$	(étape finale)

En "pratique" (enfin, dans les exercices . . .)

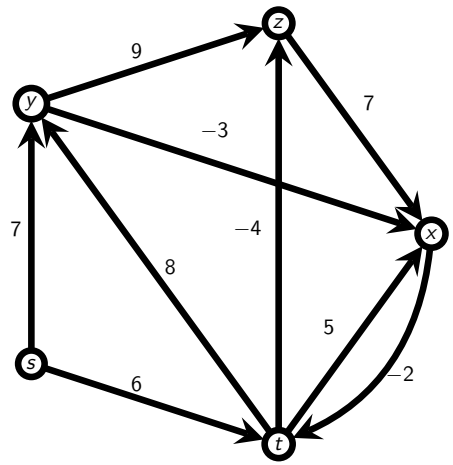
Le déroulement complet étant long, on ne vous demandera généralement les détails que pour une passe sur tous les arcs.



étape	s	t	x	y	z	après traitement des arcs issus de ...
0	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$	(étape finale)
1	0	6	$+\infty$	7	$+\infty$	$s : (s,t), (s,y)$

En "pratique" (enfin, dans les exercices . . .)

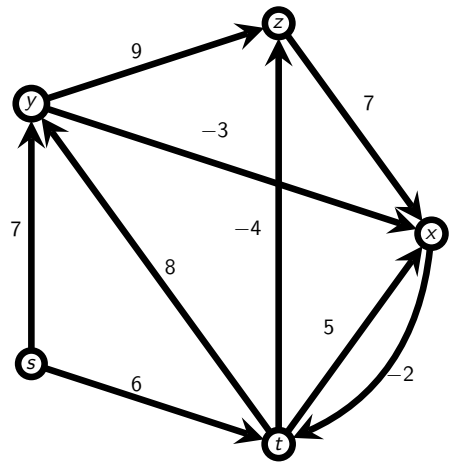
Le déroulement complet étant long, on ne vous demandera généralement les détails que pour une passe sur tous les arcs.



étape	s	t	x	y	z	après traitement des arcs issus de ...
0	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$	(étape finale)
1	0	6	$+\infty$	7	$+\infty$	$s : (s,t), (s,y)$
	0	6	11	7	2	$t : (t,x), (t,y), (t,z)$

En "pratique" (enfin, dans les exercices . . .)

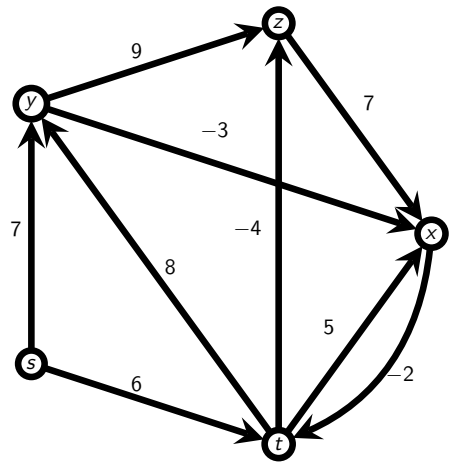
Le déroulement complet étant long, on ne vous demandera généralement les détails que pour une passe sur tous les arcs.



étape	s	t	x	y	z	après traitement des arcs issus de ...
0	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$	(étape finale)
1	0	6	$+\infty$	7	$+\infty$	$s : (s,t), (s,y)$
	0	6	11	7	2	$t : (t,x), (t,y), (t,z)$
	0	6	11	7	2	$x : (x,t)$

En "pratique" (enfin, dans les exercices . . .)

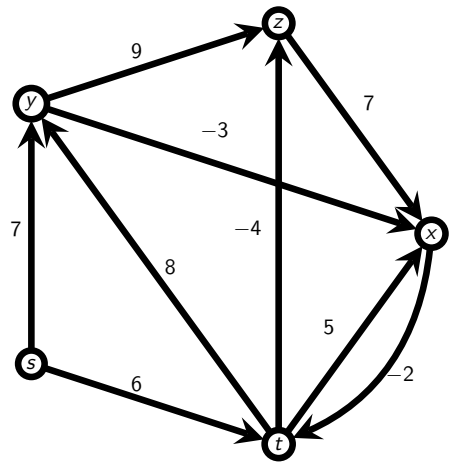
Le déroulement complet étant long, on ne vous demandera généralement les détails que pour une passe sur tous les arcs.



étape	s	t	x	y	z	après traitement des arcs issus de ...
0	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$	(étape finale)
1	0	6	$+\infty$	7	$+\infty$	$s : (s,t), (s,y)$
	0	6	11	7	2	$t : (t,x), (t,y), (t,z)$
	0	6	11	7	2	$x : (x,t)$
	0	6	4	7	2	$y : (y,x), (y,z)$

En "pratique" (enfin, dans les exercices . . .)

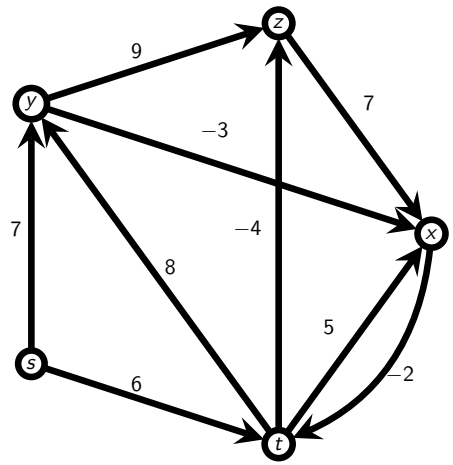
Le déroulement complet étant long, on ne vous demandera généralement les détails que pour une passe sur tous les arcs.



étape	s	t	x	y	z	après traitement des arcs issus de ...
0	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$	(étape finale)
1	0	6	$+\infty$	7	$+\infty$	$s : (s,t), (s,y)$
	0	6	11	7	2	$t : (t,x), (t,y), (t,z)$
	0	6	11	7	2	$x : (x,t)$
	0	6	4	7	2	$y : (y,x), (y,z)$
	0	6	4	7	2	$z : (z,x)$

En "pratique" (enfin, dans les exercices . . .)

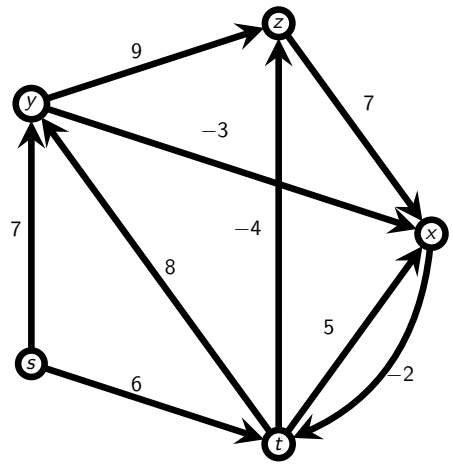
Le déroulement complet étant long, on ne vous demandera généralement les détails que pour une passe sur tous les arcs.



étape	s	t	x	y	z	après traitement des arcs issus de ...
0	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$	(étape finale)
1	0	6	$+\infty$	7	$+\infty$	$s : (s,t), (s,y)$
	0	6	11	7	2	$t : (t,x), (t,y), (t,z)$
	0	6	11	7	2	$x : (x,t)$
	0	6	4	7	2	$y : (y,x), (y,z)$
	0	6	4	7	2	$z : (z,x)$
2	0	2	4	7	2	(étape finale)

En "pratique" (enfin, dans les exercices . . .)

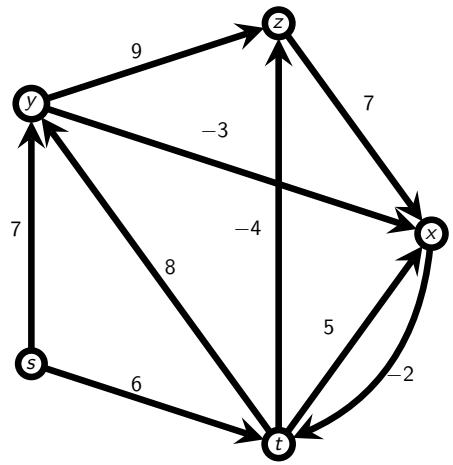
Le déroulement complet étant long, on ne vous demandera généralement les détails que pour une passe sur tous les arcs.



étape	s	t	x	y	z	après traitement des arcs issus de ...
0	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$	(étape finale)
1	0	6	$+\infty$	7	$+\infty$	$s : (s,t), (s,y)$
	0	6	11	7	2	$t : (t,x), (t,y), (t,z)$
	0	6	11	7	2	$x : (x,t)$
	0	6	4	7	2	$y : (y,x), (y,z)$
	0	6	4	7	2	$z : (z,x)$
2	0	2	4	7	2	(étape finale)
3	0	2	4	7	-2	(étape finale)

En "pratique" (enfin, dans les exercices . . .)

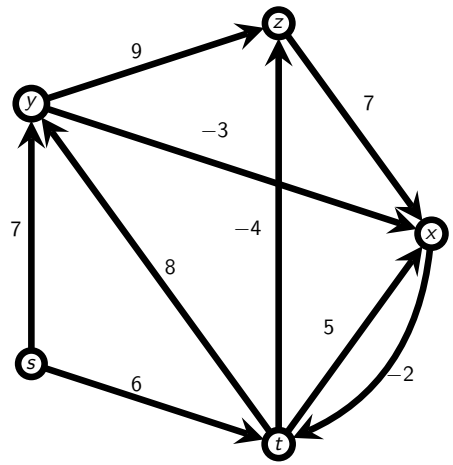
Le déroulement complet étant long, on ne vous demandera généralement les détails que pour une passe sur tous les arcs.



étape	s	t	x	y	z	après traitement des arcs issus de ...
0	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$	(étape finale)
1	0	6	$+\infty$	7	$+\infty$	$s : (s,t), (s,y)$
	0	6	11	7	2	$t : (t,x), (t,y), (t,z)$
	0	6	11	7	2	$x : (x,t)$
	0	6	4	7	2	$y : (y,x), (y,z)$
	0	6	4	7	2	$z : (z,x)$
2	0	2	4	7	2	(étape finale)
3	0	2	4	7	-2	(étape finale)
4	0	2	4	7	-2	(étape finale)

En "pratique" (enfin, dans les exercices . . .)

Le déroulement complet étant long, on ne vous demandera généralement les détails que pour une passe sur tous les arcs.



étape	s	t	x	y	z	après traitement des arcs issus de ...
0	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$	(étape finale)
1	0	6	$+\infty$	7	$+\infty$	$s : (s,t), (s,y)$
	0	6	11	7	2	$t : (t,x), (t,y), (t,z)$
	0	6	11	7	2	$x : (x,t)$
	0	6	4	7	2	$y : (y,x), (y,z)$
	0	6	4	7	2	$z : (z,x)$
2	0	2	4	7	2	(étape finale)
3	0	2	4	7	-2	(étape finale)
4	0	2	4	7	-2	(étape finale)
5	0	2	4	7	-2	(étape finale)

Cycles négatifs

- Un **cycle négatif** C est un cycle tel que $\sum_{a \in A(C)} w(a) < 0$;

Cycles négatifs

- Un **cycle négatif** C est un cycle tel que $\sum_{a \in A(C)} w(a) < 0$;
- Les poids négatifs ne posent pas problème à Bellman-Ford, mais les cycles négatifs oui — pour les mêmes raisons que Dijkstra ;

Cycles négatifs

- Un **cycle négatif** C est un cycle tel que $\sum_{a \in A(C)} w(a) < 0$;
- Les poids négatifs ne posent pas problème à Bellman-Ford, mais les cycles négatifs oui — pour les mêmes raisons que Dijkstra ;
- L'algorithme de Bellman-Ford comprendra un morceau permettant de détecter la présence d'un cycle négatif ;

Intuitions sur Bellman-Ford

- Chaque arc (u, v) a le potentiel d'améliorer le meilleur chemin actuel de s à v en passant par u ;

Intuitions sur Bellman-Ford

- Chaque arc (u, v) a le potentiel d'améliorer le meilleur chemin actuel de s à v en passant par u ;
- Pourquoi parcourir tous les arcs exactement $|V|$ fois ?

Intuitions sur Bellman-Ford

- Chaque arc (u, v) a le potentiel d'améliorer le meilleur chemin actuel de s à v en passant par u ;
- Pourquoi parcourir tous les arcs exactement $|V|$ fois ?
  - s'il existe un plus court chemin entre s et u, il existe un plus court chemin entre s et u qui contient au plus $|V| - 1$ arcs (s'il n'y a pas de cycle de poids négatif) ;

Intuitions sur Bellman-Ford

- Chaque arc (u, v) a le potentiel d'améliorer le meilleur chemin actuel de s à v en passant par u ;
- Pourquoi parcourir tous les arcs exactement $|V|$ fois ?
  - s'il existe un plus court chemin entre s et u, il existe un plus court chemin entre s et u qui contient au plus $|V| - 1$ arcs (s'il n'y a pas de cycle de poids négatif) ;
  - si on arrive à améliorer un chemin contenant le nombre maximal d'arcs, c'est forcément grâce à un cycle de poids négatif.

L'algorithme de Bellman-Ford

Algorithme 4 : BellmanFord(G, source)

Entrées : un graphe pondéré orienté G, un sommet source.
Sortie : la longueur d'un plus court chemin de la source à chacun des sommets du graphe ($+\infty$ pour les sommets non accessibles), ou NIL si le graphe contient un cycle négatif accessible à partir de la source.

```
1 distances ← tableau(G.nombre_sommets(), +∞);
2 distances[source] ← 0;
  // parcourir chaque arc |V| − 1 fois
3 pour i allant de 1 à G.nombre_sommets() −1 faire
4 |   pour chaque (u, v, p) ∈ G.arcs() faire
5 |   |   distances[v] ← min(distances[v], distances[u] + p);
  // vérifier la présence d'un cycle négatif
6 pour chaque (u, v, p) ∈ G.arcs() faire
7 |   si distances[v] > distances[u] + p alors renvoyer NIL;
8 renvoyer distances;
```

Complexité de l'algorithme de Bellman-Ford

- La complexité est assez simple à calculer :

Complexité de l'algorithme de Bellman-Ford

- La complexité est assez simple à calculer :
  - on examine **tous** les arcs $|V|$ fois ;

Complexité de l'algorithme de Bellman-Ford

- La complexité est assez simple à calculer :
  - on examine **tous** les arcs $|V|$ fois ;
  - tous les calculs sont en $O(1)$.

Complexité de l'algorithme de Bellman-Ford

- La complexité est assez simple à calculer :
  - on examine **tous** les arcs $|V|$ fois ;
  - tous les calculs sont en $O(1)$.
- On a donc :

Complexité de l'algorithme de Bellman-Ford

- La complexité est assez simple à calculer :
  - on examine **tous** les arcs $|V|$ fois ;
  - tous les calculs sont en $O(1)$.
- On a donc :
  - du $O(|V|^3)$ pour une matrice d'adjacence ;

Complexité de l'algorithme de Bellman-Ford

- La complexité est assez simple à calculer :
  - on examine **tous** les arcs $|V|$ fois ;
  - tous les calculs sont en $O(1)$.
- On a donc :
  - du $O(|V|^3)$ pour une matrice d'adjacence ;
  - du $O(|V||A|)$ pour une liste d'adjacence.

Complexité de l'algorithme de Bellman-Ford

- La complexité est assez simple à calculer :
  - on examine **tous** les arcs $|V|$ fois ;
  - tous les calculs sont en $O(1)$.
- On a donc :
  - du $O(|V|^3)$ pour une matrice d'adjacence ;
  - du $O(|V||A|)$ pour une liste d'adjacence.
- Remarque : on peut s'arrêter quand l'algorithme "se stabilise", donc quand les estimations ne subissent plus aucun changement.

Dijkstra vs. Bellman-Ford

- Quand utiliser Dijkstra ?

Dijkstra vs. Bellman-Ford

- Quand utiliser Dijkstra ?
  - quand tous les poids sont positifs ou nuls ;

Dijkstra vs. Bellman-Ford

- Quand utiliser Dijkstra ?
  - quand tous les poids sont positifs ou nuls ;
  - sa complexité est meilleure que Bellman-Ford.

Dijkstra vs. Bellman-Ford

- Quand utiliser Dijkstra ?
  - quand tous les poids sont positifs ou nuls ;
  - sa complexité est meilleure que Bellman-Ford.
- Quand utiliser Bellman-Ford ?

Dijkstra vs. Bellman-Ford

- Quand utiliser Dijkstra ?
  - quand tous les poids sont positifs ou nuls ;
  - sa complexité est meilleure que Bellman-Ford.
- Quand utiliser Bellman-Ford ?
  - quand le graphe est a des poids négatifs.

Dijkstra vs. Bellman-Ford

- Quand utiliser Dijkstra ?
  - quand tous les poids sont positifs ou nuls ;
  - sa complexité est meilleure que Bellman-Ford.
- Quand utiliser Bellman-Ford ?
  - quand le graphe est a des poids négatifs.
- S'il y a des cycles négatifs accessibles à partir de la source, il n'y a pas de plus courts chemins à partir de la source.

Motivations

- Jusqu'ici, on s'est intéressés au calcul de tous les plus courts chemins issus d'une source donnée ;

Motivations

- Jusqu'ici, on s'est intéressés au calcul de tous les plus courts chemins issus d'une source donnée ;
- Comment faire pour calculer les plus courts chemins entre chaque paire de sommets ?

Motivations

- Jusqu'ici, on s'est intéressés au calcul de tous les plus courts chemins issus d'une source donnée ;
- Comment faire pour calculer les plus courts chemins entre chaque paire de sommets ?
- On pourrait lancer $|V|$ fois Dijkstra ou Bellman-Ford ;

Motivations

- Jusqu'ici, on s'est intéressés au calcul de tous les plus courts chemins issus d'une source donnée ;
- Comment faire pour calculer les plus courts chemins entre chaque paire de sommets ?
- On pourrait lancer $|V|$ fois Dijkstra ou Bellman-Ford ;
- L'algorithme de Floyd-Warshall permet d'obtenir le résultat voulu avec une meilleure complexité.

L'algorithme de Floyd-Warshall

- L'algorithme de Floyd-Warshall procède comme suit :

L'algorithme de Floyd-Warshall

- L'algorithme de Floyd-Warshall procède comme suit :
  - au départ, les plus courts chemins entre chaque paire de sommets sont les poids des arcs les reliant (ou $+\infty$) ;

L'algorithme de Floyd-Warshall

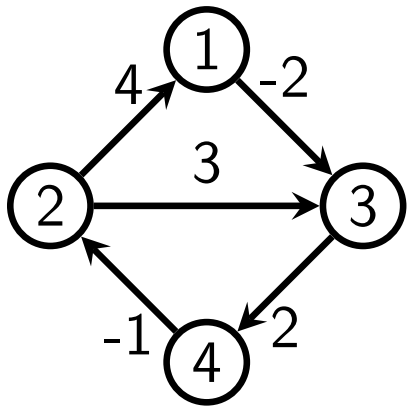
- L'algorithme de Floyd-Warshall procède comme suit :
  - au départ, les plus courts chemins entre chaque paire de sommets sont les poids des arcs les reliant (ou $+\infty$) ;
  - à la $k^{\text{ème}}$ itération, on cherche à améliorer le chemin $u \rightsquigarrow v$ en s'autorisant les **sommets intermédiaires d'indice** 0**,** 1**,** 2**,** ...**,** k pour un certain k fixé.

L'algorithme de Floyd-Warshall

- L'algorithme de Floyd-Warshall procède comme suit :
  - au départ, les plus courts chemins entre chaque paire de sommets sont les poids des arcs les reliant (ou $+\infty$) ;
  - à la $k^{\text{ème}}$ itération, on cherche à améliorer le chemin $u \rightsquigarrow v$ en s'autorisant les **sommets intermédiaires d'indice** 0**,** 1**,** 2**,** ...**,** k pour un certain k fixé.
- Remarque : la seule modification à l'étape k consiste à vérifier si le chemin $u \rightsquigarrow k \rightsquigarrow v$ est plus court que le chemin $u \rightsquigarrow v$, puisque les sommets d'indices inférieurs ont déjà été utilisés.

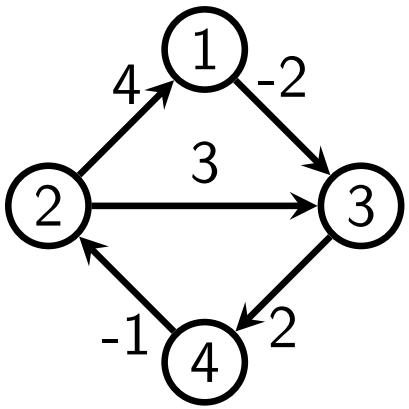
Déroulement de l'algorithme de Floyd-Warshall

Exemple 5



Déroulement de l'algorithme de Floyd-Warshall

Exemple 5

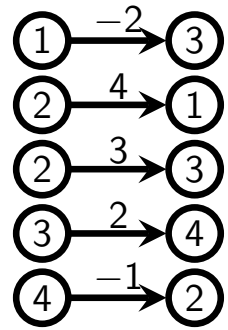


	1	2	3	4
1	0	$+\infty$	-2	$+\infty$
2	4	0	3	$+\infty$
3	$+\infty$	$+\infty$	0	2
4	$+\infty$	-1	$+\infty$	0

matrice de distances

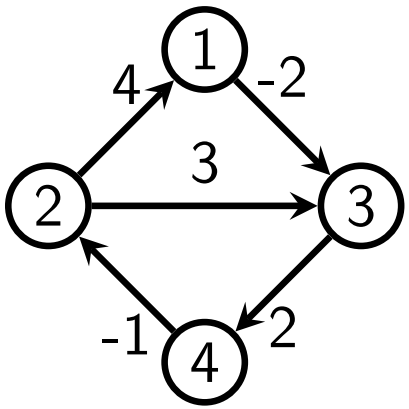
Les chemins

$k = 0$



Déroulement de l'algorithme de Floyd-Warshall

Exemple 5

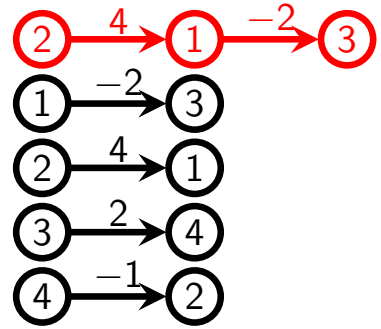


	1	2	3	4
1	0	$+\infty$	-2	$+\infty$
2	4	0	3	$+\infty$
3	$+\infty$	$+\infty$	0	2
4	$+\infty$	-1	$+\infty$	0

matrice de distances

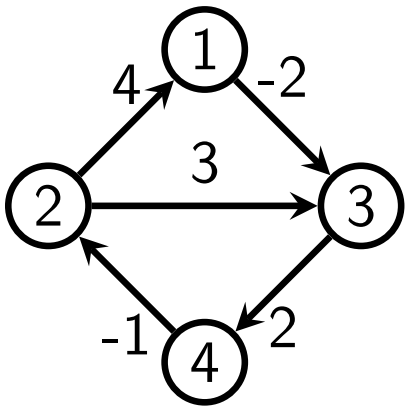
Les chemins

$k = 1$



Déroulement de l'algorithme de Floyd-Warshall

Exemple 5

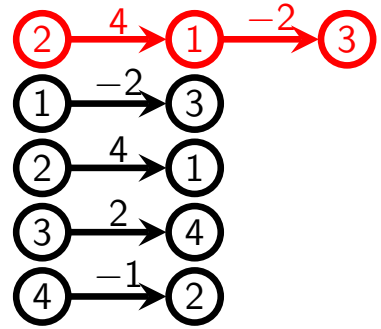


	1	2	3	4
1	0	$+\infty$	-2	$+\infty$
2	4	0	2	$+\infty$
3	$+\infty$	$+\infty$	0	2
4	$+\infty$	-1	$+\infty$	0

matrice de distances

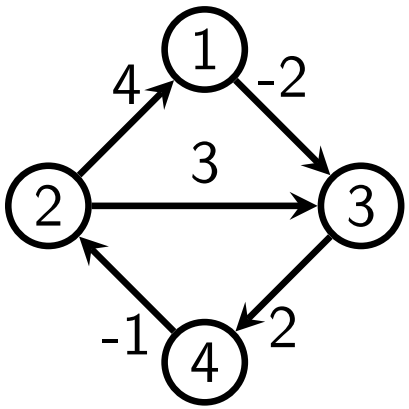
Les chemins

$k = 1$



Déroulement de l'algorithme de Floyd-Warshall

Exemple 5

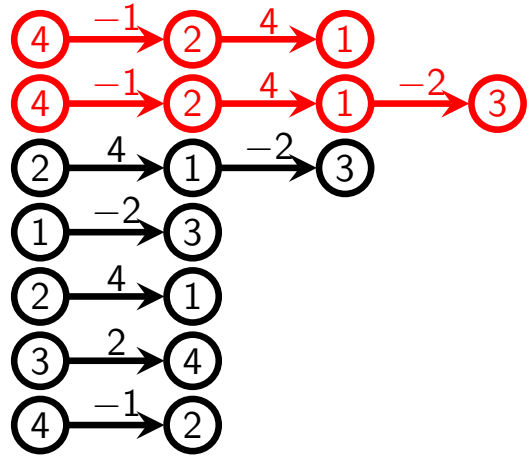


	1	2	3	4
1	0	$+\infty$	-2	$+\infty$
2	4	0	2	$+\infty$
3	$+\infty$	$+\infty$	0	2
4	$+\infty$	-1	$+\infty$	0

matrice de distances

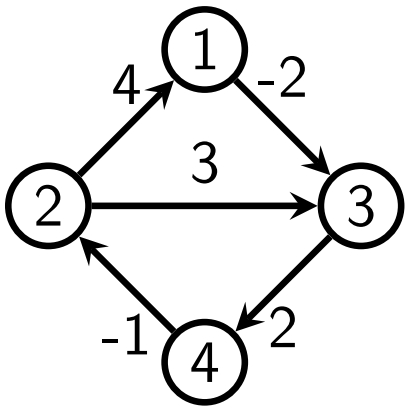
Les chemins

$k = 2$



Déroulement de l'algorithme de Floyd-Warshall

Exemple 5

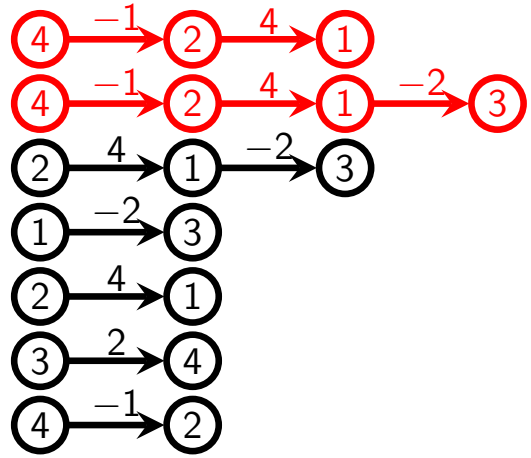


	1	2	3	4
1	0	$+\infty$	-2	$+\infty$
2	4	0	2	$+\infty$
3	$+\infty$	$+\infty$	0	2
4	3	-1	1	0

matrice de distances

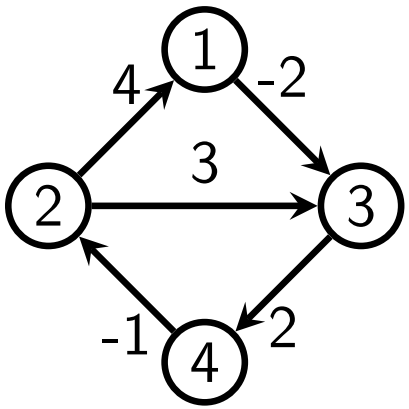
Les chemins

$k = 2$



Déroulement de l'algorithme de Floyd-Warshall

Exemple 5

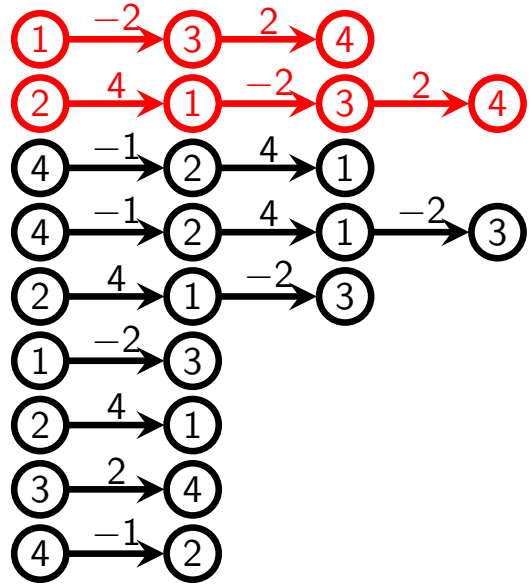


	1	2	3	4
1	0	$+\infty$	-2	$+\infty$
2	4	0	2	$+\infty$
3	$+\infty$	$+\infty$	0	2
4	3	-1	1	0

matrice de distances

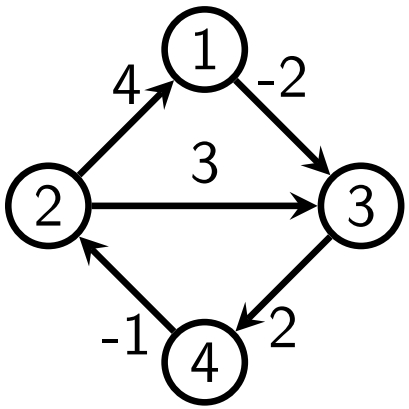
Les chemins

$k = 3$



Déroulement de l'algorithme de Floyd-Warshall

Exemple 5

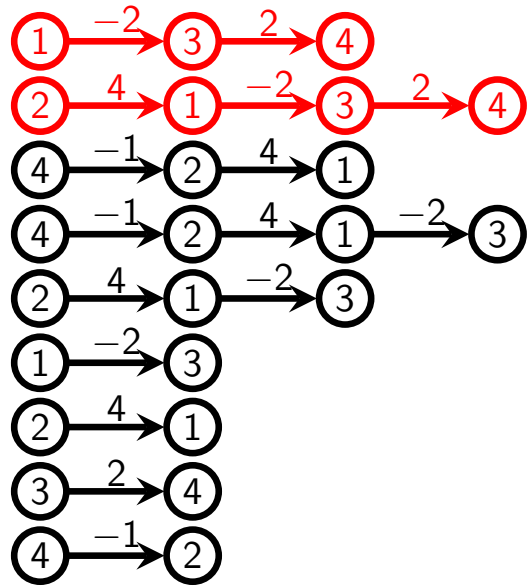


	1	2	3	4
1	0	$+\infty$	-2	0
2	4	0	2	4
3	$+\infty$	$+\infty$	0	2
4	3	-1	1	0

matrice de distances

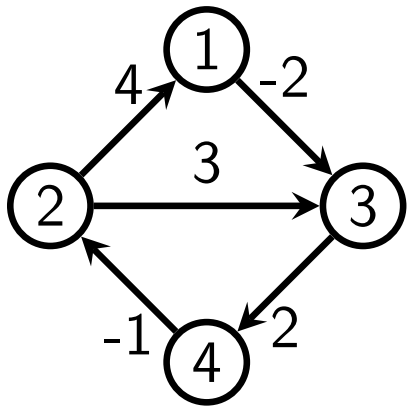
Les chemins

$k = 3$



Déroulement de l'algorithme de Floyd-Warshall

Exemple 5

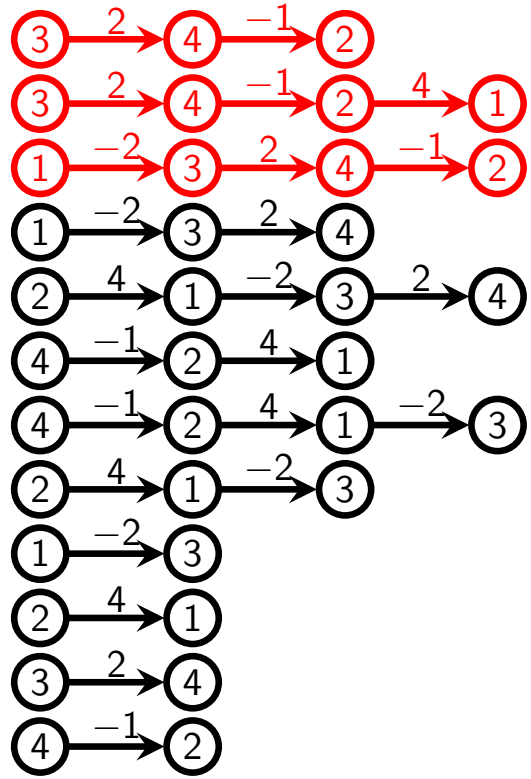


	1	2	3	4
1	0	$+\infty$	-2	0
2	4	0	2	4
3	$+\infty$	$+\infty$	0	2
4	3	-1	1	0

matrice de distances

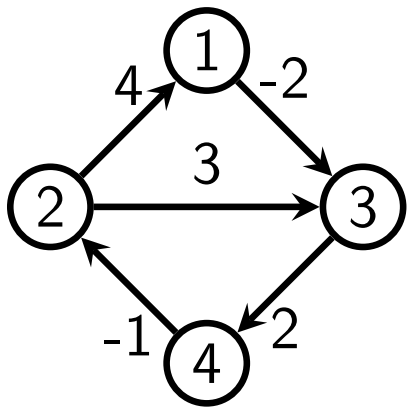
Les chemins

$k = 4$



Déroulement de l'algorithme de Floyd-Warshall

Exemple 5

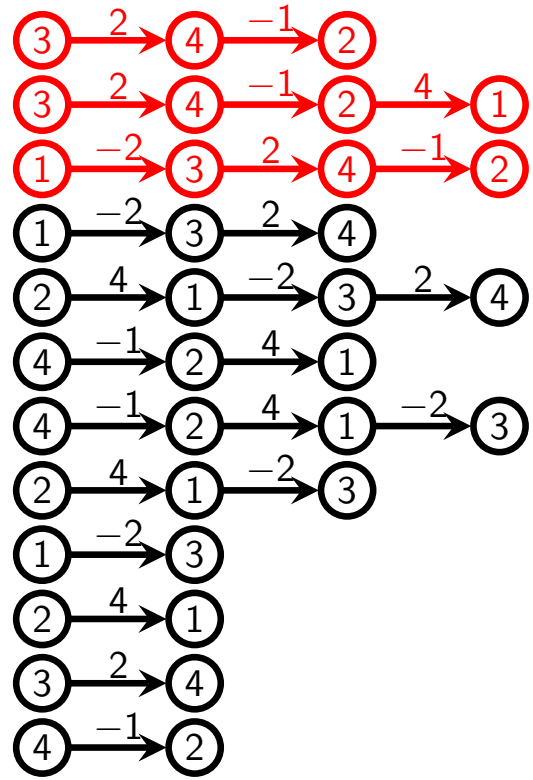


	1	2	3	4
1	0	−1	−2	0
2	4	0	2	4
3	5	1	0	2
4	3	−1	1	0

matrice de distances

Les chemins

$k = 4$



L'algorithme de Floyd-Warshall

Si seules les distances nous intéressent, l'algorithme est assez simple à implémenter, car peu d'accès au graphe sont nécessaires.

Algorithme 5 : FloydWarshall(G)

Entrées : un graphe orienté pondéré G.
Sortie : les distances entre toute paire de sommets du graphe.

```
1  n ← G.nombre_sommets();
2  distances ← matrice(n, n, +∞);
3  pour i allant de 0 à n − 1 faire
4  |   distances[i][i] ← 0;
5  pour chaque (u, v, p) ∈ G.arcs() faire
6  |   distances[u][v] ← p;
   // chercher les améliorations en passant par k = 0, 1, 2, ...
7  pour k allant de 0 à n − 1 faire
8  |   pour i allant de 0 à n − 1 faire
9  |   |   pour j allant de 0 à n − 1 faire
10 |   |   |   distances[i][j] ← min(distances[i][j],
   |   |   |     distances[i][k]+distances[k][j]);
11 renvoyer distances;
```

Complexité de l'algorithme de Floyd-Warshall

- La complexité est assez simple à calculer :

Complexité de l'algorithme de Floyd-Warshall

- La complexité est assez simple à calculer :
  - on accède une fois à tous les arcs ;

Complexité de l'algorithme de Floyd-Warshall

- La complexité est assez simple à calculer :
  - on accède une fois à tous les arcs ;
  - on a trois boucles imbriquées indépendantes comportant chacune $|V|$ itérations ;

Complexité de l'algorithme de Floyd-Warshall

- La complexité est assez simple à calculer :
  - on accède une fois à tous les arcs ;
  - on a trois boucles imbriquées indépendantes comportant chacune $|V|$ itérations ;
  - les opérations sur la matrice de distances se font en $O(1)$;

Complexité de l'algorithme de Floyd-Warshall

- La complexité est assez simple à calculer :
  - on accède une fois à tous les arcs ;
  - on a trois boucles imbriquées indépendantes comportant chacune $|V|$ itérations ;
  - les opérations sur la matrice de distances se font en $O(1)$;
- $\Rightarrow$ total : $O(|V|^3)$;

Complexité de l'algorithme de Floyd-Warshall

- La complexité est assez simple à calculer :
  - on accède une fois à tous les arcs ;
  - on a trois boucles imbriquées indépendantes comportant chacune $|V|$ itérations ;
  - les opérations sur la matrice de distances se font en $O(1)$;
- $\Rightarrow$ total : $O(|V|^3)$;
- Dans ce cas précis, la représentation du graphe importe peu : qu'on obtienne les arcs en $O(|V|^2)$ (matrice) ou en $O(|A|)$ (listes), c'est le $O(|V|^3)$ qui domine ;

Correction de l'algorithme de Floyd-Warshall

- À la fin de l'étape k, distances$[u][v]$ contient le poids d'un plus court chemin de u à v dont les sommets intermédiaires ont des numéros entre 0 et k.

Correction de l'algorithme de Floyd-Warshall

- À la fin de l'étape k, distances$[u][v]$ contient le poids d'un plus court chemin de u à v dont les sommets intermédiaires ont des numéros entre 0 et k.
    - En effet supposons que ce soit vrai par récurrence jusqu'à l'étape $k-1$. Soit P un plus court chemin de u à v dont les sommets intermédiaires ont des numéros entre 0 et k.

Correction de l'algorithme de Floyd-Warshall

- À la fin de l'étape k, distances$[u][v]$ contient le poids d'un plus court chemin de u à v dont les sommets intermédiaires ont des numéros entre 0 et k.
  - En effet supposons que ce soit vrai par récurrence jusqu'à l'étape $k-1$. Soit P un plus court chemin de u à v dont les sommets intermédiaires ont des numéros entre 0 et k.
  - Si P passe par k, $P = u \rightsquigarrow k \rightsquigarrow k \cdots \rightsquigarrow k \rightsquigarrow v$,
    avec les numéros intermédiaires de chaque tronçon entre 0 et $k-1$.

Correction de l'algorithme de Floyd-Warshall

- À la fin de l'étape k, distances$[u][v]$ contient le poids d'un plus court chemin de u à v dont les sommets intermédiaires ont des numéros entre 0 et k.
  - En effet supposons que ce soit vrai par récurrence jusqu'à l'étape $k-1$. Soit P un plus court chemin de u à v dont les sommets intermédiaires ont des numéros entre 0 et k.
  - Si P passe par k, $P = u \rightsquigarrow k \rightsquigarrow k \cdots \rightsquigarrow k \rightsquigarrow v$, avec les numéros intermédiaires de chaque tronçon entre 0 et $k-1$.
  - Comme il n'existe pas de cycle de poids négatif, on coupe dans P les morceaux allant de k à k pour avoir un autre plus court chemin P' qui va de u à k avec des sommets intermédiaires dans 0 et $k-1$ puis de k à v avec des sommets intermédiaires dans 0 et $k-1$. Son poids est donc distances_etape_$k-1$$[u][k]$ + distances_etape_$k-1$$[k][v]$.

Correction de l'algorithme de Floyd-Warshall

- À la fin de l'étape k, distances$[u][v]$ contient le poids d'un plus court chemin de u à v dont les sommets intermédiaires ont des numéros entre 0 et k.
  - En effet supposons que ce soit vrai par récurrence jusqu'à l'étape $k-1$. Soit P un plus court chemin de u à v dont les sommets intermédiaires ont des numéros entre 0 et k.
  - Si P passe par k, $P = u \rightsquigarrow k \rightsquigarrow k \cdots \rightsquigarrow k \rightsquigarrow v$, avec les numéros intermédiaires de chaque tronçon entre 0 et $k-1$.
  - Comme il n'existe pas de cycle de poids négatif, on coupe dans P les morceaux allant de k à k pour avoir un autre plus court chemin P' qui va de u à k avec des sommets intermédiaires dans 0 et $k-1$ puis de k à v avec des sommets intermédiaires dans 0 et $k-1$. Son poids est donc distances_etape_$k-1$$[u][k]$ + distances_etape_$k-1$$[k][v]$.
  - Si P ne passe pas par k, son poids est distances_etape_$k-1$$[u][v]$.

Correction de l'algorithme de Floyd-Warshall suite

- Si les sommets sont numérotés de 0 à $n-1$, à la fin distances[u][v] contient donc le poids d'un plus court chemin de u à v.