Algorithmique des graphes

6 — Plus courts chemins, la suite

Anthony Labarre

10 mars 2021

Université Gustave Eiffel

Correction de l'algorithme de Kosaraju-Sharir

- Pour prouver que l'algorithme de Kosaraju-Sharir est correct, il est plus simple de s'imaginer que chaque CFC est supprimée du graphe dès qu'on l'identifie ;

Correction de l'algorithme de Kosaraju-Sharir

- Pour prouver que l'algorithme de Kosaraju-Sharir est correct, il est plus simple de s'imaginer que chaque CFC est supprimée du graphe dès qu'on l'identifie ;
- Le résultat suivant sera crucial :

Correction de l'algorithme de Kosaraju-Sharir

- Pour prouver que l'algorithme de Kosaraju-Sharir est correct, il est plus simple de s'imaginer que chaque CFC est supprimée du graphe dès qu'on l'identifie ;
- Le résultat suivant sera crucial :

Lemme 1

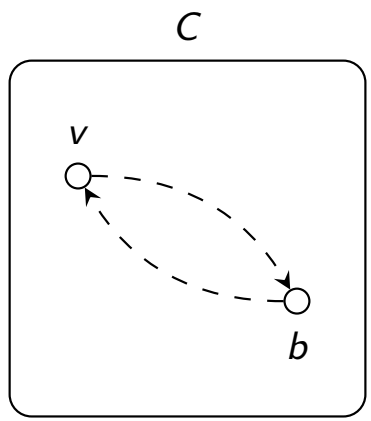
Soit v le sommet de G dont la date de fin d'exploration est maximale. Alors la CFC C de G contenant v est une source, c'est-à-dire qu'il n'existe pas d'arc (a, b) avec $a \notin C$ et $b \in C$.

Correction de l'algorithme de Kosaraju-Sharir

Soit v le sommet de G dont la date de fin d'exploration est maximale. Alors la CFC C de G contenant v est une source, c'est-à-dire qu'il n'existe pas d'arc (a, b) avec $a \notin C$ et $b \in C$.

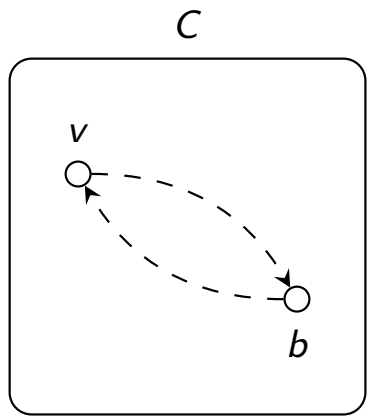
Correction de l'algorithme de Kosaraju-Sharir

Soit v le sommet de G dont la date de fin d'exploration est maximale. Alors la CFC C de G contenant v est une source, c'est-à-dire qu'il n'existe pas d'arc (a, b) avec $a \notin C$ et $b \in C$.



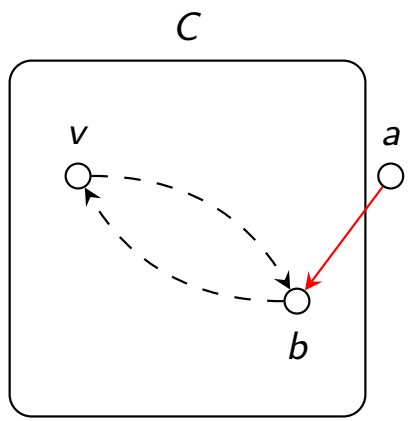
Correction de l'algorithme de Kosaraju-Sharir

Soit v le sommet de G dont la date de fin d'exploration est maximale. Alors la CFC C de G contenant v est une source, c'est-à-dire qu'il n'existe pas d'arc (a, b) avec $a \notin C$ et $b \in C$. Procédons par contradiction : on suppose



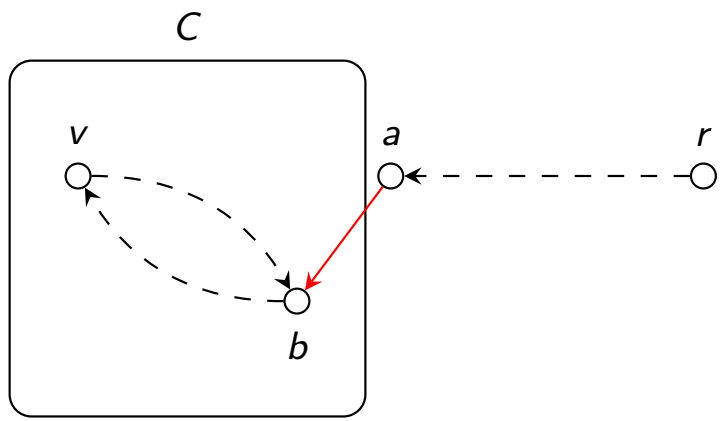
Correction de l'algorithme de Kosaraju-Sharir

Soit v le sommet de G dont la date de fin d'exploration est maximale. Alors la CFC C de G contenant v est une source, c'est-à-dire qu'il n'existe pas d'arc (a, b) avec $a \notin C$ et $b \in C$. Procédons par contradiction : on suppose qu'il existe un arc (a, b) avec $a \notin C$ et $b \in C$.



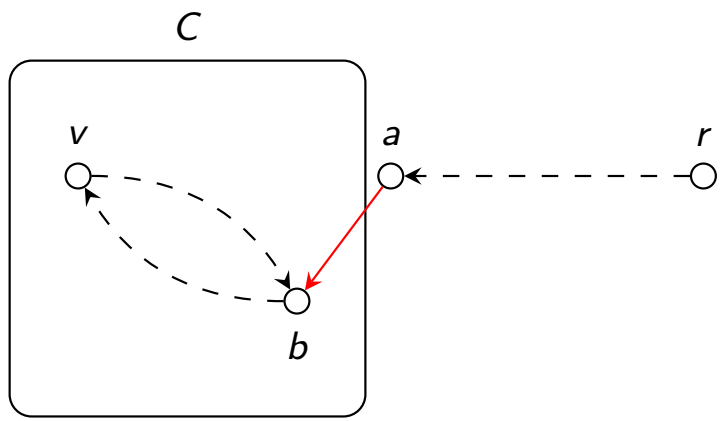
Correction de l'algorithme de Kosaraju-Sharir

Soit v le sommet de G dont la date de fin d'exploration est maximale. Alors la CFC C de G contenant v est une source, c'est-à-dire qu'il n'existe pas d'arc (a, b) avec $a \notin C$ et $b \in C$. Procédons par contradiction : on suppose qu'il existe un arc (a, b) avec $a \notin C$ et $b \in C$. Soit r le sommet à partir duquel l'exploration en profondeur mène au sommet a ; on sait que $r \notin C$, sinon on aurait $a \in C$; donc $r \neq v$.



Correction de l'algorithme de Kosaraju-Sharir

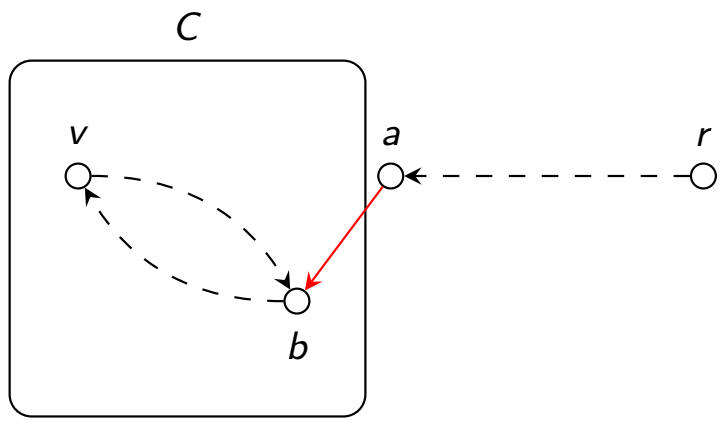
Soit v le sommet de G dont la date de fin d'exploration est maximale. Alors la CFC C de G contenant v est une source, c'est-à-dire qu'il n'existe pas d'arc (a, b) avec $a \notin C$ et $b \in C$. Procédons par contradiction : on suppose qu'il existe un arc (a, b) avec $a \notin C$ et $b \in C$. Soit r le sommet à partir duquel l'exploration en profondeur mène au sommet a ; on sait que $r \notin C$, sinon on aurait $a \in C$; donc $r \neq v$.



Il existe un chemin $r \rightsquigarrow a \to b \rightsquigarrow v$;

Correction de l'algorithme de Kosaraju-Sharir

Soit v le sommet de G dont la date de fin d'exploration est maximale. Alors la CFC C de G contenant v est une source, c'est-à-dire qu'il n'existe pas d'arc (a, b) avec $a \notin C$ et $b \in C$. Procédons par contradiction : on suppose qu'il existe un arc (a, b) avec $a \notin C$ et $b \in C$. Soit r le sommet à partir duquel l'exploration en profondeur mène au sommet a ; on sait que $r \notin C$, sinon on aurait $a \in C$; donc $r \neq v$.

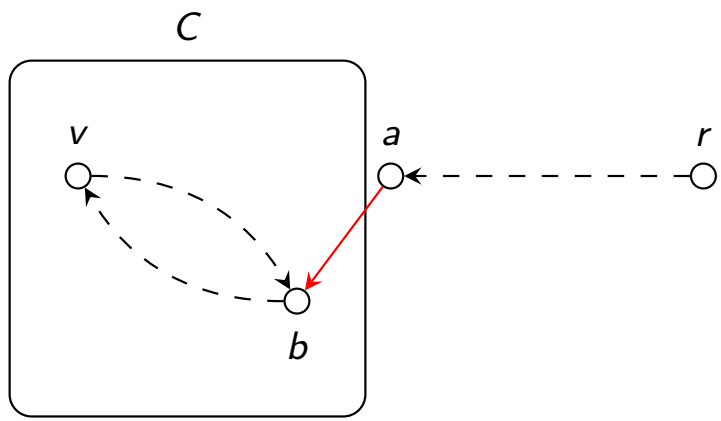


Il existe un chemin $r \rightsquigarrow a \to b \rightsquigarrow v$;

$\Rightarrow$ on finit d'explorer v avant r

Correction de l'algorithme de Kosaraju-Sharir

Soit v le sommet de G dont la date de fin d'exploration est maximale. Alors la CFC C de G contenant v est une source, c'est-à-dire qu'il n'existe pas d'arc (a, b) avec $a \notin C$ et $b \in C$. Procédons par contradiction : on suppose qu'il existe un arc (a, b) avec $a \notin C$ et $b \in C$. Soit r le sommet à partir duquel l'exploration en profondeur mène au sommet a ; on sait que $r \notin C$, sinon on aurait $a \in C$; donc $r \neq v$.

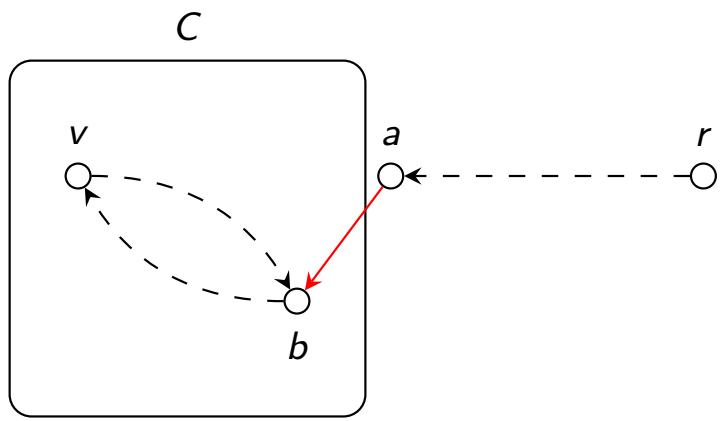


Il existe un chemin $r \rightsquigarrow a \to b \rightsquigarrow v$;

$\Rightarrow$ on finit d'explorer v avant $r \Rightarrow$ date_fin$(v) <$ date_fin(r) ;

Correction de l'algorithme de Kosaraju-Sharir

Soit v le sommet de G dont la date de fin d'exploration est maximale. Alors la CFC C de G contenant v est une source, c'est-à-dire qu'il n'existe pas d'arc (a, b) avec $a \notin C$ et $b \in C$. Procédons par contradiction : on suppose qu'il existe un arc (a, b) avec $a \notin C$ et $b \in C$. Soit r le sommet à partir duquel l'exploration en profondeur mène au sommet a ; on sait que $r \notin C$, sinon on aurait $a \in C$; donc $r \neq v$.



Il existe un chemin $r \rightsquigarrow a \to b \rightsquigarrow v$;

$\Rightarrow$ on finit d'explorer v avant $r \Rightarrow$ date_fin$(v) <$ date_fin(r) ;

$\Rightarrow$ contradiction : date_fin(v) est censée être maximale. □

Correction de l'algorithme de Kosaraju-Sharir

On peut maintenant prouver que l'algorithme de Kosaraju-Sharir est correct :

- le premier parcours nous donne les descendants de chaque sommet ;

Correction de l'algorithme de Kosaraju-Sharir

On peut maintenant prouver que l'algorithme de Kosaraju-Sharir est correct :

- le premier parcours nous donne les descendants de chaque sommet ;
- le second parcours reconstruit chaque CFC C "sans déborder", puisqu'il n'existe pas d'arc sortant de C dans G' ;

Correction de l'algorithme de Kosaraju-Sharir

On peut maintenant prouver que l'algorithme de Kosaraju-Sharir est correct :

- le premier parcours nous donne les descendants de chaque sommet ;
- le second parcours reconstruit chaque CFC C "sans déborder", puisqu'il n'existe pas d'arc sortant de C dans G' ;
- à chaque identification d'une CFC, on la supprime et on recommence ; □

Plus courts chemins : poids < 0

- L'algorithme de Dijkstra fonctionne bien tant qu'il n'y a pas de poids négatifs dans notre graphe ;

Plus courts chemins : poids < 0

- L'algorithme de Dijkstra fonctionne bien tant qu'il n'y a pas de poids négatifs dans notre graphe ;
- Cela n'arrive pas dans le cas de distances ; quand pourrait-on en avoir ?

Plus courts chemins : poids < 0

- L'algorithme de Dijkstra fonctionne bien tant qu'il n'y a pas de poids négatifs dans notre graphe ;
- Cela n'arrive pas dans le cas de distances ; quand pourrait-on en avoir ?
    - quand les poids correspondent à des dépenses énergétiques ;

Plus courts chemins : poids < 0

- L'algorithme de Dijkstra fonctionne bien tant qu'il n'y a pas de poids négatifs dans notre graphe ;
- Cela n'arrive pas dans le cas de distances ; quand pourrait-on en avoir ?
  - quand les poids correspondent à des dépenses énergétiques ;
  - quand les poids correspondent à des coûts ou à des remises ;

Plus courts chemins : poids < 0

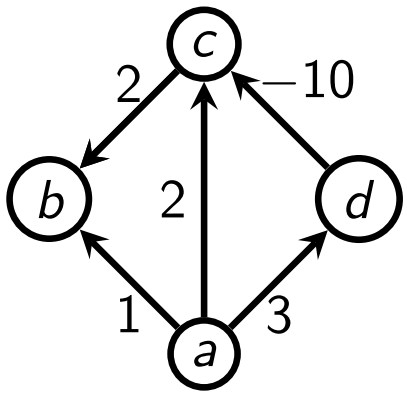
- L'algorithme de Dijkstra fonctionne bien tant qu'il n'y a pas de poids négatifs dans notre graphe ;
- Cela n'arrive pas dans le cas de distances ; quand pourrait-on en avoir ?
  - quand les poids correspondent à des dépenses énergétiques ;
  - quand les poids correspondent à des coûts ou à des remises ;
  - ...

Plus courts chemins : poids < 0

- L'algorithme de Dijkstra fonctionne bien tant qu'il n'y a pas de poids négatifs dans notre graphe ;
- Cela n'arrive pas dans le cas de distances ; quand pourrait-on en avoir ?
  - quand les poids correspondent à des dépenses énergétiques ;
  - quand les poids correspondent à des coûts ou à des remises ;
  - ...
- Comme on va le voir, l'algorithme de Dijkstra n'est plus fiable dans ces cas-là ;

Dijkstra en présence de poids < 0

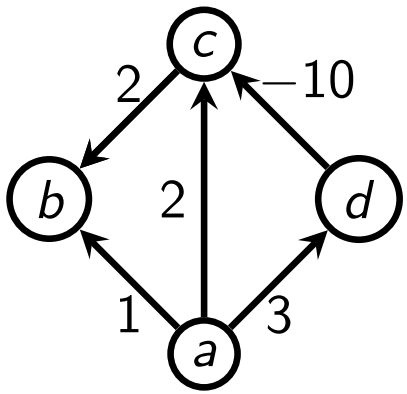
Exemple 1 (plus courts chemins depuis a)



étape	a	b	c	d
0	0	$+\infty$	$+\infty$	$+\infty$

Dijkstra en présence de poids < 0

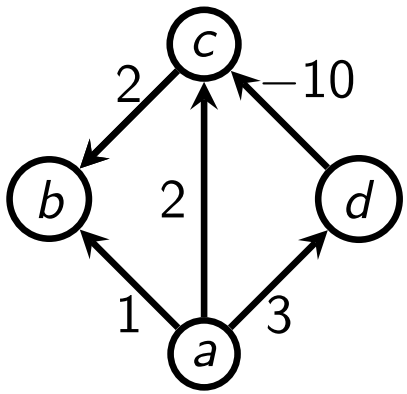
Exemple 1 (plus courts chemins depuis a)



étape	a	b	c	d
0	0	$+\infty$	$+\infty$	$+\infty$

Dijkstra en présence de poids < 0

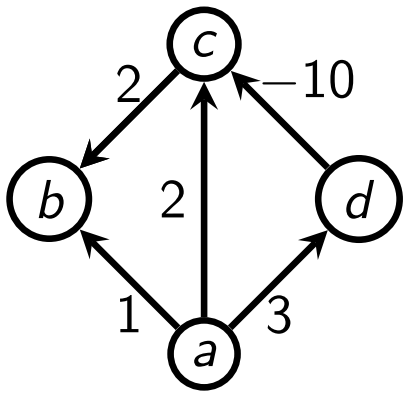
Exemple 1 (plus courts chemins depuis a)



étape	a	b	c	d
0	0	$+\infty$	$+\infty$	$+\infty$
1	0	1	2	3

Dijkstra en présence de poids < 0

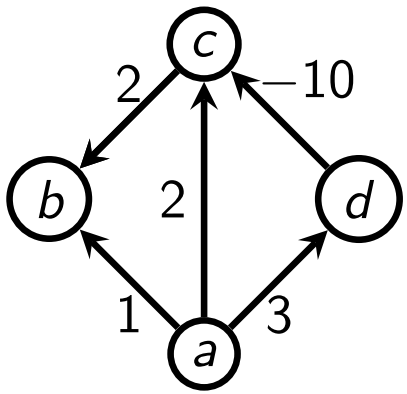
Exemple 1 (plus courts chemins depuis a)



étape	a	b	c	d
0	0	$+\infty$	$+\infty$	$+\infty$
1	0	1	2	3

Dijkstra en présence de poids < 0

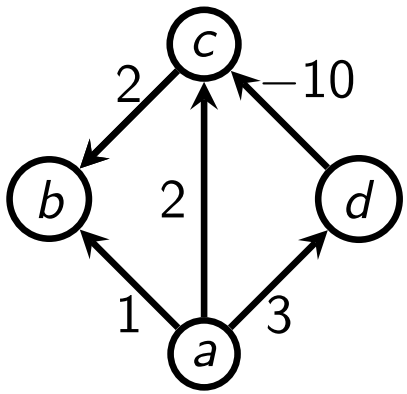
Exemple 1 (plus courts chemins depuis a)



étape	a	b	c	d
0	0	$+\infty$	$+\infty$	$+\infty$
1	0	1	2	3
2	0	1	2	3

Dijkstra en présence de poids < 0

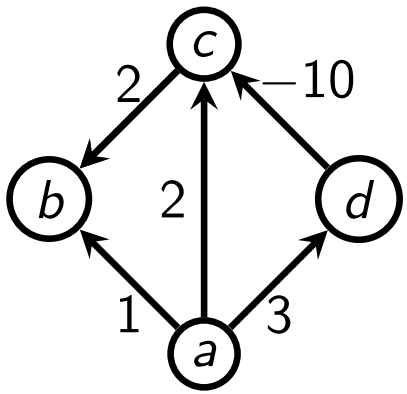
Exemple 1 (plus courts chemins depuis a)



étape	a	b	c	d
0	0	$+\infty$	$+\infty$	$+\infty$
1	0	1	2	3
2	0	1	2	3

Dijkstra en présence de poids < 0

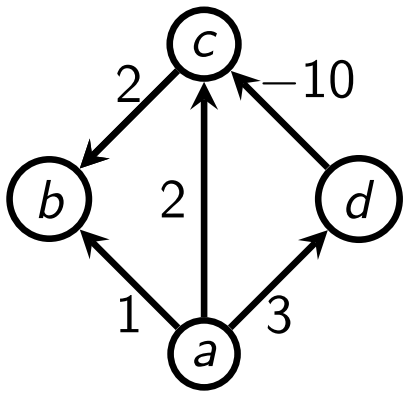
Exemple 1 (plus courts chemins depuis a)



étape	a	b	c	d
0	0	$+\infty$	$+\infty$	$+\infty$
1	0	1	2	3
2	0	1	2	3
3	0	1	2	3

Dijkstra en présence de poids < 0

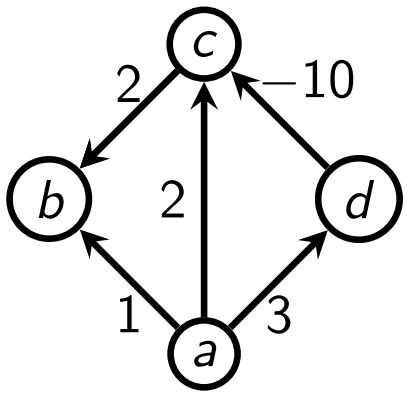
Exemple 1 (plus courts chemins depuis a)



étape	a	b	c	d
0	0	$+\infty$	$+\infty$	$+\infty$
1	0	1	2	3
2	0	1	2	3
3	0	1	2	3

Dijkstra en présence de poids < 0

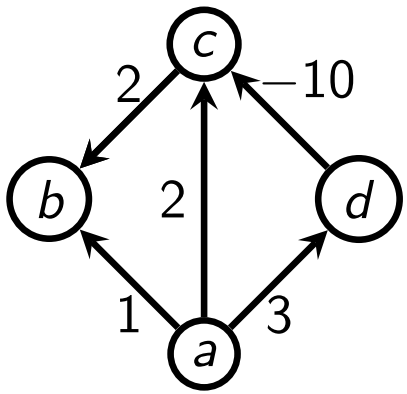
Exemple 1 (plus courts chemins depuis a)



étape	a	b	c	d
0	0	$+\infty$	$+\infty$	$+\infty$
1	0	1	2	3
2	0	1	2	3
3	0	1	2	3
4	0	1	-7	3

Dijkstra en présence de poids < 0

Exemple 1 (plus courts chemins depuis a)



étape	a	b	c	d
0	0	$+\infty$	$+\infty$	$+\infty$
1	0	1	2	3
2	0	1	2	3
3	0	1	2	3
4	0	1	-7	3

On rate le chemin optimal $a \rightsquigarrow d \rightsquigarrow c \rightsquigarrow b$ de poids -5.

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 ;
- Dans le contre-exemple, un arc de poids < 0 nous a permis de trouver un meilleur chemin ;

Problèmes de Dijkstra

- En l'absence de poids négatifs, Dijkstra s'en sortait puisqu'il n'était pas nécessaire de revenir sur ses décisions ;
- Dans le contre-exemple, un arc de poids < 0 nous a permis de trouver un meilleur chemin ;
- Mais comme l'algorithme se termine, les autres chemins n'en bénéficieront pas ;

Problèmes de Dijkstra

- En l'absence de poids négatifs, Dijkstra s'en sortait puisqu'il n'était pas nécessaire de revenir sur ses décisions ;
- Dans le contre-exemple, un arc de poids < 0 nous a permis de trouver un meilleur chemin ;
- Mais comme l'algorithme se termine, les autres chemins n'en bénéficieront pas ;
- 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

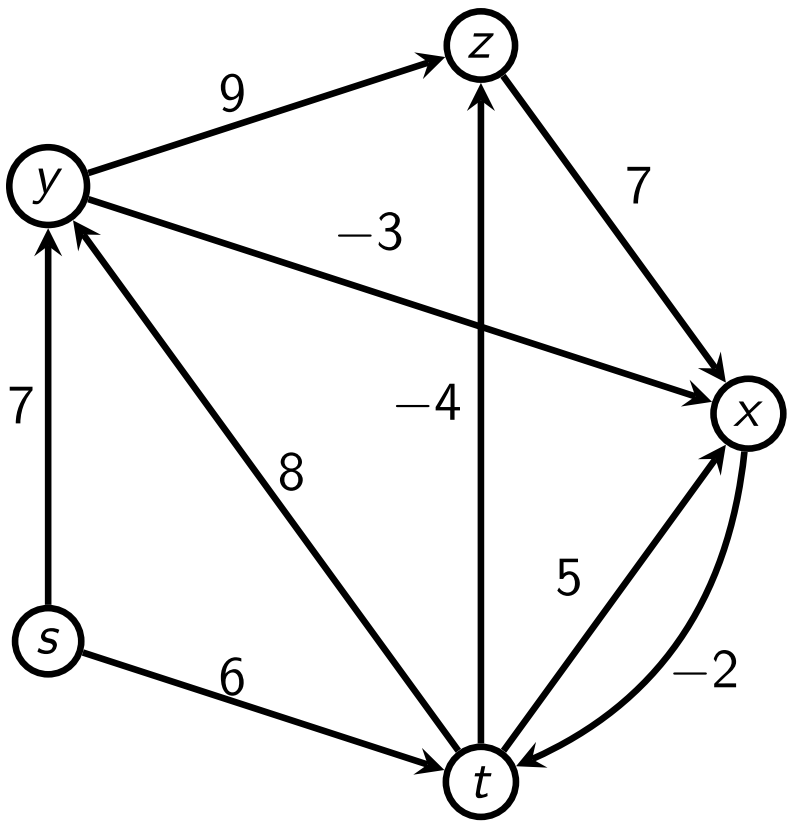
- L'algorithme de Bellman-Ford recherche également des raccourcis entre les sommets ;
- Le fonctionnement est différent de celui de Dijkstra, qui cherchait des chemins moins coûteux de la source s à un sommet u en passant par un autre sommet v ;
- Ici, on vérifie pour chaque arc (u, v) s'il existe un 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 2 (source $= s$)



Les coulisses

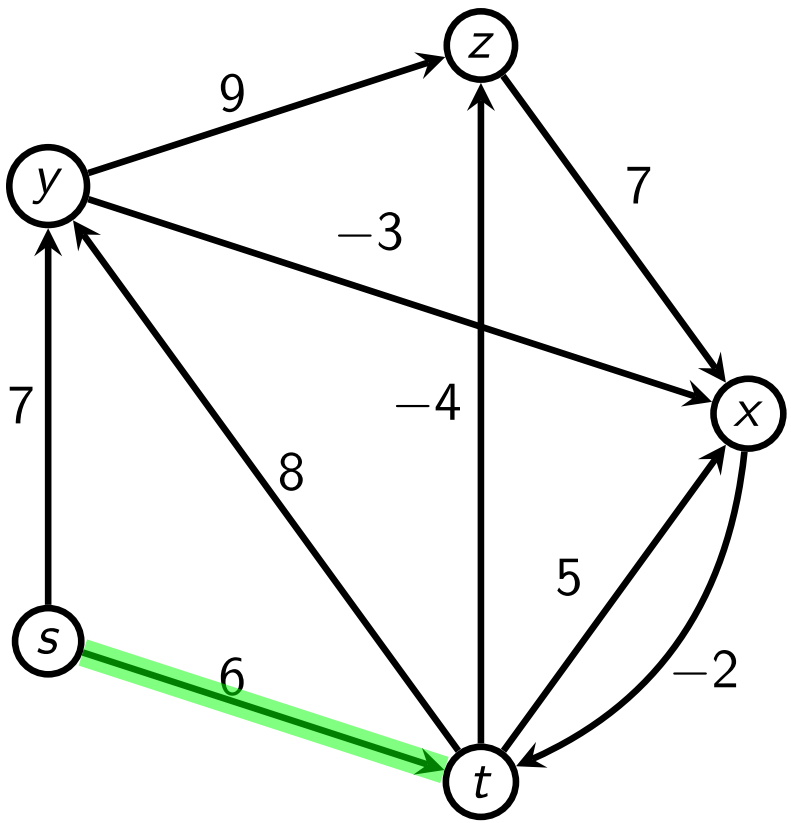
étape	s	t	x	y	z
(a)	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$

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 2 (source $= s$)



Les coulisses

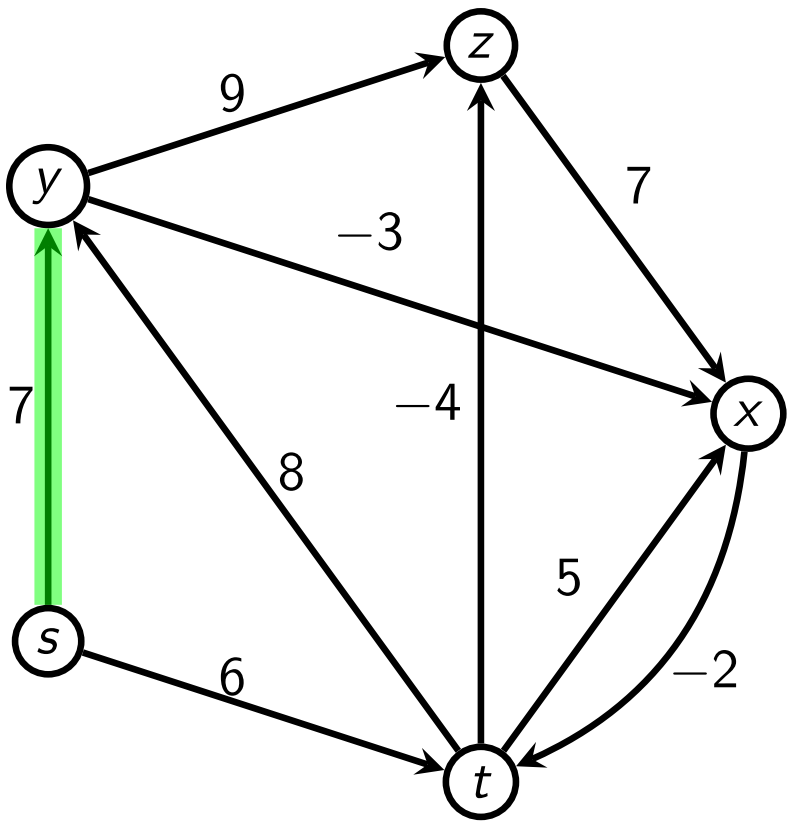
étape	s	t	x	y	z
(a)	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$

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 2 (source $= s$)



Les coulisses

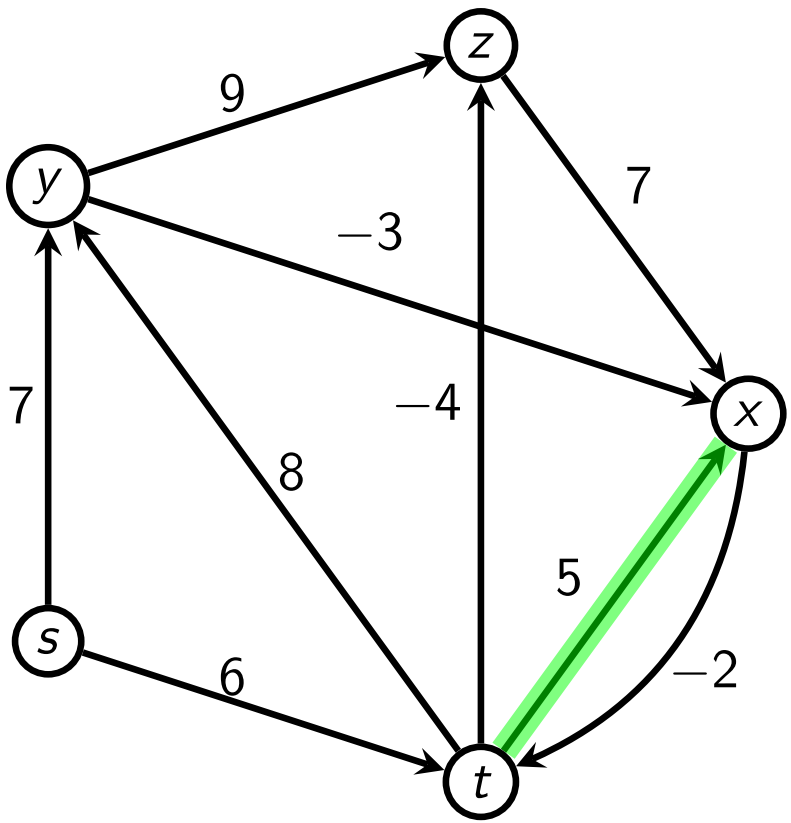
étape	s	t	x	y	z
(a)	0	6	$+\infty$	$+\infty$	$+\infty$

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 2 (source $= s$)



Les coulisses

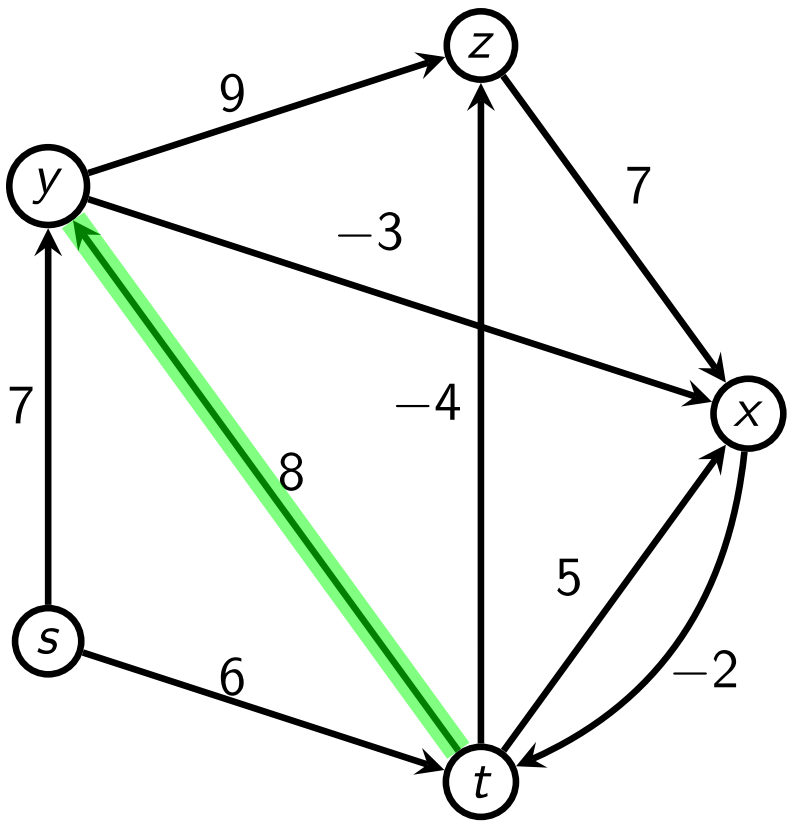
étape	s	t	x	y	z
(a)	0	6	$+\infty$	7	$+\infty$

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 2 (source $= s$)



Les coulisses

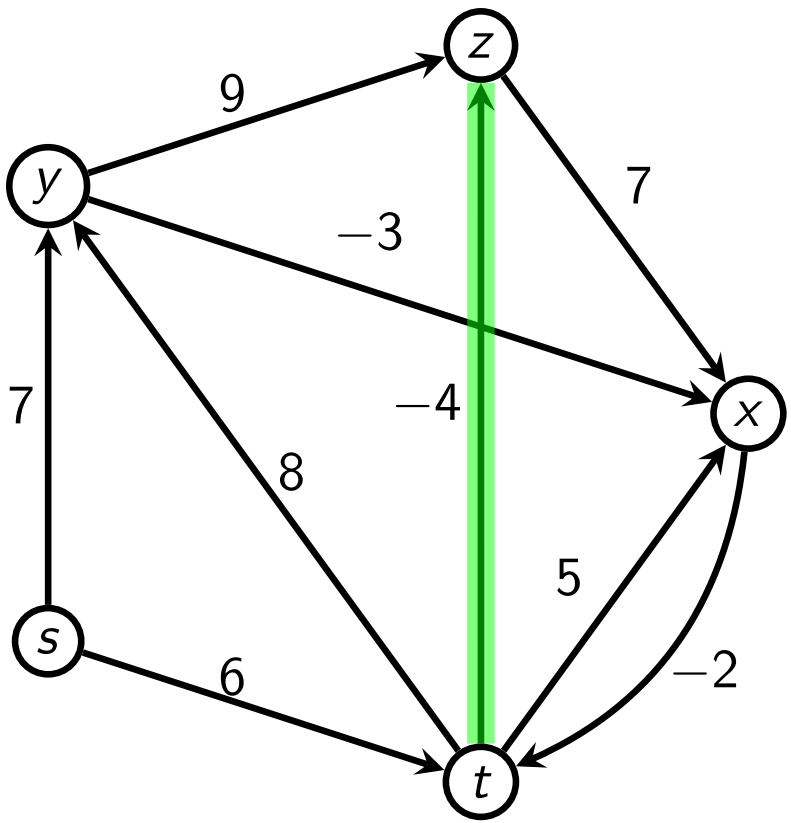
étape	s	t	x	y	z
(a)	0	6	11	7	$+\infty$

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 2 (source $= s$)



Les coulisses

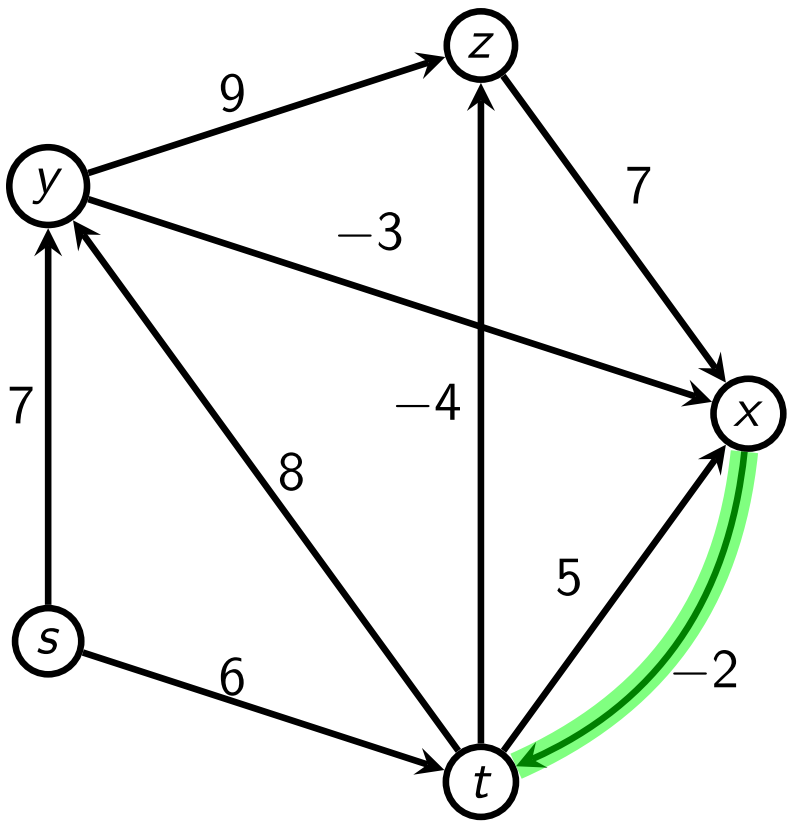
étape	s	t	x	y	z
(a)	0	6	11	7	$+\infty$

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 2 (source $= s$)



Les coulisses

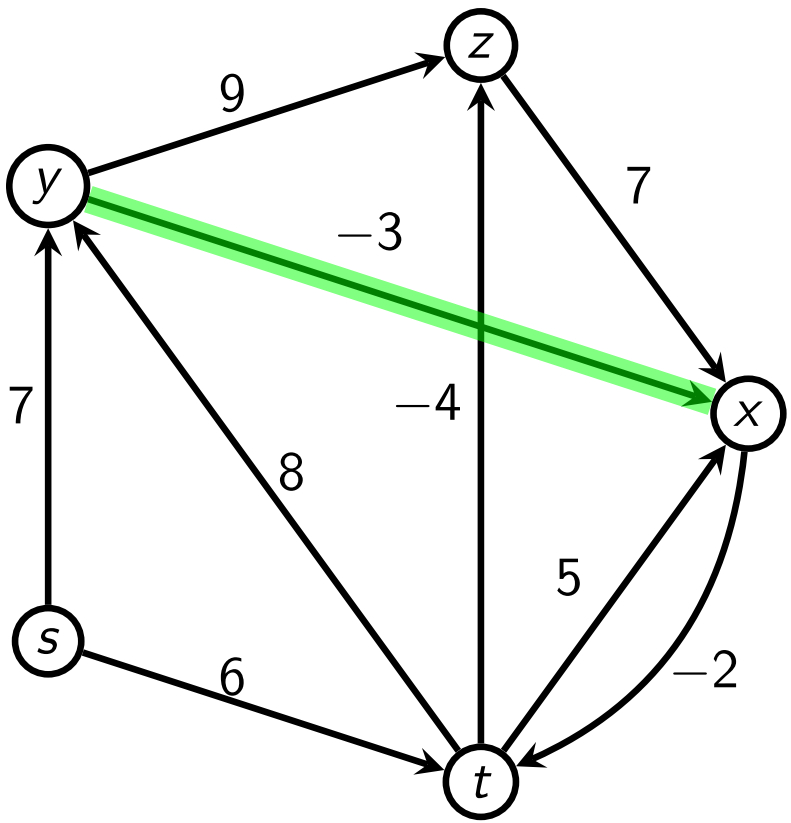
étape	s	t	x	y	z
(a)	0	6	11	7	2

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 2 (source $= s$)



Les coulisses

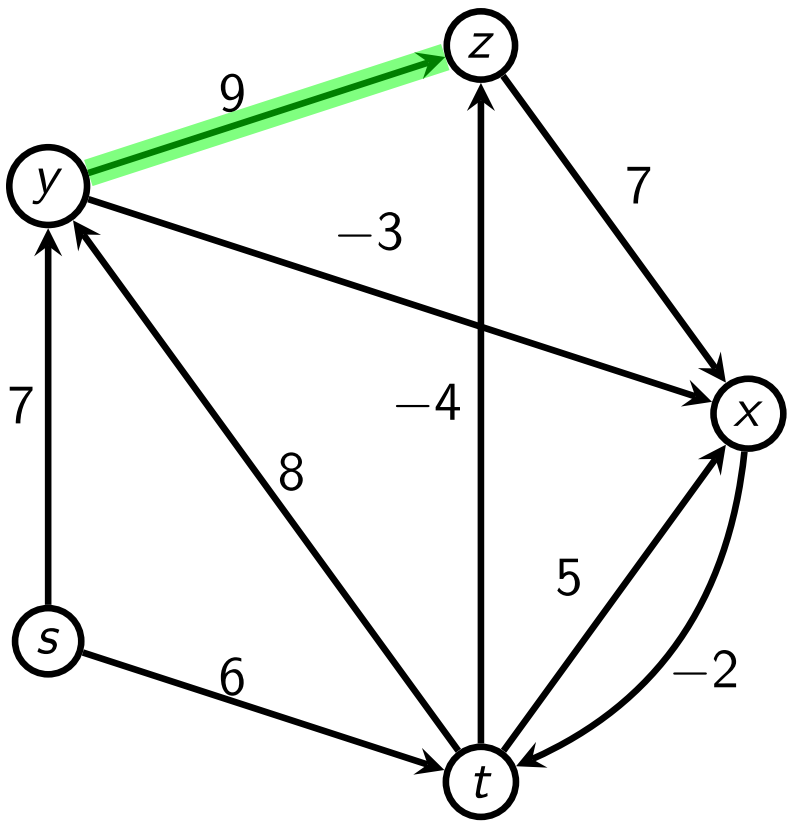
étape	s	t	x	y	z
(a)	0	6	11	7	2

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 2 (source $= s$)



Les coulisses

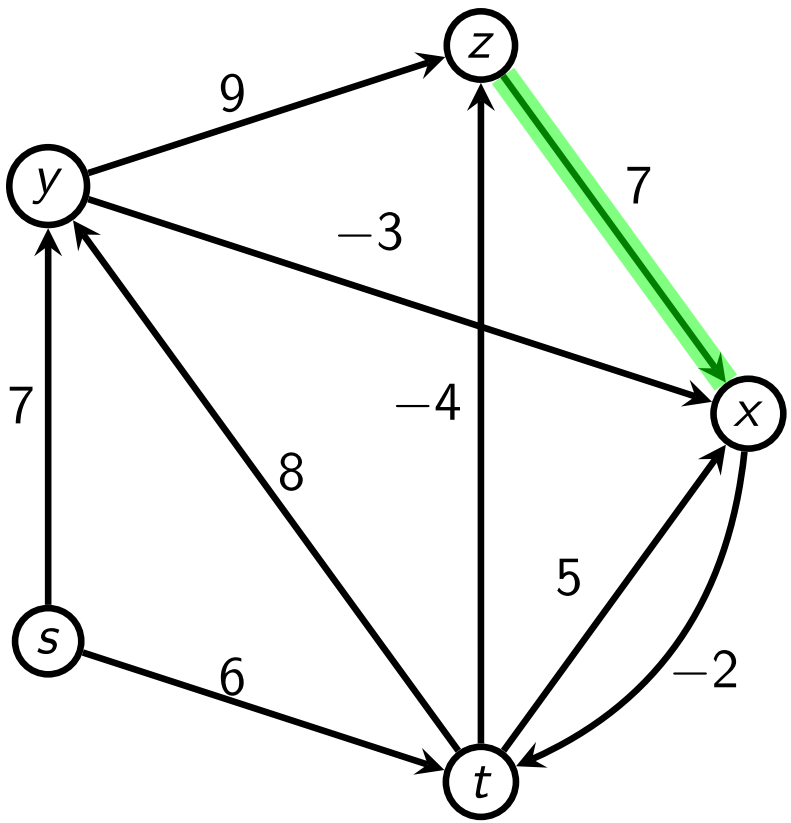
étape	s	t	x	y	z
(a)	0	6	4	7	2

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 2 (source $= s$)



Les coulisses

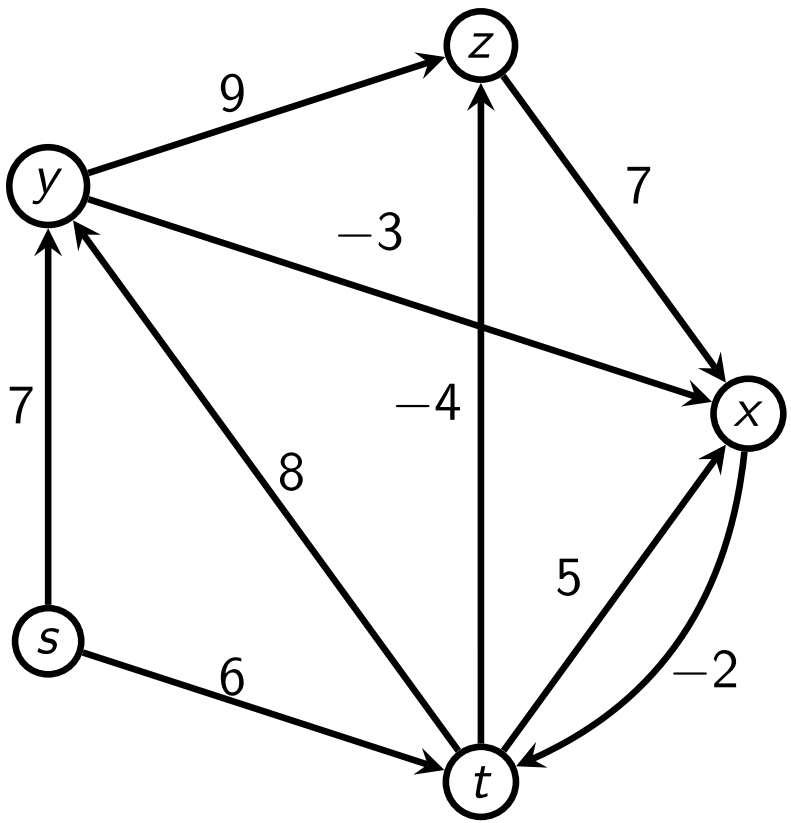
étape	s	t	x	y	z
(a)	0	6	4	7	2

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 2 (source $= s$)



Les coulisses

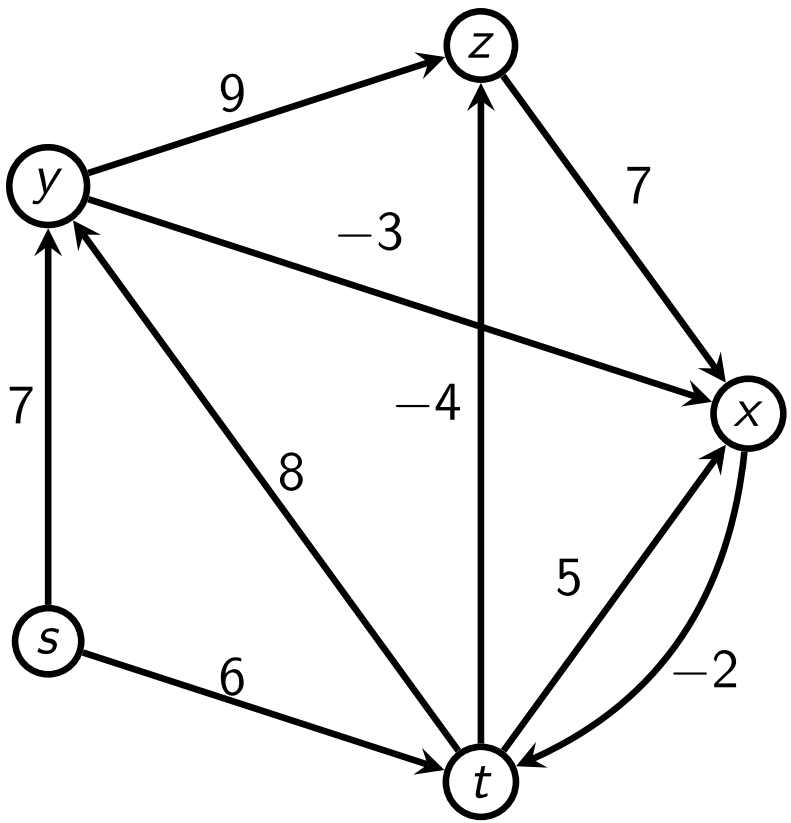
étape	s	t	x	y	z
(a)	0	6	4	7	2

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 2 (source $= s$)



Les coulisses

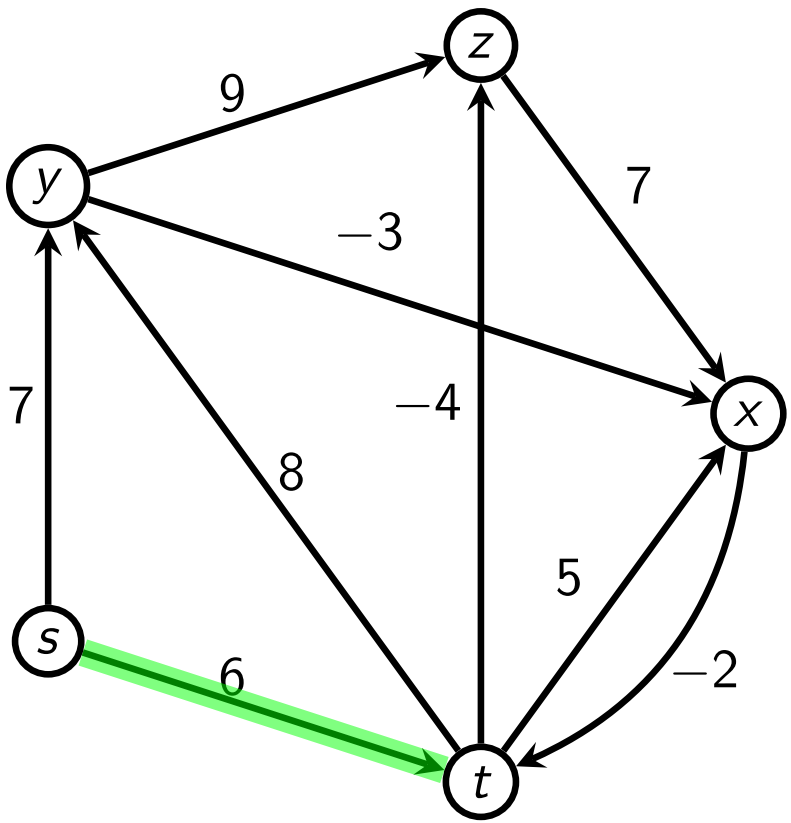
étape	s	t	x	y	z
(a)	0	6	4	7	2
(b)	0	6	4	7	2

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 2 (source $= s$)



Les coulisses

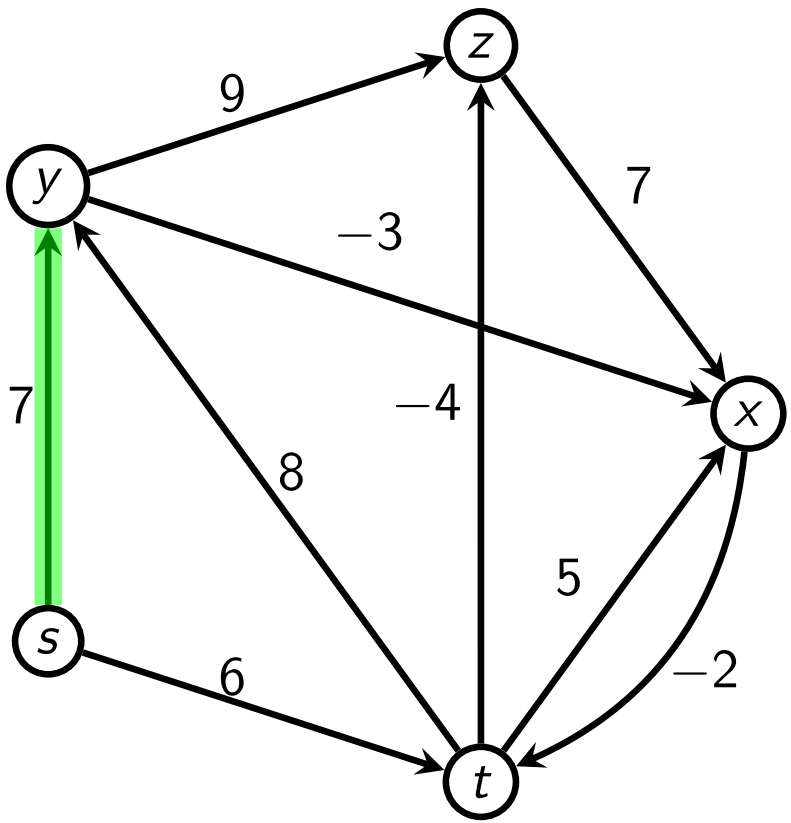
étape	s	t	x	y	z
(a)	0	6	4	7	2
(b)	0	6	4	7	2

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 2 (source $= s$)



Les coulisses

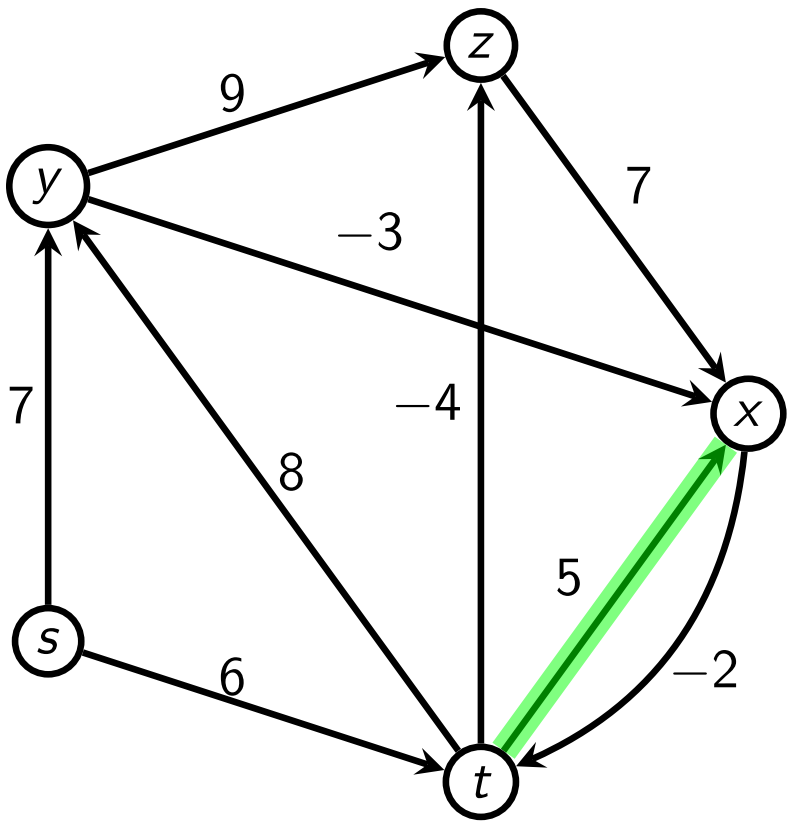
étape	s	t	x	y	z
(a)	0	6	4	7	2
(b)	0	6	4	7	2

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 2 (source $= s$)



Les coulisses

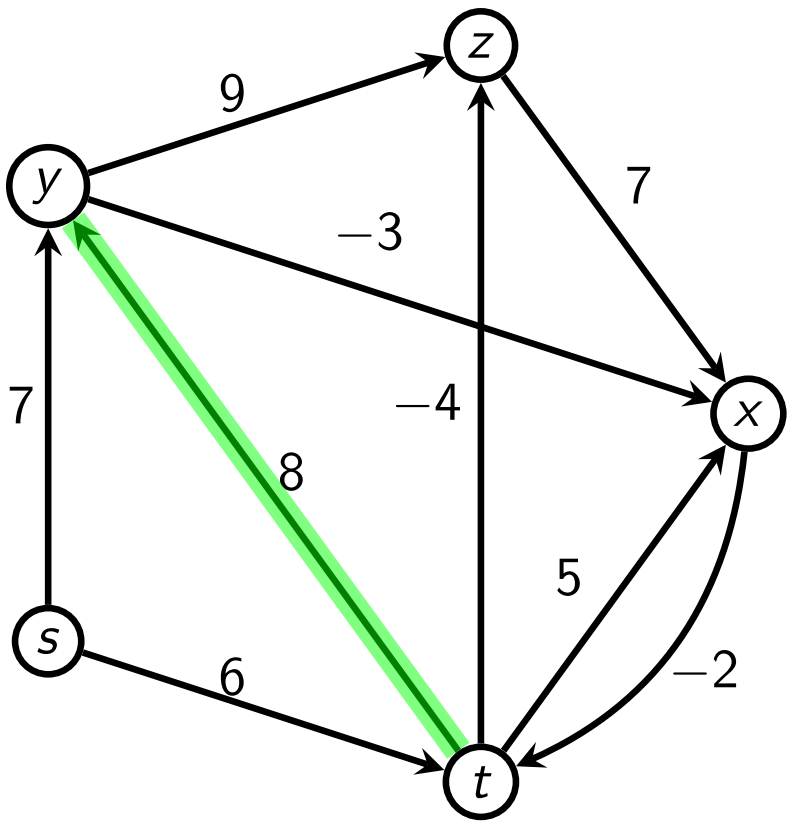
étape	s	t	x	y	z
(a)	0	6	4	7	2
(b)	0	6	4	7	2

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 2 (source $= s$)



Les coulisses

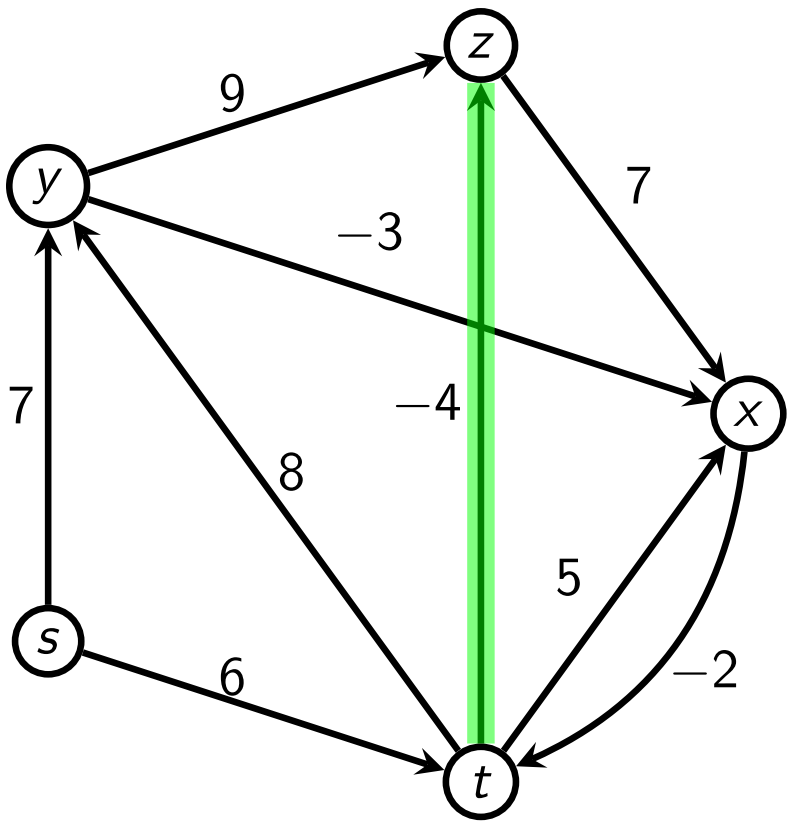
étape	s	t	x	y	z
(a)	0	6	4	7	2
(b)	0	6	4	7	2

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 2 (source $= s$)



Les coulisses

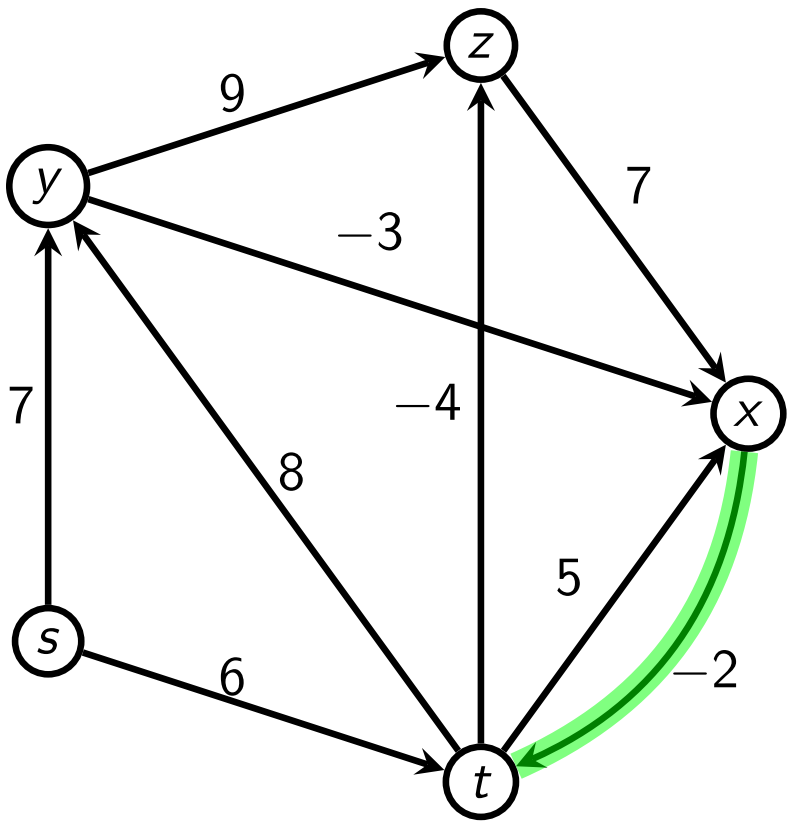
étape	s	t	x	y	z
(a)	0	6	4	7	2
(b)	0	6	4	7	2

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 2 (source $= s$)



Les coulisses

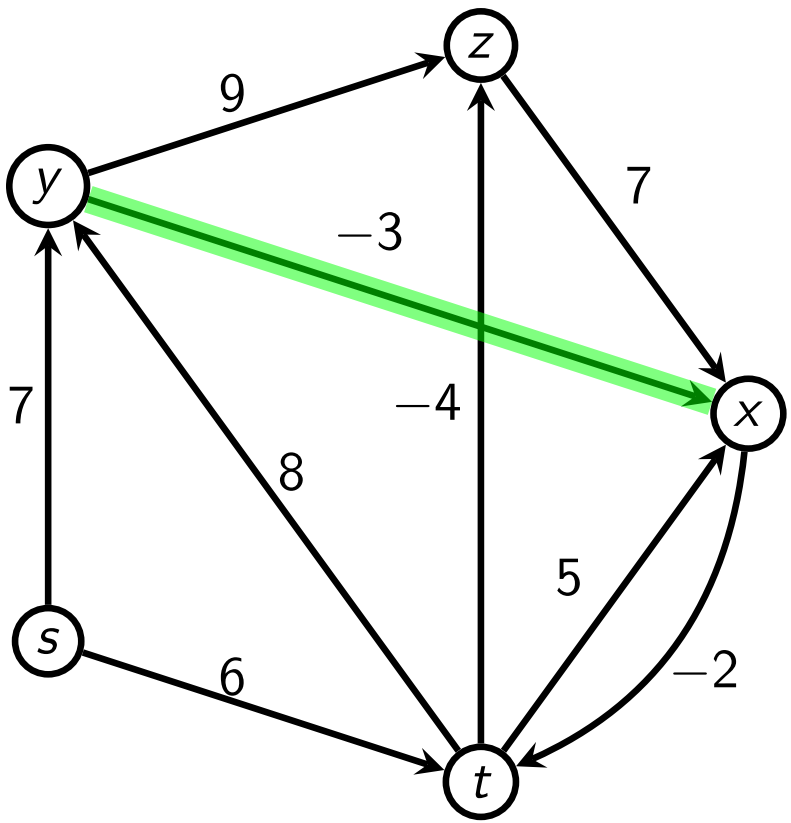
étape	s	t	x	y	z
(a)	0	6	4	7	2
(b)	0	6	4	7	2

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 2 (source $= s$)



Les coulisses

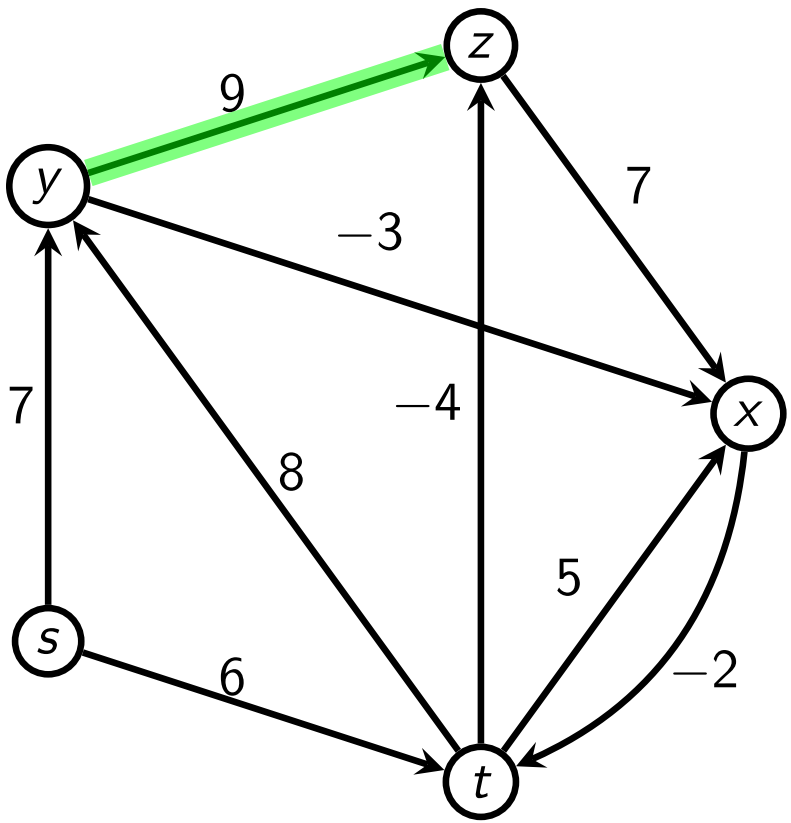
étape	s	t	x	y	z
(a)	0	6	4	7	2
(b)	0	2	4	7	2

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 2 (source $= s$)



Les coulisses

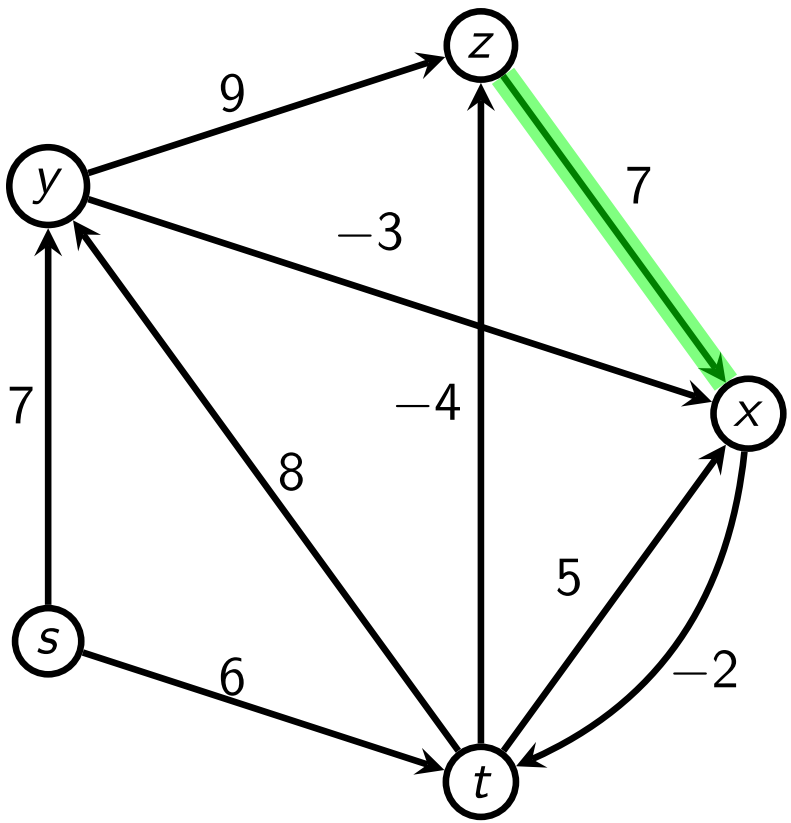
étape	s	t	x	y	z
(a)	0	6	4	7	2
(b)	0	2	4	7	2

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 2 (source $= s$)



Les coulisses

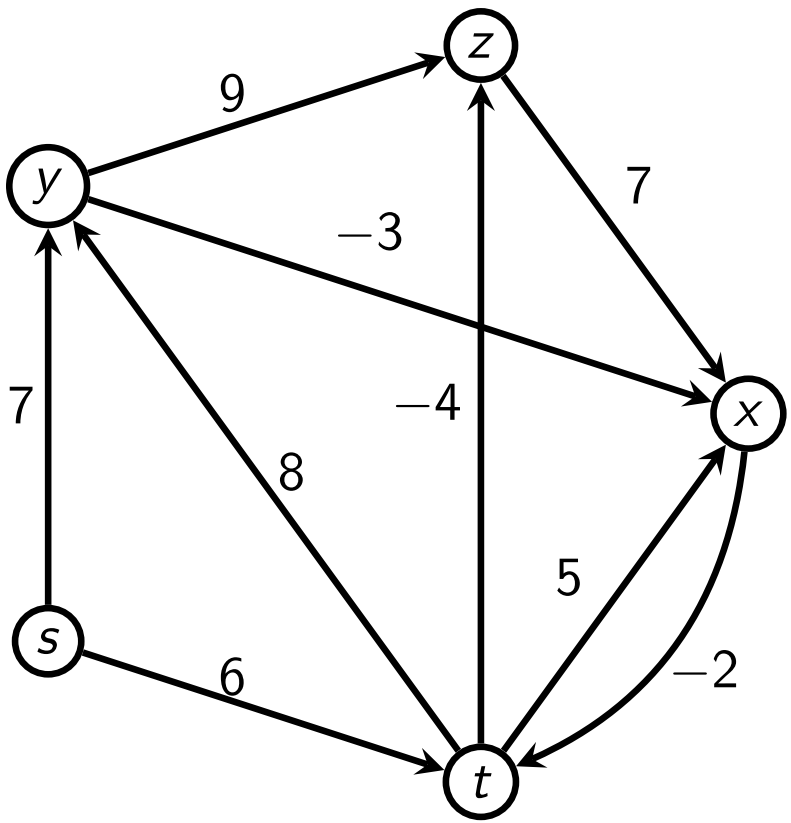
étape	s	t	x	y	z
(a)	0	6	4	7	2
(b)	0	2	4	7	2

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 2 (source $= s$)



Les coulisses

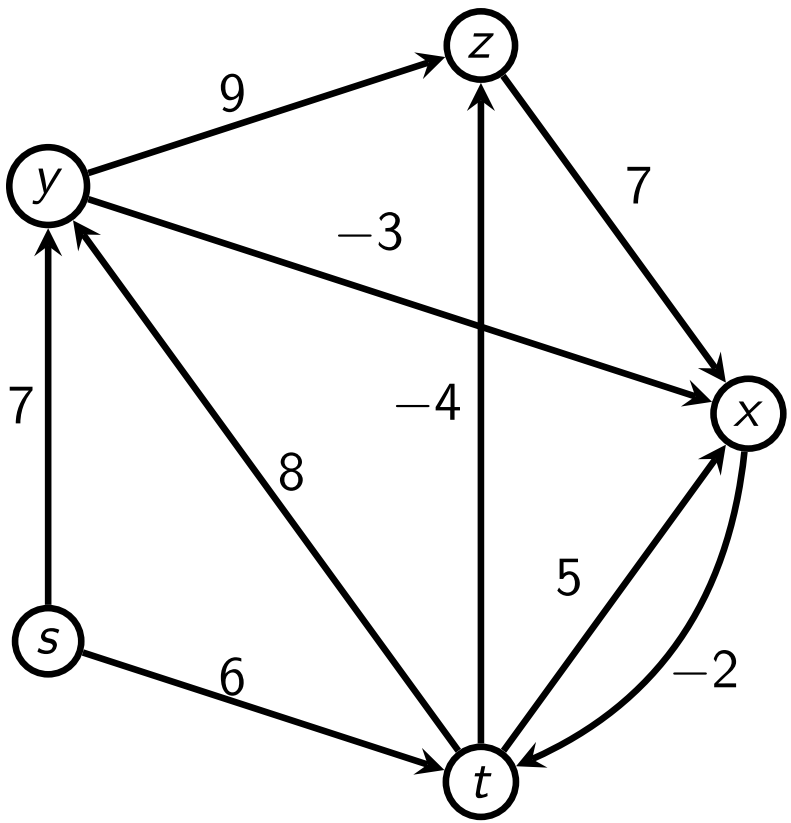
étape	s	t	x	y	z
(a)	0	6	4	7	2
(b)	0	2	4	7	2

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 2 (source $= s$)



Les coulisses

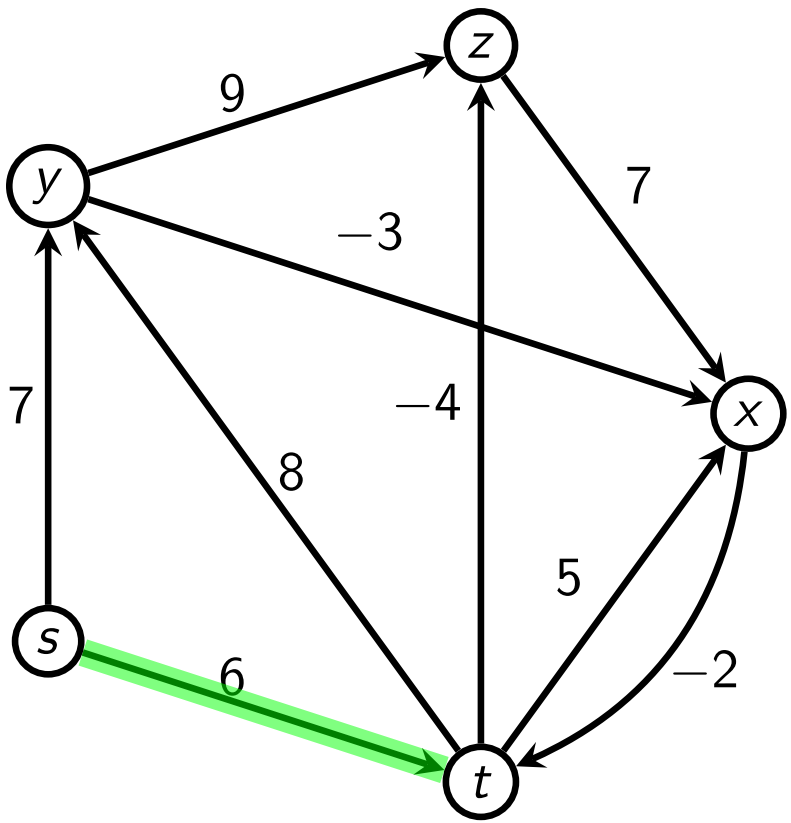
étape	s	t	x	y	z
(a)	0	6	4	7	2
(b)	0	2	4	7	2
(c)	0	2	4	7	2

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 2 (source $= s$)



Les coulisses

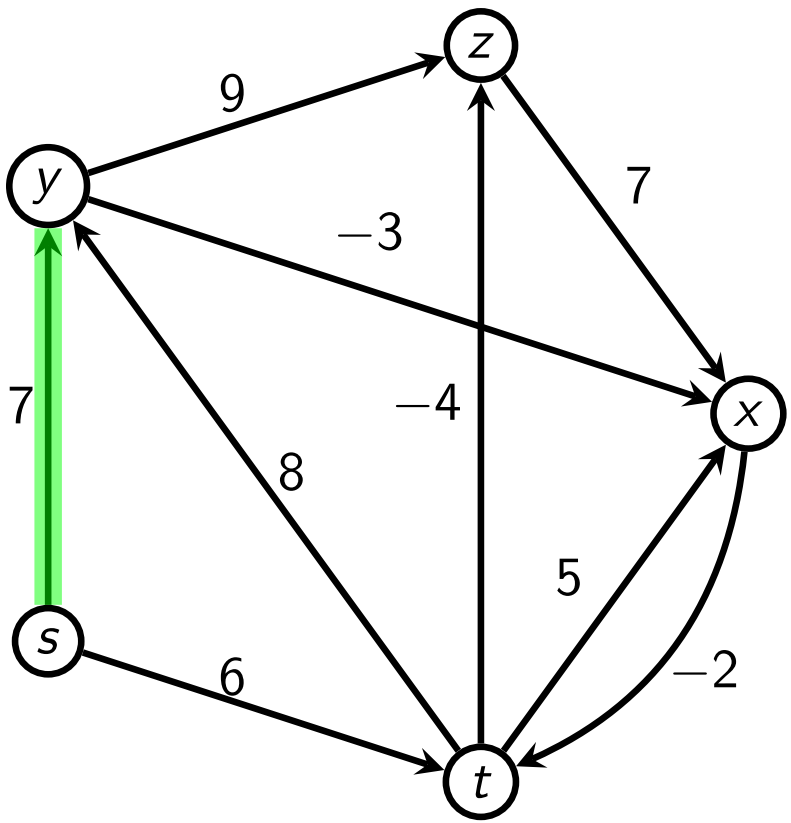
étape	s	t	x	y	z
(a)	0	6	4	7	2
(b)	0	2	4	7	2
(c)	0	2	4	7	2

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 2 (source $= s$)



Les coulisses

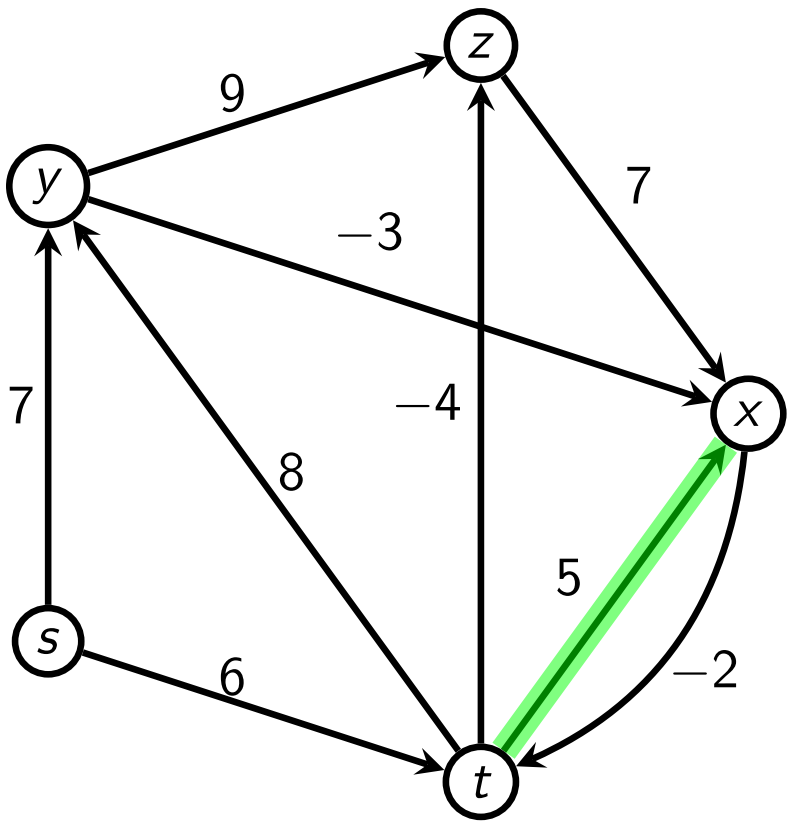
étape	s	t	x	y	z
(a)	0	6	4	7	2
(b)	0	2	4	7	2
(c)	0	2	4	7	2

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 2 (source $= s$)



Les coulisses

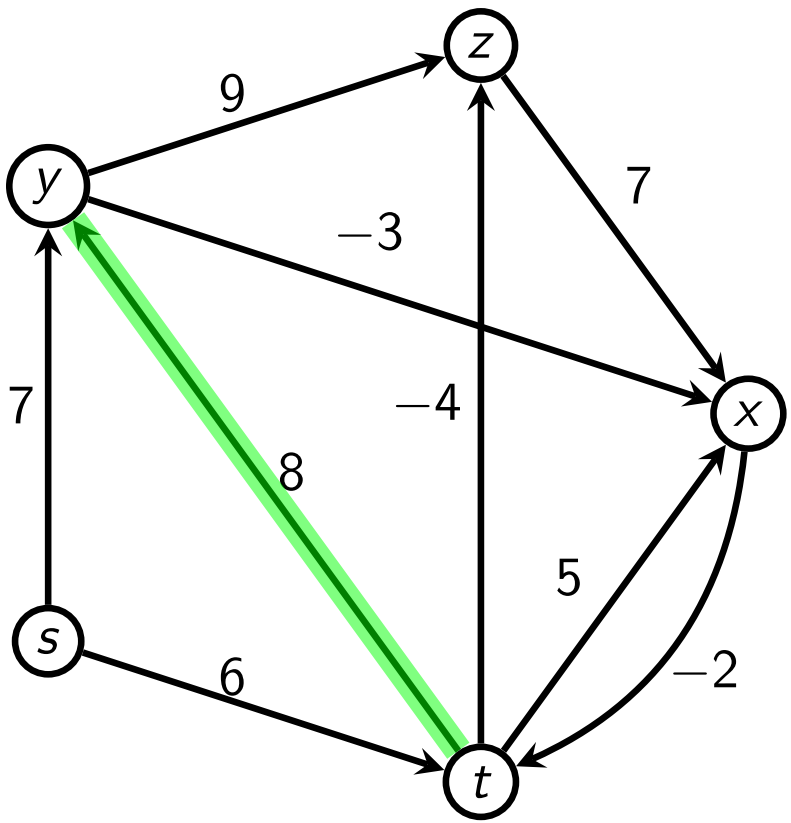
étape	s	t	x	y	z
(a)	0	6	4	7	2
(b)	0	2	4	7	2
(c)	0	2	4	7	2

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 2 (source $= s$)



Les coulisses

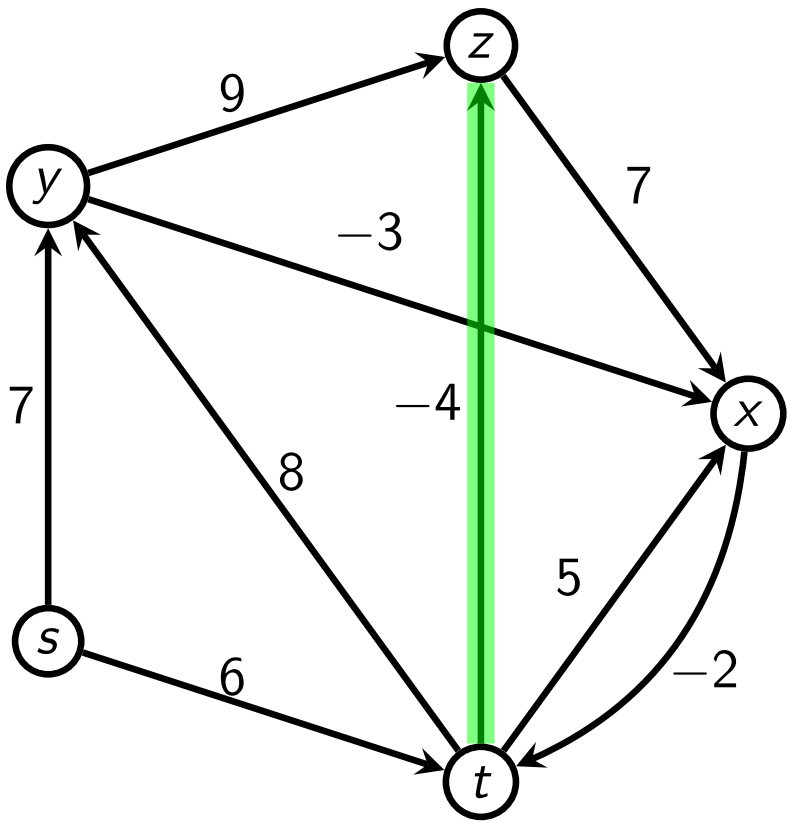
étape	s	t	x	y	z
(a)	0	6	4	7	2
(b)	0	2	4	7	2
(c)	0	2	4	7	2

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 2 (source $= s$)



Les coulisses

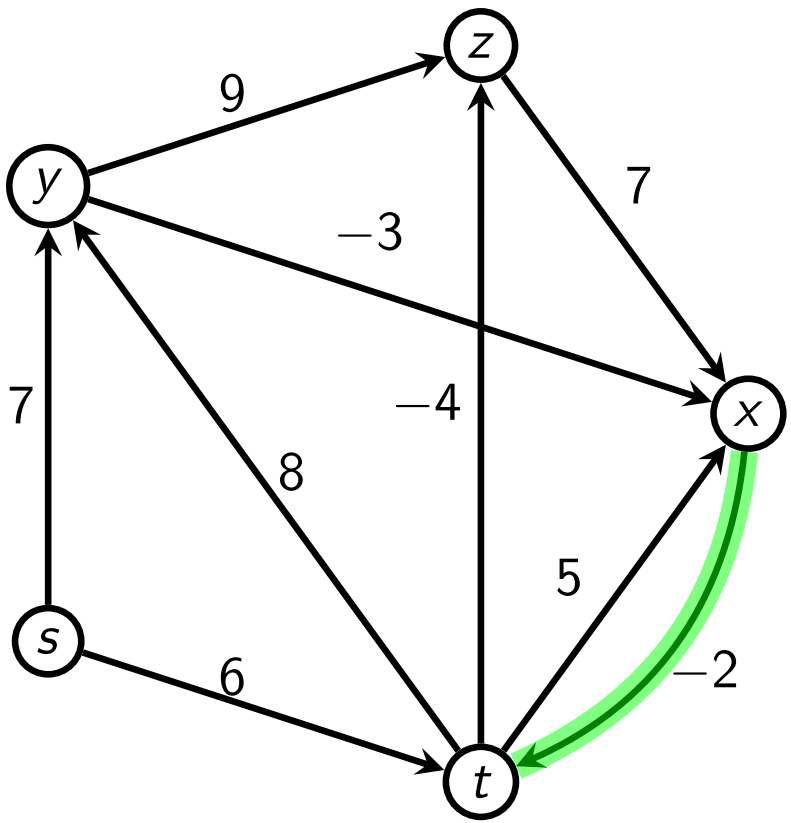
étape	s	t	x	y	z
(a)	0	6	4	7	2
(b)	0	2	4	7	2
(c)	0	2	4	7	2

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 2 (source = s)



Les coulisses

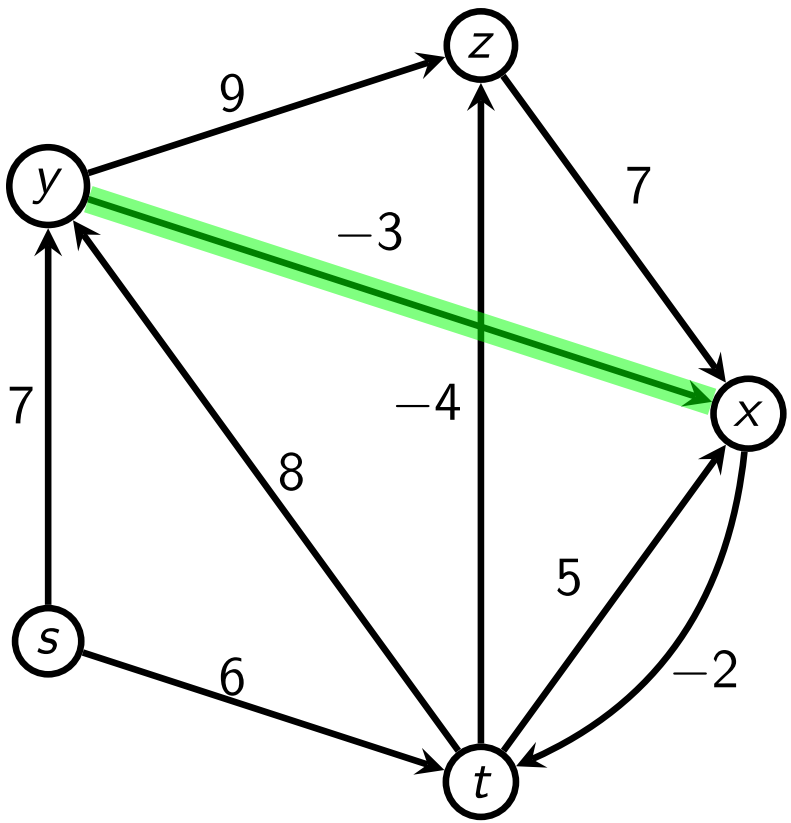
étape	s	t	x	y	z
(a)	0	6	4	7	2
(b)	0	2	4	7	2
(c)	0	2	4	7	-2

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 2 (source = s)



Les coulisses

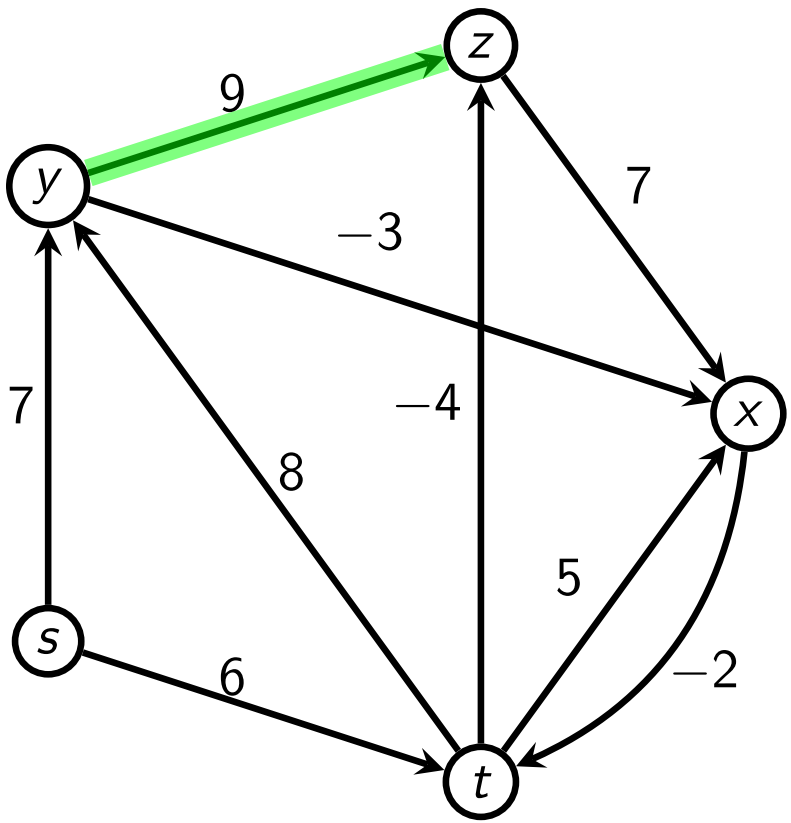
étape	s	t	x	y	z
(a)	0	6	4	7	2
(b)	0	2	4	7	2
(c)	0	2	4	7	-2

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 2 (source $= s$)



Les coulisses

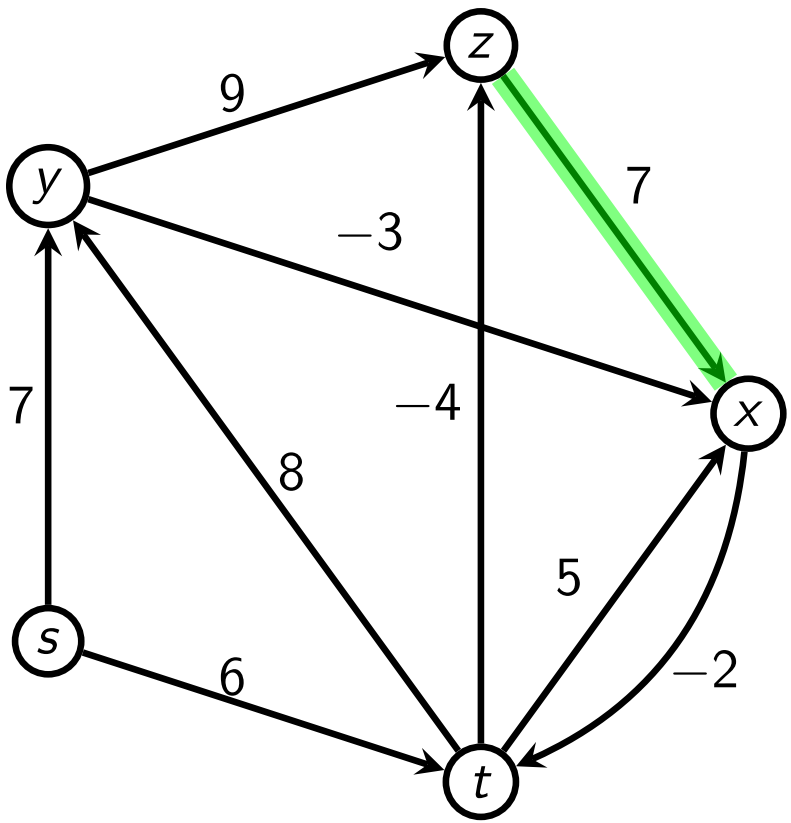
étape	s	t	x	y	z
(a)	0	6	4	7	2
(b)	0	2	4	7	2
(c)	0	2	4	7	-2

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 2 (source $= s$)



Les coulisses

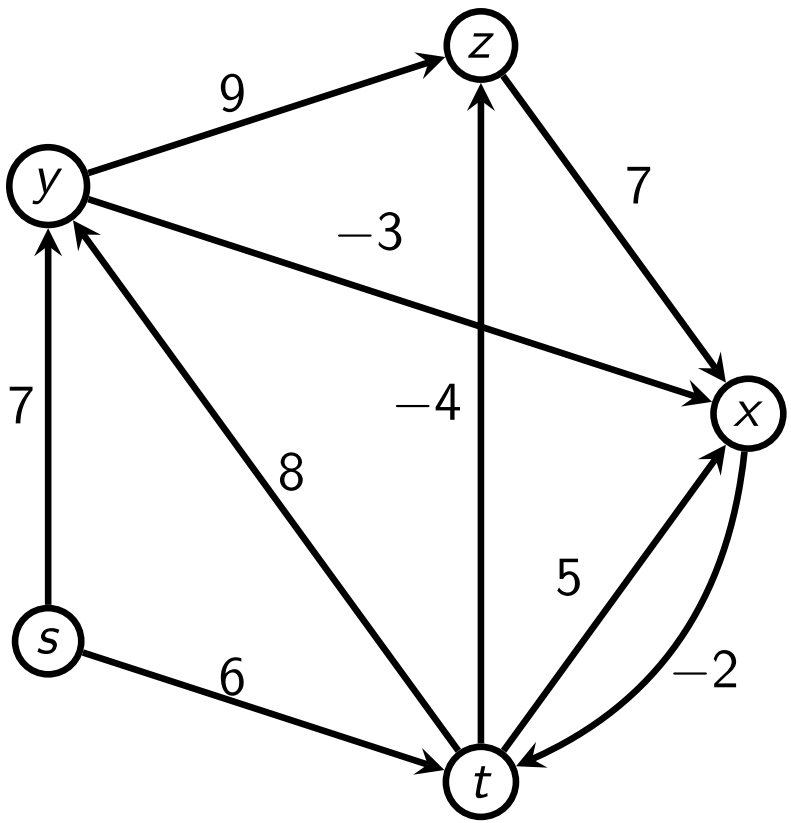
étape	s	t	x	y	z
(a)	0	6	4	7	2
(b)	0	2	4	7	2
(c)	0	2	4	7	-2

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 2 (source = s)



Les coulisses

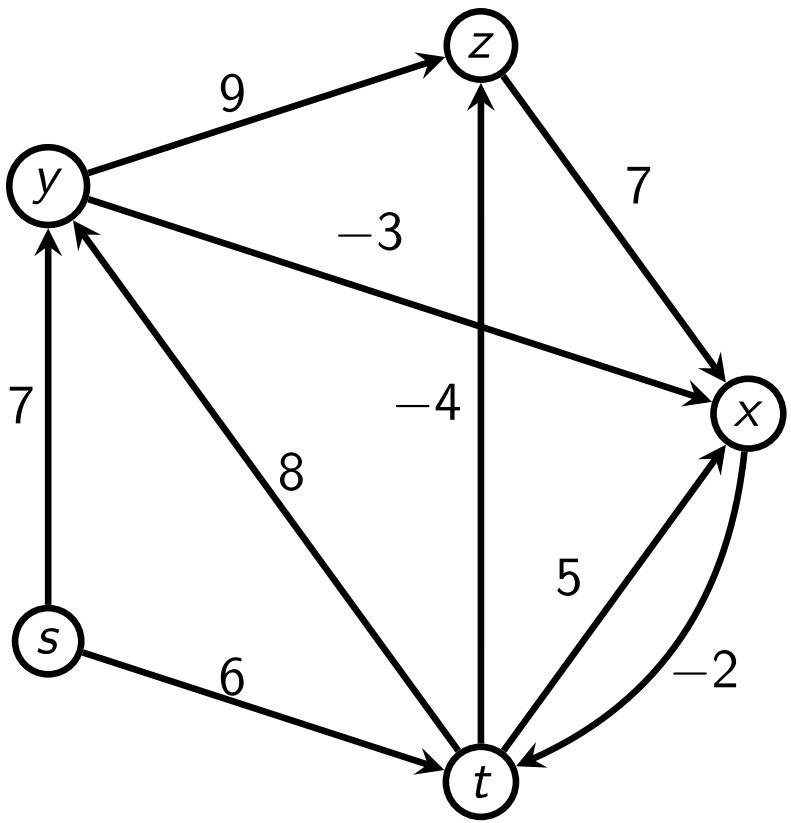
étape	s	t	x	y	z
(a)	0	6	4	7	2
(b)	0	2	4	7	2
(c)	0	2	4	7	-2

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 2 (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

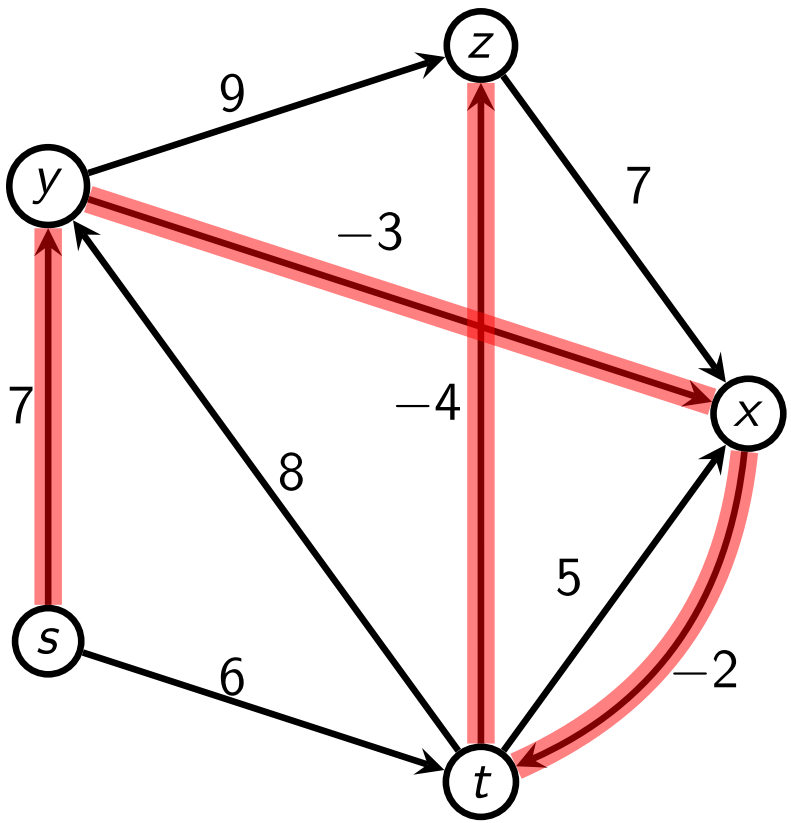
(les étapes suivantes ne changent plus rien)

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 2 (source $= s$)



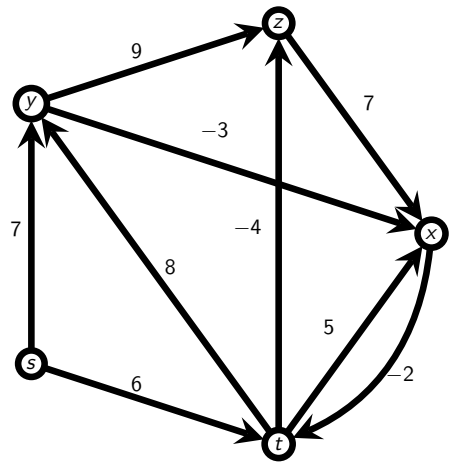
Les coulisses

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 . . .)

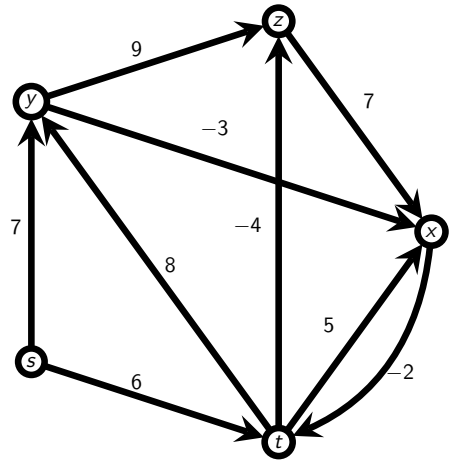
Le déroulement complet étant long, on ne vous demandera généralement les détails que pour une passe sur tous les arcs.



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 . . .)

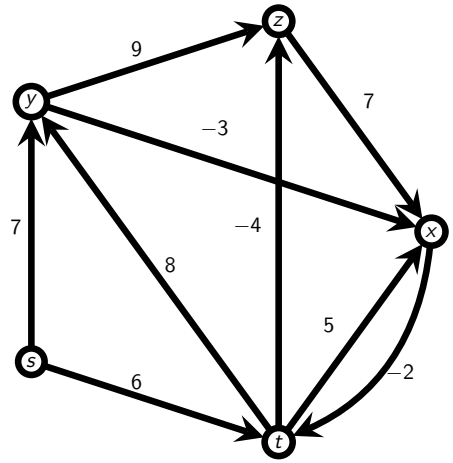
Le déroulement complet étant long, on ne vous demandera généralement les détails que pour une passe sur tous les arcs.



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 . . .)

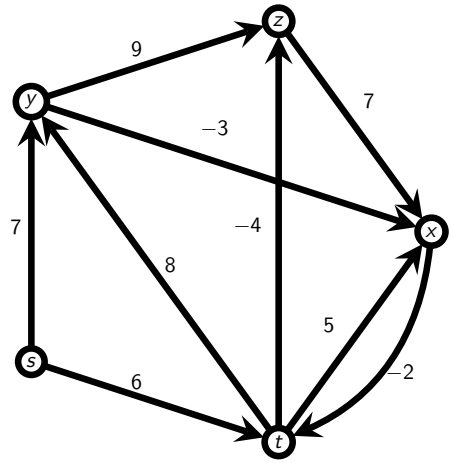
Le déroulement complet étant long, on ne vous demandera généralement les détails que pour une passe sur tous les arcs.



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 . . .)

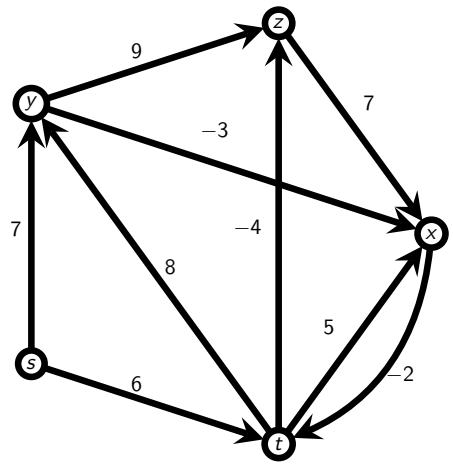
Le déroulement complet étant long, on ne vous demandera généralement les détails que pour une passe sur tous les arcs.



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 . . .)

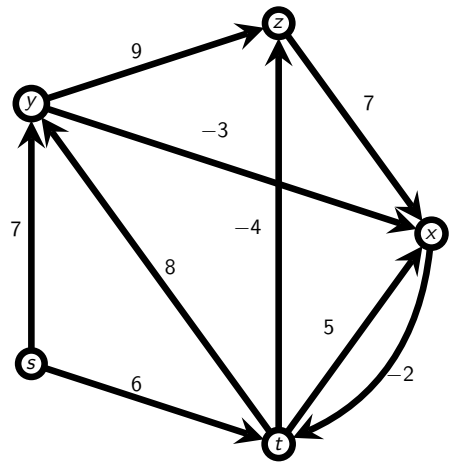
Le déroulement complet étant long, on ne vous demandera généralement les détails que pour une passe sur tous les arcs.



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 . . .)

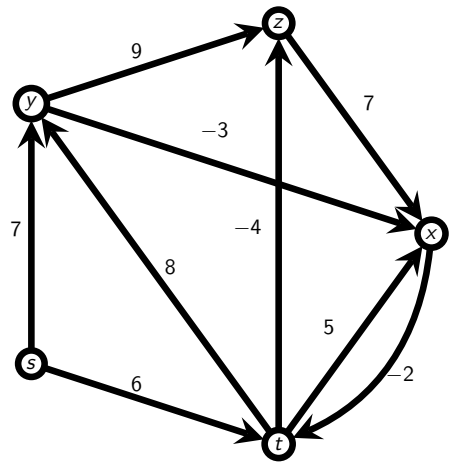
Le déroulement complet étant long, on ne vous demandera généralement les détails que pour une passe sur tous les arcs.



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 . . .)

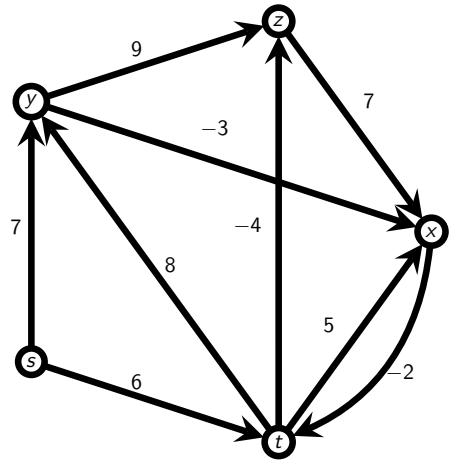
Le déroulement complet étant long, on ne vous demandera généralement les détails que pour une passe sur tous les arcs.



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 . . .)

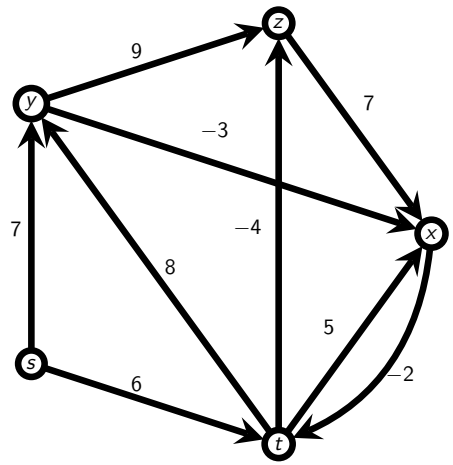
Le déroulement complet étant long, on ne vous demandera généralement les détails que pour une passe sur tous les arcs.



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 . . .)

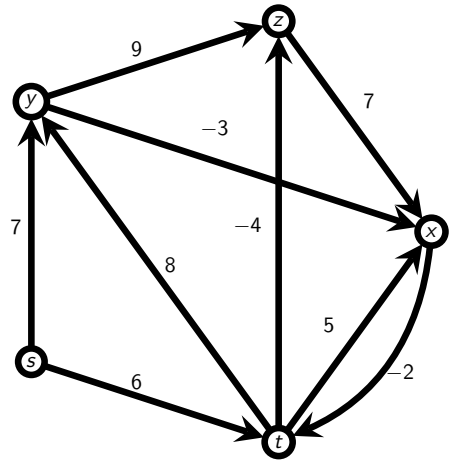
Le déroulement complet étant long, on ne vous demandera généralement les détails que pour une passe sur tous les arcs.



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 ?
  - tout chemin contient au plus $|V| - 1$ arcs ;

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 ?
  - tout chemin contient au plus $|V| - 1$ arcs ;
  - 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 proprement dit

Algorithme 1 : 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.

```
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 dans le cas orienté ;

Dijkstra vs. Bellman-Ford

- Quand utiliser Dijkstra ?
  - quand tous les poids sont positifs ou nuls ;
  - sa complexité est meilleure que Bellman-Ford dans le cas orienté ;
- 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 dans le cas orienté ;
- Quand utiliser Bellman-Ford ?
  - quand le graphe est orienté avec 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 dans le cas orienté ;
- Quand utiliser Bellman-Ford ?
  - quand le graphe est orienté avec des poids négatifs ;
- S'il y a des cycles négatifs dont le chemin cherché peut tirer profit, on ne peut rien faire ;

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

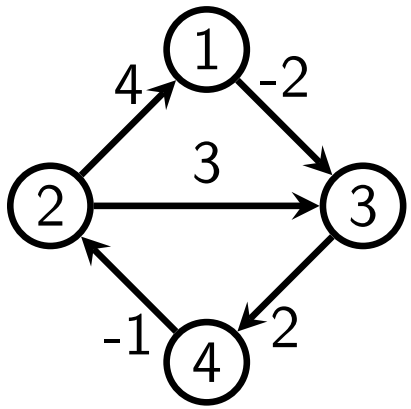
- L'algorithme de Floyd-Warshall procède comme suit :
  - au départ, les plus courts chemins entre chaque paire de sommets sont les poids des arcs les reliant (ou $+\infty$) ;
  - à la $k^{\text{ème}}$ itération, on cherche à améliorer le chemin $u \rightsquigarrow v$ en s'autorisant les sommets intermédiaires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

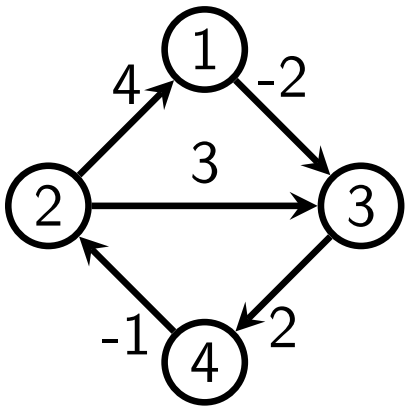
Déroulement de l'algorithme de Floyd-Warshall

Exemple 3



Déroulement de l'algorithme de Floyd-Warshall

Exemple 3

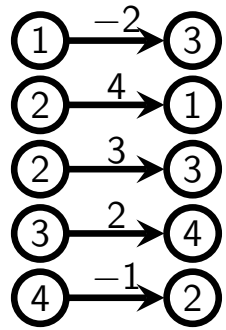


	1	2	3	4
1	0	$+\infty$	-2	$+\infty$
2	4	0	3	$+\infty$
3	$+\infty$	$+\infty$	0	2
4	$+\infty$	-1	$+\infty$	0

matrice de distances

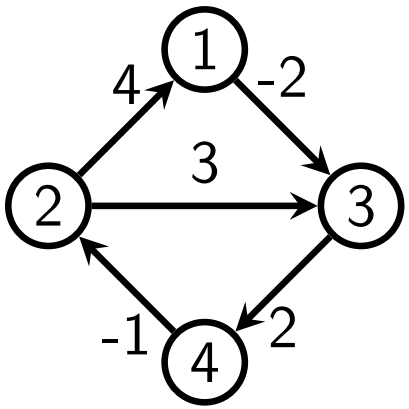
Les chemins

$k = 0$



Déroulement de l'algorithme de Floyd-Warshall

Exemple 3

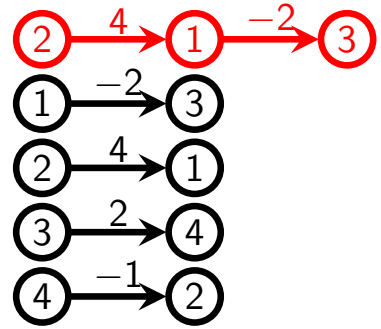


	1	2	3	4
1	0	$+\infty$	-2	$+\infty$
2	4	0	3	$+\infty$
3	$+\infty$	$+\infty$	0	2
4	$+\infty$	-1	$+\infty$	0

matrice de distances

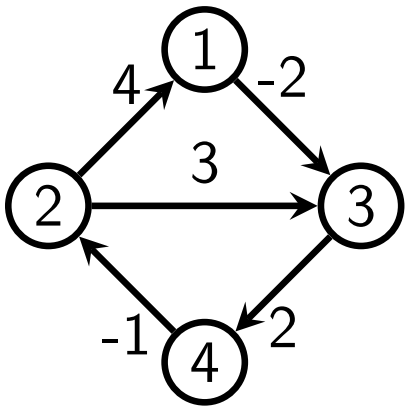
Les chemins

$k = 1$



Déroulement de l'algorithme de Floyd-Warshall

Exemple 3

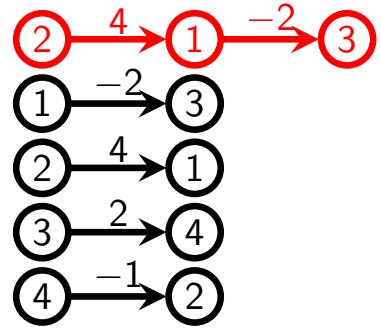


	1	2	3	4
1	0	$+\infty$	-2	$+\infty$
2	4	0	2	$+\infty$
3	$+\infty$	$+\infty$	0	2
4	$+\infty$	-1	$+\infty$	0

matrice de distances

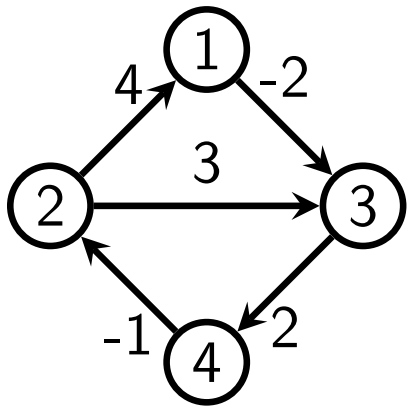
Les chemins

$k = 1$



Déroulement de l'algorithme de Floyd-Warshall

Exemple 3

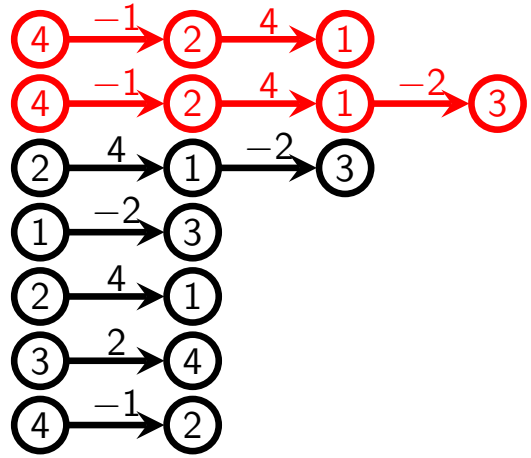


	1	2	3	4
1	0	$+\infty$	-2	$+\infty$
2	4	0	2	$+\infty$
3	$+\infty$	$+\infty$	0	2
4	$+\infty$	-1	$+\infty$	0

matrice de distances

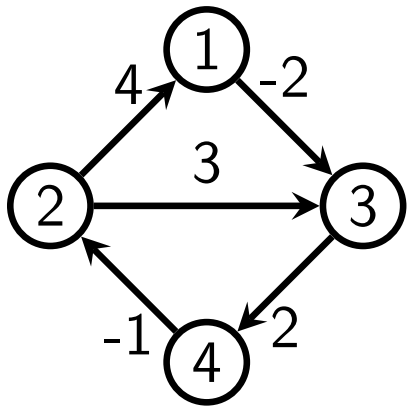
Les chemins

$k = 2$



Déroulement de l'algorithme de Floyd-Warshall

Exemple 3

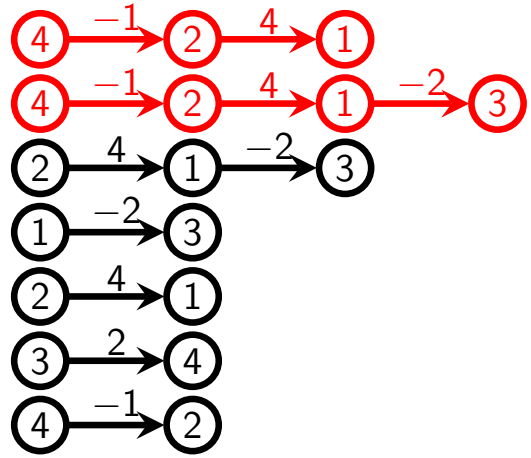


	1	2	3	4
1	0	$+\infty$	-2	$+\infty$
2	4	0	2	$+\infty$
3	$+\infty$	$+\infty$	0	2
4	3	-1	1	0

matrice de distances

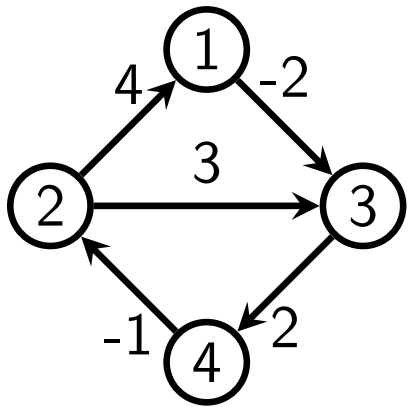
Les chemins

$k = 2$



Déroulement de l'algorithme de Floyd-Warshall

Exemple 3

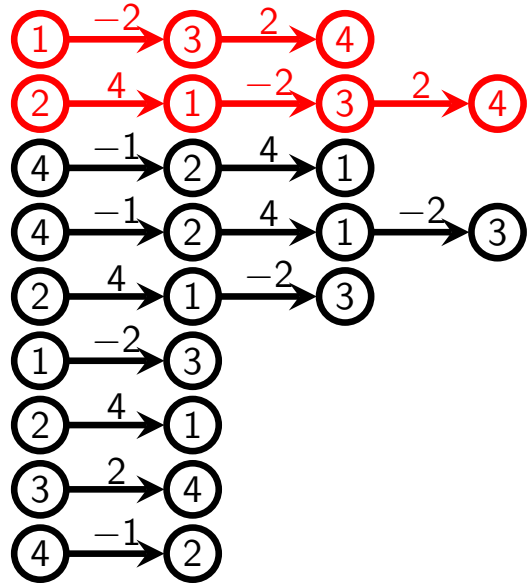


	1	2	3	4
1	0	$+\infty$	-2	$+\infty$
2	4	0	2	$+\infty$
3	$+\infty$	$+\infty$	0	2
4	3	-1	1	0

matrice de distances

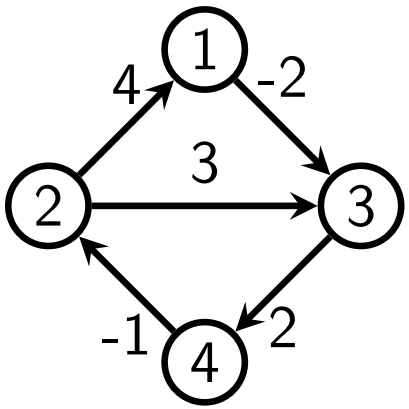
Les chemins

$k = 3$



Déroulement de l'algorithme de Floyd-Warshall

Exemple 3

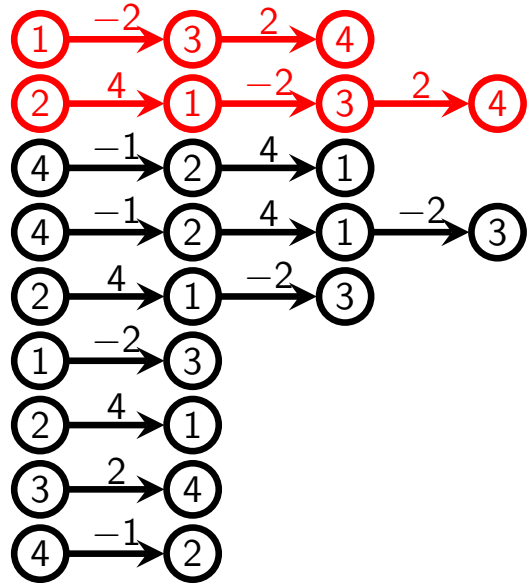


	1	2	3	4
1	0	$+\infty$	-2	0
2	4	0	2	4
3	$+\infty$	$+\infty$	0	2
4	3	-1	1	0

matrice de distances

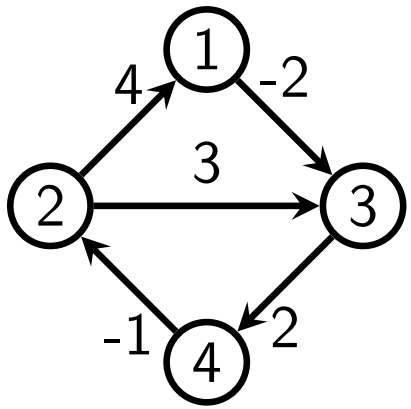
Les chemins

$k = 3$



Déroulement de l'algorithme de Floyd-Warshall

Exemple 3

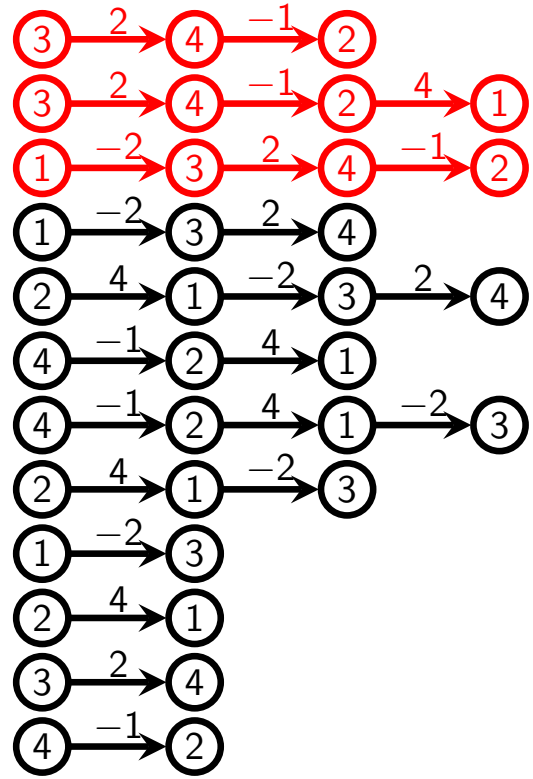


	1	2	3	4
1	0	$+\infty$	-2	0
2	4	0	2	4
3	$+\infty$	$+\infty$	0	2
4	3	-1	1	0

matrice de distances

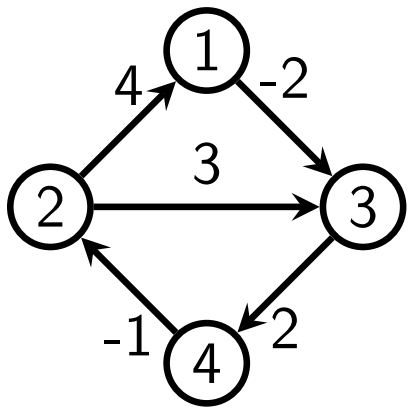
Les chemins

$k = 4$



Déroulement de l'algorithme de Floyd-Warshall

Exemple 3

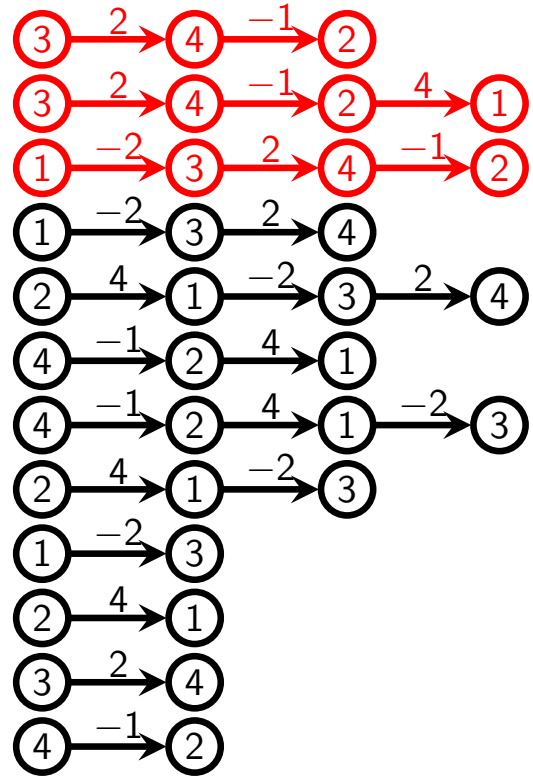


	1	2	3	4
1	0	-1	-2	0
2	4	0	2	4
3	5	1	0	2
4	3	-1	1	0

matrice de distances

Les chemins

$k = 4$



L'algorithme de Floyd-Warshall proprement dit

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

Algorithme 2 : 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]);
```

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 ;

Techniques algorithmiques

- Les algorithmes vus jusqu'ici sont des exemples de techniques algorithmiques dont on reparlera ;

Techniques algorithmiques

- Les algorithmes vus jusqu'ici sont des exemples de techniques algorithmiques dont on reparlera ;
- On a vu des algorithmes :

Techniques algorithmiques

- Les algorithmes vus jusqu'ici sont des exemples de techniques algorithmiques dont on reparlera ;
- On a vu des algorithmes :
  1. **gloutons** : Dijkstra, Kruskal, Prim ;

Techniques algorithmiques

- Les algorithmes vus jusqu'ici sont des exemples de techniques algorithmiques dont on reparlera ;
- On a vu des algorithmes :
  1. **gloutons** : Dijkstra, Kruskal, Prim ;
  2. **de programmation dynamique** : Bellman-Ford, Floyd-Warshall ;

Algorithmes gloutons

- Les **algorithmes gloutons** (*greedy* en anglais) font des choix localement optimaux selon le(s) critère(s) visé(s) ;

Algorithmes gloutons

- Les **algorithmes gloutons** (*greedy* en anglais) font des choix localement optimaux selon le(s) critère(s) visé(s) ;
  - dans un problème de minimisation, on sélectionne le choix le moins cher à chaque étape ;

Algorithmes gloutons

- Les **algorithmes gloutons** (*greedy* en anglais) font des choix localement optimaux selon le(s) critère(s) visé(s) ;
  - dans un problème de minimisation, on sélectionne le choix le moins cher à chaque étape ;
  - dans un problème de maximisation, on sélectionne le choix le plus profitable à chaque étape ;

Algorithmes gloutons

- Les **algorithmes gloutons** (*greedy* en anglais) font des choix localement optimaux selon le(s) critère(s) visé(s) ;
  - dans un problème de minimisation, on sélectionne le choix le moins cher à chaque étape ;
  - dans un problème de maximisation, on sélectionne le choix le plus profitable à chaque étape ;
- Kruskal et Prim suivent clairement ce schéma : à chaque étape, on sélectionne l'arête de poids minimum (+ autres conditions) ;

Bilan des algorithmes gloutons

- Une fois qu'on a compris le principe, les algorithmes gloutons se révèlent très utiles :

Bilan des algorithmes gloutons

- Une fois qu'on a compris le principe, les algorithmes gloutons se révèlent très utiles :
  - cette approche sert de bon "candidat de départ" si on n'a pas d'autre idée ;

Bilan des algorithmes gloutons

- Une fois qu'on a compris le principe, les algorithmes gloutons se révèlent très utiles :
  - cette approche sert de bon "candidat de départ" si on n'a pas d'autre idée ;
  - elle est généralement simple à implémenter ;

Bilan des algorithmes gloutons

- Une fois qu'on a compris le principe, les algorithmes gloutons se révèlent très utiles :
  - cette approche sert de bon "candidat de départ" si on n'a pas d'autre idée ;
  - elle est généralement simple à implémenter ;
- Malheureusement, rares sont les situations où ils sont optimaux ;

Bilan des algorithmes gloutons

- Une fois qu'on a compris le principe, les algorithmes gloutons se révèlent très utiles :
  - cette approche sert de bon "candidat de départ" si on n'a pas d'autre idée ;
  - elle est généralement simple à implémenter ;
- Malheureusement, rares sont les situations où ils sont optimaux ;
- Ils restent très utiles dans la conception d'**algorithmes d'approximation** ;

Programmation dynamique

- Les algorithmes gloutons échouent quand une solution optimale ne peut pas s'obtenir en se contentant de combiner des choix optimaux localement ;

Programmation dynamique

- Les algorithmes gloutons échouent quand une solution optimale ne peut pas s'obtenir en se contentant de combiner des choix optimaux localement ;
- La **programmation dynamique** résoud ce problème en examinant aussi les choix non optimaux localement ;

Programmation dynamique

- Les algorithmes gloutons échouent quand une solution optimale ne peut pas s'obtenir en se contentant de combiner des choix optimaux localement ;
- La **programmation dynamique** résoud ce problème en examinant aussi les choix non optimaux localement ;
- De manière très informelle :

Programmation dynamique

- Les algorithmes gloutons échouent quand une solution optimale ne peut pas s'obtenir en se contentant de combiner des choix optimaux localement ;
- La **programmation dynamique** résoud ce problème en examinant aussi les choix non optimaux localement ;
- De manière très informelle :
  - pour chaque choix possible à une étape donnée, on évalue l'impact de ce choix ;

Programmation dynamique

- Les algorithmes gloutons échouent quand une solution optimale ne peut pas s'obtenir en se contentant de combiner des choix optimaux localement ;
- La **programmation dynamique** résoud ce problème en examinant aussi les choix non optimaux localement ;
- De manière très informelle :
  - pour chaque choix possible à une étape donnée, on évalue l'impact de ce choix ;
  - on optimise ensuite sur toutes les décisions possibles, en gardant le critère global en tête ;

Programmation dynamique

- La programmation dynamique ne se fait en général pas avoir comme les algorithmes gloutons ;

Programmation dynamique

- La programmation dynamique ne se fait en général pas avoir comme les algorithmes gloutons ;
- Cela ne veut pas dire qu'elle fonctionne toujours !

Programmation dynamique

- La programmation dynamique ne se fait en général pas avoir comme les algorithmes gloutons ;
- Cela ne veut pas dire qu'elle fonctionne toujours !
- Le problème étudié doit présenter une certaine structure : on parle de "sous-structure optimale", ce qui signifie que l'on peut atteindre l'optimum global en résolvant des sous-problèmes de manière optimale ;