Algorithmes de plus court chemin

Heike Ripphausen-Lipa - Beuth University of Applied Science - Berlin
J.M. Adam - Université de Grenoble Alpes - Grenoble

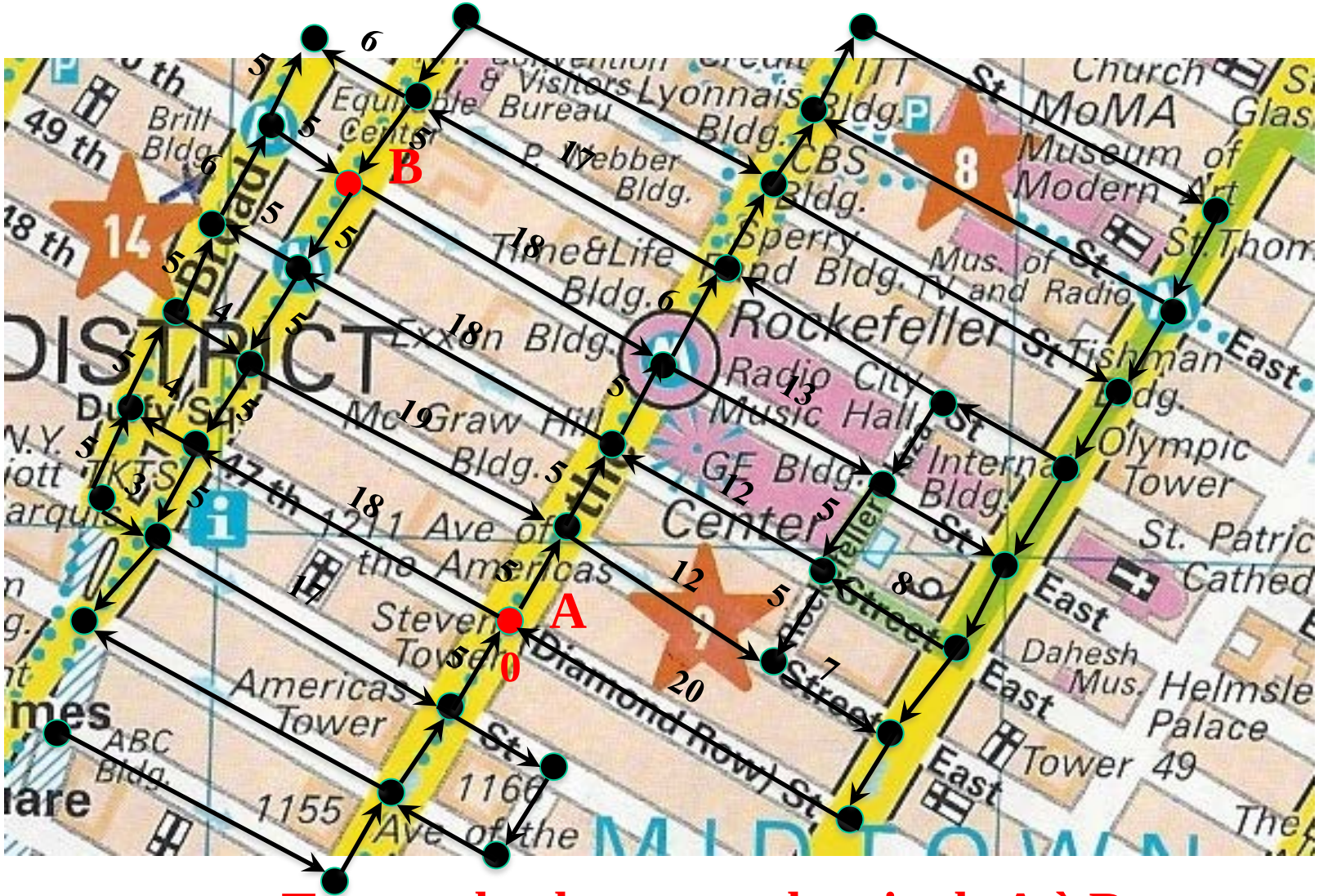
Objectif

- Se familiariser avec les différentes sortes de problèmes de plus court chemin
- Savoir quel algorithme peut être appliqué efficacement pour quel type de problème de plus courts chemins

Sommaire

- Introduction
- Algorithme de Dijkstra
- Algorithme de Bellman-Ford
- Algorithme de Floyd-Warshall

Problème: trouver le chemin le plus court



Introduction

On considère un graphe orienté pondéré :

- Graph **`G = (V,E)`** avec

- Fonction poids **`w: E -> R`**

(les poids des arcs sont des nombres réels)

Définition: Poids d‘un chemin

Le ***poids w d‘un chemin p*** avec

$p = (v_0, v_1, \ldots, v_k)$ et $(v_i, v_{i+1}) \in E$

Est défini ainsi :

$$w(p) = \sum_{i=1..k} w(v_{i-1}, v_i)$$

Definition: poids du chemin le plus court

Le **poids du chemin le plus court** $\delta(u,v)$ entre deux sommets **u** et **v** est défini ainsi :

$$\delta(u,v) = \begin{cases} \min \{ w(p) : p \text{ est un chemin de } u \text{ à } v \text{ dans } G \}, \text{ si un chemin existe} \\ \infty \text{ , sinon} \end{cases}$$

Idée de base pour trouver le chemin le plus court

De nombreux algorithmes de calcul de plus court chemin utilisent la propriéte suivante :

Les sous-chemins des chemins les plus courts sont eux-mêmes les chemins les plus courts !

Plus précisément:

si $\mathbf{p = (v_0, v_1, \ldots, v_k)}$ est un chemin le plus court entre les sommets $\mathbf{v_0}$ et $\mathbf{v_k}$

alors pour $\mathbf{i}$ et $\mathbf{j}$ $\mathbf{(0 \leq i \leq j \leq k)}$ le chemin $\mathbf{p_{ij} = (v_i, v_{i+1}, \ldots, v_j)}$ est le chemin le plus court du sommet $\mathbf{v_i}$ au sommet $\mathbf{v_j}$.

Idée de base pour trouver le chemin le plus court

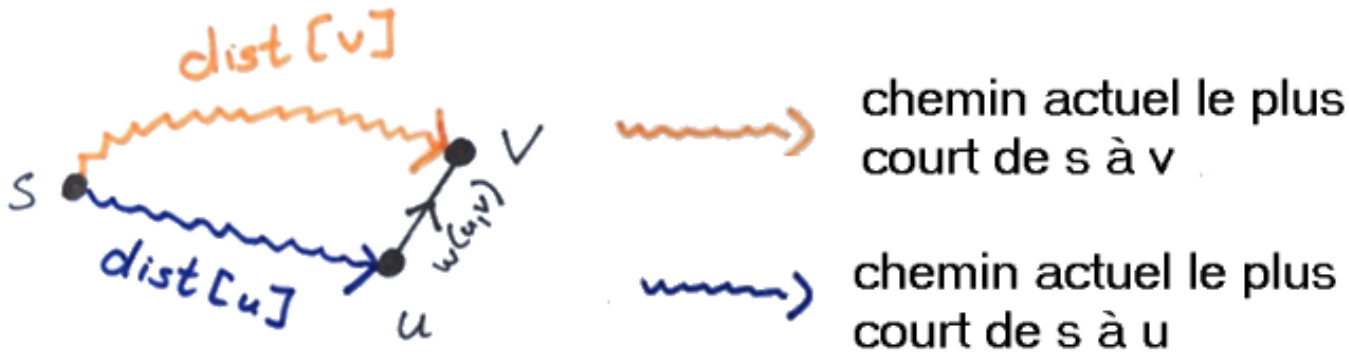
- Nous nous concentrons sur le problème des **chemins les plus courts d'une source unique,** c'est-à-dire que l'on calcule les chemins les plus courts du sommet de départ **s** vers tous les sommets **v** du graphe.

- Beaucoup d'algorithme de calcul de plus court chemin utilisent la technique de ***relaxation*** :

  A chaque sommet **v** on associe un attribut **`dist`** qui donne une borne supérieure de la distance du plus court chemin d'un sommet **s** (la source) au sommet **v**.

Relaxation

La méthode ***relaxation*** d'un arc **(u, v)** consiste à voir si le chemin le plus court calculé jusqu'à **v** peut être amélioré en prenant le chemin le plus court actuel de **s** vers **u,** en lui ajoutant l'arc **(u, v)**:



relaxation si dist[v] > dist[u] + w(u,v)

initialisation-Source-Unique

```
// Initialisation d'un algorithme de relaxation
// pour la recherche des plus courts chemins
// depuis une source unique : le sommet s
initialisation-Source-Unique(G, s)
    pourtout sommet v de G
        dist[v] ← VMAX  // valeur réelle max
        pred[v] ← NIL
    fpour
    dist[s] ← 0
```

Relax(u,v,w)

```
Relax(u, v, w)
  // arc (u,v), w(u,v) est le poids de l'arc
  si dist[v] > dist[u] + w(u,v)
  alors
       dist[v] ← dist[u] + w(u,v)
       pred[v] ← u
  fsi
```

Algorithme de Dijkstra

(publication 1959)

Algorithme de Dijkstra

- L’ Algorithme de Dijkstra résout le problème des chemins les plus courts depuis une source unique, le sommet **s .**
- L’ Algorithme de Dijkstra ne fonctionne que pour des poids positifs.
- Il fonctionne de manière similaire au parcours en largeur, mais au lieu d‘utiliser une file, il utilise une file prioritaire.

Algorithme de Dijkstra

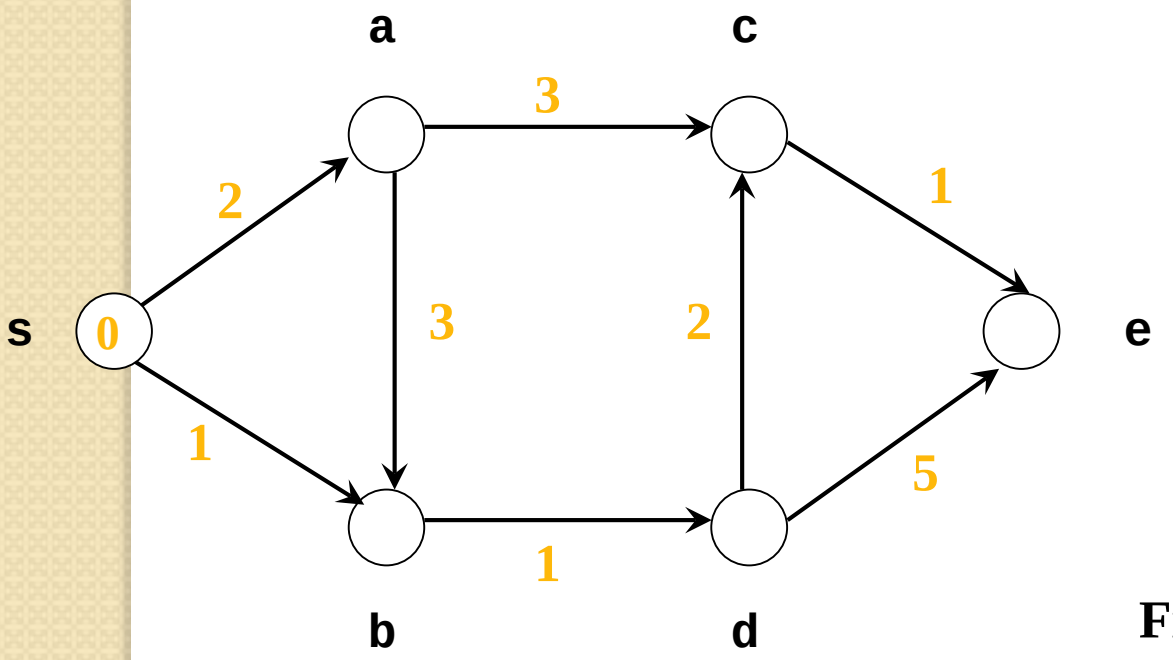
- L'algorithme de Dijkstra gère un ensemble (virtuel) **`S`** qui contient tous les sommets pour lesquels les distances les plus courtes depuis s sont déjà calculées.
- Etape par étape, le sommet **`u`** de **`S`** dont la valeur **`dist`** est minimale, est sélectioné et tous les arcs **`(u,v)`**incidents à **`u`** sont relaxés
- Pour gérer l'ensemble **`S`,** des sommets dont la distance minimale depuis s n'a pas encore été déterminée, on utilise la file prioritaire **`Q`**.

Algorithme de Dijkstra

```
Dijkstra(G, w, s)
   initialisation-Source-Unique(G, s)
   S ← Ø;   // tous les sommets atteints depuis s
   Q ← tous les sommets    // tous les sommets dans Q
   tantque Q ≠ Ø
      u ← Extraire-Min(Q)     // sommet dont dist minimale
      S ← S.ajouter(u)
      pour tout sommet v adjacent à u
          Relax(u,v,w);
      fpour
   ftq
```

Remarque: l'ensemble de sommets S n'est pas vraiment utile; il est introduit pour mieux illustrer l'algorithme.

Algorithme de Dijkstra



i Valeur de dist : `i`

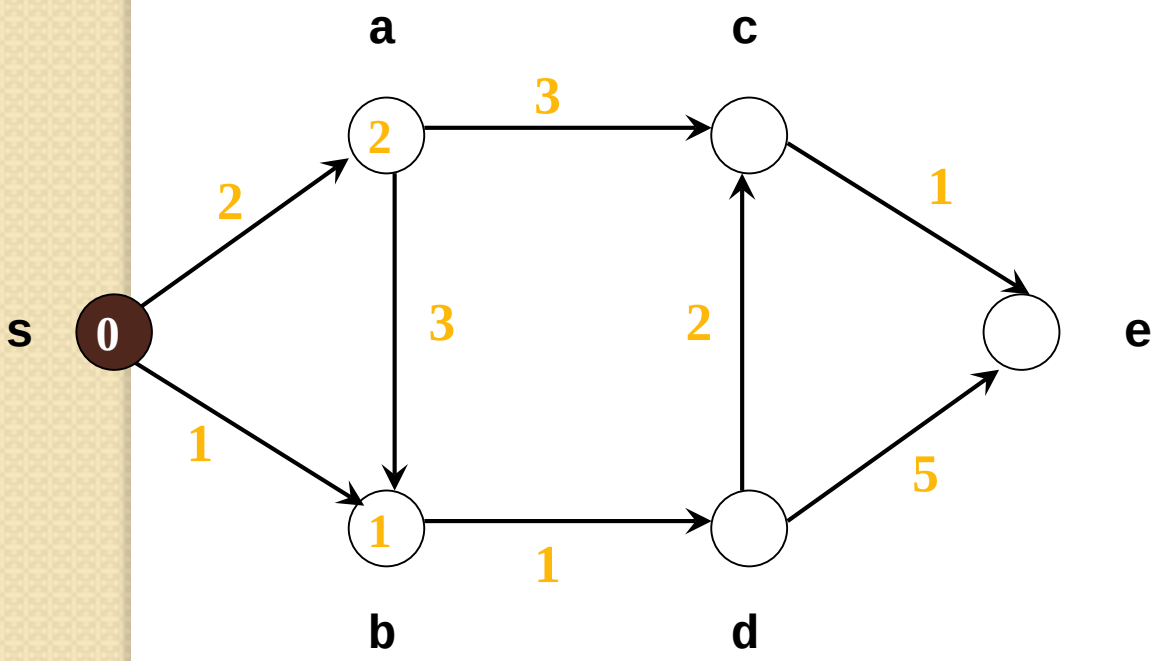
Vide : VMAX

File prioritaire Q

s	a	b	c	d	e

Remarque : Q est ordonnée selon les valeurs croisantes de **`dist`**

Algorithme de Dijkstra

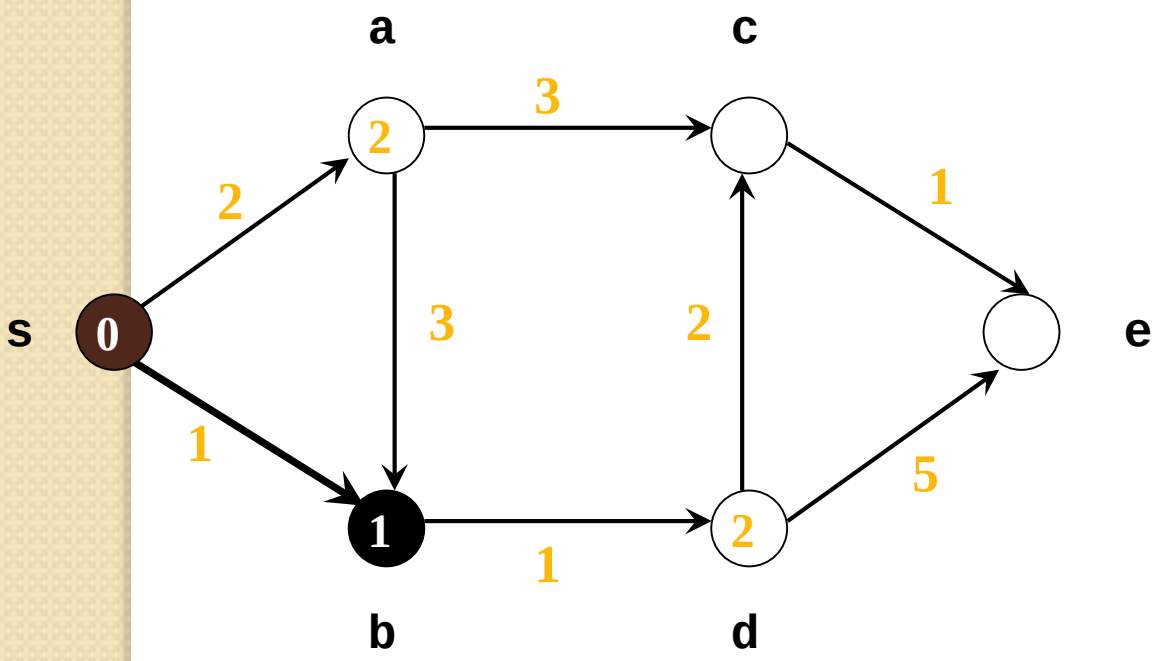


File prioritaire Q

b	a	c	d	e

Les sommets de S sont colorés en noir

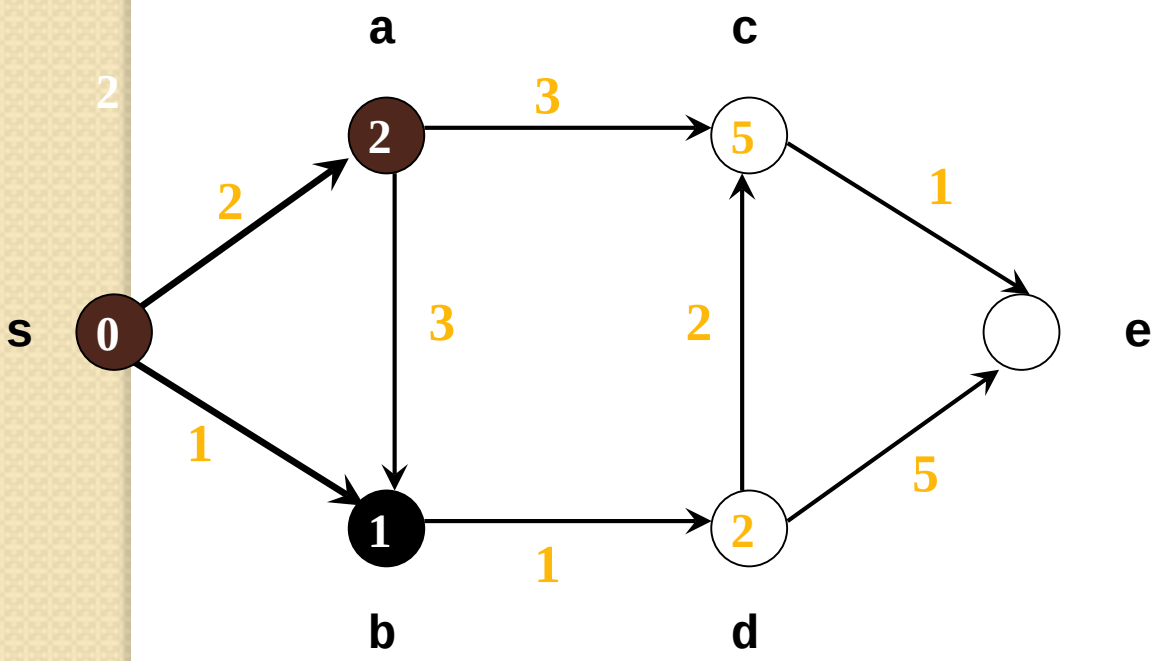
Algorithme de Dijkstra



File prioritaire Q

a	d	c	e

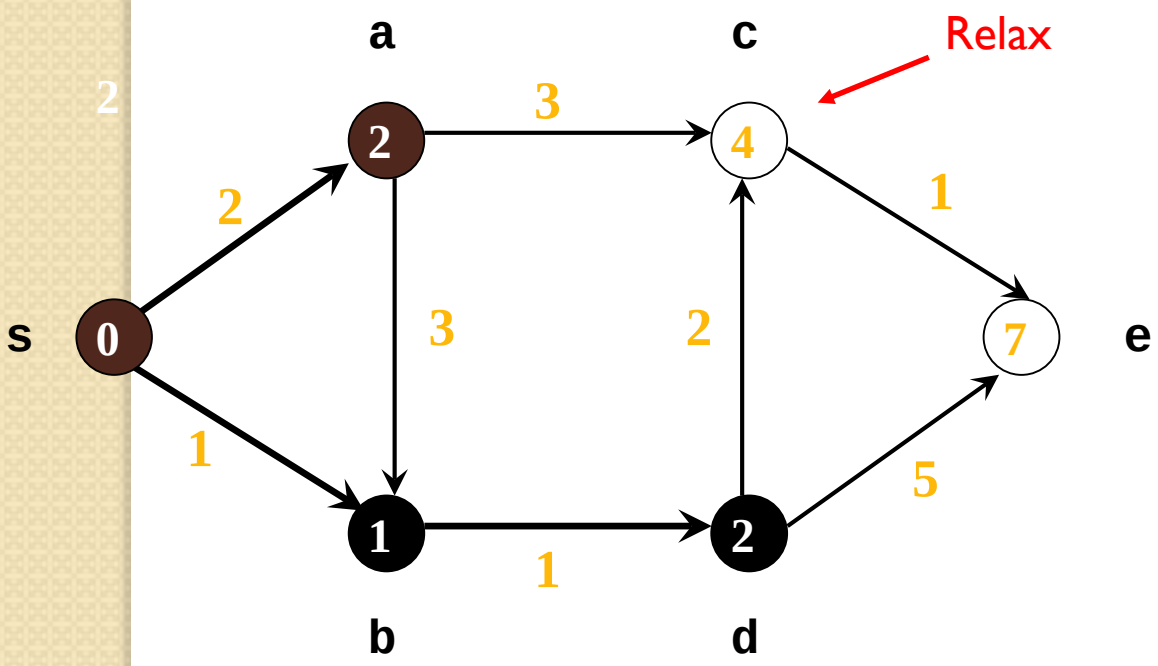
Algorithme de Dijkstra



File prioritaire Q

d	c	e

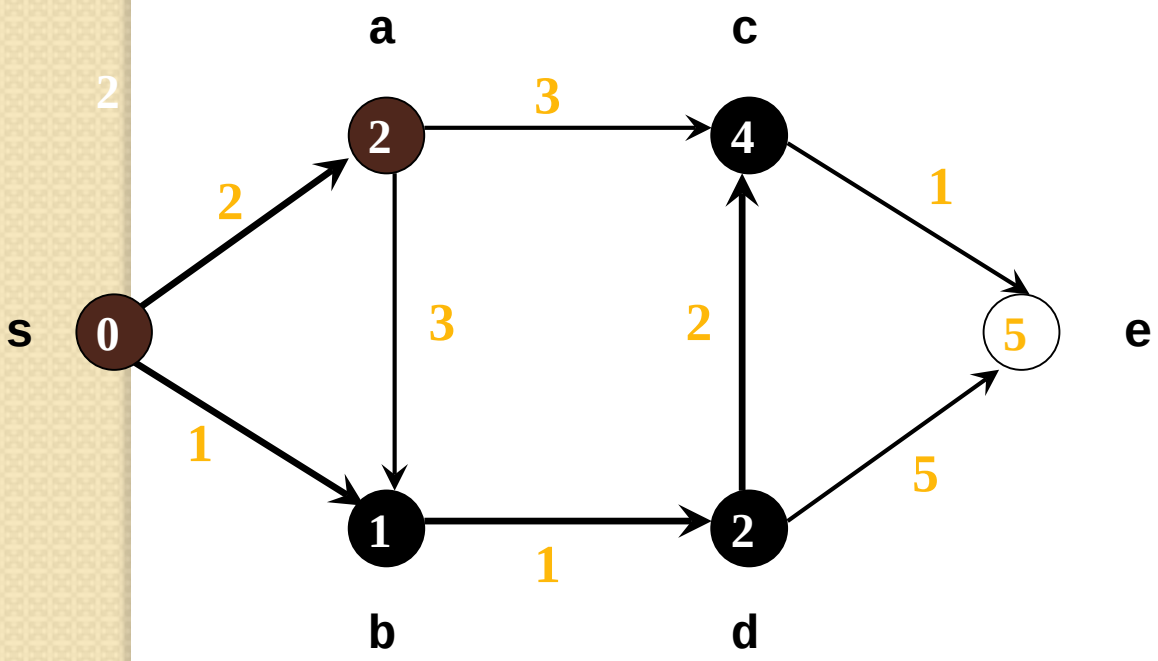
Algorithme de Dijkstra



File prioritaire Q

c	e

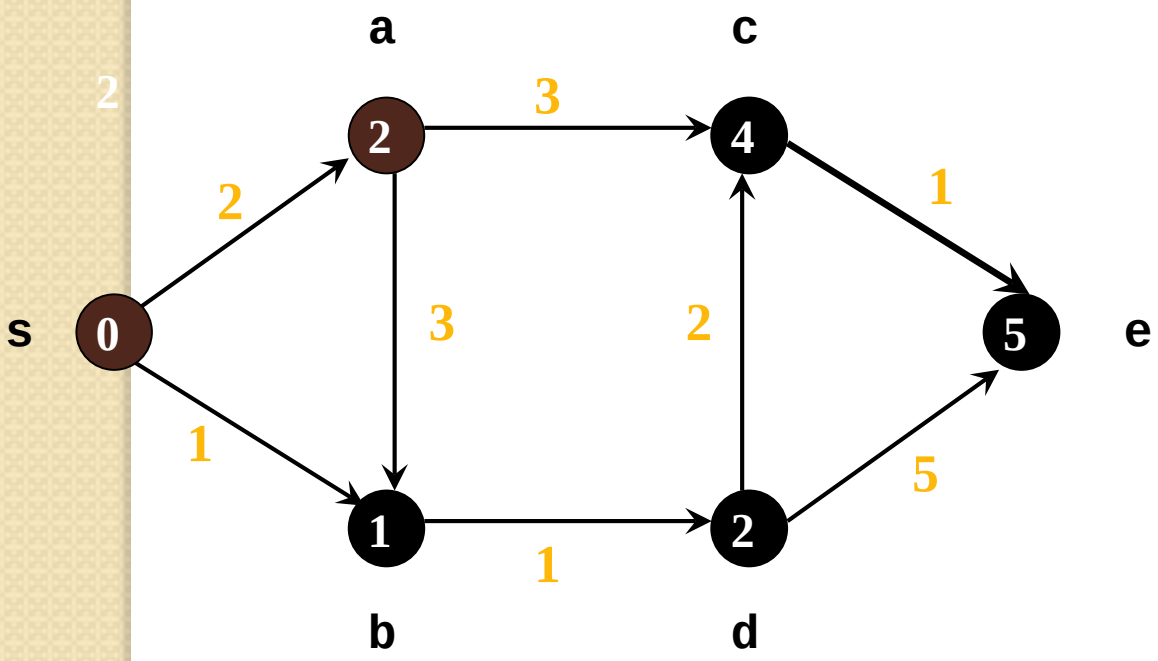
Algorithme de Dijkstra



File prioritaire Q

e

Algorithme de Dijkstra



File prioritaire Q

Algorithme de Dijkstra dans une table

Initialisation

sommet	**s**	**a**	**b**	**c**	**d**	**e**
pred	NIL	NIL	NIL	NIL	NIL	NIL
dist	0	∞	∞	∞	∞	∞

Après avoir supprimé s de Q

sommet	**s**	**a**	**b**	**c**	**d**	**e**
pred	NIL	s	s	NIL	NIL	NIL
dist	0	2	1	∞	∞	∞

Algorithme de Dijkstra dans une table

Après avoir supprimé b de Q

sommet	s	a	b	c	d	e
pred	NIL	s	s	NIL	b	NIL
dist	0	2	1	∞	2	∞

Après avoir supprimé a de Q

sommet	s	a	b	c	d	e
pred	NIL	s	s	a	b	NIL
dist	0	2	1	5	2	∞

Algorithme de Dijkstra dans une table

Après avoir supprimé d de Q

sommet	s	a	b	c	d	e
pred	NIL	s	s	d	b	d
dist	0	2	1	4	2	7

Après avoir supprimé c de Q

sommet	s	a	b	c	d	e
pred	NIL	s	s	d	b	c
dist	0	2	1	4	2	5

Algorithme de Dijkstra dans une table

Après avoir supprimé e de Q

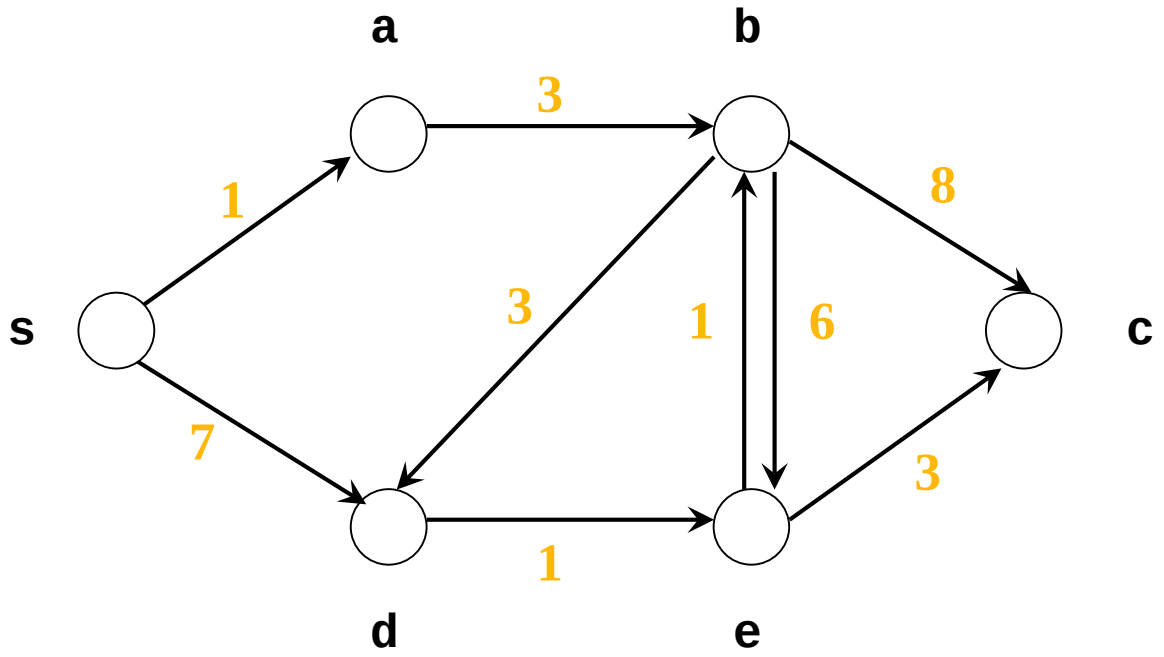
sommet	s	a	b	c	d	e
pred	NIL	s	s	d	b	c
dist	0	2	1	4	2	5

Détermination du plus court chemin de s à e :

e ← c ← d ← b ← **s**

Exercice: Algorithme de Dijkstra

Avec l'algorithme de Dijkstra déterminez tous les Chemins les plus courts partant du sommet s



Temps d'exécution de l'algorithme de Dijkstra

Temps d'exécution total

```
Initialize-Source-Unique(G, s)

S ← Ø;
Q ← V(G);

tantque (Q ≠ Ø)
   u ← Extraire-Min(Q)
   S ← S.ajouter(u)
   pour tout sommet v adjacent à u
       Relax(u,v,w)
  fpour
ftq
```

O(n)

O(1)

O(n)

O(n)

O(n * log n)

O(n)

O(m)

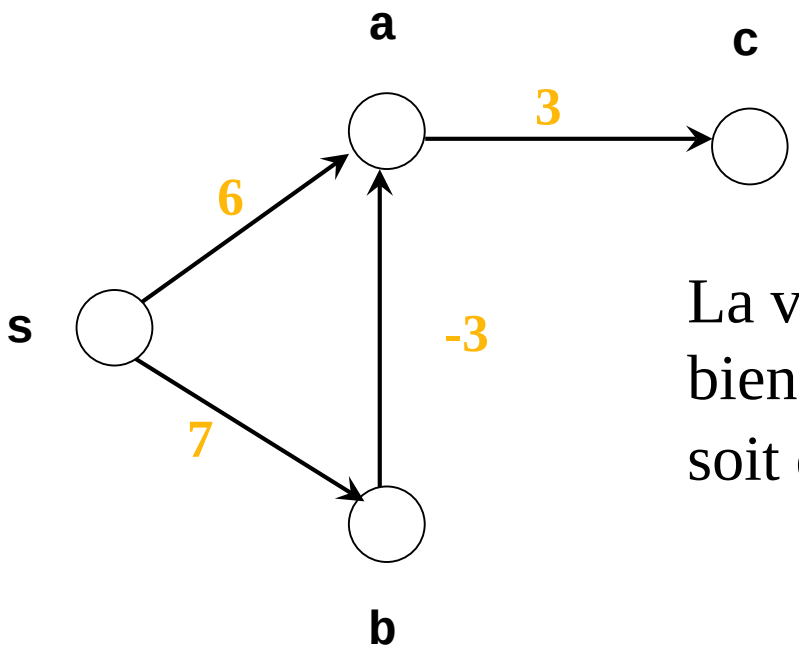
Extraire-Min dans la queue pripritaire Q

O(n* log n + m) = O(n^2)

Exercice: poids négatif

Donnez un exemple montrant que l‘algorithme de Dijkstra ne fonctionne pas avec un arc portant un poids négatif.

Exercice: poids négatif

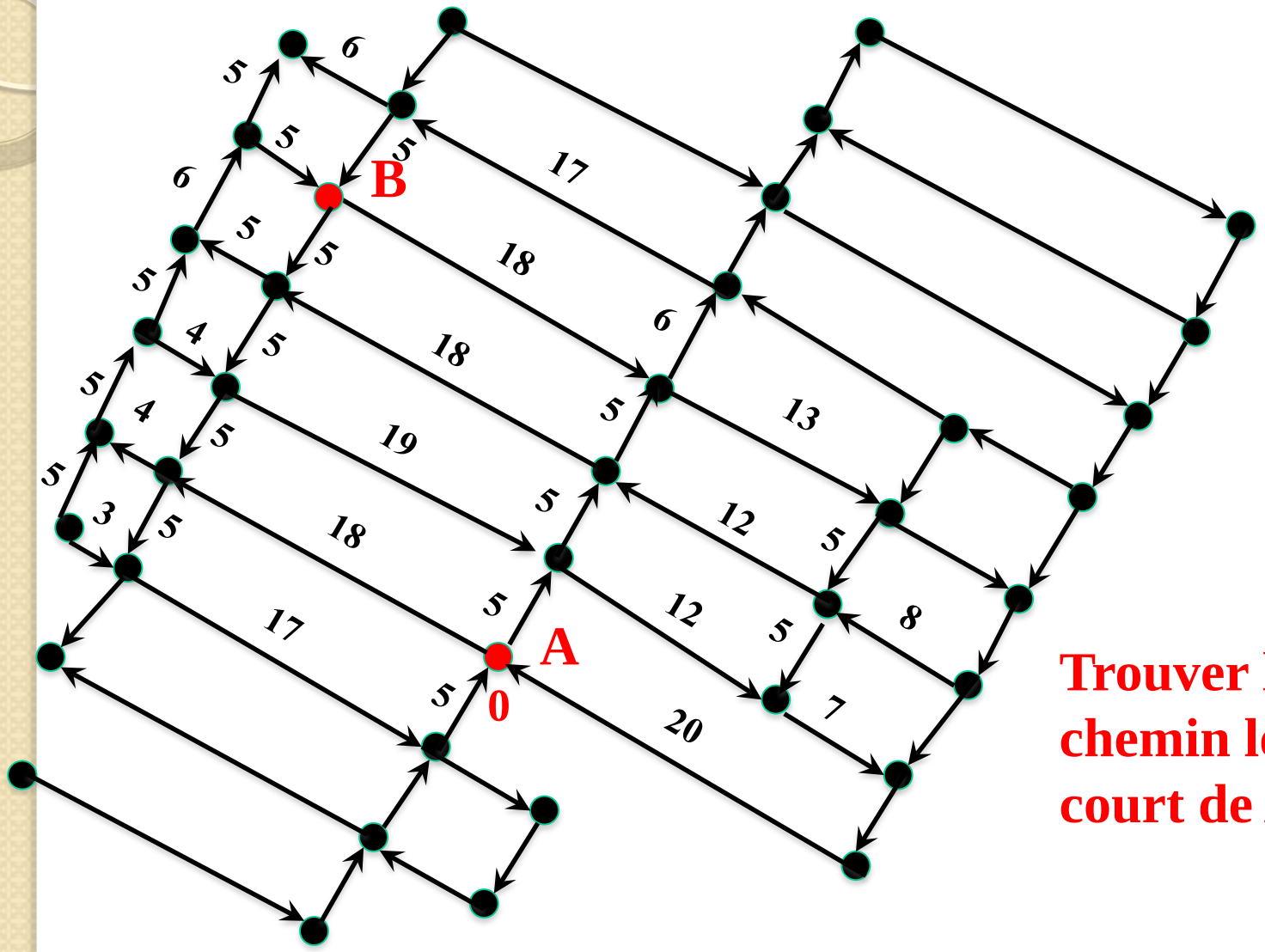


La valeur **`dist`** de **`c`** reste 9 bien que la valeur **`dist`** de **`a`** soit diminuée de 6 à 4

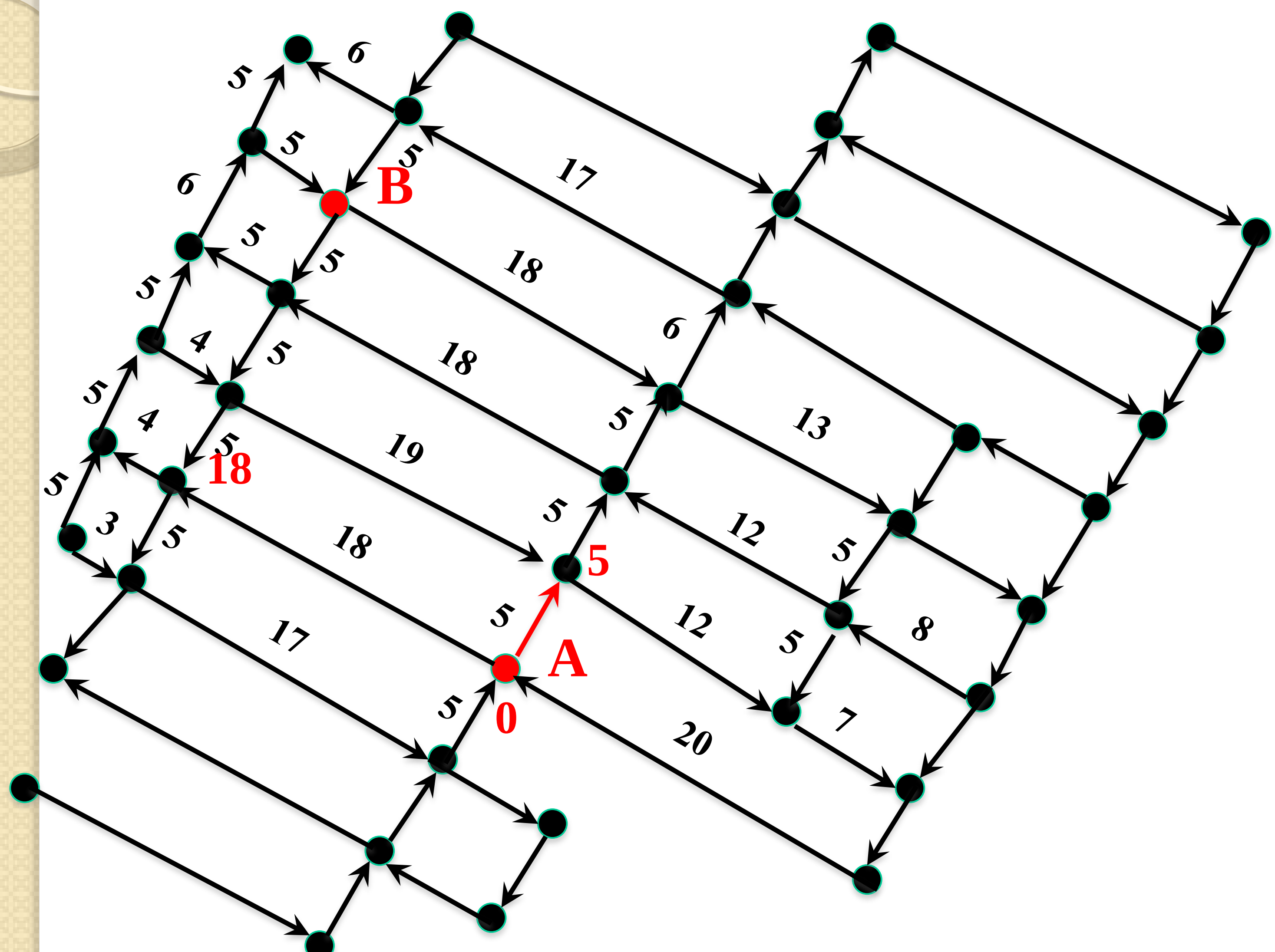
Problème avec l'algorithme de Dijkstra

L'algorithme de Dijkstra n'est pas très "orienté cible" : dans le cas où un chemin particulier de la source vers la destination doit être trouvé, l'algorithme n'est pas très performant.

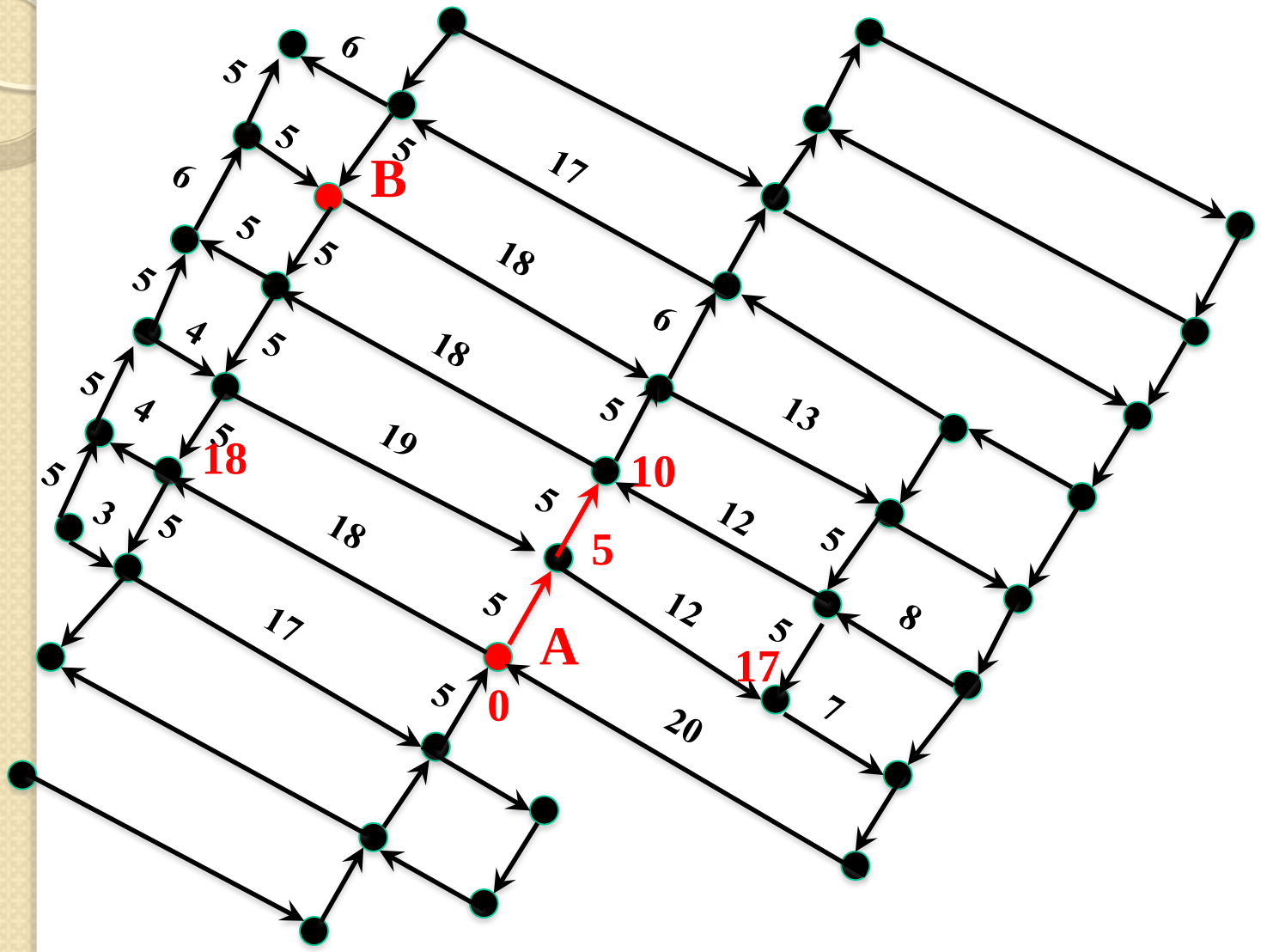
Algorithme de Dijkstra



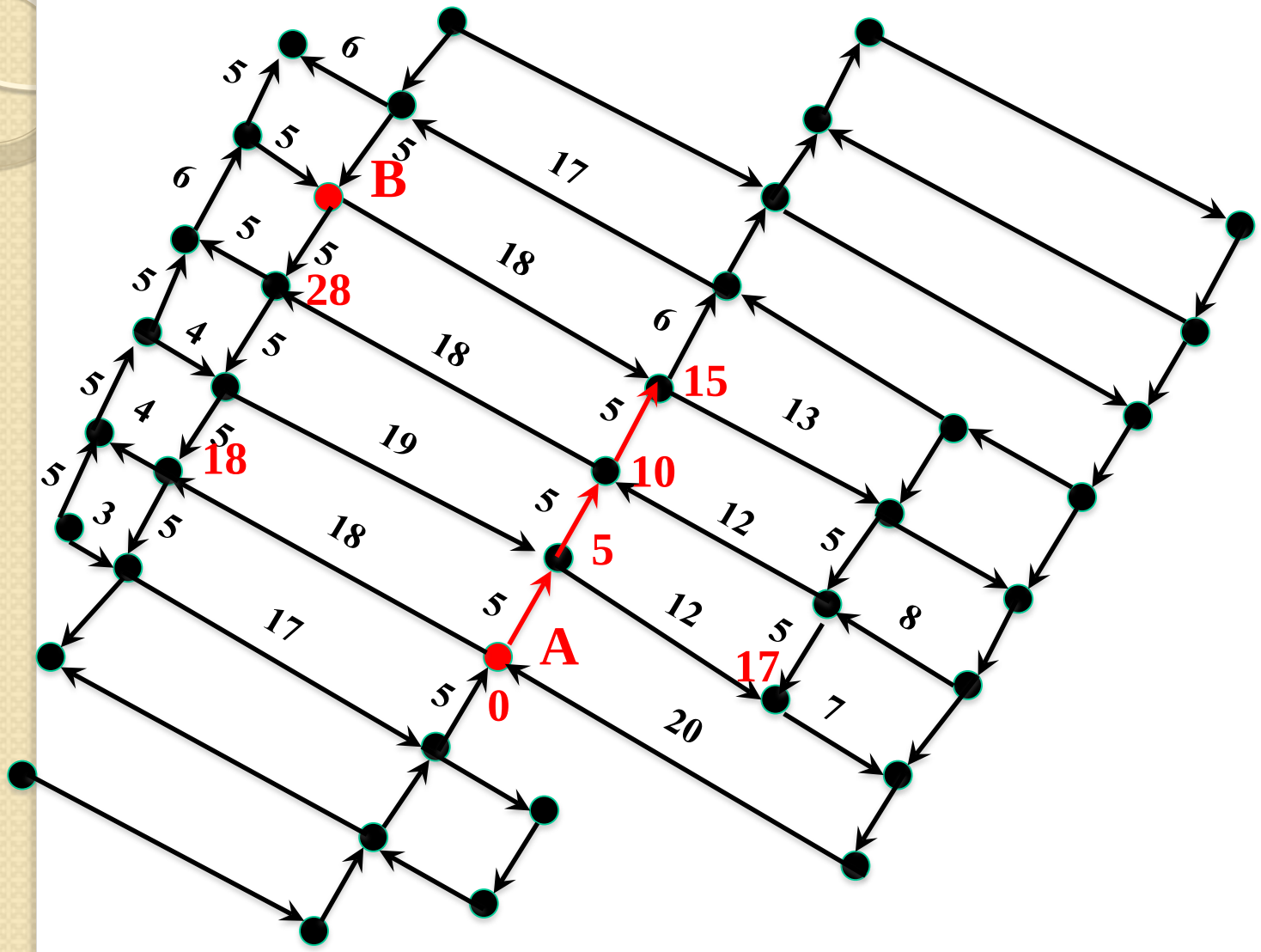
Algorithme de Dijkstra



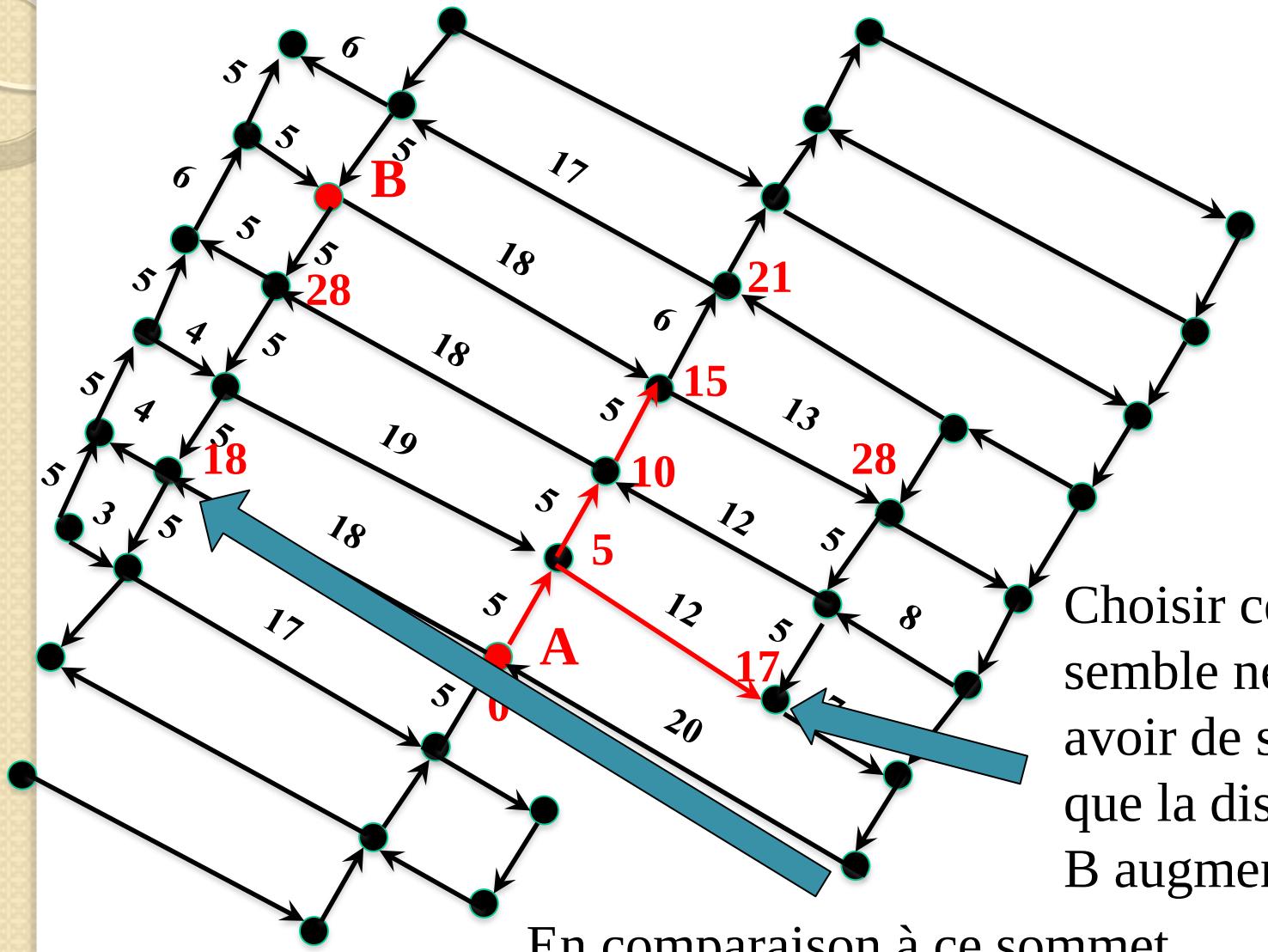
Algorithme de Dijkstra



Algorithme de Dijkstra



Algorithme de Dijkstra



Algorithme A* (1968)

L'algorithme A* est un algorithme pour trouver un chemin entre deux sommets, et qui tente de corriger le problème présenté précédemment sur l'algorithme de Dijkstra.

Pour cela, l'algorithme A* utilise une fonction heuristique h_t qui estime, pour chaque sommet u, sa distance vers la destination t.

L'algorithme ne fonctionne que si

$h_t(u) \leq$ la distance réelle entre u et t.

Pour l'exemple précédent, $h_t(u)$ peut être la distance à vol d'oiseau entre u et t.

Algorithme A*

L’ algorithme A* est similaire à l’algorithme de Dijkstra avec la difference suivante :

La file prioritaire est triée selon f avec :

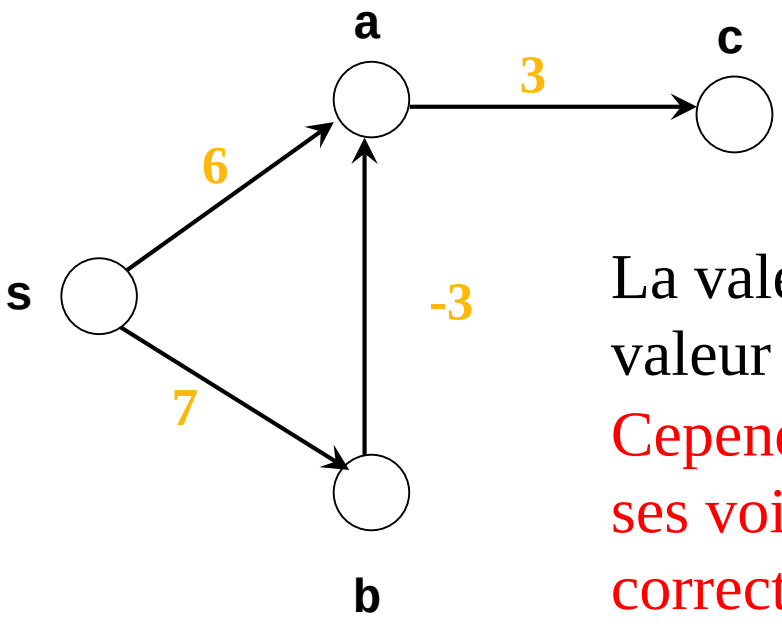
$$f(u) = dist[u] + h_t(u)$$

Algorithme de Bellman-Ford
(publications en 1956 et 1958)

Algorithme de Bellman-Ford

- L‘algorithme de Bellman-Ford résoud le problème de recherche de plus court chemin depuis une source unique, mais en autorisant des arcs portant des poids négatifs.
- Il permet de détecter l‘existence d‘un circuit absorbant, c‘est-à-dire de poids total strictement négatif, accessible depuis le sommet source.
- Cependant, si le graphe contient un circuit absorbant, il n‘existe pas de solution.
- L‘algorithme de Bellman-Ford utilise également la méthode de relaxation

Rappel : problème des poids négatifs dans l'algorithme de Dijkstra



La valeur **`dist`** de **`c`** reste 9 bien que la valeur **`dist`** de **`a`** soit diminuée de 6 à 4.

Cependant, si on retraite a pour relaxer ses voisins, on peut trouver la solution correcte.

Idée de l‘Algorithme de Bellman-Ford

Idée fondamentale de l’algorithme de Bellman-Ford :

`répéter`

relaxer tous les arcs

`Jusqu’à` ce qu’il n’y ait plus de changement de la valeur **`dist`**

Algorithme de Bellman-Ford

- Dans la première phase, les chemins les plus courts sont déterminés, si une solution existe
- Dans la seconde phase, l'algorithme détermine s'il existe des circuits absorbants.
- Si c'est le cas, l'algorithme renvoie la valeur **`faux`**, sinon la valeur **`vrai`**

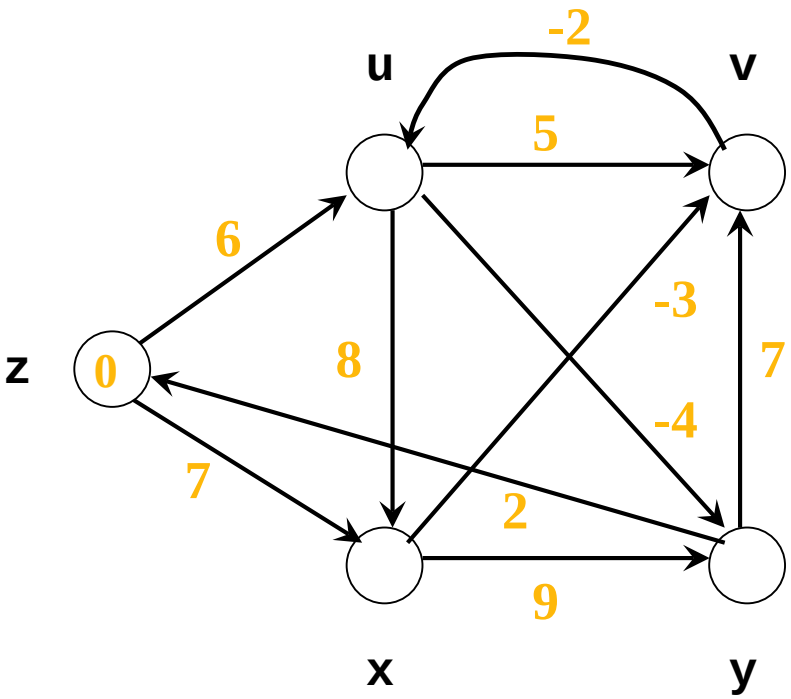
Algorithme de Bellman-Ford

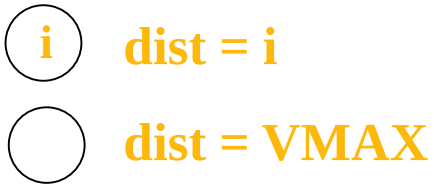
```
Bellman-Ford(G, w, s)
  Initialize-Single-Source(G, s);
  // Determiner les plus courts chemins
  pour (i=1; i < nbsommets; i++)
      pourtout arc (u,v) de G
            Relax(u,v,w)
      fpour
  fpour

  // Déterminer s'il y a des circuits absorbants
  non_absorbant ← vrai
  pour tout arc (u,v) de G
      si (dist[v] > dist[u] + w(u,v))
      alors  non_absorbant ← faux
      fsi
  fpour
  renvoyer non_absorbant
```

Algorithme de Bellman-Ford

Trouver les plus courts Chemins partant de z





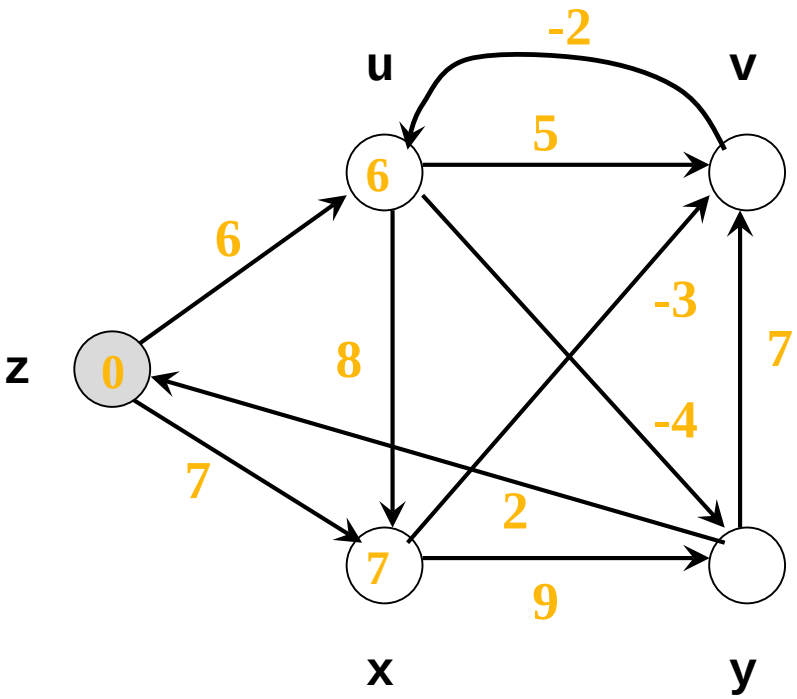
On fait l'hypothèse que l'itération intérieure parcourt les arcs dans l'ordre lexicographique : (u, v), (u,x), (u,y), (v,u), …

Algorithme de Bellman-Ford

- Première exécution de l'itération intérieure :

- Seuls les arcs incidents à **`z`** améliorent leurs valeurs **`dist`**.

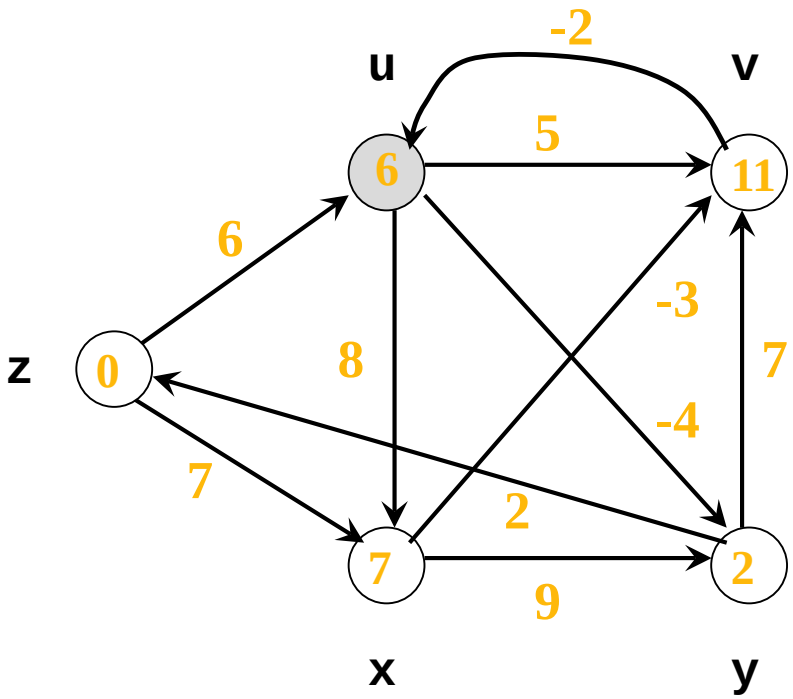
Algorithme de Bellman-Ford



Algorithme de Bellman-Ford

- Seconde exécution de l‘itération intérieure

Algorithme de Bellman-Ford



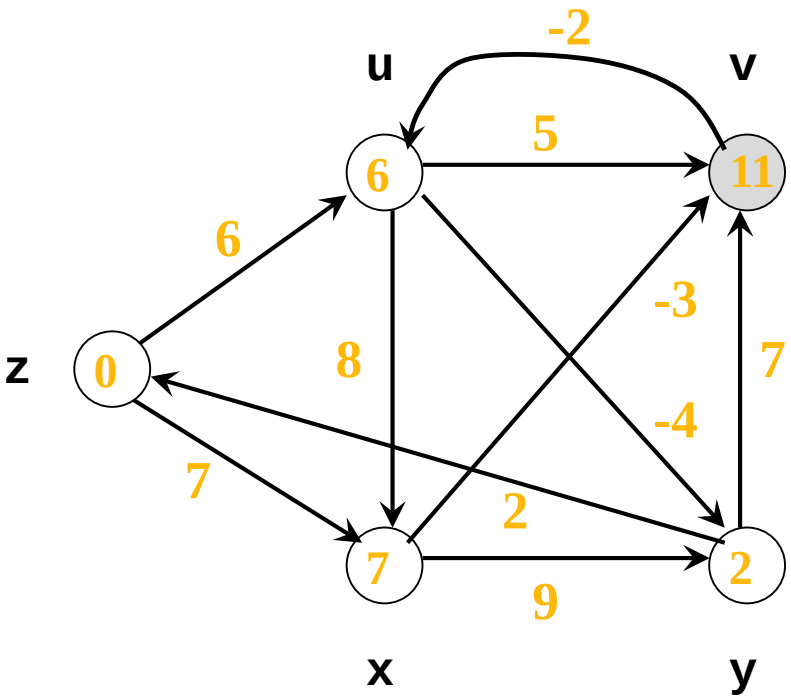
Relaxation avec l'arc (u,v) :

dist[v] décroit de VMAX à 11

Relaxation avec l'arc (u,y) :

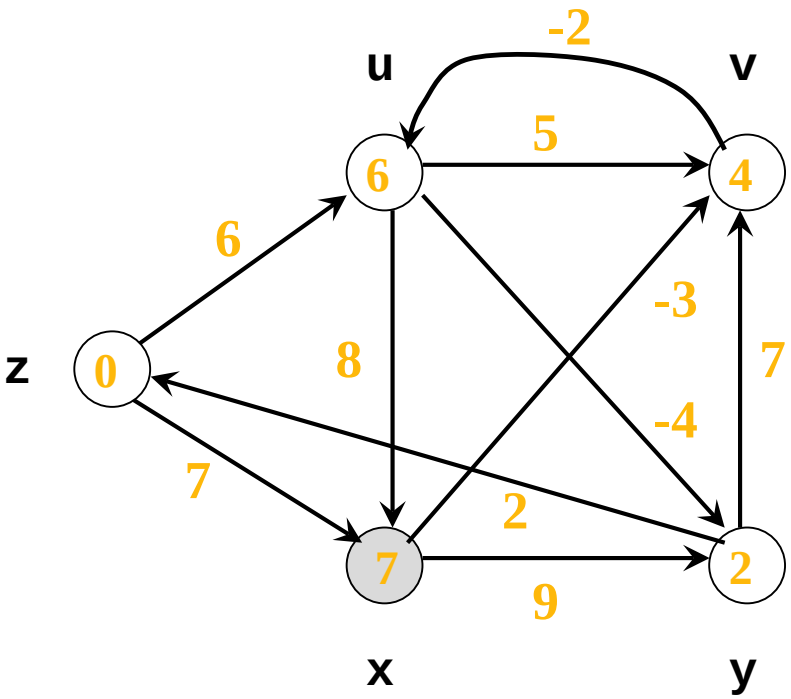
dist[y] décroit de VMAX à 2

Algorithme de Bellman-Ford



Les arcs incidents à **v** n'améliorent pas les valeurs dist

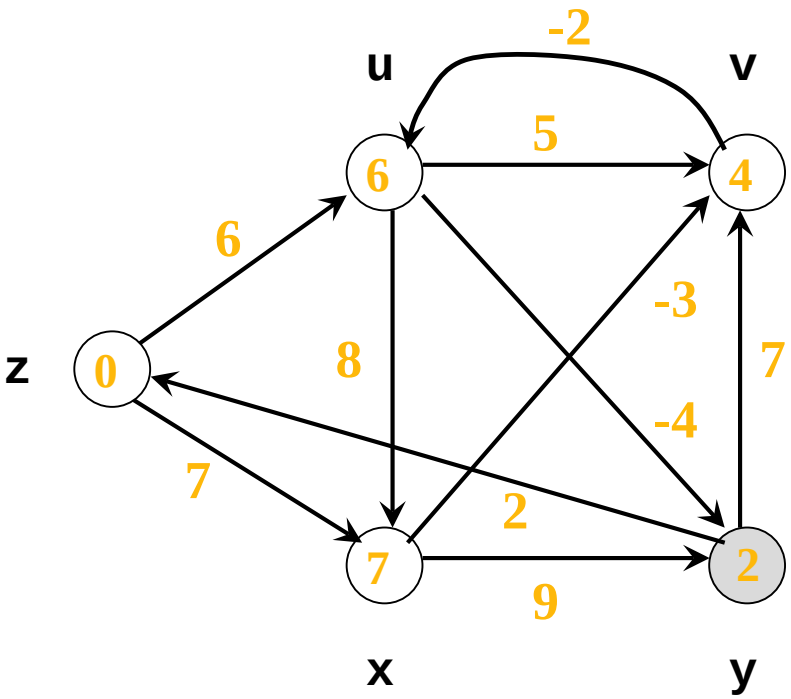
Algorithme de Bellman-Ford



Relaxation avec l'arc (x,v) :

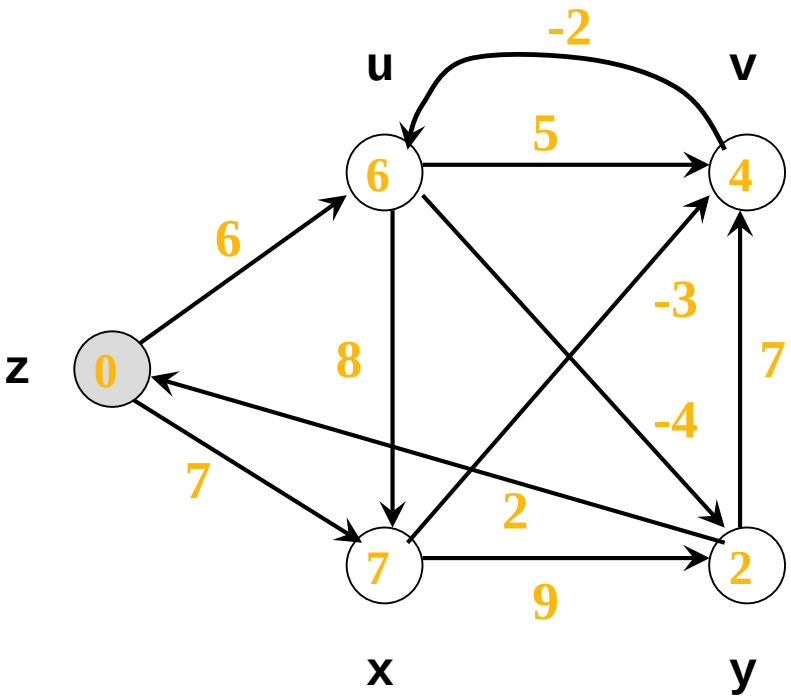
dist[v] décroit de 11 à 4

Algorithme de Bellman-Ford



Les arcs incidents à **y** n'améliorent pas les valeurs dist

Algorithme de Bellman-Ford

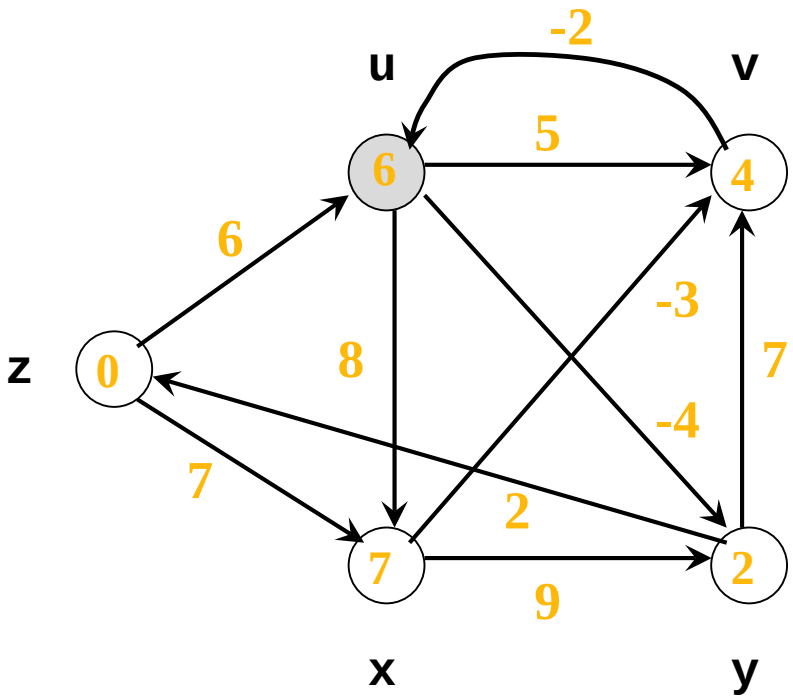


Les arcs incidents à **z** n'améliorent pas les valeurs dist

Algorithme de Bellman-Ford

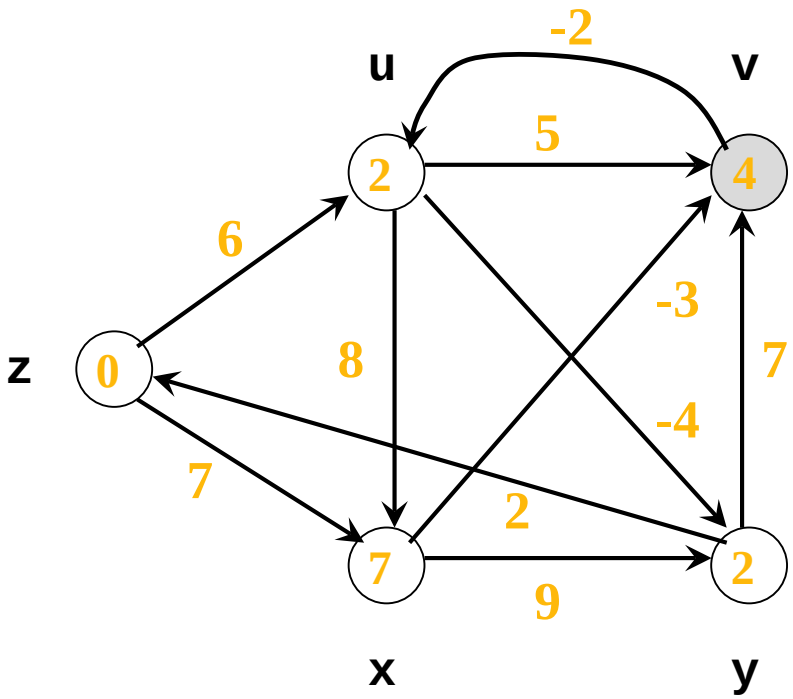
- Troisième exécution de l‘itération intérieure

Algorithme de Bellman-Ford



Les arcs incidents à **u** n'améliorent pas les valeurs dist

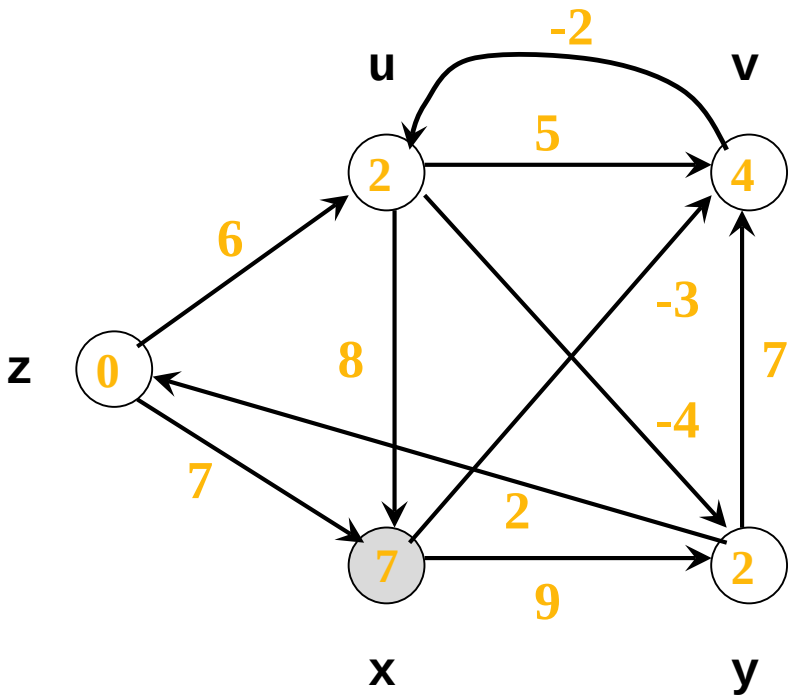
Algorithme de Bellman-Ford



Relaxation avec l'arc (v,u) :

dist[u] décroit de 6 à 2

Algorithme de Bellman-Ford

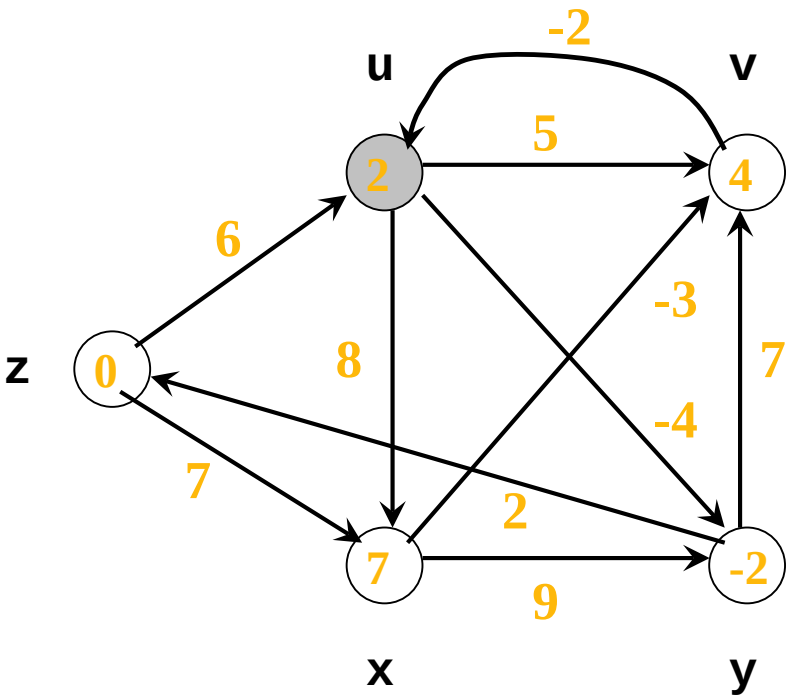


Les arcs incidents à **`x, y et z`** n'améliorent pas les valeurs dist

Algorithme de Bellman-Ford

- Quatrième exécution de l'itération intérieure

Algorithme de Bellman-Ford



Relaxation avec l'arc (u,y) :

dist[y] décroit de 2 à -2

Algorithme de Bellman-Ford

- Toutes les exécutions supplémentaires de l‘itération intrieure n‘améliorent plus la solution actuelle

Algorithme de Bellman-Ford dans une table

Initialisation

sommet	u	v	x	y	z
pred	NIL	NIL	NIL	NIL	NIL
dist	∞	∞	∞	∞	0

Après la première execution de l'iteration intérieure

sommet	u	v	x	y	z
pred	z	NIL	z	NIL	NIL
dist	6	∞	7	∞	0

Algorithme de Bellman-Ford dans une table

Après la seconde execution de l'iteration intérieure

sommet	u	v	x	y	z
pred	z	x	z	u	NIL
dist	6	4	7	2	0

Après la troisième execution de l'iteration intérieure

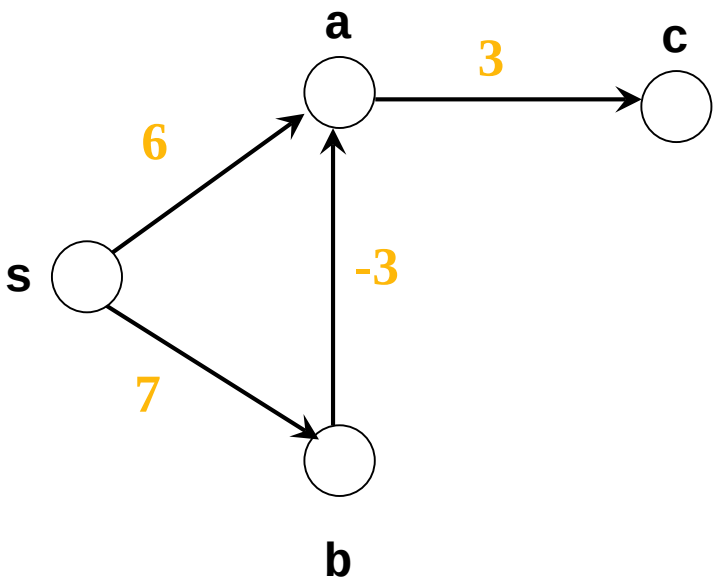
sommet	u	v	x	y	z
pred	v	x	z	u	NIL
dist	2	4	7	2	0

Algorithme de Bellman-Ford dans une table

Après la quatrième execution de l'iteration intérieure

sommet	u	v	x	y	z
pred	v	x	z	u	NIL
dist	2	4	7	-2	0

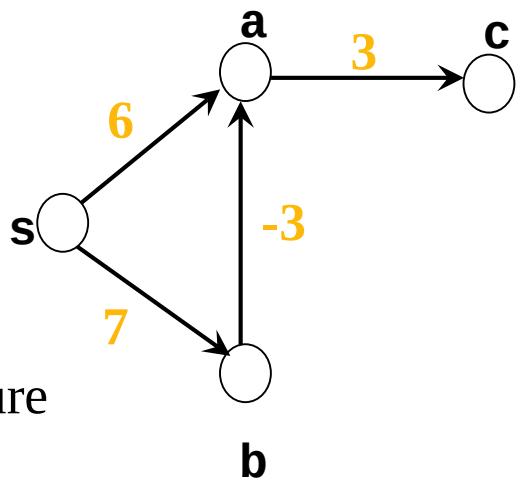
Exercice: appliquer l'algorithme de Bellman-Ford



Exercice: appliquer l'algorithme de Bellman-Ford

Initialisation

sommet	a	b	c	s
pred				
dist				



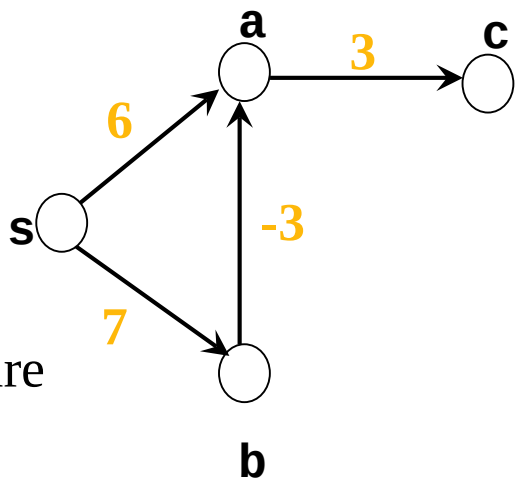
Après la première execution de l'iteration intérieure

sommet	a	b	c	s
pred				
dist				

Exercice: appliquer l'algorithme de Bellman-Ford

Après la seconde execution de l'iteration intérieure

sommet	a	b	c	s
pred				
dist				



Après la troisième execution de l'iteration intérieure

sommet	a	b	c	s
pred				
dist				

Temps d'exécution de l'algorithme de Bellman-Ford

```
Bellman-Ford(G, w, s)
  Initialize-Single-Source(G, s);                    O(n)
  // Determiner les plus courts chemins
  pour (i=1; i < nbsommets; i++)                     O(n * m)
      pourtout arc (u,v) de G                        O(m)
            Relax(u,v,w)
      fpour
  fpour

  // Déterminer s'il y a des circuits absorbants
  non_absorbant ← vrai
  pour tout arc (u,v) de G                           O(m)
      si (dist[v] > dist[u] + w(u,v))
      alors  non_absorbant ← faux
      fsi
  fpour
  renvoyer non_absorbant
```

Temps d'exécution total de l'algorithme de Bellman-Ford : **O(n * m)**

Algorithme de Floyd-Warshall (1959)

Chemins les plus courts entre toutes les paires de sommets

- Si l‘on veut trouver pour toutes les paires de sommets les chemins les plus courts, l‘algorithme décrit pécédemment peut être appliqué en considérant successivement chaque sommet comme sommet de départ
- L‘algorithme de Floyd-Warshall est cependant plus efficace pour le **problème de plus courts chemins entre tous les sommets**

Algorithme de Floyd-Warshall

Données :

Un graphe orienté **G =(V, E)** avec l'ensemble des sommets **V = {1, … , n}** et les poids **w(e)** pour tous les arcs **e ϵ E**

G est représenté par une matrice d'adjacence contenant les poids de chaque arc ou VMAX.

Résultat:

- Une matrice **n x n DIST = ($dist_{ij}$)**, telle que pour **i ≠ j** **$dist_{ij}$** est la longueur du plus court chemin de **i** à **j** et **$dist_{ii}$** est la longueur du circuit le plus court contenant **i**.
- Une matrice **n x n P = (p_{ij})**, telle que **p_{ij}** est le sommet relié à i sur le plus court chemin **(i,j)** respectivement le plus court circuit **(i,i)**

Algorithme de Floyd-Warshall

```
Floyd-Warshall-Algorithm(G, w)
  // Initialization
  Initialisation de DIST avec VMAX, P avec NIL

  // Determination des chemins sans sommet intermédiaire
  pour (i = 1; i <= n; i++)
      pour (j = 1; j <= n; j++)
            si ((i,j)є E)
            alors dist_ij ← w_ij // w_ij = w(i,j)
                  p_ij ← i
            fsi
      fpour
  fpour
```

Algorithme de Floyd-Warshall

```
// suite de l'algorithme de Floyd-Warshall (G, w)
// on tente de raccourcir les chemns par le sommet
// intermédiaire k

  pour (k = 1; k <= n; k++)
      pour (i = 1; i <= n; i++)
            pour (j = 1; j <= n; j++)
                  si (dist_ij > dist_ik + dist_kj)
                  alors dist_ij ← dist_ik + dist_kj
                        p_ij ← p_kj
                  fsi
            fpour
      fpour
  fpour
```

Algorithme de Floyd-Warshall

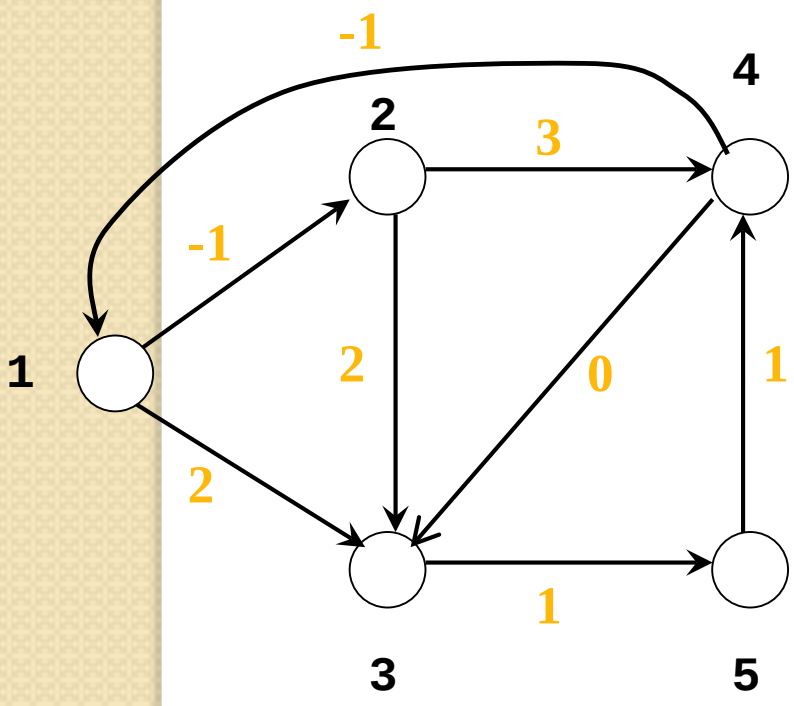
- Si pendant l‘exécution de l‘algorithme on a **`i`** avec $\mathbf{dist_{ii} < 0}$ l‘algorithme peut s‘arrêter car il y a un circuit absorbant, et donc il n‘y a pas de solution.

Algorithme de Floyd-Warshall

Remarque:

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

Algorithme de Floyd-Warshall



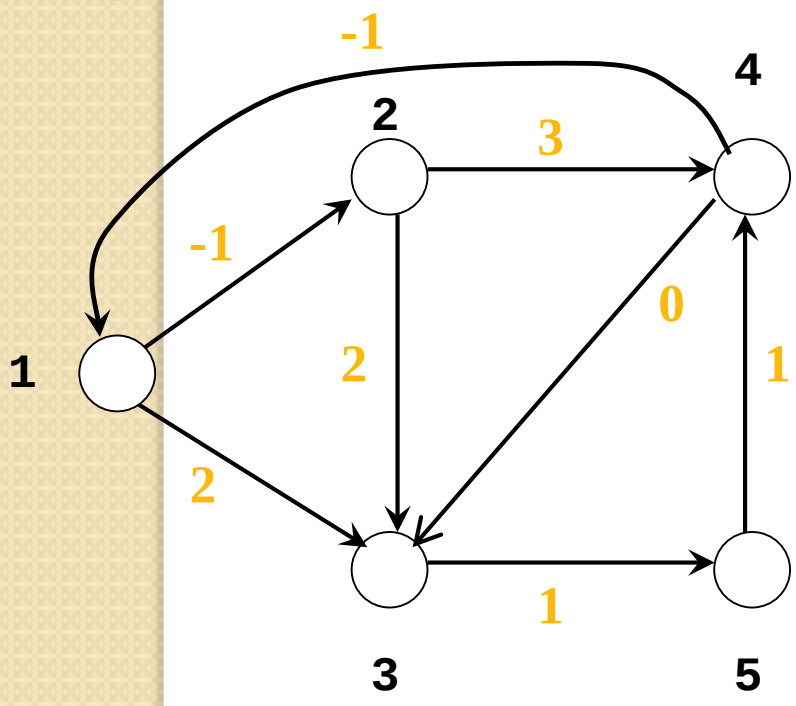
DIST^0

	-1	2		
		2	3	
				1
-1		0		
			1	

DIST^1

	-1	2		
		2	3	
				1
-1	-2	0		
			1	

Algorithme de Floyd-Warshall



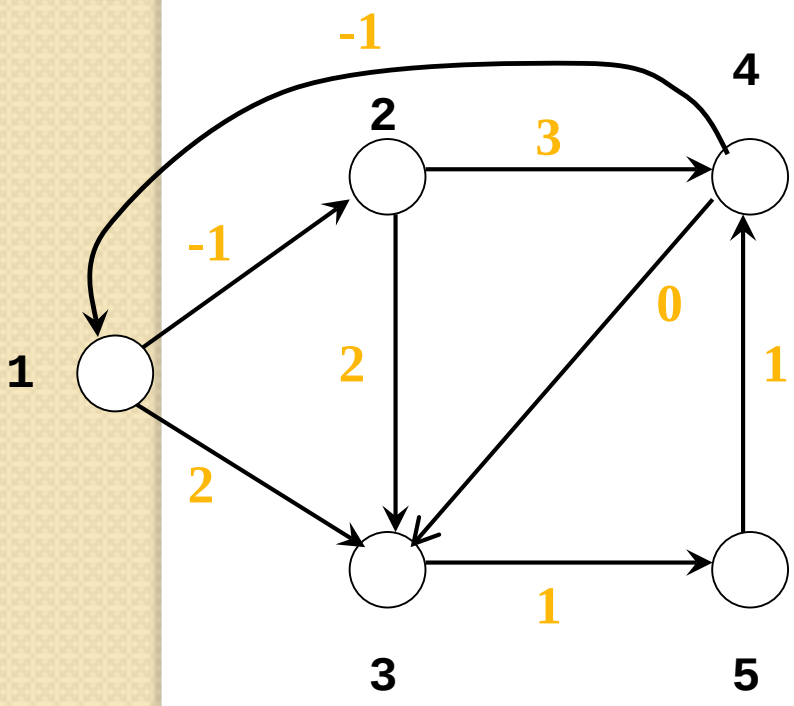
DIST¹

	-1	2		
		2	3	
				1
-1	-2	0		
			1	

DIST²

	-1	1	2	
		2	3	
				1
-1	-2	0	1	
			1	

Algorithme de Floyd-Warshall



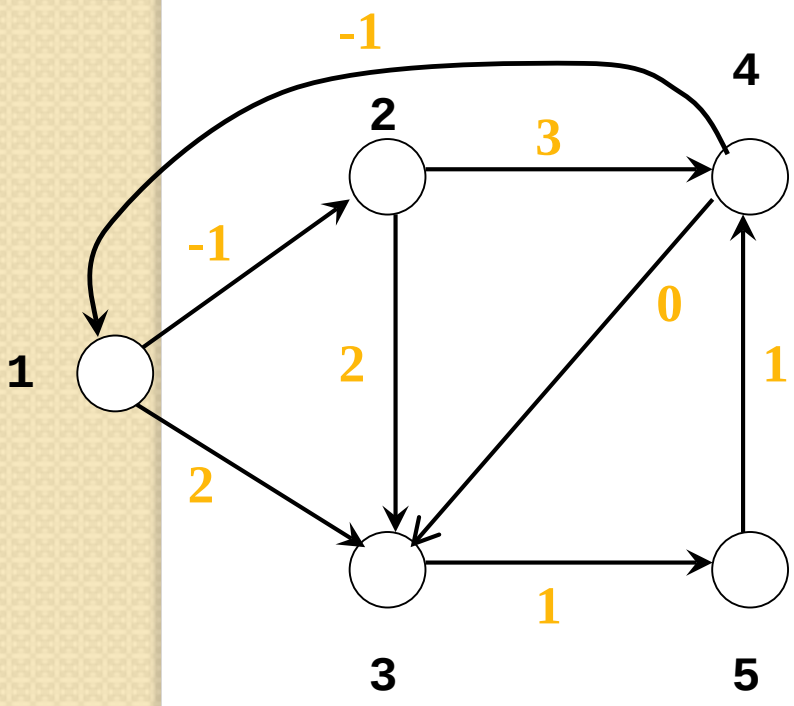
DIST²

	-1	1	2	
		2	3	
				1
-1	-2	0	1	
			1	

DIST³

	-1	1	2	2
		2	3	3
				1
-1	-2	0	1	1
			1	

Algorithme de Floyd-Warshall



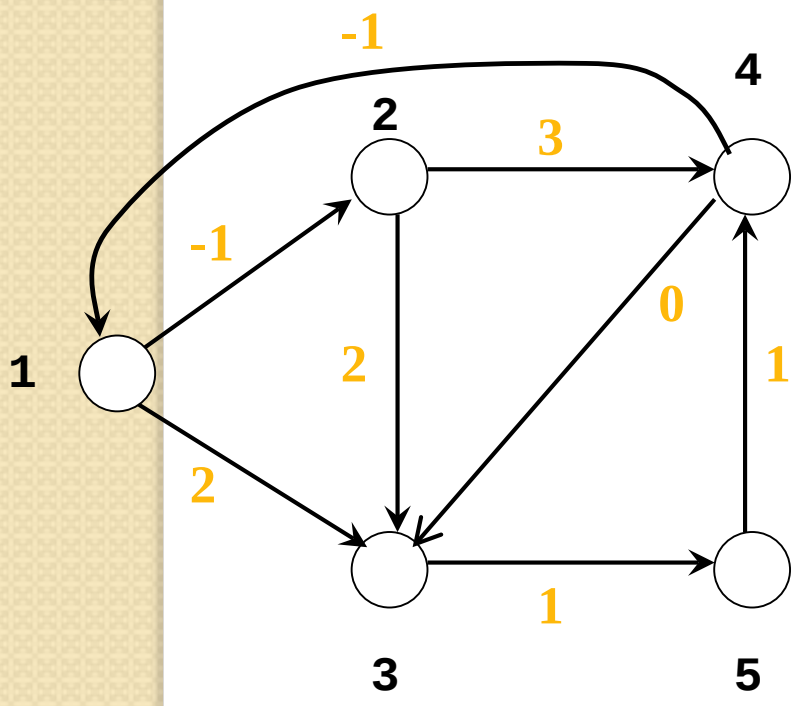
DIST^3

	-1	1	2	2
		2	3	3
				1
-1	-2	0	1	1
			1	

DIST^4

1	-1	1	2	2
2	1	2	3	3
				1
-1	-2	0	1	1
0	-1	1	1	2

Algorithme de Floyd-Warshall



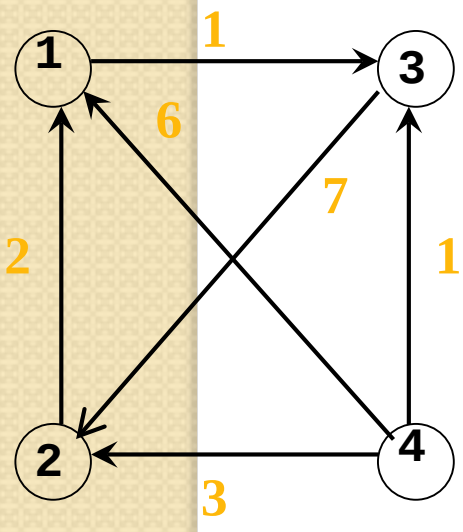
DIST^4

1	-1	1	2	2
2	1	2	3	3
				1
-1	-2	0	1	1
0	-1	1	1	2

DIST^5

1	-1	1	2	2
2	1	2	3	3
1	0	2	2	1
-1	-2	0	1	1
0	-1	1	1	2

Exercise: Appliquer l'algorithme de Floyd-Warshall



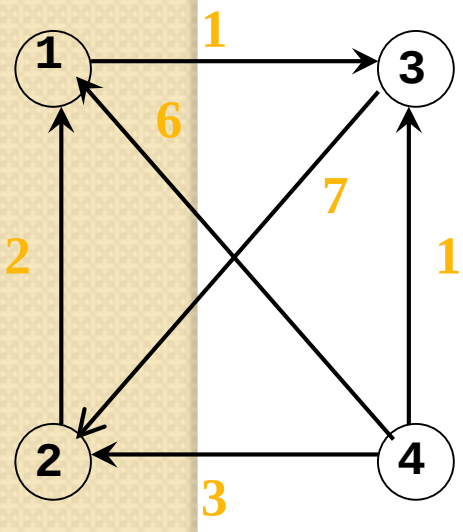
$DIST^0$	1	2	3	4
1				
2				
3				
4				

$PRED^0$	1	2	3	4
1				
2				
3				
4				

$DIST^1$	1	2	3	4
1				
2				
3				
4				

$PRED^1$	1	2	3	4
1				
2				
3				
4				

Exercise: Appliquer l'algorithme de Floyd-Warshall



$DIST^1$	1	2	3	4
1				
2				
3				
4				

$PRED^1$	1	2	3	4
1				
2				
3				
4				

$DIST^2$	1	2	3	4
1				
2				
3				
4				

$PRED^2$	1	2	3	4
1				
2				
3				
4				

Temps d'exécution de l'algorithme de Floyd-Warshall

Le temps d'exécution est dominé par la seconde partie

```
// suite de l'algorithme de Floyd-Warshall (G, w)
// on tente de raccourcir les chemns par le sommet
// intermédiaire k

  pour (k = 1; k <= n; k++)
      pour (i = 1; i <= n; i++)
            pour (j = 1; j <= n; j++)
                  si (dist_ij > dist_ik + dist_kj)
                  alors dist_ij ← dist_ik + dist_kj
                        p_ij ← p_kj
                  fsi
            fpour
      fpour
  fpour
```

O(n) O(n²) O(n³)