

Plus court chemin entre toutes les paires de sommets

CM n°6 — Algorithmique (AL5)

Matěj Stehlík

21/10/2022

Deux classes naturelles de graphes orientés sans cycles négatifs

- Rappelons que le concept du plus court chemin n'a pas de sens s'il existe un cycle négatif.
- Nous nous intéressons donc aux graphes orientés sans cycles négatifs.
- Il y en a deux classes naturelles :
 - les graphes sans arcs négatifs
 - les graphes sans cycles orientés.
- Dans les graphes sans arcs négatifs, on peut utiliser l'algorithme de Dijkstra, de complexité $O(m + n \log n)$.
- Peut-on faire mieux que Bellman–Ford (de complexité $O(nm)$) pour les graphes sans cycles orientés ?

Plus court chemin dans les graphes orientés acycliques (DAG)

- Rappelons qu'il faut effectuer une séquence de mises à jour qui inclut chaque plus court chemin comme sous-séquence.
- Dans tout chemin d'un DAG, les sommets apparaissent dans un ordre topologique croissant.
- Par conséquent, il suffit de faire un tri topologique du DAG par une recherche en profondeur, et puis de parcourir les sommets dans l'ordre topologique, en mettant chaque fois à jour tous les arcs sortants du sommet.
- La complexité de cet algorithme est de $O(n + m)$.

Algorithme de plus court chemin dans les DAG

Entrées : Graphe orienté $G = (V, E)$, pondération $w \in \mathbb{R}^m$, source $s \in V$

Sorties : Distances de s aux autres sommets

$D[s] \leftarrow 0$

$\text{prev}[s] \leftarrow s$

pour tous les $u \in V \setminus \{s\}$ **faire**

$D[u] \leftarrow +\infty$
 $\text{prev}[u] \leftarrow \emptyset$

Tri topologique de G

pour tous les $u \in V$ *dans l'ordre topologique* **faire**

pour tous les $(u, v) \in E$ **faire**
 $\text{maj}(u, v)$

retourner D, prev

Illustration de l'algorithme de plus court chemin dans les DAG

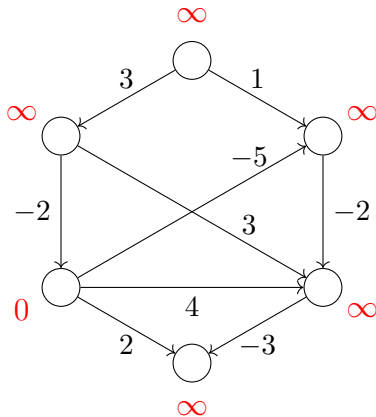


Illustration de l'algorithme de plus court chemin dans les DAG

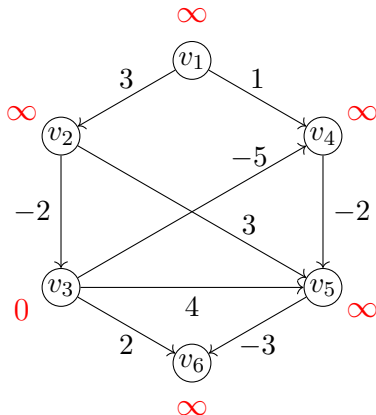


Illustration de l'algorithme de plus court chemin dans les DAG

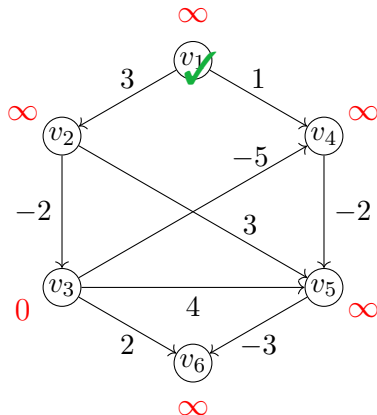


Illustration de l'algorithme de plus court chemin dans les DAG

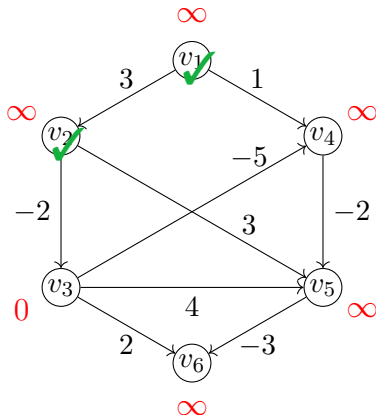


Illustration de l'algorithme de plus court chemin dans les DAG

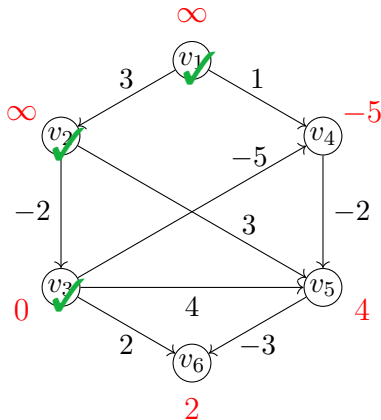


Illustration de l'algorithme de plus court chemin dans les DAG

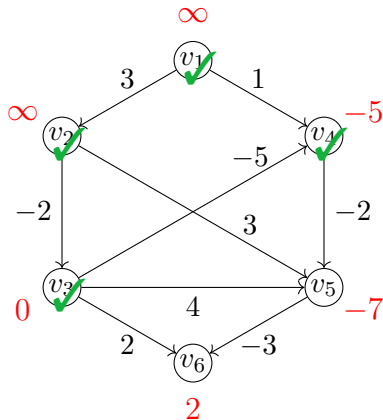


Illustration de l'algorithme de plus court chemin dans les DAG

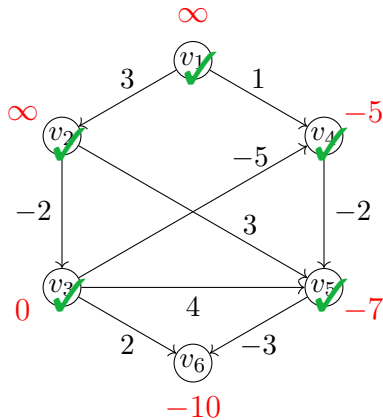
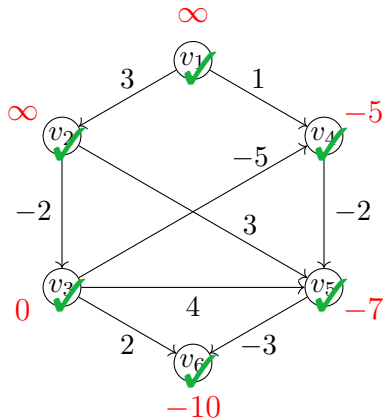


Illustration de l'algorithme de plus court chemin dans les DAG



Distances entre toutes les paires de sommets

- Dijkstra et Bellman–Ford trouvent la distance d'un sommet fixe (la source) aux autres sommets.
- Et si on veut trouver la distance entre *toutes les paires* de sommets ?
- Une approche naïve : exécuter Dijkstra ou Bellman–Ford n fois : une fois pour chaque sommet.
- La complexité de l'algorithme ainsi obtenu est de :
 - $O(nm + n^2 \log n)$ (cas avec poids non négatifs)
 - $O(n^2m)$ (cas général)
- Si l'on ignore le terme logarithmique, le premier algorithme (poids non négatifs) est de la même complexité que Bellman–Ford.
- Pour les graphes denses, la complexité du deuxième algorithme est de $O(n^4)$.
- Peut-on faire mieux ?

Sommets intermédiaires

- Le plus court chemin $(u, w_1, \dots, w_\ell, v)$ de u à v utilise un certain nombre de sommets “intermédiaires”.
- Supposons que nous n'autorisions aucun sommet intermédiaire.
- Nous pouvons alors trouver les plus courts chemins entre toutes les paires en un seul coup : le plus court chemin de u à v est simplement l'arc (u, v) , si il existe.
- On élargit progressivement (d'un sommet à chaque étape) l'ensemble des sommets intermédiaires autorisés, en mettant à jour les longueurs des plus courts chemins à chaque étape.

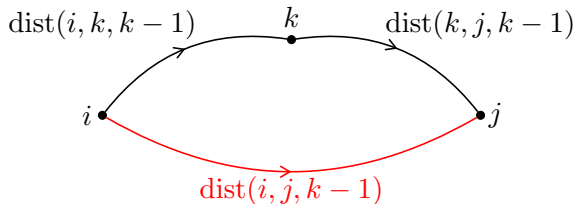
Distances partielles

- Soit $V = \{1, 2, \dots, n\}$ l'ensemble des sommets.
- Soit $\text{dist}(i, j, k)$ la longueur minimum d'un chemin de i à j dont tous les sommets intermédiaires sont dans $\{1, 2, \dots, k\}$.
- En particulier,

$$\text{dist}(i, j, 0) = \begin{cases} \ell(i, j) & \text{si } (i, j) \in E \\ \infty & \text{si } (i, j) \notin E. \end{cases}$$

- Un plus court chemin de i à j qui emprunte k (et éventuellement d'autres sommets intermédiaires qui précèdent k) passe par k une seule fois.

Mise à jour des distances partielles



- On a déjà calculé la longueur d'un plus court chemin passant uniquement par les sommets intermédiaires dans $\{1, \dots, k\}$.
- Passer par k donne un chemin plus court de i à j ssi

$$\text{dist}(i, k, k-1) + \text{dist}(k, j, k-1) < \text{dist}(i, j, k-1).$$

Algorithme de Floyd–Warshall

Entrées : Graphe orienté $G = (V, E)$ avec pondération $\ell \in \mathbb{R}^{|E|}$

Sorties : Distances entre chaque paire de sommets

pour tous les $i \in \{1, \dots, n\}$ **faire**

pour tous les $j \in \{1, \dots, n\}$ **faire**
 $\text{dist}(i, j, 0) \leftarrow \infty$

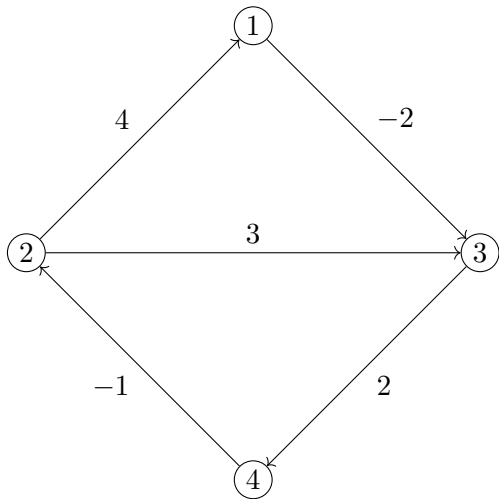
pour tous les $(i, j) \in E$ **faire**

$\text{dist}(i, j, 0) \leftarrow \ell(i, j)$

pour tous les $k \in \{1, \dots, n\}$ **faire**

pour tous les $i \in \{1, \dots, n\}$ **faire**
 pour tous les $j \in \{1, \dots, n\}$ **faire**
 $\text{dist}(i, j, k) = \min\{\text{dist}(i, k, k-1) + \text{dist}(k, j, k-1), \text{dist}(i, j, k-1)\}$

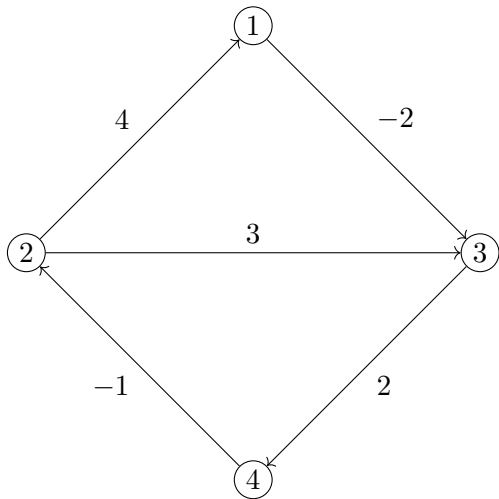
Illustration de l'algorithme de Floyd-Warshall



	1	2	3	4
1	0	∞	-2	∞
2	4	0	3	∞
3	∞	∞	0	2
4	∞	-1	∞	0

$\text{dist}(\cdot, \cdot, 0)$

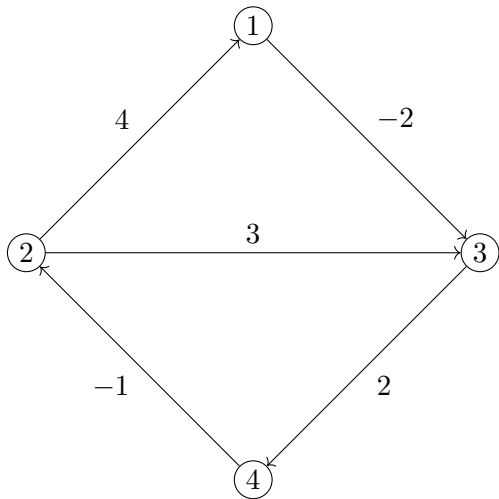
Illustration de l'algorithme de Floyd-Warshall



	1	2	3	4
1	0	∞	-2	∞
2	4	0	2	∞
3	∞	∞	0	2
4	∞	-1	∞	0

$\text{dist}(\cdot, \cdot, 1)$

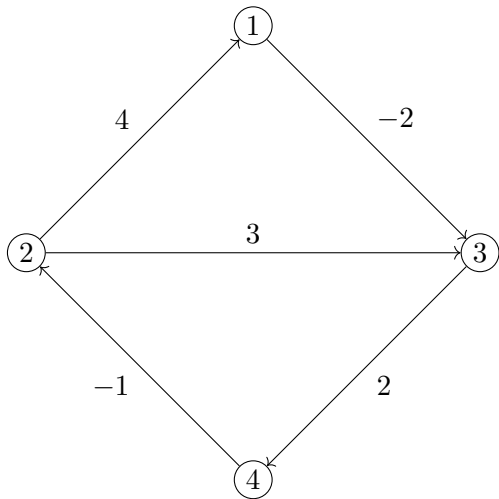
Illustration de l'algorithme de Floyd-Warshall



	1	2	3	4
1	0	∞	-2	∞
2	4	0	2	∞
3	∞	∞	0	2
4	3	-1	1	0

$\text{dist}(\cdot, \cdot, 2)$

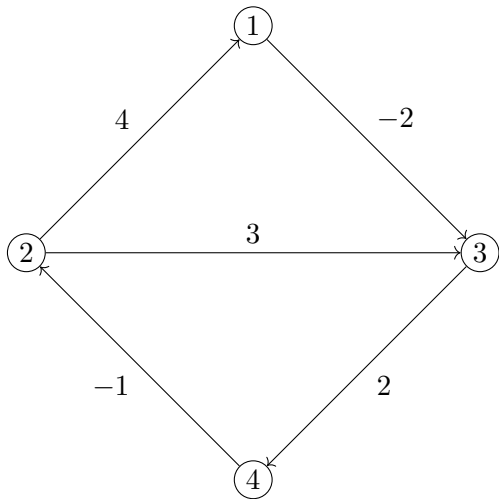
Illustration de l'algorithme de Floyd-Warshall



	1	2	3	4
1	0	∞	-2	0
2	4	0	2	4
3	∞	∞	0	2
4	3	-1	1	0

$\text{dist}(\cdot, \cdot, 3)$

Illustration de l'algorithme de Floyd-Warshall



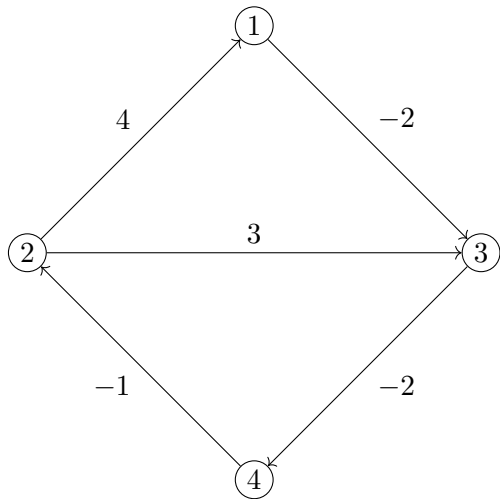
	1	2	3	4
1	0	-1	-2	0
2	4	0	2	4
3	5	1	0	2
4	3	-1	1	0

$\text{dist}(\cdot, \cdot, 4)$

Remarques sur l'algorithme de Floyd–Warshall

- La complexité est de $O(n^3)$.
- Pour les graphes *denses*, cela représente une amélioration d'un facteur de n par rapport à l'approche naïve.
- On verra un autre algorithme (de Johnson) mieux adapté aux graphes *peu denses*.
- L'algorithme de Floyd–Warshall peut être utilisé pour détecter les circuits négatifs.
- Il y a un nombre négatif sur la diagonale de la matrice de distances ssi le graphe contient au moins un circuit négatif.

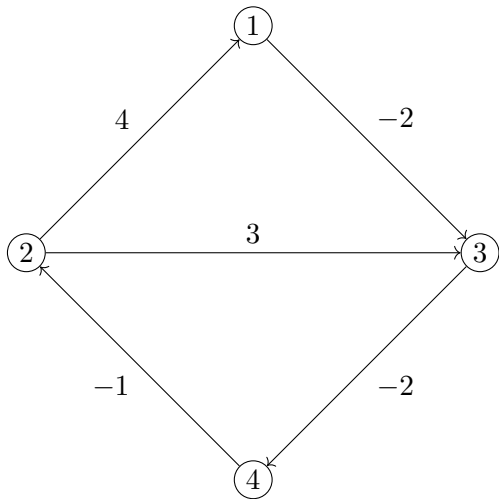
Détection de circuit négatif avec Floyd-Warshall



	1	2	3	4
1	0	∞	-2	∞
2	4	0	3	∞
3	∞	∞	0	-2
4	∞	-1	∞	0

$\text{dist}(\cdot, \cdot, 0)$

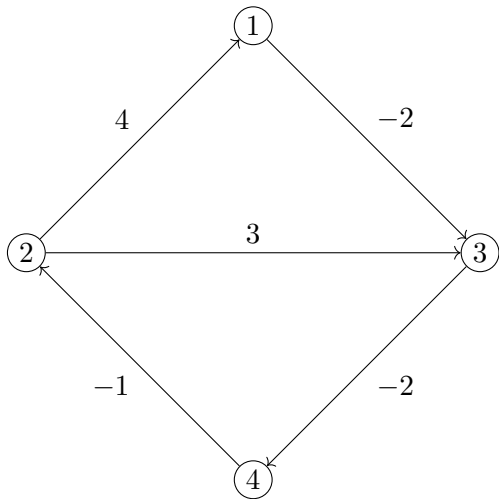
Détection de circuit négatif avec Floyd-Warshall



	1	2	3	4
1	0	∞	-2	∞
2	4	0	2	∞
3	∞	∞	0	-2
4	∞	-1	∞	0

$\text{dist}(\cdot, \cdot, 1)$

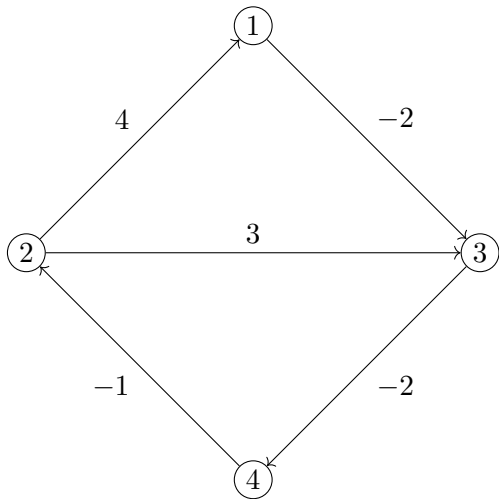
Détection de circuit négatif avec Floyd-Warshall



	1	2	3	4
1	0	∞	-2	∞
2	4	0	2	∞
3	∞	∞	0	-2
4	3	-1	1	0

$\text{dist}(\cdot, \cdot, 2)$

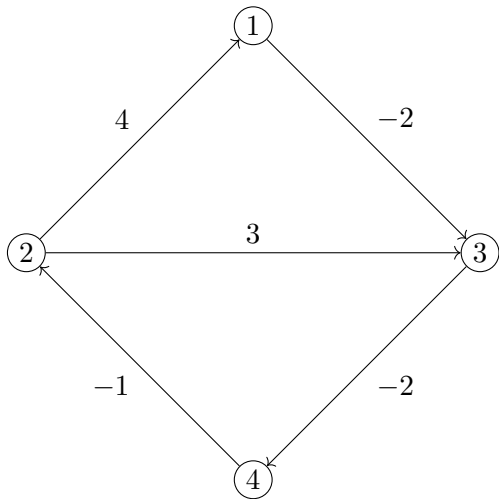
Détection de circuit négatif avec Floyd-Warshall



	1	2	3	4
1	0	∞	-2	-4
2	4	0	2	1
3	∞	∞	0	-2
4	3	-1	1	0

$\text{dist}(\cdot, \cdot, 3)$

Détection de circuit négatif avec Floyd-Warshall



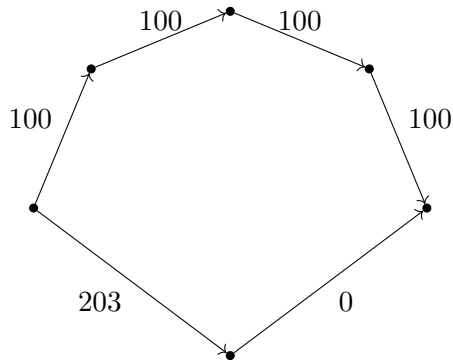
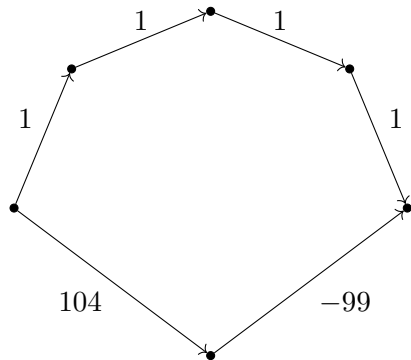
	1	2	3	4
1				
1	-1	-1	-2	0
2	4	-1	2	4
3	1	-3	-1	2
4	3	-1	1	-1

$\text{dist}(\cdot, \cdot, 4)$

Peut-on faire mieux que $O(n^3)$ dans le cas des graphes peu denses ?

- Idée naïve : repondérer le graphe de sorte que les poids deviennent non-négatifs, et les plus courts chemins soient préservés.
- Ensuite, exécuter Dijkstra n fois (une fois par sommet) ; complexité $O(nm + n^2 \log n)$.
- Comment trouver une telle repondération ?
- Première tentative : ajouter une constante au poids de chaque arc de sorte d'éliminer les poids négatifs

Cette repondération naïve ne préserve pas les plus court chemins !



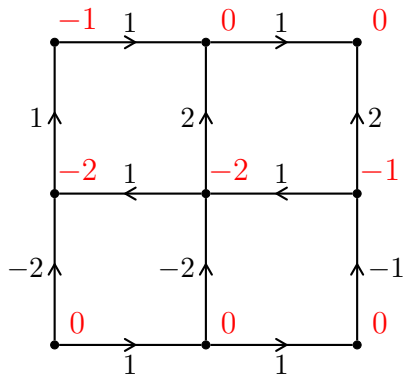
Une repondération préservant les plus courts chemins

- Soit $G = (V, E)$ un graphe avec pondération $\ell \in \mathbb{R}^m$.
- Soit $h \in \mathbb{R}^n$ un vecteur associant à chaque sommet un nombre réel.
- On définit une nouvelle pondération $\ell' \in \mathbb{R}^m$ de G par
$$\ell'_{(u,v)} = \ell_{(u,v)} + h_u - h_v.$$

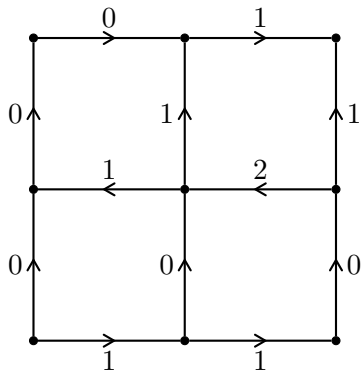
Lemme

P est un plus court chemin de u à v dans G par rapport à ℓ ssi P est un plus court chemin de u à v dans G par rapport à ℓ' .

Example



ℓ, h



ℓ'

Preuve du lemme (1/2)

- Soit P un chemin quelconque dans G .

$$\begin{aligned}\ell'(P) &= \sum_{i=1}^k \ell'_{(v_{i-1}, v_i)} \\ &= \sum_{i=1}^k (\ell_{(v_{i-1}, v_i)} + h_{v_{i-1}} - h_{v_i}) \\ &= \sum_{i=1}^k \ell_{(v_{i-1}, v_i)} + h_{v_0} - h_{v_k} \\ &= \ell(P) + h_{v_0} - h_{v_k}\end{aligned}$$

- Donc, non seulement le plus court chemin, mais *tout* chemin P de u à v vérifie $\ell'(P) = \ell(P) + h_u - h_v$.

Preuve du lemme (2/2)

- En particulier, si P est un chemin de u à v , alors $\ell(P) = \text{dist}_\ell(u, v)$ ssi $\ell'(P) = \text{dist}_{\ell'}(u, v)$.
- La longueur de cycles ne change pas si l'on passe de la pondération ℓ à ℓ' (car on a $v_0 = v_k$ dans l'équation de la diapo précédente).
- En particulier, il n'y a pas de cycle négatif par rapport à ℓ ssi il n'y a pas de cycle négatif par rapport à ℓ' .

Comment trouver la repondération ?

- Il suffit de prouver l'existence d'une pondération $\ell' \in \mathbb{R}^m$ t.q. $\ell \geq 0$.
- Soit G' le graphe construit à partir de G en ajoutant un nouveau sommet s et les arcs $\{(s, v) : v \in V\}$.
- On étend la pondération ℓ à une pondération de G' en posant $\ell_{(s,v)} = 0$ pour tout $v \in V$.
- G' ne contient aucun cycle négatif ssi G ne contient aucun cycle négatif.
- Supposons que G et G' ne contiennent aucun cycle négatif.
- On définit $h_v = \text{dist}(s, v)$ pour tout sommet $v \in V(G')$.
- On a $h_v \leq h_u + \ell_{(u,v)}$ pour tout arc $(u, v) \in E(G')$.
- Donc, $\ell'_{(u,v)} = \ell_{(u,v)} + h_u - h_v \geq 0$.

Algorithme de Johnson

1. Calculer G' .
2. Appliquer Bellman–Ford à G' , avec source s , pour calculer $h_v := \text{dist}(s, v)$ pour tout $v \in V(G)$ (ou trouver un cycle négatif)
3. Repondérer chaque arc $(u, v) \in E(G)$ par $\ell'_{(u,v)} = \ell_{(u,v)} + h(u) - h(v)$.
4. Pour chaque $u \in V(G)$, exécuter Dijkstra pour calculer $\text{dist}_{\ell'}(u, v)$ pour tout $v \in V(G)$.
5. Pour chaque couple u, v , on a $\text{dist}_{\ell}(u, v) = \text{dist}_{\ell'}(u, v) + h(v) - h(u)$.

Complexité de l'algorithme de Johnson

- L'étape 1 : $O(n)$
- L'étape 2 : $O(nm)$
- L'étape 3 : $O(m)$
- L'étape 4 : $O(nm + n^2 \log n)$
- L'étape 5 : $O(n^2)$
- Donc, l'algorithme de Johnson est de complexité $O(nm + n^2 \log n)$.
- Pour des graphes *peu denses*, l'algorithme de Johnson est donc plus rapide que l'algorithme de Floyd–Warshall.

Illustration de l'algorithme de Johnson

