Algorithme de Bellman–Ford

Entrées : Graphe orienté $G = (V, E)$, pondération $\ell \in \mathbb{R}^m$, source $s \in V$
Sorties : Distances de s aux autres sommets

$D[s] \leftarrow 0$
$\texttt{prev}[s] \leftarrow s$
pour tous les $u \in V \setminus \{s\}$ **faire**
  $D[u] \leftarrow +\infty$
  $\texttt{prev}[u] \leftarrow \emptyset$
repéter $|V| - 1$ **fois**
  pour tous les $e \in E$ **faire**
    $\texttt{maj}(e)$
retourner $D, \texttt{prev}$

Illustration de l'algorithme de Bellman–Ford

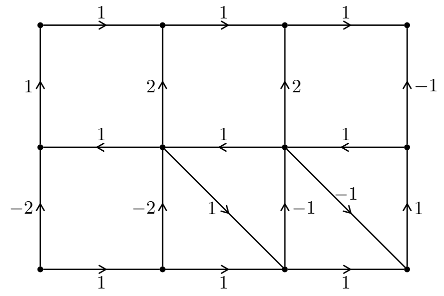
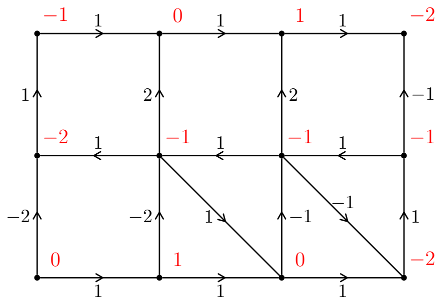


Illustration de l'algorithme de Bellman–Ford



Complexité et correction de l'algorithme de Bellman–Ford

- La complexité de Bellman–Ford est de $O(nm)$.
- Pour la correction de l'algorithme de Bellman–Ford, il suffit de prouver le lemme suivant.
- La démonstration peut se faire par récurrence.

Lemme
Après i itérations de la boucle principale :

- Si $D[u] \neq +\infty$, alors $D[u]$ est la longueur d'un chemin de s à u ;
- S'il existe un chemin de s à u comprenant au plus i arcs, alors la valeur de $D[u]$ est inférieure ou égale à la longueur d'un plus court chemin de s à u comprenant au plus i arcs.

Détection de cycles négatifs

- Une légère modification de l'algorithme de Bellman–Ford nous permet de détecter les cycles négatifs.
- Après avoir fait les $|V| - 1$ itérations de la boucle, faire une itération supplémentaire.
- Un cycle négatif existe dans G ssi il y a au moins un changement dans le tableau D lors de la dernière itération.
- Détecter les cycles négatifs a des applications importantes dans la vie réele.
- Une application classique est l'arbitrage de devises (voir le TD).

Deix classes naturelles de graphes orientés sans cycles négatifs

- Il y a deux classes naturelles de graphes orientés sans cycles négatifs :
  - les graphes sans arcs négatifs
  - les graphes sans cycles orientés.
- Dans les graphes sans arcs négatifs, on peut utiliser l'algorithme de Dijkstra.

Plus court chemin dans les graphes orientés acycliques (DAG)

- 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
$D[s] \leftarrow 0$
$\texttt{prev}[s] \leftarrow s$
pour tous les $u \in V \setminus \{s\}$ **faire**
    $D[u] \leftarrow +\infty$
    $\texttt{prev}[u] \leftarrow \emptyset$
Tri topologique de G
pour tous les $u \in V$ *dans l'ordre topologique* **faire**
    pour tous les $(u, v) \in E$ **faire**
        $\texttt{maj}(u, v)$
retourner $D, \texttt{prev}$

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

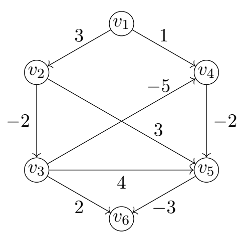
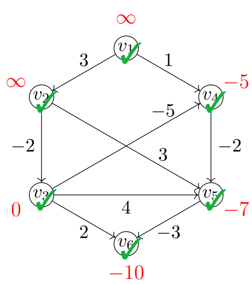


Illustration de l'algorithme de plus court chemin dans les DAG



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)
  - $O(n^2 m)$ (cas général)
- Si l'on ignore le terme logarithmique, le premier algorithme (poids non négatifs) a la même complexité que Bellman–Ford.
- Pour les graphes denses, complexité du deuxième algorithme est de $O(n^4)$.
- Peut-on faire mieux ?

Sommets intermédiaires

- Le plus court chemin $(u, w_1, \ldots, w_\ell, v)$ 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), s'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.

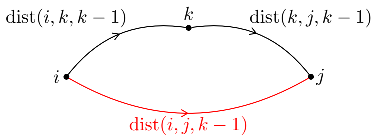
Distances partielles

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

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

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

Mise à jour des distances partielles



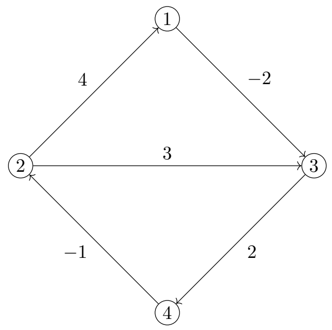
- On a déjà calculé la longueur d'un plus court chemin passant uniquement par les sommets intermédiaires dans $\{1, \dots, k\}$.
- Passer par k donne un chemin plus court de i à j ssi

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

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**
    $\mathrm{dist}(i, j, 0) \leftarrow \infty$
pour tous les $(i, j) \in E$ **faire**
  $\mathrm{dist}(i, j, 0) \leftarrow \ell(i, j)$
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**
      $\mathrm{dist}(i, j, k) = \min\{\mathrm{dist}(i, k, k-1) + \mathrm{dist}(k, j, k-1), \mathrm{dist}(i, j, k-1)\}$

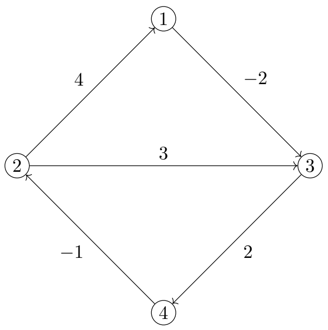
Illustration de l'algorithme de Floyd–Warshall



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

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

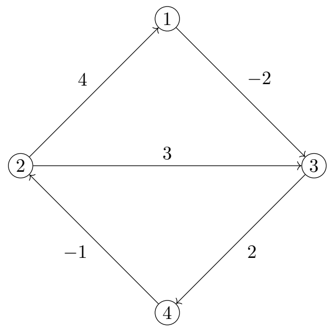
Illustration de l'algorithme de Floyd–Warshall



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

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

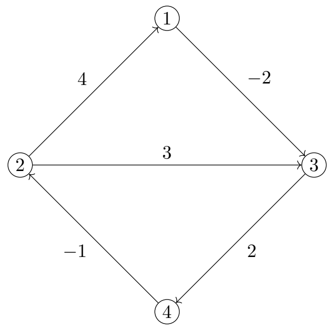
Illustration de l'algorithme de Floyd–Warshall



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

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

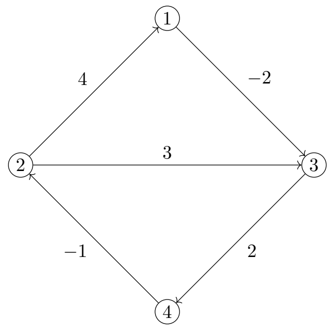
Illustration de l'algorithme de Floyd–Warshall



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

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

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

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

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.
- On verra un autre algorithme mieux adapté aux graphes peu denses.
- L'algorithme de Floyd–Warshall peut être utilisé pour détecter les cycles négatifs.
- Il y a un nombre négatif sur la diagonale de la matrice de distances ssi le graphe contient au moins un cycle négatif.

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

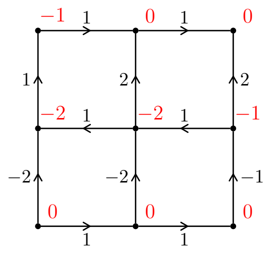
- Idée naïve : repondérer le graphe de sorte que les poids deviennent non-négatifs, et les plus courts chemins soient préservés.
- Ensuite, exécuter Dijkstra n fois (une fois par sommet) ; complexité $O(nm + n^2 \log n)$.
- On ne peut pas simplement ajouter une constante à tous les arcs (pourquoi ?)
- Cependant, il existe une façon de le faire, de complexité $O(nm)$

Repondération préservant les plus courts chemins

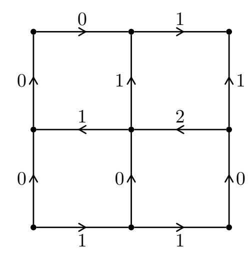
- Soit $G = (V, E)$ un graphe avec pondération $\ell \in \mathbb{R}^m$.
- Soit $h \in \mathbb{R}^n$ un vecteur associant à chaque sommet un nombre réel.
- On définit une nouvelle pondération $\ell' \in \mathbb{R}^m$ de G par $\ell'_{(u,v)} = \ell_{(u,v)} + h_u - h_v$.

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'.

Exemple



ℓ, h



ℓ'

Preuve du lemme (1/2)

- Soit P un chemin quelconque dans G.

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

- Donc, *tout* chemin P de u à v (pas seulement le plus court chemin) 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 $\ell(P) = \mathrm{dist}_\ell(u, v)$ ssi $\ell'(P) = \mathrm{dist}_{\ell'}(u, v)$.
- 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'.