Algorithmique I
Année 2007-2008

A. Benoit, B. Depardon
JF. Pineau, C. Rezvoy

TD n°8 - Recherche de plus courts chemins

1 L'algorithme de Bellman-Ford

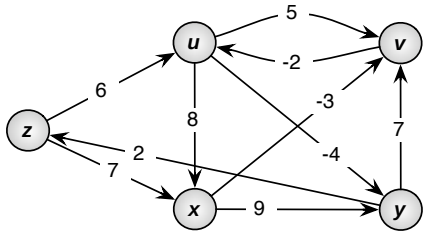
L'algorithme de Bellman-Ford résout le problème des plus courts chemins avec origine unique dans le cas le plus général où les poids des arcs peuvent avoir des valeurs négatives. Étant donné un graphe orienté pondéré $G = (V, E)$, de fonction de poids w, et une origine s, l'algorithme retourne une valeur booléenne indiquant s'il existe un circuit de poids négatif accessible depuis s. S'il n'en existe pas, l'algorithme donne les plus courts chemins ainsi que leurs poids.

Les notations sont les suivantes : $\pi[v]$ contient le prédécesseur de v sur le chemin (NIL s'il n'y en a pas), $\delta(u, v)$ est le poids du plus court chemin de u vers v (∞ s'il n'existe pas), $d[v]$ est une variable qui est une borne supérieure du poids du plus court chemin de s vers v (cf. question 4). On définit le poids d'un chemin $p = \langle v_0, v_1, \ldots, v_k \rangle$, est $w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$.

Algorithme 1 Bellman-Ford(G, w, s)

```
1: pour tout sommet v ∈ V faire // Initialisation
2:    d[v] ← ∞, π[v] ← NIL
3: d[s] ← 0
4: pour i de 1 à |V| − 1 faire
5:    pour tout arc (u, v) ∈ E faire // relâchement de l'arc (u, v)
6:       si d[v] > d[u] + w(u, v) alors
7:          d[v] ← d[u] + w(u, v), π[v] ← u
8: pour tout arc (u, v) ∈ E faire // détection des circuits négatifs
9:    si d[v] > d[u] + w(u, v) alors
10:      retourner Faux
11: retourner Vrai
```

Question 1.1 On considère le graphe ci-dessous. Faire tourner l'algorithme en prenant comme source le sommet z. Même question en changeant le poids de l'arc (y, v) à 4.



Solution : *Exemple lorsque l'algorithme tourne en considérant les arcs dans l'ordre suivant :* (u, v), (u, x), (u, y), (v, u), (x, v), (x, y), (y, v), (y, z), (z, u), (z, x)

Avec $w(y, v) = 4$ *: l'algorithme renvoie faux à cause d'un circuit de poids négatif entre les sommets* u *et* v*.*

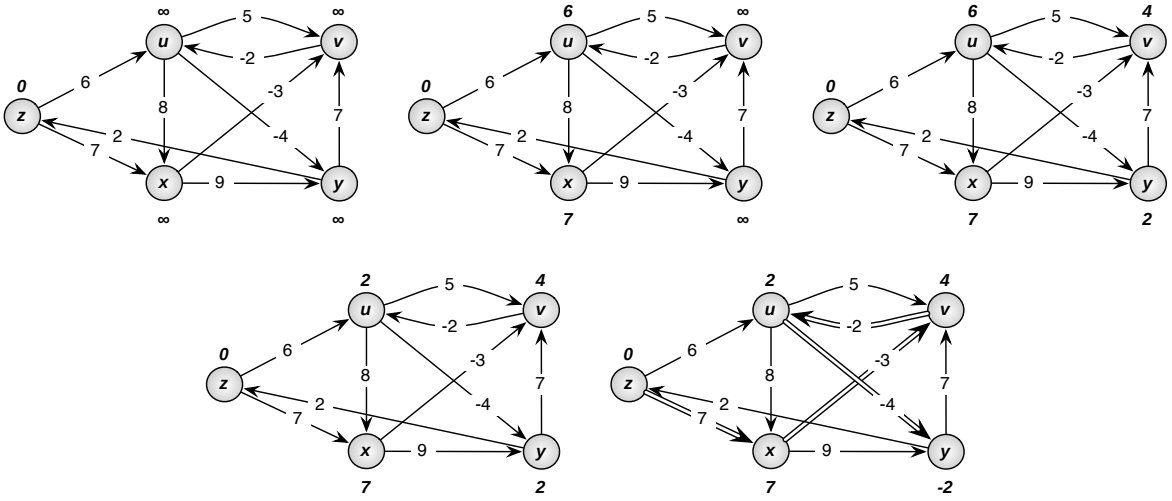


FIG. 1: Bellman-Ford, renvoie vrai, les plus courts chemins sont en blancs

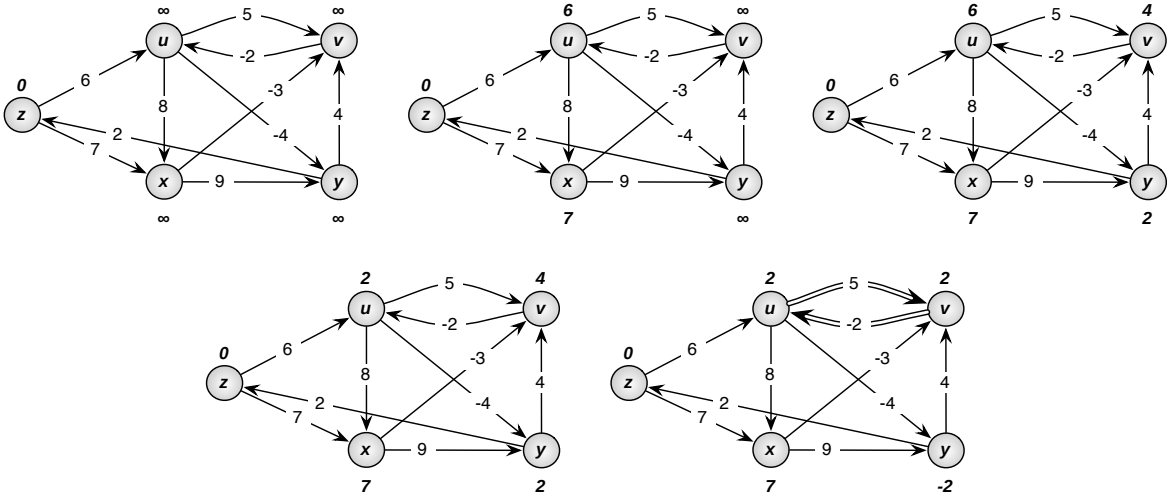


FIG. 2: Bellman-Ford, renvoie faux à cause d'un circuit de poids négatif

□

Question 1.2 Quelle est la complexité de cet algorithme ?

Solution : *La phase d'initialisation s'exécute en $O(V)$, la phase de relâchement en $O(VE)$, et la phase de recherche de circuit négatif en $O(E)$. Soit une complexité totale en $O(VE)$.* □

Question 1.3 Montrer qu'après le relâchement de l'arc (u, v), $d[v] \leq d[u] + w(u, v)$.

Solution : *Le relâchement se fait ainsi :*

si $d[v] > d[u] + w(u, v)$ ***alors***
$d[v] \leftarrow d[u] + w(u, v)$, $\pi[v] \leftarrow u$

Donc on a 2 cas :
- *soit $d[v] \leq d[u] + w(u, v)$, et dans ce cas rien n'est fait, la propriété est donc conservée*
- *soit $d[v] > d[u] + w(u, v)$, et dans ce cas on fait $d[v] \leftarrow d[u] + w(u, v)$, donc $d[v] = d[u] + w(u, v)$, la propriété est vraie.*

□

Question 1.4 Propriété du majorant : Montrer que $d[v] \geq \delta(s, v)$ pour tout sommet v à tout instant de l'algorithme. Montrer que lorsque $d[v]$ a atteint sa borne inférieure $\delta(s, v)$, il n'est plus jamais modifié.

Solution : *Montrons l'invariant $d[v] \geq \delta(s, v)$ pour tout $v \in V$ en raisonnant par récurrence sur le nombre d'étapes de relâchement. En phase d'initialisation on a bien $d[v] \geq \delta(s, v)$ puisque $d[v] = \infty, \forall v \in V - \{s\}$, et $d[s] = 0 \geq \delta(s, s)$ (on a $\delta(s, s) = -\infty$ si s se trouve sur un circuit de longueur strictement négative, et 0 sinon).*

Considérons le relâchement d'un arc (u, v). L'hypothèse de récurrence implique que $d[x] \geq \delta(s, x)$ pour tout $x \in V$ avant le relâchement. La seule valeur d pouvant changer est $d[v]$. Si elle change, on a

$$\begin{aligned} d[v] &= d[u] + w(u, v) \\ &\geq \delta(s, u) + w(u, v) \text{ (d'après l'hypothèse de récurrence)} \\ &\geq \delta(s, v) \text{ (d'après l'inégalité triangulaire : } \forall (u, v),\ \delta(s, v) \leq \delta(s, u) + w(u, v)) \end{aligned}$$

et donc l'invariant est conservé.

La valeur n'est plus modifiée par la suite une fois que $d[v] = \delta(s, v)$, car après avoir atteint sa borne inférieure $d[v]$ ne peut plus diminuer, et il ne peut pas non plus augmenter puisque les étapes de relâchement n'augmentent pas les valeurs de d. □

Question 1.5 Propriété de convergence : Considérons un plus court chemin $(s, \ldots, u, v)$ pour un sommet v donné. Montrer que si $d[u] = \delta(s, u)$ à un moment précédent le relâchement de (u, v) dans l'algorithme, alors on a toujours $d[v] = \delta(s, v)$ après l'appel.

Solution : *D'après la propriété du majorant, si $d[u] = \delta(s, u)$ à un moment donné précédant le relâchement de l'arc (u, v), alors $d[v] \geq \delta(s, v)$ est vérifiée par la suite. En particulier, après le relâchement de l'arc (u, v), on a*

$$\begin{aligned} d[v] &\leq d[u] + w(u,v) \text{ (question 3)} \\ &= \delta(s,u) + w(u,v) \\ &= \delta(s,v) \end{aligned}$$

La dernière égalité provient de la sous-structure optimale d'un plus court chemin. □

Question 1.6 Montrer que, si G ne contient aucun circuit de poids négatif accessible depuis s, alors, à la fin de l'exécution de l'algorithme, $d[v] = \delta(s,v)$ pour tout sommet v accessible depuis s.

Solution : *On démontre ce lemme en faisant appel à la* ***propriété de relâchement de chemin*** *: si $p = \langle v_0, v_1, \ldots, v_k \rangle$ est un plus court chemin dans G de $s = v_0$ à v_k et si les arcs de p sont relâchés dans l'ordre (v_0, v_1), (v_1, v_2), ..., (v_{k-1}, v_k), alors $d[v_k] = \delta(s, v_k)$. Cette propriété est vraie même si d'autres étapes de relâchement interviennent entre les relâchements des arcs de p.*

Démonstration : par récurrence, montrons qu'après le relâchement du ième arc de p, on a $d[v_i] = \delta(s, v_i)$. Pour $i = 0$, on n'a pas encore réalisé de relâchement, et on a donc $d[v_0] = d[s] = 0 = \delta(s,s)$ (pas de circuit de poids négatif). Induction : on suppose que $d[v_{i-1}] = \delta(s, v_{i-1})$. D'après la propriété de convergence on a $d[v_i] = \delta(s, v_i)$, et cette inégalité ne bouge plus ensuite.

Montrons maintenant que, si G ne contient aucun circuit de poids négatif accessible depuis s, alors, à la fin de l'exécution de l'algorithme, $d[v] = \delta(s,v)$ pour tout sommet v accessible depuis s. Soit un sommet v accessible depuis s, et soit un plus court chemin de s à v : $p = \langle v_0, v_1, \ldots, v_k \rangle$, avec $v_0 = s$ et $v_k = v$. Le chemin p a au plus $|V| - 1$ arcs, et donc $k \leq |V| - 1$. Chacune des $|V| - 1$ itérations de la boucle ***pour*** *des lignes 4 à 7 relâche tous les $|E|$ arcs. Parmi les arcs relâchés dans la ième itération, pour $i = 1, 2, \ldots, k$, il y a (v_{i-1}, v_i). D'après la propriété de relâchement de chemin, on a donc $d[v] = d[v_k] = \delta(s, v_k) = \delta(s,v)$.* □

Question 1.7 Montrer que pour chaque sommet v, il existe un chemin de s vers v si et seulement si BELLMAN-FORD se termine avec $d[v] < \infty$.

Solution : ***Propriété aucun-chemin :*** *S'il n'y a pas de chemin de s à v, alors on a toujours $d[v] = \delta(s,v) = \infty$.*
Démonstration : d'après la propriété de majorant, on a toujours $\infty = \delta(s,v) \leq d[v]$, d'où $d[v] = \infty = \delta(s,v)$. □

Nous pouvons à présent démontrer la validité de l'algorithme.

Question 1.8 Montrer que si G ne contient aucun circuit de poids négatif accessible depuis s, alors l'algorithme retourne VRAI, on a $d[v] = \delta(s,v)$ pour tous les sommets $v \in V$, et le sous-graphe G_π des sommets v dont $\pi(v) \neq$ NIL et des arcs $(\pi(v), v)$ est une arborescence des plus courts chemins de racine s. Montrer que si G contient un circuit de poids négatif accessible à partir de s, l'algorithme renvoie FAUX (on pourra raisonner par l'absurde).

Solution : *Supposons que G ne contienne aucun circuit de longueur strictement négative accessible depuis s. Si v est accessible depuis s, alors d'après la question 1.6 on a $d[v] = \delta(s,v)$. Si v n'est pas accessible depuis s, alors la propriété se déduit de la propriété aucun-chemin, et $d[v] = \infty = \delta(s,v)$. La propriété est donc vérifiée.*

Propriété de sous-graphe prédécesseur : *une fois que $d[v] = \delta(s,v)$ pour tout $v \in V$, le sous-graphe prédécesseur est une arborescence de plus courts chemins de racine s.*
Démonstration : Il nous faut les trois propriétés suivantes

- *les nœuds de G_π sont accessibles depuis s : par définition $\delta(s,v)$ est fini si et seulement si v est accessible depuis s, et donc les sommets accessibles depuis s sont exactement ceux dont l'attribut d a une valeur finie. Mais $d[v]$ a une valeur finie pour un sommet $v \in V - \{s\}$ si et seulement si $\pi[v] \neq NIL$. Donc les sommets de G_π sont accessibles depuis s.*
- *G_π forme une arborescence de racine s : après l'initialisation, G_π ne contient que le sommet d'origine. Considérons maintenant un sous-graphe prédécesseur G_π obtenu après une séquence d'étape de relâchement. Montrons que G_π est sans circuit par l'absurde. On suppose que G_π contient un circuit $c = \langle v_0, v_1, \ldots, v_k \rangle$, avec $v_0 = v_k$. On a donc $\pi[v_i] = v_{i-1}$ pour $i = 1, 2, \ldots, k$, et on peut supposer sans perte de généralité que le circuit est du au relâchement de l'arc (v_{k-1}, v_k).*

  Tous les sommets du circuit c sont accessibles depuis l'origine s, car ils ont tous un prédécesseur différent de NIL, et on donc une longueur de plus court chemin finie.

  Juste avant l'appel à relâcher pour (v_{k-1}, v_k), on a $\pi[v_i] = v_{i-1}$ pour $i = 1, 2, \ldots, k-1$, et la dernière mise à jour de $d[v_i]$ était $d[v_i] \leftarrow d[v_{i-1}] + w(v_i, v_{i-1})$, et si $d[v_{i-1}]$ a été modifié depuis, il a diminué. On a donc juste avant relâcher : $d[v_i] \geq d[v_{i-1}] + w(v_i, v_{i-1})$. Puisque $\pi[v_k]$ est modifié par l'appel, on a également avant l'appel $d[v_k] > d[v_{k-1}] + w(v_k, v_{k-1})$. Donc la longueur du plus court chemin le long du cycle c vaut :

$$\sum_{i=1}^{k} d[v_i] > \sum_{i=1}^{k} (d[v_{i-1}] + w(v_{i-1}, v_i))$$

  Mais, $\sum_{i=1}^{k} d[v_i] = \sum_{i=1}^{k} d[v_{i-1}]$. On a donc $0 > \sum_{i=1}^{k} w(v_{i-1}, v_i)$. Donc, le circuit a un poids strictement négatif, ce qui contredit l'hypothèse selon laquelle G ne contient pas de circuit de longueur strictement négative. Donc G_π est un graphe orienté sans circuit.

  Il faut ensuite montrer par récurrence que G_π forme une arborescence de racine s : montrer que pour chaque sommet $v \in S_\pi$, il existe un unique chemin de s à v dans G_π.
- *G_π arborescence de plus courts chemins : $\forall v \in V_\pi$, le chemin simple unique p de s à v dans G_π est un plus court chemin entre s et v dans G. Soit $p = \langle v_0, v_1, \ldots, v_k \rangle$, où $v_0 = s$ et $v_k = v$. Pour $i = 1, 2, \ldots, k$, on a à la fois $d[v_i] = \delta(s, v_i)$ et $d[v_i] \geq d[v_{i-1}] + w(v_{i-1}, v_i)$, d'où l'on conclut que $w(v_{i-1}, v_i) \leq \delta(s, v_i) - \delta(s, v_{i-1})$. En sommant les poids le long du chemin p, on obtient*

$$\begin{aligned} w(p) &= \sum_{i=1}^{k} w(v_{i-1}, v_i) \\ &\leq \sum_{i=1}^{k} (\delta(s, v_i) - \delta(s, v_{i-1})) \\ &= \delta(s, v_k) - \delta(s, v_0) \\ &= \delta(s, v_k) \end{aligned}$$

  Donc, $w(p) \leq \delta(s, v_k)$. Comme $\delta(s, v_k)$ est une borne inférieure de la longueur d'un chemin quelconque de s à v_k, on conclut que $w(p) = \delta(s, v_k)$ et donc que p est un plus court chemin de s vers $v = v_k$.

Revenons à la démonstration de la validité de l'algorithme. Avec la propriété précédente, et le fait que $d[v] = \delta(s, v)$, cela implique que G_π est une arborescence de plus courts chemins.

L'algorithme retourne vrai si G ne contient aucun circuit de poids négatif ? À la fin de l'algorithme on a $\forall (u, v) \in E$

$$\begin{aligned} d[p] &= \delta(s, v) \\ &\leq \delta(s, u) + w(u, v) \\ &= d[u] + w(u, v) \end{aligned}$$

et donc aucun des tests de la ligne 9 de l'algorithme ne permet de retourner faux.

Supposons maintenant qu'il existe un circuit de longueur strictement négative $c = \langle v_0, v_1, \ldots, v_k \rangle$, où $v_0 = v_k$, accessible depuis l'origine s. Alors,

$$\sum_{i=1}^{k} w(v_{i-1}, v_i) < 0$$

Supposons que l'algorithme retourne vrai. Alors $d[v_i] \leq d[v_{i-1}] + w(v_{i-1}, v_i)$, pour $i = 1, 2, \ldots, k$. On a donc, en sommant sur le circuit c

$$\begin{array}{rcl} \sum_{i=1}^{k} d[v_i] & \leq & \sum_{i=1}^{k} (d[v_{i-1}] + w(v_{i-1}, v_i)) \\ & = & \sum_{i=1}^{k} d[v_{i-1}] + \sum_{i=1}^{k} w(v_{i-1}, v_i) \end{array}$$

Comme $v_0 = v_k$, chaque sommet de c apparaît exactement une fois dans chacune des sommes. Donc

$$\sum_{i=1}^{k} d[v_i] = \sum_{i=1}^{k} d[v_{i-1}]$$

De plus, vu que $d[v_i]$ est fini, on a

$$0 \leq \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

Ce qui contredit l'inégalité précédente $\sum_{i=1}^{k} w(v_{i-1}, v_i) < 0$. Donc l'algorithme retourne vrai uniquement s'il n'y a pas de circuit de longueur strictement négative, et faux sinon. □

2 L'algorithme de Johnson

On s'intéresse maintenant au problème consistant à calculer les plus courts chemins pour tout couple de sommets du graphe. L'algorithme de Johnson a une bonne complexité pour les graphes creux (peu d'arêtes en comparaison du nombre de sommets). Il utilise les algorithmes de Dijkstra et de Bellman-Ford, et renvoie soit la matrice des poids des plus courts chemins, soit l'assertion que le graphe possède un circuit de poids négatif.

La technique utilisée consiste à se ramener au cas avec uniquement des poids positifs, puis de faire tourner l'algorithme de Dijkstra en partant de chaque sommet. Si les poids ne sont pas tous positifs, on les redéfinit pour pouvoir se ramener au premier cas.

Soit w la fonction des poids avant redéfinition. La nouvelle fonction de poids $\hat{w}$ doit respecter deux propriétés :

1. Pour chaque couple de sommets $u, v \in V$, un plus court chemin de u à v en utilisant la fonction de poids w est également un plus court chemin pour la fonction $\hat{w}$.
2. Pour chaque arête (u, v), le nouveau poids $\hat{w}(u, v)$ n'est pas négatif.

Dans la suite, on note comme dans l'exercice précédent δ pour la fonction des poids des plus courts chemins (avec w), et $\hat{\delta}$ représente la fonction des poids des plus courts chemins en utilisant la fonction de poids $\hat{w}$.

Voici l'algorithme de Dijkstra. Avec $\forall u \in V, adjacent[u]$ la liste des sommets adjacents à u.

Algorithme 2 Dijkstra(G, w, s)

```
1: pour tout sommet v ∈ V faire // Initialisation
2:    d[v] ← ∞, π[v] ← NIL
3: d[s] ← 0
4: E ← ∅
5: F ← V[G] // ensemble des sommets de G
6: tantque F ≠ ∅ faire
7:    u ← v|d[v] = min{d[x]|x ∈ F} // on choisit le sommet avec la plus petite valeur de d
8:    F ← F − {u}
9:    E ← E ∪ {u}
10:   pour tout sommet v ∈ adjacent[u] faire // relâchement de l'arc (u, v)
11:      si d[v] > d[u] + w(u, v) alors
12:         d[v] ← d[u] + w(u, v), π[v] ← u
```

Question 2.1 Propriété 1
Soit $h : V \to \mathbb{R}$ une fonction quelconque faisant correspondre un réel à chaque sommet. Pour chaque arête $(u, v) \in E$, on définit $\hat{w}(u, v) = w(u, v) + h(u) - h(v)$.

Soit $p = \langle v_0, v_1, ..., v_k \rangle$ un chemin allant de v_0 à v_k. Montrer que $w(p) = \delta(v_0, v_k)$ si et seulement si $\hat{w}(p) = \hat{\delta}(v_0, v_k)$. De plus, montrer que G possède un circuit de poids négatif en utilisant w si et seulement si G possède un circuit de poids négatif en utilisant $\hat{w}$.

Solution : *Commençons par montrer que* $\hat{w}(p) = w(p) + h(v_0) - h(v_k)$. *On a*

$$\begin{aligned} \hat{w}(p) &= \sum_{i=1}^{k} \hat{w}(v_{i-1}, v_i) \\ &= \sum_{i=1}^{k} (w(v_{i-1}, v_i) + h(v_{i-1}) - h(v_i)) \\ &= \sum_{i=1}^{k} w(v_{i-1}, v_i) + h(v_0) - h(v_k) \\ &= w(p) + h(v_0) - h(v_k) \end{aligned}$$

Donc, pour tout chemin p entre v_0 et v_k vérifie $w(p) + h(v_0) - h(v_k)$. Si un chemin est plus court qu'un autre pour la fonction w, alors il l'est également pour $\hat{w}$. Donc, $w(p) = \delta(v_0, v_k)$ si et seulement si $\hat{w}(p) = \hat{\delta}(v_0, v_k)$.

Montrons que si G possède un circuit de poids négatif en utilisant w alors il en est de même pour la fonction de poids $\hat{w}$. Soit $c = \langle v_0, v_1, ..., v_k \rangle$ un circuit quelconque tel que $v_0 = v_k$. On a donc $\hat{w}(p) = w(p) + h(v_0) - h(v_k) = w(c)$. □

Question 2.2 Propriété 2
On désire maintenant définir $\hat{w}$ vérifiant également la deuxième propriété (fonction non négative). On crée un nouveau graphe $G' = (V', E')$ avec $V' = V \cup \{s\}$ $(s \notin V)$ et $E' = E \cup \{(s, v) : v \in V\}$. La fonction de poids w est étendue de manière à ce que $w(s, v) = 0$ pour chaque $v \in V$.

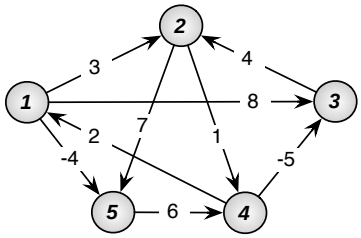
De plus, si G n'a pas de circuit de poids négatif, on définit $h(v) = \delta(s, v)$ pour chaque $v \in V'$.

2.2.1 A partir des indications précédentes, définir une nouvelle fonction de poids et montrer qu'elle vérifie les deux propriétés.

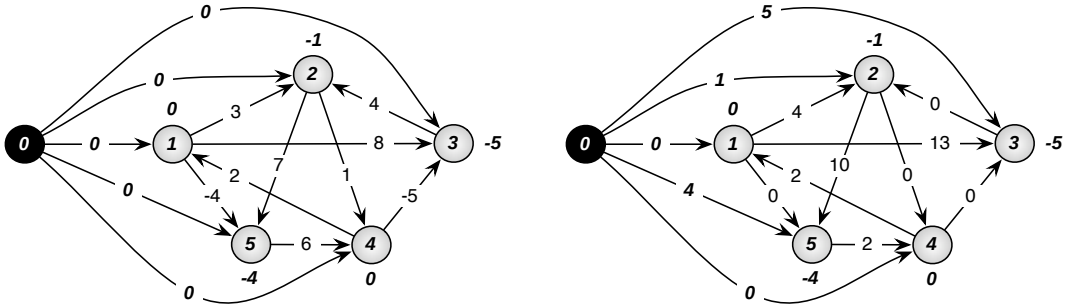
Solution : *En étendant la fonction de pondération de la sorte, on a vu qu'aucun arc n'entre dans s, aucun plus court chemin de G', hormis ceux d'origine s, ne contient s. De plus G' ne contient aucun circuit de longueur strictement négative si et seulement si G n'en contient aucun.*

Supposons que G et G' ne contiennent aucun circuit de longueur strictement négative. On définit $h(v) = \delta(s, v)$ pour tout $v \in V'$. D'après l'inégalité triangulaire on a $h(v) \leq h(u) + w(u, v)$ pour tout arc $(u, v) \in E'$. Donc, si l'on définit les nouveaux poids $\hat{w}$ comme étant : $\hat{w}(u, v) = w(u, v) + h(u) - h(v)$, on a bien $\hat{w}(u, v) \geq 0$, et la seconde propriété est vérifiée. □

2.2.2 Générer G' à partir du graphe G ci-dessous, et calculer les nouveaux poids.



Solution : *Voir figure 3.* □



(a) Fonction de pondération initiale w

(b) Fonction de pondération modifiée $\hat{w}$

FIG. 3: Graphe G'

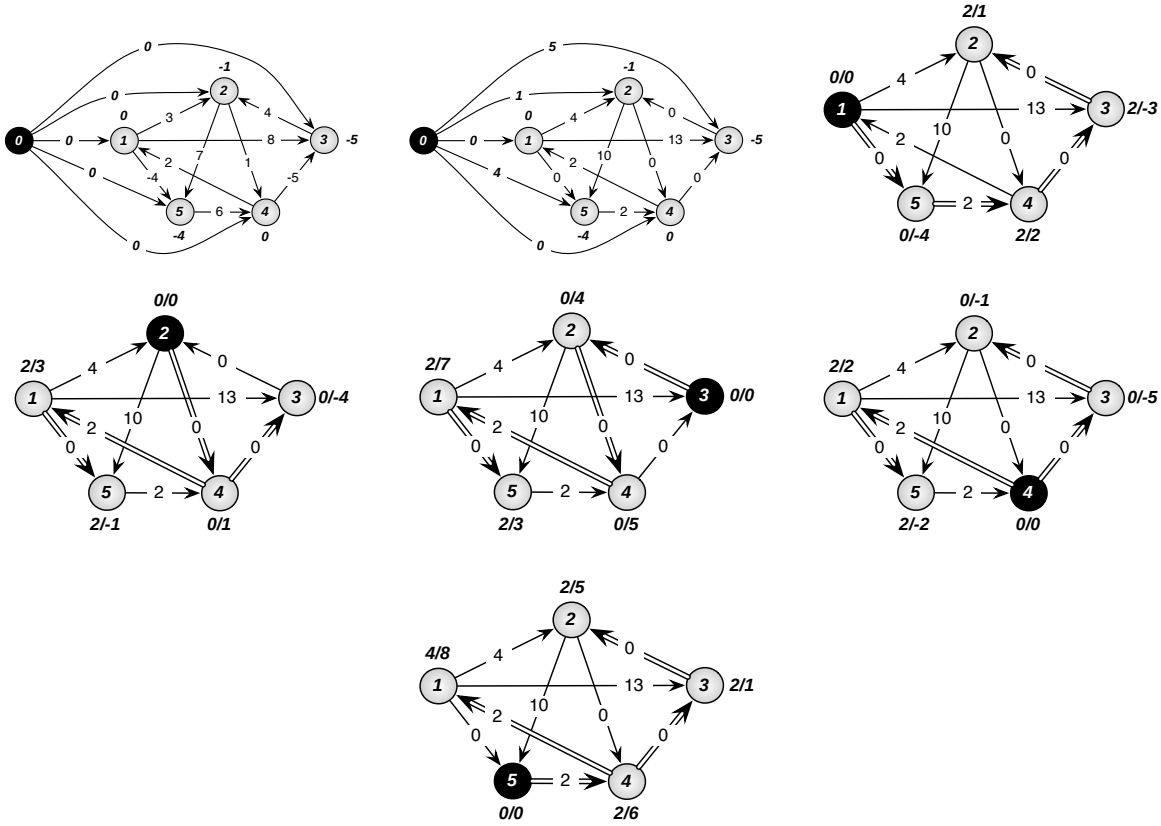
Question 2.3 Algorithme de Johnson
Écrire l'algorithme de Johnson en s'inspirant des questions précédentes. Faire tourner l'algorithme sur le graphe ci-dessus. Justifier la correction de l'algorithme et calculer sa complexité. On rappelle que l'algorithme de Dijkstra implémenté par un tas de Fibonacci a une complexité en $O(V log V + E)$.

Solution :

Algorithme 3 Johnson(G, w, s)

1: Calculer G', où $V[G'] = V[G] \cup \{s\}$, $E[G'] = E[G] \cup \{(s, v) : v \in V[G]\}$, et $w(s, v) = 0$ $\forall v \in V[G]$
2: **si** $Bellman - Ford(G', w, s) =$**Faux alors**
3: **print** "le graphe contient un circuit de longueur strictement négative"
4: **sinon**
5: **pour tout** sommet $v \in S[G']$ **faire**
6: $h(v) \leftarrow \delta(s, v)$ // $\delta(s, v)$ calculé avec Bellman-Ford
7: **pour tout** arc $(u, v) \in E[G']$ **faire**
8: $\hat{w}(u, v) \leftarrow w(u, v) + h(u) - h(v)$
9: **pour tout** sommet $u \in V[G]$ **faire**
10: Exécuter $Dijkstra(G, \hat{w}, u)$ // afin de calculer $\hat{\delta}(u, v)$, $\forall v \in V[G]$
11: **pour tout** sommet $v \in V[G]$ **faire**
12: $d_{u,v} \leftarrow \hat{\delta}(u, v) + h(v) - h(u)$
13: **retourner** D

La ligne 1 construit G'. La ligne 2 exécute l'algorithme de Bellman-Ford sur G' en utilisant la fonction de pondération w et le sommet d'origine s. Si G', et donc G, contient un circuit de longueur strictement négative, alors on signale le problème. Le reste de l'algorithme suppose donc que le graphe ne contient aucun circuit de longueur strictement négative, et calcule les longueurs des plus courts chemins $\hat{\delta}(u, v)$ en utilisant Dijkstra un fois pour chaque sommet de V. La ligne 12 stocke dans $d_{u,v}$ (élément de la matrice D de taille $|V| \times |V|$) la valeur $\delta(u, v)$.



Si la file de priorité min de Dijkstra est gérée avec un tas de Fibonacci, le temps d'exécution de l'algorithme est alors de $O(V^2 \log V + VE)$. L'implémentation, plus simple, basée sur un tas min binaire donne un temps d'exécution en $O(VE \log V)$. □