Plus courts chemins

Dans un problème de plus courts chemins, on possède en entrée un graphe orienté pondéré $G = (S,A)$ de fonction de pondération $w : A \to \mathbb{R}$. Le poids du chemin $p = \langle v_0, v_1, ..., v_k \rangle$ est la somme des poids de ses arcs :

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

Le poids $\delta(u,v)$ d'un plus court chemin d'un sommet u à un sommet v est bien évidemment le minimum des poids des chemins de u à v (si celui-ci est défini, ce qui peut ne pas être le cas si le graphe contient un circuit de poids strictement négatif). Un **plus court chemin** d'un sommet u à un sommet v est alors un chemin de u à v de poids $\delta(u,v)$.

0.1 Plus courts chemins à origine unique

On souhaite dans cette section trouver les plus courts chemins depuis un sommet origine s et vers n'importe quel autre sommet.

Dans la suite, $\pi[u]$ désignera le prédécesseur de u dans l'estimation du plus court chemin de s à u et $d[u]$ désignera la longueur de ce chemin.

0.1.1 Algorithme de Dijkstra

L'algorithme de Dijkstra résout le problème de la recherche d'un plus court chemin à origine unique pour un graphe orienté pondéré $G = (S,A)$ dans le cas où **tous les arcs ont un poids positif ou nul** : $\forall (u,v) \in A, w(u,v) \geq 0$.

L'algorithme de Dijkstra maintient à jour un ensemble E des sommets de G dont le plus court chemin à partir de l'origine s est connu et calculé. À chaque itération, l'algorithme choisit parmi les sommets de $S \setminus E$ — c'est-à-dire parmi les sommets dont le plus court chemin à partir de l'origine n'est pas connu — le sommet u dont l'estimation de plus court chemin est minimale. Cet algorithme est donc **glouton**. Une fois un sommet u choisi, l'algorithme met à jour, si besoin est, les estimations des plus courts chemins de ses successeurs (les sommets qui peuvent être atteint directement à partir de u).

SOURCE-UNIQUE-INITIALIZATION initialise les valeurs de $\pi[u]$ et de $d[u]$ pour chaque sommet u : initialement, il n'y a pas de chemin connu de s à u (si $u \neq s$) et u est estimé être à une distance infinie de s.

RELÂCHER(u, v, w) compare le plus court chemin de s à v connu avec une nouvelle proposition (chemin estimé de s à u puis arc de u à v), et met les différentes données à jour si besoin est.

L'algorithme est présenté figure 1.

Cet algorithme fourni effectivement les plus courts chemins. L'algorithme glouton fonctionne uniquement parce que les poids sont positifs. On montre la correction de l'algorithme par récurrence.

— Le premier sommet ajouté à E est s car $d[s]$ vaut alors 0 quand toutes les autres distances estimées sont infinies.

— Supposons qu'à un instant donné pour chaque sommet u de E, $d[u]$ est bien la longueur du plus court chemin de s à u. On rajoute alors un sommet v à E. $d[v]$ est alors minimale parmi les sommets de $S \setminus E$. Montrons que $d[v] = \delta(s,v)$.

Soit p un plus court chemin de s à v Soit y le premier sommet de p n'appartenant pas à E. Par minimalité de $d[v]$ on a : $d[v] \leq d[y]$. De plus on a $d[y] = \delta(s,y)$: parce que p contient le plus court chemin de s à x et donc de s au prédécesseur z de y, parce que $d[z] = \delta(s,z)$ par hypothèse de récurrence, et finalement parce que z a été relâché.

Par positivité des poids, $\delta(s,y) \leq \delta(s,v)$. Donc $d[v] \leq d[y] = \delta(s,y) \leq \delta(s,v)$ et $d[v] = \delta(s,v)$.

```
SOURCE-UNIQUE-INITIALIZATION[G, s]
  pour chaque sommet v de G faire
    d[v] ← +∞
    π[v] ← NIL
  d[s] ← 0

RELÂCHER(u, v, w)
  si d[v] > d[u] + w(u,v) alors
    d[v] ← d[u] + w(u,v)
    π[v] ← u

DIJKSTRA(G, w, s)
  SOURCE-UNIQUE-INITIALIZATION(G, s)
  E ← ∅
  F ← S
  tant que F ≠ ∅ faire
    u ← EXTRAIRE-MIN(F)
    E ← E ∪ {u}
    pour chaque arc (u,v) de G faire
      RELÂCHER(u, v, w)
```

FIGURE 1 – Algorithme de Dijkstra pour le calcul des plus courts chemins.

La figure 2 présente un exemple d'exécution de l'algorithme de Dijkstra.

Complexité

La complexité de l'algorithme dépend de la complexité de l'opération EXTRAIRE-MIN. Dans le cas (défavorable) où on implémente l'ensemble F au moyen d'un simple tableau, la recherche du minimum coûte à chaque fois $\Theta(|F|) = O(|S|)$. La boucle « tant que » s'exécutant exactement $|S|$ fois, et chaque arc étant visité une unique fois, la complexité de l'algorithme est $O(|S|^2 + |A|) = O(|S|^2)$.

0.1.2 Algorithme de Bellman-Ford

L'algorithme de Bellman-Ford résout le problème des plus courts chemins avec origine unique dans le cas général où le poids des arcs peut être négatif. Appelé sur un graphe $G = (S,A)$, l'algorithme de Bellman-Ford renvoie un booléen indiquant si le graphe contient ou non un circuit de poids strictement négatif accessible à partir de l'origine. L'algorithme est présenté figure 3.

Correction

La correction de l'algorithme de Bellman-Ford peut se montrer par récurrence sur le nombre d'arcs des plus courts chemins : à la fin de la i^e itération de la première boucle, les plus courts chemins contenant au plus i arcs sont connus, à la condition que le graphe ne contienne aucun circuit de poids strictement négatif. $|S|-1$ itérations suffisent car un plus court chemin est élémentaire (sans perte de généralité) et contient donc au plus $|S|-1$ arcs.

Vu ce qui précède, l'algorithme renvoie VRAI s'il n'y a pas de circuit de poids strictement négatif. Montrons qu'il renvoie FAUX sinon. Pour s'en convaincre, prenons un circuit c de sommets $u_0, u_1, ..., u_{p-1}, u_p u_0$. Si l'algorithme renvoie VRAI, alors pour tout $i \in [1,p]$, on a $d(u_i) \leq d(u_{i-1} + w(u_{i-1}, u_i)$. Par sommation on obtient :

$$\sum_{i=1}^{p} d(u_i) \leq \sum_{i=1}^{p} d(u_{i-1} + \sum_{i=1}^{p} w(u_{i-1}, u_i) \Leftrightarrow \sum_{i=1}^{p} d(u_i) \leq \sum_{i=0}^{p-1} d(u_i + w(c) \Leftrightarrow d(u_p) \leq d(u_0) + w(c) \Leftrightarrow 0 \leq w(c).$$

Donc, si l'algorithme renvoie VRAI le graphe ne contient pas de circuit de poids strictement négatif.

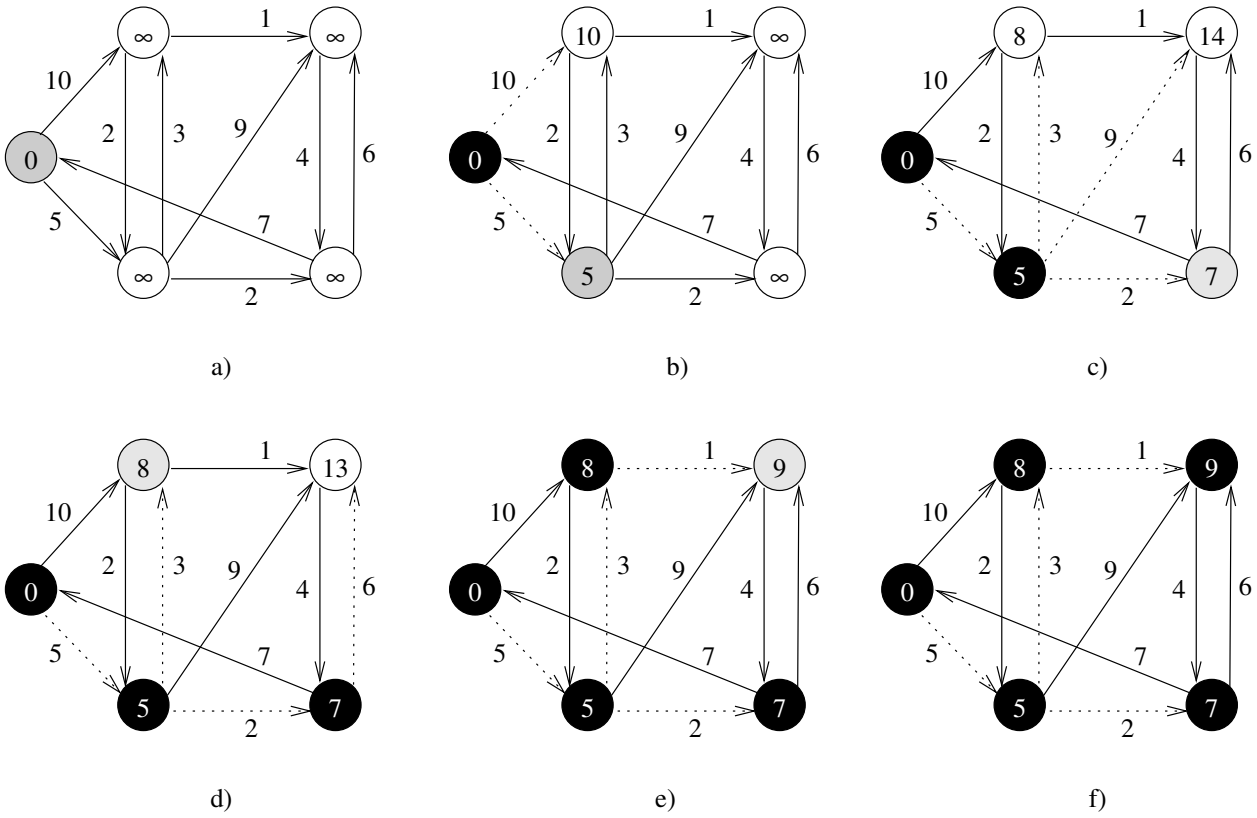


FIGURE 2 – Exemple d'exécution de l'algorithme de Dijkstra : l'origine est le sommet le plus à gauche ; dans chaque graphe, les sommets noirs sont éléments de E, le sommet grisé est celui qui va être rajouté à E et les arcs en pointillés sont ceux utilisés pour les estimations des plus courts chemins, les longueurs de ces chemins étant indiquées dans les sommets.

BELLMAN-FORD(G, s, w)
  SOURCE-UNIQUE-INITIALIZATION(G, s)
  pour $i \leftarrow 1$ **à** $|S|-1$ **faire**
    pour chaque arc $(u,v) \in A$ **faire**
      RELÂCHER(u, v, w)
  pour chaque arc $(u,v) \in A$ **faire**
    si $d[v] > d[u] + w(u,v)$ **alors renvoyer** FAUX
  renvoyer VRAI

FIGURE 3 – Algorithme de Bellman-Ford pour le calcul des plus courts chemins.

Complexité

Cet algorithme est en $\Theta(|S|.|A|)$ car l'initialisation et la vérification de la non-existence d'un circuit de poids strictement négatif sont en $\Theta(|S|)$ et $\Theta(|A|)$ respectivement, et car la boucle « pour » s'exécute exactement $(|S|-1)$ fois et que chaque itération visite chaque arc exactement une fois ce qui nous coûte $\Theta(|S|.|A|)$.

La figure 4 présente un exemple d'exécution de cet algorithme.

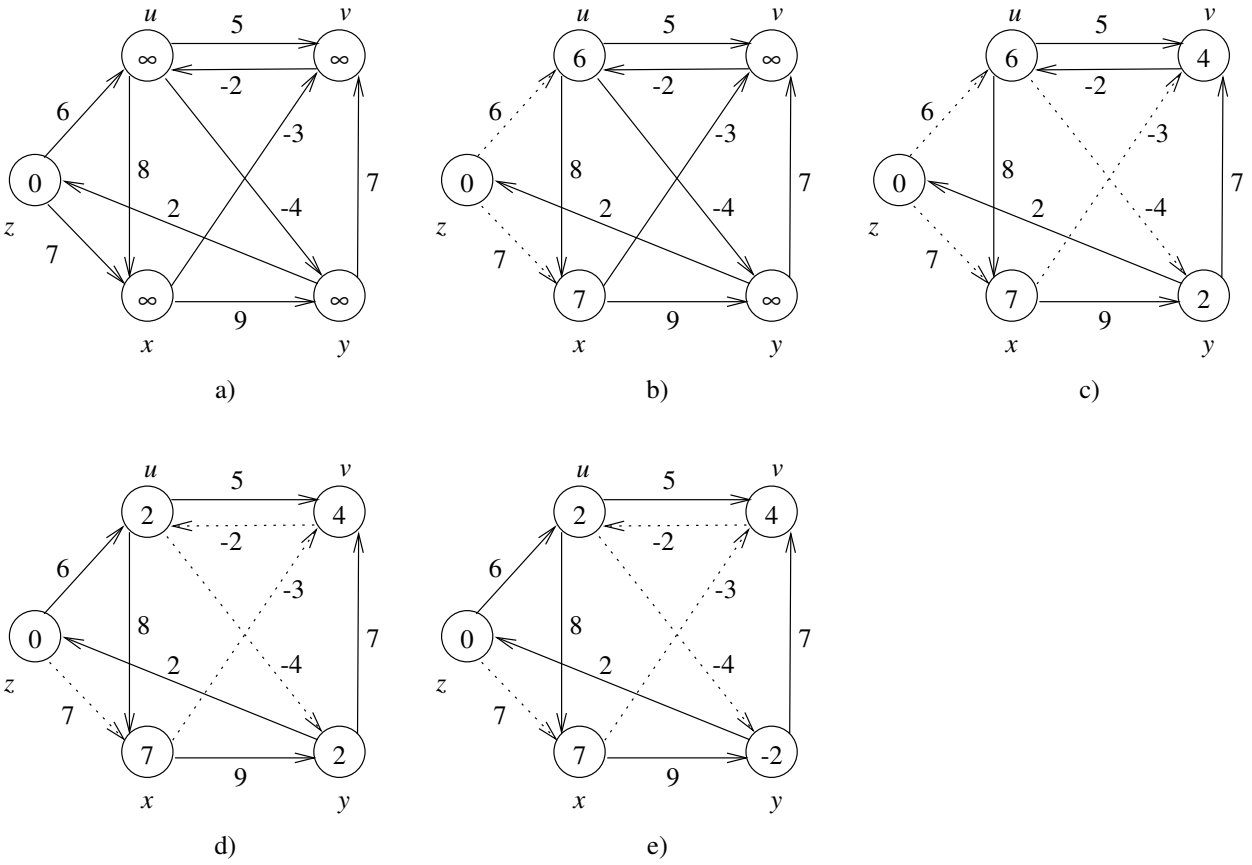


FIGURE 4 – Exemple d'exécution de l'algorithme de Bellman-Ford : l'origine est le sommet le plus à gauche ; dans chaque graphe les arcs en pointillés sont ceux utilisés pour les estimations des plus courts chemins, les longueurs de ces chemins étant indiquées dans les sommets. Les arcs sont considérés dans l'ordre lexicographique : (u,v), (u,x), (u,y), (v,u), (x,v), (x,y), (y,v), (z,u) et (z,x).

0.2 Plus courts chemins pour tout couple de sommets

Nous nous intéressons ici à la recherche des plus courts chemins entre tous les couples de sommets d'un graphe (typiquement on cherche à élaborer la table des distances entre tous les couples de villes d'un atlas routier). On dispose en entrée d'un graphe $G = (S,A)$ et d'une fonction de pondération w.

Nous supposons dans cette section qu'il peut y avoir des arcs de poids négatifs, mais qu'il n'existe pas de circuits de poids strictement négatifs.

0.2.1 Algorithme de Floyd-Warshall

Principe

On effectue une itération sur les sommets intermédiaires des chemins, un sommet intermédiaire étant un sommet autre que les extrémités du chemin en question. On note $\{1,2,...,n\}$ les n sommets de G. Ici, $d_{i,j}^{(k)}$ est la lon-

gueur du plus court chemin de i à j n'utilisant comme sommets intermédiaires que des sommets parmi $\{1,2,...,k\}$. De deux choses l'une, un plus court chemin de i à j n'ayant comme sommets intermédiaires que des sommets de $\{1,2,..,k\}$ contient ou ne contient pas le sommet k :

1. Si le plus court chemin p de i à j et n'ayant comme sommets intermédiaires que des sommets de $\{1,2,..,k\}$ a effectivement comme sommet intermédiaire k, alors p est de la forme $i \overset{p_1}{\rightsquigarrow} k \overset{p_2}{\rightsquigarrow} j$ où p_1 (resp. p_2) est un plus court chemin de i à k (resp. de k à j) n'ayant comme sommets intermédiaires que des sommets de $\{1,2,...,k-1\}$.
2. Si le plus court chemin p de i à j et n'ayant comme sommets intermédiaires que des sommets de $\{1,2,..,k\}$ ne contient pas k, alors c'est un plus court chemin p de i à j et n'ayant comme sommets intermédiaires que des sommets de $\{1,2,..,k-1\}$.

Résolution récursive

Le principe explicité au paragraphs précédent nous donne directement une récursion définissant $d_{i,j}^{(k)}$:

$$d_{i,j}^{(k)} = \begin{cases} w_{i,j} & \text{si } k = 0, \\ \min(d_{i,j}^{(k-1)}, d_{i,k}^{(k-1)} + d_{k,j}^{(k-1)}) & \text{sinon.} \end{cases}$$

Calcul itératif des poids des plus courts chemins

L'algorithme est présenté par la figure 5.

```
FLOYD-WARSHALL(W)
  n ← lignes(W)
  D^(0) ← W
  pour k ← 1 à n faire
    pour i ← 1 à n faire
      pour j ← 1 à n faire
        d_{i,j}^(k) ← min(d_{i,j}^(k-1), d_{i,k}^(k-1) + d_{k,j}^(k-1))
  renvoyer D^(n)
```

FIGURE 5 – Algorithme de Floyd-Warshall pour le calcul des plus courts chemins.

Construction des plus courts chemins

Tout comme on a défini les longueurs des plus courts chemins, on peut définir les prédécesseurs dans les plus courts chemins : $\pi_{i,j}^{(k)}$ représente ici le prédécesseur du sommet j dans le plus court chemin de i à j n'utilisant comme sommets intermédiaires que des sommets parmi $\{1,2,...,k\}$. Pour $k = 0$, un plus court chemin ne possède aucun sommet intermédiaire, donc :

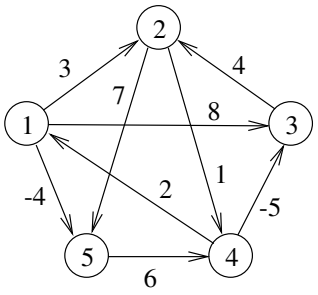
$$\pi_{i,j}^{(0)} = \begin{cases} \text{NIL} & \text{si } i = j \text{ ou } w_{i,j} = \infty, \\ i & \text{si } i \neq j \text{ et } w_{i,j} < \infty. \end{cases}$$

Dans le cas général, si le plus court chemin est de la forme $i \rightsquigarrow k \rightsquigarrow j$ le prédécesseur de j est le même que celui du plus court chemin de k à j et n'utilisant comme sommets intermédiaires que des sommets parmi $\{1,2,...,k-1\}$. Autrement, on prend le même prédécesseur de j que celui qui se trouvait sur le plus court chemin de i à j et n'utilisant comme sommets intermédiaires que des sommets parmi $\{1,2,...,k-1\}$. Nous avons donc, dans tous les cas :

$$\pi_{i,j}^{(k)} = \begin{cases} \pi_{i,j}^{(k-1)} & \text{si } d_{i,j}^{(k-1)} \leq d_{i,k}^{(k-1)} + d_{k,j}^{(k-1)}, \\ \pi_{k,j}^{(k-1)} & \text{si } d_{i,j}^{(k-1)} > d_{i,k}^{(k-1)} + d_{k,j}^{(k-1)}. \end{cases}$$

Complexité

On remarque aisément que l'algorithme de Floyd-Warshall est de complexité $\Theta(n^3)$.
La figure 6 présente le résultat de l'exécution de l'algorithme de Floyd-Warshall sur un exemple de graphe.



$$D^{(0)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \qquad \Pi^{(0)} = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & \text{NIL} & 4 & \text{NIL} & \text{NIL} \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \qquad \Pi^{(1)} = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \qquad \Pi^{(2)} = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(3)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \qquad \Pi^{(3)} = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(4)} = \begin{pmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \qquad \Pi^{(4)} = \begin{pmatrix} \text{NIL} & 1 & 4 & 2 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(5)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \qquad \Pi^{(5)} = \begin{pmatrix} \text{NIL} & 3 & 4 & 5 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$

FIGURE 6 – Séquence des matrice $D^{(k)}$ et $\Pi^{(k)}$ calculées par l'algorithme FLOYD-WARSHALL sur le graphe en haut de la figure.