université de BORDEAUX

Année 2020-2021 — DS novembre 2020
Épreuve de : Recherche Opérationnelle
Date : 17 novembre 2020 — **Durée :** 1h30
Épreuve de : O. Baudon — **Nb. pages :** 5

Une attention toute particulière sera portée à la présentation et à la rédaction de la copie.

Dans tous les exercices, on désignera par $V(G)$ et $E(G)$ respectivement l'ensemble des sommets et l'ensemble des arêtes d'un graphe G. Les variables n et m désigneront respectivement le nombre de sommets et d'arêtes.

Dans tout le sujet, on conviendra que, dans les différentes structures de données modélisant les graphes, les sommets sont rangés dans l'ordre croissant de leur nom.

1 Bellman

L'algorithme de Bellman est rappelé en fin d'exercice.

Appliquer l'algorithme de Bellman au graphe G de la Figure 1 pour calculer les plus courts chemins de s aux autres sommets du graphe. Donner
- l'ordre d'entrée des sommets dans la file F ;
- pour chaque sommet v la distance de s à v ;
- l'arborescence des plus courts chemins.

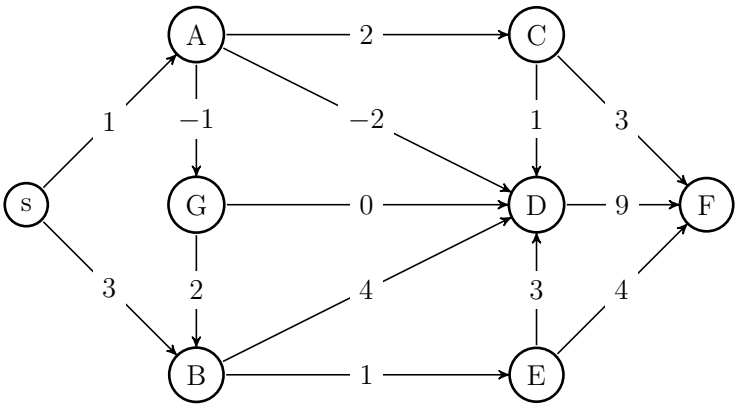

Figure 1 – Graphe G

```
0   Bellman(G,w,s)
1     pour chaque sommet v de V(G) faire
2         d[v] <- infini
3         pere[v] <- nil
4         npred[v] <- nombre de predecesseurs de v
5     d[s] <- 0
6     F <- CREER_FILE()
7     INSERER_FILE(F, s)
8     tant que F non vide faire
9         pivot <- TETE_FILE(F)
10        DEFILER(F)
11        pour tout v successeur de pivot faire
12            si d[v] > d[pivot] + w(pivot, v) alors
13                d[v] <- d[pivot] + w(pivot, v)
14                pere[v] <- pivot
15            npred[v] <- npred[v] - 1
16            si npred[v] = 0 alors
17                INSERER_FILE(F, v)
```

2 Détection d'arcs opposés

On considère un graphe orienté à n sommets et m arcs. On dit que deux arcs e_1 et e_2 sont opposés si $e_1 = (i, j)$ et $e_2 = (j, i)$ avec i et j deux sommets distincts du graphe.

Pour chacune des trois représentations : listes de successeurs, matrice d'adjacence, matrice d'incidence, écrire un algorithme pour déterminer si le graphe contient des arcs opposés et en étudier la complexité. *Attention de présenter les algorithmes de façon lisible !*

3 Plus courts chemins entre toutes paires de sommets

Rappels : l'algorithme de Floyd calcule les plus courts chemins entre toutes paires de sommets avec une complexité en $O(n^3)$. L'algorithme de Dijkstra calcule les plus courts chemins entre un sommet s et les autres sommets d'un graphe où tous les arcs ont un poids positif. En utilisant un tas de Fibonacci pour implémenter la file de priorité, la complexité de l'algorithme de Dijkstra est en $O(n.log(n) + m)$.

3.1) Il est possible, pour calculer les distances entre toutes paires de sommets, de remplacer l'algorithme de Floyd par n itérations de l'algorithme de Dijkstra, à condition que la fonction de poids soit positive. A quelle condition est-ce préférable ?

En fait, il est possible de contourner la limitation d'avoir tous les arcs de poids positifs.

Supposons que la fonction de poids w sur G ne soit pas positive. Si (G, w) ne possède pas de circuit strictement négatif, nous allons définir une nouvelle fonction de poids $\hat{w}$ sur G, positive et telle que (G, w) et $(G, \hat{w})$ aient les mêmes plus courts chemins.

Soit h une fonction de $V(G) \to \mathbb{R}$ et w une fonction de $E(G) \to \mathbb{R}$. Soit $\hat{w}$ la fonction de $E(G) \to \mathbb{R}$ définie par : $\forall (u, v) \in E(G), \hat{w}(u, v) = w(u, v) + h(u) - h(v)$

3.2) Montrer que $p = v_1, \ldots v_k$ est un plus court chemin de (G, w) si et seulement si p est un plus court chemin de $(G, \hat{w})$.

3.3) Montrer que (G, w) est sans circuit strictement négatif si et seulement si $(G, \hat{w})$ est sans circuit négatif.

Il nous faut maintenant trouver une fonction h en fonction de w et telle que $\hat{w}$ soit positive. D'après la question précédente, une telle fonction n'existera que si (G, w) est sans circuit strictement négatif. Pour cela, on se donne un graphe $G' = (V', E')$ avec $V' = V(G) \cup \{s'\}$ et $E' = E(G) \cup \{(s', v), v \in V(G)\}$. G' est donc un graphe obtenu à partir de G en rajoutant un sommet s' et tous les arcs entre s' et les sommets issus de $V(G)$. Le graphe G' est muni d'une fonction de poids w' définie par $\forall e \in E(G), w'(e) = w(e)$ et $\forall v \in V(G), w'(s', v) = 0$.

3.4) Montrer que pour tout graphe G, G' est sans circuit strictement négatif si et seulement si G est sans circuit strictement négatif.

3.5) Dessiner le graphe G' correspondant au graphe G de la figure 2.

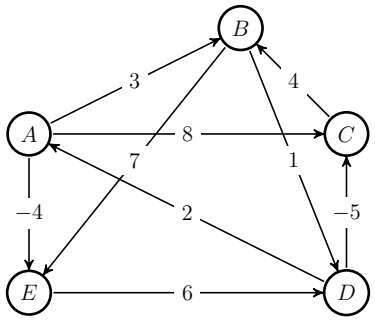

FIGURE 2 – Graphe G

La première étape de notre nouvel algorithme consiste à calculer les plus courtes distances entre s' et les autres sommets de G', à l'aide l'algorithme de Ford rappelé ci-dessous (Algorithme 1). Cette étape permet également de vérifier si G' (et donc G) est sans circuit strictement négatif.

3.6) Appliquer l'algorithme de Ford au graphe G' obtenu à la question précédente.

L'algorithme de Ford renvoie `VRAI` si et seulement si le graphe G muni de la fonction de poids sur les arcs w est sans circuit négatif. La fonction d donne les plus courtes distances entre le sommet s et les autres sommets du graphe G.

Algorithme 1 Ford(G, w, s)

```
1: pour tout v de V(G) faire
2:    d[v] ← ∞
3:    pere[v] ← NIL
4: fin pour
5: d[s] ← 0
6: pour i de 1 à n − 1 faire
7:    pour tout arc e = (u, v) ∈ E(G) faire
8:       si d[v] > d[u] + w(u, v) alors
9:          d[v] ← d[u] + w[u, v]
10:         pere[v] ← u]
11:      fin si
12:   fin pour
13: fin pour
14: pour tout arc e = (u, v) ∈ E(G) faire
15:   si d[v] > d[u] + w(u, v) alors
16:      retourner FAUX
17:   fin si
18: fin pour
19: retourner VRAI
```

Pour terminer notre algorithme, on calcule la fonction de poids $\hat{w}$ en prenant comme fonction h la distance dans le graphe G' entre s' et les sommets de G. Puis on applique l'algorithme de Dijkstra n fois en prenant à chaque fois une source différente.

3.7) Quelle est la complexité de l'algorithme de Ford ? Et celle de notre algorithme ?

3.8) Montrer que pour tout graphe G et fonction de poids w sans circuit négatif, la fonction $\hat{w}$ obtenue est positive.

3.9) Dessiner le graphe G de la figure 2 avec la fonction de poids $\hat{w}$.

3.10) Calculer les plus courtes distances entre toute paire de sommets dans $(G, \hat{w})$ en utilisant l'algorithme de Dijkstra rappelé ci-dessous (Algorithme 2) successivement à partir

de chacun des sommets. Pour chaque sommet source, on donnera l'arborescence des plus courts chemins.

Algorithme 2 Dijkstra(G, w, s)

```
1: pour tout v de V(G) faire
2:    d(v) ← ∞
3:    pere(v) ← NIL
4:    couleur(v) ← BLANC
5: fin pour
6: d(s) ← 0
7: F ← FILE_PRIORITE({s}, d)
8: tant que F ≠ ∅ faire
9:    pivot ← EXTRAIRE_MIN(F)
10:   pour tout e = (pivot, v) arc sortant de pivot faire
11:      si couleur(v) = BLANC alors
12:         si d(v) = ∞ alors
13:            INSERER(F, v)
14:         fin si
15:         si d[v] > d[pivot] + w(e) alors
16:            d[v] ← d[pivot] + w(e)
17:            pere[v] ← pivot
18:         fin si
19:      fin si
20:   fin pour
21:   couleur[pivot] ← NOIR
22: fin tant que
```

3.11) Donner la matrice des plus courtes distances dans (G, w).