université de BORDEAUX

Année 2023-2024 — Session de décembre 2023
Master MIAGe : 1ère année — UE : 4TYE705U
Épreuve de : Recherche Opérationnelle et Conduite de Projets
Partie : Recherche Opérationnellle
Durée : 1h30
Épreuve de : O. Baudon — **Nb. pages : 9**
Collège Sciences et technologies

Aucun document autorisé.
Une attention toute particulière sera portée à la présentation et à la rédaction de la copie.
Les algorithmes sont rappelés à la fin du document.

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.

1 Plus courts chemins

Les algorithmes de Dijkstra et Bellman sont rappelés à la fin du document : Algorithmes 1 et 2. Pour les deux graphes $G1$ et $G2$ des Figures 1 et 2, calculer les plus courts chemins entre le sommet s et les autres sommets du graphe. Pour chacun des deux graphes, on rappellera les conditions nécessaires à l'utilisation de l'algorithme choisi. Si vous utilisez Dijkstra, donner pour chaque itération la valeur du *pivot* et les modifications de la fonction d. Pour Bellman, donner pour chaque itération la valeur de la tête de la file F et les modifications de d et $npred$.
Dans les deux cas, dessiner l'arborescence des plus courts chemins.

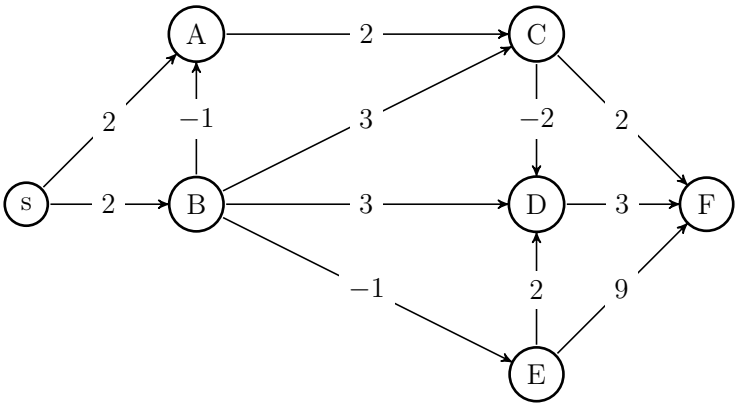

Figure 1 – Graphe $G1$

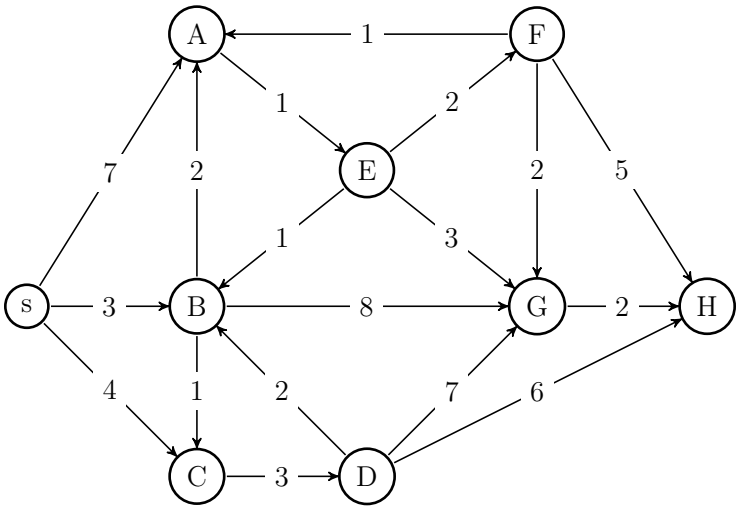

FIGURE 2 – Graphe $G2$

2 Flot Maximum

2.1) En utilisant l'algorithme de Ford-Fulkerson (Algorithmes 3 et 4), trouver le flot maximum entre les sommets s et t du réseau de la Figure 3, *en partant du flot initial déjà indiqué*. Cette valeur initiale est donnée par le premier nombre porté sur les arcs, le second étant la capacité de l'arc. Par exemple, l'arc `(s,A)` possède un flot initial de 5 et une capacité de 13. On donnera pour chaque exécution de la procédure de marquage le chemin augmentant obtenu et l'augmentation correspondante.

2.2) Donner la coupe minimum correspondante et redessiner le graphe avec le flot maximum.

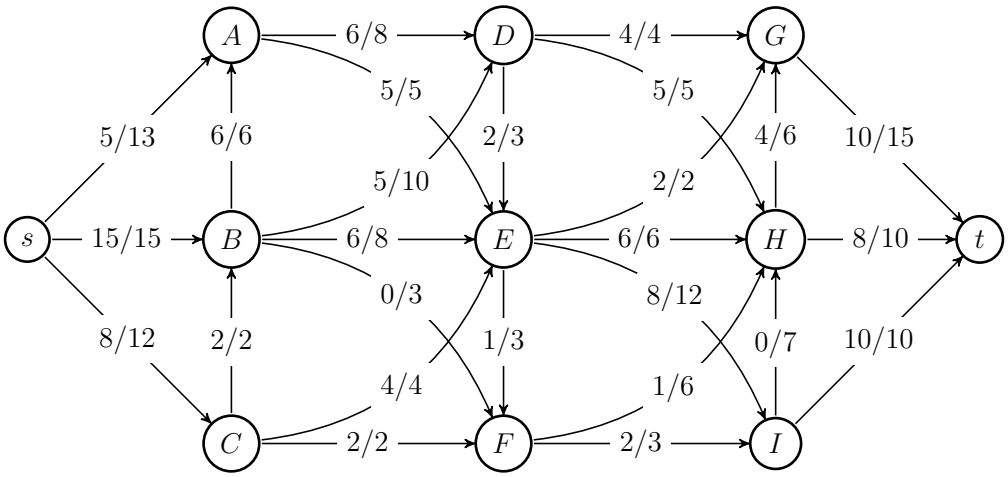

FIGURE 3 – Réseau

3 Plus courts circuits

Dans cet exercice, tous les circuits sont considérés élémentaires (sans répétition de sommet ni d'arc).

L'objectif de cet exercice est de concevoir un algorithme pour calculer la longueur minimum d'un circuit dans un graphe orienté donné, en se basant sur l'algorithme de parcours en largeur.

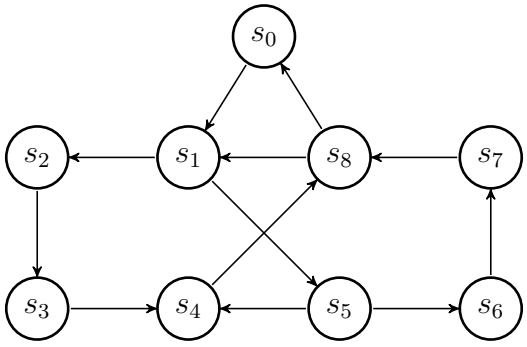

FIGURE 4 – Graphe orienté G_1.

1. Appliquer le parcours en largeur (Algorithme 5) au graphe G_1 de la Figure 4
2. Identifier tous les circuits du graphe G_1 de la Figure 4 contenant le sommet s_0 (il y en a trois). Quelle est la longueur minimum d'un circuit de $G1$ contenant s_0 ?
3. Y a-t-il dans le graphe G_1 un autre circuit encore plus court, ne passant pas par s_0 ? Si oui, quelle est sa longueur ?

4. Soit G un graphe orienté, v_0 un sommet de G et k la longueur minimum d'un circuit de G contenant v_0. Soit

$$C = v_0 e_1 v_1 e_2 v_2 \dots v_{k-1} e_k v_0$$

un circuit de longueur minimum contenant v_0 (donc de longueur k).
Montrer qu'à l'issue d'un parcours en largeur $PL(G, v_0)$ à partir du sommet v_0, on a $d[v_i] \leq i$ pour tout $i = 0, 1, \dots, k-1$.
Montrer qu'à l'issue d'un parcours en largeur $PL(G, v_0)$ à partir du sommet v_0, on a $d[v_i] = i$ pour tout $i = 0, 1, \dots, k-1$.

5. Soit deux sommets w_1 et w_2 deux sommets prédécesseurs d'un sommet $v0$ dans deux circuits distincts (respectivement C_1 et C_2) contenant v_0. Comment peut-on lors du parcours en largeur détecter lequel des deux circuits C_1 ou C_2 est le plus court ?
6. Comment peut-on détecter lors d'un parcours en largeur qu'un arc d'un graphe orienté est le dernier arc d'un circuit le plus court contenant le sommet de départ v_0 ? Proposer un algorithme $PCC(G, s)$ à partir d'une modification de $PL(G, s)$ qui calcule et retourne la longueur minimum d'un circuit contenant le sommet s de départ.
7. Proposer un algorithme pour calculer et retourner la longueur minimum d'un circuit dans un graphe orienté donné, en faisant des appels à l'algorithme PCC.
8. Sachant que le graphe G est représenté par ses listes d'adjacence, en étudier la complexité.

Algorithmes

Algorithme 1 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
```

Algorithme 2 Bellman(G, w, s)

```
1: pour tout v de V(G) faire
2:    d[v] ← ∞
3:    pere[v] ← NIL
4:    npred[v) ← deg⁻[v]
5:    si npred(v) = 0 alors
6:       INSERER_FILE(F, v)
7:    fin si
8: fin pour
9: d[s] ← 0
10: tant que F non vide faire
11:    u ← TETE_FILE(F)
12:    DEFILER(F)
13:    pour v ∈ Adj(u) faire
14:       si d[v] > d[u] + w(u, v) alors
15:          d[v] ← d[u] + w[u, v]
16:          pere[v] ← u
17:       fin si
18:       npred(v) ← npred[v] − 1
19:       si npred(v) = 0 alors
20:          INSERER_FILE(F, v)
21:       fin si
22:    fin pour
23: fin tant que
```

Dans l'Algorithme 3 (FlotMax), le paramètre c désigne les capacités du graphe G, s la source et t la destination du flot f calculé. $I(e)$ et $T(e)$ désignent respectivement le sommet initial et le sommet terminal d'un arc e.

Algorithme 3 FlotMax(G, c, s, t)

```
1: pour tout e de E(G) faire
2:    f[e] ← 0
3: fin pour
4: répéter
5:    Marquage(G, c, f, s, t)
6:    si t ∈ Y alors
7:       v ← t
8:       C⁺ ← {(t, s)}
9:       C⁻ ← ∅
10:      tant que v ≠ s faire
11:         e ← A[v]
12:         si v = T[e] alors
13:            C⁺ ← C⁺ ∪ {e}
14:            v ← I[e]
15:         sinon
16:            C⁻ ← C⁻ ∪ {e}
17:            v ← T[e]
18:         fin si
19:      fin tant que
20:   fin si
21:   pour tout e ∈ C⁺ faire
22:      f(e) ← f(e) + δ[t]
23:   fin pour
24:   pour tout e ∈ C⁻ faire
25:      f(e) ← f(e) − δ[t]
26:   fin pour
27: jusqu'à t ∉ Y
```

Algorithme 4 Marquage(G, c, f, s, t)

1: $Y \leftarrow \{s\}$
2: $\delta(s) \leftarrow +\infty$
3: $Max \leftarrow$ faux
4: **tant que** $t \notin Y et Max =$faux **faire**
5: **si** il existe $e = (u, v)$ avec $u \in Y, v \notin Y, f(e) < c(e)$ **alors**
6: $Y \leftarrow Y \cup \{v\}$
7: $A[v] \leftarrow e$
8: $\delta[v] \leftarrow min(\delta[u], c(e) - f(e))$
9: **sinon**
10: **si** il existe $e = (u, v)$ avec $v \in Y, u \notin Y, f(e) > 0$ **alors**
11: $Y \leftarrow Y \cup \{u\}$
12: $A[u] \leftarrow e$
13: $\delta[u] \leftarrow min(\delta[v], f(e))$
14: **sinon**
15: Max $\leftarrow vrai$
16: **fin si**
17: **fin si**
18: **fin tant que**

Algorithme 5 Parcours en largeur PL(G, s)

```
1: couleur(s) ← GRIS
2: d(s) ← 0
3: pere(s) ← NIL
4: F ← {s}
5: pour tout v ∈ V(G)\s faire
6:    couleur(v) ← BLANC
7:    d(v) ← ∞
8:    pere(v) ← NIL
9: fin pour
10: tant que F non vide faire
11:    v ← tete(F)
12:    pour tout w ∈ Adj(v) faire
13:       si couleur(w) = BLANC alors
14:          couleur(w) ← GRIS
15:          d(w) ← d(v) + 1
16:          pere(w) ← v
17:          Enfiler(F, w)
18:       fin si
19:    fin pour
20:    Defiler(F)
21:    couleur(v) ← NOIR
22: fin tant que
```