université de BORDEAUX

Année 2021-2022 — Session de décembre 2022
Master MIAGe : 1ère année — UE : 4TYE705U
Épreuve de : Recherche Opérationnelle et Conduite de Projets
Durée : 1h30
Épreuve de : O. Baudon — **Nb. pages : 11**
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 Composantes fortement connexes

1.1) En utilisant l'algorithme de calcul des composantes fortement connexes (Algorithme 6) basé sur le parcours en profondeur (Algorithmes 7 et 8), donnez les composantes fortement connexes du graphe de la Figure 1. Pour chaque parcours en profondeur, on dessinera l'arborescence et donnera les heures de début et fin de visite. On supposera que les listes des successeurs sont données dans l'ordre lexicographique et que le premier sommet visité lors du premier parcours en profondeur est le sommet A. On rappelle que le graphe G^{-1} désigne le graphe inverse de G, c'est à dire le graphe $(V(G), E(G^{-1}))$, avec $\forall u, v \in V(G), (u, v) \in E(G^{-1}) \iff (v, u) \in E(G)$.

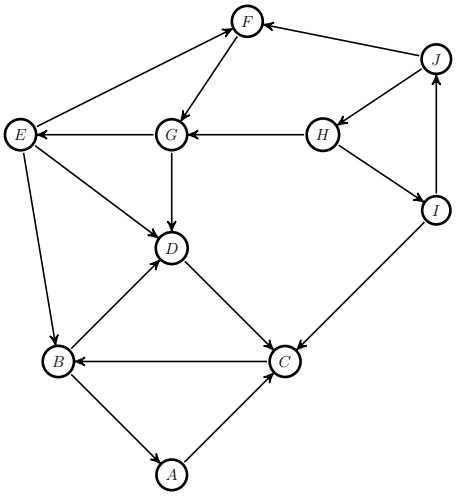

Figure 1 – Graphe G

2 Plus courts chemins

Les algorithmes de Dijkstra, Bellman et Ford sont rappelés à la fin du document : algorithmes 1, 2 et 3. Pour les trois graphes $G1$, $G2$ et $G3$ des Figures 2, 3 et 4, calculer les plus courts chemins entre le sommet s et les autres sommets du graphe. Pour chacun des graphes, on rappellera les conditions nécessaires à l'utilisation de l'algorithme choisi. Traiter les sommets dans l'ordre s, A, B, C, D, E.
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 $npred$. Pour Ford, pour chaque valeur de i, donner les modifications de d et de $pere$. Si le résultat est $FAUX$, donner le circuit négatif l'ayant engendré. Sinon, donner l'arborescence des plus courts chemins.

Dans les trois cas, dessiner l'arborescence des plus courts chemins.

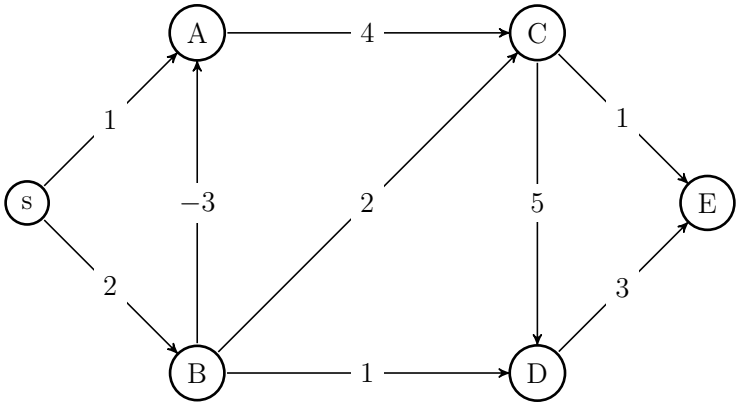

FIGURE 2 – Graphe $G1$

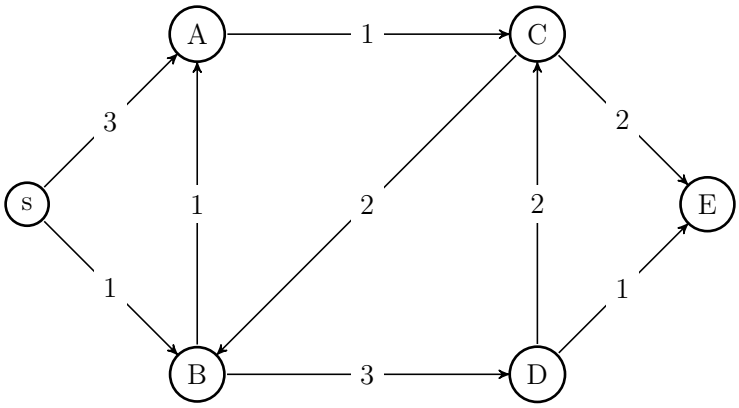

FIGURE 3 – Graphe $G2$

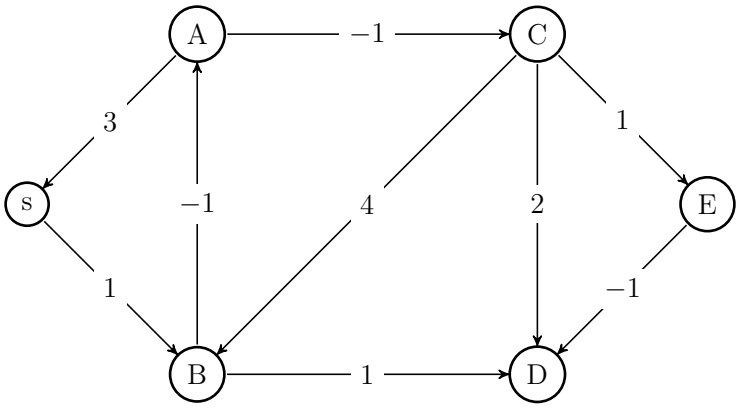

FIGURE 4 – Graphe $G3$

3 Flot Maximum

3.1) En utilisant l'algorithme de Ford-Fulkerson (Algorithmes 4 et 5), trouver le flot maximum entre les sommets s et t du réseau de la Figure 5. Le flot possède déjà une valeur initiale 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 1 et une capacité de 4. *Vous devez améliorer ce flot existant, sans repartir d'un flot nul.* On donnera pour chaque exécution de la procédure de marquage le chemin augmentant obtenu et l'augmentation correspondante.

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

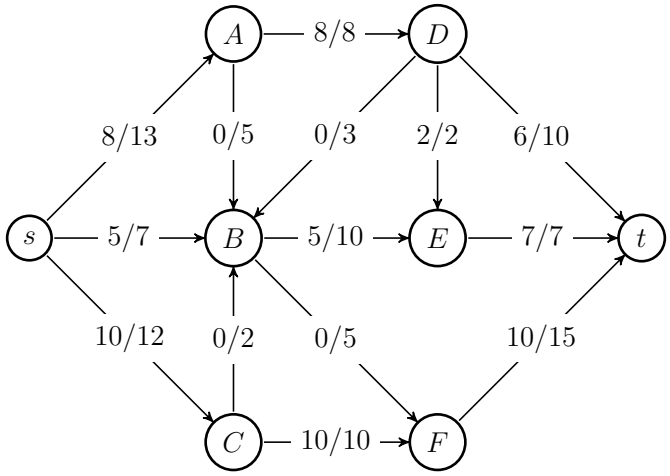

FIGURE 5 – Réseau

4 Graphe partiel connexe de poids minimum

Un graphe partiel d'un graphe $G = (V, E)$ est un graphe $G' = (V, E')$ avec $E' \subseteq E$.

4.1) Soit G un graphe simple connexe muni d'une fonction de poids w sur ses arêtes. Montrer par un exemple simple que si on autorise des valeurs négatives pour w, un graphe partiel connexe de poids total minimum n'est pas forcément un arbre.

4.2) Que faut-il modifier à l'algorithme de Kruskal (Algorithme 9), pour calculer un graphe partiel connexe de poids total minimum quand tous les poids ne sont pas positifs ?

4.3) Appliquer votre algorithme au graphe G_2 de la Figure 6. Préciser dans quel ordre les arêtes conservées ont été sélectionnées. Dessiner le graphe partiel de poids total minimum obtenu et donner la valeur de son poids.

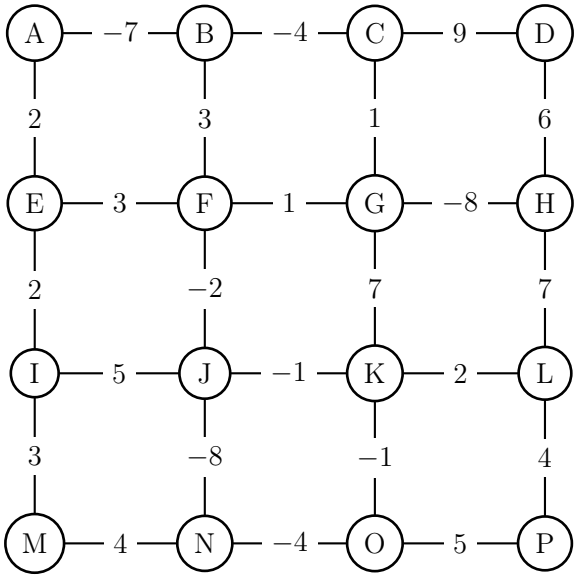

FIGURE 6 – Graphe G_2

Algorithmes

Dans les algorithmes de Dikstra (1), Bellman (2) et Ford (3), on a :
G : graphe orienté
$w : E(G) \to \mathbb{R}$
s : source de G

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
```

L'algorithme de Ford renvoie VRAI si le graphe G est sans circuit négatif, FAUX sinon.

Algorithme 3 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
```

Dans l'Algorithme 4 (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 4 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 5 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 6 Composantes fortement connexes CFC(G)

1: Exécuter PP(G)
2: Calculer G^{-1}
3: Exécuter PP(G^{-1}) en considérant les sommets dans la boucle 6 à 10 de PP(G^{-1}) dans l'ordre décroissant de la fonction f (fin de visite) obtenue à l'étape 1.
4: Retourner les arborescences obtenues comme composantes fortement connexes de G

Algorithme 7 Parcours en profondeur PP(G)

1: **pour tout** $v \in V(G)$ **faire**
2: $couleur(v) \leftarrow BLANC$
3: $pere(v) \leftarrow NIL$
4: **fin pour**
5: $temps \leftarrow 0$
6: **pour tout** $v \in V(G)$ **faire**
7: **si** $couleur(v) = BLANC$ **alors**
8: $VisiterPP(v)$
9: **fin si**
10: **fin pour**

Algorithme 8 VisiterPP(v)

1: $d(v) \leftarrow temps \leftarrow temps + 1$
2: $couleur(v) \leftarrow GRIS$
3: **pour tout** $w \in Adj(v)$ **faire**
4: **si** $couleur(w) = BLANC$ **alors**
5: $pere(w) \leftarrow v$
6: $VisiterPP(w)$
7: **fin si**
8: **fin pour**
9: $couleur(v) \leftarrow NOIR$
10: $f(v) \leftarrow temps \leftarrow temps + 1$

Dans l'algorithme de Kruskal (9), le paramètre $composante(u)$ désigne l'ensemble des sommets appartenant à la même composante connexe que u dans le graphe partiel contenant les arêtes de l'ensemble $E(T)$.

Algorithme 9 Kruskal(G, w)

pour tout v de $V(G)$ **faire**
  $composante(v) \leftarrow \{v\}$
fin pour
Trier $E(G)$ dans un ordre croissant $(e_1, \ldots, e_m)$ en fonction de $w(e)$
$i \leftarrow 1$
$E(T) \leftarrow \{\}$
tant que $|E(T)| < n - 1$ **faire**
  si $e_i = (u, v)$ et $composante(u) \neq composante(v)$ **alors**
    $E(T) \leftarrow E(T) \cup \{e_i\}$
    UNIFIER$(composante(u), composante(v))$
  fin si
  $i \leftarrow i + 1$
fin tant que
retourner $E(T)$