université de BORDEAUX

Année 2025-2026 — DS novembre 2025
Épreuve de : Recherche Opérationnelle
Date : 17 novembre 2025 — **Durée :** 1h30
Épreuve de : O. Baudon

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 Sommet source

Soit G un graphe orienté. Un sommet de G est une *source*, s'il n'a pas de voisin entrant.

1. Identifier toutes les sources du graphe G_1 de la figure 1.
2. Donner un exemple d'un graphe orienté qui n'a aucune source.
3. Pour chacune des représentations
   (a) matrice d'adjacence,
   (b) listes d'adjacence

   proposer un algorithme `Source(G,s)` qui retourne `True` si un sommet donné s d'un graphe orienté donné G est une source, et qui retourne `False` sinon.
4. Donner la complexité des algorithmes, en fonction du nombre de sommets n, du nombre d'arcs m et du degré sortant maximum Δ.

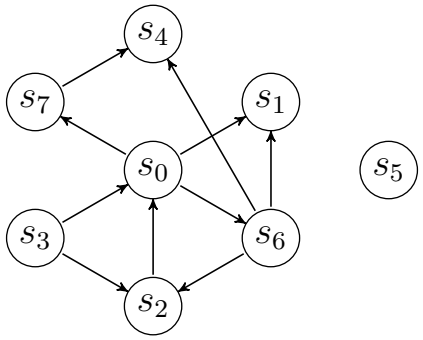

Figure 1 – Le graphe G_1

2 Plus courts chemins

Les algorithmes de Dijkstra (Algorithme 1) et Bellman (Algorithme 2) sont rappelés à la fin de l'exercice.

Pour les deux graphes $G1$ et $G2$ des Figures 2 et 3, 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.

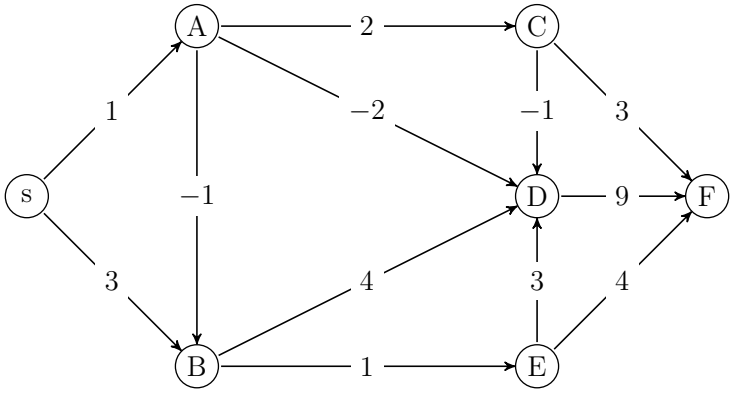

FIGURE 2 – Graphe $G1$

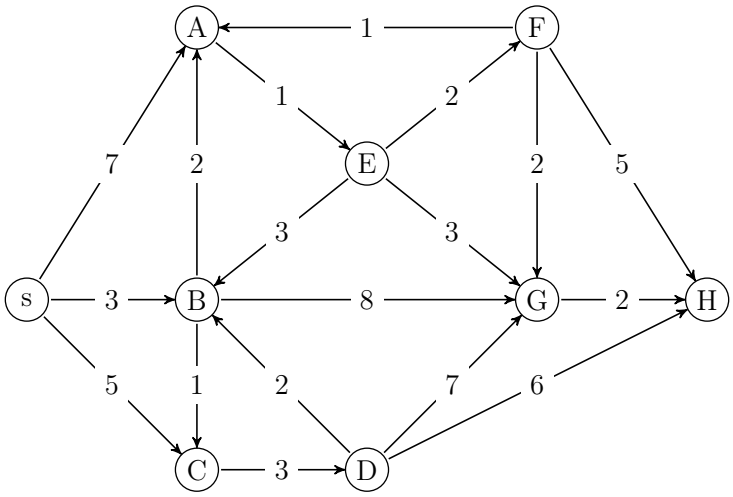

FIGURE 3 – Graphe $G2$

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)

H

1: **pour tout** v de $V(G)$ **faire**
2: $d[v] \leftarrow \infty$
3: $pere[v] \leftarrow NIL$
4: $npred[v) \leftarrow deg^{-}[v]$
5: **si** $npred(v) = 0$ **alors**
6: INSERER_FILE(F, v)
7: **fin si**
8: **fin pour**
9: $d[s] \leftarrow 0$
10: **tant que** F non vide **faire**
11: $u \leftarrow TETE\_FILE(F)$
12: DEFILER(F)
13: **pour** $v \in Adj(u)$ **faire**
14: **si** $d[v] > d[u] + w(u, v)$ **alors**
15: $d[v] \leftarrow d[u] + w[u, v]$
16: $pere[v] \leftarrow u$
17: **fin si**
18: $npred(v) \leftarrow npred[v] - 1$
19: **si** $npred(v) = 0$ **alors**
20: INSERER_FILE(F, v)
21: **fin si**
22: **fin pour**
23: **fin tant que**

3 Flot Maximum

3.1) 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 4. 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.

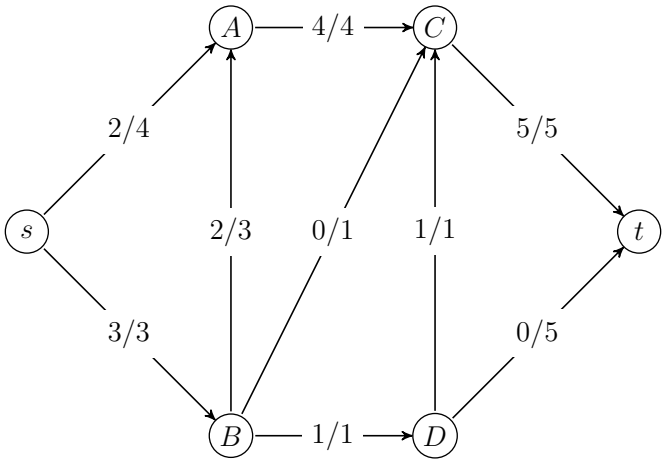

FIGURE 4 – Réseau

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 ← {s}
2: δ(s) ← +∞
3: Max ← faux
4: tant que t ∉ Y et Max =faux faire
5:    si il existe e = (u, v) avec u ∈ Y, v ∉ Y, f(e) < c(e) alors
6:       Y ← Y ∪ {v}
7:       A[v] ← e
8:       δ[v] ← min(δ[u], c(e) − f(e))
9:    sinon
10:      si il existe e = (u, v) avec v ∈ Y, u ∉ Y, f(e) > 0 alors
11:         Y ← Y ∪ {u}
12:         A[u] ← e
13:         δ[u] ← min(δ[v], f(e))
14:      sinon
15:         Max ← vrai
16:      fin si
17:   fin si
18: fin tant que
```

4 Plus petit cycle contenant un sommet donné

Dans cet exercice, les graphes sont **simples**, **connexes**, **non-orientés** et chaque cycle considéré est supposé *élémentaire* (sans répétition de sommet ou d'arête).

Tous les parcours respecteront l'**ordre lexicographique** sur les sommets.

Notons qu'un graphe peut contenir plusieurs cycles et que chaque sommet peut être contenu dans plusieurs cycles.

1. Donner un exemple de graphe connexe dans lequel il existe à la fois
   (a) au moins un sommet contenu dans aucun cycle (nommer ce sommet A) et
   (b) au moins un sommet contenu dans un et un seul cycle (nommer ce sommet B) et
   (c) au moins un sommet contenu dans plusieurs cycles (nommer ce sommet C).
2. Expliquer brièvement comment détecter lors d'un parcours en largeur l'existence d'un cycle. L'algorithme est rappelé ci-dessous.

```
0   PL(G,s)
1     pour chaque sommet u de X[G] \ {s} faire
2         couleur[u] <- BLANC
3         d[u] <- infini
4         pere[u] <- nil
5     couleur[s] <- GRIS
6     d[s] <- 0
7     pere[s] <- nil
8     Enfiler(F, s)
9     tant que non vide(F) faire
10        u <- tete(F)
11        pour chaque v de Adj(u) faire
12            si couleur[v] = BLANC
13                alors couleur[v] <- GRIS
14                      d[v] <- d[u] + 1
15                      pere[v] <- u
16                      Enfiler(F, v)
17        Defiler(F)
18        couleur[u] <- NOIR
```

3. Le premier cycle détecté par un parcours en largeur contient-il forcément le sommet de départ ? Si oui, justifier. Sinon, donner un contre-exemple.

4. Donner un exemple de graphe G connexe qui satisfasse simultanément les deux conditions ci-dessous :

   (a) chaque sommet est contenu dans au moins un cycle, et

   (b) il est possible de choisir dans G un sommet de départ v tel que le premier cycle détecté par `PL(`G, v`)` ne contienne pas v.

   On précisera bien l'ordre sur les sommets.

5. Étant donné un graphe G et un sommet v de G, on se propose de modifier l'algorithme de parcours en largeur `PL(`G, v`)` de façon à calculer la longueur (c'est-à-dire le nombre d'arêtes) du cycle le plus court dans G contenant v, s'il en existe un ; si un tel cycle n'existe pas, l'algorithme retournera 0.

   Pour cela, lors de l'exécution de `PL(`G, v`)`, on calculera pour chaque sommet u de G une nouvelle étiquette `branche[`u`]`. Cette étiquette `branche[`u`]` est définie comme suit :

$$\texttt{branche[}u\texttt{]} = \begin{cases} 0 & \text{si} \quad u = v, \\ u & \text{si} \quad \texttt{pere[}u\texttt{]} = v, \\ \texttt{branche[pere[}u\texttt{]]} & \text{sinon.} \end{cases}$$

   (a) En supposant que le sommet v est le sommet s_0, calculer `branche[`u`]` pour tous les sommets u du graphe `G2` de la FIGURE 5 :

   (b) Proposer une modification de l'algorithme de parcours en largeur qui, pour un graphe G et un sommet v de G, retourne la longueur (c'est-à-dire le nombre

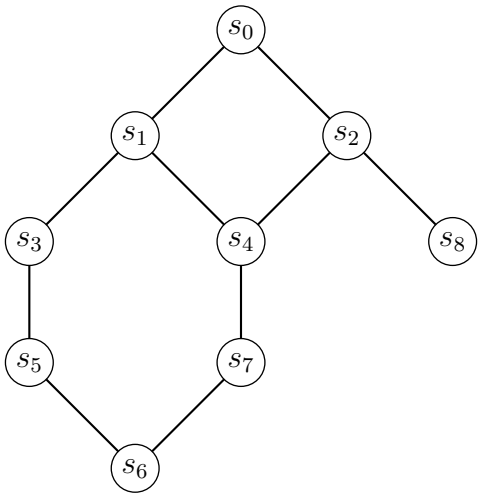

FIGURE 5 – Graphe `G2`

d'arêtes) du cycle le plus court dans G contenant v, s'il en existe un, et retourne 0 s'il n'en existe pas.

Remarque : vous noterez bien que cet algorithme ne détermine pas la longueur du plus court cycle de G mais la longueur du plus court cycle de G passant par v, le sommet de départ de $PL(G, v)$.