université de BORDEAUX

Année 2021-2022 — Session de décembre 2021
Master MIAGe : 1ère année — UE : 4TYE705U
Épreuve de : Recherche Opérationnelle et Conduite de Projets
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.

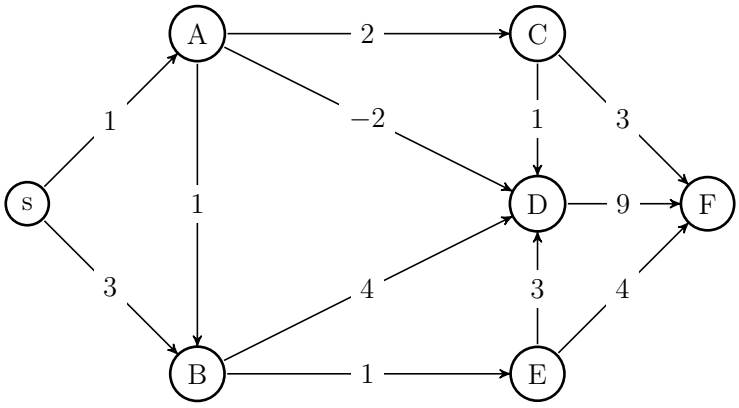

Figure 1 – Graphe $G1$

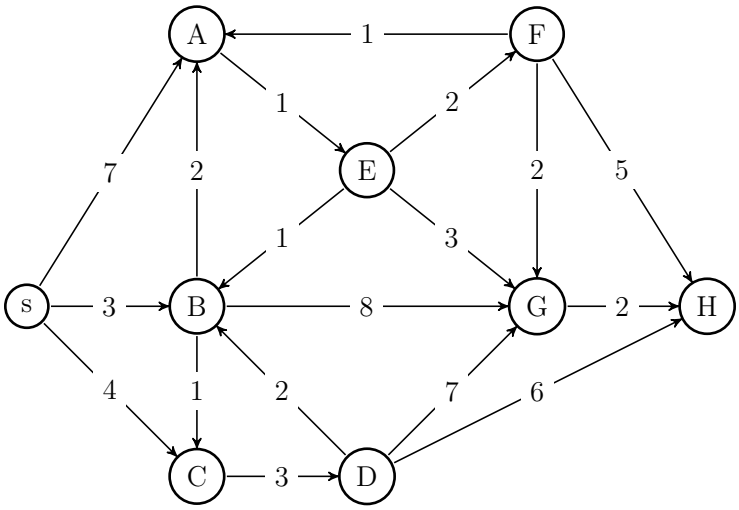

FIGURE 2 – Graphe $G2$

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.

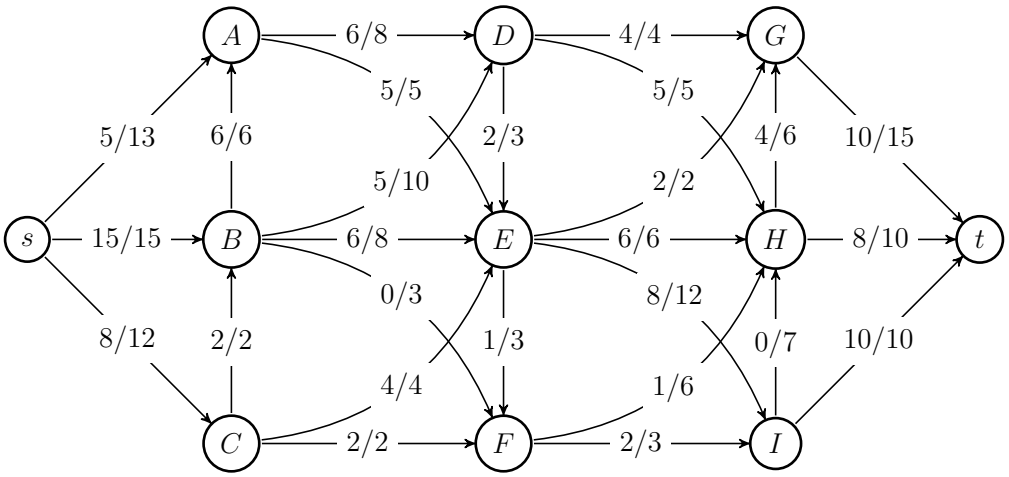

FIGURE 3 – Réseau

3 Composantes fortement connexes

3.1) En utilisant l'algorithme de calcul des composantes fortement connexes (Algorithme 5) basé sur le parcours en profondeur (Algorithmes 6 et 7), donnez les composantes fortement connexes du graphe de la Figure 4. 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 des indices et que le premier sommet visité lors du premier parcours en profondeur est le sommet s_0. 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 $\forall u, v \in V(G), (u, v) \in E(G^{-1}) \iff (v, u) \in E(G)$.

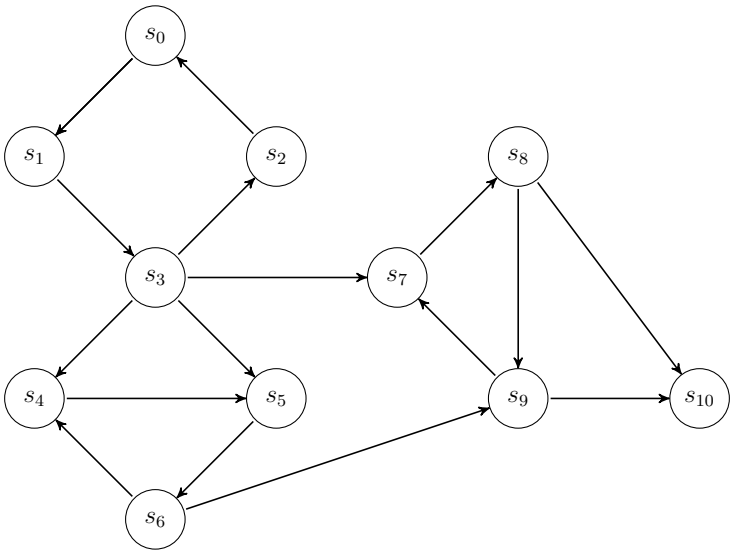

FIGURE 4 – Graphe

4 Arbre de poids maximum

On considère le problème suivant :

un réseau routier reliant un ensemble de villes est représenté par un graphe G simple, non orienté, chaque ville étant représentée par un sommet et deux sommets a et b étant reliés par une arête s'il existe une route directe reliant la ville a à la ville b.

Chaque arête e est munie d'un poids $h(e)$ donnant la hauteur maximum (en cm) d'un véhicule pouvant emprunter cette route. On souhaite calculer les itinéraires entre les villes maximisant la hauteur possible d'un véhicule.

On considère T_{max} l'arbre couvrant de G de poids maximum.

4.1) Montrer que si une arête $e = \{u, v\}$ n'appartient pas à T_{max}, et si C_{uv} désigne la chaîne reliant u et v dans T_{max}, alors $\forall e' \in C_{uv}, h(e) \leq h(e')$.

4.2) Que pouvez-vous en conclure si les poids sont tous différents ?

4.3) Que faut-il modifier à l'algorithme de Prim, rappelé à la fin du document (Algorithme 8), pour calculer un arbre de poids maximum ?

4.4) Donner l'ensemble des routes à emprunter dans le réseau routier de la Figure 5 pour maximiser la hauteur des camions utilisables pour tout transport entre deux villes.

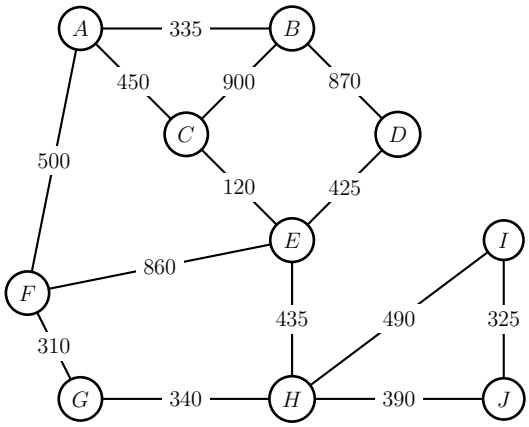

FIGURE 5 – Réseau routier

Algorithmes

Algorithme 1 Dijkstra(G, w, s)

pour tout v de $V(G)$ **faire**
  $d(v) \leftarrow \infty$
  $pere(v) \leftarrow NIL$
  $couleur(v) \leftarrow BLANC$
fin pour
$d(s) \leftarrow 0$
$F \leftarrow FILE\_PRIORITE(\{s\}, d)$
tant que $F \neq \emptyset$ **faire**
  $pivot \leftarrow EXTRAIRE\_MIN(F)$
  pour tout $e = (pivot, v)$ arc sortant de $pivot$ **faire**
    si $couleur(v) = BLANC$ **alors**
      si $d(v) = \infty$ **alors**
        INSERER(F, v)
      fin si
      si $d[v] > d[pivot] + w(e)$ **alors**
        $d[v] \leftarrow d[pivot] + w(e)$
        $pere[v] \leftarrow pivot$
      fin si
    fin si
  fin pour
  $couleur[pivot] \leftarrow NOIR$
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 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 6 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 7 VisiterPP(v)

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

Algorithme 8 Prim(G, w)

```
1: F ← FILE_PRIORITE(V(G), cle)
2: pour tout v de V(G) faire
3:    cle(v) ← ∞
4: fin pour
5: cle(r) ← 0
6: pere(r) ← NIL
7: tant que F ≠ ∅ faire
8:    u ← EXTRAIRE_MIN(F)
9:    pour tout v ∈ Adj(u) faire
10:      si v ∈ F et w(u, v) < cle(v) alors
11:         pere(v) ← u
12:         cle(v) ← w(u, v)
13:      fin si
14:   fin pour
15: fin tant que
```