université de BORDEAUX

Année 2021-2022 — Session de décembre 2021
Étape : INF501-MI501 — UE : 4TIN501U
Épreuve de : Algorithmique des graphes
Date : 13 décembre 2021 – 9h — **Durée :** 1h30
Épreuve de : O. Baudon — **Nb. pages : 9**

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 ou arcs d'un graphe G. Les variables n et m désigneront respectivement le nombre de sommets et d'arêtes.

1 Bellman

On rappelle que lors d'un parcours en profondeur sur un graphe orienté sans circuit, un tri topologique des sommets peut être obtenu en triant les sommets dans l'ordre inverse des heures de fin de visite.

1.1) Appliquer le parcours en profondeur au graphe de la Figure 1 (voir Algorithmes 1 et 2).

1.2) Qu'est-ce qui permet lors d'un parcours en profondeur de vérifier si un graphe orienté possède ou non un circuit ? Que pouvez-vous en conclure pour le graphe de la Figure 1 ? S'il existe, donner l'ordre obtenu par le tri topologique.

1.3) Appliquer l'algorithme de Bellman, rappelé à la fin du document (Algorithme 3) au graphe G_1 de la Figure 1.

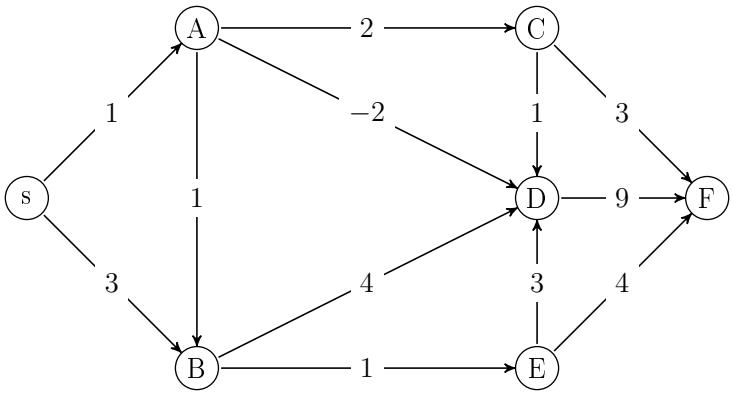


Figure 1 – Graphe G_1

2 Flot Maximum

2.1) 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 2 en améliorant le flot existant. Le flot existant est donné 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 obtenue lors de l'exécution de l'algorithme de Ford-Fulkerson et redessiner le graphe avec le flot maximum.

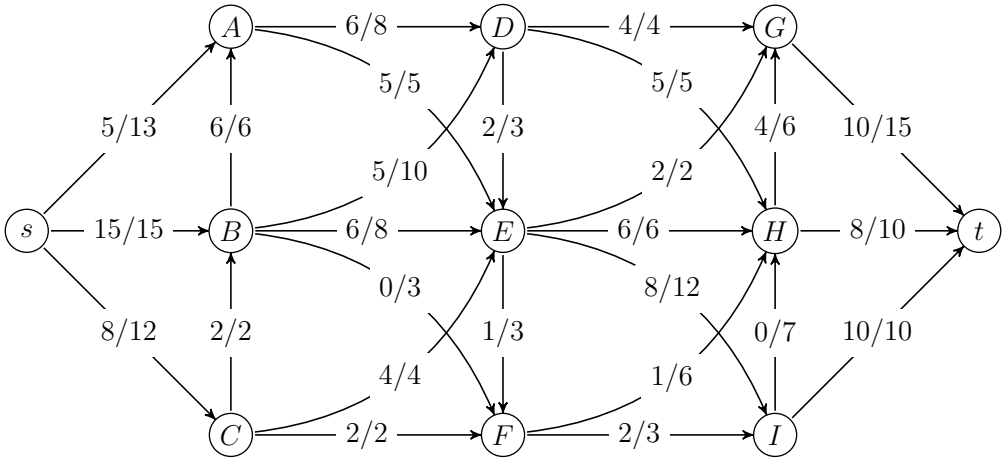


FIGURE 2 – Réseau

3 Arbre de poids minimum

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

3.1) 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.

3.2) Que faut-il modifier à l'algorithme de Kruskal (Algorithme 6), pour calculer un graphe partiel connexe de poids total minimum quand tous les poids ne sont pas positifs ?

3.3) Appliquer votre algorithme au graphe G_2 de la Figure 3. 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.

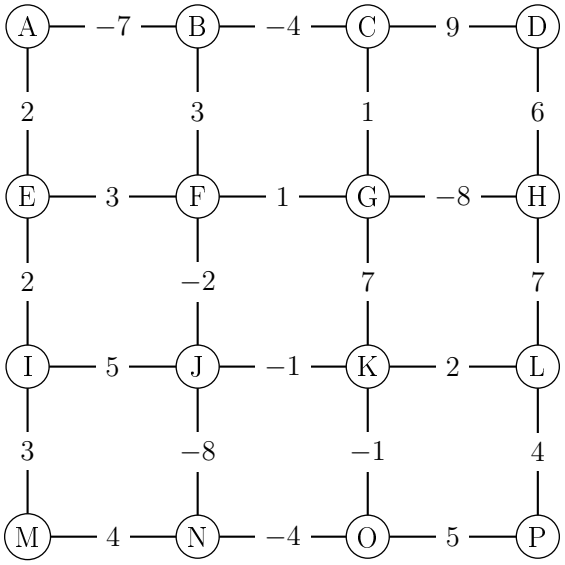


FIGURE 3 – Graphe G_2

4 Utilisation de Dijkstra dans un graphe avec un arc de poids négatif

Soit G un graphe orienté et valué par une fonction w ayant un et un seul arc de poids négatif. On note α l'origine et β l'extrémité de cet arc, *i.e.* $w(\alpha, \beta) < 0$.

Dans la suite on notera $d_H(u, v)$ la distance d'un plus court chemin du sommet u vers le sommet v dans un graphe H.

4.1) En utilisant l'algorithme de *Dijkstra* (Algorithme 7), trouver une façon de détecter l'existence d'un circuit dans G dont la somme des poids des arcs est négative (un tel circuit sera appelé un **circuit absorbant**).

Dans la suite on suppose que le graphe G ne possède pas de circuit absorbant, mais qu'il possède un arc (α, β) tel que $w(\alpha, \beta) < 0$. Pour chaque sommet de G, on veut savoir si un plus court chemin depuis un sommet distingué s doit emprunter ou non l'arc de poids négatif. Pour ce faire, on procède en deux étapes :
Soit G' le graphe obtenu en supprimant de G l'arc de poids négatif.

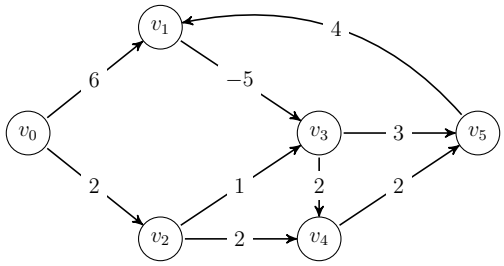
i. On calcule les distances $d_{G'}(s, v)$ entre s et tous les sommets v de G' en utilisant l'algorithme de *Dijkstra*

ii. On calcule les distances $d_{G'}(\beta, v)$ dans G' entre le sommet β et tous les sommets v de G'.

4.2) Comment utiliser les distances ainsi calculées pour savoir si le plus court chemin de s à un sommet v quelconque est l'un de ceux qui contiennent l'arc de poids négatif ?

4.3) Appliquer votre algorithme au graphe G ci-dessous en utilisant le sommet $s = v_0$ comme sommet distingué. Pour les exécutions de l'algorithme de Dijkstra, on précisera l'ordre des sommets pivots, et pour chaque sommet pivot les modifications des valeurs de $d[v]$ et on dessinera l'arborescence des plus courts chemins obtenus.

Finalement, pour chaque sommet, indiquer la longueur d'un plus court chemin depuis v_0 et si ce chemin utilise ou non l'arc (v_1, v_3) de poids négatif.



Algorithmes

Algorithme 1 Parcours en profondeur PP(G)

```
1: pour tout v ∈ V(G) faire
2:     couleur(v) ← BLANC
3:     pere(v) ← NIL
4: fin pour
5: temps ← 0
6: pour tout v ∈ V(G) faire
7:     si couleur(v) = BLANC alors
8:         VisiterPP(v)
9:     fin si
10: fin pour
```

Algorithme 2 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 3 Bellman(G, w, s)

```
1: TriTopologique(G)
2: Soit v_1, ..., v_n l'ordre topologique calculé à l'étape 1
3: pour i de 1 à n faire
4:    d[v_i] ← ∞
5:    pere[v_i] ← NIL
6: fin pour
7: d[s] ← 0
8: Soit j l'indice de s dans l'ordre topologique calculé à l'étape 1
9: pour i de j à n − 1 faire
10:   pour u ∈ Adj(v_i) faire
11:      si d[u] > d[v_i] + w(v_i, u) alors
12:         d[u] ← d[v_i] + w[v_i, u]
13:         pere[u] ← v_i
14:      fin si
15:   fin pour
16: fin pour
```

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)

$Y \leftarrow \{s\}$
$\delta(s) \leftarrow +\infty$
$Max \leftarrow$ faux
tant que $t \notin Y et Max =$faux **faire**
  si il existe $e = (u, v)$ avec $u \in Y, v \notin Y, f(e) < c(e)$ **alors**
    $Y \leftarrow Y \cup \{v\}$
    $A[v] \leftarrow e$
    $\delta[v] \leftarrow min(\delta[u], c(e) - f(e))$
  sinon
    si il existe $e = (u, v)$ avec $v \in Y, u \notin Y, f(e) > 0$ **alors**
      $Y \leftarrow Y \cup \{u\}$
      $A[u] \leftarrow e$
      $\delta[u] \leftarrow min(\delta[v], f(e))$
    sinon
      Max $\leftarrow vrai$
    fin si
  fin si
fin tant que

Algorithme 6 Kruskal(G, w)

pour tout v de $V(G)$ **faire**
  $composante(v) \leftarrow \{v\}$
fin pour
$ncomposantes \leftarrow n$
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 $ncomposantes > 1$ **faire**
  si $e_i = (u, v)$ et $composante(u) \neq composante(v)$ **alors**
    $E(T) \leftarrow E(T) \cup \{e_i\}$
    UNIFIER$(composante(u), composante(v))$
    $ncomposantes \leftarrow ncomposantes - 1$
  fin si
  $i \leftarrow i + 1$
fin tant que
retourner $E(T)$

Algorithme 7 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
```