université de BORDEAUX

Année 2020-2021 — Session de décembre 2020
Master MIAGe : 1ère année — UE : 4TYE705U
Épreuve de : Recherche Opérationnelle et Conduite de Projets
Durée : 1h30
Épreuve de : O. Baudon — **Nb. pages :**
Collège Sciences et technologies

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.

1 Composantes fortement connexes

1.1) En utilisant l'algorithme de calcul des composantes fortement connexes (Algorithme 1) basé sur le parcours en profondeur (Algorithmes 2 et 3), 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 $\forall u, v \in V(G), (u, v) \in E(G^{-1}) \iff (v, u) \in E(G)$.

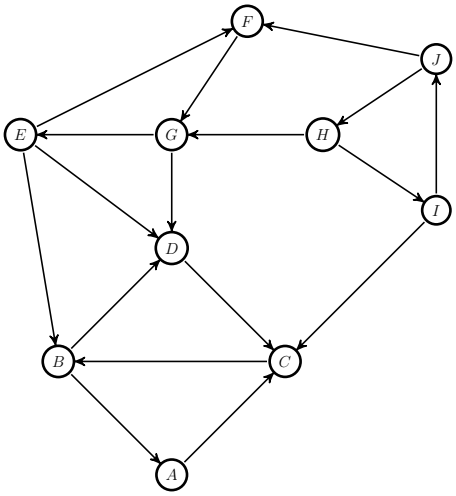

Figure 1 – Graphe G

Algorithme 1 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 2 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 3 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: $f(v) \leftarrow temps \leftarrow temps + 1$

Le parcours en profondeur de G est représenté par la Figure 2. Les heures de début et fin de visite sont données par le tableau ci-dessous :

v	A	B	C	D	E	F	G	H	I	J
$d(v)$	1	3	2	4	9	10	11	15	16	17
$f(v)$	8	6	7	5	14	13	12	20	19	18

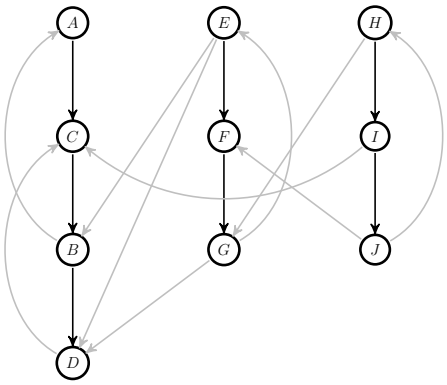

FIGURE 2 – PP(G)

Pour le parcours de G^{-1}, on doit donc considérer les sommets dans la boucle principale de PP(G^{-1}) (ligne 6) dans l'ordre $H, I, J, E, F, G, A, C, B, D$.
Cela donne les valeurs de début et fin de visite du tableau ci-dessous, l'arborescence étant représentée par la Figure 3.

v	A	B	C	D	E	F	G	H	I	J
$d(v)$	13	14	15	16	7	9	8	1	2	3
$f(v)$	20	19	18	17	12	10	11	6	5	4

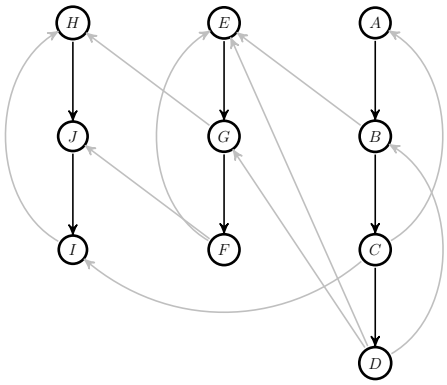

FIGURE 3 – PP(G^{-1})

Les composantes fortement connexes de G sont donc $\{A, B, C, D\}$, $\{E, F, G\}$ et $\{H, I, J\}$.

2 Algorithme de Prim

2.1) En utilisant l'algorithme de Prim (Algorithme 4), trouver l'arbre de poids minimum dans le graphe de la Figure 4, en partant du sommet A. Donner pour chaque étape le nom du sommet extrait de la file de priorité et les changements de valeurs pour la fonction *clé*. Dessiner l'arbre de poids minimum obtenu.

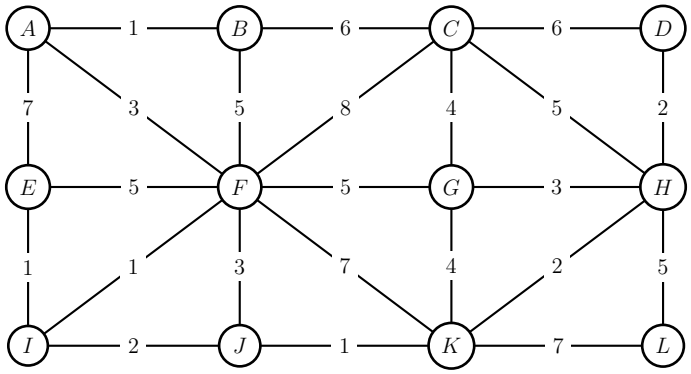

FIGURE 4 – Graphe G

Algorithme 4 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
```

Le tableau suivant donne les différentes valeurs de u et pour chacune de ces valeurs, les modifications de la clé. Une croix signifie que le sommet a été rajouté à l'arbre et donc que sa clé ne sera plus modifée. Le père d'un sommet v sera le dernier sommet u à avoir modifié sa clé. L'arbre de poids minimum obtenu est donné par la Figure 5.

u	A	B	C	D	E	F	G	H	I	J	K	L
	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
A	X	1			7	3						
B		X	6									
F					5	X	5		1	3	7	
I					1				X	2		
E					X							
J										X	1	
K							4	2			X	7
H			5	2			3	X				5
D				X								
G			4				X					
C			X									
L												X

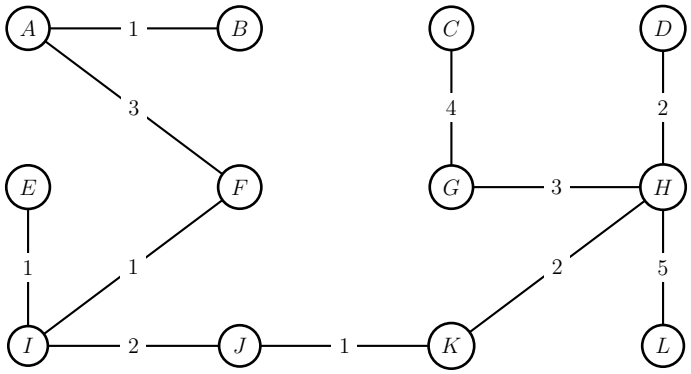

FIGURE 5 – Graphe G

3 Flot Maximum

3.1) En utilisant l'algorithme de Ford-Fulkerson (Algorithmes 5 et 6), trouver le flot maximum entre les sommets s et t du réseau de la Figure 6. 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.

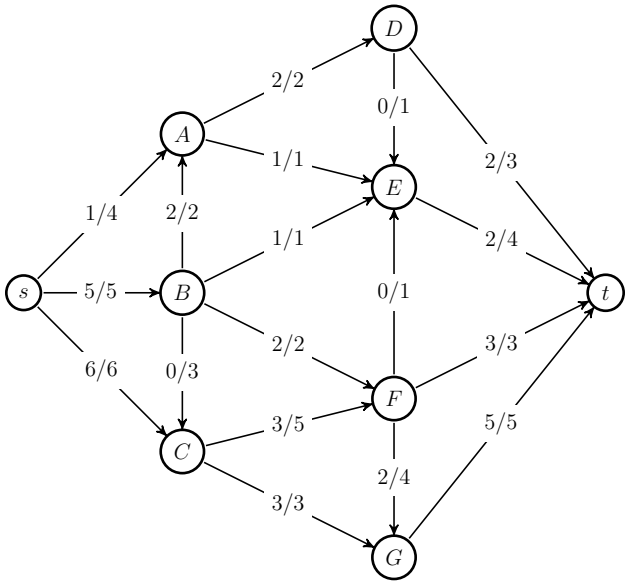

FIGURE 6 – Réseau

L'unique chaîne augmentante est : sABCFEt pour une augmentation de +1.
La valeur du flot maximum est donc $f(sA) + f(sB) + f(sC) = 2 + 5 + 6 = 13$. Lors de la deuxième exécution de Marquage, on reste bloqué avec pour valeur de Y : {s, A, B, C, F, G}. Ceci nous donne une coupe minimum de valeur : $c(AD) + c(AE) + c(BE) + c(FE) + c(Ft) + c(Gt) = 2 + 1 + 1 + 1 + 3 + 5 = 13$. Donc $val(Y) = val(f)$ et donc le flot est bien maximum. Il est donné par la Figure 7.

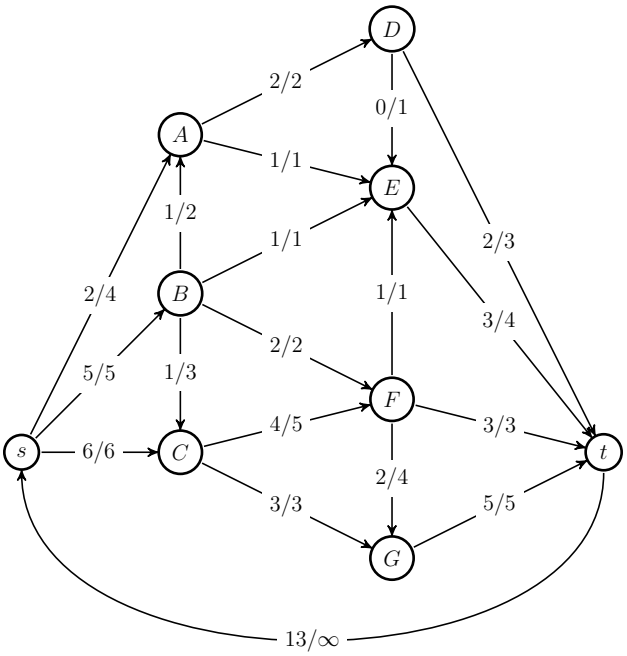

FIGURE 7 – Flot maximum

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 5 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 6 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

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 $d_s(G)$ les distances des plus courts chemins issu du sommet s vers tous les autres sommets de G. On notera $d_s(G, u)$ la distance d'un plus court chemin issu du sommet s vers le sommet u dans G.

4.1) En utilisant l'algorithme de *Dijkstra*, rappelé ci-dessous, 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**).

On calcule avec *Dijkstra* la distance de β à α dans le graphe G' obtenu à partir de G en supprimant l'arc (α, β). G possède un circuit absorbant si et seulement si $d_\beta(G', \alpha) < |w(\alpha, \beta)|$.

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 l'arc de poids négatif. Pour ce faire, on procède en deux étapes :

i. On applique l'algorithme de *Dijkstra* au graphe G' obtenu en supprimant de G l'arc de poids négatif et on calcule la distance $d_s(G')$ entre s et tous les sommets de G'.

ii. Soit G" le graphe ayant comme sommets ceux qui dans G' sont accessibles depuis β et tous les arcs de G les reliant excepté l'arc (α,β) de poids négatif. On calcule, en appliquant l'algorithme de *Dijkstra*, la distance $d_\beta(G")$ entre β et tous les sommets de $G"$.

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

Le chemin de s à u utilisera l'arc entre α et β si et seulement si $d_s(G', u) > d_s(G', \alpha) + w(\alpha, \beta) + d_\beta(G", u)$.

4.3) Appliquer votre algorithme au graphe G ci-dessous en utilisant le sommet $s = v_0$ comme sommet distingué.

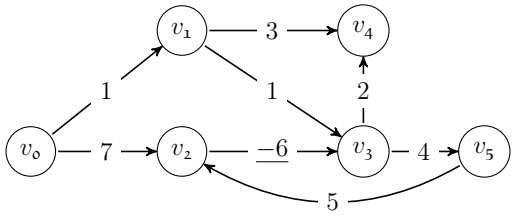

FIGURE 8 – Graphe G

```
0   Dijkstra(G,w,s)
1     pour chaque sommet v de V(G) faire
2         d[v] <- infini
3         pere[v] <- nil
4         couleur[v] <- BLANC
5     d[s] <- 0
6     F <- FILE_PRIORITE(V(G), d)
7     tant que F non vide faire
8         pivot <- EXTRAIRE_MIN(F)
9         couleur[pivot] <- GRIS
10        pour tout arc e = (pivot, v) arc sortant de pivot faire
11            si couleur(v) = BLANC et d[v] > d[pivot] + w(e)
12                alors d[v]  <- d[pivot] + w(e)
13                      pere[v]  <- pivot
14        couleur[pivot] <- NOIR
```

On calcule tout d'abord les distances de v_0 dans G'. Le résultat est donné par le tableau ci-dessous, l'arborescence des plus courts chemins dans G' étant donnée par la Figure 9.

pivot	v_0	v_1	v_2	v_3	v_4	v_5
	0	∞	∞	∞	∞	∞
v_0	X	1	7			
v_1		X		2	4	
v_3				X		6
v_4					X	
v_5						X
v_2			X			

Puis on calcule les plus courts chemins dans G' à partir de v_3, qui sont données par le tableau ci-dessous :

pivot	v_0	v_1	v_2	v_3	v_4	v_5
	∞	∞	∞	0	∞	∞
v_3				X	2	4
v_4					X	
v_5			9			X
v_2			X			

Enfin, on compare pour chaque sommet u les valeurs $d_{v_0}(G', u)$ et $d_{v_0}(G', v_2) + w(v_2, v_3) + d_{v_3}(G", u)$, soit $d_{v_0}(G', u)$ et $d_{v_3}(G", u) + 1$ et on garde le minimum. Ceux pour qui la deuxième valeur est le minimum utiliseront l'arc (v_2, v_3). Ce qui nous donnera l'arborescence des plus courts chemins représentée dans la Figure 10.

u	$d_{v_0}(G', u)$	$d_{v_3}(G", u) + 1$	$d_{v_0}(G, u)$
v_0	0	∞	0
v_1	1	∞	1
v_2	7	10	7
v_3	2	1	1
v_4	4	3	3
v_5	6	5	5

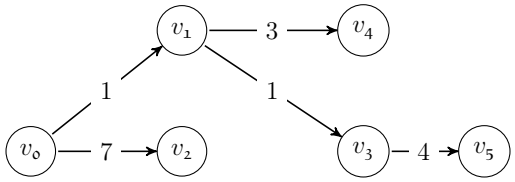

FIGURE 9 – Plus courts chemins dans G'

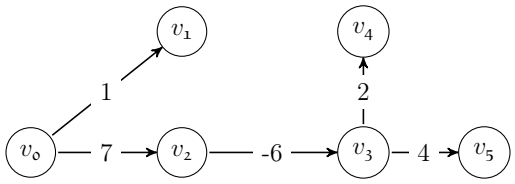

FIGURE 10 – Plus courts chemins dans G