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 7) basé sur le parcours en profondeur (Algorithmes 8 et 9), 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 $\forall u, v \in V(G), (u, v) \in E(G^{-1}) \iff (v, u) \in E(G)$.

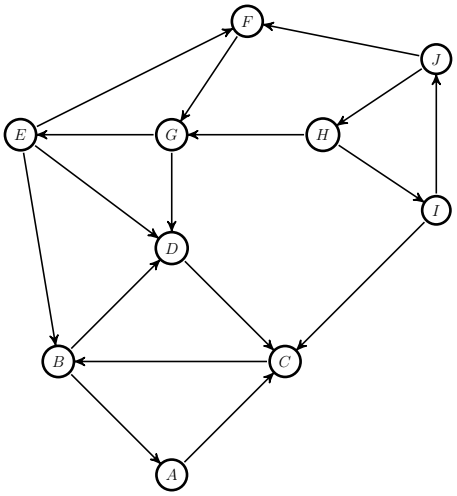

Figure 1 – Graphe G

Pour le parcours en profondeur du graphe de la Figure 1, les valeurs des tableaux d et f (début et fin de visite), et du tableau *pere* sont données par le tableau suivant :

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
$pere(v)$	NIL	C	A	B	NIL	E	F	NIL	H	I

Les arborescences sont représentées par la Figure 2.
L'ordre inverse de la fin de visite est donc $H, I, J, E, F, G, A, C, B, D$.
Le parcours en profondeur du graphe G^{-1} respectant cet ordre dans la boucle principale de $PP()$ donne comme résultat :

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

Les arborescences sont représentées par la Figure 3.
Les composantes fortement connexes obtenues sont donc $\{A, B, C, D\}$, $\{E, F, G\}$, $\{H, I, J\}$.

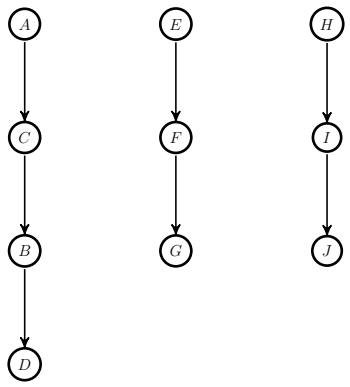

FIGURE 2 – PP(G)

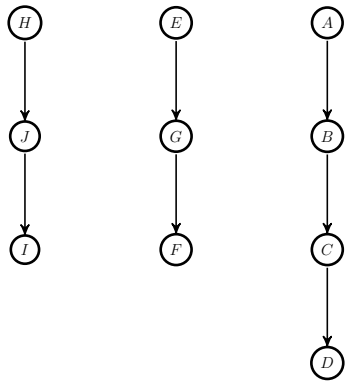

FIGURE 3 – PP(G^{-1})

2 Plus courts chemins

Les algorithmes de Dijkstra, Bellman et Ford sont rappelés à la fin du document : algorithmes 2, 3 et 4. Pour les trois graphes $G1$, $G2$ et $G3$ des Figures 4, 5 et 6, 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.

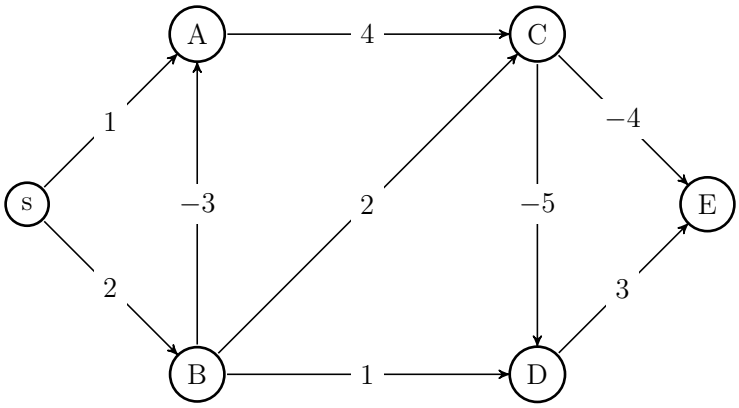

FIGURE 4 – Graphe $G1$

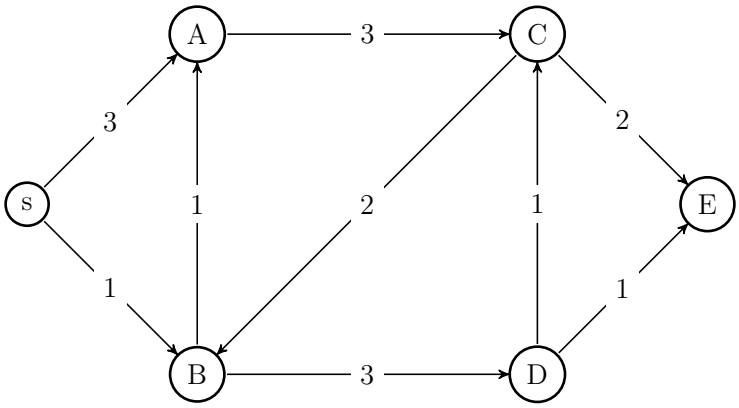

FIGURE 5 – Graphe $G2$

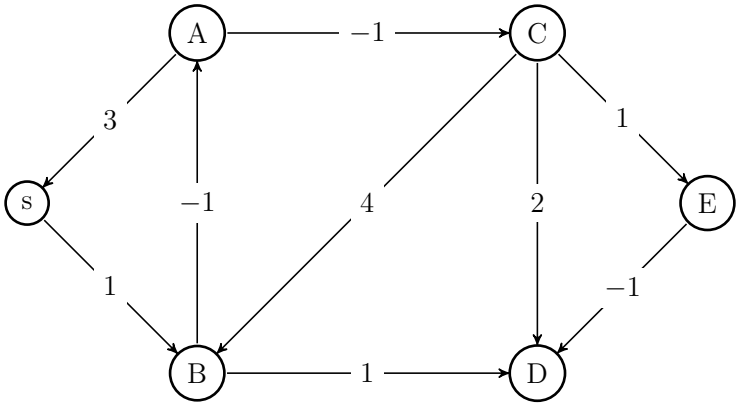

FIGURE 6 – Graphe $G3$

Pour le graphe $G1$, on peut utiliser Bellman car il ne contient pas de circuit. Pour le graphe $G2$, on peut utiliser Dijkstra car il ne contient pas d'arc de poids négatif. Pour le graphe $G3$, il est nécessaire d'utiliser l'algorithme de Ford car il possède des arcs de poids négatif et des circuits, par exemple $sBAs$ ou encore $ACBA$.

L'application de Bellman au graphe $G1$ est donnée par le tableau ci-dessous. La première ligne correspond aux initialisations de d et $npred$. La tête de file u considérée à chaque étape est donnée par la première colonne, avec sa distance avec s. Les modifications de d et $npred$ sont données sur chaque ligne pour chaque sommet v dans la colonne correspondante, séparées par un "/" :

u	s	A	B	C	D	E
	$0/0$	$\infty/2$	$\infty/1$	$\infty/2$	$\infty/2$	$\infty/2$
$s/0$	X	$1/1$	$2/0$			
$B/2$		$-1/0$	X	$4/1$	$3/1$	
$A/-1$		X		$3/0$		
$C/3$				X	$-2/0$	$-1/1$
$D/-2$					X	$-1/0$
$E/-1$						X

On peut donc constater que le graphe ne contient pas de circuit. L'arborescence des plus courts chemins est donnée par la Figure 7.

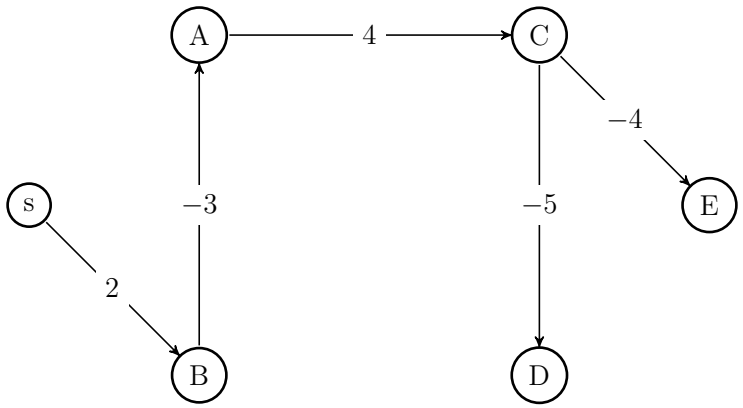

FIGURE 7 – Plus courts chemins dans $G1$

L'application de Dijkstra au graphe $G2$ est donnée par le tableau ci-dessous. La première ligne correspond aux initialisations de d. Le pivot considéré à chaque étape est donnée par la première colonne, avec sa distance avec s. Les modifications de d sont données sur chaque ligne pour chaque sommet v dans la colonne correspondante :

pivot	s	A	B	C	D	E
	0	∞	∞	∞	∞	∞
$s/0$	X	3	1			
$B/1$		2	X		4	
$A/2$		X		5		
$D/4$					X	5
$C/5$				X		
$E/5$						X

L'arborescence des plus courts chemins est donnée par la Figure 8.

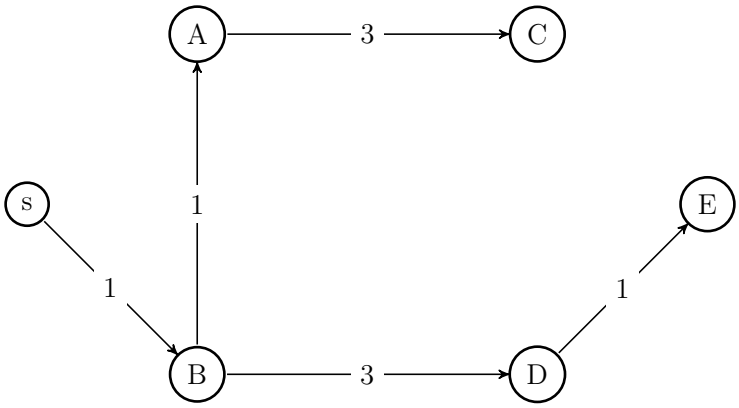

FIGURE 8 – Plus courts chemins dans $G2$

L'application de Ford au graphe $G3$ est donnée par le tableau ci-dessous. La première ligne correspond aux initialisations de d. La première colonne donne la valeur de i, suivie des arcs considérés. Les modifications de d sont données sur chaque ligne pour chaque sommet v dans la colonne correspondante :

	s	A	B	C	D	E
	0	∞	∞	∞	∞	∞
$i=1$						
sB			1			
As						
AC						
BA		0				
BD					2	
CB						
CD						
CE						
ED						
$i=2$						
sB						
As						
AC				-1		
BA						
BD						
CB						
CD					1	
CE						0
ED					-1	
$i=3$						
sB						
As						
AC						
BA						
BD						
CB						
CD						
CE						
ED						

Comme pour $i = 3$, il n'y a aucune modification, on peut en conclure que l'algorithme se terminera sans autre modification et retournera VRAI. L'arborescence des plus courts chemins est donnée par la Figure 9.

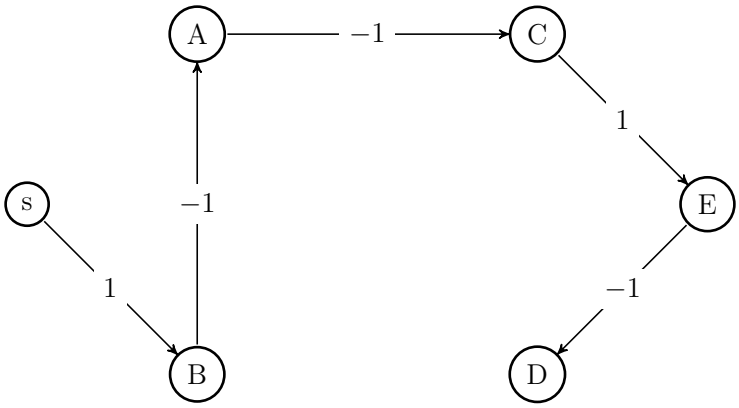

FIGURE 9 – Plus courts chemins dans $G3$

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 10. 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.

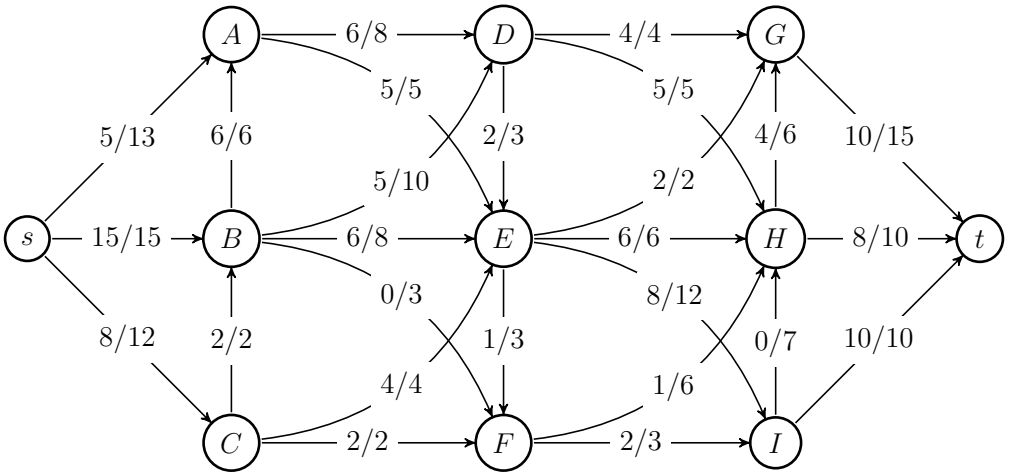

FIGURE 10 – Réseau

Les chaînes augmentantes sont les suivantes :

1. s, A, D, E, I, H, t avec une augmentation de $+1$
2. s, A, B, E, I, H, t avec une augmentation de $+1$
3. $s, A, D, B, E, I, H, G, t$ avec une augmentation de $+1$
4. s, A, B, F, H, G, t avec une augmentation de $+1$

ce qui donne un flot d'une valeur totale de 32 représenté par la Figure 11.

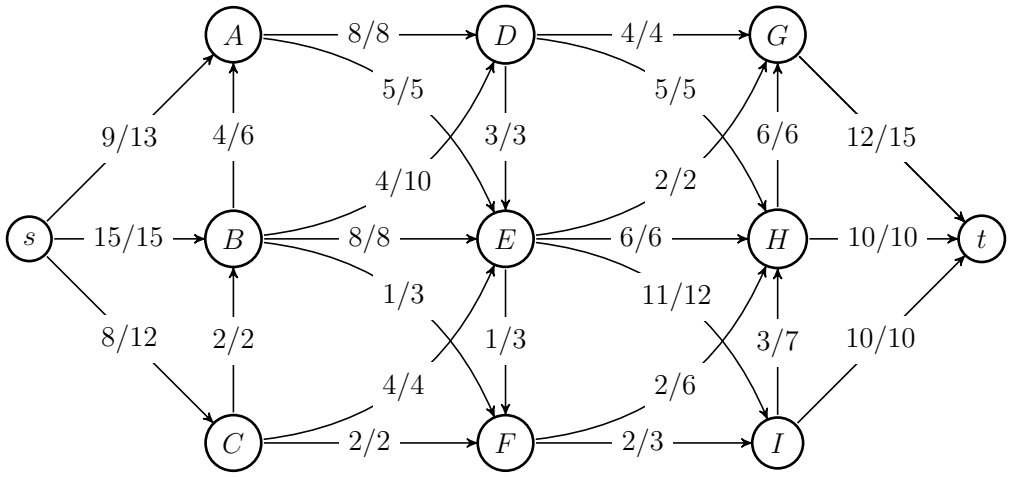

FIGURE 11 – Flot maximum

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

La coupe minimum est Y = {s, A, B, C, D,E, F, H, I} de valeur $c(DG) + c(EG) + c(HG) + c(Ht) + c(It) = 4 + 2 + 6 + 10 + 10 = 32$, ce qui correspond bien à la valeur du flot maximum trouvé.

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.

Si on prend un cycle à trois sommets dont toutes les arêtes ont un poids de -1, la solution optimale consiste à garder toutes les arêtes, ce qui donne un poids total de -3, alors que

tout arbre couvrant contiendra 2 arêtes et aura donc un poids total de -2, ce qui est plus élevé.

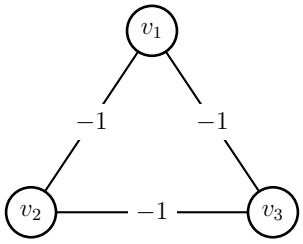

FIGURE 12 – Graphe ayant un graphe partiel connexe minimum qui n'est pas un arbre

4.2) Que faut-il modifier à l'algorithme de Kruskal (Algorithme 10), pour calculer un graphe partiel connexe de poids total minimum quand tous les poids ne sont pas positifs ?
Il faut d'abord garder toutes les arêtes de poids négatif, puis considérer les arêtes de poids positifs dans l'ordre croissant des poids en gardant celles qui relient des sommets n'appartenant pas à une même composante connexe, et ce, jusqu'à ce que le graphe soit connexe.
Ce qui donne l'algorithme suivant, où la variable *ncc* contient le nombre de composantes connexes du graphe partiel en cours et E(H) les arêtes du graphe partiel connexe de poids minimum :

Algorithme 1 KruskalModifié(G, w)

1: $ncc \leftarrow 0$
2: **pour tout** v de $V(G)$ **faire**
3: $\quad composante(v) \leftarrow \{v\}$
4: $\quad ncc \leftarrow ncc + 1$
5: **fin pour**
6: Trier $E(G)$ dans un ordre croissant $(e_1, \ldots, e_m)$ en fonction de $w(e)$
7: $i \leftarrow 1$
8: $E(H) \leftarrow \{\}$
9: **tant que** $w(e_i) < 0$ **faire**
10: $\quad E(H) \leftarrow E(H) \cup \{e_i\}$
11: $\quad$ **si** $e_i = (u, v)$ et $composante(u) \neq composante(v)$ **alors**
12: $\quad\quad$ UNIFIER$(composante(u), composante(v))$
13: $\quad\quad ncc \leftarrow ncc - 1$
14: $\quad$ **fin si**
15: $\quad i \leftarrow i + 1$
16: **fin tant que**
17: **tant que** $ncc > 1$ **faire**
18: $\quad$ **si** $e_i = (u, v)$ et $composante(u) \neq composante(v)$ **alors**
19: $\quad\quad E(H) \leftarrow E(H) \cup \{e_i\}$
20: $\quad\quad$ UNIFIER$(composante(u), composante(v))$
21: $\quad\quad ncc \leftarrow ncc - 1$
22: $\quad$ **fin si**
23: $\quad i \leftarrow i + 1$
24: **fin tant que**
25: **retourner** $E(H)$

4.3) Appliquer votre algorithme au graphe G_2 de la Figure 13. 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.

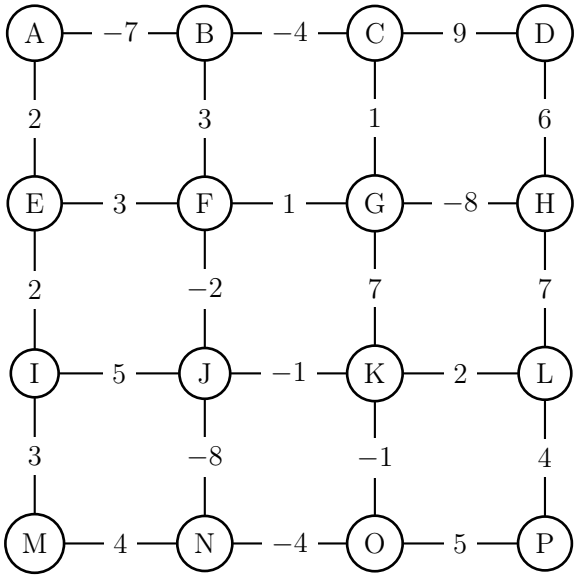

FIGURE 13 – Graphe G_2

Le graphe partiel de poids minimum est donné par la figure 14 ci-dessous. Les étiquettes sur les arêtes sont le poids de l'arête suivi de leur ordre d'insertion dans le graphe partiel.
L'ordre de rajout des arêtes est le suivant :
On rajoute tout d'abord les arêtes de poids négatifs dans l'ordre croissant de leur poids (ordre inverse de leur valeur absolue) : GH, JN, AB, BC, NO, FJ, JK, KO.
Puis on traite les arêtes de poids positif en les gardant si elles relient deux sommets n'étant pas dans les mêmes composantes connexes, qui sont alors unifiées. Sinon, on les jette : on garde CG, FG, AE, EI, KL, on jette BF, EF, on garde IM, LP, on jette MN, IJ, OP, on garde DH. Le graphe est maintenant connexe, donc on peut s'arrêter. Cela donne un poids total de -14.

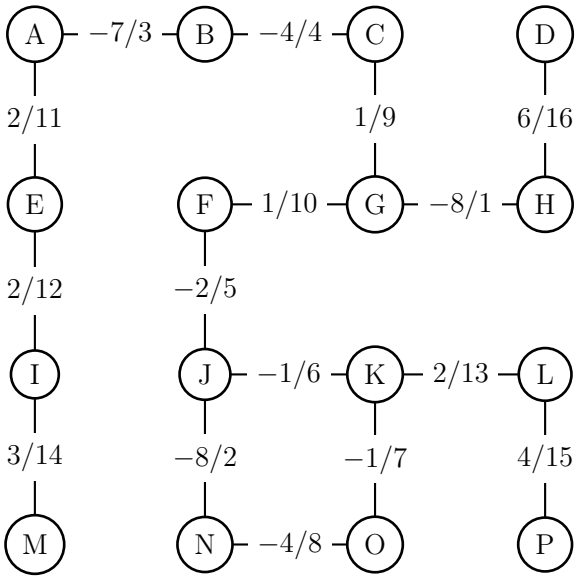

FIGURE 14 – Graphe G_2

Algorithmes

Dans les algorithmes de Dikstra (2), Bellman (3) et Ford (4), on a :
G : graphe orienté
$w : E(G) \to \mathbb{R}$
s : source de G

Algorithme 2 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 3 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 4 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 5 (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 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)

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 7 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 8 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 9 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 (10), 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 10 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 $E(T)$