université de BORDEAUX

Année 2024-2025 — Session de décembre 2024
Master MIAGe : 1ère année — UE : 4TYE705U
Épreuve de : Recherche Opérationnelle et Conduite de Projets
Durée : 1h30
Épreuve de : O. Baudon et C. Favre-Bertin
Collège Sciences et technologies

Aucun document autorisé.
Une attention toute particulière sera portée à la présentation et à la rédaction de la copie.

Partie : Recherche Opérationnelle

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 Algorithme de Prim

1.1) En utilisant l'algorithme de Prim rappelé à la fin du document (Algorithme 3), trouver l'arbre de poids minimum dans le graphe de la Figure 1, en partant du sommet A. On donnera pour chaque étape le nom du sommet extrait de la file de priorité et les changements de valeur pour la fonction *clé*.

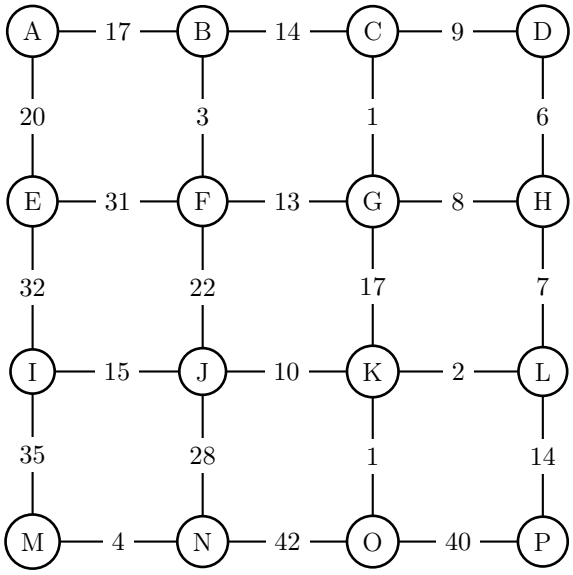

Figure 1 – Graphe

Le tableau ci-dessous donne la liste des sommets extraits de la file (sommet u) avec les modifications de la fonction *clé* qu'entraîne leur ajout à l'arbre. Au départ, toutes les clés sont à l'infini, sauf celle de A qui est à 0.

u	
A	$cle(B) = 17, cle(E) = 20$
B	$cle(C) = 14, cle(F) = 3$
F	$cle(G) = 13, cle(J) = 22$
G	$cle(C) = 1, cle(H) = 8, cle(K) = 17$
C	$cle(D) = 9$
H	$cle(D) = 6, cle(L) = 7$
D	
L	$cle(K) = 2, cle(P) = 14$
K	$cle(J) = 10, cle(O) = 1$
O	$cle(N) = 42$
J	$cle(I) = 15, cle(N) = 28$
P	
I	
E	
N	$cle(M) = 4$
M	

L'arbre de poids minimum est donné par la Figure 2 et est de poids total 149.

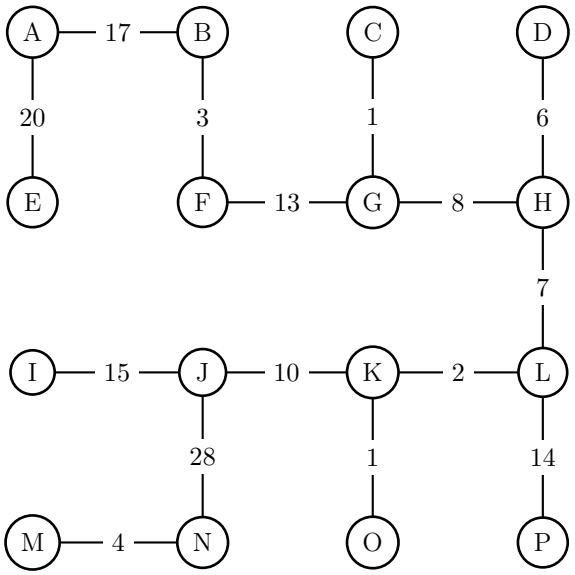

FIGURE 2 – Arbre de poids total minimum

2 Plus courts chemins

Les algorithmes de Dijkstra, Bellman et Ford sont rappelés à la fin du document : Algorithmes 4, 5 et 6. Pour les trois graphes $G1$, $G2$ et $G3$ des Figures 3, 4 et 5, 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.
En cas de choix possible, 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 si elles existent.

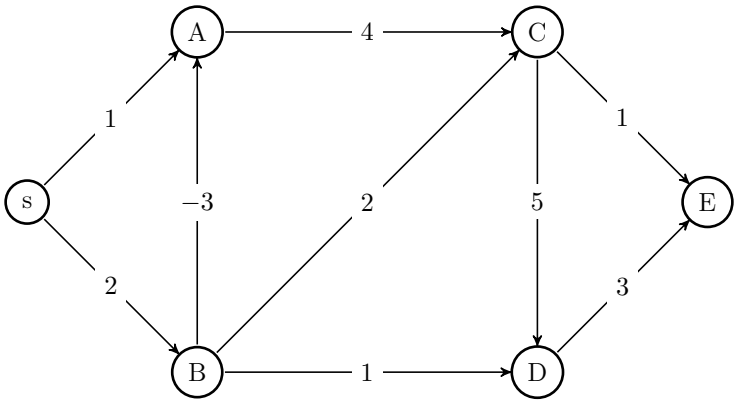

FIGURE 3 – Graphe $G1$

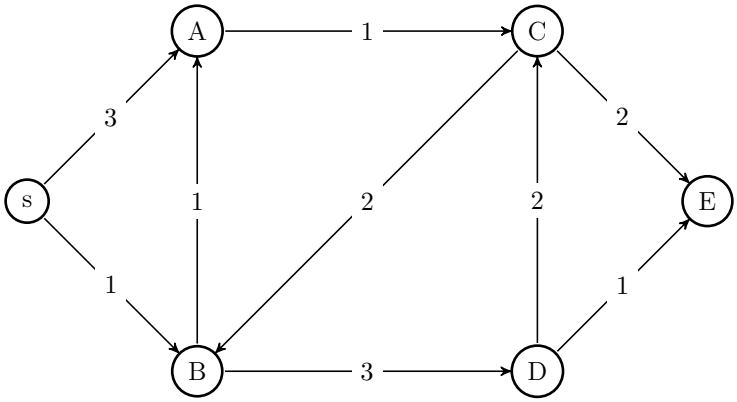

FIGURE 4 – Graphe $G2$

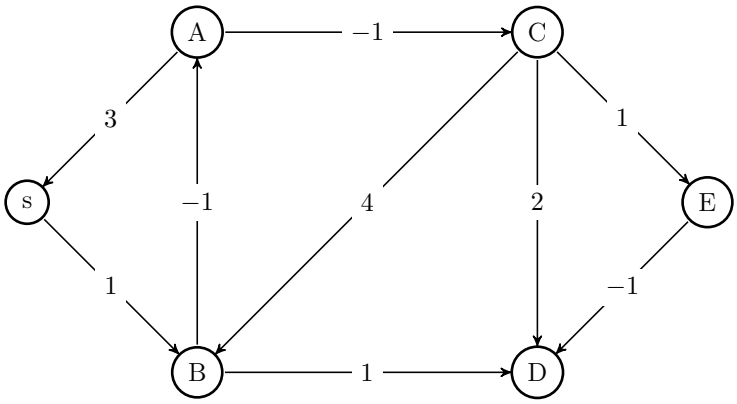

FIGURE 5 – Graphe $G3$

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	$3/0$	$4/1$
$D/3$					X	$4/0$
$E/4$						X

On peut donc constater que le graphe ne contient pas de circuit. L'arborescence des plus courts chemins est donnée par la Figure 6.

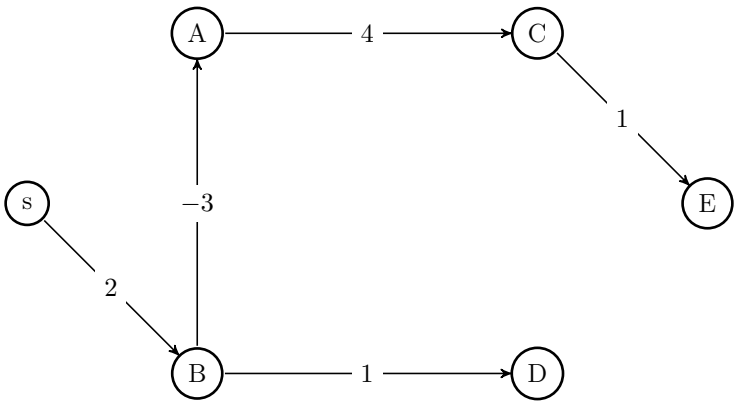

FIGURE 6 – Plus courts chemins dans $G1$

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		3		
$C/3$				X		5
$D/4$					X	
$E/5$						X

L'arborescence des plus courts chemins est donnée par la Figure 7.

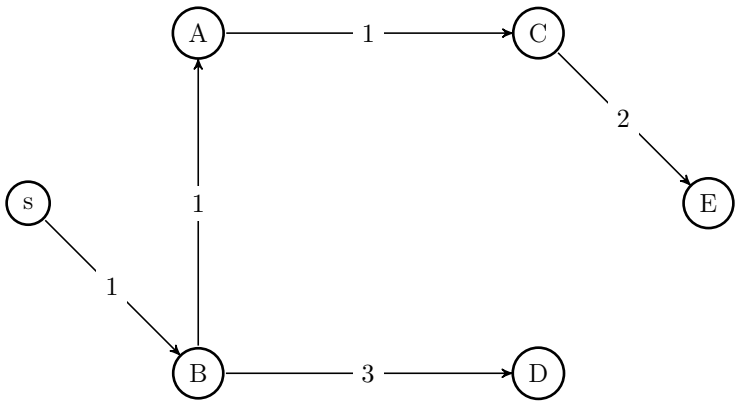

FIGURE 7 – Plus courts chemins dans $G2$

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 8.

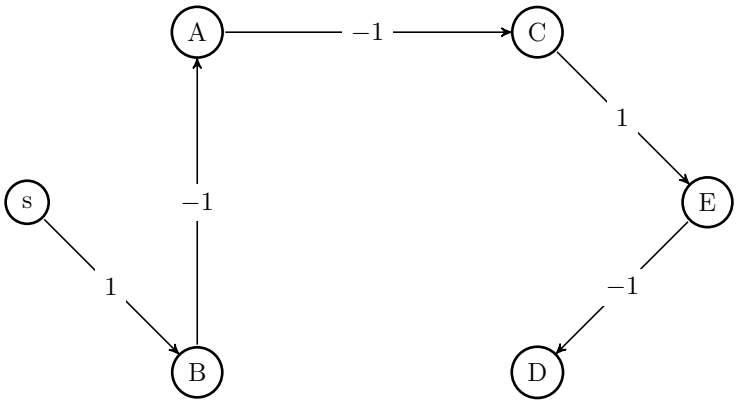

FIGURE 8 – Plus courts chemins dans $G3$

3 Flot Maximum

3.1) En utilisant l'algorithme de Ford-Fulkerson rappelé à la fin du document (Algorithmes 7 et 8), trouver le flot maximum entre les sommets s et t du réseau de la Figure 9. 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 8 et une capacité de 13. *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.

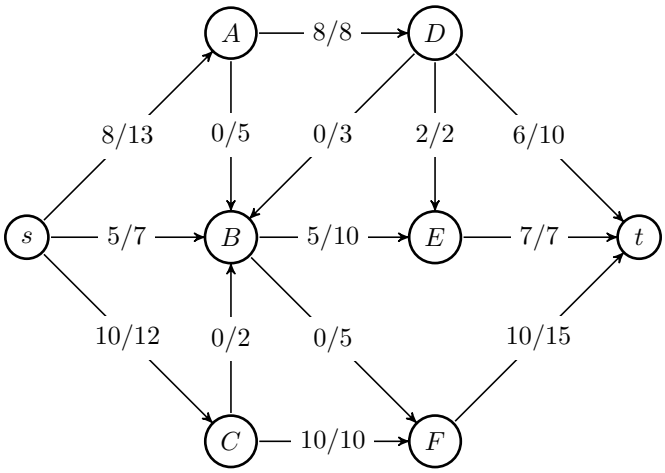

FIGURE 9 – Réseau

Les chaînes augmentantes sont les suivantes :

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

ce qui donne un flot d'une valeur totale de 30 représenté par la Figure 10.

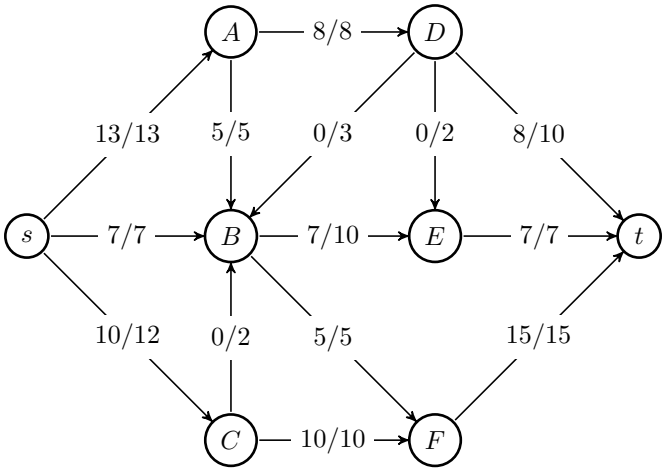

FIGURE 10 – Flot maximum

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

Lors de la dernière exécution de la procédure de marquage, les sommets marqués sont $Y = \{s, A, B, C, E\}$. La valeur de cette coupe est $c(AD) + c(BF) + c(CF) + c(Et) = 8 + 5 + 7 + 10 = 30$, ce qui est bien la valeur du flot maximum obtenu.

4 Recherche d'un chemin hamiltonien dans un tournoi

Un *tournoi* est un graphe simple complet orienté. Autrement dit, si T est un tournoi et u et v deux sommets de T, alors $(u, v) \in E(T) \Leftrightarrow (v, u) \notin E(T)$.

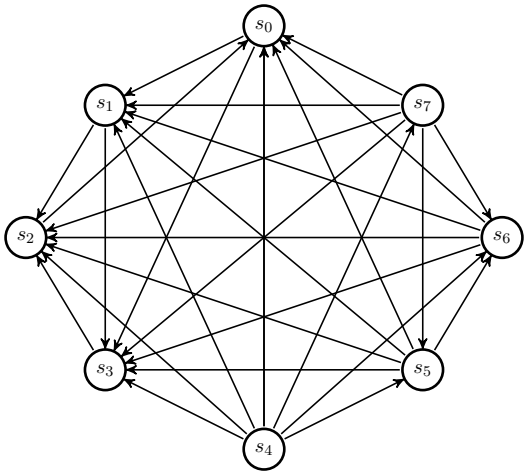

FIGURE 11 – Exemple de tournoi à 8 sommets

Un *chemin hamiltonien* est un chemin qui passe par tous les sommets du graphe une et une seule fois. Tout tournoi possède un chemin hamitonien. Par exemple, pour le tournoi de la Figure 11 : $s_4, s_7, s_5, s_6, s_0, s_1, s_3, s_2$.

4.1) Appliquer l'algorithme de parcours en profondeur rappelé à la fin du document (Algorithmes 9 et 10) au tournoi T de la Figure 11. On conviendra que dans les listes d'adjacence, les sommets sont rangés dans l'ordre lexicographique. Pour chaque sommet, donner les valeurs `d`, `f`, `pere` retournées par l'algorithme en respectant l'ordre de visite des sommets.

L'arbre obtenu en partant de s_0 et en considérant les voisins dans l'ordre lexicographique est donné par la Figure 12.

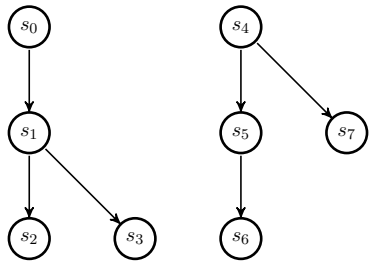

FIGURE 12 – Parcours en profondeur du tournoi de la Figure 11

Les valeurs des tableaux d, f, et père sont données ci-dessous :

	s_0	s_1	s_2	s_3	s_4	s_5	s_6	s_7
$pere$	NIL	s_0	s_1	s_1	NIL	s_4	s_5	s_4
d	1	2	3	5	9	10	11	14
f	8	7	4	6	16	13	12	15

Ordonner les sommets de T dans l'ordre inverse de leur fin de visite. Que remarquez-vous ?

L'ordre des sommets dans l'ordre inverse de leur fin de visite est : $s_4, s_7, s_5, s_6, s_0, s_1, s_3, s_2$, ce qui correspond au chemin hamiltonien donné en exemple.

4.2) Lorsque l'on range les sommets dans l'ordre inverse de leur fin de visite, quels sont les types d'arcs (parmi *liaison, retour, avant, transverse*) reliant deux sommets consécutifs ? Montrer que si les sommets sont rangés dans l'ordre inverse de leur fin de visite, alors si le sommet v suit immédiatement le sommet u dans cet ordre, l'arc entre ces deux sommets est orienté de u vers v.

Si u et v sont deux sommets consécutifs dans l'ordre inverse de l'heure de fin de visite, s'il existe un arc entre u et v, ce qui sera toujours le cas dans un tournoi, cet arc sera soit un arc de liaison, soit un arc transverse. En effet, si uv est un arc retour, alors $f[u] < f[v]$. Si uv est un arc avant, il y aura toujours des sommets intermédiaires dans l'arbre de parcours entre leur origine et leur destination. Donc leurs heures de fin de visite ne seront pas consécutives.

De plus, les arcs de liaisons et transverses sont toujours orientés du sommet ayant la plus grande heure de fin de visite vers celui ayant la plus petite. Donc si v suit u dans l'ordre inverse de fin de visite, l'arc entre u et v sera l'arc uv.

4.3) Modifier l'algorithme de parcours en profondeur pour afficher les sommets dans un ordre induisant un chemin hamiltonien. Quelle est la complexité de l'algorithme obtenu ?

Dans la partie $PP(G)$, on crée une pile vide P, par exemple après l'initialisation de la variable *temps*, puis on affiche cette pile après avoir terminé le parcours.
Dans $VisiterPP(v)$, on empile v dans P quand on met à jour son heure de fin de visite
Cela donne les algorithmes suivants PP' et $VisiterPP'$:

Algorithme 1 PP'(G)

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

Algorithme 2 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: $f(v) \leftarrow temps \leftarrow temps + 1$
11: $Empiler(P, v)$

La complexité du parcours en profondeur n'est pas modifiée et l'algorithme obtenu est donc en $O(n + m)$.

Algorithmes

Algorithme 3 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
```

Dans les algorithmes de Dikstra (4), Bellman (5) et Ford (6), on a :
G : graphe orienté
$w : E(G) \to \mathbb{R}$
s : source de G

Algorithme 4 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 5 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 6 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 7 (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 7 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 8 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 9 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 10 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: $f(v) \leftarrow temps \leftarrow temps + 1$