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 : 16**

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 2 et 3).

Le parcours est représenté par la Figure 2 dans laquelle les arcs de liaisons sont en noir et les autres arcs (ici uniquement avants et transverses) en gris. L'arc AD est avant, les arcs CD, CF, ED, et EF sont des arcs transverses.
Les heures de début et fin de visite sont données par le tableau ci-dessous :

sommet v	s	A	B	C	D	E	F
$d(v)$	1	2	3	11	4	8	5
$f(v)$	14	13	10	12	7	9	6

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.

Lors d'un parcours en profondeur, on peut dire qu'un graphe orienté possède un circuit si et seulement si il existe un arc retour.
Comme ce n'est pas le cas lors du parcours en profondeur du graphe G_1, celui-ci est sans circuit et on peut donc effectuer un tri topologique sur ses sommets, c'est à dire obtenir un ordre sur ses sommets $v_1, \ldots, v_n$ tel que si $v_i v_j \in E(G_1)$ alors $i < j$.
Cet ordre est obtenu en triant les sommets dans l'ordre inverse de leur fin de visite, ce qui nous donne ici : s, A, C, B, E, D, F

1.3) Appliquer l'algorithme de Bellman, rappelé à la fin du document (Algorithme 4) au graphe G_1 de la Figure 1.

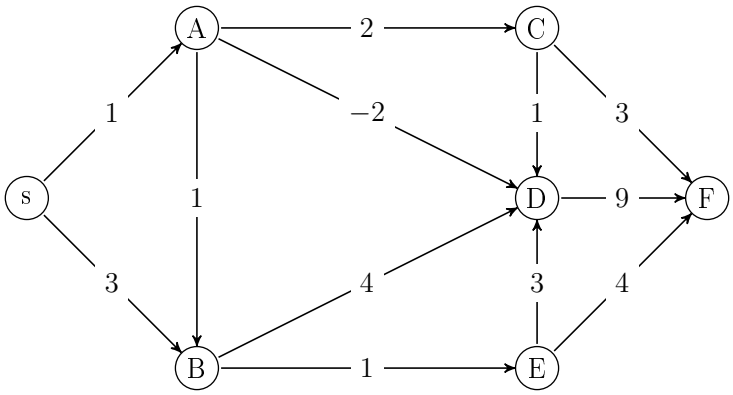


FIGURE 1 – Graphe G_1

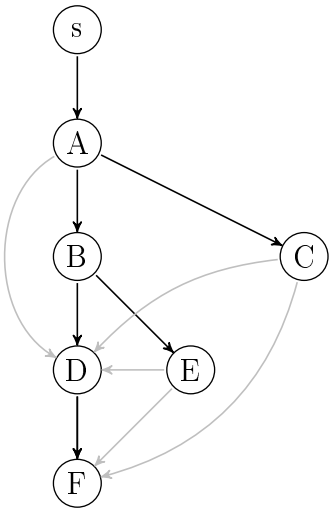


FIGURE 2 – Parcours en profondeur du graphe G_1

L'application de l'algorithme de Bellman au graphe G_1 est donné par le tableau ci-dessous. La première ligne contient les valeurs initiales de d. Les lignes suivantes correspondent aux exécutions de la boucle 9-15 et indiquent les changements du tableau d. Le sommet v_i et la valeur de $d[v_i]$ sont indiqués par la première colonne, séparé par un /.
L'arborescence des plus cours chemins est donnée par la Figure 3.

v_i	s	A	B	C	D	E	F
	0	∞	∞	∞	∞	∞	∞
$s/0$		1	3				
$A/1$			2	3	-1		
$C/3$							6
$B/2$						3	
$E/3$							
$D/-1$							
$F/6$							

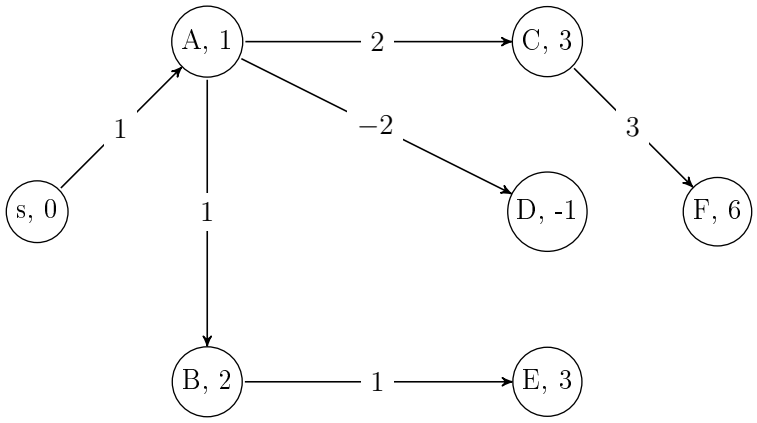


FIGURE 3 – Plus courts chemins dans le graphe G_1

2 Flot Maximum

2.1) 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 4 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.

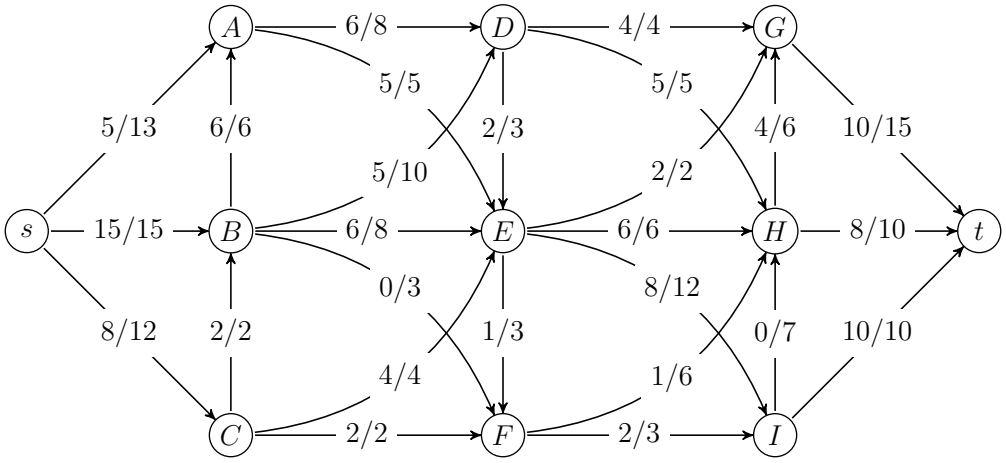


FIGURE 4 – Réseau

La première chaîne augmentante est $sADEIHt$ avec une augmentation du flot de 1.
La deuxième chaîne augmentante est $sABEIHt$ avec une augmentation du flot de 1.
La troisième chaîne augmentante est $sABFHGt$ avec une augmentation du flot de 2.
Lors de la recherche d'une nouvelle chaîne augmentante, on va marquer les sommets $s, A, B, C, D, E, F, I, H$ sans pouvoir atteindre t.
La coupe minimum obtenue par l'algorithme est donc $\{s, A, B, C, D, E, F, I, H\}$ de valeur $c(DG) + c(EG) + c(HG) + c(Ht) + c(It)$, soit $4 + 2 + 6 + 10 + 10 = 32$, ce qui correspond bien à la valeur du flot $f(sA) + f(sB) + f(sC) = 9 + 15 + 8 = 32$.
Le flot maximum est donné par la Figure 5.

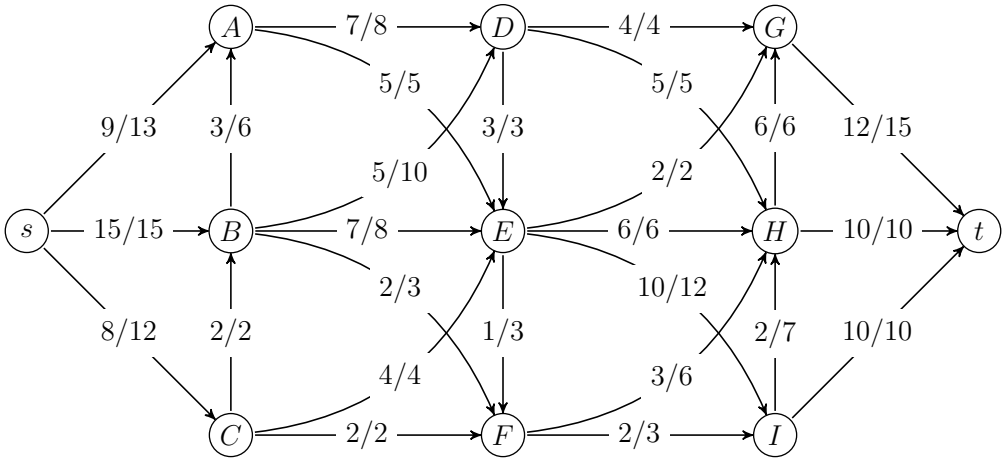


FIGURE 5 – Flot maximum

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.

Si on prend un graphe complet à 3 sommets, dans lequel toutes les arêtes ont un poids négatif (voir Figure 6), le graphe partiel connexe de poids total minimum contiendra toutes les arêtes car si toutes les arêtes sont de poids négatif, en retirer une fait augmenter le poids total. Et donc le graphe partiel connexe de poids total minimum contiendra un cycle.

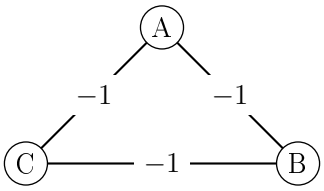


FIGURE 6 – Graphe partiel connexe de poids minimum avec cycle

3.2) Que faut-il modifier à l'algorithme de Kruskal (Algorithme 7), 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, en unifiant les composantes connexes de leurs extrémités. Ensuite, on reprend la version classique : tant que le nombre de composantes connexes est supérieur à 1, on rajoute les arêtes de poids positifs dans l'ordre croissant de leur poids si elles ne sont pas déjà entièrement incluse dans une même composante connexe.
Ce qui donne l'algorithme Kruskal-modifie suivant :

Algorithme 1 Kruskal-modifie(G, w)

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

3.3) Appliquer votre algorithme au graphe G_2 de la Figure 7. 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.

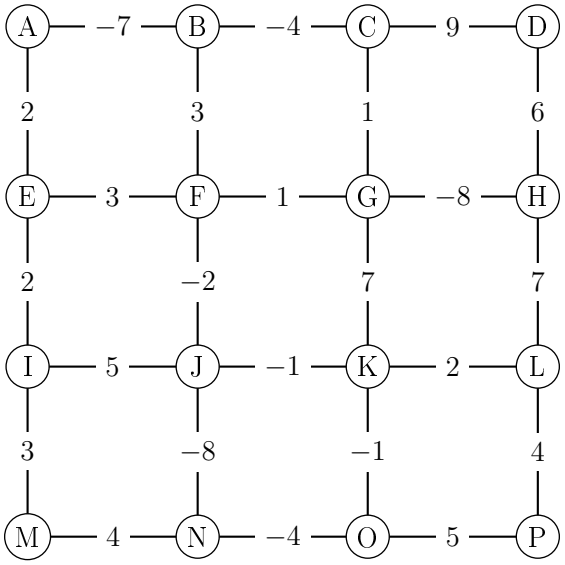


FIGURE 7 – Graphe G_2

On commence par garder toutes les arêtes négatives, soit (si on les considère par ordre croissant) : GH, JN, AB, BC, NO, FJ, JK, KO, puis on traite les arêtes de poids positif tant que le graphe partiel obtenu n'est pas connexe, ce qui nous fait rajouter CG, FG, KL, AE, EI, IM, LP et DH.
On notera que le résultat contient un cycle composé uniquement d'arêtes de poids négatif : JKONJ.
Le graphe partiel de poids minimum est représenté par la Figure 8, où les arêtes conservées sont en noir et celles supprimées en gris. Son poids total est de -14.

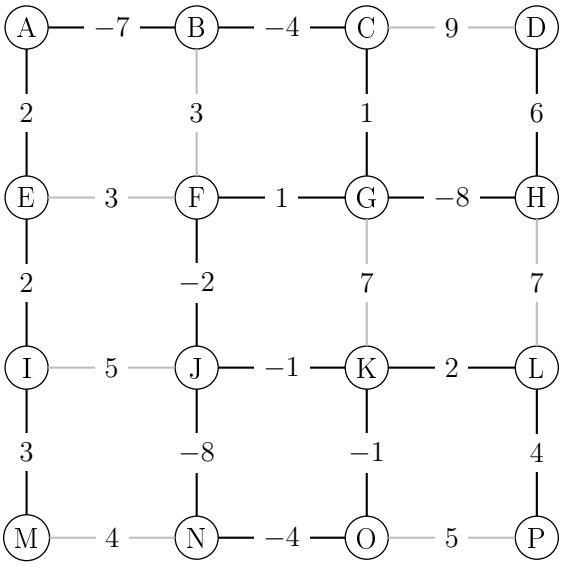


FIGURE 8 – Graphe G_2

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 8), 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**).

Il suffit (sans avoir à modifier l'algorithme) d'appliquer Dijkstra à partir de β après avoir supprimé l'arête $\alpha\beta$ pour calculer la distance entre β et α. Si cette distance est strictement inférieure à la valeur absolue de $w(\alpha, \beta)$, alors le graphe G possède un circuit de poids total négatif.

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'.

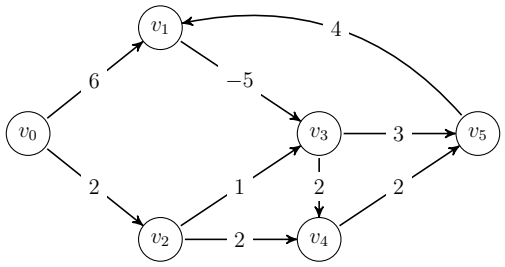
4.2) Comment utiliser les distances ainsi calculées pour savoir si le plus court chemin de s à un sommet v quelconque est l'un de ceux qui contiennent l'arc de poids négatif?

Il suffit de comparer les valeurs $d_{G'}(s, v)$ et $d_{G'}(s, \alpha) + w(\alpha, \beta) + d_{G'}(\beta, v)$:
L'arête $\alpha\beta$ devra être utilisée si

$$d_{G'}(s, v) > d_{G'}(s, \alpha) + w(\alpha, \beta) + d_{G'}(\beta, v)$$

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.



Exécution de Dijkstra(G', v_0) pour calculer $d_{G'}(v_0, v)$ pour tout sommet v :

pivot	v_0	v_1	v_2	v_3	v_4	v_5
	0	∞	∞	∞	∞	∞
v_0	X	6	2			
v_2			X	3	4	
v_3				X		6
v_4					X	
v_1		X				
v_5						X

L'arborescence des plus courts chemins est donnée par la Figure 9.

Exécution de Dijkstra(G', v_3) pour calculer $d_{G'}(v_3, v)$ pour tout sommet v (on notera que v_0 et v_2 ne peuvent être atteints depuis v_3 dans G') :

pivot	v_0	v_1	v_2	v_3	v_4	v_5
	∞	∞	∞	0	∞	∞
v_3				X	2	3
v_4					X	
v_5		7				X
v_1		X				

L'arborescence des plus courts chemins est donnée par la Figure 10.

Calcul des plus courts chemins dans G depuis v_0 (une croix dans la dernière colonne indique que le chemin de v_0 à v utilise l'arc de poids négatif entre v_1 et v_3 :

v	$d_{G'}(v_0, v)$	$d_{G'}(v_0, v_1) + w(v_1, v_3) + d_{G'}(v_3, v)$	$d_G(v_0, v)$	
v_0	0	∞	0	
v_1	6	8	6	
v_2	2	∞	2	
v_3	3	1	1	X
v_4	4	3	3	X
v_5	6	4	4	X

L'arborescence des plus courts chemins est donnée par la Figure 11.

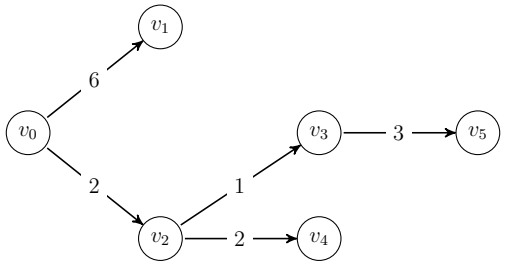


FIGURE 9 – Plus courts chemins dans G' depuis v_0

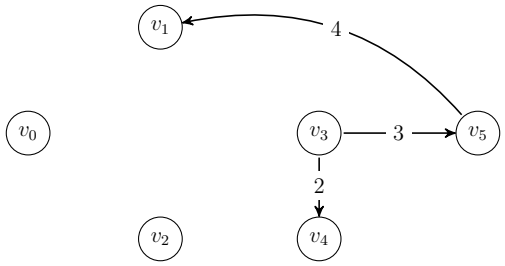


FIGURE 10 – Plus courts chemins dans G' depuis v_3

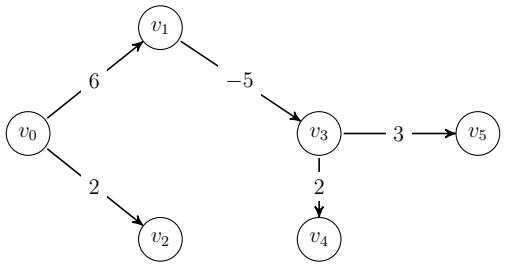


FIGURE 11 – Plus courts chemins dans G depuis v_0

Algorithmes

Algorithme 2 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 3 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 4 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 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)

$Y \leftarrow \{s\}$
$\delta(s) \leftarrow +\infty$
$Max \leftarrow$ 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 7 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 8 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
```