université de BORDEAUX

Année 2023-2024 — Session de décembre 2023
Master MIAGe : 1ère année — UE : 4TYE705U
Épreuve de : Recherche Opérationnelle et Conduite de Projets
Partie : Recherche Opérationnellle
Durée : 1h30
Épreuve de : O. Baudon — **Nb. pages : 9**
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 Plus courts chemins

Les algorithmes de Dijkstra et Bellman sont rappelés à la fin du document : Algorithmes 3 et 4. Pour les deux graphes $G1$ et $G2$ des Figures 1 et 2, calculer les plus courts chemins entre le sommet s et les autres sommets du graphe. Pour chacun des deux graphes, on rappellera les conditions nécessaires à l'utilisation de l'algorithme choisi. 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 $npred$.
Dans les deux cas, dessiner l'arborescence des plus courts chemins.

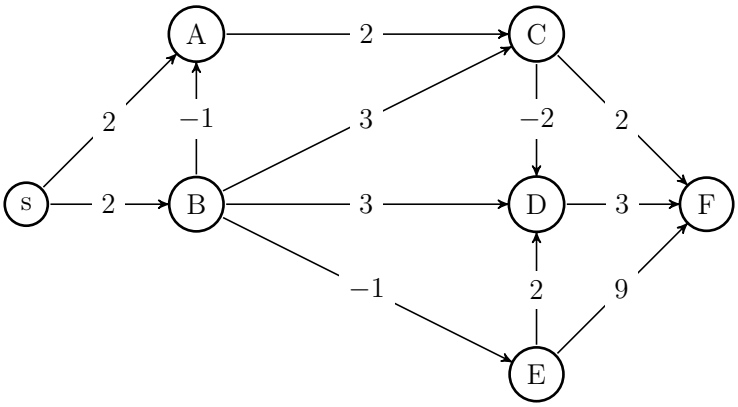

Figure 1 – Graphe $G1$

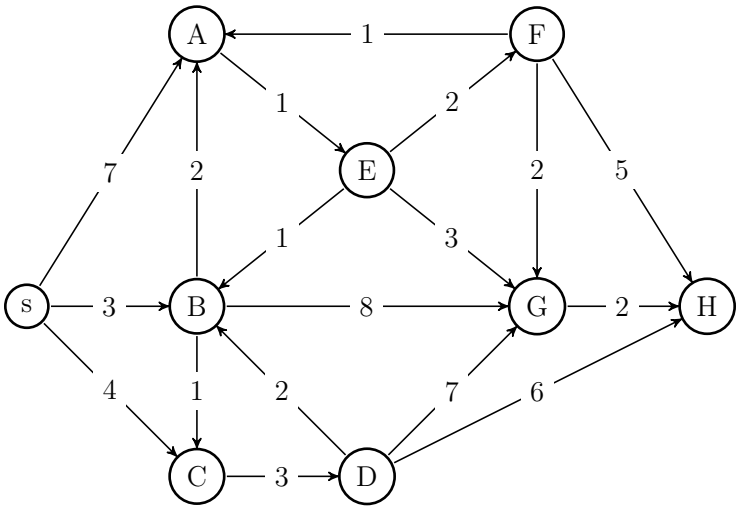

FIGURE 2 – Graphe $G2$

Dans le cas du graphe $G1$, il faut utiliser Bellman car le graphe possède des arcs de poids négatif (par exemple BA de poids -1), mais ne possède pas de circuit.
Dans le cas du graphe $G2$, il faut utiliser Dijkstra car le graphe possède des circuits (par exemple AEBA), mais ne possède pas d'arc de poids négatif.
Les tableaux montrant l'exécution des algorithmes respectivement sur $G1$ et $G2$ sont donnés ci-dessous, suivi de l'arborescence des plus courts chemins obtenue.

Application de Bellman sur $G1$

u	s	A	B	C	D	E	F	$File$
	$0/0$	$\infty/2$	$\infty/1$	$\infty/2$	$\infty/3$	$\infty/1$	$\infty/3$	s
$s/0$	X	$2/1$	$2/0$					B
$B/2$		$1/0$	X	$5/1$	$5/2$	$1/0$		A, E
$A/1$		X		$3/0$				E, C
$E/1$					$3/1$	X	$10/2$	C
$C/3$				X	$1/0$		$5/1$	D
$D/1$					X		$4/0$	F
$F/4$							X	$\emptyset$

Arborescence des plus courts chemins de $G1$ depuis s

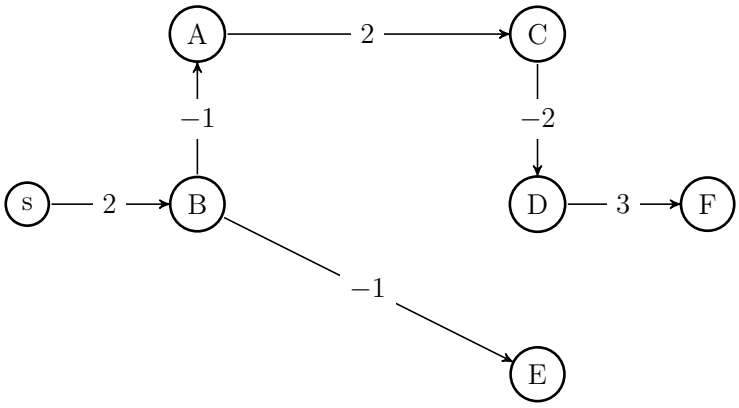

FIGURE 3 – Arborescence des plus courts chemins de $G1$ depuis s

Application de Dijkstra sur $G2$

pivot	s	A	B	C	D	E	F	G	H
	0	∞	∞	∞	∞	∞	∞	∞	∞
$s/0$	X	7	3	4					
$B/3$		5	X					11	
$C/4$				X	7				
$A/5$		X				6			
$E/6$						X	8	9	
$D/7$					X				13
$F/8$							X		
$G/9$								X	11
$H/11$									X

Arborescence des plus courts chemins de $G2$ depuis s

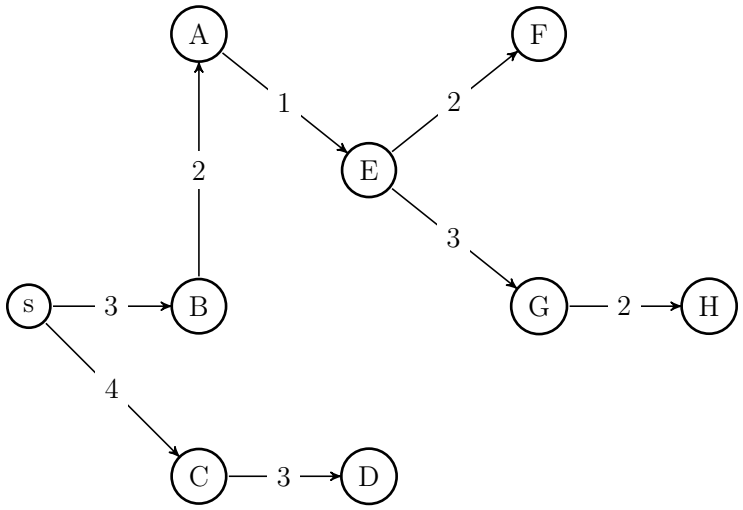

FIGURE 4 – Arborescence des plus courts chemins de $G2$ depuis s

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 5, *en partant du flot initial déjà indiqué*. Cette valeur initiale est 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 5 et une capacité de 13. On donnera pour chaque exécution de la procédure de marquage le chemin augmentant obtenu et l'augmentation correspondante.

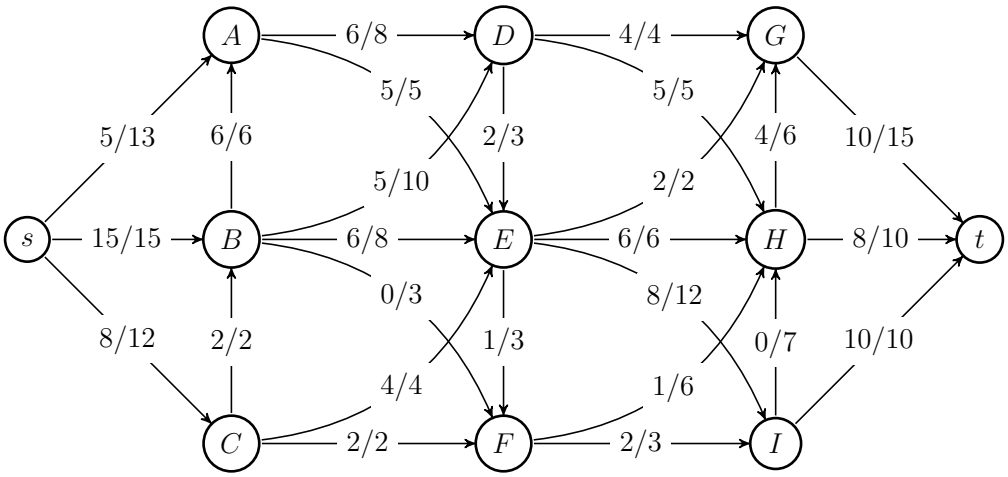

FIGURE 5 – Réseau

Les chemins augmentant sont les suivants :

— sABDEIHt, $\delta = +1$

— sABEIHt, $\delta = +1$

— sABEIHGt, $\delta = +1$

— sABFIHGt, $\delta = +1$

ce qui donne un flot maximim d'une valeur de 32.

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

Les sommets marqués lors de la dernière exécution de la procédure de marquage sont : $Y = \{s, A, B, C, D, E, F, H, I\}$.
La valeur de Y est $c(DG) + c(EG) + c(HG) + c(Ht) + c(It) = 4 + 2 + 6 + 10 + 10 = 32$.

Le flot maximum est donné sur le graphe ci-dessous :

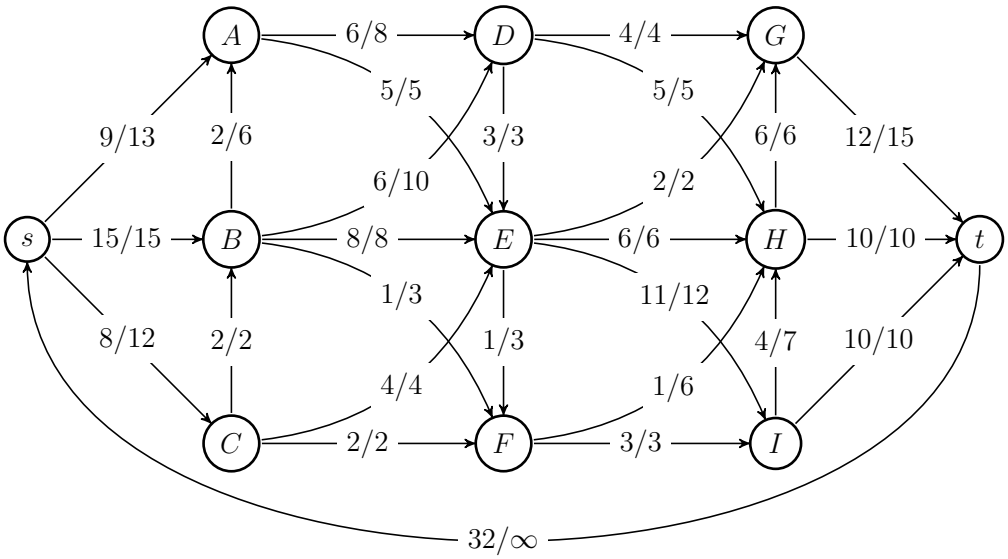

FIGURE 6 – Flot maximum

3 Plus courts circuits

Dans cet exercice, tous les circuits sont considérés élémentaires (sans répétition de sommet ni d'arc).

L'objectif de cet exercice est de concevoir un algorithme pour calculer la longueur minimum d'un circuit dans un graphe orienté donné, en se basant sur l'algorithme de parcours en largeur.

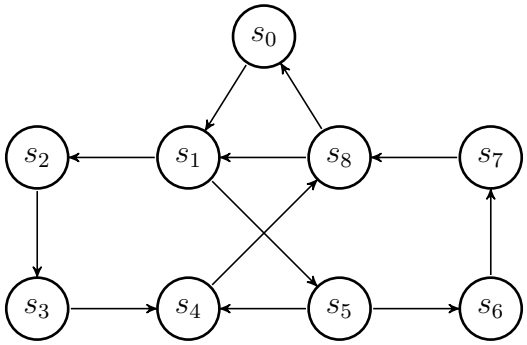

FIGURE 7 – Graphe orienté G_1.

1. Appliquer le parcours en largeur (Algorithme 7) au graphe G_1 de la Figure 8
   Parcours en largeur de G_1 :
   Les arcs n'appartenant pas à l'arbre de parcours sont dessinés en rouge.
   Les valeurs de d sont données dans le tableau sous le parcours.

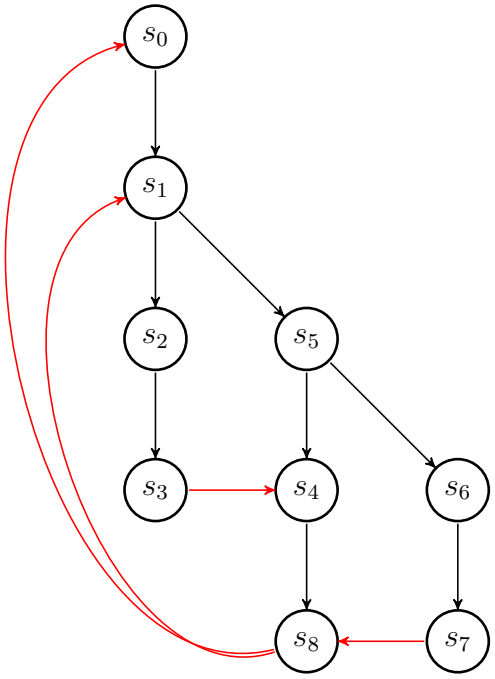

FIGURE 8 – Graphe orienté G_1.

   Valeurs de d

v	s_0	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8
$d(v)$	0	1	2	3	3	2	3	4	4

2. Identifier tous les circuits du graphe G_1 de la Figure 8 contenant le sommet s_0 (il y en a trois). Quelle est la longueur minimum d'un circuit de $G1$ contenant s_0 ?
   Voici les 3 circuits avec leur longueur respective :
   — $s_0, s_1, s_5, s_4, s_8, s_0$ de longueur 5
   — $s_0, s_1, s_2, s_3, s_4, s_8, s_0$ de longueur 6
   — $s_0, s_1, s_5, s_6, s_7, s_8, s_0$ de longueur 6
   La longueur minimum d'un circuit contenant s_0 est donc de 5.
3. Y a-t-il dans le graphe G_1 un autre circuit encore plus court, ne passant pas par s_0 ? Si oui, quelle est sa longueur ?
   La réponse est oui. Il y a un circuit de longueur 4 : s_1, s_5, s_4, s_8, s_1, qui ne passe pas par s_0.

4. Soit G un graphe orienté, v_0 un sommet de G et k la longueur minimum d'un circuit de G contenant v_0. Soit

$$C = v_0 e_1 v_1 e_2 v_2 \dots v_{k-1} e_k v_0$$

un circuit de longueur minimum contenant v_0 (donc de longueur k).
Montrer qu'à l'issue d'un parcours en largeur $PL(G, v_0)$ à partir du sommet v_0, on a $d[v_i] \leq i$ pour tout $i = 0, 1, \dots, k-1$.
Comme $d[v_i]$ a pour valeur la plus courte distance depuis le sommet de départ, donc ici v_0, à la fin de l'exécution de PL, et qu'il existe un chemin de longueur i entre v_0 et chacun des sommets v_i : $v_0 e_1 v_1 \dots e_i v_i$, on a bien $d[v_i] \leq i$ pour tout $i = 0, 1, \dots, k-1$.
Montrer qu'à l'issue d'un parcours en largeur $PL(G, v_0)$ à partir du sommet v_0, on a $d[v_i] = i$ pour tout $i = 0, 1, \dots, k-1$.
Preuve par contradiction : supposons qu'il existe i tel que $d[v_i] < i$. Cela signifie qu'il existe un chemin de longueur strictement inférieure à i entre v_0 et v_i. Si on le complète par $e_{i+1} v_{i+1} \dots v_{k-1} e_k v_0$, on obtient un circuit de longueur strictement inférieure à k, ce qui est contraire à l'hypothèse que $C = v_0 e_1 v_1 e_2 v_2 \dots v_{k-1} e_k v_0$ est un circuit de longueur minimum contenant v_0.

5. Soit deux sommets w_1 et w_2 deux sommets prédécesseurs d'un sommet $v0$ dans deux circuits distincts (respectivement C_1 et C_2) contenant v_0. Comment peut-on lors du parcours en largeur détecter lequel des deux circuits C_1 ou C_2 est le plus court ?
Le circuit le plus court correspondra au sommet w_i le plus proche de v_0, donc forcément le premier des deux visité lors du PL.

6. Comment peut-on détecter lors d'un parcours en largeur qu'un arc d'un graphe orienté est le dernier arc d'un circuit le plus court contenant le sommet de départ v_0 ? Proposer un algorithme $PCC(G, s)$ à partir d'une modification de $PL(G, s)$ qui calcule et retourne la longueur minimum d'un circuit contenant le sommet s de départ.
Le dernier arc d'un circuit le plus court contenant le sommet de départ v_0 sera le premier arc examiné n'appartenant pas à l'arbre (donc ayant comme extrémité terminale un sommet non BLANC) et ayant comme extrémité terminale le sommet v_0.
L'algorithme $PCC(G, s)$ est donné ci-dessous : Algorithme 1. La modification par rapport à l'algorithme $PL(G, s)$ consiste en l'ajout des lignes 18 à 22 et la ligne 27.

7. Proposer un algorithme pour calculer et retourner la longueur minimum d'un circuit dans un graphe orienté donné, en faisant des appels à l'algorithme PCC.
L'algorithme est donné ci-dessous : Algorithme 2.

8. Sachant que le graphe G est représenté par ses listes d'adjacence, en étudier la complexité.
L'algorithme $PCC(G, s)$ a la même complexité que le parcours en largeur, soit $O(n + m)$. Donc l'algorithme $PCCG(G)$ a une complexité de $O(n^2 + n \times m)$.

Algorithme 1 Plus court circuit contenant s PCC(G, s)

```
1: couleur(s) ← GRIS
2: d(s) ← 0
3: pere(s) ← NIL
4: F ← {s}
5: pour tout v ∈ V(G)\s faire
6:    couleur(v) ← BLANC
7:    d(v) ← ∞
8:    pere(v) ← NIL
9: fin pour
10: tant que F non vide faire
11:    v ← tete(F)
12:    pour tout w ∈ Adj(v) faire
13:       si couleur(w) = BLANC alors
14:          couleur(w) ← GRIS
15:          d(w) ← d(v) + 1
16:          pere(w) ← v
17:          Enfiler(F, w)
18:       sinon
19:          si w = s alors
20:             retourner d(w) + 1
21:          fin si
22:       fin si
23:    fin pour
24:    Defiler(F)
25:    couleur(v) ← NOIR
26: fin tant que
27: retourner ∞
```

Algorithme 2 Plus court circuit général PCCG(G)

```
1: min ← ∞
2: pour tout v ∈ V(G) faire
3:    pccv ← PCC(G, v)
4:    si pccv < min alors
5:       min ← pccv
6:    fin si
7: fin pour
8: retourner min
```

Algorithmes

Algorithme 3 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 4 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
```

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 Parcours en largeur PL(G, s)

```
1: couleur(s) ← GRIS
2: d(s) ← 0
3: pere(s) ← NIL
4: F ← {s}
5: pour tout v ∈ V(G)\s faire
6:    couleur(v) ← BLANC
7:    d(v) ← ∞
8:    pere(v) ← NIL
9: fin pour
10: tant que F non vide faire
11:    v ← tete(F)
12:    pour tout w ∈ Adj(v) faire
13:       si couleur(w) = BLANC alors
14:          couleur(w) ← GRIS
15:          d(w) ← d(v) + 1
16:          pere(w) ← v
17:          Enfiler(F, w)
18:       fin si
19:    fin pour
20:    Defiler(F)
21:    couleur(v) ← NOIR
22: fin tant que
```