

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.

Dans tout le sujet, on conviendra que, dans les différentes structures de données modélisant les graphes, les sommets sont rangés dans l'ordre croissant de leur nom.

1 Sommet source

Soit G un graphe orienté. Un sommet de G est une *source*, s'il n'a pas de voisin entrant.

1. Identifier toutes les sources du graphe G_1 de la Figure 1.

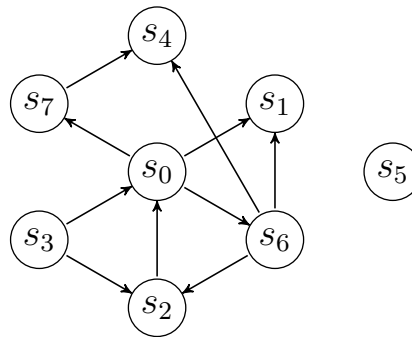


FIGURE 1 – Le graphe G_1

Réponse

Les sommets s_3 et s_5 sont des sources. Ils n'ont aucun prédécesseur.

2. Donner un exemple d'un graphe orienté qui n'a aucune source.

Réponse

Un circuit ne possède aucune source, comme celui à 3 sommets de la Figure 2.

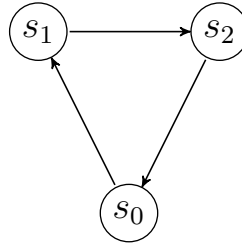


FIGURE 2 – Circuit à 3 sommets

3. Pour chacune des représentations

- (a) matrice d'adjacence,
- (b) listes d'adjacence

proposer un algorithme **Source**(G, s) qui retourne **True** si un sommet donné s d'un graphe orienté donné G est une source, et qui retourne **False** sinon.

Réponse pour la matrice d'adjacence

Il faut vérifier si s est de degré entrant 0, donc s'il n'y a aucun 1 dans sa colonne. Le graphe G est représenté par sa matrice d'adjacence A . On suppose également que s est représenté un entier.

Algorithme 1 **Source**(A, s)

```

1: pour tout  $i$  de 1 à  $n$  faire
2:   si  $A[i, s] \geq 1$  alors
3:     retourner False
4:   fin si
5: fin pour
6: retourner True

```

Réponse pour les listes d'adjacence

Il faut vérifier si s est de degré entrant 0, donc si s n'apparaît dans aucune des listes.

Algorithme 2 **Source**(G, s)

```

1: pour tout  $v \in V(G)$  faire
2:   pour tout  $w \in Adj(v)$  faire
3:     si  $w = s$  alors
4:       retourner False
5:     fin si
6:   fin pour
7: fin pour
8: retourner True

```

4. Donner la complexité des algorithmes, en fonction du nombre de sommets n , du nombre d'arcs m et du degré sortant maximum Δ .

Réponse pour la matrice d'adjacence

On parcourt toute la colonne du sommet s , donc la complexité est en $O(n)$.

Réponse pour les listes d'adjacence

On parcourt toutes les listes d'adjacence, donc la complexité est en $O(n + m)$.

2 Plus courts chemins

Les algorithmes de Dijkstra (Algorithme 3) et Bellman (Algorithme 4) sont rappelés à la fin de l'exercice.

Pour les deux graphes $G1$ et $G2$ des Figures 3 et 4, 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.

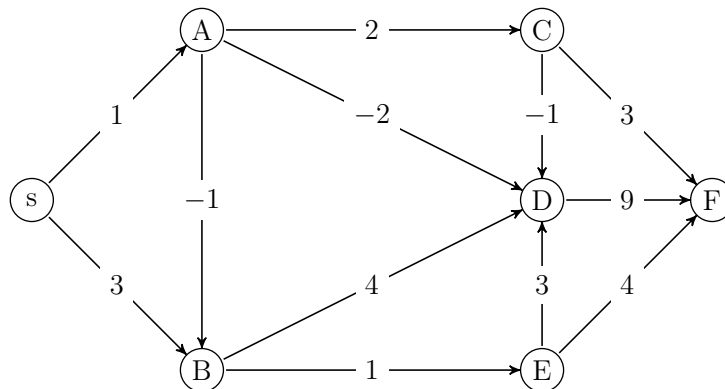


FIGURE 3 – Graphe $G1$

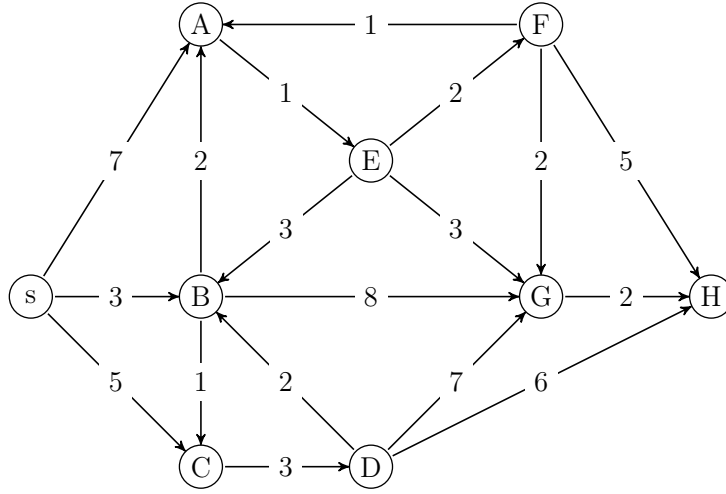


FIGURE 4 – Graphe G_2

Algorithme 3 Dijkstra(G, w, s)

```

1: pour tout  $v$  de  $V(G)$  faire
2:    $d(v) \leftarrow \infty$ 
3:    $pere(v) \leftarrow NIL$ 
4:    $couleur(v) \leftarrow BLANC$ 
5: fin pour
6:  $d(s) \leftarrow 0$ 
7:  $F \leftarrow FILE\_PRIORITE(\{s\}, d)$ 
8: tant que  $F \neq \emptyset$  faire
9:    $pivot \leftarrow EXTRAIRE\_MIN(F)$ 
10:  pour tout  $e = (pivot, v)$  arc sortant de  $pivot$  faire
11:    si  $couleur(v) = BLANC$  alors
12:      si  $d(v) = \infty$  alors
13:        INSERER( $F, v$ )
14:      fin si
15:      si  $d[v] > d[pivot] + w(e)$  alors
16:         $d[v] \leftarrow d[pivot] + w(e)$ 
17:         $pere[v] \leftarrow pivot$ 
18:      fin si
19:    fin si
20:  fin pour
21:   $couleur[pivot] \leftarrow NOIR$ 
22: fin tant que

```

Algorithme 4 Bellman(G, w, s)

H

```
1: pour tout  $v$  de  $V(G)$  faire
2:    $d[v] \leftarrow \infty$ 
3:    $pere[v] \leftarrow NIL$ 
4:    $npred[v] \leftarrow deg^-[v]$ 
5:   si  $npred(v) = 0$  alors
6:     INSERER_FILE( $F, v$ )
7:   fin si
8: fin pour
9:  $d[s] \leftarrow 0$ 
10: tant que  $F$  non vide faire
11:    $u \leftarrow TETE\_FILE(F)$ 
12:   DEFILER( $F$ )
13:   pour  $v \in Adj(u)$  faire
14:     si  $d[v] > d[u] + w(u, v)$  alors
15:        $d[v] \leftarrow d[u] + w[u, v]$ 
16:        $pere[v] \leftarrow u$ 
17:     fin si
18:      $npred(v) \leftarrow npred[v] - 1$ 
19:     si  $npred(v) = 0$  alors
20:       INSERER_FILE( $F, v$ )
21:     fin si
22:   fin pour
23: fin tant que
```

Solution du graphe de la Figure 3

On peut voir que le graphe de la Figure 3 a des poids négatifs, mais constater qu'il est sans circuit. On va donc lui appliquer l'algorithme de Bellman.

Le tableau ci-dessous donne son exécution, et la Figure 5 l'arborescence des plus courts chemins obtenus.

	s	A	B	C	D	E	F
	0/0	$\infty/1$	$\infty/2$	$\infty/1$	$\infty/4$	$\infty/1$	$\infty/3$
$s/0$	X	1/0	3/1				
$A/1$		X	0/0	3/0	-1/3		
$B/0$			X		-1/2	1/0	
$C/3$				X	-1/1		6/2
$E/1$					-1/0	X	5/1
$D/-1$					X		5/0
$F/5$							X

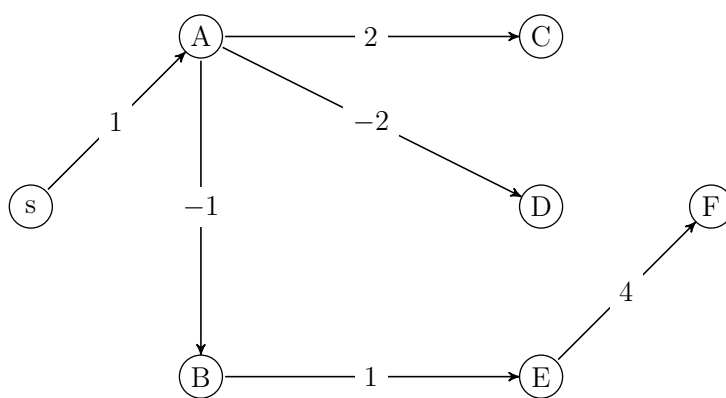


FIGURE 5 – Plus courts chemins du graphe $G1$

Solution du graphe de la Figure 4

On peut voir que le graphe de la Figure 4 n'a que des poids positifs, mais constater qu'il possède des circuits (par exemple $AEBA$). On va donc lui appliquer l'algorithme de Dijkstra.

Le tableau ci-dessous donne son exécution, et la Figure 6 l'arborescence des plus courts chemins obtenus.

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

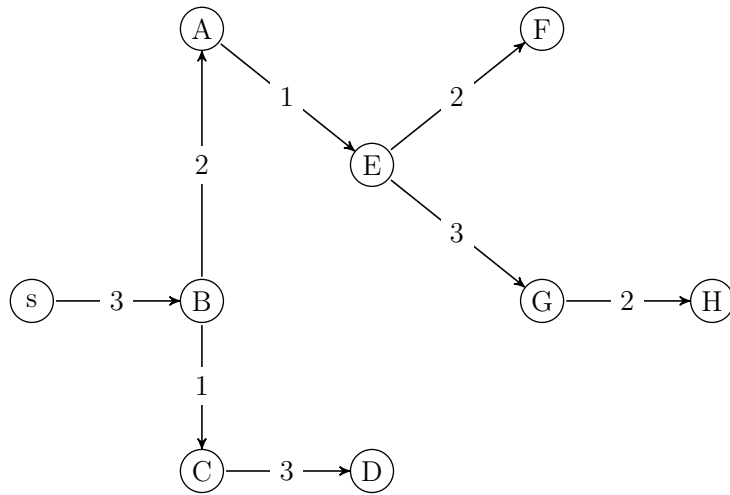


FIGURE 6 – Plus courts chemins du graphe G_2

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 7. 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 2 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.

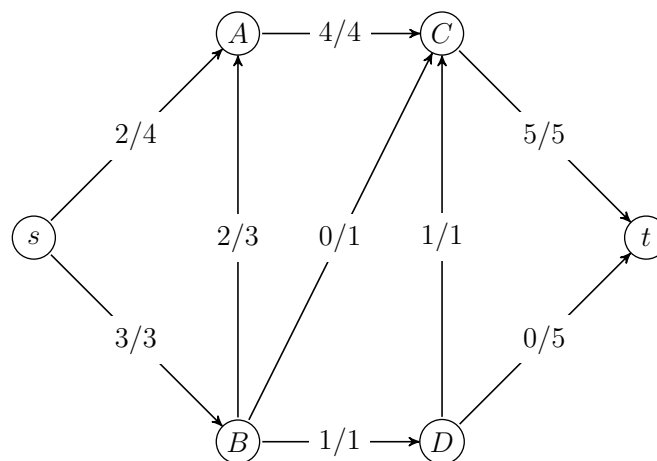


FIGURE 7 – Réseau

Réponse

Il n'y a qu'un seul chemin augmentant, la valeur de l'augmentation étant $+1$: $sABCDt$. On notera que l'utilisation des arcs AB et CD se font par marquage indirect.

Le flot maximum est donc de 6.

Lors de la deuxième tentative, l'ensemble des sommets marqués est l'ensemble $Y = \{s, A, B\}$. Les arcs de la coupe minimum sont donc AC , BC et BD .

$val(Y) = c(A, C) + c(B, C) + c(B, D) = 4 + 1 + 1 = 6$. On a bien la valeur de la coupe minimum qui égale la valeur du flot maximum.

Le flot maximum est donné par la Figure 8.

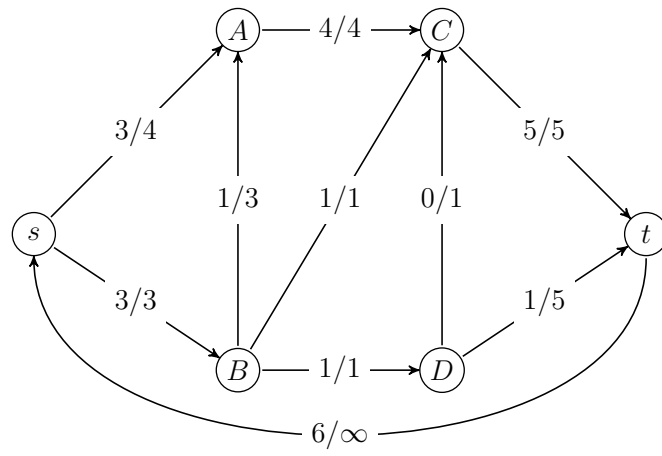


FIGURE 8 – Flot maximum

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:    $f[e] \leftarrow 0$ 
3: fin pour
4: répéter
5:   Marquage( $G, c, f, s, t$ )
6:   si  $t \in Y$  alors
7:      $v \leftarrow t$ 
8:      $C^+ \leftarrow \{(t, s)\}$ 
9:      $C^- \leftarrow \emptyset$ 
10:    tant que  $v \neq s$  faire
11:       $e \leftarrow A[v]$ 
12:      si  $v = T[e]$  alors
13:         $C^+ \leftarrow C^+ \cup \{e\}$ 
14:         $v \leftarrow I[e]$ 
15:      sinon
16:         $C^- \leftarrow C^- \cup \{e\}$ 
17:         $v \leftarrow T[e]$ 
18:      fin si
19:    fin tant que
20:  fin si
21:  pour tout  $e \in C^+$  faire
22:     $f(e) \leftarrow f(e) + \delta[t]$ 
23:  fin pour
24:  pour tout  $e \in C^-$  faire
25:     $f(e) \leftarrow f(e) - \delta[t]$ 
26:  fin pour
27: jusqu'à  $t \notin Y$ 

```

Algorithme 6 Marquage(G, c, f, s, t)

```
1:  $Y \leftarrow \{s\}$ 
2:  $\delta(s) \leftarrow +\infty$ 
3:  $Max \leftarrow \text{faux}$ 
4: tant que  $t \notin Y$  et  $Max = \text{faux}$  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:       $Max \leftarrow \text{vrai}$ 
16:    fin si
17:  fin si
18: fin tant que
```

4 Plus petit cycle contenant un sommet donné

Dans cet exercice, les graphes sont **simples**, **connexes**, **non-orientés** et chaque cycle considéré est supposé *élémentaire* (sans répétition de sommet ou d'arête).

Tous les parcours respecteront l'**ordre lexicographique** sur les sommets.

Notons qu'un graphe peut contenir plusieurs cycles et que chaque sommet peut être contenu dans plusieurs cycles.

1. Donner un exemple de graphe connexe dans lequel il existe à la fois
 - (a) au moins un sommet contenu dans aucun cycle (nommer ce sommet A) et
 - (b) au moins un sommet contenu dans un et un seul cycle (nommer ce sommet B)
et
 - (c) au moins un sommet contenu dans plusieurs cycles (nommer ce sommet C).

Réponse

La réponse est donnée par le graphe H de la Figure 9 ci-dessous.

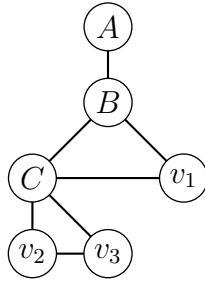


FIGURE 9 – Graphe H

2. Expliquer brièvement comment détecter lors d'un parcours en largeur l'existence d'un cycle. L'algorithme est rappelé ci-dessous.

```

0  PL(G,s)
1    pour chaque sommet u de X[G] \ {s} faire
2      couleur[u] <- BLANC
3      d[u] <- infini
4      pere[u] <- nil
5  couleur[s] <- GRIS
6  d[s] <- 0
7  pere[s] <- nil
8  Enfiler(F, s)
9  tant que non vide(F) faire
10   u <- tete(F)
11   pour chaque v de Adj(u) faire
12     si couleur[v] = BLANC
13       alors couleur[v] <- GRIS
14       d[v] <- d[u] + 1
15       pere[v] <- u
16       Enfiler(F, v)
17   Defiler(F)
18   couleur[u] <- NOIR

```

Réponse

Soit T l'arborescence obtenue par l'exécution de $PL(G, s)$ via le table *pere*.

Si à la ligne 12, le sommet v voisin de u n'est pas blanc et n'est pas le père de u dans T , cela signifie qu'il existe un ancêtre commun de u et v dans T .

Soit p cet ancêtre commun le plus éloigné de s , il existe un cycle composé de l'arête entre u et v et les chaînes de T reliant p à u et p à v . Donc après la ligne 16, il faut rajouter :

```

17   sinon si couleur[v] = GRIS et pere[u] != v
18     alors cycle <- VRAI

```

3. Le premier cycle détecté par un parcours en largeur contient-il forcément le sommet de départ ? Si oui, justifier. Sinon, donner un contre-exemple.

Réponse

Si on part d'un sommet n'appartenant à aucun cycle (par exemple le sommet A du graphe H de la Figure 9), le premier cycle obtenu ne contiendra pas le sommet de départ.

4. Donner un exemple de graphe G connexe qui satisfasse simultanément les deux conditions ci-dessous :
- (a) chaque sommet est contenu dans au moins un cycle, et
 - (b) il est possible de choisir dans G un sommet de départ v tel que le premier cycle détecté par $PL(G, v)$ ne contienne pas v .

On précisera bien l'ordre sur les sommets.

Réponse

Le graphe $H2$ ci-dessous de la Figure 10 répond à la question.

On exécute $PL(H2, v_1)$ en considérant que les sommets sont rangés en ordre croissant dans les listes d'adjacence. Le premier cycle détecté sera v_2, v_4, v_5, v_2 lors de l'examen de l'arête v_4v_5 alors que le cycle contenant $v_1 : v_1, v_2, v_5, v_6, v_3, v_1$ ne sera détecté que lors de l'examen des voisins de v_5 .

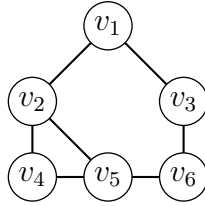


FIGURE 10 – Graphe $H2$

5. Étant donné un graphe G et un sommet v de G , on se propose de modifier l'algorithme de parcours en largeur $PL(G, v)$ de façon à calculer la longueur (c'est-à-dire le nombre d'arêtes) du cycle le plus court dans G contenant v , s'il en existe un ; si un tel cycle n'existe pas, l'algorithme retournera 0.

Pour cela, lors de l'exécution de $PL(G, v)$, on calculera pour chaque sommet u de G une nouvelle étiquette `branche[u]`. Cette étiquette `branche[u]` est définie comme suit :

$$\text{branche}[u] = \begin{cases} 0 & \text{si } u = v, \\ u & \text{si } \text{pere}[u] = v, \\ \text{branche}[\text{pere}[u]] & \text{sinon.} \end{cases}$$

- (a) En supposant que le sommet v est le sommet s_0 , calculer `branche[u]` pour tous les sommets u du graphe $G2$ de la FIGURE 11.

Réponse

Les valeurs de `branche` sont données dans le tableau ci-dessous.

u	s_0	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8
<code>branche[u]</code>	0	s_1	s_2	s_1	s_1	s_1	s_1	s_1	s_2

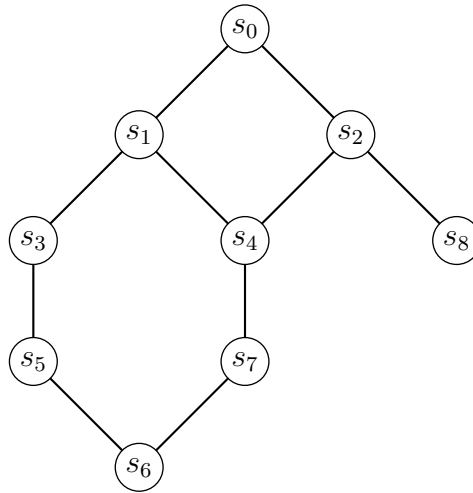


FIGURE 11 – Graphe G_2

- (b) Proposer une modification de l'algorithme de parcours en largeur qui, pour un graphe G et un sommet v de G , retourne la longueur (c'est-à-dire le nombre d'arêtes) du cycle le plus court dans G contenant v , s'il en existe un, et retourne 0 s'il n'en existe pas.

Remarque : vous noterez bien que cet algorithme ne détermine pas la longueur du plus court cycle de G mais la longueur du plus court cycle de G passant par v , le sommet de départ de $PL(G, v)$.

Réponse

L'algorithme est l'algorithme PL-modifié(G, s) donné ci-dessous.

```

0  PL-modifie(G,s)
1    pour chaque sommet u de X[G] \ {s} faire
2      couleur[u] <- BLANC
3      d[u] <- infini
4      pere[u] <- nil
5  couleur[s] <- GRIS
6  d[s] <- 0
7  pere[s] <- nil
8  branche[s] <- 0
9  Enfiler(F, s)
10 tant que non vide(F) faire
11   u <- tete(F)
12   pour chaque v de Adj(u) faire
13     si couleur[v] = BLANC
14       alors couleur[v] <- GRIS
15         d[v] <- d[u] + 1
16         pere[v] <- u
17         Enfiler(F, v)
18     si branche[u] = 0
19       alors branche[v] <- v
20     sinon
21       branche[v] <- branche[u]
22     sinon si couleur[v] = GRIS et pere[u] != v et branche[v] != branche[u]
23       alors retourner d[u] + d[v] + 1
24   Defiler(F)
25   couleur[u] <- NOIR
26 retourner 0

```