

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 Bellman

L'algorithme de Bellman est rappelé en fin d'exercice.

Appliquer l'algorithme de Bellman au graphe G de la Figure 1 pour calculer les plus courts chemins de s aux autres sommets du graphe. Donner

- l'ordre d'entrée des sommets dans la file F ;
- pour chaque sommet v la distance de s à v ;
- l'arborescence des plus courts chemins.

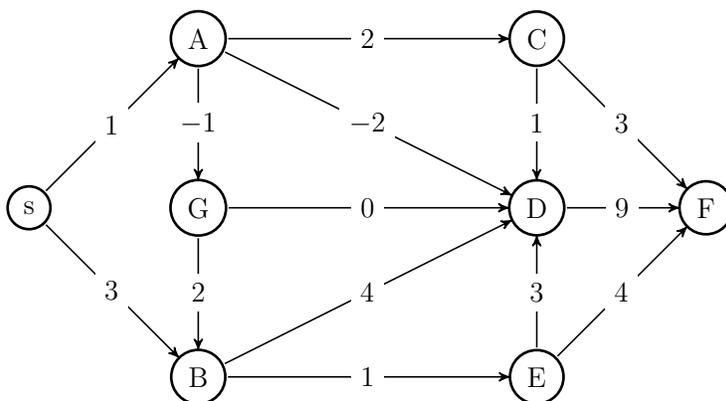


FIGURE 1 – Graphe G

```

0 Bellman(G,w,s)
1   pour chaque sommet v de V(G) faire
2     d[v] <- infini
3     pere[v] <- nil
4     npred[v] <- nombre de predecesseurs de v
5   d[s] <- 0
6   F <- CREER_FILE()
7   INSERER_FILE(F, s)
8   tant que F non vide faire
9     pivot <- TETE_FILE(F)
10    DEFILER(F)
11    pour tout v successeur de pivot faire
12      si d[v] > d[pivot] + w(pivot, v) alors
13        d[v] <- d[pivot] + w(pivot, v)
14        pere[v] <- pivot
15      npred[v] <- npred[v] - 1
16      si npred[v] = 0 alors
17        INSERER_FILE(F, v)

```

L'exécution de Bellman peut-être résumée par le tableau suivant, où dans chaque case, on note les modifications liées au traitement de la tête de file de d , la distance par rapport à s et $npred$ le nombre de prédécesseurs restant à traiter : $d/npred$.

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

L'ordre d'entrée dans la file F est donc s, A, C, G, B, E, D, F .

L'arborescence des plus courts chemins et le tableau des distances sont donnés ci-dessous :

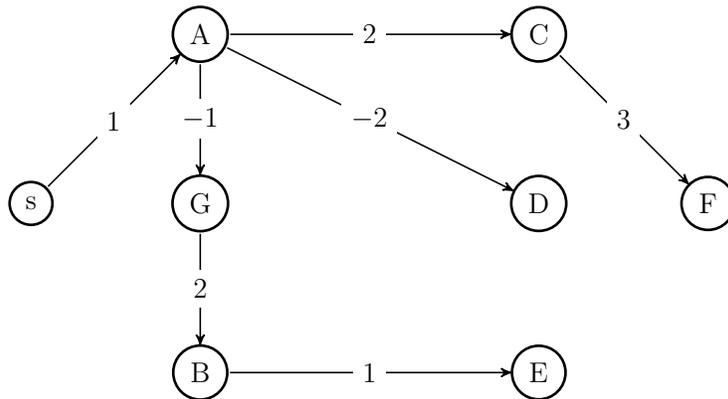


FIGURE 2 – Arborescence des plus courts chemins de G

Tableau des distances :

s	A	B	C	D	E	F	G
0	1	2	3	-1	3	6	0

2 Détection d'arcs opposés

On considère un graphe orienté à n sommets et m arcs. On dit que deux arcs e_1 et e_2 sont opposés si $e_1 = (i, j)$ et $e_2 = (j, i)$ avec i et j deux sommets distincts du graphe.

Pour chacune des trois représentations : listes de successeurs, matrice d'adjacence, matrice d'incidence, écrire un algorithme pour déterminer si le graphe contient des arcs opposés et en étudier la complexité. *Attention de présenter les algorithmes de façon lisible!*

Représentation par les listes de successeurs

Données : G graphe orienté

ContientArcsOpposes(G) :

- 1: **pour tout** $u \in V(G)$ **faire**
- 2: **pour tout** $v \in Adj(u)$ **faire**
- 3: **pour tout** $w \in Adj(v)$ **faire**
- 4: **si** $w = u$ **alors**
- 5: **retourner** VRAI
- 6: **fin si**
- 7: **fin pour**
- 8: **fin pour**
- 9: **fin pour**
- 10: **retourner** FAUX

Complexité :

La boucle 3-7 est en $O(\Delta)$

Cette boucle est exécutée une fois par arc.

La boucle 1 passe en revue tous les sommets du graphe

Donc l'algorithme est en $O(n + m \times \Delta(G))$

Représentation par la matrice d'adjacence

Données : A , matrice d'adjacence d'un graphe G

ContientArcsOpposes(A) :

```
1: pour  $i$  de 1 à  $n$  faire
2:   pour  $j$  de  $i + 1$  à  $n$  faire
3:     si  $A[i, j] = 1$  et  $A[j, i] = 1$  alors
4:       retourner VRAI
5:     fin si
6:   fin pour
7: fin pour
8: retourner FAUX
```

Complexité :

dans le pire des cas (aucune paire d'arcs opposés), on parcourt tout le triangle de la matrice au dessus de la diagonale et donc on exécute $\frac{n \times (n-1)}{2}$ fois les lignes 3 à 5.

Par conséquent, la complexité est en $O(n^2)$.

Représentation par la matrice d'adjacence

Données : B , matrice d'incidence d'un graphe G

ContientArcsOpposes(B) :

```
1: pour  $j$  de 1 à  $m$  faire
2:   pour  $i$  de 1 à  $n$  faire
3:     si  $B[i, j] = -1$  alors
4:       origine  $\leftarrow i$ 
5:     sinon si  $B[i, j] = 1$  alors
6:       destination  $\leftarrow i$ 
7:     fin si
8:   fin pour
9:   pour  $j2$  de  $j + 1$  à  $m$  faire
10:    si  $B[origine, j2] = 1$  et  $B[destination, j2] = -1$  alors
11:      retourner VRAI
12:    fin si
13:  fin pour
14: fin pour
15: retourner FAUX
```

Complexité :

dans le pire des cas (aucune paire d'arcs opposés), on exécute la boucle 1 m fois. Dans cette boucle, on va exécuter la boucle 2 n fois (soit au total du $O(n \times m)$). La boucle 9 combinée avec la boucle 1 sera exécutée en tout $\frac{m \times (m-1)}{2}$ fois, soit du $O(m^2)$.

Par conséquent, la complexité est en $O(m \times (n + m))$.

3 Plus courts chemins entre toutes paires de sommets

Rappels : l'algorithme de Floyd calcule les plus courts chemins entre toutes paires de sommets avec une complexité en $O(n^3)$. L'algorithme de Dijkstra calcule les plus courts chemins entre un sommet s et les autres sommets d'un graphe où tous les arcs ont un poids positif. En utilisant un tas de Fibonacci pour implémenter la file de priorité, la complexité de l'algorithme de Dijkstra est en $O(n \cdot \log(n) + m)$.

3.1) Il est possible, pour calculer les distances entre toutes paires de sommets, de remplacer l'algorithme de Floyd par n itérations de l'algorithme de Dijkstra, à condition que la fonction de poids soit positive. A quelle condition est-ce préférable ?

Il faut que $n^2 \times \log(n) + n \times m$ soit d'ordre inférieur à n^3 , donc que le nombre d'arcs ne soit pas de l'ordre de n^2 .

En fait, il est possible de contourner la limitation d'avoir tous les arcs de poids positifs. Supposons que la fonction de poids w sur G ne soit pas positive. Si (G, w) ne possède pas de circuit strictement négatif, nous allons définir une nouvelle fonction de poids \hat{w} sur G , positive et telle que (G, w) et (G, \hat{w}) aient les mêmes plus courts chemins.

Soit h une fonction de $V(G) \rightarrow \mathbb{R}$ et w une fonction de $E(G) \rightarrow \mathbb{R}$. Soit \hat{w} la fonction de $E(G) \rightarrow \mathbb{R}$ définie par : $\forall (u, v) \in E(G), \hat{w}(u, v) = w(u, v) + h(u) - h(v)$

3.2) Montrer que $p = v_1, \dots, v_k$ est un plus court chemin de (G, w) si et seulement si p est un plus court chemin de (G, \hat{w}) .

$$\hat{w}(p) = \hat{w}(v_1 v_2) + \dots + \hat{w}(v_{k-1} v_k)$$

$$\text{Donc } \hat{w}(p) = w(v_1 v_2) + h(v_1) - h(v_2) + w(v_2 v_3) + h(v_2) - h(v_3) + \dots + w(v_{k-1} v_k) + h(v_{k-1}) - h(v_k)$$

$$\text{Donc } \hat{w}(p) = w(v_1 v_2) + \dots + w(v_{k-1} v_k) + h(v_1) - h(v_k)$$

$$\text{Donc } \hat{w}(p) = w(p) + h(v_1) - h(v_k)$$

Donc $\hat{w}(p)$ sera minimum si et seulement si $w(p)$ est minimum.

Donc p est un plus court chemin de (G, w) si et seulement si p est un plus court chemin de (G, \hat{w}) .

3.3) Montrer que (G, w) est sans circuit strictement négatif si et seulement si (G, \hat{w}) est sans circuit strictement négatif.

Soit $C = v_1, \dots, v_k, v_1$ un circuit de G .

$$\hat{w}(C) = \hat{w}(v_1v_2) + \dots + \hat{w}(v_{k-1}v_k) + \hat{w}(v_kv_1)$$

$$\text{Donc } \hat{w}(C) = w(v_1v_2) + h(v_1) - h(v_2) + w(v_2v_3) + h(v_2) - h(v_3) + \dots + w(v_{k-1}v_k) + h(v_{k-1}) - h(v_k) + w(v_kv_1) + h(v_k) - h(v_1)$$

$$\text{Donc } \hat{w}(C) = w(v_1v_2) + \dots + w(v_{k-1}v_k) + w(v_kv_1)$$

$$\text{Donc } \hat{w}(C) = w(C)$$

Donc C est un circuit strictement négatif de (G, w) si et seulement si C est un circuit strictement négatif de (G, \hat{w}) .

Donc (G, w) est sans circuit strictement négatif si et seulement si (G, \hat{w}) est sans circuit strictement négatif.

Il nous faut maintenant trouver une fonction h en fonction de w et telle que \hat{w} soit positive. D'après la question précédente, une telle fonction n'existera que si (G, w) est sans circuit strictement négatif. Pour cela, on se donne un graphe $G' = (V', E')$ avec $V' = V(G) \cup \{s'\}$ et $E' = E(G) \cup \{(s', v), v \in V(G)\}$. G' est donc un graphe obtenu à partir de G en rajoutant un sommet s' et tous les arcs entre s' et les sommets issus de $V(G)$. Le graphe G' est muni d'une fonction de poids w' définie par $\forall e \in E(G), w'(e) = w(e)$ et $\forall v \in V(G), w'(s', v) = 0$.

3.4) Montrer que pour tout graphe G , G' est sans circuit strictement négatif si et seulement si G est sans circuit strictement négatif.

Comme s' n'a aucun arc entrant, il ne peut appartenir à aucun circuit de G' .

Par conséquent, tous les circuits de G' sont des circuits de G . Donc G' est sans circuit strictement négatif si et seulement si G est sans circuit strictement négatif.

3.5) Dessiner le graphe G' correspondant au graphe G de la figure 3.

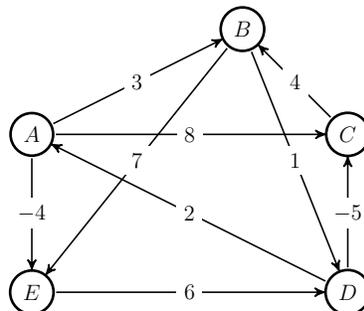


FIGURE 3 – Graphe G

Le graphe G' est donné par la Figure 4

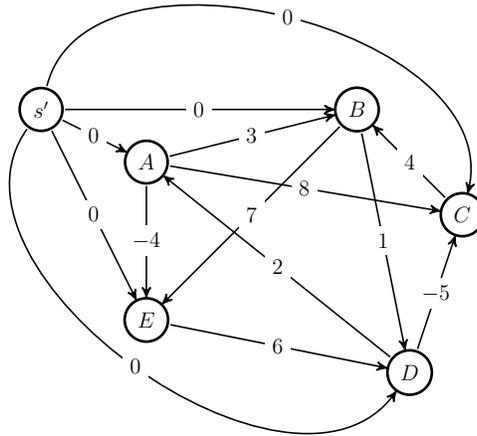


FIGURE 4 – Graphe G'

La première étape de notre nouvel algorithme consiste à calculer les plus courtes distances entre s' et les autres sommets de G' , à l'aide l'algorithme de Ford rappelé ci-dessous (Algorithme 1). Cette étape permet également de vérifier si G' (et donc G) est sans circuit strictement négatif.

3.6) Appliquer l'algorithme de Ford au graphe G' obtenu à la question précédente.

L'algorithme de Ford renvoie **VRAI** si et seulement si le graphe G muni de la fonction de poids sur les arcs w est sans circuit négatif. La fonction d donne les plus courtes distances entre le sommet s et les autres sommets du graphe G .

Algorithme 1 Ford(G, w, s)

```
1: pour tout  $v$  de  $V(G)$  faire
2:    $d[v] \leftarrow \infty$ 
3:    $pere[v] \leftarrow NIL$ 
4: fin pour
5:  $d[s] \leftarrow 0$ 
6: pour  $i$  de 1 à  $n - 1$  faire
7:   pour tout arc  $e = (u, v) \in E(G)$  faire
8:     si  $d[v] > d[u] + w(u, v)$  alors
9:        $d[v] \leftarrow d[u] + w[u, v]$ 
10:       $pere[v] \leftarrow u$ 
11:    fin si
12:  fin pour
13: fin pour
14: pour tout arc  $e = (u, v) \in E(G)$  faire
15:   si  $d[v] > d[u] + w(u, v)$  alors
16:     retourner FAUX
17:   fin si
18: fin pour
19: retourner VRAI
```

Exécution de Ford sur G' depuis s'

i	u	s'	A	B	C	D	E
		0	∞	∞	∞	∞	∞
1	s'	0	0	0	0	0	0
	A						-4
	B						
	C						
	D				-5		
	E						
2	C			-1			
	E						
3	B						
4							
5							
6							

L'exécution de Ford sur G' depuis s' renvoie VRAI avec les valeurs de d suivantes :

v	s'	A	B	C	D	E
$d[v]$	0	0	-1	-5	0	-4

Pour terminer notre algorithme, on calcule la fonction de poids \hat{w} en prenant comme fonction h la distance dans le graphe G' entre s' et les sommets de G . Puis on applique

l'algorithme de Dijkstra n fois en prenant à chaque fois une source différente.

3.7) Quelle est la complexité de l'algorithme de Ford ? Et celle de notre algorithme ?

La complexité de l'algorithme de Ford est en $O(n \times m)$.

Le calcul de G' et de \hat{w} à partir de h sera en $O(n + m)$.

Le calcul des plus courtes distances dans G' entre toute paire de sommets en utilisant Dijkstra sera en $O(n \times (n \times \log(n) + m))$. Donc la complexité de notre algorithme sera en $O(n \times (n \times \log(n) + m))$, soit $O(n^2 \times \log(n) + n \times m)$.

3.8) Montrer que pour tout graphe G et fonction de poids w sans circuit négatif, la fonction \hat{w} obtenue est positive.

Soit uv un arc de G .

$$\hat{w}(uv) = w(uv) + h(u) - h(v).$$

$$\text{Donc } \hat{w}(uv) = w(uv) + d_{G'}(s', u) - d_{G'}(s', v).$$

$$\text{On a } d_{G'}(s', v) \leq d_{G'}(s', u) + w(uv).$$

$$\text{Donc } 0 \leq d_{G'}(s', u) + w(uv) - d_{G'}(s', v).$$

Donc $w(uv) + d_{G'}(s', u) - d_{G'}(s', v) \geq 0$. Donc pour tout arc uv de G , $\hat{w}(uv)$ est positive.

3.9) Dessiner le graphe G de la figure 3 avec la fonction de poids \hat{w} .

Le graphe G avec la fonction de poids \hat{w} est donné par la Figure 5.

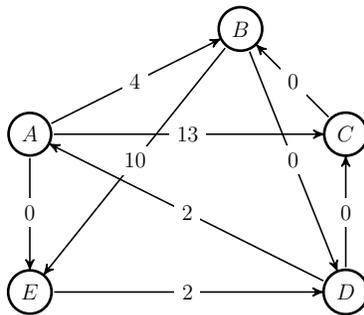


FIGURE 5 – Graphe (G, \hat{w})

3.10) Calculer les plus courtes distances entre toute paire de sommets dans (G, \hat{w}) en utilisant l'algorithme de Dijkstra rappelé ci-dessous (Algorithme 2) successivement à partir de chacun des sommets. Pour chaque sommet source, on donnera l'arborescence des plus courts chemins.

Algorithme 2 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
```

Les exécutions de Dijkstra et les arborescences sont données ci-dessous :

Plus courts chemins à partir de A

<i>Pivot</i>	<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>E</i>
	0	∞	∞	∞	∞
<i>A</i>	<i>X</i>	4	13		0
<i>E</i>				2	<i>X</i>
<i>D</i>			2	<i>X</i>	
<i>C</i>		2	<i>X</i>		
<i>B</i>		<i>X</i>			

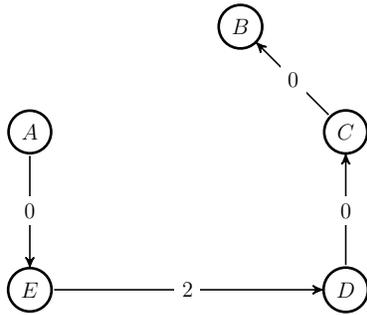


FIGURE 6 – Plus courts chemins à partir de A

Plus courts chemins à partir de B

<i>Pivot</i>	<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>E</i>
	∞	0	∞	∞	∞
<i>B</i>		<i>X</i>		0	10
<i>D</i>	2		0	<i>X</i>	
<i>C</i>			<i>X</i>		
<i>A</i>	<i>X</i>				2
<i>E</i>					<i>X</i>

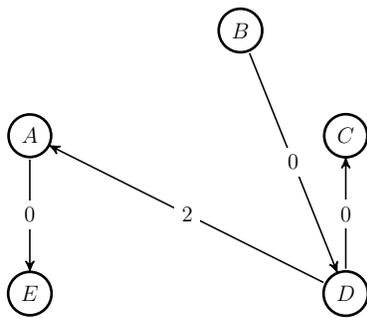


FIGURE 7 – Plus courts chemins à partir de B

Plus courts chemins à partir de C

<i>Pivot</i>	<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>E</i>
	∞	∞	0	∞	∞
<i>C</i>		0	<i>X</i>		
<i>B</i>		<i>X</i>		0	10
<i>D</i>	2			<i>X</i>	
<i>A</i>	<i>X</i>				2
<i>E</i>					<i>X</i>

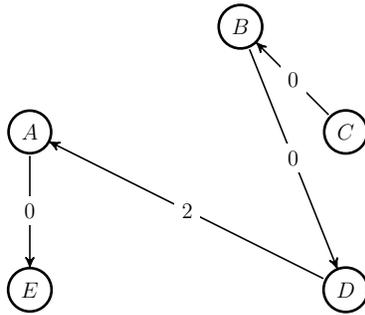


FIGURE 8 – Plus courts chemins à partir de C

Plus courts chemins à partir de D

<i>Pivot</i>	<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>E</i>
	∞	∞	∞	0	∞
<i>D</i>	2		0	X	
<i>C</i>		0	X		
<i>B</i>		X			10
<i>A</i>	X				2
<i>E</i>					X

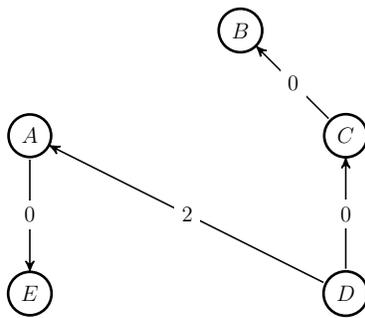


FIGURE 9 – Plus courts chemins à partir de D

Plus courts chemins à partir de E

<i>Pivot</i>	<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>E</i>
	∞	∞	∞	∞	0
<i>E</i>				2	X
<i>D</i>	4		2	X	
<i>C</i>		2	X		
<i>B</i>		X			
<i>A</i>	X				

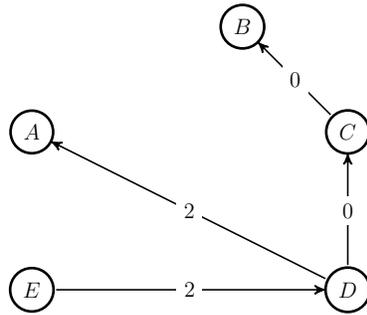


FIGURE 10 – Plus courts chemins à partir de E

3.11) Donner la matrice des plus courtes distances dans (G, w) .

Plus courtes distances dans (G, w)

	<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>E</i>
<i>A</i>	0	1	-3	2	-4
<i>B</i>	3	0	-4	1	-1
<i>C</i>	7	4	0	5	3
<i>D</i>	2	-1	-5	0	-2
<i>E</i>	8	5	1	6	0