

*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 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 deux représentations :

1. listes de successeurs,
2. matrice d'adjacence,

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 !*

### Corrigé Listes d'adjacence

```

pour tout sommet u de V(G) faire
  pour tout sommet v de Adj(u) faire
    si v != u alors
      pour tout sommet w de Adj(v) faire
        si w = u alors
          retourner VRAI
        fin si
      fin pour
    fin si
  fin pour
fin pour
retourner FAUX

```

### Complexité

La complexité de l'algorithme ci-dessus est en  $O(n + m \times \Delta^+)$  où  $n$  est le nombre de sommets,  $m$  le nombre d'arcs et  $\Delta^+$  le degré sortant maximum.

On peut améliorer un peu l'algorithme en divisant le temps de calcul par 2, ce qui ne change pas sa complexité.

```
pour tout sommet u de V(G) faire
  marque[u] <- FAUX
fin pour
pour tout sommet u de V(G) faire
  marque[u] <- VRAI
  pour tout sommet v de Adj(u) faire
    si marque[v] = FAUX alors
      pour tout sommet w de Adj(v) faire
        si w = u alors
          retourner VRAI
        fin si
      fin pour
    fin si
  fin pour
fin pour
retourner FAUX
```

### Corrigé Matrice d'adjacence

```
pour tout i de 1 à n faire
  pour tout j de i+1 à n faire
    si A[i, j] > 0 et A[j, i] > 0 alors
      retourner VRAI
    fin si
  fin pour
fin pour
retourner FAUX
```

### Complexité

La complexité de l'algorithme ci-dessus est en  $O(n^2)$ .

## 2 Flot Maximum

**2.1)** En utilisant l'algorithme de Ford-Fulkerson rappelé ci-dessous (Algorithme 1), trouver le flot maximum entre les sommets  $s$  et  $t$  du réseau de la figure 1. 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 1 et une capacité de 4. On donnera pour chaque exécution de la procédure de marquage le chemin augmentant obtenu et l'augmentation correspondante.

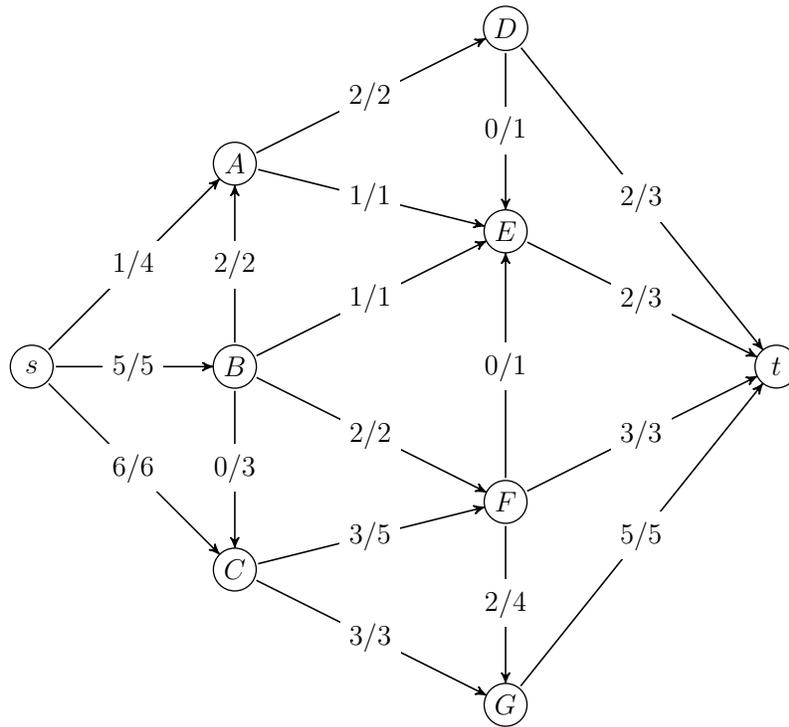


FIGURE 1 – Réseau

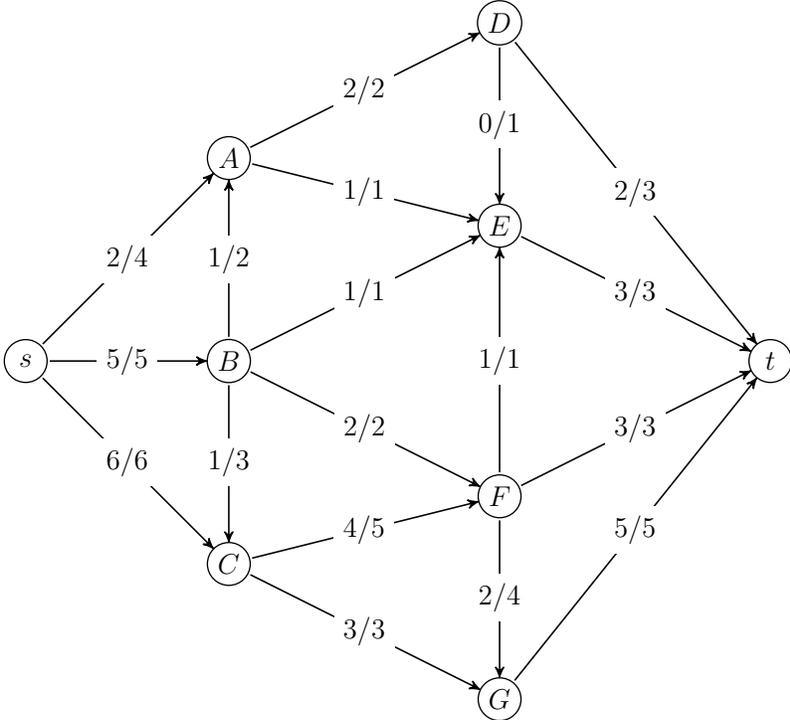
chaîne augmentante :  $sABCFEt$  avec  $\delta = +1$ .

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

Les sommets marqués lors du dernier appel à Marquage sont :

$Y = \{s, A, B, C, F, G\}$ , ce qui donne une coupe de valeur  $c(AD) + c(AE) + c(BE) + c(FC) + c(Ft) + c(Gt) = 2 + 1 + 1 + 1 + 3 + 5 = 13$ . La valeur du flot trouvée étant également de 13, on a bien un flot maximum et une coupe minimum.

Flot maximum



Dans l'Algorithme 1 (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 1** 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 2** 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
```

---

### 3 Algorithme de Johnson

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 ?

Si on exécute  $n$  fois Dijkstra, on aura une complexité de  $O(n^2 \cdot \log(n) + n \cdot m)$ . C'est préférable si le nombre d'arcs n'est pas en  $O(n^2)$ , par exemple si le nombre d'arcs sortant de chaque sommet est borné par  $\log(n)$ .

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  $(G, \hat{w})$ .

Soit  $w(p)$  le poids total du chemin  $p$  avec la pondération  $w$  et  $\hat{w}(p)$  le poids total du chemin  $p$  avec la pondération  $\hat{w}$ .

$$\begin{aligned}\hat{w}(p) &= \sum_{1 \leq i < k} \hat{w}(v_i v_{i+1}) = \sum_{1 \leq i < k} [w(v_i v_{i+1}) + h(v_i) - h(v_{i+1})] \\ &= \sum_{1 \leq i < k} w(v_i v_{i+1}) + h(v_1) - h(v_k) = w(p) + h(v_1) - h(v_k)\end{aligned}$$

La différence entre  $w(p)$  et  $\hat{w}(p)$  ne dépend donc que des sommets de départ et d'arrivée et donc  $w(p)$  et  $\hat{w}(p)$  seront minimaux si et seulement si  $p$  est un plus court chemin de  $v_1$  à  $v_k$  dans  $(G, w)$  et dans  $(G, \hat{w})$ .

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

Soit  $C = v_1, \dots, v_k, v_1$  un circuit de  $G$ . D'après le calcul de question précédente, on a  $\hat{w}(C) = w(C)$ . 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  $E' = E(G) \cup \{(s', v), v \in V(G)\}$ . Le graphe  $G'$  est muni d'une fonction de poids  $w'$  définie par  $\forall e \in E(G), w'(e) = w(e)$  et  $\forall v \in V(G), w'(s', v) = 0$ . En d'autres termes, le graphe  $G'$  est obtenu en rajoutant au graphe  $G$  un nouveau sommet  $s'$  qui est relié à tous les sommets déjà existants par un arc de poids nul.

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

Le rajout du sommet  $s'$  et des arcs sortants de  $s'$  ne rajoute pas de circuit à  $G$ . Donc les circuits de  $G'$  sont les mêmes que ceux de  $G$ . Et 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 2.

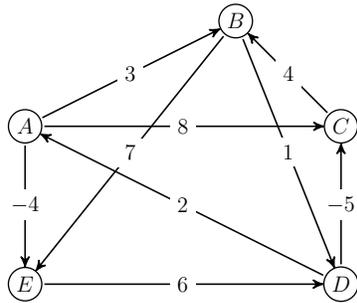
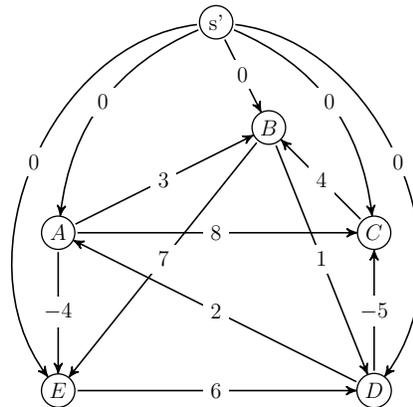


FIGURE 2 – Graphe G

Graphe  $G'$  issu du graphe  $G$  de la Figure 2



La première étape de l'algorithme de Johnson 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 3). 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 3** 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
```

---

### Application de Ford à $G'$

|                        | $s'$ | $A$      | $B$      | $C$      | $D$      | $E$      |
|------------------------|------|----------|----------|----------|----------|----------|
|                        | 0    | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ |
| $i = 1$                |      |          |          |          |          |          |
| arcs sortant de $s'/0$ |      | 0        | 0        | 0        | 0        | 0        |
| arcs sortant de $A/0$  |      |          |          |          |          | -4       |
| arcs sortant de $B/0$  |      |          |          |          |          |          |
| arcs sortant de $C/0$  |      |          |          |          |          |          |
| arcs sortant de $D/0$  |      |          |          | -5       |          |          |
| arcs sortant de $E/-4$ |      |          |          |          |          |          |
| $i = 2$                |      |          |          |          |          |          |
| arcs sortant de $s'/0$ |      |          |          |          |          |          |
| arcs sortant de $A/0$  |      |          |          |          |          |          |
| arcs sortant de $B/0$  |      |          |          |          |          |          |
| arcs sortant de $C/-5$ |      |          | -1       |          |          |          |
| arcs sortant de $D/0$  |      |          |          |          |          |          |
| arcs sortant de $E/-4$ |      |          |          |          |          |          |

Il n'y a plus aucune modification pour  $i = 3, 4$  et  $5$ , ni lors de l'exécution de la boucle 14. Donc Ford retourne VRAI.

Pour terminer l'algorithme de Johnson, 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 l'algorithme de Johnson ?

L'algorithme de Ford est en  $O(n.m)$ . Et donc l'algorithme de Johnson est en  $O(n.m) + O(n^2 \cdot \log(n) + n.m)$ , soit  $O(n^2 \cdot \log(n) + n.m)$ .

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

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

Or  $d_{G'}(s', v) \leq d_{G'}(s', u) + w(u, v)$ .

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

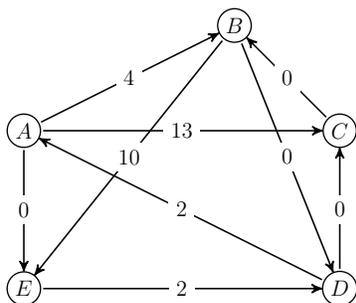
Et donc  $\hat{w}(uv) \geq 0$ .

3.9) Dessiner le graphe  $G$  de la figure 2 avec la fonction de poids  $\hat{w}$ .

On rappelle dans le tableau suivant les valeurs de  $d_{G'}(s', v)$  pour tous les sommets  $v$  de  $G$  obtenues par l'exécution de l'algorithme de Ford :

|                 |     |     |     |     |     |
|-----------------|-----|-----|-----|-----|-----|
|                 | $A$ | $B$ | $C$ | $D$ | $E$ |
| $d_{G'}(s', v)$ | 0   | -1  | -5  | 0   | -4  |

Graphe  $G$  muni de la fonction de poids  $\hat{w}$



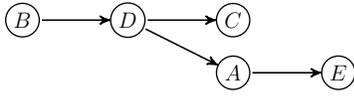
3.10) Calculer les plus courtes distances entre toute paire de sommets dans  $(G, \hat{w})$  en utilisant l'algorithme de Dijkstra rappelé ci-dessous (Algorithme 4) successivement à partir de chacun des sommets. Pour chaque sommet source, on donnera l'arborescence des plus courts chemins.

Les tableaux suivants donnent le calcul des distances à partir de chaque sommet, de  $A$  à  $E$  et sont suivis de l'arborescence des plus courts chemins correspondant.

|              |     |          |          |          |          |
|--------------|-----|----------|----------|----------|----------|
| <i>pivot</i> | $A$ | $B$      | $C$      | $D$      | $E$      |
|              | 0   | $\infty$ | $\infty$ | $\infty$ | $\infty$ |
| $A/0$        | X   | 4        | 13       |          | 0        |
| $E/0$        | X   |          |          | 2        | X        |
| $D/2$        | X   |          | 2        | X        | X        |
| $C/2$        | X   | 2        | X        | X        | X        |
| $B/2$        | X   | X        | X        | X        | X        |



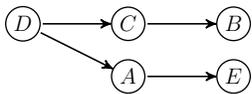
|              |          |     |          |          |          |
|--------------|----------|-----|----------|----------|----------|
| <i>pivot</i> | $A$      | $B$ | $C$      | $D$      | $E$      |
|              | $\infty$ | 0   | $\infty$ | $\infty$ | $\infty$ |
| $B/0$        |          | X   |          | 0        | 10       |
| $D/0$        | 2        | X   | 0        | X        |          |
| $C/0$        |          | X   | X        | X        |          |
| $A/2$        | X        | X   | X        | X        | 2        |
| $E/2$        | X        | X   | X        | X        | X        |



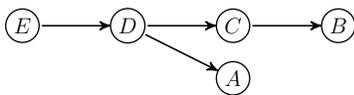
| <i>pivot</i> | <i>A</i> | <i>B</i> | <i>C</i> | <i>D</i> | <i>E</i> |
|--------------|----------|----------|----------|----------|----------|
|              | $\infty$ | $\infty$ | 0        | $\infty$ | $\infty$ |
| <i>C/0</i>   |          | 0        | <i>X</i> |          |          |
| <i>B/0</i>   |          | <i>X</i> | <i>X</i> | 0        | 10       |
| <i>D/0</i>   | 2        | <i>X</i> | <i>X</i> | <i>X</i> |          |
| <i>A/2</i>   | <i>X</i> | <i>X</i> | <i>X</i> | <i>X</i> | 2        |
| <i>E/2</i>   | <i>X</i> | <i>X</i> | <i>X</i> | <i>X</i> | <i>X</i> |



| <i>pivot</i> | <i>A</i> | <i>B</i> | <i>C</i> | <i>D</i> | <i>E</i> |
|--------------|----------|----------|----------|----------|----------|
|              | $\infty$ | $\infty$ | $\infty$ | 0        | $\infty$ |
| <i>D/0</i>   | 2        |          | 0        | <i>X</i> |          |
| <i>C/0</i>   |          | 0        | <i>X</i> | <i>X</i> |          |
| <i>B/0</i>   |          | <i>X</i> | <i>X</i> | <i>X</i> | 10       |
| <i>A/2</i>   | <i>X</i> | <i>X</i> | <i>X</i> | <i>X</i> | 2        |
| <i>E/2</i>   | <i>X</i> | <i>X</i> | <i>X</i> | <i>X</i> | <i>X</i> |



| <i>pivot</i> | <i>A</i> | <i>B</i> | <i>C</i> | <i>D</i> | <i>E</i> |
|--------------|----------|----------|----------|----------|----------|
|              | $\infty$ | $\infty$ | $\infty$ | $\infty$ | 0        |
| <i>E/0</i>   |          |          |          | 2        | <i>X</i> |
| <i>D/2</i>   | 4        |          | 2        | <i>X</i> | <i>X</i> |
| <i>C/2</i>   |          | 2        | <i>X</i> | <i>X</i> | <i>X</i> |
| <i>B/2</i>   |          | <i>X</i> | <i>X</i> | <i>X</i> | <i>X</i> |
| <i>A/4</i>   | <i>X</i> | <i>X</i> | <i>X</i> | <i>X</i> | <i>X</i> |



---

**Algorithme 4** 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
```

---

