2021-04-28

Plus court chemin

Christophe Viroulaud

Terminale NSI

Plus court chemin

Christophe Viroulaud

Terminale NSI

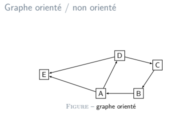
Comment fonctionnent les algorithmes de plus court chemin ?

RIP déjà utilisé avec ARPANET !

Comment fonctionnent les algorithmes de plus court chemin ?

2021-04-28

Plus court chemin
└Retour des graphes
└Graphe orienté
└Graphe orienté / non orienté



Graphe orienté / non orienté

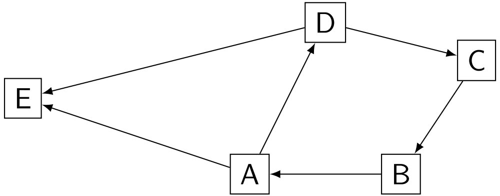


Figure – graphe orienté

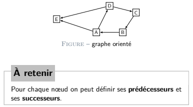


Figure – graphe orienté

À retenir

Pour chaque nœud on peut définir ses **prédécesseurs** et ses **successeurs**.

1. Le nœud E ne possède pas de *successeur*.

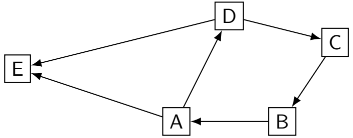


Figure – graphe orienté

À retenir

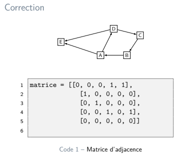
Pour chaque nœud on peut définir ses **prédécesseurs** et ses **successeurs**.

Activité 1 :
1. Établir la matrice d'adjacence du graphe figure 2.
2. Établir le dictionnaire des successeurs du graphe figure 2.
3. Déterminer un cycle dans le graphe.

on peut faire dictionnaire des prédécesseurs

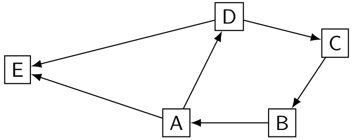
Activité 1 :

1. Établir la matrice d'adjacence du graphe figure 2.
2. Établir le dictionnaire des successeurs du graphe figure 2.
3. Déterminer un cycle dans le graphe.



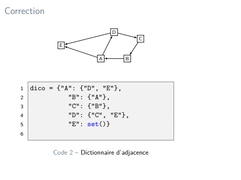
La matrice n'est plus symétrique

Correction



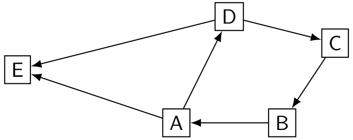

```
matrice = [[0, 0, 0, 1, 1],
           [1, 0, 0, 0, 0],
           [0, 1, 0, 0, 0],
           [0, 0, 1, 0, 1],
           [0, 0, 0, 0, 0]]

```

Code 1 – Matrice d'adjacence



On peut utiliser des tableaux en place des ensembles.

Correction

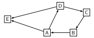



```
dico = {"A": {"D", "E"},
        "B": {"A"},
        "C": {"B"},
        "D": {"C", "E"},
        "E": set()}

```

Code 2 – Dictionnaire d'adjacence

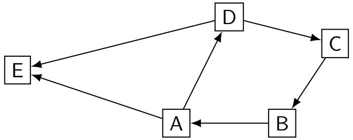
2021-04-28

Correction



A → D → C → B → A

Correction



$$A \to D \to C \to B \to A$$

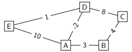


Figure – graphe non orienté pondéré

pondération peut être négative

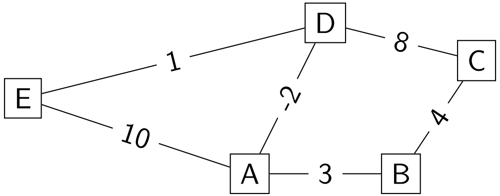
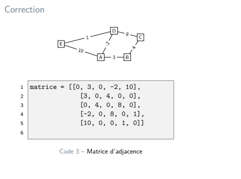


Figure – graphe non orienté pondéré

Activité 2 :
1. Établir la matrice d'adjacence du graphe figure 3.
2. Établir le dictionnaire d'adjacence du graphe figure 3.

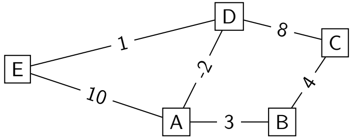
Activité 2 :

1. Établir la matrice d'adjacence du graphe figure 3.
2. Établir le dictionnaire d'adjacence du graphe figure 3.



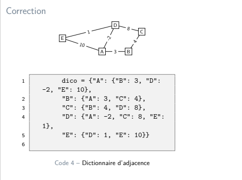
La matrice est symétrique

Correction



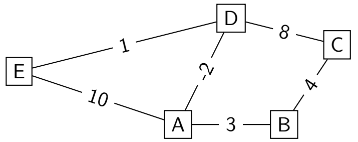

```
matrice = [[0, 3, 0, -2, 10],
           [3, 0, 4, 0, 0],
           [0, 4, 0, 8, 0],
           [-2, 0, 8, 0, 1],
           [10, 0, 0, 1, 0]]

```

Code 3 – Matrice d'adjacence



Dictionnaire de dictionnaires

Correction




```
        dico = {"A": {"B": 3, "D": -2, "E": 10},
        "B": {"A": 3, "C": 4},
        "C": {"B": 4, "D": 8},
        "D": {"A": -2, "C": 8, "E": 1},
        "E": {"D": 1, "E": 10}}
```

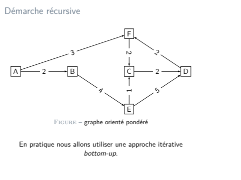
Code 4 – Dictionnaire d'adjacence

Algorithme de Bellman-Ford : Fin des années 50

La distance pour atteindre chaque nœud correspond à la distance pour atteindre son prédécesseur à laquelle on ajoute le poids de l'arête les séparant.

Algorithme de Bellman-Ford : Fin des années 50

La distance pour atteindre chaque nœud correspond à la distance pour atteindre son prédécesseur à laquelle on ajoute le poids de l'arête les séparant.



1. Bellman « père de l'approche dynamique »

Remarque

2. Dans le graphe figure 4 les pondérations représentent un nombre de routeurs traversés pour atteindre le nœud (routeur) suivant.

Démarche récursive

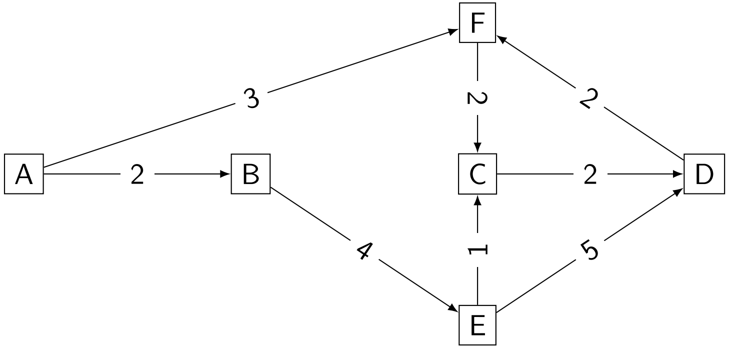


FIGURE – graphe orienté pondéré

En pratique nous allons utiliser une approche itérative *bottom-up*.

peut servir pour retrouver chemin, distance mini...

On obtient un tableau contenant la distance minimale entre le routeur de départ et chaque autre routeur.

Pour chaque routeur

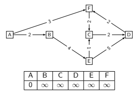
On obtient un tableau contenant la distance minimale entre le routeur de départ et chaque autre routeur.

1 Créer un tableau des distances entre A et les routeurs (A inclus), initialisées à l'infini.
2 Dans le tableau modifier la distance vers A à 0.
3
4 Tant que (le nombre d'itérations) < (nombre de routeurs)
5 Pour chaque arc du graphe
6 Si (la distance du routeur) > (la distance de son prédécesseur + poids de l'arc entre les deux routeurs)
7 La distance du routeur est remplacée par cette nouvelle valeur
8
Code 5 – Algorithme de Bellman Ford

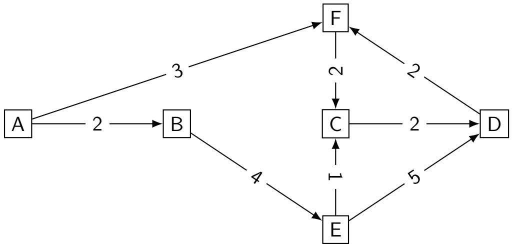
1. ligne 2 de l'algo : en effet distance de A à A = 0
2. ligne 4 : on fait autant d'itérations qu'il y a de routeurs = pire des cas → on pourra améliorer


```
1 Créer un tableau des distances entre A et les routeurs (A inclus
      ), initialisées à l'infini.
2 Dans le tableau modifier la distance vers A à 0.
3
4 Tant que (le nombre d'itérations) < (nombre de routeurs)
5     Pour chaque arc du graphe
6         Si (la distance du routeur) > (la distance de son prédé
      cesseur + poids de l'arc entre les deux routeurs)
7             La distance du routeur est remplacée par cette
      nouvelle valeur
8
```

Code 5 – Algorithme de Bellman Ford

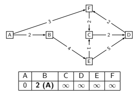


Initialisation



A	B	C	D	E	F
0	∞	∞	∞	∞	∞

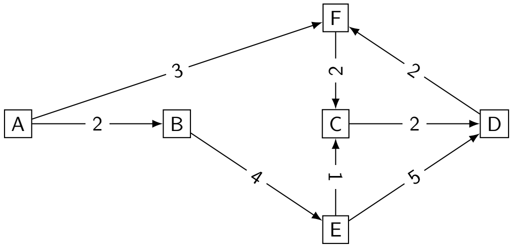
Première itération



La distance calculée pour atteindre B correspond à la distance (du tableau) de son prédécesseur (A) augmentée du poids de l'arc A-B :

$$0 + 2 = 2 < \infty$$

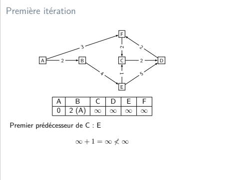
Première itération



A	B	C	D	E	F
0	**2 (A)**	∞	∞	∞	∞

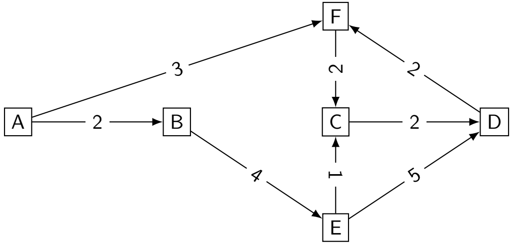
Activité 3 : Continuer de dérouler l'algorithme sur le graphe figure 4.

Activité 3 : Continuer de dérouler l'algorithme sur le graphe figure 4.



On est toujours dans la première itération ; on vérifie chaque arc.

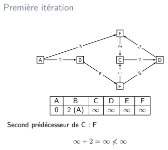
Première itération



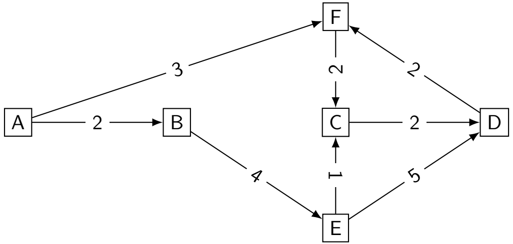
A	B	C	D	E	F
0	2 (A)	∞	∞	∞	∞

Premier prédécesseur de C : E

$$\infty + 1 = \infty \nless \infty$$



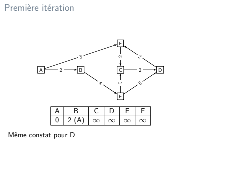
Première itération



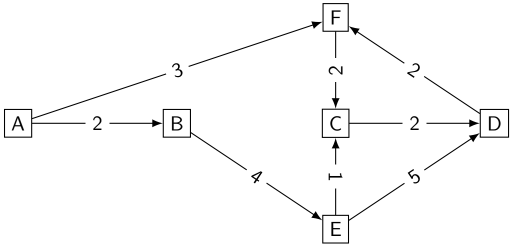
A	B	C	D	E	F
0	2 (A)	∞	∞	∞	∞

Second prédécesseur de C : F

$$\infty + 2 = \infty \nless \infty$$

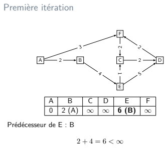


Première itération

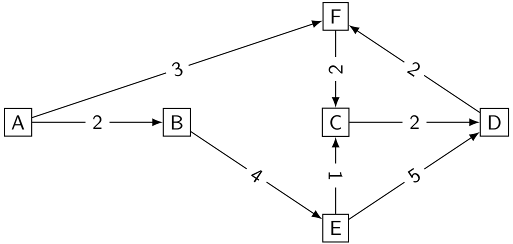


A	B	C	D	E	F
0	2 (A)	∞	∞	∞	∞

Même constat pour D



Première itération

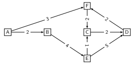


A	B	C	D	E	F
0	2 (A)	∞	∞	**6 (B)**	∞

Prédécesseur de E : B

$$2 + 4 = 6 < \infty$$

Première itération

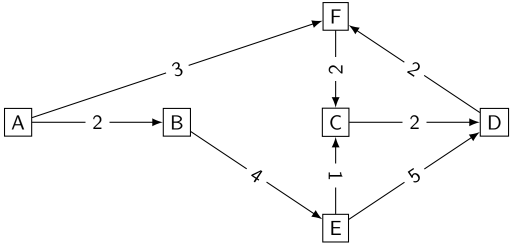


A	B	C	D	E	F
0	2 (A)	∞	∞	6 (B)	**3 (A)**

Premier prédécesseur de F : A

$0+3=3<\infty$

Première itération



A	B	C	D	E	F
0	2 (A)	∞	∞	6 (B)	**3 (A)**

Premier prédécesseur de F : A

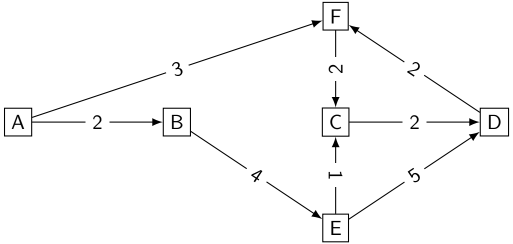
$$0+3=3<\infty$$

Première itération

Second prédécesseur de F : D

$\infty + 2 = \infty \not< 3$

Première itération

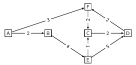


A	B	C	D	E	F
0	2 (A)	∞	∞	6 (B)	3 (A)

Second prédécesseur de F : D

$$\infty + 2 = \infty \not< 3$$

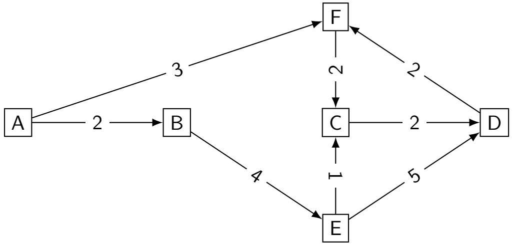
Première itération



A	B	C	D	E	F
0	2 (A)	∞	∞	6 (B)	3 (A)

Fin de la première itération

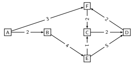
Première itération



A	B	C	D	E	F
0	2 (A)	∞	∞	6 (B)	3 (A)

Fin de la première itération

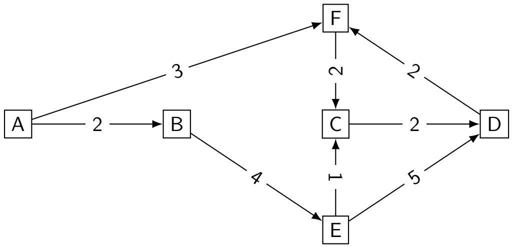
Deuxième itération



A	B	C	D	E	F
0	2 (A)	∞	∞	6 (B)	3 (A)
0	2 (A)	∞	∞	6 (B)	3 (A)

Pas de modification pour A et B

Deuxième itération



A	B	C	D	E	F
0	2 (A)	∞	∞	6 (B)	3 (A)
0	2 (A)	∞	∞	6 (B)	3 (A)

Pas de modification pour A et B

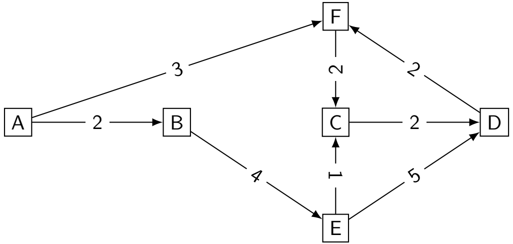
Deuxième itération

A	B	C	D	E	F
0	2 (A)	∞	∞	6 (B)	3 (A)
0	2 (A)	**7 (E)**	∞	6 (B)	3 (A)

Premier prédécesseur de C : E

$$6 + 1 = 7 < \infty$$

Deuxième itération

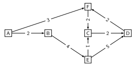


A	B	C	D	E	F
0	2 (A)	∞	∞	6 (B)	3 (A)
0	2 (A)	**7 (E)**	∞	6 (B)	3 (A)

Premier prédécesseur de C : E

$$6 + 1 = 7 < \infty$$

Deuxième itération

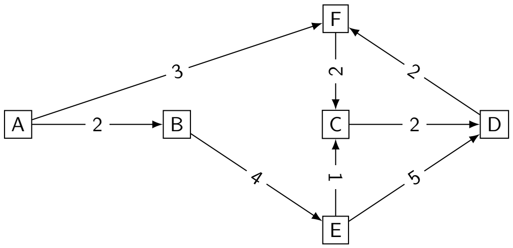


A	B	C	D	E	F
0	2 (A)	∞	∞	6 (B)	3 (A)
0	2 (A)	**5 (F)**	∞	6 (B)	3 (A)

Second prédécesseur de C : F

$$3 + 2 = 5 < 7$$

Deuxième itération

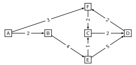


A	B	C	D	E	F
0	2 (A)	∞	∞	6 (B)	3 (A)
0	2 (A)	**5 (F)**	∞	6 (B)	3 (A)

Second prédécesseur de C : F

$$3 + 2 = 5 < 7$$

Deuxième itération

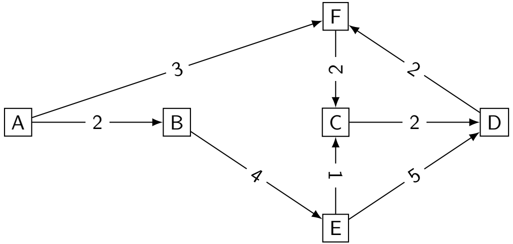


A	B	C	D	E	F
0	2 (A)	∞	∞	6 (B)	3 (A)
0	2 (A)	5 (F)	**7 (C)**	6 (B)	3 (A)

Premier prédécesseur de D : C

$$5 + 2 = 7 < \infty$$

Deuxième itération

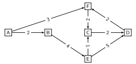


A	B	C	D	E	F
0	2 (A)	∞	∞	6 (B)	3 (A)
0	2 (A)	5 (F)	**7 (C)**	6 (B)	3 (A)

Premier prédécesseur de D : C

$$5 + 2 = 7 < \infty$$

Deuxième itération

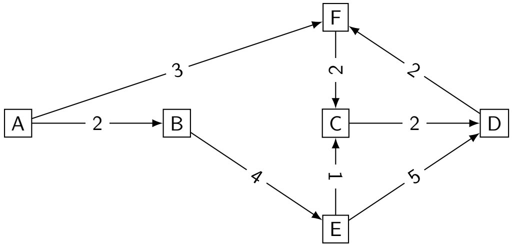


A	B	C	D	E	F
0	2 (A)	∞	∞	6 (B)	3 (A)
0	2 (A)	5 (F)	7 (C)	6 (B)	3 (A)

Second prédécesseur de D : E

$$6 + 5 = 11 \nless 7$$

Deuxième itération

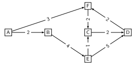


A	B	C	D	E	F
0	2 (A)	∞	∞	6 (B)	3 (A)
0	2 (A)	5 (F)	7 (C)	6 (B)	3 (A)

Second prédécesseur de D : E

$$6 + 5 = 11 \nless 7$$

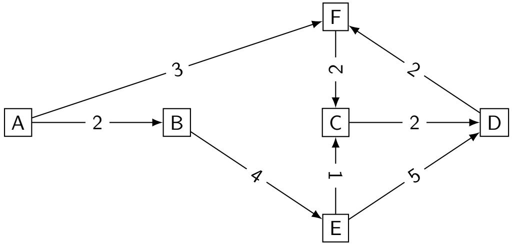
Deuxième itération



A	B	C	D	E	F
0	2 (A)	∞	∞	6 (B)	3 (A)
0	2 (A)	5 (F)	7 (C)	6 (B)	3 (A)

Pas de changement pour E et F

Deuxième itération

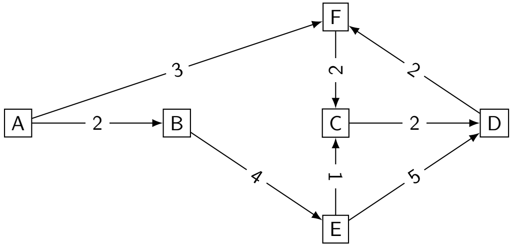


A	B	C	D	E	F
0	2 (A)	∞	∞	6 (B)	3 (A)
0	2 (A)	5 (F)	7 (C)	6 (B)	3 (A)

Pas de changement pour E et F

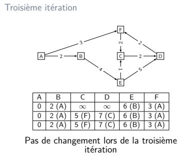
Deuxième itération
Fin de la deuxième itération

Deuxième itération



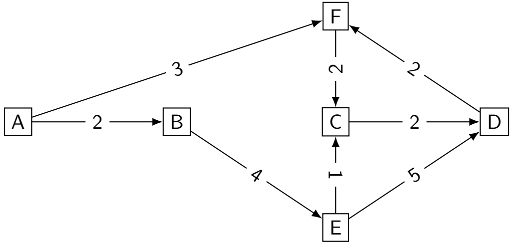
A	B	C	D	E	F
0	2 (A)	∞	∞	6 (B)	3 (A)
0	2 (A)	5 (F)	7 (C)	6 (B)	3 (A)

Fin de la deuxième itération



1. amélioration de l'algorithme : on peut s'arrêter là
2. On peut retracer le chemin en partant de la fin : D → C → F → A

Troisième itération



A	B	C	D	E	F
0	2 (A)	∞	∞	6 (B)	3 (A)
0	2 (A)	5 (F)	7 (C)	6 (B)	3 (A)
0	2 (A)	5 (F)	7 (C)	6 (B)	3 (A)

Pas de changement lors de la troisième itération

▶ du nombre de sommets (notée S) : on visite chaque sommet (ligne 4) ;
1 Tant que (le nombre d'itérations)
< (nombre de routeurs)
2

ligne 4 = on peut faire une amélioration : si pas de modification pendant l'itération, on peut s'arrêter

- du nombre de sommets (notée S) : on visite chaque sommet (ligne 4) ;

```
1 Tant que (le nombre d'itérations)
     < (nombre de routeurs)
2
```

▶ du nombre de sommets (notée S) : on visite chaque sommet (ligne 4) ;
Tant que (le nombre d'itérations) < (nombre de routeurs)
▶ du nombre d'arcs (notée A) : pour chaque sommet on regarde tous les arcs du graphe (ligne 5).
Pour chaque arc du graphe

ligne 4 = on peut faire une amélioration : si pas de modification pendant l'itération, on peut s'arrêter

- du nombre de sommets (notée S) : on visite chaque sommet (ligne 4) ;

```
Tant que (le nombre d'itérations)
    < (nombre de routeurs)

```

- du nombre d'arcs (notée A) : pour chaque sommet on regarde tous les arcs du graphe (ligne 5).

```
Pour chaque arc du graphe

```

Complexité de l'algorithme de Bellman Ford

$O(S.A)$

Complexité de l'algorithme de Bellman Ford

$$O(S.A)$$

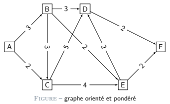
1. peut se contenter de renvoyer la distance départ/arrivée (et le chemin)
2. graphe non orienté possible
3. parfois appelé Moore-Dijkstra

Edsger Dijkstra

principe : construire un sous-graphe en ajoutant à chaque itération un sommet de distance minimale.

Edsger Dijkstra

principe : construire un sous-graphe en ajoutant à chaque itération un sommet de distance minimale.



Remarque

Les poids de chaque arc représentent le coût de chaque connexion.

coût 5 $\rightarrow$ 20Mbit/s
coût 2 $\rightarrow$ 50Mbit/s

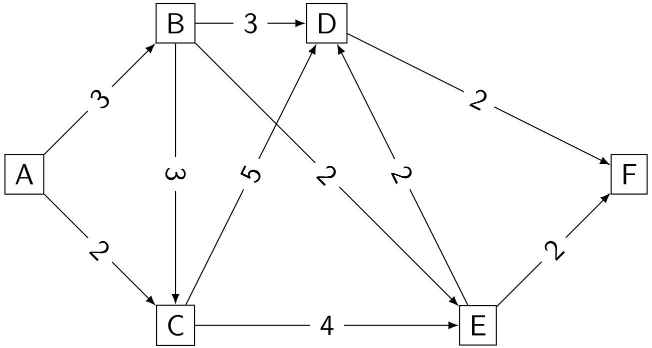


FIGURE – graphe orienté et pondéré

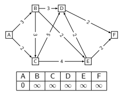
```
1 Créer un tableau des distances entre A et les routeurs (A inclus
     ), initialisées à l'infini.
2 Dans le tableau modifier la distance vers A à 0.
3
4 Tant qu'il reste des routeurs non sélectionnés
5     Parmi les routeurs non−sélectionnés, choisir le routeur (noté
       S) ayant la plus petite distance.
6     Pour chaque routeur adjacent à S (noté V) et non déjà sé
      lectionné:
7         Si (la distance de V) > (la distance de S + poids S−V)
8             La distance de V est remplacée par cette nouvelle
      valeur
9
```

Code 6 – Algorithme de Dijkstra

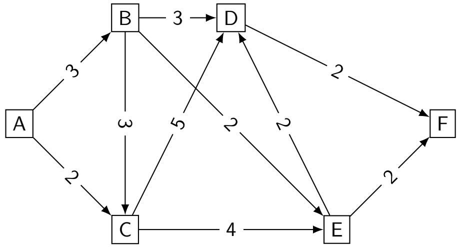
1. S = suivant ; V = voisin
2. ligne 7 : on compare encore la distance déjà enregistrée à distance du voisin + arc


```
1 Créer un tableau des distances entre A et les routeurs (A inclus
     ), initialisées à l'infini.
2 Dans le tableau modifier la distance vers A à 0.
3
4 Tant qu'il reste des routeurs non sélectionnés
5     Parmi les routeurs non−sélectionnés, choisir le routeur (noté
       S) ayant la plus petite distance.
6     Pour chaque routeur adjacent à S (noté V) et non déjà sé
      lectionné:
7         Si (la distance de V) > (la distance de S + poids S−V)
8             La distance de V est remplacée par cette nouvelle
      valeur
9
```

Code 6 – Algorithme de Dijkstra

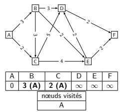


Initialisation

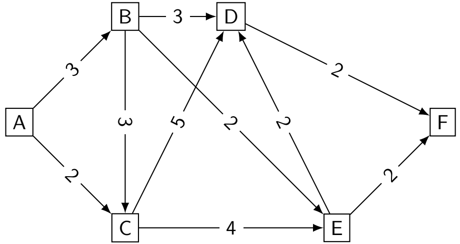


A	B	C	D	E	F
0	∞	∞	∞	∞	∞

Sélection de A



Sélection de A



A	B	C	D	E	F
0	**3 (A)**	**2 (A)**	∞	∞	∞

nœuds visités
A

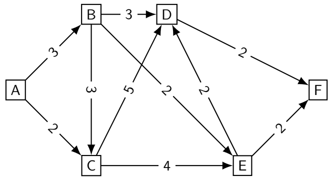
Activité 4 : Continuer de dérouler l'algorithme sur le graphe figure 5.

Activité 4 : Continuer de dérouler l'algorithme sur le graphe figure 5.

Correction
On sélectionne le nœud non encore visité et avec la plus petite distance : C.

cellule grisée = nœud déjà visité, on a déjà sa route la + courte

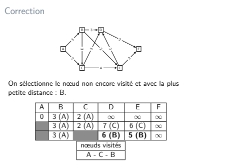
Correction



On sélectionne le nœud non encore visité et avec la plus petite distance : C.

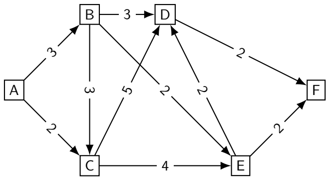
A	B	C	D	E	F
0	3 (A)	2 (A)	∞	∞	∞
	3 (A)	2 (A)	**7 (C)**	**6 (C)**	∞

nœuds visités
A - C



C n'est pas regardé : on a déjà trouvé la + petite distance pour lui = il a déjà été visité

Correction



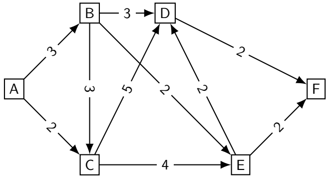
On sélectionne le nœud non encore visité et avec la plus petite distance : B.

A	B	C	D	E	F
0	3 (A)	2 (A)	∞	∞	∞
	3 (A)	2 (A)	7 (C)	6 (C)	∞
	3 (A)		**6 (B)**	**5 (B)**	∞

nœuds visités
A - C - B

Correction
On sélectionne le nœud non encore visité et avec la plus petite distance : E.
nœuds visités
A - C - B - E

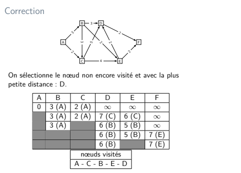
Correction



On sélectionne le nœud non encore visité et avec la plus petite distance : E.

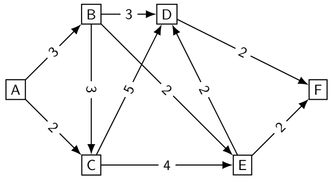
A	B	C	D	E	F
0	3 (A)	2 (A)	∞	∞	∞
	3 (A)	2 (A)	7 (C)	6 (C)	∞
	3 (A)		6 (B)	5 (B)	∞
			6 (B)	5 (B)	**7 (E)**

nœuds visités
A - C - B - E



pas de modification

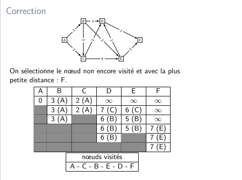
Correction



On sélectionne le nœud non encore visité et avec la plus petite distance : D.

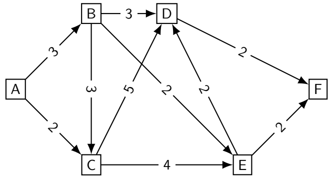
A	B	C	D	E	F
0	3 (A)	2 (A)	∞	∞	∞
	3 (A)	2 (A)	7 (C)	6 (C)	∞
	3 (A)		6 (B)	5 (B)	∞
			6 (B)	5 (B)	7 (E)
			6 (B)		7 (E)

nœuds visités
A - C - B - E - D



pas de modification

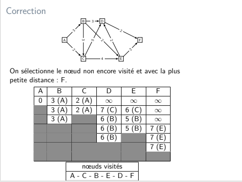
Correction



On sélectionne le nœud non encore visité et avec la plus petite distance : F.

A	B	C	D	E	F
0	3 (A)	2 (A)	∞	∞	∞
	3 (A)	2 (A)	7 (C)	6 (C)	∞
	3 (A)		6 (B)	5 (B)	∞
			6 (B)	5 (B)	7 (E)
			6 (B)		7 (E)
					7 (E)

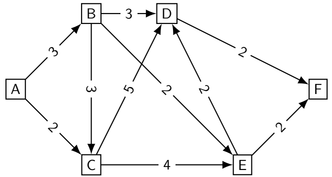
nœuds visités
A - C - B - E - D - F



tous les nœuds ont été visités = fin de l'algorithme. On peut reconstruire le chemin.

pour F : F → E → B → A

Correction



On sélectionne le nœud non encore visité et avec la plus petite distance : F.

A	B	C	D	E	F
0	3 (A)	2 (A)	∞	∞	∞
	3 (A)	2 (A)	7 (C)	6 (C)	∞
	3 (A)		6 (B)	5 (B)	∞
			6 (B)	5 (B)	7 (E)
			6 (B)		7 (E)
					7 (E)

nœuds visités
A - C - B - E - D - F

- La complexité dépend du nombre de sommets S et du nombre d'arcs A.

1. hors programme
2. utilisation de tas par exemple

- La complexité dépend du nombre de sommets S et du nombre d'arcs A.

1. hors programme
2. utilisation de tas par exemple

- La complexité dépend du nombre de sommets S et du nombre d'arcs A.
- Le point clé de l'algorithme tient dans la recherche de la distance minimale.

Parmi les routeurs non−sélectionnés, choisir le routeur (noté S) ayant la plus petite distance.

- La complexité dépend du nombre de sommets S et du nombre d'arcs A.
- Le point clé de l'algorithme tient dans la recherche de la distance minimale.

```
Parmi les routeurs non−sélectionnés, choisir le routeur (
    noté S) ayant la plus petite distance.

```

Complexité de l'algorithme de Dijkstra

$O((A+S)\times \log S)$

Complexité de l'algorithme de Dijkstra

$$O((A+S)\times \log S)$$