Chapitre III :

Le plus court chemin

INTRODUCTION

En théorie des graphes, le **problème de plus court chemin** est le problème algorithmique qui consiste à trouver un chemin d'un sommet à un autre de façon que la somme des poids des arcs de ce chemin soit minimale. Pour calculer un plus court chemin, il existe de nombreux algorithmes, selon la nature des valeurs et des contraintes supplémentaires qui peuvent être imposées. Dans de nombreux cas, il existe des algorithmes de complexité en temps polynomiale, comme **l'algorithme de Dijkstra** dans des graphes avec poids positifs.

Pour accéder à un sommet u depuis s, plusieurs chaînes/chemins optimaux peuvent exister. La chaîne optimale (le chemin optimal) n'est pas forcément celle (celui) d'un plus petit nombre d'arêtes (arcs). Si le prédécesseur de u dans une chaîne optimale (un chemin optimal) est un voisin (entrant) v de u, alors la sous-chaîne (le sous-chemin) de s à v est optimal.e aussi.

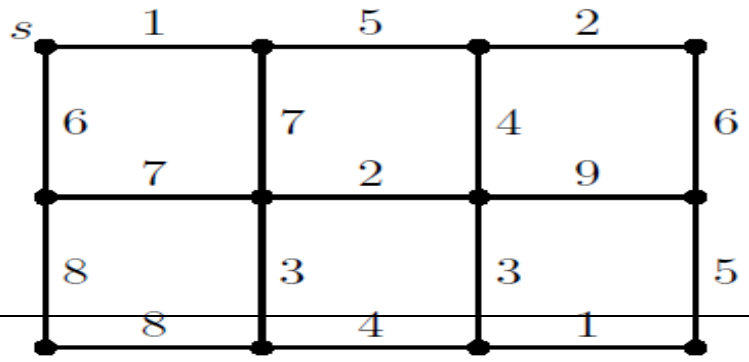
On a donc :

$$\text{Dist}(s; u) = \min_{vu \in E(G)} \{ \text{Dist}(s; v) + W(vu) \}:$$

Exemple : Calcul de distances

Entrée : Un graphe (orienté ou pas) G, avec une pondération des arêtes **W : E(G)→ R+**; un sommet de départ s.
Sortie : La distance dist(s; v) pour tout sommet v
Rappel : la distance entre deux sommets est le poids minimum d'une chaîne (d'un chemin) les reliant.

Graphe G avec les arêtes pondérées, calculer la distance minimale ; sommet de départ est s.



Pour accéder à un sommet u depuis s, plusieurs chaînes/chemins optimaux peuvent exister.

I. Algorithme de Dijkstra

Edgser Wybe Dijkstra (1930-2002) a proposé en 1959 un algorithme qui permet de calculer le plus court chemin entre un sommet particulier et tous les autres. Les sommets sont visités dans l'ordre croissant de leur distance à partir du sommet de départ s. A chaque étape, la distance d'un sommet est marquée définitive (le sommet est colorié noir) ;
Les distances d'autres sommets sont éventuellement mises à jour. Si pour un sommet *u* la valeur de *d[u]* est différente de ∞, alors il existe au moins une chaîne de *s à u* ; parmi les chaînes qui relient *s et u*, dans celle de longueur (poids total) *d[u]* le prédécesseur de *u* est mémorisé comme *père[u].*

Le résultat est une arborescence, c'est-à-dire un arbre avec un sommet particulier appelé racine. Numérotons les sommets du graphe *G = (V,E)* de *1 à n*. Supposons que l'on s'intéresse aux chemins partant du sommet 1. On construit un vecteur $\lambda = (\lambda(1);\lambda(2); \ldots ;\lambda(n))$ ayant n composantes tel que $\lambda(j)$ soit égal à la longueur du plus court chemin allant de 1 au sommet j . On initialise ce vecteur à $C1j$, c'est-à-dire à la première ligne de la matrice des coûts du graphe, définie comme indiqué ci-dessous :

où δ (i, j) > 0 est le poids de l'arc (i, j). On construit un autre vecteur p pour mémoriser le chemin pour aller du sommet 1 au sommet voulu. La valeur p(i) donne le sommet qui précède i dans le chemin.

$$c_{ij} = \begin{cases} 0 & \text{si } i = j \\ \infty & \text{si } i \neq j \text{ et } (i,j) \notin E \\ \delta(i,j) & \text{si } i \neq j \text{ et } (i,j) \in E \end{cases}$$

On considère ensuite deux ensembles de sommets, *S* initialisé à {1} et *T* initialisé à *{2,3, . . . ,n}.* À chaque pas de l'algorithme, on ajoute à *S* un sommet jusqu'à ce que *S* =*V* de telle sorte que le vecteur λ donne à chaque étape la longueur minimale des chemins de 1 aux sommets de *S*.

I.1. Principe de l'algorithme de Dijkstra

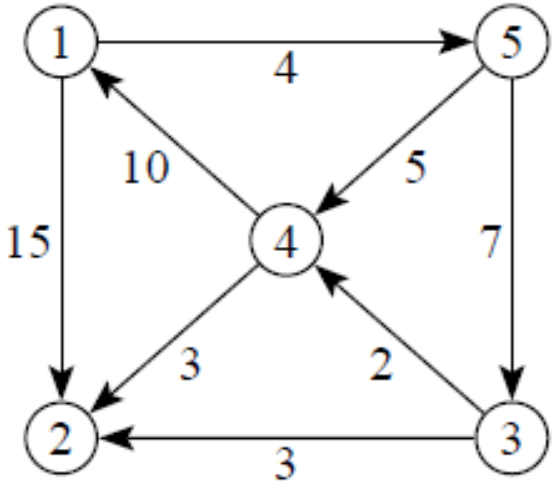
On suppose que le sommet de départ (qui sera la racine de l'arborescence) est le sommet numéroté 1. Notons qu'on peut toujours renuméroter les sommets pour que ce soit le cas.

Initialisation :

$\lambda(j) = c_{1j}$ et $p(j) =$ NIL, pour $1 \leq j \leq n$
Pour $2 \leq j \leq n$ **faire**
Si $c_{1j} < \infty$ **alors** $p(j) = 1$.
$S = 1$; $T = \{2,3, \ldots ,n\}$.

Itérations :

Tant que *T* n'est pas vide **faire**
Choisir *i* dans *T* tel que $\lambda(i)$ est minimum
Retirer *i* de *T* et l'ajouter à *S*
Pour chaque successeur *j* de *i*, avec *j* dans *T*, **faire**
Si $\lambda(j) > \lambda(i) + \delta(i, j)$ **alors**
$\lambda(j) = \lambda(i) + \delta(i, j)$
$p(j) = i$



Initialisations
S = {1} ; ***T*** = {2,3,4,5} ; **λ** = (0,15,∞,∞,4) ; *p* = (NIL,1,NIL,NIL,1)
1ère itération
i = 5 car **λ (5)** = min(15,∞,∞,4) = 4
S = {1,5} ; *T* = {2,3,4} les successeurs de 5 dans *T* sont 3 et 4
λ (3) prend la nouvelle valeur min(∞; λ (5) + δ (5;3)) = min(λ ;4+7) = 11 ; *p*(3) = 5
λ (4) prend la nouvelle valeur min(∞; λ (5) + δ (5;4)) = 9 ; *p*(4) = 5
d'où les nouveaux vecteurs λ = (0,15,11,9,4) et *p* = (NIL,1,5,5,1)

2ème itération

i = 4 ; **λ (4)** = 9
S = {1,5,4} ; ***T*** = {2,3} le seul successeur de 4 dans *T* est 2
λ (2) prend la nouvelle valeur min(15;λ (4)+δ (4;2)) = min(15;9+3) = 12 ; *p*(2) = 4
d'où les nouveaux vecteurs λ = (0,12,11,9,4) et *p* = (NIL,4,5,5,1)

3ème itération

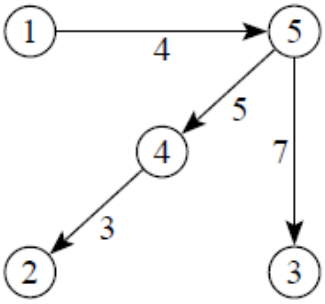
i = 3 ; **λ (3)** = 11
S = {1,5,4,3} ; ***T*** = {2} le seul successeur de 3 dans *T* est 2
λ (2) garde sa valeur car min(12;λ (3)+δ (3;2)) = min(12;11+3) = 12
d'où les vecteurs inchangés λ = (0,12,11,9,4) et *p* = (NIL,4,5,5,1)

4ème itération

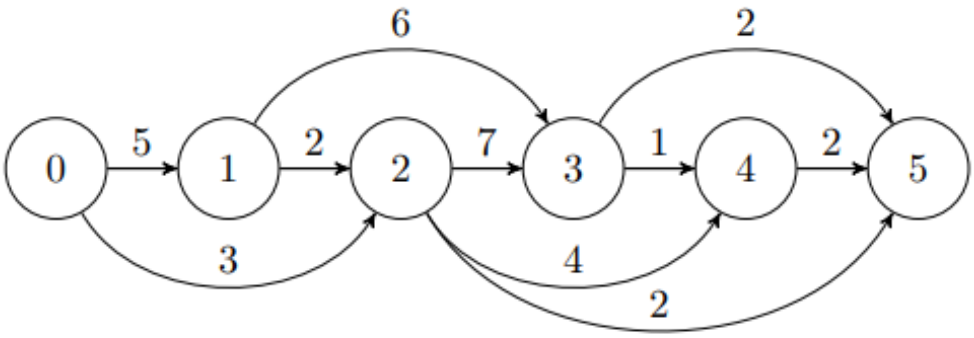
i = 2 ; **λ (2)** = 12
S = {1,5,4,3,2} ; ***T* = { }**
λ = (0,12,11,9,4)
p = (NIL,4,5,5,1)

L'algorithme se termine, **car *T* = { }.**
On peut lire les coûts des chemins les plus courts dans λ et les chemins eux-mêmes grâce à la vectrice *p*. Par exemple, le chemin minimal de 1 à 4 est de coût 9, car λ (4) = 9. C'est le chemin 1–5–4, car *p*(4) = 5 et *p*(5) = 1.

Voici la réponse sous forme d'arborescence :

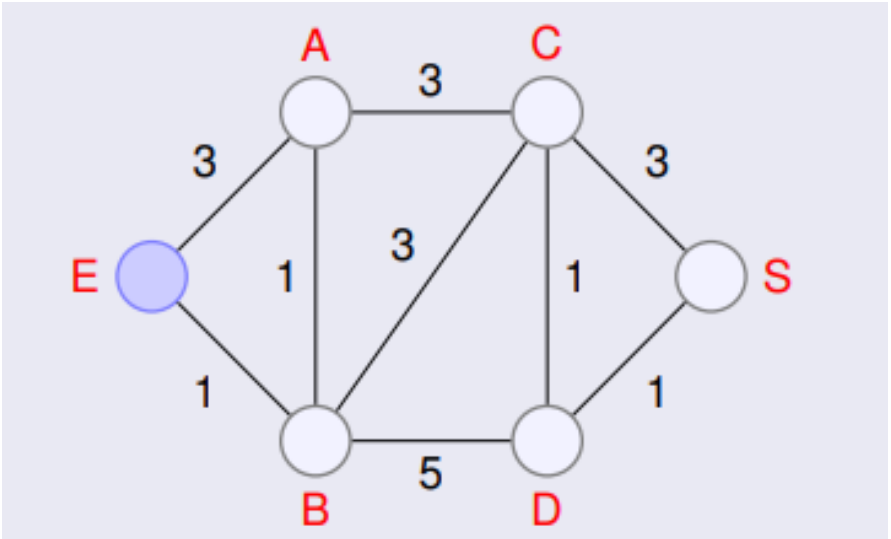


<u>Exercice d'application :</u>

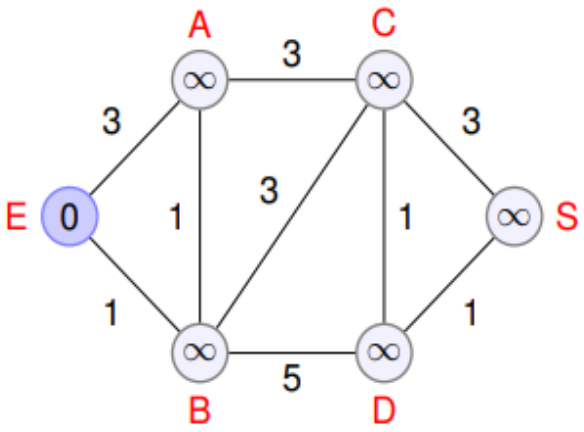


Donnez le résultat de plus court chemin avec l'algorithme de DIJKSTRA

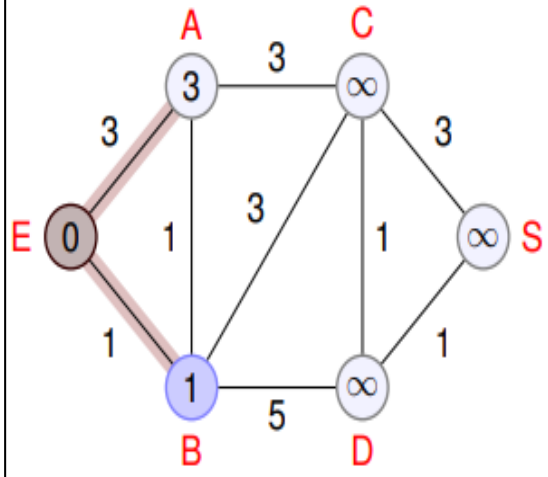
Dans un graphe non orienté cherchons les plus courts chemins d'origine E du graphe ci-dessous :



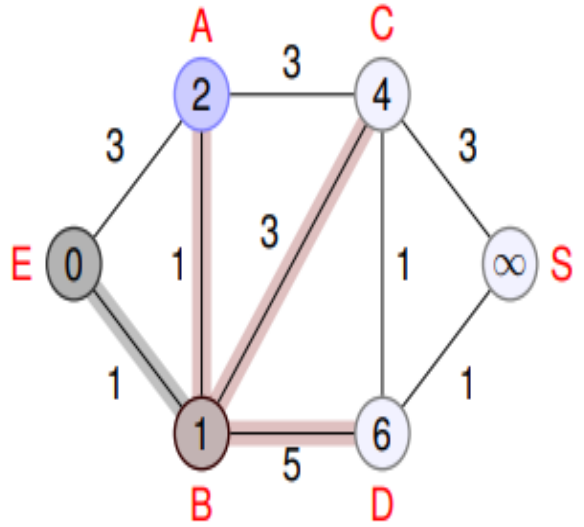
La Solution



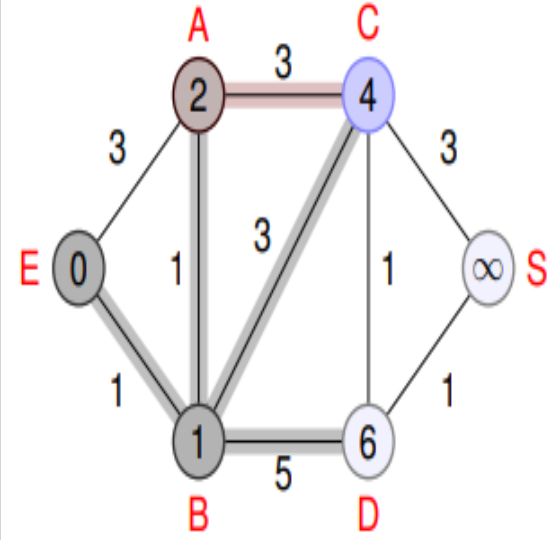
E	A	B	C	D	S
0	∞	∞	∞	∞	∞
•					
•					
•					
•					
•					



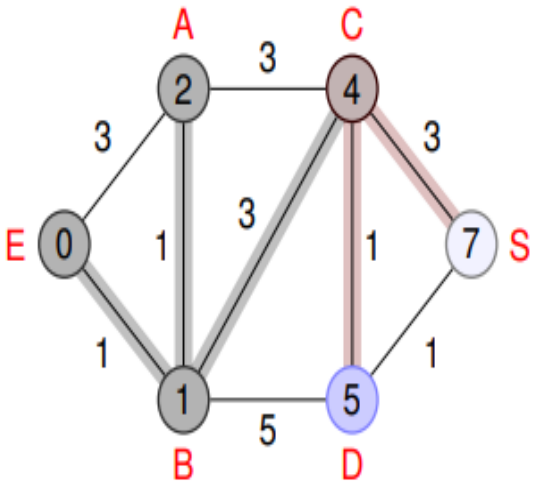
E	A	B	C	D	S
0	∞	∞	∞	∞	∞
•	3_E	1_E	∞	∞	∞
•		•			
•		•			
•		•			
•		•			



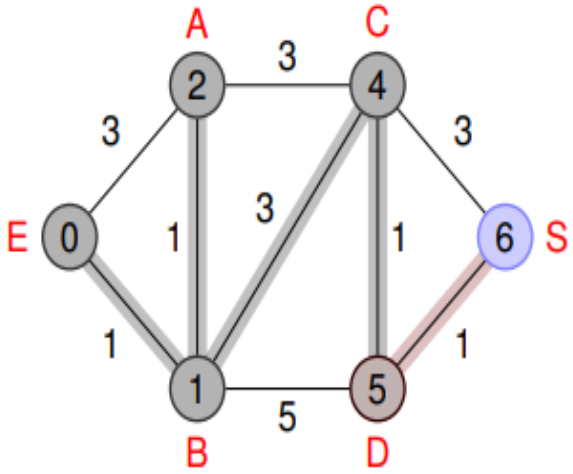
E	A	B	C	D	S
0	∞	∞	∞	∞	∞
•	3_E	1_E	∞	∞	∞
•	2_B	•	4_B	6_B	∞
•	•	•			
•	•	•			
•	•	•			



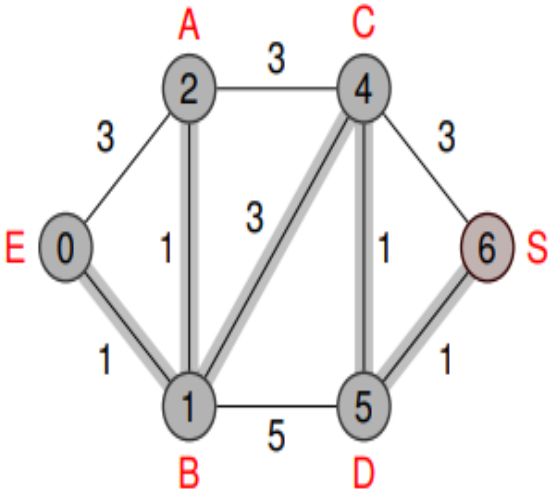
E	A	B	C	D	S
0	∞	∞	∞	∞	∞
•	3_E	1_E	∞	∞	∞
•	2_B	•	4_B	6_B	∞
•	•	•	4_B	6_B	∞
•	•	•	•		
•	•	•	•		



E	A	B	C	D	S
0	∞	∞	∞	∞	∞
•	3_E	1_E	∞	∞	∞
•	2_B	•	4_B	6_B	∞
•	•	•	4_B	6_B	∞
•	•	•	•	5_C	7_C
•	•	•	•	•	



E	A	B	C	D	S
0	∞	∞	∞	∞	∞
•	3_E	1_E	∞	∞	∞
•	2_B	•	4_B	6_B	∞
•	•	•	4_B	6_B	∞
•	•	•	•	5_C	7_C
•	•	•	•	•	6_D



E	A	B	C	D	S
0	∞	∞	∞	∞	∞
•	3_E	1_E	∞	∞	∞
•	2_B	•	4_B	6_B	∞
•	•	•	4_B	6_B	∞
•	•	•	•	5_C	7_C
•	•	•	•	•	6_B

Conclusion :

Les sommets sont visités dans l'ordre de poids minimum de leur distance. Les distances d'autres sommets sont éventuellement mises à jour.

II) Algorithme de Bellman-Ford

L'**algorithme de Bellman-Ford**, aussi appelé **algorithme de Bellman–Ford–Moore**, est un algorithme qui calcule des plus courts chemins depuis un sommet source donné dans un **graphe orienté** pondéré. Il porte le nom de ses inventeurs **Richard Bellman** et **Lester Randolph Ford junior** (publications en 1956 et 1958), et de **Edward Forrest Moore** qui le redécouvrit en 1959. Contrairement à **l'algorithme de Dijkstra**, l'algorithme de Bellman-Ford autorise la présence de certains arcs de poids négatif et permet de détecter l'existence d'un circuit absorbant, c'est-à-dire de poids total strictement négatif, accessible depuis le sommet source.

Principe de l'algorithme :

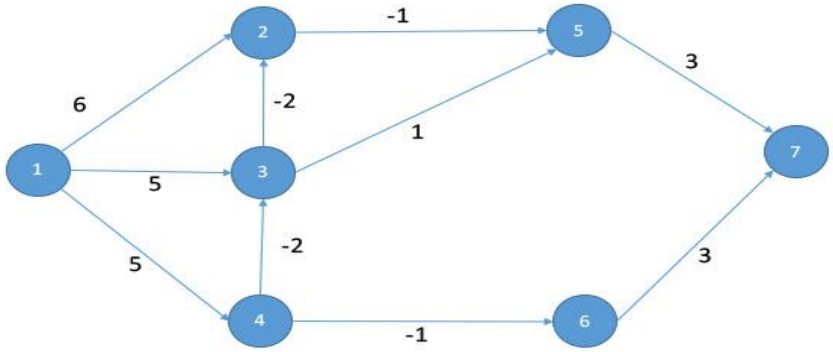
Soit $\mathbf{G = (X, U)}$ un graphe dont les arcs sont munis de longueur réelles quelconques. On cherche les pcch de $\mathbf{i_0}$ à tous les autres sommets du graphe.
Remarque préliminaire : Soit un graphe G présentant sur certains arcs de poids négative et, envisageons de rendre les valuations toutes positives ou nulles en ajoutant à chaque valeur la valeur absolue maximale des valeurs négatives.

1. initialise les distances de la source à tous les sommets en tant qu'infini et la distance à la source elle-même en tant que 0. Créez un tableau **dist[]** de taille **|V|** avec toutes les valeurs infinies sauf **dist[src]** où **src** est le sommet source.
2. Faire **|V|-1** fois où **|V|** est le nombre de sommets dans le graphe les instructions suivantes.
    - Répéter pour chaque arête u-v
        - Si **dist[v]>dist[u]+poids l'arête u-v,** mettez à jour **dist[v]=dist[u]+poind de l'arête u-v**
3. Si **dist[v] > dist [u] + poids de l'arête u-v**, alors "le graphe contient un cycle de pondération négatif"

L'idée de l'étape 3 est que l'étape 2 garantit les distances les plus courtes si le graphe ne contient pas de cycle de pondération négatif. Si nous

parcourons toutes les arêtes une fois de plus et nous obtenons un chemin plus court pour tout sommet, il y a un cycle de pondération négatif.

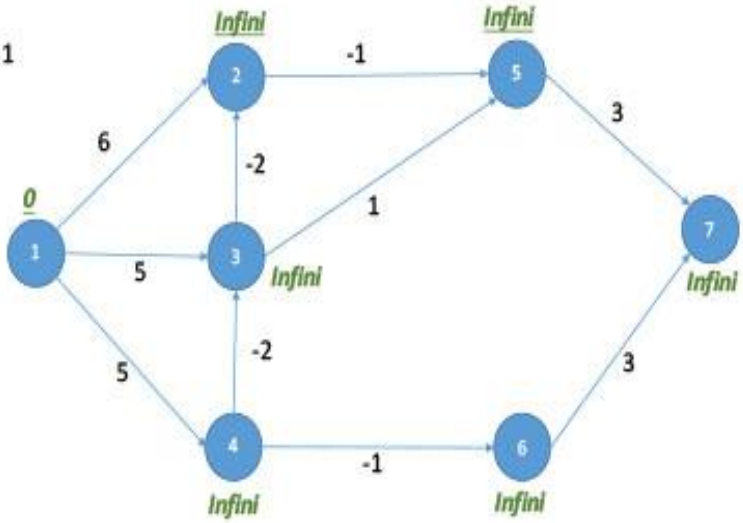
Exemple d'application



L'ordre des arêtes

[(1,2), (1,3), (1,4), (3,2), (2,5), (3,5), (4,3), (4,6), (6,7), (5,7)]

Etape 1

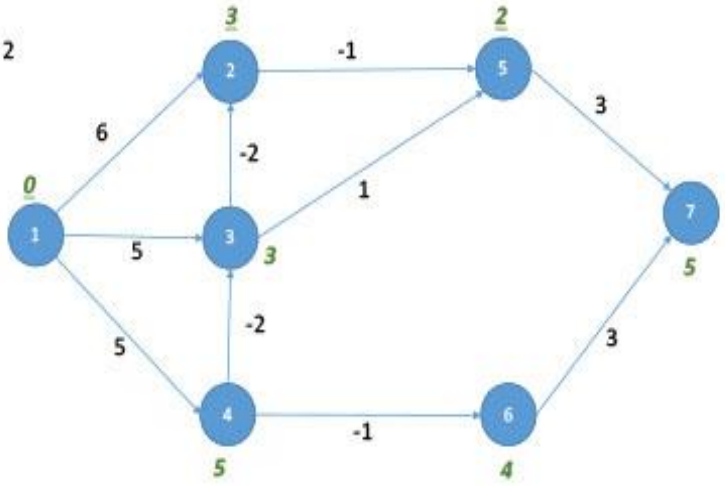


Arête	relaxation (mise à jour)
(1,2)	Distance[2]=6
(1,3)	Distance[3]=5
(1,4)	Distance[4]=5
(3,2)	Distance[2]=3
(2,5)	Distance[5]=2
(3,5)	Rien à faire (2<4)
(4,3)	Distance[3]=3
(4,6)	Distance[6]=4
(6,7)	Distance[7]=7
(5,7)	Distance[7]=5

L'ordre des arêtes

[(1,2), (1,3), (1,4), (3,2), (2,5), (3,5), (4,3), (4,6), (6,7), (5,7)]

Etape 2

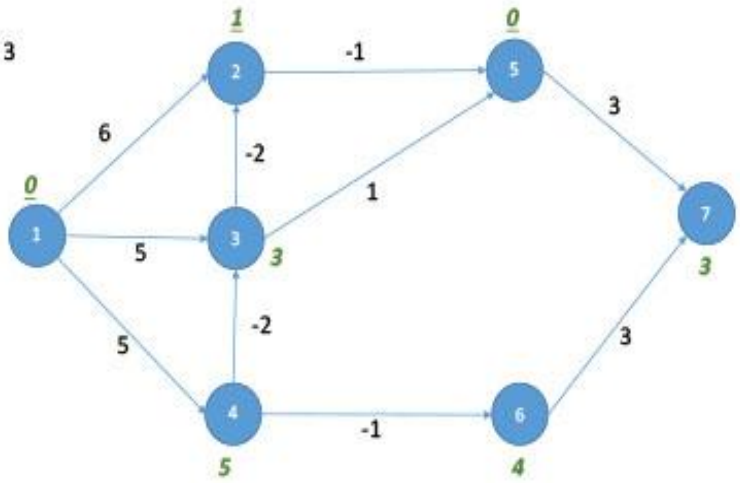


Arête	relaxation (mise à jour)
(1,2)	Rien à faire
(1,3)	Rien à faire
(1,4)	Rien à faire
(3,2)	Distance[2]=1
(2,5)	Distance[5]=0
(3,5)	Rien à faire
(4,3)	Rien à faire
(4,6)	Rien à faire
(6,7)	Rien à faire
(5,7)	Distance[7]=3

L'ordre des arêtes

[(1,2), (1,3), (1,4), (3,2), (2,5), (3,5), (4,3), (4,6), (6,7), (5,7)]

Etape 3



Arête	relaxation (mise à jour)
(1,2)	Rien à faire
(1,3)	Rien à faire
(1,4)	Rien à faire
(3,2)	Rien à faire
(2,5)	Rien à faire
(3,5)	Rien à faire
(4,3)	Rien à faire
(4,6)	Rien à faire
(6,7)	Rien à faire
(5,7)	Rien à faire

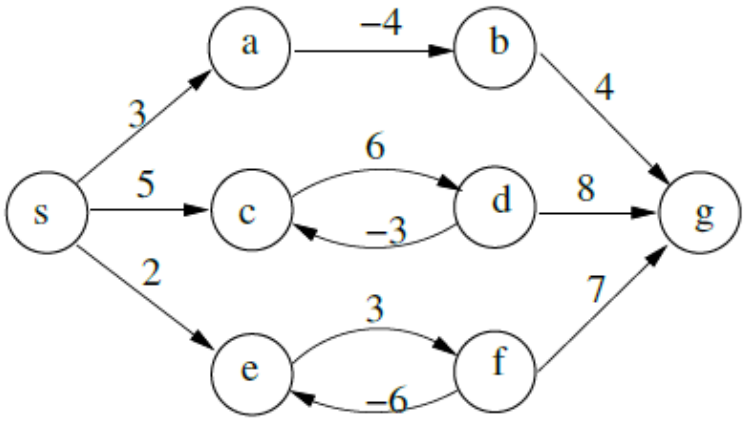
L'ordre des arêtes

[(1,2), (1,3), (1,4), (3,2), (2,5), (3,5), (4,3), (4,6), (6,7), (5,7)]

STOP

Exercice d'application

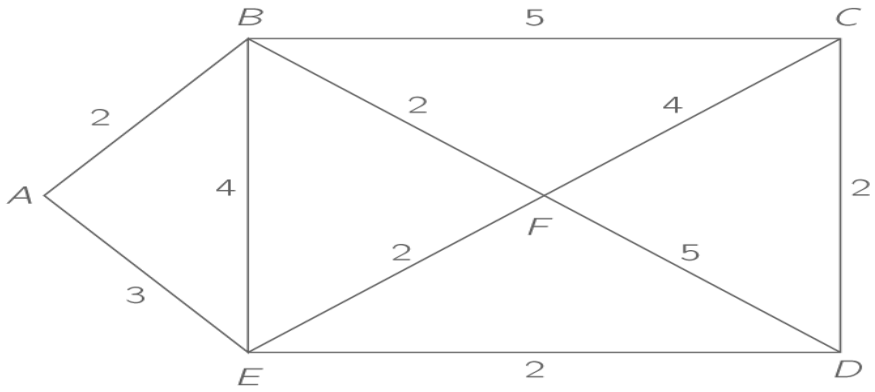
Trouvez le plus court chemin du graphe ci-dessous en appliquant l'algorithme de Bellman-Ford.



TD1-Chapitre III : Plus court chemin

Exercice : QCM

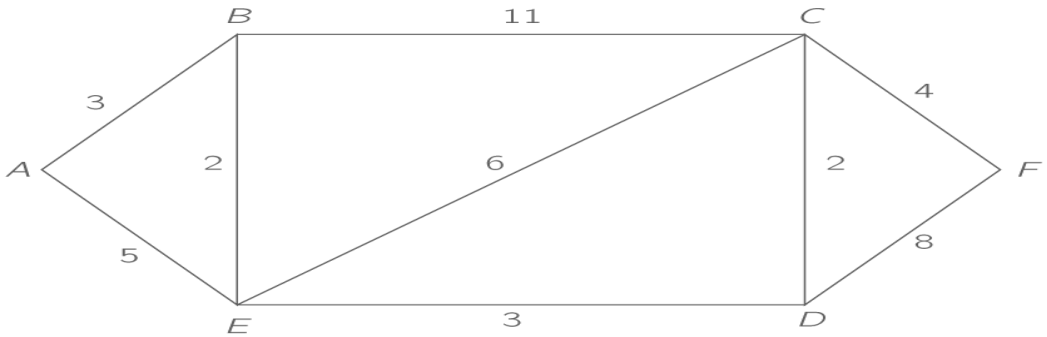
On représente sur le graphe *G* ci-dessous les liaisons routières entre six places (*A*, *B*, *C*, *D*, *E*, *F*) d'un centre-ville. Sur chaque route est indiqué le nombre de feux tricolores présents entre les deux places qu'elle relie.



Un automobiliste souhaite se rendre de *B* à *D* en empruntant le trajet comportant le moins de feux tricolores possible. Quel itinéraire cet automobiliste doit-il emprunter ?

- ☐ F – B – D – E
- ☐ D– E – F – B
- ☐ A – C – B – D
- ☐ B – F – E – D

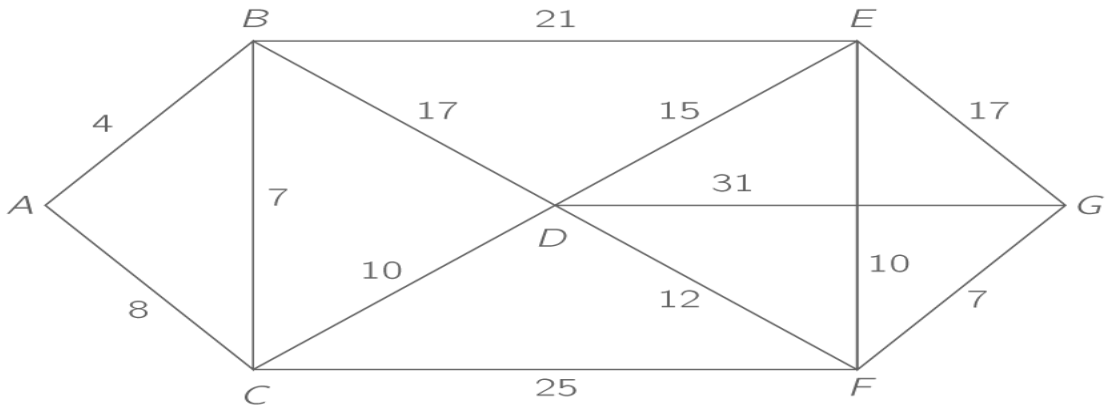
On représente sur le graphe ci-dessous les liaisons routières entre six places (*A*, *B*, *C*, *D*, *E*, *F*) d'un centre-ville. Sur chaque route est indiqué le nombre de feux tricolores présents entre les deux places qu'elle relie.



Un automobiliste souhaite se rendre de *A* à *F* en empruntant le trajet comportant le moins de feux tricolores possible. Quel itinéraire cet automobiliste doit-il emprunter ?

- ☐ A-B-F-E-C-D
- ☐ A-B-C-D-E-F
- ☐ F-C-E-D-B-A
- ☐ A-B-E-D-C-F

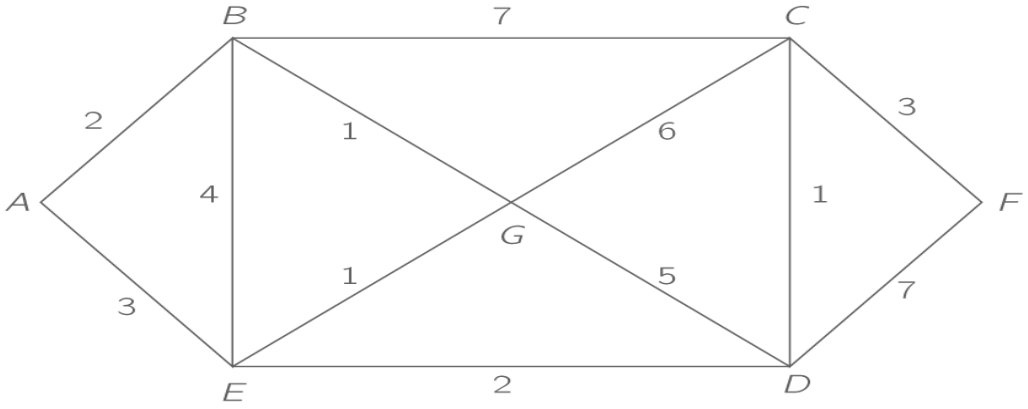
On représente sur le graphe ci-dessous les liaisons ferroviaires entre sept gares (*A*, *B*, *C*, *D*, *E*, *F*, *G*). Sur chaque ligne est indiqué le temps de trajet en minutes (correspondance comprise) entre les deux gares qu'elle relie.



Un usager souhaite se rendre le plus rapidement possible de *B* à *G*. Quel itinéraire cet automobiliste doit-il emprunter ?

- B-C-D-A-F
- B-C-D-F-G
- G-F-D-C-B
- B-C-D-A-G

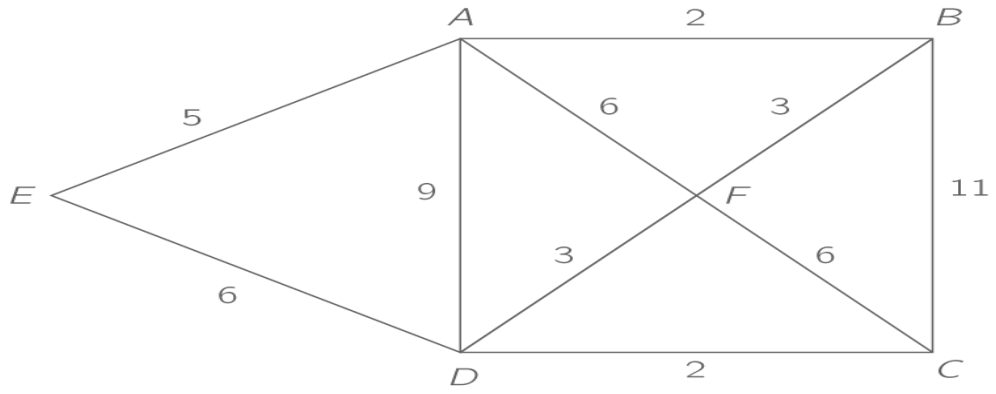
On représente sur le graphe ci-dessous les liaisons ferroviaires entre sept gares (*A*, *B*, *C*, *D*, *E*, *F*, *G*). Sur chaque ligne est indiqué le temps de trajet en minutes (correspondance comprise) entre les deux gares qu'elle relie.



Un usager souhaite se rendre le plus rapidement possible de *B* à *F*. Quel itinéraire cet automobiliste doit-il emprunter ?

- F-C-D-E-G-B
- F-C-D-E-G-A
- B-G-E-D-C-F
- B-C-A-D-G-F

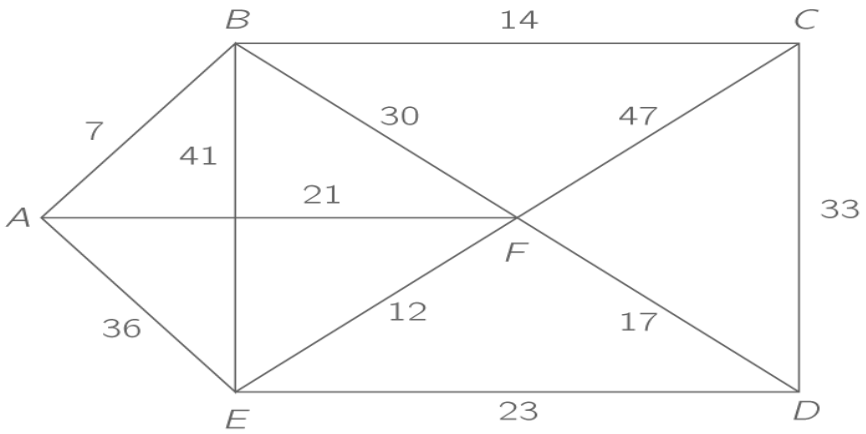
On représente sur le graphe *G* ci-dessous les liaisons routières entre six villes (*A*, *B*, *C*, *D*, *E*, *F*). Sur chaque route est indiqué le temps de trajet (en minutes) entre les deux villes qu'elle relie.



Un automobiliste souhaite se rendre le plus rapidement possible de *C* à *A*. Quel itinéraire cet automobiliste doit-il emprunter ?

- A– B – F – D – C
- C – E – F – D – A
- C – F – D – E – A
- C – D – F – B – A

On représente sur le graphe G ci-dessous les liaisons routières entre six villes (A, B, C, D, E, F). Sur chaque route est indiqué le temps de trajet (en minutes) entre les deux villes qu'elle relie.

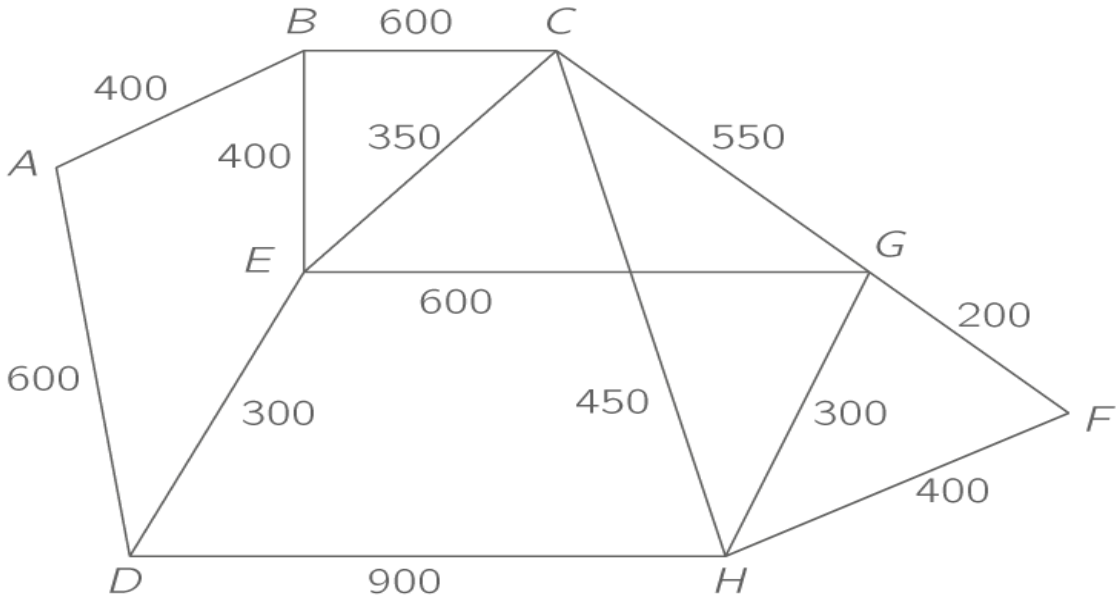


Un automobiliste souhaite se rendre le plus rapidement possible de E à C. Quel itinéraire cet automobiliste doit-il emprunter ?

- E – D – C
- E – F – D – C
- E – F – A – B – C
- E – F – C

Exercice 2 : Algorithme de Djikstra

On représente sur le graphe *G* ci-dessous les liaisons routières entre 8 villes (*A*, *B*, *C*, *D*, *E*, *F, G, H*). Sur chaque route est indiquée la distance en km entre les deux villes qu'elle relie.

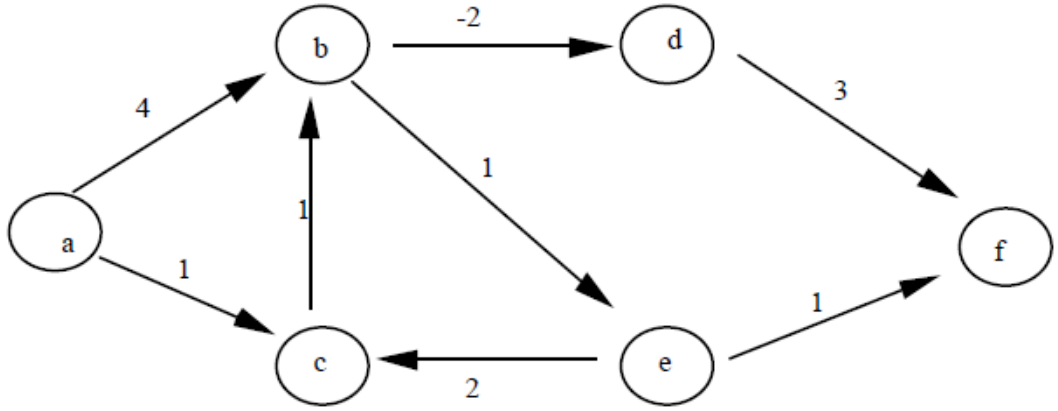


En partant de la ville A quelle serait le plus court chemin entre A et les autres villes. Appliquez l'algorithme de DIJKSTRA ?

TD-2 Plus Court Chemin

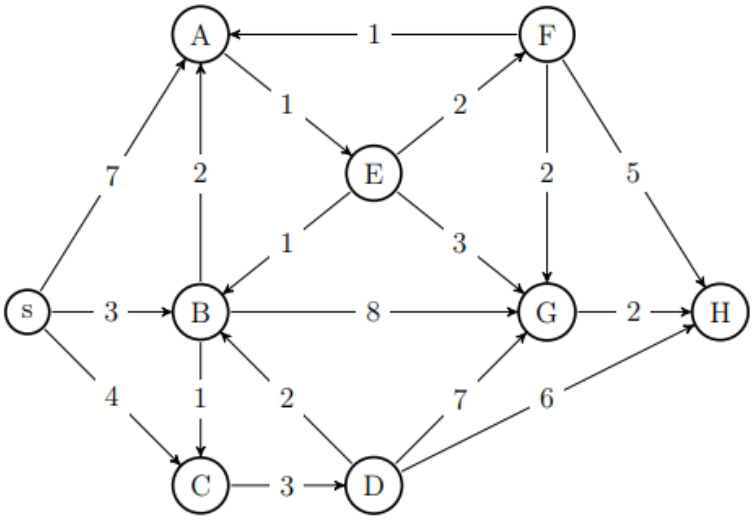
Exercice 1 :

Déterminer dans le graphe suivant les plus courts chemins à partir du sommet **f.** Utiliser l'algorithme de Ford-Bellman (préciser pourquoi cela est nécessaire) en parcourant les sommets dans l'ordre alphabétique.



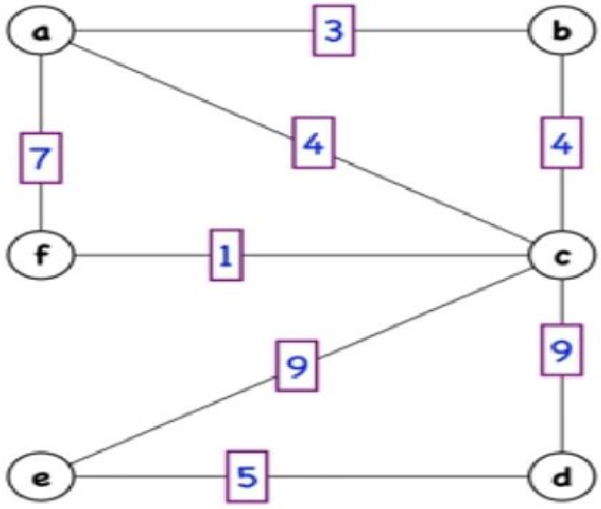
Exercice 2 :

Déterminer dans le graphe suivant les plus courts chemins à partir du sommet**.** Utiliser l'algorithme de Dijkstra.



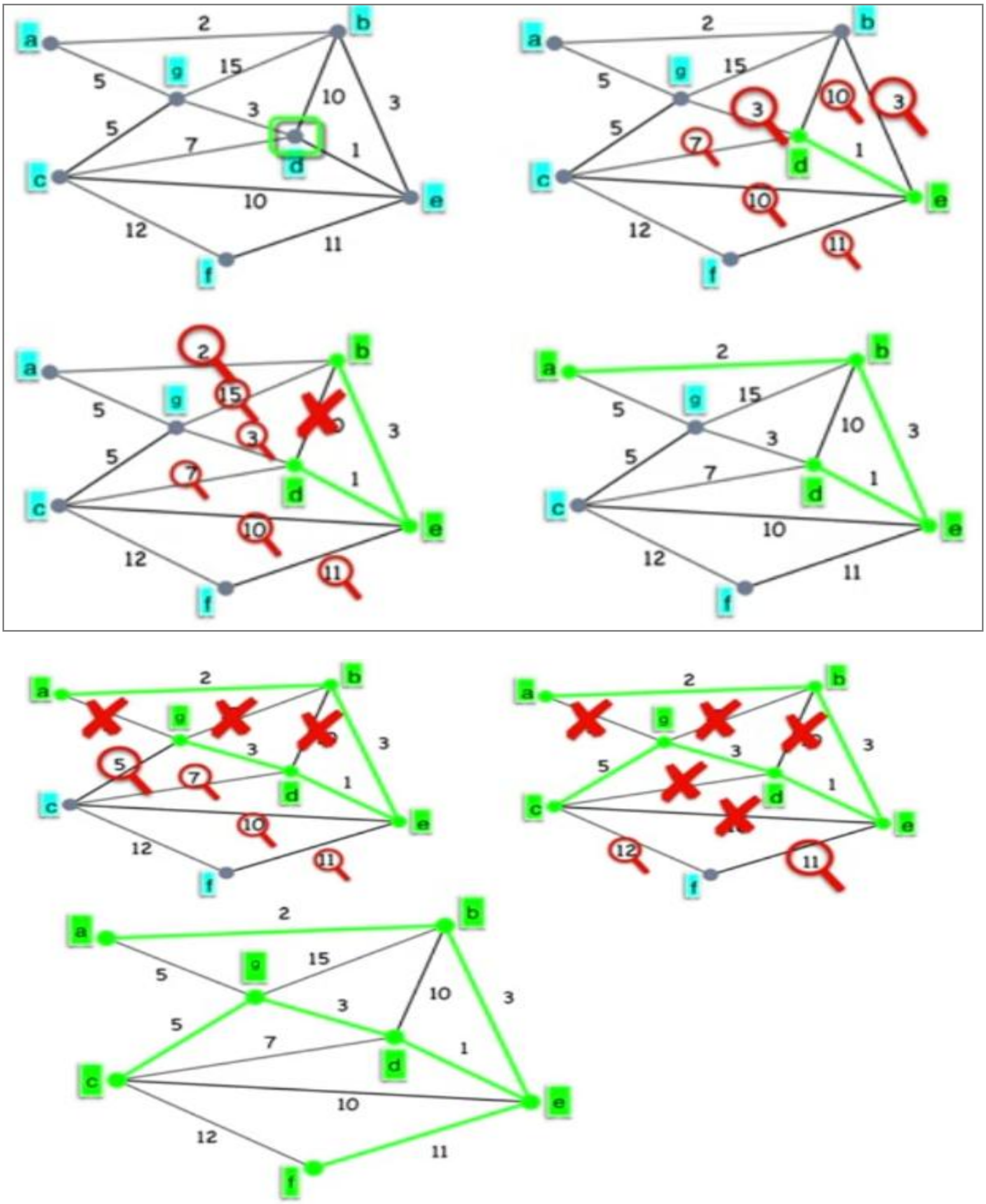
Exercice 3 :

Déterminer dans le graphe suivant les plus courts chemins à partir du sommet**.** Utiliser l'algorithme de **KRUSKAL**.



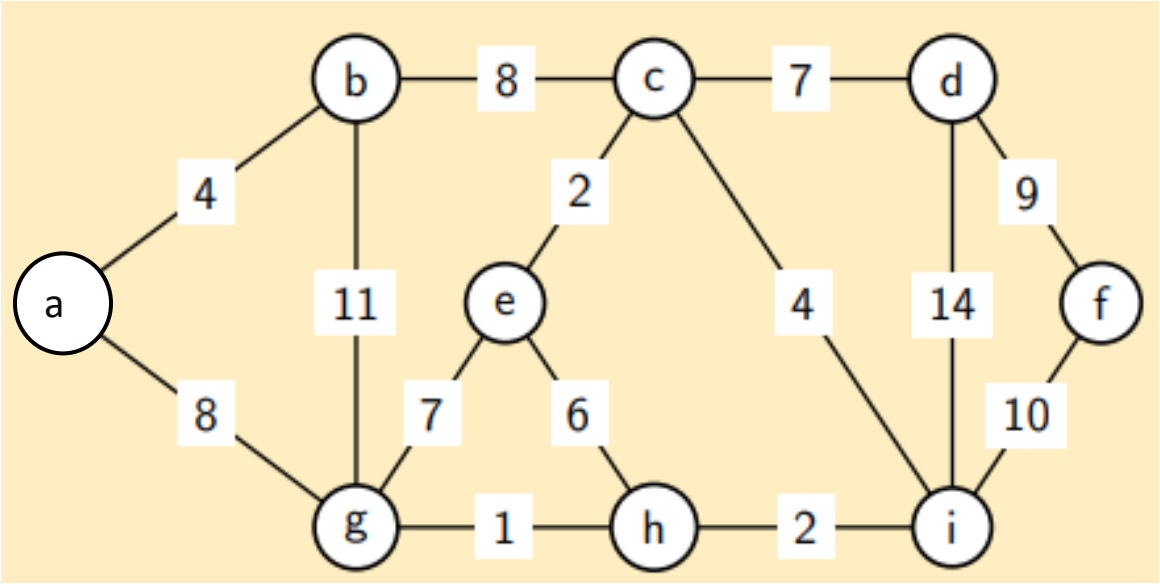
Exercice 4 :

Déterminer dans le graphe suivant les plus courts chemins à partir du sommet**.**
Utiliser l'algorithme de **PRIM**.



Principe :

L'algorithme de **PRIM** construit un arbre T en rajoutant à chaque étape le sommet qui n'est pas encore dans l'arbre et qui possède l'arête de poids minimum parmi celles reliant les sommets de T aux sommets de G − T.



Chapitre 4 : COLORATION D'UN GRAPHE

Introduction

En théorie des graphes, la coloration de graphe consiste à attribuer une couleur à chacun de ses sommets de manière que deux sommets reliés par une arête soient de couleur différente. On cherche souvent à utiliser le nombre minimal de couleurs, appelé nombre chromatique.

La coloration fractionnaire consiste à chercher non plus une mais plusieurs couleurs par sommet et en associant des coûts à chacune. Le champ d'applications de la coloration de graphe couvre notamment le problème de l'attribution de fréquences dans les télécommunications, la conception de puces électroniques ou l'allocation de registres en compilation.

Coloration usuelle

On appelle **coloriage** ou **coloration usuelle** d'un graphe G=(X,E), une application ϕ de X dans l'ensemble des entiers telle que $\phi(x) \neq \phi(y)$ lorsque x et y sont adjacents. Deux sommets adjacents ne peuvent pas être coloriés de la même couleur et tous les sommets doivent être coloriés.

Définition : Nombre chromatique d'un graphe

On appelle **nombre chromatique** d'un graphe G=(X,E), le plus petit nombre de couleurs nécessaires pour colorier ce graphe. Ce nombre est noté **γ(X,E)**.

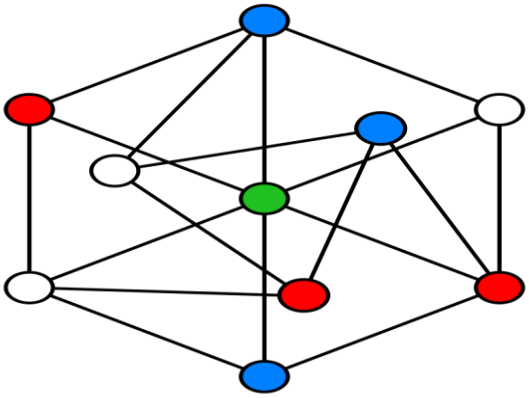
Le nombre chromatique est toujours compris entre 1 et le nombre de points.

Principe des Algorithmes de coloriage :

La notion de coloration n'est définie que pour les graphes sans boucle, et la multiplicité des arêtes ne joue aucun rôle. Donc, soit *G* un graphe simple (sans boucle ni arête multiple)

Algorithme de Welsh et Powell

1. Repérer le degré de chaque sommet.
2. Ranger les sommets par ordre de degrés décroissants (dans certains cas plusieurs possibilités).
3. Attribuer au premier sommet (A) de la liste une couleur.
4. Suivre la liste en attribuant la même couleur au premier sommet (B) qui ne soit pas adjacent à (A).
5. Suivre (si possible) la liste jusqu'au prochain sommet (C) qui ne soit adjacent ni à A ni à B.
6. Continuer jusqu'à ce que la liste soit finie.
7. Prendre une deuxième couleur pour le premier sommet (D) non encore coloré de la liste.
8. Continuer jusqu'à avoir coloré tous les sommets.



<u>Exercice d'application :</u>