Chapitre 4

Problèmes de cheminement

1. Recherche des composantes connexes

1. Présentation des objectifs

La vérification de la connexité d'un graphe est un des premiers problèmes de la théorie des graphes. De nombreux algorithmes permettent de trouver l'ensemble des sommets appartenant à une même composante connexe.

(Trémeaux 1882-Tarjan 1972).

2. La recherche des composantes connexes

Les composantes connexes d'un graphe se déterminent en utilisant un algorithme de marquage simple.

1. Recherche des composantes connexes

Le principe

L'idée de cet algorithme est la suivante: pour un sommet quelconque x du graphe G, il s'agit de trouver toutes les chaînes reliant ce sommet aux autres sommets du graphe; ainsi les sommets reliés au sommet x par une chaîne forment la composante connexe qui contient le sommet x.

Enoncé

Données: un graphe G=(X,U)

Résultat: le nombre k de composantes connexes de G, et la liste de ses composantes connexes.{C_1, C_2,..C_k }

1. Recherche des composantes connexes

(0) Initialisation: k=0, W=X.

(1) (1.1) Choisir un sommet de W et le marquer d'un signe (+), puis marquer tous ses voisins d'un (+). On continue cette procédure jusqu'à ce qu'on ne puisse plus marquer de sommets.

(1.2) Poser k=k+1 et C_k l'ensemble de sommets marqués.

(1.3) Retirer de W les sommets de C_k et poser $W=W-C_k$.

(1.4) On teste si W=Ø. - Si oui terminer aller à (2).

- Si non aller à (1)

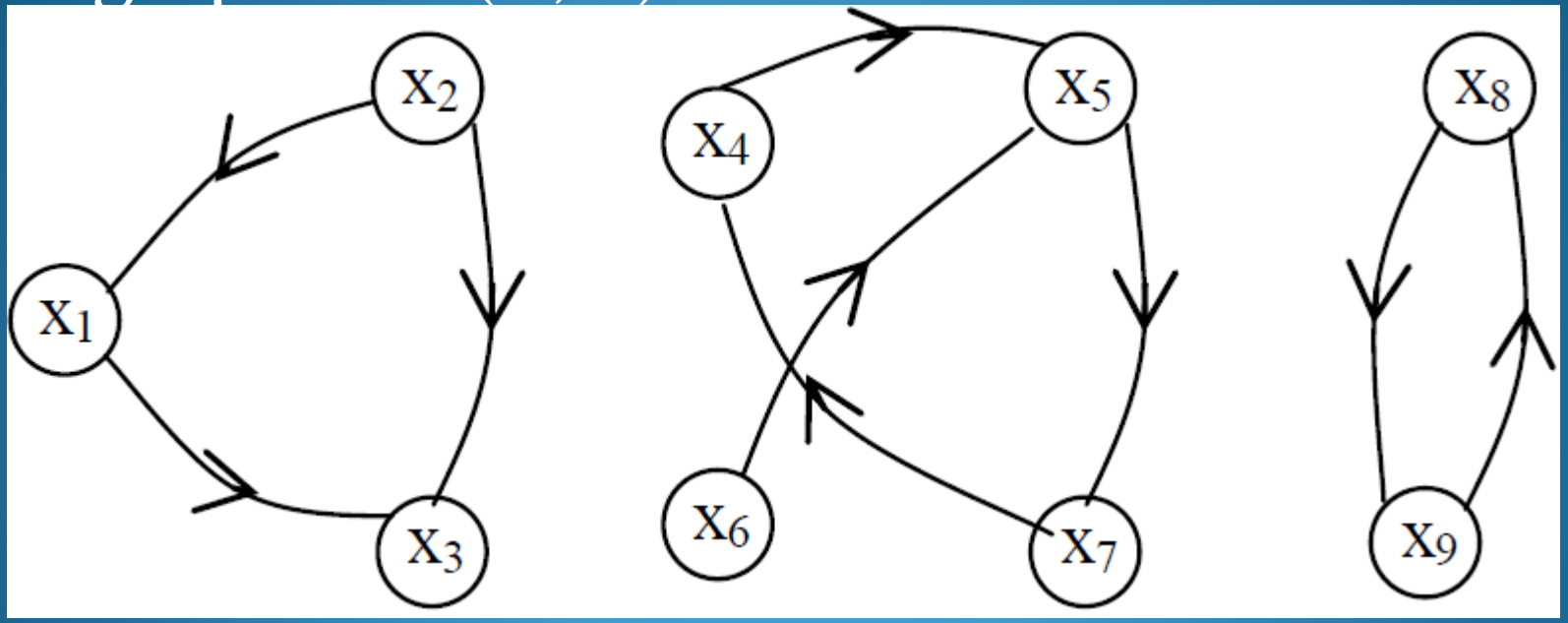
(2) Le nombre de composantes connexes de (G) est k.

Chaque ensemble Ci, i=1,..,k correspond aux sommets d'une composante connexe de (G).

1. Recherche des composantes connexes

Application:

Soit le graphe G=(X,U) suivant:

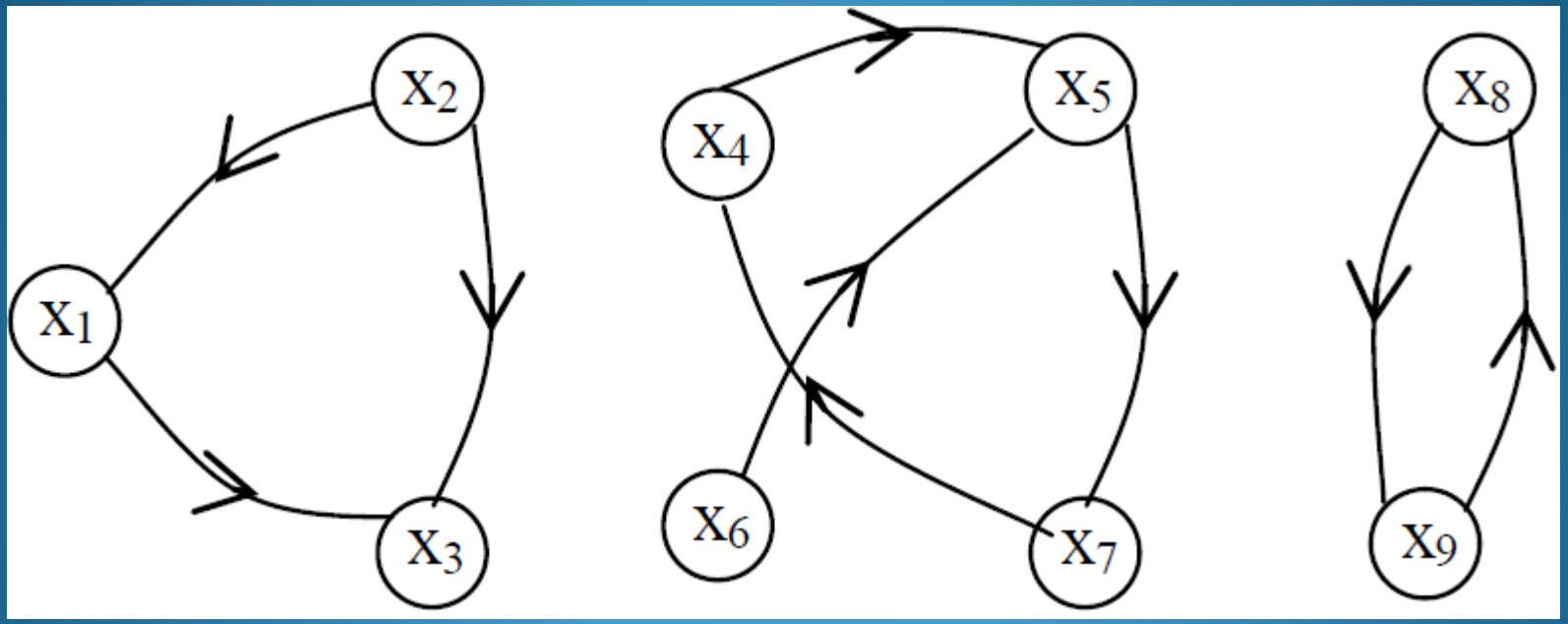


Le graphe (G) n'est pas connexe, car il n'existe pas de chaîne reliant les sommets x_3 et x_8.

Initialisation: k=0, W={x_1,x_2,x_3,...x_9}

1. Recherche des composantes connexes

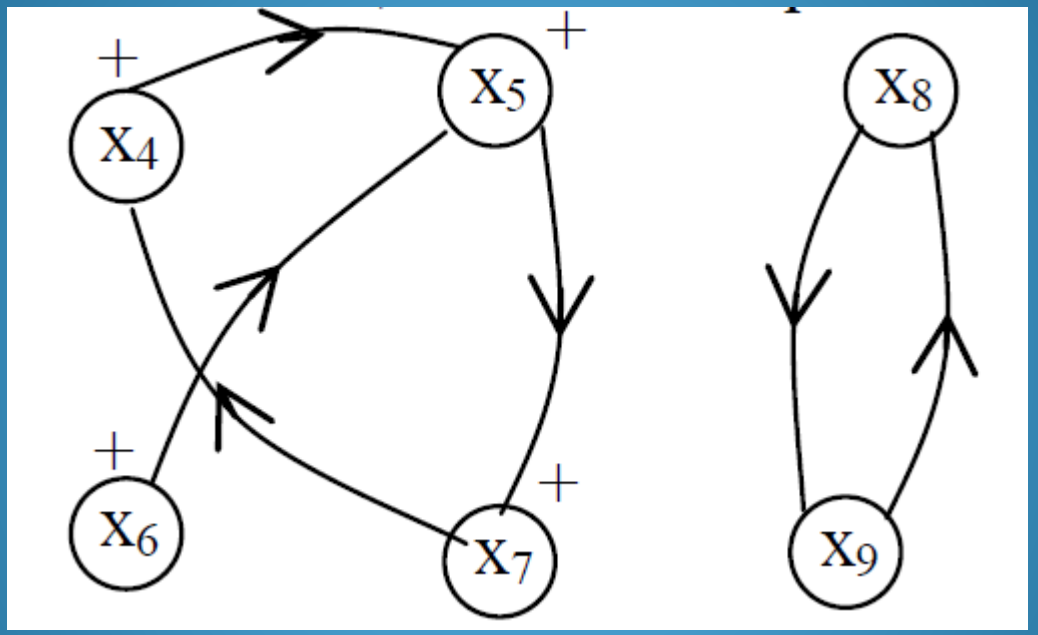
Itération 1: on choisir dans W le sommet x2, et on le marque d'un (+), on marque ensuite ses voisins x_1 et x_3



Soit $C_1=\{x_1,x_2,x_3\}$ l'ensemble des sommets marqués. On retire de W les sommets C1, on obtient: $W=\{x_4,x_5,x_6,x_7,x_8,x_9\} \neq \phi$

1. Recherche des composantes connexes

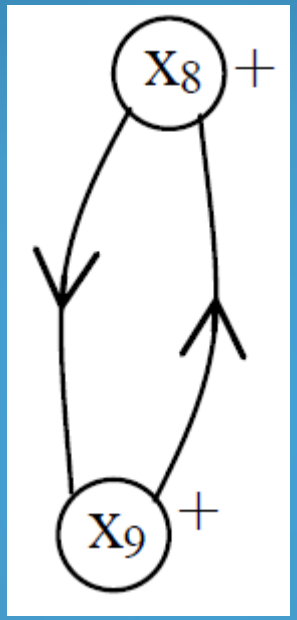
Itération 2: on choisir dans W le sommet x5, et on le marque d'un (+), on marque ensuite ses voisins x_4 ,x_6 et x7



Soit $C_2=\{x_4,x_5,x_6,x_7\}$ l'ensemble des sommets marqués. On retire de W les sommets C2, on obtient: $W=\{x_8,x_9\} \neq \phi$

1. Recherche des composantes connexes

Itération 3: on choisir dans W le sommet x8, et on le marque d'un (+), on marque ensuite ses voisins x_9



Soit $C_3=\{x_8,x_9\}$ l'ensemble des sommets marqués. On retire de W les sommets C_3, on obtient: $W= \phi$

Le nombre de composantes connexes de (G) est 3, elles sont: $C_1=\{x_1,x_2,x_3\}$,$C_2=\{x_4,x_5,x_6,x_7\}$, $C_3=\{x_8,x_9\}$

2.Recherche du plus court chemin

1. La recherche de plus court chemin

Si un plus court chemin d'un sommet s à un sommet x existe dans un réseau. La longueur de ce plus court chemin sera appelée "plus courte distance de s à x " et se notera $\pi(x)$.

1.1. Présentation des conditions

Soit un graphe G, on associe à chaque arc u de G une **longueur** $l(u)$ telle que $\forall\ u \in U$, $l(u) \geq 0$. Soit a et b deux sommets de G, il s'agit de trouver un chemin $\mu(a,b)$ tel que: $l(\mu) = \sum_{u \in \mu} l(u)$,

$\mu(a,b)$ est appelé le **plus court chemin** de a à b.

2.Recherche du plus court chemin

1.2. Algorithme de Dijkstra

Principe de l'algorithme :

On applique cette algorithme pour déterminer une arborescence des plus courtes distances sur un réseau R=(X,U,d), où les longueurs des arcs sont positives ou nulles .

L'idée est de calculer de proche en proche, l'arborescence des plus courtes distances, issue du sommet s à un sommet donné p. Une particularité de cet algorithme est que les distances s'introduisent dans l'ordre croissant.

2.Recherche du plus court chemin

1.2. Algorithme de Dijkstra

Principe de l'algorithme :

Algorithme 7: Algorithme de Dijkstra

Données : Un graphe orienté pondéré $G = (X, A, W)$ et un sommet $s \in X$
Résultat : Le plus court chemin de s vers tous les autres sommets de G

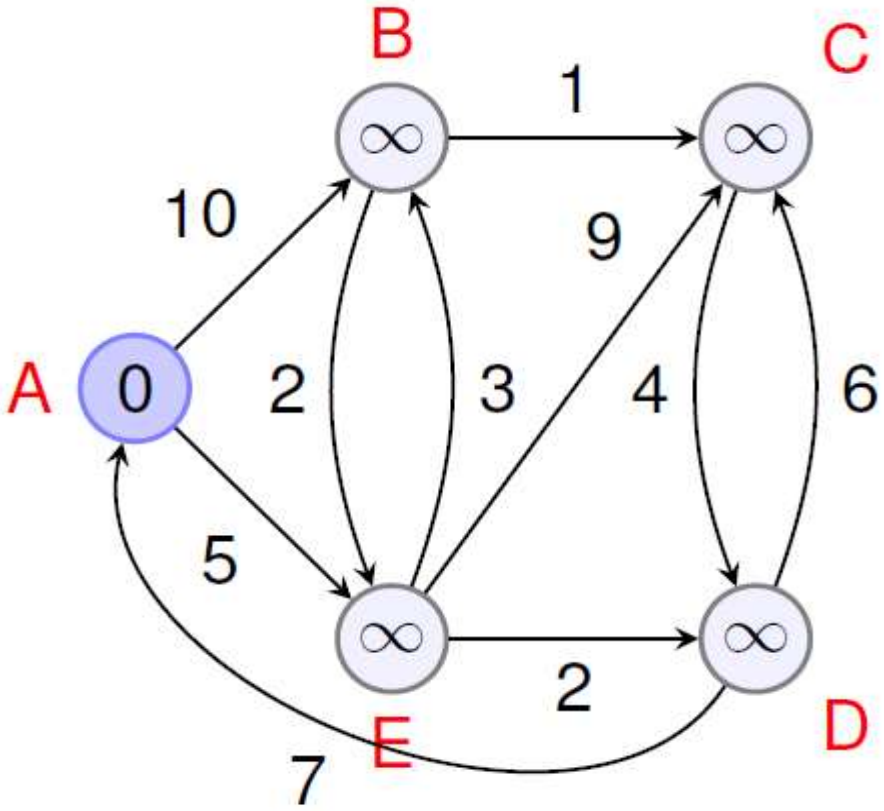
```
// V : Tableau stockant les étiquettes des sommets de G
Initialiser V à +∞
V[s] = 0
// P : Tableau permettant de retrouver la composition des chemins
Initialiser P à 0
P[s] = s
répéter
    // Recherche du sommet x non fixé de plus petite étiquette
    V_min = +∞
    pour y allant de 1 à N faire
        si y non marqué et V[y] < V_min alors
            x ← y
            V_min ← V[y]
    // Mise à jour des successeurs non fixés de x
    si V_min < +∞ alors
        Marquer x
        pour tout successeur y de x faire
            si y non marqué et V[x] + W[x, y] < V[y] alors
                V[y] = V[x] + W[x, y]
                P[y] = x
jusqu'à V_min = +∞
```

2.Recherche du plus court chemin

1.2. Algorithme de Dijkstra

Exemple: Cherchons les plus courts chemins d'origine A dans ce graphe:

On se place au sommet de plus petit poids, ici le sommet A.

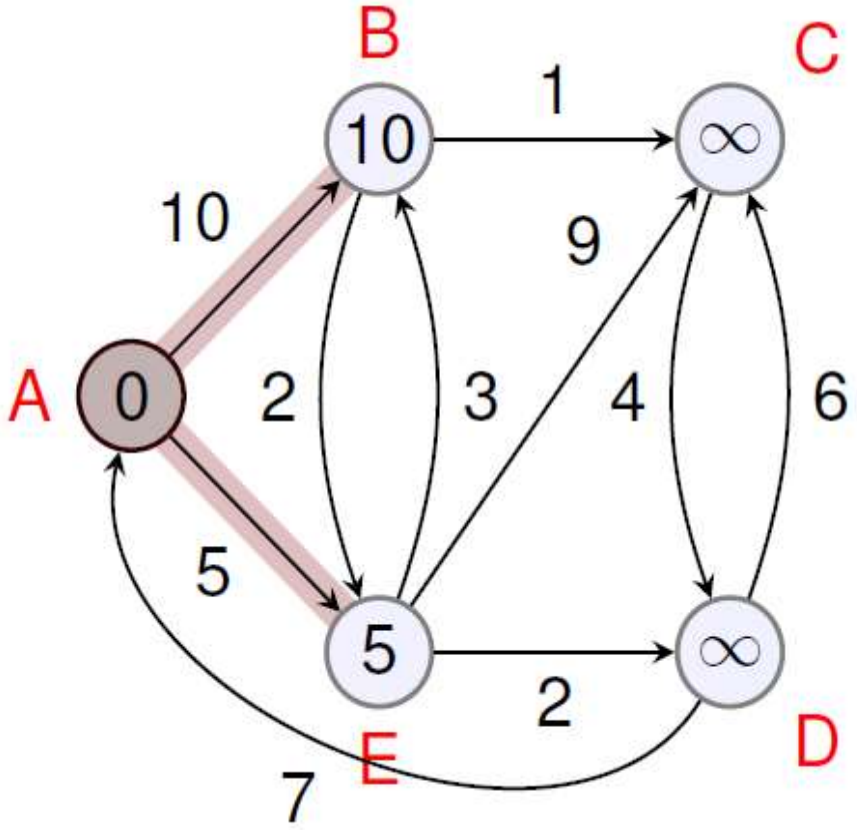


A	B	C	D	E
0	∞	∞	∞	∞
•				
•				
•				
•				
•				

2.Recherche du plus court chemin

1.2. Algorithme de Dijkstra

Exemple: On étudie chacune des arêtes partant du sommet choisi. Dans les colonnes, on mets la distance à A, et le sommet d'où l'on vient.

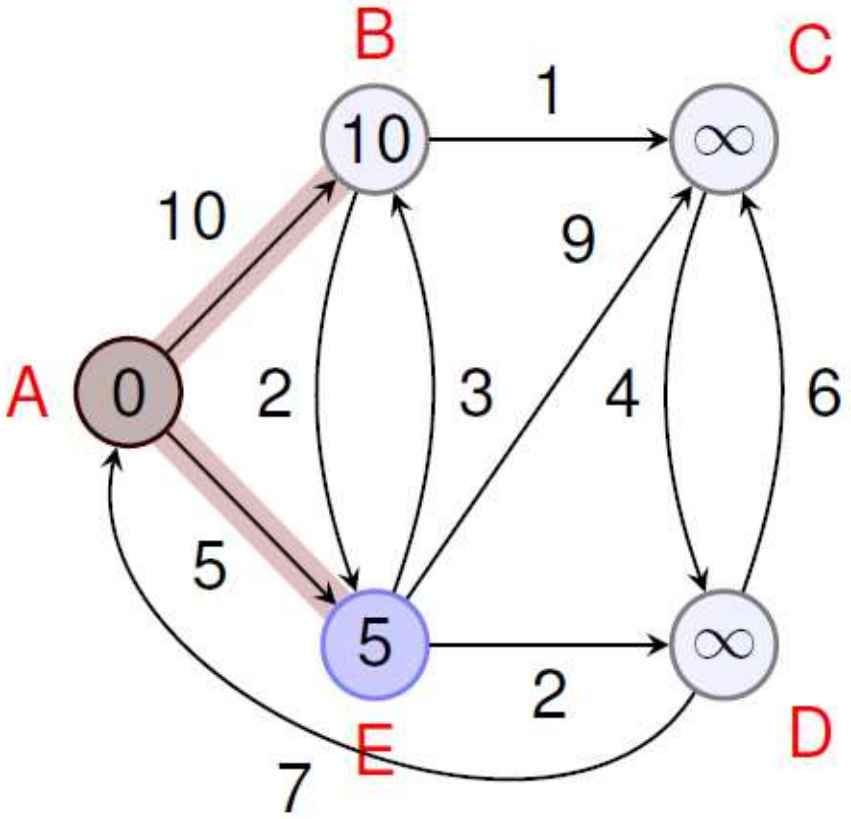


A	B	C	D	E
0	∞	∞	∞	∞
•	10_A	∞	∞	5_A
•				
•				
•				
•				

2.Recherche du plus court chemin

1.2. Algorithme de Dijkstra

Exemple: On se place de nouveau au sommet de plus petit poids, ici E.

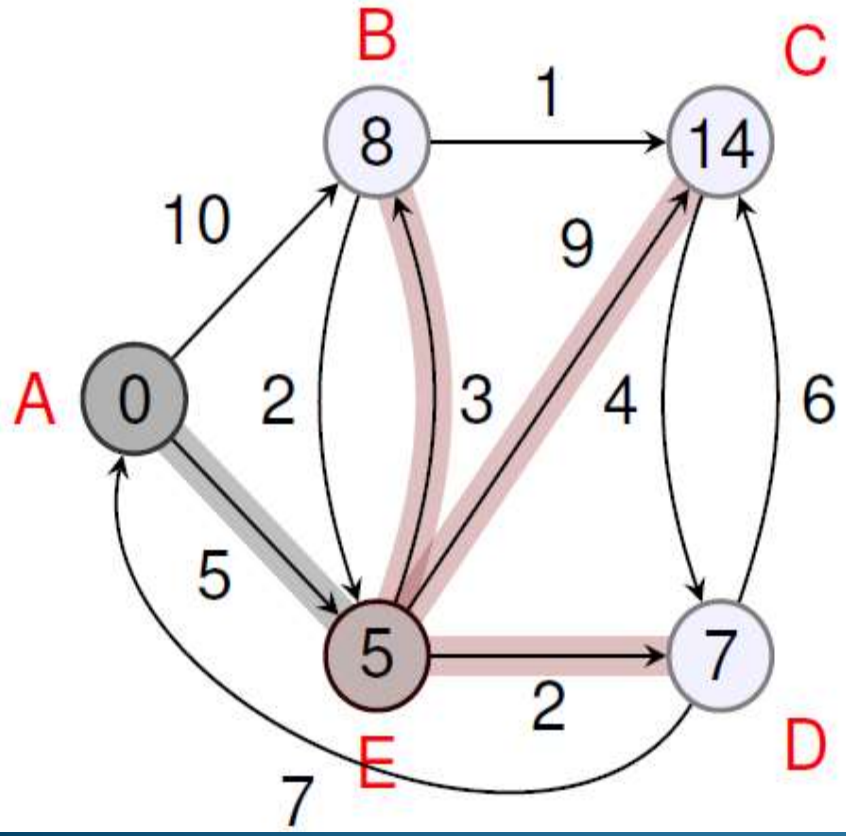


A	B	C	D	E
0	∞	∞	∞	∞
•	10_A	∞	∞	$\boxed{5_A}$
•				•
•				•
•				•
•				•

2.Recherche du plus court chemin

1.2. Algorithme de Dijkstra

Exemple: En applique l'algorithme

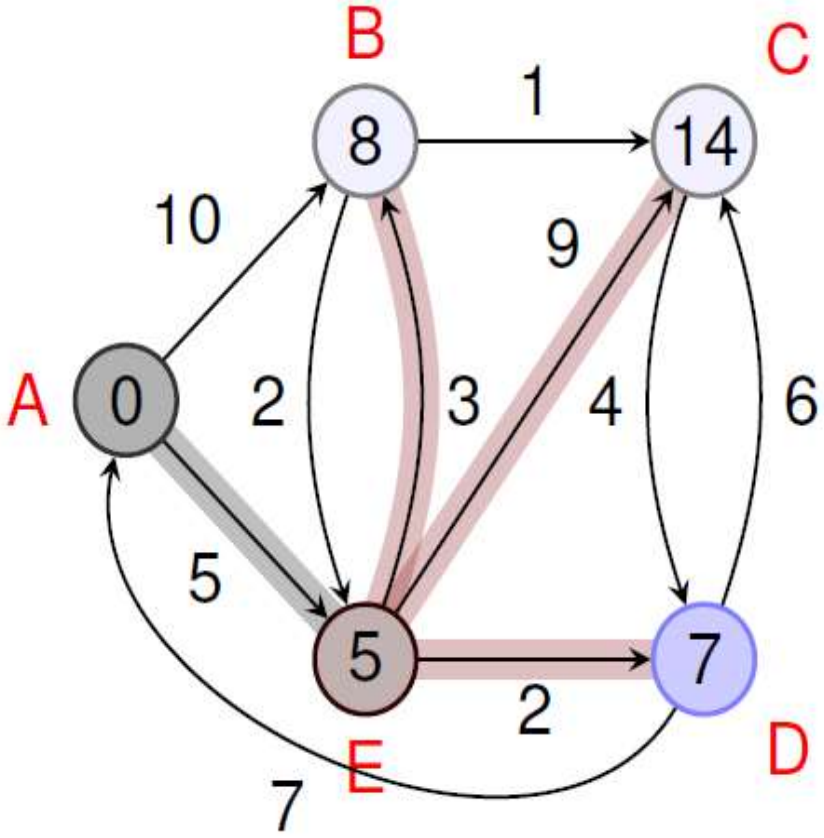


A	B	C	D	E
0	∞	∞	∞	∞
•	10_A	∞	∞	$\boxed{5_A}$
•	8_E	14_E	7_E	•
•				•
•				•

2.Recherche du plus court chemin

1.2. Algorithme de Dijkstra

Exemple: On se place de nouveau au sommet de plus petit poids, ici D.

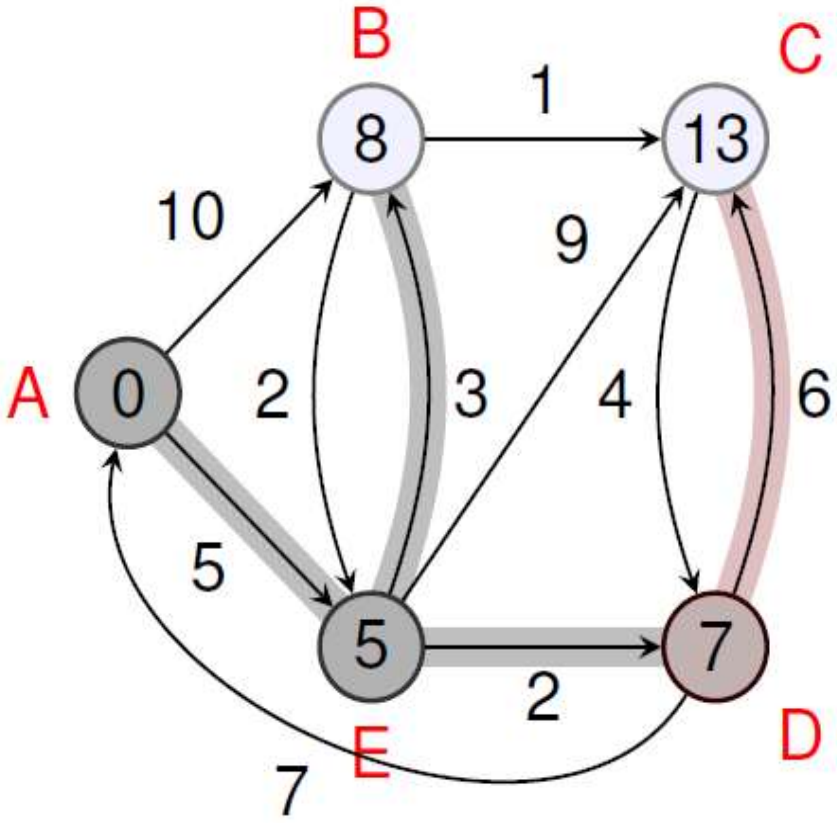


A	B	C	D	E
0	∞	∞	∞	∞
•	10_A	∞	∞	5_A
•	8_E	14_E	$\boxed{7_E}$	•
•			•	•
•			•	•

2.Recherche du plus court chemin

1.2. Algorithme de Dijkstra

Exemple: En applique l'algorithme

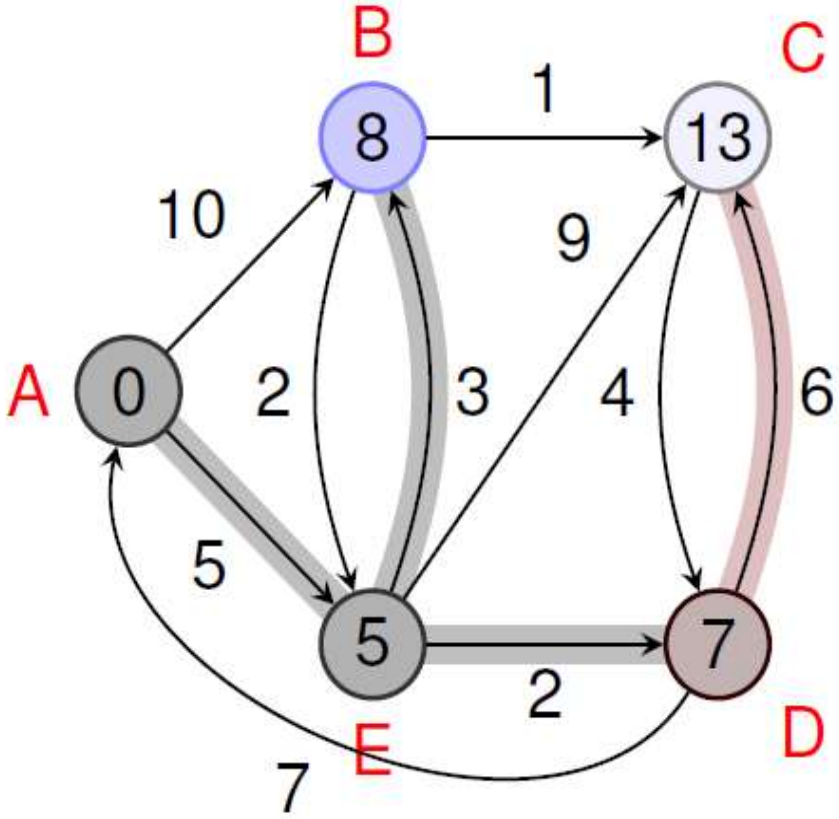


A	B	C	D	E
0	∞	∞	∞	∞
•	10_A	∞	∞	5_A
•	8_E	14_E	$\boxed{7_E}$	•
•	8_E	13_D	•	•
•			•	•
•			•	•

2.Recherche du plus court chemin

1.2. Algorithme de Dijkstra

Exemple: On se place de nouveau au sommet de plus petit poids, ici B.

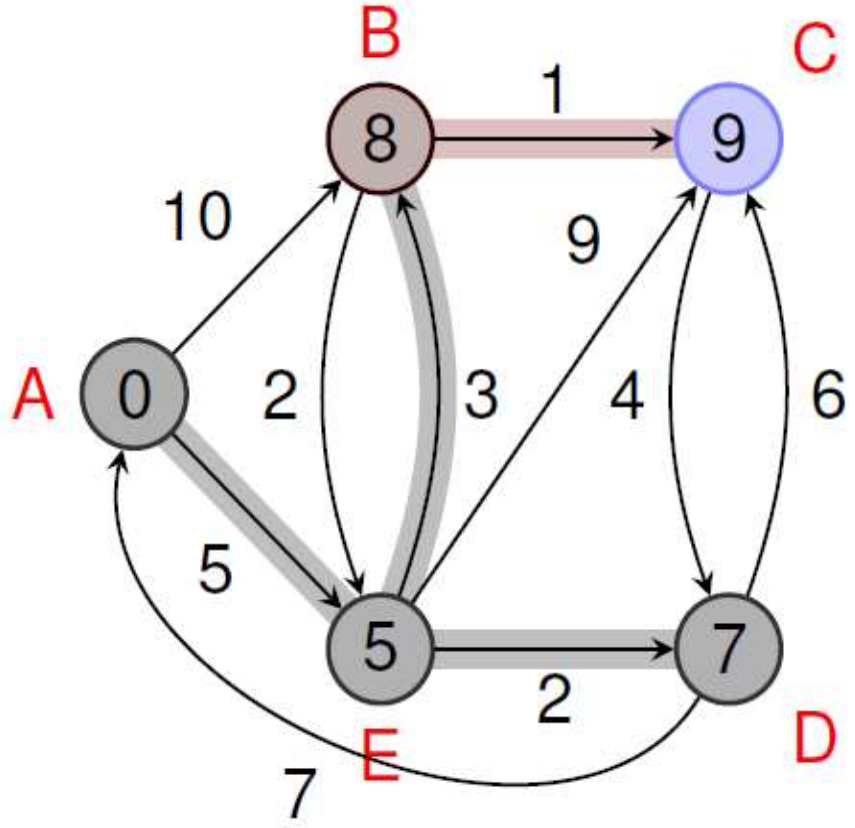


A	B	C	D	E
0	∞	∞	∞	∞
•	10_A	∞	∞	5_A
•	8_E	14_E	7_E	•
•	$\boxed{8_E}$	13_D	•	•
•	•		•	•
•	•		•	•

2.Recherche du plus court chemin

1.2. Algorithme de Dijkstra

Exemple: En applique l'algorithme

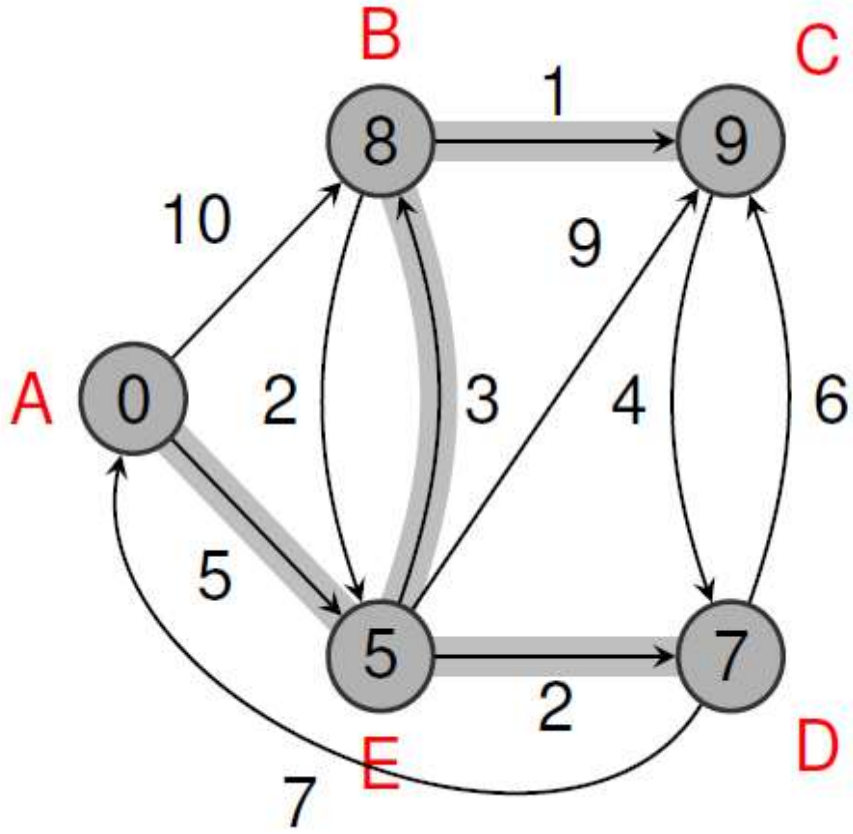


A	B	C	D	E
0	∞	∞	∞	∞
•	10_A	∞	∞	5_A
•	8_E	14_E	7_E	•
•	8_E	13_D	•	•
•	•	$\boxed{9_B}$	•	•
•	•	•	•	•

2.Recherche du plus court chemin

1.2. Algorithme de Dijkstra

Exemple: Si l'on ne considère que les flèches soulignées, on obtient un arbre, un graphe sans cycle.



A	B	C	D	E
0	∞	∞	∞	∞
•	10_A	∞	∞	5_A
•	8_E	14_E	7_E	•
•	8_E	13_D	•	•
•	•	9_B	•	•
•	•	•	•	•

3. Recherche d'un arbre de poids extrémum

3.1. Présentation des objectifs

Si on associe à chaque arc d'un graphe G=(X,U) une valeur (un poids). Le problème de l'arbre de coût minimum (maximum) consiste à trouver un sous graphe qui est un arbre, dont la somme des poids des arcs est minimale (maximale).

Par exemple: minimiser le coût d'installation des lignes téléphoniques dans une localité peut être représenté comme un problème de recherche d'un arbre de coût minimum.

3. Recherche d'un arbre de poids extrémum

3.1. Présentation des objectifs

En effet, on veut relier tous les points de la localité sans avoir de lignes inutiles, d'où la recherche d'un arbre. Ensuite on veut avoir un coût d'installation minimum, alors on associera à chaque possibilité d'installation d'une ligne le coût nécessaire et on cherchera à minimiser le coût total de toute l'installation.

Plusieurs algorithmes existent pour résoudre les problèmes de recherche d'un arbre de poids minimum, nous ne présenterons que les méthodes qui paraissent les plus célèbres.

3. Recherche d'un arbre de poids extrémum

3.2. Algorithme de Kruskal 1956

Données :

– Graphe $G = (V,E)$ ($|V| = n$, $|E| = m$)

– Pour chaque arête e de E , son poids $c(e)$.

Résultat :

Arbre ou forêt maximale $A = (V,F)$ de poids minimum.

Trier et renuméroter les arêtes de G dans l'ordre croissant de leur poids : $c(e_1) \leq c(e_2) \leq \ldots \leq c(e_m)$.

Poser $F \leftarrow \emptyset$, $k \leftarrow 0$;

Tant que $k < m$ **et** $|F| < n - 1$ **faire**

Début

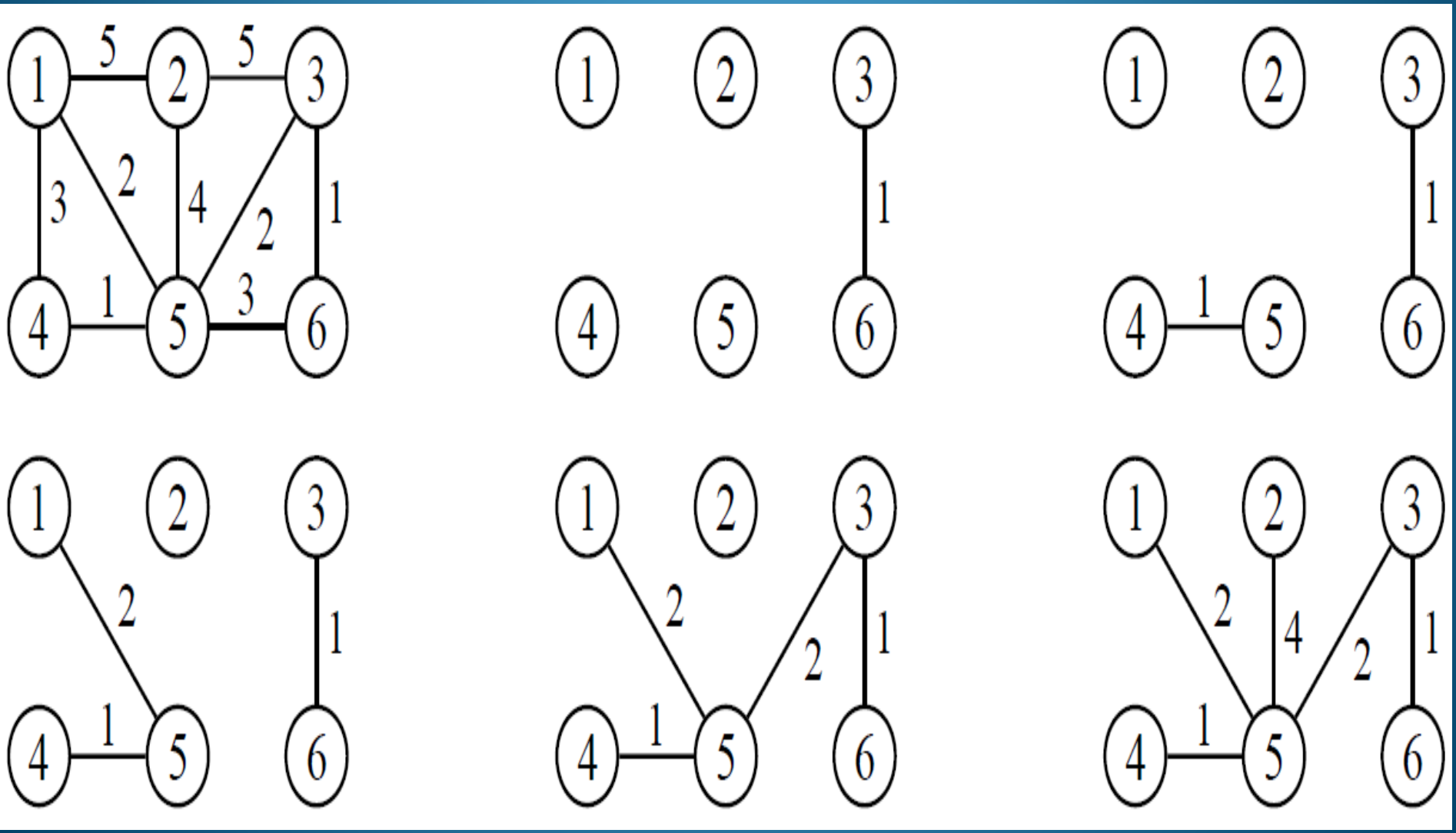
si e_{k+1} ne forme pas de cycle avec F **alors** $F \leftarrow F \cup \{e_{k+1}\}$ $k \leftarrow k+1$

Fin

3. Recherche d'un arbre de poids extrémum

3.2. Algorithme de Kruskal 1956

Exemple:



3. Recherche d'un arbre de poids extrémum

3.2. Algorithme de Kruskal 1956

Exemple:

Les arêtes de poids 3 n'ont pas pu être placées, car elles auraient formé un cycle.

L'algorithme s'est arrêté dès que cinq arêtes ont été placées. Toute arête supplémentaire aurait créé un cycle.

S'il y a plusieurs arêtes de même poids, il peut y avoir plusieurs arbres couvrants de poids minimum: tout dépend de l'ordre dans lequel ces arêtes ont été triées.