Chapitre 3 : Graphes particuliers

I. Graphe biparti :

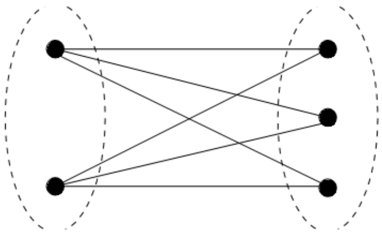
On appelle graphe biparti, noté $\boldsymbol{G = (X_1, X_2, U)}$, un graphe dont l'ensemble des sommets est partitionné en deux ensembles $\boldsymbol{(X_1, X_2)}$ et tel que pour toute arête (ou arc) $\boldsymbol{u_i}$: une extrémité est dans $\boldsymbol{X_1}$ et l'autre extrémité est dans $\boldsymbol{X_2}$.

$$X = (X_1 \cup X_2) \quad \text{et si} \quad \forall\ u_{ij} = (x_i, x_j), \begin{cases} x_i \in X_1 \Rightarrow x_j \in X_2 \\ x_i \in X_2 \Rightarrow x_j \in X_1 \end{cases}$$

- Un graphe biparti complet est noté $\boldsymbol{K_{r,s}}$ avec $r = |X_1|$ et $s = |X_2|$.
- Un graphe biparti ne possède aucun cycle impair (nombre impair d'arêtes).
- Un graphe biparti est 2-coloriable.

Exemple :

Graphe biparti : $\boldsymbol{K_{2,3}}$



Théorème de Konig : Le nombre chromatique d'un graphe G est 2 si et seulement s'il n'admet pas de cycle de longueur impaire.

II. Graphe Planaire :

C'est un graphe qui peut être représenté sur un plan tel que deux arcs (ou arêtes) ne se coupent pas.

- Tous les graphes de moins de 5 sommets sont planaires.
- Et aussi, tous les graphes bipartis de moins de 6 sommets,

Les autres, on connaît des algorithmes pour déterminer si un graphe est planaire.

Exemple d'application : la conception de circuits électriques.

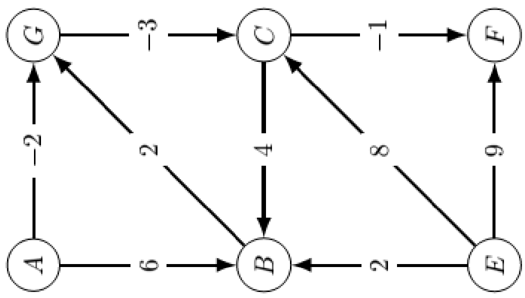
III. Graphe Complémentaire :

On appelle graphe complémentaire (on le noté $\bar{G}$) du graphe G le graphe dont les sommets sont ceux de G et dans lequel deux sommets sont adjacents si et seulement si ils ne sont pas adjacents dans G.

IV. Graphes Valués :

Un graphe valué est un graphe orienté G = (X, *U*) muni d'une fonction *f* : U → R* appelée fonction de coût.

	A	B	C	E	F	G
A		6				2
B						2
C		4			1	
E		2	8		9	
F						
G			3			



C'est-à-dire un graphe où des réels sont associés aux arêtes.

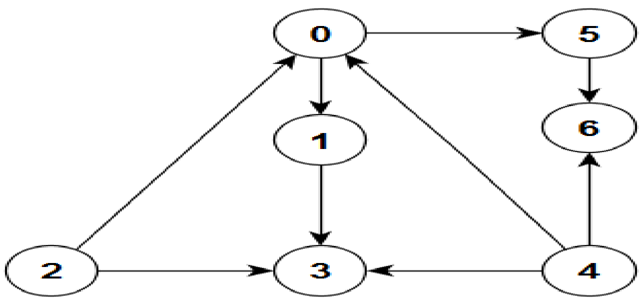
Exemple:La matrice d'adjacence du graphe valué suivant est :

V. *Graphes sans circuit :*

Un graphe sans circuit est un graphe ne contenant aucun circuit.

Algorithme 4 : l'identification d'un graphe sans circuit
Entrées : G=(X ; U) un graphe orienté, Sortie : oui / non
Début // $\Gamma(s_i)$= L'ensemble des successeurs de s_i ▪ **$X(0) = \{s_i \,/ s_i \in X,\ \Gamma(s_i) = \emptyset\}$** ▪ **$X(1) = \{s_i \,/ s_i \in X - \{X(0)\},\ \Gamma(s_i) \subset \{X(0)\}\}$** ▪ **$X(2) = \{s_i \,/ s_i \in X - \{X(0) \cup X(1)\},\ \Gamma(s_i) \subset \{X(0) \cup X(1)\}\}$** ▪ … ▪ **$X(n) = \{s_i \,/ s_i \in X - \{X(0) \cup X(1) \cup \ldots \cup X(n-1)\},\ \Gamma(s_i) \subset \{X(0) \cup X(1) \cup \ldots \cup X(n-1)\}\}$** **Si** $X = X(0) \cup X(1) \cup \ldots \cup X(n-1)$ **alors** *le graphe est sans circuit* **Fin**

Exemple 1 : soit le graphe



- $X(0) = \{ x_3 , x_6\}$
- $X(1) = \{ x_5 , x_1\}$
- $X(2) = \{ x_0 \}$
- $X(3) = \{ x_4 , x_2\}$

Algorithme 5: l'identification d'un graphe sans circuit
Entrées: La matrice d'adjacence **M** d'un graphe orienté, Sortie : oui / non
Début 1. **Si** ∃ une ligne ***i*** ne contient que des « **0** » **alors** Supprimer la ligne i et la colonne i. **Si** La matrice M ne contient que des « **0** » **alors** Le graphe est sans circuit **Sinon** Allez **1** à **FinSI** **Fin**

Exemple 1 : soit le graphe

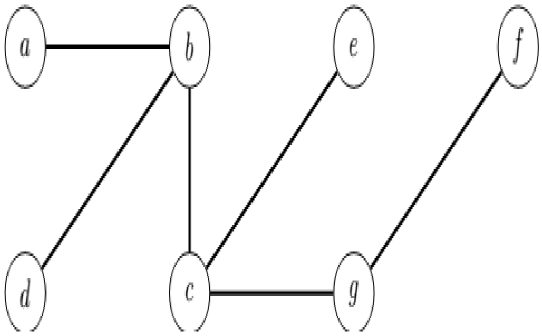
	x_0	x_1	x_2	x_3	x_4	x_5	x_6
x_0	0	1	0	0	0	1	0
x_1	0	0	0	1	0	0	0
x_2	1	0	0	1	0	0	0
x_3	0	0	0	0	0	0	0
x_4	1	0	0	1	0	0	1
x_5	0	0	0	0	0	0	1
x_6	0	0	0	0	0	0	0

	x_0	x_1	x_2	x_4	x_5
x_0	0	1	0	0	1
x_1	0	0	0	0	0
x_2	1	0	0	0	0
x_4	1	0	0	0	0
x_5	0	0	0	0	0

	x_0	x_2	x_4
x_0	0	0	0
x_2	1	0	0
x_4	1	0	0

	x_2	x_4
x_2	0	0
x_4	0	0

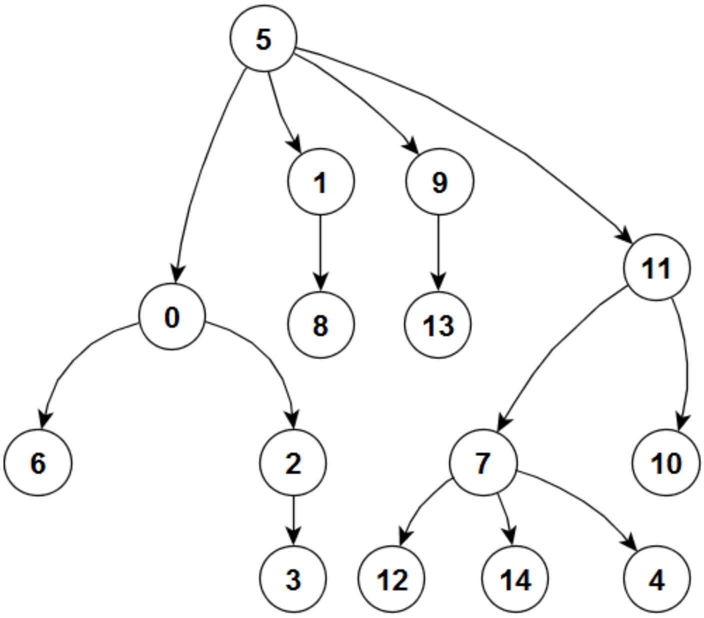
- $X(0) = \{ x_3 , x_6\}$
- $X(1) = \{ x_5 , x_1\}$
- $X(2) = \{ x_0 \}$
- $X(3) = \{ x_4 , x_2\}$



VI. ***Arbres, arborescences et forêts :***

1. *Définition d'un Arbre :*

- Un arbre est un graphe ***connexe*** et ***sans cycle.***
- Un arbre de ***n*** nœuds a donc exactement ***n-1*** arêtes.



2. *Définition d'une Arborescence :*

Une arborescence (***G, r***) est un arbre orienté de ***racine r*** tel que :

- la ***racine r*** (ou sommet ***r***) n'a pas de prédécesseur
- Tout sommet ***s***, autre que ***r***, a un seul prédécesseur.

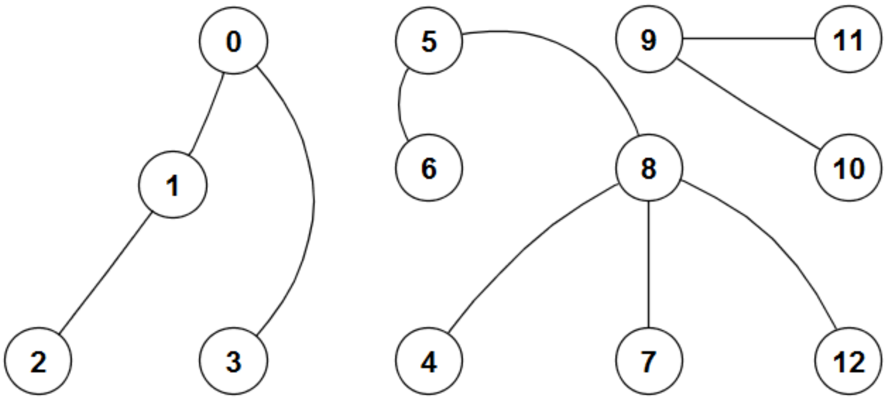
3. *Propriétés*

- Dans un **G** orienté, *la **racine*** de l'arbre est un nœud qui n'a pas d'arcs entrants (unique).
- Une ***feuille*** est un nœud de degré 1 (dans un graphe orienté : uniquement un arc entrant et sans arcs sortants).
- Le graphe ***G*** possède au moins une ***feuille.***

- Si on ajoute une arête à un arbre, on crée un cycle (et un seul).
- Un arbre ne comporte pas de boucles. En effet, toute boucle est un cycle.
- Si on retire une arête à un arbre, on rompt la connexité.
- Dans un arbre, il y a une chaîne (et une seule) entre deux sommets.
- Dans une arborescence, il existe un chemin unique joignant la racine à tout autre sommet.

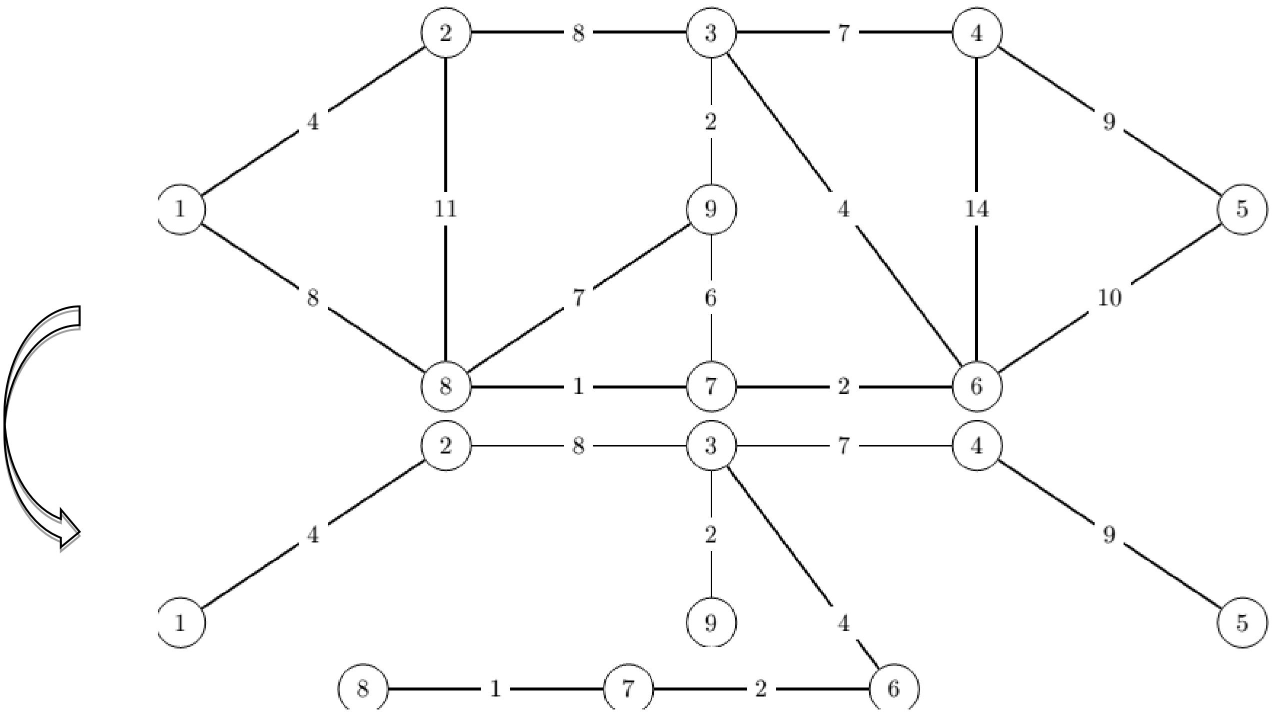
4. Définition d'une forêt:

C'est un graphe non orienté sans cycle (pas forcément connexe). Chaque composante connexe d'une forêt est un arbre.



5. Problème de l'arbre minimal

Le problème de l'arbre de coût minimum (ou maximum) consiste à trouver un arbre (*Graphe partiel de G*), dont la somme des poids des arcs est minimale (ou maximale). En pratique, les problèmes se ramène plutôt à trouver l'arbre de coût minimum, ce qui ne change pas fondamentalement le principe des algorithmes proposés ici.



Exemple d'application : minimiser le coût d'installation de lignes électriques entre des maisons peut être modélisé par la recherche d'un arbre de coût minimum. En effet, on veut :

- Utiliser moins de câbles possibles (chercher à minimiser la longueur totale de câble utilisé).
- Connecter toutes les maisons entre elles sans avoir de lignes inutile.

Remarque :

- Le nombre d'arrêtes dans un ACPM (Arbre couvrant de poids minimum) est n-1

5.1. Algorithme de Kruskal

L'idée de l'Algorithme de *Kruskal* est de trier les arcs par ordre croissant de leur poids. Ensuite dans cet ordre, les arcs sont ajoutés tant qu'il n'y a pas de cycle introduit, sinon on passe à l'arc suivant dans l'ordre de tri.

```
Algorithme 6 : permet d'extraire l'arbre de poids minimum
Entrées : G=(X ; U) orienté, Sortie : Arbre = ( X, U')
Début
   ▪ Supprimer toutes les boucles.
   ▪ Supprimer toutes les arêtes parallèles entre deux
     sommets sauf l'arête au poids minimum.
   ▪ trier les arêtes en ordre de poids croissant et les
     ranger dans une liste L.
   ▪ U' ← {}

Pour tous les arcs ui de U : i allant de 1 à m faire
  Si G = (X ; U'∪ ui) ne contient pas un cycle  alors
        U'← U'∪ ui
  FinSI
  Si |U'|= m - 1 alors RETOURNER Arbre = (X,U') et Quitter
Finpour
Fin
```

Dans l'exemple précédant : On trie les arêtes du graphe. On obtient l'ordre suivant : {7, 8} < {3, 9} = {6, 7} < {1, 2} = {3, 6} < {7, 9} < {8, 9} = {3, 4} < {2, 3} = {1, 8} < {4, 5} < {5, 6} < {2, 8} < {4, 6}.

On ajoute alors : successivement dans l'Arbre les arêtes : {7, 8}, {3, 9}, {6, 7}, {1, 2}, {3, 6}, {3, 4}, {2, 3}, {4, 5}.

5.2. Algorithme de Prim

Le principe de Prim est de partir d'un arbre initial réduit à un seul sommet, puis d'augmenter à chaque itération la taille de l'arbre en le connectant au plus proche voisin libre.

***Algorithme* 7**:permet d'extraire l'arbre de poids minimum

Entrées : G=(X ; U) orienté, Sortie : **Arbre = (X, U')**

Début

1. Supprimer toutes les boucles.

2. Supprimer toutes les arêtes parallèles entre deux sommets sauf l'arête au poids minimum.

3. **X'** ← {}

4. Choisir arbitrairement un sommet ***s*** non traité et ajouter le à **X'**

X' ← **X'** + {s}

5. Examiner toutes les arêtes connectées à l'arbre **X'** et Choisir l'arête u_i avec le poids plus faible.

Ajouter cette arête à l'arbre

U' ← **U'** + u_i

Si tous les sommets sont examinés : X' = X **alors**

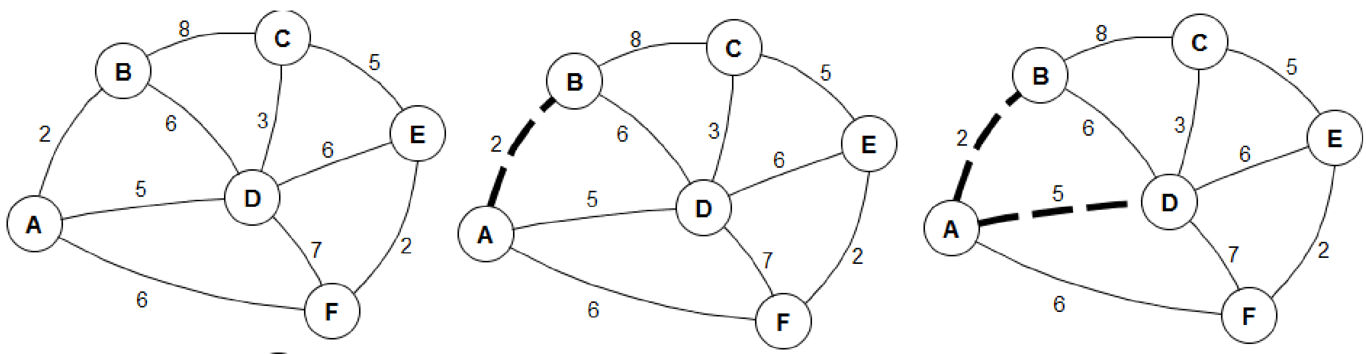
Retourner **Arbre = (X, U')**

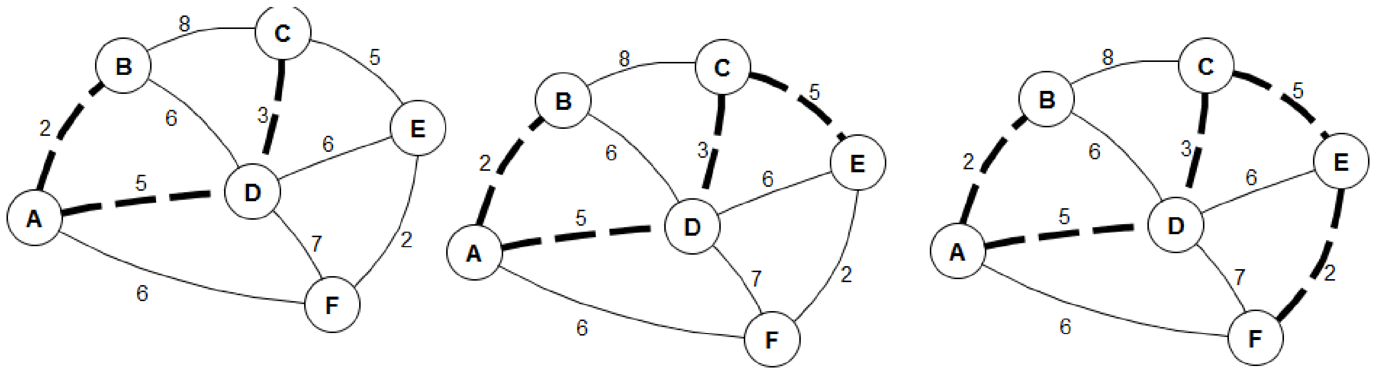
Sinon

Allez à 4

FinSi

Fin





5.3. Algorithme de SOLLIN

L'idée de cet algorithme est une combinaison entre l'algorithme de Kruskal et l'algorithme de Prim. Le principe est de réduire G, à chaque fois que l'on en choisit une arrête, on fusionne les nœuds que cette arête relie. Ainsi, il ne reste plus qu'un sommet à la fin.

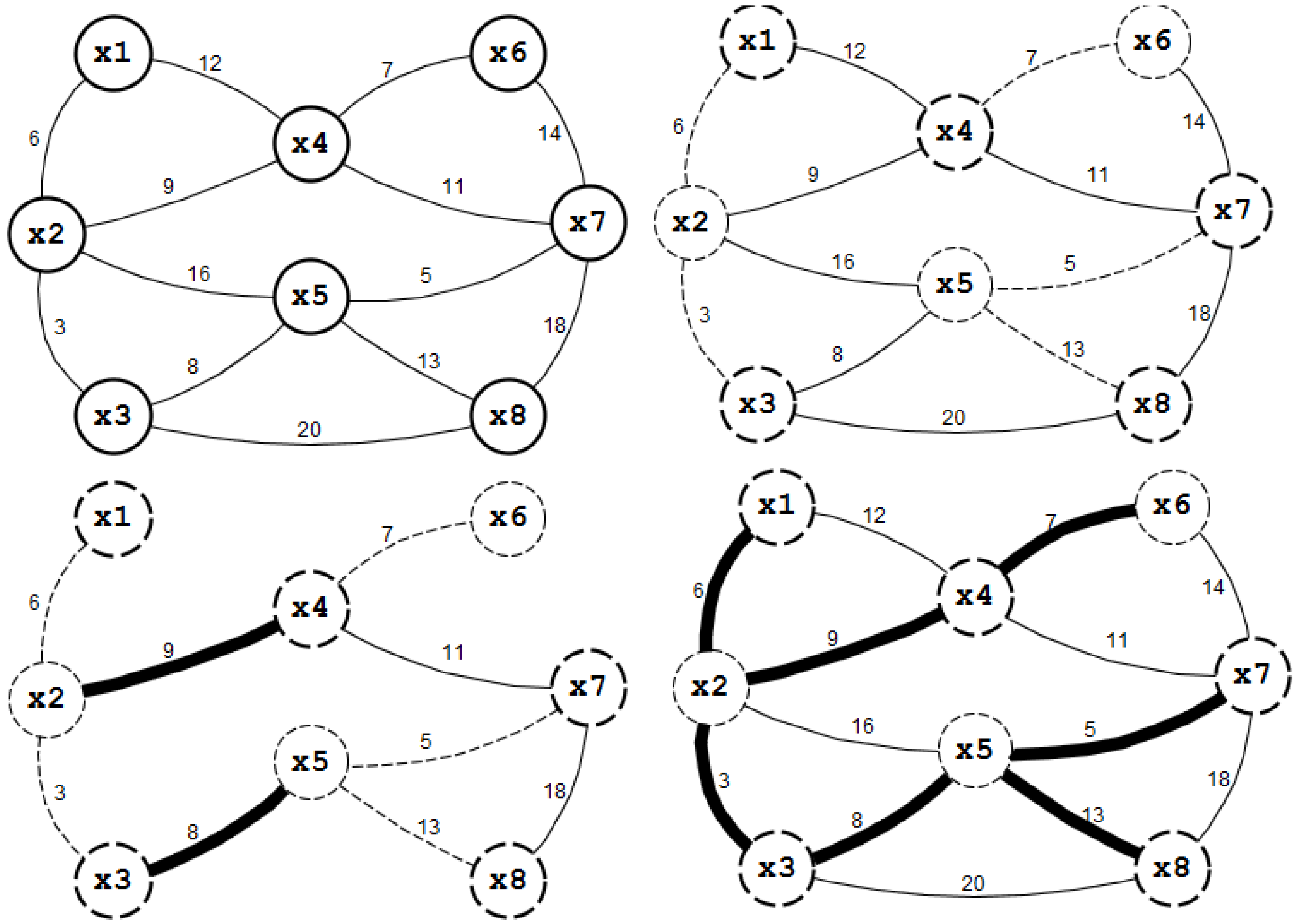
Algorithme 8 : extraire l'arbre de poids minimum
Entrées : G = (X ; U) orienté, Sortie : **Arbre = (X, U')**

Début

  U' ← {},
  Répéter
    1. X' ← **X**
    2. Supprimer toutes les boucles.
    3. Supprimer toutes les arêtes parallèles entre deux sommets sauf l'arête au poids minimum.
    Tant que **X'** ≠ ∅ **faire**
      a. Choisir arbitrairement un sommet ***s*** non traité, ***s*** ∈ *X'*
      b. Examiner toutes les arêtes connectées au sommet ***s*** et Choisir l'arête u_i avec le poids le plus faible.
      c. Ajouter cette arête à l'arbre : **U'** ← **U'** + u_i
      d. Retirer de **X'** les extrémités de u_i
    Fin TQ
    Fusionner les sommets de la même composante en un seul sommet
  Jusqu'à ***G est réduit à un seul sommet.***
  Retourner **Arbre = (X, U')**

Fin

Exemple :


x1
x2
x3
x4
x5
x6
x7
x8
12
7
6
14
9
11
16
5
3
18
8
13
20
x1
x2
x3
x4
x5
x6
x7
x8
12
7
6
14
9
11
16
5
3
18
8
13
20
x1
x2
x3
x4
x5
x6
x7
x8
7
6
9
11
5
3
18
8
13
x1
x2
x3
x4
x5
x6
x7
x8
12
7
6
14
9
11
16
5
3
18
8
13
20