CHAPITRE 2.

STRUCTURES ARBORESCENTES.

2.1 DEFINITIONS.

2.1.1 Arbres.

C'est un graphe non orienté, connexe, acyclique.

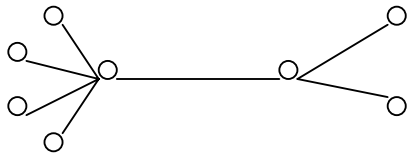


FIG. 2.1. Arbre.

Un arbre comprend n – 1 arêtes. L'addition à un arbre d'une arête entre deux sommets crée un cycle et un seul.

2.1.2 Forêts.

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

2.1.3 Arborescence.

C'est un graphe orienté où chaque sommet possède un seul précédent sauf un qui n'en a pas : la RACINE. Pour tout x de X, il existe un chemin unique de la racine à x.

On considère un nœud x d'une arborescence T, de racine r.

- Un nœud y quelconque sur le chemin unique de r à x est appelé ANCETRE de x ; x est un DESCENDANT de y.
- Si le dernier arc sur le chemin de r vers x est (y, x), alors y est le **père** de x, x est un **fils** de y. Si deux nœuds ont le même père, ils sont **frères**. Un nœud sans fils est une **feuille**. Un noeud qui n'est pas une feuille est dit un **noeud interne.**
- La longueur du chemin entre r et x est la **profondeur** de x dans T.

- **La hauteur d'un noeud x** est définie récursivement de la façon suivante :
  h(x) = 0 si x est la racine.
  h(x) = 1 + h(y) si y est le père de x.
- **Degré d'un noeud & Degré d'une aborescence**.
  - ❖ Degré d'un noeud est le nombre de ses sous-aborescences.
  - ❖ Degré d'une aborescence est le degré maximal des noeuds. Si une aborescence T a le degré m, T est dit l' aborescence à m- aires.

- Si chaque nœud a au maximum deux **fils**, on parle d'arborescence binaire.

EXEMPLE. Arborescence 3-aires de 8 nœuds, de hauteur 4 avec la racine.

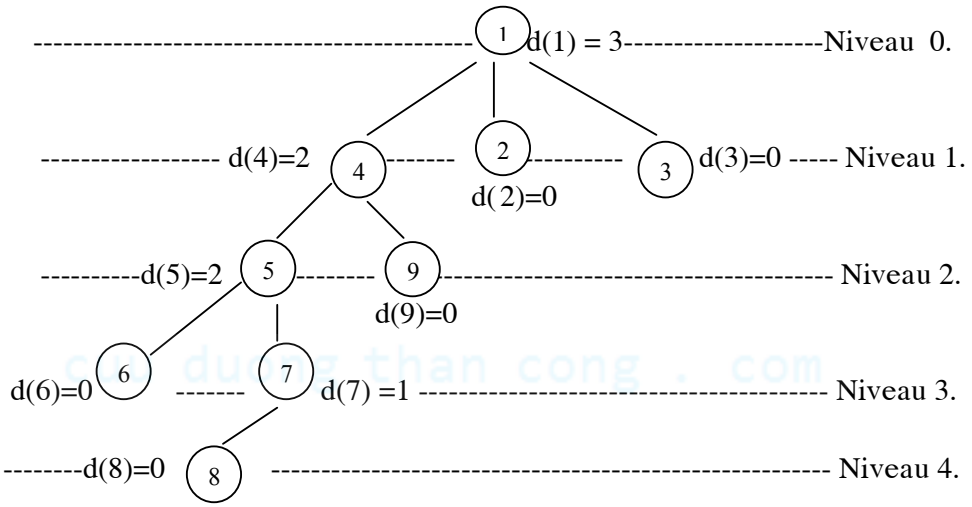
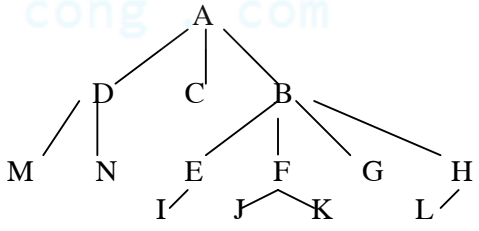


FIG.2.2.

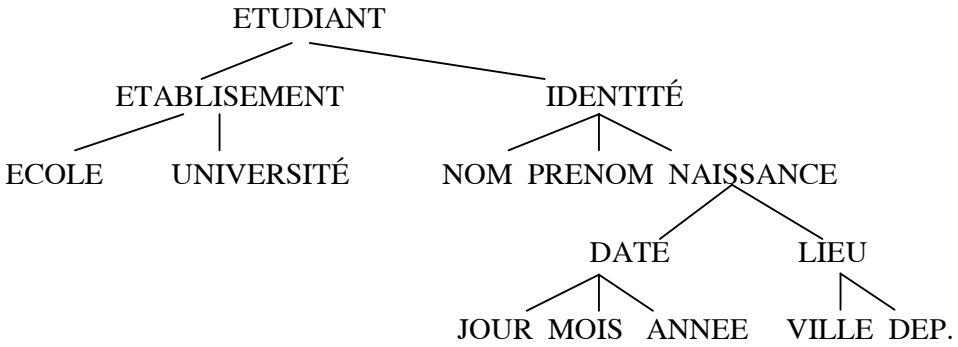
2.1.4. EXEMPLE.

- On peut parfois représenter une relation d'inclusion entre plusieurs ensembles par une aborescence :

B, C, D ⊂ A.
E, F, G, H ⊂ B.
M, N ⊂ D.
I ⊂ E.
J,K ⊂ F.
L ⊂ H.



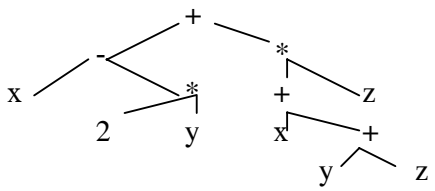
- **Une variable structurée** peut être représentée sous forme d'un arbre. Par exemple :



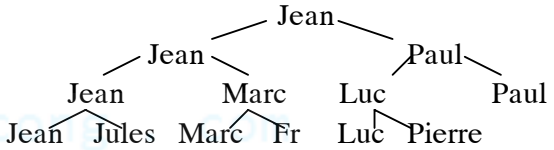
- **Une expression arithmétique**

  X = (x – (2* y) +((x+(y+z)) *z)

  A pour représentation :

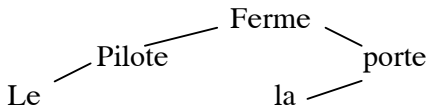


- **Les résultats d'un tournoi de tennis :**
  - **Premier tour**. Marc a battu François, Jean a battu Jules, et Luc a battu Pierre.
  - **Deuxième tour**. Jean a battu Marc et Paul a battu Luc.
  - Jean a gagné en final contre Paul.



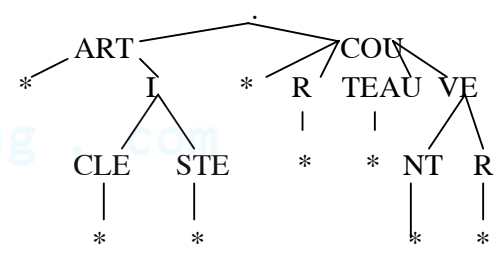
- **Les Phrases d'une langue naturelle** (ou d'un langage de programmation)**.**

  La phrase « Le Pilote ferme la porte »
  peut se représenter sous la forme :



- **Le dictionaire « aborescence »**.

  Par exemple, le dictionaire composé des mots ART, ARTICLE, ARTISTE, COU, COUR, COUTEAU, COUVE,COUVENT,COUVER peuvent se représenter par la figure suivante. Le caractère « * » indique la fin d'un mot. On notera que l'ordre alphabétique est respecté de gauche à droite à chaque niveau.



2.2 PROPRIETES FONDAMENTALES.

2.2.1 THEOREME 1.

Soit G un arbre d'ordre n > 1. Les propriétés suivantes sont équivalentes :

1. G est connexe et sans cycle.
2. G est connexe et admet n – 1 arêtes.
3. G est sans cycle et admet n – 1 arêtes.
4. G est sans cycle et en ajoutant une arête entre deux sommets non adjacents, on crée un cycle (et un seul).
5. G est connexe et en supprimant une arête quelconque, il n'est plus connexe.
6. Tout couple de sommets est relié par une chaîne et une seule.

2.2.2 THEOREME 2.

Un graphe G = (X,U) admet un graphe partiel qui soit un arbre si et seulement si il est connexe.

2.2.3 THEOREME 3.

Toute arborescence est un arbre.

2.3 ARBRES BINAIRES.

2.3.1. DEFINITION (EN RECURSIVE).

Un arbre binaire est soit vide (noté ∅) soit de la forme :

$B = < O, B_1, B_2 >$ où :

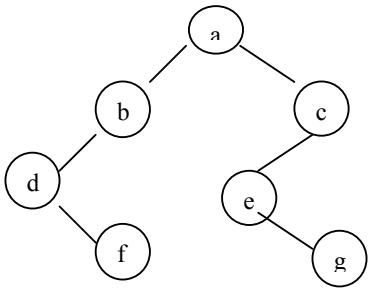
O : racine,

B_1 : sous arbre gauche et

B_2 : sous arbre droit.

2.3.2. REPRÉSENTATION DES ARBRES BINAIRES.

EXEMPLE.



❖ **UTLISATION DE TABLEAU.**

```
Type  Arbtab  = Array [1..n] of Record   v : t ;
                                         G : integer ;
                                         D : integer ;
                                End ;
```

		Gauche	Droit
1			
2	d	0	8
3	a	5	6
4	e	0	9
5	b	2	0
6	c	4	0
7			
8	f	0	0
9	g	0	0
10			

❖ **UTILISATION DE POINTEURS :**

```
Type   Pt   =   ^nut ;
       nut  =   Record
                    G   : Pt ;
                    Val : t ;
                    D   : Pt ;
                End ;
```

2.3.3. PARCOURS D'UN ARBRE BINAIRE.

Nous nous limitons ci-dessous à trois parcours classiques suivants :

1. **PREFIXÉ(en préordre).**
   - Le traitement de la racine.
   - Le parcours du sous arbre gauche.
   - Le parcours du sous arbre droit.

2. **INFIXÉ.**
   - Le parcours du sous arbre gauche.
   - Le traitement de la racine.
   - Le parcours du sous arbre droit.

3. **POSTFIXÉ (SUFFIXÉ).**
   - Le parcours du sous arbre gauche.
   - Le parcours du sous arbre droit.
   - Le traitement de la racine.

EXEMPLE. Pour le graphe de l'exemple ci-dessus, on a :

1. Parcours préfixé : a b d f c e g
2. Parcours infixé : d f b a e g c
3. Parcours suffixé : f d b g e c a

2.4 ARBRES DE RECOUVREMENT.

2.4.1. DÉDINITION.

Soit G un graphe non orienté. Un arbre H est dit **l'arbre de recouvrement** de G si H est sous arbre partiel de G et contenant tous les noeuds de G.

2.4.2. THÉORÈME.

Un graphe G a un arbre de recouverement si et seulement si G est connexe.

2.4.3. ALGORITHME DE RECHERCHE DE L'ARBRE DE RECOUVREMENT.

Considèrons un graphe G.

ALGORITHME.

- **1er étape.** H := { un noeud quelconque de G}.
- **2è étape.** Si tous les noeuds de G appartiennent à H , l'algorithme termine.
- **3è étape.** Si non, choisir un noeud de G, relié à un noeud de H par une arête. Ajouter ce noeud à H. Retouner à la 2è étape.

EXEMPLE . Considèrons le graphe G de la figure suivante :

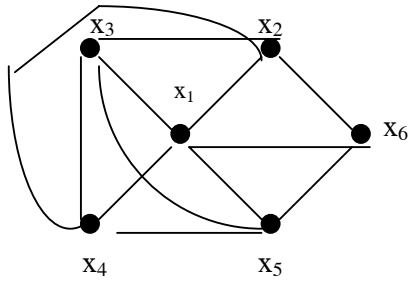


FIG. 2.3.

- À partir de x_1. T= ∅.
- **1er étape**. Choisir x_2, T = {(x_1,x_2)}.
- **2è étape**. Choisir x_3, T = {(x_1,x_2), (x_2,x_3)}.
- **3è étape**. Choisir x_4, T = {(x_1,x_2), (x_2,x_3), (x_3,x_4)}.
- **4è étape**. Choisir x_5, T = {(x_1,x_2), (x_2,x_3), (x_3,x_4), (x_4,x_5)}.
- **5è étape**. Choisir x_6, T = {(x_1,x_2), (x_2,x_3), (x_3,x_4), (x_4,x_5), (x_5,x_6)}.

Résultat : T est un arbre de recouvrement du graphe G .

2.4.4. THÉORÈME.

Soit H un arbre de recouvrement du graphe G.

Ajouter à H une arête du G n'appartenant pas à H, on a un cycle du H. Supprimer une arête quelconque de ce cycle, on a un nouvel arbre de recouvrement du graphe G.

2.4.5. ALGORITHME DE JUSTIFICATION DE CONNEXITÉ.

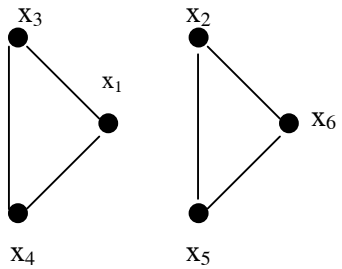
Considèrons un graphe non orienté G.

Appliquer l'algorithme ci-dessus à G. Alors, après la termination de l'algorithme:

- Si H contenant tous les noeuds du G, alors G est connexe et H est un arbre de recouvrement du graphe G.
- Sinon, G n'est pas connexe et H est un arbre de recouvrement d'une composante connexe du graphe G.

EXEMPLE 1 . Dans le cas du graphe G de la figure FIG. 2.3. , on a G connexe.

EXEMPLE 2. Soit G un graphe de la figure suivante :



- À partir de x_1. T= ∅.
- **1er étape**. Choisir x_3, T = {(x_1,x_3)}.
- **2è étape**. Choisir x_4, T = {(x_1,x_3), (x_3,x_4)}.
  - L'algorithme se termine. T est un arbre de recouvrement d'une composante connexe du graphe G.

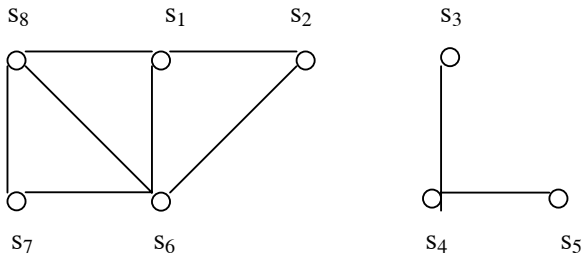
2.4.6. ALGORITHME DE RECHERCHE DE COMPOSANTES CONNEXES.

À l'aide de parcours en profondeur PROF(s), on peut visiter tous les noeuds appartenant à la même composante connexe du noeud s, alors le nombre de composantes connexes est égal au nombre de l'appel de cette procedure. On peut améliorer cette procedure PROF(s) pour indiquer les noeuds de même composante connexe comme suit :

```
PROCEDURE PROF(k :integer) ;
//Parcours en profondeur à partir du noeud k
Int i;
{
      Mark[k]:= Nocomp;
      for (i =1; i≤ n ; i++)
            if (a[i,k]==1) && (Mark[i]= =0) PROF(i);
 }

PROCEDURE CONNEXE ;
Int i ;
{//Initialisation de Mark (des noeuds a déjà marqué) et Nocomp (nombre de composantes connexes)
   for (j= 1 ;j≤n ; j++) { Mark[j] =0 ; Nocomp =0 ;}
  //Appel de la procedure pour determiner des composantes connexes
   for (i =1; i≤ n ; i++)
      If (Mark [i] = =0) { Nocomp =Nocomp +1 ; PROF(i) ;}
  }
End ;
```

EXEMPLE.



- À partir de s_1 . Appel de DFS(1) , on a l'ensemble marqué $\{s_1, s_2, s_6, s_7, s_8\}$.
- i= 3 Appel de DFS(3) , on a l'ensemble marqué $\{s_3, s_4, s_5\}$.

- **Résultat.** On a deux composantes connexes.
  $\mathbf{C_1} = \{s_1, s_2, s_6, s_7, s_8\}$.
  $\mathbf{C_2} = \{s_3, s_4, s_5\}$.

2.5 ARBRES DE RECOUVREMENT MINIMAUX.

PROBLEME 1. Considérons un graphe G = (X,U) connexe, et, à toute arête u, associons un nombre l(u) que nous appellerons sa **longueur**. Il s'agit de trouver un arbre partiel H=(X,V) du graphe d'une longueur totale $\sum_u l(u)$ minimum.

EXEMPLE. Ce problème se rencontre très souvent en télécommunications et en des occasions diverses. Posons nous, par exemple, la question suivante : quelle est la plus courte longueur de câble nécessaire pour relier entre elles n villes données ? Les villes sont alors les sommets du graphe, et l(x, y) est la distance kilométrique séparant les villes x et y. Le réseau de câbles cherché doit être connexe, et, puisque il est de longueur minimum, il n'admet pas de cycles : c'est donc un arbre. On cherche ici l'arbre le plus « court » possible qui soit un graphe partiel du graphe complet de n sommets.

Etablissons tout d'abord un lemme.

LEMME. Si G=(X,U) est un graphe complet, et si les longueurs l(u) associées aux arêtes sont toutes différentes, le Problème 1 admet une solution et une seule (X,V) ; l'ensemble $V=\{v_1,v_2,\ldots,v_{n-1}\}$ est obtenu de la façon suivante :on prend pour v_1 la plus courte arête ; pour v_2 la plus courte arête telle que $v_2 \neq v_1$ et $V_2 = \{v_1,v_2\}$ ne contienne pas de cycles;
pour v_3 la plus courte arête telle que $v_3 \neq v_2 \neq v_1$ et $V_3 = \{v_1,v_2,v_3\}$ ne contienne pas de cycles ; etc…

2.5.1. Algorithme de PRIM (pour le graphe non orienté, valué et connexe).

Notations :

- ♦ M = L' ensemble de noeuds non marqués .
- ♦ Pr(p) = L'ensemble des sommets précédant p à chaque étape.
- ♦ d = L'ensemble des distance à chaque étape.
- ♦ Mark = L'ensemble des noeuds marqués.

PRINCIPE DE L'ALGORITHME.

- ❖ On part d'un arbre initial T réduit à un seul sommet s (e. g. ; s =1)
- ❖ Ensuite, à chaque itération, on augmente l'arbre T en le connectant au «Plus proche » sommet libre au sens des poids.

En détaillé, on a comme suit :

1. Au départ du noeud 1. M = {2,…n}
2. À chaque itération, Choisir un noeud à marquer :c' est le noeud qui a la plus courte distance.
   - ❖ $k = Argmin_{x \in M}$ d[x], c'à d d[k] = Min { d[x] : x ∈ M}
   - ❖ Mises à jour d[i], Pr[i] avec i∈ M \{k} à l'aide de la formule:
     - d[i] = l[k,i] si d[i] > l[k,i].
     - Pr[i] = k.
   - ❖ Remplacer M := M\{k}.

   Si M = ∅. L' algorithme se termine, sinon retourner à 2.

PROCEDURE PRIM ;

```
❖ //Suppose que l' on a la matrice de longuers l est Stocké sous la forme de matrice
  d'adjacence
❖ //Initialisations de M, d, Pr, Mark
  for (i= 1 ; i≤ n ;i++)
        {d[i] = l(1,i) ; pr[i] :=1 ; Mark[i] :=0 ;}
  Mark[1] :=1 ; n0 :=n-1 ;
❖ WHILE (n0 > 0)
  {
     k:= Argmin {d[i] : i∈ M} ;
  //Remise à jour d, Pr, M et Mark
  Mark[k] :=1 ;
     ∀ i ∈ M { d[i] := l[k,i] si d[i] > l[k,i].
                  Pr[i] = k.}
        //Supprimer le noeud k
        M := M\{k} ;
  }END WHILE ;
```

Complexité : O(m log n).

EXEMPLE. Voir FIG. 2.3.

Les étapes de l'algorithme comme suivant :

- **Initialisation : M, d, Pr :**
  - M = { 2, 3, 4, 5, 6}
  - d = [0, 2, 3, 11, 5, 8]
  - Pr = [1, 1, 1, 1, 1, 1]

- **1er étape.** Choisir s_2. Remise à jour M, d, Pr :
  M = { , 3, 4, 5, 6}
  d = [0, 2, 1, 10, 5, 8]
  Pr = [1, 1, 2, 2, 1, 1]

- **2è étape.** s_2 est le sommet actuel. Choisir s_3. Remise à jour M, d, Pr :
  M = { , 3, 4, 5, 6}
  d = [0, 2, 1, 6, 5, 8]
  Pr = [1, 1, 2, 3, 1, 1]

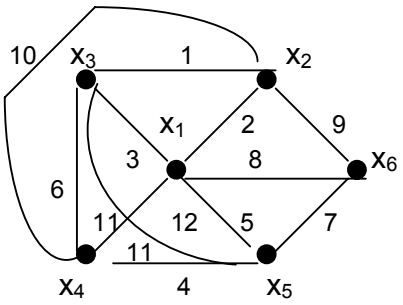
- **3è étape.** s_3 est le sommet actuel. Choisir s_5. Remise à jour M, d, Pr :
  M = { , 3, 4, 5, 6}
  d = [0, 2, 1, 4, 5, 7]
  Pr = [1, 1, 2, 5, 1, 5]

- **4è étape.** s_5 est le sommet actuel. Choisir s_4. Remise à jour M, d, Pr :
  M = { , 3, 4, 5, 6}
  d = [0, 2, 1, 4, 5, 7]
  Pr = [1, 1, 2, 5, 1, 5]

- **5è étape.** s_4 est le sommet actuel. Choisir s_7. Algorithme se termine car M = ∅.

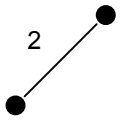
T = {(x_1,x_2) ,(x_2,x_3) ,(x_5,x_4), (x_1,x_5), (x_5,x_6)}
l(T) = { 2, 1, 4, 5, 7,}

Somme de poids minimal = 19.

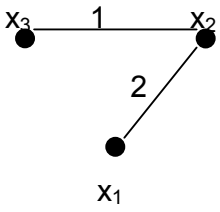


Arbre de départ.

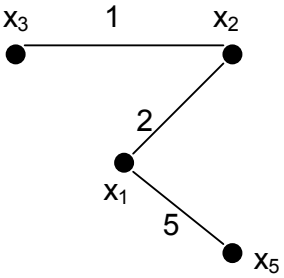
Arbre initial



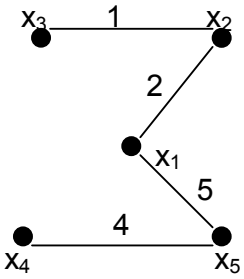
1 ère arête



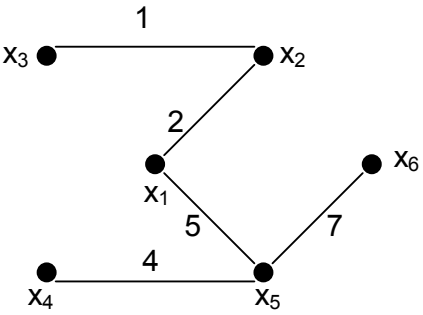
2 ème arête



3 ème arête



4 ème arête



5 ème arête.

FIG. 2.3. Recherche d'un arbre à coût minimum par Prim (s=1).

2.5.2. Algorithme de KRUSKAL (1956).

On procédera par étapes en choisissant chaque fois la plus courte arête qui ne forme pas de cycles avec les arêtes déjà choisies.

On s'arrête lorsque tous les sommets du graphe sont connectés ou, ce qui revient au même, lorsque le nombre d'arêtes retenues égale n – 1. C'est un algorithme **glouton,** i.e., il fait un choix optimal localement dans l'espoir que ce choix mènera à la solution optimale globalement. Ici, il rajoute à chaque étape l'arête de poids minimal à la forêt qu'il construit. L'arbre obtenu est unique si toutes les arêtes sont initialement de valeurs différentes.

Complexité : O(m log m).

EXEMPLE. Voir FIG. 2.3.

$U=\{(x_2, x_3),(x_1,x_2),(x_1,x_3),(x_4,x_5),(x_1,x_5),(x_3,x_4), (x_5,x_6),(x_1,x_6),(x_2,x_6),(x_2,x_4),(x_1,x_4),(x_3,x_4)\}$

$L(U) = \{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$

Les étapes de l'algorithme comme suivant :

- **1er étape.** $T= \{(x_2, x_3)\}$, $L(T) = \{ 1\}$
- **2è étape**. $T= \{(x_2, x_3),(x_1,x_2)\}$, $L(T) = \{ 1, 2 \}$
- **3è étape** . $T= \{(x_2, x_3),(x_1,x_2),),(x_4,x_5)\}$, $L(T) = \{ 1, 2 , 4 \}$
- 4è **étape**. $T= \{(x_2, x_3),(x_1,x_2),),(x_4,x_5) ,(x_1,x_5)\}$, $L(T) = \{ 1, 2 , 4, 5 \}$
- 5è **étape**. $T= \{(x_2, x_3), (x_1,x_2),),(x_4,x_5) ,(x_1,x_5) , (x_5,x_6)\}$

Algorithme se termine car $Card(T) = 5 = 6$ (noeuds) –1.
Somme de poids minimal = 19.

REMARQUE. Sur cet exemple, on retrouve l'arbre à coût minimum calculé par l'algorithme de PRIM. Dans le cas général, on peut trouver un arbre différent, mais de même poids.

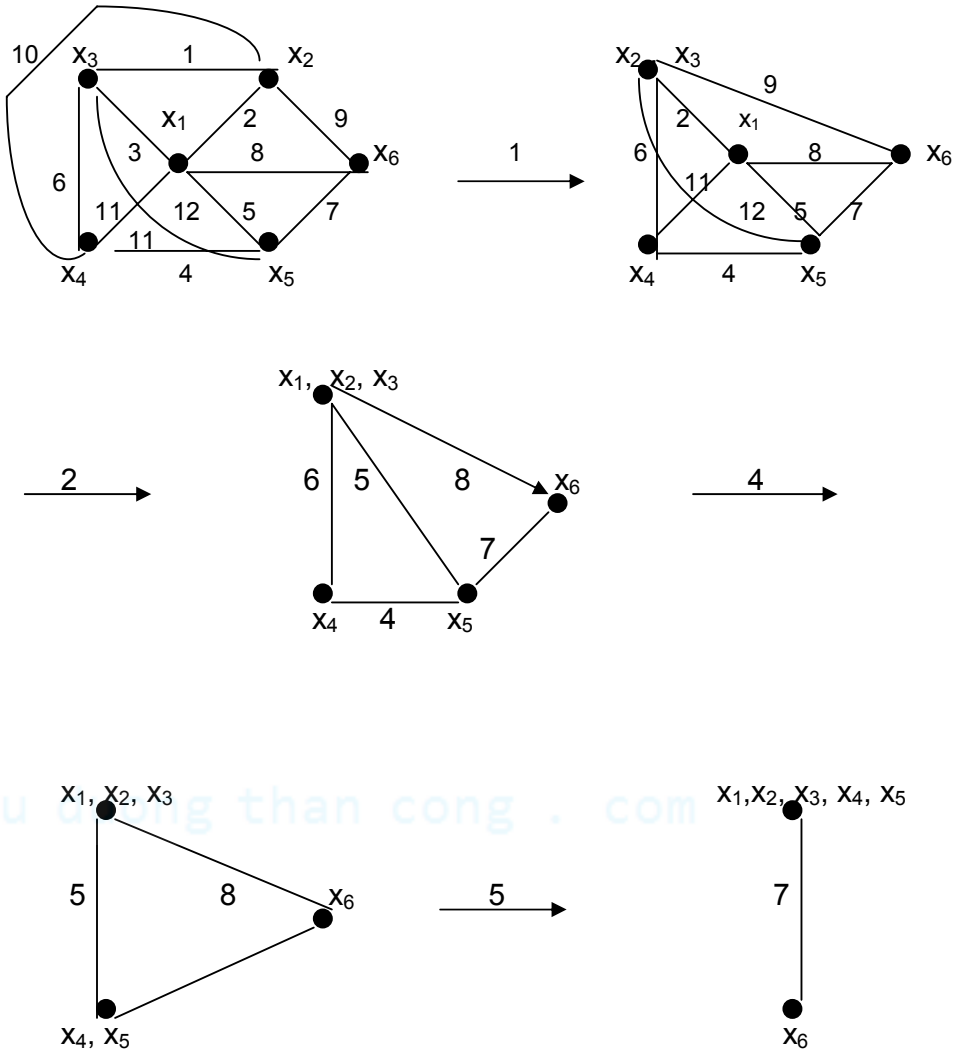


FIG. 2.4. Recherche d'un arbre à coût minimum par Kruskal.