Chapitre 6

Les graphes

6.1 Introduction

La notion de graphe est une structure qui permet de représenter plusieurs situations réelles, en but de leur apporter des solutions mathématiques et informatique, tel que :

- Les réseaux de transport (routiers, ferrés, aériens $\cdots$),
- Les réseaux téléphoniques, électriques, de gaz,$\cdots$ etc,
- Les réseaux d'ordinateurs,
- Ordonnancement des tâches,
- Circuits électroniques,
- $\ldots$

Les arbres et les listes linéaires chaînées ne sont que des cas particuliers des graphes.

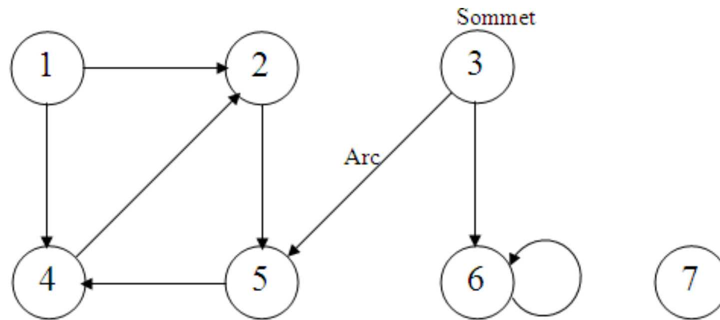
6.2 Définitions

6.2.1 Graphe

Un graphe est défini par un couple (S, A) où S est un ensemble de sommets (nœuds ou points) et A est un sous ensemble du produit cartésien $(S \times S)$ représentant les relations existant entre les sommets.

Exemple : $S = 1, 2, 3, 4, 5, 6, 7$

$A = (1, 2), (1, 4), (2, 5)(3, 5), (3, 6), (4, 2), (5, 4), (6, 6)$

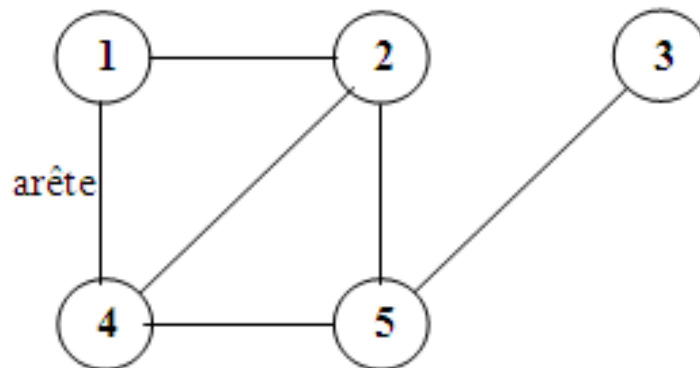


6.2.2 Graphe orienté

C'est un graphe où les relations entre les sommets sont définies dans un seul sens (exemple précédent). Dans ce cas les relations sont appelées "arcs".

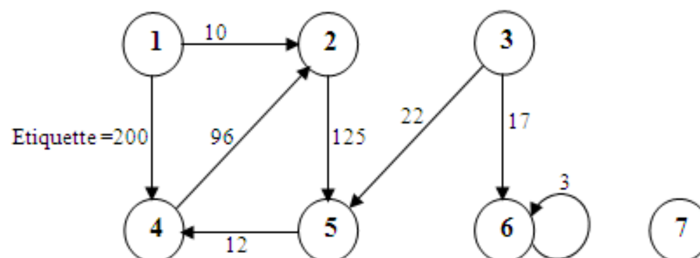
6.2.3 Graphe non orienté

C'est un graphe où les relations sont définies dans les deux sens. Dans ce cas, les relations sont appelées "arêtes".



6.2.4 Graphe étiqueté ou pondéré

C'est un graphe orienté ou non où à chaque arc ou arête correspond une valeur ou une étiquette représentant son coût (ou distance).



6.2.5 Origine et extrémité

Si $a = (x, y)$ est un arc de x vers y alors :

- x est l'origine de a et y est son extrémité
- x est un prédécesseur de y et y est un successeur de x

Si l'origine et l'extrémité d'un arc se coïncident on l'appelle une boucle (6,6).

6.2.6 Chemin

Un chemin est un ensemble d'arcs $a_1, a_2, \cdots, a_p$ où Origine(a_{i+1}) = Extrémité(a_i), $1 \leq i \leq p$

On dit que le chemin est de longueur $p - 1$

Exemple : $\{(1,2),(2,5),(5,4)\}$

6.2.7 Circuit

Un circuit est un chemin $a_1, a_2, \cdots, a_p$ où Origine(a_1) = Extrémité(a_p)

Exemple : $\{(2,5),(5,4),(4,2)\}$

6.2.8 Chaîne

Une chaîne est un chemin non orienté.

Exemple : $\{(1,4),(5,4),(3,5)\}$

6.2.9 Cycle

Un cycle est une chaîne fermée.

Exemple : $\{(1,2),(2,5),(5,4),(1,4)\}$

6.2.10 Graphe connexe

Un graphe connexe est un graphe où pour tout couple de sommets (x, y), il existe une chaîne d'arcs les joignant.

Exemple : Pour le couple $(1,6)$, il existe une chaîne d'arcs, et il n'en existe pas pour (1,7).

6.2.11 Graphe fortement connexe

Un graphe fortement connexe est un graphe où pour tout couple de sommets (x, y), il existe un chemin d'arcs les joignant.

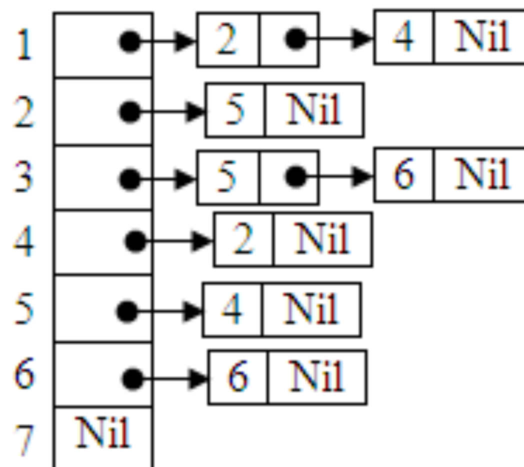
Exemple : Pour $(3,2)$ il existe un chemin $\{(3,5),(5,4),(4,2)\}$ mais il n'en existe pas pour $(2,3)$.

6.3 Représentation des graphes

Les graphes peuvent être représentés en deux manières : en listes d'adjacence (dynamique) ou en matrice d'adjacence (statique)

6.3.1 Listes d'adjacence

Dans cette représentation, les successeurs d'un nœud sont rangés dans une liste linéaire chainée. Le graphe est représenté par un tableau T où $T[i]$ contient la tête de la liste des successeurs du sommet numéro i. Le graphe (S,A) présenté au début de cette section peut être représenté comme suit :



Les listes d'adjacence sont triées par numéro de sommet, mais les successeurs peuvent apparaître dans n'importe quel ordre.

```
Type  TMaillon = Structure
        Successeur : entier ;
        Suivant : Pointeur(TMaillon) ;
Fin ;
Var  Graphe : Tableau[1..n] de Pointeur(TMaillon) ;
```

6.3.2 Matrice d'adjacence

Dans cette représentation le graphe est stocké dans un tableau à deux dimensions de valeurs booléennes ou binaires. Chaque case (x, y) du tableau est égale à *vrai* (1) s'il existe un arc de x vers y, et *faux* (0) sinon. Dans la matrice suivante, on représente le graphe précédent (1 pour *vrai* et 0 pour *faux*) :

	1	2	3	4	5	6	7
1	0	1	0	1	0	0	0
2	0	0	0	0	1	0	0
3	0	0	0	0	1	1	0
4	0	1	0	0	0	0	0
5	0	0	0	1	0	0	0
6	0	0	0	0	0	1	0
7	0	0	0	0	0	0	0

Il est clair que la matrice d'adjacence d'un graphe non orienté est symétrique puisque chaque arête existe dans les deux sens.

Var Graphe : **Tableau[1..n,1..n] de booléen ;**

Pour un graphe étiqueté les valeurs de la matrice peuvent être les étiquettes elles-mêmes, avec une valeur particulière pour les arcs inexistant. Par exemple le graphe étiqueté de l'exemple peut être représenté comme suit :

	1	2	3	4	5	6	7
1	-1	10	-1	200	-1	-1	-1
2	-1	-1	-1	-1	125	-1	-1
3	-1	-1	-1	-1	22	17	-1
4	-1	96	-1	-1	-1	-1	-1
5	-1	-1	-1	12	-1	-1	-1
6	-1	-1	-1	-1	-1	3	-1
7	-1	-1	-1	-1	-1	-1	-1

La représentation matricielle permet de tirer des conclusions intéressantes sur les graphes en utilisant les calculs matriciels.

6.4 Parcours de graphes

De même que pour les arbres, il est important de pouvoir parcourir un graphe selon certaines règles, cependant, le parcours des graphe est un peut différent de celui des arbres. Dans un arbre, si on commence à partir de la racine on peut atteindre tous les nœuds, malheureusement, ce n'est pas le cas pour un graphe où on est obligé de reprendre le parcours tant qu'il y a des sommets non visités. En plus un graphe peut contenir des cycles, ce qui conduit à des boucles infinies de parcours.
Il existe deux types de parcours de graphes : le parcours en profondeur d'abord (Depth First Search) et le parcours en largeur d'abord (Breadth First Search).

6.4.1 En profondeur d'abord (DFS)

Le principe du DFS est de visiter tous les sommets en commençant par aller le plus profondément possible dans le graphe.

```
Var  Graphe : Tableau[1..n,1..n] de booleen ;
     Visité : Tableau[1..n] de booleen ;
Procédure DFS( Sommet : entier ;);
var  i : entier ;
Début
   Visité[Sommet] ← Vrai ;
   Afficher(sommet) ;
   Pour i de 1 à n faire
      Si (Graphe[sommet,i] et non Visité[i]) Alors
         DFS(i)
      Fin Si;
   Fin Pour;
Fin;

Appel :
Pour s de 1 à n faire
   Si (Non Visité[s]) Alors
      DFS(s)
   Fin Si;
Fin Pour;

Trace : 1 2 5 4 3 6 7
```

La procédure DFS peut être utilisée pour divers objectifs :

- Pour vérifier s'il existe un chemin d'un sommet s_1 vers un autre s_2 en initialisant le tableaux Visité à faux et en appelant DFS(s_1) pour ce sommet. Si Visité[s_2] est à vrai à la fin de l'appel de DFS, alors un chemin existe entre s_1 et s_2.
- Pour vérifier si un circuit contentent deux sommets s_1 et s_2 existe, en appelant DFS(s_1) pour un tableaux Visité1 et DFS(s_2) pour un tableaux Visité2, si Visité1[s_2] et Visité2[s_1] sont à Vrai après l'appel alors un tel circuit existe.
- De la même manière pour vérifier si un graphe est cyclique (contient des circuits) ou non.
- Pour trouver les chemins minimums d'un sommet s vers tous les autres.

6.4.2 En largeur d'abord (BFS)

Dans ce parcours, un sommet s est fixé comme origine et l'on visite tous les sommet situés à une distance k de s avant de passer à ceux situés à $k+1$. On utilise pour cela une file d'attente.

```
Var  Graphe : Tableau[1..n,1..n] de booleen ;
     Visité : Tableau[1..n] de booleen ;
     F : File d'attente ;
Procédure BFS( Sommet : entier ;);
Var  i,s : entier ;
Début
 | Enfiler(F,Sommet) ;
 | Tant que (non File_Vide(F)) faire
 |  | Defiler(F,s) ;
 |  | Afficher(s) ;
 |  | Pour i de 1 à n faire
 |  |  | Si (Graphe[s,i] et non Visité[i]) Alors
 |  |  |  | Enfiler(F,i) ;
 |  |  | Fin Si;
 |  | Fin Pour;
 | Fin TQ;
Fin;

Appel :
Pour s de 1 à n faire
 | Si (non Visité[s]) Alors
 |  | BFS(s)
 | Fin Si;
Fin Pour;

Trace : 1 2 4 5 3 6 7
```

6.5 Plus court chemin (Algorithme de Dijkstra)

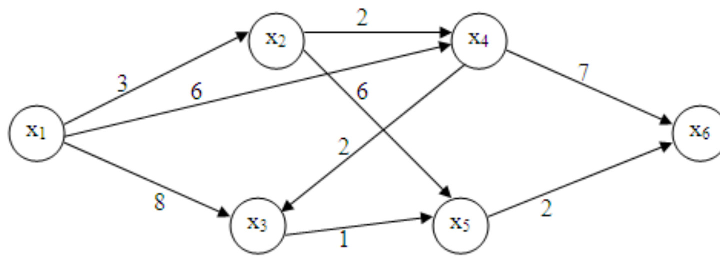
Soit un graphe $G(S,U)$ orienté sans boucles avec n sommets,
$(x_i, x_j) \in U \Rightarrow l_{ij}$ = longueur de l'arc (x_i, x_j)
La longueur d'un chemin $\mu = \sum_{(x_i,x_j)\in\mu} l_{ij}$
On peut trouver plusieurs chemins reliant un sommet à un autre. Le problème du chemin minimum ou de plus court chemin consiste à trouver le chemin de moindre coût. On trouve plusieurs algorithmes pour la résolution de ce problème : Dijkstra, Ford, Belman-Kalaba,$\cdots$ etc.

6.5.1 Algorithme de Dijkstra

L'algorithme de Dijkstra résout le problème de la recherche d'un plus court chemin à origine unique pour un graphe orienté pondéré $G = (S,U)$ dans le cas où tous les arcs ont un poids positif ou nul.
L'algorithme de Dijkstra maintient à jour un ensemble D des sommets de S dont le plus court chemin à partir de l'origine s est connu et calculé. A chaque itération, l'algorithme choisit parmi les sommets de (S - D) c'est-à-dire parmi les sommets dont le plus court chemin à partir de l'origine n'est pas connu, le sommet x dont l'estimation de plus court chemin est minimale. Une fois un sommet u choisi, l'algorithme met à jour, si besoin est, les estimations des plus courts chemins de ses successeurs (les sommets qui peuvent être atteint directement à partir de u).

1- $D = x_i, \lambda_i = 0$ // x_i sommet de départ
  $\lambda_j = l_{ij}$, $j \neq i$ (si un arc (x_i, x_j) n'existe pas $l_{ij} = +\infty$)
  //λ_j coût du chemin minimum entre x_i et x_j
2- $\lambda_k = min\ \lambda_j$ / $x_j \notin D$
3- $D = D \cup \{x_k\}$
  $\lambda_j = min\{\lambda_j, \lambda_k + l_{kj}\}$
4- Aller à 2 si $D \neq S$

Exemple :



D	λ_1	λ_2	λ_3	λ_4	λ_5	λ_6
x_1	0	3	8	6	∞	∞
x_1, x_2	0	3	8	5	9	∞
x_1, x_2, x_4	0	3	7	5	9	12
x_1, x_2, x_4, x_3	0	3	7	5	8	12
x_1, x_2, x_4, x_3, x_5	0	3	7	5	8	10
$x_1, x_2, x_4, x_3, x_5, x_6$	0	3	7	5	8	10