CHAPITRE 4 : LES GRAPHES

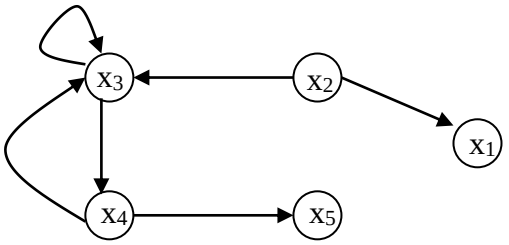
Plan

1. Introduction aux graphes

Les graphes sont actuellement l'outil privilégié pour modéliser des ensembles de structures complexes. Ils sont indispensables pour représenter et étudier des relations entre les objets.

Les graphes sont utilisés en :

- Economie (diagramme d'organisation)
- Electronique (circuits intégrés)
- Base de données (liens entre informations)
- Communications (réseaux de télécommunications)
- Transport (réseaux routiers)
- Ordonnancement (structure des projets)
- Etc.



2. Définitions

2.1 Graphes orientés (Directed Graphs)

- Un graphe $\boldsymbol{G}$ est un couple $(\boldsymbol{X}, \boldsymbol{U})$ où :
  - X est un ensemble $\{x_1, \dots, x_n\}$ de ***nœuds ou sommets***.
  - $U = \{u_1, u_2, \dots, u_m\}$ est une famille de couples ordonnées de sommets appelées ***arcs***.

- Un graphe est dit **valué** s'il ∃ une application $C : U \rightarrow R$, associant à chaque arc u un réel c_u.

- Une **boucle** est un arc reliant un nœud à lui-même.

- Chaque arc $u = (x, y)$ a deux extrémités appelées **initiale** (x) et **terminale** (y). u est dit **incident intérieurement** à x et **incident extérieurement** à y.

- Dans un arc de la forme (x, y), x est appelé **prédécesseur** de y et y est dit **successeur** de x. x et y sont appelés sommets voisins (ou adjacents).

- L'ensemble des successeurs de x est noté $\Gamma(x)$ et celui de ses prédécesseurs est noté $\Gamma^{-}(x)$.

- Le nombre de successeurs de x est appelé **demi degré extérieur** et est noté $d_G^+(x) = |\Gamma(x)|$. Le **demi degré intérieur** de x est défini comme étant $d_G^-(x) = |\Gamma^-(x)|$. Le **degré** de x est défini comme suit : $d_G(x) = d_G^+(x) + d_G^-(x)$.

- La **densité** d'un graphe est définie par le rapport $\boldsymbol{m/n^2}$ c'est-à-dire le nombre actuel d'arcs de G divisé par le nombre maximum d'arcs que peut avoir G. La plupart des graphes rencontrés en pratique ne sont pas très dense (à faible densité). Ils sont appelés creux (en Anglais sparse).

2.2 Graphes non orientés

Dans les applications où on s'intéresse uniquement aux paires de sommets reliées et non à leurs orientations, on parle de graphes non orientés. Dans ce type de graphe, on parle d'une arête (edge en Anglais) $e = [x, y]$ au lieu de l'arc (x, y). Un graphe non orienté est noté $G = (X, E)$ où E désigne une famille d'arêtes. Les termes qui s'appliquent aux graphes orientés et sont indépendants de l'orientation restent valables pour les graphes non orientés.

2.3 Parcours

- On appelle **chemin** μ de longueur p toute suite de p arcs $(u_1, \dots, u_p)$ telle que l'extrémité initiale de u_i est égale à l'extrémité terminale de u_{i-1} $\forall\, i > 1$, et l'extrémité terminale de u_i est égale à l'extrémité initiale de u_{i+1} $\forall\, i < p$.

- On appelle **circuit** tout chemin fermé, c'est-à-dire un chemin tel que $u_1 = u_p$.

- Un arc est un chemin de longueur 1 et une boucle est un circuit de longueur 1 aussi.

- Si le graphe est non orienté, on parle de **chaine** au lieu de chemin, et de **cycle** au lieu de circuit.

- Un **parcours** est un élément de l'ensemble des **chemins**, **circuits**, **chaînes** et **cycles**.

2.4 Graphes particuliers

- Un graphe orienté $G = (X, U)$ est dit **symétrique** si $(x, y) \in U \Longrightarrow (y, x) \in U$. Les graphes non orientés peuvent être représentés par des graphes orientés symétriques.

- Le graphe **complémentaire** de $G = (X, U)$ est le graphe $H = (X, X^2 - U)$.

- Un graphe est dit **complet** si toute paire de sommets est connectée par un arc ou une arête.

3. Représentation des graphes

3.1 Matrice d'adjacence

Considérons un graphe orienté $G = (X, U)$ dont le nombre de sommets est n et celui des arcs étant m. Une **matrice d'adjacence** est une matrice M $(n \times n)$ dont les éléments sont booléens indiquant si les sommets correspondants sont reliés.

Un graphe valué $G = (X, U, W)$ peut être représenté par une matrice W $(n \times n)$ qui donne à la fois les coûts des arcs et fait fonction de la matrice d'adjacence. Les matrices d'adjacence permettent de détecter les boucles, la symétrie ainsi que la connectivité. On peut facilement à l'aide de cette structure de données obtenir la liste des successeurs d'un sommet ainsi que celle de ses prédécesseurs. Cependant, ces structures ne sont pas adéquates pour les graphes dont la densité est faible.

Exemple : Le graphe donné en exemple à la section 2.1. est représenté par la matrice d'adjacence M :

	x_1	x_2	x_3	x_4	x_5
x_1	0	0	0	0	0
x_2	1	0	1	0	0
x_3	0	0	1	1	0
x_4	0	0	1	0	1
x_5	0	0	0	0	0

3.2 Listes d'adjacence

a. Listes Contigües

Les listes d'adjacence implémentent la structure $G = (X, \Gamma)$ où les sommets sont rangés consécutivement dans des tableaux ou bien dans des listes chaînées. Dans le cas des tableaux, on note par **Succ** le tableau dont les éléments sont les listes des successeurs des sommets $0,1, \dots, n-1$ dans l'ordre, c'est-à-dire $\Gamma(0), \Gamma(1), \dots, \Gamma(n-1)$. Le nombre d'éléments de **Succ** est donc le nombre d'arcs du graphe à savoir m. Pour délimiter ces listes successives des successeurs, on fait usage d'une structure **Tête** sous forme de tableau à n éléments, qui donne pour chaque sommet l'indice dans le tableau **Succ** où commence ses successeurs. En effet, les successeurs d'un sommet y sont rangés dans **Succ** de **Tete[y]** à **Tete[y+1]-1**. Dans le cas où un sommet y n'a pas de successeurs, on pose **Tete[y]=Tete[y+1]**.

Les graphes non orientés sont codés comme des graphes orientés symétriques. Si $[x, y]$ est une arête, x apparaît dans la liste des successeurs de y, et y dans celle de x. Dans le cas d'un graphe valué $G = (X, U, W)$, on fait usage d'un tableau de poids **W** en regard du tableau **Succ**.

L'avantage majeur de ce codage étant sa compacité. En effet, un graphe consomme $n + m + 1$ mots mémoire qui est bien moins que n^2 mots d'une matrice dans le cas où G n'est pas dense.

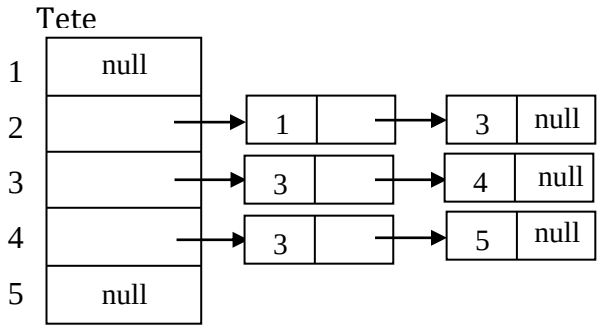
Exemple : (Graphe section 2.1)

1	2	3	4	5	6	
1	1	3	5	7	7	Tete

1	3	3	4	3	5	
1	2	3	4	5	6	Succ

b. Listes Chaînées

Le même principe peut être appliqué en utilisant des listes chaînées comme suit :



Là aussi, il y a lieu d'ajouter pour chaque bloc une case supplémentaire dans le cas de graphe valué pour stocker le poids de l'arc.

4. Parcours des graphes

4.1 Construction des listes de prédécesseurs

a. Représentation en matrice d'adjacence

Soit M la matrice d'adjacence d'un graphe G d'ordre n. Les successeurs d'un sommet quelconque i peuvent être obtenus en parcourant la ligne i de la matrice M en $O(n)$ opérations. Les prédécesseurs sont obtenus en parcourant la colonne i également en $O(n)$.

b. Représentation en listes d'adjacence

Pour les graphes de faible densité, il est avantageux de coder le graphe par des structures linéaires représentant les listes d'adjacence. En effet, la structure nécessite seulement $O(m+n)$ mots-mémoire et la liste des successeurs ne nécessite que $O(d^+(i))$ opérations. Cependant, les listes d'adjacence ne peuvent pas être manipulées efficacement pour retrouver la liste des prédécesseurs d'un sommet i. Pour retrouver la liste des prédécesseurs, il faut balayer tout le graphe pour détecter les sommets dont i est successeur et ceci nécessite $O(m)$ opérations et $O(nm)$ opérations pour construire les listes des prédécesseurs de tous les sommets. Dans le cas où la liste des prédécesseurs est requise fréquemment, il est plus avantageux, en termes de complexité temps, de construire le graphe inverse G^- de G où les listes des successeurs ne sont autres que les listes de prédécesseurs dans G. Dans ce cas on consomme $2(m+n)$ mots pour les deux graphes, mais c'est nettement inférieur à n^2 pour les graphes de faible densité.

4.2. Exploration des Graphes

L'exploration consiste à déterminer l'ensemble des descendants d'un sommet s, c'est-à-dire l'ensemble des sommets situés sur des chemins d'origine s. Le principe de l'exploration d'un graphe peut être décrit comme suit : Au début on marque le sommet s, ensuite à chaque fois qu'on rencontre un arc (x, y) avec x marqué et y non marqué, on marque alors y.

a. Exploration en largeur (BFS : Breadth-First Search)

Dans ce type d'exploration, on implémente l'ensemble des sommets marqués comme étant une file F. Les sommets sont marqués par ordre de nombre d'arcs croissant à s : on commence par les successeurs de s, puis les successeurs des successeurs de s, etc. Un sommet x en tête de la file F reste tant que ses successeurs ne sont pas examinés. Pendant ce temps, tout successeur de x non marqué est marqué et rangé en fin de F.

Algorithme :

```
Marquer s ; Enfiler(F,s)
Repeter  ……N
  x = Defiler(F)
  pour tout successeur y non marqué de x ……∑d⁺(i) = M
   Enfiler(F,y)
   Marquer y
  Finpour
Jusqu'à FileVide(F)
```

Complexité : O(N+M) ≈ O(M)

b. Exploration en Profondeur (DFS : Depth-First Search)

Il s'agit cette fois ci d'implémenter l'ensemble des sommets marqués comme étant une pile P et l'exploration progresse en se déplaçant le plus loin possible le long d'un chemin dont s est l'origine avant de rebrousser chemin. La pile P contient les sommets explorés et permet à la procédure de rebrousser chemin.

Algorithme :

```
Marquer s ; Empiler(P,s)
Repeter
  Tant qu'il y a un successeur y à TetePile(P) et non marqué faire
    Marquer y
    Empiler(P,y)
  Fintq
  x = Depiler(P)
Jusqu'à PileVide(P)
```

Idem pour la complexité que BFS.

c. Exemple

Soit un graphe G orienté, dont les sommets sont (s_1, s_2, s_3, s_4, s_5, s_6) représenté par la matrice d'adjacence suivante. Donner les ordres de parcours en largeur BFS et en profondeur DFS, à partir du sommet s1.

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

BFS :

File	s1	s6 s3 s2	s5 s4 s6 s3	s5 s4 s6	s5 s4	s5	File Vide
Sommets Marqués	s1	s2,s3,s6	s4,s5				

Ordre de défilement : s1,s2,s3,s6,s4,s5

DFS :

Pile	s1	s2 s1	s4 s2 s1	s3 s4 s2 s1	s5 s3 s4 s2 s1	s6 s5 s3 s4 s2 s1	s5 s3 s4 s2 s1	s3 s4 s2 s1	s4 s2 s1	s2 s1	s1	Pile Vide
Sommets Marqués	s1	s2	s4	s3	s5	s6						

Ordre de dépilement : s6,s5,s3,s4,s2,s1

5. Problème des chemins optimaux

5.1. Typologie, algorithmes et applications des problèmes des chemins optimaux

Soit $G = (X, A, W)$ un graphe orienté valué et considérons le coût d'un chemin comme étant la somme des coûts de ses arcs. Les principaux problèmes consistent à trouver des chemins de coût minimal. Dans ce contexte, il y a lieu de distinguer les trois problèmes suivants :

1. Soient *s*, *t* deux sommets, trouver un plus court chemin de *s* à *t*.
2. Trouver un plus court chemin de *s* vers tout autre sommet de G.
3. Trouver un plus court chemin entre tout couple de sommets.

Dans cette section, on s'intéressera au second problème en calculant pour chaque sommet x la valeur du plus court chemin du sommet de départ vers x ; V[*x*], appelée aussi étiquette ou label.
Il existe une famille d'algorithmes qui calculent V[*x*] d'une manière définitive pour chaque sommet x. Ces algorithmes sont appelés **à fixation d'étiquettes** et le plus répandu de cette classe est l'algorithme de Dijkstra. D'autres algorithmes affinent jusqu'à la dernière itération l'étiquette de chaque sommet *x*. Cette classe est appelée **à correction d'étiquettes**. Il y a lieu de distinguer les cas suivants :

- Cas W constante. Le problème se réduit à celui de la recherche des chemins contenant le plus petit nombre d'arcs qui peut être résolu par une exploration en largeur.

- Cas $W \geq 0$. Le problème peut être résolu par l'algorithme de Dijkstra qui est du type à fixation d'étiquettes et dont la complexité est $O(n^2)$. Il existe une implémentation en structure de tas dont la complexité est $O(n \log n)$.

- Cas W quelconque. Il existe un algorithme dû à Bellman du type à correction d'étiquettes et dont le principe repose sur la programmation dynamique. La complexité étant $O(nm)$ qui peut être réduite à $O(m)$ lorsque le graphe G ne contient pas de circuits.

Les applications des problèmes des chemins optimaux sont nombreuses et diverses. Dans le domaine des transports, par exemple, on s'intéresse aux chemins optimaux d'une ville x à une autre ville y. Dans le routage du trafic réseau, on parle des protocoles OSPF (open **shortest path** first).

5.2. Algorithme de DIJKSTRA

Il fait partie des algorithmes à fixation d'étiquettes et ne peut être appliqué que pour les graphes à valuations **positives**. L'algorithme fixe l'étiquette de chaque sommet *x* à chaque itération en mettant à jour un tableau de booléens.

Lors de sa phase d'initialisation, l'algorithme initialise un tableau V à ∞, P à zéros et D à faux. Pour un sommet donné *s*, on initialise V[*s*] à zéro et P[*s*] à *s*. L'itération principale de l'algorithme est constituée de deux boucles. La première consiste à trouver un sommet non encore fixé de l'ensemble V qui est minimal. Pour cela, la valeur ∞ est d'abord affectée à Vmin, ensuite pour tout *y* allant de 1 à n dont D[*y*] est faux et V[*y*] strictement inférieure à la valeur Vmin, l'algorithme conserve *y* dans une variable *x* et affecte V[*y*] à Vmin. Dans le cas où Vmin n'est pas ∞, D[x] est remplacé par vrai. La seconde boucle consiste à parcourir tous les sommets *k* successeurs de *x* pour mettre à jour la liste des successeurs. Si V[*x*] +W[*k*] est inférieur à V[*y*] alors V[*x*] + W[*k*] est affectée à V[*y*]. Finalement, *x* est affecté à P[*y*].

Algorithme de Dijkstra

```
s : sommet de départ
Initialiser le tableau V à +∞      //Valeur des plus court chemins
Initialiser le tableau P à 0       //Détail des chemins (Path)
V[s]=O
P[s]=s
Répéter
  // Chercher sommet non fixé de V minimal
  Vmin=+∞
  Pour i=1 à n faire
    Si (i non fixé  et  V[i]<Vmin) alors  x=i; Vmin=V[i]  fsi
  Finpour
  Si Vmin<+∞
    Fixer x
    // Mise à jour des successeurs
    Pour chaque successeur y de x faire
      Si V[x]+W[x][y] < V[y]
          V[y] = V[x]+W[x][y];
          P[y] = x
      Fsi
    Finpour
  Fsi
Jusqu'à Vmin=+∞
```

La complexité de l'algorithme de Dijkstra dans le pire des cas est $O(n^2)$. Elle peut être améliorée en O(nlogn) en utilisant un tas pour représenter le tableau V.

Exemple : Soit le graphe orienté valué représenté par la figure suivante. Dérouler l'algorithme de Dijkstra en prenant comme sommet de départ 1.

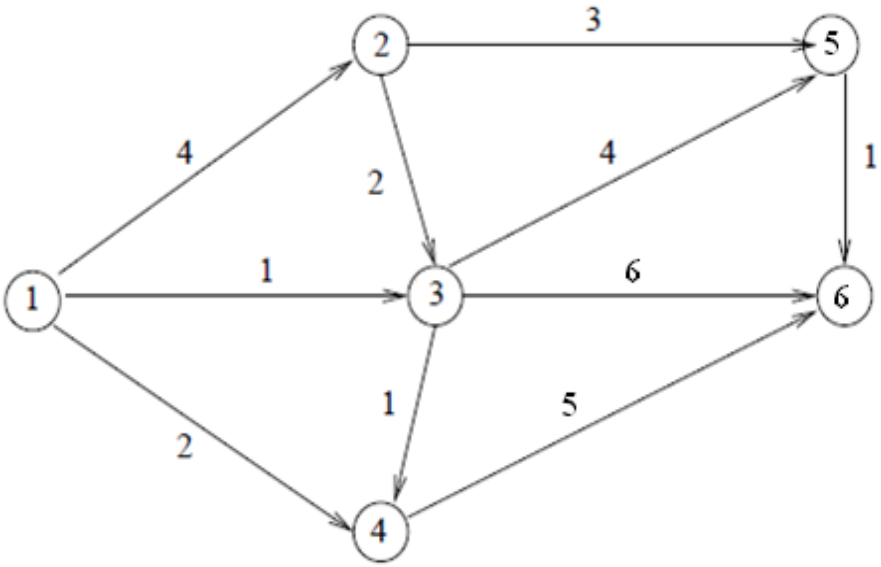


Tableau V (Values)

1	**2**	**3**	**4**	**5**	**6**	**Sommets Marqués**
0	∞	∞	∞	∞	∞	1
	4	1	2	∞	∞	3
	4		2	5	7	4
	4			5	7	2
				5	7	5
					6	6
0	**4**	**1**	**2**	**5**	**6**	**Résultat Final**

Tableau P (Paths)

1	**2**	**3**	**4**	**5**	**6**
1	0 1	0 1	0 1	0 3	0 3 5

On en déduit que les chemins optimaux à partir du sommet 1 sont :

1→1	0
1→2	4
1→3	1
1→4	2
1→3→5	5
1→3→5→6	6