Miage

Théorie des graphes et algorithmes

Joëlle Cohen

8 octobre 2012

Introduction

Ce résumé de cours ne prétend pas être exhaustif ni se substituer en aucune manière aux ouvrages publiés sur ce sujet. Ce document n'a d'autre but que de fournir aux étudiants un support de cours. Après une présentation des notions de base sur les graphes seront abordés quelques problèmes et des algorithmes les résolvant :

- parcours de graphe
- arbre couvrant minimum
- plus court chemin
- détection de circuit dans un graphe orienté , détermination des composantes fortement connexes d'un graphe orienté , tri topologique
- ordonnancement

remarque : dans les algorithmes qui seront décrits, le terme FIN signifie une sortie du programme en cours d'exécution.

Table des matières

Chapitre 1

Généralités sur les graphes

1.1 Définitions

1.1.1 graphe non orienté (graph)

Définition 1.1 *Un graphe non orienté G est un triplet (X, E, φ) où*
- *X est un ensemble non vide de sommets (vertices)*
- *E est un ensemble d'arêtes (edges)*
- *φ est une fonction d'incidence qui à chaque arête de E associe une paire d'éléments de X non nécessairement distincts.*

exemple : Soit G_1 défini par $X = \{a, b, c, d, f, g\}$, $E = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8\}$ et
$\varphi(e_1) = \{a, b\}$, $\varphi(e_2) = \{b, c\}$, $\varphi(e_3) = \{c\}$, $\varphi(e_4) = \{c, d\}$, $\varphi(e_5) = \{b, d\}$, $\varphi(e_6) = \{d, g\}$, $\varphi(e_7) = \{b, g\}$, $\varphi(e_8) = \{b, c\}$.

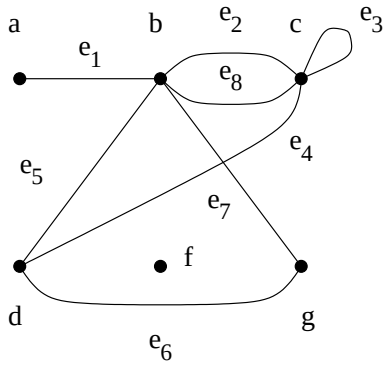


FIGURE 1.1 – le graphe non orienté G_1

On dit les sommets a et b sont adjacents, que l'arête e_4 est incidente au sommet c et au sommet d.

Les arêtes e_2 et e_8 sont parallèles, l'arête e_3 est une boucle, le sommet f est isolé et le sommet a est pendant.

exemple : Soit H_1 défini par $Y = \{u, v, w, x, y, z\}$, $F = \{a, b, c, d, e, f, g, h\}$ et

$\psi(a) = \{x, y\},\ \psi(b) = \{y, w\},\ \psi(c) = \{u, w\}\psi(d) = \{u, y\},\ \psi(e) = \{u, z\},\ \psi(f) = \{z\},\ \psi(g) = \{z, y\},\ \psi(h) = \{z, y\}$.

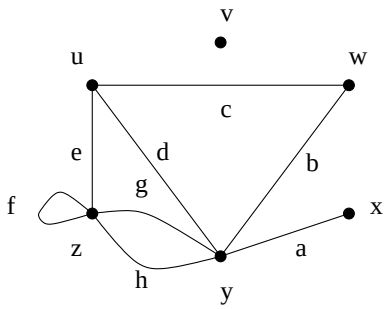


FIGURE 1.2 – le graphe non orienté H_1

Le cardinal de X, lorsque X est fini, est appelé l'*ordre* du graphe.

Par la suite, on ne considèrera que des graphes finis c'est-à-dire X et E seront finis.

1.1.2 graphe non orienté simple

Un graphe non orienté est simple s'il n'a ni arêtes parallèles ni boucle. On identifie alors une arête e à $\varphi(e)$, son image par φ.

Propriété 1.1 *Si n est l'ordre d'un graphe non orienté simple G et m est son nombre d'arêtes alors $m \leq C_n^2$.*

En effet, φ est alors une injection de X dans l'ensemble des parties à deux éléments de X. Or il y a C_n^2 paires d'éléments de X.

1.1.3 graphe orienté fini (digraph)

Définition 1.2 *Un graphe orienté fini G est un triplet (X, E, φ) où*
- *X est un ensemble fini non vide de sommets (vertices)*
- *E est un ensemble fini d'arcs*
- *φ est une fonction d'incidence qui à chaque arc e de E associe un couple (x, y) d'éléments de X non nécessairement distincts. x est l'origine de e et y est le but de e.*

exemple : Soit G_2 (voir figure 1.3) le graphe orienté défini par $X = \{a, b, c\}$, $E = \{e_1, e_2, e_3, e_4, e_5, e_6\}$ et $\varphi(e_1) = (a, b)$, $\varphi(e_2) = (b, c)$, $\varphi(e_3) = (a, b)$, $\varphi(e_4) = (c, b)$, $\varphi(e_5) = (c, c)$, $\varphi(e_6) = (a, c)$.

Le sommet a est l'origine de e_1 et le sommet b est le but de e_1.

Les arcs e_1 et e_3 sont parallèles, l'arc e_5 est une boucle mais les arcs e_2 et e_4 ne sont pas parallèles.

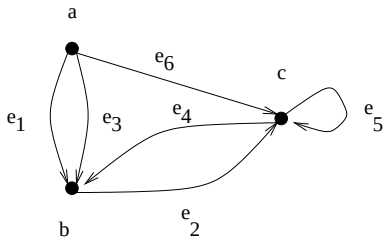


FIGURE 1.3 – graphe orienté G_2

1.2 Isomorphisme de graphes

Définition 1.3 *Soient $G = (X, E, \varphi)$ et $H = (Y, F, \psi)$ deux graphes .*
G_1 et H_1 sont isomorphes s'il existe deux bijections $\theta : X \to Y$, $\tau : E \to F$ telles que
pour tout $e \in E$, $\varphi(e) = xy \Leftrightarrow \psi(\tau(e)) = \theta(x)\theta(y)$

Les accolades - graphe non orienté - ou parenthèses - graphe orienté - nécessaires dans les fonctions d'incidence ont été volontairement omises puisque la définition d'isomorphisme est la même pour les deux types de graphes .

exemple : G_1 et H_1 sont isomorphes. En effet, les bijections suivantes θ et τ vérifient l'équivalence ci-dessus.

θ	
a	x
b	y
c	z
d	u
f	v
g	w

τ	
e_1	a
e_2	g
e_3	f
e_4	e
e_5	d
e_6	c
e_7	b
e_8	h

Par la suite on considérera souvent des graphes à un isomorphisme près : on parle alors de graphes non étiquetés.

exemple : les deux graphes suivants ne sont pas isomorphes.

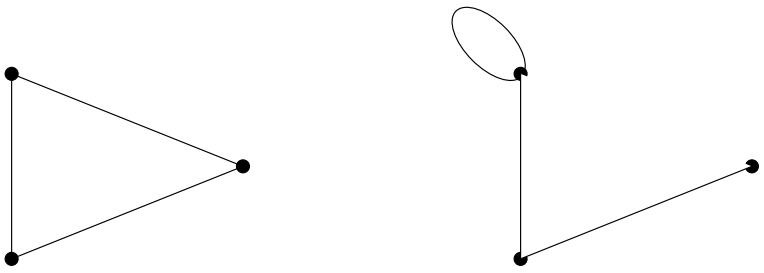


FIGURE 1.4 – deux graphes non isomorphes

remarque : comme le montre l'exemple précédent, deux graphes isomorphes ont le même ordre et le même nombre d'arêtes mais le contraire n'est pas vrai.

exercice : Trouver les 11 graphes simples non isomorphes d'ordre 4.

Définition 1.4 *un graphe est complet s'il est simple et si deux sommets quelconques sont adjacents. On a alors exactement* $\frac{n(n-1)}{2}$ *arêtes si l'ordre du graphe est* n.

exemple : le graphe complet d'ordre 5 est noté K_5.

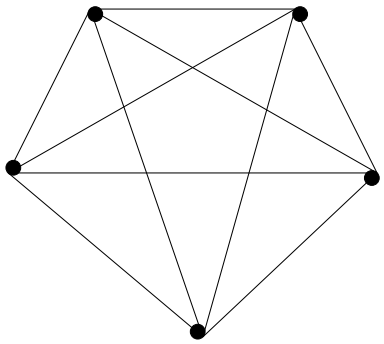


FIGURE 1.5 – le graphe complet d'ordre 5

1.3 Sous-graphes et graphes partiels

Définition 1.5 *Soit* $G(X, E, \varphi)$ *un graphe et* $Y \subset X$. *Le sous-graphe* G_Y *de* G *engendré par* Y *est le graphe dont*
- *l'ensemble des sommets est* Y
- *l'ensemble des arêtes (arcs) sont les arêtes de* G *joignant* 2 *sommets de* Y
- *la fonction d'incidence* φ_Y *est la restriction de* φ *aux arêtes (arcs) de* G_Y.

Définition 1.6 *Soit* $G(X, E, \varphi)$ *un graphe et* $F \subset E$. *Le graphe partiel* G_F *de* G *engendré par* F *est le graphe dont*
- *l'ensemble des sommets est* X
- *l'ensemble des arêtes (arcs) est* F
- *la fonction d'incidence* φ_F *est la restriction de* φ *à* F.

exemple : cf figure 1.6

1.4 Degrés

1.4.1 degré dans un graphe non orienté

Définition 1.7 *le degré* $d(x)$ *d'un sommet* x *de* G *est le nombre d'arêtes incidentes à* x *dans* G *(chaque boucle incidente à* x *compte deux fois).*

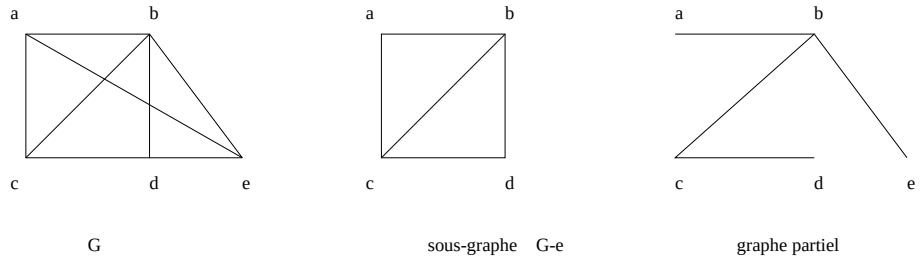


FIGURE 1.6 – sous-graphe et graphe partiel

Propriété 1.2 *dans un graphe non orienté,* $\sum_{x\in X} d(x) = 2m$ *où* m *est le nombre d'arêtes.*

En effet, dans la somme $\sum_{x\in X} d(x)$, chaque arête est comptée deux fois dans $d(x)$ si c'est un boucle sur x et et une fois dans $d(x)$ plus une fois dans $d(y)$ si elle est incidente à x et y.

conséquence : dans un graphe non orienté le nombre de sommets de degré impair est pair.

exemple :

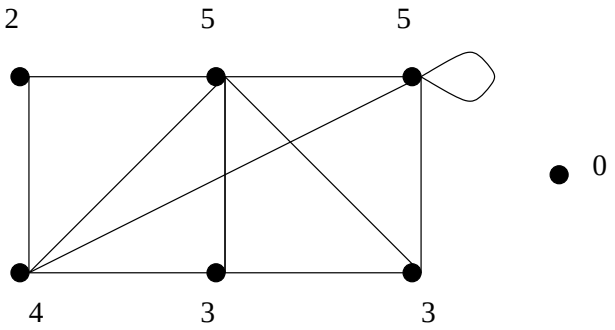


FIGURE 1.7 – Degrés

Définition 1.8 *on dit qu'un graphe non orienté est* $k-$*régulier si tous ses sommets sont de degré* k*.*

Propriété 1.3 *soient* δ *le plus petit degré et* Δ *le plus grand degré d'un graphe non orienté* G *d'ordre* n *à* m *arêtes. On a* $n\delta \leq 2m \leq n\Delta$*.*

Définition 1.9 *On dit qu'une suite d'entiers naturels* $k_1, \cdots, k_n$ *est graphique s'il existe un graphe non orienté simple d'ordre* n *tel que ses sommets* $x_1, \cdots, x_n$ *ont respectivement les degrés* $k_1, \cdots, k_n$*.*

Théorème 1.4 (Havel-Hakimi) *: une suite* **décroissante** *d'entiers naturels* $k_1 \geq \cdots \geq k_n$ *avec* $n \geq 2$ *et* $k_1 \geq 1$ *est graphique ssi la suite* $k_2 - 1, k_3 - 1, \cdots, k_{k_1+1} - 1, k_{k_1+2}, \cdots k_n$ *est graphique. (on a supprimé* k_1 *de la suite puis on a retranché* 1 *aux* k_1 *premiers termes)*

Ce théorème fournit un algorithme pour déterminer si une suite est graphique ou non.

1.4.2 degré dans un graphe orienté

Définition 1.10 *dans un graphe orienté on distingue*
- *le degré entrant $d^-(x)$ égal au nombre d'arcs de but x,*
- *le degré sortant $d^+(x)$ égal au nombre d'arcs d'origine x et*
- *le degré $d(x)$ égal à $d^-(x) + d^+(x)$.*

exemple :

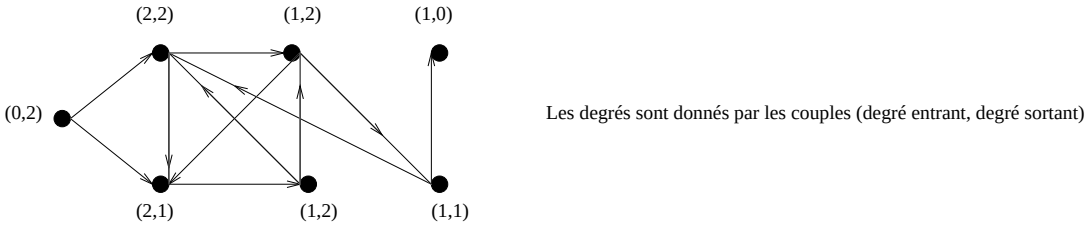


FIGURE 1.8 – Degrés entrants et degrés sortants

On retrouve la propriété 1.2 sous la forme suivante

Propriété 1.5 *dans un graphe orienté, $\sum_{x\in X} d^-(x) = \sum_{x\in X} d^+(x) = m$ où m est le nombre d'arcs.*

1.5 Chaînes - Cycles - Connexité

1.5.1 Définitions

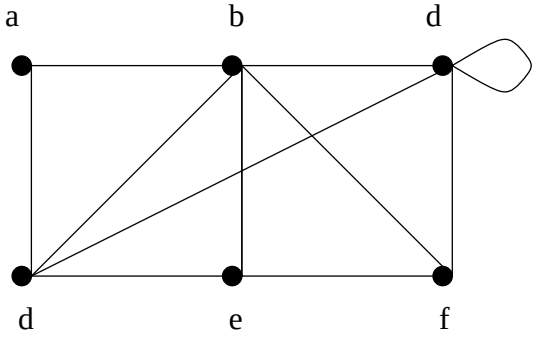
Définition 1.11 *Soit $G = (X, E, \varphi)$ un graphe non orienté.*
- *une chaîne est une suite $x_0e_1x_2e_2x_3\cdots e_kx_k$ ave $\varphi(e_i) = x_{i-1}x_i$ pour $i = 1\cdots k$. On dit que cette chaîne est de longueur k et relie x_0 à x_k.*
- *une chaîne est simple si elle ne contient pas deux fois la même arête.*
- *un cycle est une chaîne simple fermée i.e. $x_0 = x_k$.*
- *une chaîne est élémentaire si elle ne contient pas deux fois le même sommet (sauf aux extrêmités si c'est un cycle élémentaire).*
- *une chaîne est eulérienne si elle est simple et contient toutes les arêtes.*
- *une chaîne est hamiltonienne si elle est élémentaire et contient tous les sommets.*

exemple : dans le graphe G_1, la chaîne $be_2ce_4de_5be_8ce_4de_6g$ n'est pas simple. La chaîne $ge_7be_2ce_4de_6g$ est un cycle. La chaîne $ge_7be_2ce_8be_5d$ est simple mais elle n'est pas élémentaire. G_1 n'a pas de chaîne hamiltonienne puisque le sommet f est isolé. G_1 n'a pas de chaîne eulérienne.

remarque : une chaîne élémentaire est simple mais le contraire n'est pas nécessairement vrai.

Propriété 1.6 *si x et y sont reliées par une chaîne alors il existe une chaîne élémentaire qui relie x à y.*

exemple : dans le graphe de la figure suivante, la chaîne *abde* est élémentaire (les arêtes sont volontairement omises puisqu'en l'absence d'arêtes parallèles il n'y a pas d'ambigüité).



1.5.2 Composantes connexes

Définition 1.12 *La relation "être reliés par une chaîne" est une relation d'équivalence sur l'ensemble des sommets de G. Les sous-graphes engendrés par cette relation sont les composantes connexes de G (ce sont les sous-graphes maximaux par rapport à cette relation).*

La composante connexe $CC(x)$ du sommet x est alors l'ensemble des sommets de G reliés à x par une chaîne sachant que x est relié à lui- même par une chaîne de longueur nulle.

On dit que G est un graphe connexe s'il n'a qu'une composante connexe *i.e.* si deux sommets quelconques sont reliés par une chaîne.

exemple : le graphe G_1 a deux composantes connexes.

Propriété 1.7 *Soit G un graphe non orienté d'ordre n, à m arêtes et à p composantes connexes. On a l'inégalité suivante : $m - n + p \geq 0$.*

1.5.3 Distance dans un graphe non orienté

Définition 1.13 *On définit la distance $d(x, y)$ entre deux sommets x, y d'un graphe non orienté G par :*
- *$d(x, y) = 0$ si $x = y$.*
- *$d(x, y)$ est la longueur d'une plus courte chaîne reliant x à y s'il existe une telle chaîne*
- *$d(x, y) = \infty$ sinon*

Propriété 1.8 *Pour tous sommets x, y, z*
- $d(x, x) = 0$
- $d(x, y) = d(y, x)$
- $d(x, y) \leq d(x, z) + d(y, z)$

C'est une distance au sens métrique du terme.

Définition 1.14 *la diamètre d'un graphe non orienté est le maximum des distances entre deux sommets de ce graphe.*

remarque : dans un graphe non connexe, le diamètre est infini.

exemple : le diamètre du graphe de la figure 1.9 est 2.

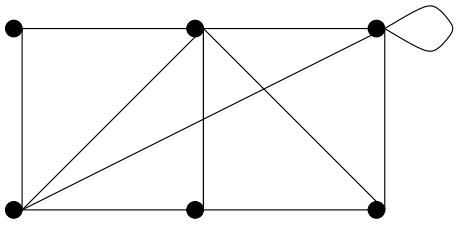


FIGURE 1.9 – Un graphe de diamètre 2

1.6 Chemins - Circuits - Forte Connexité

1.6.1 Définitions

Les définitions sont les mêmes que pour les graphes non orientés en remplaçant chaîne par *chemin* et cycle par *circuit*.

exemple : dans le graphe G_2, le chemin ce_4be_2c est un circuit. Le chemin ae_1be_2c est hamiltonien mais G_2 n'a pas de chemin eulérien.

1.6.2 graphe orienté fortement connexe

Pour un graphe orienté G, la relation "être reliés par un chemin" n'est pas une relation d'équivalence puisqu'elle n'est pas nécessairement symétrique.

Les *composantes fortement connexes* de G sont les sous-graphes maximaux pour cette relation "être reliés par un chemin" et on a la notion suivante :

Définition 1.15 *un graphe orienté G est fortement connexe si pour tout couple de sommets (x, y), il existe un chemin de x à y.*

exemple : G_2 n'est clairement pas fortement connexe. Ses composantes fortement connexes de G_2 sont $\{a\}$, $\{b, c\}$.

exemple : le graphe de la figure 1.10 est fortement connexe.

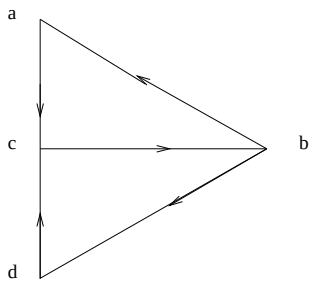


FIGURE 1.10 – un graphe fortement connexe

1.7 Fermeture transitive d'un graphe

Définition 1.16 *La fermeture (ou clôture) transitive d'un graphe simple G est le graphe ayant les mêmes sommets que G et dont les arêtes (arcs) relient x à y s'il existe dans G une chaîne (chemin) de x à y.*

exemple : cf figure 1.11

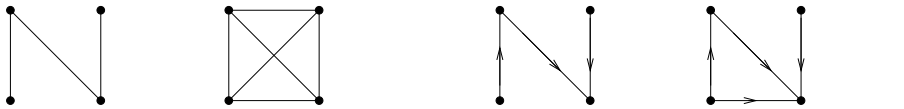


FIGURE 1.11 – fermeture transitive

1.8 Représentation des graphes

On ne parlera que de deux façons de représenter les graphes selon que l'on considère ce graphe comme ensemble d'arêtes ou comme ensemble de sommets.

1.8.1 Matrice d'adjacence

Soit G un graphe d'ordre n dont l'ensemble des sommets est $\{x_1, \cdots, x_n\}$.

La matrice d'adjacence M_a de G est la matrice carrée $n \times n$ définie comme suit : m_{ij} est le nombre d'arêtes (arcs) de x_i à x_j.

remarque :
– si G est non orienté alors M_a est symétrique
– si G est simple, M_a est une matrice de booléens
Une consultation complète de M_a prend un temps $\Theta(n^2)$. Si G a peu d'arêtes (arcs) alors cette représentation est coûteuse en mémoire.

exemple : le graphe G_1 est représenté par la matrice suivante

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 2 & 1 & 0 & 1 \\ 0 & 2 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \end{pmatrix}$$

exemple : le graphe G_2 est représenté par la matrice suivante

$$\begin{pmatrix} 0 & 2 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

1.8.2 Un premier algorithme : algorithme de Roy-Warshall

Cet algorithme utilise la matrice d'adjacence pour calculer la matrice d'adjacence de la fermeture transitive d'un graphe simple et déterminer ainsi la relation d'accessibilité dans le graphe initial.

L'algorithme de Roy-Warshall est basé sur la propriété du paragraphe 1.5.1 : ainsi pour déterminer quels sommets sont reliés par une chaîne il suffit de calculer toutes les chaînes élémentaires de longueur au plus n si n est l'ordre de G.

PROCÉDURE Roy-Warshall (M_a : matrice de booléens)

$A \leftarrow M_a$ {A sera la matrice de la fermeture transitive}

POUR i de 1 à n FAIRE $A_{i,i} \leftarrow 1$ FINPOUR

POUR k de 1 à n FAIRE

POUR i de 1 à n FAIRE

POUR j de 1 à n FAIRE $A_{i,j} \leftarrow A_{i,j} \vee (A_{i,k} \wedge A_{k,j})$ FINPOUR FINPOUR FINPOUR

FINPROCÉDURE

Complexité : $\Theta(n^3)$.

Justification : à la k-ième itération, $A_{i,j} = 1$ *si et seulement si* il existe une chaîne de x_i à x_j ne passant que par $x_1, \cdots x_k$.

1.8.3 Liste d'adjacence

A chaque sommet de G on associe la liste de ses successeurs dans G. L'espace mémoire occupé est alors de l'ordre de $n + 2m$ (pour un graphe non orienté)

exemple : le graphe G_1 est représenté par la liste suivante

$a\bullet \rightarrow b$

$b\bullet \rightarrow a \rightarrow c \rightarrow c \rightarrow d \rightarrow g$

$c\bullet \rightarrow b \rightarrow b \rightarrow c \rightarrow d$

$d\bullet \rightarrow b \rightarrow c \rightarrow g$

$f\bullet$

$g\bullet \rightarrow b \rightarrow d$

exemple : le graphe G_2 est représenté par les listes suivantes

$a\bullet \rightarrow b \rightarrow b \rightarrow c$

$b\bullet \rightarrow c$

$c\bullet \rightarrow b \rightarrow c$

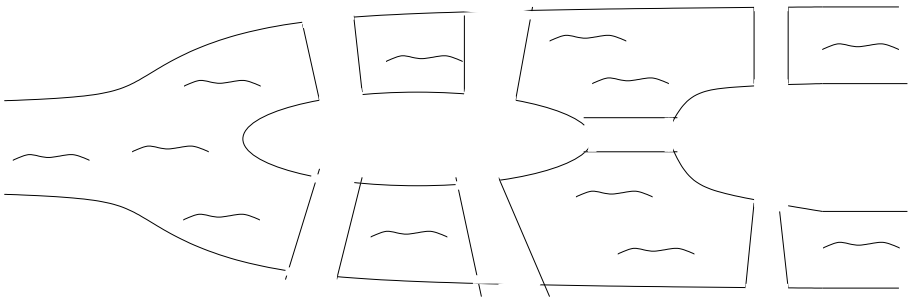


FIGURE 1.12 – les sept ponts de Königsberg

1.9 Quelques notions particulières

1.9.1 Les 7 ponts de Königsberg

Cet exemple historique est important puisque sa résolution par L.Euler initia la théorie des graphes.

Le problème posé était le suivant : peut-on visiter toute la ville en emprutant chaque pont une et une seule fois ? La réponse est non et L.Euler montra pourquoi en établissant le théorème suivant :

Théorème 1.9 *Soit G un graphe non orienté sans sommet isolé. G a une chaîne eulérienne si et seulement si G est connexe et G a 0 ou 2 sommets de degré impair.*

Ce théorème explique pourquoi on peut dessiner d'un seul trait sans repasser la "maison" avec un toit mais qu'il est impossible de le faire pour la maison sans toit (cf figure 1.13).

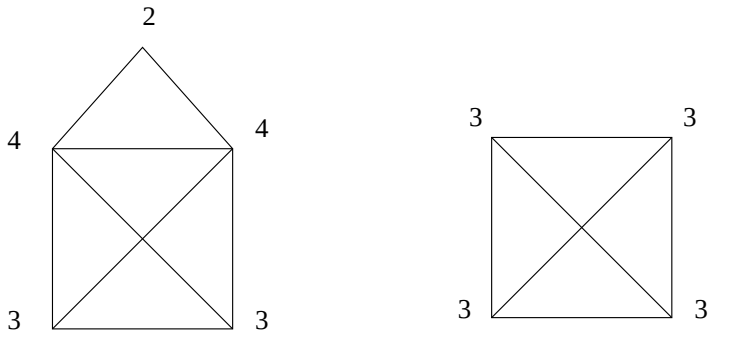


FIGURE 1.13 – les deux maisons

1.9.2 Coloration des sommets

Colorier les sommets d'un graphe consiste à associer une couleur à chaque de façon que deux sommets adjacents n'aient pas la même couleur. Plus formellement,

Définition 1.17 *Soit G un graphe non orienté sans boucle. Une coloration de G est une application $\gamma : X \to C$ de l'ensemble des sommets de G dans un ensemble fini C de couleurs, telle que si une arête e est incidente à x et y alors $\gamma(x) \neq \gamma(y)$.*

Le *nombre chromatique* de G est le nombre minimal de couleurs permettant de colorier G.

exercice : Cinq étudiants A, B, C, D, E doivent passer les examens suivants.

A	Ada, Anglais, Logique
B	Probabilités, Compilation
C	Réseau, Logique
D	Anglais, Probabilités, Réseau
E	Ada, Compilation

Sachant que chaque étudiant ne peut se présenter qu'à une épreuve par jour, quel est le nombre minimal de jours nécessaire à l'organisation de toutes les épreuves ?

Pour résoudre ce problème, on définit le graphe simple non orienté des "incompatibilités" ainsi : les sommets sont les matières et deux sommets sont adjacents si un étudiant doit passer ces deux matières (*i.e.* elles sont incompatibles en e sens qu'elles ne peuvent pas se dérouler la même journée). Un fois ce graphe établi, on le colorie et chaque couleur représente une demie journée nécessaire ; deux épreuves ayant la même couleur donc ne concernant aucun étudiant en commun pourront se dérouler la même journée. Ici la réponse est 3.

Les propriétes suivantes sont faciles à prouver :

Propriété 1.10
- *le nombre chromatique du graphe complet K_n est n,*
- *le nombre chromatique d'un graphe d'ordre n est inférieur ou égal à n,*
- *si Δ est le degré maximal d'un graphe alors son nombre chromatique est inférieur ou égal à $\Delta + 1$,*
- *si un graphe d'ordre n a un sous-graphe complet d'ordre k alors son nombre chromatique est compris entre k et n.*

Dans le paragraphe suivant nous allons nous intéresser aux graphes dont le nombre chromatique est 2.

1.9.3 Graphe biparti

Les graphes non orientés dont le nombre chromatique est 2 peuvent aussi se définir ainsi

Définition 1.18 *Un graphe non orienté G est biparti s'il existe une partition de l'ensemble de ses sommets en deux classes disjointes telle que toute arête a une extrêmité dans chaque classe.*

Ces graphes ont une caractérisation :

Théorème 1.11 *un graphe est biparti si et seulement si il n'a pas de cycle impair.*

Preuve : • Supposons que G soit biparti et ait un cycle impair $x_0e_1x_2e_2x_3\cdots x_{2k}e_{2k+1}x_{2k+1}$ avec $x_{2k+1} = x_0$. Soit $Y \cup Z$ la bipartition des sommets de G. Si $x_0 \in Y$ alors $x_1 \in Z$ et donc $x_2 \in Y$. On obtient $x_0, x_2, \cdots x_{2k} \in Y$ et $x_1, \cdots x_{2k+1} \in Z$. Mais $x_{2k+1} = x_0$: contradiction.

• Réciproquement soit G sans cycle impair. On supposera dans un premier temps que G est connexe et on choisit un sommet x. On définit alors les deux ensembles suivants :

$X_p = \{y \in X \mid$ une plus courte chaîne élémentaire reliant x à y est de longueur paire$\}$ et $X_i = \{y \in X \mid$ une plus courte chaîne élémentaire reliant x à y est de longueur impaire$\}$.

On a bien $X_p \cup X_i = X$ puisque G est connexe. Supposons qu'il existe une arête e incidente à deux sommets y et z de X_p. On a alors une chaîne élémentaire reliant x à y de de longueur paire et une chaîne élémentaire reliant x à z de de longueur paire. On a donc un cycle $x\cdots yez\cdots x$ de longueur impaire (pair + pair + 1) : contradiction. On montre de même qu'il n'existe pas d'arête incidente à deux sommets de X_i. Donc G est biparti.

Si G n'est pas connexe, on applique le raisonnement précédent à ses composantes connexes pour obtenir une bipartition des sommets de G. □

Définition 1.19 *G est biparti complet si G est biparti et si tout sommet d'une partie est adjacent à tout sommet de l'autre partie. On note $K_{p,q}$ le graphe biparti complet dont les deux parties ont respectivement p et q sommets.*

exemple : le graphe de la figure 1.14 est biparti.

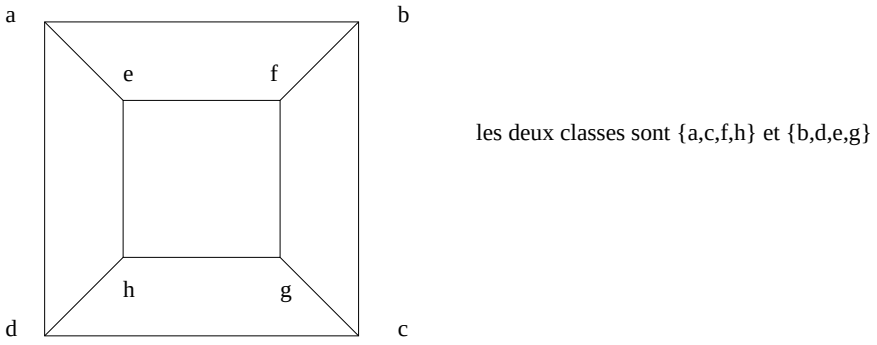


FIGURE 1.14 – Un graphe biparti

exemple : le graphe de la figure 1.15 est biparti complet.

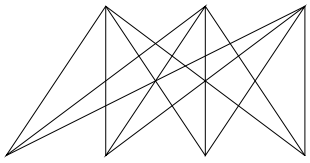


FIGURE 1.15 – Le graphe biparti complet $K_{3,4}$

1.9.4 Coloration des arêtes

On peut aussi colorier les arêtes d'un graphe pour que deux arêtes incidentes à un même sommet n'aient pas la même couleur. Bien cela nécessite que le graphe non orienté n'ait aucune boucle. Plus formellement,

Définition 1.20 *Soit G un graphe non orienté sans boucle. Une coloration des arêtes de G est une application $\gamma : E \to C$ de l'ensemble des arêtes de G dans un ensemble fini C de couleurs, telle que si un sommet x est une extrêmité des arêtes e et e' alors alors $\gamma(e) \neq \gamma(e')$.*

On appelle *indice chromatique* de G le nombre minimal de couleurs permettant de colorier les arêtes de G.

Le problème de coloration des arêtes peut se ramener à celui des sommets pour un graphe en définissant le graphe adjoint G^a ainsi : les sommets de G^a sont les arêtes de G et deux sommets de G^a sont adjacents si, dans G ces deux arêtes ont une extrêmité communes.

exercice : 5 amis font un tournoi d'échecs, dont les parties se déroulent en temps limité d'une heure, chacun devant rencontrer les quatre autres. Combien d'heures nécessite ce tournoi ?

L'idée est le suivante : chaque partie est caractérisée par ses deux joueurs ; donc on peut représenter ce tournoi par un graphe dans lequel les sommets seront les joueurs et les arêtes seront les parties. Ici on obtient le graphe complet K_5. En coloriant les arêtes de ce graphe on détermine par une couleur une plage horaire durant laquelle les parties de cette couleur pourront se dérouler. L'indice chromatique de K_5 nous donne donc la réponse.

On a un résultat équivalent à la coloration des sommets :

Propriété 1.12 *si Δ est le degré maximal d'un graphe alors son indice chromatique est inférieur ou égal à $\Delta + 1$.*

Plus précisement :

Propriété 1.13 (Vizing) *si Δ est le degré maximal d'un graphe simple alors son indice chromatique est inférieur ou égal à Δ ou $\Delta + 1$.*

En ce qui concerne les graphes bipartis le résultat est

Propriété 1.14 *si Δ est le degré maximal d'un graphe biparti alors son indice chromatique est égal à Δ.*

Preuve : éliminons le cas trivial $\Delta = 1$ pour lequel une seule couleur est clairement suffisante dans un graphe biparti.

On définit un ensemble de Δ couleurs (on les notera $1, 2, \ldots, \Delta$) puis on affecte à chaque arête une couleur sans chercher à colorier les arêtes : si cette affectation est une coloration alors c'est terminé.

Dans le cas contraire on va améliorer cette affectation pour la faire devenir une coloration. On s'occupe tout d'abord d'un sommet posant problème : il existe un sommet x avec deux

arêtes e, e' incidentes de même couleur i. Comme x a au plus Δ arêtes incidentes, alors il existe nécessairement une couleur j qui ne colore aucune arête incidente à x.

Changeons alors la couleur de $e = \{x, y\}$ en couleur j.

Si y n'est pas incident à deux arêtes de couleur j, on s'arrête.

Dans le cas contraire le changement de couleur de e a fait que y a maintenant deux arêtes incidentes e, e'' de couleur j. Changeons la couleur de e'' en couleur i.

De proche en proche, on a ainsi une chaîne élémentaire de couleur alternée et on a éliminé le problème pour x : en effet il se pourrait que cette chaîne se ferme sur x avec la couleur i sur la dernière arête qui revient sur x ; mais alors ce cycle élémentaire serait de longueur impaire ce qui est impossible dans un graphe biparti.

On continue ainsi à éliminer les problèmes de sommet en sommet. □

exemple : prenons $K_{2,3}$ avec la partition de sommets $\{a, b\}$ et $\{A, B, C\}$.

On colorie les 2 arêtes incidentes à A en rouge, à B en bleu et à C en vert.

Partons de A : on change la couleur de $\{A, b\}$ en bleu, puis de $\{b, B\}$ en rouge.

Maintenant partons de C : on change la couleur de $\{C, a\}$ en rouge, puis de $\{a, A\}$ en vert.□

La remarque suivante est importante et nous permet d'introduire la notion de couplages : en regroupant les arêtes de même couleur nous obtenons une partition de l'ensemble des arêtes ; l'ensemble des arêtes ayant la même couleur forme un *couplage i.e.* un ensemble d'arêtes n'ayant aucune extrêmité commune deux à deux.

Pour l'instant, il n'existe pas d'algorithme en temps polynomial qui permet de décider entre Δ et $\Delta + 1$ l'indice chromatique d'un graphe simple.

Nous allons exploiter ces résultats dans le problème de l'emploi du temps : un ensemble de classes doit recevoir une série de cours d'une heure réalisés par un ensemble de professeurs. Bien sûr, chaque classe reçoit un cours unique par heure et chaque professeur donne un cours unique par heure. Le problème est de minimiser le nombre d'heures totales permettant la réalisation de tous les cours.

Le problème peut donc être modélisé ainsi : on établit un graphe (simple) dont les sommets sont classes et professeurs et les arêtes les cours ; le graphe est clairement biparti et une coloration des arêtes résout le problème, donc nous savons que la réponse est Δ.

Mais nous allons compliquer le problème en limitant le nombre de salles disponibles par un entier σ.

Donc il ne peut pas y avoir plus de σ cours par tranche horaire : si le nombre total de cours est m (nombre total d'arêtes) on ne peut donc pas avoir moins de $\frac{m}{\sigma}$ heures pour réaliser tous les cours. L'algorithme suivant permet de colorier les arêtes avec un nombre de couleurs égal à $max\{\Delta, \lceil \frac{m}{\sigma} \rceil\}$ ce qui est un résultat optimal.

Remarquons qu'une nouvelle fois le cas $\Delta = 1$ est trivial. Nous supposerons donc que $\Delta \geq 2$

Bien sûr cet algorithme ne nous sera utile que si $\Delta < \lceil \frac{m}{\sigma} \rceil$, puisque le cas opposé a déjà été traité.

Soit $k = \lceil \frac{m}{\sigma} \rceil > \Delta$, et un ensemble $\{1, \ldots k\}$ de k couleurs. On commence par déterminer une coloration du graphe biparti en Δ couleurs. Soit C_i l'ensemble des arêtes de couleur i pour

$1 \leq i \leq \Delta$; on ajoute les ensembles vides suivants $C_{\Delta+1}, \dots C_k$. Ces k ensembles forment une partition de l'ensemble des arêtes.

Le but est d'obtenir une partition de l'ensemble des arêtes en k ensembles non vides. Pour cela on va équilibrer les couleurs entre elles.

Soit C_i l'ensemble de plus petit cardinal - 0 - et C_j de plus grand cardinal. On a alors $|C_j| > 1$ (en effet le nombre d'arêtes m est supérieur à Δ). On considère le graphe partiel engendré par $C_i \cup C_j$: dans ce graphe les composantes connexes sont soit des sommets isolés soit des chaînes élémentaires soit des cycles élémentaires (de longueur pair) ; de plus leurs couleurs d'arêtes sont alternées entre i et j donc parmi les chaînes élémentaires on choisit une chaîne qui a le plus de couleur i : on fait alors un échange arête par arête entre C_i et C_j dans cette chaîne.

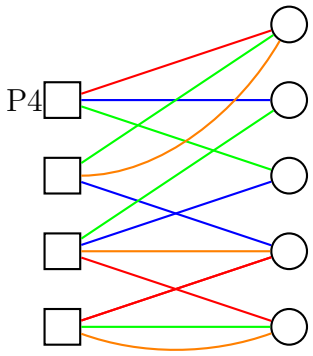
On réapplique ce procédé jusqu'à ce que tous les couplages aient le même cardinal à 1 unité près.

exemple : On considère un ensemble de 4 professeurs $p_1, \dots p_4$ et 5 classes $c_1, \dots c_5$ avec les cours correspondant au tableau suivant.

On a ici $\Delta = 4$ et $m = 13$. On limite de plus le nombre de salles à 3 salles. Donc $\lceil \frac{m}{\sigma} \rceil = 5$: on doit donc trouver 5 couleurs.

	c_1	c_2	c_3	c_4	c_5
p_1	1	1	1	0	0
p_2	2	0	0	1	0
p_3	0	1	1	1	1
p_4	0	0	0	1	2

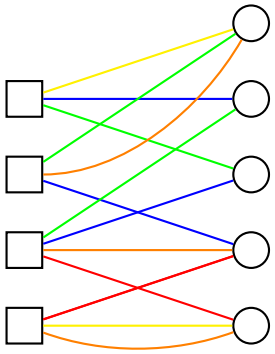
On commence par appliquer la méthode qui nous permet de trouver 4 couleurs (sans contrainte de salles). Ici on peut la trouver facilement.



On a les couplages suivants $C_r = \{(p_1, c_1), (p_3, c_5), (p_4, c_4)\}, C_b = \{(p_1, c_2), (p_2, c_4), (p_3, c_3)\}, C_v = \{(p_1, c_3), (p_2, c_1), (p_3, c_2), (p_4, c_5)\}, C_o = \{(p_2, c_1), (p_3, c_4), (p_4, c_5)\}$. On rajoute $C_j = \emptyset$.

On rééquilibre C_r et C_j en affectant (p_1, c_1) à C_j. Puis on rééquilibre C_v et C_j : le graphe partiel engendré par C_v et C_j a 4 composantes connexes ; on peut affecter (p_4, c_5) à C_j.

Cela donne au final



ce qui correspond à l'emploi du temps suivant

	h_1 (vert)	h_2(bleu)	h_3(jaune)	h_4(rouge)	h_5(orange)
S_1	(p_1, c_3)	(p_1, c_2)	(p_1, c_1)	(p_4, c_4)	(p_2, c_1)
S_2	(p_2, c_1)	(p_2, c_4)	(p_4, c_5)	(p_3, c_5)	(p_4, c_5)
S_3	(p_3, c_2)	(p_3, c_3)			(p_3, c_4)

1.9.5 Graphe planaire

Définition 1.21 *un graphe est planaire s'il peut être dessiné dans le plan sans qu'aucune paire d'arêtes ne s'intersecte.*

Propriété 1.15 *K_5, $K_{3,3}$ ne sont pas planaires.*

exemple : le graphe de la figure 1.16 est planaire et sa représentation plane est donnée par le graphe à côté.

On peut se poser le problème de la planarité lors de la construction d'un réseau de voies ferrées afin de minimiser le nombre d'aiguillages.

Cett notion de graphe planaire a un lien célèbre avec le problème des 4 couleurs ; l'assertion est la suivante : toute carte géographique peut être coloriée avec 4 couleurs seulement de telle sorte que deux pays frontaliers n'aient pas la même couleur. Cette assertion n'a été prouvée que récemment (à l'aide de nombreuses heures de calcul sur ordinateur). Une carte géographique peut être modélisée par un graphe planaire (les sommets sont les pays et les arêtes représentent la relation frontalière). Ce qui mène à l'expression suivante de ce résultat :

Théorème 1.16 *Le nombre chromatique d'un graphe planaire est inférieur ou égal à 4.*

Pour continuer sur des notions géométriques, on peut noter que si G est plan alors G délimite dans le plan un certain nombre de régions.

exemple : le graphe **plan** de la figure 1.16 délimite 7 régions.

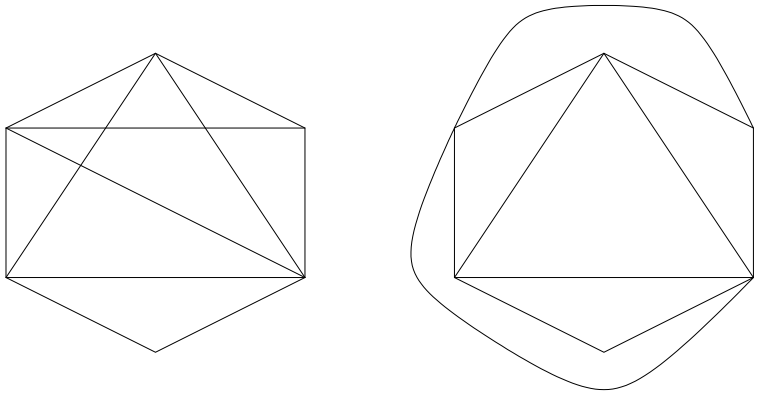


FIGURE 1.16 – Un graphe planaire et sa représentation plane

Propriété 1.17 *Soit G un graphe plan connexe d'ordre n à m arêtes délimitant r régions. On a alors la formule d'Euler :* $n - m + r = 2$.

On peut remarquer que si G est un graphe plan à m arêtes délimitant r régions alors le fait d'ajouter une arête à G en conservant sa propriété d'être plan coupe une des r régions en deux et le nombre de régions augmente d'un comme le nombre d'arêtes.

Comme on le verra dans le chapitre 2, un graphe non orienté connexe d'ordre n a au moins $n-1$ arêtes. Un tel graphe connexe (d'ordre n ayant $n-1$ arêtes) est appelé un arbre et il n'a pas de cycle ; par conséquent sa représentation plane délimite une unique région.

Chapitre 2

Arbres et Parcours de graphe

2.1 Définitions - Caractérisations

2.1.1 Arbre en théorie des graphes

Définition 2.1 *Un arbre est un graphe non orienté connexe sans cycle (acyclique).*

exemple :

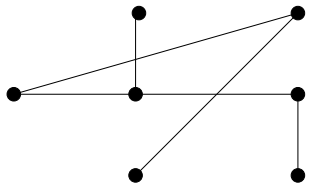


FIGURE 2.1 – Un arbre

exercice : trouver les 6 arbres d'ordre 6 (non isomorphes).

2.1.2 Arborescence

Définition 2.2 *Une arborescence est un graphe orienté possédant un sommet priviligié, la racine, tel qu'il existe un et un seul chemin depuis la racine à tout autre sommet.*

exemple : cf figure 2.2

Propriété 2.1 *On peut orienter les arêtes d'un arbre à partir de n'importe quel sommet de façon à obtenir une arborescence.*

exemple : dans la figure 2.3, l'arbre de la figure 2.1 devient une arborescence de racine r.

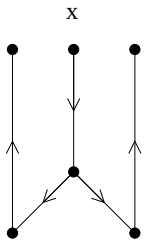


FIGURE 2.2 – Une arborescence de racine x

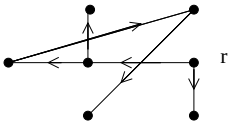


FIGURE 2.3 – Une arborescence de racine r

2.2 Propriétes et caractérisations des arbres

Théorème 2.2 *Soit G un graphe non orienté d'ordre n. Les propositions suivantes sont équivalentes :*

1. *G est un arbre*
2. *G est connexe et a $n-1$ arêtes*
3. *G est sans cycle et a $n-1$ arêtes*
4. *G est sans boucle et pour tous sommets x, y, $x \neq y$, il existe une unique chaîne reliant x à y*

Preuve : elle repose en partie sur les deux lemmes et la propriété suivants

Lemme 2.1 *un graphe non orienté connexe d'ordre n a au moins n-1 arêtes.*

Lemme 2.2 *un graphe non orienté d'ordre n qui a au moins n arêtes possède un cycle.*

Propriété 2.3 *Soit G un graphe non orienté d'ordre n, à m arêtes et p composantes connexes G est sans cycle si et seulement si $m - n + p = 0$.*

Preuve des lemmes :

• On remarquera tout d'abord que si G est un graphe connexe d'ordre supérieur ou égal à 2 tel qu'il existe un graphe partiel $G-e$ de G qui ne soit pas connexe alors $G-e$ a deux composantes connexes En effet, soit $e = \{x, y\}$. On peut alors partitionner l'ensemble des sommets de G en deux parties : les sommets reliés à x par une chaîne ne passant pas par e et les sommets reliés à y par une chaîne ne passant pas par e. Ce sont les deux composantes connexes de $G - e$.

Soit G un graphe non orienté connexe d'ordre n. Montrons par récurrence sur n que G a au moins $n-1$ arêtes.

Pour $n = 1$, le résultat est clair.

Soit G connexe d'ordre $n+1$, $n \geq 1$. Supposons qu'il existe un graphe partiel $G-e$ de G qui ne soit pas connexe. Soient alors C_1, C_2 les deux composantes connexes de $G-e$. L'ordre n_i de

C_i est au plus égal à n pour $i = 1, 2$ et $n_1 + n_2 = n + 1$. Par hypothèse de récurrence, C_i a donc au moins $n_i - 1$ arêtes. Donc $G - e$ a au moins $(n_1 - 1) + (n_2 - 1) = n + 1 - 2 = n - 1$ arêtes. D'où G a au moins n arêtes. □

• Soit G un graphe d'ordre n ayant au moins n arêtes. Montrons par récurrence sur n que G possède un cycle.

Pour $n = 1$, c'est clair (c'est une boucle).

Soit G d'ordre $n+1$ ayant au moins $n+1$ arêtes, $n \geq 1$. Alors G a au moins une composante connexe C_j d'ordre n_j ayant au moins n_j arêtes. En effet, si toutes les composantes connexes C_i de G d'ordre n_i ont m_i arêtes avec $m_i < n_i$ alors, puisque $\sum_{i=1}^{i=p} n_i = n + 1$, G aurait $m = \sum_{i=1}^{i=p} m_i < \sum_{i=1}^{i=p} n_i$ arêtes d'où $m < n + 1$: contradiction.

Dans la composante connexe C_j d'ordre $n_j \leq n$, par hypothèse de récurrence il y a un cycle. □

• Montrons maintenant la propriété. Soit G un graphe sans cycle. Montrons par récurrence sur n que $m - n + p = 0$.

Pour $n = 1$, on a nécessairement $m = 0$ et $p = 1$, l'égalité est vérifiée.

Soit G d'ordre $n + 1$, $n \geq 1$ sans cycle. Alors ses p composantes connexes sont sans cycle. L'ordre n_i de chacune d'entre elles est inférieur ou égal à n. Donc par hypothèse de récurrence, on a $m_i - n_i + 1 = 0$. D'où $m - n + p = \sum_{i=1}^{i=p} m_i - \sum_{i=1}^{i=p} n_i + p = \sum_{i=1}^{i=p} (m_i - n_i) + p = -p + p = 0$.

Réciproquement, on montre par récurrence sur n que si G a un cycle alors $m - n + p > 0$.

Pour $n = 1$, l'existence d'un cycle implique l'existence d'au moins une arête et comme $n = p = 1$, l'inégalité est vérifiée.

Soit G d'ordre $n + 1$, $n \geq 1$ ayant un cycle. Soient alors $C_{i_j}, 1 \leq j \leq k$, ses k composantes connexes C_j ayant un cycle. Par hypothèse de récurrence, $m_{i_j} - n_{i_j} + 1 > 0$ pour $1 \leq j \leq k$. Pour les $p - k$ autres, on a nécessairement $m_{i_j} - n_{i_j} + 1 = 0$ puisqu'elles sont sans cycle. D'où $m - n + p = \sum_{i=1}^{i=p} (m_i - n_i + 1) > 0$. □

On peut maintenant montrer le théorème :

• $1 \Rightarrow 2$ et $1 \Rightarrow 3$ par définition et d'après les lemmes.

• $2 \Rightarrow 1$ et $3 \Rightarrow 1$ d'après la propriété. Donc $2 \Leftrightarrow 3$.

• $4 \Rightarrow 1$: sous l'hypothèse 4, G est connexe et l'existence d'un cycle de longueur supérieure ou égale à 2 entrainant l'existence de deux chaînes simples distinctes reliant deux sommets distincts du cycle, G est sans cycle.

• $1 \Rightarrow 4$: l'existence provient de la connexité et l'unicité provient de l'absence de cycle. □

remarque : attention, les réciproques des deux lemmes précédents ne sont pas vraies.

remarque : la propriété peut s'énoncer de façon plus générale : $m-n+p \geq 0$ pour tout graphe d'ordre n, ayant m arêtes et p composantes connexes

On peut aussi caractériser un arbre par le résultat suivant :

Propriété 2.4 *G est un arbre*
- *si et seulement si G est connexe et si on lui retire une arête, il n'est plus connexe*
- *si et seulement si G est sans cycle et si on lui rajoute une arête, on forme un cycle.*

Définition 2.3 *On appelle isthme une arête e telle que le nombre de composantes connexes de $G-e$ est strictement supérieur au nombre de composantes connexes de G.*

Toute arête d'un arbre est un isthme et sa suppression engendre deux composantes connexes.

2.3 Arbre couvrant

2.3.1 Définition

Soit G un graphe. Un *arbre couvrant* de G est un graphe partiel de G qui est un arbre.

exemple : dans la figure 2.4, un arbre couvrant d'un graphe d'ordre 7 est dessiné en gras.

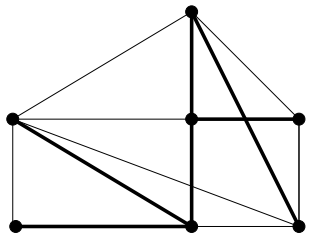


FIGURE 2.4 – Un graphe et un de ses arbres couvrants

2.3.2 Propriétés

Propriété 2.5 *tout graphe connexe a un arbre couvrant.*

remarque : pour un graphe connexe, il peut y avoir plusieurs arbres couvrants non isomorphes.

Pour un graphe non connexe, on parle de *forêt couvrante*.

Les arbres couvrants peuvent être caractérisés de plusieurs façons

Propriété 2.6 *Un graphe partiel d'un graphe connexe G est un arbre couvrant de G si et seulement si il est connexe et minimal avec cette propriété relativement à la suppression d'arêtes.*

Propriété 2.7 *Un graphe partiel d'un graphe connexe G est un arbre couvrant de G si et seulement si il est acyclique et maximal avec cette propriété relativement l'ajout d'arêtes.*

Les deux propriétés suivantes permettent de préciser la conséquence d'un ajout d'arête à un arbre couvrant.

Propriété 2.8 *Soit T un arbre couvrant de G et e une arête de G n'appartenant pas à T. Le graphe partiel $T + e$ contient un unique cycle élémentaire.*

Propriété 2.9 *Soit T un arbre couvrant de Get e une arête de G n'appartenant pas à T. Soit e' une arête du cycle de $T + e$. Alors $T + e - e'$ est un arbre couvrant de G non nécessairement isomorphe à T. (cf figure 2.5)*

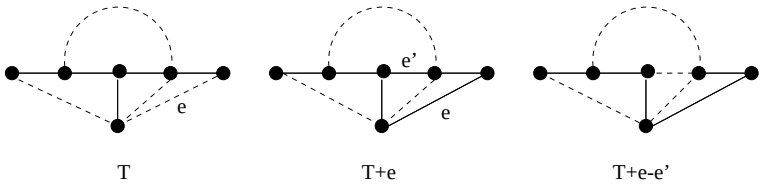


FIGURE 2.5 – deux arbres couvrants T et $T + e - e'$ non isomorphes

2.4 Parcours de graphes

2.4.1 Parcours en profondeur

Le principe du parcours en profondeur d'un graphe (orienté ou non) est celui du parcours d'un labyrinthe : on va de sommet en sommet en marquant au fur et à mesure les sommets visités. La visite se poursuit le plus loin possible tant qu'il reste des sommets accessibles non encore marqués. Quand on atteint un sommet z dont tous les voisins ont été déjà marqués alors on revient au sommet précédant z dans la visite.

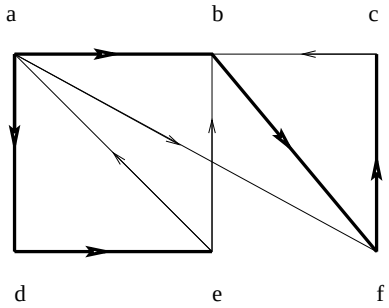
Algorithme

C'est une procédure essentiellement récursive. Pour la mettre en oeuvre on utilisera un tableau de booléens *marque* indexé par les sommets du graphe. Ce tableau est initialisé à $false$. Un parcours de G à partir d'un sommet x de G est réalisé par la procédure suivante :

PROCÉDURE P_prof (G : graphe ; x : sommet)
  $marque[x] \leftarrow true$
  POUR chaque sommet y adjacent à x FAIRE SI $marque[y] = false$ ALORS P_prof(G ; y)FINSI
  FINPOUR
FINPROCÉDURE

Cette procédure appelée sur le sommet x marquera tous les sommets de la composante connexe de x. Pour effectuer un parcours complet du graphe il faudra relancer la procédure à partir d'un sommet non marqué de G jusqu'à de que tous les sommets soient marqués.

On peut ainsi déterminer les composantes connexes d'un graphe non orienté mais attention on ne détermine pas ainsi les composantes fortement connexes d'un graphe orienté (cf figure 2.6 : on peut voir en gras les arcs empruntés par un parcours en profondeur depuis le sommet a en suivant l'ordre alphabétique pour la boucle POUR).



{a,d,e} et {b,c,f} sont les deux composantes fortement connexes de ce graphe pourtant le parcours en profondeur depuis a marque tous les sommets

FIGURE 2.6 – un parcours en profondeur d'un graphe orienté

On peut modifier la procédure afin de numéroter les sommets dans l'ordre de visite. Pour cela, on initialise une variable globale num à 0 et on met à jour la variable $i\_prof[z]$ d'un tableau i_prof indexé par les sommets de G à chaque nouveau sommet visité z.

PROCÉDURE P_prof2 (x : sommet)

$marque[x] \leftarrow true$

$num \leftarrow num + 1$

$i\_prof[x] \leftarrow num$

POUR chaque sommet y adjacent à x FAIRE SI $marque[y] = false$ ALORS P_prof2(y)

FINPOUR

FINPROCÉDURE

Arbre couvrant issu d'un parcours en profondeur

En effectuant le parcours en profondeur d'un graphe à partir d'un sommet, on note que l'on engendre un arbre couvrant de la composante connexe de x. Il suffit de recueillir l'arête reliant le sommet visité (paramètre de l'appel courant de la procédure) au sommet adjacent que l'on va visiter (paramètre du prochain appel de la procédure). Cet arbre peut être orienter afin d'obtenir une arborescence de racine x

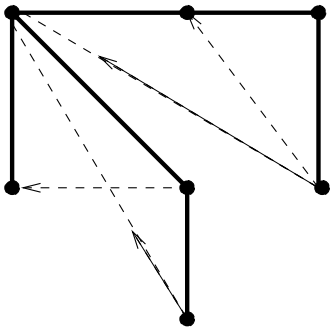
On obtiendra une forêt couvrante si l'on visite complètement le graphe.

Les arêtes de G n'appartenant pas à la forêt couvrante de G issu d'un parcours complet sont appelées *arêtes de retour*.

remarque : les arêtes de retour ont la propriété suivante : si l'arête xy est une arête de retour et si x est visité avant y ($i\_prof[x] < i\_prof[y]$) alors y est un descendant de x dans l'arbre couvrant de parcours en profondeur.

En effet, l'appel récursif x induit un sous-graphe de la forêt couvrante qui est un arbre que l'on peut orienter afin d'obtenir une arborescence de racine x. Dans cet arbre figurent nécessairement tous les sommets adjacents à x et visités après x : donc y est dans cet arbre et est un descendant de x.

exemple :



en gras, l'arbre couvrant et en pointillés

les aretes de retour

FIGURE 2.7 – un parcours en profondeur

Complexité

Soit G d'ordre n à m arêtes. On suppose que G est représenté par listes d'adjacence.

Chaque sommet est paramètre de la procédure une et une seule fois donc il y a n appels récursifs. A chaque sommet visité, tous les sommets adjacents à x sont testés pour le tableau *marque* ce qui se fait globalement en un temps proportionnel à m. L'algorithme demande donc un temps $O(n)$ pour les appels et un temps $O(m)$ pour les marques. Donc le temps de calcul est de l'ordre de $O(max(n, m))$.

2.4.2 Parcours en largeur

Le principe est de visiter tous les voisins d'un sommet avant de visiter le sommet suivant qui sera le premier voisin à avoir été visité auparavent. Pour conserver les voisins d'un sommet en cours de visite que l'on visitera ultérieurement, on utilisera une file.

Algorithme

Pour mettre en oeuvre un parcours en largeur on utilisera un tableau de booléens *marque* indexé par les sommets du graphe et une file ϕ de sommets. Le tableau *marque* est initialisé à $false$ et la file est initialisée à *vide*. Un parcours en largeur de G à partir d'un sommet x de G est réalisé par la procédure suivante :

PROCÉDURE P_larg (G : graphe ; x : sommet)

$marque[x] \leftarrow true$

$enfiler(x, \phi)$

TANT QUE $\phi \neq \emptyset$ FAIRE

$y \leftarrow premier(\phi)$

POUR tout sommet z adjacent à y FAIRE

SI $marque[z] = false$ ALORS $enfiler(z, \phi)$; $marque[z] \leftarrow true$ FINSI

FINPOUR

$defiler(\phi)$

FINTANT QUE

FINPROCÉDURE

Cette procédure appelée sur le sommet x marquera tous les sommets de la composante connexe de x. Pour effectuer un parcours complet du graphe il faudra relancer la procédure à partir d'un sommet non marqué de G jusqu'à de que tous les sommets soient marqués.

On peut ainsi déterminer les composantes connexes d'un graphe non orienté mais attention, comme dans le cas du parcours en profondeur, on ne détermine pas ainsi les composantes fortement connexes d'un graphe orienté (cf 2.9).

exemple : dans la figure 2.8 la boucle POUR suit l'ordre d'indexation des sommets. Les arêtes en gras sont celles de l'arbre couvrant issu de ce parcours en largeur depuis le sommet x_1.

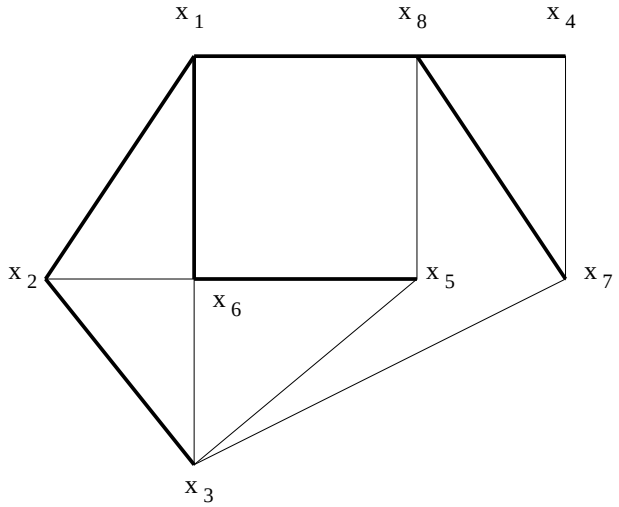


FIGURE 2.8 – un parcours en largeur d'un graphe non orienté

exemple : dans la figure 2.9 la boucle POUR suit l'ordre alphabétique des sommets. Les arcs en gras sont ceux de l'arborescence couvrante issue de ce parcours en largeur depuis le sommet a.

On peut modifier la procédure afin de numéroter les sommets dans l'ordre de visite. Pour cela, on initialise une variable globale num à 0 et on met à jour la variable $i\_prof\_lar[z]$ d'un tableau i_prof_lar indexé par les sommets de G à chaque nouveau sommet visité z.

On obtient ainsi une numérotation différente que celle du parcours en profondeur ; cette numérotation correspond à l'ordre hiérarchique.

exemple : dans la figure 2.8, les sommets $x_1, \cdots, x_8$ ont respectivement comme indice de parcours en largeur : $1, 2, 5, 7, 6, 3, 8, 4$.

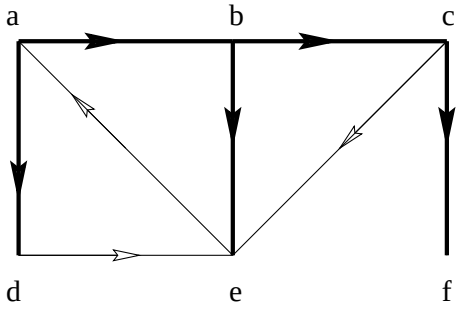


FIGURE 2.9 – un parcours en largeur d'un graphe orienté

Complexité

Soit G d'ordre n à m arêtes. On suppose que G est représenté par listes d'adjacence.

Chaque sommet est enfilé au plus une fois. A chaque sommet visité, tous les sommets adjacents sont testés pour le tableau *marque* ce qui se fait globalement en un temps proportionnel à m. L'algorithme demande donc un temps $O(n)$ pour la boucle tant que et un temps $O(m)$ pour les marques. Donc le temps de calcul est de l'ordre de $O(max(n, m))$.

2.4.3 Algorithme de Moore

Cet algorithme repose sur un parcours en largeur et permet de déterminer une plus courte chaîne, et donc la distance, entre deux sommets fixés d'un graphe non orienté.

Soient x, y les sommets du graphe dont on veut déterminer la distance. On associe initialement à x l'index 0 et à tous les autres sommets l'index ∞. Puis tous les voisins de x sont indexés par 1 (ce qui correspond à la distance de x à chacun de ses voisins). Puis à chaque sommet d'index ∞ adjacent à un sommet d'index 1 on associe l'index 2 et on continue ainsi jusqu'à ce que y ait un index fini ou bien jusqu'à ce que tous les sommets d'index fini soient adjacents seulement à des sommets d'index fini.

On utilisera
- une file dont les éléments seront des sommets de G.
- un tableau *parent* indexé par les sommets de G et dont les éléments seront des sommets de G.
- un tableau d'entiers (plus l'élément ∞) *index* indexé par les sommets de G.
- un tableau *plus_court* de sommets de G.

PROCÉDURE Moore(G : graphe ; x, y : sommet)

$index[x] \leftarrow 0$

POUR tout sommet $z \neq x$ FAIRE $index[z] \leftarrow \infty$ FINPOUR

$enfiler(x, \phi)$

TANT QUE $\phi \neq \emptyset$ FAIRE

$t \leftarrow premier(\phi)$

POUR tout sommet z adjacent à t FAIRE

SI $index[z] = \infty$ ALORS $enfiler(z, \phi)$; $index[z] \leftarrow index[t] + 1$; $parent[z] \leftarrow t$ FINSI

FINPOUR

$defiler(\phi)$

FINTANT QUE

SI $index[y] \neq \infty$ ALORS $k \leftarrow index[y]; plus\_court[k] \leftarrow y$ FINSI

TANT QUE $k \neq 0$ FAIRE

$plus\_court[k-1] \leftarrow parent[plus\_court[k]]$

$k \leftarrow k-1$

FINTANT QUE

$\{plus\_court[0] \cdots plus\_court[k]$ est une plus courte chaîne reliant x à $y\}$

FINPROCÉDURE

Complexité : la complexité est la même que pour le parcours en largeur $O(max(n,m))$.

2.5 Points d'articulation dans un graphe non orienté

Soit G un graphe non orienté.

Définition 2.4 *un sommet x est un point d'articulation de G si le nombre de composantes connexes de $G-x$ est strictement supérieur au nombre de composantes connexes de G.*

exemple : dans la figure 2.10, b, c et g sont les points d'articulation du graphe.

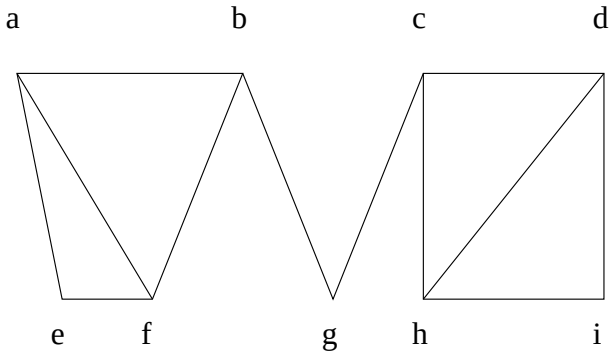


FIGURE 2.10 – points d'articulation

Définition 2.5 *un graphe connexe sans point d'articulation est dit non séparable ou inarticulé.*

Définition 2.6 *Soit G un graphe non orienté connexe. Un bloc de G est un sous-graphe qui est un graphe non séparable et maximal vis à vis de cette propriété.*

exemple : dans le graphe de la figure 2.10, les blocs sont $\{a,b,e,f\}$, $\{g\}$ et $\{c,d,h,i\}$.

Les blocs d'un graphe induisent une partition de l'ensemble des arêtes de G et deux blocs distincts ont au plus un sommet en commun qui est alors nécessairement un point d'articulation.

Si G représente un réseau de communication, le fait que G soit non séparable nous assure que le réseau fonctionnera malgré une panne de l'équipement de l'un de ses noeuds.

Pour déterminer les points d'articulation d'un graphe connexe G, on procède à un parcours en profondeur qui fournit un arbre couvrant T de G et au cours duquel on calcule l'indice de parcours en profondeur. Cet indice permet alors de calculer une autre valeur pour chaque sommet de G, l'indice *plus-pas* défini comme suit :

Soit x un sommet. $plus\_bas[x]$ est le minimum de

- $i\_prof[x]$
- $i\_prof[y]$ pour tout sommet y tel que $\{xy\}$ est une arête dans G mais pas dans T.
- $plus\_bas[t]$ pour tout sommet t qui est un fils de x dans T (*i.e.* $i\_prof[x] < i\_prof[t]$ et $\{xt\}$ est une arête dans T.

remarque : cette définition est récursive et le calcul de l'indice $plus\_bas$ se fait par un parcours en profondeur de T dans l'ordre postfixe.

Soit x_0 le sommet de départ du parcours en profondeur. x_0 est donc la racine de T et on a la propriété suivante :

Propriété 2.10 *les points d'articulation sont caractérisés par*

x_0 est un point d'articulation si et seulement si x_0 a au moins deux fils dans T.

Soit x différent de x_0. x est un point d'articulation si et seulement si x a un fils t dans T tel que $plus\_bas[t] \geq i\_prof[x]$.

Preuve : soit x_0 la racine de T.

Supposons que x_0 soit un point d'articulation de G et soit y le premier sommet visité après x_0. Alors $G - x_0$ a au moins deux composantes connexes et y appartient à l'une d'elles, soit C_1. Soit C_2 une composante connexe distincte de C_1. Avant de visiter C_2, on doit visiter de nouveau x_0 puisqu'il n'existe aucune arête de C_1 à C_2 dans G. Donc x_0 a au moins deux fils dans T soit y et un sommet de C_2.

Réciproquement, supposons que x_0 a au moins deux fils dans T et soit y le premier sommet visité après x_0 (y est donc un fils de x_0 dans T). Soit T_y le sous-arbre de T engendré par l'appel récursif du parcours en profondeur sur y ; y est la racine de T_y et T_y contient tous les descendants de y dans T. Supposons que uv soit une arête de G telle que u soit un sommet de T_y et v un sommet de $T - x_0 - T_y$. Alors v est visité après u et uv n'est pas une arête de T_y. Donc cette arête est une arête de retour et d'après la remarque du paragraphe 2.4.1 v doit être un descendant de u donc un sommet de T_y : contradiction. Donc G ne contient pas de telle arête donc toute chaîne reliant un sommet de T_y à un sommet de $T - x_0 - T_y$ passe par x_0 et x_0 est un point d'articulation de G.

Soit $x \neq x_0$ un sommet de G.

Supposons que x soit un point d'articulation de G. Alors x_0 appartient à l'une de composantes connexes de $G - x$ soit C_1. Puisque x est un point d'articulation, le parcours en profondeur doit visiter x une première fois alors qu'il reste encore des sommets non visités. Soit y le premier

sommet adjacent à x qui n'appartient pas à C_1. y appartient à une composante connexe C_2 de $G - x$ et y est visité après x d'où $i\_prof[x] < i\_prof[y]$.

Puisqu'aucun sommet de C_2 n'est adjacent à un sommet de $G - C_2 - x$ y est visité directement aprés x, donc xy est une arête de T ; de plus, $i_prof[z] > i_prof[y]$ pour toute arête yz telle que z soit visité après y. Donc $plus\_bas[y] \geq i\_prof[x]$.

Réciproquement, supposons que y est un fils de x dans T tel que $plus\_bas[y] \geq i\_prof[x]$. Soit S l'ensemble des sommets de la chaîne élémentaire reliant x_0 à x dans T en excluant x. Soit S' l'ensemble des sommets du sous-arbre maximal de T de racine y. Par la remarque du paragraphe 2.4.1, il n'existe pas d'arête reliant un sommet de S' à un sommet de $G - (S \cup S' \cup x)$. Supposons qu'il existe une arête ss' de G telle $s \in S$ et $s' \in S'$. alors ss' doit être une arête de retour et on doit avoir $i\_prof[s] < i\_prof[x]$. En prenet la chaîne élémentaire reliant y à s' dans T suivie de l'arête ss', on remarque que $plus\_bas[y] \leq i\_prof[s]$. Donc $plus\_bas[y] < i\_prof[x]$ ce qui contredit l'hypothèse. Donc toute chaîne reliant un sommet de S à un sommet de S' passe par x et x est un point d'articulation de G. □

exemple : dans le graphe de la figure 2.10, si le parcours en profondeur depuis a suit l'ordre alphabétique, on obtient les valeurs suivantes :

sommet	a	b	c	d	e	f	g	h	i
i_prof	1	2	6	7	9	3	5	8	9
$plus\_bas$	1	1	6	6	1	1	5	6	7

2.6 Isthme

On a vu la définition des isthmes (bridges) comme étant une arête indispensable à la connexité du graphe. Le théorème suivant permet de déterminer les isthmes après un parcours en profondeur.

Théorème 2.11 *si T est l'arbre couvrant issu d'un parcours en profondeur d'un graphe connexe G alors l'arête $\{xy\}$ avec $i\_prof[x] < i\_prof[y]$ est un isthme si et seulement si $\{xy\}$ est une arête de T et $plus\_bas[y] > i\_prof[x]$.*

exemple : dans le graphe de la figure 2.10, si le parcours en profondeur depuis a suit l'ordre alphabétique, les isthmes sont bien les arêtes $\{gc\}$ et $\{bg\}$ avec $plus\_bas[c] > i\_prof[g]$ et $plus\_bas[g] > i\_prof[b]$.

2.7 k-connexité

La perte d'un sommet pour un graphe non séparable ne provoque pas la perte de la connexité. De façon plus générale, on se pose la question pour un graphe connexe du nombre minimum de sommets à retirer pour perdre la connexité de ce graphe. Plus ce nombre est important, plus un réseau représenté par un tel graphe est robuste vis à vis des pannes.

Plus précisement, on définit la notion de *k-connexité*

Définition 2.7 *soit G un graphe non orienté ayant au moins une paire de sommets non adjacents. On appelle $\kappa(G)$, le nombre de connexité de G, le plus petit entier k tel qu'il existe des sommets $x_1, \cdots x_k$ de G tels que $G - \{x_1, \cdots x_k\}$ soit non connexe.*

On dira que G est *k-connexe* si $\kappa(G) \geq k$.

Dans le cas où G n'a pas de paire de sommets non adjacents (*i.e.* G est complet si de plus G est simple), on posera $\kappa(G) = n - 1$ où n est l'ordre de G.

exemple : pour le graphe complet K_n, $\kappa(K_n) = n - 1$.

exemple : pour le graphe de la figure 2.10, $\kappa(G) = 1$.

pour le graphe de la figure 2.8, $\kappa(G) = 2$.

On peut définir la notion équivalente pour les arêtes et définir ainsi le *nombre d'arête -conexité* $\kappa'(G)$ d'un graphe connexe d'ordre supérieur ou égal à 2. On a alors les résultats suivants :

Théorème 2.12 *$\kappa'(G) \leq \kappa(G) \leq \delta$ où δ est le degré minimum des sommets du graphe.*

exemple : pour le graphe de la figure 2.8, $\kappa'(G) = \kappa(G) = 2$ et $\delta = 2$.

exercice : chercher un exemple de graphe pour lequel les inégalités ci-dessus sont strictes.

Théorème 2.13 *si $\delta \geq \dfrac{n + k - 2}{2}$ alors G est k-connexe.*

exemple : dans la figure 2.11 le graphe a un nombre de connexité égale à 4.

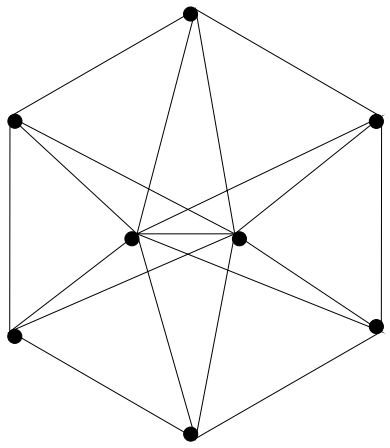


FIGURE 2.11 – nombre de connexité d'un graphe

remarque : tout graphe non séparable d'ordre supérieur à 3 (en particulier un bloc d'au moins 3 sommets) est 2-connexe.

Deux résultats importants (dont on ne donnera pas la preuve) fournissent une caractérisation de graphe k-connexe.

Théorème 2.14 (Menger) • *Un graphe d'ordre $n \geq k+1$ est k-connexe si et seulement si deux sommets distincts quelconques sont reliés par k chaînes qui n'ont aucun sommet commun (hormis leurs extrêmités).*

• *Un graphe est k-arête-connexe si et seulement si deux sommets distincts quelconques sont reliés par k chaînes élémentaires qui n'ont aucune arête commune.*

Chapitre 3

Problème de l'Arbre Couvrant Minimum

Le problème abordé ici correspond à la minimisation du coût de construction d'un réseau minimal de communications connaissant le coût de chaque liaison possible entre les noeuds du réseau à construire. Puisqu'on souhaite un réseau minimal, on recherche un arbre couvrant (*i.e.* le graphe induit connexe qui a le minimum d'arêtes). On doit de plus tenir compte du coût de construction : chaque arête sera affectée d'un *poids* par une fonction p à valeurs dans $\mathbb{R}_+^*$ et le coût total sera égal à la somme des coûts des arêtes de l'arbre. On dit que l'on recherche un arbre couvrant minimal.

exemple :

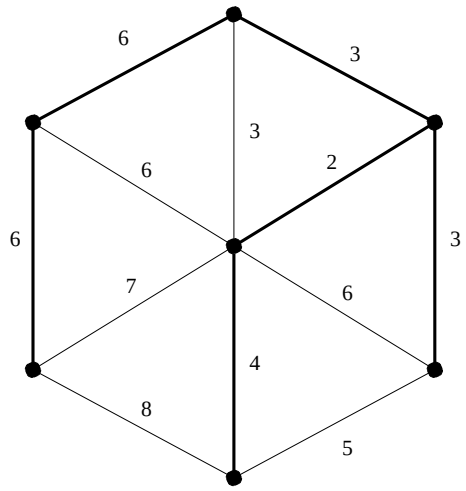


FIGURE 3.1 – un réseau à construire et une solution minimale en gras

remarque : par la suite, on utilisera la même notation pour un ensemble d'arêtes et le graphe engendré par cet ensemble.

Trouver un arbre couvrant minimal en examinant tous les arbres couvrants est en pratique irréalisable : pour un graphe complet d'ordre 20, il faudrait plus de 83000 siècles pour déterminer l'arbre couvrant minimal à raison d'un million d'arbres couvrants calculés par milliseconde.

On étudiera deux algorithmes de complexité raisonnable qui résolvent le problème de l'arbre couvrant minimal.

3.1 Algorithme de Kruskal

L'algorithme de Kruskal est un algorithme glouton qui repose sur le fait qu'un arbre couvrant d'un graphe d'ordre n a $n-1$ arêtes et est acyclique. Le choix des arêtes se fera donc sur deux critères : la conservation de l'acyclicité et un coût minimal.

3.1.1 Algorithme

Soit E l'ensemble des sommets de G. On utilisera un ensemble d'arêtes T qui sera en sortie l'arbre couvrant minimal et un ensemble d'arêtes F qui représentera les arêtes qui peuvent être choisies.

PROCÉDURE Kruskal(G : graphe).

$T \leftarrow \emptyset$

$F \leftarrow E$

TANT QUE $|T| < n-1$ FAIRE

trouver une arête $e \in F$ de poids minimal

$F \leftarrow F - e$

SI $T + e$ est acyclique ALORS $T \leftarrow T \cup e$ FINSI

FINTANT QUE

FINPROCÉDURE

exemple :

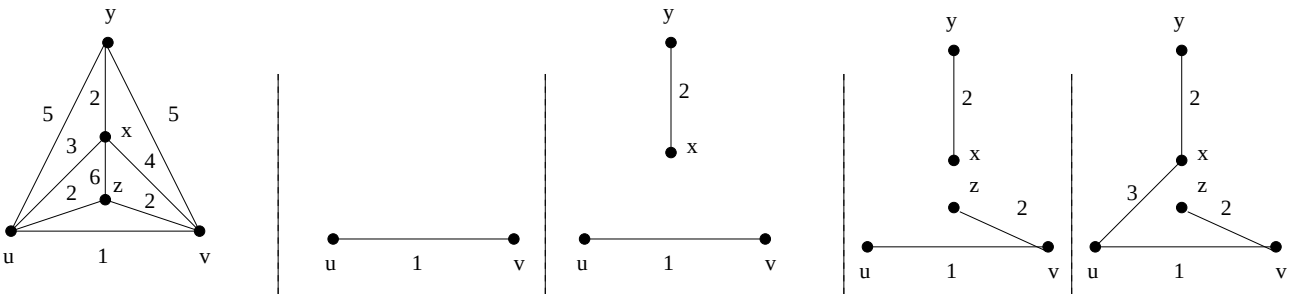


FIGURE 3.2 – l'algorithme de Kruskal

3.1.2 Preuve de l'algorithme

Soit n est l'ordre de G. L'algorithme produit bien un arbre couvrant de G puisqu'il termine lorsque $n-1$ arêtes sont choisies et T est acyclique.

Supposons que T ne soit pas minimal.

Si $e_1, \cdots, e_{n-1}$ sont les arêtes de T dans l'ordre de choix de l'algorithme, alors $p(e_1) \leq \cdots \leq p(e_{n-1})$ par construction. Soit alors A un arbre couvrant minimal tel que l'indice de la première arête de T qui ne soit pas une arête de A soit maximum. Soit donc k cet indice, on a alors $e_1 \cdots e_k \in T$, $e_1 \cdots e_{k-1} \in A$ et $e_k \notin A$. Alors $A + e_k$ contient un unique cycle C. C ne peut pas être constitué uniquement d'arête de T (hormis e_k) car sinon T contiendrait un cycle.Soit alors e' une arête de C qui appartient à A mais qui n'appartient pas à T. $A + e_k - e'$ est donc un arbre couvrant de G et $p(A+e_k-e') = p(A)+p(e_k)-p(e')$. Dans l'exécution de l'algorihtme de Kruskal, e_k a été choisie de poids minimal tel que $G[e_1 \cdots e_k]$ soit acyclique. Mais $G[e_1 \cdots e_{k-1}, e']$ est aussi acyclique puisque sous-graphe de A. Donc $p(e') \geq p(e_k)$ et $p(A + e_k - e') \leq p(A)$. On en déduit que $A+e_k-e'$ est optimal et diffère de T par un arête d'indice strictement supérieur à k : contradiction.

3.1.3 Complexité

Si les arêtes sont classées par ordre de coût croissant, ce qui peut se faire en $O(m \log m)$ par un tri *rapide*, il reste à évaluer la complexité du test d'acyclicité. Si les composantes connexes du graphe T sont connues, ce test se réduit à vérifier que les extrêmités de l'arête choisie appartiennent à deux composantes connexes distinctes. Il reste alors à gérer ces composantes connexes de façon à *fusionner* les deux composantes connexes reliées par l'arête choisie. Cela peut se faire par la technique de *fusion rapide* en $O(m \log n)$. L'algorithme de Kruskal a une complexité de l'ordre $O(m \times max(\log n, \log m))$.

remarque : elle peut être améliorée par un algorithme de fusion encore plus efficace.

3.2 Algorithme de Prim

L'algorithme de Kruskal veille à maintenir la propriété d'acyclicité d'un arbre alors que l'algorithme de Prim se base sur la connexité d'un arbre. L'algorithme de Prim fait pousser un arbre couvrant minimal en ajoutant au sous-arbre T déjà construit une nouvelle branche parmi les arêtes de poids minimal joignant un sommet de T à un sommet n'appartenant pas à ce dernier. L'algorithme s'arrête lorsque tous les sommets du graphe appartiennent à T.

3.2.1 Algorithme

Soit X l'ensemble des sommets de G. On utilisera un ensemble d'arêtes T qui sera en sortie l'arbre couvrant minimal et un ensemble de sommets S qui contiendra les sommets de T.

PROCÉDURE Prim(G : graphe)
  choisir un sommet x de G.
  $S \leftarrow S \cup x$
  $T \leftarrow \emptyset$
  TANT QUE $S \neq X$ FAIRE
    trouver une arête $\{ys\}$ de poids minimal tel que $y \in X - S$ et $s \in S$

$T \leftarrow T \cup \{ys\}$

$S \leftarrow S \cup y$

FINTANT QUE

FINPROCÉDURE

exemple : dans la figure 3.3, en partant du sommet u, T contient successivement les arêtes $\{uv\}$, $\{uz\}$, $\{ux\}$, $\{xy\}$.

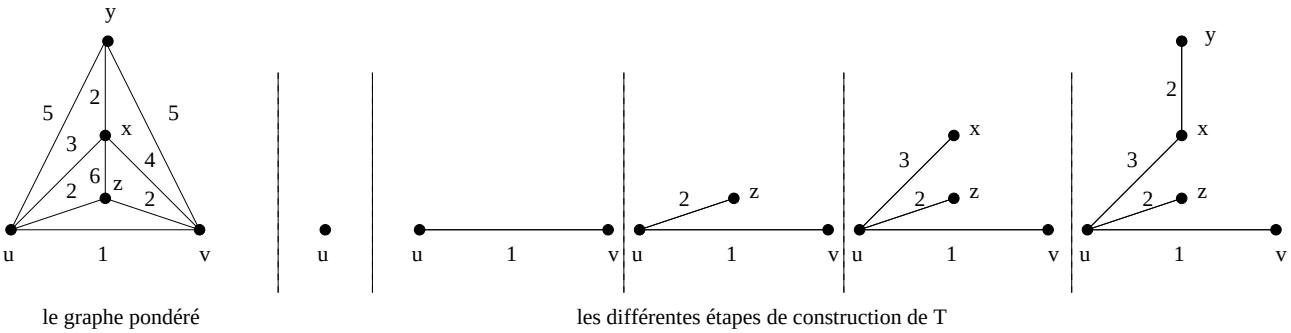


FIGURE 3.3 – l'algorithme de Prim

3.2.2 Preuve de l'algorithme

Soit n est l'ordre de G. L'algorithme produit bien un arbre couvrantde G puisqu'il termine lorsque tous les sommets de X sont dans S et à chaque itération une arête est choisie. Donc T contient tous les sommets de G et a exactement $n-1$ arêtes.

Supposons que T ne soit pas minimal. Soit alors A un arbre couvrant minimal qui a le maximum d'arêtes en commun avec T.

Si $e_1, \cdots, e_{n-1}$ sont les arêtes de T dans l'ordre de choix de l'algorithme, le poids de T est $p(T) = \sum_{i=1}^{i=n-1} p(e_i)$. Soit j tel que $j = inf\{i, 1 \leq i \leq n; e_i \notin A\}$ (e_j est la première arête de T qui n'appartient pas à A). On notera S_i l'ensemble des sommets du sous-graphe induit par $e_1, \cdots, e_i$. Le graphe $A + e_j$ n'est pas un arbre donc contient un unique cycle C. Or, par construction, e_j relie nécessairement un sommet de S_{j-1} à un sommet de $X - S_{j-1}$. Donc C doit contenir une arête $e \neq e_j$ qui elle aussi relie un sommet de S_{j-1} à un sommet de $X - S_{j-1}$. Le graphe $A + e_j - e$ est alors un arbre couvrant de G. Mais par construction de T, $p(e_j) \leq p(e)$, d'où $p(A + e_j - e) \leq p(A)$: $A + e_j - e$ est alors un arbre couvrant minimal qui a plus d'arête en commun avec T que A ce qui contredit l'hypothèse sur A.

3.2.3 Complexité

Soit n est l'ordre de G. Si S a k sommets, pour choisir une arête de poids minimal relient un sommet de S à un sommet de $X - S$, on doit trouver le poids minimum parmi au plus $k(n-k)$ arêtes puisque chaque sommet de S est adjacent à au plus $(n-k)$ sommets de $X - S$. Or k varie de 1 à n, donc $k(n-k) < (n-1)^2$. Ce choix est effectué n fois dans la boucle tant que. Donc la complexité est de l'ordre $O(n^3)$.

Chapitre 4

Problème du plus court chemin

Le problème posé dans ce chapitre est de déterminer dans un graphe *valué* (orienté ou non orienté) le plus court chemin entre deux sommets. Les arêtes (arcs) du graphe seront donc affectées d'une valeur par une fonction v à valeurs dans $\mathbb{R}$ et on calculera la longueur d'un chemin (chaîne) par la somme des valeurs de ses arêtes (arcs).

exemple :

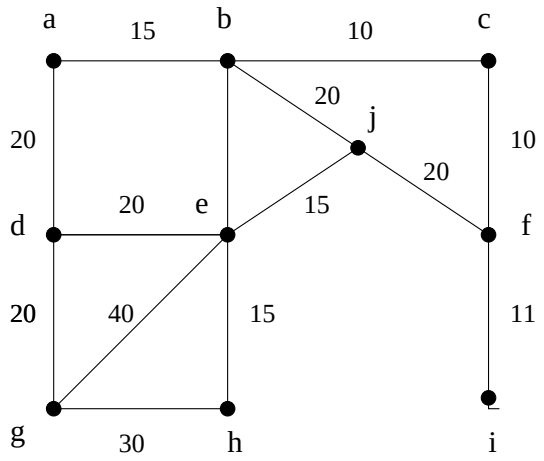


FIGURE 4.1 – Trouver un plus court chemin

Selon la fonction v, on peut être amené à des solutions différentes. Plus précisement, il est aisé de remarquer qu'il n'y a pas de solution lorsque le graphe contient un circuit (ou un cycle) absorbant *i.e.* un circuit pour lequel la somme des valeurs de ses arcs est négative (cf figure 4.2).

On étudiera quatre solutions principales : les algorithmes de Dijkstra, Floyd et Bellman et une variante l'algorithme PAPS.

L'algorithme de Dijkstra fournit une solution pour une fonction v à valeurs dans $\mathbb{R}_+$. L'algorithme de Floyd fonctionne pour toute fonction v en l'absence de circuit absorbant. Une légère modification permet de corriger ce problème. L'algorithme de Bellman permet de trouver une solution et de détecter la présence d'un circuit absorbant. L'algorithme PAPS lui aussi détecte d'éventuel circuit absorbant mais il diffère du précédent par l'utilisation d'une file pour un parcours en largeur.

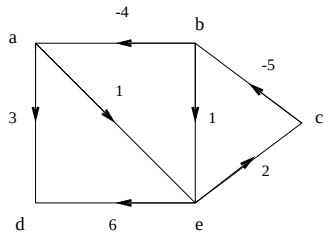


FIGURE 4.2 – un circuit absorbant

4.1 Algorithme de Dijkstra

Un sommet x étant fixé, cet algorithme est un algorithme glouton qui construit progressivement un ensemble de sommets pour lesquels on connaît un plus court chemin depuis x. A chaque étape on choisit un sommet dont la distance à x est minimale parmi ceux qui n'ont pas encore été choisis.

4.1.1 Algorithme

On utilisera un ensemble C de sommets et deux tableaux D et P indexés par les sommets du graphe : D tableau de réels et P tableau de sommets.

PROCÉDURE Dijkstra(G : graphe , x : sommet)
  $C \leftarrow \{x\}$; $D[x] \leftarrow 0$
  POUR tout sommet $s \neq x$ FAIRE
    $D[s] \leftarrow v(x,s)$ ou $+\infty$ si s n'est pas adjacent à x
    $P[s] \leftarrow x$
  FINPOUR
  TANT QUE $|C| < n$ FAIRE
    choisir y tel que $D[y] = min\{D[z]; z \notin C\}$
    SI $D[y] = +\infty$ ALORS fin FINSI
    $C \leftarrow C \cup \{y\}$
    POUR chaque sommet $z \notin C$ FAIRE
      SI $D[z] > D[y] + v(yz)$ ALORS $D[z] \leftarrow D[y] + v(yz)$; $P[z] \leftarrow y$ FINSI
    FINPOUR
  FINTANT QUE
FINPROCÉDURE

exemple : pour le graphe de la figure 4.1, à partir du sommet a on a le tableau suivant :

sommet	a	b	c	d	e	f	g	h	i	j	sommets choisis
	0	15	$+\infty$	20	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	a
	0	**15**	25	20	30	$+\infty$	$+\infty$	$+\infty$	$+\infty$	35	a, b
	0	**15**	25	**20**	30	$+\infty$	40	$+\infty$	$+\infty$	35	a, b, d
	0	**15**	**25**	**20**	30	35	40	$+\infty$	$+\infty$	35	a, b, d, c
D	0	**15**	**25**	**20**	**30**	35	40	45	$+\infty$	35	a, b, d, c, e
	0	**15**	**25**	**20**	**30**	**35**	40	45	46	35	a, b, d, c, e, f
	0	**15**	**25**	**20**	**30**	**35**	40	45	46	**35**	a, b, d, c, e, f, j
	0	**15**	**25**	**20**	**30**	**35**	**40**	45	46	**35**	a, b, d, c, e, f, j, g
	0	**15**	**25**	**20**	**30**	**35**	**40**	**45**	46	**35**	$a, b, d, c, e, f, j, g, h$
	0	**15**	**25**	**20**	**30**	**35**	**40**	**45**	**46**	**35**	$a, b, d, c, e, f, j, g, h, i$

Le tableau P contient les pères des sommets dans un plus court chemin depuis le sommet x.

remarque : **attention**, cet algorithme n'est valable que pour une valuation positive *i.e.* une fonctin v à valeurs dans $\mathbb{R}_+$, comme le montre l'exemple suivant

exemple : dans la figure 4.3, le tableau D évolue de la façon suivante :

sommet	a	b	c	sommets choisis
D	0	1	2	a
D	0	**1**	2	a, b
D	0	**1**	**−1**	a, c, b

Pour relier a à b, le plus court chemin passe par c et vaut 0. Mais l'algorithme ne le détecte pas.

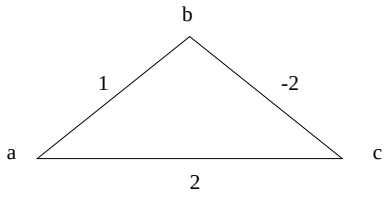


FIGURE 4.3 – une valuation négative

4.1.2 Preuve

L'algorithme termine puisqu'on augmente le nombre de sommets choisis à chaque itération. Avant la boucle TANT QUE, C ne contient que x et $D[x] = 0$ est la longueur d'un plus court chemin depuis x. Au bout de k itérations, supposons que pour chaque sommet z de C, $D[z]$ est la longueur d'un plus court chemin depuis x. Soit y le sommet choisi dans $X - C$ à la $k+1$ itération. Soit $x = x_0, x_1, x_2, \cdots, x_{l-1}, x_l = y$ un chemin σ quelconque de x à y s'il existe. Soit i le plus grand entier tel que $x, x_1, x_2, \cdots, x_i$ soient des sommets de C. Par hypothèse de récurrence, $D[x_i] \leq \sum_{j=0}^{j=i-1} p(x_j, x_{j+1})$. Donc, puisque la fonction p est à valeurs positives, la longueur $l(\sigma)$ du chemin σ est supérieure à $D[x_i] + \sum_{j=i+1}^{j=l-1} p(x_j, x_{j+1})$.

Le sommet x_i a été choisi au cours des k itérations précédentes donc on a nécessairement $D[x_i] + p(x_i, x_{i+1}) \geq D[x_{i+1}]$.

Mais par choix de y, $D[y] \leq D[x_{i+1}]$.

Il reste donc $D[y] \leq D[x_i] + p(x_i, x_{i+1}) \leq l(\sigma)$ *i.e.* $D[y]$ est inférieur à la longueur d'un chemin quelconque depuis x. □

4.1.3 Complexité

Dans l'algorithme on utilise implicitement l'ensemble $X - C$ que l'on notera CC. C'est l'ensemble des sommets parmi lesquels on choisira le prochain y.

La boucle TANT QUE a au plus $(n-1)$ itérations car on choisit un sommet à chaque itération.

Le choix du sommet y se fera en temps $\Theta(|CC|)$ et puisque $|CC|$ diminue d'une unité à chaque itération, on aura alors $\sum_{i=1}^{i=n}(i-1)$ comparaisons au total soit $\Theta(n^2)$.

Supprimer y de CC prend un temps au maximum $\Theta(n)$ opérations.

La boucle POUR chaque sommet z de CC FAIRE la comparaison $D[z] > D[y] + v(yz)$ se fera au plus $d(y)$ fois donc au plus $\Theta(m)$ fois au total.

On peut améliorer le choix et la suppression du sommet y en représentant CC par un tas ordonné selon les valeurs en cours du tableau D.

Dans ce cas la complexité est au pire $\Theta(max(m, n \log n))$.

4.2 Algorithme de Floyd

Cet algorithme est une adaptation de l'algorithme de Roy-Warshall(cf 1.8.2).

4.2.1 Algorithme

On utilise une matrice carrée M d'ordre n initialisée par les valeurs $v(i,j)$. On utilise une matrice carrée P d'ordre n dans laquelle $P_{i,j}$ sera le père de j dans un plus court chemin depuis i.

PROCÉDURE Floyd $(M, P$: matrice)
  POUR i de 1 à n FAIRE
    POUR j de 1 à n FAIRE $M_{i,j} \leftarrow v(i,j)$ $P_{i,j} \leftarrow i$ FINPOUR FINPOUR
  POUR k de 1 à n FAIRE
    POUR i de 1 à n FAIRE
      POUR j de 1 à n FAIRE
      SI $M_{i,k} + M_{k,j} < M_{i,j}$ ALORS $M_{i,j} \leftarrow M_{i,k} + M_{k,j}; P_{i,j} \leftarrow P_{k,j}$ FINSI
  FINPOUR FINPOUR FINPOUR
FINPROCÉDURE

exemple :

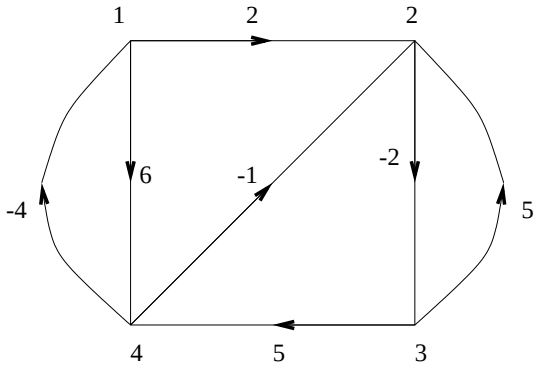


FIGURE 4.4 – Un graphe sans circuit absorbant

On a

$$M = \begin{pmatrix} 0 & 2 & +\infty & 6 \\ +\infty & 0 & -2 & +\infty \\ +\infty & 5 & 0 & 5 \\ -4 & -1 & +\infty & 0 \end{pmatrix} \; et \; P = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 \end{pmatrix}$$

puis pour $k = 1$

$$M = \begin{pmatrix} 0 & 2 & +\infty & 6 \\ +\infty & 0 & -2 & +\infty \\ +\infty & 5 & 0 & 5 \\ -4 & -2 & +\infty & 0 \end{pmatrix} \; et \; P = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 \\ 4 & 1 & 4 & 4 \end{pmatrix}$$

puis pour $k = 2$

$$M = \begin{pmatrix} 0 & 2 & 0 & 6 \\ +\infty & 0 & -2 & +\infty \\ +\infty & 5 & 0 & 5 \\ -4 & -2 & -4 & 0 \end{pmatrix} \; et \; P = \begin{pmatrix} 1 & 1 & 2 & 1 \\ 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 \\ 4 & 1 & 2 & 4 \end{pmatrix}$$

puis pour $k = 3$

$$M = \begin{pmatrix} 0 & 2 & 0 & 5 \\ +\infty & 0 & -2 & 3 \\ +\infty & 5 & 0 & 5 \\ -4 & -2 & -4 & 0 \end{pmatrix} \; et \; P = \begin{pmatrix} 1 & 1 & 2 & 3 \\ 2 & 2 & 2 & 3 \\ 3 & 3 & 3 & 3 \\ 4 & 1 & 2 & 4 \end{pmatrix}$$

puis pour $k = 4$

$$M = \begin{pmatrix} 0 & 2 & 0 & 5 \\ -1 & 0 & -2 & 3 \\ 1 & 3 & 0 & 5 \\ -4 & -2 & -4 & 0 \end{pmatrix} \; et \; P = \begin{pmatrix} 1 & 1 & 2 & 3 \\ 4 & 2 & 2 & 3 \\ 4 & 1 & 3 & 3 \\ 4 & 1 & 2 & 4 \end{pmatrix}$$

On peut modifier l'algorithme de Floyd afin qu'il détecte la présence d'un circuit de longueur négative de la façon suivante :

```
PROCÉDURE Floyd 2 (M, P : matrice)
  POUR i de 1 à n FAIRE
    POUR j de 1 à n FAIRE M_{i,j} ← v(i,j) ; P_{i,j} ← i FINPOUR FINPOUR
  POUR k de 1 à n FAIRE
    POUR i de 1 à n FAIRE
      SI M_{i,j} ≠ +∞ ALORS
        SI M_{i,k} + M_{k,i} < 0 ALORS FIN ; circuit absorbant
        SINON
          POUR j de 1 à n FAIRE
          M_{i,j} ← min{M_{i,j}, M_{i,k} + M_{k,j}}
          SI M_{i,j} < M_{i,k} + M_{k,j} ALORS P_{i,j} ← P_{k,j} FINSI
          FINPOUR
        FINSI
      FINSI
    FINPOUR
  FINPOUR
FINPROCÉDURE
```

4.2.2 Preuve

La matrice M initiale contient les plus courts chemins de i à j ne passant par aucun autre sommet. On peut montrer par récurrence qu'à l'étape k, $M_{i,j}$ est égal à un plus court chemin de i à j passant uniquement par les sommets $1, \cdots, k$. Cela constitue l'invariant de la boucle POUR.

4.2.3 Complexité

Elle est clairement en $\Theta(n^3)$.

4.3 Algorithme de Bellman

Cet algorithme recherche un plus court chemin depuis un sommet fixé x dans un graphe quelconque ou bien détecte la présence d'un circuit absorbant.

4.3.1 Algorithme

On utilise un tableau de distance D indexé par les sommets du graphe et la liste des voisins (pour un graphe non orienté) ou bien la liste des prédécesseurs (dans la cas d'un graphe orienté).

PROCÉDURE Bellman (G : graphe ; x : sommet)

{initialisation} $D^0[x] \leftarrow 0$,

POUR tout sommet $y \neq x$ FAIRE $D^0[y] \leftarrow +\infty$ FINPOUR

$k \leftarrow 0$

TANT QUE $k < n$ FAIRE

$D^k[x] \leftarrow 0$

POUR tout sommet y FAIRE

$D^{k+1}[y] \leftarrow min(D^k[y], min\{D^k[z] + w(zy)) \; ; zy \in E\}$

FINPOUR

SI $D^{k+1} = D^k$ ALORS FIN FINSI

$k \leftarrow k + 1$

FINTANT QUE

SI $k = n$ ALORS il existe un circuit de longueur négative.

FINPROCÉDURE

exemple : dans la figure 4.5, si on choisit $x = a$, le tableau D évolue ainsi

sommet	a	b	c	d	e
D^0	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$
D^1	0	3	-1	$+\infty$	$+\infty$
D^2	0	3	-1	-4	0
D^3	0	-3	-1	-4	0
D^4	0	-3	-2	-4	0
D^5	0	-3	-2	-5	-1

L'algorithme s'arrête avec $k = 5$ et le circuit absorbant a été detecté.

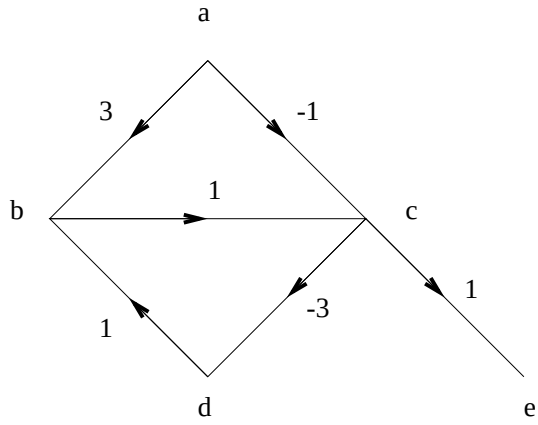


FIGURE 4.5 – Un graphe avec un circuit absorbant

4.3.2 Preuve

$D^0[y]$ représente la valeur d'un chemin de x à y de longueur minimum et contenant 0 arête (arc). On peut montrer par récurrence sur k que, à la k-ième itération de la boucle TANT QUE, $D^k[y]$ représente la valeur d'un chemin de x à y de longueur minimum et ne contenant pas plus que k arêtes (arcs).

S'il n'existe pas de circuit absorbant, il existe un plus court chemin (élémentaire) de x à y de moins de $n-1$ arêtes (arcs) et le tableau D se stabilise en moins de $n-1$ itérations à la valeur d'un plus court chemin de x à y, pour chaque sommet y.

S'il n'y a pas de stabilisation alors il existe un circuit absorbant.

4.3.3 Complexité

La boucle TANT QUE demande chaque fois m additions et m comparaisons donc la complexité est en $O(m \times n)$.

4.4 Algorithme PAPS

Cet algorithme est une variante du précédent.

4.4.1 Algorithme

On utilisera un tableau de distance D indexé par les sommets du graphe, un tableau de sommets P indexé par les sommets du graphe une file Φ de sommets , une variable entière l qui sera égale à la longueur de la file et un compteur k qui permet d'arrêter l'algorithme en présence d'un circuit absorbant.

PROCÉDURE PAPS (G : graphe ; x : sommet)
  POUR tout sommet y FAIRE $D[y] \leftarrow +\infty$; $P[y] \leftarrow x$ FINPOUR
  $k \leftarrow 0$; $\Phi \leftarrow \emptyset$; $enfiler(\Phi, x)$; $l \leftarrow 1$
  TANT QUE $k < n$ ET $\Phi \neq \emptyset$ FAIRE
    $c \leftarrow 0$;
    POUR i de 1 à l FAIRE
      $y \leftarrow premier(\Phi)$; $defiler(\Phi)$; $l \leftarrow l - 1$
      POUR tout successeur z de y FAIRE
        SI $D[y] + v(yz) < D[z]$ ALORS
        $D[z] \leftarrow D[y] + v(yz)$; $P[z] \leftarrow y$; $enfiler(\Phi, z)$; $c \leftarrow c + 1$
        FINSI
      FINPOUR
    FINPOUR

$k \leftarrow k+1$; $l \leftarrow c$

FINTANT QUE

SI $\Phi \neq \emptyset$ ALORS il existe un circuit absorbant.

FINPROCÉDURE

4.4.2 Preuve

Elle est laissée au lecteur.

4.4.3 Complexité

Elle est identique à la complexité de algorithme de Bellman. En effet, la boucle POUR i de 1 à l a au maximum m itérations et la boucle TANT QUE n itérations. Donc la complexité de l'algorithme PAPS est en $\Theta(m \times n)$.

Chapitre 5

Graphes orientés

Dans ce chapitre on s'intéressera à certains problèmes spécifiques aux graphes orientés comme la détermination des composantes fortement connexes d'un graphe orienté, la détection de circuits, le tri topologique *etc.*

5.1 Composantes fortement connexes

Contrairement au cas d'un graphe non orienté, un parcours de graphe (en largeur ou en profondeur) ne permet pas de déterminer les composantes fortement connexes d'un graphe orienté.

Un parcours d'un graphe orienté depuis un sommet x permet de visiter les sommets *accessibles* depuis x. L'idée est d'alors *inverser* les arcs du graphe orienté et de recommencer le parcours. Il est alors nécessaire de choisir les sommets à partir desquels on effectue ces parcours.

Pour cela, on numérote les sommets en ordre de *post-visite* au cours du premier parcours en profondeur en suivant l'algorithme suivant dans lequel c est une variable globale entière initialisée à 0 et $i\_post[x]$ est l'indice de post-visite.

PROCÉDURE P_prof_or(G : graphe ; x : sommet)

$marque[x] \leftarrow true$

POUR chaque successeur y de x FAIRE SI $marque[y] = false$ ALORS P_prof_or(G ; y)FINSI

FINPOUR

$c \leftarrow c + 1$

$i\_post[x] \leftarrow c$

FINPROCÉDURE

On peut alors déterminer les composantes fortement connexes en suivant les instructions suivantes :

1. effectuer un parcours en profondeur à partir d'un sommet quelconque (relancer le parcours jusqu'à ce que tous les sommets soient marqués)
2. construire le graphe G^r qui a les mêmes sommets que G mais dont les arcs ont l'orientation inverse.

3. effectuer un parcours en profondeur de G^r à partir du sommet qui a l'indice de post-visite le plus élevé. L'arborescence obtenue est la composante fortement connexe de ce sommet.
4. s'il reste des sommets non marquées par cette deuxième visite, il faut choisir celui qui a l'indice de post-visite le plus élevé pour relancer l'instruction 3.

exemple : sur les figures 5.1 et 5.2 suivantes, on peut voir les parcours des graphes G et G^r ainsi que les arborescences obtenues (le parcours suit l'ordre alphabétique). Dans le cas de G^r, ce sont les composantes fortement connexes de G.

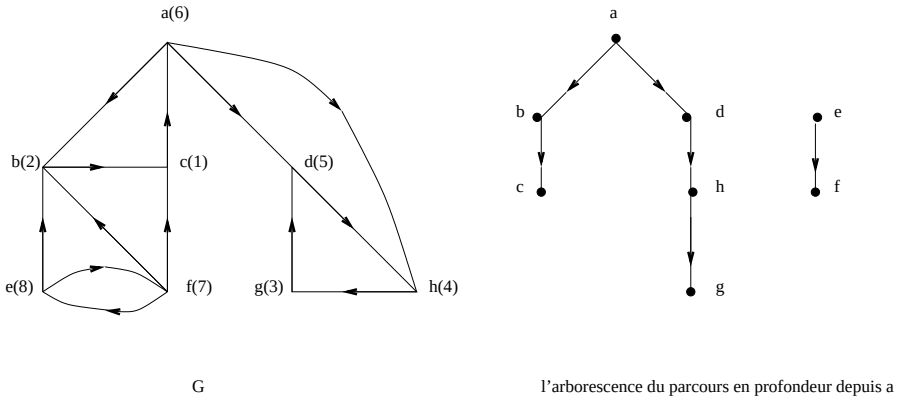


FIGURE 5.1 – Un parcours de graphe orienté

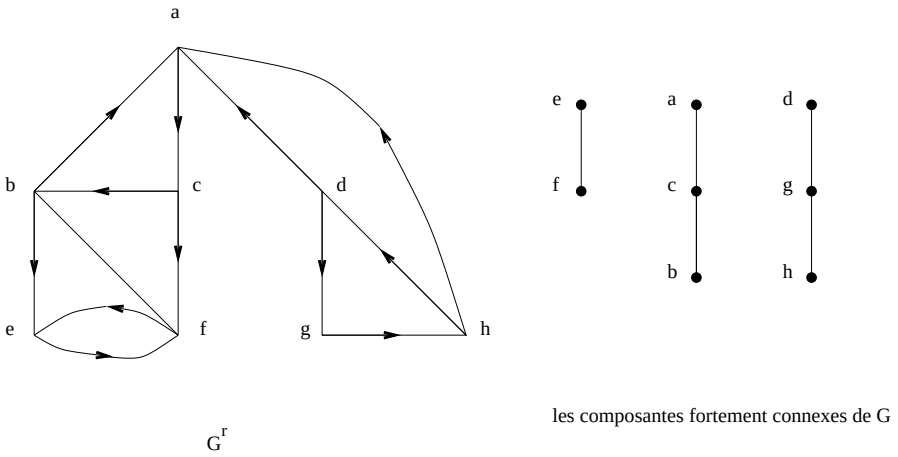


FIGURE 5.2 – Le parcours du graphe renversé

Les arcs n'appartenant à aucune composante fortement connexe sont des arcs inter-composantes. On peut alors obtenir le *graphe réduit* dont les sommets sont les composantes fortement connexes de G et les arcs sont les arcs inter-composantes (cf figure 5.3).

Ce graphe réduit n'a évidemment pas de circuit.

5.2 Graphe sans circuit

Ces graphes sont notamment importants dans le problème de l'ordonnancement.

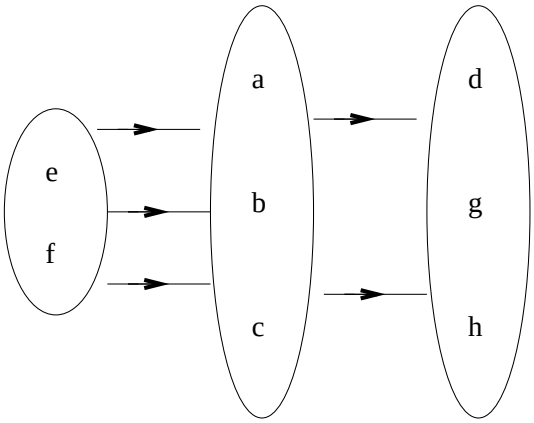


FIGURE 5.3 – Le graphe réduit

5.2.1 Algorithmes

Un graphe orienté comporte un circuit *si et seulement si* lors d'un parcours initié depuis le sommet x, on retombe sur ce sommet. On peut donc adapter l'algorithme de parcours en profondeur pour détecter le présence d'un circuit. On verra deux variantes : l'une utilise une liste des sommets en cours de visite et l'autre un double marquage des sommets.

avec une liste

La procédure de parcours suivante maintient à jour une liste L de sommets dont l'appel sur cette procédure n'est encore fini. Cette liste est initialisée à $\emptyset$. On utilise de plus un tableau de booléens *visite* initialisé à $false$ et indexé par les sommets du graphe.

PROCÉDURE visite_liste(G : graphe ; x : sommet)
  SI $x \in L$ ALORS répondre 'non' ; FIN
  SINON
    $visite[x] \leftarrow true$
    $L \leftarrow L \cup \{x\}$
    POUR chaque successeur y de x FAIRE visite_liste(G ; y)FINPOUR
    $L \leftarrow L - \{x\}$
  FINSI
FINPROCÉDURE

Cette procédure visite_liste est utilisée dans la procédure suivante

PROCÉDURE sans circuit(G : graphe)
  POUR chaque sommet x FAIRE SI $visite[x] = false$ ALORS visite_liste(G ; x)FINSI FINPOUR
  répondre 'oui'
FINPROCÉDURE

double marquage

La procédure de parcours suivante marque de deux façons les sommets : $marque[x]$ est $true$ lorsque x est ou a été visité et $en\_cours[x]$ est $true$ lorsque l'appel de la procédure sur x n'est pas terminé. Les tableaux de booléens $marque$ et en_cours sont initialisés à $false$ et indexés par les sommets du graphe.

PROCÉDURE visite_marque(G : graphe ; x : sommet)
  $marque[x] \leftarrow true$
  $en\_cours[x] \leftarrow true$
  POUR chaque successeur y de x FAIRE
    SI $en\_cours[y] = true$ ALORS répondre 'non' ; FIN
    SINON SI $marque[y] = false$ ALORS visite_marque(G ; y) FINSI
    FINSI
  FINPOUR
  $en\_cours[x] \leftarrow false$
FINPROCÉDURE

Cette procédure est alors utilisé dans la procédure suivante :

PROCÉDURE sans circuit 2(G : graphe orienté)
  POUR chaque sommet x FAIRE
    SI $marque[x] = false$ ALORS visite_marque(G ; x)FINSI
  FINPOUR
  répondre 'oui'
FINPROCÉDURE

Dans un graphe sans circuit, il y a nécessairement parmi les sommets des sommets qui n'ont pas de successeur et des sommets qui n'ont pas de prédécesseur : les premiers sont appelés *sommet puits* et les seconds *sommet source*. Comment considérer les autres sommets ? On peut effectuer une sorte de classement appelé tri topologique.

5.2.2 Tri topologique

Définition 5.1 *Soit G un graphe orienté d'ordre n. On appelle tri topologique des sommets de G une bijection σ de X dans $\{1 \cdots n\}$ telle que si (x, y) est un arc alors $\sigma(x) < \sigma(y)$.*

Il est clair que si G a un circuit, un tel tri est impossible. La réciproque est vraie :

Propriété 5.1 *Un graphe orienté est sans circuit si et seulement si G admet un tri topologique de ses sommets.*

remarque : un graphe orienté sans circuit peut avoir plusieurs tris topologiques.

exemple : dans la figure 5.4, on peut classer ainsi les sommets

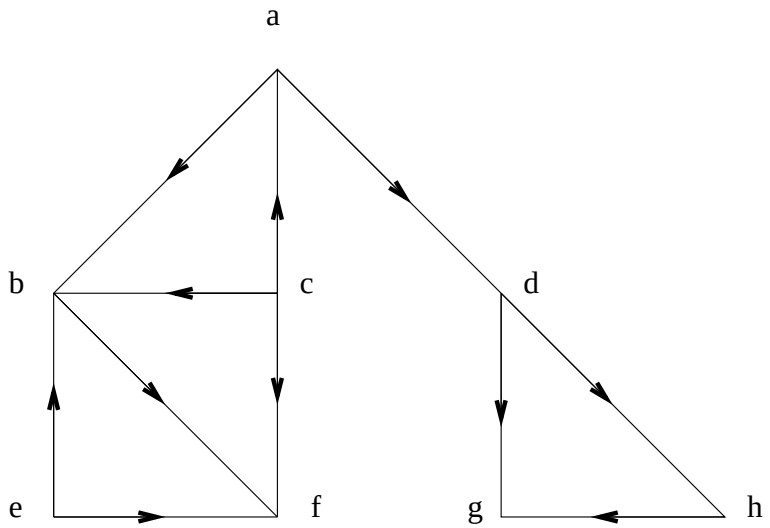


FIGURE 5.4 – Un graphe sans circuit

– e, c, a, b, d, h, g, f
– c, e, a, b, f, d, h, g
– e, c, a, b, f, d, h, g

c est un sommet source et g est un sommet puits.

Pour effectuer un tri topologique, on peut utiliser un parcours en profondeur dans lequel les sommets sont empilés après avoir été visités. Soit π une pile de sommets initialisée à $\emptyset$.

PROCÉDURE P_prof 3(G : graphe ; x : sommet)
  $marque[x] \leftarrow true$
  POUR chaque sommet y successeur de x FAIRE
    SI $marque[y] = false$ ALORS P_prof 3(G ; y)FINSI
  FINPOUR
  empiler(x,π)
FINPROCÉDURE

Le dépilement de π permet alors le tri topologique :

PROCÉDURE tri topologique(G : graphe ; x : sommet)
  POUR chaque sommet x FAIRE $marque[x] \leftarrow false$ FINPOUR
  $c \leftarrow 1$
  POUR chaque sommet x FAIRE SI $marque[x] = false$ ALORS P_prof 3(G ; x)FINSI
  FINPOUR
  TANT QUE $\pi \neq \emptyset$ FAIRE
    $\sigma[(premier(\pi))] \leftarrow c$
    $depiler(\pi)$
    $c \leftarrow c + 1$
  FINTANT QUE
FINPROCÉDURE

La fonction σ réalise un tri topologique.

exemple : dans la figure 5.4, si la procédure tri topologique(G ; a) suit l'ordre alphabétique la fonction σ a les valeurs suivantes

sommet	a	b	c	d	e	f	g	h
σ	3	7	2	4	1	8	6	5

5.3 Orientation

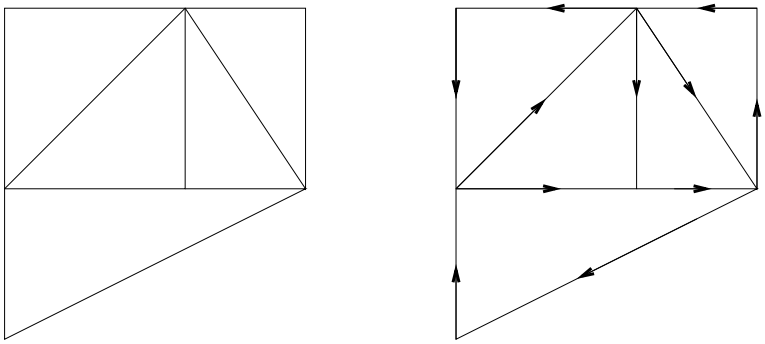
Le problème abordé dans ce paragraphe peut être illustré cd la façon suivante :

Dans une ville, un quartier formé de rues étroites à double sens est perpétuellement encombré. Afin de désengorger ce quartier, la mairie décide de rendre toutes ses rues à sens unique : est-ce possible ? Comment réaliser ce projet ?

La question de la possibilité d'une telle démarche sous-entend le problème suivant : peut-on orienter les voies de ce quartier tel qu'aucune rue ne devienne inaccessible depuis une autre ?

Les rues seront représentées par des arêtes et les croisements seront les sommets du graphe qui représentera le quartier.

Dans le graphe de la figure 5.5, c'est possible alors que dans le graphe de la figure 5.6, c'est impossible.



le graphe non orienté sa version orientée

FIGURE 5.5 – On peut orienter

Le résultat suivant fournit une caractérisation et un algorithme pour l'orientation de tels graphes.

Définition 5.2 *Un graphe non orienté connexe est fortement orientable s'il existe une orientation de ses arêtes qui le transforme en graphe orienté fortement connexe.*

Théorème 5.2 *Un graphe non orienté connexe est fortement orientable si et seulement si il n'a pas d'isthme.*

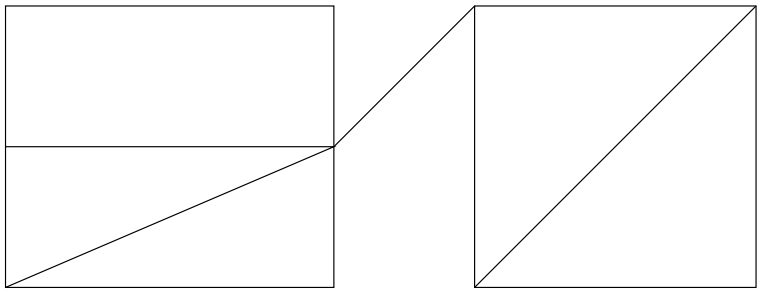


FIGURE 5.6 – On ne peut pas orienter

Preuve : La condition nécessaire est claire.

Réciproquement, soit G un graphe non orienté sans isthme. Alors chaque arête de G appartient à un cycle. Soit $C = x_1, \cdots x_k, x_1$ un cycle de G. On oriente alors l'arête $\{x_k, x_1\}$ en arc (x_k, x_1) puis chaque arête $\{x_i, x_{i+1}\}$ en arc (x_i, x_{i+1}) pour $i = 1, \cdots k-1$. S'il existe des arêtes reliant deux sommets du cycle C qui n'ont pas encore été orientées, on les oriente indifféremment. Le sous-graphe orienté $G_0 = G(x_1, \cdots x_k)$ est fortement connexe puisqu'il a un circuit hamiltonien.

Si tous les sommets de G sont dans C, le problème est résolu. Sinon, puisque G est connexe, il existe un sommet y_1 n'appartenant pas à C adjacent à un sommet x_j de C. L'arête $\{y_1, x_j\}$ n'est pas un isthme donc appartient à un cycle $C_1 = y_1, y_2 = x_j, \cdots, y_l, y_1$. On oriente alors l'arête $\{y_1, x_j\}$ en arc (y_1, x_j), l'arête $\{y_l, y_1\}$ en arc (y_l, y_1), puis pour tout $i = 2, \cdots, l-1$ on oriente les arêtes $\{y_i, y_{i+1}\}$ non encore orientées en arc (y_i, y_{i+1}). Remarquons que si les sommets étaient déjà reliés par un arc, alors ces deux sommets appartiennent à G_0. Après ces orientations, si une arête reliant deux sommets de C_1 ou un sommet de C à un sommet de C_1 n'a pas encore reçu d'orientation, on l'oriente indifféremment.

On montre alors que le sous-graphe orienté G_1 engendré par $x_1, \cdots, x_k, y_1, \cdots, y_l$ est fortement connexe. En effet, on construit le circuit suivant : on part de y_1 vers y_2 puis pour $i = 2, \cdots, l-1$ on ajoute l'arc (y_i, y_{i+1}) s'il existe sinon le chemin reliant y_i à y_{i+1} dans G_0. Puis on revient à y_1 et à y_2 par les arcs (y_l, y_1) et (y_1, y_2). On peut alors continuer par le circuit $y_2, x_2, \cdots, x_k, x_1, x_2$. Si tous les sommets de G ne sont pas dans G_1 on poursuit cette procédure jusqu'à ce que ce soit le cas. □

Chapitre 6

Ordonnancement

Dans ce chapitre, on étudiera le problème de coordination de travaux : un ensemble de tâches doit être accompli en respectant certaines contraintes. Ces contraintes peuvent être de plusieurs natures, par exemple :

- contraintes de précédence : une tâche ne peut être exécutée que si une ou plusieurs autres sont achevées
- contraintes absolues : une tâche doit être achevée avant une date précise *etc.*
- contraintes d'incompatibilité : deux tâches ne peuvent s'exécuter simultanément.

On ne verra que les contraintes de précédence simples pour lesquelles les tâches ont une durée fixée.

exemple : la réfection d'un amphithéâtre nécessite les travaux suivants :

- pose de haut-parleur
- pose d'une moquette
- peinture
- pose d'un tableau
- pose de sièges

La pose de haut-parleur demande une demi-journée de travail ; la pose de la moquette demande deux jours de travail ; la pose du tableau demande une demi-journée de travail ; la pose des sièges demande deux jours de travail et enfin la peinture nécessite quatre jours de travail.

Evidemment, la peinture doit précéder tout autre travail et les sièges ne peuvent être installés qu'après la moquette.

On peut résumer ces données dans le tableau suivant :

tâche	durée	contraintes de précédence
haut-parleur HP	1	
moquette M	4	S
tableau T	1	
sièges S	4	
peinture P	8	HP, M, T, S

Dans la méthode potentiels–tâches (MPM pour *Method of Project Management*), la modélisation se fait par un graphe orienté dont les sommets sont les tâches, un arc indique que la tâche origine doit précéder la tâche but et chaque arc est valué par la durée de son origine. On complète

alors le graphe avec deux sommets α et ω : α représente la tâche fictive de début des travaux et précède chaque sommet sans prédécesseur par un arc de valeur nulle ; ω représente la tâche fictive de fin des travaux et succède à chaque sommet sans successeur par un arc valué par la durée de la tâche origine (cf figure 6.1).

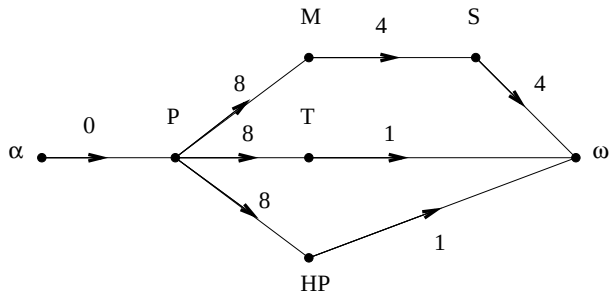


FIGURE 6.1 – Le graphe orienté représentant les contraintes d'exécution

Le problème est d'établir un calendrier des travaux respectant les contraintes et minimisant la durée totale.

Par exemple, pour la réfection d'un amphithéâtre décrite par le graphe 6.1, la durée minimale des travaux est de 16 demi-journées.

On peut noter les deux remarques suivantes :
- il est clair que ce calendrier ne peut être établi si le graphe a un circuit (ce qu'on appellerait un cercle vicieux)
- le graphe peut être bien plus important que celui de l'exemple précedent et un "bon" schéma permet une résolution plus aisée. "Bon" signifie en particulier que la lecture du graphe depuis le sommet α jusqu'au sommet ω permet de voir les tâches pouvant être exécutées simultanément et l'ordre de ces groupes de tâches.

Par exemple, pour la réfection d'un amphithéâtre décrite par le graphe 6.1, la peinture doit être faite en premier puis parallèlement moquette, tableau et haut-parleur peuvent être posés et enfin les sièges après la moquette.

Cela nous amène à définir la notion de rang dans un graphe orienté acyclique.

6.1 Rang d'un sommet dans un graphe orienté

Définition 6.1 *Dans un graphe orienté sans circuit, pour un sommet x, $rang(x)$ est la longueur maximale d'un chemin de but x.*

exemple : dans le graphe de la figure 5.4, on a
- $rang(c) = rang(e) = 0$
- $rang(a) = 1$
- $rang(b) = rang(d) = 2$
- $rang(f) = rang(h) = 3$
- $rang(g) = 4$

Pour calculer cette fonction *rang*, il suffit donc de résoudre le problème du plus long chemin par opposition au problème du plus court chemin (cf chapitre 4).

Une solution possible consiste à utiliser les degrés entrants des sommets :

FONCTION rang(G : graphe) : entier naturel
  $k \leftarrow 0$; $S_0 \leftarrow \emptyset$
  POUR chaque sommet x FAIRE
    $d[x] \leftarrow d^-(x)$;
    SI $d^-(x) = 0$ ALORS $S_0 \leftarrow S_0 \cup \{x\}$ FINSI FINPOUR
  TANT QUE $S_k \neq \emptyset$ FAIRE
    $S_{k+1} \leftarrow \emptyset$
    POUR chaque sommet $y \in S_k$ FAIRE
      $rang[y] \leftarrow k$
      POUR chaque sommet z successeur de y FAIRE
        $d[z] \leftarrow d[z] - 1$;
        SI $d[z] = 0$ ALORS $S_{k+1} \leftarrow S_{k+1} \cup \{z\}$ FINSI
      FINPOUR
    FINPOUR
    $k \leftarrow k + 1$
  FINTANT QUE
FINFONCTION

La complexité de cet algorithme est $O(max(m, n))$ puisque chaque sommet est vu une fois dans S_k pour une valeur de k correspondant à son rang et chaque arc est vu une fois pour les successeurs de chaque sommet.

6.2 Dates au plus tôt et au plus tard

On utilise la méthode MPM. Soit G le graphe orienté valué par une fonction d à valeurs dans $\mathbb{R}_+$; G représente les contraintes de précédence d'un problème d'ordonnancement. G a un unique sommet α de degré entrant nul et un unique sommet ω de degré sortant nul qui représentent respectivement les tâches fictives de début de travaux et de fin de travaux.

On cherche donc un ordonnancement des travaux qui minimise la durée totale des travaux donc la date de fin des travaux.

Pour qu'une tâche puisse commencer, il est nécessaire que toutes les tâches qui la précèdent soient terminées : on va donc calculer la longueur d'un plus long chemin depuis le sommet α pour déterminer cette date que l'on appellera date au plus tôt.

Pour une tâche x, la date au plus tôt de x est $d_{tot}[x] = max_{(zx)\in E}\ (d_{tot}[z] + d(z))$.

La durée minimale des travaux sera alors la date au plus tôt de la tâche ω soit la longueur d'un plus long chemin depuis α jusqu'à ω.

On envisagera deux solutions pour déterminer cette durée minimale :

– parcourir l'arborescence des chemins acycliques au moyen d'un parcours en profondeur
– utiliser la fonction *rang*

Cela donne dans le premier cas la procédure suivante dans laquelle D est un tableau de réels indexé par les sommets de G, P un tableau de sommets indexé par les sommets de G et L est une liste de sommets. On initialise D et P par

POUR chaque sommet $z \neq \alpha$ FAIRE $D[z] \leftarrow \infty$; $P[z] \leftarrow \alpha$ FINPOUR

$D[\alpha] \leftarrow 0$

PROCÉDURE Durée_minimale(G : graphe , x : sommet)

$L \leftarrow L + x$

POUR tout successeur y de x tel que $y \notin L$ FAIRE

SI $D[\alpha, y] = \infty$ OU $D[\alpha, y] < D[\alpha, x] + v(x, y)$

ALORS $D[\alpha, y] \leftarrow D[\alpha, x] + v(x, y)$; $P[y] \leftarrow x$ FINSI

Durée_minimale(G , y)

FINPOUR

$L \leftarrow L - x$

FINPROCÉDURE

L'appel de Durée_minimale(G, α) donne les valeurs des dates au plus tôt pour chaque tâche ainsi que la durée minimale du projet.

remarque : la liste L a pour rôle d'éviter les circuits mais dans le cas de l'ordonnancement on peut supposer qu'elle est inutile.

exemple : pour l'exemple de la réfection d'un amphithéâtre, on a alors

tâche	α	P	M	T	HP	S	ω
D	0	∞	∞	∞	∞	∞	∞
D	0	0	∞	∞	∞	∞	∞
D	0	0	8	∞	∞	∞	∞
D	0	0	8	∞	∞	12	∞
D	0	0	8	∞	∞	12	16
D	0	0	8	8	∞	12	16
D	0	0	8	8	8	12	16

Dans l'autre solution, la fonction *rang* permet d'écrire efficacement la fonction suivante :

FONCTION date au plus tôt(G : graphe) : entier naturel

$d_{tot}[\alpha] \leftarrow 0$;

POUR chaque sommet x par ordre de rang croissant FAIRE

$d_{tot}[x] \leftarrow max_{(zx)\in E}\ (d_{tot}[z] + d(z))$;

FINPOUR

FINFONCTION

Une fois déterminée la durée minimale du projet, on peut remarquer ques le retard de certaines tâches entraîne un retard global des travaux : ce sont des tâches *critiques*. Il existe au moins un

chemin dit critique dont tous les sommets sont des tâches critiques : c'est un plus long chemin de α à ω. Au contraire, certaines autres tâches peuvent être différées, dans une certaine limite appelée marge, sans pour autant retarder la fin des travaux. Pour calculer cette marge, on va déterminer la date au plus tard d_{tard} pour chaque tâche en posant $d_{tard}[\omega] \leftarrow d_{tot}[\omega]$. On cherche donc le "dernier moment" auquel exécuter une tâche sans provoquer de retard global.

Pour une tâche x, on détermine la première tâche (dans le temps) qui doit lui succéder et on tient compte alors de la durée de x : $d_{tard}[x] = min_{(xy)\in E}\ (d_{tard}[y]) - d(x)$. Cela correspond à $d_{tard}[x] = d_{tot}[\omega] - l(x,\omega)$ où $l(x,\omega)$ est la longueur d'un plus long chemin de x à ω.

Le graphe est donc parcouru depuis le sommet ω et les sommets pour lesquels les dates au plus tôt et au plus tard sont égales correspondent aux tâches critiques. Pour les autres sommets, leur marge est obtenue par la différence entre leurs date au plus tard et date au plus tôt.

FONCTION date au plus tard(G : graphe) : entier naturel
  $d_{tard}[\omega] \leftarrow d_{tot}[\omega]$;
  POUR chaque sommet x par ordre de rang décroissant FAIRE
    $d_{tard}[x] \leftarrow min_{(xy)\in E}\ (d_{tard}[y]) - d(x)$;
  FINPOUR
FINFONCTION

exemple : pour le problème de la figure 6.1, on a le tableau des valeurs suivantes :

tâche	P	M	T	HP	S	ω
d_{tot}	0	8	8	8	12	16
d_{tard}	0	8	15	15	12	16
marge	0	0	7	7	0	0

Le chemin critique est ici α,P,M,S,ω.

remarque : une autre méthode potentiels–étapes (PERT pour Project Evaluation and Review Technique) que l'on ne détaillera pas ici, modélise chaque tâche par un arc valué par sa durée et dont l'origine représente le début de la tâche et le but sa fin.

6.3 Marge libre

On a vu la notion de marge totale. Cette marge, si elle est utilisée peut obliger les tâches suivantes à elles-même utiliser leur marge totale.

Une notion plus fine de marge libre permet de calculer pour chaque tâche une marge qui n'aura aucune influence sur le comportement des tâches suivantes.

Par exemple, si une tâche A de durée 10 a pour date au plus tôt 12, date au plus tard 17 et si elle précède deux tâches B et C dont les dates au plus tôt sont respectivement 24 et 27 (à cause d'autres prédécesseurs) alors A pourra commencer au plus tard à la date 14 (et se terminer à la date 24) sans que B et C voient leur date au plus tôt perturbée. Sa marge libre est alors de 2.

Le calcul de la marge libre se fait par la fonction suivante :

FONCTION marge libre(G : graphe) : entier naturel

POUR chaque sommet x FAIRE

$marge\_libre[x] \leftarrow min_{(xy)\in E}\ (d_{tot}[y]) - d_{tot}[x] - d(x)$;

FINPOUR

FINFONCTION

6.4 Algorithme de Bellman

L'algorithme de Bellman modifié permet de calculer les trois fonctions lorsque le graphe ne contient pas de circuit ce qui est le cas pour ces problèmes. On utilise un ensemble S de sommets initialisé à $\emptyset$. Cet ensemble permet de choisir les sommets par ordre de rang croissant sans calculer ce rang.

PROCÉDURE Bellman_ modifié (G : graphe)

$S \leftarrow S \cup \{\alpha\}$; $d_{tot}[\alpha] \leftarrow 0$

TANT QUE il existe un sommet $x \notin S$ dont tous les prédécesseurs sont dans S FAIRE

$d_{tot}[x] \leftarrow max_{(zx)\in E}(d_{tot}[z] + d(z))$; $S \leftarrow S \cup \{x\}$

FINTANTQUE

$S \leftarrow \emptyset$

$S \leftarrow S \cup \{\omega\}$; $d_{tard}[\omega] \leftarrow d_{tot}[\omega]$

TANT QUE il existe un sommet $x \notin S$ dont tous les successeurs sont dans S FAIRE

$d_{tard}[x] \leftarrow min_{(xz)\in E}(d_{tard}[z]) - d(x)$; $S \leftarrow S \cup \{x\}$

FINTANTQUE

POUR chaque sommet x FAIRE $marge(x) \leftarrow d_{tard}[x] - d_{tot}[x]$ FINPOUR

FINFONCTION

Index