Graphes : Théorie, Algorithmes et Implémentation

Olivier Baudon
Université Bordeaux 1
351, cours de la Libération, 33405 Talence Cedex, France

21 février 2012

Table des matières

1 Eléments de théorie des Graphes **2**
- 1.1 Définitions . . . 2
  - 1.1.1 Graphe non orienté . . . 2
  - 1.1.2 Graphe orienté . . . 4
  - 1.1.3 Propriétés élémentaires . . . 5
  - 1.1.4 Graphes non étiquetés . . . 6
  - 1.1.5 Chaînes . . . 6
  - 1.1.6 Connexité . . . 7
- 1.2 Sous-graphe . . . 8
- 1.3 Zoologie . . . 9

2 Algorithmes **10**
- 2.1 Parcours de graphes . . . 10
  - 2.1.1 Parcours en largeur . . . 10
  - 2.1.2 Parcours en profondeur . . . 11
  - 2.1.3 Applications du parcours en profondeur . . . 11
- 2.2 Arbre de poids minimum . . . 12
  - 2.2.1 Algorithme de Kruskal . . . 12
  - 2.2.2 Algorithme de Prim . . . 12
- 2.3 Plus court chemin . . . 13
  - 2.3.1 Plus court chemin entre deux sommets . . . 13
  - 2.3.2 Plus courts chemins entre toutes paires de sommets . . . 15
- 2.4 Flots . . . 17
  - 2.4.1 Algorithme de Ford et Fulkerson . . . 17

Résumé

Ces notes présentent la théorie des graphes, les algorithmes et les structures de données nécessaires à une utilisation de ce modèle par ordinateur.

Chapitre 1

Eléments de théorie des Graphes

1.1 Définitions

1.1.1 Graphe non orienté

Un *graphe* G est un couple (V, E) formé de deux ensembles finis $V = \{v_1, \ldots, v_n\}$ et $E = \{e_1, \ldots, e_m\}$, avec $n > 0$, $m \geq 0$, et où pour tout i, e_i est composée de deux éléments de V, pas forcément distincts. V est appelé l'ensemble des *sommets* et E l'ensemble des *arêtes*. L'*ordre* de G est le nombre de sommets, généralement noté n et sa *taille* le nombre d'arêtes, généralement désigné par m.

Dans la suite, nous noterons par $V(G)$, $E(G)$, $n(G)$ et $m(G)$, l'ensemble des sommets, l'ensemble des arêtes, l'ordre et la taille d'un graphe G, ou plus simplement par V, E, n et m s'il n'y a pas d'ambiguité sur le graphe.

Relations dans un graphe non orienté

Soient v_1 et v_2 les deux sommets (pas forcément distincts) constituant l'arête e. e est notée v_1v_2. On dit que v_1 et v_2 sont *incidents* à e, ou encore que ce sont les *extrémités* de e, ou que e relie v_1 et v_2. v_1 et v_2 sont dits *adjacents*, ou encore *voisins*.

L'ensemble des voisins d'un sommet v, appelé le voisinage de v, sera noté $Adj(v)$.

On voit ici que la vision d'un graphe peut être ensembliste (ensembles de sommets, d'arêtes, de voisins) ou relationnelle (relation d'incidence, de voisinage). D'autre part, la relation de voisinage peut être déduite de celle d'incidence, l'inverse n'étant pas vrai.

Multigraphe et graphe simple

Deux arêtes sont dites *parallèles* si elles ont les mêmes extrémités. Un ensemble d'arêtes parallèles est appelé *arête multiple* .

Une arête reliant un sommet à lui-même est appelée une *boucle*.

Un graphe est dit *simple* s'il est sans boucle, ni arête multiple. Un graphe possédant des arêtes multiples sera parfois appelé un *multigraphe*.

Seule la relation d'incidence permet de définir complètement un graphe possédant des arêtes multiples. Dans le cas d'un graphe simple, la relation de voisinage suffit.

Dégré

Le degré d'un sommet v, noté $deg(v)$, est le nombre d'arêtes incidentes à v, une boucle étant comptée deux fois.

On notera par $\delta(G)$ le degré minimum d'un sommet de G et par $\Delta(G)$ le degré maximum. En d'autres termes :

$$\delta(G) = min\{deg(v), v \in V(G)\}$$

$$\Delta(G) = max\{deg(v), v \in V(G)\}$$

Représentation graphique

Le plus souvent, pour définir un graphe, on préférera l'utilisation d'un dessin plutôt que de donner ses ensembles de sommets et d'arêtes de manière exhaustive. Par exemple, le graphe $G = (\{a, b, c, d, e, f\}, \{ab, af, bc, bd, be, cd, cf, de, ef\})$ est représenté par la figure 1.1.

Ce graphe est simple, de degré minimum 2 (degré du sommet a) et maximum 4 (degré du sommet b). Les voisins du sommets c, de degré 3, sont les sommets b, d et f.

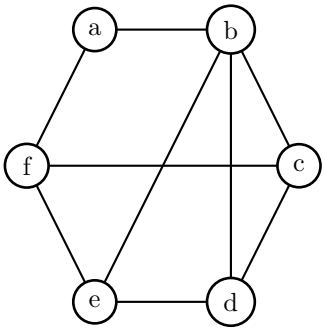


FIGURE 1.1 – Graphe simple

Le graphe de la figure 1.2 est un exemple de graphe "non simple". En effet, il existe une arête multiple entre les sommets c et d et une boucle sur le sommet e. Les degrés des sommets c, d et e sont respectivement 3, 3 et 4.

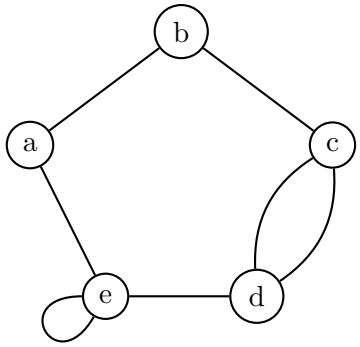


FIGURE 1.2 – Graphe avec boucle et arête multiple

1.1.2 Graphe orienté

Un graphe G peut être *orienté*. On ne parlera plus d'arêtes, mais d'*arcs*. Si $e = v_1v_2$ est un arc de G, v_1 sera appelé *origine* ou *extrémité initiale* de e et v_2 *destination* ou *extrémité terminale* de e.
De plus, v_1 sera dit *prédécesseur* de v_2 et v_2 *successeur* de v_1. Enfin, on dira que e relie v_1 à v_2. On notera par $Adj^+(v)$ (respectivement $Adj^-(v)$) l'ensemble des successeurs (respectivement prédécesseurs) d'un sommet v.

Dans un graphe orienté, le degré *entrant* d'un sommet v est le nombre d'arcs ayant v comme destination. Le degré *sortant* de v est le nombre d'arcs ayant v comme origine.
Certains auteurs parlent également de demi-degré intérieur et de demi-degré extérieur.

Dans l'exemple représenté sur la figure 1.3, le sommet b est de degré entrant 2, sortant 1 avec comme ensembles de prédécesseurs $Adj^-(b) = \{a, f\}$ et successeurs $Adj^+(b) = \{c\}$.

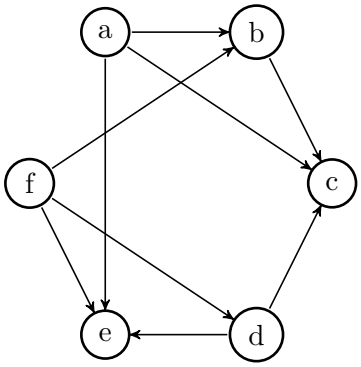


FIGURE 1.3 – Graphe orienté

1.1.3 Propriétés élémentaires

Théorème 1 *La somme des degrés d'un graphe est égale à deux fois son nombre d'arêtes.*

Preuve Il suffit de constater que chaque arête ajoute 2 à la somme des degrés d'un graphe. □

Théorème 2 *Le nombre de sommets de degrés impairs dans un graphe est pair.*

Preuve Puisque la somme des degrés est toujours paire (cf. théorème 1), le nombre d'entiers impairs dans cette somme doit être pair. □

Théorème 3 *Dans un graphe G orienté, la somme des degrés entrant est égale à la somme des degrés sortant et à son nombre d'arêtes.*

Preuve Comme pour le théorème 1, il suffit de constater que chaque arc contribue pour 1 à la somme des degrés entrant et pour 1 à celle des degrés sortants d'un graphe. □

Théorème 4 *Le nombre maximum d'arêtes dans un graphe simple à n sommets est $\frac{n\times(n-1)}{2}$.*

Preuve En effet, dans un graphe simple ayant un maximum d'arêtes, chaque sommet est de degré $n-1$. Par conséquent, la somme des degrés est $n\times(n-1)$ et le nombre d'arêtes $\frac{n\times(n-1)}{2}$. □

1.1.4 Graphes non étiquetés

Isomorphisme de graphes

Deux graphes seront dits *isomorphes* si l'on peut passer de l'un à l'autre par un simple renommage des sommets et des arêtes.

Plus formellement, soit deux graphes $G_1 = (V_1, E_1)$ et $G_2 = (V_2, E_2)$. G_1 et G_2 sont isomorphes si et seulement si il existe une bijection φ de V_1 dans V_2 telle que $\forall v, w \in V_1, vw \in E_1 \Leftrightarrow \varphi(v)\varphi(w) \in E_2$.

Remarque : cette notion s'applique aussi bien à des graphes orientés que non orientés.

Il arrive fréquemment que l'on ne fasse pas la distinction entre un graphe et sa classe d'équivalence. Nous parlerons alors de *graphes non étiquetés*. Inversement, l'ensemble des noms donnés aux sommets d'un graphe est appelé un *étiquetage*.

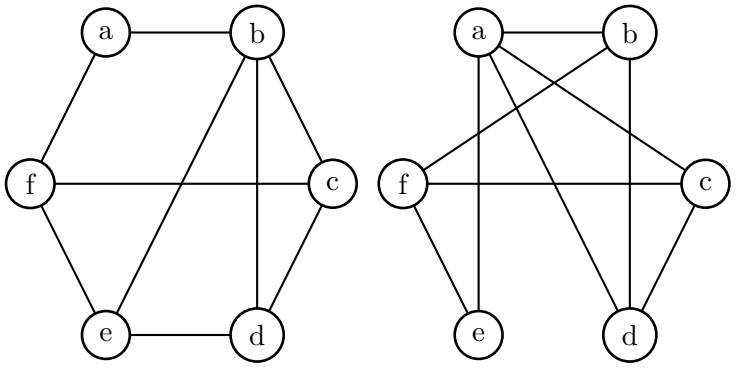


FIGURE 1.4 – Graphes isomorphes

Exemple

Les deux graphes de la figure 1.4 sont isomorphes, l'isomorphisme étant le suivant : $\{(a, e), (b, a), (c, c), (d, d), (e, b), (f, f)\}$.

1.1.5 Chaînes

Notions non orientées

Une *chaîne* est une séquence alternée de sommets et d'arêtes $P_k = v_1, e_1, v_2, \ldots, v_k, e_k, v_{k+1}$, où chaque arête e_i a pour extrémités v_i et v_{i+1}. On dira que P_k est une chaîne de *longueur* k reliant v_1 à v_{k+1}. v_1 et v_{k+1} seront appelés les *extrémités* de la chaîne P_k.

Si $k > 0$ et $v_1 = v_{k+1}$, on dit que P_k est fermée.

Une *chaîne simple* est une chaîne ne passant pas deux fois par la même arête.

Une *chaîne élémentaire* est une chaîne ne passant pas deux fois par le même sommet. Une chaîne élémentaire est donc forcément simple.

Un *cycle* est une chaîne simple fermée, c'est à dire une chaîne simple $C_k = v_1, e_1, v_2, \ldots, v_k, e_k, v_{k+1}$, telle que $v_{k+1} = v_1$.

Un *cycle élémentaire* est une chaîne simple fermée dont tous les sommets sont distincts à l'exception de ses extrémités.

Si le graphe G est simple, il n'est pas nécessaire de préciser l'arête entre deux sommets consécutifs dans une chaîne ou un cycle. Dans ce cas, la chaîne sera donc exprimée comme une simple suite de sommets $v_1, \ldots, v_{k+1}$ ayant comme propriété que deux sommets consécutifs sont toujours voisins.

Notions orientées

Un *chemin* est une séquence alternée de sommets et d'arcs $P_k = v_1, e_1, v_2, \ldots, v_k, e_k, v_{k+1}$, où chaque arc e_i a pour extrémité initiale v_i et extrémité terminale v_{i+1}. De même que pour les chaînes, on dira que P_k est un chemin de *longueur* k reliant v_1 à v_{k+1}.

De même qu'à une chaîne correspond la notion orientée de chemin, à chaîne simple, chaîne élémentaire, cycle, cycle élémentaire, correspondent les notions orientées de *chemin simple*, *chemin élémentaire*, *circuit*, *circuit élémentaire*.

Exemples

Dans le graphe non orienté de la figure **??**,

1.1.6 Connexité

Définitions

Un graphe est dit *connexe* si et seulement si pour toute paire de sommets u, v, il existe une chaîne reliant u et v.

Un graphe orienté est dit *fortement connexe* si et seulement si pour toute paire de sommets u, v, il existe un chemin reliant u à v.

La *composante connexe* d'un sommet v est l'ensemble des sommets u tels qu'il existe une chaîne entre u et v. Un graphe connexe est donc un graphe ayant une seule composante connexe.

La *composante fortement connexe* d'un sommet v d'un graphe orienté G est l'ensemble des sommets u de $V(G)$ tels qu'il existe un chemin de u à v et un chemin de v à u. Un graphe fortement-connexe est donc un graphe ayant une seule composante fortement connexe.

Arbre

Un *arbre* est un graphe connexe sans cycle. Un sommet de degré 1 dans un arbre est appelé une *feuille*.

Une *forêt* est un graphe sans cycle. Autrement dit, une forêt est un graphe dont toutes les composantes connexes sont des arbres.

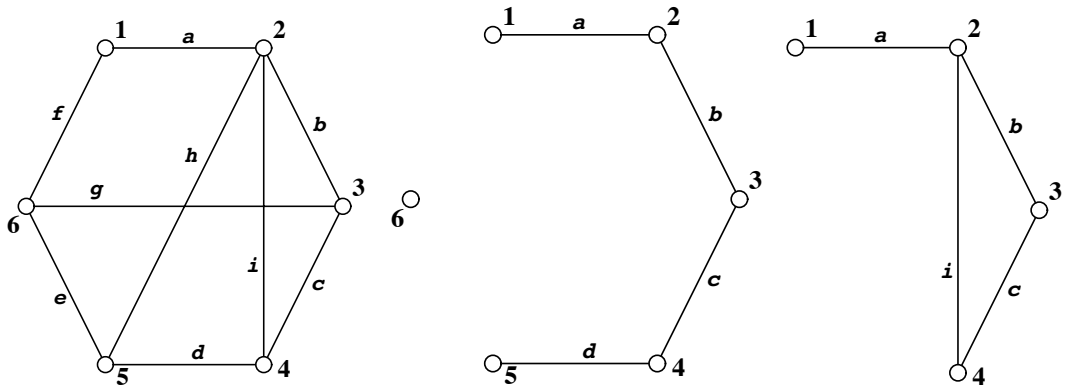
On peut remarquer qu'une forêt est forcément simple, une boucle ou une arête multiple étant considérées comme des cycles.

Une *arborescence* est un arbre dont un sommet r, appelé *racine* est distingué. Une arborescence est naturellement orientée de sa racine vers les feuilles.

1.2 Sous-graphe

Définition 1 *Un graphe* partiel *de $G = (V, E)$ est un graphe $G[F] = (V, F)$ où F est un sous-ensemble de E.*

Définition 2 *Un* sous-graphe induit *de $G = (V, E)$ est un graphe $G(X) = (X, F)$ où $X \subseteq V$ et F est l'ensemble des arêtes ou arcs de E ayant leurs deux extrémités dans X.*



(a) Graphe G (b) Graphe partiel (c) Sous-graphe induit

FIGURE 1.5 – Graphe G, graphe partiel $G[a, b, c, d]$ et sous-graphe induit $G(1, 2, 3, 4)$

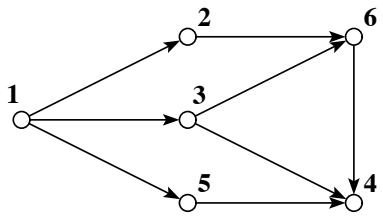


FIGURE 1.6 – Graphe orienté sans circuit

Exemples 1,a,2,b,3,c,4 est une chaîne de longueur 3 du graphe de la figure 1.5a. 2,b,3,c,4,d,5,h,2 est un cycle de longueur 4 de ce même graphe. Par contre, le sous-graphe induit par $\{2, 3, 4, 5\}$ n'est pas un cycle car il contient une arête de trop : l'arête i. Le sous-graphe induit par $\{1, 3, 4\}$ n'est pas connexe.
Le graphe de la figure 1.6 ne possède aucun circuit. Le sous-graphe induit par les sommets 1,2,6 et 4 forme un chemin de ce graphe.

1.3 Zoologie

- Le **graphe complet** à n sommets, noté $\mathbf{K}_n$, est le graphe simple où toute paire de sommets est reliée par une arête.
- Le **cycle** $\mathbf{C}_n$ est le graphe simple connexe d'ordre n où tous les sommets sont de degré 2.
- La **chaîne** $\mathbf{P}_n$ est l'arbre à n sommets où tous les sommets sont de degré au plus 2.

Théorème 5 *La taille du graphe complet K_n est $\frac{n(n-1)}{2}$.*

Théorème 6 *Soit G un graphe simple d'ordre n et de taille m. Les propriétés suivantes sont équivalentes :*

1. *G est un arbre.*
2. *G est connexe et $m = n - 1$.*
3. *G est sans cycle et $m = n - 1$.*
4. *G est connexe et toute suppression d'arête le déconnecte.*
5. *G est sans cycle et tout ajout d'arête crée un cycle.*
6. *Entre tout couple de sommets, il existe une chaîne unique.*

Chapitre 2

Algorithmes

2.1 Parcours de graphes

2.1.1 Parcours en largeur

Algorithm 1 Parcours en largeur PL(G, s)

1: $couleur(s) \leftarrow GRIS$
2: $d(s) \leftarrow 0$
3: $pere(s) \leftarrow NIL$
4: $F \leftarrow \{s\}$
5: **pour tout** $v \in V(G) \backslash s$ **faire**
6: $couleur(v) \leftarrow BLANC$
7: $d(v) \leftarrow \infty$
8: $pere(v) \leftarrow NIL$
9: **fin pour**
10: **tant que** F non vide **faire**
11: $v \leftarrow tete(F)$
12: **pour tout** $w \in Adj(v)$ **faire**
13: **si** $couleur(w) = BLANC$ **alors**
14: $couleur(w) \leftarrow GRIS$
15: $d(w) \leftarrow d(v) + 1$
16: $pere(w) \leftarrow v$
17: $Enfiler(F, w)$
18: **fin si**
19: **fin pour**
20: $Defiler(F)$
21: $couleur(v) \leftarrow NOIR$
22: **fin tant que**

Complexité

2.1.2 Parcours en profondeur

Algorithm 2 Parcours en profondeur PP(G)

1: **pour tout** $v \in V(G)$ **faire**
2: $couleur(v) \leftarrow BLANC$
3: $pere(v) \leftarrow NIL$
4: **fin pour**
5: $temps \leftarrow 0$
6: **pour tout** $v \in V(G)$ **faire**
7: **si** $couleur(v) = BLANC$ **alors**
8: $VisiterPP(v)$
9: **fin si**
10: **fin pour**

Algorithm 3 VisiterPP(v)

1: $couleur(v) \leftarrow GRIS$
2: $d(v) \leftarrow temps \leftarrow temps + 1$
3: **pour tout** $w \in Adj(v)$ **faire**
4: **si** $couleur(w) = BLANC$ **alors**
5: $pere(w) \leftarrow v$
6: $VisiterPP(w)$
7: **fin si**
8: **fin pour**
9: $couleur(v) \leftarrow NOIR$
10: $f(v) \leftarrow temps \leftarrow temps + 1$

2.1.3 Applications du parcours en profondeur

Tri topologique

Au cours d'un parcours en profondeur, à chaque fois que l'on noircit un sommet, il est inséré en tête de liste.

A $PP(G)$, rajouter une ligne `5 bis` :
$liste \leftarrow \{\}$

A $VisiterPP(v)$, rajouter
trois lignes `3.1, 3.2, 3.3`
si $couleur(w) = GRIS$ **alors**
retourner"Existence d'un circuit"

fin si

une ligne `11` :

$INSERER\_TETE(liste, v)$

Composantes fortement connexes

Algorithm 4 CFC(G)

1: Exécuter PP(G) et trier les sommets selon un ordre décroissant de f
2: Calculer G^{-1}
3: Exécuter PP(G^{-1})
4: Retourner les arborescences obtenues comme composantes fortement connexes de G

2.2 Arbre de poids minimum

2.2.1 Algorithme de Kruskal

Algorithm 5 Kruskal(G, w)

1: **pour tout** v de $V(G)$ **faire**
2: $composante(v) \leftarrow \{v\}$
3: **fin pour**
4: Trier $E(G)$ dans un ordre croissant $(e_1, \ldots, e_m)$ en fonction de $w(e)$
5: $i \leftarrow 1$
6: $E(T) \leftarrow \{\}$
7: **tant que** $|E(T)| < n - 1$ **faire**
8: **si** $e_i = (u, v)$ et $composante(u) \neq composante(v)$ **alors**
9: $E(T) \leftarrow E(T) \cup \{e_i\}$
10: UNIFIER($composante(u), composante(v)$)
11: **fin si**
12: $i \leftarrow i + 1$
13: **fin tant que**
14: **retourner** $E(T)$

2.2.2 Algorithme de Prim

Algorithm 6 Prim(G, w)

```
1: F ← FILE_PRIORITE(V(G), cle)
2: pour tout v de V(G) faire
3:    cle(v) ← ∞
4: fin pour
5: cle(r) ← 0
6: pere(r) ← NIL
7: tant que F ≠ ∅ faire
8:    u ← EXTRAIRE_MIN(F)
9:    pour tout v ∈ Adj(u) faire
10:      si v ∈ F et w(u, v) < cle(v) alors
11:         pere(v) ← u
12:         cle(v) ← w(u, v)
13:      fin si
14:   fin pour
15: fin tant que
```

2.3 Plus court chemin

2.3.1 Plus court chemin entre deux sommets

Dijkstra

G : graphe orienté
$w : E(G) \to \mathbb{R}^+$
s : source de G

Bellman

G : graphe orienté sans circuit
$w : E(G) \to \mathbb{R}$
s : source de G

Algorithme de Ford

G : graphe orienté
$w : E(G) \to \mathbb{R}$
s : source de G
L'algorithme renvoie vrai si le graphe G est sans circuit.

Algorithm 7 Dijkstra(G, w, s)

pour tout v de $V(G)$ **faire**
  $d[v] \leftarrow \infty$
  $pere[v] \leftarrow NIL$
  $couleur[v] \leftarrow BLANC$
fin pour
$d[s] \leftarrow 0$
$F \leftarrow FILE\_PRIORITE(V(G), d)$
tant que $F \neq \emptyset$ **faire**
  $pivot \leftarrow EXTRAIRE\_MIN(F)$
  $couleur[pivot] \leftarrow GRIS$
  pour tout $e = (pivot, v)$ arc sortant de $pivot$ **faire**
    si $couleur(v) = BLANC$ et $d[v] > d[pivot] + w(e)$ **alors**
      $d[v] \leftarrow d[pivot] + w(e)$
      $pere[v] \leftarrow pivot$
    fin si
  fin pour
  $couleur[pivot] \leftarrow NOIR$
fin tant que

Algorithm 8 Bellman(G, w, s)

pour tout v de $V(G)$ **faire**
  $d[v] \leftarrow \infty$
  $pere[v] \leftarrow NIL$
  $npred[v) \leftarrow deg^-[v]$
fin pour
$d[s] \leftarrow 0$
INSERER_FILE(F, s)
tant que F non vide **faire**
  $u \leftarrow TETE\_FILE(F)$
  DEFILER(F)
  pour $v \in Adj(u)$ **faire**
    si $d[v] > d[u] + w(u, v)$ **alors**
      $d[v] \leftarrow d[u] + w[u, v]$
      $pere[v] \leftarrow u]$
    fin si
    $npred(v) \leftarrow nped[v] - 1$
    si $npred(v) = 0$ **alors**
      INSERER_FILE(F, v)
    fin si
  fin pour
fin tant que

Algorithm 9 PlusCourtChemin(G, w, s)

```
1: pour tout v de V(G) faire
2:    d[v] ← ∞
3:    pere[v] ← NIL
4: fin pour
5: d[s] ← 0
6: pour i de 1 à n − 1 faire
7:    pour tout arc e ∈ E(G) faire
8:       si d[v] > d[u] + w(u, v) alors
9:          d[v] ← d[u] + w[u, v]
10:         pere[v] ← u]
11:      fin si
12:   fin pour
13: fin pour
14: pour tout arc e ∈ E(G) faire
15:   si d[v] > d[u] + w(u, v) alors
16:      retourner FAUX
17:   fin si
18: fin pour
19: retourner VRAI
```

2.3.2 Plus courts chemins entre toutes paires de sommets

Algorithme de Floyd

G est un graphe orienté, munis d'une fonction de poids sur les arcs w.
Les sommets sont numérotés de 1 à n.
$W_{i,j}$ contiendra la plus courte distance entre le sommet i et le sommet j, $Pred_{i,j}$ le sommet prédécesseur de j sur un plus court chemin de i à j.

Algorithme de Warshall

L'algorithme de Warshall est une adaptation de l'algorithme de Floyd au calcul de la fermeture transitive d'un graphe orienté. $W_{i,j}$ vaudra 1 s'il existe un chemin de i à j dans le graphe G, 0 sinon.

Algorithm 10 Floyd(G, w)

```
1: pour i de 1 à n faire
2:   pour j de 1 à n faire
3:     si i = j alors
4:       W(i,j) ← 0
5:       Pred(i,j) ← i
6:     sinon si ij ∈ E(G) alors
7:       W(i,j) ← w(i,j)
8:       Pred(i,j) ← i
9:     sinon
10:      W(i,j) = ∞
11:      Pred(i,j) ← NIL
12:    fin si
13:  fin pour
14: fin pour
15: pour k de 1 à n faire
16:   pour i de 1 à n faire
17:     pour j de 1 à n faire
18:       si W(i,k) + W(k,j) < W(i,j) alors
19:         W(i,j) ← W(i,k) + W(k,j)
20:         Pred(i,j) ← Pred(k,j)
21:       fin si
22:     fin pour
23:   fin pour
24: fin pour
```

Algorithm 11 Warshall(G, w)

```
1: pour i de 1 à n faire
2:   pour j de 1 à n faire
3:     si i = j || ij ∈ E(G) alors
4:       W(i,j) ← 1
5:     sinon
6:       W(i,j) = 0
7:     fin si
8:   fin pour
9: fin pour
10: pour k de 1 à n faire
11:   pour i de 1 à n faire
12:     pour j de 1 à n faire
13:       W(i,j) ← (W(i,k) & W(k,j)) || W(i,k)
14:     fin pour
15:   fin pour
16: fin pour
```

2.4 Flots

2.4.1 Algorithme de Ford et Fulkerson

G : graphe orienté
$c : E(G) \to \mathbb{R}^+$
s : source de G
t : puit de G
On considère un arc (t, s) de capacité $c(t, s)$ infinie. La valeur du flot sur cet arc sera la "valeur du flot de s à t".
On note respectivement par $I(e)$ et $T(e)$ l'extrémité initiale et l'extrémité terminale d'un arc e.

Algorithm 12 FlotMax(G, c, s, t)

```
1:  pour tout e de E(G) faire
2:     f[e] ← 0
3:  fin pour
4:  répéter
5:     Marquage(G, c, f, s, t)
6:     si t ∈ Y alors
7:        v ← t
8:        C⁺ ← {(t, s)}
9:        C⁻ ← ∅
10:       tant que v ≠ s faire
11:          e ← A[v]
12:          si v = T[e] alors
13:             C⁺ ← C⁺ ∪ {e}
14:             v ← I[e]
15:          sinon
16:             C⁻ ← C⁻ ∪ {e}
17:             v ← T[e]
18:          fin si
19:       fin tant que
20:    fin si
21:    pour tout e ∈ C⁺ faire
22:       f(e) ← f(e) + δ[t]
23:    fin pour
24:    pour tout e ∈ C⁻ faire
25:       f(e) ← f(e) − δ[t]
26:    fin pour
27: jusqu'à t ∉ Y
```

Algorithm 13 Marquage(G, c, f, s, t)

1: $Y \leftarrow \{s\}$
2: $\delta(s) \leftarrow +\infty$
3: $Max \leftarrow$ faux
4: **tant que** $t \notin Y et Max =$faux **faire**
5: **si** il existe $e = (u, v)$ avec $u \in Y, v \notin Y, f(e) < c(e)$ **alors**
6: $Y \leftarrow Y \cup \{v\}$
7: $A[v] \leftarrow e$
8: $\delta[v] \leftarrow min(\delta[u], c(e) - f(e))$
9: **sinon**
10: **si** il existe $e = (u, v)$ avec $v \in Y, u \notin Y, f(e) > 0$ **alors**
11: $Y \leftarrow Y \cup \{u\}$
12: $A[u] \leftarrow e$
13: $\delta[u] \leftarrow min(\delta[v], f(e))$
14: **sinon**
15: Max $\leftarrow vrai$
16: **fin si**
17: **fin si**
18: **fin tant que**

Bibliographie

[1] T.H Cormen, C. E. Leiserson, R.L. Rivest, and C. Stein. *Algorithmique - 3ème édition - Cours avec 957 exercices et 158 problèmes.* Dunod, 2010.

[2] T.H Cormen, C. E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms, Third Edition.* The MIT Press, 2009.

[3] N.H. Xuong. *Mathématiques discrètes et informatique.* Masson, 1992.

[4] I. Charon, A. Germa, and O. Hudry. *Méthodes d'optimisation combinatoire.* Masson, 1997.

[5] Michel Gondran and Michel Minoux. *Graphes et algorithmes.* Tec & Doc Lavoisier, 2009.

[6] Shimon Even. *Graph Algorithms.* Computer Science Press, 1979.

[7] Michel Sakarovitch. *Graphes et programmation linéaire.* Hermann, 1984.