Cheminement dans les Graphes

1. Chaînes / Chemins

1.1 Chaîne

On appelle chaîne dans un graphe non orienté (resp. orienté) $G=(X, E)$ (resp. $G=(X, U)$), une suite alternée de sommets et d'arêtes (resp. d'arcs) :

$\mu = x_0\, e_1\, x_1 \ldots x_{k-1}\, e_k\, x_k$ (resp. $\mu = x_0\, u_1\, x_1 \ldots x_{k-1}\, u_k\, x_k$)

Tel que pour i de 1 à k, x_{i-1} et x_i sont extrémités de l'arête e_i (resp. de l'arc u_i).

On dit que μ est une chaîne joignant les sommets x_0 et x_k de longueur k.

1.2 Chemin

On appelle chemin dans un graphe orienté $G=(X, U)$, une suite alternée de sommets et d'arcs :

$\gamma = x_0\, u_1\, x_1 \ldots x_{k-1}\, u_k\, x_k$

Tel que pour i de 1 à k,

le sommet x_{i-1} est extrémité initiale de l'arc u_i et le sommet x_i est son extrémité terminale.

On dit que γ est un chemin de x_0 vers x_k de longueur k.

1.3 Chaîne / Chemin simple

On dit qu'un chemin ou une chaîne est simple si tous les arcs ou les arêtes les composant sont distincts.

1.4 Chaîne / Chemin élémentaire

On dit qu'un chemin ou une chaîne est élémentaire si tous les sommets les composant sont distincts.

1.5 Remarques

- La longueur d'une chaîne (chemin) est égale au nombre d'arêtes (arcs) formant cette chaîne (chemin).
- S'il existe dans un graphe un chemin d'un sommet x vers un sommet y, on notera : $x\ \alpha\ y$.
- Toute chaîne (ou chemin) élémentaire est aussi simple. L'inverse n'est pas toujours vrai.
- Dans un graphe simple, une chaîne ou un chemin peuvent être déterminés juste en énumérant la suite des sommets qui les composent.

1.6 Proposition

Soit $G=(X, E)$ un graphe non orienté. De toute chaîne joignant deux sommets x et $y \in X$, on peut extraire une chaîne élémentaire joignant x et y.

Soit $G=(X, U)$ un graphe orienté. De tout chemin allant du sommet $x \in X$ vers le sommet $y \in X$, on peut extraire un chemin élémentaire allant de x à y.

2. Cycles / Circuits

2.1 Chaîne fermée / Chemin fermé

Un chemin (resp. chaîne) dont les extrémités sont confondues est dit chemin fermé (resp. chaîne fermée).

2.2 Cycle

On appelle cycle dans un graphe non orienté (resp. orienté) $G=(X, E)$ (resp. $G=(X, U)$), toute chaîne fermée simple :

$\mu = x_0\, e_1\, x_1 \ldots x_{k-1}\, e_k\, x_k$ (resp. $\mu = x_0\, u_1\, x_1 \ldots x_{k-1}\, u_k\, x_k$) Tel que $k > 0$, et $x_0 = x_k$.

On dit que μ est un cycle de longueur k.

2.3 Circuit

On appelle circuit dans un graphe orienté $G=(X, U)$, tout chemin fermé simple :

$\gamma = x_0\, u_1\, x_1 \ldots x_{k-1}\, u_k\, x_k$ Tel que $k > 0$, et $x_0 = x_k$.

On dit que μ est un circuit de longueur k.

2.4 Cycle / Circuit élémentaire

On dit qu'un cycle ou circuit est élémentaire si tous les sommets qui les composent sont distincts.

2.5 Remarques

- La longueur d'un cycle ou d'un circuit élémentaire est aussi égale au nombre de sommets formant ce cycle ou circuit.
- Dans un graphe simple, un cycle ou un circuit peuvent être déterminés juste en énumérant la suite des sommets qui les composent.
- Une boucle est un cycle élémentaire de longueur 1.
- Une boucle dans un graphe orienté est un circuit élémentaire de longueur 1.
- Tout cycle est aussi chaîne. Tout circuit est aussi chemin. Tout circuit est aussi cycle. Tout chemin est aussi chaîne.
- Dans le cas d'une k-coloration, tout graphe contenant un cycle de longueur impaire nécessite au minimum 3 couleurs.

2.6 Proposition

Soit $G=(X, E)$ un graphe non orienté (resp. $G=(X, U)$ un graphe orienté). De tout cycle (resp. tout circuit) passant par une arête $e \in E$ (resp. un arc $u \in U$), on peut extraire un cycle (resp. un circuit) élémentaire passant par e (resp. u).

2.7 Existence d'un cycle

Si G est un graphe vérifiant $\delta(G) \geq k \geq 2$ Alors G contient un cycle.

Si de plus G est simple alors G admet un cycle élémentaire de longueur $\geq k+1$ et une chaine élémentaire de longueur $\geq k$.

Conséquence : Si $m \geq n$ (m étant la taille de G et n le nombre de sommets dans G) alors G admet un cycle.

2.8 Existence d'un circuit

Si G est un graphe vérifiant $\delta^+(G) \geq k \geq 1$ (resp. $\delta^-(G) \geq k \geq 1$) Alors G contient un circuit.

Si de plus G est simple alors G admet un circuit élémentaire de longueur $\geq k+1$ et un chemin élémentaire de longueur $\geq k$.

3. Matrice de fermeture transitive

Soit $G=(X, U)$ un 1-graphe orienté d'ordre n. A partir de sa matrice d'adjacence M, on peut calculer la matrice de fermeture transitive de G qu'on note $\hat{M}$ où chaque élément $\hat{m}_{ij} = \begin{cases} 1 \ si \ \exists i \alpha j \\ 0 \ sinon \end{cases}$

3.1 Calcul Matriciel Direct

On peut avoir la matrice de fermeture transitive par calcul matriciel comme suit : $\hat{M} = \bigvee_{l=1}^{n} M^{[l]}$

Où chaque matrice $M^{[l]}$ se calcule par récurrence (sur l) à travers le produit matriciel booléen comme suit :

$$\begin{cases} M^{[1]} = M \\ M^{[l+1]} = M^{[l]} * M \end{cases}$$

Chaque élément de $M^{[l]}$: $m_{ij}^{[l]} = \vee (\underset{k=1}{\overset{n}{m}}_{ik}^{[l-1]} \wedge m_{kj})$ où l varie de 2 à n.

La matrice $M^{[l]}$ représente tous les chemins dans G de longueur l.

3.2 Algorithme de Warshall

Le calcul direct de la matrice transitive $\hat{M}$ nécessite trop d'opérations matricielles. L'algorithme de Warshall, donné ci-dessous, permet de la calculer avec un gain considérable en nombre d'opérations : n^2 tests et au plus n^3 opérations $\vee$, c'est donc un algorithme en $O(n^3)$

```
Algorithme Warshall (Entrée : M : « Matrice »; Sortie : M : « Matrice »)
   Début
     Pour j de 1 à n Faire
         Pour i de 1 à n Faire
             Si M[i,j] = 1 Alors
                Pour k de 1 à n Faire M[i,k]  = M[i,k] ∨ M[j,k]  Fait;
             Fsi;
         Fait;
     Fait;
   Fin.
```

4. Exploration (Parcours) d'un graphe

L'exploration d'un graphe est un parcours (via les arcs ou les arêtes) permettant d'examiner de façon exhaustive (visiter) les sommets. L'exploration d'un graphe permet d'étudier une ou plusieurs propriétés du graphe tel que la connexité, la forte connexité, biparti, …

4.1 Algorithme

Principe :

- Il consiste à déterminer l'ordre dans lequel seront visités les sommets. Ainsi, le parcours commence d'un sommet de départ *r* qu'on appelle *racine* et donne comme résultat une liste ordonnée de sommets où *r* apparaît en premier et les autres sommets apparaissent une seule fois.

Données :

- En entrée :
  - `X` : ensemble de *n* éléments représentant les identificateurs des sommets.
  - `U` : ensemble de *m* éléments sous forme (`i`, `j`) représentant les arcs où $i \in X$ et $j \in X$.
  - `r` : identificateur du sommet de départ (la racine).
- En sortie :
  - `P` : Liste ordonnée de sommets.
- Autres :
  - `L` : Liste ordonnée de sommets qui peut être implémentée en tant que pile ou file (liste FIFO).

L'algorithme :

```
P ← ∅ ;     L ← {r} ;
Tant que ((L ≠ ∅) et (P ≠ X))
      Choisir_extraire (i∈L) ;
      Pour (tout (i, j)∈U)   Si (j∉P) Alors Ajouter j à L ;
      Pour (tout (j, i)∈U)   Si (j∉P) Alors Ajouter j à L ;
      Ajouter i à la fin de P ;
```

Remarque :

- Nous supposons que la fonction `choisir_extraire` existe et consiste à choisir un sommet de *L* de façon déterministe puis le supprime de *L*. Nous expliquons ci-dessous ses différentes implémentations.
- Il est possible d'appliquer cet algorithme sur un graphe non-orienté.

4.2 Exploration en largeur

Cette exploration consiste à parcourir le graphe à partir du sommet de départ la racine (*r*) puis ses voisins puis les voisins des voisins non explorés et ainsi de suite jusqu'à la fin.

Nous pouvons utiliser l'algorithme d'exploration donné ci-dessus en déclarant `L` comme une liste FIFO (premier arrivé, premier sorti) et la fonction `choisir_extraire` devient `defiler` et la fonction `Ajouter` devient `enfiler`.

4.3 Exploration en profondeur

Cette exploration consiste à parcourir le graphe à partir du sommet de départ la racine (*r*) puis tracer une chaîne à partir de ce sommet puis choisir un autre sommet (parmi les voisins des sommets de cette chaîne dans l'ordre) et faire de même jusqu'à la fin.

Nous pouvons utiliser l'algorithme d'exploration donné ci-dessus en déclarant `L` comme une pile (dernier arrivé, premier sorti) et la fonction `choisir_extraire` devient `depiler` et la fonction `Ajouter` devient `empiler`.

5. Connexité

5.1 Définition

Un graphe est dit connexe s'il existe une chaîne joignant chaque paire de sommets *x* et *y* ($x \neq y$).

5.2 Composante connexe

Soit un graphe $G = (X, E)$ (resp. $G = (X, U)$):

- Le sous graphe engendré par un sommet isolé est considéré comme une composante connexe de *G*.
- Si tout sous graphe engendré par un ensemble de sommets $S \subseteq X$ (G_S) est connexe et le sous graphe engendré par $S \cup \{x\}$ et $x \notin S$ n'est pas connexe Alors G_S est une composante connexe de *G*.

Remarque : Un graphe connexe contient une seule composante connexe.

5.3 Algorithme de Connexité

Principe :

- A partir d'un sommet *r*, on tentera d'atteindre tous les autres sommets du graphe en faisant un prolongement d'une chaîne.
- Pour chaque sommet nous allons vérifier les arcs sortants et les arcs entrants.

Données :

- En entrée :
  - `X` : ensemble de *n* éléments représentant les identificateurs des sommets.
  - `U` : ensemble de *m* éléments sous forme (`i`, `j`) représentant les arcs où $i \in X$ et $j \in X$.
  - `r` : identificateur d'un sommet de *X* qu'on appelle racine.
- En sortie :
  - `Connexe` : Booléen, sera à `Vrai` si le graphe est connexe.
- Autres :
  - `CC` : sous-ensemble de *X*. Le sous graphe engendré par cet ensemble représente une Composante Connexe.
  - `Marque` : vecteur de *n* éléments booléens pour marquer les sommets traités.

L'algorithme :

```
C ← {r} ;
Pour (tout i ∈ X)        Marque[i] ← faux ;
Tant que (∃ i∈C tel que Non(Marque[i]))
      Pour (tout (i, j)∈U)   C ← C ∪ {j} ;
      Pour (tout (j, i)∈U)   C ← C ∪ {j} ;
      Marque[i] ← vrai ;
Si C=X Alors Connexe ← Vrai ;
       Sinon Connexe ← Faux ;
```

Remarque :

Nous pouvons aussi utiliser l'algorithme d'exploration afin de vérifier la connexité. Il s'agit juste de vérifier si la sortie `P = X`.

5.4 Graphe *k*-connexe

Un graphe *G* est dit *k*-connexe si et seulement si *G* est connexe d'ordre $n \geq k+1$ et il n'existe pas d'ensemble $S \subset X$ de cardinal *k*-1 tel que le sous graphe engendré par $X-S$ n'est pas connexe.

En d'autres termes, en supprimant moins de *k* sommets, le graphe sera toujours connexe.

5.5 Point d'articulation

Dans un graphe *G* qui est 1-connexe, un point d'articulation est tout sommet $x \in X$, tel que le sous graphe de *G* engendré par le sous-ensemble de sommets $X-\{x\}$ n'est non connexe.

En d'autres termes, en supprimant moins de *k* sommets, le graphe sera toujours connexe.

6. Forte Connexité

6.1 Définition

Un graphe orienté $G=(X, U)$ est fortement connexe (f.c.) s'il existe entre chaque paire de sommets *x* et $y \in X$ ($x \neq y$) un chemin de *x* à *y* ($x \alpha y$) et un chemin de *y* à *x* ($y \alpha x$).

6.2 Composantes fortement connexes (C.F.C.)

Soit $G=(X, U)$ un graphe orienté :

- Le sous graphe engendré par un sommet $x \in X$ tel que $d_G^+(x) = 0$ ou $d_G^-(x) = 0$ forme une composante fortement connexe de *G*.
- Si le sous graphe engendré par un ensemble de sommets $S \subseteq X$ (G_S) est fortement connexe et le sous graphe engendré par $S \cup \{x\}$ et $x \notin S$ n'est pas fortement connexe Alors G_S est une composante fortement connexe de *G*.

6.3 Ascendants / Descendants

Soit $G=(X, U)$ un graphe orienté, On définit pour chaque sommet $x \in X$ deux ensembles :

- L'ensemble des descendants de x : $D(x) = \{ y \in X / x \alpha y \}$
- L'ensemble des ascendants de x : $A(x) = \{ y \in X / y \alpha x \}$

6.4 Algorithme de Calcul de C.F.C.

Principe :

- A partir d'un sommet *r*, on calcule la c.f.c. à laquelle appartient *r*.
- On calcule l'ensemble des descendants de *r* (noté *D*) puis l'ensemble des ascendants de *r* (noté *A*).
- La c.f.c. est $A \cap D \cup \{r\}$

Données :

- En entrée :
  - `X` : ensemble de *n* éléments représentant les identificateurs des sommets.
  - `U` : ensemble de *m* éléments sous forme (`i`, `j`) représentant les arcs où $i \in X$ et $j \in X$.
  - `r` : identificateur d'un sommet de *X* qu'on appelle racine.
- En sortie :
  - `CFC` : sous-ensemble de *X*. Le sous graphe engendré par cet ensemble représente la C.F.C.
- Autres :
  - `Marque` : vecteur de *n* éléments booléens.
  - `A, D`: sous-ensembles de *X*. Ils représentant l'ensemble des ascendants et l'ensemble des descendants de la racine.

L'algorithme :

```
(* Initialisation *)
   D ← {r} ;   Pour (tout i ∈ X)    Marque[i] ← faux ;
(* Recherche des descendants de r *)
   Tant que ((∃ i ∈ D) et (Marque[i]=faux) )
         Marque[i] ← vrai ;
         Pour (tout (i, j) ∈ U )  D ← D ∪ {j} ;
(* Réinitialisation *)
   A ← {r} ;          Pour (tout i ∈ X)      Marque[i] ← faux ;
(* Recherche des ascendants de r *)
   Tant que ((∃ i ∈ A) et (Marque[i]=faux))
         Marque[i] ← vrai ;
         Pour (tout (j, i) ∈ U)   A ← A ∪ {j}
(* Calcul de la C.F.C. *)
   CFC  ← D ∩ A
```

Remarque : Il est possible d'utiliser cet algorithme comme procédure et l'appeler à répétition jusqu'à obtention de toutes les CFC d'un graphe. Il faut aussi savoir qu'il existe d'autres algorithmes plus simplifiés.

6.5 Graphe réduit

A tout graphe orienté $G=(X, U)$ on associe le graphe simple $G_R=(X_R, U_R)$ appelé graphe réduit de *G* défini comme suit :

X_R = { A chaque c.f.c. de *G* correspond un sommet C_i }

$U_R = \{ (C_i, C_j) / i \neq j$ et $\exists x \in C_i$ et $\exists y \in C_j$ et $(x, y) \in U \}$

6.6 Remarques

- Un graphe fortement connexe possède une seule C.F.C.
- Le graphe réduit d'un graphe ne possède pas de circuits.

7. Parcours Eulériens

7.1 Définition

Un parcours Eulerien passe une fois et une seule fois par chaque arête (resp. arc) du graphe. Le parcours peut être une chaîne, un chemin, un cycle ou un circuit. Soit *G* un graphe contenant *m* arêtes (resp. *m* arcs) :

Une chaîne simple, un chemin simple, un cycle simple ou un circuit de longueur *m* est appelé Eulérien.

7.2 Théorème 1 – Euler 1766

Un multigraphe G admet une chaîne Eulérienne si et seulement si il est connexe (à des points isolés près) et le nombre de sommets de degré impair est 0 ou 2.

7.3 Conséquences

Un graphe G admet une chaîne Eulérienne d'un sommet x à un sommet y ($x \neq y$) si et seulement si $d_G(x)$ et $d_G(y)$ sont impairs et $\forall z$ sommet de G ($z \neq x$ et $z \neq y$), on a $d_G(z)$ pair.

Un graphe G admet un cycle Eulérien si et seulement $\forall x$ sommet de G, on a $d_G(x)$ pair.

7.4 Détermination d'une chaîne Eulérienne

- On part d'un sommet a de degré impair (S'il n'y pas de sommets de degrés impairs, on démarre de n'importe quel sommet). On construit une chaîne à partir de a comme suit :
- A chaque étape k on obtient une chaîne de longueur k et le graphe sera réduit à G_k qui correspond au graphe partiel de G engendré par l'ensemble des arêtes (resp. d'arcs) initial auquel on supprime ceux faisant partie de la chaîne.
- A chaque étape k, en arrivant à un sommet x, il faut éviter de prendre toute arête (resp. arc) qui est isthme dans G_k. Sauf s'il s'agit de la seule et unique possibilité, on la prend et on termine.
- On termine lorsqu'il n'y a plus d'arêtes (resp. arcs) dans le graphe G_k.

7.5 Proposition

Un graphe $G=(X, U)$ admet un circuit Eulérien si et seulement si pour tout sommet x, on a $d^+_G(x) = d^-_G(x)$. (On dit que G est pseudo-symétrique)

8. Parcours Hamiltoniens

8.1 Définition

Un parcours Hamiltonien passe une fois et une seule fois par chaque sommet du graphe. Le parcours peut être une chaîne, un chemin, un cycle ou un circuit. Soit G un graphe d'ordre n (contenant n sommets) :

Une chaîne (resp. un chemin) élémentaire de longueur n-1 est appelé chaîne Hamiltonienne (resp. chemin Hamiltonien).

Un cycle (resp. circuit) élémentaire de longueur n est appelé cycle (resp. circuit) Hamiltonien.

8.2 Remarque

- Un graphe qui contient un cycle Hamiltonien est appelé graphe Hamiltonien.
- Un graphe semi-Hamiltonien est un graphe qui contient une chaîne Hamiltonienne, mais pas de cycle Hamiltonien.
- Le plus petit graphe Hamiltonien d'ordre n est le graphe cycle (Graphe connexe non-orienté à n arêtes. Il est 2-régulier)

8.3 Propositions

- Un graphe complet d'ordre $n \geq 3$ est Hamiltonien.
- Tout graphe tournoi (un graphe orienté simple et complet) d'ordre n, noté T_n contient un chemin Hamiltonien.
- Tout tournoi d'ordre n (T_n) fortement connexe contient un circuit Hamiltonien.

8.4 Théorème

Le théorème ci-dessous est utilisé pour démontrer qu'un graphe n'est pas Hamiltonien (ne contient pas de cycle Hamiltonien).

Si $G=(X,E)$ est un graphe Hamiltonien, alors pour tout ensemble de sommets $S \subset X$, on a : $p(G_{X-S}) \leq |S|$

où $p(G_{X-S})$ est le nombre de composantes connexes du sous graphe de G induit par l'ensemble $X-S$

Généralités sur les graphes – Concepts Fondamentaux

1. Introduction

Un graphe est défini par deux ensembles : Un ensemble de sommets noté X et un ensemble de relations entre les sommets noté U ou E. Selon le type de cette relation. On distingue deux grandes classes de graphes : les graphes orientés si la relation est orientée et les graphes non orientés dans le cas contraire.

1.1 Graphe orienté

Un graphe orienté $G=(X, U)$ est défini par les deux ensembles :

- $X = \{ x_1, x_2, \ldots, x_n \}$ avec n entier fini et $n \geq 1$, où chaque $x_i \in X$ est un **sommet** du graphe.
- $U = \{ u_1, u_2, \ldots, u_m \}$ avec m entier positif ou nul et fini, est appelé ensemble des **arcs**. Chaque $u_j \in U$ est une paire ordonnée de sommets, $u_j = (x, y)$. x est appelé extrémité initiale de u_j et y est appelé extrémité terminale de u_j. U peut être vide.

1.2 Graphe non orienté

Un graphe non orienté $G=(X, E)$ est défini par les deux ensembles :

- $X = \{ x_1, x_2, \ldots, x_n \}$ avec n entier fini et $n \geq 1$, où chaque $x_i \in X$ est un **sommet** du graphe.
- $E = \{ e_1, e_2, \ldots, e_m \}$ avec m entier positif ou nul et fini, est appelé ensemble des **arêtes**. Chaque $e_j \in E$ est une paire non ordonnée de sommets, $e_j = \{x, y\} = \{y, x\}$. x et y sont appelés extrémités de e_j. E peut être vide.

1.3 Représentation

On représente généralement un sommet par un point ou un cercle. Un arc est représenté par une flèche et une arête par un trait qui peuvent être courbés.

2. Définitions

- Le nombre de sommets dans un graphe est appelé l'ordre du graphe. Le nombre d'arcs (resp. arêtes) dans un graphe est appelé taille du graphe.
- Si les deux extrémités d'un arc (resp. une arête) sont confondus alors cet arc (resp. cette arête) est appelé(e) boucle**.**
- Si deux arcs (resp. deux arêtes) possèdent les mêmes extrémités, on dit alors qu'ils sont parallèles.
- Un graphe est dit simple s'il ne contient ni boucles ni arêtes parallèles.
- Soit un arc $u=(x, y)$ (resp. une arête $e=\{x, y\}$) :
  - x *et* y sont dits deux sommets adjacents.
  - Pour le cas de l'arc u : x est dit prédécesseur de y. y est dit successeur de x.
  - x et y sont incidents à l'arc u (resp. à l'arête e).
  - L'arc u (resp. à l'arête e) est incident aux sommets x et y.
  - u est incident vers l'extérieur de x et u est incident vers intérieur de y.
- Soit $x \in X$, un sommet du graphe orienté $G=(X, U)$. On définit :
  - $\Gamma^+(x) = \text{Succ}(x) = \{ y \in X / (x, y) \in U \}$ appelé ensemble des successeurs du sommet x.
  - $\Gamma^-(x) = \text{Pred}(x) = \{ y \in X / (y, x) \in U \}$ appelé ensemble des prédécesseurs du sommet x.
- Soit $x \in X$, un sommet du graphe G, on appelle voisin de x tout sommet $y \in X$ différent de x et qui est adjacent à x. Ainsi, on définit l'ensemble V comme suit :
  - $V(x) = \{y \in X-\{x\} / \{x, y\} \in E\}$ pour les graphes non orientés.
  - $V(x) = V^+(x) \cup V^-(x)$ où $V^+(x) = \{y \in X-\{x\} / (x, y) \in U\}$ et $V^-(x) = \{y \in X-\{x\} / (y, x) \in U\}$.
  - $V^+(x)$ (resp. $V^-(x)$) est appelé ensemble des voisins externes (resp. internes) de x.
- Deux arcs (resp. arêtes) sont dits adjacents s'ils ont une extrémité en commun.
- Pour tout graphe orienté, on définit deux applications donnant l'extrémité initiale et terminale d'un arc donné :

$$I : U \to X \qquad T : U \to X$$
$$u=(x, y) \to x \qquad u=(x, y) \to y$$

- On appelle multiplicité d'un arc (x_i, x_j), la valeur m_{ij} correspondant au nombre d'arcs qui relient x_i à x_j. La multiplicité d'un graphe G est le nombre $m(G)$ correspondant au maximum des m_{ij}.
- Un graphe orienté est dit p-graphe si $m(G) = p$.

3. Notion de degré

3.1 Définition 1

Soit $G = (X, E)$ un graphe non orienté (resp. $G = (X, U)$ un graphe orienté). A tout sommet $x \in X$, on peut associer une valeur entière positive ou nulle, notée $d_G(x)$, qu'on appelle degré du sommet x. Cette valeur est définie comme suit :

$d_G(x)$ = nombre de fois où x est extrémité d'un arc (resp. d'une arête).

3.2 Définition 2

Soit $G = (X, U)$ un graphe orienté.

On appelle demi-degré extérieur d'un sommet $x \in X$, la valeur suivante : $d_G^+(x) = |\{ u \in U / I(u)=x \}|$.

On appelle demi-degré intérieur d'un sommet $x \in X$, la valeur suivante : $d_G^-(x) = |\{ u \in U / T(u)=x \}|$.

3.3 Remarques

- Pour tout graphe orienté, nous avons : $d_G(x) = d_G^+(x) + d_G^-(x)$
- Pour tout graphe, nous avons : $d_G(x) \geq |V(x)|$. Si G est simple Alors On a $d_G(x) = |V(x)|$.
- Pour tout graphe orienté, nous avons : $d_G^+(x) \geq |V^+(x)|$ et $d_G^-(x) \geq |V^-(x)|$. Si G est 1-graphe sans boucles Alors On a $d_G^+(x) = |V^+(x)|$ et $d_G^-(x) = |V^-(x)|$.
- On appelle degré minimal d'un graphe G qu'on note par $\delta(G)$, le plus petit degré dans le graphe G. $\delta(G) = \underset{x \in X}{Min} \{d_G(x)\}$
- On appelle degré maximal d'un graphe G qu'on note par $\Delta(G)$, le plus grand degré dans le graphe G. $\Delta(G) = \underset{x \in X}{Max} \{d_G(x)\}$
- Si $d_G(x) = 0$ Alors x est dit sommet isolé.
- Si $d_G(x) = 1$ Alors x est dit sommet pendant.
- Un arc (resp. Une arête) incident(e) à un sommet pendant est appelé(e) pendant(e).

3.4 Formule des degrés

Cas non orienté :

Pour tout graphe : $G = (X, E)$, On a : $\sum_{x \in X} d_G(x) = 2\,|E|$.

Preuve :

Chaque arête a exactement deux extrémité ⇒ Elle est comptée deux fois dans le degré de chaque sommet ⇒ La somme totale des degrés est égale à deux fois le nombre total d'arêtes.

Cas orienté :

Pour tout graphe : $G = (X, U)$, On a : $\sum_{x \in X} d_G(x) = 2\,|U|$ et $\sum_{x \in X} d_G^+(x) = \sum_{x \in X} d_G^-(x) = |U|$.

Preuve :

Chaque arc a exactement une extrémité initiale et une extrémité terminale ⇒ Chaque arc est comptabilisé une fois dans d^+ pour son extrémité initiale et une autre fois dans d^- pour son extrémité finale. Le nombre total d'arcs ayant une extrémité initiale (resp. terminale) est exactement la somme des demi-degrés extérieurs (resp. intérieurs).

Conséquence : De là, on peut déduire que le nombre de sommets de degrés impairs dans un graphe est toujours pair.

4. Représentation machine

4.1 Matrice d'adjacence

A tout graphe d'ordre n on associe une matrice M de n lignes et n colonnes dont les éléments sont notés M_{ij}.

Pour un graphe non orienté $G=(X, E)$, $M_{ij}=M_{ji}$ représente le nombre d'arêtes ayant les sommets i et j comme extrémités. Ainsi, la matrice d'adjacence M d'un graphe non orienté est toujours symétrique. Pour la boucle, on la compte deux fois.

- La somme d'une ligne k = la somme d'une colonne $k = d_G(k)$

Pour un graphe orienté $G=(X, U)$, M_{ij} représente le nombre d'arcs ayant i comme extrémité initiale et j comme extrémité terminale.

- Les coefficients de M pour un graphe simple est binaire avec la diagonale complètement à 0.
- La somme d'une ligne $i = d_G^+(i)$. La somme d'une colonne $j = d_G^-(j)$.

4.2 Matrice d'incidence

A tout **graphe non orienté** $G=(X, E)$, on peut associer une matrice M de n lignes et m colonnes. Où n est le nombre de sommets dans G et m est le nombre d'arêtes dans G.

M_{ij} représente le nombre de fois où le sommet i est incident à l'arête j. Les éléments de M sont dans $\{0, 1, 2\}$

- Si deux colonnes j_1 et j_2 sont identiques alors les arêtes j_1 et j_2 sont parallèles.
- Si un élément $M_{ij}=2$ alors l'arête j est une boucle.

A tout **graphe orienté** $G=(X, U)$, on peut associer une matrice M de n lignes et m colonnes. Où n est le nombre de sommets dans G et m est le nombre d'arcs dans G.

$$M_{ij}=\begin{cases} 1 & \text{si } I(j)=i \\ -1 & \text{si } T(j)=i \\ 0 & \text{si } j \text{ est une boucle ou autre} \end{cases}$$

- Une colonne nulle représente une boucle.
- Dans ce cas, toute boucle dans le graphe est détectée mais son emplacement ne peut pas être précisé à partir de cette matrice.

4.3 Listes

A tout graphe orienté $G=(X, U)$ avec $|X| = n$ et $|U| = m$, on peut associer deux tableaux (vecteurs) PS et LS :

PS : tableau de pointeurs à $n+1$ éléments, où :

- $PS[i]$: pointe sur la case contenant le premier successeur du sommet i dans LS.
- On pose $PS[1] = 1$.
- On pose pour tout $n \geq i \geq 2$: $PS[i] = k$, où $k=PS[i-1]+$ *nombre de successeurs de* i-1
- On pose $PS[n+1] = m+1$
- Si un sommet i n'a pas de successeur, on aura $PS[i]=PS[i+1]$

LS : tableau de m éléments, où :

- Les successeurs d'un sommet i se trouvent entre la case numéro $PS[i]$ et la case $PS[i+1]-1$ du tableau LS.

5. Graphes particuliers

Soit $G = (X, U)$ un graphe orienté (resp. $G = (X, E)$ un graphe non orienté). Soit $A \subset X$ un sous ensemble de sommets et $V \subset U$ (resp. $V \subset E$).

5.1 Sous graphe

Un sous graphe de G engendré par l'ensemble de sommets A est le graphe :

$G_A = (A, U_A)$ où $U_A = \{u \in U / I(u) \in A$ et $T(u) \in A\}$ dans le cas orienté.

$G_A = (A, E_A)$ où $E_A = \{e = \{x, y\} \in E / x \in A$ et $y \in A\}$ dans le cas non orienté.

5.2 Graphe partiel

Un graphe partiel de G engendré par l'ensemble d'arcs (resp. d'arêtes) V est le graphe $G_V = (X, V)$.

5.3 Sous graphe partiel

Un sous graphe partiel de G engendré par l'ensemble de sommets A et l'ensemble d'arcs (resp. d'arêtes) V est le graphe $G_{A,V} = (A, V_A)$.

V_A est l'ensemble d'arcs (resp. arêtes) qui ont leurs deux extrémités dans le sous ensemble V.

5.4 Complément d'un graphe

Le graphe complémentaire de G est noté $\overline{G} = (X, \overline{U})$ (resp. $\overline{G} = (X, \overline{E})$) où :

$\overline{U} = \{(x, y) \in X^2 / x \neq y$ et $(x, y) \notin U\}$ (resp. $\overline{E} = \{ \{x, y\} \in X^2 / x \neq y$ et $\{x, y\} \notin E\}$)

5.5 Line-graph

On appelle graphe représentatif des arêtes d'un graphe non orienté G (ou Line-graph), le graphe $L(G)$ dont les sommets représentent les arêtes de G et deux sommets sont adjacents dans $L(G)$ si et seulement si les arêtes correspondantes de G sont incidentes à un même sommet de G.

6. Propriétés des graphes

6.1 Graphe Simple

Un graphe est dit simple s'il ne contient ni boucles ni arcs parallèles. Si G est simple, on a $d_G(x) = |V(x)|$.

6.2 Graphe Complet

Dans le cas orienté : G est complet **ssi** $\forall x \neq y \in X, (x, y) \notin U \Rightarrow (y, x) \in U$
Dans le cas non orienté : G est complet **ssi** $\forall x \neq y \in X, \{ x, y \} \in E$.
Un graphe simple complet d'ordre n est noté K_n.

6.3 Graphe Régulier

Un graphe G est dit k-régulier si $\forall x$ sommet de G, on a $d_G(x) = k$. En d'autres termes, $\delta(G) = \Delta(G) = k$.
Si $k = 0$, G est un graphe sans arêtes (sans arcs) appelé stable. G est constitué seulement de sommets isolés.
Si $k = 1$, G est constitué d'arcs (arêtes) dispersé(e)s dans l'espace.

6.4 Graphe Symétrique

Cette notion est spécifique aux graphes orientés.
G est symétrique **ssi** $\forall x \neq y \in X, (x, y) \in U \Rightarrow (y, x) \in U$

6.5 Graphe Antisymétrique

Cette notion est spécifique aux graphes orientés.
G est antisymétrique **ssi** $\forall x \neq y \in X, (x, y) \in U \Rightarrow (y, x) \notin U$

6.6 Graphe Transitif

Cette notion est spécifique aux graphes orientés.
G est transitif **ssi** $\forall x, y, z \in X, (x, y) \in U$ et $(y, z) \in U \Rightarrow (x, z) \in U$

6.7 Graphe Biparti

G est dit biparti ssi l'ensemble de ses sommets X admet une partition en 2 sous ensembles X_1 et X_2 avec $X_1 \cap X_2 = \emptyset$ et $X_1 \cup X_2 = X$.
Dans le cas orienté : $\forall (x, y) \in U \Rightarrow x \in X_1$ et $y \in X_2$
Dans le cas non orienté : $\forall \{x, y\} \in E$ $(x \in X_1$ et $y \in X_2)$ ou $(x \in X_2$ et $y \in X_1)$
G est dit biparti complet **ssi** G est dit biparti et $\forall x \in X_1$ et $\forall y \in X_2 \Rightarrow (x, y) \in U$.
Un graphe biparti complet et simple $G=(X_1 \cup X_2, U)$ (resp. $G=(X_1 \cup X_2, E)$)avec $|X_1|=p$ et $|X_2|=q$ est noté $K_{p,q}$.

6.8 Graphe Multiparti

G est dit multiparti **ssi** l'ensemble de ses sommets X admet une partition en p sous ensembles $X_1 \ldots X_p$ ($p \geq 3$). Avec $X_i \cap X_j = \emptyset$ $(i \neq j)$ et $X_1 \cup \ldots \cup X_p = X$.
Dans le cas orienté : $\forall (x, y) \in U \Rightarrow x \in X_k$ et $y \in X_{k+1}$ (avec $1 \leq k \leq p\text{-}1$)
Dans le cas non orienté : $\forall \{x, y\} \in E$ $(x \in X_k$ et $y \in X_{k+1})$ ou $(y \in X_k$ et $x \in X_{k+1})$ avec $1 \leq k \leq p\text{-}1$

7. Stable / Clique

On appelle stable dans un graphe G un sous-ensemble de sommets $S \subseteq X$ et le sous graphe engendré par S est formé de sommets isolés (ne contient aucun arc ou arête).
Chaque partition d'un graphe biparti forme un stable.
On appelle clique dans un graphe G un sous-ensemble de sommets $C \subseteq X$ où le sous graphe engendré par C est un graphe complet.

8. Coloration des sommets d'un graphe

8.1 Définition 1

On appelle k-coloration d'un graphe non orienté $G=(X, E)$, une application φ qui associe à chaque sommet $x \in X$ de G une couleur représentée par un entier entre 1 et k de telle façon que les sommets ont des couleurs distinctes, comme suit :

$$\varphi : X \rightarrow \{1, 2, \ldots, k\}$$
$$x \rightarrow \varphi(x) \quad \text{tel que } \forall y \neq x \in X \text{ si } \{x, y\} \in E \text{ Alors } \varphi(x) \neq \varphi(y).$$

En d'autres termes, deux sommets adjacents ne peuvent pas être coloriés de la même couleur et tous les sommets doivent être coloriés. De ce fait, une k-coloration partitionne l'ensemble des sommets X en k stables où tous les sommets du même stable ont la même couleur.

8.2 Nombre chromatique

On appelle nombre chromatique d'un graphe $G=(X, E)$, le nombre minimal de couleurs nécessaires pour colorier les sommets de ce graphe. Ce nombre est noté $\chi(G)$.

Ainsi, le nombre chromatique est toujours compris entre 1 et le nombre de sommets $n=|X|$.

8.3 Problème de coloration

Il s'agit de réaliser une k-coloration d'un graphe G. k doit être le plus proche possible du nombre chromatique $\chi(G)$. L'algorithme de *Welsh & Powell* est l'un des plus connus pour résoudre ce problème :

Ordonner les sommets par de degrés (ordre décroissant : du plus grand au plus petit). $X = \{x_1, x_2, \ldots, x_n\}$ tel que $d_G(x_i) \geq d_G(x_{i+1})$

Pour i de 1 à n : Affecter à x_i la plus petite couleur possible distincte des couleurs de $V(x_i)$ colorés.

8.4 Proposition 1

Pour tout graphe $G=(X, E)$ tel que $\Delta(G)$ est le degré maximal dans G et $\chi(G)$ est le nombre chromatique de G, nous avons : $\chi(G) \leq \Delta(G)+1$.

8.5 Proposition 2

Pour tout graphe $G=(X, E)$ complet K_n où $n \geq 2$ est l'ordre de G et $\chi(G)$ est le nombre chromatique de G, nous avons : $\chi(G)= \Delta(G)+1$.

8.6 Proposition 3

Pour tout graphe $G=(X, E)$ où $\chi(G)$ est le nombre chromatique de G et $C \subseteq X$ est la plus grande clique dans G, nous avons : $\chi(G) \geq |C|$.

9. Isomorphisme

9.1 Définition 1

Soient deux graphes orientés $G_1= (X_1, U_1)$ et $G_2= (X_2, U_2)$. On dit que G_1 et G_2 sont isomorphes (on note $G_1 \equiv G_2$) **ssi** $\exists f : X_1 \rightarrow X_2$ et $\exists g : U_1 \rightarrow U_2$ deux bijections avec $\forall u \in U_1$, $u=(x, y) \Leftrightarrow g(u)=(f(x), f(y))$.

9.2 Définition 2

Soient deux graphes non orientés $G_1= (X_1, E_1)$ et $G_2= (X_2, E_2)$. On dit que G_1 et G_2 sont isomorphes (on note $G_1 \equiv G_2$) **ssi** $\exists \varphi : X_1 \rightarrow X_2$ une bijection avec $\forall x, y \in X_1$, $e=\{x, y\} \in E_1 \Rightarrow \{\varphi(x), \varphi(y)\} \in E_2$.

9.3 Proposition

Soient deux graphes isomorphes $G_1= (X_1, U_1) \equiv G_2= (X_2, U_2)$ (resp. non $G_1= (X_1, E_1) \equiv G_2= (X_2, E_2)$) alors $|X_1| = |X_2|$ et $|U_1| = |U_2|$ (resp. $|E_1| = |E_2|$) et $\forall x \in X_1$ de degré $d_G(x)$, $\exists y \in X_2$ de degré $d_G(y) = d_G(x)$.

Remarque : La réciproque n'est pas toujours vraie. On peut trouver deux graphes non isomorphes ayant le même nombre de sommets et le même nombre d'arc (ou arêtes).

Les Flots

1. Réseau de transport

Un réseau de transport est un graphe orienté pondéré $G=(X, U, c)$ constitué d'un ensemble de sommets X, un ensemble d'arcs U et on associe à chaque arc $u \in U$ un poids $c(u) \geq 0$ appelé capacité de l'arc u.
Un réseau de transport possède deux (02) sommets particuliers :

- e : entrée du graphe, c'est une source (n'ayant pas de prédécesseurs)
- s : sortie du graphe, c'est un puits (n'ayant pas de successeurs)

2. Flot dans un réseau de transport

Soit $G=(X, U, c)$ un réseau de transport avec $|X|=n$ et $|U|=m$. On appelle flot compatible dans G un vecteur f de m composantes réelles (une par arc) vérifiant :

$\forall u \in U,\ 0 \leq f(u) \leq c(u)$

$\forall x \in X - \{e, s\}, \sum_{u \in \omega^+(\{x\})} f(u) - \sum_{u \in \omega^-(\{x\})} f(u) = 0$ (loi de Kirchhoff)

Tel que : $\omega^+(\{x\})$ est l'ensemble des arcs sortants de x.
$\omega^-(\{x\})$ est l'ensemble des arcs entrants vers x.

De ce fait, nous aurons : $\sum_{u \in \omega^+(\{e\})} f(u) = \sum_{u \in \omega^-(\{s\})} f(u)$

On dit aussi que le flot f est réalisable.

3. Arc saturé par un flot

Soit $G=(X, U, c)$ un réseau de transport. Un arc $u \in U$ est dit saturé par un flot compatible f si et seulement si $f(u)=c(u)$.
Un flot est dit complet si tout chemin de l'entrée du réseau vers la sortie du réseau passe nécessairement par un arc saturé.

4. Réseau résiduel

A partir d'un réseau de transport $G=(X, U, c)$. et d'un flot compatible f, on peut construire un réseau résiduel $G_f=(X, U_f, c_f)$ comme suit (voir algorithme ci-dessous) :

```
Début
 Uf←∅ ;
 Pour tout arc u=(x,y)∈U
  Faire
   Si (f(u)≠0)
    Alors Uf ← Uf ∪ {(y,x)} ;
          cf(y,x) ← f(u) ;
   fSi
   Si (c(u) – f(u)≠0)
    Alors Uf ← Uf ∪ {(x,y)} ;
          cf(x,y) ← c(u) – f(u) ;
   fSi
  Fait
Fin.
```

5. Chemin d'augmentation

Soit un réseau de transport $G=(X, U, c)$. avec $e \in X$ l'entrée de G et $s \in X$ la sortie de G, f un flot compatible, et $G_f=(X, U_f, c_f)$ le réseau résiduel de G associé à f. Un chemin γ dans G_f de e vers s est appelé chemin d'augmentation de capacité $c(\gamma)$ où $c(\gamma) = \underset{u \in \gamma}{Min}(c_f(u))$.

6. Flot maximal

Soit $G=(X, U, c)$ un réseau de transport. Les sommets e et $s \in X$ sont respectivement les entrée et sortie de G.

1. Un flot compatible f est dit maximal dans le réseau de transport G si et seulement si $\sum_{x \in X} f(e,x) = \sum_{x \in X} f(x,s)$ est de valeur maximale.
2. Un flot compatible f est maximal, s'il n'existe aucun chemin d'augmentation dans le réseau résiduel G_f.

7. Algorithme de Ford-Fulkerson

Donnée

En Entrée : Le réseau de transport $G=(X,U,c)$
$e \in X$ et $s \in X$

En Sortie : f vecteur de $m=|U|$ éléments $\in \mathcal{R}$.

L'algorithme

```
Début
     /*Initialisation*/
     Pour tout (u∈U)
      Faire
        f(u) ← 0 ; /*On peut l'initialiser à n'importe quel flot réalisable ≠ 0*/
      Fait
     Gf ← G ;
      /*Processus itératif*/
     Tant Que (∃γ chemin d'augmentation dans Gf)
      Faire
        Pour tout (u=(x,y)∈γ)
         Faire
           Si (u∈U)
            Alors f(u) ← f(u)+c(γ) ;
            Sinon f(y,x) ← f(y,x)-c(γ) ;
           fSi
         Fait
        Construire Gf à partir du nouveau flot f ;
      Fait
Fin.
```

8. Coupe minimale

On appelle coupe minimale du réseau de transport pour un flot donné f, une partition de l'ensemble des sommets X en deux sous-ensembles X_1 et X_2 ($X= X_1 \cup X_2$ et $X_1 \cap X_2=\varnothing$) tel que tout arc allant X_1 de vers X_2 est saturé.
Si un flot admet une coupe minimale alors ce flot est maximal.

Problèmes de cheminement dans les Graphes

1. Graphes sans circuits

1.1 Propositions

Un graphe orienté $G=(X, U)$ est dit sans circuits (S.C.) si et seulement si toute composante fortement connexe est réduite à un sommet.
Un graphe orienté $G=(X, U)$ est dit sans circuits (S.C.) si et seulement si il est isomorphe à son graphe réduit.
Un graphe orienté $G=(X, U)$ est dit sans circuits (S.C.) si et seulement si tout chemin dans G est élémentaire.

1.2 Source et Puits

Un sommet s est appelé source si et seulement si $d^-(s)=0$.
Un sommet p est appelé puits si et seulement si $d^+(p)=0$.
Tout graphe sans circuits possède une source et un puits.
Dans un graphe sans circuits $G=(X, U)$ $\forall x \in X$, l'extrémité initiale ($\boldsymbol{s} \in X$) d'un plus long chemin vers x est une source.
Dans un graphe sans circuit $G=(X, U)$ $\forall x \in X$, l'extrémité terminale ($\boldsymbol{p} \in X$) d'un plus long chemin commençant en x est un puits.

2. Partition en niveaux d'un graphe sans circuits

2.1 Niveau d'un sommet

Soit $G=(X, U)$, un graphe orienté sans circuits. A tout $x \in X$, on associe un entier :
$v(x)$: **niveau** du sommet x et représente la longueur maximale d'un chemin élémentaire se terminant à x.
On affecte par convention à une source s la valeur $v(s)=0$.
Soit $G=(X,U)$ un graphe orienté, et soit $\lambda(G)$ la longueur du plus long chemin élémentaire dans G,
$\lambda(G)=\underset{x\in X}{Max}\ \{v(x)\}$.

2.2 Partition en niveaux :

Soit $G=(X,U)$ est un graphe sans circuits. L'ensemble des sommets X peut être partitionné au maximum en $\lambda(G)+1$ stables. Où chaque sommet de niveau i sera placé dans le stable N_i. Chaque stable N_i représente un niveau de G.

2.3 Proposition :

$G=(X,U)$ est un graphe sans circuits si et seulement si
X admet une partition $\{N_0 \cup N_1 \cup \ldots\ldots \cup N_p\} / x \in N_i \Leftrightarrow v(x)=i$

3. Graphe pondéré (Réseau)

3.1 Poids d'un arc

Soit $G=(X,U)$ un graphe orienté,
On définit $p : U \rightarrow \mathfrak{R}$ une application qui associe pour chaque arc $u \in U$ de G une valeur réelle $p(u)$ appelée poids de l'arc u. Un tel graphe est appelé graphe pondéré, graphe valué ou réseau.

3.2 Poids d'un chemin

On définit le poids d'un chemin γ comme la somme des poids des arcs de γ, $p(\gamma)=\sum_{u\in\gamma} p(u)$. On l'appelle aussi distance.

3.3 Circuit absorbant

Un circuit γ est dit absorbant si son poids est négatif ($p(\gamma)< 0$).

4. Algorithme de Bellman-Ford

4.1 Problème traité

A chaque sommet $x \in X$, on veut associer un chemin de poids optimal joignant la source du graphe $r \in X$ à x dans le réseau $R=(X, U, p)$

4.2 Principe

L'algorithme consiste à affecter à chaque sommet $x \in X$ d'un réseau $R=(X,U,p)$ sans circuits, une valeur $\pi(x)$ (appelée potentiel de x) qui représente le poids du chemin optimal reliant r à x. Il se base sur la liste des prédécesseurs de chaque sommet.

4.3 Les données

En entrée

X : ensemble des sommets
n : Nombre de sommets
U : ensemble d'arcs sous forme (x,y)
p : un vecteur de m éléments où $p[u]$ représente le poids du l'arc u dans le réseau.
r : La source du graphe.

En sortie :

π : vecteur de n éléments. Chaque entrée $\pi[x]$ représente le poids du chemin optimal de r vers x
Pré : vecteur de n éléments. Chaque entrée *Pré[x]* représente le prédécesseur de x dans le chemin optimal de r vers x.

Autres :

S : ensemble des sommets qui sont extrémité terminale d'un chemin commençant par r et pour lesquels $\pi[i]$ est calculé définitivement.

4.4 Algorithme

```
Debut
     S←{r};
     Pour tout x∈X
     Faire
          π[x]← +∞ ;
          Pré[x] ←NULL ;
     Fait
     π[r]← 0;
     Pour tout (x ∈ X-S) tel que (∀ u ∈ U avec T(u)= x on a I(u)∈S)
     Faire
           Pour tout ((y, x) ∈ U)
           Faire
                Si π[x] > π[y]+p[(y,x)]
                      Alors π[x] ← π[y]+p[(y,x)];
                            Pré[x] ← y;
                fSi
           Fait
               S ← S ∪{x} ;
      Fait
Fin
```

4.5 Remarques

- On dit que l'algorithme retourne une arborescence optimale.
- Si le chemin optimal est le chemin de poids maximal, on change dans l'algorithme « *min* » par « *max* » et « $+\infty$ » par « $-\infty$ ».
- Si $\forall x \in X\ p(x)=1$, l'algorithme calculera le plus court chemin en nombre d'arcs.
- La complexité de l'algorithme est $o(n^2)$.

- Si le graphe contient plusieurs sources ou le sommet initial *r* n'est pas la source du graphe, il est exigé de partitionner le graphe en niveaux au préalable et mettre tous les sommets de niveau ≤ *v(r)* dans *S*. A chaque itération, faire le choix du prochain sommet celui qui n'est pas dans *S* ayant le plus petit niveau.

5. Algorithme de Dijkstra

5.1 Problème traité

Le problème est le même que celui traité par Bellman, sauf qu'ici le réseau *R=(X,U,p)* peut être avec ou sans circuits mais à condition que les poids soient tous positifs ou nuls. Le sommet de départ *r* n'est pas nécessairement une source.

5.2 Principe

L'algorithme calcule le chemin de poids minimal en partant du sommet *r* et en prolongeant le chemin à chaque itération. Cette méthode s'appelle, calcul de la plus courte distance de proche en proche. A chaque étape, pour un sommet donné *x*, on ajuste les valeurs $\pi[y]$ pour tout sommet *y* successeur de *x*.

5.3 Les données

En entrée

X : ensemble des sommets
n : Nombre de sommets
U : ensemble d'arcs sous forme *(x,y)*
p : un vecteur de *m* éléments où *p*[*u*] représente le poids du l'arc *u* dans le réseau.
r : un sommet donné de *X*.

En sortie :

π : vecteur de *n* éléments. Chaque entrée $\pi[x]$ représente le poids du chemin optimal de *r* vers *x*
Pré : vecteur de *n* éléments. Chaque entrée *Pré[x]* représente le prédécesseur de *x* dans le chemin optimal de *r* vers *x*.

Autres :

f : vecteur de *n* éléments. Chaque entrée *f*[*k*] contient un sommet de *X*. le vecteur *f* à la fin sera trié dans un ordre croissant par rapport aux valeurs de $\pi[f[k]]$.
S : ensemble des sommets qui sont extrémité terminale d'un chemin commençant par *r*.
I(*u*) : donne l'extrémité initiale de l'arc *u.* *T*(*u*) : donne l'extrémité terminale de l'arc *u.*

5.4 L'Algorithme

```
Début
     s←{r};     k←1;     f[k]←r;
     Pour tout x∈X faire π[x]← +∞ ; fait
     π[r]←0 ;
     Tant que (k<n) et (π[x]<+∞)
     Faire
          Pour tout u∈U / (I(u)=f[k]) et (T(u)∉S)
          Faire
               x = T(u);
               Si (π[x] > π[f[k]]+p[u])
                     Alors π(x) ← π[f[k]]+p[u];
                          Pré[x] ← f[k];
               fSi
          Fait
          x ← y / y ∈ X-S et π[y] minimal;
          k ← k+1;
          f[k] ← x;
          S ←S ∪ {x};
     Fait
Fin
```

5.5 Remarque

- On dit que l'algorithme retourne une arborescence optimale.
- La complexité de l'algorithme est $o(m^2)$ où m est le nombre d'arcs.
- Si le graphe est non orienté, on peut associer à chaque arête $\{x,y\}$ de poids p, deux (02) arcs (x,y) et (y,x) de même poids p, puis on applique l'algorithme de Dijkstra.

6. Algorithme de Bellman-Kalaba

6.1 Problème traité

Le problème est de trouver le poids du chemin optimal à partir d'un sommet donné r vers tout autre sommet dans le graphe. Le graphe peut contenir des circuits et les poids peuvent être positifs nuls ou négatifs. L'algorithme permet de détecter les circuits absorbants.

6.2 Les données

En entrée

X : ensemble des sommets
n : Nombre de sommets
U : ensemble d'arcs sous forme (x,y)
p : un vecteur de m éléments où $p[u]$ représente le poids du l'arc u dans le réseau.
r : un sommet donné de X.

En sortie :

π : vecteur de n éléments. Chaque entrée $\pi[i]$ représente le poids du chemin optimal de r vers i.
Pré : vecteur de n éléments. Chaque entrée *Pré[x]* représente le prédécesseur de x dans le chemin optimal de r vers x.

Autres :

π_old : vecteur de n éléments. Contient les valeurs de π de l'itération précédente.

6.3 L'Algorithme

```
Debut
     Pour tout x∈X
        Faire
          π_old[x]← +∞ ;
          Si (r,x) ∈ U Alors  π[x]← p(r,x);
                              Pré[x] ← NULL ;
                       Sinon  π[x]← +∞ ;
                              Pré[x] ← NULL ;
          fSi
        Fait
     π[r]← 0; i ← 1;
     Tant que ((π_old ≠ π) et(i <= n))
     Faire
        π_old ← π; i ← i+1;
        Pour tout (x ∈ X-{r})
        Faire
           Pour tout ((y, x) ∈ U)
           Faire
                Si π[x] > π_old [y]+ p[(y, x)]
                       Alors π[x] ← π_old[y]+p[(y, x)];
                             Pré[x] ← y;
                fSi
           Fait
        Fait
     Fait
     Si (π_old ≠ π)    Alors Retourner("Il existe un circuit absorbant");
                       Sinon retourner(π,Pré) ;
Fin
```

6.4 Remarques

- Si à la fin de l'exécution on obtient toujours $\pi\_old \neq \pi$ alors le graphe contient un circuit absorbant.
- On dit que l'algorithme retourne une arborescence optimale.
- De même que pour Dijkstra, si le graphe est non orienté, on peut associer à chaque arête $\{x,y\}$ de poids p, deux (02) arcs (x,y) et (y,x) dont les poids sont p, puis on applique l'algorithme de Bellman-Kalaba.

7. Algorithme de Floyd

7.1 Problème traité

Le problème est de trouver le poids du chemin optimal entre toute paire de sommets dans le graphe. Le graphe peut contenir des circuits et les poids peuvent être positifs nuls ou négatifs.

7.2 Les données

En entrée

X : ensemble des sommets
n : Nombre de sommets
U : ensemble d'arcs sous forme *(x,y)*
p : un vecteur de m éléments où $p[u]$ représente le poids du l'arc u dans le réseau.

En sortie :

π: matrice de n x n éléments. Chaque entrée $\pi[i,j]$ représente le poids du chemin optimal de i vers j.
Pré : matrice de n éléments. Chaque entrée *Pré[i,j]* représente le prédécesseur de j dans le chemin optimal de i vers j.

7.3 L'Algorithme

```
Début
     Pour i de 1 à n
     Faire
        Pour j de 1 à n
        Faire
           Si (i=j) Alors π[i,j] ← 0;     Pré[i,j] ← NULL;
                    Sinon  Si((i,j)∈U)  Alors  π[i,j] ← p[(i,j)];       Pré[i,j] ← i;
                                        Sinon  π[i,j] ← +∞ ;    Pré[i,j] ← NULL;
                           fSi
           fSi
        Fait
     Fait
     Pour k de 1 à n
        Pour i de 1 à n
           Pour j de 1 à n
                Si (π[i,j] > π[i,k]+ π[k,j]) Alors π[i,j] ← π[i,k]+ π[k,j];
                                                   Pré[i,j] ← Pré[k,j];
                fSi
Fin
```

7.4 Remarques

- La complexité de l'algorithme est $o(n^3)$
- L'algorithme ne permet pas de détecter les circuits absorbants mais (à la différence de Dijkstra et Bellman-Ford) il se termine malgré la présence de circuits absorbants.
- Si le graphe est non orienté, on applique le même principe que pour Dijkstra et Bellman-Kalaba pour transformer le graphe en graphe orienté puis on pourra applique l'algorithme de Floyd.

Problème d'ordonnancement

1. Identification du problème d'ordonnancement

L'examen d'un projet (industriel, administratif, informatique, …) comporte en général deux phases importantes.

Phase 1 :

La division du projet en plusieurs tâches (ou étapes) élémentaires, l'étude des liaisons (contraintes logiques, chronologiques, …) et l'estimation de la durée de chaque tâche. Cette phase d'analyse amène à la construction d'un graphe orienté (où les sommets représentent les tâches élémentaires) dans lequel les arcs seront pondérés.

Phase 2 :

Elle consiste à analyser et étudier le graphe obtenu en phase1. Parmi les résultats de cette phase : la détermination d'un planning (ou ordre chronologique) et la recherche de la durée totale du projet dans le but de la minimiser. L'étude du graphe revient déterminer un chemin de poids optimal…

2. Graphe potentiel tâches

Appelée aussi méthode des potentiels métra (MPM) ou méthode du chemin critique. Elle consiste à planifier plusieurs tâches dont certaines sont liées entre elles par des contraintes dites de potentielles, s'exprimant sous la forme $t_j - t_i \geq a_{ij}$ où t_i et t_j représentent les dates de début au plus tôt des tâches i et j.
Ce genre de contrainte s'interprète par : « la tâche j, ne peut commencer qu'après que i aura consommée a_{ij} unités de temps depuis son début ».
La méthode modélise le problème sous forme de graphe appelé graphe potentiels tâches où chaque tâche est représentée par un sommet et chaque arc $(i, j) \in U$ est dessiné si on a une contrainte de potentiel entre i et j ($t_j - t_i \geq a_{ij}$) en lui associant un poids égal à a_{ij}.
On introduit dans le graphe deux **tâches fictives** *«D»* et *«F»* représentant le début et la fin du projet.
Pour chaque tâche i n'ayant de tâche précédente, on rajoute l'arc (D,i). Le poids de l'arc est généralement 0 sauf s'il a été précisé dans l'énoncé un délai entre le début du projet et cette tâche.
Pour chaque tâche i, on rajoute l'arc (i,F). Le poids de cet arc sera égal à la durée de la tâche i. On peut optimiser le graphe en supprimant certains arcs (i,F) qui risquent de représenter des contraintes redondantes.

3. Etude et Analyse du graphe potentiel tâches

3.1 Dates au plus tôt

La durée minimale de l'ordonnancement est donnée par la date au plus tôt (t_F) de la tâche fictive «fin projet».
La date au plus tôt d'une tâche j est obtenue par la formule suivante :

$$t_j = \max_{i \in LP(j)} \{t_i + a_{ij}\},\ j \neq 0.\ LP(j) \text{ est la liste des prédécesseurs du sommet } j.$$

$$t_0 = t_D = 0$$

Remarque :

La date au plus tôt d'une tâche j est égale à la plus longue distance (chemin de poids maximum) du sommet «début projet» au sommet j dans le graphe «potentiels tâches».

3.2 Marge totale

Le délai de retard pour une tâche i qui n'affecte pas la durée minimale du projet est noté M_i et est appelé **marge totale de la tâche *i***.

$M_i = t_i^* - t_i$ où t_i : date au plus tôt de la tâche i

t_i^* : date au plus tard de la tâche i calculée comme suit :

$$t_F^* = t_F$$

$$t_i^* = \min_{j \in LS(i)} \{t_i^* - a_{ij}\} \quad i \neq F = \text{«fin projet»}.\ LS(i) \text{ est la liste des successeurs du sommet } i.$$

Une tâche i ayant une marge totale $M_i = 0$ et est appelé **tâche critique**.

3.3 Marge libre

La **marge libre d'une tâche *i***, notée m_i, est le délai de retard d'une tâche i sans affecter les dates de début au plus tôt des tâches postérieures.

$$m_i = \min_{j \in LS(i)} \{t_j - t_i - a_{ij}\}$$

3.4 Marge certaine

La **marge certaine d'une tâche *i***, notée μ_i, est le délai de retard d'une tâche i, quand les tâches antérieures commencent à leurs dates au plus tard et les tâches postérieures à leurs dates plus tôt

$$\mu_i = \max \{ 0, \min_{j \in LS(i)} \{t_j - a_{ij}\} - \max_{k \in LP(i)} \{t^*_k + a_{ki}\}\}$$

4. Exemple

Soit un projet dont la liste des tâches est donnée dans le tableau ci-dessous. Il comporte également la durée d'exécution de chaque tâche (en jours) ainsi que les contraintes liées au début d'exécution :

Tâche *i*	Durée de *i*	Contraintes liées au début d'exécution de la tâche *i*
1	6	-
2	10	Ne peut débuter que 2 jours après le début des travaux.
3	20	Après la fin de 1 et 2.
4	7	Après la fin de 1.
5	8	Peut débuter 2 jours après le début de 2.
6	4	Après la fin de 5 et 10 jours au maximum après le début de 4.
7	10	Peut débuter après la fin de 4 et lorsque 3 est à moitié réalisé.
8	5	Ne doit pas dépasser 5 jours après la fin de 6 et la 5 doit être achevée.
9	12	-
10	20	Peut commencer 5 jours après la fin de 9.

1) Les contraintes…
2) Le graphe potentiel tâches associé à ce problème…
3) Les dates au plus tôt…
4) La durée minimale du projet…
5) Les dates au plus tard…
6) Les tâches critiques…

Arbre de couverture optimal

1. Définition d'un arbre

Soit $G=(X,U)$ un graphe orienté d'ordre $n \geq 2$. G est un **arbre** si l'une des **six (6) propriétés** suivantes est vérifiée :

1. G est connexe et sans cycles.
2. G est sans cycles et admet n-1 arcs.
3. G est connexe et admet n-1 arcs.
4. G est sans cycle maximal (tout arc supplémentaire créé un cycle dans G).
5. G est connexe minimal (la suppression d'un arc quelconque le rend non connexe).
6. Pour toute paire de sommets ($x, y \in X, x \neq y$) Il existe dans G une chaîne et une seule joignant x à y.

Les 6 caractéristiques ci-dessus (propres aux arbres) sont équivalentes.
Ceci est valable pour un graphe non-orienté.

2. Codage de Prufer

Un arbre représenté par un graphe d'ordre n peut être représenté par séquence ordonnée de n-2 éléments. Chaque élément représente le numéro (indice) d'un sommet. Pour cela, chaque sommet du graphe doit être numéroté. Dans ce qui suit, nous présentons un algorithme de codage qui génère pour un graphe arbre le code Prufer correspondant ainsi qu'un algorithme de décodage qui génère un graphe arbre à partir d'un code de Prufer.

2.1 Algorithme de codage

Principe

A partir d'un graphe qui est arbre d'ordre n (dont les sommets sont numérotés 1, 2, …, n), l'algorithme rajoute pour chaque feuille (sommet de degré 1) ayant le plus petit numéro son voisin dans le code de Prufer et supprime cette feuille ainsi que l'arête qui lui est incidente. L'algorithme s'arrête après n-2 itérations. En d'autres termes, l'algorithme se termine après avoir inséré dans le code de Prufer n-2 sommets.

Donnée

En Entrée : Le graphe arbre $G=(X,E)$
En Sortie : Une suite ordonnée de sommets P
Autres :
x : variable sommet

L'algorithme

```
Début
      P ← ∅;
      Tant que |X|>2
          Faire
                 Choisir x dans X tel que dG(x)=1 et x minimal; // feuille de numéro minimal
                 P ← P . Adjacent(x);
                 // Rajouter le sommet adjacent à x dans la liste ordonnée P
                 X ← X - {x}; // Supprimer le sommet x
                 E ← E - {x, Adjacent(x)}; // Supprimer l'arête incidente à x
          Fait
Fin.
```

2.2 Algorithme de décodage

Principe

A partir d'un code de Prufer P qui est une suite de n-2 numéros de sommets, où n est l'ordre du graphe, l'algorithme va générer un graphe arbre d'ordre n. L'algorithme calcule d'abord les degrés des sommets à partir de P. Par la suite, pour chaque sommet qui apparaît dans P, il crée une arête vers le sommet de degré 1 puis décrémente les degrés des deux extrémités de cette arête. A la fin, les deux sommets restants ayant un degré égal à 1 seront reliés par une arête.

Donnée

En Entrée : Le code de Prufer sous forme d'une liste ordonnée *P*
En Sortie : Un graphe non-orienté qui est arbre *G=(X, E)*
Autres :
x : variable sommet
D : vecteur de n éléments. Chaque *D*[*i*] représente au début le degré de *i*.

L'algorithme

```
Début
      n ← longueur(P)+ 2;
      X ← ∅;
      E ← ∅;
      Pour i de 1 à n
            Faire
                  X ← X ∪ {i};
                  D[i] ← 1;
            Fait
      Pour chaque valeur j de P
            Faire
                  D[i] ← D[i] + 1;
            Fait
      Pour chaque valeur j de P
            Faire
                  Chercher k tel que D[k]=1 et k minimal;
                  E ← E ∪ {j,k};
                  D[j] ← D[j]-1;
                  D[k] ← 0;
            Fait
      Relier les deux sommets ayant D[i]=1;
Fin.
```

3. Graphe pondéré (Réseau)

3.1 Poids d'une arête

Soit *G=(X,E)* un graphe non-orienté,
On définit $p : E \rightarrow \mathcal{R}$ une application qui associe pour chaque arête $e \in E$ de *G* une valeur réelle *p(e)* appelée poids de l'arête *e*.

3.2 Poids d'un arc

Soit *G=(X,U)* un graphe orienté,
On définit $p : U \rightarrow \mathcal{R}$ une application qui associe pour chaque arc $u \in U$ de *G* une valeur réelle *p(u)* appelée poids de l'arc *u*.

3.3 Poids d'un arc

Soit *G=(X, U, p)* (resp. *G=(X, E, p)*) un graphe doté d'une application *p* de pondération des arcs (resp. des arêtes).
Un tel graphe est appelé graphe pondéré, graphe valué ou réseau.

4. Arbre de couverture d'un graphe

Soit *G=(X,U)* un graphe orienté d'ordre $n \geq 2$. On appelle arbre dans *G* un sous-graphe partiel de *G*, *H=(Y,V)* connexe et sans cycles. Un arbre est maximal dans *G* s'il contient le maximum possible de sommets de *G* (c'est-à-dire *Y=X* si *G* est connexe).

5. Arbre de couverture de poids optimal

5.1 Identification du problème

Soit *G=(X,U)* un graphe orienté connexe muni d'une application poids *p*.
La recherche dans *G* d'un arbre de poids optimal revient à trouver un graphe partiel *G'=(X,U')* de *G* qui soit un arbre et pour lequel la somme des poids des arcs de *G'* soit optimale (maximale ou minimale selon la situation).

5.2 Algorithme de Kruksal

Principe

Dans cet algorithme, il faut d'abord trier dans un tableau les arcs du graphe selon l'ordre croissant des poids. Le tableau trié doit être parcouru dans l'ordre. Un arc est sélectionné s'il ne forme pas avec les arcs déjà sélectionnés un cycle. Les (n-1) premiers arcs sélectionnés forment l'arbre recouvrant de poids minimal.

Donnée

En Entrée : Le graphe pondéré (réseau) $G=(X,U,p)$
En Sortie : Ensemble d'arcs H
Autres :
x : variable sommet
u : variable arc
V : tableau d'arcs trié dans un ordre de poids croissants ($V[i] \leq V[i+1]$)
n : nombre de sommets dans G, c'est-à-dire $n=|X|$

L'algorithme

```
Début
     V = tri_poids_croissant(U,p);
     H ← {V[1]};
     i ← 1 ;       j ← 1;
     Tant que (j < n-1)
     Faire
          i ← i+1; u ← V[i];
          Si (H ∪ {u} ne contient pas de cycle)
            Alors H ← H ∪ {u}; j ← j+1; // L'arc sélectionné ne doit pas créer de cycle
          fSi
     Fait
Fin.
```

Remarques

- L'algorithme a une complexité de l'ordre de $o(n^2)$.
- On peut utiliser l'algorithme sur un graphe non-orienté.
- On peut trier les arcs dans un ordre décroissant. Dans ce cas, on met tous les arcs dans H et on supprime dans l'ordre les arcs si ça ne déconnecte pas le graphe. On s'arrête quand on aura n-1 arcs.
- Il est possible d'utiliser le même algorithme pour trouver l'arbre de couverture poids maximal.

5.3 Définition (Contraction)

Soit $G=(X,U)$ un graphe et $u=(x,y) \in U$. On appelle contraction, le graphe $G_u=(X_u,U_u)$ obtenu à partir de G comme suit :

- Les sommets x et y seront remplacés par un sommet unique $c\{x,y\}$ dans G_u.
- Pour tout arc v_l dans G ayant x ou y comme extrémité initiale (resp. terminale), on lui remplace dans G_u son extrémité initiale (resp. terminale) par $c\{x,y\}$.
- On supprime l'arc u dans G_u.

5.4 Théorème 1

Soit $G=(X,U)$ un graphe et $u=(x,y) \in U$. G est un arbre si et seulement si G_u est un arbre.

5.5 Théorème 2

Soit $G=(X,U,p)$ un graphe connexe et $x \in X$.
Soit U_x un sous-ensemble de U défini comme suit : $U_x=\{u \in U / (I(u)=x \text{ ou } T(u)=x) \text{ et } u \neq (x,x)\}$
Soit $u(x)$ l'arc de U_x ayant le plus grand poids : $u(x) / p(u(x)) = min\{p(u) / u \in U_x\}$.
L'arbre $H=(X,V) / V \subset U$ et $u(x) \in V$ est un arbre de poids minimum.

5.6 Algorithme de Prim

Principe

Cet algorithme se base sur les deux théorèmes précédents. A partir d'un sommet x quelconque, on sait que l'arc adjacent à ce sommet de poids minimum ($u(x)$) fait certainement partie de l'arbre de couverture de poids minimum. A partir de cet arc là, on fait une contraction du graphe.

On répète ces opérations jusqu'à ce que le graphe soit contracté en un seul sommet. L'ensemble *V* formé des différents arcs $u(x)$ obtenus à chaque itération donnera l'arbre de poids minimal.

Donnée

En Entrée : Le graphe pondéré (réseau) *R=(X,U,p)*
En Sortie : Ensemble d'arcs *H*
Autres :
x : variable sommet
u : variable arc
u(x) : arc de poids minimal dont l'une des extrémités est *x*
G : variable graphe valué
G_u : Graphe contraction de *G* par rapport à l'arc *u*

L'algorithme

```
Début
      G ← R ;
      H ← ∅;
      Tant que |X|≥ 2
      Faire
            Choisir x dans X;
            u ← u(x); // choisir l'arc u de poids min ayant x comme extrémité
            H ← H ∪ {u};
            G ← Gu ; // Contracter G par rapport à l'arc u
      Fait
Fin.
```

Remarques

- L'algorithme a une complexité de l'ordre de $o(n^2)$.
- Il est possible d'utiliser le même algorithme pour trouver l'arbre de couverture poids maximal, en choisissant à chaque itération l'arc de poids maximal.