Recherche Opérationnelle

M Weinberg Benjamin

Docteur ès Infirmatique & Agrégé de mathématiques

Ce cours a été présenté aux étudiant de Master I et Master II Informatique de FLST.

Mathématiques appliquées, deux parties:

Théorie des graphes

programmation linéaire

Table des matières

Chap.1Théorie des Graphes 4

IIntroduction 4

1 )Historique 4

2 )Les Langages Différents 4

3 )De nombreux problèmes 4

IIGraphe orienté 5

1 )Généralités 5

a )Exemple : 5

2 )Dessin 5

3 )Mathématiques 5

a )Vocabulaire 5

b )Autres définitions et notations 5

c )Dèrnières définitions 5

IIIGraphe non orienté 6

1 )Français 6

2 )Sagittal 6

3 )Mathématique 6

a )vocabulaire 6

b )Autres définitions et notations 6

c )Dèrnières définitions 6

IVReprésentation Informatique 7

1 )Sur un exemple 7

2 )Liste d'adjacence 7

a )Exemple 7

3 )Matrice d'adjacence 7

a )Exemple 7

b )Nombre de chemin de longeur donné 7

Vmulti-graphe 8

1 )Par l'exemple 8

2 )Vocabulaire 8

Chap.2Un peu de théorie 8

ILemme des poignées de mains 8

1 )dans les graphes orienté 8

2 )des graphes non orienté 8

3 )Corollaire 8

4 )Exercice 8

IISous-structure 8

1 )Sous-graphe 8

a )Definition 8

b )Exemple 9

2 )Graphe partiel 9

a )Definition 9

b )Exemple 9

c )Remarque 9

3 )Connexités 9

a )Fortement connexe 9

b )connexité (simple) 9

c )Composante connexe 9

IIIRecherche de chemin Eulerien 10

1 )Condition nécessaire 10

2 )Est elle suffisante ? Oui cf ci dessous 10

3 )Algorithme. 10

4 )Exercice 11

Chap.3Plus court chemin 12

IGraphe valué 12

1 )Définitions 12

a )Remarque 12

b )Définitions 12

c )Exemple 12

d )Contexte général 12

2 )Longueur d'un chemin 12

3 )Codage par Matrice de valuations 13

a )Expression mathématique 13

b )Exemple 13

4 )Exercice 13

5 )Inégalité triangulaire et chemin absorbant 13

a )Remarque : vers une condition nécessaire 13

b )Proposition 13

c )Définition 14

d )Théorème 14

e )Exemple 14

6 )Codage des chemins partant d'un sommet 14

7 )Exercice 14

IICas des graphes orienté acyclique 15

1 )Définition 15

2 )Exemple 15

3 )Fonction ordinale 15

4 )Exercice / Exemples 15

5 )Algorithme 15

6 )Algorithme de Bellman 16

7 )Exemple / Exercice 16

8 )Complexité 16

IIICas d'un graphe à valuations positives 16

1 )Définition 16

2 )Algorithme de Dijkstra 16

3 )Exercice / Exemple 17

4 )Complexité 17

5 )Et les valuations négatives ? 17

6 )Exercice 18

a )Donner la matrice d’adjacence M de ce graphe. On classera les sommets par ordre alphabétique. 18

b )Ce graphe est il complet ? Est il connexe ? Justifier ses réponses. 18

c )Est il possible que l’agent de sécurité passe une et une seule fois par toutes les allées de cette usine ? On justifiera précisément sa réponse en utilisant un théorème vu en cours. S’il existe, donner un exemple d’un tel parcours. 18

d )Même question en ajoutant la contrainte de revenir à son point de départ. 18

e )Appliquer l'algorithme de Dijkstra en détaillant son fonctionnement. On donnera les temps minimaux et les plus courts chemins pour aller du bâtiment A vers les autres bâtiments. 18

IVCas Général 18

1 )Algorithme de Bellman-Ford 18

2 )Exemple 19

3 )Complexité 19

4 )Remarques 19

Chap.4Problème d'ordonnancement 20

IGénéralités 20

1 )Le problème 20

2 )Exemple 20

3 )Méthodes 20

IIMéthode PERT 20

1 )Construction du graphe 20

a )Échéances 20

b )Contraintes 21

c )2 échéances de plus 21

d )Exemple 21

2 )Pause Exercice 21

3 )Calcul des dates au plus tôt 29

4 )Calcul des dates au plus tard 29

5 )Chemin critique et tâche critiques 29

6 )Simplifications 33

IIIExercices 34

Chap.5Problèmes de Flot Maximum 35

IGénéralités 35

1 )Réseau de transport 35

2 )Flot 35

3 )Flot compatible 35

4 )Flot complet 35

5 )PROBLEME DU FLOT MAXIMUM 36

6 )Applications 36

IIMéthode 36

1 )Principe 36

2 )Algorithmes 36

a )Exemple de chaines augmentantes 36

b )Recherche de chaines augmentantes 36

c )Algorithme de Ford-Fulkerson 37

d )Exemple 37

IIIExercice 38

IVConvergeance de l'algorithme de Ford-Fulkerson 38

1 )Cas des flux entiers 38

2 )Cas des flux irrationnels 38

3 )Remarque 38

Chap.6Programmation Linéaire 39

IGénéralités 39

1 )Rappel 39

2 )Introduction 39

3 )Exemple 39

4 )Modélisation 39

5 )Résolution graphique 40

6 )Exercice 43

IIFormalisation 44

1 )Données brutes 44

2 )Canonisation 44

a )Forme canonique de maximisation avec inégalités 44

b )Forme canonique de maximisation avec égalités 45

3 )Exerices 45

4 )Résolution analytique et interprétation géométrique 45

5 )Exercice une résolution Analytique avec le problème de l'usine 47

6 )Algorithme du simplex 47

a )Choix du pivot 48

b )Exercice écrire l'algorithme du pivot de Gauss 48

c )Exercice écrire l'algorithme du Simplex 48

7 )Exercice 48

IIIDifférents cas possibles d'un PL (exo) 49

1 )Exercice 49

2 )Méthode du grand M 50

a )Variables supplémentaires 50

b )Trouver le 1re système 50

c )Sur l'exemple 50

3 )Cas d'arrêts 50

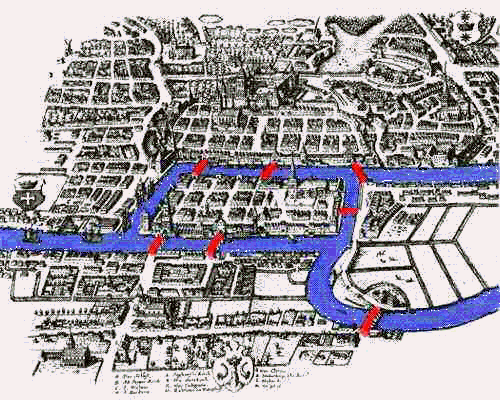
4 )Convergence 51

# Théorie des Graphes

## Introduction

### Historique

Apparition en 1736, avec Euler et Le problème des ponts de Koeningsberg.



### Les Langages Différents

Français

Dessin

Mathématiques

Programmation

### De nombreux problèmes

Parmi ceux qu'on va aborder:

Problèmes de chemin minimum

Problèmes de flux

Problèmes de d'ordonnancement

## Graphe orienté

### Généralités

Un graphe orienté permet de représenter des relations binaires entre des objets. Cette relation n'est pas forcément symétriques

#### Exemple

La construction d'un entrepôt est découpée en plusieurs tâches

* Acceptation des plans par le propriétaire
* Préparation du terrain
* Commande des matériaux
* Creusage des fondations
* ...

On sait que

* la commande des matériaux doit se faire après l'accéptation des plans
* Le creusage des fondations doit ce faire après l'accéptation des plans et après la préparation du terrain
* ...

### Dessin

Diagramme sagital

### Mathématiques

Un graphe G est la donnée d'un ensemble de sommet V et d'un ensemble d'arcs E. Un arc relie 2 sommet donc E est une partie de V x V.

Sur l'exemple : V = {A, B, C, D} et E = { (A, C) , (A, D), (B, D) }

#### Vocabulaire

Sommet

arc

origine, extremité

#### Autres définitions et notations

Successeur +

Prédécesseur -

Degré entrant d-

Degré sortant d+

Degré

#### Dèrnières définitions

Graphe Simple

Complet

Chemin

Circuit

(chemin /circuit) Simple et élémentaire

(chemin /circuit) eulerien

(chemin /circuit) hamiltonien

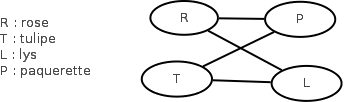
## Graphe non orienté

Un graphe non orienté permet également de représenter des relations binaires entre des objets. Or ici, la relation est symétrique.

### Français

Un jardinier doit décorer un jardin privatif en répartissant des variétés de fleurs. Certaines de ces variétés ne peuvent pas être plantées ensemble pour des raisons diverses (tailles, couleurs, conditions climatiques, …) voici des roses et les paquerettes, les tulipes et les paquerettes, Lys et roses, Lys et tulipes, ...

### Sagittal



### Mathématique

G(V, E) est une graphe non orienté si pour tout e = (x,y) in E e' = (y,x) in E

V = {R, P, T, L)

E = {(R, P), (R, L), (T, L), (T, P), (P, R), (L, R), (L, T), (P, T)}

#### vocabulaire

Sommet

arc<--> arête

chemin <--> chaine

circuit <--> cycle

origine, extremité

#### Autres définitions et notations

Degré entrant / Degré sortant

Degré = degré sortant = degré entrant

Successeur / Prédécesseur

adjacent

#### Dèrnières définitions

Graphe Simple

Complet

chemin / circuit

Chaine / Cycle

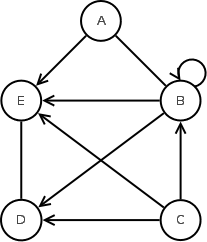
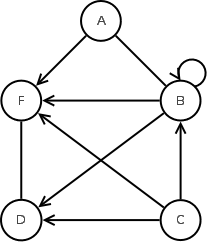
(chaine /cycle) Simple et élémentaire

(chaine /cycle) eulerien

(chaine /cycle) hamiltonien

## Représentation Informatique

### Sur un exemple



### Liste d'adjacence

Pour chaque sommet, on donne la liste des successeurs de ce sommet

#### Exemple

A : {B E}

B : {A, B}

C : {B, D, E}

D : {E}

E : {D}

Il existe une variante avec la liste des successeurs et des prédécesseurs

### Matrice d'adjacence

La matrice d'adjacence d'un graphe G(V,E) dont les sommets sont triés par ordre croissant est une matrice M = (mij)

#### Exemple

#### Nombre de chemin de longueur donné

Remarque M = le nombre de chemin de longueur 1 pour aller d'un sommet à un autre.

M est une matrice carré, on peut calculer M².

Remarque M2 = le nombre de chemin de longueur 2 pour aller d'un sommet à un autre.

...

Remarque Mk = le nombre de chemin de longueur k pour aller d'un sommet à un autre.

## multi-graphe

### Par l'exemple

### Vocabulaire

Sommet, arête, degré, chaine, cycle, (eulérien)

Matrice d'incidence

# Un peu de théorie

## Lemme des poignées de mains

### dans les graphes orienté

### des graphes non orienté

Attention les boucles sont comptées 2 fois.

### Corollaire

Dans un graphe orienté ou non le nombre de sommet de degré impair est pair.

### Exercice

On souhaite organiser un tournoi de handball entre 7 équipes. Pour des raisons d’organisation, on limite le nombre de matchs.

Est-il possible de faire jouer à chaque équipe exactement 5 matchs ?

Est-il possible de faire jouer à chaque équipe exactement 4 matchs ?

(si la réponse à une question est oui, représenter le graphe correspondant)

## Sous-structure

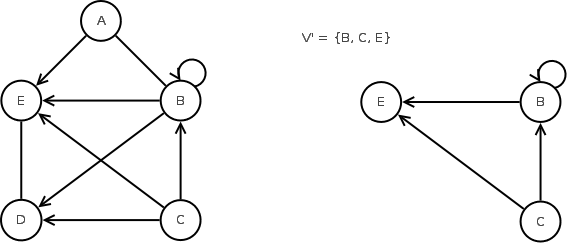
### Sous-graphe

#### Definition

Soit G(V, E) un graphe et V' partie de V

On définit le sous-graphe de G induit par V' comme le graphe G'(V', E') avec E' = { (x,y) in E | x in V' et y in V' }

#### Exemple



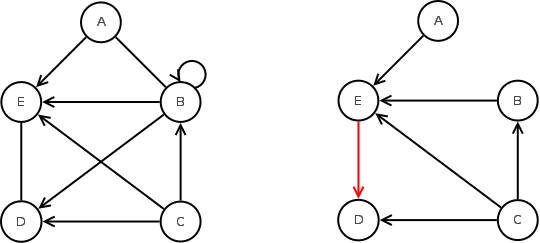
### Graphe partiel

#### Definition

Soit G(V,E) un graphe et E' partie de E

On définit le graphe partiel de G induit par E' comme le graphe G'(V, E')

#### Exemple



#### Remarque

Sous graphe partiel.

### Connexités

#### Fortement connexe

Chemin

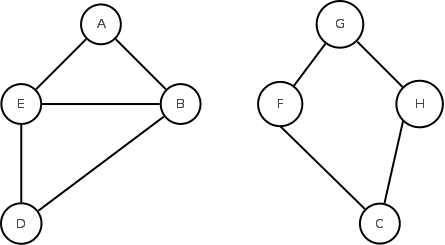
#### connexité (simple)

Chaîne

orientation des arc importe peu !

#### Composante connexe

Exemple :



Définition :

Les sous graphes maximaux connexes.

## Recherche de chemin Eulerien

### Condition nécessaire

Connexité

Degré

### Est elle suffisante ? Oui cf ci dessous

### Algorithme.

**FONCTION** CHEMIN\_EULERIEN(g : graphe) **EST**

Départ, arrivée : Sommet;

Solution : Liste(sommet) <-- vide;

CS : Liste(sommet) <-- vide;

**DEBUT**

choisir\_Depart\_arrivée(départ, arrivée, g);

**TANT QUE** E  **FAIT**

Extraire\_une\_chaine\_simple (départ,arrivee, CS, G);

inserer(dans => Solution, quoi => C, devant => depart);

départ <-- choisir\_sommet\_de\_degré\_pair(g);

arrivee <-- départ;

FAIT

FINSI

**PROCEDURE** choisir\_depart\_arrivée(départ, arrivée : out sommet; G : graphe) **EST**

cmpt : entier <-- 0;

**DEBUT**

**POUR** i  G.V **FAIT**

**SI** d(i) est impair **ALORS**

cmpt <-- compt +1;

**SI** cmpt = 1 **ALORS**

départ <-- i;

**SINON SI** cmpt = 2 **ALORS**

arrivée <-- i;

**SINON**

**LEVE** erreur\_graphe;

**FIN SI**

**FIN SI**

**FAIT**

**SI** cmpt  0 **ALORS** // le degrés de chaque sommet est pair

départ <-- choisir\_sommet\_de\_degré\_pair(g);

arrivee <-- départ;

**SINON SI** cmpt  2 **ALORS**

**LEVE** erreur\_graphe;

**FIN SI**

**FIN**

**FONCTION** choisir\_sommet\_de\_degré\_pair(g : graphe) **RETOURNE** sommet **EST**

**DEBUT**

**POUR** i  G.V **FAIT**

**SI** d(i)  0 **ALORS**

**RETOURNE** i;

**FIN SI**

**FAIT**

**LEVE erreur\_graphe;**

**FIN**

**PROCEDURE** Extraire\_une\_chaine\_simple (départ,arrivee : sommet; CS : liste(sommet), G : graphe) **EST**

courant : sommet : départ;

suivant : sommet <-- voisin(courant);

**DEBUT**

vider(CS);

**TANT QUE** suivant  départ **FAIT**

ajoute(courant, CS);

retirer\_arc((courant, suivant), g.v);

courant <-- suivant;

suivant <-- voisin(courant);

**FAIT**

ajoute(départ, CS); //  **ne pas mettre cette ligne si on souhaite utilisé l'insertion déjà codée :s**

**FIN**

### Exercice

Pour chaque graphe, donner une chaîne eulerienne, si une telle chaîne existe, sinon justifier sa non existance.

|  |  |  |
| --- | --- | --- |
|  |  |  |

# Plus court chemin

## Graphe valué

### Définitions

#### Remarque

On ne considère à partir de maintenant uniquement des graphes simples.

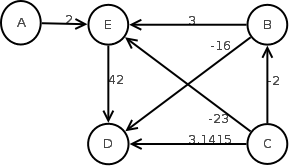
#### Définitions

On dit que G(V, E, d) est un graphe valué si G(V,E) est un graphe et d est une application de E dans IR.

On dit que G(V, E, d) est un graphe (valué) à valuation positive si d est à valeur dans IR+.

On représente généralement un graphe valué en plaçant les valuations sur les arcs.

#### Exemple



#### Contexte général

Dans la suite du chapitre on considère G(V, E, d) un graphe valué.

### Longueur d'un chemin

Soit C = (s0, s1, ... , sn) un chemin de G. On appelle longueur du chemin C (notée L(C)), la somme des valuations de ces arcs.

### Codage par Matrice de valuations

#### Expression mathématique

#### Exemple

### Exercice

Calculer la longueur du chemin (C – B – E – D) de l'exemple.

### Inégalité triangulaire et chemin absorbant

#### Remarque : vers une condition nécessaire

Soient x, y et z trois sommets si notons A, B et C des chemins reliant respectivement x à y, y à z et x à z.

Si A, B et C sont minimaux alors L(C) <= L(A) + L(B)

preuve : notons A = (a0, a1, ... , an), B = (b0, b1, ... , bk), C = (c0, c1, ... , cl),

on a a0 = x, an = y = b0 et bk = z donc le chemin C' = (a0, a1, ... , an-1, b0, b1, ... , bk) est un chemin allant de x à z

or on a supposé C minimal donc

L(C) <= L(C') donc L(C) <= L(A) + L(B)

#### Proposition

On remarque que pour tout sommet x de V le chemin minimal allant de x à x doit avoir une longueur positive ou nulle.

Preuve : par l'absurde

Supposons que D = (x, s1, s2, ... , sn-1, x) est une longueur négative strictement :L(D) = l <0

Applicons la remarque précédente au chemin D (pour A, B et C).

l = L(C) <= L(C) + L(C) = 2 \* l donc l >= 0 (Aburde)

#### Définition

Un circuit dont la valuation est négative est dit absorbant.

#### Théorème

Si un graphe admet un circuit absorbant alors ce graphe n'admet pas plus court chemin mettant ayant comme extremité un sommet de ce circuit.

#### Exemple

### Codage des chemins partant d'un sommet

Un sommet d étant choisit, on note la fonction qui retourne la longueur du plus cours chemin trouvé allant de d à v.

Techniquement, il s'agit d'un tableau L (on notera indifféremment L(v) ou L[v]).

**PROCEDURE** initialisation (debut : Sommet) **EST**

**DEBUT**

**POUR** v in V **FAIT**

L[v] <-- INIFINI

P[v] <-- NULL;

**FAIT**

L[debut] <-- 0

**FIN**

**PROCEDURE** ameliorer (depuis : Sommet ; jusqu\_a : sommet) **EST**

**DEBUT**

**SI** L[jusqu\_a] > L[depuis] + v(depuis, jusqu\_a) **ALORS**

L[jusqu\_a] <-- L[depuis] + d(depuis, jusqu\_a)

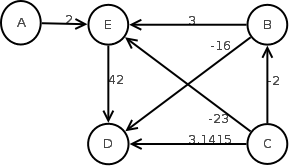
P[jusqu\_a] <-- depuis

**FINSI**

**FIN**

### Exercice

Pour le graphe suivant touver une amélioration et appliquer la



|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| L | | | | |  | P | | | | |
| A | B | C | D | E |  | A | B | C | D | E |
| 0 |  |  |  | 2 |  | -- | -- | -- | -- | A |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| L | | | | |  | P | | | | |
| A | B | C | D | E |  | A | B | C | D | E |
| 0 |  |  | 44 | 2 |  | -- | -- | -- | E | A |

## Cas des graphes orienté acyclique

### Définition

On dit qu'un graphe orienté est acyclique si il ne présente aucun circuit. (n'est pas malheureux)

### Exemple

### Fonction ordinale

On appelle fonction ordinale d'un graphe acyclique une bijection f définie par

f : V --> [[ 1 ; |V| ]]

n --> f(n)

Vérifiant xVyxf(y fx)

On recherchera ici la fonction réciprique à la fonction ordinale.

### Exercice / Exemples

Donner 2 fonctions ordionales compatible avec le graphe ci-dessus.

### Algorithme

**FONCTION** OrdinaleReciproque(Graphe : G) **EST**

s : Sommet;

tab : Tab[1..|G.V|] DE sommet;

**DEBUT**

**POUR** i allant de 1 à |V| **FAIT**

s <-- chercherSansPred(G);

tab[i] <-- s;

G.V <-- G.V \ {s};

G.sousgrapheInduit(V); // on retire les arcs qui viennent de nulle part

**FAIT**

**RETOURNE** tab;

**FIN**

**FONCTION** chercherSansPred(Graphe : G) **RETOURNE** sommet **EST**

**DEBUT**

**POUR** x  G.V **FAIT**

**SI** d(x) = 0 **ALORS**

**RETOURNE** x;

**FINSI**

**FAIT**

**FIN**

### Algorithme de Bellman

**PROCEDURE** Bellman(origine : Sommet) **EST**

s : Sommet;

tab : Tab[1..|G.V|] DE sommet;

**DEBUT**

initialisation(origine);

tab = OrdionaleReciproque(G);

**POUR** i allant de 1 à |V| **FAIRE**

s <-- tab[i];

**POUR** t  G-(s) **FAIRE**

ameliorer (s, t);

**FAIT**

**FAIT**

**FIN**

### Exemple / Exercice

On prendra comme origine le sommet C.

### Complexité

En supposant que la recherche de la fonction ordinale soit déjà fait O( | V | + | E | ).

Dans un graphe connexe, on dira simplement O(| E |).

L'algorithme présenté ici laisse supposé que la recherche de la fonction ordinale est en O(| V |²) or on peut faire descendre la complexité en O(| E |) en plaçant en tête de liste les sommets de degré entrant null dans V à chaque construction de sous-graphe.

## Cas d'un graphe à valuations positives

### Définition

On dit qu'un graphe est (un graphe valué) à valuations positives si toutes les valuations sont positives ou nulles.

### Algorithme de Dijkstra

**PROCEDURE** Dijkstra (debut : sommet) **EST**

i,j : Sommet;

S, T : Ensemble de Sommet;

**DEBUT**

Initialisation();

S =  ;

T = V;

**TANT QUE** T  **FAIT**

i <-- ChoisirSommetMin(T);

T <-- T \ {i};

S <-- S U {i}

**POUR** j  (i)  T **FAIT**

améliore(i,j)

**FAIT**

**FAIT**

**FIN**

**PROCEDURE** ChoisirSommetMin(T : ensemble de sommets) **RETOURNE** sommet **EST**

b : sommet : premierElement(T);

**DEBUT**

**POUR** r  T\{b} **FAIRE**

**SI** L(r)< L(b) **ALORS**

b <-- r;

**FINSI**

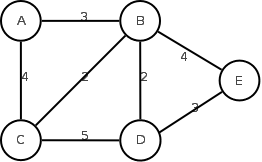
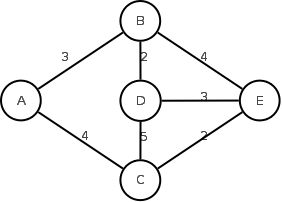
**FAIT**

**RETOURNE** b;

**FIN**

### Exercice / Exemple

Pour les deux graphes ci-dessous appliquer l'algorithme de Dijkstra.



### Complexité

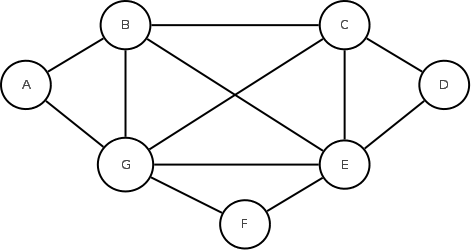
La version présentée ici et en O(| V |²), comme pour Bellman, cette complexité peut descendre en O(| E |) si la structure pour gerer les sommets et plus adapté qu'une simple liste.

### Et les valuations négatives ?

Par un exemple,

### Exercice

Le graphe ci-dessous indique, sans respecter d’échelle, les allées existantes entre les sept bâtiments d’une entreprise importante :



#### Donner la matrice d’adjacence M de ce graphe. On classera les sommets par ordre alphabétique.

#### Ce graphe est il complet ? Est il connexe ? Justifier ses réponses.

#### Est il possible que l’agent de sécurité passe une et une seule fois par toutes les allées de cette usine ? On justifiera précisément sa réponse en utilisant un théorème vu en cours. S’il existe, donner un exemple d’un tel parcours.

#### Même question en ajoutant la contrainte de revenir à son point de départ.

On donne maintenant les temps de parcours des allées par l’agent de surveillance :

AB : 16 minutes ;

AG : 12 minutes ;

BC : 8 minutes ;

BE : 12 minutes ;

BG : 8 minutes ;

CD : 7 minutes ;

CE : 4 minutes ;

CG : 10 minutes ;

DE : 2 minutes ;

EF : 8 minutes ;

EG : 15 minutes ;

FG : 8 minutes.

Nota Bene : les temps sont indépendants du sens de parcours.

#### Appliquer l'algorithme de Dijkstra en détaillant son fonctionnement. On donnera les temps minimaux et les plus courts chemins pour aller du bâtiment A vers les autres bâtiments.

## Cas Général

Pour se prémunir des cas gènants, on s'était placé avec Bellman dans des cas sans de circuit et avec Dijkstra dans des cas sans valeurs négatives.

Pas de circuit absorbant.

### Algorithme de Bellman-Ford

**FONCTION** améliore (depuis : Sommet ; jusqu\_a : sommet) **RETOURNE** booléen **EST**

**DEBUT**

**SI** L[v] > L[u] + v(u,v) **ALORS**

L[v] <-- L[u] + d(u,v)

P[v] <-- u

**RETOURNE** VRAI

**FINSI**

**RETOURNE** FAUX

**FIN**

**PROCEDURE** Bellman-Ford (G : graphe)

changement : BOOLEAN <-- VRAI

**DEBUT**

intialisation ();

**TANT** **QUE** changement **FAIT**

**POUR** i  G.V **FAIT**

**POUR** j (i) **FAIT**

**SI** améliore(i,j) **ALORS**

changement <-- VRAI

**FINSI**

**FAIT**

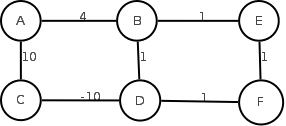
**FAIT**

**FAIT**

**FIN**

### Exemple

Appliquer l'algorithme de Bellman-Ford sur le graphe suivant en partant de A.



### Complexité

La boucle de I en O(| E |)

elle est excutée | V | fois au maximum

D'où la complexité de Bellman-ford en O(| V | . | E |)

### Remarques

Si le graphe présente un circuit absorbant, l'algorithme boucle sans fin !!

Il existe un algorithme permettant de calculer la longueur à partir de n'importe quel sommet vers chaque autre sommet. Il s'agit de l'algorithme de Floyd.

Annecdote : rechercher le chemin le plus cours semble naturel ; Qu'en est-il de rechercher le chemin le plus long ?

# Problème d'ordonnancement

## Généralités

### Le problème

Un problème d'ordonnancement consiste à organiser dans le temps la réalisation de tâches, compte tenu de contraintes temporelles (délais, contraintes d'enchaînement) et de contraintes portant sur la disponibilité des ressources requises.

Une tâche est une entité élémentaire localisée dans le temps par une date de début et/ou de fin, dont la réalisation nécessite une durée, et qui consomme un moyen selon une certaine intensité.

### Exemple

|  |  |  |  |
| --- | --- | --- | --- |
| tâches | nature | prédécesseurs | durée en jours |
| A | acceptation des plans par le propriétaire |  | 4 |
| B | préparation du terrain |  | 2 |
| C | commande des matériaux | A | 1 |
| D | creusage des fondations | A, B | 1 |
| E | commande des portes et fenêtres | A | 2 |
| F | livraison des matériaux | C | 2 |
| G | coulage des fondations | D, F | 2 |
| H | livraison des portes et fenêtres | E | 10 |
| I | pose des murs, de la charpente et du toit | G | 4 |
| J | mise en place des portes et fenêtres | H, I | 1 |

### Méthodes

Il existe plusieurs méthode pour résoudre ce type de problème

Diagramme de Gantt : permet de visualiser le temps total du projet.

Méthode PERT : construction d'un graphe pour modéliser le problème puis le résoudre.

## Méthode PERT

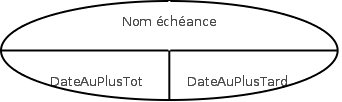
### Construction du graphe

Le première étape de la méthode PERT est de construire un graphe orienté valué G(V, E, d). Les sommets représentent les échéances importantes du projet comme le début d'une tâche par exemple. Les arcs représentent soit les tâches soit des contraintes de précédence ; On parle alors de tâches fictives. La valuation des arcs correspondent aux durées des différentes tâches. On ajoute également deux échéances supplémentaires : le début du projet, et la fin du projet.

#### Échéances

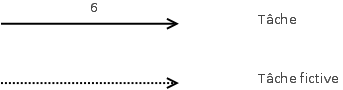
Pour chaque tâche décrivant le projet, on crée deux sommets : un pour le début de la tache, un pour la fin de la tâche. On peut créer des échéances supplémentaires en cas de besoin : par exemple "30 min avant la fin de la cuisson" signifie qu'il y a une échéance importante durant la tâche "cuisson".

On représente une échance de la manière suivante :

On place sous le nom deux dates qui seront calculer par la suite

#### Contraintes

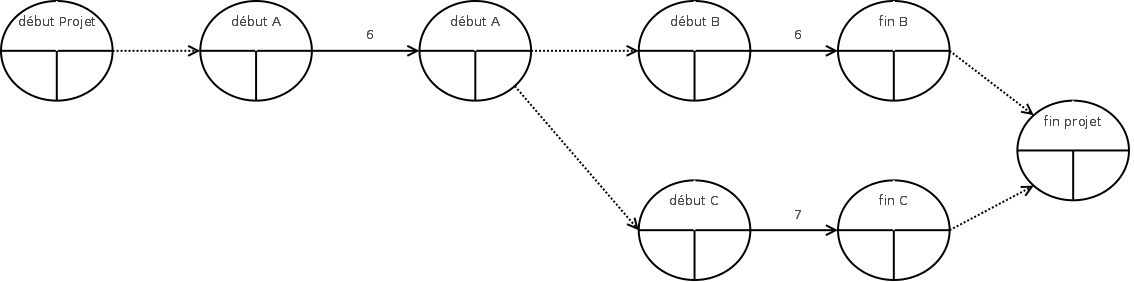
Il s'agit d'un arc du graphe. On le trace en pointillé quand il modélise une tâche fictive. On place la durée de la tâche au dessus de l'arc. Par convention les valuations nulles ne sont pas notées.



#### 2 échéances de plus

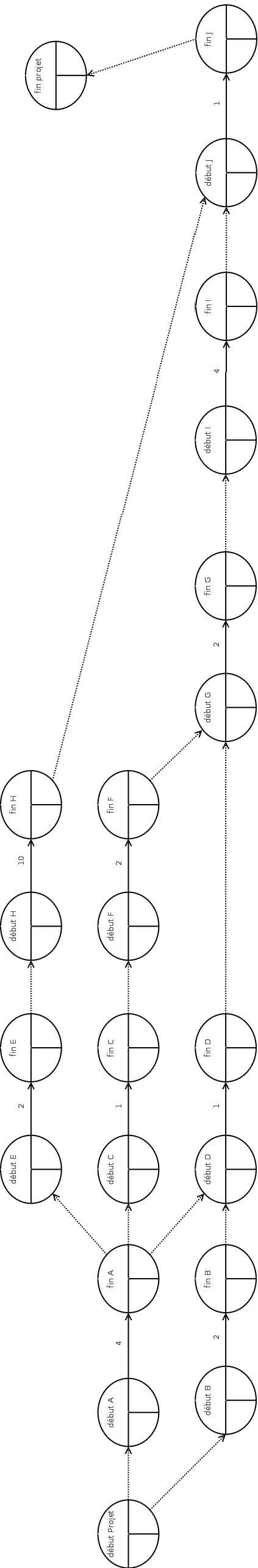
L'échéance de "début de projet" est placée avant tout les échéances sans prédécesseur et réliées à celle-ci par des arcs (allant du début de projet vèrs ces dernières). L'échéance de "fin de projet" est placée après tout les échéances sans successeur et réliées à celle-ci par des arcs (allant de ces dérnières vers la fin de projet).

#### Exemple



### Pause Exercice

Faire de graphe de PERT de l'exemple du début.



### Calcul des dates au plus tôt

Pour ordonnancer, le projet, on cherche dans un premier temps les dates au plus tôt. C'est à dire la date minimale à partir de laquelle on va pouvoir franchir cette étape. Pour connaitre la date au plus tôt d'un sommet x (note do(x)), il faut que toutes les échéances qui précédent x, soit finies ainsi que la contrainte temporelle qui mène à cette échéance.

En d'autres termes, le calcul de la date au plus tôt se fait par un chemin le plus long et en fixant la do(debutPojet) = 0

### Calcul des dates au plus tard

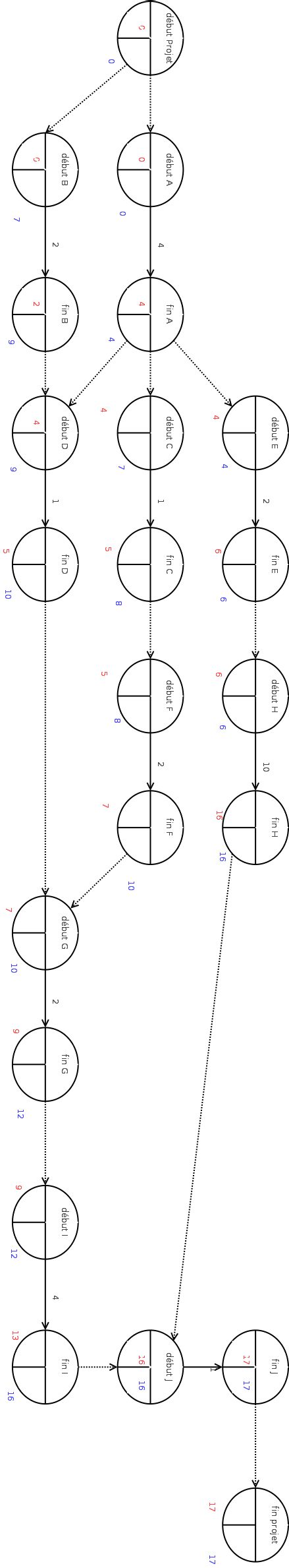
La date au plus tard d'une échéance est la date maximale autorisée à cette étape pour ne pas retarder la projet. On obtient le da(x) pour chaque sommet x du graphe grâce à la formule

et en posant da(Fin) = do(Fin)

### Chemin critique et tâche critiques

La marge sur une échéance x est la différence entre la da(x) et do(x). Si cette différence est nulle l'échéance est dite critique. Il existe au moins un chemin allant du début du projet à la fin du projet dont toutes les échéances sont critiques. Un tel chemin est dit critique.

Exemple

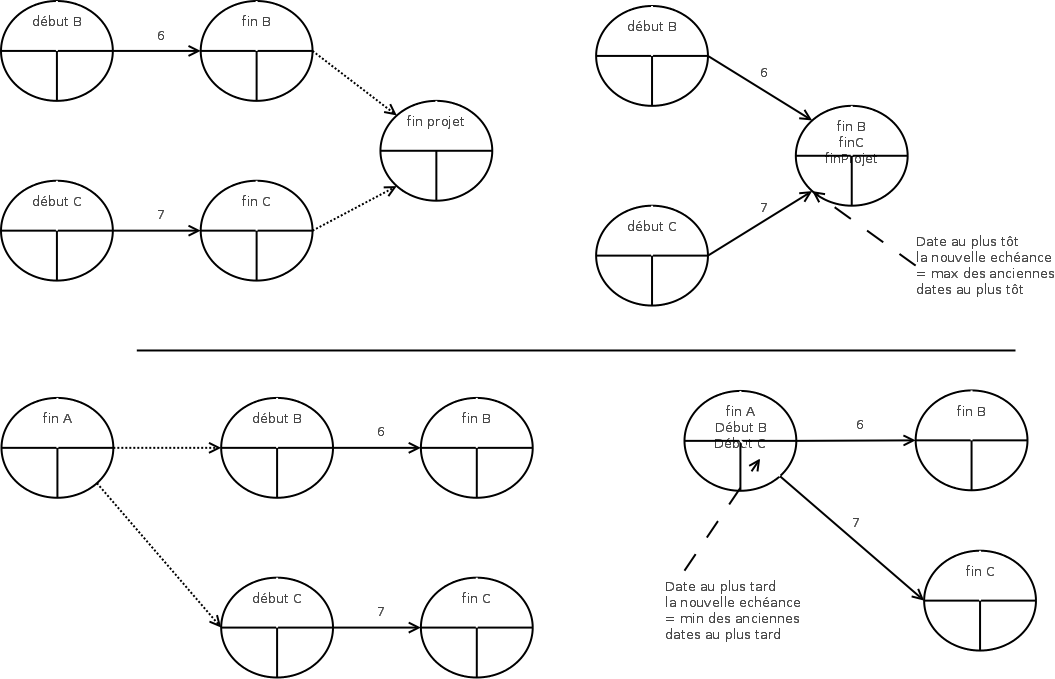


### Simplifications

Il est possible de simplifier le graphe PERT. Voici Deux simplifications possibles :

Une échéance étant le seul succésseur de tous ses prédécésseurs peut fusionner avec ceux-ci, si les contraintes de temps sont toutes nulles.

Une échéance étant le seul prédécesseur de tous ses successeurs peut fusionner avec ceux-ci, si les contraintes de temps sont toutes nulles.



## Exercices

La mise en exploitation d’un nouveau gisement minier demande la réalisation d’un certain nombre de tâches. Le tableau suivant représente ces différentes tâches avec leurs relations d’antériorité.

|  |  |  |  |
| --- | --- | --- | --- |
| Tâche | Description | Durée (en jours) | Tâches antérieures |
| A | obtention d’un permis d’exploitation | 120 | - |
| B | établissement d’une piste de 6 km | 180 | A |
| C | transport et installation à pied d’œuvre de 2 sondeuses | 3 | B |
| D | création de bâtiments provisoires pour le bureau des plans, le logement des ouvriers sondeurs | 30 | B |
| E | goudronnage de la piste | 60 | B |
| F | adduction d’eau | 90 | D |
| G | campagne de sondage | 240 | C,D |
| H | forage et équipement de trois puits | 180 | E,F,G |
| I | transport et installation au fond du matériel d’exploitation | 30 | J,H |
| J | construction de bureaux et logements, ouvriers et ingénieurs | 240 | E,F,G |
| K | traçage et aménagement du fond | 360 | J,H |
| L | construction d’une laverie | 240 | J,H |

Etablir le Graphe de PERT ; Puis déterminer les dates au plus tôt puis les dates au plus tard. En déduire les marges et les tâches critiques.

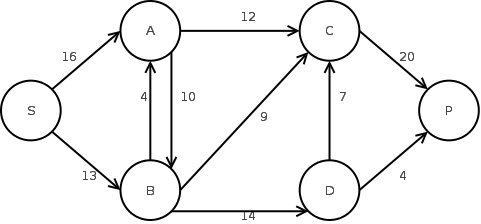
# Problèmes de Flot Maximum

## Généralités

### Réseau de transport

Un réseau de transport est un graphe orienté, où chaque arc est associé à un nombre c(u)  0, appelé "capacité" de l'arc u. En outre, un tel réseau vérifie les hypothèses suivantes:

* Il existe un seul noeud s qui n'a pas de prédécesseurs, tous les autres en ont au moins un. Ce noeud est appelé l'entrée du réseau, ou la source.
* Il existe également un seul noeud p qui n'a pas de successeurs, tous les autres en ont au moins un. Ce noeud est appelé la sortie du réseau, ou le puits.



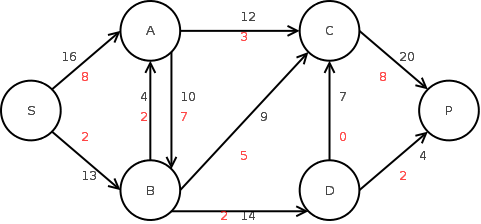
### Flot

Un flot **f** dans un réseau de transport est une fonction qui associe à chaque arc **u** une quantité **f(u)** qui représente la quantité de flot qui passe par cet arc, en provenance de la source et en destination du puits. Un flot doit respecter la règle suivante:

* la somme des quantités de flot sur les arcs entrants dans un noeud (autre que s et p) doit être égale à la somme des quantités de flot sur les arcs sortants de ce même noeud. ~~En d'autres termes, la quantité totale de flot qui entre dans un noeud est égale à la quantité totale de flot qui en sort.~~
* On remarque que ce qui sort de la source est égale à ce qui entre dans le puits.

### Flot compatible

Un flot f est compatible avec un réseau si pour tout arc u, 0  f(u)  c(u). Autrement dit, pour chaque arc, le flot qui le traverse ne doit pas dépasser la capacité de l'arc.



### Flot complet

Un flot **f** est complet si pour tout chemin allant de la source au puits, il y a au moins un arc saturé, i.e. le flot qui le traverse est égal à la capacité de l'arc.

### PROBLEME DU FLOT MAXIMUM

Connaissant les capacités des arcs d'un réseau de transport, le problème du flot maximum consiste à trouver un flot compatible complet.

### Applications

Ce problème pose notamment dans la conception de circuit élecrtique ou de réseau de hydraulique, de télécom ...

## Méthode

### Principe

On part d'un flot compatible. Le plus évident est le flot nul, i.e. pour tout arc u, f(u) = 0. Ensuite, on cherche une chaîne reliant la source au puits telle que son flot peut être augmenté. Si on n'en trouve pas, le problème est résolu. Sinon, on augmente le flot sur cette chaîne. Ensuite, on recommence à chercher une chaîne augmentante et ainsi de suite. Une chaîne augmentante, i.e. une chaîne pour laquelle le flot peut être augmenté est une chaîne pour laquelle les arcs dans le sens direct n'ont pas atteint leur limite maximum et les arcs en sens indirect ont un flot non nul qui les traverse. L'augmentation de flot maximum pour une chaîne est le minimum des écarts entre le flot courant et le flot maximal pour les arcs directs ou le flot courant pour les arcs indirects. Autrement dit, une chaîne C est augmentante si:

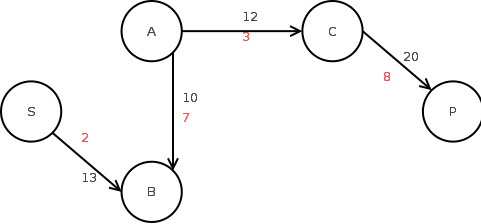
* pour tout arc u direct, f(u) < c(u),
* pour tout arc u indirect, f(u) > 0.

Le flot sur cette chaîne C peut être augmenté de:

min({c(u) - f(u) | u C et u sens direct}  {f(u) | u C et u sens indirect})

### Algorithmes

#### Exemple de chaines augmentantes



#### Recherche de chaines augmentantes

Titre: RechercherChaineAugmentante

Entrées:

R = (X;U;c) un réseau, s et p deux sommets, f() le flot courant.

Sorties:

pred() une fonction indiquant par quel arc on arrive à un noeud donné à partir de s.

Variables intermédiaires:

X' un sous-ensemble de noeuds, accessible() une fonction qui indique si un noeud est accessible à partir de s,

x et y deux noeuds.

Début

X' <-- {s};

pour tout x  X faire

accessible(x) <-- faux;

pred(x) <-- nil;

fin pour;

marqué(s) <-- vrai;

tant que X' et non accessible(p) faire

choisir x  X';

X' <-- X' – {x};

pour tout u = (x;y)  U faire

si non accessible(y) et f(u) < c(u) alors

X' <-- X'  {y};

pred(y) <-- x;

accessible(y) <-- vrai;

fin si;

fait;

pour tout u = (y;x) U faire

si non accessible(y) et f(u) > 0 alors

X' <-- X'  {y};

pred(y) <-- x;

accessible(y) <-- vrai;

fin si;

fait;

fait;

Fin

#### Algorithme de Ford-Fulkerson

Titre: FordFulkerson1

Entrées:

R = (X;U;c) un réseau, s et p deux sommets.

Sorties:

f() une fonction indiquant le flot circulant à travers chaque arc.

Variables intermédiaires:

C un sous-ensemble d'arcs,

u un arc,

m et a deux réels.

Début

pour tout arc u  U faire

f(u) <-- 0;

fait;

tant que une chaîne C augmentante entre s et p R faire // pred(P) != nil

m <-- + ;

pour tout u C faire

si u en sens direct alors

a <-- c(u) – f(u)

sinon

a <-- f(u);

fin si

si a < m

alors m <-- a;

fin si

fin pour;

pour tout u C faire

si u en sens direct alors

f(u) f(u) + m

sinon

f(u) f(u) – m;

fin Si

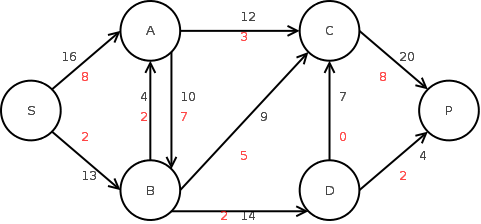
fin pour;

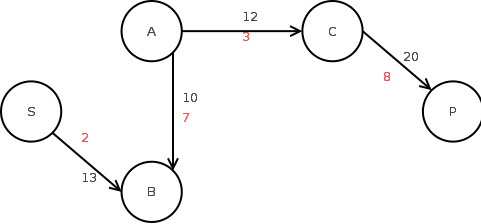
fin tant que;

Fin

#### Exemple

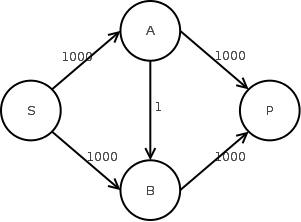
Augmenter l'exemple

avec la chaîne

Achever l'execution de FF

## Exercice

Faire FF avec le réseau suivant

Que pourrait-il se passer dans le pire des cas ?

## Convergeance de l'algorithme de Ford-Fulkerson

### Cas des flux entiers

La complexité de l'algorithme de FF est en O(|E|  f) ou f = max {f(u) | u  E}

### Cas des flux irrationnels

Dans le cas où le flux présente des valeurs irrationnelles la convergeance de l'algorithme n'est pas garantie : l'algorithme peut boucler indéfiniment.

### Remarque

Il existe toute fois une variation de l'algorithme de FF : l'algorithme de Edmond-karp qui a une complexité en O(VE²)

# Coloration de graphe

#### Algorithme de Welsh et Powell

Voici un exemple qui, en plus, fournit une preuve constructive du [théorème de Vizing](http://fr.wikipedia.org/wiki/Théorème_de_Vizing), qui établit la formule : , où représente le *degré* maximum d'un sommet de *G*. Pour s'en convaincre, appliquons l'algorithme suivant (Welsh et Powell) :

1. Repérer le degré de chaque sommet.
2. Ranger les sommets par ordre de degrés décroissants (dans certains cas plusieurs possibilités).
3. Attribuer au premier sommet (A) de la liste une couleur.
4. Suivre la liste en attribuant la même couleur au premier sommet (B) qui ne soit pas adjacent à (A).
5. Suivre (si possible) la liste jusqu'au prochain sommet (C) qui ne soit adjacent ni à A ni à B.
6. Continuer jusqu'à ce que la liste soit finie.
7. Prendre une deuxième couleur pour le premier sommet (D) non encore colorié de la liste.
8. Répéter les opérations 4 à 7.
9. Continuer jusqu'à avoir coloré tous les sommets.

Remarquons que cette méthode peut aboutir à la pire des colorations possibles, par-exemple si le graphe *G* a la structure de [couronne à n sommets](http://fr.wikipedia.org/wiki/Graphe_couronne), son nombre chromatique est 2 (si n est pair) tandis que Welsh-Powell donne dans certains cas (selon l'ordre dans lequel sont rangés les sommets) une coloration utilisant n/2 couleurs ! L'heuristique DSATUR permet d'éviter ce problème et donne une coloration moins mauvaise dans le pire cas.

#### DSATUR

On considère un graphe *G=(V,E)* simple connexe et non-orienté. Pour chaque sommet *v* de *V*, on calcule le degré de saturation *DSAT(v)* de la manière suivante:

*DSAT(v)=* nombre de couleurs différentes dans les sommets adjacents à *v*

L'algorithme DSATUR est un algorithme de coloration séquentiel, au sens où il colorie un seul sommet à la fois et tel que :

au départ le graphe n'est pas coloré

on colorie un sommet non déjà coloré

on stoppe DSATUR quand tous les sommets de *G* sont colorés.

Dans un premier temps on voit bien d'une part que la preuve de terminaison est triviale et d'autre part que l'algorithme est séquentiel. Dans le détail l'algorithme est le suivant :

1. Ordonner les sommets par ordre décroissant de degré.

2. Colorer un des sommets de degré maximum avec la couleur 1.

3. Choisir un sommet non coloré avec DSAT maximum. En cas d'égalité, choisir un sommet de degré maximal.

4. Colorer ce sommet par la plus petite couleur possible.

5. Si tous les sommets sont colorés alors stop. Sinon aller en 3.

(source : http://fr.wikipedia.org/wiki/Coloration\_de\_graphe)

# Programmation Linéaire

## Généralités

### Rappel

Soit A une matrice réelle de ***p*** lignes et ***n*** colonnes et C un vecteur de IRp. On cherche à résoudre l'équation A . X = C avec X un vecteur de ***n*** variables réelles.

Pour celà, nous allons utiliser le pivot de Gauss. Cette méthode repose sur les opérations élémentaires suivantes:

* multiplier une ligne par un nombre réelle ***non nul***;
* ajouter à une ligne un multiple d'une autre ligne.

Cette deux opérations permettent de réaliser un **"Pivot"** de la manière suivant:

Soit ***aij*** un coefficient non nul de la matrice A, on commence par diviser la ième ligne par ***aij***. Puis on retranche à chaque ligne ***k***  ***i*** du nouveau système ***akj*** fois la nouvelle ligne ***i***.

On obtient ainsi un nouveau système A' . X = C' vérifiant ***aij*** = 1 et pour tout ***k***  ***i,*** ***akj*** = 0.

On répète l'opération de pivotage afin d'obtenir un système résolu.

Ex : Pivoter le système

(Quels sont les cas possibles lors de la fin de la résolution?)

### Introduction

La Programmation Linéaire consiste à traduire une situation concréte à l’aide:

* d'équations et d'inéquations appelées "contraintes"
* une fonction appelée "objectif".

La fonction "objectif", les équations et les inéquations manipule des variables dites de décisions et différentes types de variables auxiliaires. De plus, les contraintes et la fonction "objectif" seront linéaires (i.e. toutes les variables apparaitront avec un degré 1).

La résolution d'un programme linéaire consiste **généralement** à trouver des valeurs pour les variables de sortes que toutes les contraintes soient vérifiés et que la fonction "objectif" soit optimisée (maximisée ou minimisée).

Dans le cours nous présenterons la méthode du SIMPLEX. (Nous commencerons par laisser de côté toutes les difficultés pour nous interesser au coeur de la méthode puis nous reviendrons dans un second temps sur ces différents points.)

### Exemple

Un fleuriste dispose de 50 lys, 80 roses et 80 jonquilles en stocke. Il réalise deux types de

* comprenant 10 lys, 10 roses et 20 jonquilles pour 40 euros;
* comprenant 10 lys, 20 roses et 10 jonquilles pour 50 euros.

Comment le fleuriste doit-il former les bouquets pour réaliser une recette maximale?

### Modélisation

On cherche ici le nombre de bouquet de type 1 et le nombre de bouquet de type 2. On note *x* le nombre de bouquets de type 1 et *y* le nombre de bouquets de type de 2.

On obtient alors 3 contraintes:

avec *x* et *y* positifs.

L'objectif est de maximiser *f(x) = 40 x + 50 y .*

### Résolution graphique

Dans un premier temps, on trace les droites permettant de délimiter les contraintes: Chaque inégalité est représentée par une demi-espace (ici un demi-plan).

Ici deux contraintes supplémentaires sont données de manière implicite: x 0 et y  0.

Si une contrainte est donnée par une équation, on aura nos variable qui prendrons leurs valeurs dans un hyperplan (sous espace de dimension n-1) : ici une droite.

Une fois l'ensemble des valeurs possibles délimité, on place l'hyperplan (la droite) correspondant à notre fonction objectif.

Puis, on la fait coulisser pour obtenir des valeurs d'objectif superieur.

Au final, la position limite de la droite correspond à la meilleur valeur possible. Sur notre exemple, on obtient le maximum pour x = 2 et y = 3 d'où f est maximale pour f(2, 3) = 80 + 150 = 230.



### Exercice

Une usine produit deux modèles de machines, l'une que l'on appellera modèle A exige 2 kg de matière première et de 30 heures de fabrication et donne un bénéfice de 7 €.  L'autre que l'on appellera B exige 4 kg de matière première et de 15 heures de fabrication et donne un bénéfice de 6 €. On dispose de 200 kg de matière première et de 1200 h de travail.

Formaliser ce problème en un programme linéaire.

Avec A, et B positifs

Résoudre ce problème graphiquement.



On trouve graphiquement le maximum en C(20, 40)

## Formalisation

### Données brutes

Un vecteur X de n variables :.

Une f : IRn  IR linéaire à maximiser ou à minimiser : f(X) = tB . X = < B | X >

p1 contraintes linéaires d'inégalités gk(X)  ck ou gk(X)  ck avec gk linéaire.

p2 contraintes linéaires d'égalités gk(X)  ck avec gk linéaire.

Remarque : les inégalités sont larges

### Canonisation

Afin de simplifier l'énoncé formel d'un PL, on peut imposer que la donnée respecte une certaine forme sans perte de généralité.

#### Forme canonique de maximisation avec inégalités

* On considère que la fonction ***f*** doit être maximiser.
  + Minimiser ***f*** revient à maximiser ***g = -f***
* Les ***xi*** sont tous positifs.
  + Si ***xi*** est négatif, on remplace toutes les occurences de cet variable par ***xi' = - xi*** avec ***xi'*** positifs.
  + Si ***xi*** est de signe quelconque, on remplace toutes les occurence de ***xi*** par ***xi* = *xi*+ - *xi*-** avec ***xi***+ et ***xi***- positifs
* Les containtes sont toutes des containtes du type inférieur ou égale
  + Si la contrainte ***gk(X)*  ck**, on remplace cette contrainte par ***-gk(X)  -ck***.
  + Si la contrainte ***gk(X)  ck***, on la remplace par deux contraintes ***gk(X)  ck***et ***gk(X)  ck***.

Le problème se réécrit de manière à l'aide :

* d'un vecteur X de ***n*** variables (toutes) positives
* d'un vecteur B de ***n*** valeurs réelles
* d'une matrice ***A*** avec ***p*** lignes et ***n*** colonnes.
* Et d'un vecteur C de ***p*** valeurs réelles.

Comme *max{tB . X | A . X* ****** *C, X }*

*! Ajouter un exemple !*

#### Forme canonique de maximisation avec égalités

* Les deux premiers points sur la maximisation de l'objectif et sur la positivité des variables sont les identiques ;
* Les containtes sont toutes des containtes d'égalités.
  + Si la contrainte ***gk(X)  ck***, on remplace cette contrainte par ***gk(X) + k  ck*** , avec k positif.
  + Si la contrainte ***gk(X)  ck***, on remplace cette contrainte par ***gk(X) - k  ck*** , avec k positif.

Le problème se réécrit de manière à l'aide :

* d'un vecteur X de ***n*** variables (toutes) positives
* d'un vecteur B de ***n*** valeurs réelles
* d'une matrice ***A*** avec ***p*** lignes et ***n*** colonnes.
* Et d'un vecteur C de ***p*** valeurs réelles.

Comme max{tB . X | A . X = C}

*! Ajouter un exemple !*

### Exerices

Donner la forme canonique de maximisation avec inégalités puis la forme canonique de maximisation avec égalité pour les problèmes du fleuriste puis de l'usine

### Résolution analytique et interprétation géométrique

Question ?

Quelle valeur peut-on donner aux variables ?

x <-- 0 ; y <-- 0 ; 1 <-- 50 ;  <-- 50 ; 3 <-- 50 ;

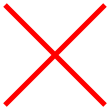
Comment augmenter la valeur de f ?

augmenter la valeur de x ou de y => choix x ! (ce n'est pas le meilleur)

jusqu'à quelle valeur je peux augmenter x ?

tant que 1 , 2 et 3 sont positifs (ou nuls)

I.E. Ici on peut annuler 3 en laissant 1 et 2 non nuls on effectue le pivot au tour du coefficient de la permier ligne et de la dernière colonne.



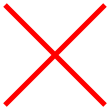
Observer la comportement de f(x, y, 1, 2, ) au cours de l'algorithme. On maximise une autre expression de la même fonction.

Premier critère de Dantzig : choisir une colonne ***j*** dont le coefficient sur la ligne "objectif" est positif.

On préfèrera le coefficient maximale mais ce critère n'est pas optimale.

Deuxième critère de Dantzig : choisir la ligne ***i*** tel que ***ci / aij*** soit minimale parmi les ***aij*** > 0

### Exercice une résolution Analytique avec le problème de l'usine



### Algorithme du simplex

L'algorithme du simlex consiste à faire des pivots (de Gauss) successif afin de faire apparaître la solution par lecture des valeurs des variables de bases et en annulant les varaibles hors-base.

Algorithme Simplex

p variables de bases (y1, ... yp)

Donnée une matrice A in Mp, n (IR)

les colonnes correspondant aux variables de bases presentent uniquement des 0 sauf sur une ligne. Les coefficients non nul sur une colonne d'une vecteur de bases sont sur des lignes différentes.

Un vecteur C de IRp dont toutes les valeurs sont positives ou nulles

un vecteur B de IRn.

Posons

#### Choix du pivot

#### Exercice écrire l'algorithme du pivot de Gauss

#### Exercice écrire l'algorithme du Simplex

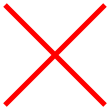
### Exercice

Une compagnie fabrique 3 types de plats P1, P2 et P, à partir de 4 ingrédients différents I1, I2, I3 et I4. Le tableau suivant rassemble toutes les contraintes matérielles :

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| INGEDIENT | P1 | P2 | P3 | Prix chez le fournisseur | Quantité maximale fournie |
| I1 | Au moins 15% du poids total | Au plus 20% du poids total | Sans spécification | 45 | 5200 |
| I2 | Au plus 15% du poids total | Sans spécification | Au plus 15% du poids total | 30 | 8000 |
| I3 | Sans spécification | Au plus 30% du poids total | Au plus 50% du poids total | 19 | 8000 |
| I4 | Au moins 20% du poids total | Au moins 40% du poids total | Au plus 15% du poids total | 35 | 7500 |
| Prix de ventre au kg | 80 | 75 | 54 |  |  |

Sachant que l'industriel veut maximiser les bénéfices, fournissez un programme linéaire modélisant le problème.

"Qij est la quantité d'ingrédiant i dans le produit j"

Donner le premier tableau du simplex. Pouvons nous le résoudre directement ?

## Différents cas possibles d'un PL (exo)

### Exercice

Donner le premier tableau du simplex du prblème suivant :

### Méthode du grand M

#### Variables supplémentaires

On ajoute une variable supplémentaire par contraintes non satisfaire (dont le second membre est négatif).

<Ck | X > = -z devient < Ck | X > - e' = -z puis <-Ck | X > + z = e

Puis, on pénalise la fonction objectif d'un coefficient -M pour chaque nouvelle variable :

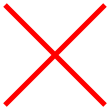
max <B | X > devient max <B | X> - M . z

#### Trouver le 1re système

Les nouvelles variables vont servir de variables de base or pour l'instant, il y a un coefficient non nul au niveau de la fonction objectif pour ces variables.

On effectue alors un pivot autour de ces nouvelles variables pour "nettoyer" leur colonne.

#### Sur l'exemple



### Cas d'arrêts

### Convergence