

# Recherche Opérationnelle - Cours 2

Olivier Baudon

Université de Bordeaux

5 octobre 2023

# Notations asymptotiques

## Notation $O$ ("grand o")

$$O(g(n)) = \{f(n) \mid \exists c \in \mathbb{R}^+, \exists n_0 \in \mathbb{N}, \forall n \geq n_0, 0 \leq f(n) \leq c \times g(n)\}$$

*Remarque : on ne notera pas  $f(n) \in O(g(n))$ , mais  $f(n) = O(g(n))$*

*Attention ! Cette notation "=" n'est pas symétrique !*

## Exemple

$$3 * n^2 + 2 * n + 3 = O(n^2)$$

Il suffit de prendre  $c = 5$  et  $n_0 = 2$ .

*On pourrait aussi dire que  $3 * n^2 + 2 * n + 3 = O(n^3)$ , mais l'approximation est moins précise !*

# Notations asymptotiques

## Notation $\Omega$

$$\Omega(g(n)) = \{f(n) \mid \exists c \in \mathbb{R}^+, \exists n_0 \in \mathbb{N}, \forall n \geq n_0, 0 \leq c \times g(n) \leq f(n)\}$$

## Notation $\Theta$

$$\Theta(g(n)) = \{f(n) \mid \exists c_1, c_2 \in \mathbb{R}^+, \exists n_0 \in \mathbb{N}, \forall n \geq n_0, 0 \leq c_1 \times g(n) \leq f(n) \leq c_2 \times g(n)\}$$

## Théorèmes

$$f(n) = O(g(n)) \Leftrightarrow g(n) = \Omega(f(n))$$

$$f(n) = \Theta(g(n)) \Leftrightarrow f(n) = O(g(n)) \text{ et } f(n) = \Omega(g(n))$$

$$f(n) = \Theta(g(n)) \Leftrightarrow f(n) = O(g(n)) \text{ et } g(n) = O(f(n))$$

## Existence d'une arête ou d'un arc entre deux sommets

$A$  : matrice d'adjacence d'un graphe  $G$  orienté ou non

$i, j$  : deux sommets de  $G$

Teste si  $j$  est voisin ou successeur de  $i$

**SontVoisins**( $A, i, j$ ) :

1: retourner  $A[i, j] \neq 0$

Complexité :  $O(1)$

## Degré d'un sommet

$A$  : matrice d'adjacence d'un graphe  $G$  non orienté

$i$  : sommet de  $G$

**Degré**( $A, i$ ) :

- 1:  $degre \leftarrow 0$
- 2: **pour**  $j$  de 1 à  $n$  **faire**
- 3:   **si**  $i \neq j$  **alors**
- 4:      $degre \leftarrow degre + A[i, j]$
- 5:   **sinon**
- 6:      $degre \leftarrow degre + 2 \times A[i, j]$
- 7:   **fin si**
- 8: **fin pour**
- 9: **retourner**  $degre$

Complexité :  $O(n)$

# Complexité selon la représentation - Matrice d'adjacence

## Degrés de tous les sommets

$A$  : matrice d'adjacence d'un graphe  $G$  non orienté

**Degres( $A$ ) :**

- 1: **pour**  $i$  de 1 à  $n$  **faire**
- 2:      $degre[i] \leftarrow 0$
- 3:     **pour**  $j$  de 1 à  $n$  **faire**
- 4:         **si**  $i \neq j$  **alors**
- 5:              $degre[i] \leftarrow degre[i] + A[i, j]$
- 6:         **sinon**
- 7:              $degre[i] \leftarrow degre[i] + 2 \times A[i, j]$
- 8:         **fin si**
- 9:     **fin pour**
- 10: **fin pour**
- 11: **retourner**  $degre$

Complexité :  $O(n^2)$

## Existence d'une arête

$B$  : matrice d'incidence d'un graphe  $G$  non orienté

$i, j$  : deux sommets de  $G$

**SontVoisins**( $B, i, j$ ) :

- 1: **pour**  $e$  de 1 à  $m$  **faire**
- 2:     **si**  $B[i, e] \neq 0$  **et**  $B[j, e] \neq 0$  **alors**
- 3:         **retourner** VRAI
- 4:     **fin si**
- 5: **fin pour**
- 6: **retourner** FAUX

Complexité :  $O(m)$

Pour tester l'existence d'un arc de  $i$  vers  $j$ , il suffit de remplacer la condition :  $B[i, e] \neq 0$  et  $B[j, e] \neq 0$  par :  $B[i, e] = -1$  et  $B[j, e] = 1$

Par contre, il est impossible avec la matrice d'incidence d'un graphe orienté de tester s'il existe une boucle sur un sommet  $i$  donné.

## Degré d'un sommet

$B$  : matrice d'incidence d'un graphe  $G$  non orienté

$i$  : sommet de  $G$

**Degré**( $B, i$ ) :

- 1:  $degre \leftarrow 0$
- 2: **pour**  $e$  de 1 à  $m$  **faire**
- 3:      $degre \leftarrow degre + B[i, e]$
- 4: **fin pour**
- 5: **retourner**  $degre$

Complexité :  $O(m)$



## Degrés de tous les sommets

$B$  : matrice d'incidence d'un graphe  $G$  non orienté

**Degres( $A$ ) :**

- 1: **pour**  $i$  de 1 à  $n$  **faire**
- 2:      $degres[i] \leftarrow 0$
- 3:     **pour**  $e$  de 1 à  $m$  **faire**
- 4:          $degre[i] \leftarrow degre[i] + B[i, e]$
- 5:     **fin pour**
- 6: **fin pour**
- 7: **retourner**  $degres$

Complexité :  $O(n \times m)$

## Existence d'une arête

$G$  : un graphe (orienté ou non)

$u, v$  : deux sommets de  $G$

Teste si  $v$  est voisin ou successeur de  $u$

**SontVoisins**( $G, u, v$ ) :

- 1: **pour tout**  $w \in Adj(u)$  **faire**
- 2:     **si**  $w = v$  **alors**
- 3:         **retourner** VRAI
- 4:     **fin si**
- 5: **fin pour**
- 6: **retourner** FAUX

Complexité :  $O(\Delta(G)) = O(n)$

## Degré d'un sommet

$G$  : un graphe non orienté

$u$  : un sommet de  $G$

**Degré**( $G, u$ ) :

1:  $degre \leftarrow 0$

2: **pour tout**  $v \in Adj(u)$  **faire**

3:   **si**  $v \neq u$  **alors**

4:      $degre(u) \leftarrow degre(u) + 1$

5:   **sinon**

6:      $degre(u) \leftarrow degre(u) + 2$

7:   **fin si**

8: **fin pour**

9: **retourner**  $degre$

Complexité :  $O(\Delta(G)) = O(n)$

# Complexité selon la représentation : Listes d'adjacence

## Degrés de tous les sommets

$G$  : un graphe non orienté

**Degres( $G$ ) :**

- 1: **pour tout**  $u \in V(G)$  **faire**
- 2:      $degres(u) \leftarrow 0$
- 3:     **pour tout**  $v \in Adj(u)$  **faire**
- 4:         **si**  $v \neq u$  **alors**
- 5:              $degres(u) \leftarrow degres(u) + 1$
- 6:         **sinon**
- 7:              $degres(u) \leftarrow degres(u) + 2$
- 8:         **fin si**
- 9:     **fin pour**
- 10: **fin pour**
- 11: **retourner**  $degres$

Complexité :  $O(n + m)$

## Problématiques

Soit  $G$  un graphe orienté et  $\omega$  une fonction de poids sur les arcs de  $G$ .

*Si  $G$  n'est pas orienté, on peut considérer chaque arête comme une paire d'arcs symétriques, sauf pour les boucles qui seront simplement orientées.*

1. Quel est le plus court chemin pour aller de  $u$  à  $v$  ?
2. Quel est le plus court chemin d'un sommet  $u$  à tous les sommets du graphe ?
3. Quel est le plus court chemin entre toute paire de sommets ?

## Théorème

Il existe un plus court chemin de  $u$  à  $v$  dans  $(G, w)$  si et seulement si il existe un chemin de  $u$  à  $v$  et aucun chemin de  $u$  à  $v$  ne contient de circuit de longueur totale strictement négative.

## Arborescence

L'ensemble des plus courts chemins à partir d'un sommet  $s$  forme une arborescence de racine  $s$ .

On notera par  $d(v)$  la distance trouvée de  $s$  à  $v$  et par  $\pi(v)$  le prédécesseur de  $v$  sur le chemin de  $s$  à  $v$  de longueur  $d(v)$ .

## Principe du relâchement

Soient

$d$  la valeur du plus court chemin trouvé à l'instant  $t$  entre un sommet  $s$  et les sommets de  $G$ ,

$\pi$  les prédécesseurs de chaque sommet sur les plus courts chemins trouvés à l'instant  $t$  depuis le sommet  $s$ ,

$uv$  un arc de  $G$ ,

**Relacher**( $G, \omega, u, v$ ) :

- 1: **si**  $d(u) + \omega(uv) < d(v)$  **alors**
- 2:      $d(v) \leftarrow d(u) + \omega(uv)$
- 3:      $\pi(v) \leftarrow u$
- 4: **fin si**

# Algorithme de Dijkstra

## Utilisation

On peut utiliser l'algorithme de Dijkstra à condition que la fonction  $\omega$  de pondération des arcs soit positive.

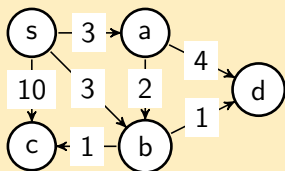
## Énoncé

Voir document "Algorithmes des graphes" page 2.



# Algorithme de Dijkstra

## Exemple



Graphe  $G$

<i>pivot</i>	<i>s</i>	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>
	0	$\infty$	$\infty$	$\infty$	$\infty$
<i>s</i>	0	3	3	10	$\infty$
<i>a</i>	0	3	3	10	7
<i>b</i>	0	3	3	4	4
<i>c</i>	0	3	3	4	4
<i>d</i>	0	3	3	4	4

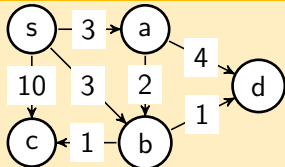
$Dijkstra(G, \omega, s)$

Remarque :  $\pi(s) = NIL$  et  $\pi(v)$  est le pivot quand  $d(v)$  a été modifié pour la dernière fois.

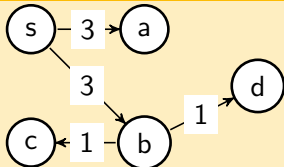
Ici :  $\pi(s) = NIL, \pi(a) = s, \pi(b) = s, \pi(c) = b, \pi(d) = b$ .

# Algorithme de Dijkstra

## Exemple - Arborescence des plus courts chemins



Graphe G



Arborescence

## Complexité

Si la file de priorité est implémentée sous la forme d'une liste, alors la complexité de l'algorithme de Dijkstra est en  $O(n^2 + m)$ . Si  $G$  ne possède pas d'arcs parallèles, cela nous donne du  $O(n^2)$ .

Si le nombre d'arcs est faible par rapport à  $n^2$ , on a intérêt à implémenter la file de priorité à l'aide d'un tas binaire. Dans ce cas, la complexité de Dijkstra devient  $O((n + m) \times \log(n))$ .

# Algorithme de Dijkstra

## Validité

On note par  $\delta(u)$  la plus courte distance entre  $s$  et  $u$  dans  $G$ .

On va montrer que  $d(u) = \delta(u)$  à chaque fois qu'un sommet  $u$  est extrait de la file de priorité  $F$ .

Supposons que ce ne soit pas le cas. Soit  $u$  le premier sommet extrait de  $F$  tel que  $d(u) \neq \delta(u)$  à cet instant.

$u$  ne peut pas être égal à  $s$ , car  $s$  est le premier sommet extrait de  $F$  et  $d(s) = \delta(s) = 0$ .

De plus,  $d(u) \neq \infty$ , sinon,  $u$  n'aurait pas été inséré dans  $F$ . Donc il existe lors de l'extraction de  $u$  un chemin de  $s$  à  $u$  de longueur  $d[u] \neq \infty$  et donc il existe un plus court chemin dans  $G$  de longueur  $\delta(u) \neq \infty$ . Soit  $p$  ce chemin et soit  $y$  le premier sommet de  $p$  non extrait de  $F$  quand on extrait  $u$ .  $y$  existe, sinon on aurait  $d(u) = \delta(u)$ . Soit  $x$  le prédécesseur de  $y$  dans  $p$ . Comme  $x$  a été extrait de  $F$  avant  $u$ , cela signifie que  $d(y) = \delta(y)$  quand on extrait  $u$  et comme  $\delta(y) < \delta(u) < d[u]$ ,  $y$  aurait dû être extrait de  $F$  avant  $u$ . Donc  $u$  n'existe pas !

# Algorithme de Bellman

## Utilisation

On peut utiliser l'algorithme de Bellman à condition que le graphe  $G$  soit sans circuit.

Dans ce cas, on peut obtenir un **tri topologique** sur les sommets de  $G$ , c'est à dire un ordre tel que si un arc va du sommet de rang  $i$  vers le sommet de rang  $j$ , alors  $i < j$ .

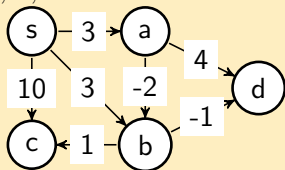
## Énoncé

Voir document "Algorithmes des graphes" pages 3 et 4.

# Algorithme de Bellman

## Exemple

Le tri topologique de  $G$  donne deux ordres possibles :  $s, a, b, c, d$  et  $s, a, b, d, c$ .



Graphe  $G$

$u$	$s$	$a$	$b$	$c$	$d$	File
	0	$\infty$	$\infty$	$\infty$	$\infty$	$s$
$s$	0	3	3	10	$\infty$	$a$
$a$	0	3	1	10	7	$b$
$b$	0	3	1	2	0	$cd$
$c$	0	3	1	2	0	$d$
$d$	0	3	1	2	0	$\emptyset$

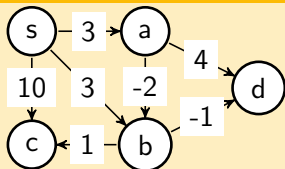
$Bellman(G, \omega, s)$

Remarque :  $\pi(s) = NIL$  et  $\pi(v)$  est la tête de liste quand  $d(v)$  a été modifié pour la dernière fois.

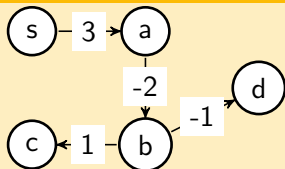
Ici :  $\pi(s) = NIL, \pi(a) = s, \pi(b) = a, \pi(c) = b, \pi(d) = b$ .

# Algorithme de Bellman

## Exemple - Arborescence des plus courts chemins



Graphe G



Arborescence

# Algorithme de Bellman

## Complexité

La complexité de l'algorithme de Bellman est en  $O(n + m)$ .

## Validité

On a bien  $d(s) = \delta(s)$

Si  $v_1 = s, v_2, \dots, v_k$  est l'ordre obtenu par le tri topologique, il est clair que si pour tout  $j < i$ ,  $d(v_j) = \delta(v_j)$ , alors  $d(v_i) = \delta(v_i)$ .

Donc par récurrence, on a bien  $\forall i, 1 \leq i \leq k, d(v_i) = \delta(v_i)$ .



## Objectifs

L'objectif d'un graphe PERT est d'aider à la gestion de projets, en identifiant les tâches pour lesquelles un retard entraînera un retard sur le projet global et pour les autres le temps que l'on peut rajouter à leur exécution sans retarder la fin du projet.

## Modélisation

Chaque tâche est représentée par :

- ▶ deux sommets :  
un pour le début de la tâche et un pour la fin ;
- ▶ les deux sommets liés à une tâche  $t$  sont reliés par un arc ayant comme poids la durée de la tâche  $t$  ;

Si une tâche  $t$  doit précéder immédiatement une tâche  $t'$ , on rajoute un arc de poids 0 entre le sommet représentant la fin de  $t$  et celui représentant le début de  $t'$ .

## Modélisation - suite

On rajoute également deux sommets :

- ▶ un pour le début de projet, relié par un arc de poids 0 à tous les sommets représentant les débuts des tâches n'ayant aucune contrainte ;
- ▶ un pour la fin de projet auquel sont reliés tous les sommets représentant la fin des tâches n'étant une contrainte pour aucune autre.

## Exemple - Projet

Le projet est composé de 5 tâches : A, B, C, D, E.

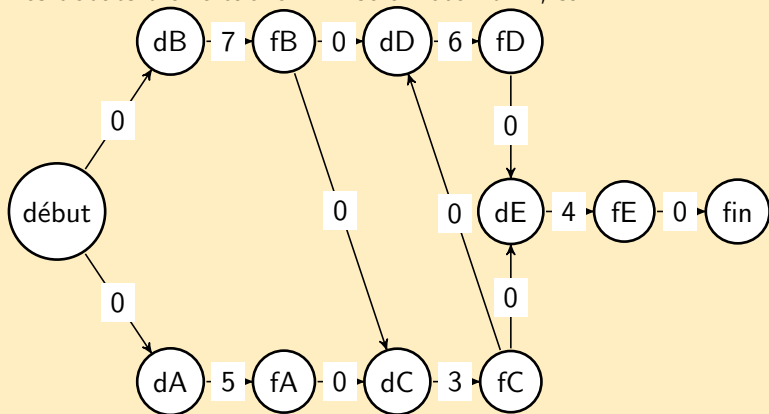
Le tableau suivant donne pour chaque tâche  $t$  sa durée et les tâches devant être achevées avant de pouvoir commencer  $t$ . Le graphe est le graphe PERT associé.

tâche	durée	contraintes
<i>A</i>	5	
<i>B</i>	7	
<i>C</i>	3	<i>A, B</i>
<i>D</i>	6	<i>B, C</i>
<i>E</i>	4	<i>C, D</i>

# Graphe PERT

## Exemple - Graphe

Les débuts d'une tâche "X" sera noté "dX", sa fin "fX".



## Calculs

On peut tout d'abord remarquer que par essence, un graphe PERT est forcément sans circuit.

On va donc appliquer l'algorithme de Bellman, modifié, pour dans un premier temps calculer les plus longs chemins entre le début du projet et les autres noeuds, ce qui nous donnera les dates au plus tôt de chaque début et fin de tâche.

On va ensuite re-appliquer l'algorithme de Bellman dans le graphe inverse, pour calculer les dates au plus tard, afin de savoir sur chaque tâche de quel délai on peut disposer pour la réaliser et déterminer les tâches critiques, c'est à dire les tâches où les dates au plus tôt et au plus tard sont égales, et donc pour lesquelles tout retard entrainera un retard du projet global.

## Exemple - Calculs

Le tableau suivant donne les dates aux plus tôt et au plus tard de chaque début et fin de tâche.

tâche	date au plus tôt	date au plus tard
début A	0	2
fin A	5	7
début B	0	0
fin B	7	7
début C	7	7
fin C	10	10
début D	10	10
fin D	16	16
début E	16	16
fin E	20	20

## Exemple - Tâches critiques

La durée minimum du projet est de 20. Les tâches critiques sont B, C, D et E.