Introduction aux graphes et aux matrices

Cours n°5

EPITA Cyber 2
2025-2026

Ce cours présente le problème de recherche d'un plus court chemin et deux algorithmes résolvant ce problème, l'algorithme de Bellman et l'algorithme de Dijkstra.

1 Introduction

Le problème de **recherche d'un plus court chemin (PCCH)** est un problème essentiel de la théorie des graphes. Ce problème possède des applications dans de nombreux domaines, tels que la recherche d'un itinéraire sur une carte, le routage internet ou téléphonique, ou le calcul de trajectoire d'un robot. Il existe différents algorithmes résolvant ce problème, nous étudierons deux algorithmes : l'agorithme de Bellman et l'algorithme de Dijkstra.

2 Problème du Plus Court Chemin

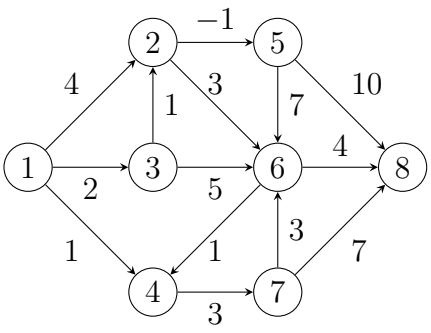
2.1 Définition

Le problème de recherche d'un plus court chemin nécessite un graphe pondéré, orienté ou non-orienté. Dans le cas d'un graphe non-orienté, on considère que les arêtes peuvent être empruntées dans les deux sens.

Soit $G = (X, U)$ un graphe orienté et pondéré. Pour un arc (i, j) de U, on note $w(i, j)$ **le poids associé à l'arc** (i, j). Dans le contexte du problème de plus court chemin, le poids d'un arc (i, j) est parfois appelé **sa longueur**. Le **poids d'un chemin** μ dans G correspond à la somme des poids des arcs qui composent ce chemin.

Définition 1. *Soit $G = (X, U)$ un graphe orienté et pondéré et $w(i, j)$ la fonction de poids. La recherche d'un plus court chemin d'un sommet i_0 vers un sommet j_0 consiste à déterminer le chemin μ^* tel que, parmi tous les chemins allant de i_0 à j_0, μ^* possède un poids total, $\sum_{(i,j)\in\mu^*} w(i, j)$, minimal.*

Exemple :



Recherche d'un plus court chemin de 1 vers 8 :

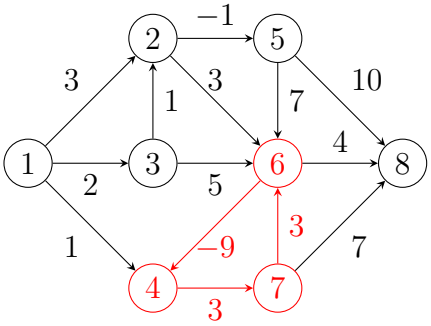
$\mu_1 = (1, 3, 6, 8)$ est un chemin de poids (ou longueur) 11 ;
$\mu_2 = (1, 2, 5, 8)$ est un chemin de poids 13 ;
$\mu_3 = (1, 4, 7, 6, 8)$ est un chemin de poids 11 ;
$\mu_4 = (1, 2, 6, 8)$ est un chemin de poids 11 ;
$\mu_5 = (1, 3, 2, 6, 8)$ est un chemin de poids 10 ;
μ_5 est un chemin de 1 vers 8 de poids minimal ;
$\Rightarrow$ μ_5 est un plus court chemin de 1 vers 8.

FIGURE 1 – Graphe G_1 orienté d'ordre 8.

2.2 Condition d'existence

Dans un graphe orienté et pondéré, un **circtuit absorbant** est un circuit dont le poids total est strictement négatif. Il existe une solution au problème de recherche d'un plus court chemin si et seulement si **il n'existe pas de circuit absorbant** dans le graphe étudié.

En effet, si on emprunte un circuit absorbant sur le chemin de i_0 à j_0, alors le poids du chemin diminue strictement. En empruntant autant de fois que l'on souhaite le circuit absorbant, le poids du chemin peut diminuer de façon arbitraire.
Considérons l'exemple du graphe G_1 où le poids de l'arc $(6, 4)$ passe à -9.



Le circuit $(4, 7, 6, 4)$ est un circuit de poids -3. En l'empruntant, le poids total du chemin diminue de -3.
Ainsi, le chemin de 1 vers 8 $(1, 4, 7, 6, 4, 7, 6, 8)$ a pour poids total 2, et le chemin $(1, 4, 7, 6, 4, 7, 6, 4, 7, 6, 8)$ a pour poids total -1.
$\rightarrow$ On peut construire un chemin de 1 vers 8 de poids aussi faible que l'on souhaite en empruntant le circuit absorbant $(4, 7, 6, 4)$.

Dans les graphes orientés, pour assurer l'existence d'un plus court chemin, on peut considérer les graphes qui ne contiennent pas de circuits **ou** les graphes qui ne contiennent de valeurs négatives :

- Dans le cas où le **graphe ne contient pas de circuit** (mais contient peut-être des valeurs négatives), l'**algorithme de Bellman** permet de déterminer les plus courts chemins d'un sommet i_0 vers tous les autres sommets.
- Dans le cas où le **graphe ne contient pas de valeurs négatives** (mais contient peut-être des circuits), l'**algorithme de Dijkstra** permet de déterminer les plus courts chemins d'un sommet i_0 vers tous les autres sommets. En particulier, l'algorithme de Dijkstra s'applique aux graphes non orientés où l'on considère qu'une arête est en fait deux arcs de sens opposés.

Remarque : Il existe une extension de l'algorithme de Bellman, nommé algorithme de Bellman-Ford, qui est utilisable sur des graphes (orientés ou non) avec circuit et valeurs négatives, et qui détecte l'existence de circuits absorbants. Nous n'étudierons pas cet algorithme dans le cadre de ce cours.

2.3 Catégorie d'algorithme

En déterminant le plus court chemin d'un sommet i_0 vers un sommet j_0, on détermine également les plus courts chemins de i_0 vers les autres sommets du graphes. Les algorithmes de recherche d'un plus court chemin se décompose en deux catégories : les algorithmes de recherche des plus courts chemins à partir d'un sommet i_0 vers tous les autres sommets du graphes, et les algorithmes de recherche des plus courts chemins entre chaque paire de sommet du graphe.

3 Algorithme de Bellman

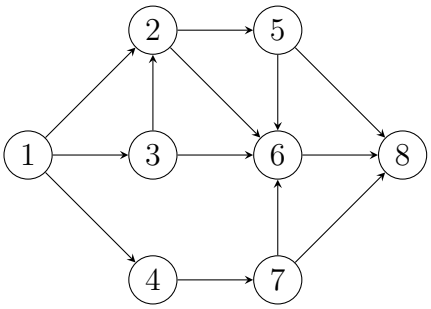
Dans le cas où le graphe ne contient pas de circuit (mais contient peut-être des valeurs négatives), l'algorithme de Bellman permet de déterminer les plus courts chemins d'un sommet i_0 vers tous les autres sommets.
Commençons par définir la notion d'**ordre topologique**.

3.1 Ordre topologique

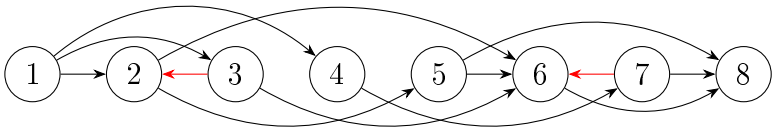
Définition 2. *Dans un graphe orienté $G = (X, U)$, un **ordre topologique** est une numérotation $v(\cdot)$ des sommets telle que : $\forall (i, j) \in U, v(i) < v(j)$. Autrement dit, si il existe un arc (i, j) dans le graphe, alors le sommet i est rangé avant le sommet j dans un ordre topologique.*

Pourquoi parle-t-on d'ordre "topologique" ? Le mot topologique fait référence au positionnement des nœuds dans l'espace. Lorsque l'on représente les nœuds sur une ligne et dans un ordre topologique (de gauche à droite), alors tous les arcs vont vers l'avant (la droite). Cette vision de l'ordre topologique nous aidera à comprendre pourquoi l'algorithme de Bellman fonctionne.

Exemple : Considérons le graphe G_2 qui ne contient pas de circuit.



L'ordre naturel $(1, 2, 3, 4, 5, 6, 7, 8)$ ne constitue pas un ordre topologique car les arcs $(3, 2)$ et $(7, 6)$ ne suivent pas cet ordre :



L'ordre $(1, 3, 2, 4, 5, 7, 6, 8)$ est un ordre topologique :

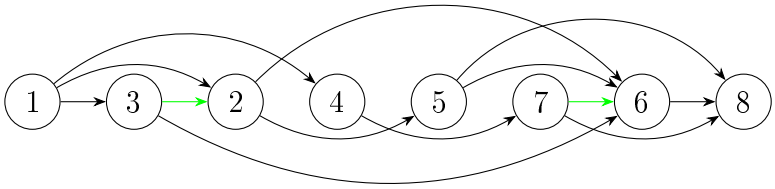
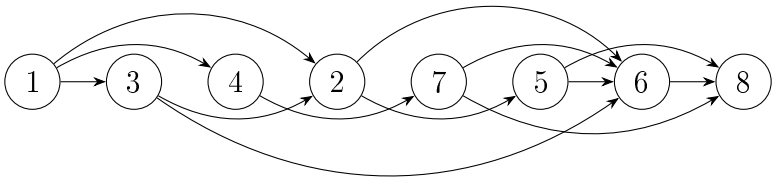


FIGURE 2 – Graphe G_2 orienté d'ordre 8.

Noter qu'un ordre topologique n'est pas nécessairement unique. Par exemple, dans le graphe G_2 de la figure 2, un autre ordre topologique possible est $(1, 3, 4, 2, 7, 5, 6, 8)$:



Propriété : *Lorsque le* ***graphe ne contient pas de circuit****, il existe un ordre topologique des sommets.*

Propriété : *La* ***numérotation suffixe inverse*** *obtenue par un parcours en profondeur du graphe constitue un ordre topologique.*

3.2 Principe de l'algorithme de Bellman

L'algorithme de Bellman réalise un **marquage des sommets** : une marque $\lambda(j)$ est associée à chaque sommet j, et représente à la fin de l'algorithme le poids d'un plus court chemin de i_0 vers j.

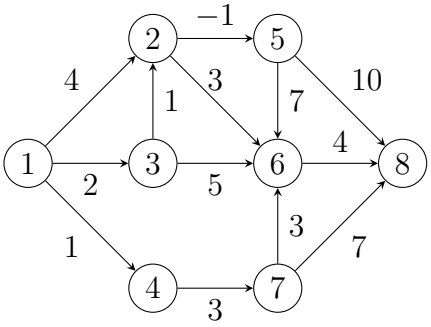
Dans l'agorithme de Bellman, les sommets sont marqués en suivant un ordre topologique du graphe. Considérons que l'ordre topologique obtenu est l'ordre $(1, 2, 3, \ldots, n)$ et que l'on cherche le plus court chemin de 1 vers tous les autres sommets.

L'algorithme suit alors les étapes suivantes :

- À l'itération 1, le sommet de départ 1 est marqué avec la valeur $\lambda(1) = 0$.
- On parcours les nœuds dans l'ordre topologique. À l'itération j, on détermine la marque du sommet j par
$$\lambda(j) = \min_{i \in N^-(j)} \Big(\lambda(i) + w(i,j) \Big)$$
c'est-à-dire, pour chaque sommet i prédécesseur de j, on ajoute le poids $w(i,j)$ à la marque $\lambda(i)$, et la marque de j est la plus petite parmi ces marques.
- A la fin de l'algorithme, pour tout sommet j du graphe, la marque $\lambda(j)$ contient le poids d'un plus court chemin de 1 vers j.

Lors du marquage d'un sommet j, on enregistre également le sommet prédécesseur de j qui fournit la marque la plus petite. Cela permet de reconstituer le plus court chemin de 1 vers j.

Exemple : Considérons le graphe G_3 qui ne contient pas de circuit.



L'ordre $(1, 3, 2, 4, 5, 7, 6, 8)$ est un ordre topologique.

$\rightarrow$ Les sommet seront marqués suivant l'ordre $(1, 3, 2, 4, 5, 7, 6, 8)$ lors de l'algorithme de Bellman.

FIGURE 3 – Graphe G_3 orienté d'ordre 8.

Algorithme de Bellman sur G_3 :

sommet	$\lambda(\cdot)$	parent$(\cdot)$
1	0	/
3	2	1
2	3	3
4	1	1
5	2	2
7	4	4
6	6	2
8	10	6

Les sommet ont été marqués en suivant l'ordre topologique $(1, 3, 2, 4, 5, 7, 6, 8)$. À la fin de l'algorithme, la marque $\lambda(j)$ représente le poids d'un plus court chemin de 1 vers j.

$\rightarrow$ Le plus court chemin du sommet 1 vers le sommet 8 est $(1, 3, 2, 6, 8)$ avec comme poids total 10.

Implémentation.

Algorithme de Bellman :

```
bellman(graphe G):
    λ(1) = 0
    p(1) = 1
    pour chaque sommet j dans l'ordre topologique:
        λ(j) = min_{i∈N⁻(j)} ( λ(i) + w(i,j) )
        p(j) = argmin_{i∈N⁻(j)} ( λ(i) + w(i,j) )
```

Complexité. La complexité de l'algorithme de Bellman est en $O(n + m)$. En effet, déterminer un ordre topologique s'effectue en $O(n + m)$, en prenant l'ordre suffixe inverse d'un parcours en profondeur. Puis, pour chaque sommet j, le marquage des sommets s'effectue $O(d^-(j))$ opérations, soit $O(m)$ opérations au total.

4 Algorithme de Dijkstra

Dans le cas où le graphe ne contient pas de longueurs négatives (mais contient peut-être des circuits), l'algorithme de Dijkstra permet de déterminer les plus courts chemins d'un sommet i_0 vers tous les autres sommets.

Principe de l'algorithme

L'algorithme de Dijkstra réalise un **marquage des sommets** : une **marque temporaire** $\lambda(j)$ est associée à chaque sommet j, cette marque évolue à chaque itération et représente à la fin de l'algorithme le poids d'un plus court chemin de i_0 vers j.

A chaque itération, une **marque temporaire** d'un sommet devient **définitive** et les marques des sommets voisins évoluent.

Ainsi, à chaque itération, l'ensemble X des sommets est partagé en **deux sous-ensembles** S qui contient les sommets dont la marque est définitive, et $X \setminus S$ qui contient les sommets dont la marque peut encore évoluer.

L'algorithme de Dijkstra suit les étapes suivantes :

— A l'itération 1, le sommet de départ i_0 est marqué avec la valeur $\lambda(i_0) = 0$ et les autres sommets avec $\lambda(i) = +\infty$.

— A une itération quelconque, on détermine le **sommet i de plus petite marque temporaire**, la marque de i devient définitive, et les marques des successeurs de i non définitivement marqués évoluent de la façon suivante :

$$\forall j \in N^+ \setminus S, \lambda(j) = min\{\lambda(j); \lambda(i) + w(i,j)\}$$

c'est-à-dire, pour chaque sommet j successeur de i, la nouvelle marque de j est le minimum entre $\lambda(j)$ et $\lambda(i) + w(i,j)$.

— Après n itérations, toutes les marques sont définitives et une marque $\lambda(j)$ contient le poids d'un plus court chemin de i_0 vers j.

Lors du marquage d'un sommet j, on enregistre également le sommet prédécesseur de j qui fournit la marque la plus petite. Cela permet de reconstituer le plus court chemin de i_0 vers j.
Exemple : Considérons le graphe G_3 qui ne contient pas de valeurs négatives.

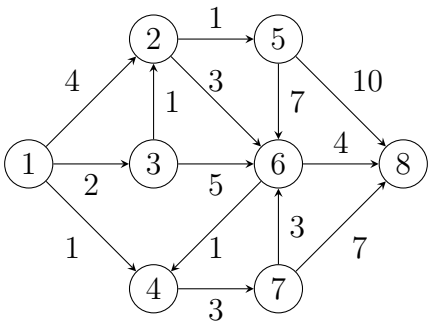


FIGURE 4 – Graphe G_3 orienté d'ordre 8.

Itération	$\lambda(1)$	$\lambda(2)$	$\lambda(3)$	$\lambda(4)$	$\lambda(5)$	$\lambda(6)$	$\lambda(7)$	$\lambda(8)$	sommet sélectionné
1	0* (1)	∞	∞	∞	∞	∞	∞	∞	1
2	/	4 (1)	2 (1)	1* (1)	∞	∞	∞	∞	4
3	/	4 (1)	2* (1)	/	∞	∞	4 (4)	∞	3
4	/	3* (3)	/	/	∞	6 (2)	4 (4)	∞	2
5	/	/	/	/	4* (2)	6 (2)	4 (4)	∞	5
6	/	/	/	/	/	6 (2)	4* (4)	14 (5)	7
7	/	/	/	/	/	6* (2)	/	11 (7)	6
8	/	/	/	/	/	/	/	10* (6)	8

FIGURE 5 – Application de Dijkstra : les marques avec une étoile sont définitives.

Explication :

— L'itération #1 initialise la marque du sommet 1 à 0, et des autres sommets à ∞.

— A l'itération #2, la marque du sommet 1 est considéré comme définitive.
On met à jour les marques des sommets successeurs du sommet 1, en enregistrant le sommet parent qui conduit à la nouvelle marque.

— A l'itération #3, on sélectionne le sommet de marque temporaire la plus petite, c'est-à-dire le sommet 4, et cette marque devient définitive.
On met à jour les marques des sommets successeurs du sommet 4, en enregistrant le sommet parent qui conduit à la nouvelle marque.

— A l'itération #4, on sélectionne le sommet de marque temporaire la plus petite, c'est-à-dire le sommet 3, et cette marque devient définitive.
On met à jour les marques des sommets successeurs du sommet 3, en enregistrant le sommet parent qui conduit à la nouvelle marque.

— A l'itération #5, on sélectionne le sommet de marque temporaire la plus petite, c'est-à-dire le sommet 2, et cette marque devient définitive.
On met à jour les marques des sommets successeurs du sommet 2, en enregistrant le sommet parent qui conduit à la nouvelle marque.

— ...

— A la fin de l'itération #8, le sommet 8 est marqué définitivement avec la valeur 10.
$\Rightarrow$ Le plus court chemin du sommet 1 vers 8 est $(1, 3, 2, 6, 8)$ de poids total 10.

Implémentation

Algorithme de Dijkstra :

```
dijkstra(graphe G) :
    λ(i_0) = 0
    p(i_0) = 1
    pour chaque sommet j ≠ i_0:
        λ(j) = ∞
    S = ∅
    pour n iterations :
        Selectionner i ∈ X \ S dont la marque est minimale
        S = S ∪ {i}
        pour chaque sommet j ∈ N+(i) \ S :
            Si λ(j) > λ(i) + w(i,j) alors
                λ(j) = λ(i) + w(i,j)
                p(j) = i
```

Complexité

La complexité de l'algorithme de Dijkstra est en $O(n^2)$.
En effet, l'algorithme effectue n itérations. A chaque itération, on sélectionne le sommet i de marque temporaire minimal en $O(n)$ opérations. Puis, on met à jour les marques de ses sommets successeurs en $O(1 + d^+(i))$ opérations. La complexité est donc en $O(n^2) + O(n + m)$, soit $O(n^2)$.