

Plus courts chemins dans un graphe avec l'algorithme de Dijkstra

Quentin Fortier

Graphe pondéré : Poids

On considère dans ce cours seulement des graphes orientés.

Définition

Un graphe **pondéré** est un graphe $\vec{G} = (V, \vec{E})$ muni d'une fonction de poids $w : \vec{E} \rightarrow \mathbb{R}$.

$w(u, v)$ est le **poids** de l'arête de u vers v .

Graphe pondéré : Poids

On considère dans ce cours seulement des graphes orientés.

Définition

Un graphe **pondéré** est un graphe $\vec{G} = (V, \vec{E})$ muni d'une fonction de poids $w : \vec{E} \rightarrow \mathbb{R}$.

$w(u, v)$ est le **poids** de l'arête de u vers v .

Il est pratique de définir $w(u, v) = \infty$ s'il n'y a pas d'arête entre u et v .
En Python, on peut utiliser `float("inf")` pour représenter $+\infty$.

Graphe pondéré : Poids

On considère dans ce cours seulement des graphes orientés.

Définition

Un graphe **pondéré** est un graphe $\vec{G} = (V, \vec{E})$ muni d'une fonction de poids $w : \vec{E} \rightarrow \mathbb{R}$.

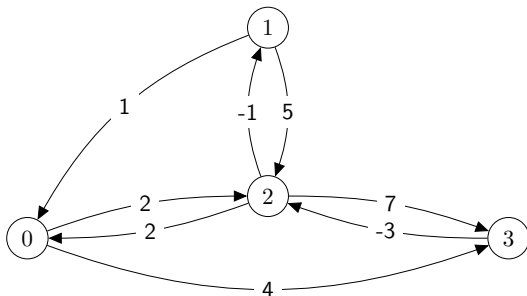
$w(u, v)$ est le **poids** de l'arête de u vers v .

Il est pratique de définir $w(u, v) = \infty$ s'il n'y a pas d'arête entre u et v .
En Python, on peut utiliser `float("inf")` pour représenter $+\infty$.

Pour représenter un graphe pondéré, on utilisera une **matrice d'adjacence pondéré**, contenant $w(u, v)$ sur la ligne u , colonne v .

Graphe pondéré : Poids

Exemple de graphe représenté par matrice d'adjacence pondérée :



$$\begin{pmatrix} 0 & \infty & 2 & 4 \\ 1 & 0 & 5 & \infty \\ 2 & -1 & 0 & 7 \\ \infty & \infty & -3 & 0 \end{pmatrix}$$

```
G = [  
  [0, float("inf"), 2, 4],  
  [1, 0, 5, float("inf")],  
  [2, -1, 0, 7],  
  [float("inf"), float("inf"), -3, 0]  
]
```

Définition

- ① Le **poids d'un chemin** C , noté $w(C)$ est la somme des poids de ses arêtes.
- ② Un chemin de $u \in V$ à $v \in V$ est un **plus court chemin** s'il n'existe pas de chemin de poids plus petit.

Définition

- 1 Le **poids d'un chemin** C , noté $w(C)$ est la somme des poids de ses arêtes.
- 2 Un chemin de $u \in V$ à $v \in V$ est un **plus court chemin** s'il n'existe pas de chemin de poids plus petit.

Exercice

Écrire une fonction `poids_chemin(C, G)` qui calcule le poids d'un chemin C (donné par la liste des ses sommets) dans le graphe représenté par la matrice d'adjacence pondérée G .

Définition

La **distance** $d(u, v)$ est le poids d'un plus court chemin de u à v .
Autrement dit :

$$d(u, v) = \inf\{w(C) \mid C \text{ est un chemin de } u \text{ à } v\}$$

Définition

La **distance** $d(u, v)$ est le poids d'un plus court chemin de u à v .
Autrement dit :

$$d(u, v) = \inf\{w(C) \mid C \text{ est un chemin de } u \text{ à } v\}$$

Il peut ne pas y avoir de plus court chemin de u à v ...

Définition

La **distance** $d(u, v)$ est le poids d'un plus court chemin de u à v .
Autrement dit :

$$d(u, v) = \inf\{w(C) \mid C \text{ est un chemin de } u \text{ à } v\}$$

Il peut ne pas y avoir de plus court chemin de u à v ...

- ... si v n'est pas atteignable depuis u : on pose $d(u, v) = \infty$.

Définition

La **distance** $d(u, v)$ est le poids d'un plus court chemin de u à v .
Autrement dit :

$$d(u, v) = \inf\{w(C) \mid C \text{ est un chemin de } u \text{ à } v\}$$

Il peut ne pas y avoir de plus court chemin de u à v ...

- ... si v n'est pas atteignable depuis u : on pose $d(u, v) = \infty$.
- ... s'il existe un cycle de poids négatif : on pose $d(u, v) = -\infty$.

Inégalité triangulaire

S'il n'y a pas de cycle de poids négatif :

$$d(v_1, v_2) \leq d(v_1, v_3) + d(v_3, v_2)$$

Inégalité triangulaire

S'il n'y a pas de cycle de poids négatif :

$$d(v_1, v_2) \leq d(v_1, v_3) + d(v_3, v_2)$$

Preuve :

Concaténer un plus court chemin de v_1 à v_3 et un plus court chemin de v_3 à v_2 donne un chemin de v_1 à v_2 de poids $d(v_1, v_3) + d(v_3, v_2)$.

Ce poids est donc supérieur au poids $d(v_1, v_2)$ d'un plus court chemin de v_1 à v_2 .

Sous-optimalité

Soit C un plus court chemin de u à v et u' , v' deux sommets de C . Alors le sous-chemin de C de u' à v' est aussi un plus court chemin.

Sous-optimalité

Soit C un plus court chemin de u à v et u' , v' deux sommets de C . Alors le sous-chemin de C de u' à v' est aussi un plus court chemin.

Preuve :

Si ce n'était pas le cas on pourrait le remplacer par un chemin plus court pour obtenir un chemin de u à v plus court que C (absurde).

Plus courts chemins

L'objectif du cours est de résoudre le problème suivant :

Problème

Entrée : $\vec{G} = (V, \vec{E})$ un graphe pondéré orienté et $s \in V$.

Sortie : Une liste contenant $d(s, v)$, pour tout $v \in V$.

Plus courts chemins

L'objectif du cours est de résoudre le problème suivant :

Problème

Entrée : $\vec{G} = (V, \vec{E})$ un graphe pondéré orienté et $s \in V$.

Sortie : Une liste contenant $d(s, v)$, pour tout $v \in V$.

Cas particuliers :

- Tous les poids sont égaux :

Plus courts chemins

L'objectif du cours est de résoudre le problème suivant :

Problème

Entrée : $\vec{G} = (V, \vec{E})$ un graphe pondéré orienté et $s \in V$.

Sortie : Une liste contenant $d(s, v)$, pour tout $v \in V$.

Cas particuliers :

- Tous les poids sont égaux : **parcours en largeur** depuis s .

Plus courts chemins

L'objectif du cours est de résoudre le problème suivant :

Problème

Entrée : $\vec{G} = (V, \vec{E})$ un graphe pondéré orienté et $s \in V$.

Sortie : Une liste contenant $d(s, v)$, pour tout $v \in V$.

Cas particuliers :

- Tous les poids sont égaux : **parcours en largeur** depuis s .
- Tous les poids sont positifs : **algorithme de Dijkstra** depuis s .

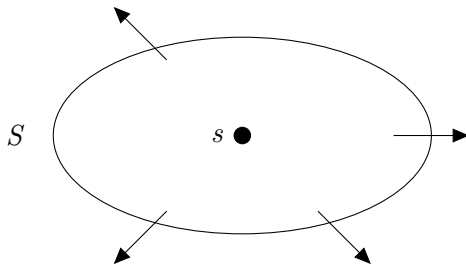
Algorithme de Dijkstra

Idée : Calculer les distances par ordre croissant depuis s .

Algorithme de Dijkstra

Idée : Calculer les distances par ordre croissant depuis s .

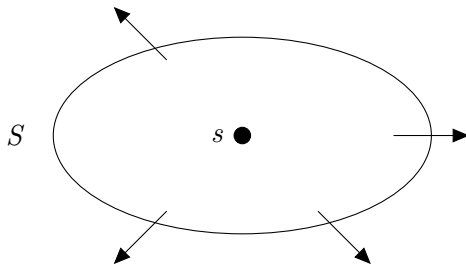
Soit $S \subset V$ l'ensemble des sommets de distance connue.



Algorithme de Dijkstra

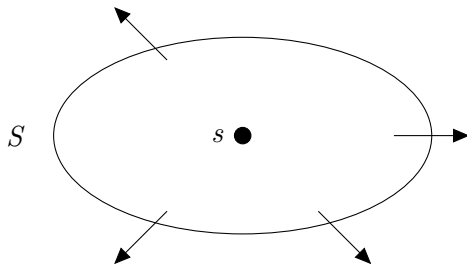
Idée : Calculer les distances par ordre croissant depuis s .

Soit $S \subset V$ l'ensemble des sommets de distance connue.



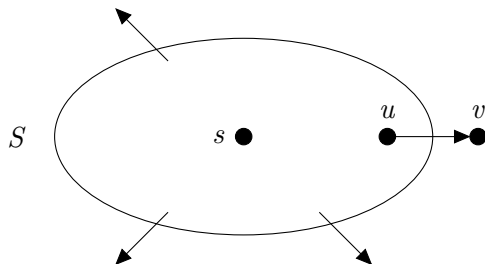
À chaque étape, on déduit la distance à un sommet de plus (et on l'ajoute à S).

Algorithme de Dijkstra



Soit $(u, v) \in \vec{E}$ tel que $v \notin S$ et $d(s, u) + w(u, v)$ est minimum.

Algorithme de Dijkstra

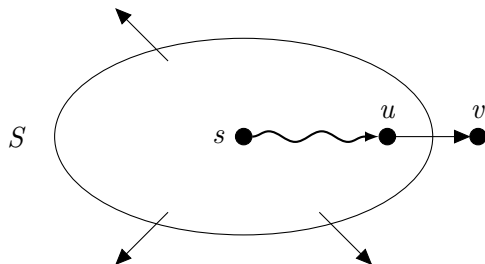


Soit $(u, v) \in \vec{E}$ tel que $v \notin S$ et $d(s, u) + w(u, v)$ est minimum.

Alors :

$$d(s, v) = d(s, u) + w(u, v)$$

Algorithme de Dijkstra



Soit $(u, v) \in \vec{E}$ tel que $v \notin S$ et $d(s, u) + w(u, v)$ est minimum.

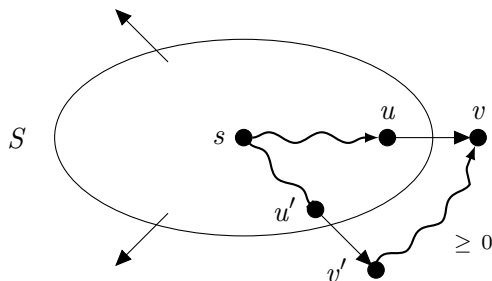
Alors :

$$d(s, v) = d(s, u) + w(u, v)$$

Preuve :

- ❶ Il existe un chemin de longueur $d(s, u) + w(u, v)$.

Algorithme de Dijkstra



Soit $(u, v) \in \vec{E}$ tel que $v \notin S$ et $d(s, u) + w(u, v)$ est minimum.

Alors :

$$d(s, v) = d(s, u) + w(u, v)$$

Preuve :

- ② Un chemin C de s à v doit sortir de S avec un arc (u', v') . Comme les poids sont ≥ 0 :

$$\text{poids}(C) \geq d(s, u') + w(u', v') \geq d(s, u) + w(u, v)$$

Algorithme de Dijkstra

On stocke les sommets restants à visiter dans q ($= \overline{S}$) et on conserve un tableau dist des distances estimées tel que :

$$\textcircled{1} \quad \forall v \notin q : \text{dist}[v] = d(s, v).$$

$$\textcircled{2} \quad \forall v \in q : \text{dist}[v] = \min_{u \notin q} d(s, u) + w(u, v).$$

Algorithme de Dijkstra

On stocke les sommets restants à visiter dans q ($= \overline{S}$) et on conserve un tableau dist des distances estimées tel que :

- ❶ $\forall v \notin q : \text{dist}[v] = d(s, v).$
- ❷ $\forall v \in q : \text{dist}[v] = \min_{u \notin q} d(s, u) + w(u, v).$

Algorithme de Dijkstra :

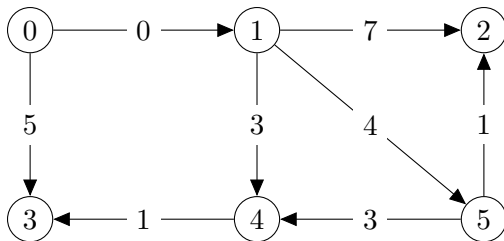
```
Initialement : q contient tous les sommets
                dist[s] = 0
                dist[v] = float("inf"), si v != s
```

```
Tant que q est non vide:
    Extraire u de q tel que dist[u] soit minimum
    Pour tout voisin v de u:
        Si dist[u] + w(u, v) < dist[v]:
            dist[v] = dist[u] + w(u, v)
```

Algorithme de Dijkstra : Exemple

Exercice

Appliquer l'algorithme de Dijkstra depuis $s = 0$ sur le graphe suivant, en mettant $\text{dist}[v]$ à côté de chaque sommet v :



Algorithme de Dijkstra : File de priorité

L'algorithme de Dijkstra demande d'extraire le minimum de q , ce qui peut être réalisé efficacement avec une file de priorité :

Définition

Une **file de priorité** est une structure de données dont chaque élément possède une **priorité** et avec les opérations suivantes :

- Ajout d'un élément avec sa priorité.
- Extraction de l'élément de priorité minimum.
- (Éventuellement) mise à jour de la priorité d'un élément.

Algorithme de Dijkstra : File de priorité

L'algorithme de Dijkstra demande d'extraire le minimum de q , ce qui peut être réalisé efficacement avec une file de priorité :

Définition

Une **file de priorité** est une structure de données dont chaque élément possède une **priorité** et avec les opérations suivantes :

- Ajout d'un élément avec sa priorité.
- Extraction de l'élément de priorité minimum.
- (Éventuellement) mise à jour de la priorité d'un élément.

Il existe plusieurs implémentation de file de priorité, notamment par arbre binaire de recherche ou tas. [Voir le cours de MP2I pour plus de détails.](#)

Algorithme de Dijkstra : File de priorité

L'algorithme de Dijkstra demande d'extraire le minimum de q , ce qui peut être réalisé efficacement avec une file de priorité :

Définition

Une **file de priorité** est une structure de données dont chaque élément possède une **priorité** et avec les opérations suivantes :

- Ajout d'un élément avec sa priorité.
- Extraction de l'élément de priorité minimum.
- (Éventuellement) mise à jour de la priorité d'un élément.

Il existe plusieurs implémentation de file de priorité, notamment par arbre binaire de recherche ou tas. [Voir le cours de MP2I pour plus de détails.](#)

En Python, on peut utiliser [heapq](#) (qui implémente une file de priorité par tas). Mais `heapq` ne permet pas de mettre à jour une priorité... À la place j'ai implémenté une structure de file de priorité très simple.

Algorithme de Dijkstra : File de priorité

Utilisation de mon implémentation de file de priorité (il faut préalablement installer le package cpge : `pip install cpge`) :

```
from cpge import PriorityQueue

q = PriorityQueue() # file de priorité vide
q.add(2, 7) # ajoute l'élément 2 avec la priorité 7
q.add(5, 2)
q.add(3, 9)
q.take_min() # renvoie 5
q.take_min() # renvoie 2
```

Algorithme de Dijkstra : Implémentation

```
def dijkstra(G, s):
    n = len(G)
    dist = n*[float("inf")]
    dist[s] = 0
    q = PriorityQueue()
    for v in range(n):
        q.add(v, dist[v])

    while not q.is_empty():
        u = q.take_min()
        for v in range(n):
            d = dist[u] + G[u][v]
            if v in q and d < dist[v]:
                q.update(v, d)
                dist[v] = d

    return dist
```
