Informatique tronc commun - PCSI

# Cours 7 : Recherche de plus court chemin

Florent Pompigne
pompigne@crans.org

Lycée Buffon

année 2021/2022

# Objectif

Nous avons vu qu'un parcours en largeur permet d'obtenir les distances et plus courts chemins à partir d'un sommet $s$ dans un graphe. L'objectif de ce chapitre est d'adapter cette approche au cas d'un graphe **pondéré**, ainsi qu'au cas où on vise un sommet destination fixé.

# Objectif

Nous avons vu qu'un parcours en largeur permet d'obtenir les distances et plus courts chemins à partir d'un sommet $s$ dans un graphe. L'objectif de ce chapitre est d'adapter cette approche au cas d'un graphe **pondéré**, ainsi qu'au cas où on vise un sommet destination fixé.

Dans la suite, nous considérerons donc un graphe pondéré, dont les poids sont **positifs**.

# Objectif

Nous avons vu qu'un parcours en largeur permet d'obtenir les distances et plus courts chemins à partir d'un sommet $s$ dans un graphe. L'objectif de ce chapitre est d'adapter cette approche au cas d'un graphe **pondéré**, ainsi qu'au cas où on vise un sommet destination fixé.

Dans la suite, nous considérerons donc un graphe pondéré, dont les poids sont **positifs**.

## Définition

# Objectif

Nous avons vu qu'un parcours en largeur permet d'obtenir les distances et plus courts chemins à partir d'un sommet $s$ dans un graphe. L'objectif de ce chapitre est d'adapter cette approche au cas d'un graphe **pondéré**, ainsi qu'au cas où on vise un sommet destination fixé.

Dans la suite, nous considérerons donc un graphe pondéré, dont les poids sont **positifs**.

### Définition

Le **poids** d'un chemin est la somme des poids des arcs (ou arêtes) le constituant.

# Objectif

Nous avons vu qu'un parcours en largeur permet d'obtenir les distances et plus courts chemins à partir d'un sommet $s$ dans un graphe. L'objectif de ce chapitre est d'adapter cette approche au cas d'un graphe **pondéré**, ainsi qu'au cas où on vise un sommet destination fixé.

Dans la suite, nous considérerons donc un graphe pondéré, dont les poids sont **positifs**.

## Définition

Le **poids** d'un chemin est la somme des poids des arcs (ou arêtes) le constituant.

un chemin de $u$ à $v$ est **minimal** s'il est de poids minimal parmi les chemins de $u$ à $v$.

# Objectif

Nous avons vu qu'un parcours en largeur permet d'obtenir les distances et plus courts chemins à partir d'un sommet $s$ dans un graphe. L'objectif de ce chapitre est d'adapter cette approche au cas d'un graphe **pondéré**, ainsi qu'au cas où on vise un sommet destination fixé.

Dans la suite, nous considérerons donc un graphe pondéré, dont les poids sont **positifs**.

## Définition

Le **poids** d'un chemin est la somme des poids des arcs (ou arêtes) le constituant.

un chemin de $u$ à $v$ est **minimal** s'il est de poids minimal parmi les chemins de $u$ à $v$.

La **distance** d'un sommet $u$ à un sommet $v$ dans un graphe pondéré est le poids d'un chemin minimal de $u$ à $v$, ou $+\infty$ s'il n'en existe pas. On notera cette quantité $\delta(u, v)$.

# Relâchement

Un algorithme de recherche de plus courts chemins cherche à remplir une liste de distances D et une liste de pères P. Pour cela, il se base généralement sur des relâchements :

### Définition

Lors du **relâchement** de l'arc $(u, v)$ de poids $w$ :
si $D[v] > D[u] + w$,
alors on remplace $D[v]$ par $D[u] + w$ et $P[v]$ par $u$.

# Relâchement

Un algorithme de recherche de plus courts chemins cherche à remplir une liste de distances D et une liste de pères P. Pour cela, il se base généralement sur des relâchements :

### Définition

Lors du **relâchement** de l'arc $(u, v)$ de poids $w$ :
si $D[v] > D[u] + w$,
alors on remplace $D[v]$ par $D[u] + w$ et $P[v]$ par $u$.

### Propriétés

- Lors du relâchement de $(u, v)$, $D[v]$ ne peut que diminuer
- Un relâchement préserve la propriété :

$$\forall u \in S,\ \delta(s, u) \leq D[u]$$

# Algorithme de Dijkstra

```
Dijkstra(G,s):
    D[s] vaut 0
    D[u] est infinie pour les autres sommets u
    Les sommets n'ont initialement pas de père
    on crée une liste L de tous les sommets
    tant que L est non vide:
        on sort de L le sommet u de D[u] minimal
        pour chaque successeur v de u:
            on relâche l'arc (u,v)
    on renvoie D et P
```

# Algorithme de Dijkstra

```
Dijkstra(G,s):
    D[s] vaut 0
    D[u] est infinie pour les autres sommets u
    Les sommets n'ont initialement pas de père
    on crée une liste L de tous les sommets
    tant que L est non vide:
        on sort de L le sommet u de D[u] minimal
        pour chaque successeur v de u:
            on relâche l'arc (u,v)
    on renvoie D et P
```

## Complexité

Chaque arc va être relâché une fois, pour un coût umulé en $m \times O(1) = O(m)$ ($m = \mathrm{card}(A)$). Chaque recherche du minimum de $L$ est de complexité en $O(n)$, pour un coût cumulé en $n \times O(n) = O(n^2)$ ($n = \mathrm{card}(S)$).
La complexité totale est donc en $O(m + n^2) = O(n^2)$.

# Algorithme A*

L'algorithme de Dijkstra permet de trouver *toutes* les distances et plus courts chemins depuis un sommet source $s$. Lorsqu'on s'intéresse à un unique sommet destination, il est possible d'adapter l'algorithme pour orienter la recherche et donc réduire le temps de calcul.

# Algorithme A*

L'algorithme de Dijkstra permet de trouver *toutes* les distances et plus courts chemins depuis un sommet source $s$. Lorsqu'on s'intéresse à un unique sommet destination, il est possible d'adapter l'algorithme pour orienter la recherche et donc réduire le temps de calcul.

L'algorithme $A^*$ utilise pour cela une fonction **heuristique** donnant une estimation *a priori* de la distance entre un sommet $u$ et le sommet destination $d$. Par exemple, si les sommets ont des positions spatiales, l'heuristique peut calculer la distance à vol d'oiseau.

# Algorithme A*

L'algorithme de Dijkstra permet de trouver *toutes* les distances et plus courts chemins depuis un sommet source $s$. Lorsqu'on s'intéresse à un unique sommet destination, il est possible d'adapter l'algorithme pour orienter la recherche et donc réduire le temps de calcul.

L'algorithme $A^*$ utilise pour cela une fonction **heuristique** donnant une estimation *a priori* de la distance entre un sommet $u$ et le sommet destination $d$. Par exemple, si les sommets ont des positions spatiales, l'heuristique peut calculer la distance à vol d'oiseau.

La différence principale avec Dijkstra est qu'à chaque étape, c'est le sommet u qui minimise D[u] + h(u,d) est qui sorti de L : les sommets proches de la destination sont priorisés.

# Algorithme A*

```
A_etoile(G,s,d,h):
    D[s] vaut 0
    D[u] est infinie pour les autres sommets u
    Les sommets n'ont initialement pas de père
    on crée une liste L contenant seulement s
    tant que L est non vide:
        Soit u dans L de D[u] + h(u,d) minimal
        on sort u de L
        si u = d :
            on s'arrête en renvoyant D[d] et P
        sinon, pour chaque successeur v de u:
            on relâche (u,v)
            si on a diminué la distance de v
            et que v n'est pas dans L :
                on l'y ajoute
    on renvoie "d n'est pas accessible depuis s"
```

# Choix de l'heuristique

- Si on prend une heuristique qui renvoie toujours 0, alors l'algorithme $A^*$ se comporte comme l'algorithme de Dijkstra, en s'arrêtant dès que $d$ est atteint.

# Choix de l’heuristique

- Si on prend une heuristique qui renvoie toujours 0, alors l’algorithme $A^*$ se comporte comme l’algorithme de Dijkstra, en s’arrêtant dès que $d$ est atteint.
- Si on prend une heuristique qui ne surestime jamais la distance de $u$ à $d$, ie qui vérifie $\forall u \in S, h(u, d) \leq \delta(u, d)$, alors $A^*$ est en général plus rapide que Dijkstra pour atteindre $d$, tout en trouvant bien un chemin minimal.

# Choix de l'heuristique

- Si on prend une heuristique qui renvoie toujours 0, alors l'algorithme $A^*$ se comporte comme l'algorithme de Dijkstra, en s'arrêtant dès que $d$ est atteint.
- Si on prend une heuristique qui ne surestime jamais la distance de $u$ à $d$, ie qui vérifie $\forall u \in S, h(u,d) \leq \delta(u,d)$, alors $A^*$ est en général plus rapide que Dijkstra pour atteindre $d$, tout en trouvant bien un chemin minimal.
- Si on prend une heuristique qui peut surestimer la distance de $u$ à $d$ (par exemple : deux fois la distance à vol d'oiseau), cela peut encore accélérer la vitesse à laquelle $A^*$ trouve $d$, mais on n'a en revanche plus la garantie que le chemin et la distance calculés sont minimaux.