

/tikz/,/tikz/graphs/
conversions/canvas coordinate/.code=1 ,
conversions/coordinate/.code=1
force, layered, circular

Plus courts chemins dans les graphes pondérés

MP/MP* Option info

Graphe pondéré

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

Graphe pondéré

Un graphe **pondéré** est un graphe $G = (V, E)$ muni d'une fonction de poids $w : E \rightarrow \mathbb{R}$ (en Caml, `w : int -> int -> int` ou `w : int array array`).

Graphe pondéré

Un graphe **pondéré** est un graphe $G = (V, E)$ muni d'une fonction de poids $w : E \rightarrow \mathbb{R}$ (en Caml, `w : int -> int -> int` ou `w : int array array`).

Le **poids d'un chemin** est la somme des poids de ses arêtes.

Graphe pondéré

Un graphe **pondéré** est un graphe $G = (V, E)$ muni d'une fonction de poids $w : E \rightarrow \mathbb{R}$ (en Caml, `w : int -> int -> int` ou `w : int array array`).

Le **poids d'un chemin** est la somme des poids de ses arêtes.

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.

Graphe pondéré

Un graphe **pondéré** est un graphe $G = (V, E)$ muni d'une fonction de poids $w : E \rightarrow \mathbb{R}$ (en Caml, `w : int -> int -> int` ou `w : int array array`).

Le **poids d'un chemin** est la somme des poids de ses arêtes.

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.

La **distance** $d(u, v)$ de u à v est le poids d'un plus court chemin de u à v , s'il en existe un.

Distance

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

Distance

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

- ① ... si v n'est pas atteignable depuis u , on pose $d(u, v) = \infty$.

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

- ① ... si v n'est pas atteignable depuis u , on pose $d(u, v) = \infty$.
- ② ... s'il existe un cycle de poids négatif, on pose $d(u, v) = -\infty$.

Remarque: s'il n'y a pas de cycle de poids ≤ 0 , les plus courts chemins sont élémentaires (ils passent au plus une fois sur un sommet) donc sont de longueur au plus $V - 1$.

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:

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

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.

Inégalité triangulaire

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

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

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.

L'optimalité des sous-problèmes est cruciale pour montrer que certains algorithmes/raisonnements sont corrects, par exemple:

- ❶ pour utiliser la programmation dynamique / diviser pour régner.
- ❷ un sous-arbre d'un ABR est un ABR.
- ❸ un sous-arbre d'un tas est un tas.

Pensez à le mentionner et le justifier si besoin est...

Problèmes de plus courts chemins

Pour trouver des plus courts chemins...

- 1 ...depuis un sommet à tous les autres quand tous les **poids** sont **égaux** et positifs:

Problèmes de plus courts chemins

Pour trouver des plus courts chemins...

- ① ...depuis un sommet à tous les autres quand tous les **poids** sont **égaux** et positifs: BFS en $O(V + E)$
- ② ...depuis un sommet à tous les autres quand il n'y a **pas de cycle**:

Problèmes de plus courts chemins

Pour trouver des plus courts chemins...

- ① ...depuis un sommet à tous les autres quand tous les **poids** sont **égaux** et positifs: BFS en $O(V + E)$
- ② ...depuis un sommet à tous les autres quand il n'y a **pas de cycle**: tri topologique + prog. dyn. en $O(V + E)$
- ③ ...depuis un sommet à tous les autres quand les **poids** sont **positifs**:

Problèmes de plus courts chemins

Pour trouver des plus courts chemins...

- ① ...depuis un sommet à tous les autres quand tous les **poids** sont **égaux** et positifs: BFS en $O(V + E)$
- ② ...depuis un sommet à tous les autres quand il n'y a **pas de cycle**: tri topologique + prog. dyn. en $O(V + E)$
- ③ ...depuis un sommet à tous les autres quand les **poids** sont **positifs**: Dijkstra en $O(V \log V)$
- ④ ...depuis un sommet à tous les autres (et détecter un cycle de poids négatif):

Problèmes de plus courts chemins

Pour trouver des plus courts chemins...

- ① ...depuis un sommet à tous les autres quand tous les **poids** sont **égaux** et positifs: BFS en $O(V + E)$
- ② ...depuis un sommet à tous les autres quand il n'y a **pas de cycle**: tri topologique + prog. dyn. en $O(V + E)$
- ③ ...depuis un sommet à tous les autres quand les **poids** sont **positifs**: Dijkstra en $O(V \log V)$
- ④ ...depuis un sommet à tous les autres (et détecter un cycle de poids négatif): Bellman-Ford (prog. dyn.) en $O(V^2)$
- ⑤ ...**entre tout couple de sommets** (et détecter un cycle de poids négatif):

Problèmes de plus courts chemins

Pour trouver des plus courts chemins...

- 1 ...depuis un sommet à tous les autres quand tous les **poids** sont **égaux** et positifs: BFS en $O(V + E)$
- 2 ...depuis un sommet à tous les autres quand il n'y a **pas de cycle**: tri topologique + prog. dyn. en $O(V + E)$
- 3 ...depuis un sommet à tous les autres quand les **poids** sont **positifs**: Dijkstra en $O(V \log V)$
- 4 ...depuis un sommet à tous les autres (et détecter un cycle de poids négatif): Bellman-Ford (prog. dyn.) en $O(V^2)$
- 5 ...**entre tout couple de sommets** (et détecter un cycle de poids négatif): Floyd-Warshall (prog. dyn.) en $O(V^3)$.

Graphe non-orienté

Nous allons écrire les algorithmes de plus courts chemins pour des graphes orientés, mais ils fonctionnent aussi pour des graphes non-orientés, **à condition que tous les poids soient positifs.**

Graphe non-orienté

Nous allons écrire les algorithmes de plus courts chemins pour des graphes orientés, mais ils fonctionnent aussi pour des graphes non-orientés, **à condition que tous les poids soient positifs.**

Soit G un graphe non-orienté et \vec{G} obtenu à partir de G en remplaçant chaque arête u, v par deux arcs de même poids (u, v) et (v, u) :

Graphe non-orienté

Nous allons écrire les algorithmes de plus courts chemins pour des graphes orientés, mais ils fonctionnent aussi pour des graphes non-orientés, **à condition que tous les poids soient positifs**.

Soit G un graphe non-orienté et \vec{G} obtenu à partir de G en remplaçant chaque arête u, v par deux arcs de même poids (u, v) et (v, u) :

- 1 Si les poids de G sont ≥ 0 , les distances entre sommets sont les mêmes dans G et \vec{G} .
- 2 G et \vec{G} ont même matrice d'adjacence et même liste d'adjacence.

Graphe non-orienté

Nous allons écrire les algorithmes de plus courts chemins pour des graphes orientés, mais ils fonctionnent aussi pour des graphes non-orientés, **à condition que tous les poids soient positifs**.

Soit G un graphe non-orienté et \vec{G} obtenu à partir de G en remplaçant chaque arête u, v par deux arcs de même poids (u, v) et (v, u) :

- 1 Si les poids de G sont ≥ 0 , les distances entre sommets sont les mêmes dans G et \vec{G} .
- 2 G et \vec{G} ont même matrice d'adjacence et même liste d'adjacence.

Pourquoi cela ne marche-t-il pas s'il y a des poids négatifs?

Algorithme de Dijkstra

But: calculer les distances d'un sommet r aux autres, où les poids sont ≥ 0 .

Algorithme de Dijkstra

But: calculer les distances d'un sommet r aux autres, où les poids sont ≥ 0 .

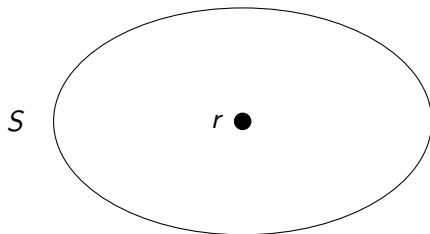
Idée: calculer les distances par ordre croissant, en partant de r .

Algorithme de Dijkstra

But: calculer les distances d'un sommet r aux autres, où les poids sont ≥ 0 .

Idée: calculer les distances par ordre croissant, en partant de r .

Supposons connaître les distances de r à tous les sommets de S .

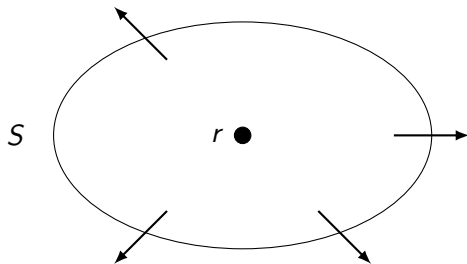


Algorithme de Dijkstra

But: calculer les distances d'un sommet r aux autres, où les poids sont ≥ 0 .

Idée: calculer les distances par ordre croissant, en partant de r .

Supposons connaître les distances de r à tous les sommets de S .

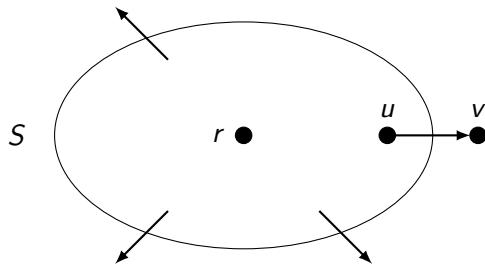


Algorithme de Dijkstra

But: calculer les distances d'un sommet r aux autres, où les poids sont ≥ 0 .

Idée: calculer les distances par ordre croissant, en partant de r .

Supposons connaître les distances de r à tous les sommets de S .



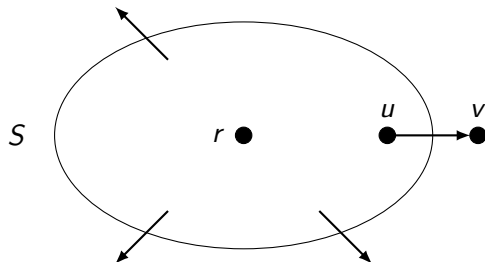
Soit (u, v) sortant de S tel que $d(r, u) + w(u, v)$ soit **minimum**.

Algorithme de Dijkstra

But: calculer les distances d'un sommet r aux autres, où les poids sont ≥ 0 .

Idée: calculer les distances par ordre croissant, en partant de r .

Supposons connaître les distances de r à tous les sommets de S .



Soit (u, v) sortant de S tel que $\mathbf{d(r, u) + w(u, v)}$ soit **minimum**.

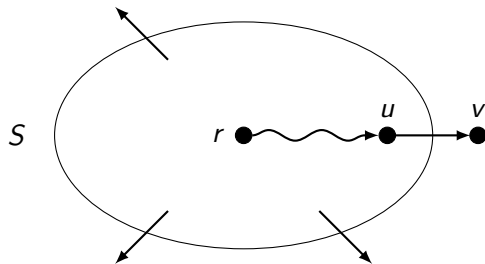
Alors, si tous les poids sont ≥ 0 : $\boxed{d(r, v) = d(r, u) + w(u, v)}$.

Algorithme de Dijkstra

But: calculer les distances d'un sommet r aux autres, où les poids sont ≥ 0 .

Idée: calculer les distances par ordre croissant, en partant de r .

Supposons connaître les distances de r à tous les sommets de S .



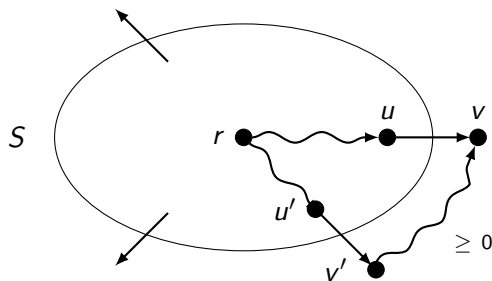
En effet: 1) Il existe un chemin de longueur $d(r, u) + w(u, v)$.

Algorithme de Dijkstra

But: calculer les distances d'un sommet r aux autres, où les poids sont ≥ 0 .

Idée: calculer les distances par ordre croissant, en partant de r .

Supposons connaître les distances de r à tous les sommets de S .



En effet: 1) Il existe un chemin de longueur $d(r, u) + w(u, v)$.

2) Un chemin C de r à v doit sortir de S avec un arc (u', v') . Comme les poids sont ≥ 0 , $\text{poids}(C) \geq d(r, u') + w(u', v') + w(v', v) \geq d(r, u) + w(u, v)$.

Algorithme de Dijkstra

On stocke les sommets restants à visiter dans `next` ($= \bar{S}$) et on conserve un tableau `dist` tel que:

- 1 $\forall v \notin \text{next}: \text{dist.}(v) = d(r, v).$
- 2 $\forall v \in \text{next}: \text{dist.}(v) = \min_{u \notin \text{next}} (d(r, u) + w(u, v)).$

Algorithme de Dijkstra

On stocke les sommets restants à visiter dans `next` ($= \bar{S}$) et on conserve un tableau `dist` tel que:

- 1 $\forall v \notin \text{next}: \text{dist.}(v) = d(r, v).$
- 2 $\forall v \in \text{next}: \text{dist.}(v) = \min_{u \notin \text{next}} (d(r, u) + w(u, v)).$

Initialement: `next` contient tous les sommets

`dist.(r) <- 0` et `dist.(v) <- ∞` , $\forall v \neq r$.

Tant que `next` $\neq \emptyset$:

Extraire `u` de `next` tel que `dist.(u)` soit minimum

Pour tout voisin `v` de `u`:

`dist.(v) <- min dist.(v) (dist.(u) + w u v)`

Algorithme de Dijkstra

On stocke les sommets restants à visiter dans `next` ($= \bar{S}$) et on conserve un tableau `dist` tel que:

- 1 $\forall v \notin \text{next}: \text{dist.}(v) = d(r, v).$
- 2 $\forall v \in \text{next}: \text{dist.}(v) = \min_{u \notin \text{next}} (d(r, u) + w(u, v)).$

Initialement: `next` contient tous les sommets

`dist.(r) <- 0` et `dist.(v) <- ∞` , $\forall v \neq r$.

Tant que `next` $\neq \emptyset$:

Extraire `u` de `next` tel que `dist.(u)` soit minimum

Pour tout voisin `v` de `u`:

`dist.(v) <- min dist.(v) (dist.(u) + w u v)`

Remarque: ne marche pas avec des poids négatifs.

Algorithme de Dijkstra

Initialement: `next` contient tous les sommets

`dist.(r) <- 0` et `dist.(v) <- ∞` , $\forall v \neq r$

Tant que `next` $\neq \emptyset$:

Extraire `u` de `next` tel que `dist.(u)` soit minimum

Pour tout voisin `v` de `u`:

`dist.(v) <- min dist.(v) (dist.(u) + w u v)`

Complexité

Algorithme de Dijkstra

Initialement: `next` contient tous les sommets

`dist.(r) <- 0` et `dist.(v) <- ∞` , $\forall v \neq r$

Tant que `next` $\neq \emptyset$:

Extraire `u` de `next` tel que `dist.(u)` soit minimum

Pour tout voisin `v` de `u`:

`dist.(v) <- min dist.(v) (dist.(u) + w_{uv})`

Complexité si `next` est un **tableau de booléens**:

Algorithme de Dijkstra

Initialement: next contient tous les sommets

$\text{dist.}(r) \leftarrow 0$ et $\text{dist.}(v) \leftarrow \infty, \forall v \neq r$

Tant que next $\neq \emptyset$:

Extraire u de next tel que $\text{dist.}(u)$ soit minimum

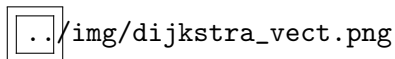
Pour tout voisin v de u:

$\text{dist.}(v) \leftarrow \min(\text{dist.}(v), (\text{dist.}(u) + w_{uv}))$

Complexité si next est un **tableau de booléens**:

chaque recherche de minimum est en $O(V)$ donc la complexité totale est $O(V^2)$.

Implémentation de Dijkstra avec un tableau



`w : int -> int -> int` est la fonction de poids.

Implémentation de Dijkstra avec une file de priorité

Initialement: next contient tous les sommets

$\text{dist.}(r) \leftarrow 0$ et $\text{dist.}(v) \leftarrow \infty, \forall v \neq r$

Tant que $\text{next} \neq \emptyset$:

Extraire u de next tel que $\text{dist.}(u)$ soit minimum

Pour tout voisin v de u :

$\text{dist.}(v) \leftarrow \min \text{dist.}(v) \text{ (dist.}(u) + w_{uv})$

Complexité si next est une **file de priorité min**:

Implémentation de Dijkstra avec une file de priorité

Initialement: next contient tous les sommets

$\text{dist.}(r) \leftarrow 0$ et $\text{dist.}(v) \leftarrow \infty, \forall v \neq r$

Tant que $\text{next} \neq \emptyset$:

Extraire u de next tel que $\text{dist.}(u)$ soit minimum

Pour tout voisin v de u :

$\text{dist.}(v) \leftarrow \min \text{dist.}(v) \text{ (dist.}(u) + w_{uv})$

Complexité si next est une **file de priorité min**:

① V extractions du minimum: $O(V \log(V))$

② au plus, mises à jour: $O(\log(V))$

Total: $O(V \log(V)) + O(\log(V)) = \boxed{O(V \log(V))}$.

Implémentation de Dijkstra avec une file de priorité

Initialement: `next` contient tous les sommets

`dist.(r) <- 0` et `dist.(v) <- ∞ , $\forall v \neq r$`

Tant que `next $\neq \emptyset$` :

Extraire `u` de `next` tel que `dist.(u)` soit minimum

Pour tout voisin `v` de `u`:

`dist.(v) <- min dist.(v) (dist.(u) + w u v)`

Complexité si `next` est une **file de priorité min**:

① V extractions du minimum: $O(V \log(V))$

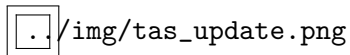
② au plus, mises à jour: $O(\log(V))$

Total: $O(V \log(V)) + O(\log(V)) = \boxed{O(\log(V))}$.

C'est mieux qu'avec un tableau si $, = O\left(\frac{V^2}{\log(V)}\right)$.

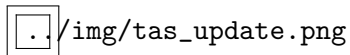
Implémentation de Dijkstra avec une file de priorité

Problème: la fonction de mise à jour d'un élément dans un tas demande de connaître l'indice de l'élément à modifier.



Implémentation de Dijkstra avec une file de priorité

Problème: la fonction de mise à jour d'un élément dans un tas demande de connaître l'indice de l'élément à modifier.



Il faudrait maintenir un tableau qui donne l'indice (dans le tas) d'un sommet. C'est fastidieux (cf infollg)...

On pourrait utiliser un ABR équilibré: pour mettre à jour il suffit d'appeler `del` puis `add`.

Implémentation de Dijkstra avec une file de priorité

Ma solution jj personnelle ll plus simple: ajouter des couples (distance estimée de v , v) sans jamais mettre à jour les éléments de la FP. On peut donc avoir plusieurs fois le même sommet dans la FP:

Implémentation de Dijkstra avec une file de priorité

Ma solution j'ai personnellement trouvée plus simple: ajouter des couples (distance estimée de r , v) sans jamais mettre à jour les éléments de la FP. On peut donc avoir plusieurs fois le même sommet dans la FP:

Initialement: fp contient $(0, r)$

$\text{dist.}(r) \leftarrow 0$ et $\text{dist.}(v) \leftarrow \infty, \forall v \neq r$

Tant que $\text{fp} \neq \emptyset$:

Extraire (d, u) de fp tel que d soit minimum

Si $\text{dist.}(u) = \infty$:

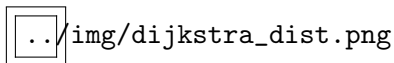
$\text{dist.}(u) \leftarrow d$ (* d est la distance de r à u *)

Pour tout voisin v de u :

Ajouter $(d + w(u, v), v)$ à fp

Implémentation de Dijkstra avec une file de priorité

Ma solution *à* personnelle *est* plus simple: ajouter des couples (distance estimée de v , v) sans jamais mettre à jour les éléments de la FP. On peut donc avoir plusieurs fois le même sommet dans la FP.



Plus courts chemins avec Dijkstra

Initialement: `next` contient tous les sommets

`dist.(r) <- 0` et `dist.(v) <- ∞` , $\forall v \neq r$

Tant que `next` $\neq \emptyset$:

Extraire `u` de `next` tel que `dist.(u)` soit minimum

Pour tout voisin `v` de `u`:

`dist.(v) <- min dist.(v) (dist.(u) + w u v)`

Question

Comment modifier l'algorithme pour connaître les plus courts chemins?

Plus courts chemins avec Dijkstra

Tant que $\text{next} \neq \emptyset$:

Extraire u de next tel que $\text{dist.}(u)$ soit minimum

Pour tout voisin v de u :

Si $\text{dist.}(v) > \text{dist.}(u) + w_{uv}$:

$\text{dist.}(v) \leftarrow \text{dist.}(u) + w_{uv}$

$\text{pere.}(v) \leftarrow u$

On peut conserver dans $\text{pere.}(v)$ le prédécesseur de v dans un plus court chemin de r à v .

$v, \text{pere.}(v), \text{pere.}(\text{pere.}(v)), \dots$ jusqu'à r donne un chemin (à l'envers) de r à v .

Plus courts chemins avec Dijkstra

Tant que $\text{next} \neq \emptyset$:

Extraire u de next tel que $\text{dist.}(u)$ soit minimum

Pour tout voisin v de u :

Si $\text{dist.}(v) > \text{dist.}(u) + w_{uv}$:

$\text{dist.}(v) \leftarrow \text{dist.}(u) + w_{uv}$

$\text{pere.}(v) \leftarrow u$


On peut conserver dans $\text{pere.}(v)$ le prédécesseur de v dans un plus court chemin de r à v .

$v, \text{pere.}(v), \text{pere.}(\text{pere.}(v)), \dots$ jusqu'à r donne un chemin (à l'envers) de r à v .

Le graphe des pères est un arbre (**un arbre des plus courts chemins**), non unique a priori.

Plus courts chemins avec Dijkstra

On stocke des triplets (distance estimé de v , v , père de v) dans la FP:

../img/dijkstra_pere.png

(On n'a plus besoin de dist)

Plus courts chemins entre toutes paires de sommets

On veut connaître les plus courts chemins entre toutes les paires de sommets dans un graphe $G = (V, \vec{E})$.

Plus courts chemins entre toutes paires de sommets

On veut connaître les plus courts chemins entre toutes les paires de sommets dans un graphe $G = (V, \vec{E})$.

- ① Si tous les poids sont ≥ 0 : on peut utiliser Dijkstra depuis chaque sommet, en $O(V \times \log(V))$.

Plus courts chemins entre toutes paires de sommets

On veut connaître les plus courts chemins entre toutes les paires de sommets dans un graphe $G = (V, \vec{E})$.

- ① Si tous les poids sont ≥ 0 : on peut utiliser Dijkstra depuis chaque sommet, en $O(V \times \log(V))$.
- ② Si les poids sont quelconques: on va voir l'algorithme de **Floyd-Warshall** par programmation dynamique en $O(V^3)$.

Programmation dynamique

Pour pouvoir résoudre un problème P par programmation dynamique:

- ❶ S'intéresser plus généralement à $P(k)$, où k est un paramètre et $P(n) = P$.
- ❷ Montrer comment résoudre le cas de base (souvent $P(0)$).
- ❸ Montrer que la résolution de $P(k)$ se ramène à $P(k')$, $k' < k$ (trouver une équation de récurrence).

Programmation dynamique

Pour pouvoir résoudre un problème P par programmation dynamique:

- ❶ S'intéresser plus généralement à $P(k)$, où k est un paramètre et $P(n) = P$.
- ❷ Montrer comment résoudre le cas de base (souvent $P(0)$).
- ❸ Montrer que la résolution de $P(k)$ se ramène à $P(k')$, $k' < k$ (trouver une équation de récurrence).

Pour implémenter la méthode, stocker les solutions des $P(k)$ dans un tableau et calculer les $P(k)$ de proche en proche:

```
for k = ... to ... do p.(k) <- ...
```

Programmation dynamique

Notre problème est $P = \{ \text{trouver } d(u, v), \text{ pour tous sommets } u, v \}$.

u ●

● v

Programmation dynamique

Notre problème est $P = \{ \text{trouver } d(u, v), \text{ pour tous sommets } u, v \}$.



Programmation dynamique

Notre problème est $P = \{ \text{trouver } d(u, v), \text{ pour tous sommets } u, v \}$.



On peut écrire l'équation:

$$d(u, v) = \min_{v'} (d(u, v') + w(v', v))$$

Programmation dynamique

Notre problème est $P = \{ \text{trouver } d(u, v), \text{ pour tous sommets } u, v \}$.



On peut écrire l'équation:

$$d(u, v) = \min_{v'} (d(u, v') + w(v', u))$$

Mais ça ne permet pas de faire de la programmation dynamique: on ne se ramène pas à des sous-problèmes $\{ \text{plus petits} \}$!

Programmation dynamique: Floyd-Warshall

Soit $d_k(u, v)$ la longueur d'un plus court chemin de u à v n'utilisant que des sommets intermédiaires de numéro $< k$

Programmation dynamique: Floyd-Warshall

Soit $d_k(u, v)$ la longueur d'un plus court chemin de u à v n'utilisant que des sommets intermédiaires de numéro $< k$ (∞ s'il n'y a pas de chemin).

Programmation dynamique: Floyd-Warshall

Soit $d_k(u, v)$ la longueur d'un plus court chemin de u à v n'utilisant que des sommets intermédiaires de numéro $< k$ (∞ s'il n'y a pas de chemin).

On va calculer $P(k) = \text{jj trouver } d_k(u, v), \text{ pour tous sommets } u, v \text{ jj}$.

Programmation dynamique: Floyd-Warshall

Soit $d_k(u, v)$ la longueur d'un plus court chemin de u à v n'utilisant que des sommets intermédiaires de numéro $< k$ (∞ s'il n'y a pas de chemin).

On va calculer $P(k) = \{ \{u, v\} \mid \text{trouver } d_k(u, v), \text{ pour tous sommets } u, v \}$.

Soit C un plus court chemin de u à v n'utilisant que des sommets intermédiaires de numéro $< k + 1$.

Programmation dynamique: Floyd-Warshall

Soit $d_k(u, v)$ la longueur d'un plus court chemin de u à v n'utilisant que des sommets intermédiaires de numéro $< k$ (∞ s'il n'y a pas de chemin).

On va calculer $P(k) = \{ \{ u, v \} \mid \text{trouver } d_k(u, v), \text{ pour tous sommets } u, v \}$.

Soit C un plus court chemin de u à v n'utilisant que des sommets intermédiaires de numéro $< k + 1$.

- 1 Si C n'utilise pas k comme sommet intermédiaire:

Programmation dynamique: Floyd-Warshall

Soit $d_k(u, v)$ la longueur d'un plus court chemin de u à v n'utilisant que des sommets intermédiaires de numéro $< k$ (∞ s'il n'y a pas de chemin).

On va calculer $P(k) = \forall u, v \text{ trouver } d_k(u, v)$, pour tous sommets u, v .

Soit C un plus court chemin de u à v n'utilisant que des sommets intermédiaires de numéro $< k + 1$.

- 1 Si C n'utilise pas k comme sommet intermédiaire:

$$d_{k+1}(u, v) = d_k(u, v)$$

- 2 Si C utilise k comme sommet intermédiaire:

Programmation dynamique: Floyd-Warshall

Soit $d_k(u, v)$ la longueur d'un plus court chemin de u à v n'utilisant que des sommets intermédiaires de numéro $< k$ (∞ s'il n'y a pas de chemin).

On va calculer $P(k) = \text{jj trouver } d_k(u, v), \text{ pour tous sommets } u, v \text{ jj}$.

Soit C un plus court chemin de u à v n'utilisant que des sommets intermédiaires de numéro $< k + 1$.

- 1 Si C n'utilise pas k comme sommet intermédiaire:

$$d_{k+1}(u, v) = d_k(u, v)$$

- 2 Si C utilise k comme sommet intermédiaire:

$$d_{k+1}(u, v) = d_k(u, k) + d_k(k, v)$$

Équation de récurrence:

Programmation dynamique: Floyd-Warshall

Soit $d_k(u, v)$ la longueur d'un plus court chemin de u à v n'utilisant que des sommets intermédiaires de numéro $< k$ (∞ s'il n'y a pas de chemin).

On va calculer $P(k) = \text{jj trouver } d_k(u, v), \text{ pour tous sommets } u, v \text{ jj}$.

Soit C un plus court chemin de u à v n'utilisant que des sommets intermédiaires de numéro $< k + 1$.

- 1 Si C n'utilise pas k comme sommet intermédiaire:

$$d_{k+1}(u, v) = d_k(u, v)$$

- 2 Si C utilise k comme sommet intermédiaire:

$$d_{k+1}(u, v) = d_k(u, k) + d_k(k, v)$$

Équation de récurrence:

$$d_{k+1}(u, v) = \min(d_k(u, v), d_k(u, k) + d_k(k, v))$$

Programmation dynamique: Floyd-Warshall

Initialiser $d_0(u, v) \leftarrow w(u, v)$ si $(u, v) \in \vec{E}$,
 ∞ sinon.

Pour $k = 0$ à $V - 1$:

 Pour tout sommet u :

 Pour tout sommet v :

$$d_{k+1}(u, v) \leftarrow \min(d_k(u, v), d_k(u, k) + d_k(k, v))$$

Programmation dynamique: Floyd-Warshall

Initialiser $d_0(u, v) \leftarrow w(u, v)$ si $(u, v) \in \vec{E}$,
 ∞ sinon.

Pour $k = 0$ à $V - 1$:

 Pour tout sommet u :

 Pour tout sommet v :

$$d_{k+1}(u, v) \leftarrow \min(d_k(u, v), d_k(u, k) + d_k(k, v))$$

On peut utiliser un tableau d à 3 dimensions pour stocker $d_k(u, v)$ dans $d.(u).(v).(k)$.

Programmation dynamique: Floyd-Warshall

Initialiser $d_0(u, v) \leftarrow w(u, v)$ si $(u, v) \in \vec{E}$,
 ∞ sinon.

Pour $k = 0$ à $V - 1$:

 Pour tout sommet u :

 Pour tout sommet v :

$$d_{k+1}(u, v) \leftarrow \min(d_k(u, v), d_k(u, k) + d_k(k, v))$$

On peut utiliser un tableau d à 3 dimensions pour stocker $d_k(u, v)$ dans $d.(u).(v).(k)$.

On a en fait juste besoin de d_k pour calculer d_{k+1} : on peut donc utiliser une matrice d telle que $d.(u).(v)$ contienne le dernier $d_k(u, v)$ calculé

Programmation dynamique: Floyd-Warshall

Initialiser $d_0(u, v) \leftarrow w(u, v)$ si $(u, v) \in \vec{E}$,
 ∞ sinon.

Pour $k = 0$ à $V - 1$:

 Pour tout sommet u :

 Pour tout sommet v :

$$d_{k+1}(u, v) \leftarrow \min(d_k(u, v), d_k(u, k) + d_k(k, v))$$

On peut utiliser un tableau d à 3 dimensions pour stocker $d_k(u, v)$ dans $d.(u).(v).(k)$.

On a en fait juste besoin de d_k pour calculer d_{k+1} : on peut donc utiliser une matrice d telle que $d.(u).(v)$ contienne le dernier $d_k(u, v)$ calculé (ça marche car $d_{k+1}(u, k) = d_k(u, k)$).

Programmation dynamique: Floyd-Warshall

$d.(u).(v)$ contient le dernier $d_k(u, v)$ calculé:

Initialiser $d.(u).(v) \leftarrow w(u, v)$ si $(u, v) \in \vec{E}$, ∞ sinon.

Pour $k = 0$ à $V - 1$:

 Pour tout sommet u :

 Pour tout sommet v :

$d.(u).(v) \leftarrow \min d.(u).(v) \ (d.(u).(k) + d.(k).(v))$

Programmation dynamique: Floyd-Warshall

$d.(u).(v)$ contient le dernier $d_k(u, v)$ calculé:

Initialiser $d.(u).(v) \leftarrow w(u, v)$ si $(u, v) \in \vec{E}$, ∞ sinon.

Pour $k = 0$ à $V - 1$:

 Pour tout sommet u :

 Pour tout sommet v :

$d.(u).(v) \leftarrow \min d.(u).(v) \ (d.(u).(k) + d.(k).(v))$

Complexité:

Programmation dynamique: Floyd-Warshall

$d.(u).(v)$ contient le dernier $d_k(u, v)$ calculé:

Initialiser $d.(u).(v) \leftarrow w(u, v)$ si $(u, v) \in \vec{E}$, ∞ sinon.

Pour $k = 0$ à $V - 1$:

 Pour tout sommet u :

 Pour tout sommet v :

$d.(u).(v) \leftarrow \min d.(u).(v) \ (d.(u).(k) + d.(k).(v))$

Complexité: $O(V^3)$

Floyd-Warshall

Initialiser $d.(u).(v) \leftarrow w(u, v)$ si $(u, v) \in \vec{E}$, ∞ sinon.

Pour $k = 0$ à $V - 1$:

 Pour tout sommet u :

 Pour tout sommet v :

$d.(u).(v) \leftarrow \min d.(u).(v) \ (d.(u).(k) + d.(k).(v))$

Question

Comment détecter un cycle de poids négatif?

Floyd-Warshall

Initialiser $d.(u).(v) \leftarrow w(u, v)$ si $(u, v) \in \vec{E}$, ∞ sinon.

Pour $k = 0$ à $V - 1$:

 Pour tout sommet u :

 Pour tout sommet v :

$d.(u).(v) \leftarrow \min d.(u).(v) \ (d.(u).(k) + d.(k).(v))$

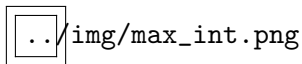
Question

Comment détecter un cycle de poids négatif?

Il y a un cycle de poids négatif $\iff \exists u, d.(u).(u) < 0$.



`max_int` est le plus grand entier représentable en Caml (dépend du processeur: 32 ou 64 bits).

Problèmes :



`max_int` est le plus grand entier représentable en Caml (dépend du processeur: 32 ou 64 bits).


Problèmes :


 `../img/max_int.png` `../img/sum.png`

Matrice d'adjacence pondérée

On utilise souvent une matrice d'adjacence $A = (a_{i,j})$ modifiée pour représenter un graphe $\vec{G} = (V, \vec{E})$ pondéré par w :

- $a_{i,j} = w(i,j)$, si $(i,j) \in \vec{E}$
- $a_{i,j} = 0$, si $i = j$
- $a_{i,j} = \infty$, si $(i,j) \notin \vec{E}$

../img/warshall_ex.png


../img/warshall_ex.png


Ma-
trice
d'adjacence

Matrice d'adjacence pondérée

On utilise souvent une matrice d'adjacence $A = (a_{i,j})$ modifiée pour représenter un graphe $\vec{G} = (V, \vec{E})$ pondéré par w :

- $a_{i,j} = w(i,j)$, si $(i,j) \in \vec{E}$
- $a_{i,j} = 0$, si $i = j$
- $a_{i,j} = \infty$, si $(i,j) \notin \vec{E}$

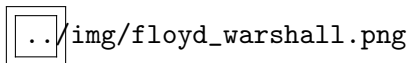
../img/warshall_ex.png

../img/warshall_ex2

Ma-
trice
des
dis-
tances
ren-
voyée
par
Floyd-
Warshall

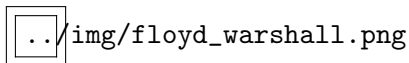
Floyd-Warshall

On suppose que le graphe G est représenté par matrice d'adjacence pondérée A :



Floyd-Warshall

On suppose que le graphe G est représenté par matrice d'adjacence pondérée W :



On pourrait copier l'entrée pour éviter de modifier la matrice d'adjacence à l'extérieur de la fonction...

Programmation dynamique: Floyd-Warshall

Initialiser $d.(u).(v) \leftarrow w(u, v)$ si $(u, v) \in \vec{E}$, ∞ sinon.

Pour $k = 0$ à $V - 1$:

 Pour tout sommet u :

 Pour tout sommet v :

$d.(u).(v) \leftarrow \min d.(u).(v) (d.(u).(k) + d.(k).(v))$

Question

Comment connaître un plus court chemin de n'importe quel sommet u à n'importe quel autre v ?

Programmation dynamique: Floyd-Warshall

Initialiser $d.(u).(v) \leftarrow w(u, v)$ si $(u, v) \in \vec{E}$, ∞ sinon.

Pour $k = 0$ à $V - 1$:

 Pour tout sommet u :

 Pour tout sommet v :

$d.(u).(v) \leftarrow \min d.(u).(v) (d.(u).(k) + d.(k).(v))$

Question

Comment connaître un plus court chemin de n'importe quel sommet u à n'importe quel autre v ?

Utiliser une matrice $pere$ telle que $pere.(u).(v)$ est le prédécesseur de v dans un plus court chemin de u à v .

Plus courts chemins avec Floyd-Warshall

Initialiser $d.(u).(v) \leftarrow w(u, v)$ si $(u, v) \in \vec{E}$, ∞ sinon.

Initialiser $pere.(u).(v) \leftarrow u$, $\forall u, v \in V$.

Pour $k = 0$ à $V - 1$:

 Pour tout sommet u :

 Pour tout sommet v :


 Si $d.(u).(v) > d.(u).(k) + d.(k).(v)$:


$d.(u).(v) \leftarrow d.(u).(k) + d.(k).(v)$

$pere.(u).(v) \leftarrow pere.(k).(v)$


On obtient une matrice $pere$ telle que $pere.(u).(v)$ est le prédécesseur de v dans un plus court chemin de u à v .


Floyd-Warshall

 `../img/init_pere.png`

 `../img/floyd_warshall_pere.png`

Floyd-Warshall

 `../img/floyd_warshall_pere.png`

 `../img/warshall_chemin.png`