Avant le cours 3 proprement dit

La preuve corrigée de la validité de BFS pour le calcul des plus courts chemins

M1 Info - Graphes et programmation dynamique

Cours 2 - Plus courts chemins a origine unique

BFS et Dijkstra

Semestre Automne 2022-2023 - Université Côte D'azur

Christophe Crespelle

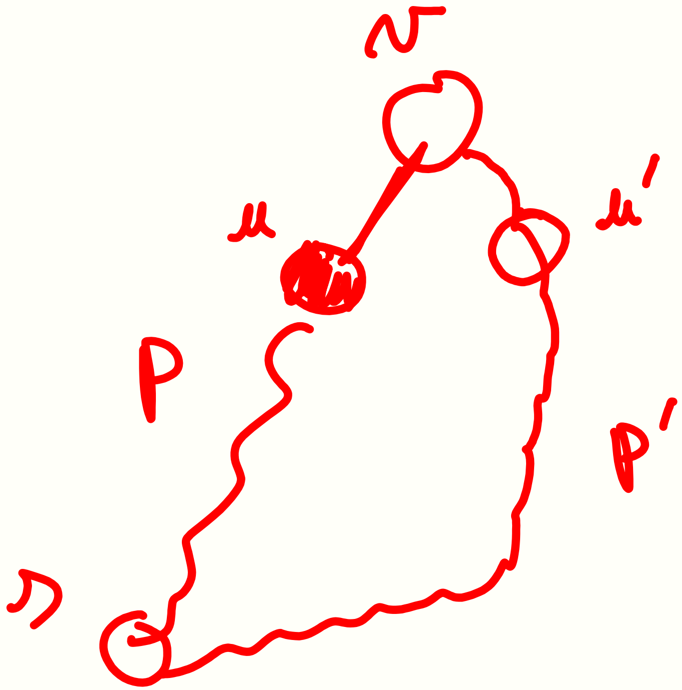
christophe.crespelle@univ-cotedazur.fr

Preuve de correction de l'algorithme

Theoreme : A la fin de l'algorithme, la distance $dist[u]$ affectee a chaque sommet u est exactement la distance $dist(s,u)$ entre u et s.

Démonstration.

Considerons la premiere fois au cours de l'algorithme que la distance affectee a un sommet v (ligne 10) n'est pas correcte et soit u le sommet affecte a la ligne 7 et qui a decouvert v.



$dist(s,u') < dist(s,u)$

□

Preuve de correction de l'algorithme

Theoreme : A la fin de l'algorithme, la distance $dist[u]$ affectee a chaque sommet u est exactement la distance $dist(s, u)$ entre u et s.

Démonstration.

Considerons la premiere fois au cours de l'algorithme que la distance affectee a un sommet v (ligne 10) n'est pas correcte et soit u le sommet affecte a la ligne 7 et qui a decouvert v.

On a a la fois $dist[v] > dist(s, v)$, d'apres le corollaire precedent, et $dist[u] = dist(s, u)$ par definition de v.

Considerons un plus court chemin de s a v, et notons u' le precedent de v sur ce chemin. On a
$dist(s, u') = dist(s, v) - 1 < dist[v] - 1 = dist[u] = dist(s, u)$.

Or d'apres la propriete 3, les sommets entrent dans F, et donc en sortent, a distance croissante. Ainsi, comme $dist(s, u') < dist(s, u)$, u' aurait donc du etre traite, et decouvrir v, avant u : contradiction. $\square$

M1 Info - Graphes et programmation dynamique

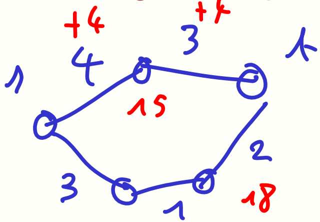
Cours 3 - Plus courts chemins (suite)

Bellman-Ford et Floyd-Warshall

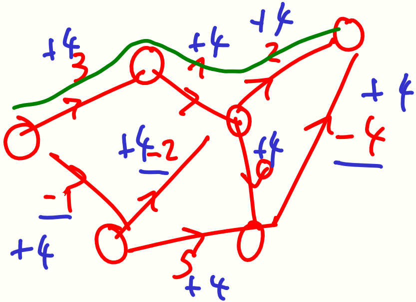
Semestre d'Automne 2022-2023 - Université Côte D'azur

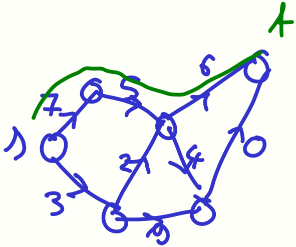
Christophe Crespelle

christophe.crespelle@univ-cotedazur.fr



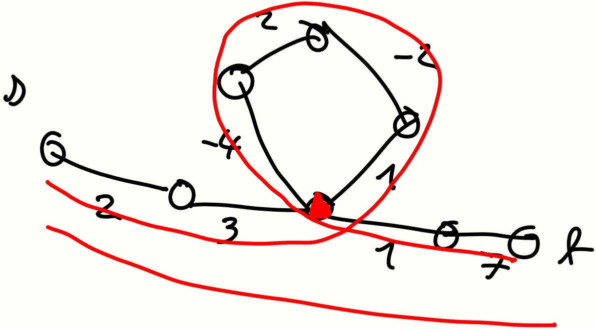
Graphes orientes avec poids negatifs





Un petit probleme avec les poids negatifs

- Les plus courts chemins ne sont plus necessairement **simples**



Un petit probleme avec les poids negatifs

- Les plus courts chemins ne sont plus necessairement **simples**

- En fait, il n'existe plus toujours de plus court chemin ! ! !

Un petit probleme avec les poids negatifs

- Les plus courts chemins ne sont plus necessairement **simples**

- En fait, il n'existe plus toujours de plus court chemin ! ! !

$\Longrightarrow$ **il faut interdire les cycles de poids negatifs**

Un petit probleme avec les poids negatifs

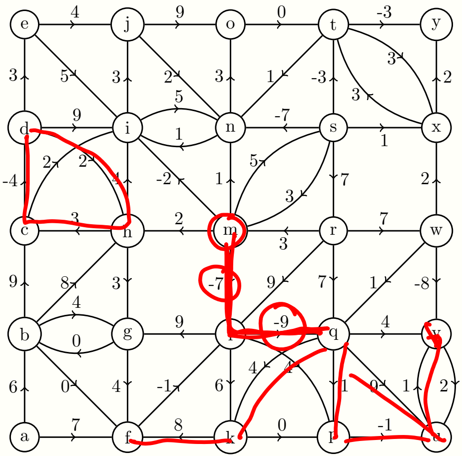
- Les plus courts chemins ne sont plus necessairement **simples**

- En fait, il n'existe plus toujours de plus court chemin ! ! !

$\Longrightarrow$ **il faut interdire les cycles de poids negatifs**

- **Nouveau cadre :** graphes orientes, ponderes sans cycles de poids negatifs

Exemple


e
j
o
t
y
d
i
n
s
x
c
h
m
r
w
b
g
l
q
v
a
f
k
p
u
4
9
0
-3
3
5
3
2
3
1
-3
3
2
3
9
5
1
-7
1
-4
2
2
4
-2
1
5
3
7
2
3
2
3
7
9
8
3
-7
9
7
1
-8
4
9
-9
4
0
6
0
4
-1
6
4
4
1
0
1
2
7
8
0
-1

Algorithme : Bellman-Ford

Algorithme 1 : Bellman-Ford(G,s)

Deux tableaux *dist* et *pere* de taille n;
pour u *de* 0 *a* $n-1$ **faire**
  $dist[u] \leftarrow +\infty$; $pere[u] \leftarrow \perp$;
fin
$dist[s] \leftarrow 0$;
pour $k = 1$ *a* $n-1$ **faire** n-1 fois
  pour $uv \in A$ **faire**
    si $dist[v] > dist[u] + w(u,v)$ **alors**
      $dist[v] \leftarrow dist[u] + w(u,v)$; $pere[v] \leftarrow u$;
    fin
  fin
fin
pour $uv \in A$ **faire**
  si $dist[v] > dist[u] + w(u,v)$
    alors retourner "cycle de poids negatif";
fin

relax(u,v)

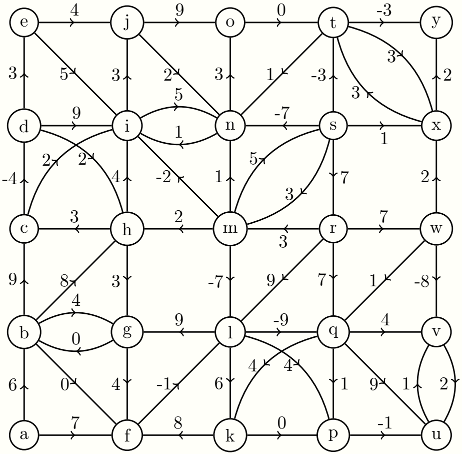
PCC

$O(m + n \log n)$

$O(n\,m)$

vérifie pas de cycle < 0

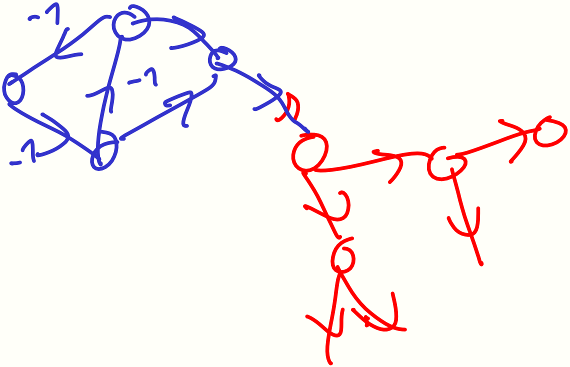
Exemple d’execution de Bellman-Ford



Preuve de correction de l'algorithme

Theoreme : L'algorithme de Bellman-Ford(G,s) est correct.

Lemme 1 : Si G ne contient pas de cycle de poids negatif accessible depuis s, alors apres les $n-1$ iterations de la boucle ligne 6, on a $\forall u \in V, dist[u] = dist(s,u)$.

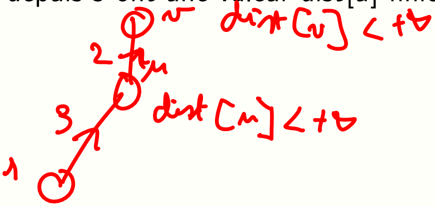


Preuve de correction de l'algorithme

Theoreme : L'algorithme de Bellman-Ford(G,s) est correct.

Lemme 1 : Si G ne contient pas de cycle de poids negatif accessible depuis s, alors apres les $n-1$ iterations de la boucle ligne 6, on a $\forall u \in V, dist[u] = dist(s,u)$.

Remarque 1 : Apres les $n-1$ iterations de la boucle ligne 6, tous les sommets u accessibles depuis s ont une valeur $dist[u]$ finie.



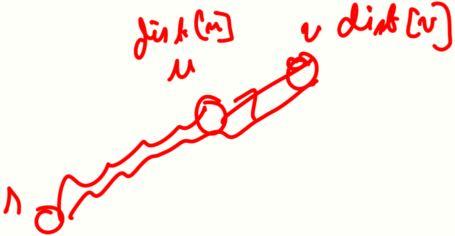
Preuve de correction de l'algorithme

Theoreme : L'algorithme de Bellman-Ford(G,s) est correct.

Lemme 1 : Si G ne contient pas de cycle de poids negatif accessible depuis s, alors apres les $n-1$ iterations de la boucle ligne 6, on a $\forall u \in V, dist[u] = dist(s,u)$.

Remarque 1 : Apres les $n-1$ iterations de la boucle ligne 6, tous les sommets u accessibles depuis s ont une valeur $dist[u]$ finie.

Remarque 2 : Au cours de l'algo, $\forall u \in V$, si $dist[u] < +\infty$, alors $\exists$ un chemin de s a u de longueur $dist[u]$. D'ou $dist[u] \geq dist(s,u)$.



Preuve de correction de l'algorithme

Theoreme : L'algorithme de Bellman-Ford(G,s) est correct.

Lemme 1 : Si G ne contient pas de cycle de poids negatif accessible depuis s, alors apres les $n-1$ iterations de la boucle ligne 6, on a $\forall u \in V, dist[u] = dist(s,u)$.

Remarque 1 : Apres les $n-1$ iterations de la boucle ligne 6, tous les sommets u accessibles depuis s ont une valeur $dist[u]$ finie.

Remarque 2 : Au cours de l'algo, $\forall u \in V$, si $dist[u] < +\infty$, alors $\exists$ un chemin de s a u de longueur $dist[u]$. D'ou $dist[u] \geq dist(s,u)$.

Preuve du Lemme 1.

Soit u accessible depuis s.

Comme il n'y a pas de cycle de poids negatif accessible depuis s, $\exists$ un plus court chemin $P = su_1u_2\cdots \underbrace{u_k}_{=u}$ de s a $u = u_k$ et il est simple.

Preuve de correction de l'algorithme

Theoreme : L'algorithme de Bellman-Ford(G,s) est correct.

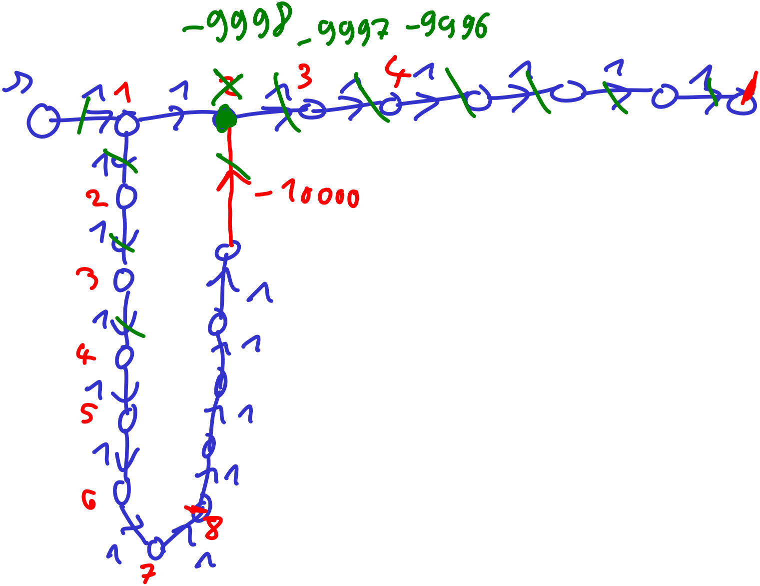
Lemme 1 : Si G ne contient pas de cycle de poids negatif accessible depuis s, alors apres les $n-1$ iterations de la boucle ligne 6, on a $\forall u \in V, dist[u] = dist(s,u)$.

Remarque 1 : Apres les $n-1$ iterations de la boucle ligne 6, tous les sommets u accessibles depuis s ont une valeur $dist[u]$ finie.

Remarque 2 : Au cours de l'algo, $\forall u \in V$, si $dist[u] < +\infty$, alors $\exists$ un chemin de s a u de longueur $dist[u]$. D'ou $dist[u] \geq dist(s,u)$.

Preuve du Lemme 1.

Soit u accessible depuis s.
Comme il n'y a pas de cycle de poids negatif accessible depuis s, $\exists$ un plus court chemin $P = s u_1 u_2 \cdots u_k$ de s a $u = u_k$ et il est simple.
$k \leq n-1$
Il contient au plus $n-1$ aretes et apres l'iteration i, $1 \leq i \leq k$, $dist[u_i] = dist(s, u_i)$.



-9998
-9997
-9996
-10000
1
2
3
4
5
6
7
8

Preuve de correction de l'algorithme

Lemme 2 : L'algorithme Bellman-Ford(G,s) retourne "cycle de poids negatif" ssi il existe un cycle de poids negatif dans G qui est accessible depuis s.

Preuve de correction de l'algorithme

Lemme 2 : L'algorithme Bellman-Ford(G,s) retourne "cycle de poids negatif" ssi il existe un cycle de poids negatif dans G qui est accessible depuis s.

Démonstration.

- pas de cycle de poids negatif $\Longrightarrow \forall uv \in A$, test ligne 14 faux

□

Preuve de correction de l'algorithme

<u>Lemme 2 :</u> L'algorithme Bellman-Ford(G,s) retourne "cycle de poids negatif" ssi il existe un cycle de poids negatif dans G qui est accessible depuis s.

Démonstration.

- pas de cycle de poids negatif $\Longrightarrow \forall uv \in A$, test ligne 14 faux

Si G ne contient pas de cycle negatif accessible depuis s, alors d'apres le lemme 1, on a $dist[u] = dist(s, u)$ pour tous les sommets u accessibles.

□

Preuve de correction de l'algorithme

Lemme 2 : L'algorithme Bellman-Ford(G,s) retourne "cycle de poids negatif" ssi il existe un cycle de poids negatif dans G qui est accessible depuis s.

Démonstration.

- pas de cycle de poids negatif $\Longrightarrow \forall uv \in A$, test ligne 14 faux

Si G ne contient pas de cycle negatif accessible depuis s, alors d'apres le lemme 1, on a $dist[u] = dist(s, u)$ pour tous les sommets u accessibles.
L'inegalite triangulaire donne alors $dist[v] \leq dist[u] + w(u, v)$.

□

Preuve de correction de l'algorithme

<u>Lemme 2 :</u> L'algorithme Bellman-Ford(G,s) retourne "cycle de poids negatif" ssi il existe un cycle de poids negatif dans G qui est accessible depuis s.

Démonstration.

- pas de cycle de poids negatif $\Longrightarrow \forall uv \in A$, test ligne 14 faux

Si G ne contient pas de cycle negatif accessible depuis s, alors d'apres le lemme 1, on a $dist[u] = dist(s, u)$ pour tous les sommets u accessibles.
L'inegalite triangulaire donne alors $dist[v] \leq dist[u] + w(u, v)$.

- $\forall uv \in A$, test ligne 14 faux $\Longrightarrow$ pas de cycle de poids negatif

□

Preuve de correction de l'algorithme

Lemme 2 : L'algorithme Bellman-Ford(G,s) retourne "cycle de poids negatif" ssi il existe un cycle de poids negatif dans G qui est accessible depuis s.

Démonstration.

- pas de cycle de poids negatif $\Longrightarrow \forall uv \in A$, test ligne 14 faux

Si G ne contient pas de cycle negatif accessible depuis s, alors d'apres le lemme 1, on a $dist[u] = dist(s, u)$ pour tous les sommets u accessibles.
L'inegalite triangulaire donne alors $dist[v] \leq dist[u] + w(u, v)$.

- $\forall uv \in A$, test ligne 14 faux $\Longrightarrow$ pas de cycle de poids negatif

Soit C un cycle accessible depuis s.

□

Preuve de correction de l'algorithme

Lemme 2 : L'algorithme Bellman-Ford(G,s) retourne "cycle de poids negatif" ssi il existe un cycle de poids negatif dans G qui est accessible depuis s.

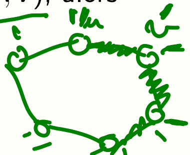
Démonstration.

- pas de cycle de poids negatif $\Longrightarrow \forall uv \in A$, test ligne 14 faux

Si G ne contient pas de cycle negatif accessible depuis s, alors d'apres le lemme 1, on a $dist[u] = dist(s, u)$ pour tous les sommets u accessibles.
L'inegalite triangulaire donne alors $dist[v] \leq dist[u] + w(u, v)$.

- $\forall uv \in A$, test ligne 14 faux $\Longrightarrow$ pas de cycle de poids negatif

Soit C un cycle accessible depuis s.
Si $\forall(u, v) \in A(C)$, $dist[v] \leq dist[u] + w(u, v)$, alors



□

Preuve de correction de l'algorithme

Lemme 2 : L'algorithme Bellman-Ford(G,s) retourne "cycle de poids negatif" ssi il existe un cycle de poids negatif dans G qui est accessible depuis s.

Démonstration.

- pas de cycle de poids negatif $\Longrightarrow \forall uv \in A$, test ligne 14 faux

Si G ne contient pas de cycle negatif accessible depuis s, alors d'apres le lemme 1, on a $dist[u] = dist(s, u)$ pour tous les sommets u accessibles.
L'inegalite triangulaire donne alors $dist[v] \leq dist[u] + w(u, v)$.

- $\forall uv \in A$, test ligne 14 faux $\Longrightarrow$ pas de cycle de poids negatif

Soit C un cycle accessible depuis s.
Si $\forall (u, v) \in A(C)$, $dist[v] \leq dist[u] + w(u, v)$, alors
$\sum_{u \in V(C)} d[v] \leq \sum_{u \in V(C)} d[u] + \sum_{uv \in A(C)} w(u, v)$. < 0 ???

$0 \leq 0$

□

Preuve de correction de l'algorithme

Lemme 2 : L'algorithme Bellman-Ford(G,s) retourne "cycle de poids negatif" ssi il existe un cycle de poids negatif dans G qui est accessible depuis s.

Démonstration.

- pas de cycle de poids negatif $\Longrightarrow \forall uv \in A$, test ligne 14 faux

Si G ne contient pas de cycle negatif accessible depuis s, alors d'apres le lemme 1, on a $dist[u] = dist(s, u)$ pour tous les sommets u accessibles.
L'inegalite triangulaire donne alors $dist[v] \leq dist[u] + w(u, v)$.

- $\forall uv \in A$, test ligne 14 faux $\Longrightarrow$ pas de cycle de poids negatif

Soit C un cycle accessible depuis s.
Si $\forall (u, v) \in A(C), dist[v] \leq dist[u] + w(u, v)$, alors
$\sum_{u \in V(C)} d[v] \leq \sum_{u \in V(C)} d[u] + \sum_{uv \in A(C)} w(u, v)$.
D'ou $\sum_{uv \in A(C)} w(u, v) \geq 0$.

$\square$

Analyse de complexite de l'algorithme

- Toutes les instructions s'executent en temps $O(1)$
- Boucle ligne 6 : $n - 1$ fois
- Boucle ligne 7 : m fois
- Boucle ligne 14 : m fois
- Total : $O(nm)$

Graphes orientes ponderes
sans cycle de poids negatif

Plus courts chemins entre toutes paires

- **Nouveau probleme :** plus courts chemins entre toutes paires
- **Meme cadre :** graphes orientes ponderes sans cycle de poids negatif

Plus courts chemins entre toutes paires $\Theta(n^2)$

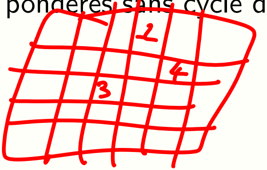
- **Nouveau probleme :** plus courts chemins entre toutes paires

- **Meme cadre :** graphes orientes ponderes sans cycle de poids negatif

- **Entrée :** un graphe $G = (V, E)$ non oriente et non pondere, un sommet $s \in V$
  - G represente par sa matrice d'adjacence ponderee W

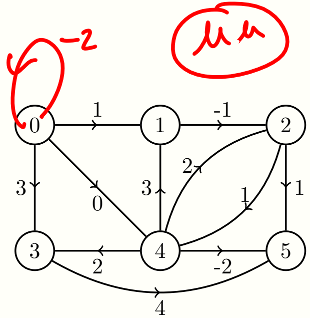
Plus courts chemins entre toutes paires

- **Nouveau probleme :** plus courts chemins entre toutes paires
- **Meme cadre :** graphes orientes ponderes sans cycle de poids negatif



- **Entrée :** un graphe $G = (V, E)$ non oriente et non pondere, un sommet $s \in V$
  - G represente par sa matrice d'adjacence ponderee W
- **Sortie :** la distance $dist(u, v)$ pour tous les couples de sommets $u, v \in V$
  - Une matrice de distance D avec d_{uv} la distance de u a v
  - Une matrice de predecesseur Π avec π_{uv} le predecesseur de v sur un plus court chemin de u a v

Exemple



u → 0

uv

uvw

	0					5
0	0					-2
1	+∞	0				
2	+∞		0			
3	+∞			0		
4	+∞				0	
5	+∞					0

Solution avec plus courts chemins a origine unique

- Graphes non ponderes : BFS depuis chaque sommet
  - temps $O(nm)$

Solution avec plus courts chemins a origine unique

- Graphes non ponderes : BFS depuis chaque sommet
  - temps $O(nm)$

- Graphes ponderes positivement : Dijkstra depuis chaque sommet
  - temps $O(nm + n^2 \log n)$ (avec implementation tas de Fibonacci)

Solution avec plus courts chemins a origine unique

- Graphes non ponderes : BFS depuis chaque sommet
  - temps $O(nm)$
- Graphes ponderes positivement : Dijkstra depuis chaque sommet
  - temps $O(nm + n^2 \log n)$ (avec implementation tas de Fibonacci)
- Graphes ponderes negativement sans cycle de poids negatif : Bellman-Ford depuis chaque sommet
  - temps $O(n^2 m)$ $n \times O(nm)$

Solution avec plus courts chemins a origine unique

- Graphes non ponderes : BFS depuis chaque sommet
  - temps $O(nm)$
- Graphes ponderes positivement : Dijkstra depuis chaque sommet
  - temps $O(nm + n^2 \log n)$ (avec implementation tas de Fibonacci)
- Graphes ponderes negativement sans cycle de poids negatif : Bellman-Ford depuis chaque sommet
  - temps $O(n^2 m)$

m

Graphes ponderes negativement sans cycle de poids negatif

- Solution basee programmation dynamique : Floyd-Warshall
  - temps $O(n^3)$

Idee : programmation dynamique

- Pas de cycle negatif $\Longrightarrow$ plus courts chemins simples

Idee : programmation dynamique

- Pas de cycle negatif $\Longrightarrow$ plus courts chemins simples
- Pour $k \in [\![0, n-1]\!]$, on note $d^{[k]}(u, v)$ la longueur du plus court chemin de u a v qui est
  - **simple** et
  - qui n'utilise **que les sommets [0,k] comme intermediaire**

Idee : programmation dynamique

- Pas de cycle negatif $\Longrightarrow$ plus courts chemins simples

$d^{[0]}$

- Pour $k \in [\![0, n-1]\!]$, on note $d^{[k]}(u, v)$ la longueur du plus court chemin de u a v qui est

  $[0, 0]$

  - **simple** et
  - qui n'utilise **que les sommets [0,k] comme intermediaire**

  Par convention, $d^{[-1]}(u, v) = w(u, v)$ si $uv \in A$ et $d^{[-1]}(u, v) = +\infty$ sinon.

$d^{[-1]} = W$

0

Idee : programmation dynamique

- Pas de cycle negatif $\implies$ plus courts chemins simples

- Pour $k \in [\![0, n-1]\!]$, on note $d^{[k]}(u, v)$ la longueur du plus court chemin de u a v qui est
  - **simple** et
  - qui n'utilise **que les sommets [0,k] comme intermediaire**

  Par convention, $d^{[-1]}(u, v) = w(u, v)$ si $uv \in A$ et $d^{[-1]}(u, v) = +\infty$ sinon.

  u, v

- On a la formule de recurrence suivante $\forall k \in [\![0, n-1]\!]$

$$d^{[k]}(u, v) = \min\{d^{[k-1]}(u, v), d^{[k-1]}(u, k) + d^{[k-1]}(k, v)\}$$

$d^{[0,k]}(u,v)$???

$d^{[0,\, k-1]}(u,v)$

Idee : programmation dynamique

- Pas de cycle negatif $\Longrightarrow$ plus courts chemins simples

- Pour $k \in [\![0, n-1]\!]$, on note $d^{[k]}(u, v)$ la longueur du plus court chemin de u a v qui est
  - **simple** et
  - qui n'utilise **que les sommets [0,k] comme intermediaire**

  Par convention, $d^{[-1]}(u, v) = w(u, v)$ si $uv \in A$ et $d^{[-1]}(u, v) = +\infty$ sinon.

  $[0, k-1] + k$ u, v

- On a la formule de recurrence suivante $\forall k \in [\![0, n-1]\!]$

$$d^{[k]}(u, v) = \min\{\underbrace{d^{[k-1]}(u, v)}_{[0,k-1]}, d^{[k-1]}(u, k) + d^{[k-1]}(k, v)\}$$

Algorithme : Floyd-Warshall

Algorithme 2 : Floyd-Warshall(W)

reserver une matrice $D^{[-1]}$ de taille $n \times n$;
pour $(u, v) \in V^2$ **faire**
    si $uv \in A$ **alors** $d^{[-1]}(u, v) = w(u, v)$;
                sinon $d^{[-1]}(u, v) = +\infty$;
fin
pour $k = 0$ *a* $n - 1$ **faire** n
    Reserver une matrice $D^{[k]}$ de taille $n \times n$
    pour $u = 0$ *a* $n - 1$ **faire** n
        pour $v = 0$ *a* $n - 1$ **faire** n
            $d^{[k]}(u, v) \leftarrow \min\{d^{[k-1]}(u, v), d^{[k-1]}(u, k) + d^{[k-1]}(k, v)\}$;
        fin
    fin
fin
retourner $D^{[n-1]}$;

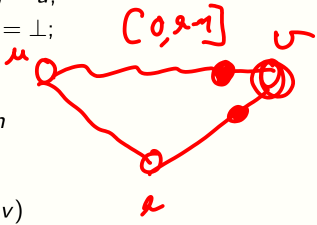
$d^{[-1]}$ $d^{[0]}$ $d^{[1]}$ $d^{[2]}$

O n O_v $\boxed{O(n^3)}$

Algorithme : Floyd-Warshall augmente

Algorithme 3 : Floyd-Warshall(W) augmente

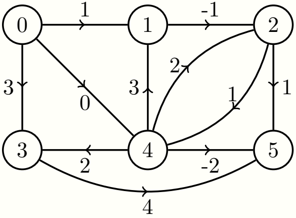
reserver deux matrices $D^{[-1]}$ et $\Pi^{[-1]}$ de taille $n \times n$;
pour $(u, v) \in V^2$ **faire**
  si $uv \in A$ **alors** $d^{[-1]}(u, v) = w(u, v)$; $\pi^{[-1]}(u, v) = u$;
    sinon $d^{[-1]}(u, v) = +\infty$; $\pi^{[-1]}(u, v) = \perp$;
fin
pour $k = 0$ *a* $n - 1$ **faire**
  Reserver deux matrice $D^{[k]}$ et $\Pi^{[k]}$ de taille $n \times n$
  pour $u = 0$ *a* $n - 1$ **faire**
    pour $v = 0$ *a* $n - 1$ **faire**
      si $d^{[k-1]}(u, k) + d^{[k-1]}(k, v) < d^{[k-1]}(u, v)$
        alors $d^{[k]}(u, v) \leftarrow d^{[k-1]}(u, k) + d^{[k-1]}(k, v)$;
          $\pi^{[k]}(u, v) \leftarrow \pi^{[k-1]}(k, v)$;
        sinon $d^{[k]}(u, v) \leftarrow d^{[k-1]}(u, v)$;
          $\pi^{[k]}(u, v) \leftarrow \pi^{[k-1]}(u, v)$;
    fin
  fin
fin
retourner $(D^{[n-1]}, \Pi^{[n-1]})$;



passe par k

passe pas par k

Exemple d'execution de Floyd-Warshall



Preuve de correction de l'algorithme

Theoreme : L'algorithme de Floyd-Warshall(W) est correct.

Preuve de correction de l'algorithme

Theoreme : L'algorithme de Floyd-Warshall(W) est correct.

- Pas de cycles negatif $\Longrightarrow$ plus court chemins simples

Preuve de correction de l'algorithme

Theoreme : L'algorithme de Floyd-Warshall(W) est correct.

- Pas de cycles negatif $\Longrightarrow$ plus court chemins simples
- La formule de recurrence est correcte (deja prouvee)

Preuve de correction de l'algorithme

<u>Theoreme :</u> L'algorithme de Floyd-Warshall(W) est correct.

- Pas de cycles negatif $\Longrightarrow$ plus court chemins simples
- La formule de recurrence est correcte (deja prouvee)
- $d^{[n-1]}(u, v)$ est la longueur d'un
  - plus court chemin **simple** de u a v
  - dont les **sommets intermediaires sont parmi** $[\![0, n-1]\!]$

Preuve de correction de l'algorithme

Theoreme : L'algorithme de Floyd-Warshall(W) est correct.

- Pas de cycles negatif $\Longrightarrow$ plus court chemins simples
- La formule de recurrence est correcte (deja prouvee)
- $d^{[n-1]}(u, v)$ est la longueur d'un
  - plus court chemin **simple** de u a v
  - dont les **sommets intermediaires sont parmi** $[\![0, n-1]\!]$

$$\boxed{\Longrightarrow d^{[n-1]} \text{ est exactement la distance de } u \text{ a } v}$$

Preuve de correction de l'algorithme

Theoreme : L'algorithme de Floyd-Warshall(W) est correct.

- Pas de cycles negatif $\Longrightarrow$ plus court chemins simples
- La formule de recurrence est correcte (deja prouvee)
- $d^{[n-1]}(u, v)$ est la longueur d'un
  - plus court chemin **simple** de u a v
  - dont les **sommets intermediaires sont parmi** $[\![0, n-1]\!]$

$$\boxed{\Longrightarrow d^{[n-1]} \text{ est exactement la distance de } u \text{ a } v}$$

- Idem pour $\pi^{[n-1]}(u, v)$

Analyse de complexite de l'algorithme

- Toutes les instructions s'executent en temps $O(1)$
- Boucle ligne 2 : temps $O(n^2)$
- 3 boucles imbriquees lignes 6, 8 et 9 : temps $O(n^3)$
- Total : $O(n^3)$

A savoir

toutes paires

- Graphes ponderes negativement sans cycle de poids negatif : Floyd-Warshall
  - temps $O(n^3)$
- Graphes ponderes positivement : Dijkstra depuis chaque sommet
  - temps $O(nm + n^2 \log n)$ (avec implementation tas de Fibonacci)

A savoir

- Graphes ponderes negativement sans cycle de poids negatif : Floyd-Warshall
  - temps $O(n^3)$

- Graphes ponderes positivement : Dijkstra depuis chaque sommet
  - temps $O(nm + n^2 \log n)$ (avec implementation tas de Fibonacci)

Un meilleur algo pour poids negatifs

- Graphes ponderes negativement sans cycle de poids negatif : algorithme de Johnson
  - temps $O(nm + n^2 \log n)$

A savoir

- Graphes ponderes negativement sans cycle de poids negatif : Floyd-Warshall
  - temps $O(n^3)$
- Graphes ponderes positivement : Dijkstra depuis chaque sommet
  - temps $O(nm + n^2 \log n)$ (avec implementation tas de Fibonacci)

Un meilleur algo pour poids negatifs

- Graphes ponderes negativement sans cycle de poids negatif : algorithme de Johnson
  - temps $O(nm + n^2 \log n)$
- Repondere pour n'avoir que des poids positifs !

A savoir

- Graphes ponderes negativement sans cycle de poids negatif : Floyd-Warshall
  - temps $O(n^3)$
- Graphes ponderes positivement : Dijkstra depuis chaque sommet
  - temps $O(nm + n^2 \log n)$ (avec implementation tas de Fibonacci)

Un meilleur algo pour poids negatifs

- Graphes ponderes negativement sans cycle de poids negatif : algorithme de Johnson
  - temps $O(nm + n^2 \log n)$
- Repondere pour n'avoir que des poids positifs !
  - Utilise Bellman-Ford (une seule fois) pour calculer une reponderation qui ne perturbe pas les PCC

A savoir

- Graphes ponderes negativement sans cycle de poids negatif : Floyd-Warshall
  - temps $O(n^3)$

- Graphes ponderes positivement : Dijkstra depuis chaque sommet
  - temps $O(nm + n^2 \log n)$ (avec implementation tas de Fibonacci)

$O(nm)$

Un meilleur algo pour poids negatifs

- Graphes ponderes negativement sans cycle de poids negatif : algorithme de Johnson
  - temps $O(nm + n^2 \log n)$
- Repondere pour n'avoir que des poids positifs !
  - Utilise Bellman-Ford (une seule fois) pour calculer une reponderation qui ne perturbe pas les PCC
  - Applique n fois Dijkstra sur le graphe repondere