Université de Nantes

UNIVERSITÉ DE NANTES

{département} INFORMATIQUE

M1 Informatique – Graphes
Problème des plus courts chemins

Irena.Rusu@univ-nantes.fr

LS2N, bât. 34, bureau 303

tél. 02.51.12.58.16

- Généralités
- Variante « Chemins de même origine »
  - Algorithme de Dijkstra
  - Algorithme de Bellman-Ford
  - Cas particulier : les graphes acycliques
- Variante « Toutes paires de sommets »
  - Algorithme de Johnson
  - Algorithme de Floyd-Warshall

- Généralités
- Variante « Chemins de même origine »
  - Algorithme de Dijkstra
  - Algorithme de Bellman-Ford
  - Cas particulier : les graphes acycliques
- Variante « Toutes paires de sommets »
  - Algorithme de Johnson
  - Algorithme de Floyd-Warshall

> Votre itinéraire : De Nantes A Aéroport International de Nantes-Atlantique

Graphe orienté valué : $G = (S, A, v)$ avec poids $v : A \rightarrow R$

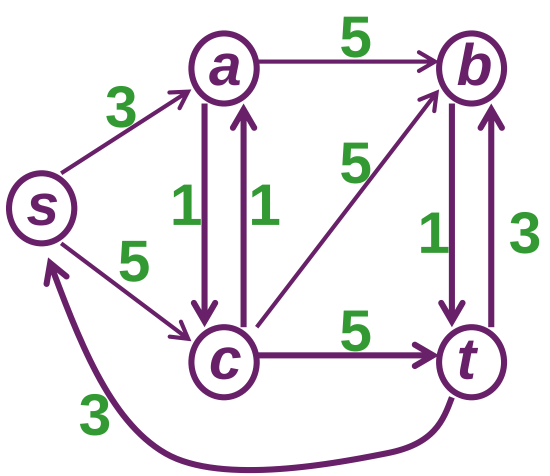
Poids d 'un chemin $c = ((s_0,s_1), (s_1,s_2), \ldots, (s_{k-1},s_k))$

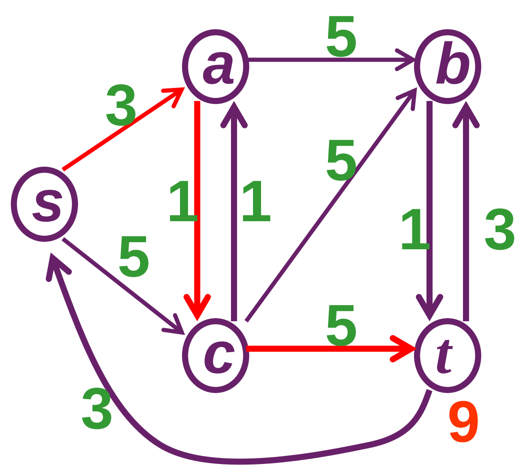
$$p(c) = \Sigma_{i=1}^{k} \; v[s_{i-1},s_i]$$

Distance de s à t : $d(s,t)=\min\{p(c),\, c$ chemin de s à $t\}\cup\{+\infty\}$

Plus court chemin de s à t : chemin c de s à t t.q.

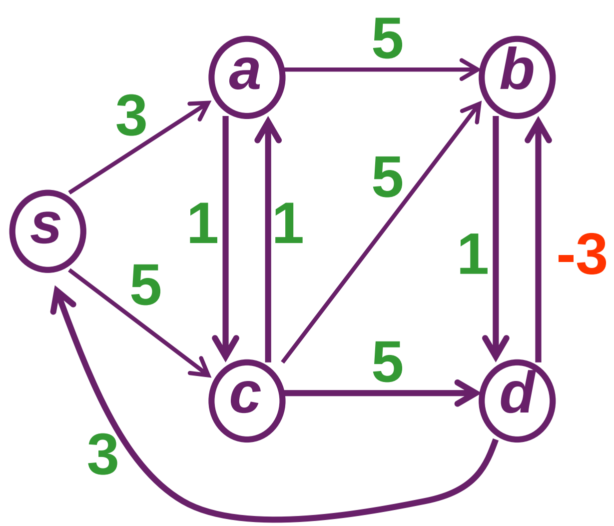
$p(c)=d(s,t)$, si un tel chemin existe





Proposition

pour tout $t \in S$ $d(s, t) > -\infty$ ssi
le graphe n'a pas de circuit de poids < 0
accessible depuis s



Problème : étant donnés $s, t \in S$ calculer

$$d(s, t) = \min \{ p(c) \; ; \; c \text{ chemin de } s \text{ à } t \} \cup \{+\infty\}$$

Variantes :

Chemins de même origine (ou même destination)

(selon les hypothèses)

algorithme de Dijkstra $O(|S|^2)$

ou $O((|S|+|A|).\log|S|)$

algorithme de Bellman-Ford $O(|A|.|S|)$

Toutes paires de sommets

algorithme de Johnson $O(|S|^2\log|S|+|S|.|A|)$

ou

(graphes peu denses) $O(|S|.|A|.\log|S|)$

algorithme de Floyd-Warshall $O(|S|^3)$

- Généralités
- Variante « Chemins de même origine »
  - Algorithme de Dijkstra
  - Algorithme de Bellman-Ford
  - Cas particulier : les graphes acycliques
- Variante « Toutes paires de sommets »
  - Algorithme de Johnson
  - Algorithme de Floyd-Warshall

Graphe valué : $G = (S, A, v)$ avec poids $v : A \to \mathbb{R}$

s un sommet fixé, $s \in S$

Problème : pour tout $t \in S$ calculer

$$d(s, t) = \min \{ p(c) \; ; \; c \text{ chemin de } s \text{ à } t \} \cup \{+\infty\}$$

Résultat : un arbre des plus courts chemins (c.à.d. un arbre dont les branches sont les plus courts chemins)

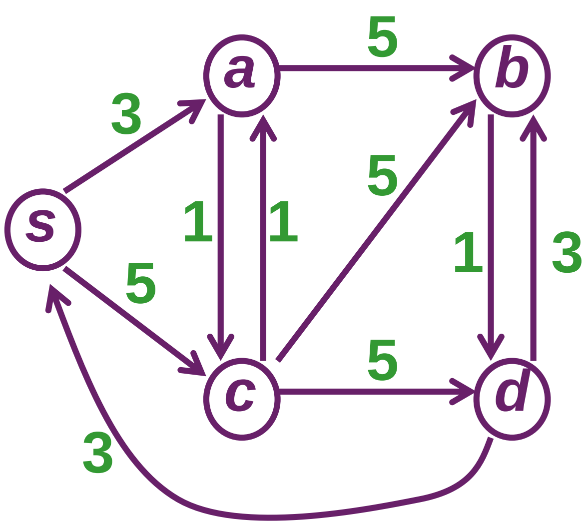
Graphe valué : $G = (S, A, v)$ avec poids $v : A \to \mathbb{R}$

s un sommet fixé, $s \in S$

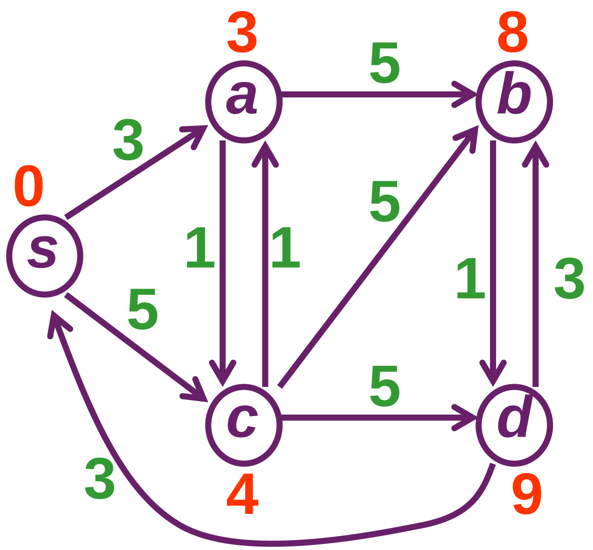
Problème : pour tout $t \in S$ calculer

$$d(s, t) = \min \{ p(c) \; ; \; c \text{ chemin de } s \text{ à } t \} \cup \{+\infty\}$$

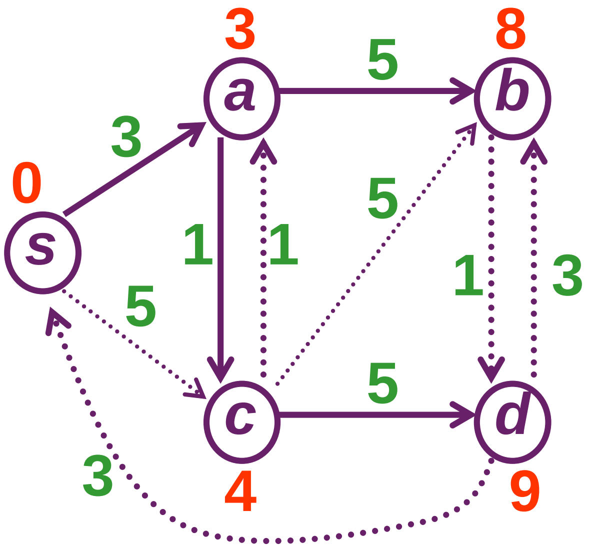
Exemple

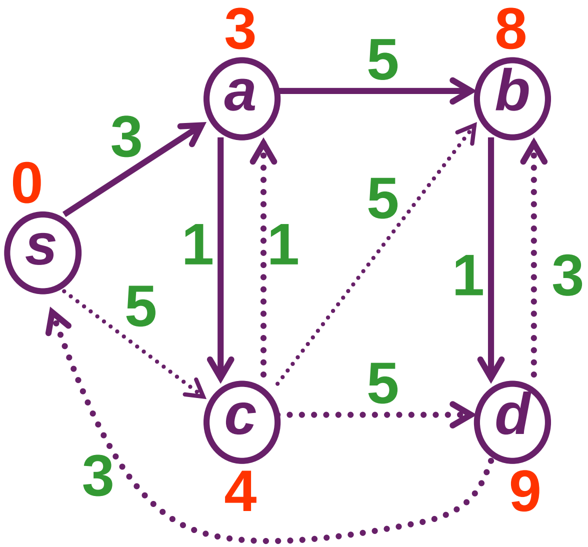


Poids *d(s,.)*

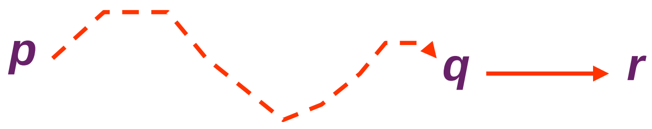


Arbres de racine *s*
dont les branches sont des chemins de poids minimum





Propriété 1 : $G = (S, A, v)$
soit c un plus court chemin de p à r
dont l'avant-dernier sommet est q
Alors $d(p, r) = d(p, q) + v(q, r)$



Propriété 2 : $G = (S, A, v)$
soit c un chemin de p à r
dont l'avant-dernier sommet est q
Alors $d(p, r) \leq d(p, q) + v(q, r)$

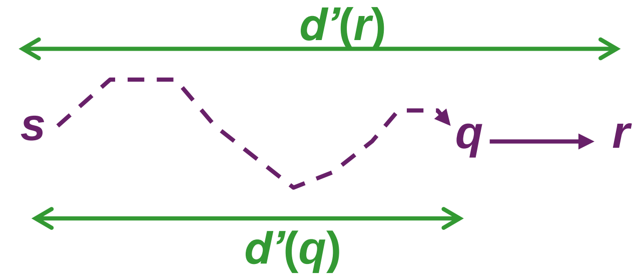
Calcul des $d(s,t)$ par approximations successives

$t \in S$ $d'(t)$ = estimation de $d(s, t)$

$\pi(t)$ = prédécesseur de t : avant-dernier sommet d'un chemin de s à t ayant pour poids $d'(t)$

Initialisation de d' et π

```
INIT
    pour chaque  t ∈ S faire
        { d'(t) ← ∞ ;   π(t) ← nil; }
    d'(s) ← 0;
```



Relaxation de l 'arc (q, r)

```
RELAX(q, r)
    si  d'(q) + v(q, r) < d'(r)
    alors { d'(r) ← d'(q) + v(q, r) ;   π(r) ← q ; }
```

Proposition :

la propriété « pour tout $t \in S$, $d'(t) \geq d(s, t)$ »
est un invariant de `relax`

Preuve par induction sur le nombre
d 'exécutions de `relax`

- Généralités
- Variante « Chemins de même origine »
  - Algorithme de Dijkstra
  - Algorithme de Bellman-Ford
  - Cas particulier : les graphes acycliques
- Variante « Toutes paires de sommets »
  - Algorithme de Johnson
  - Algorithme de Floyd-Warshall

Condition : $v(q, r) \geq 0$ pour tout arc (q, r)

```
début
    INIT;
    Q ← S ;
    tant que Q ≠ ∅ faire {
        soit q tel que d′(q)= min{d′(v), v ∈ Q};
        Q ← Q - {q} ;
        pour chaque r∈Q successeur de q faire
            RELAX(q, r) ;
    }
fin
```

C'est un algorithme glouton.

Par matrice d'adjacence

temps global $O(|S|^2)$

Par listes de successeurs

Q : implémenté comme un AVL (ordonné selon les d'(.))

$|S|$ opérations `min` : $O(|S|.\log|S|)$

$|A|$ opérations `RELAX` : $O(|A|.\log|S|)$

temps global $O((|S|+|A|).\log|S|)$

Condition : *aucune*

```
début
      INIT;
      Q ← S ;
      tant que Q ≠ ∅ faire {
            soit q tel que d′(q) = min{d′(v), v ∈ Q} ;
            Q ← Q - {q} ;
            pour chaque r successeur de q faire {
                  RELAX(q, r) ;
                  si changement alors Q ← Q ∪ {r}
                  }
      }
fin
```

Remarque. Dans le pire des cas, exponentiel.

- Généralités
- Variante « Chemins de même origine »
  - Algorithme de Dijkstra
  - Algorithme de Bellman-Ford
  - Cas particulier : les graphes acycliques
- Variante « Toutes paires de sommets »
  - Algorithme de Johnson
  - Algorithme de Floyd-Warshall

- RELAX sur tous les arcs 1 fois = parcourir (en relaxant) tous les chemins de longueur 1 du graphe
- RELAX sur tous les arcs 2 fois = parcourir (en relaxant) tous les chemins de longueur 2 du graphe
- …
- RELAX sur tous les arcs n-1 fois = parcourir (en relaxant) tous les chemins de longueur n-1 du graphe

- Le plus long chemin possible sans cycle a n-1 arcs

→ un n-ème RELAX n'améliore pas, si pas de cycle négatif (inversement, s'il améliore → il y a un cycle de poids négatif)

Aucune condition : pour tout arc (q, r), $v(q, r) \in \mathbb{R}$

```
début
    INIT;
    Q ← S ;
    répéter |S| -1 fois
        pour chaque (q, r) ∈ A faire
            RELAX(q, r) ;
    pour chaque (q, r) ∈ A faire
        si d'(q) + v(q, r) < d'(r) alors
            retour « cycle de poids négatif »
    retour « poids calculés »
fin
```

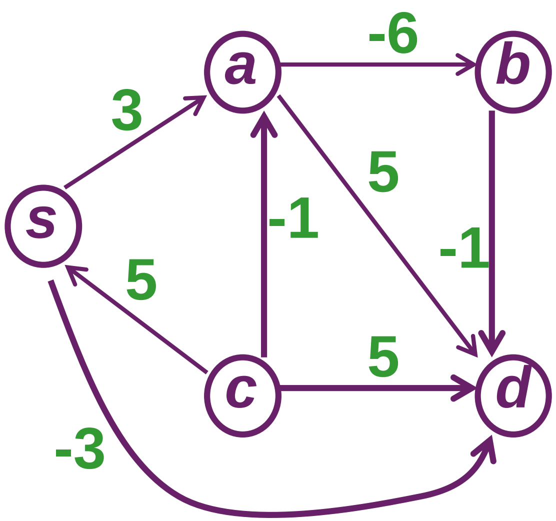
Temps : $O(|S|.|A|)$

- Généralités
- Variante « Chemins de même origine »
  - Algorithme de Dijkstra
  - Algorithme de Bellman-Ford
  - Cas particulier : les graphes acycliques
- Variante « Toutes paires de sommets »
  - Algorithme de Johnson
  - Algorithme de Floyd-Warshall

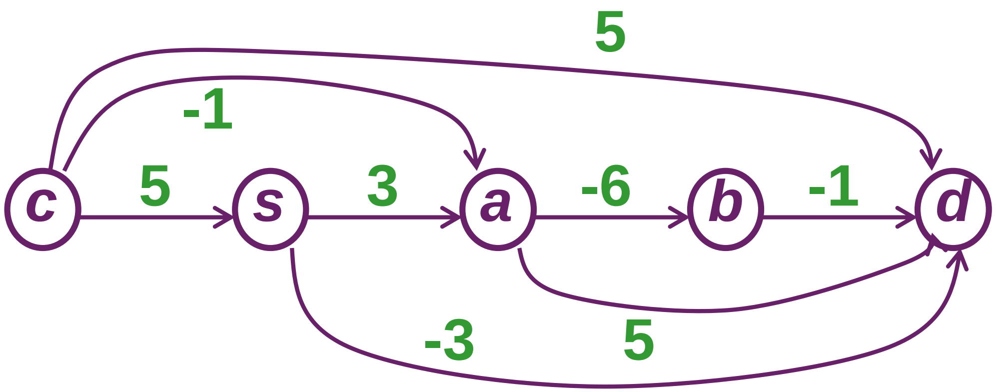
Aucune condition : pour tout arc (q, r), $v(q, r) \in \mathrm{R}$
Calcul après ordre topologique

```
début
      INIT;
      pour chaque q ∈ S en ordre topologique faire
            pour chaque r successeur de q faire
                  RELAX(q, r) ;
fin
```

Temps : $O(|S| + |A|)$
chaque sommet et chaque arc est examiné une fois



Ordre topologique
c, *s*, *a*, *b*, *d*



- Généralités
- Variante « Chemins de même origine »
  - Algorithme de Dijkstra
  - Algorithme de Bellman-Ford
  - Cas particulier : les graphes acycliques
- Variante « Toutes paires de sommets »
  - Algorithme de Johnson
  - Algorithme de Floyd-Warshall

$G = (S, A, v)$ graphe orienté valué

$$S = \{1,2,\ldots,n\} \qquad v : A \to \mathrm{R}$$

Matrice des poids : $W = (W[i,j])$ avec

$$W[i,j] = \begin{cases} 0 & \text{si } i = j \\ v(i,j) & \text{si } (i, j) \in A \\ \infty & \text{sinon} \end{cases}$$

Problème. Calculer la matrice des distances

$$D = (\ d(i, j) \mid 1 \leq i, j \leq n\ )$$

- Retourne une matrice des poids des plus courts chemins, *ou*
- Indique si le graphe en entrée contient un circuit de poids négatif

Idée :

définir un nouveau poids ≥ 0 sur les arcs, tout en gardant les plus courts chemins

appliquer l’algorithme de Dijkstra à partir de chaque sommet.

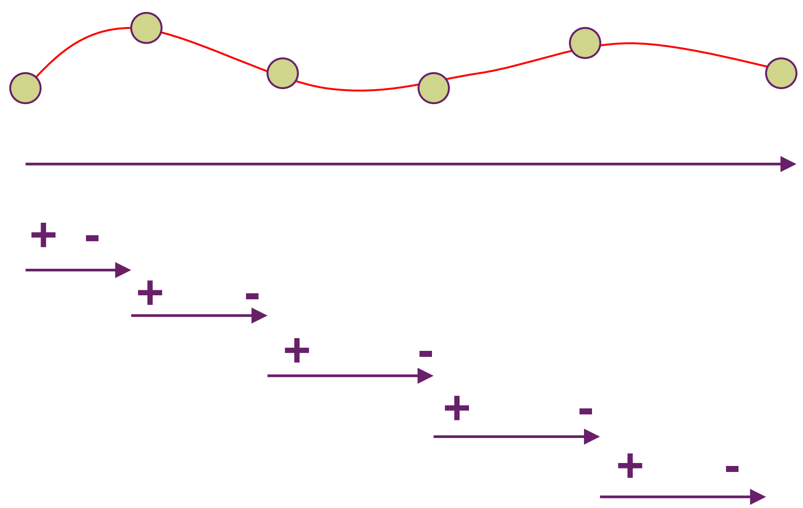
Propriété. $G=(S,A,v)$ graphe orienté valué avec $v: A \to \mathbf{R}$

$h: S \to \mathbf{R}$ fonction donnée

Soit $v': A \to \mathbf{R}$ définie pour tous $s,t \in S$ par $v'(s,t)=v(s,t)+h(s)-h(t)$.

Alors pour tout chemin $c = ((s_0,s_1), (s_1,s_2), \ldots, (s_{k-1},s_k))$ on a

$p'(c)=p(c)+h(s_0) - h(s_k)$, où p' poids obtenu avec v'.



- Les longueurs relatives des chemins

  si *c, c'* chemins de s_0 à s_k tels que $p(c) \leq p(c')$

  alors $p'(c) \leq p'(c')$

  ⇨ *les plus courts chemins sont les mêmes*

- Les circuits de longueur négative

  dans ce cas $s_0 = s_k$ donc $p(c) = p(c')$

  (où *c* est un chemin fermé, c.à.d. un circuit)

$G=(S,A,v)$ graphe

Propriété. Soit v prolongé par des 0 dans le nouveau graphe

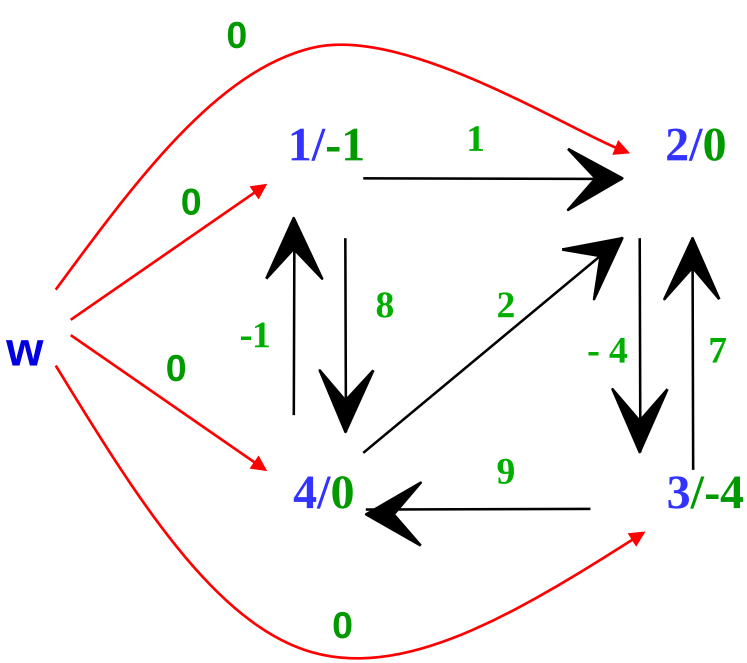
$$G'=(S \cup \{w\}, A \cup \{ws \mid s \in S\}, v).$$

Alors G a un circuit de longueur négative si et seulement si G' a un circuit de longueur négative.

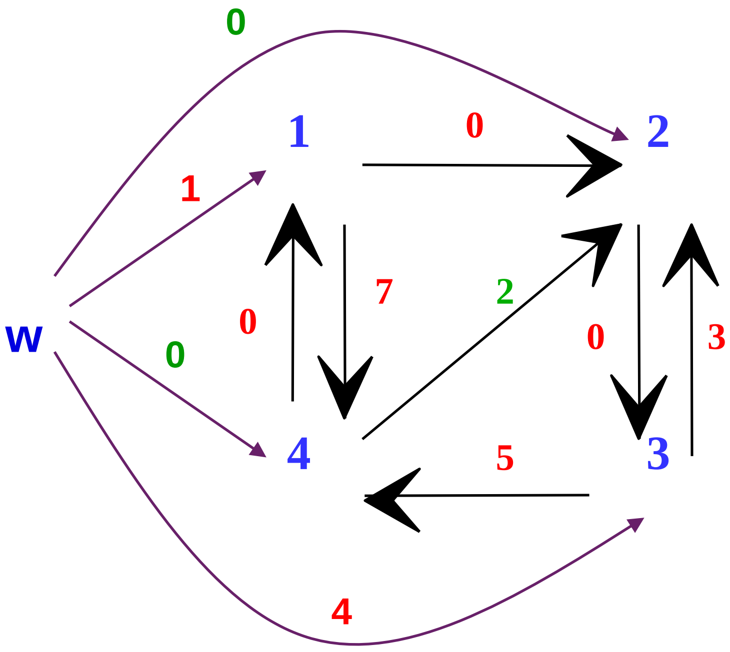
Soit $h(s)=d(w,s)$ pour tout $s \in S$.

$\rightarrow h(t) \leq h(s) + v(s,t), \forall s,t$ (inégalité triangulaire)

$\rightarrow$ **$v'(s,t)=v(s,t)+h(s)-h(t) \geq 0, \forall s,t$**



Poids *v*



Poids *v'*

Algorithme **Johnson**$(G=(S,A,v))$

Début

$G' \leftarrow (S \cup \{ w \}, A \cup \{ ws \mid s \in S \})$;

pour tout $s \in S$ **faire** $v(w,s) \leftarrow 0$ **fin pour**

Bellman-Ford (G',w,v) :

si existe circuit de poids négatif **alors** stop

sinon

pour chaque sommet $s \in S \cup \{ w \}$ **faire**

$h(s) \leftarrow d(w,s)$ calculée par Bellman-Ford **fin pour**

pour chaque arc $(s,t) \in A \cup \{ws \mid s \in S\}$ **faire**

$v'(s,t) \leftarrow v(s,t)+h(s)-h(t)$ **fin pour**

pour chaque sommet $s \in S$ **faire**

avec **Dijkstra**(G,s,v') obtenir $d'(s,u)$, $\forall u \in S$

pour chaque $u \in S$ **faire**

$d(s,u)=d'(s,u)-h(s)+h(u)$ **fin pour**

fin pour

fin si

Fin

Par listes de successeurs

Bellman-Ford : $O(|A|.|S|)$

Dijkstra : $O((|S|+|A|)log|S|)$ répété $|S|$ fois

Temps total : $O(|S|.|A|.log|S|)$

Peut être implémenté en $O(|S|^2 log\ |S| + |S|.|A|)$.

- Généralités
- Variante « Chemins de même origine »
  - Algorithme de Dijkstra
  - Algorithme de Bellman-Ford
  - Cas particulier : les graphes acycliques
- Variante « Toutes paires de sommets »
  - Algorithme de Johnson
  - Algorithme de Floyd-Warshall

- En général, problèmes d'optimisation (min/max d'un coût)
- « Diviser pour régner », mais :
  - Séparer la recherche du coût min/max de celle de l'« objet » produisant le min/max (c.à.d. de la solution elle-même)
  - Exprimer le min/max du problème en combinant des solutions de sous-problèmes, sous forme récursive
  - Résoudre les sous-problèmes les plus faciles d'abord, et garder leurs solutions dans un tableau
  - Garder de l'information permettant de construire à la fin la solution.
  - Programmation itérative, pas récursive

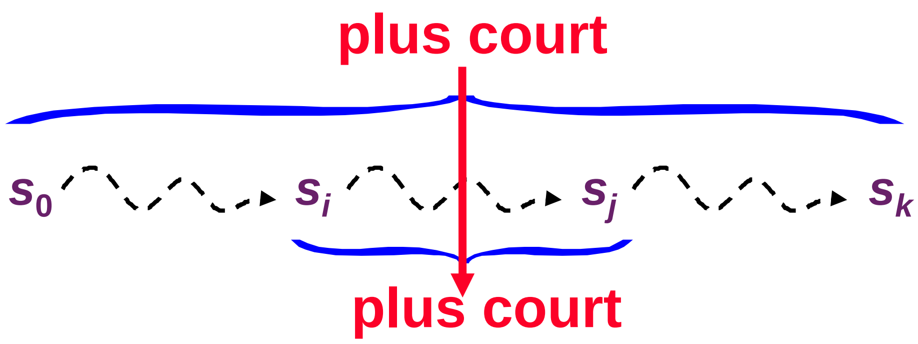
Condition: pas de circuit de poids négatif dans G

Lemme de base

$((s_0,s_1), \ldots, (s_i ,s_{i+1}), \ldots, (s_{j-1}, s_j), \ldots, (s_{k-1},s_k))$

plus court chemin de s_0 à s_k dans G

$\Rightarrow$ $((s_i ,s_{i+1}), \ldots, (s_{j-1},s_j))$ plus court chemin de s_i à s_j dans G



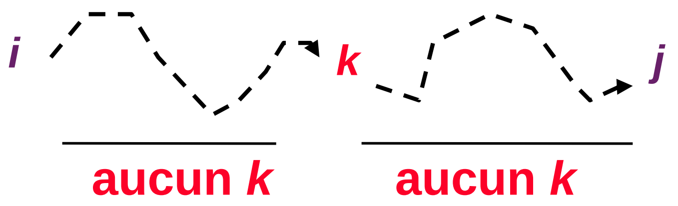
Notation

D_k = ($D_k[i, j]$ | $1 \le i, j \le n$) avec

$D_k[i, j]$ = min{ $W(c)$ | c chemin de i à j dont les sommets intermédiaires sont tous $\le k$ }

D_0 = W

D_n = matrice des distances de G = D



Lemme. Pour tout $k \ge 1$,

$D_k[i,j] = \min\{ D_{k-1}[i,j] , D_{k-1}[i,k] + D_{k-1}[k,j] \}$

Calcul :

de D_k à partir de D_{k-1} en temps $O(n^2)$

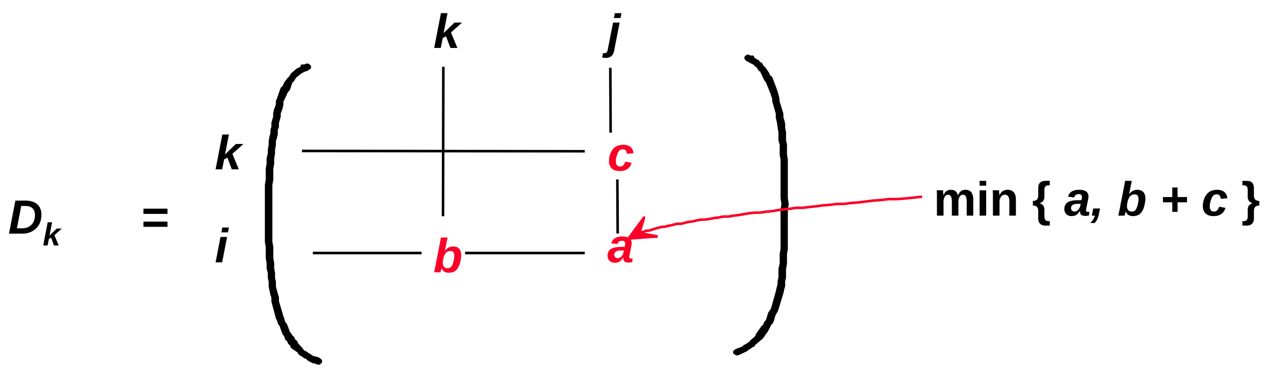
de $D = D_n$ en temps $O(n^3)$

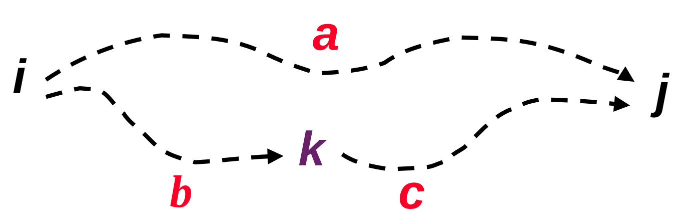
pour $k \leftarrow 1$ à n faire

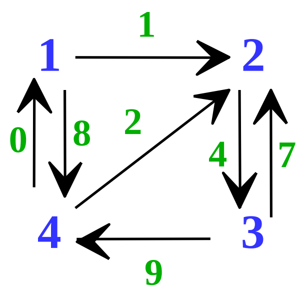
  pour $i \leftarrow 1$ à n faire

    pour $j \leftarrow 1$ à n faire

      $D[i, j] \leftarrow \min \{ D[i, j] , D[i, k] + D[k, j] \}$ **;**







$$D_0 = W = \begin{bmatrix} 0 & 1 & \infty & 8 \\ \infty & 0 & 4 & \infty \\ \infty & 7 & 0 & 9 \\ 0 & 2 & \infty & 0 \end{bmatrix}$$

$$D_1 = \begin{bmatrix} 0 & 1 & \infty & 8 \\ \infty & 0 & 4 & \infty \\ \infty & 7 & 0 & 9 \\ 0 & 1 & \infty & 0 \end{bmatrix}$$

$$D_2 = \begin{bmatrix} 0 & 1 & 5 & 8 \\ \infty & 0 & 4 & \infty \\ \infty & 7 & 0 & 9 \\ 0 & 1 & 5 & 0 \end{bmatrix}$$

$$D_3 = \begin{bmatrix} 0 & 1 & 5 & 8 \\ \infty & 0 & 4 & 13 \\ \infty & 7 & 0 & 9 \\ 0 & 1 & 5 & 0 \end{bmatrix}$$

$$D_4 = \begin{bmatrix} 0 & 1 & 5 & 8 \\ 13 & 0 & 4 & 13 \\ 9 & 7 & 0 & 9 \\ 0 & 1 & 5 & 0 \end{bmatrix}$$

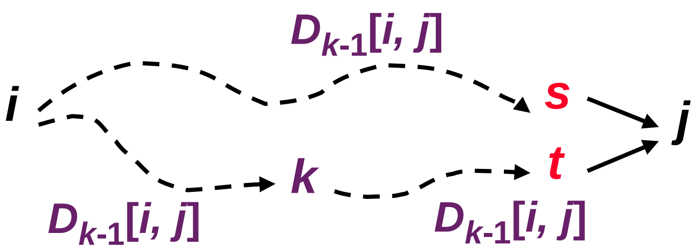
Mémorisation explicite des plus courts chemins de i à j, $1 \leq i, j \leq n$

n^2 chemins de longueur maximale n-1 : espace $O(n^3)$

Matrice des prédécesseurs : espace $\Theta(n^2)$

$P_k = (P_k[i, j] \mid 1 \leq i, j \leq n)$ avec

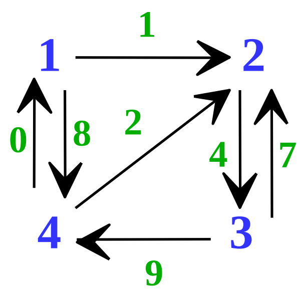
$P_k[i, j]$ = prédécesseur de j sur un plus court chemin de i à j dont les sommets intermédiaires sont tous $\leq k$



Récurrence

$$P_0[i, j] = \begin{cases} i & \text{si } (i, j) \in A \\ - & \text{sinon} \end{cases}$$

$$P_k[i, j] = \begin{cases} P_{k-1}[i, j] & \text{si } D_{k-1}[i, j] \leq D_{k-1}[i, k] + D_{k-1}[k, j] \\ P_{k-1}[k, j] & \text{sinon} \end{cases}$$



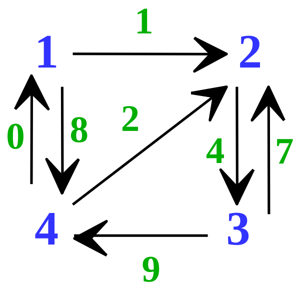
$$D_0 = W = \begin{pmatrix} 0 & 1 & \infty & 8 \\ \infty & 0 & 4 & \infty \\ \infty & 7 & 0 & 9 \\ 0 & 2 & \infty & 0 \end{pmatrix} \qquad P_0 = \begin{pmatrix} - & 1 & - & 1 \\ - & - & 2 & - \\ - & 3 & - & 3 \\ 4 & 4 & - & - \end{pmatrix}$$

$$D_1 = \begin{pmatrix} 0 & 1 & \infty & 8 \\ \infty & 0 & 4 & \infty \\ \infty & 7 & 0 & 9 \\ 0 & 1 & \infty & 0 \end{pmatrix} \qquad P_1 = \begin{pmatrix} - & 1 & - & 1 \\ - & - & 2 & - \\ - & 3 & - & 3 \\ 4 & 1 & - & - \end{pmatrix}$$

$$D_2 = \begin{pmatrix} 0 & 1 & 5 & 8 \\ \infty & 0 & 4 & \infty \\ \infty & 7 & 0 & 9 \\ 0 & 1 & 5 & 0 \end{pmatrix} \qquad P_2 = \begin{pmatrix} - & 1 & 2 & 1 \\ - & - & 2 & - \\ - & 3 & - & 3 \\ 4 & 1 & 2 & - \end{pmatrix}$$

$$D_3 = \begin{pmatrix} 0 & 1 & 5 & 8 \\ \infty & 0 & 4 & 13 \\ \infty & 7 & 0 & 9 \\ 0 & 1 & 5 & 0 \end{pmatrix} \qquad P_3 = \begin{pmatrix} - & 1 & 2 & 1 \\ - & - & 2 & 3 \\ - & 3 & - & 3 \\ 4 & 1 & 2 & - \end{pmatrix}$$

$$D_4 = \begin{pmatrix} 0 & 1 & 5 & 8 \\ 13 & 0 & 4 & 13 \\ 9 & 7 & 0 & 9 \\ 0 & 1 & 5 & 0 \end{pmatrix} \qquad P_4 = \begin{pmatrix} - & 1 & 2 & 1 \\ 4 & - & 2 & 3 \\ 4 & 3 & - & 3 \\ 4 & 1 & 2 & - \end{pmatrix}$$



$$D_4 = \begin{pmatrix} 0 & 1 & 5 & 8 \\ 13 & 0 & 4 & 13 \\ 9 & 7 & 0 & 9 \\ 0 & 1 & 5 & 0 \end{pmatrix} \qquad P_4 = \begin{pmatrix} - & 1 & 2 & 1 \\ 4 & - & 2 & 3 \\ 4 & 3 & - & 3 \\ 4 & 1 & 2 & - \end{pmatrix}$$

Exemple de chemin

distance de 2 à 1 = $D_4[2,1]$ = 13

$P_4[2,1] = 4$; $P_4[2,4] = 3$; $P_4[2,3] = 2$;

$$2 \xrightarrow{4} 3 \xrightarrow{9} 4 \xrightarrow{0} 1$$

- Applications diverses :
  - Pratiques (dans les réseaux d'ordinateurs, de téléphonie …)
  - Pour modéliser d'autres problèmes (reconstruire des séquences à partir de fragments, élaborer des stratégies de jeu …)

- Algorithmes simples et efficaces

- Implémentations pas vraiment compliquées