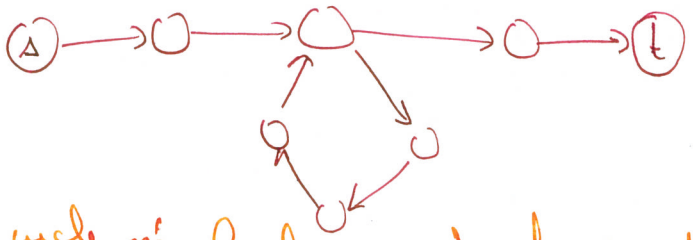
Généralisation aux poids négatifs [Bellman-Ford]

⚠ s'il existe un cycle de long. négative ⇒ pas de plus court chemin.

Approche par prog. dyn.



- Si pas de cycle nég. le plus court chemin de s à t est forcément élémentaire (ie. sans cycle)
- Si $s \to \cdots \to x \to t$ est optimal pour $s \to t$

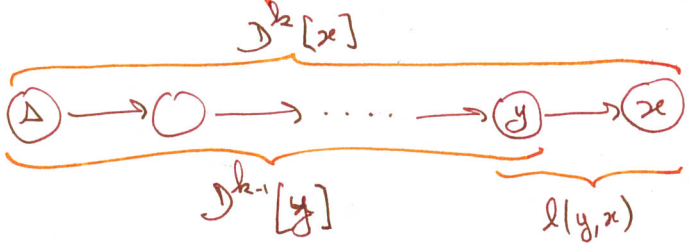
alors $s \to \cdots \to x$ ——— $s \to x$

⇒ on a une sous-structure optimale ⇒ la prog. dyn. s'applique.

Notons $D^k[x]$ la long. des plus court chemin pour aller de s à x avec au plus k arcs. On convient que $D^k[x] = \infty$ quand ce n'est pas possible.

Objectif : calculer $D^{|V|-1}[x]$ pour tout x.

Def. récursive :



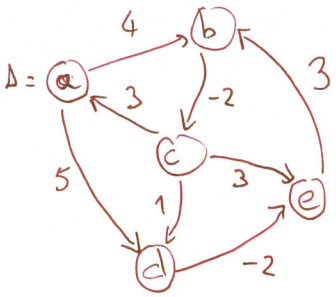
d'où

$$\begin{cases} D^k[x] = \min\limits_{y \in V \setminus \{x\}} \left(D^{k-1}[y] + l(y,x) \right) \\ \text{ou} \\ D^k[x] = D^{k-1}[x] \text{ si } s \to x \text{ n'a besoin que de } k-1 \text{ arcs.} \end{cases}$$

On fusionne les deux cas avec :

$$D^k[x] = \min_{y \in V} \left(D^{k-1}[y] + l(y,x) \right) \quad \text{en convenant que } l(x,x) = 0.$$

et on pose $D^0[s] = 0$ et $D^0[x] = \infty$ pour $x \neq s$



ℓ

	a	b	c	d	e
a	0	4	∞	5	∞
b	∞	0	-2	∞	∞
c	3	∞	0	1	3
d	∞	∞	∞	0	-2
e	∞	3	∞	∞	0

	a	b	c	d	e
D^0	0	∞	∞	∞	∞
D^1	0	4	∞	5	∞
D^2	0	4	2	5	3
D^3	0	4	2	3	3
D^4	0	4	2	3	1

Premier jet :

$\forall x,\ D^0[x] \Leftarrow \infty$
$D^0[s] \leftarrow 0$
for $k \leftarrow 1$ to $|V|-1$
  for $x \in V$
    $D^k[x] = \min_{y \in V} D^{k-1}[y] + \ell(y,x)$

$\Theta(|V|^3)$

Améliorations :

1) au lieu de for $x \in V$ puis $\min_{y \in V}$ il vaudrait mieux itérer sur tous les arcs :

$\forall x,\ D^0[x] \leftarrow \infty$
$D^0[s] \leftarrow 0$
for $k \leftarrow 1$ to $|V|-1$
  $\forall x,\ D^k[x] = D^{k-1}[x]$
  for $(y,x) \in E$
    $D^k[x] \leftarrow \min\left(D^k[x], D^{k-1}[y] + \ell(y,x)\right)$

$\Theta(|V|.|E|)$ avec liste d'adj.

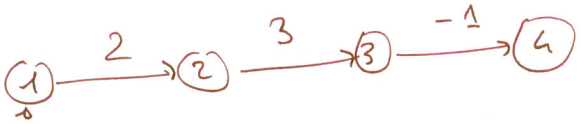
2) On peut travailler en place, sans retenir D pour les différentes valeurs de k :

```
∀x ∈ V,  D[x] ← ∞
D[s] ← 0
for k ← 1 to |V| - 1
    for (y,x) ∈ E
        D[x] ← min(D[x], D[y] + ℓ(y,x))
```

boucles $\Theta(|V| \cdot |E|)$

⚠ à l'itération k, D[x] peut contenir une distance obtenue avec plus que k arcs. Pas grave, car D ne doit plus bouger après (au plus) |V|-1 itérations.

ex.



k	D[1]	D[2]	D[3]	D[4]
0	0	∞	∞	∞
1	0	2	5	4
2	0	2	5	4
3	0	2	5	4

3) s'arrêter dès que plus rien ne bouge

l'algo passe en $O(|V| \cdot |E|)$

4) n'itérer que sur les arcs dont la source a changé de poids

```
∀x∈V,  D[x] ← ∞
D[s] ← 0,  Set1 ← {s}
for k ← 1 to |V| - 1                                   O(|V|)  ~~O(n²)~~
    if Set1 = ∅                                                 bougeons en
        break                                                   O(|V|·|E|)
    for each x in Set1                                 O(|V|)×O(|E|)
        for each y in Neighb(x)
            v ~~D[x] ← min(D[x],~~ D[y] + ℓ(y,x)~~}~~
            if D[x] =/= v
                D[x] ← v                                (*)
                Set2.insert(x)
    Set1, Set2 ← Set2, ∅
```

→ Pour retenir les chemins correspondants aux distance, il suffit de maintenir des pointeurs "père" à chaque modification de D.

(*) Father[x] ← y

avec initialement

Father[x] ← x pour tout x.

Ces liens "pères" forment un arbre couvrant du graphe s'il est connexe -

Application (de l'arbre) jeu où l'on ordonne à toutes les unités de rejoindre un même point (la source).

Application de Bellman-Ford

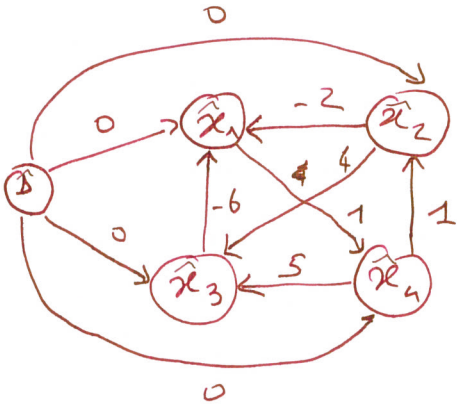
On considère un système d'équations de la forme

$$x_i - x_j \leq c_k .$$

Cela peut arriver par exemple si x_i et x_j représentent des dates d'événements et que l'événement x_i ne doit pas se produire moins de c_k jours ~~avant~~ après x_j.

Par exemple considérons

$$\begin{aligned} x_1 - x_2 &\leq -2 \\ x_1 - x_3 &\leq -6 \\ x_3 - x_4 &\leq 5 \\ x_3 - x_2 &\leq 4 \\ x_2 - x_4 &\leq 1 \\ x_4 - x_1 &\leq 1 \end{aligned}$$



On représente le système par un graphe avec

pour chaque $x_i - x_j \leq c_k$

et

pour chaque x_i

On exécute Bellman-Ford à partir de s :

s	x_1	x_2	x_3	x_4	
0	∞	∞	∞	∞	
0	0	0	0	0	
0	0	0	0	0	
0	-2	0	0	0	nœuds actifs
0	-6	0	0	0	
0	-6	0	0	0	
0	-6	0	0	-5	
0	-6	-4	0	-5	
0	-6	-4	0	-5	

nœud visité

$x_1 = -6, x_2 = -4, x_3 = 0, x_4 = -5$ est une solution du système

Mais on peut aussi tous les décaler (= changer la valeur de s), par exemple

en ajoutant 6 partout : $x_1 = 0$, $x_2 = 2$, $x_3 = 6$, $x_4 = 1$

Théorème : si le graphe construit possède un cycle négatif, alors le système d'équations n'a pas de solution.

Preuve : Sans perte de généralité, supposons qu'il existe un cycle $C = (\hat{x}_1, \hat{x}_2, \dots, \hat{x}_u, \hat{x}_1)$. Le cycle ne contient pas s car s n'a aucun arc entrant.

$(\hat{x}_1, \hat{x}_2)$ représente $x_2 - x_1 \leq c(\hat{x}_1, \hat{x}_2)$

$(\hat{x}_2, \hat{x}_3)$ $x_3 - x_2 \leq c(\hat{x}_2, \hat{x}_3)$

$\vdots$

$(\hat{x}_{u-1}, \hat{x}_u)$ $x_u - x_{u-1} \leq c(\hat{x}_{u-1}, \hat{x}_u)$

$(\hat{x}_u, \hat{x}_1)$ $x_1 - x_u \leq c(\hat{x}_u, \hat{x}_1)$

$$0 \leq \sum_{e \in C} c(e)$$

Le coût du cycle doit donc être positif pour que le système ait une solution.

Théorème : si le système possède une solution, les longueurs calculées par Bellman-Ford à partir de s en sont une.

Preuve : si la solution existe, il n'y a pas de cycle négatif et Bellman-Ford s'applique. Notons $d(\hat{x})$ la longueur affectée à $\hat{x}$ par Bellman-Ford.

Par définition de d, on a $d(\hat{x}_i) \leq d(\hat{x}_j) + c(\hat{x}_j, \hat{x}_i)$

d'où $d(\hat{x}_i) - d(\hat{x}_j) \leq c(\hat{x}_j, \hat{x}_i) = c_q$

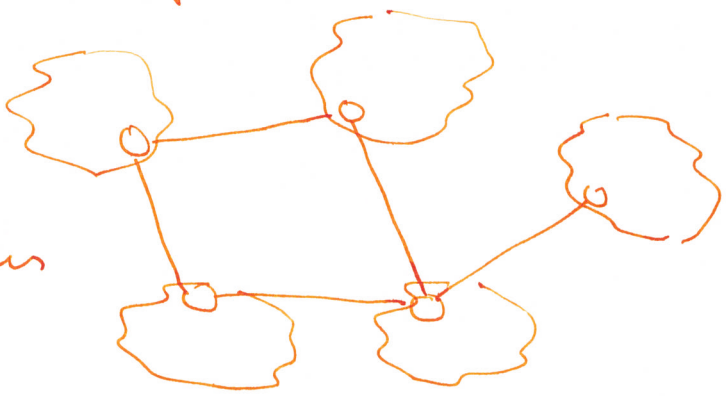
Donc poser $x_i = d(\hat{x}_i)$ satisfait bien $x_i - x_j \leq c_q$

Application dérivée de Bellman Ford : Routing Information Protocol

(/sbin/routed) RFC 1058.

Contexte : plusieurs réseaux interconnectés par des gateways.

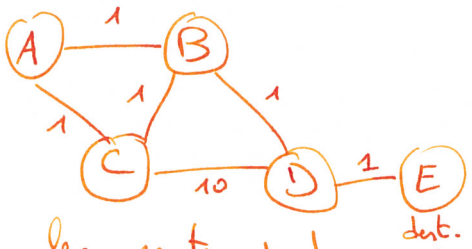
- les gateways s'échangent des infos pour maintenir les meilleures routes (estimées).



Fonctionnement

- Chaque nœud connaît ses voisins et le coût de les contacter (en général, 1).
- Chaque nœud maintient un vecteur de distance à tous les autres nœuds (accompagné de l'info "père")
- Chaque nœud envoie à ses voisins sa table de distance (toutes les 30 sec.)
- Chaque nœud met à jour sa propre table de distance lorsqu'il reçoit celle d'un voisin.
- Si un nœud n'entend pas son voisin pendant 180 sec, il marque la route comme "invalide" (distance = ∞, ou en pratique, 16) et la propagera aux voisins par la prochaine màj. Ou bien il "revalidera" la route en passant par un autre voisin à réception de son update.

problème du compte vers l'infini



regardons les routes de tous les noeuds vers E

D: connecté directement, dist = 1
B: —— via D, dist = 2
C: —— B, dist = 3
A: —— B, ——

supposons que le liens B-D meure.
voici les màj des routes vers E avec le temps →

D: direct(1)	direct(1)	direct(1)	...	direct(1)	direct(1)
B: unreach	C(4)	C(5)		C(11)	C(12)
C: B(3)	A(4)	A(5)		A(11)	D(11)
A: B(3)	C(4)	C(5)		C(11)	C(12)

Améliorations:

- "Split Horizon" : ne pas dire à un voisin ~~quelle~~ qu'on connait des routes si elles passent par lui.
- "Split Horizon with poisoned update" : si une route passe par un voisin, prétendre qu'elle vaut 16 lorsqu'on lui parle.

D: direct(1)	direct(1)	direct(1)	direct(1)
B: unreach	unreach.	C(5)	C(12)
C: B(3)	A(4)	D(11)	D(11)
A: B(3)	C(4)	unreach.	C(12)

- "Triggered Updates" faire une màj à chaque changement de métrique/route.