M1 Informatique # Protocoles réseaux

TD n.4

Couche réseau
Routage

Algorithmes de routage en plan de contrôle

Routage par vecteur de distances.

Rappel : 5 couches numérotées de 1 à 7 :

Application	(7)
Transport	(4)
Internet ou Réseau	(3)
Lien	(2)
Physique	(1)

La couche réseau est divisée en deux *plans* :

- le plan de données s'occuppe de pousser les paquets
  plusieurs millions de fois par seconde ;
- le plan de contrôle calcule par où il faut les pousser
  latence de l'ordre de la seconde.

La semaine dernière ⟶ plan de données.
Cette semaine ⟶ plan de contrôle.

Avant de transférer un paquet, un routeur consulte la table de routage afin de déterminer l'adresse IP du voisin.
Le plan de contrôle construit la table de routage.

Source : Cours 2020 Prof. Juliusz Chroboczek

Tables de routage

- Lors de ce TD, on va voir qu'est-ce que c'est **l'algorithme de Bellman-Ford distribué**.
- Quelle sont les données manipules par cet algorithme ? **les tables de routage**.
- Une table de routage c'est une table a une dimension essentiellement avec :
- A droite, un numéro d'interface et le « next-hop » (nh dans le poly). Dans le poly nh est l'adresse du « next hop ».
- A gauche : la destination.
- Donc c'est un tableau indexé par chaque destination et dedans pour chaque destination on associe un nexthop.

- Pour chaque paquet qu'elle manipule, la couche réseau effectue une décision de routage qui dépend de la destination d'un paquet.
- Pour cela, elle consulte une structure de données — la table de routage, qui conceptuellement est une fonction qui à toute adresse IP de destination affecte un voisin, c'est-à-dire une paire $(interface, IP)$:

D ↦ (if, nh)

- Il n'est bien-sûr pas raisonnable de représenter une table de routage de façon exhaustive — en IPv4 il faudrait plusieurs giga-octets de mémoire, sans même parler d'IPv6.
- La table de routage est donc compressée : elle est représentée par une structure de données qui est indexée par des préfixes, et elle associe donc des voisins à tout un préfixe sans énumérer toutes les adresses qu'il contient.
- La table de routage contient deux types d'entrées :

P ↦ (if, nh) (entrée normale)
P ↦ if (entrée connectée)

- Une entrée normale affecte le même voisin (if, nh) à chacune des adresses D d'un préfixe.
- Une entrée connectée, par contre, affecte à toute adresse D d'un préfixe le voisin (if, D) — le voisin a la même adresse que la destination, la destination est donc directement connectée.

- Par exemple, la table de routage suivante envoie les paquets du $/24$ à une destination directement connectée, et les autres à un routeur par défaut :

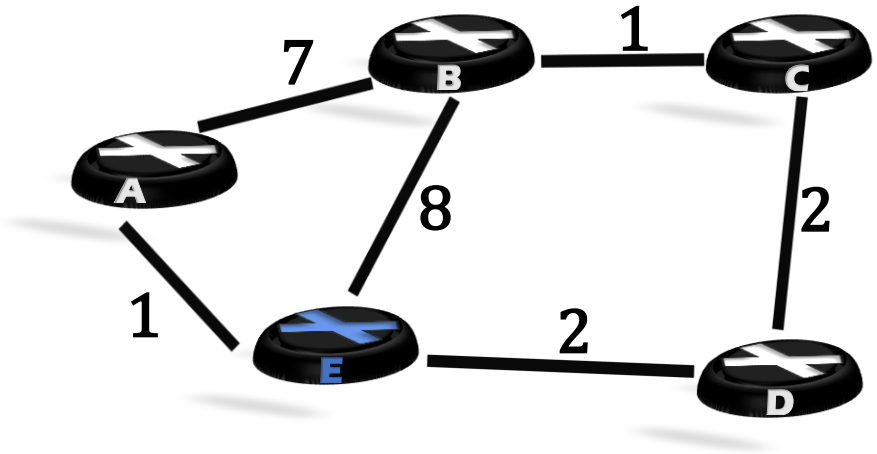
203.0.113.0/24 ↦ eth0
0.0.0.0/0 ↦ (eth0, 203.0.113.1)

- La table de routage est une structure de données ambigüe : deux préfixes peuvent se recouvrir, et donc plusieurs entrées peuvent s'appliquer à la même adresse de destination.
- Cependant, si deux préfixes distincts ont une intersection non-nulle, alors l'un est inclus dans l'autre — il y en a donc un qui est plus spécifique que l'autre, et c'est celui-ci qui s'applique. On appelle cette règle **la règle du préfixe le plus long (longest – prefix rule)**.

Source : Poly "Protocoles Réseau V" / Prof. Juliusz Chroboczek

Algorithme de routage à vecteur de distance

- Les réseaux informatiques emploient généralement des algorithmes de routage dynamique plus complexes que l'inondation, mais plus efficaces car ils trouvent les chemins les plus courts correspondant à la topologie actuelle.
- Les deux algorithmes dynamiques les plus connus sont celui à **vecteur de distance** et celui par informations d'état de lien.
- Alors que l'algorithme d'état de lien repose sur une connaissance de la topologie globale du réseau, l'algorithme à **vecteur de distance** (DV, *Distance Vector*) est de nature ***itérative***, ***asynchrone*** et ***distribuée***.
    - **Distribuée** : parce que les calculs se font au niveau de chaque nœud individuel, à partir des informations fournies par les *nœuds voisins directs*, et parce que les résultats sont partagés de la même manière.
    - **Itérative** : car ce processus se répète jusqu'à ce qu'il n'y ait plus d'informations à échanger entre nœuds voisins. De fait, cet algorithme s'interrompt de lui-même.
    - **Asynchrone** : dans la mesure où il n'impose pas à tous les nœuds de travailler ensemble.
    - Un algorithme distribué, asynchrone, itératif et à interruption automatique est plus intéressant qu'un algorithme centralisé.
- Avec les algorithmes de **routage par vecteur de distance** (*distance vector routing*), chaque routeur maintient sa propre **table de routage** : La principale structure de données dans un algorithme DV.
- **Un exemple** permet d'appréhender le contenu d'une table de distances. La topologie d'un réseau est illustrée à la figure 1, avec la table de distances présentée à droite, correspondant au nœud **E** après convergence de l'algorithme.
- Il met régulièrement à jour sa table de routage avec les informations reçues des routeurs voisins. Finalement, chaque routeur connait le meilleur lien à utiliser pour atteindre chaque destination.
- L'établissement des routes vers les meilleurs chemins est appelé **convergence**.



Destination	
A	d = 1 , nh = A
B	d = 5 , nh =D
C	d = 4 , nh =D
D	d = 2 , nh = D
E	d = 0 , nh = E

Figure 1 · Table de distances du nœud d'origine E

A	d = 1 , nh = A

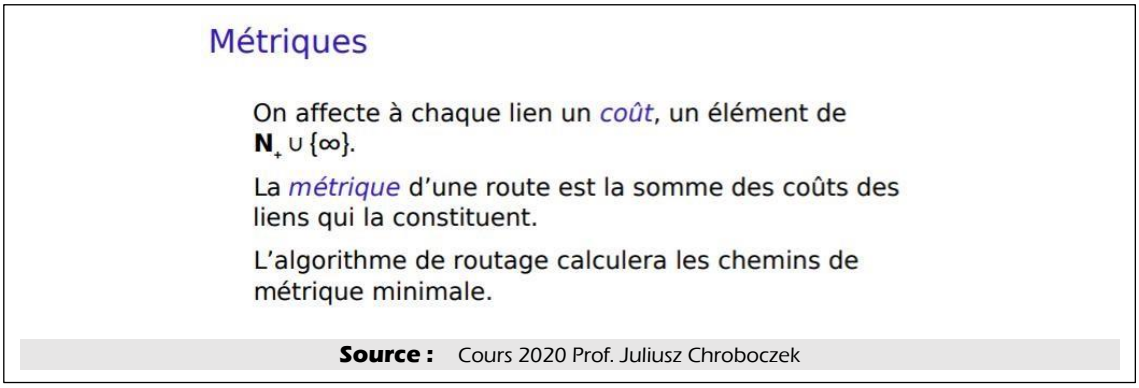
- Le cout du trajet direct de **E** vers A présente une valeur de 1.
  Par conséquent, $\boldsymbol{d_A(E) = 1}$

- Cette technique est également souvent appelée algorithme de routage de **Bellman-Ford**, en hommage aux chercheurs qui l'ont développée. C'est l'algorithme original du réseau ARPAnet, qui a également été utilisé par le protocole RIP (*Routing Information Protocol*) sur l'Internet.
  - Il est utilisé par de nombreux protocoles de routage supplémentaires, parmi lesquels les protocoles RIP et BGP de l'internet, IRDP de l'ISO, IPX de Novell.

- Dans cette technique chaque routeur maintient une table de routage indexée contenant une entrée par routeur. Chaque entrée se compose de deux parties, la ligne de sortie préférée vers la destination en question et une estimation du temps ou de la distance séparant cette destination du routeur.

- Ainsi, la **table d'acheminement** d'un nœud (qui indique la liaison de sortie appropriée pour transmettre des paquets vers une destination donnée) peut aisément être obtenue à partir de sa table de distances.

- La distance est mesurée en nombre de sauts ou toute autre métrique.



- Le routeur est supposé connaitre la distance vers chacun des routeurs adjacents. Si la métrique est le nombre de sauts, cette distance est de 1.

- Si la métrique est le délai de propagation, le routeur peut le mesurer directement en envoyant des paquets ECHO spéciaux que le destinataire retourne le plus vite possible avec des informations de temps

Sources :

Andrew Tanenbaum, David Wetherall. Computer Networks.. (Edition française)
James W. Kurose, Keith W. Ross. Computer Networking, A Top-Down Approach. (Edition française)

Algorithme de routage à vecteur de distance

Bellman-Ford distribué

- **Bellman-Ford** est un algorithme de plus court chemin (PCC). PCC c'est un arbre partant de l'origine.
- **Là on utilise Bellman-Ford dans l'autre sens, pour voir comment accéder à une destination.0**
- Ici, pour chaque destination S on calcule un arbre de façon distribué, Bellman -Ford va calculer un arbre du plus court chemin et pour cela il nous manque les distances. un arbre peut être vu comme allant vers la racine et comme partant de la racine.
- Donc, pour chaque destination on fait un Bellman-Ford.

Quelque mots à propos de l'algorithme

- On fix S,. Il faut voir que entre parentaise X c'est pas la destination, mais celui qui fait le calcul.
- L'opération de base en terme de réseaux : tous les algorithmes de routage se base sur le fait que les paquets sont à destination de ses voisins ! **c'est donc un algorithme local**, on échange avec ces voisins.
- Chaque routeur envoie à ses voisins (c'est ça qui chaque d'algo de routage a un autre, dans Bellman-Ford il envoie sa table de routage).
- C'est ça l'idée de la partie distribué, on envoie au voisin sa table de routage... faut le faire souvent mais pas trop souvent.

L'algorithme : version du poly du cours :

Le cout de la liaison directe entre X et Y

L'entrée de la table de distances du nœud X pour la destination S

- **Initialement,** $d_S(S) = 0$ et $d_S(X) = \infty$
- **Assez souvent,** si $d_S(Y) < \infty$, Y annonce $d_S(Y)$ à ses voisins.
- **Quant** X reçoit $d_S(Y)$:
  - **Si** $nh_S(X) = Y$, **alors** $d_S(X) := C_{XY} + d_S(Y)$
  - **Si** $C_{XY} + d_S(Y) < d_S(X)$ **alors** $d_S(X) := C_{XY} + d_S(Y)$ **et** $nh_S(X) := Y$
- **Timeout :** **Si** $nh_S(X) = Y$, **et** X n'entend plus Y,
  - $d_S(X) := \infty$ **et** $nh_S(X) := \perp$

En plus de son next-hop sélectionné $nh\ (X)$, chaque nœud maintient une estimation $d_S(X)$ de la métrique de la route sélectionnée. Comme on a fixé S, on peut se permettre d'écrire $nh(X)$ et $d(X)$ pour $nh_S(X)$ et $d_S(X)$ respectivement.

Source : Poly "Protocoles Réseau V" / Prof. Juliusz Chroboczek

Algorithme à vecteur de distance (DV)

Version du livre Computer Networking, A Top-Down Approach

A chaque nœud X :

```
// Initialisation
    Pour tous les nœuds adjacents v :
        D^x(*, v) = ?  // Le signe * signifie "pour toutes les lignes"
        D^x(v, v) = c(X,v)
    Pour toutes les destinations y :
        envoyer  min_w D(y, w) à chaque voisin // w à tous les voisins de X

Effectuer une boucle
    Attendre jusqu'à constater un changement de cout de ligne v  OU
             jusqu'à recevoir une mise à jour venant de v

    Si ( c(x,v) devient d) // Note : d peut être positif ou négatif
        // Changer les couts de toutes les destination via v par d
        Pour toutes les destinations y :
            D^x(y, v) = D^x(y, v) + d
    Ou bien
        // mise à jour reçue de v wrt à destination de y : le chemin le plus court de v à y a changé
        // v a transmis une nouvelle valeur pour : min_w D^v(y, w)
        // Cette nouvelle valeur est : "newval"
        Pour la destination y :
            D^x(y, v) = c(X, v) 1 newval

    Si nouveau min_w D^X(y, w) pour toute destination y
        envoyer une nouvelle valeur de min_w D^X(y, w) à tous les voisins
Sans cesse
```

Par exemple, Imaginez un nœud X souhaitant acheminer un paquet vers une destination Y *via* son voisin direct Z.

- L'entrée de la table de distances du nœud X, $D^x(Y,Z)$ correspond à la somme du cout de la liaison directe entre X et Z, $c(X,Z)$ et de la valeur du parcours le moins onéreux actuel entre Z et Y, ce qui conduit à l'expression :

$$D^x(Y,Z) = c(X,Z) + min_w\{D^z(Y,w)\}$$

dans laquelle le terme min_w est choisi parmi tous les voisins directs de Z (y compris X).

- Cette équation donne une idée du type de communication de voisinage qui s'établit avec l'algorithme DV, dans lequel chaque nœud doit connaitre la valeur du parcours le moins onéreux de tous ses voisins pour toutes les destinations

possibles. Dès qu'un nœud trouve un nouveau parcours moins onéreux (couteux) vers une destination donnée, il est chargé d'en informer tous ses voisins.

- **Par exemple,** *imaginons que la métrique utilisée soit le délai d'acheminement et que chaque routeur connaisse le temps qu'il met pour atteindre chacun de ses routeurs voisins.*
  - Toutes les T ms, chaque routeur envoie à chacun des routeurs contigus une liste des délais estimés pour atteindre chaque destination proposée.
  - Supposons que l'une de ces tables vienne d'arriver de la part du routeur voisin X, avec X_i le délai estimé pour atteindre le routeur i.
  - Si le routeur récepteur sait que le délai qui le sépare de X est m ms, il sait aussi qu'il peut atteindre i par l'intermédiaire de X avec un délai de $X_i + m$ ms.
  - En effectuant ce calcul pour chaque routeur adjacent, un routeur peut déterminer le délai estimé qui lui semble le meilleur et le porter dans sa table de routage en y associant le lien de sortie appropriée.
  - Notez que l'ancienne table de routage n'est pas utilisée dans le calcul.

Source :
James W. Kurose, Keith W. Ross. Computer Networking, A Top-Down Approach. (Edition française)

Exercice 1 algorithme de vecteur de distances

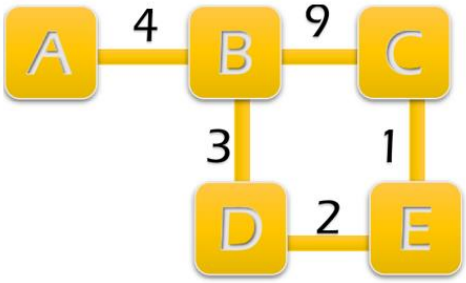
1#

Faites tourner l'algorithme de vecteur de distances, A étant la source.

On va router en destination de A. Il est écrit que "A est la source".
Dans l'algo de Bellman-Ford ça veut dire que A est la destination.

L'algorithme : version du poly du cours :

- **Initialement,** $d_S(S) = 0$ et $d_S(X) = \infty$
- **Assez souvent,** si $d_S(Y) < \infty$, Y annonce $d_S(Y)$ à ses voisins.
- **Quant** X reçoit $d_S(Y)$:
  - **Si** $nh_S(X) = Y$, **alors** $d_S(X) := C_{XY} + d_S(Y)$
  - **Si** $C_{XY} + d_S(Y) < d_S(X)$ **alors** $d_S(X) := C_{XY} + d_S(Y)$ **et** $nh_S(X) := Y$
- **Timeout :** Si $nh_S(X) = Y$, **et** X n'entend plus Y,
  - $d_S(X) := \infty$ **et** $nh_S(X) := \perp$



La c'est nous qui avons choisi l'ordre

Temp	0	1	2	3	4	5	6
Qui annonce ?	-	B	A	B	C	D	E
A	d = 0, nh = A	d = 0, nh = A	d = 0, nh = A	d = 0, nh = A	d = 0, nh = A	d = 0, nh = A	d = 0, nh = A
B	d = ∞, nh = ⊥	d = ∞, nh = ⊥	d = 4, nh = A	d = 4, nh = A	d = 4, nh = A	d = 4, nh = A	d = 4, nh = A
C	d = ∞, nh = ⊥	d = ∞, nh = ⊥	d = ∞, nh = ⊥	d = 13, nh = B	d = 13, nh = B	d = 13, nh = B	d = 10, nh = E
D	d = ∞, nh = ⊥	d = ∞, nh = ⊥	d = ∞, nh = ⊥	d = 7, nh = B	d = 7, nh = B	d = 7, nh = B	d = 7, nh = B
E	d = ∞, nh = ⊥	d = ∞, nh = ⊥	d = ∞, nh = ⊥	d = ∞, nh = ⊥	d = 14, nh = C	d = 9, nh = D	d = 9, nh = D

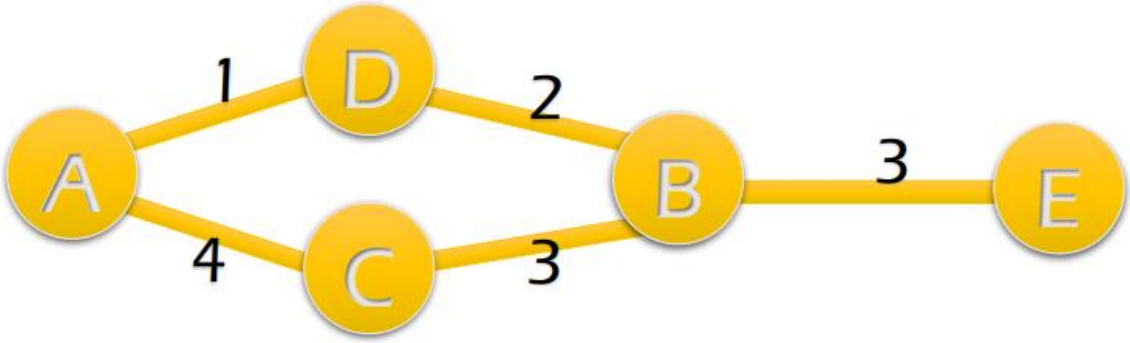
Rien ne change

Inconnu

Exercice 2 : un état étrange

#

On considère l'algorithme de vecteur de distances.
À un certain instant on a le réseau de cinq routeurs et les tables de routage suivants :



Dans cet exercice on nous demande si c'est possible que les tables de routage soient comme ça ?
C.-à-d. : on a juste nommé les distances, les next-hop sont pas écrit. Donc la question 2 consiste à rajouter les next-hop.
Question 1 : exécution.

1#

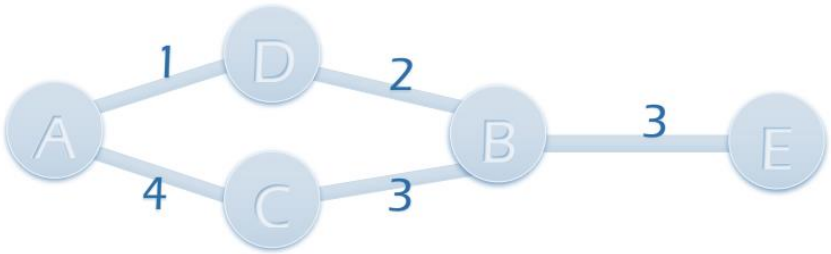
Un tel état est-il possible (il peut y avoir des pertes de paquets) ?

Si oui décrire le scénario qui y conduit (qui a échangé avec qui dans quel ordre) ;
Si non expliquer pourquoi (en précisant ce qui impossible).

2#

Il manque le champ next hop dans les tables de routage. L'ajouter avec sa valeur.

Destination	La table de routage de A	
B	d = 7 , nh = C	
C	d = 4 , nh = C	
D	d = ∞ , nh = ⊥	
E	d = 10 , nh = C	



	La table de routage de A	pas optimal / optimal
B	d = 7 , nh = ?	pas optimal
C	d = 4 , nh = ?	optimal
D	d = ∞ , nh = ?	pas optimal
E	d = 10 , nh = ?	pas optimal

Conclusion : table de routage pas optimal.

Mais, une tel table de routage est possible ?

peut-il arrivé un scénario qui amène a cette table de routage ?

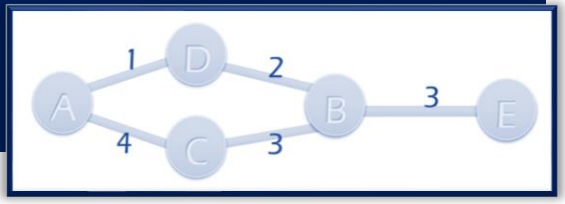
Ca na pas été possible s'il y avait une distance qui n'est pas la longueur d'une route.

	La table de routage de A	pas optimal / optimal	possible / pas possible
B	d = 7 , nh = ?	pas optimal	possible si le next-hop est C
C	d = 4 , nh = ?	optimal	possible si le next-hop est C
D	d = ∞ , nh = ?	pas optimal	
E	d = 10 , nh = ?	pas optimal	possible si le next-hop est C

Donc, c'est possible, mais faut construire un scénario.

Scénario 1

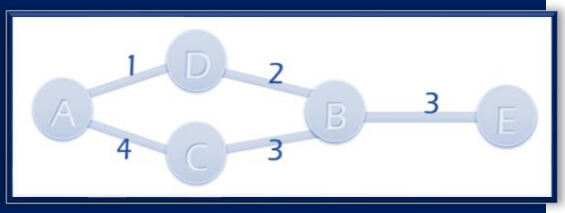
- D est mort / absent. A, B, C, E convergent bien.
- D arrive, émet, mais paquet vers A perdu.
- B émet
- C émet
- A émet



Scénario 2

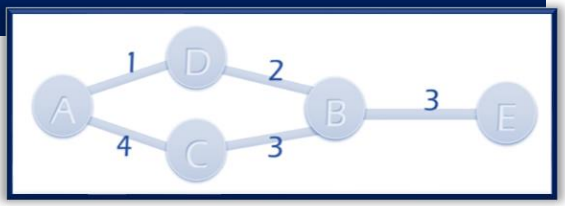
- Lien AD et DB morts.
- DB up.
- AD up.

Mais D n'a pas encore émis.

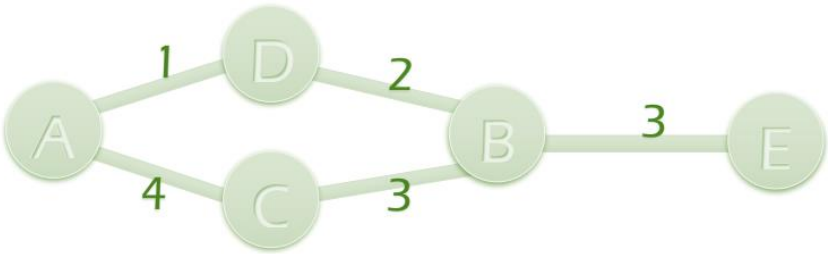


Scénario 3 (encore plus simple que le scénario 2)

- Lien AD mort.
- AD up mais D n'a pas encore émis.



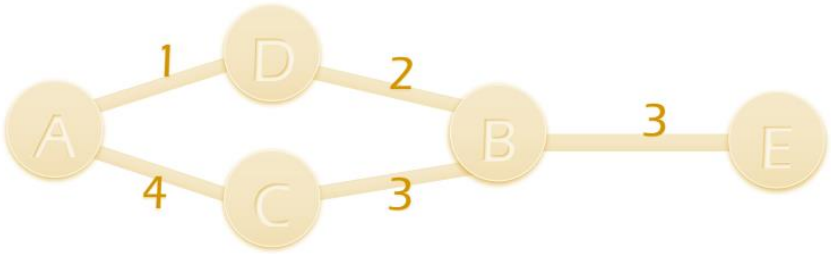
Destination	La table de routage de B
A	d = 3 , nh = D
C	d = 3 , nh = C
D	d = 2 , nh = D
E	d = 3 , nh = E



	La table de routage de B	pas optimal / optimal
A	d = 3 , nh = ?	optimal
C	d = 3 , nh = ?	optimal
D	d = 2 , nh = ?	optimal
E	d = 3 , nh = ?	optimal

Conclusion : table de routage optimal.

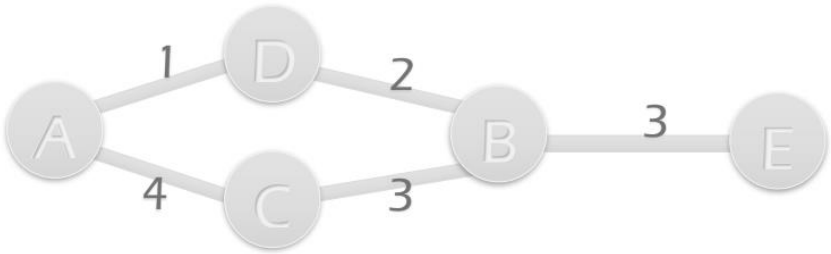
Destination	La table de routage de C	
A	d = 4 , nh = A	
B	d = 3 , nh = B	
D	d = 5, nh = A (nh peut être aussi B)	
E	d = 6 , nh = B	



	La table de routage de C	pas optimal / optimal
A	d = 4 , nh = ?	optimal
B	d = 3 , nh = ?	optimal
D	d = 5, nh = ?	optimal
E	d = 6 , nh = ?	optimal

Conclusion : table de routage optimal.

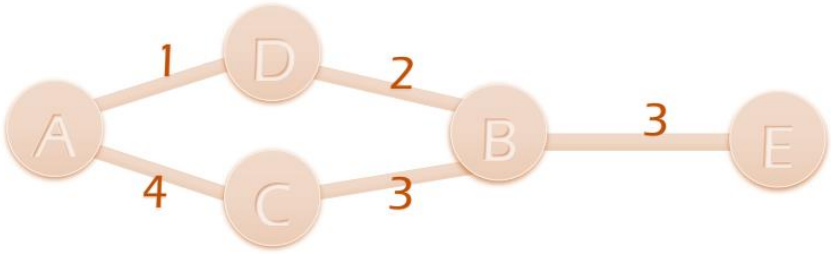
Destination	La table de routage de D	
A	d = 1 , nh = A	
B	d = 2 , nh = B	
C	d = 5, nh = B (nh peut être aussi A)	
E	d = 5 , nh = B	



	La table de routage de D	pas optimal / optimal
A	d = 1 , nh = ?	optimal
B	d = 2 , nh = ?	optimal
C	d = 5, nh = ?	optimal
E	d = 5 , nh = ?	optimal

Conclusion : table de routage optimal.

Destination	La table de routage de E	
	A	d = 6 , nh = B
	B	d = 3 , nh = B
	C	d = 6, nh = B
	D	d = 5 , nh = B



	La table de routage de E	pas optimal / optimal
A	d = 6 , nh = ?	optimal
B	d = 3 , nh = ?	optimal
C	d = 6, nh = ?	optimal
D	d = 5 , nh = ?	optimal

Conclusion : table de routage optimal.

Exercice 3 : convergence statique

On se place dans le cas d'un réseau statique (le coût des liens ne bouge pas), à l'état initial (tous les routeurs viennent de démarrer) et où toutes les sources sont connues de tout le monde.
(c.-à-d. tlm connait tlm, tlm connait qui il y a sur le réseau)

1#

Montrer que pour tous S et Y, $d_S(Y)$ ne fait que diminuer au cours du temps

$d_S(Y)$: La distance de Y à S

Le relâchement fait seulement baisser la distance.
Statique ⟹ C_{xy} ne bouge pas, donc « **Si** $nh_X(S) == Y$ » ne fait pas monter la distance.

2ème cas impossible (ne fait que diminuer)

$$\underbrace{X \text{ nœuds}}_{d(X) > d(X)} \rightarrow Y \text{ a augmente la valeur sucesseur}$$

Soit X le premier à augmenter d(X) Si d' > d

Impossible

1ère cas

C'est parce que $d'(X) = C_{xy'} + d(Y')$ où $C_{xy'} + d(Y') < d(X)$

2ème cas

$d'(X) = C_{yx} + d'(Y)$

$d'(Y) > d(Y)$, impossible car x est le premier à augmenter sa métrique

Donc la métrique diminue → donc on converge.

2#

Montrer que $d_S(S)$ et $nh_S(S)$ ont déjà les bonnes valeurs (celles obtenues après convergence de l'algorithme) à l'état initial.

$d_S(S) = 0 \rightarrow$ imédiat

Dans Bellman-Ford : d < 0 ⟹ boucle.
Si on prend l'hyp (hypothèse) de cours « > 0 » : $d_S(X) \geq 0$
on a $d_S(S)$ initialiser à 0, ne peut pas baisser.

3#

Supposons qu'une plus courte route de A_1 à A_k soit $A_1, A_2, \ldots, A_k$, et $d_{A_k}(A_i)$ soit optimale pour tout $i \in \{2, \ldots k\}$.

Montrer que quand A_2 envoie sa table de routage à A_1, A_1 calcule le coût optimal de la route minimale vers A_k.

$$A_1 - \underbrace{A_2 - A_3 - \ldots A_k}_{convergé}$$

$$d \rightarrow deja\ optimal$$

A_2 annonce à $A_1 \rightarrow A_1$ aura un cout immédiat.

Remarque : Avant l'annonce $\boldsymbol{d(A_1)} \geq$ à la valeur optimale.

2ème cas : soit $\boldsymbol{d_{A_k}(A_1)}$ est déjà optimale → donc rien ne change.

soit $\boldsymbol{d(A_1)}$ devient égal à $C_{A_1A_2} + \underbrace{d(A_2)}_{optimale}$, optimale.

4#

En déduire la convergence de l'algorithme sur un réseau statique, en raisonnant par induction sur la longueur des routes de coût minimal.

Récurrence sur le nombre k de hops de la route (cf cas c3, preuve Bellman-Ford, Dijkstra...)

Par récurrence

On veut démontrer que pour tout x, d(x) atteint la valeur optimal.

Démonstration : par récurrence sur la longueur x.

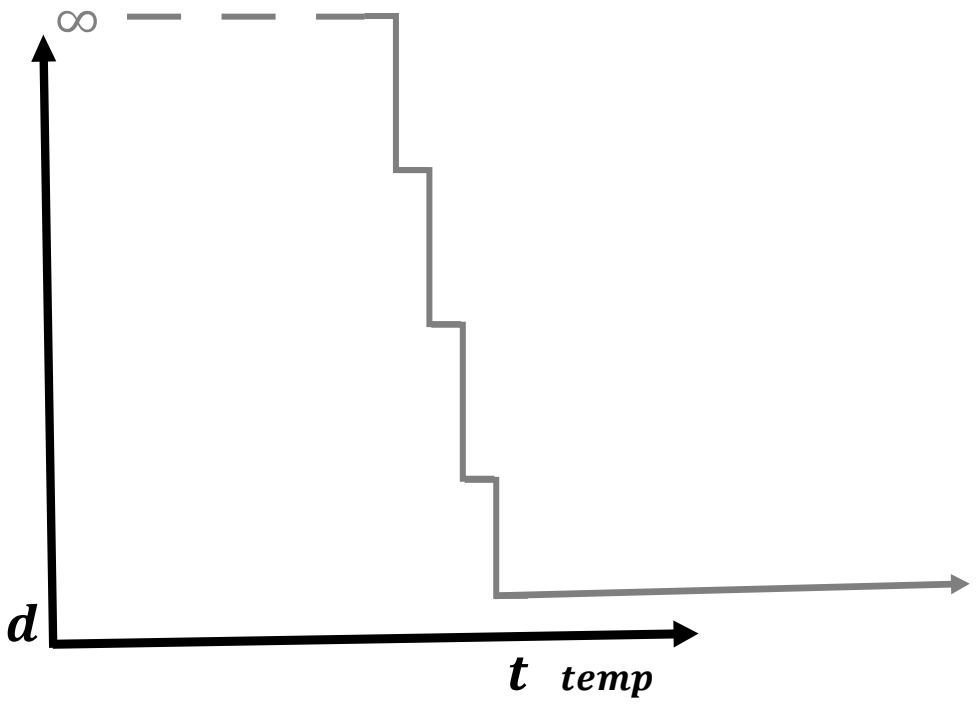
- **Cas de base** : x est la source (question 2).
- **Cas inductif** : question 3

5#

Comment un routeur sait-il que l'algorithme a convergé vers des chemins de métrique minimale, c'est -à-dire que sa table de routage est correcte ?

Comment on sait qu'on a convergé vers les bonnes distances ?

On avait vu que typiquement la distance au cours du temp pour un routeur elle commence à l'infini et puit ça baisse par paliers (étages. phases), et puit après il y a la valeur correcte au bout d'un certain temp **t**.



Au temp **t** le routeur a la valeur correcte. Comment on le sait ? Comment savoir qu'on a arrivé à la valeur la plus petite ?

Le routeur ne connait même pas les chemins en fait. Il connait ses voisins. Quand un package sort de notre ordinateur pour aller chez Google, tous ce qu'il sait, le package, c'est qu'il doit aller vers notre Freebox, mais il ne sait pas où il doit aller par la suite.

Donc, on ne peut jamais être sûr qu'on a testé tous les chemins, de plus que l'internet c'est grand.

La réponse : C'est une question piège, on ne sait pas. Donc la réponse est : jamais.

Ce genre d'algorithme, tous ce qu'on sait, c'est qu'il y a la propriété de stabilisation, On veut en perméance l'optimum, mais on sait pas du tout qu'on a l'optimum.

Par exemple, dans l'exercice 2, au début, selon notre scénario, le lien AD était mort donc le routeur A avait des routes incorrectes. Quand le lien s'allume, dans combien de temps il va être au courant ? ça dépend de la question au bout de combien de temp on propage et etc, donc en fait on n'a pas

de borne et l'algorithme est asynchrone, de plus que dans la réalité, la situation est dynamique et il y a sens arrête des nœuds qui arrives et qui partes.

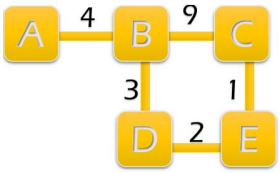
CCL(Conclusion) : quand les couts ne bougent pas, l'algo converge vers ce qu'il faut.
➔ Bellman-Ford correct en cout statique (pas le cas d'Internet).

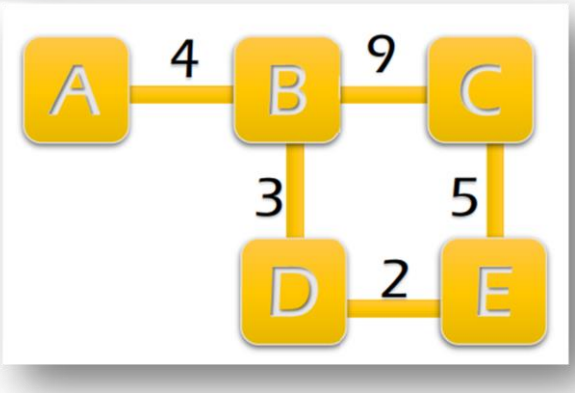
Exercice 4 dynamique

1#

Dans l'exercice 1, le coût du lien C-E passe à 5.

Que se passe-t-il ?





A	d = 0, nh = A
B	d = 4, nh = A
C	d = 10, nh = E
D	d = 7, nh = B
E	d = 9, nh = D

A	d = 0, nh = A
B	d = 4, nh = A
C	d = 14, nh = E
D	d = 7, nh = B
E	d = 9, nh = D

On a des routes qui ne sont pas optimal :

- $C \longrightarrow A$: non optimale car on passe par E et ça nous donne 14 alors qu'on peut faire 13 en passant par B.
- $A \longrightarrow C$: (le graphe est symétrique).

- $B \longrightarrow C$: non optimale (10 VS. 9)[1].
- $C \longrightarrow B$: (le graphe est symétrique).

[1] Car on continue a passer par E ce qui couté avant 6 (< 9 pour le lien direct B-C), mais maintenant que C-E coute 5 alors le cout est 10 (> 9 pour le lien direct B-C).

2#

A quoi sert la ligne

$$\textbf{si } nh_S(Y) = X, \quad \textbf{alors } d_S(Y) := c_{YX} + d_S(X)$$

de l'algorithme ?

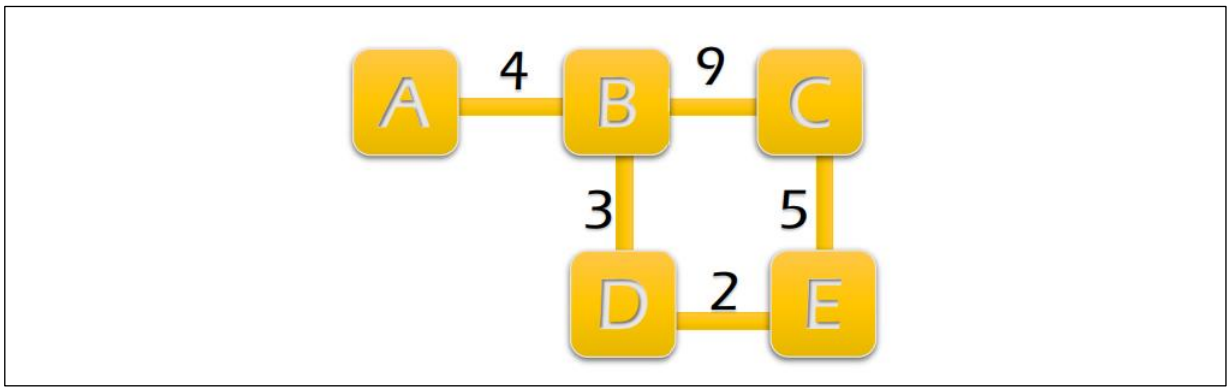
Si on n'aurais pas cette ligne, les distances ne serait pas mise à jour.

Par exemple,

C croit qu'il est une route de longueur 10 vers **A**,
actuellement dans la table de routage de **C** on a :

$$\mathbf{d_A(C) = 10}$$

alors que en réalité c'est de longueur 14.



- **Temp 1 :** reçoit $d_A(E) = 9$, update $d_A(C) = 9 + 5 = 14$
- **Temp 2 :** C reçoit $nh_A(C) = B$ et met à jour $d_A(C) = 4 + 9 = 13$

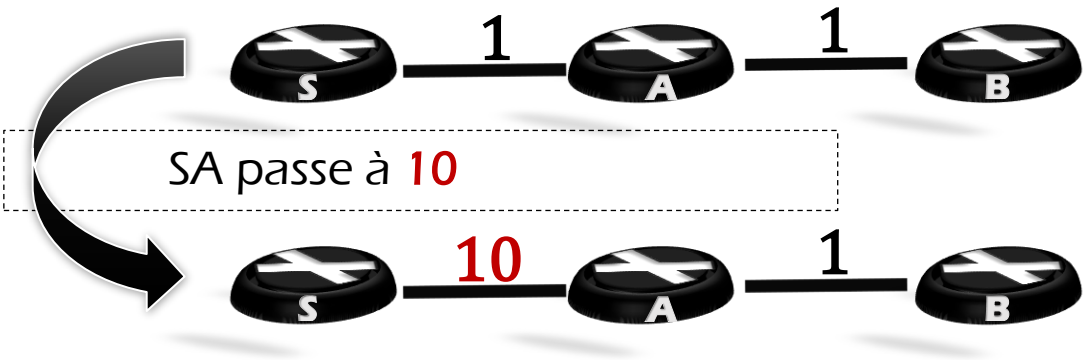
L'algorithme : version du poly du cours :

- **Initialement,** $d_S(S) = 0$ et $d_S(X) = \infty$
- **Assez souvent,** si $d_S(Y) < \infty$, Y annonce $d_S(Y)$ à ses voisins.
- **Quant** X reçoit $d_S(Y)$:
  - **Si** $nh_S(X) = Y$, **alors** $d_S(X) := c_{XY} + d_S(Y)$
  - **Si** $c_{XY} + d_S(Y) < d_S(X)$ **alors** $d_S(X) := c_{XY} + d_S(Y)$ **et** $nh_S(X) := Y$
- **Timeout :** **Si** $nh_S(X) = Y$, **et** X n'entend plus Y,
  - $d_S(X) := \infty$ **et** $nh_S(X) := \perp$

3#

Donner un scénario où une boucle de routage apparait.

Comment disparaît-elle ?

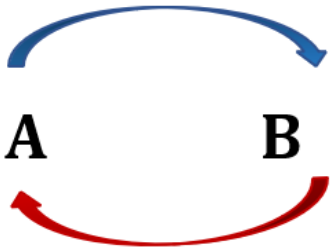


Temp	0	1	2
	SA passe a 10		Boucle !
Qui annonce ?	-	S	B
S	d = 0, nh = S	d = 0, nh = S	d = 0, nh = S
A	d = 1, nh = S	d = 10, nh = S	d = 3, nh = B
B	d = 2, nh = A	d = 2, nh = A	d = 2, nh = A

Résultat :

Boucle !

$nh_s(A) = B$ $nh_s(B) = A$



A calcule $d_s(A) = 1 + 2 = 3 < 10$ et maj (met à jour) $nh_s(A) = B$

L'algorithme : version du poly du cours :

- **Initialement,** $d_S(S) = 0$ et $d_S(X) = \infty$
- **Assez souvent,** si $d_S(Y) < \infty$, Y annonce $d_S(Y)$ à ses voisins.
- **Quant** X reçoit $d_S(Y)$:
  - **Si** $nh_S(X) = Y$, **alors** $d_S(X) := C_{XY} + d_S(Y)$
  - **Si** $C_{XY} + d_S(Y) < d_S(X)$ **alors** $d_S(X) := C_{XY} + d_S(Y)$ **et** $nh_S(X) := Y$
- **Timeout :** **Si** $nh_S(X) = Y$, **et** X n'entend plus Y,
  - $d_S(X) := \infty$ **et** $nh_S(X) := \perp$

Disparition de la boucle :

Temp	0	1	2	3	4	5	6
	Boucle !						
Qui annonce ?	-	A	B	A	B	A	B
S	d = 0, nh = S	d = 0, nh = S	d = 0, nh = S	d = 0, nh = S	d = 0, nh = S	d = 0, nh = S	d = 0, nh = S
A	d = 3, nh = B	d = 3, nh = B	d = 5, nh = B	d = 5, nh = B	d = 7, nh = B	d = 7, nh = B	d = 9, nh = B
B	d = 2, nh = A	d = 4, nh = A	d = 4, nh = A	d = 6, nh = A	d = 6, nh = A	d = 8, nh = A	d = 8, nh = A
	$d_s(B) = 3 + 1 = 4$ (car $nh_s(B) = A$)	$d_s(A) = 4 + 1$	$d_s(B) = 5 + 1 = 6$	$d_s(A) = 6 + 1$	$d_s(B) = 7 + 1 = 8$	$d_s(A) = 8 + 1 = 9$	

Temp	7	8	9	10
			Disparition de la boucle	
Qui annonce ?	A	B	S	A
S	d = 0, nh = S	d = 0, nh = S	d = 0, nh = S	d = 0, nh = S
A	d = 9, nh = B	d = 11, nh = B	d = 10, nh = S	d = 10, nh = S
B	d = 10, nh = A	d = 10, nh = A	d = 10, nh = A	d = 11, nh = A
	$d_s(B) = 9 + 1 = 10$	$d_s(A) = 10 + 1 = 11$		

L'algorithme : version du poly du cours :

- **Initialement,** $d_S(S) = 0$ et $d_S(X) = \infty$
- **Assez souvent,** si $d_S(Y) < \infty$, Y annonce $d_S(Y)$ à ses voisins.
- **Quant** X reçoit $d_S(Y)$:
  - **Si** $nh_S(X) = Y$, **alors** $d_S(X) := C_{XY} + d_S(Y)$
  - **Si** $C_{XY} + d_S(Y) < d_S(X)$ **alors** $d_S(X) := C_{XY} + d_S(Y)$ **et** $nh_S(X) := Y$
- **Timeout :** **Si** $nh_S(X) = Y$, **et** X n'entend plus Y,
  - $d_S(X) := \infty$ **et** $nh_S(X) := \perp$

Exercice 5 : routeur défectueux

#

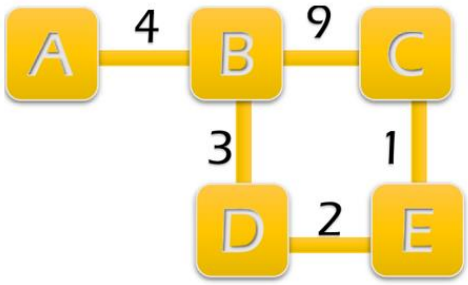
Un des routeurs a des problèmes de *boot* :

- **Au démarrage sa table est remplie de valeurs quelconques.**
- **Mais après, il effectue l'algorithme de vecteur de distances correctement.**

Que se passe-t-il ?

Qu'est-ce qu'il se passe si on démarre avec n'importe quoi dans le mémoire ? Est-que ça peut être gênent ? Le fait qu'au début la table est mal initialisé – peut causer des problèmes ou pas ?

Reprenons l'exemple de l'exercice 1 :



A annonce a B que sa distance a C est 0, 0 c'est un petit nombre, B met à jour 4+0 = 4, donc on a une boucle de routage qui se créé.

A annonce à B que sa distance à C est 0 : $d_C(A) = 0$

B met à jour : $nh_C(B) = A \quad d_C(B) = 4$

On peut dire que si c'était comme ça alors B aussi enverrai ses paquets...

Lorsque on annonce des routes petites, on va toucher un tas de paquets incorrectement, Si par exemple, dans le monde globale, si vous dites « je suis à distance minuscule au sites de routeurs américains.... » ça s'appelle une attaque ..... . (parfois c'est juste de mauvaises configuration). A l'Internet ça marche car la plus part de personne sont de bonne volonté.

Exercice 6 : l'horizon scindé

« Split horizon »

Idée : éviter boucle de routage.

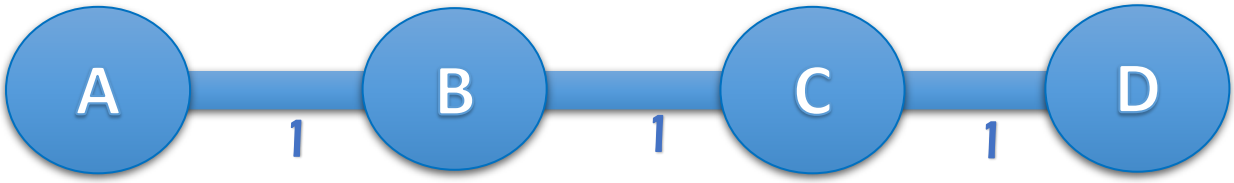
→ Ça sert à rien d'envoyer sa distance à quelqu'un par qui on passe pour aller à S.

B n'envoit pas $d_s(B)$ car $nh_s(B) = A$

Mais ne résout pas la topologie de la question 2.

Seul moyen : comptage à l'infini.

On a la topologie suivante :



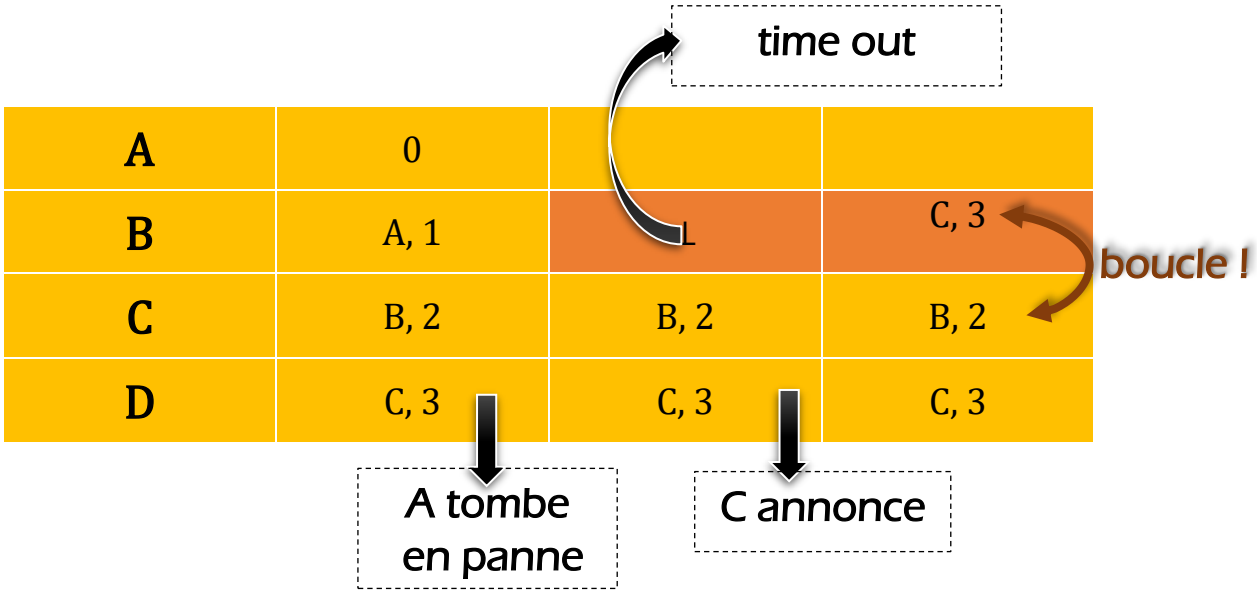
1 (a)#

Après que l'algorithme de vecteur de distances ait convergé, donnez les valeurs de $nh_A(X)$ et $d_A(X)$ pour $X = B, C, D$.

A	0
B	A, 1
C	B, 2
D	C, 3

1 (b)#

On suppose maintenant A est tombée en panne. Proposez un scénario où la valeur de $d_A(X)$ pour $X = B, C, D$ augmente à l'infini.

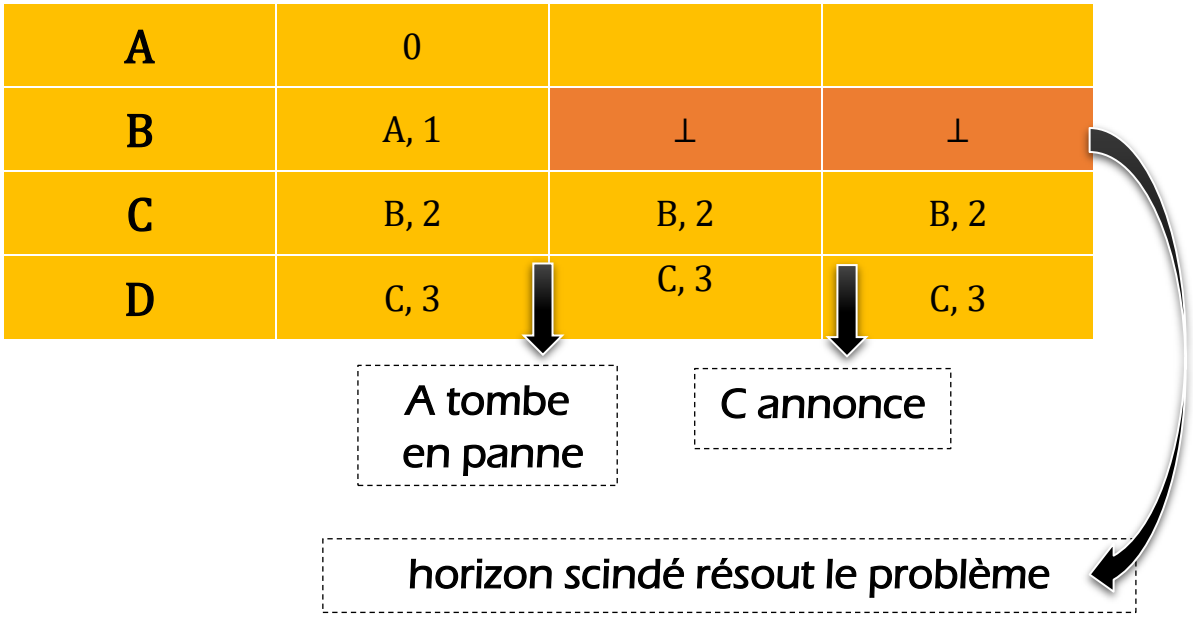


1 (c)#

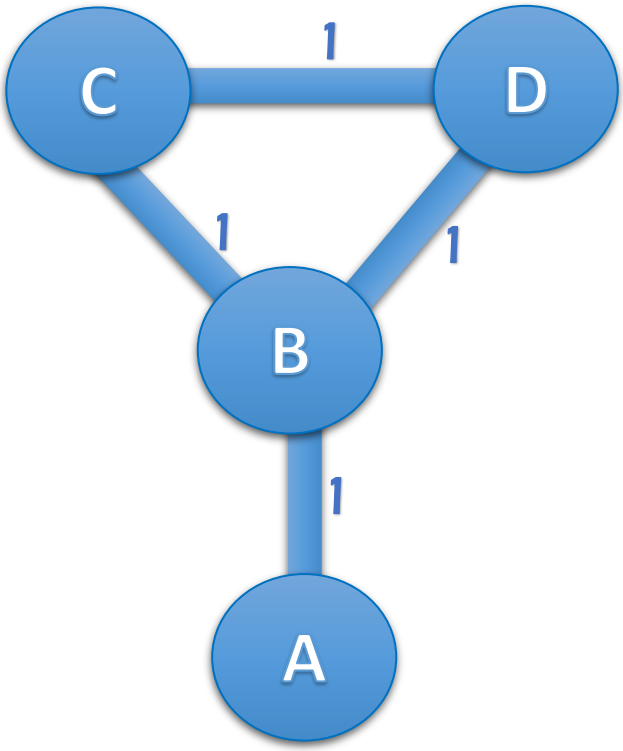
L'horizon scindé avec poison inverse consiste, pour un routeur X donné, à ne pas annoncer la vraie valeur de $d_S(X)$ à son voisin Y lorsque $nh_S(X) = Y$. Il annonce à Y que $d_S(X) = \infty$.

Le scénario de la question précédente est-il toujours possible ?

Broadcast tous les interfaces autres que le *next hop*.



On a maintenant la topologie suivante :



2(a)#

Après que l'algorithme de vecteur de distances ait convergé, donnez les valeurs de $nh_A(X)$ et $d_A(X)$ pour $X = B, C, D$.