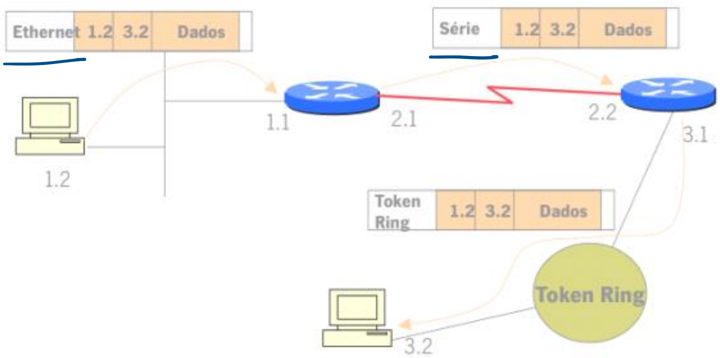
Encaminhamento

23 de janeiro de 2024 23:54

- **Conceitos**
  - Processo de *Forwarding*
  - Processo de *Routing*
- **Algoritmos de encaminhamento dinâmico**
  - Estado de Ligação (LS)
  - Vetores de Distância (DV)
  - Comparação entre DV e LS
- **Protocolos de encaminhamento IP**
  - Protocolos de encaminhamento interno (IGP)
  - Protocolos de encaminhamento externo (EGP)

Routers **armazenam e reenviam datagramas IP**

"Desencapsula" no interface de entrada, encapsula no interface de saída, de acordo com o tipo de interface...



Forwarding

↳ Utiliza a tabela de reenvio (previamente preenchida)

destino → Próximo salto , Interface de saída

Encaminhamento (routing)

→ preenche a tabela com as melhores rotas para as redes de destino / ou conjunto de prefixos de endereços

↳ Processo MANUAL / AUTOMÁTICO

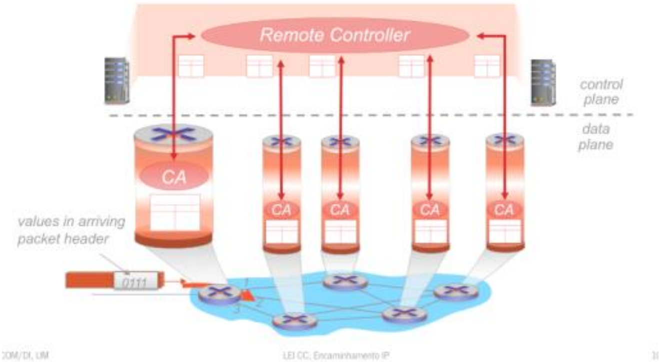
↓

Protocolo de encaminhamento dinâmico

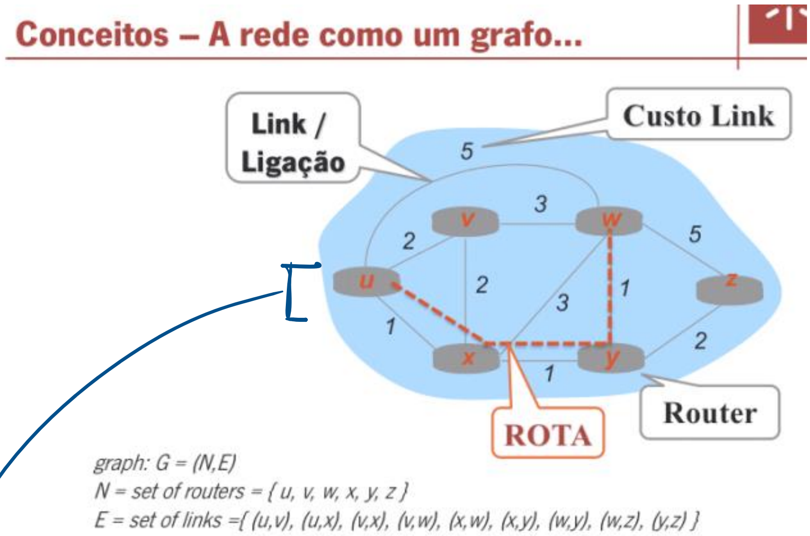
- Algoritmos de routing

• Algoritmos de routing são distribuídos

REMOTE CONTROLER
Interage com agentes locais (CA)



Algoritmo de Encaminhamento / Routing
Dada uma topologia de rede (um conjunto de encaminhadores com ligações de rede a interligá-los), representável como um grafo com pesos nos arcos / ligações dos seus nós, o objetivo do algoritmo de encaminhamento é determinar um "bom" caminho desde o nó fonte/origem até ao nó destino.



→ caminhos de custo mínimo

Nó u

Destino	Próximo Nó	Link	Custo
u	u	—	0
...	...	...	...
x	x	L_{ux}	1
...	...	...	...
w	x	L_{ux}	3

Tipos de Algoritmos

Global

- Todos os encaminhadores têm um conhecimento completo da topologia e custo das ligações;
- Algoritmos de estado das ligações (***Link State* – LS**).

Descentralizada

- Os encaminhadores só conhecem os vizinhos a que estão fisicamente/logicamente ligados e o custo das ligações respetivas;
- O processo de computação é iterativo, havendo troca de informação entre vizinhos;
- Algoritmos de vetor de distância (***Distance Vector* – DV**).

Link State (LS)

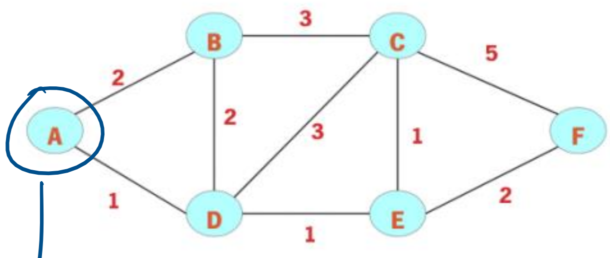
→ Base de dados topológica
- Reliable Flooding

- Inicialmente necessitam de conhecer apenas os seus vizinhos diretos, para quem enviam a identificação de todos os outros seus vizinhos, bem como o custo das ligações respetivas;
- Um nó/encaminhador ao receber esta informação atualiza a sua base de dados topológica e reenvia a informação para todos os seus vizinhos;
- Ao fim de algum tempo todos os nós possuem um conhecimento completo da topologia e dos custos de todas as ligações.

- Com esta informação, em cada nó/encaminhador, é executado um algoritmo de descoberta dos caminhos de custo mínimo, normalmente o algoritmo de **Dijkstra**.
- Com o resultado obtido da aplicação do algoritmo anterior é preenchida a tabela de encaminhamento do nó.

Algoritmos LS

- Usando um algoritmo de estado da ligação, qual a visão topológica da rede que tem o nó A no final da primeira iteração de troca de informação (*Link State Announcement* – LSA)?
- Seria correto calcular as rotas nesta fase? Quando seria?



Visão topológica de A depois da 1ª iteração depois da 1ª iteração

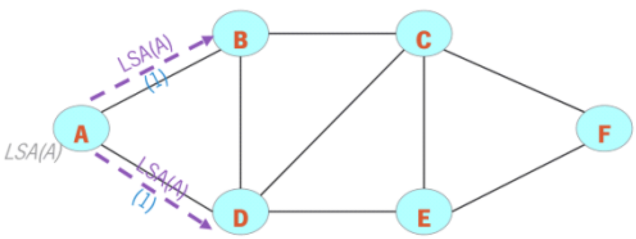
1. Cada nó gera uma mensagem LSA com o estado dos seus links

LSA(A) = { (A→B, 2); (A→D, 1) }, ... , LSA(F) = { (F→C, 5); (F→E, 2) }

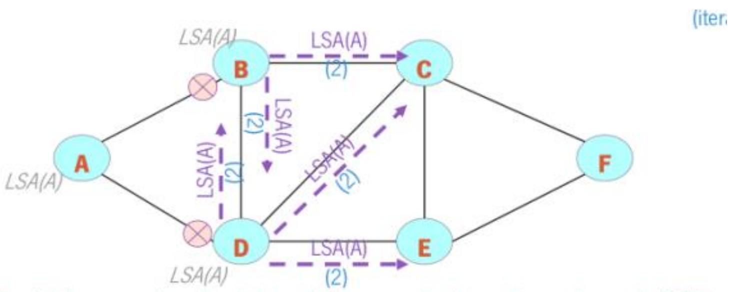
Grafo da rede = { todos os LSAs de todos os nós }

Conhecimento sobre os vizinhos

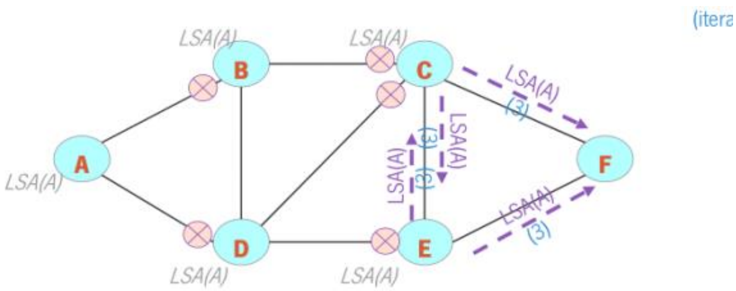
2. Cada nó tem de enviar a sua mensagem LSA a todos os outros (Flooding) -- exemplo apenas para as mensagens de A



-→ O nó A vai enviar a todos os vizinhos...

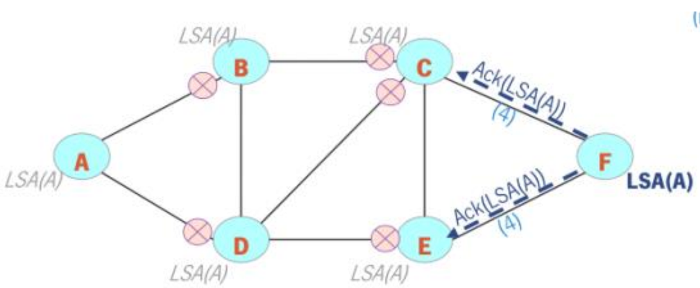


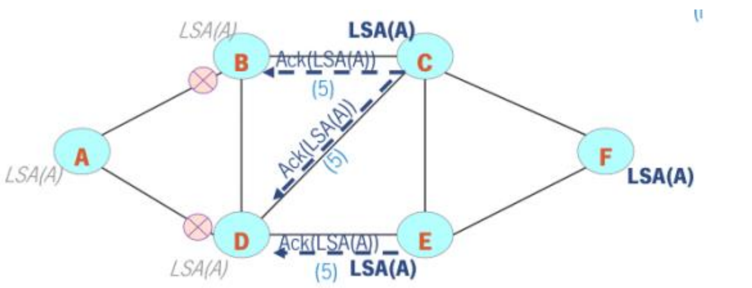
Os vizinhos reenviam também a todos, **exceto** de onde receberam! (RPF)

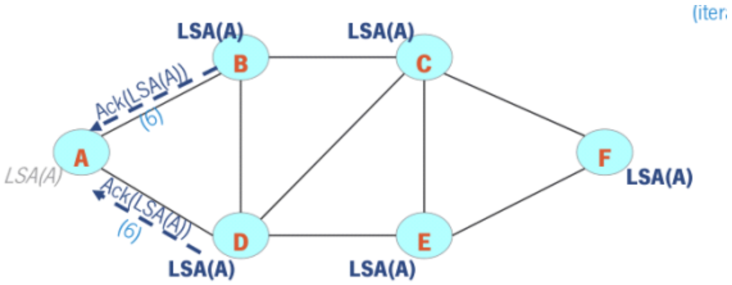


Os vizinhos reenviam também a todos, **exceto** de onde receberam! (RPF)

3. A receção tem de ser confirmada de volta até ao A (certeza que chegou)







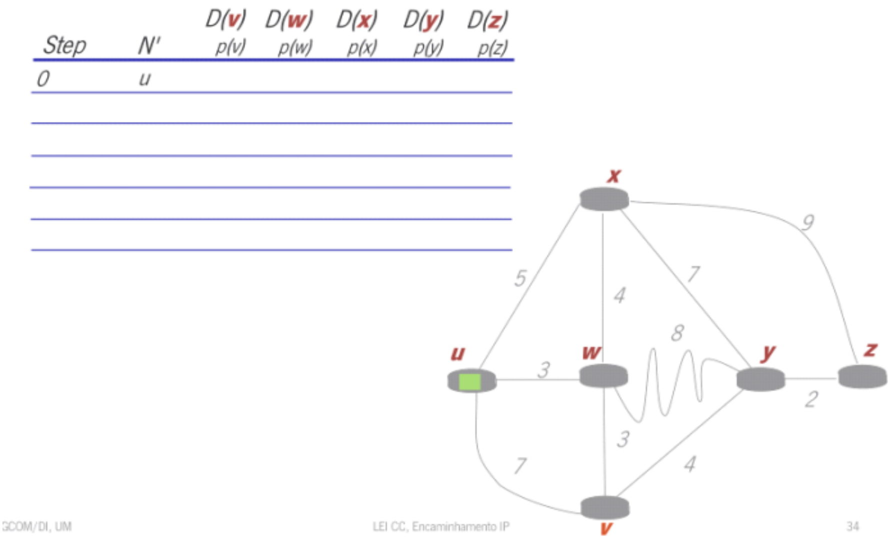
• Algoritmos LS – Dijkstra

- Seja c(i,j)≠∞ o custo da ligação do nó i para o nó adjacente j;
- Se o nó i e o nó j não estão diretamente ligados, então c(i,j)=∞;
- Seja D(v) o custo do caminho desde o nó origem até ao nó v;
- Seja p(v) o nó que antecede v no caminho desde o nó de origem até ao nó v;
- Seja N' o conjunto de todos os nós para os quais já se conhece o caminho de custo mínimo...
- Então o algoritmo Dijkstra para cálculo num nó origem dos valores mínimos D(v) para todos os nós da rede é dado por...

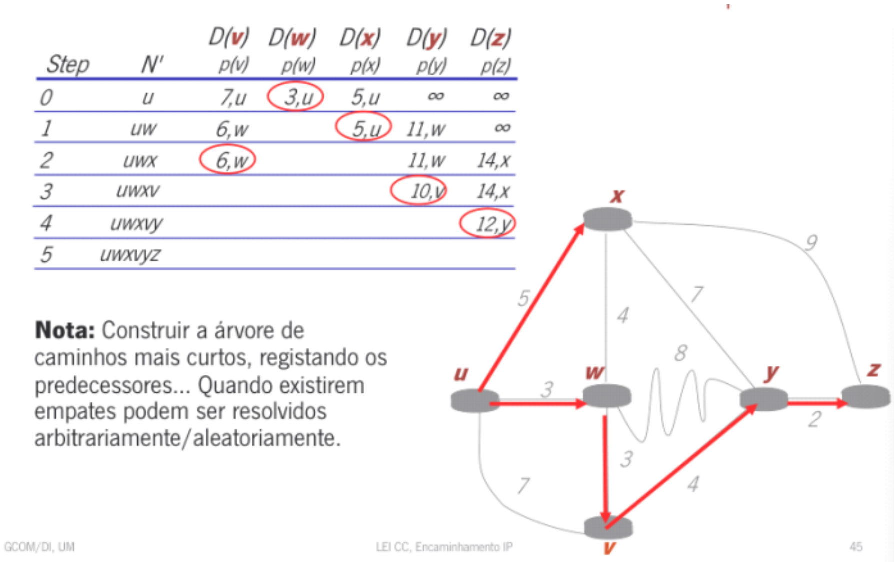
```
** Algorithm for node A
Initialization:
  N' = {A}
  for all nodes B
    if B adjacent to A
        then D(B) = c(A,B)
    else D(B) = ∞

 Loop
   find C not in N' such that D(C) is a minimum
   add C to N'
   for all B adjacent to C and not in N'
      D(B) = min( D(B), D(C) + c(C,B) )
   ** new cost to B is either old cost to B or known
   ** shortest path cost to C plus cost from C to B
 until all nodes in N'
```

Inicialização:
N' = { A }
d(B) = c(A, B)
d(C) = c(A, C)
d(others) = \infinity

é um belo algoritmo



Step	N'	D(v) p(v)	D(w) p(w)	D(x) p(x)	D(y) p(y)	D(z) p(z)
0	u					



Step	N'	D(v) p(v)	D(w) p(w)	D(x) p(x)	D(y) p(y)	D(z) p(z)
0	u	7,u	3,u	5,u	∞	∞
1	uw	6,w		5,u	11,w	∞
2	uwx	6,w			11,w	14,x
3	uwxv				10,v	14,x
4	uwxvy					12,y
5	uwxvyz					

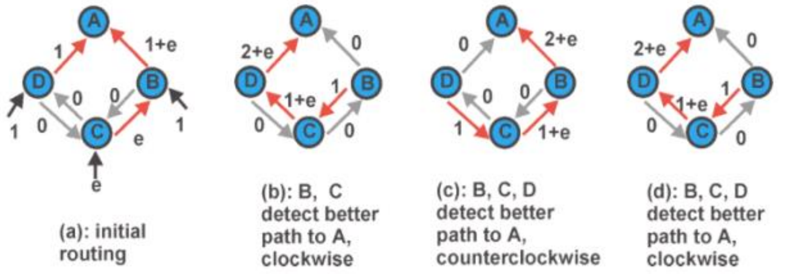
ex.: para o 10,v : D(y) = min (D(y), D(v) + c(v, y))
= min (11, 6 + 4) = 10

Tabela de encaminhamento do nó u

Destino	Próximo nó	Link	Custo
u	u	-	0
x	x	(u, x)	5
w	w	(u, w)	3
v	w	(u, w)	6
y	w	(u, w)	10
z	w	(u, w)	12

Escalabilidade e Oscilações

- Esforço/ Complexidade do algoritmo para N nós da rede é O(N^2), portanto, a implementação dos algoritmos LS têm alguns problemas de escalabilidade.
- Na presença de métricas assimétricas que espelham o estado da rede (por exemplo, se a métrica refletir a carga nas ligações) o cálculo da melhor rota sofre oscilações (ver exemplo abaixo em que a métrica reflete a quantidade de dados transmitidos)



Descentralizada:

- Os encaminhadores só conhecem os vizinhos a que estão fisicamente/logicamente ligados e o custo das ligações respetivas;
- O processo de computação é iterativo, havendo troca de informação entre vizinhos;
- Algoritmos de vetor de distância (***Distance Vector* – DV**).

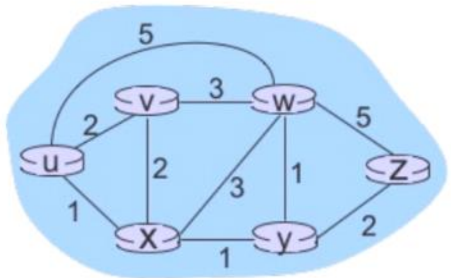
DISTANCE VECTOR - DV

Ao contrário dos algoritmos LS, os algoritmos DV não usam informação global, são distribuídos, iterativos e assíncronos.

- Cada nó recebe informação de encaminhamento de algum dos seus vizinhos diretos, recalcula a tabela de encaminhamento e envia essa informação de encaminhamento de volta;
- O processo continua até que não haja informação de encaminhamento a ser trocada entre nós vizinhos, i.e., até que a informação de encaminhamento convirja;
- Não exige que os nós estejam sincronizados uns com os outros em relação à topologia completa da rede.

Seja **c(x,v)** o custo do caminho entre **x** e **v** adjacentes e $\mathbf{V_x}$ o grupo de todos os nós vizinhos/adjacentes a **x**, então o custo do melhor caminho de **x** para **y** (ou a rota de custo mínimo entre o nó **x** e o nó **y**) é dado por:

$$\mathbf{d_x(y) = \min\{\ c(x,v) + d_v(y)\ \}},\text{ para todos os } \mathbf{v} \text{ em } \mathbf{V_x}$$



Exemplo:

Se, em certo momento, já se souber que:

$d_v(z) = 5$, $d_w(z) = 3$, $d_x(z) = 3$,

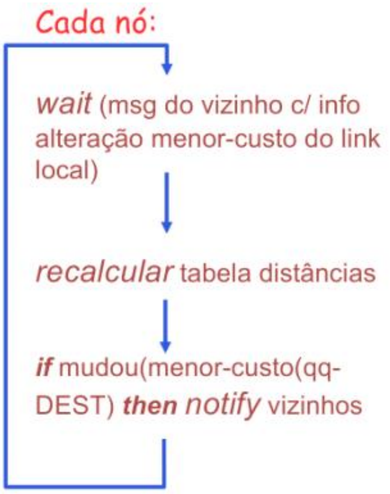
então,

$$d_u(z) = \min\{\ c(u,v) + d_v(z),\ c(u,w) + d_w(z),\ c(u,x) + d_x(z)\ \}$$
$$= \min\{2 + 5,\ 5 + 3,\ 1 + 3\} = 4$$

Sabendo que:

- **N'** é o grupo de todos os nós da rede e que $\mathbf{x} \in \mathbf{N'}$
- $\mathbf{V_x}$ é o grupo de nós adjacentes/vizinhos de **x**
- O nó **x** conhece o custo para todos os seus vizinhos
  $\mathbf{C_x = \{ c(x,v) \}, v \in V_x}$
- O custo do melhor caminho de **x** para **y** é dado por
  $\mathbf{d_x(y) = min \{ \; c(x,v) + d_v(y) \; \}, y \in N', v \in V_x}$
- Seja o custo do melhor caminho de **x** para **y** expresso por
  $\mathbf{D_x(y), y \in N'}$
- Então...

- O nó **x** mantém um vetor de distâncias próprio expresso por
  $\mathbf{DV_x = \{ D_x(y) \}, y \in N'}$
- O nó **x** também mantém os vetores de distâncias dos seus vizinhos
  $\mathbf{DV_v = \{ D_v(y) \}, y \in N', v \in V_x}$
- Cada nó **x** envia periodicamente a sua *estimativa* $\mathbf{DV_x}$ a todos os seus vizinhos
- Quando um nó **x** recebe um novo $\mathbf{DV_v}$ de um dos seus vizinhos atualiza o seu próprio vetor $\mathbf{DV_x}$
- Em condições normais, a estimativa de $\mathbf{D_x(y)}$ converge para o valor de $\mathbf{d_x(y)}$ ao fim de algum tempo
- A troca contínua dos DV mantém a convergência



O processo é iterativo e assíncrono.
Cada iteração local é causada por:

- mudança custo link local
- mensagem do vizinho: vizinho anuncia novo custo

O processo é distribuído.
Cada nó notifica vizinhos apenas quando muda o menor custo para qualquer destino.

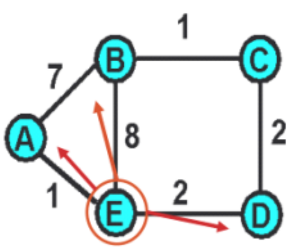
---> Algoritmo Bellman-Ford

$$D^X(Y,Z) = \text{distance } \textit{from } X \textit{ to } Y, \textit{ via } Z \text{ as next hop} = c(X,Z) + \min_w \{D^Z(Y,w)\}$$

Fonte: Computer Networking: A Top-Down Approach Featuring the Internet, J. Kurose, Addison-Wesley

At each node, X:

```
Initialization:
 for all adjacent nodes v:
    D^X(*,v) = infty  /* "for all rows" */
    D^X(v,v) = c(X,v)
 for all destinations, y
    send min_w D(y,w) to each neighbor
    /* w over all X's neighbors */

loop forever
  wait (until I see a link cost change to neighbor V
        or until I receive update from neighbor V)
  if (c(X,V) changes by d)
    /* change cost to all dest's via neighbor v by d */
    /* note: d could be positive or negative */
    for all destinations y:  D^X(y,V) =  D^X(y,V) + d

  else if (update received from V for destination Y)
    /* shortest path from V to some Y has changed */
    /* V has sent a new value for its  min_w D^V(Y,w) */
    /* call this received new value as "newval" */
    for the single destination y: D^X(Y,V) = c(X,V) + newval

  if we have a new min_w D^X(Y,w) for any destination Y
     send new value of min_w D^X(Y,w) to all neighbors
```



<- saídas

(total, estimado)

$D^E()$	cost to destination via A	B	D
destinat. A	1	14	5
B	7	8	5
C	6	9	4
D	4	11	2

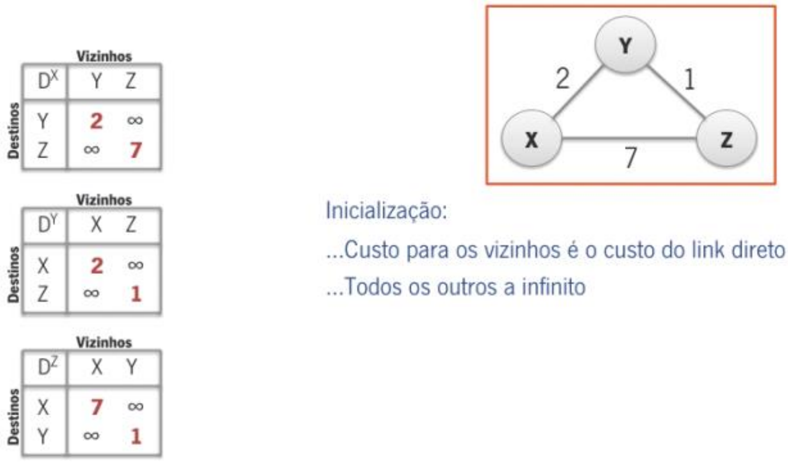
Custo para o destino via...

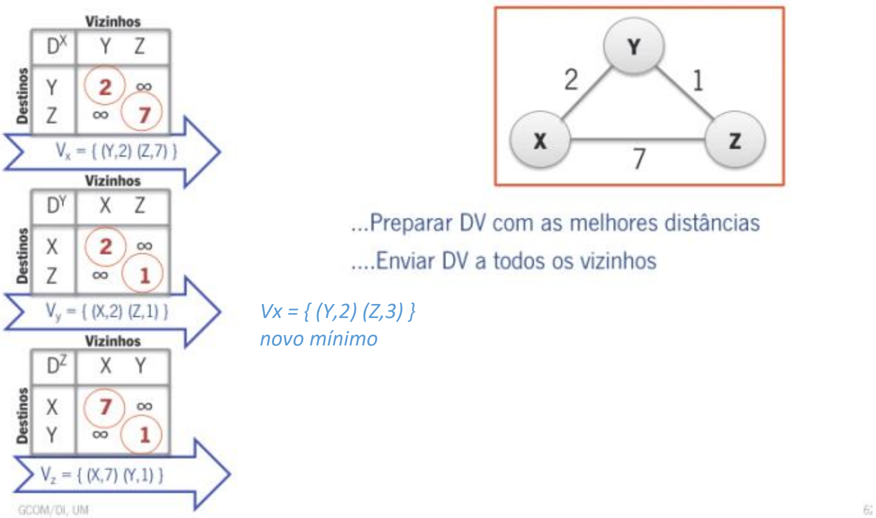
$D^E()$	A	B	D
destino A	1	14	5
B	8	8	5
C	6	9	4
D	4	11	2

	Saída	Custo
destino A	A,	1
B	D,	5
C	D,	4
D	D,	2

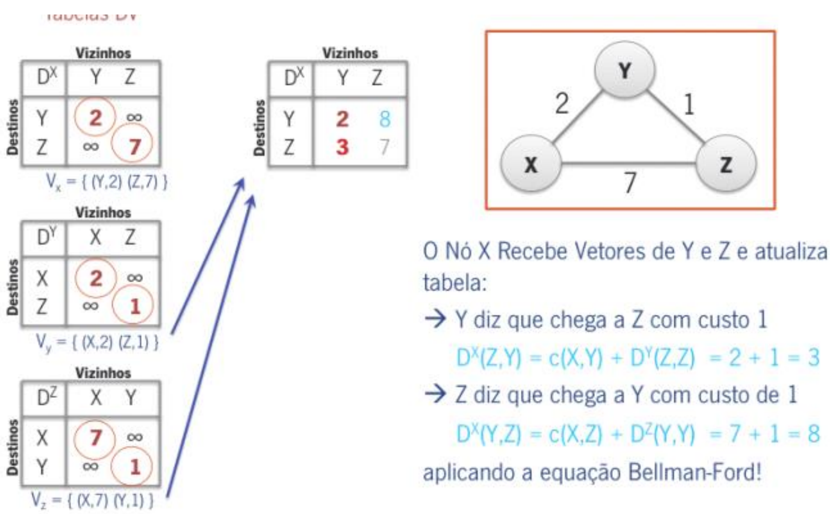
Tabela DV ⟶ Tabela Encaminhamento

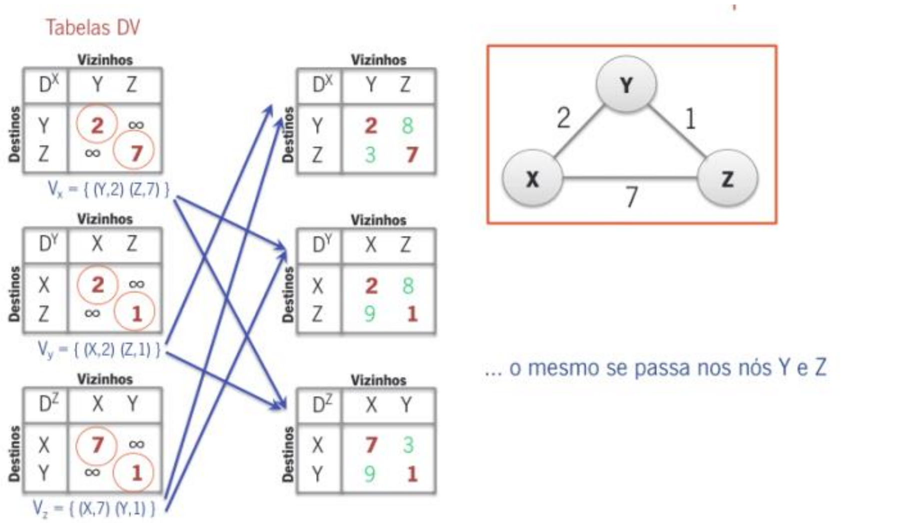
Mais um exemplo:

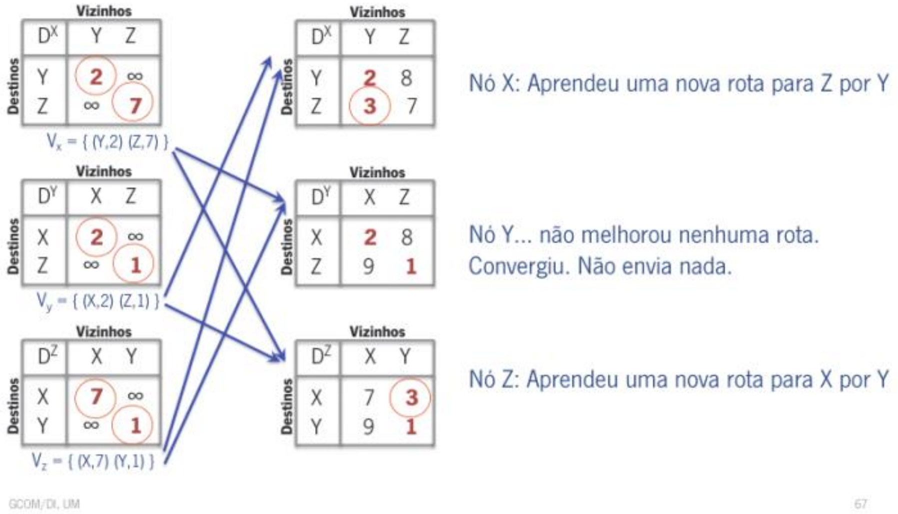


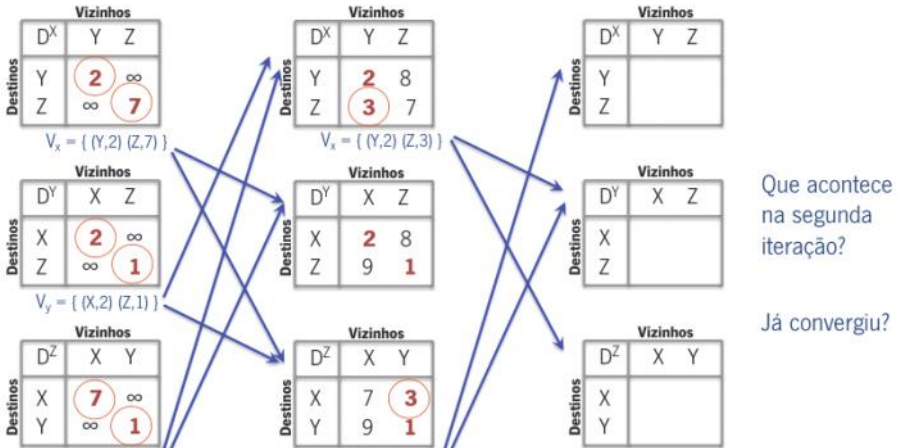


Vx = { (Y,2) (Z,3) }
novo mínimo

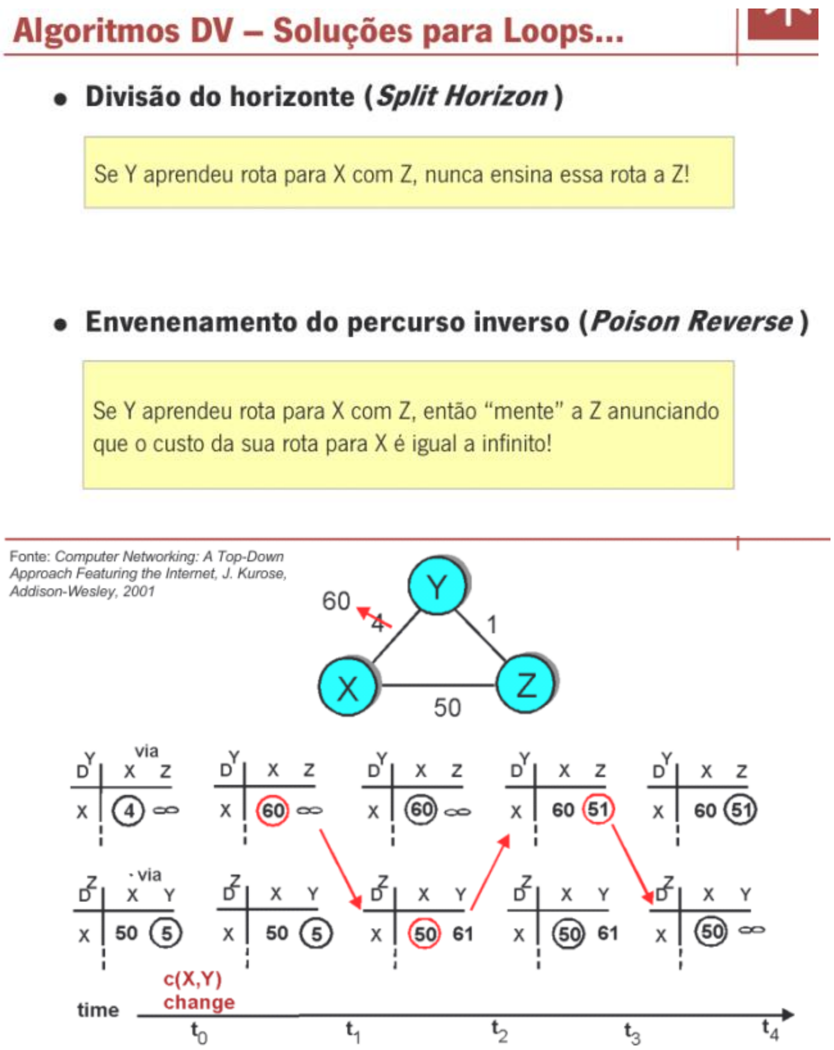


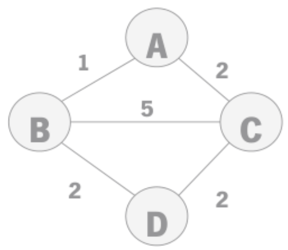






Já convergiu.
Dx(X, Z) = c(X, Z) + Dz(X, Y) = 7 + 3 = 10 > Dx(X, X) = 0





B aprendeu uma rota para C com A
⇒ B "mente" anunciando a A que o custo
da sua rota para C é ∞

D aprendeu uma rota para A com B
⇒ D "mente" anunciando a B que o custo
da sua rota para A é ∞

(evenenamento inverso)

DA	B	C
B	1	∞
C	∞	2
D	∞	∞

DA	B	C
B	1	7
C	6	2
D	3	4

DA	B	C
B	1	7
C	∞	2
D	3	4

DA	B	C
B	1	5
C	3	2
D	3	4

DB	A	C	D
A	1	∞	∞
C	∞	5	∞
D	∞	∞	2

DB	A	C	D
A	1	7	∞
C	3	5	4
D	∞	7	2

DB	A	C	D
A	1	7	∞
C	3	5	4
D	∞	7	2

DB	A	C	D
A	1	7	5
C	3	5	4
D	4	7	2

DC	A	B	D
A	2	∞	∞

DC	A	B	D
A	2	6	∞

DC	A	B	D
A	2	6	∞

DC	A	B	D
A	2	5	5

DC	A	B	D
A	2	∞	∞
B	∞	5	∞
D	∞	∞	2

DC	A	B	D
A	2	6	∞
B	3	5	4
D	∞	7	2

DC	A	B	D
A	2	6	∞
B	3	5	4
D	∞	7	2

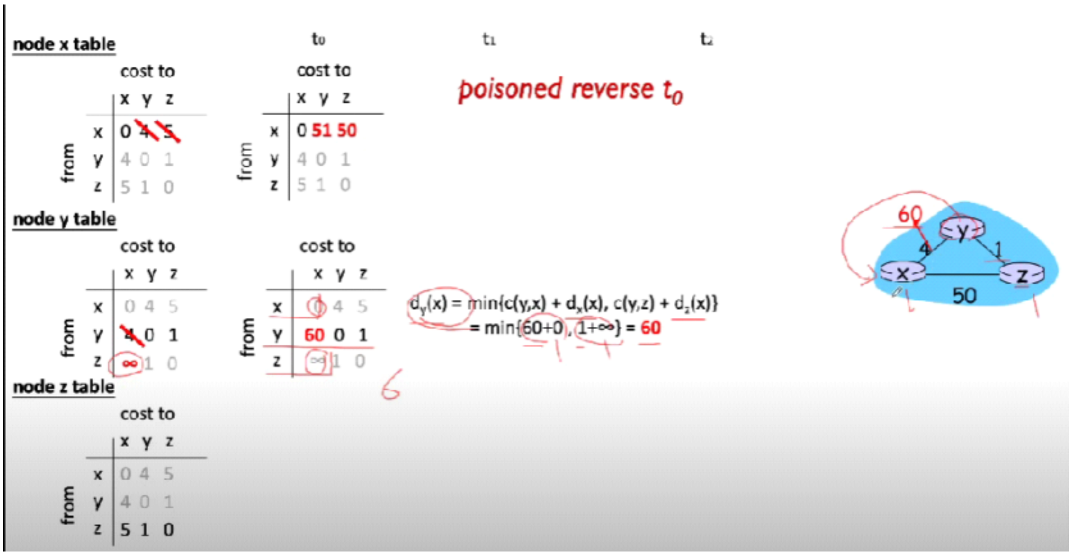
DC	A	B	D
A	2	6	5
B	3	5	4
D	5	7	2

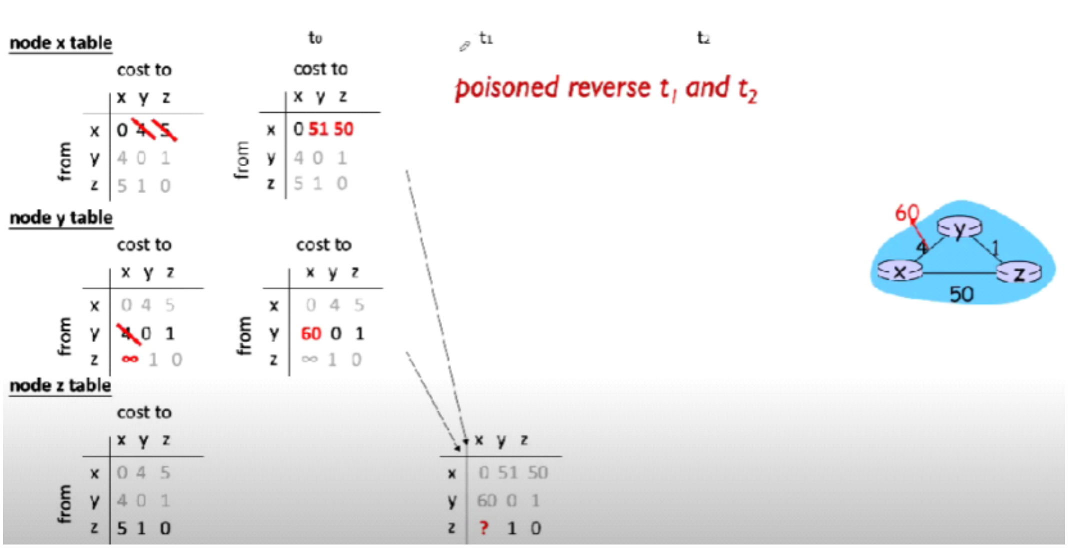
DD	B	C
A	∞	∞
B	2	∞
C	∞	2

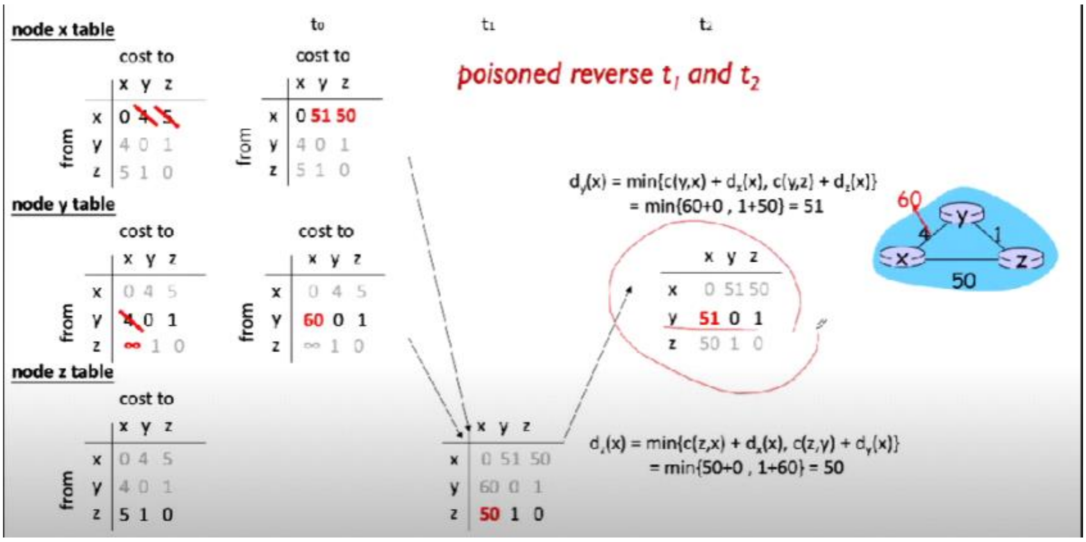
DD	B	C
A	3	4
B	2	7
C	7	2

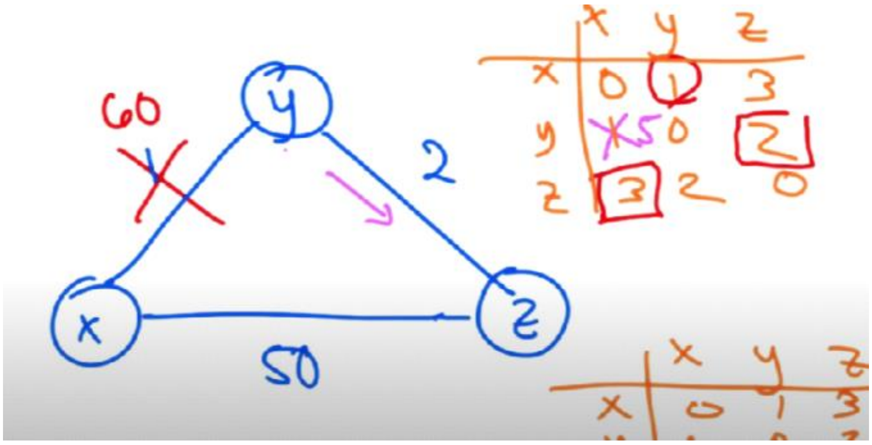
DD	B	C
A	3	4
B	2	7
C	7	2

DD	B	C
A	3	4
B	2	7
C	7	2



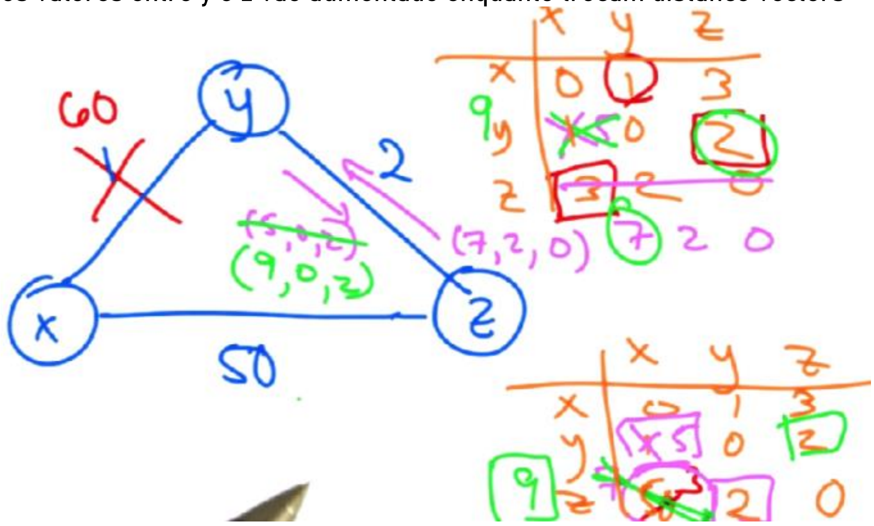




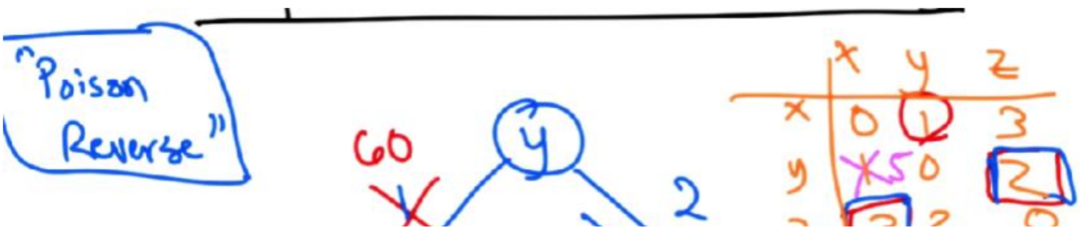


y -> x does not update to 60
y -> z is 2, and z -> x is 3
so, min (60, 2 + 3) = 5
é atualizado para 5

envia para z o seu novo distance vector (5, 0, 2)
os valores entre y e z vao aumentado enquanto trocam distance vectors

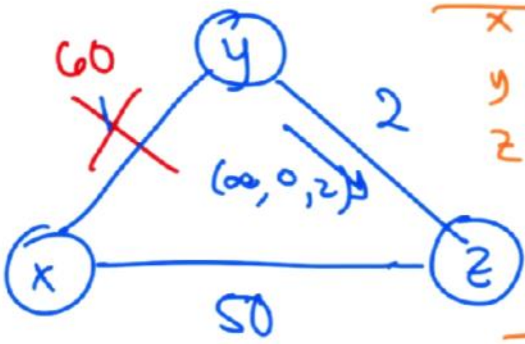


<u>Count to infinity problem</u>
solution: poison reverse -- send$(\infty, 0, 2)$ instead of $(5, 0, 2)$



"Poison Reverse"

	x	y	z
x	0	1	3 ~~50~~
y	1	0	2
z	3	2	0



	x	y	z
x	0	1	3
y	X5	0	$\boxed{2}$
z	$\boxed{3}$	2	0

	x	y	z
x	0	1	3
y	X∞	0	2
z	~~50~~	2	0