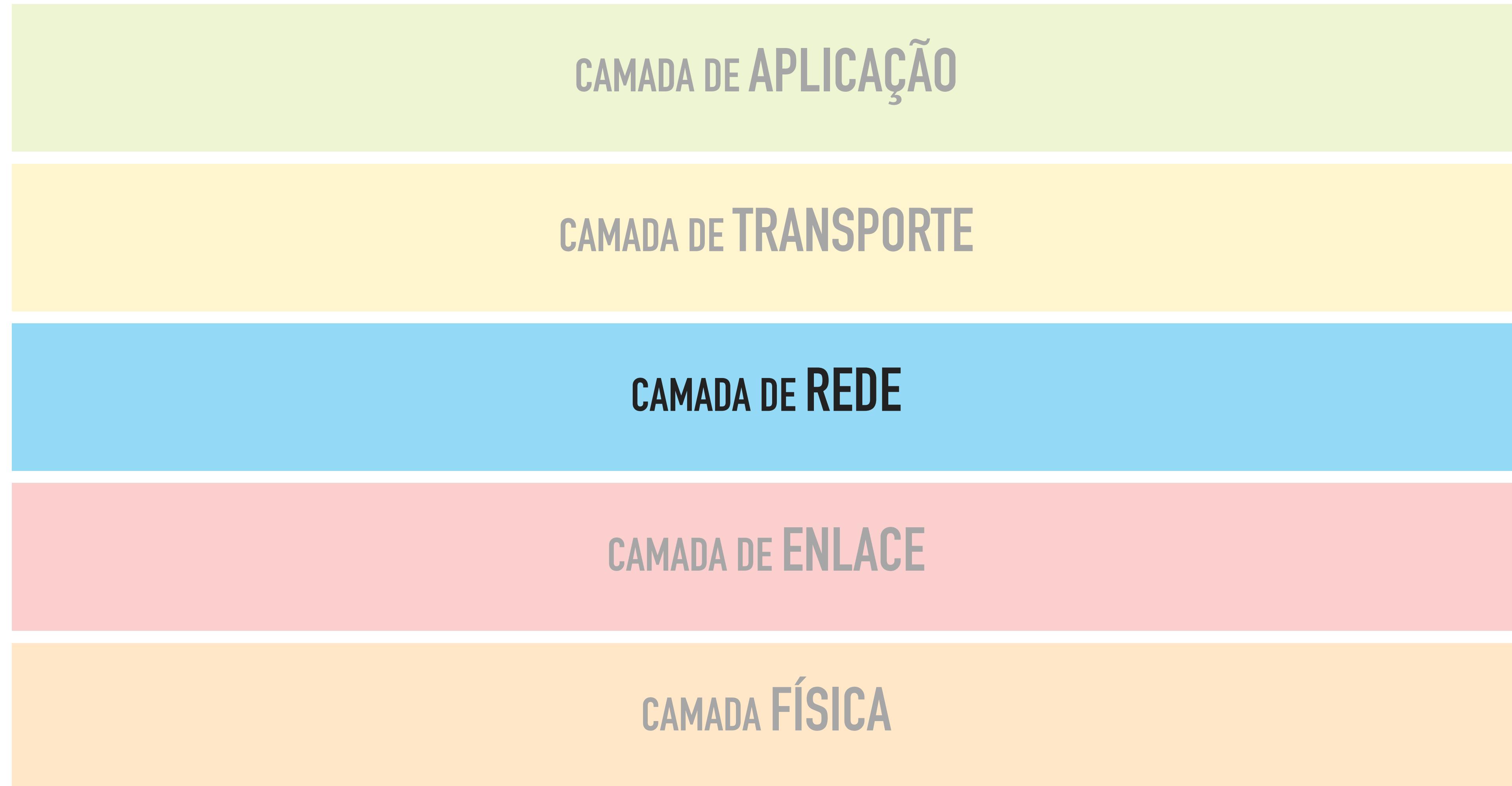


IMD0043

ROTEAMENTO POR VETOR DE DISTÂNCIAS



#3 COMPUTAÇÃO DISTRIBUÍDA DE ROTEAMENTO

- ▶ Cada nó calcula os *links* de saída baseado em:
 - ▶ Custo dos *links* locais
 - ▶ Informação anunciada pelos vizinhos
- ▶ Algoritmos diferem no que essas trocas se baseiam:
 - ▶ **Vetor de distâncias** (*distance vector*): somente a distância (e o próximo *hop*) para cada destino
 - ▶ **Vetor de caminhos** (*path vector*): o caminho completo para cada destino

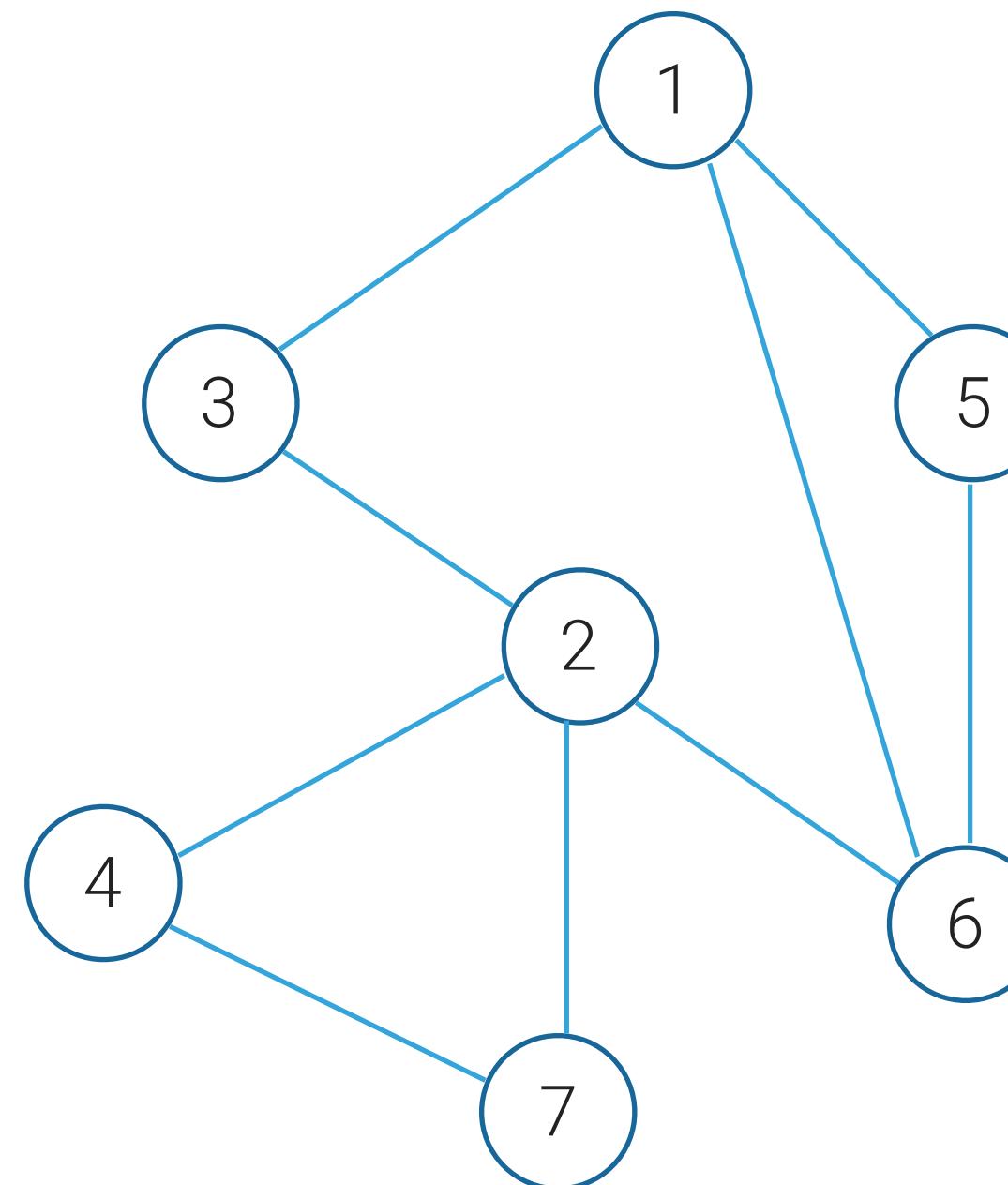
RELEMBRANDO...

Tabelas de roteamento = Coleção de *spanning trees*

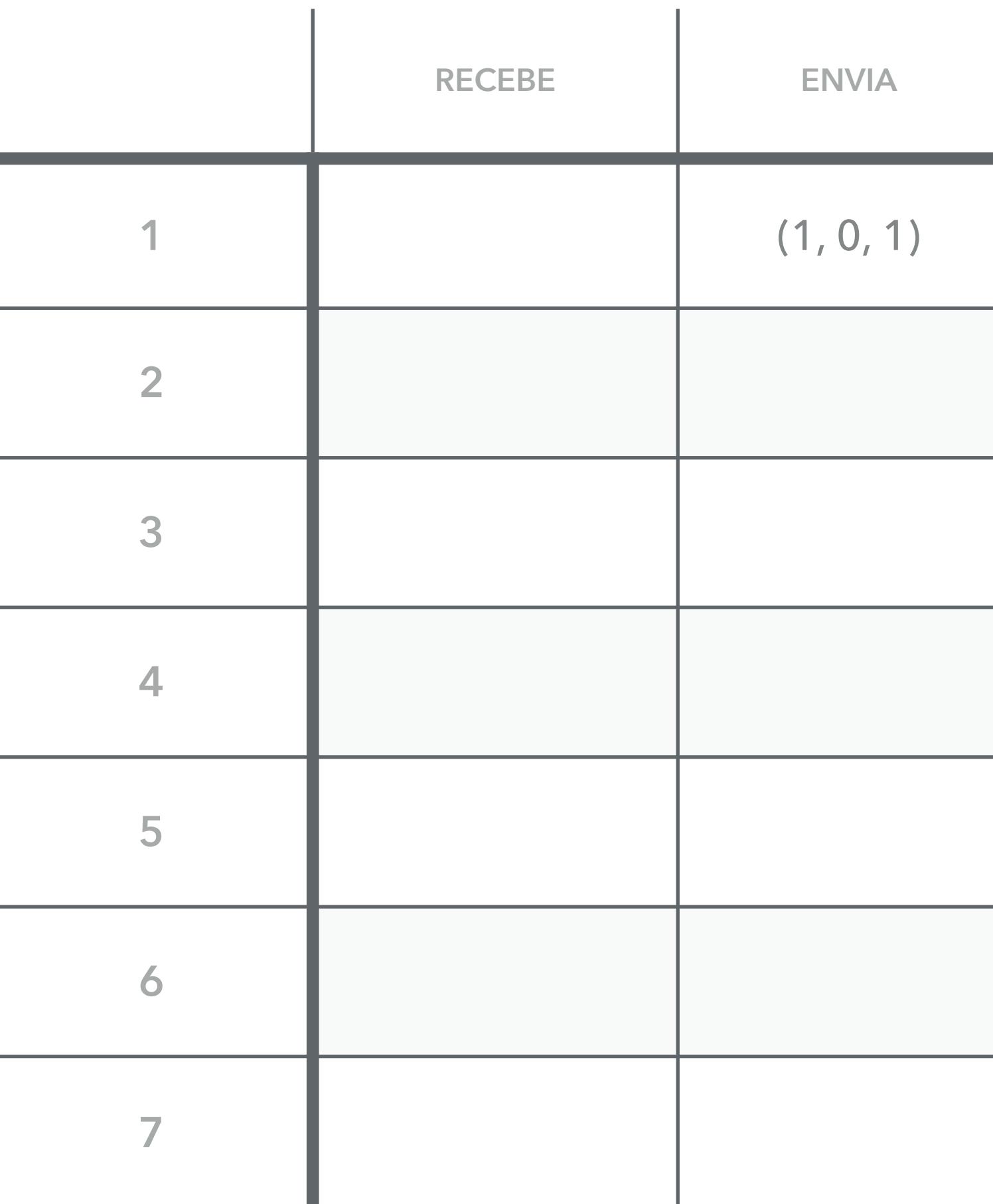
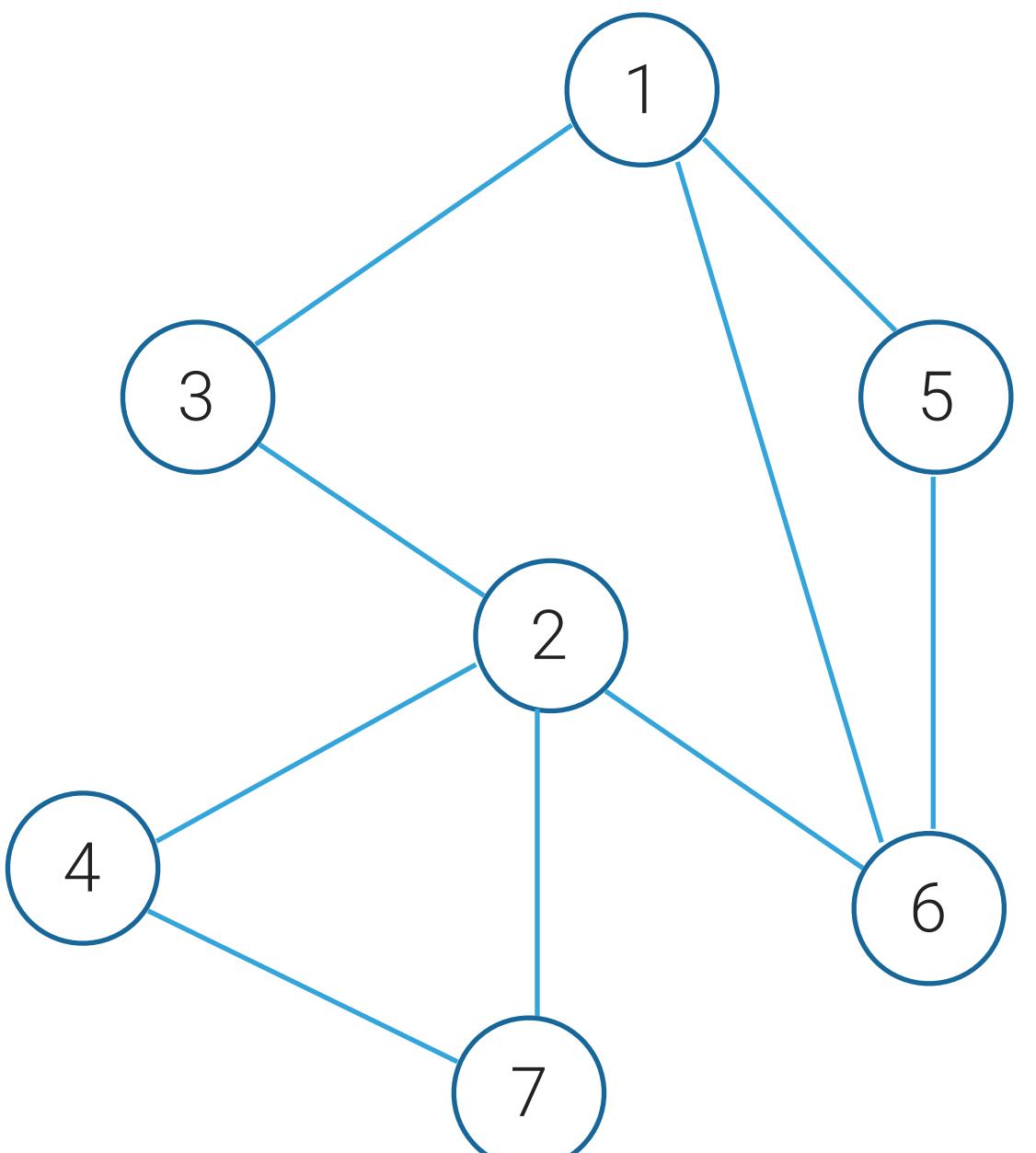
- ▶ Podemos usar o STP (com modificações)?
- ▶ Mensagens (Y, d, X)
 - ▶ Para o nó Y , a partir do nó X , com distância d para Y
- ▶ Inicialmente cada router X anuncia $(X, 0, X)$ para seus vizinhos

VETOR DE DISTÂNCIAS: UMA COLEÇÃO DE “N” STP EM PARALELO

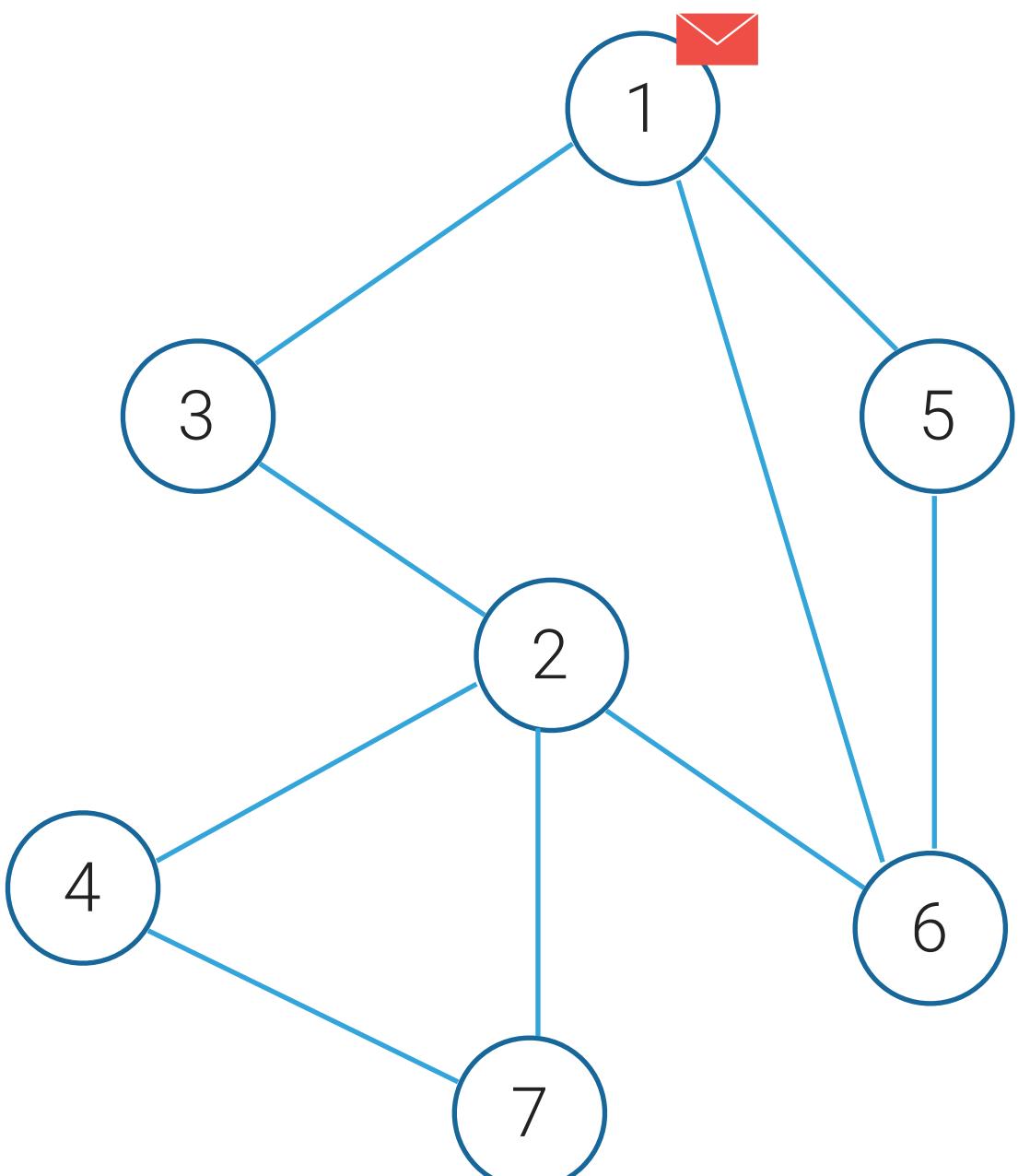
Exemplo para destino 1



ITERAÇÃO #1

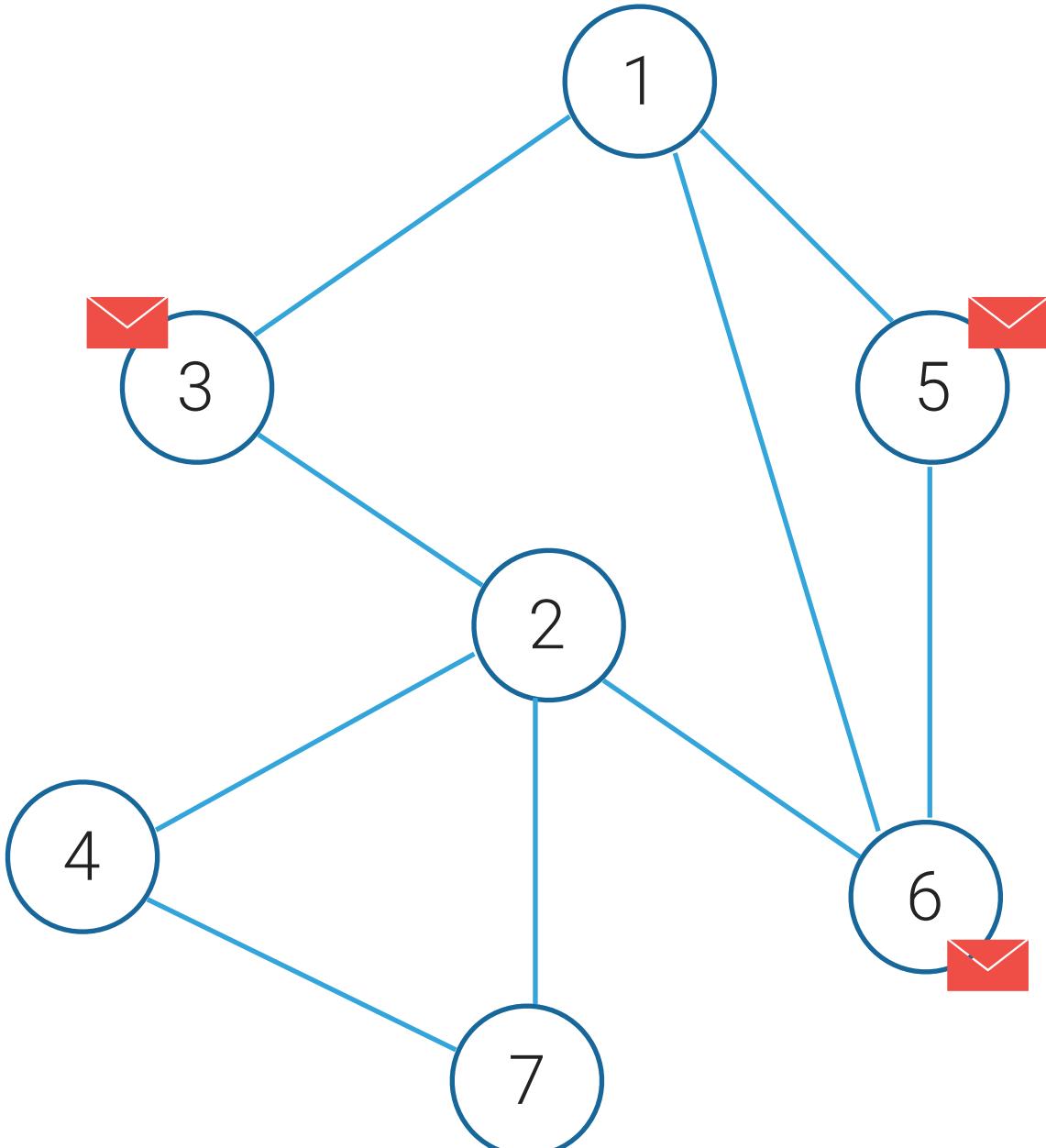


ITERAÇÃO #2



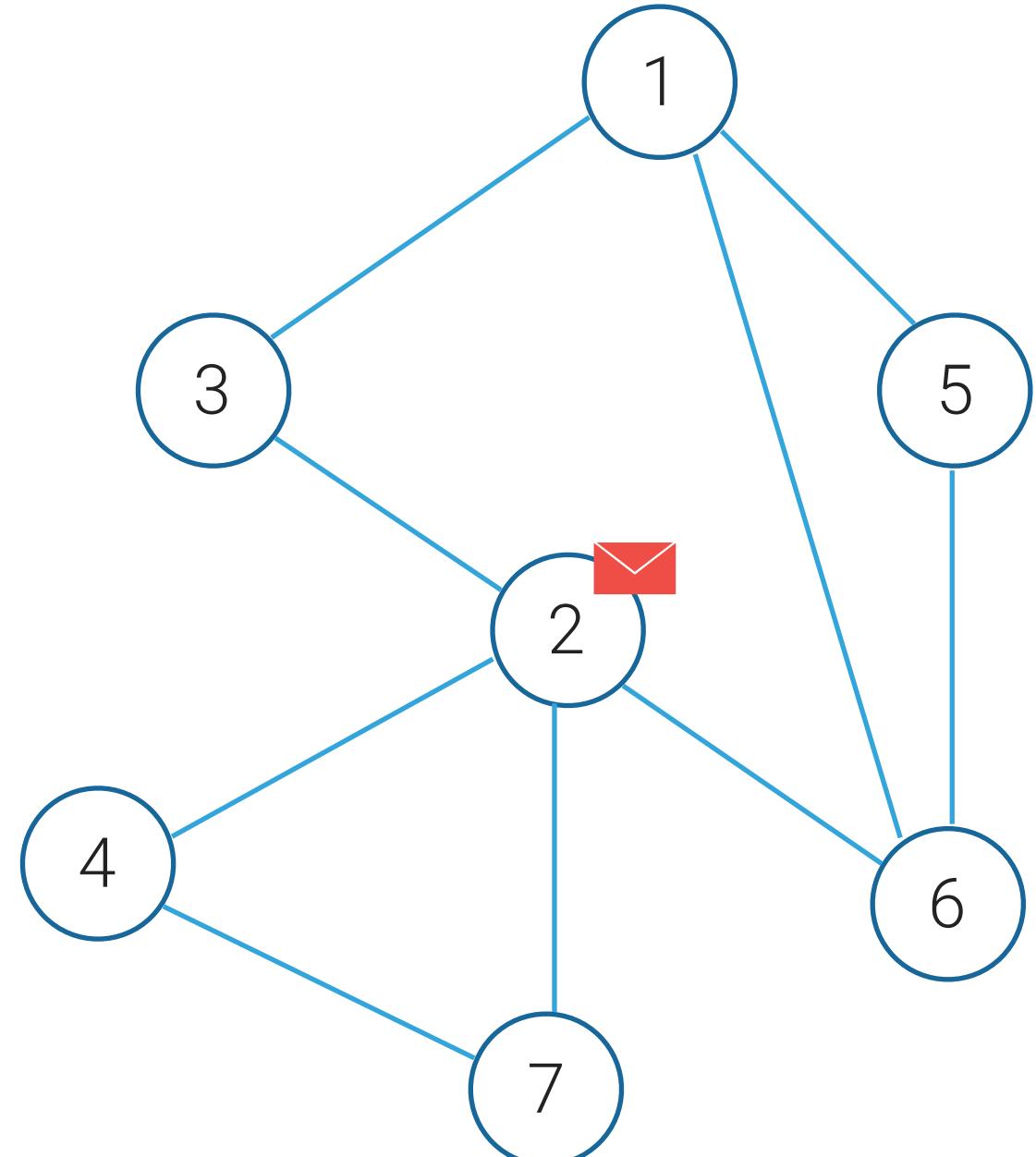
	RECEBE	ENVIA
1 (1, 0, 1)		
2		
3	(1, 0, 1)	(1, 1, 3)
4		
5	(1, 0, 1)	(1, 1, 5)
6	(1, 0, 1)	(1, 1, 6)
7		

ITERAÇÃO #3



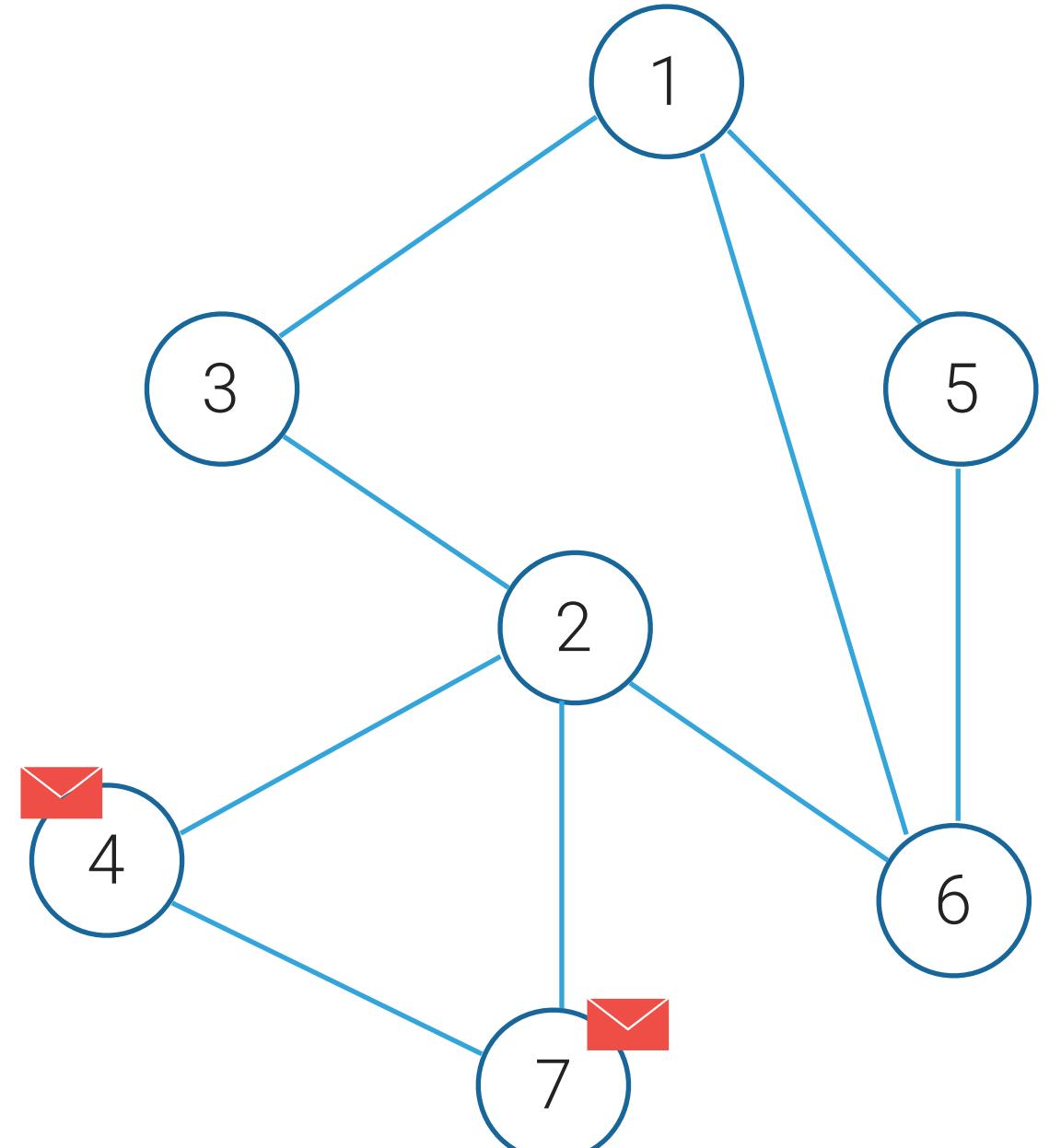
	RECEBE	ENVIA
1 (1, 0, 1)	(1, 1, 3), (1, 1, 5), (1, 1, 6)	
2	(1, 1, 3), (1, 1, 6)	(1, 2, 2)
3 (1, 1, 3)		
4		
5 (1, 1, 5)		
6 (1, 1, 6)		
7		

ITERAÇÃO #4



	RECEBE	ENVIA
1 (1, 0, 1)		
2 (1, 2, 2)		
3 (1, 1, 3)	(1, 2, 2)	
4	(1, 2, 2)	(1, 3, 4)
5 (1, 1, 5)		
6 (1, 1, 6)	(1, 2, 2)	
7	(1, 2, 2)	(1, 3, 7)

ITERAÇÃO #5



	RECEBE	ENVIA
1 (1, 0, 1)		
2 (1, 2, 2)	(1, 3, 4), (1, 3, 7)	
3 (1, 1, 3)		
4 (1, 3, 4)		(1, 3, 7)
5 (1, 1, 5)		
6 (1, 1, 6)		
7 (1, 3, 7)		(1, 3, 4)

POR QUE NÃO STP? POR QUE VETOR DE DISTÂNCIAS?

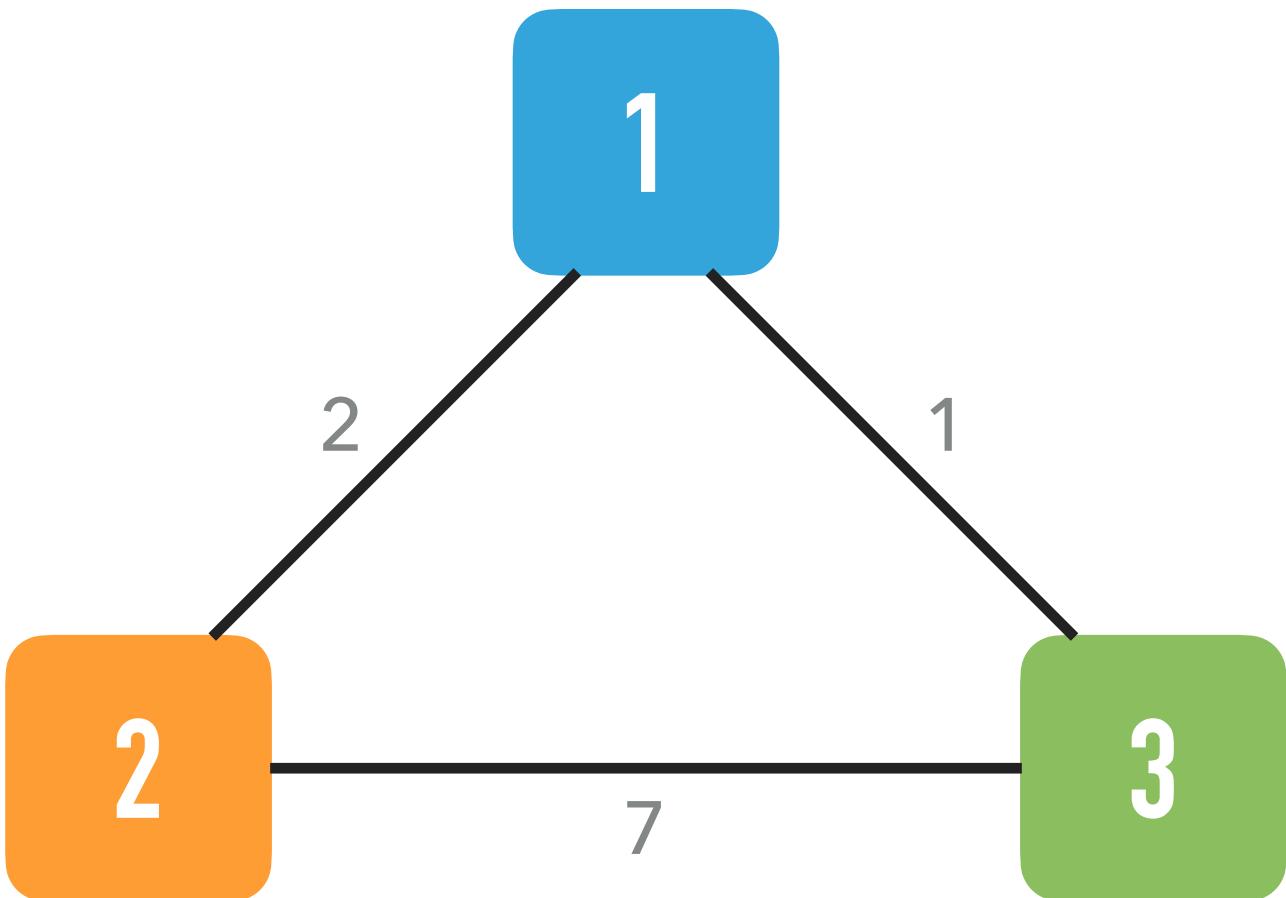
- ▶ O mesmo protocolo/algoritmo se aplica a todos os destinos
- ▶ Cada nó anuncia a distância para **cada destino**
 - ▶ Estou a 4 saltos do nó A
 - ▶ Estou a 6 saltos do nó B
 - ▶ Estou a 3 saltos do nó C
 - ▶ ...
 - ▶ Nós estamos trocando seu vetor de distâncias

PROTOCOLO DE ROTEAMENTO POR VETOR DE DISTÂNCIAS

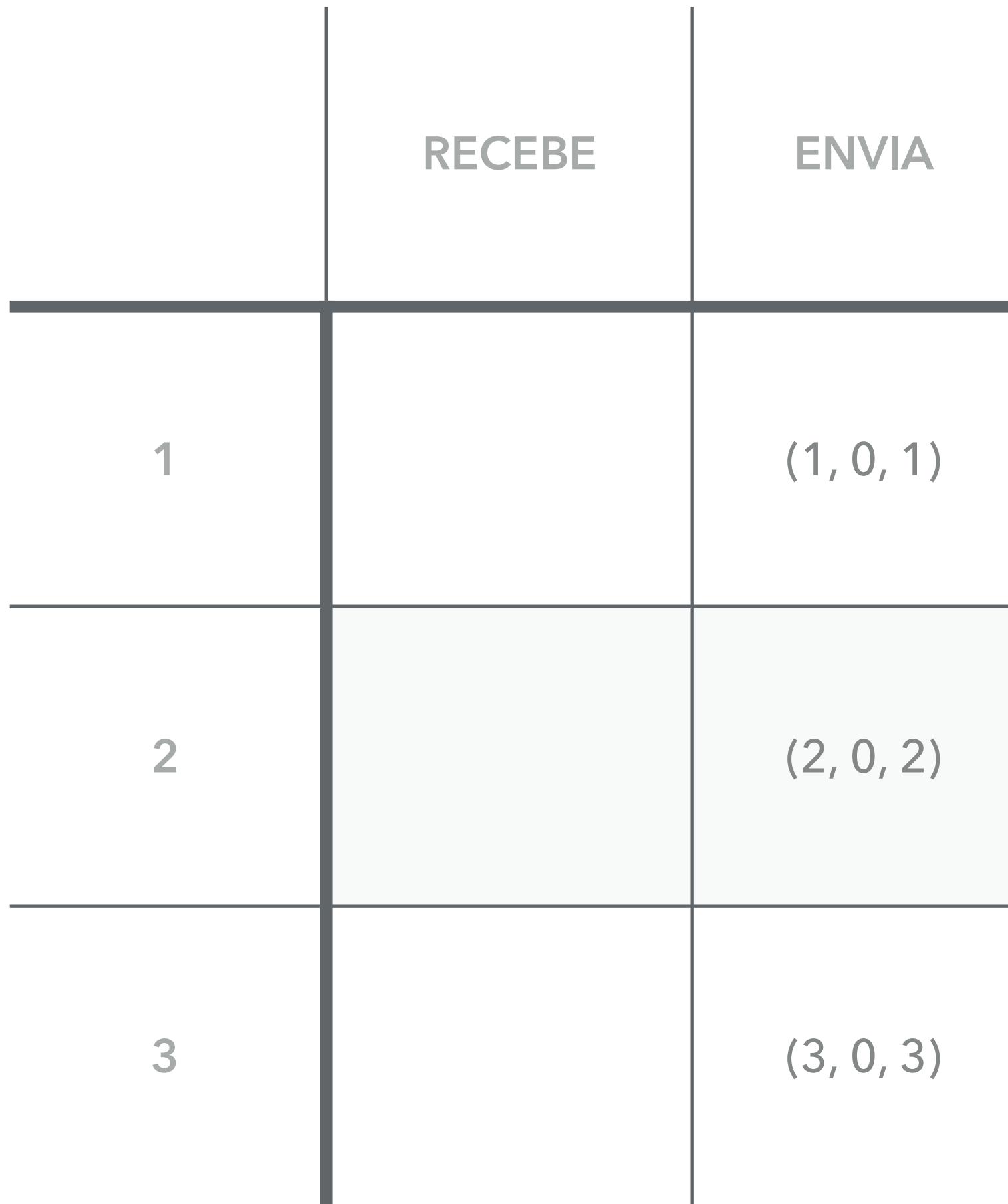
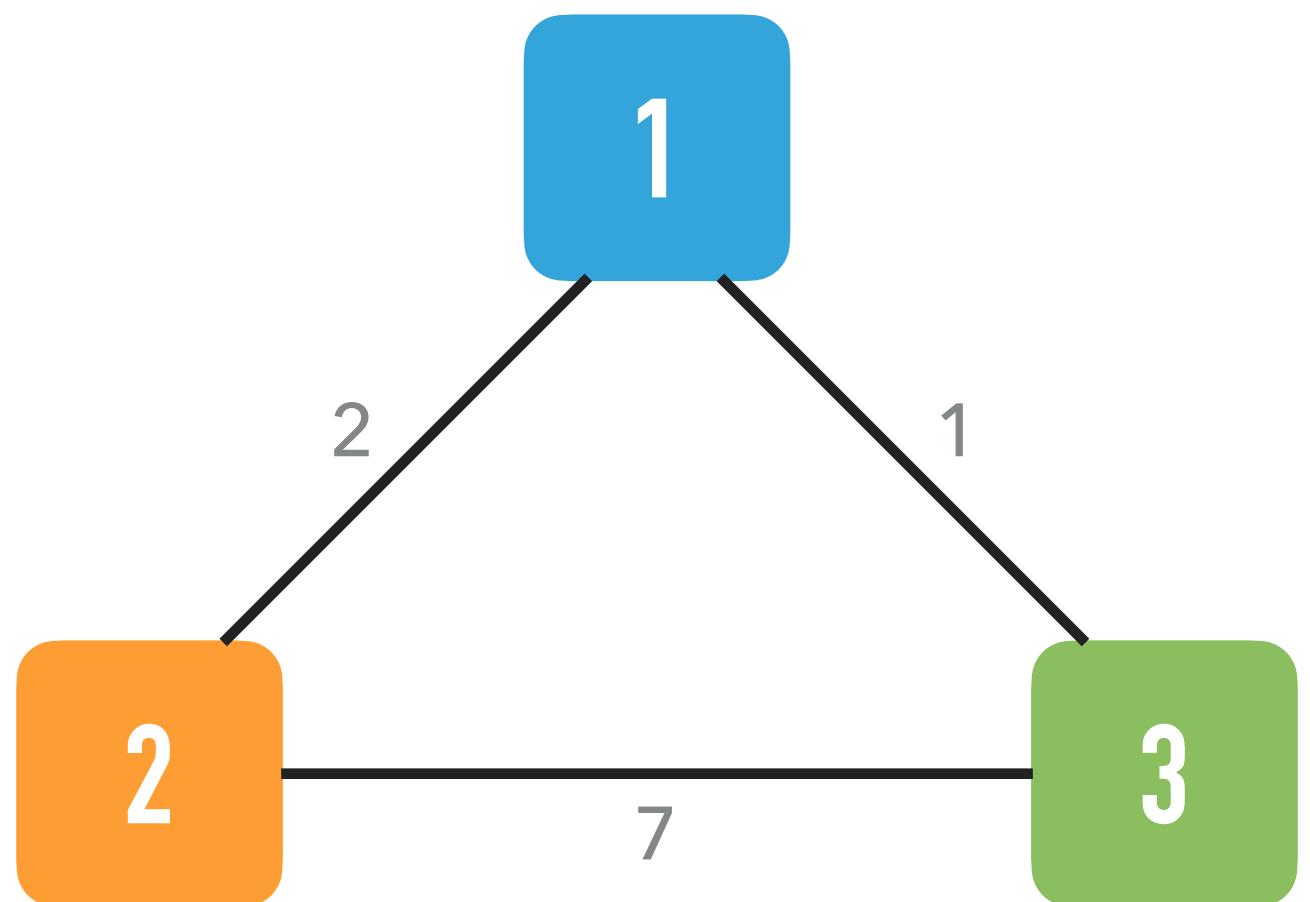
Mensagens (Y, d, X)

- ▶ Para Y , a partir do nó X , anunciando distância d para Y
- ▶ Inicialmente cada router anuncia $(X, 0, X)$ para seus vizinhos
- ▶ Routers atualizam sua visão ao receber as mensagens (Y, d, Z)
- ▶ Routers computam a menor distância até o destino
- ▶ SE $distância_{atual_para_Y} > d + custo_do_link_para_Z$ ENTÃO
 - ▶ atualize $distância_{atual_para_Y} = d + custo_do_link_para_Z$
- ▶ Se o menor caminho para o destino mudou, envie para todos os vizinhos a mensagem atualizada $(Y, d+c, X)$

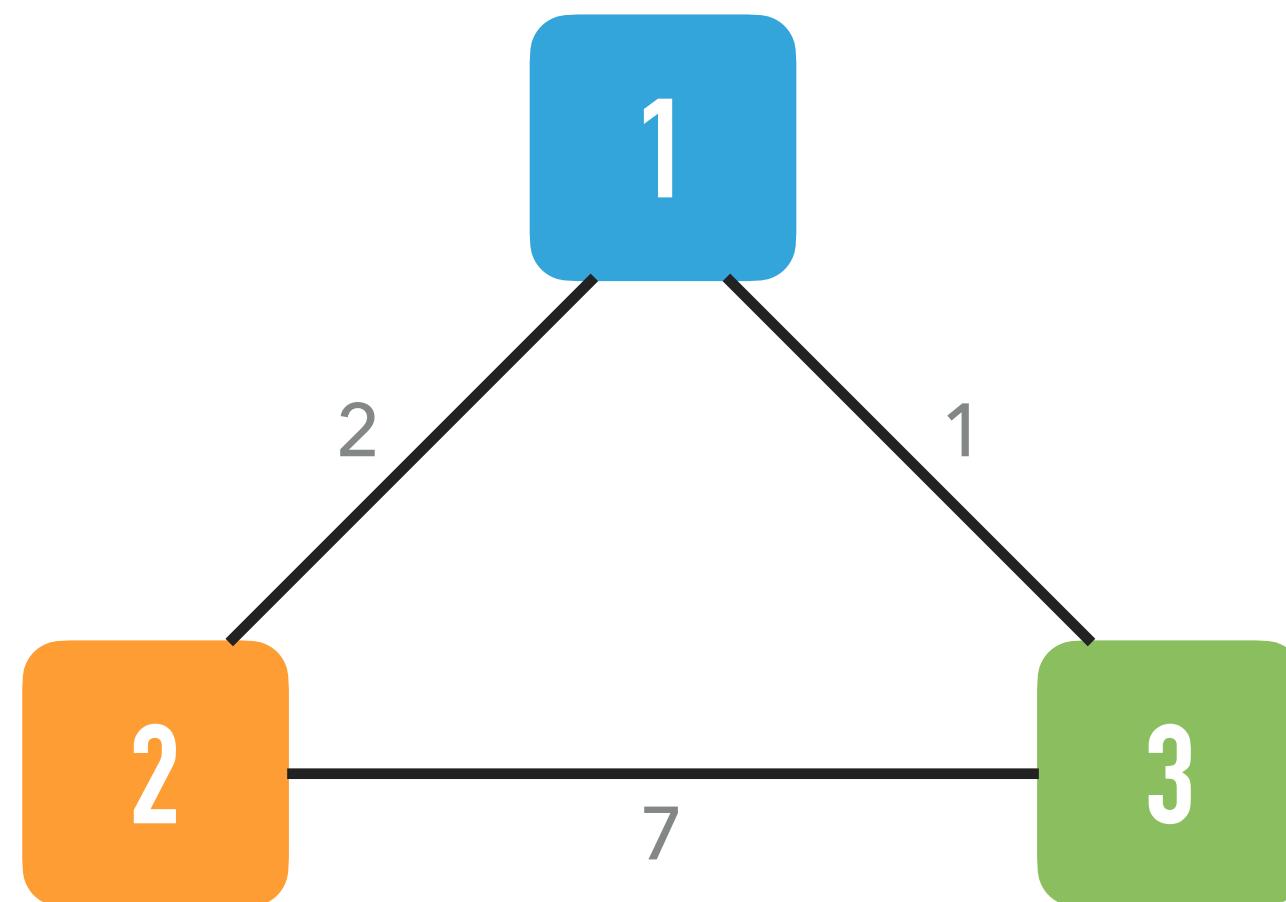
EXEMPLO COM O PROTOCOLO DESCrito



ITERAÇÃO #1

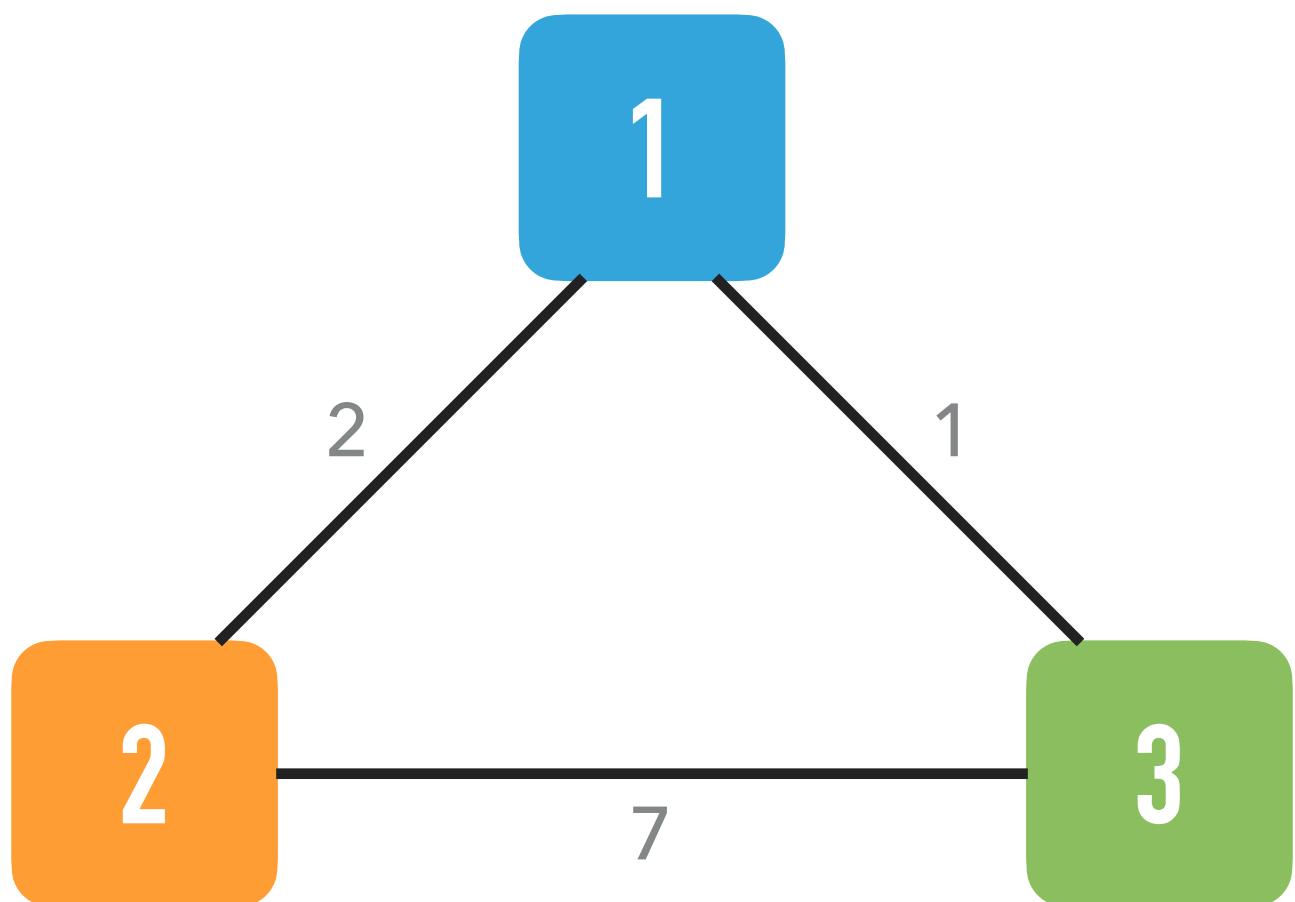


ITERAÇÃO #2



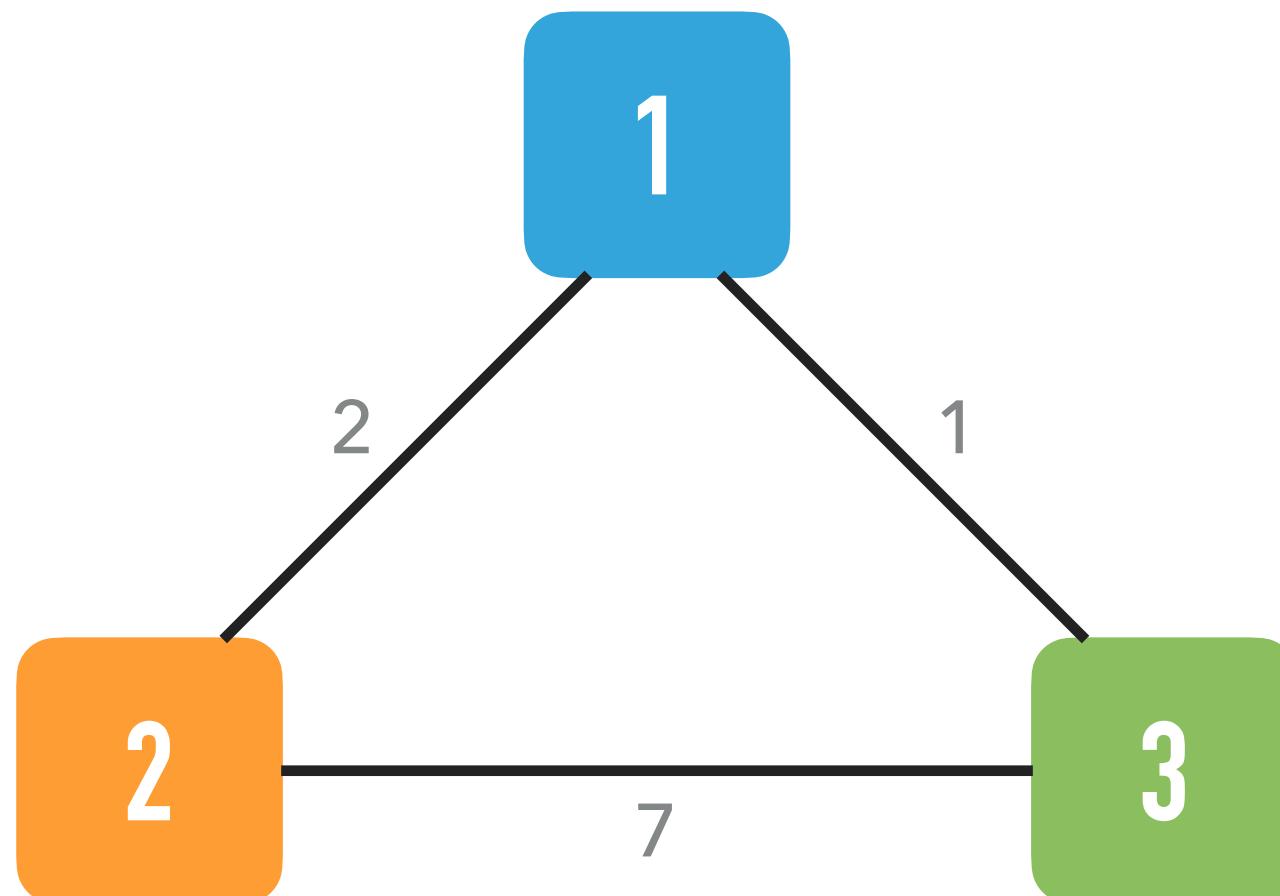
	RECEBE	ENVIA
1 (1, 0, 1)	(2, 0, 2) (3, 0, 3)	(2, 2, 1) (3, 1, 1)
2 (2, 0, 2)	(1, 0, 1) (3, 0, 3)	(1, 2, 2) (3, 7, 2)
3 (3, 0, 3)	(1, 0, 1) (2, 0, 2)	(1, 1, 3) (2, 7, 3)

ITERAÇÃO #3



	RECEBE	ENVIA
1	(1, 2, 2) (3, 7, 2) (2, 2, 1) (3, 1, 1)	
2		(2, 2, 1) (3, 1, 1) (2, 0, 2) (3, 7, 2)
3	(2, 2, 1) (3, 1, 1) (2, 7, 3) (3, 0, 3)	(2, 3, 3)

ITERAÇÃO #4



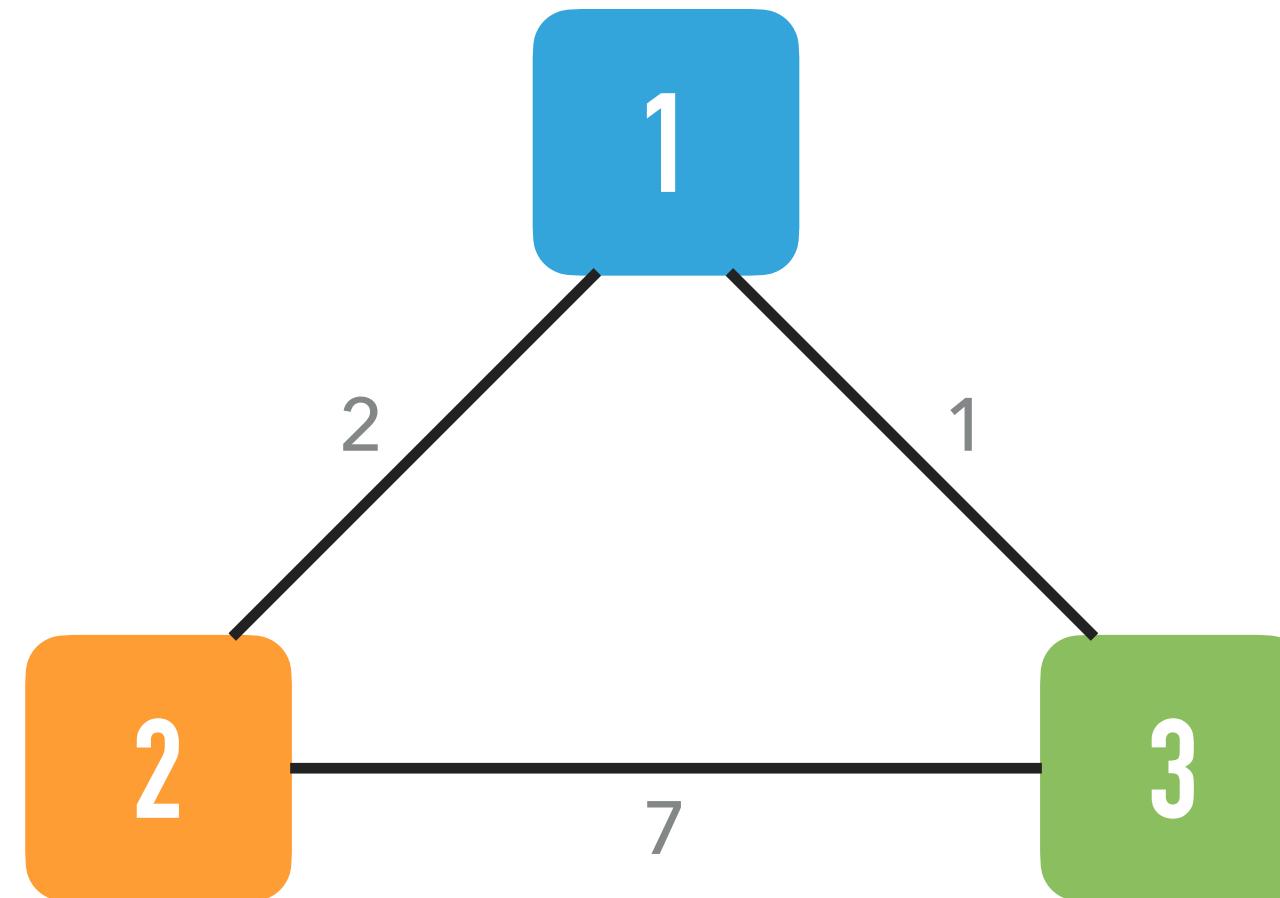
	RECEBE	ENVIA
1	(1, 2, 2) (1, 0, 1) (2, 2, 1) (3, 1, 1)	(3, 3, 2) (2, 3, 3)
2	(2, 2, 1) (1, 2, 2) (2, 0, 2) (3, 3, 2)	(2, 3, 3)
3	(2, 2, 1) (3, 1, 1) (1, 2, 2) (3, 7, 2)	(3, 3, 2)

PROTOCOLO DE ROTEAMENTO POR VETOR DE DISTÂNCIAS COM NEXT HOP

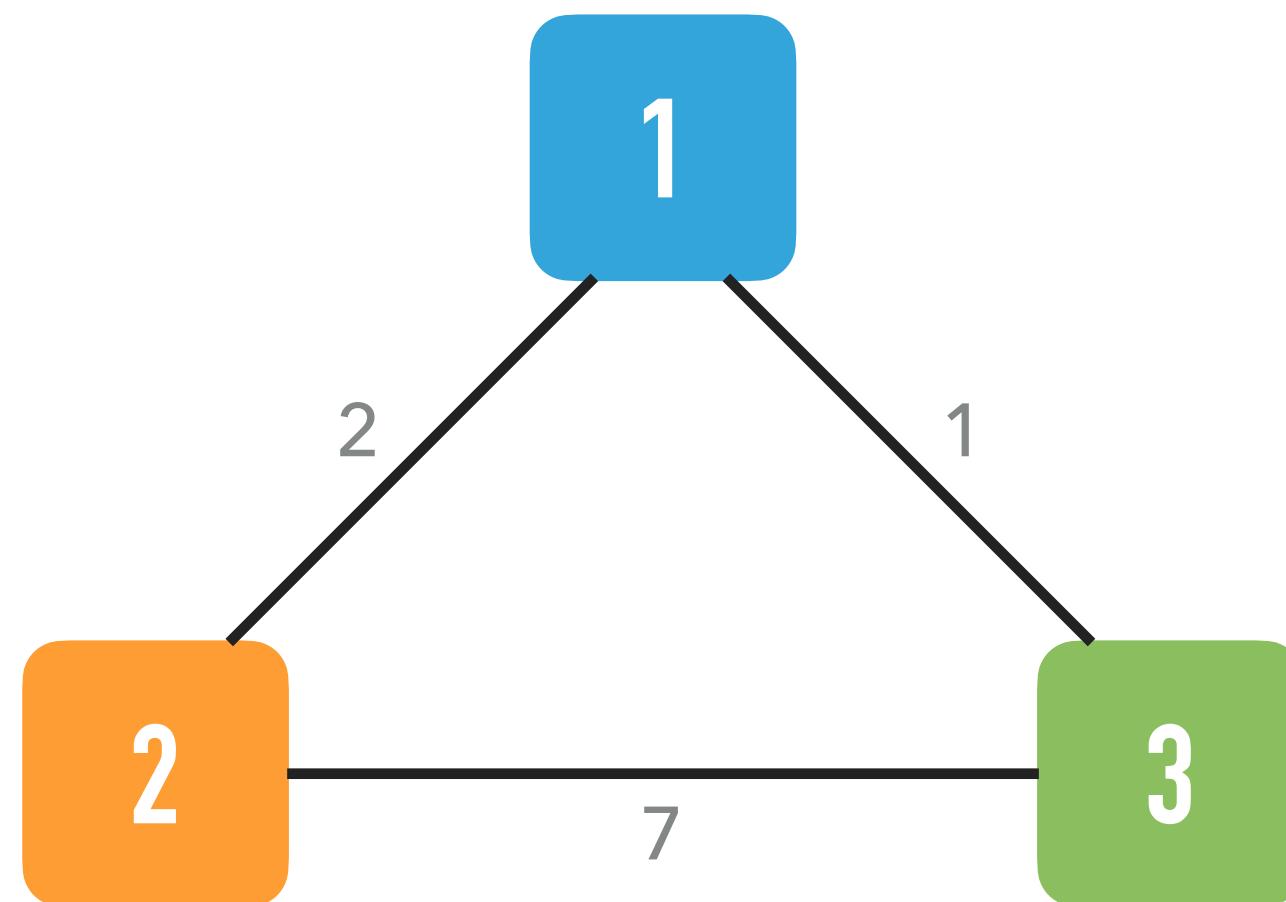
Mensagens (Y, d, X)

- ▶ Para Y , a partir do nó X , anunciando distância d para Y
- ▶ Inicialmente cada router anuncia $(X, 0, X)$ para seus vizinhos
- ▶ Router Z atualiza sua visão ao receber a mensagem (Y, d, X) de X
- ▶ Router Z computa a menor distância até o destino Y
- ▶ SE $distância_{atual_para_Y} > d + custo_do_link_para_X$ ENTÃO
 - ▶ atualize $distância_{atual_para_Y} = d + custo_do_link_para_X$
 - ▶ atualize o próximo salto (*next hop*) para destino $Y = X$
- ▶ Se o menor caminho para o destino mudou, envie para todos os vizinhos a mensagem atualizada $(Y, d+c, Z)$

EXEMPLO COM O PROTOCOLO DESCrito E COM NEXT HOP

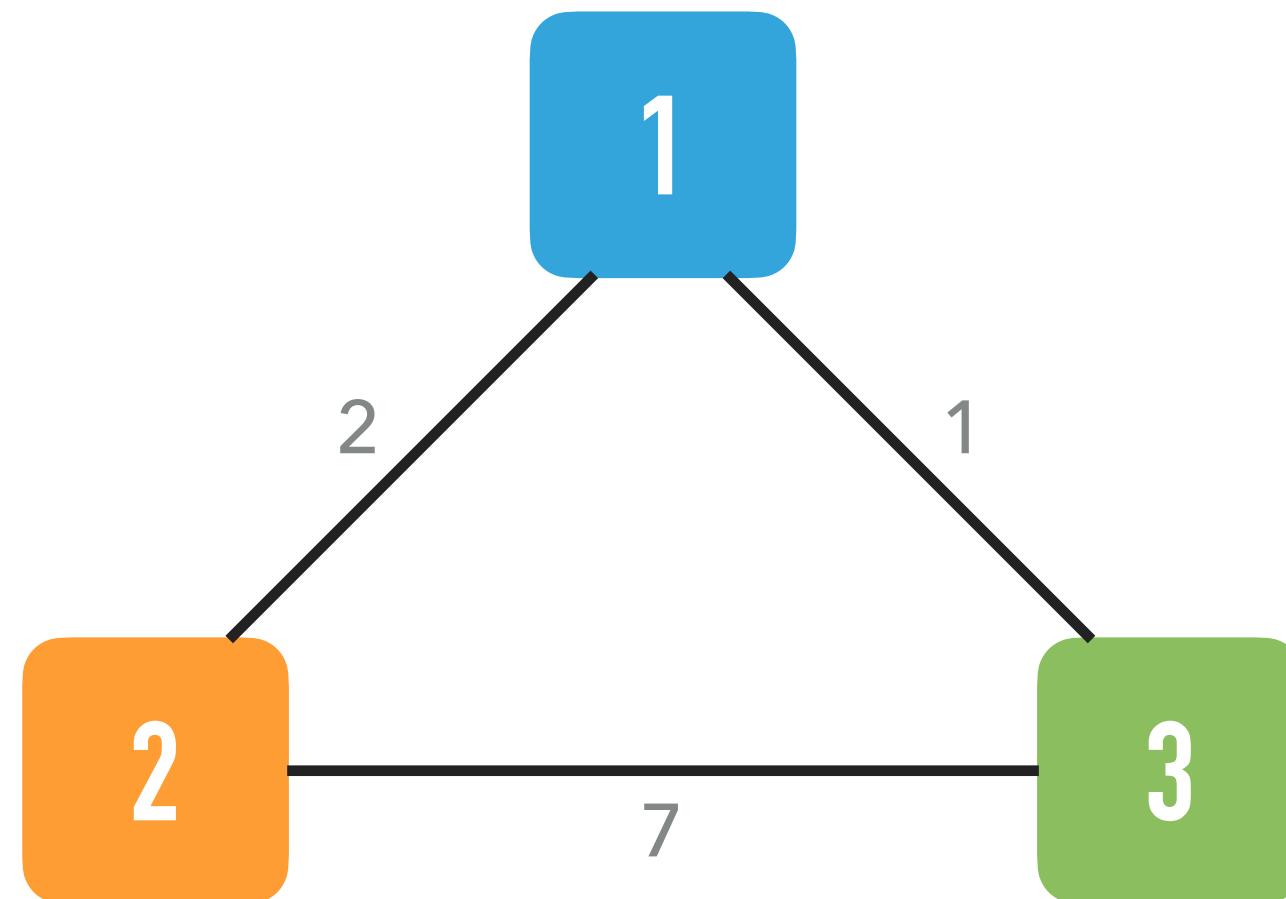


ITERAÇÃO #1



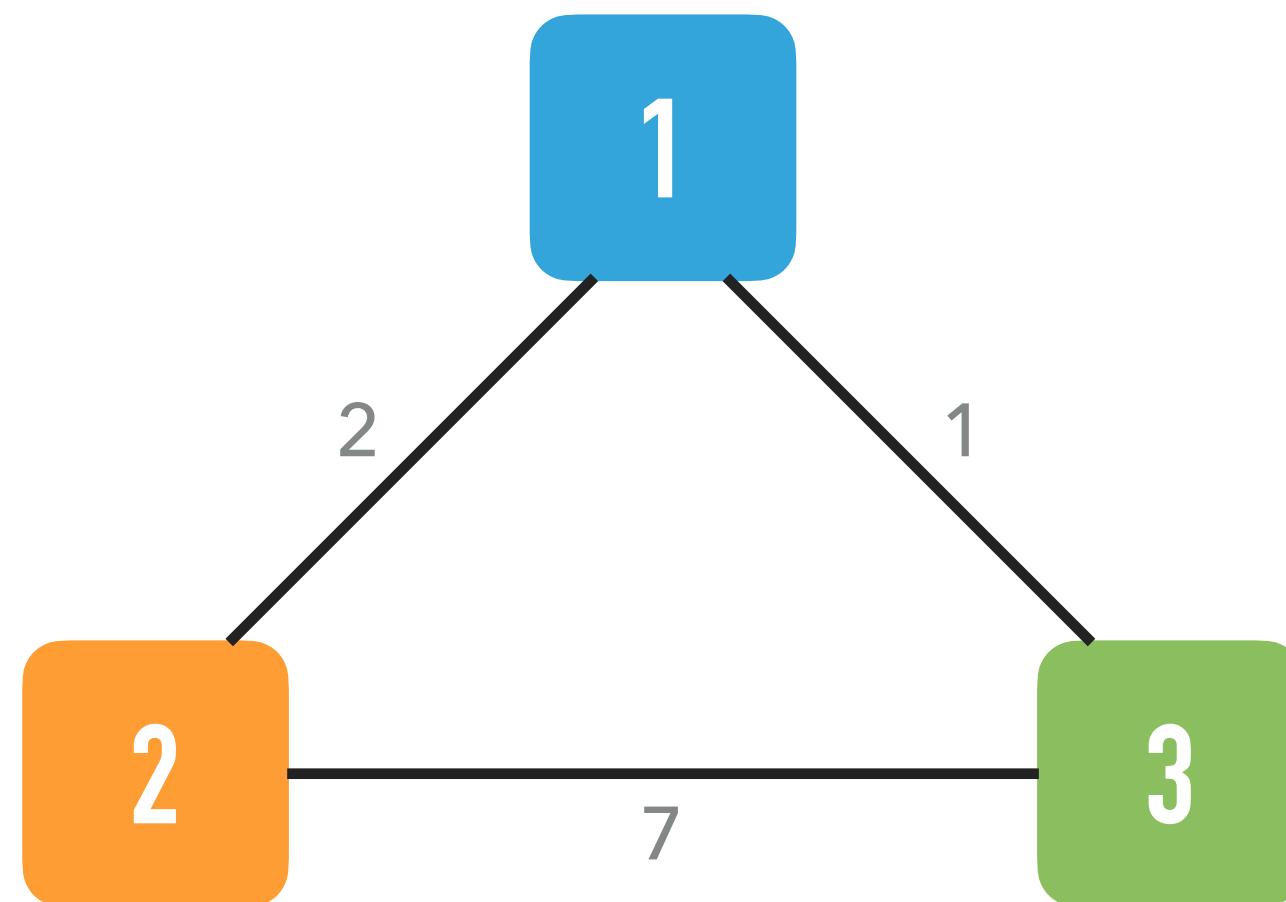
	RECEBE	ENVIA	PRÓXIMO SALTO (HOP)
1		(1, 0, 1)	-
2		(2, 0, 2)	-
3		(3, 0, 3)	-

ITERAÇÃO #2



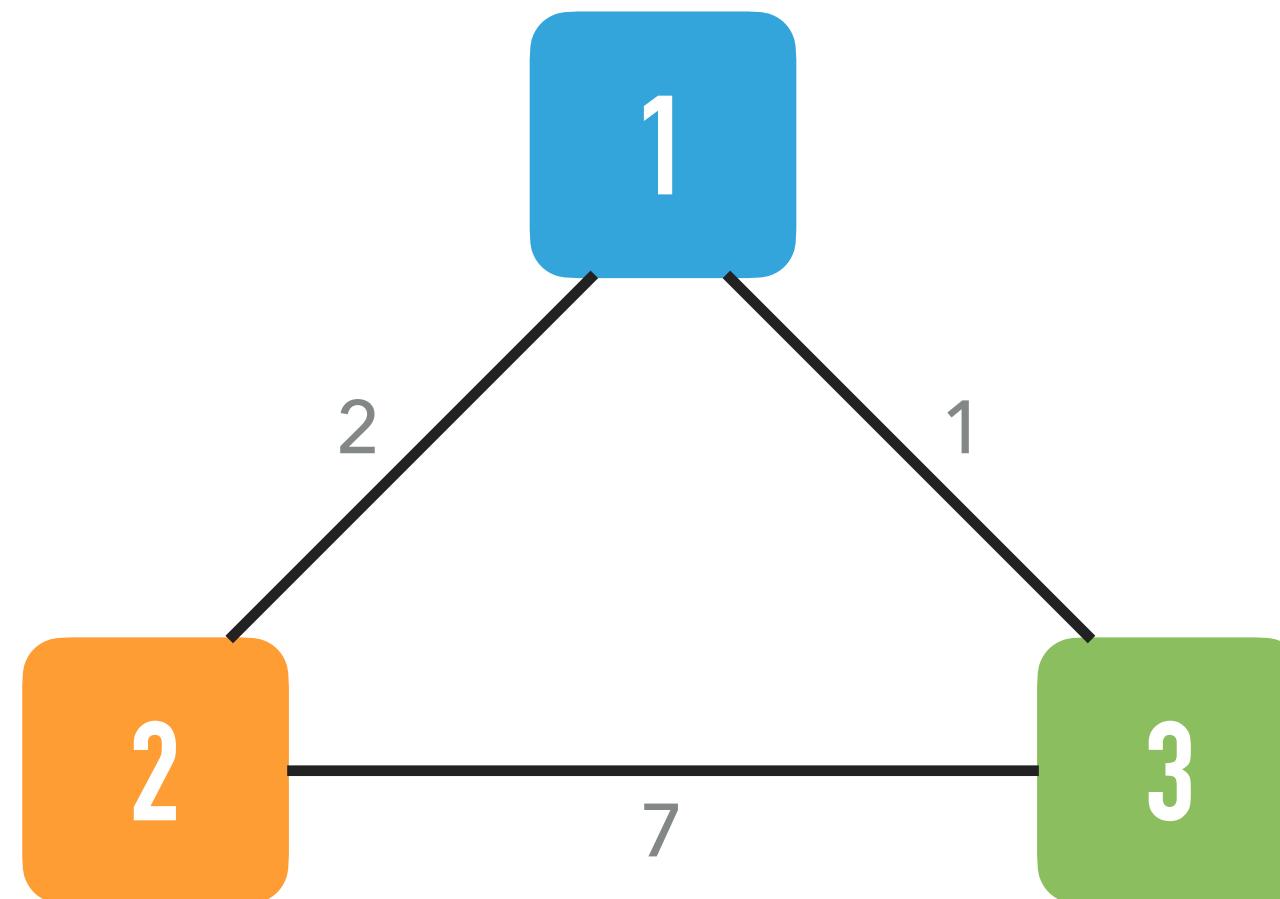
	RECEBE	ENVIA	PRÓXIMO SALTO (HOP)
1 (1, 0, 1)	(2, 0, 2) (3, 0, 3)	(2, 2, 1) (3, 1, 1)	- 2 3
2 (2, 0, 2)	(1, 0, 1) (3, 0, 3)	(1, 2, 2) (3, 7, 2)	1 - 3
3 (3, 0, 3)	(1, 0, 1) (2, 0, 2)	(1, 1, 3) (2, 7, 3)	1 2 -

ITERAÇÃO #3



	RECEBE	ENVIA	PRÓXIMO SALTO (HOP)
1	(1, 2, 2)		-
(1, 0, 1)	(3, 7, 2)		2
(2, 2, 1)	(1, 1, 3)		3
(3, 1, 1)	(2, 7, 3)		
2	(2, 2, 1)		1
(1, 2, 2)	(3, 1, 1)		-
(2, 0, 2)	(1, 1, 3)		1
(3, 7, 2)	(2, 7, 3)		
3	(2, 2, 1)		1
(1, 1, 3)	(3, 1, 1)		
(2, 7, 3)	(1, 2, 2)		1
(3, 0, 3)	(3, 7, 2)		-

ITERAÇÃO #4



	RECEBE	ENVIA	PRÓXIMO SALTO (HOP)
1	(1, 2, 2)		-
(1, 0, 1)	(3, 7, 2)	(3, 3, 2)	2
(2, 2, 1)	(1, 1, 3)	(2, 3, 3)	3
(3, 1, 1)	(2, 7, 3)		
2	(2, 2, 1)		1
(1, 2, 2)	(3, 1, 1)		-
(2, 0, 2)	(1, 1, 3)		1
(3, 3, 2)	(2, 7, 3)		
3	(2, 2, 1)		1
(1, 1, 3)	(3, 1, 1)		1
(2, 3, 3)	(1, 2, 2)	(3, 3, 2)	-
(3, 0, 3)	(3, 7, 2)		

PROTOCOLO DE ROTEAMENTO POR VETOR DE DISTÂNCIAS

Mensagens (Y, d, X)

- ▶ Para Y , a partir do nó X , anunciando distância d para Y
- ▶ Inicialmente cada router X inicializa sua tabela de roteamento para $(X, 0, -)$ e distância infinita para todos os outros destinos
- ▶ Routers anunciam seus vetores de distâncias integralmente (tabela de roteamento sem *next hops*)
- ▶ Ao receber a tabela de roteamento de um nó (X , por exemplo), cada nó:
 - ▶ Para cada destino Y da mensagem que chegou ($\text{distância}(X, Y) = d$)
 - ▶ SE $\text{distância_atual_para_}Y > d + \text{custo_do_link_para_}X$ ENTÃO
 - ▶ atualize $\text{distância_atual_para_}Y = d + \text{custo_do_link_para_}X$
 - ▶ atualize o próximo salto (*next hop*) para destino $Y = X$
- ▶ Se o menor caminho para o destino mudou, envie para todos os vizinhos

DOIS ASPECTOS DESTA ABORDAGEM

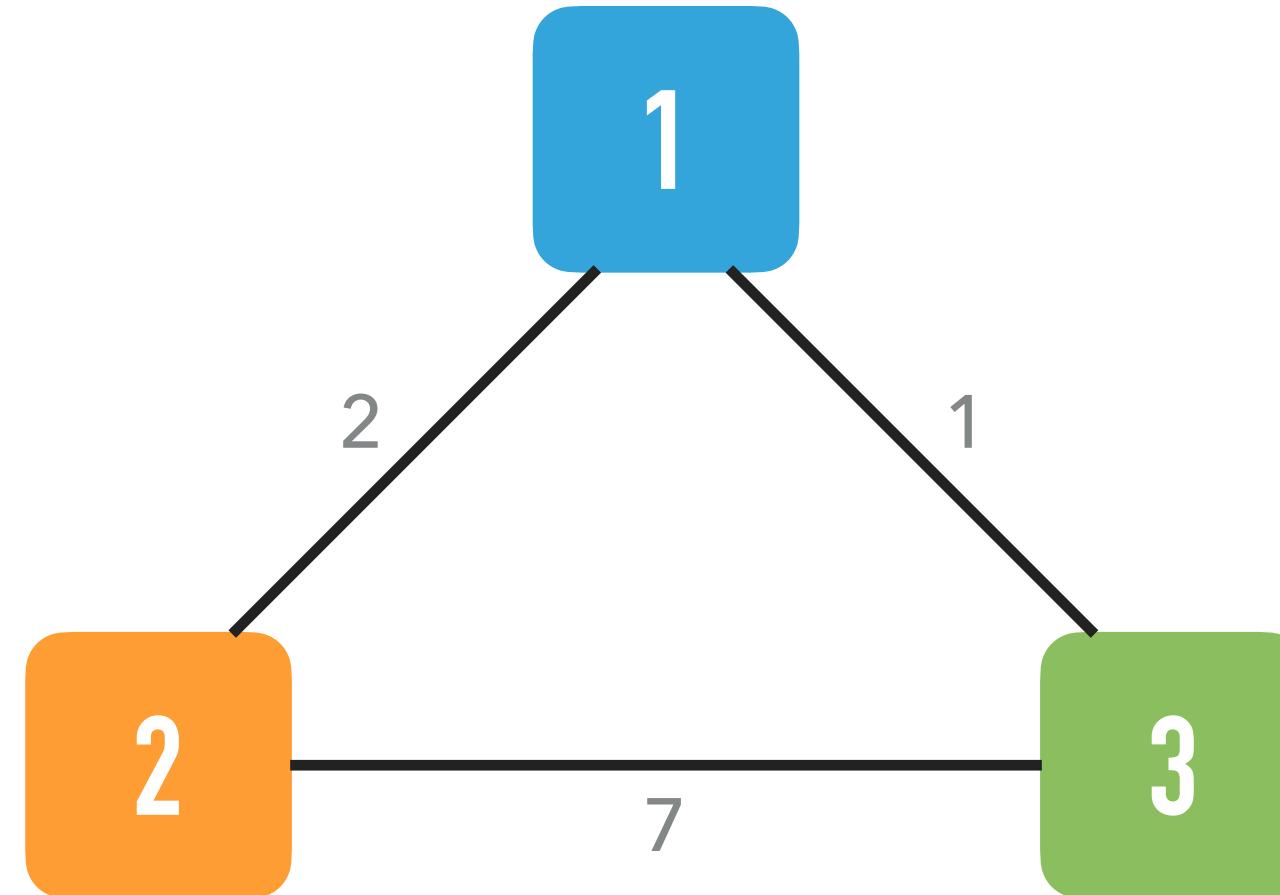
Protocolo:

- ▶ Troca de informação de roteamento entre os vizinhos
- ▶ O que trocar e quando trocar
- ▶ Protocolo RIP implementa o vetor de distâncias

Algoritmo:

- ▶ Como usar a informação dos vizinhos para atualizar sua tabela de roteamento?

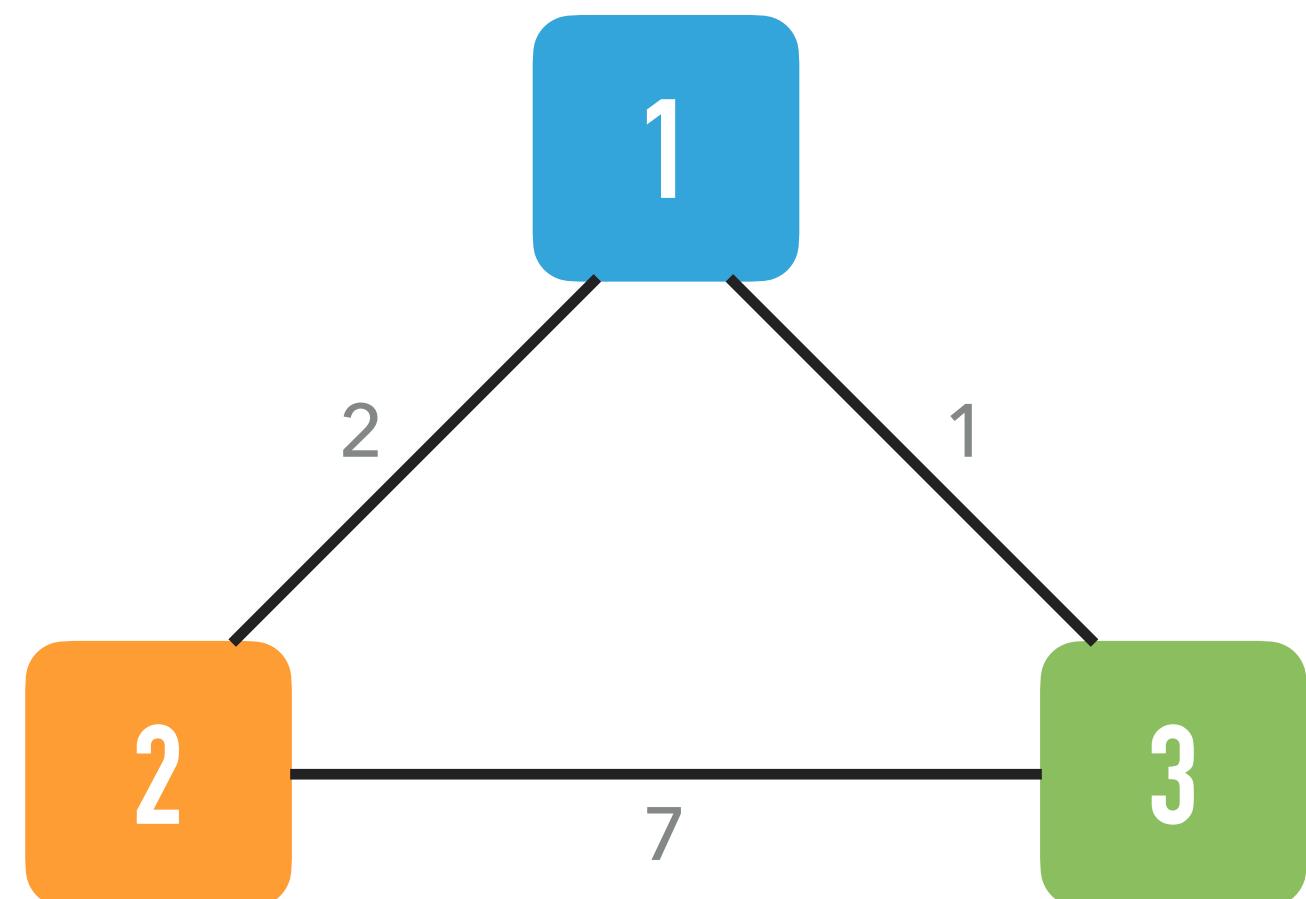
EXEMPLO COM PROTOCOLO DE VETOR DE DISTÂNCIAS



ITERAÇÃO #1

	DISTÂNCIA	PRÓXIMO SALTO (HOP)
1	0	-
2	infinito	
3	infinito	

	DISTÂNCIA	PRÓXIMO SALTO (HOP)
1	infinito	
2	0	-
3	infinito	

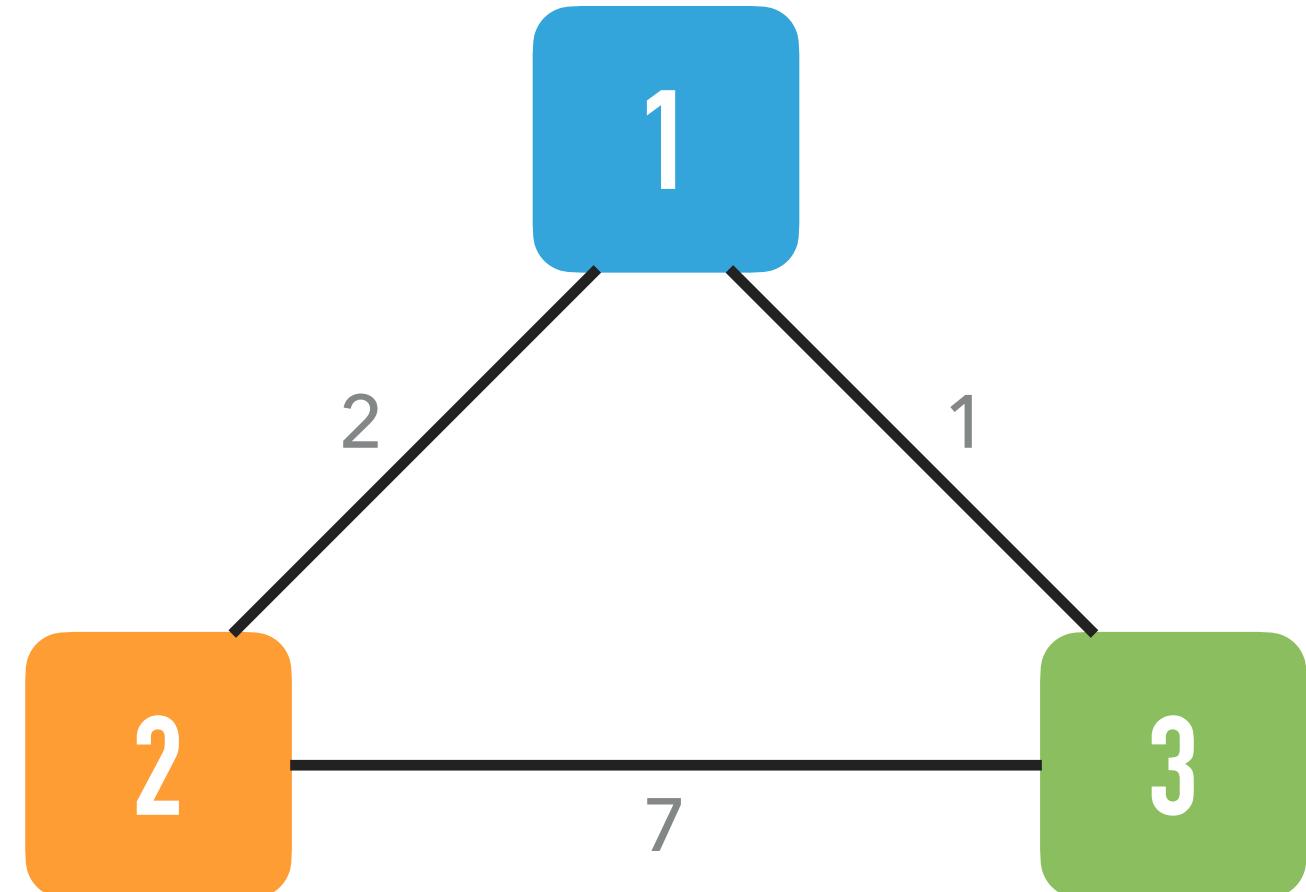


	DISTÂNCIA	PRÓXIMO SALTO (HOP)
1	infinito	
2	infinito	
3	0	-

ITERAÇÃO #2

	DISTÂNCIA	PRÓXIMO SALTO (HOP)
1	0	-
2	2	2
3	1	3

	DISTÂNCIA	PRÓXIMO SALTO (HOP)
1	2	1
2	0	-
3	7	3

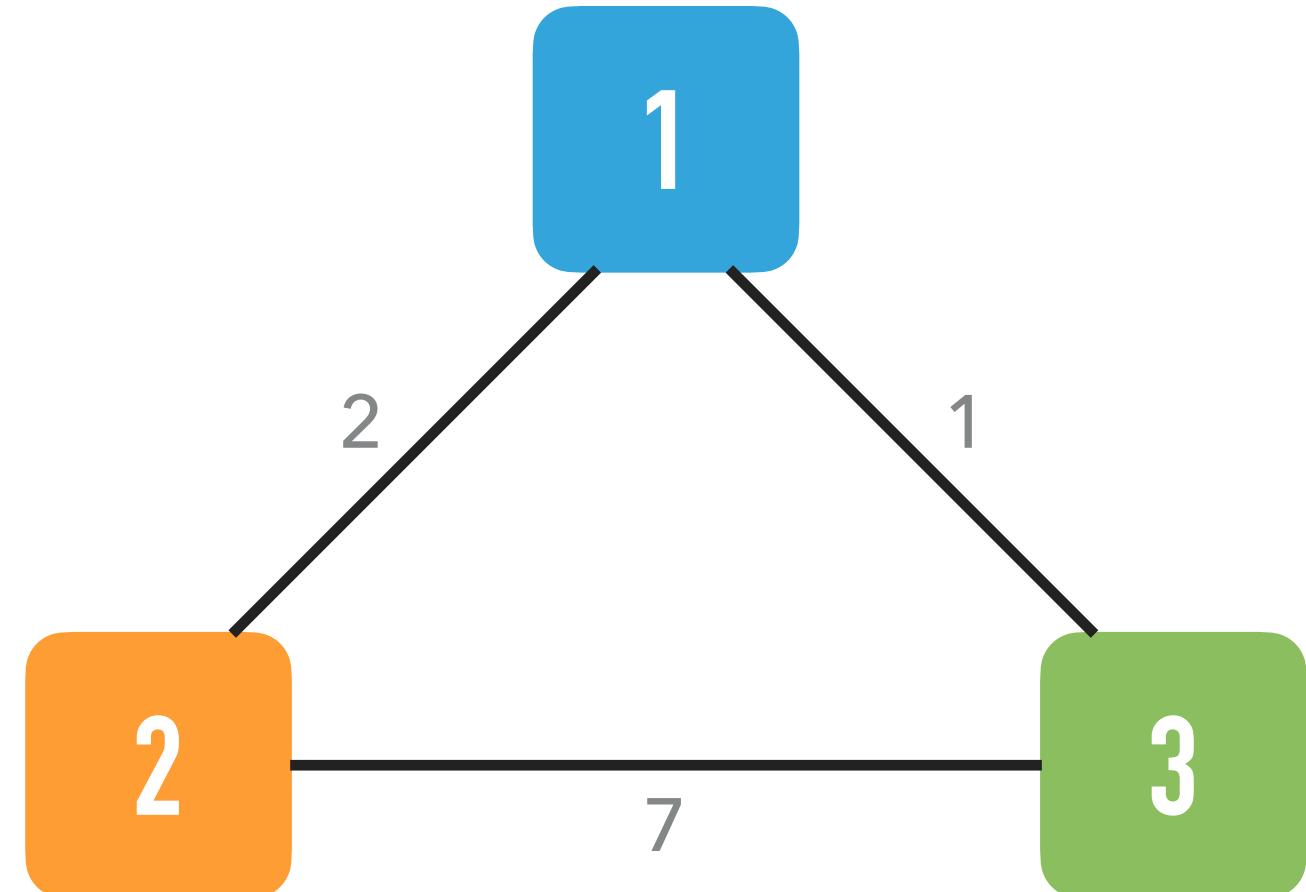


	DISTÂNCIA	PRÓXIMO SALTO (HOP)
1	1	1
2	7	2
3	0	-

ITERAÇÃO #3

	DISTÂNCIA	PRÓXIMO SALTO (HOP)
1	0	-
2	2	2
3	1	3

	DISTÂNCIA	PRÓXIMO SALTO (HOP)
1	2	1
2	0	-
3	3	1

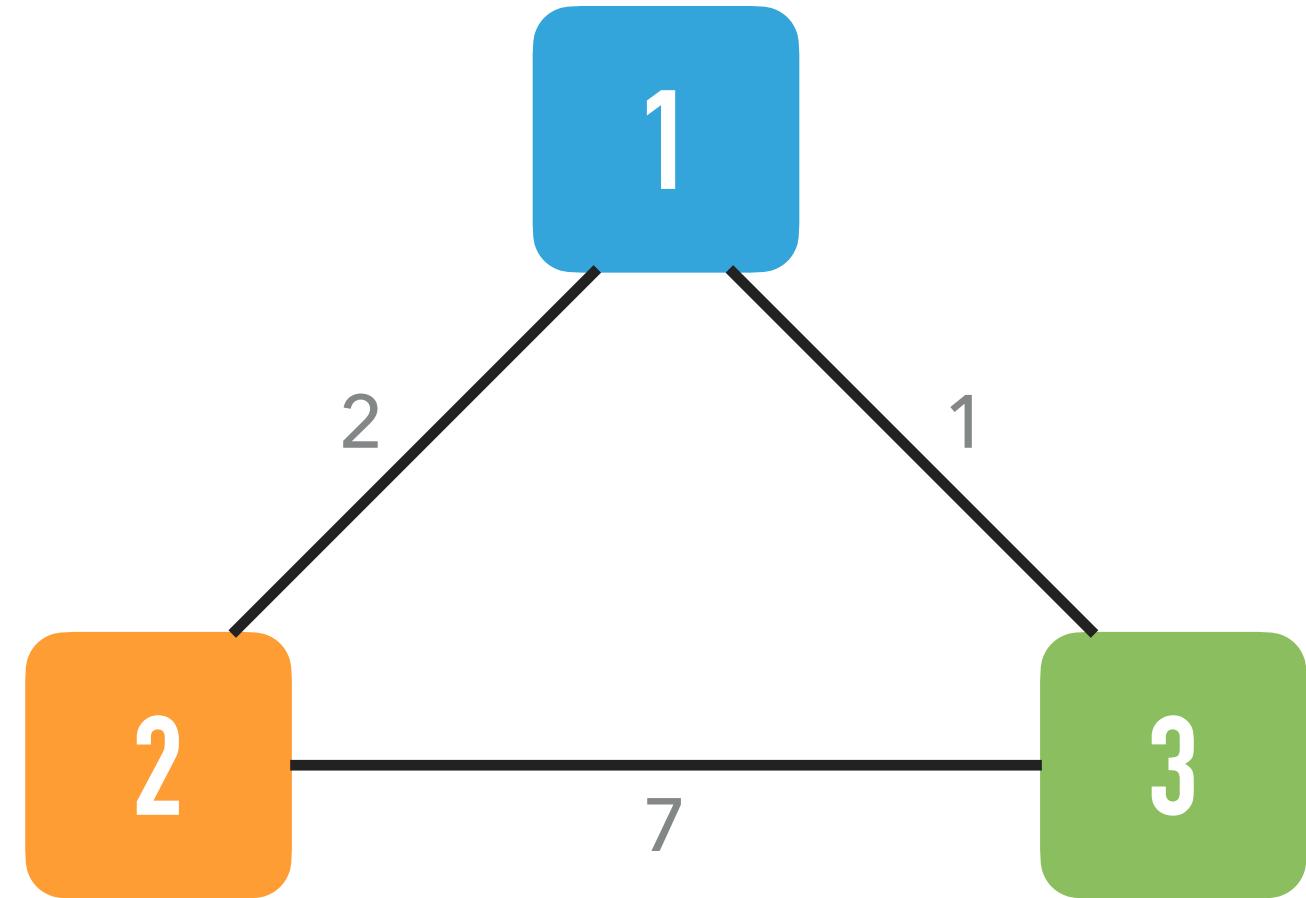


	DISTÂNCIA	PRÓXIMO SALTO (HOP)
1	1	1
2	3	1
3	0	-

ITERAÇÃO #4

	DISTÂNCIA	PRÓXIMO SALTO (HOP)
1	0	-
2	2	2
3	1	3

	DISTÂNCIA	PRÓXIMO SALTO (HOP)
1	2	1
2	0	-
3	3	1



	DISTÂNCIA	PRÓXIMO SALTO (HOP)
1	1	1
2	3	1
3	0	-

DO ALGORITMO AO PROTOCOLO

Algoritmo:

- ▶ Nós fazem uso do Bellman-Ford para calcular distâncias

Protocolo:

- ▶ Nós trocam vetores de distâncias
- ▶ Atualizam suas próprias tabelas de roteamento
- ▶ E trocam novamente...

OUTROS ASPECTOS DO PROTOCOLO

Quando enviar mensagens?

- ▶ Quando qualquer das suas distâncias $d(u, v)$ muda
- ▶ Periodicamente, para garantir consistência entre vizinhos

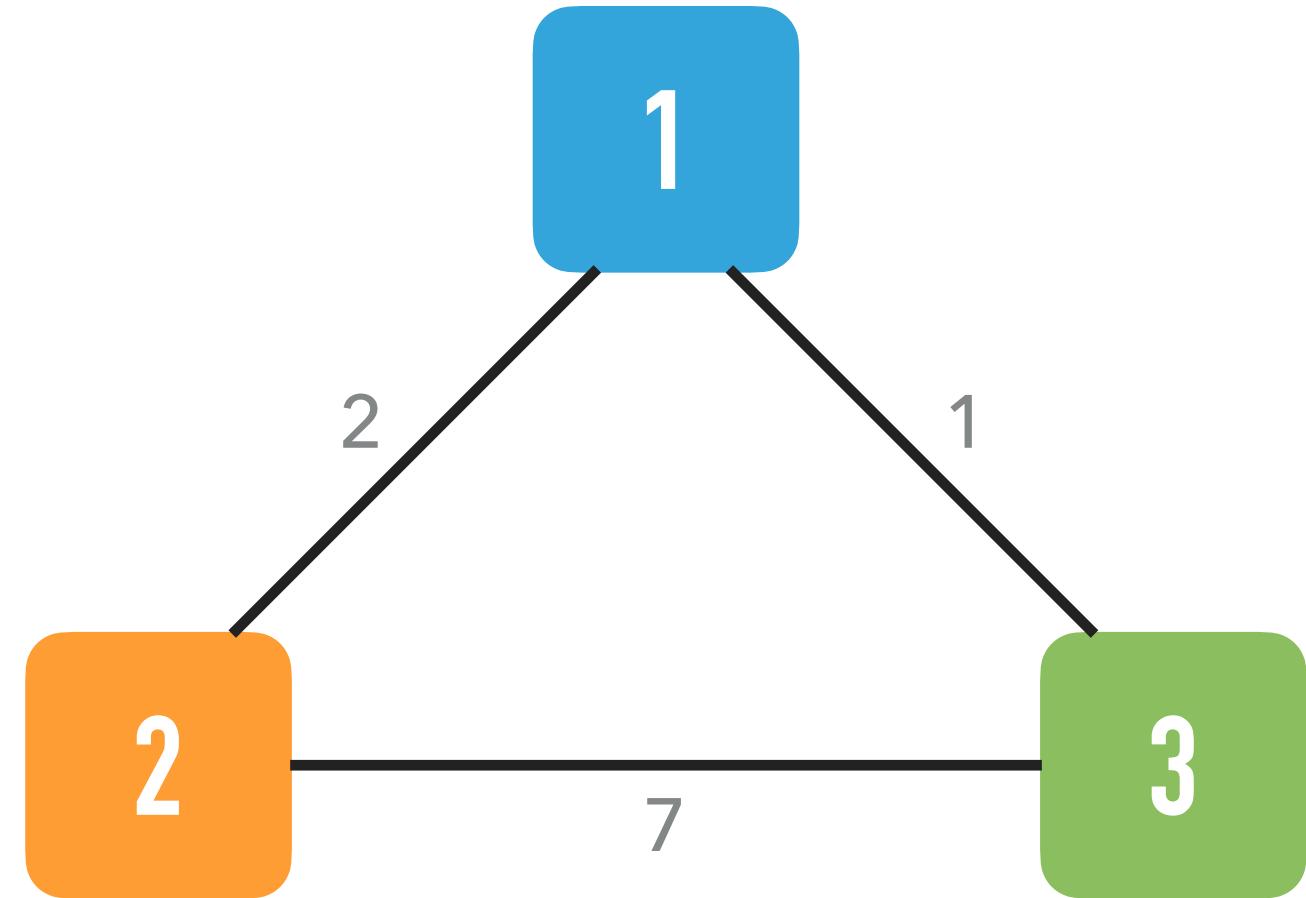
Qual informação você envia?

- ▶ Pode enviar o vetor de distâncias completo
- ▶ Enviar somente as entradas que foram atualizadas

REDE COM 3 NÓS

	DISTÂNCIA	PRÓXIMO SALTO (HOP)
1	0	-
2	2	2
3	1	3

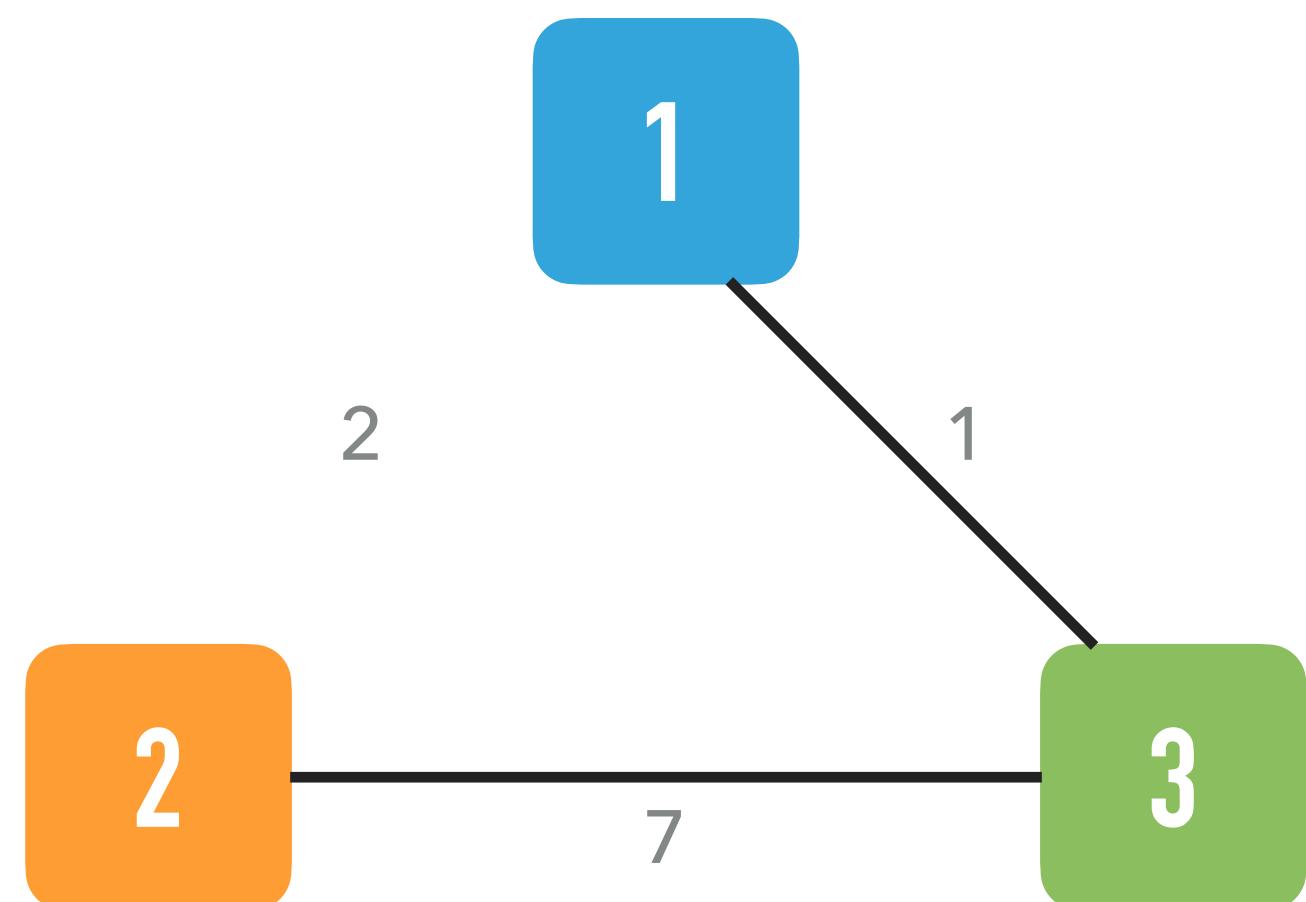
	DISTÂNCIA	PRÓXIMO SALTO (HOP)
1	2	1
2	0	-
3	3	1



	DISTÂNCIA	PRÓXIMO SALTO (HOP)
1	1	1
2	3	1
3	0	-

REDE COM 3 NÓS

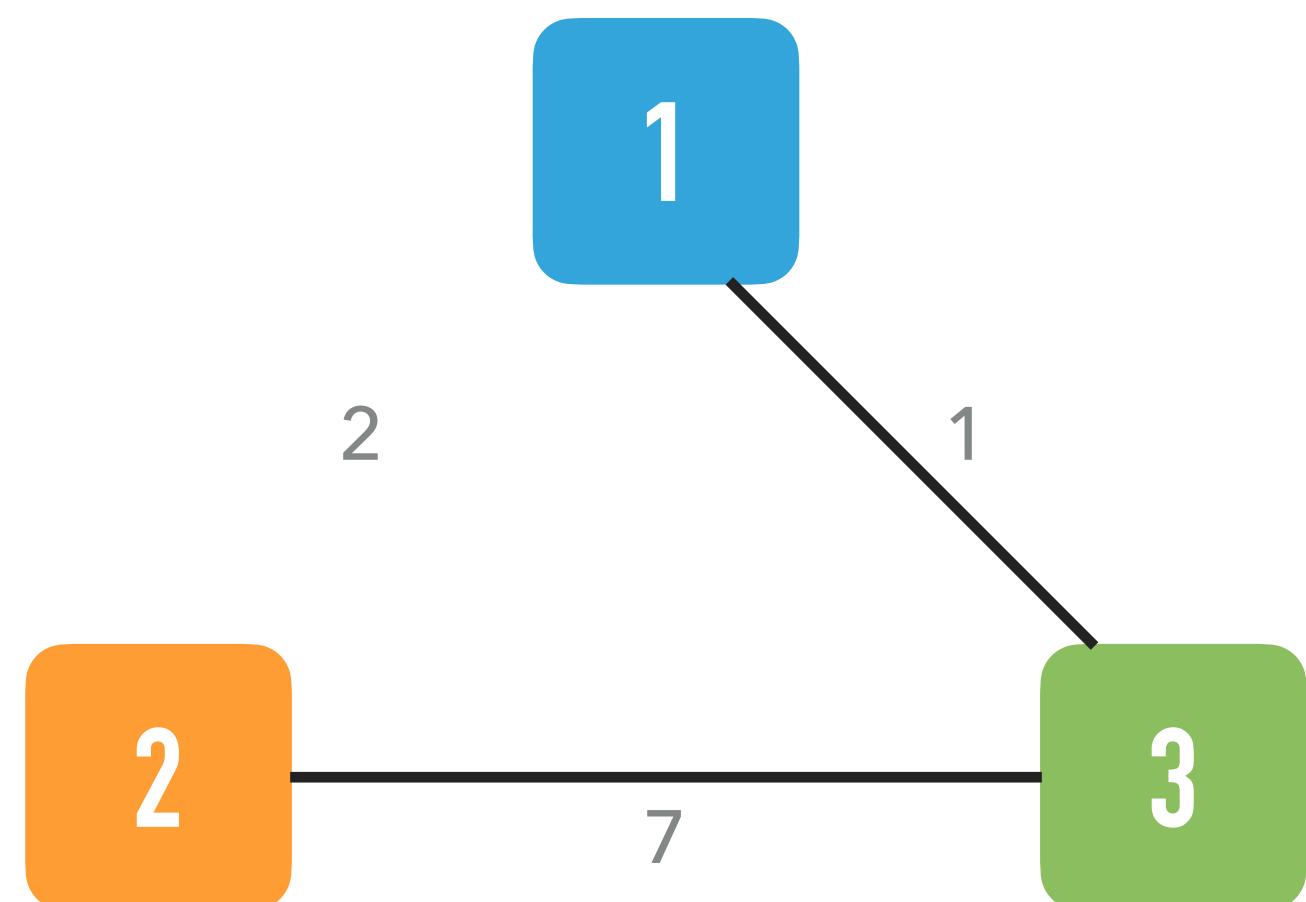
	DISTÂNCIA	PRÓXIMO SALTO (HOP)
1	0	-
2	infinito	
3	1	3



	DISTÂNCIA	PRÓXIMO SALTO (HOP)
1	1	1
2	3	1
3	0	-

ITERAÇÃO #1

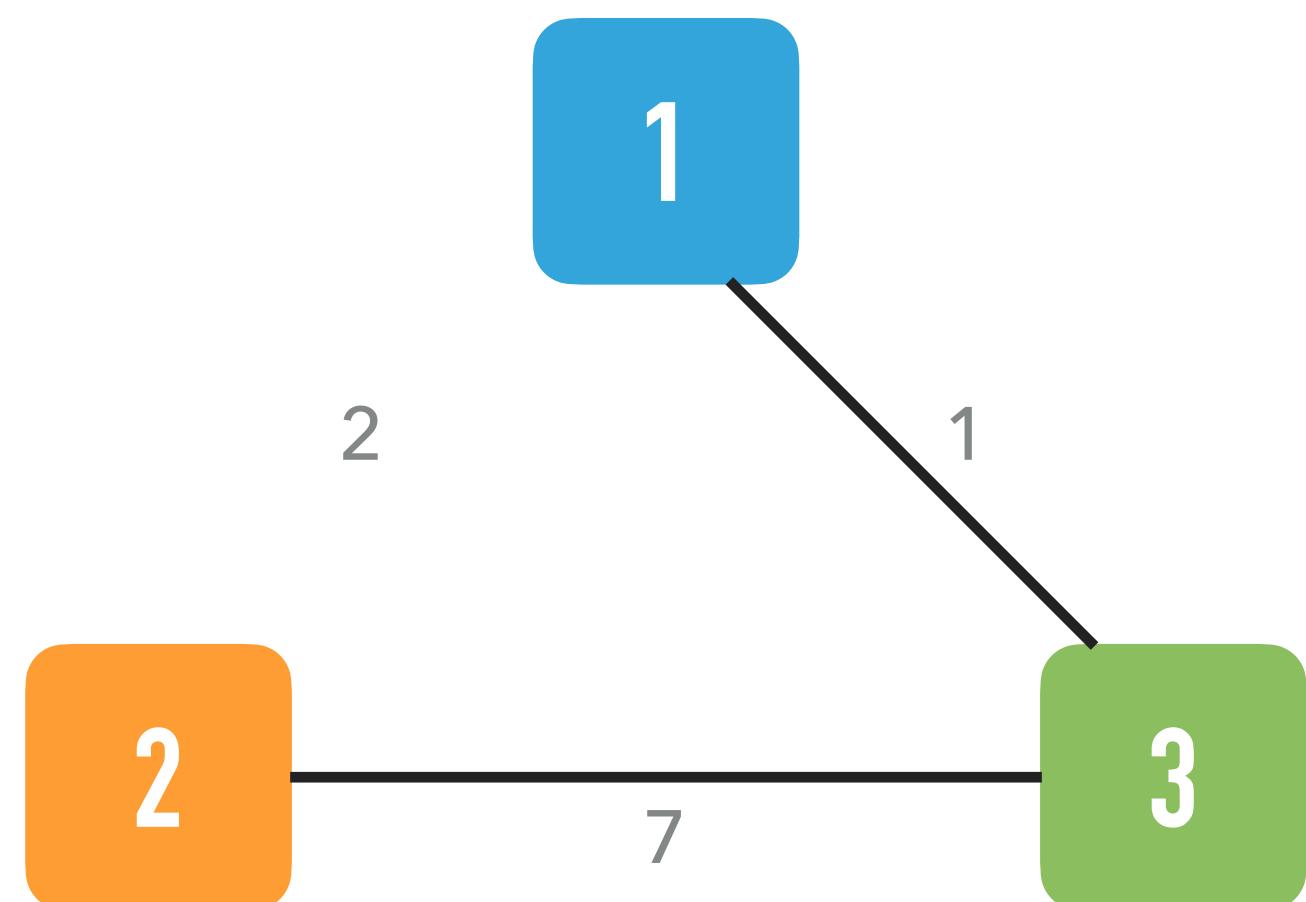
	DISTÂNCIA	PRÓXIMO SALTO (HOP)
1	0	-
2	4	3
3	1	3



	DISTÂNCIA	PRÓXIMO SALTO (HOP)
1	1	1
2	3	1
3	0	-

ITERAÇÃO #2

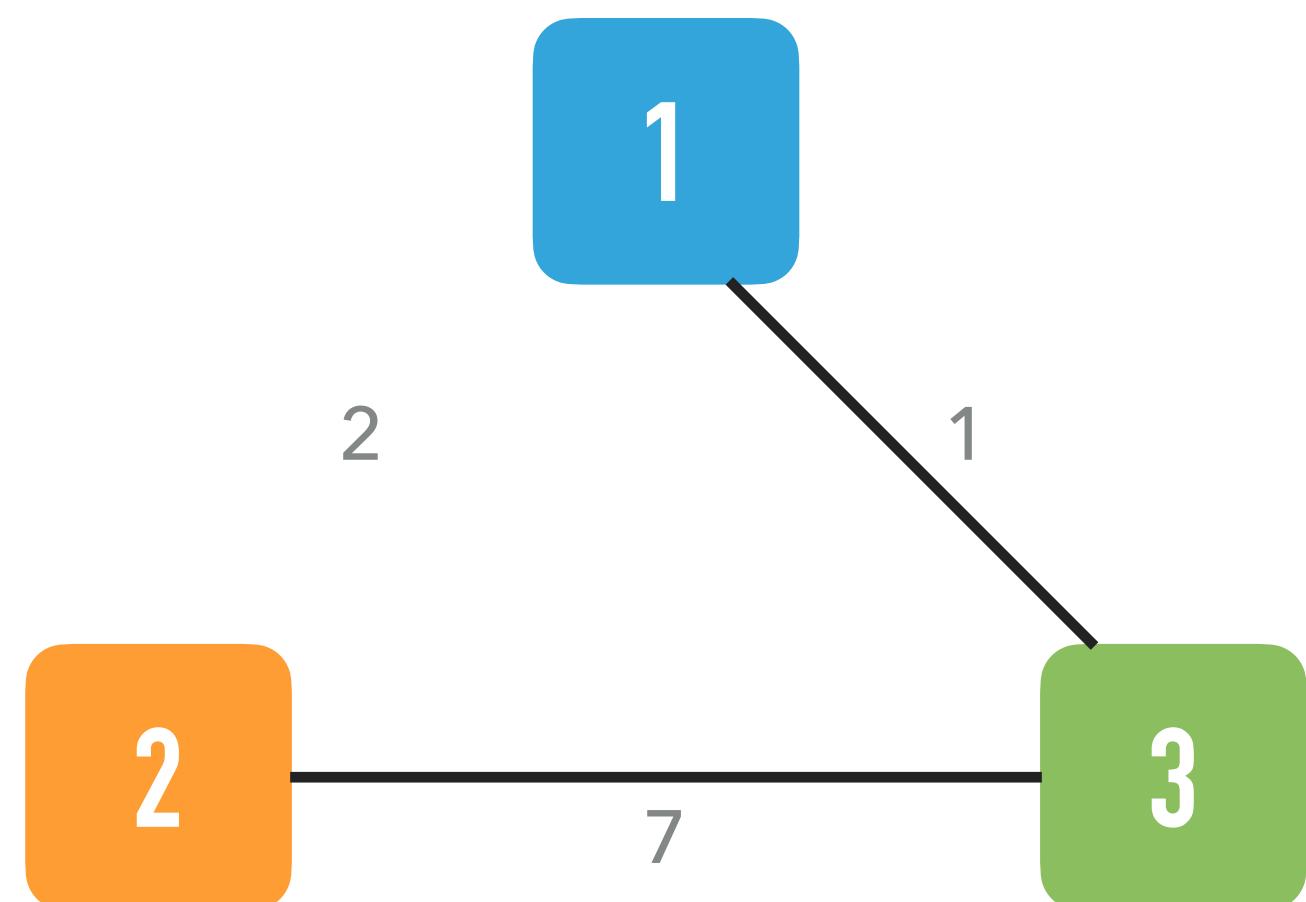
	DISTÂNCIA	PRÓXIMO SALTO (HOP)
1	0	-
2	4	3
3	1	3



	DISTÂNCIA	PRÓXIMO SALTO (HOP)
1	1	1
2	5	1
3	0	-

ITERAÇÃO #3

	DISTÂNCIA	PRÓXIMO SALTO (HOP)
1	0	-
2	6	3
3	1	3

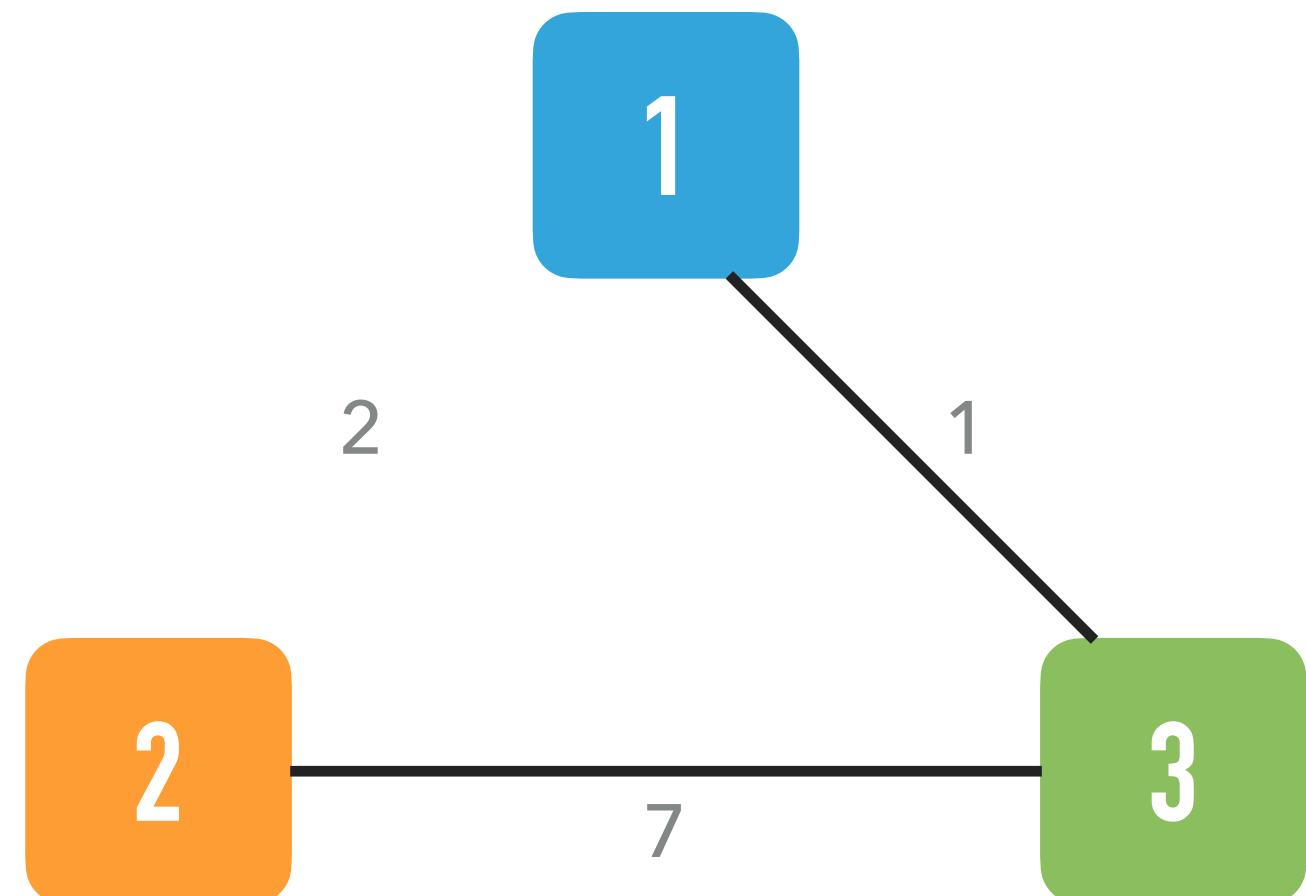


	DISTÂNCIA	PRÓXIMO SALTO (HOP)
1	1	1
2	5	1
3	0	-

ITERAÇÃO #4

	DISTÂNCIA	PRÓXIMO SALTO (HOP)
1	0	-
2	6	3
3	1	3

PROBLEMA da
CONTAGEM ao INFINITO
(count-to-infinity)



	DISTÂNCIA	PRÓXIMO SALTO (HOP)
1	1	1
2	7	1
3	0	-

COMO CORRIGIR O PROBLEMA DA CONTAGEM AO INFINITO?

Não anuncie um caminho de volta para o nó que é seu próximo salto no caminho

- ▶ Chamado de horizonte dividido (*split horizon*)
- ▶ Periodicamente, para garantir consistência entre vizinhos

Outra solução: se estiver usando o caminho do próximo salto, então:

- ▶ Avise para não usar seu caminho (informando um custo infinito)
- ▶ Chamado de envenenamento reverso (*poisoned reverse*)

Protocolos por vetor de distâncias podem convergir lentamente!

COMPARANDO A ESCALABILIDADE DOS PROTOCOLOS

Estado de enlace (*link state*)

- ▶ Flood global: estado de enlace de cada roteador
- ▶ Enviar por evento de *link*, ou periodicamente

Vetor de distâncias (*distance vector*)

- ▶ Enviar vetor somente para vizinhos
- ▶ Enviar sempre que o DV é modificado

Tradeoff

- ▶ LS: Envie para todo mundo e termine em tempo previsível
- ▶ DV: Envie localmente, e itere até atingir a convergência

O QUE VIMOS ATÉ ENTÃO...

Redes compartilhadas

- ▶ comutação de circuitos, comutação de pacotes, por que pacotes
- ▶ atraso de transmissão, atraso de propagação, atraso de enfileiramento

Princípios e objetivos arquiteturais

- ▶ modelo em camadas, princípio fim-a-fim
- ▶ objetos de projeto da Internet

Camada de enlace

- ▶ endereços MAC, meios broadcast, meios compartilhados
- ▶ quadros e enquadramento
- ▶ acesso aleatório ao meio *broadcast*: CSMA/CD
- ▶ problemas de escalabilidade de meios *broadcast*: por que Ethernet comutada
- ▶ *spanning tree protocol* (STP)

O QUE VIMOS ATÉ ENTÃO...

Camada de rede

- ▶ por que precisamos de uma camada de rede?
- ▶ a maneira correta de se pensar em tabelas de roteamento e protocolos
- ▶ protocolos de roteamento baseados no menor caminho
 - ▶ estado de enlace
 - ▶ vetor de distâncias
- ▶ Loops transientes, dead ends
- ▶ ainda vamos falar mais da camada de rede...