

# Számítógépes Hálózatok 2010

## 9. Hálózati réteg – Packet Forwarding, Link-State-Routing, Distance-Vector-Routing

### A hálózati réteg

- Lokális hálózatokat összeköthetünk hub-okkal, switch-ekkel, bridge-ekkel az alacsonyabb retegekben
  - Hub(fizikai réteg): kollíziók száma nagyon gyorsan növekszik
  - Switch (Adatkapcsolati réteg):
    - Az útvonalakról a forgalom „megfigyelésével” gyűjt információt
    - Ismeretlen célcím esetén a broadcast problémákat okoz
  - Az Internet kb.10 Mio. lokális hálózatot tartalmaz...
- Nagy hálózatokban a csomagok továbbításához útvonal információk szükségesek.
- A hálózati réteg feladatai
  - Az útvonal információk felépítése (route detection)
  - A csomagok továbbítása (packet forwarding)
- Az Internet-Protokoll lényegében hálózati réteg protokoll

## Routing-tábla és csomag továbbítás (packet forwarding)

### ● IP-Routing-tábla

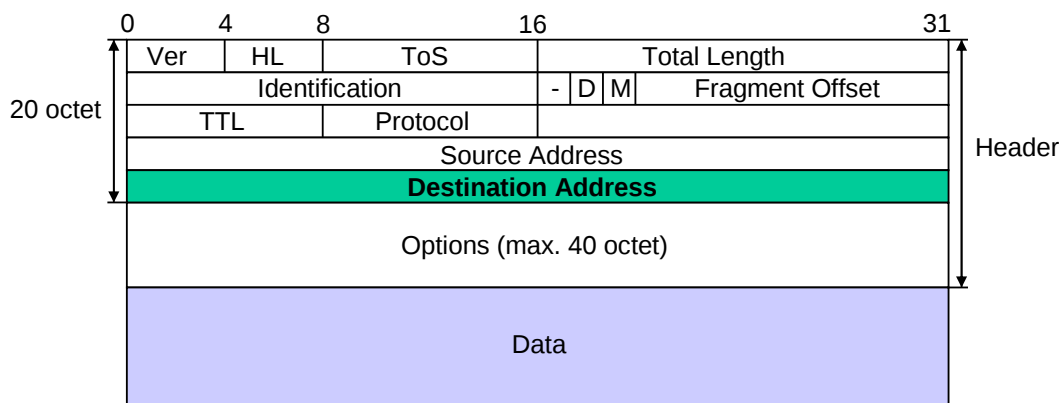
- Tartalmazza cél címekhez (destination) a következő számítógép (gateway) címét a hozzá vezető úton
- A cél meghatározhat egy számítógépet vagy egy egész sub-net-et
- Ezen kívül tartalmaz egy default-gateway-t

### ● Packet forwarding (korábban packet routing-nak nevezték)

- **IP csomag (datagram)** tartalmazza a küldő IP címét és a cél IP címét
- Amikor egy IP csomag megérkezik egy **router**hez:
  - Ha a cél IP cím = saját IP cím, akkor a csomagot kiszállítja
  - Ha a cél IP cím a routing-táblában van, továbbítja a megadott gateway-hez
  - Ha a cél IP-subnet a routing-táblában van, továbbítja a megadott gateway-nek
  - Egyébként továbbítja a default-gateway-nek

## Internet Protocol IP

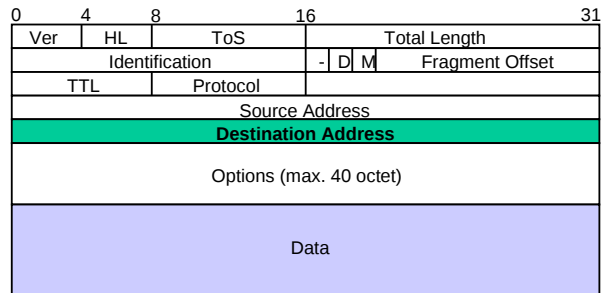
- Az adatok a küldőtől a cél-állomásig IP-csomagokban kerülnek átvitelre
- A csomagok fejléce tartalmazza a cél IP-címét
  - IPv4: 32 Bit-címek
  - IPv6: 128 Bit-címek



IPv4 csomag

## Csomag továbbítás az Internet Protokollban

- IP-csomag (datagram) tartalmazza
  - TTL (Time-to-Live): hop-ok számát
  - Küldő IP címét
  - Cél IP címét
- Egy csomag kezelése a routerben
  - $TTL = TTL - 1$
  - Ha  $TTL \neq 0$  akkor packet-forwarding a routing-tábla alapján
  - Ha  $TTL = 0$  vagy probléma lép fel a packet-forwarding-nél:
    - Töröljük a csomagot
    - Ha a csomag nem ICMP-csomag (Internet Control Message Protocol), akkor
      - Küldjük ICMP-csomagot (TTL equals 0 during transit), melyben
        - Küldő IP címe = aktuális IP cím
        - Cél IP címe = az eredeti küldő IP címe



## Statikus és dinamikus routing

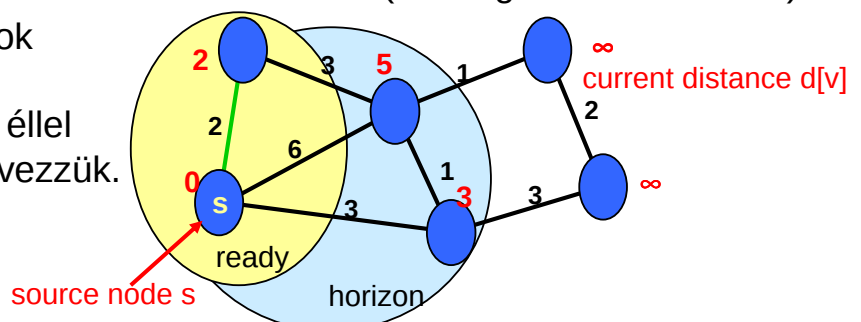
- Forwarding:
  - Csomagok továbbítása
- Routing:
  - Útvonalak meghatározása, azaz routing-tábla felépítése (route detection)
- Statikus routing
  - A routing-táblát manuálisan építjük fel
  - Kis és statikus LAN-ok esetén értelmes
- Dinamikus routing
  - A routing-tábla felépítése és aktualizálása automatizált
  - Centralizált algoritmus, pl. Link State
    - Egy/minden állomásnak ismerni kell minden információt
  - Decentrális algoritmus, pl. Distance Vector
    - minden routeren lokálisan dolgozik, lokális információkkal

## Legrövidebb utak fája – single source shortest paths

- Adott:
  - Egy irányított gráf  $G = (V, E)$ ,  $w : E \rightarrow \mathbf{R}_{\geq 0}$  nem negatív élsúlyokkal
  - Kezdő csomópont  $s \in V$
- Legyen
  - **P útvonal súlya**  $w(P) := \sum_{e \in P} w(e)$  az élek súlyainak összege P-ben
  - **u és v távolsága** G-ben,  $u, v \in V$ , egy legrövidebb út súlya G-ben u és v között :  $d(u, v) := \min\{ w(P) : P \text{ egy út } u\text{-től } v\text{-hez G-ben} \}$ .
- Keressük:
  - egy legrövidebb utat s kezdő csomóponttól minden más  $v \in V \setminus \{s\}$  csomóponthoz G-ben
  - Feltesszük, hogy minden  $v \in V \setminus \{s\}$  elérhető s-ből. Nem elérhető csomópontokhoz nem létezhet legrövidebb út sem
- Megoldás:
  - Egy fa, melynek gyökere s és minden  $v \in V \setminus \{s\}$  csomóponthoz tartalmaz egy legrövidebb utat s-től v-hez G-ben

## Dijkstra algoritmus

- **Ötlet:** A legrövidebb utakat hosszuk szerint növekvő sorrendben számítjuk ki.
- Minden  $v \in V$  csomópontokhoz kiszámítjuk a következő értékeket:
  - **$d[v]$ :** egy legrövidebb út hossza s-től v-hez,
  - **$pred[v]$ :** a v-t megelőző csomópont egy legrövidebb úton s-től v-hez.
- Az algoritmus végrehajtása után az élhalmaz  $\{ (pred[v], v) : v \in V \setminus \{s\} \}$  megadja egy legrövidebb utak fáját s gyökérrel G-ben.
- Egy v csomópontot „kész”-nek jelölünk:  **$ready[v] = true$** , ha már meghatároztunk egy legrövidebb utat s-től v-hez (röv. legrövidebb s-v utat).
- A „nem kész” csomópontok halmazát, amelyeket egy „kész” csomópontból egy éllel elérünk, **horizont**-nak nevezzük.



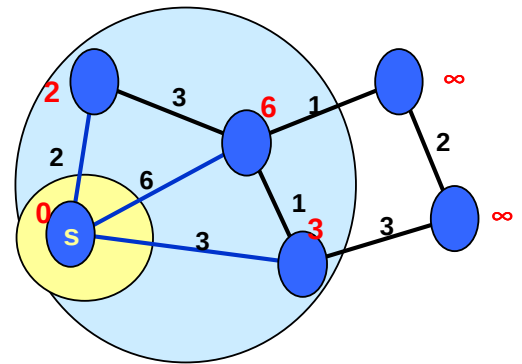
## Dijkstra algoritmus

### • Invariánsok:

- Minden horizont belüli csomópontot egy **Q priority-queue**-ban tárolunk, úgy hogy minden  $v \in Q$  csomópontra a következő érvényes:
  - $d[v]$  egy legrövidebb s-v út hossza mindazon utak között, melyek v-n kívül csak „kész” csomópontokat tartalmaznak,
  - $pred[v]$  a v-t megelőző csomópont egy ilyen úton,
  - v prioritása Q-ban  $d[v]$

### • Inicializálás

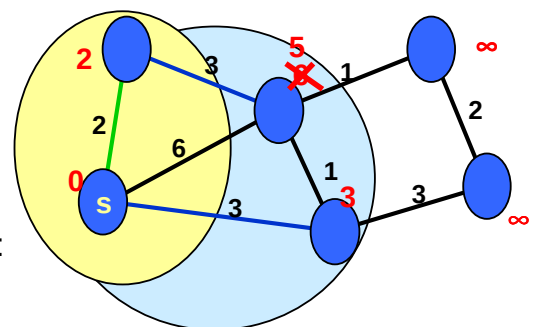
- $d[s] := 0$ ,  $ready[s] := true$ ,
- s minden v szomszédjára:
  - $d[v] := w(s, v)$ ,  $pred[v] := s$ ,  $ready[v] := false$ ,
  - $Q.Insert(v, d[v])$ .
- Minden  $v \in V \setminus \{s\}$  csomópontra:
  - $d[v] := \infty$ ,  $ready[v] := false$ .



## Dijkstra algoritmus

- Az invariánsok megőrzése egy iteráció után
  - Minden lépésben egy új csomópont lesz „kész”, egy csomópont v minimális prioritással.
  - $d[v]$  már tartalmazza a helyes értéket.
 

Mivel v minimális prioritású csomópont, minden olyan s-v út súlya, amely „nem kész” csomópontot is tartalmaz, legalább olyan nagy, mint annak az útnak a hossza, amit már megtaláltunk a csak „kész” csomópontokat tartalmazó utak között.
  - Legyen  $Adj[v] := \{ u : (v, u) \in E \}$ ,  $v \in V$ , a v-hez adjacens csomópontok halmaza
  - minden  $u \in Adj[v]$ , ha  $u \in Q$ , meg kell vizsgálni, hogy s-től u-hoz direkt v-ből egy rövidebb út vezet-e, mint azok az utak, amik csak v-től különböző „kész” csomópontot tartalmaznak. Ha igen, akkor aktualizáljuk
    - $pred[u] := v$  és  $d[u] := d[v] + w(v, u)$ ,
    - csökkentsük u prioritását Q-ban.
  - minden  $u \in Adj[v]$ , ha  $u \notin Q$  és u „nem kész”:
    - $pred[u] := v$ ,  $d[u] := d[v] + w(v, u)$ ,
    - u-t be kell szűrni Q-ba  $d[u]$  prioritással.



## Dijkstra algoritmusa

**Dijkstra(G,s,w)**

Output: egy legrövidebb utak fája  
 $T=(V,E')$  G-ben s gyökérrel

```
01  $E' := \emptyset$ ;  
02  $ready[s] := true$ ;  
03  $ready[v] := false; \quad \forall v \in V \setminus \{s\}$ ;  
04  $d[s] := 0$ ;  
05  $d[v] := \infty; \quad \forall v \in V \setminus \{s\}$ ;  
  
06 priority_queue Q;  
  
07 forall  $v \in Adj[s]$  do  
08    $pred[v] := s$ ;  
09    $d[v] := w(s,v)$ ;  
10   Q.Insert(v,d[v]);  
11 od
```

```
12 while  $Q \neq \emptyset$  do  
13    $v := Q.DeleteMin()$ ;  
14    $E' := E' \cup \{(pred[v],v)\}$ ;  
15    $ready[v] := true$ ;  
16   forall  $u \in Adj[v]$  do  
17     if  $u \in Q$  and  $d[v] + w(v,u) < d[u]$  then  
18        $pred[u] := v$ ;  
19        $d[u] := d[v] + w(v,u)$ ;  
20       Q.DecreasePriority(u,d[u]);  
21     else if  $u \notin Q$  and not  $ready[u]$  then  
22        $pred[u] := v$ ;  
23        $d[u] := d[v] + w(v,u)$ ;  
24       Q.Insert(u,d[u]);  
25     fi  
26   od  
27 od
```

## Dijkstra algoritmusa

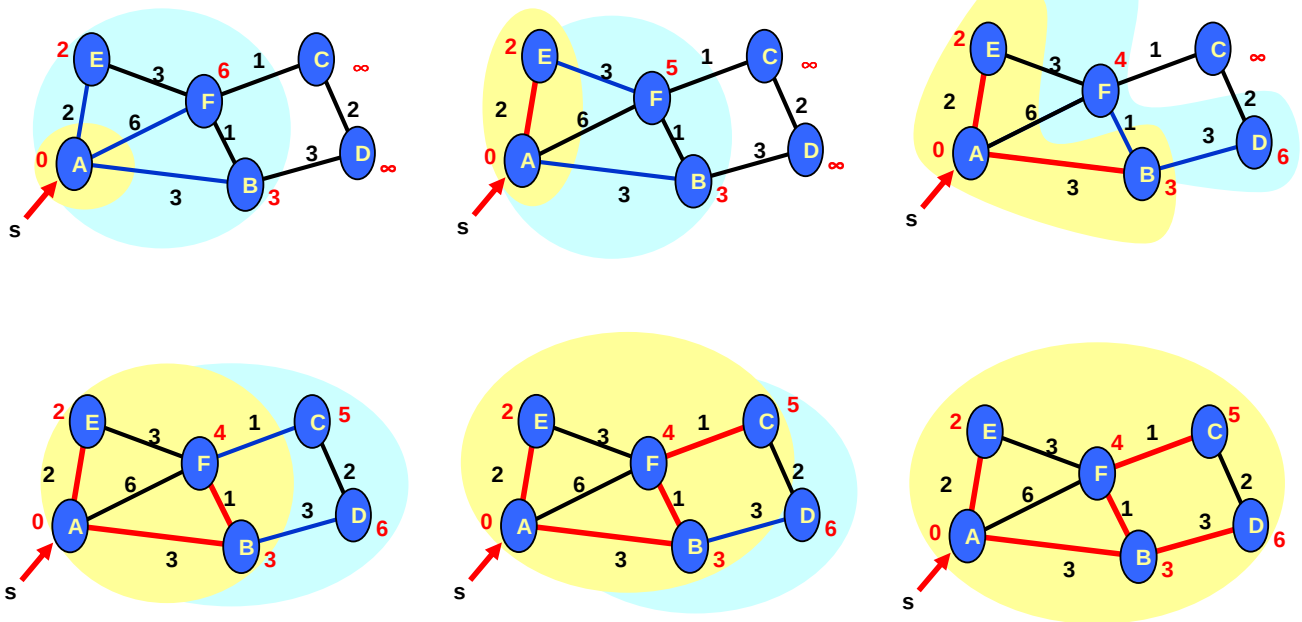
Futási idő (Fibonacci Heap-pel):

- # Q.Insert():  $n$  (csomópontonként 1) --  $O(n)$  idő
- # Q.DeleteMin():  $n$  (csomópontonként 1) --  $O(n \log n)$  idő
- # Q.DecreasePriority():  $\leq m$  (élenként  $\leq 1$ ) --  $O(m)$  idő
- # A teszt a 17. és 21. sorban:  $m$  (élenként 1) --  $O(m)$  idő
- Inicializálás:  $O(n)$  idő
- Összesen:  $O(n \log n + m)$  idő

Tárigény:  $O(n+m)$

## Dijkstra: Példa

szimmetrikusan irányított élek



## Bellman-Ford algoritmus

- Negatív élsúlyok esetén Dijkstra algoritmus nem működik
- Bellman-Ford algoritmus (1957) megoldja a problémát  $O(|V| |E|)$  idő alatt.
- Dinamikus programozás: a  $k$ -adik iteráció után,  $k=1, \dots, |V|-1$ , minden  $v \in V$ :
  - ha  $d[v] \neq \infty$ , akkor  $d[v]$  egy  $s$ - $v$  út  $P_{sv}$  súlya és  $d[v]$  nem nagyobb mint egy legrövidebb  $s$ - $v$  út súlya, amely  $\leq k$  élt tartalmaz
  - $\text{pred}[v] = \emptyset$  ha  $d[v] = \infty$ , egyébként pedig  $(\text{pred}[v], v) \in E$  az utolsó él a  $P_{sv}$  úton

### Bellman-Ford( $G, s, w$ )

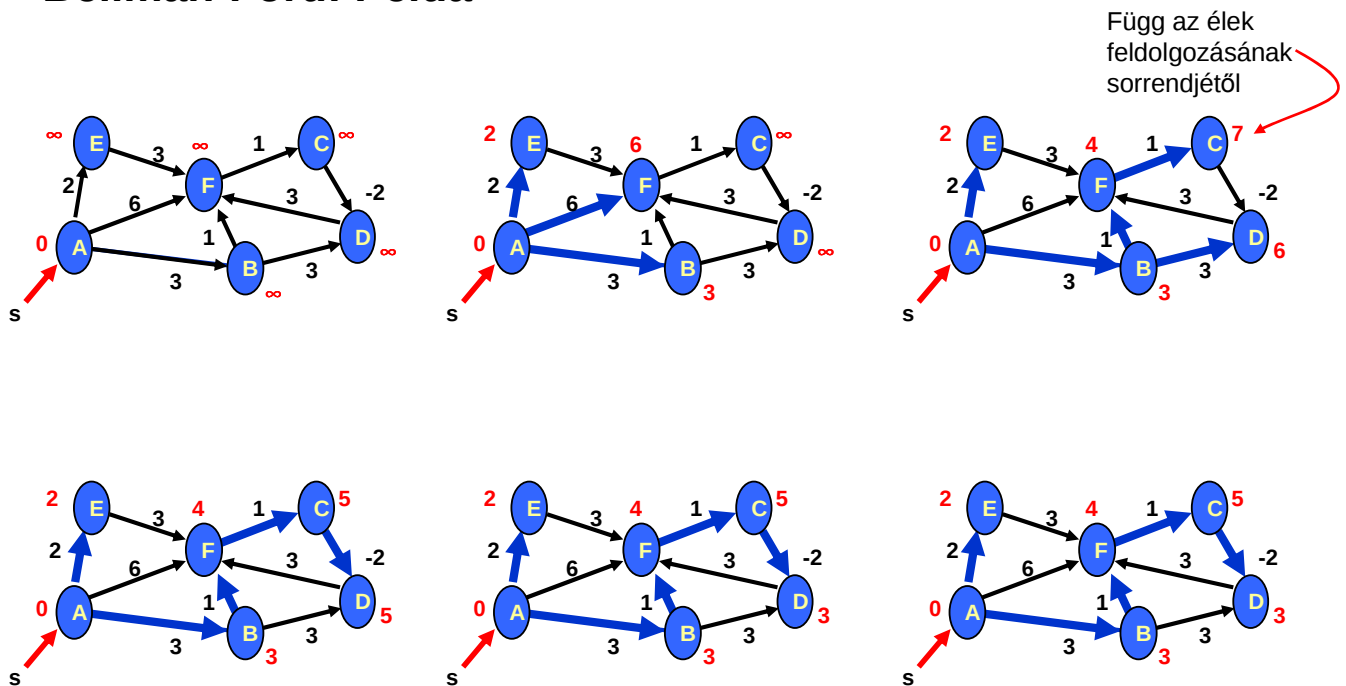
```

01 forall  $v \in V$  do
02    $d[v] := \infty$ ;  $\text{pred}[v] := \emptyset$ 
03  $d[s] := 0$ 

04 for  $k := 1$  to  $|V| - 1$  do
05   forall  $(u, v) \in E$  do
06     if  $d[u] + w(u, v) < d[v]$  then
07        $d[v] := d[u] + w(u, v)$ 
08        $\text{pred}[v] := u$ 

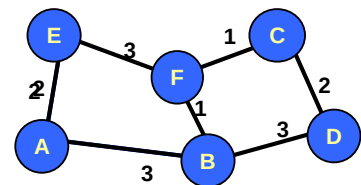
09 forall  $(u, v) \in E$  do
10   if  $d[u] + w(u, v) < d[v]$  then
11     error „negatív súlyú ciklust találtunk”
    
```

## Bellman-Ford: Példa



## Distance Vector Routing Protokoll

- A Bellman-Ford algoritmusnak az **elosztott** változatát használja, azaz minden csomópont csak a direkt szomszédjaival kommunikál
- **Asszinkron** működés
  - A csomópontoknak nem ugyanabban a „körben” kell információkat cserélniük
- Minden router nyilvántart egy táblát minden lehetséges célhoz egy bejegyzéssel (**distance vector**)
  - egy bejegyzés tartalmazza
    - a legrövidebb út (becsült) költségét (delay, vagy #hops)
    - a következő csomópont címét ezen az úton (next hop)
- minden router ismeri a költséget a direkt szomszédaihoz
- Periodikusan elküldi a tábláját minden szomszédjának
- Amikor egy router megkapja a szomszéd tábláját aktualizálja a saját tábláját



Initial distance vector of A

| A | cost     | next hop |
|---|----------|----------|
| B | 3        | B        |
| C | $\infty$ | -        |
| D | $\infty$ | -        |
| E | 2        | E        |
| F | $\infty$ | -        |

Initial distance vector of B

| B | cost     | next hop |
|---|----------|----------|
| A | 3        | A        |
| C | $\infty$ | -        |
| D | 3        | D        |
| E | $\infty$ | -        |
| F | 1        | F        |

A's vector after A received B's vector

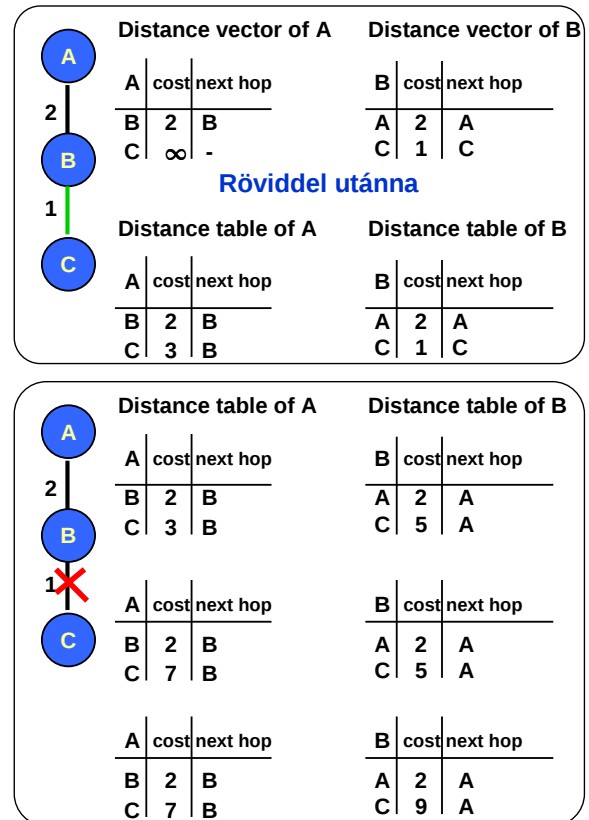
| A | cost     | next hop |
|---|----------|----------|
| B | 3        | B        |
| C | $\infty$ | -        |
| D | 6        | B        |
| E | 2        | E        |
| F | 4        | B        |

A's final distance vector

| A | cost | next hop |
|---|------|----------|
| B | 3    | B        |
| C | 5    | B        |
| D | 6    | B        |
| E | 2    | E        |
| F | 4    | B        |

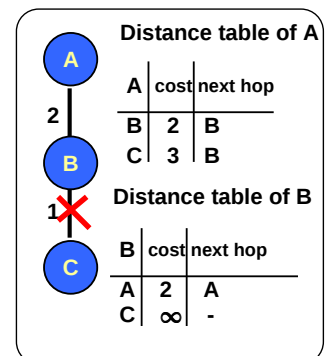
## “Count to Infinity” Probléma

- „Jó hír” gyorsan terjed
  - Új kapcsolat létrejöttkor gyorsan aktualizálódnak a táblák
- „Rossz hír” lassan terjed
  - Kapcsolat kiesik
  - A szomszédok felváltva növelik a távolságokat
  - “Count to Infinity” Probléma
    - A és B nem tudja, hogy C nem elérhető (amíg a távolság el nem ér egy limitet, amit  $\infty$ -nek tekintenek)
  - Ciklusok keletkezhetnek



## “Count to Infinity” Probléma

- Módosítások a Distance-Vector routing protollokban
  - a ping-pong-ciklusokat (count to infinity) megakadályozásához
  - **split horizon**: olyan utakat nem küld vissza a csomópont annak a szomszédjának, amit tőle „tanult”
    - a példában A nem küldi a (C,3,B) sornak megfelelő utat vissza B-nek, mert azt B-től kellett „tanulnia”
  - **split horizon with poison reverse**: negatív információt küld vissza
    - A pl. (C, $\infty$ ) utat küldi vissza B-nek
- Mindkét módszer csak két csomópontból álló ciklust kerül el



## Link State Protokoll

- Minden Link State router
  - tárolja a hálózat topológiáját
  - egy nem-elosztott legrövidebb utak algoritmust használ
- A routerek **Link State Packets (LSP)** által cserélnék ki információkat
- LSP tartalmazza
  - az LSP-t létrehozó r router IP címét
  - a költségét r minden direkt szomszédjához
  - sorozatszámot (SEQNO)
  - TTL (time to live) mezőt
- Megbízható elárasztás (Reliable Flooding)
  - minden csomópont aktuális LSP-jét tároljuk
  - továbbítjuk az LSP-ket minden szomszédos csomóponthoz
    - azon csomópont kivételével, amely az LSP-t felénk továbbította
  - A továbbításnál csökkentjük a TTL értékét
  - periodikusan létrehozunk egy új saját LSP-t
    - növekvő SEQNO-val