PB165 – Grafy a sítě

Směrování – hledání nejkratších cest

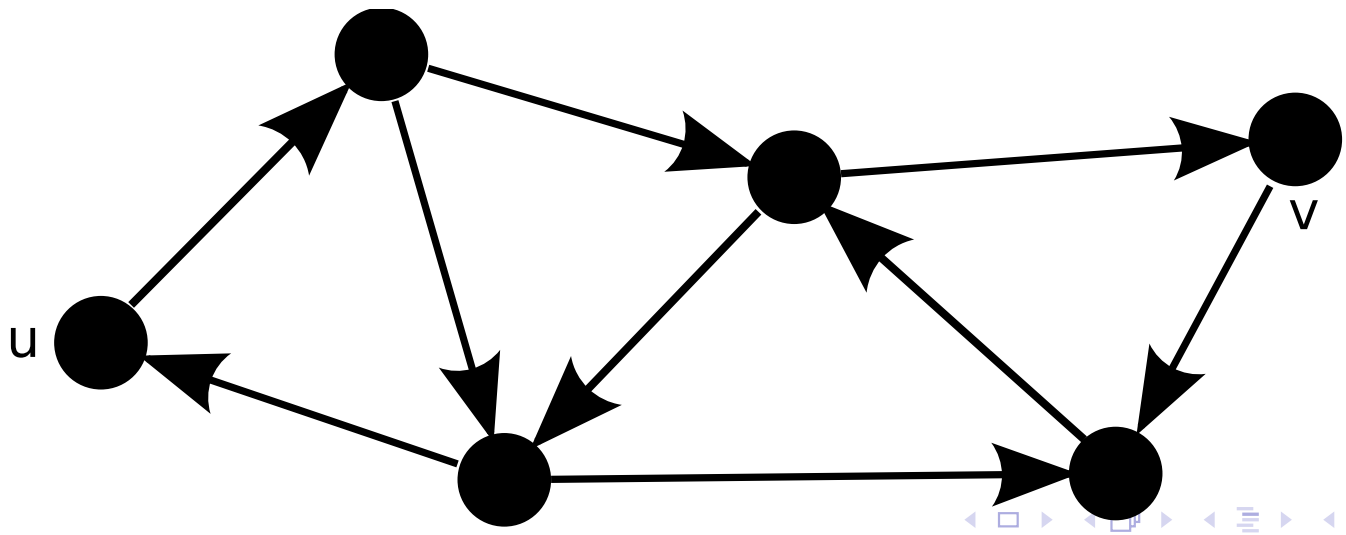
Obsah přednášky

Nejkratší cesty – vzdálenost v grafu

Pro připomenutí:

- Délka cesty v neohodnoceném grafu je rovna počtu hran na této cestě.
- Vzdálenost $\delta(u, v)$ vrcholů u, v v grafu je délka nejkratší cesty z u do v.
- Vzdálenost mezi dvěma vrcholy nemusí být v případě orientovaného grafu symetrická.

Obrázek: Nejkratší cety z u do v a naopak se liší.



Vzdálenost v ohodnoceném grafu

V reálných aplikacích hledání nejkratších cest jsou hrany grafu obvykle nějakým způsobem ohodnoceny - např. vzdálenosti mezi městy silniční sítě, latence síťových spojů.

Definice

Délka cesty v ohodnoceném grafu je rovna součtu ohodnocení hran na této cestě.

- Vzdálenost vrcholů je opět rovna délce nejkratší cesty.
- Aby měl pojem vzdálenosti význam, v grafu nemůže existovat cyklus záporné délky.

Nejkratší cesty a cykly

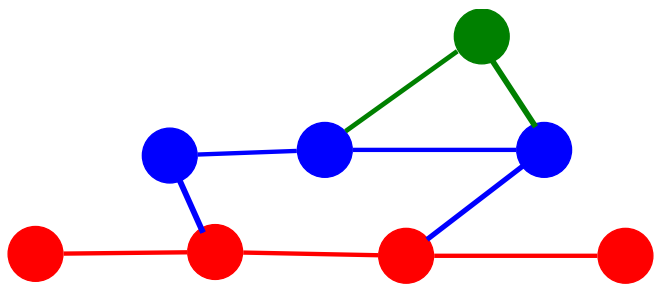
Věta

Pokud v grafu neexistuje cyklus záporné nebo nulové délky, je nejkratší sled v grafu (nejkratší) cestou.

Důkaz.

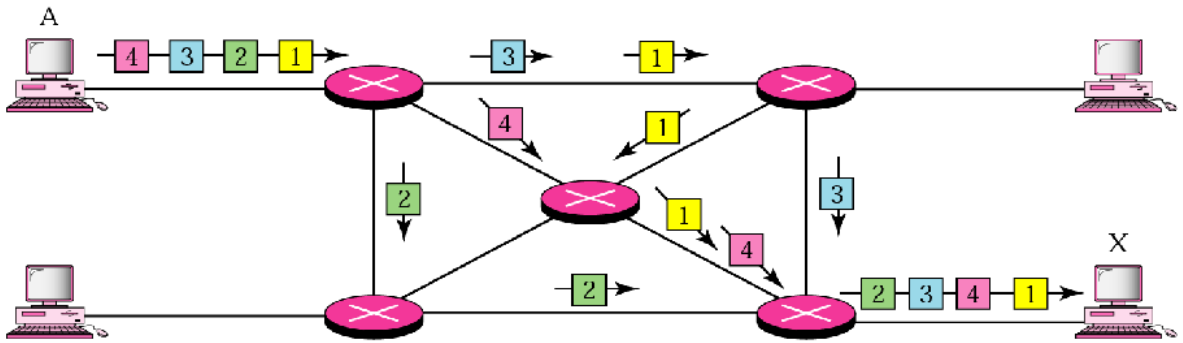
Nechť je součástí sledu s cyklus c. Tento cyklus má zřejmě kladnou délku. Potom existuje sled, který je „podsledem“ s a cyklus c je z něj „vystřižen“. Jelikož c má kladnou délku, je tento sled kratší než sled c obsahující. Takto lze pokračovat a sled zkracovat, dokud obsahuje nějaké cykly. Výsledný sled, který neobsahuje žádný cyklus, je cestou. □

Obrázek: Nejkratší sled je vyznačen červeně. Delší sledy vedou i po modrých a zelených hranách a tvoří cykly.



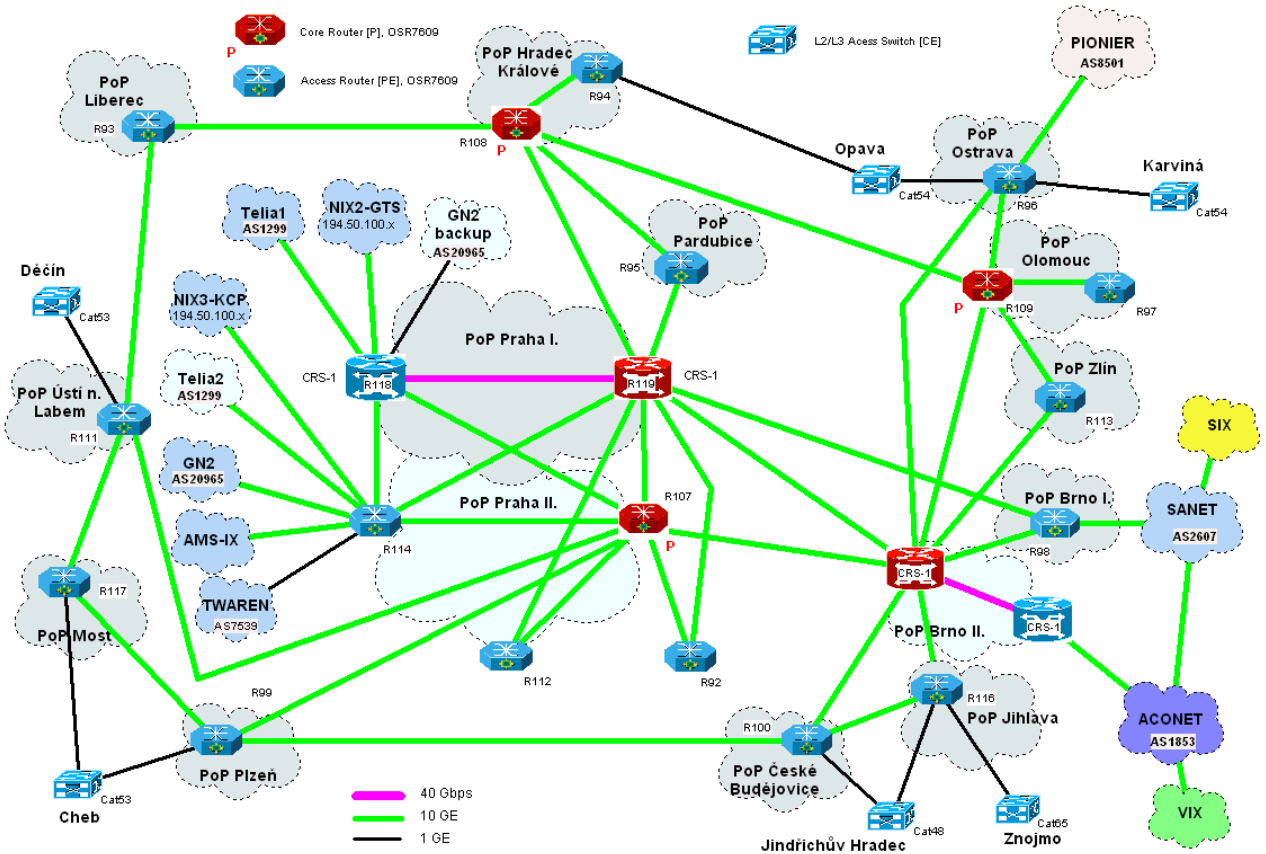
Směrování obecně

- Internet na L3 – datagramový přístup k přepínání paketů
  - data vyšších vrstev umísťována do datagramů
  - datagramy (fragmenty) putují sítí nezávisle



- **směrování (Routing)** = proces nalezení cesty mezi dvěma komunikujícími uzly
  - cesta musí splňovat určité omezující podmínky
  - ovlivňující faktory:
    - *statické:* topologie sítě
    - *dynamické:* zátěž sítě

Příklad reálné sítě



Obrázek: Logická topologie IP/MPLS vrstvy sítě CESNET2.

Směrování

- úkolem směrování je:
  - vyhledávat optimální směrovací trasy
    - kriteriem optimality je metrika
  - dopravit datový paket určenému adresátovi
- zpravidla se nezabývá celou cestou paketu
  - směrovač řeší jen jeden krok – komu paket předat jako dalšímu
    - někomu „blíže“ cíli
    - tzv. *hop-by-hop*
  - ten pak rozhoduje, co s paketem udělat dál

Směrování (Routing) vs. zasílání (Forwarding)

- *směrování*
  - společná činnost směrovačů (globální)
  - proces nalezení/vytváření a údržby směrovacích tabulek
- *zasílání*
  - lokální proces – každý směrovač samostatně
  - představuje proces průchodu paketů směrovačem
    - zaslání paketu na vybrané rozhraní směrovače (dle cílové adresy)
    - vyžaduje přístup ke směrovací tabulce

Cena komunikace

Určení ceny (ohodnocení) linky – *metrika*:

- všechny linky mají stejnou cenu (např. 1)
  - minimalizace ceny = minimalizace počtu skoků
  - nejjednodušší, nejčastěji využívané
- cena linky = převrácená hodnota kapacity ($1/prenosova\_kapacita$)
  - 10Mb linka má 100x vyšší cenu než 1Gb linka
- cena linky = zpoždění linky
  - 250ms satelitní spojení má 10x vyšší cenu než 25ms pozemní linka
- cena linky = využití linky
  - linka s 90% využitím má 10x vyšší cenu než linka s 9% využitím
  - může způsobit oscilace (nezbytné tlumení)
- cena linky = reálná cena (platba) za využití linky
  - staticky přiřazeno administrátorem
- atd.

Distribuované směrování – základní přístupy

Třídy distribuovaných směrovacích protokolů (dle charakteru směrovací informace):

- *Distance Vector (DV)* – Bellman-Fordův algoritmus
  - sousední směrovače si v pravidelných intervalech či při topologické změně (např. výpadek zařízení) vyměňují kompletní kopie svých směrovacích tabulek
  - na základe obsahu přijatých updatů si pak doplňují nové informace a inkrementují své *distance vektor číslo*
    - metrika udávající počet hopů k dané síti
  - čili „*všechny informace jen svým sousedům*“
- *Link State (LS)*
  - jednotlivé směrovače si zasílají pouze informace o stavu linek, na něž jsou bezprostředně připojeny
  - udržují si tak kompletní informace o topologii dané sítě – zařízení jsou si vědoma všech ostatních zařízení na síti
  - pak se počítá nejkratší cesta
  - čili „*informace o svých sousedech všem*“

Dijkstrův algoritmus

- „Klasický“ algoritmus hledání nejkratší cesty.
- Najde nejkratší cesty z jednoho vrcholu do všech ostatních.
  - Musí proto projít všechny vrcholy.
- Pracuje pro orientovaný i neorientovaný graf.
- Vyjma nezáporných cyklů vyžaduje nezáporné ohodnocení *všech* hran.
- Lineární paměťová složitost.
- Časová složitost záleží na použité datové struktuře.

Dijkstrův algoritmus – popis

- Počáteční vrchol označíme s.
- Pro každý vrchol v grafu je udržována hodnota `d[v]` – délka nejkratší nalezené cesty z s do v.
  - Na počátku `d[s]` $= 0$ pro počáteční vrchol a `d[v]` $= \infty$ pro ostatní vrcholy.
  - Po skončení výpočtu obsahuje `d[v]` délku nejkratší cesty v grafu, pokud taková existuje, nebo ∞ v opačném případě.
- Dále v proměnné `p[v]` ukládáme předchůdce vrcholu v na doposud nalezené nejkratší cestě z u.
  - Před výpočtem nastavíme hodnotu `p[v]` jako nedefinovanou pro všechny vrcholy.
  - Po skončení výpočtu je nejkratší cesta posloupnost vrcholů `s, p[...p[v]...], ... p[p[v]], p[v], v`.

Dijkstrův algoritmus – popis

- Všechny vrcholy jsou rozděleny do dvou vzájemně disjunktních množin:
  - S obsahuje všechny vrcholy, pro něž je v `d[v]` uložena definitivní nejkratší cesta z s do v v grafu.
  - Q obsahuje všechny ostatní vrcholy.
- Vrcholy množiny Q jsou ukládány v prioritní frontě.
  - Nejvyšší prioritu má vrchol u s nejnižší hodnotou `d[u]` – nelze do něj již nalézt kratší cestu než která je aktuální.
- V každé iteraci jsou provedeny následující kroky:
  - Odstraň vrchol u z počátku fronty.
  - Přesuň vrchol u z množiny Q do S.
  - Relaxuj všechny hrany (u, v):
    - Pokud `d[v]` $>$ `d[u]` $+w(u, v)$, uprav `d[v]`.
  - $w(u, v)$ značíme ohodnocení (weight) hrany (u, v).

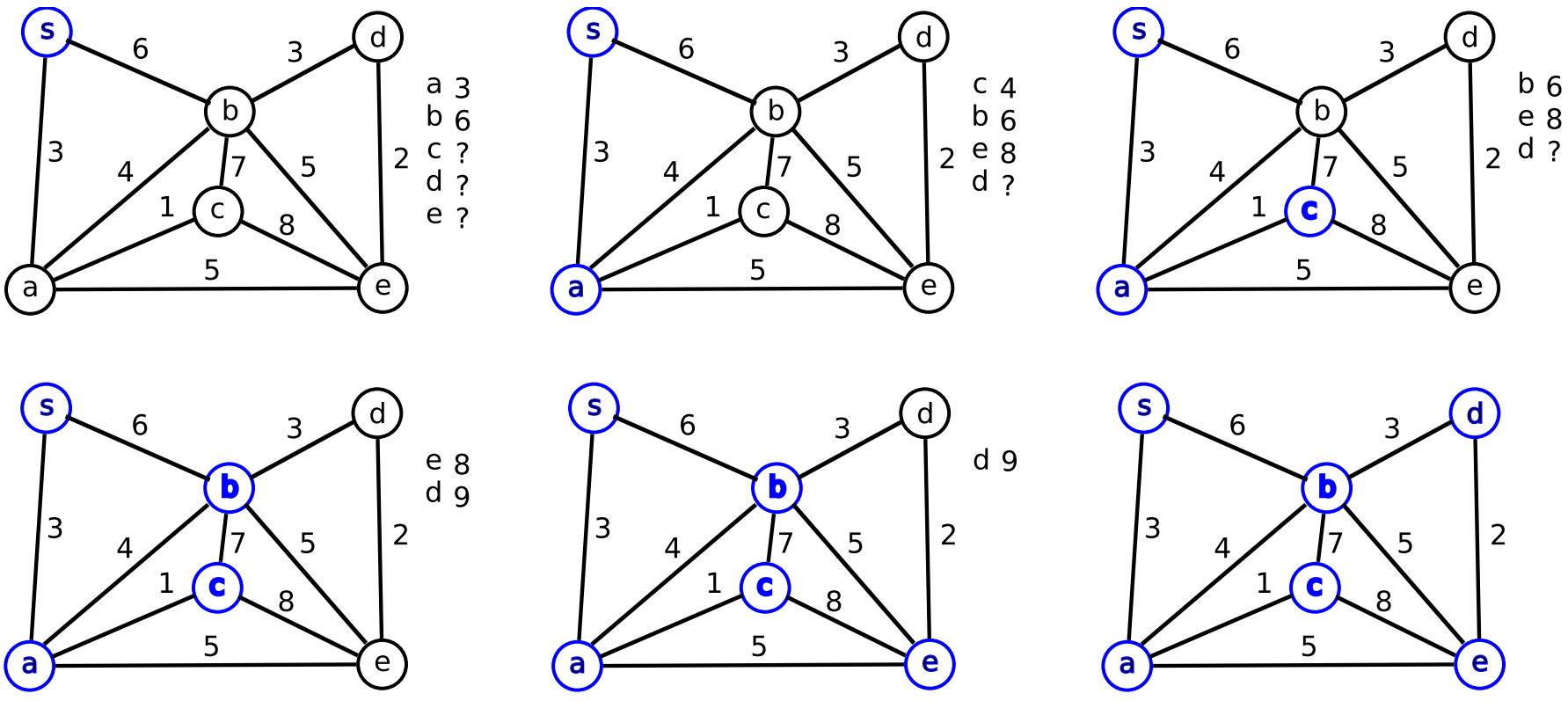
Dijkstrův algoritmus – pseudokód

Vycházíme z počátečního vrcholu s, nek. značíme ∞.

```
Vlož všechny vrcholy do Q.
d[s] = 0; p[s] = nedef;
Pro všechny vrcholy v grafu vyjma počátečního:
| d[u] = nek.
| p[u] = nedef.
Dokud není Q prázdná:
| Odstraň z Q vrchol u s nejvyšší prioritou
| Pro všechny hrany (u, v) proveď:
| | Pokud d[v] > d[u] + w(u,v)
| | | d[v] = d[u] + w(u,v)
| | | p[v] = u
```

Dijkstrův algoritmus – příklad

Obrázek: Vrcholy množiny Q jsou vyznačeny modře. Napravo od grafu je znázorněn stav prioritní fronty.



Dijkstrův algoritmus – časová složitost

Označme $n = |V|, m = |E|$.

- Inicializace je provedena v lineárním čase vzhledem k počtu vrcholů.
- Každou hranou prochází algoritmus vždy právě jednou nebo dvakrát (v případě neorientovaného grafu).
- Hlavní cyklus je proveden vždy $n$krát.
- Je tudíž provedeno vždy právě n výběrů z prioritní fronty.
- Složitost výběru z fronty záleží na její implementaci:
  - **Pole, seznam vrcholů** – výběr lze provést v lineárním čase, složitost celého algoritmu je tedy $\mathcal{O}(n^2) + m$.
  - **Binární halda** – výběr je proveden v čase $\mathcal{O}(log(n))$. Při každé relaxaci hrany může dojít k aktualizaci haldy ($\mathcal{O}(log(n))$, celková složitost je tak rovna $\mathcal{O}((n + m)log(n))$.
  - **Fibonacciho halda** – složitost výběru stejná jako v případě binární haldy, složitost úpravy haldy při relaxaci je ovšem konstantní – výsledná složtiost algoritmu $\mathcal{O}(m + nlog(n))$. `http://en.wikipedia.org/wiki/Fibonacci_heap`

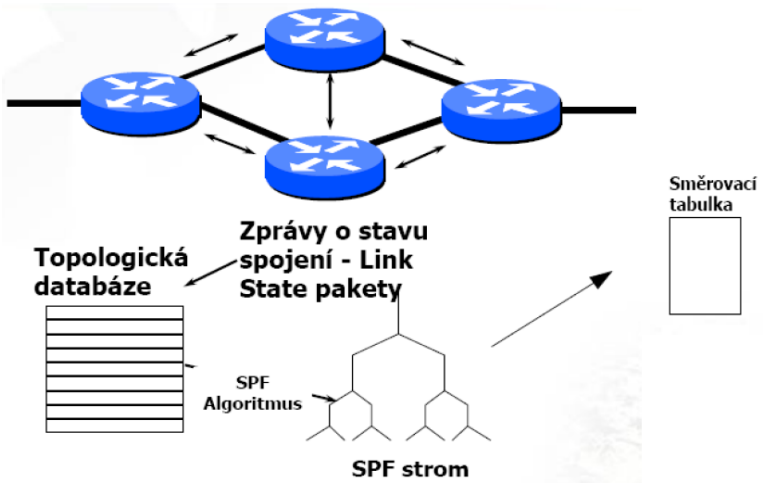
Dijkstrův algoritmus – využití

Dijkstrův algoritmus je používán v link-state směrovacích protokolech. Ty pracují na principu zasílání sousedních „vrcholů“ aktivními síťovými prvky, které se směrování zůčastní.

- Každý aktivní prvek periodicky rozesílá seznam sousedních vrcholů.
- Ten je propagován skrze síť ke všem aktivním prvkům.
- Každý aktivní prvek nezávisle na ostatnívh vypočítá strom nejkratších cest do všech ostatních aktivních prvků.
- Riziko vzniků smyček v routovacích tabulkách.

Nejpoužívanější link-state protokoly jsou OSPF a IS-IS. Oba používají Dijkstrův algoritmus.

Link State I.



- směrovače si zasílají pouze informaci o stavu linek, na něž jsou bezprostředně připojeny
- získají tím kompletní mapu sítě
  - pak si počítají nejkratší cesty (např. s využitím Dijkstrova algoritmu)
    - při každé změně stavu linek
- směrovače testují pouze dosažitelnost svých bezprostředních sousedů
- výhoda: zaručená a rychlá konvergence, vhodné i pro rozsáhlé sítě
- nevýhoda: složitější algoritmus $\Rightarrow$ větší nároky na CPU a paměť směrovače

Link State II.
Algoritmus

- Předpoklad:
  - každý směrovač zná pouze cestu a cenu ke svým sousedům
- Cíl:
  - v každém směrovači směrovací tabulka pro každý cíl
- Idea:
  - šíří se topologie, cesty si počítají směrovače samy
    - fáze 1: šíření topologie (broadcast)
    - fáze 2: výpočet nejkratší cesty – (Dijkstra)
  - směrovače si udržují databázi stavů linek a periodicky posílají LS pakety svým sousedům
    - obsah LS paketu: identifikátor uzlu, cena spojů k sousedům, pořadové číslo, doba platnosti
    - každý směrovač přeposílá LS pakety dále (kromě toho, od nějž informaci dostal)

Link State III. – protokol OSPF

- *Open Shortest Path First*
- nejpoužívanější LS protokol současnosti
- metrika: *cena (cost)*
  - číslo (v rozsahu 1 až 65535) přiřazené ke každému rozhraní směrovače
  - čím menší číslo, tím má cesta lepší metriku (bude tedy preferována)
  - standardně je ke každému rozhraní přiřazena cena automaticky odvozená z šířky pásma daného rozhraní
    - $cost = 100000000/bandwidth$ (bw v bps)
    - možno ručně měnit
- rozšíření:
  - autentizace zpráv
  - směrovací oblasti – další úroveň hierarchie
  - load-balancing – více cest se stejnou cenou

Bellman-Ford algoritmus

- Stejně jako Dijkstrův algoritmus vypočítá vzdálenosti všech vrcholů grafu z jednoho zdroje.
- Základní strukturou podobný Dijkstrovu algoritmu.
- Graf smí obsahovat i záporně ohodnocené hrany.
- Cykly s celkovým záporným ohodnocením jsou algoritmem detekovány.
- Namísto výběru hrany k relaxaci v každé iteraci relaxuje všechny hrany.
- Vyšší časová složitost než Dijkstrův algoritmus – $\mathcal{O}(mn)$.

Bellman-Ford – pseudokód

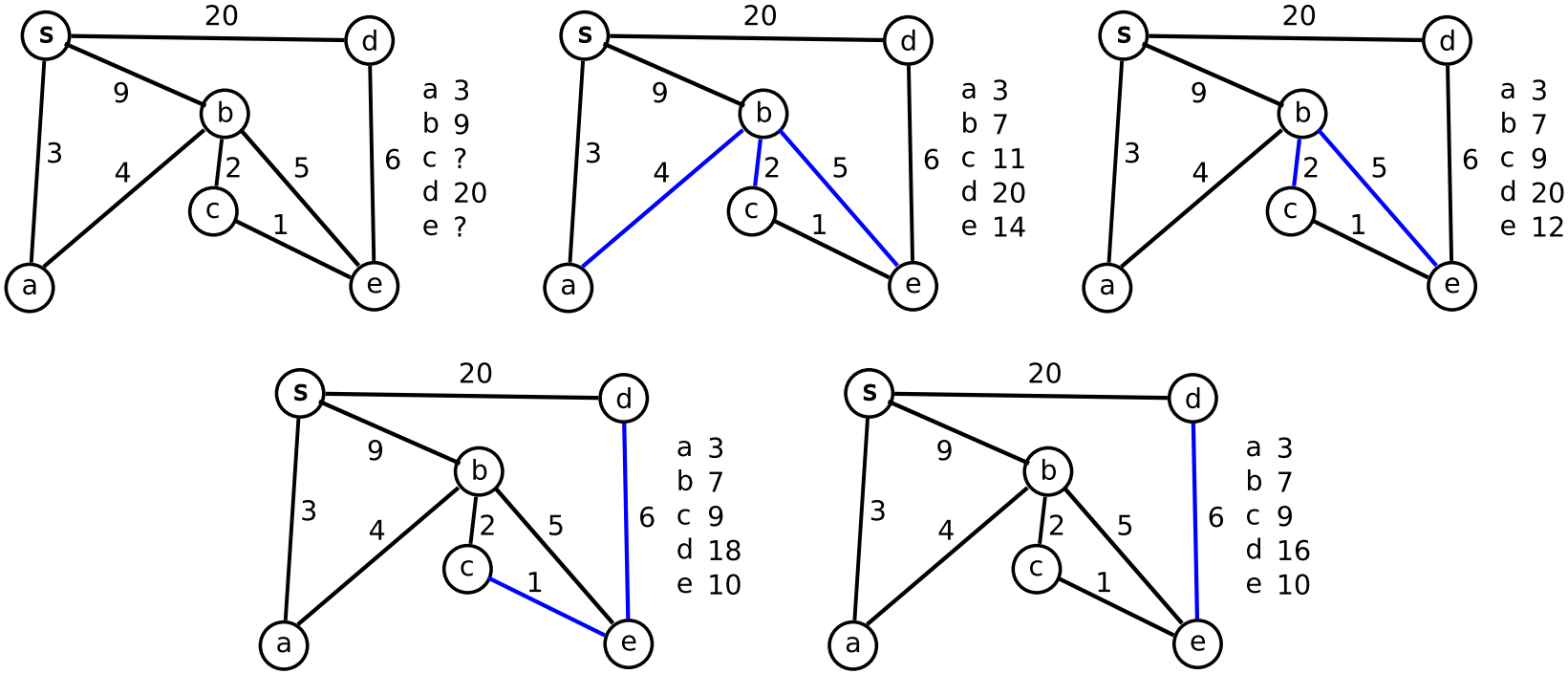
Proměnné d[v] a p[v] mají stejný význam jako u Dijkstrova algoritmu. Počet vrcholů grafu označme n.

```
d[s] = 0; p[s] = undef;
Pro všechny vrcholy v grafu vyjma počátečního:
| d[u] = nek.
| p[u] = nedef.
Opakuj n-krát:
| Pro všechny hrany (u,v):
| | Pokud d[v] > d[u] + w(u,v)
| | | d[v] = d[u] + w(u,v)
| | | p[v] = u

Pro všechny hrany (u,v) opakuj:
| Pokud d[v] > d[u] + w(u,v):
  | | Chyba: graf obsahuje cyklus neg. váhy
```

Bellman-Ford – příklad

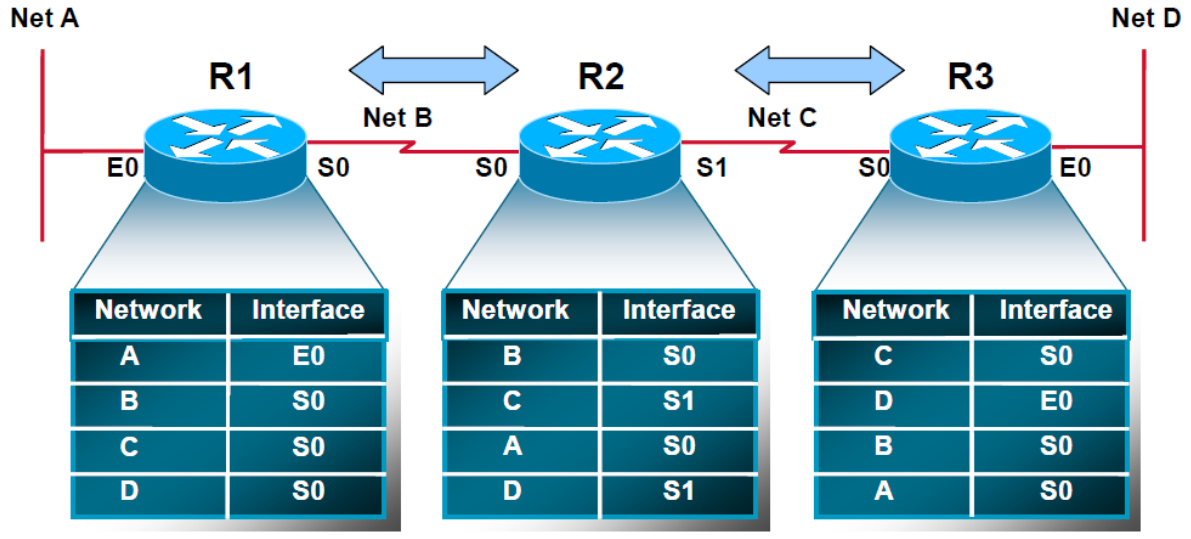
Obrázek: Relaxované hrany v každém kroku jsou vyznačeny modře.
Napravo od grafu je vypsána délka nejkratších cest do vrcholů.
V poslední (nezobrazené) iteraci již nedojde k žádným změnám.



Bellman-Ford – aplikace

- Bellman-Ford algoritmus je používán v druhé třídě směrovacích protokolů – distance-vector.
- Namísto struktury celého grafu jsou mezi aktivními prvky přenášeny jim známé vzdálenosti do ostatních aktivních prvků.
- Každý aktivní prvek periodicky rozesílá tyto sobě známé vzdálenosti k ostatním.
- Obdrží-li aktivní prvek tabulku vzdáleností, provede relaxaci hran, případně aktualizuje svoji směrovací tabulku a rozešle svým sousedům.
- Distance-vector protokoly mají nižší výpočetní složitost než link-state protokoly.
- Nejznámějšími distance-vector protokoly jsou RIP, BGP, EGP.

Distance Vector I.



- směrovač si udržuje všechny známé routy v tabulce ve formě uspořádaných trojic (N, G, D), kde:
  - N . . . cílová síť
  - G . . . adresa následujícího směrovače
  - D . . . vzdálenost do cílové síťě (metrika)
- tabulky se upravují tak, aby se směrovalo nejkratší cestou
- problémy: pomalá konvergence, příliš mnoho režijních dat

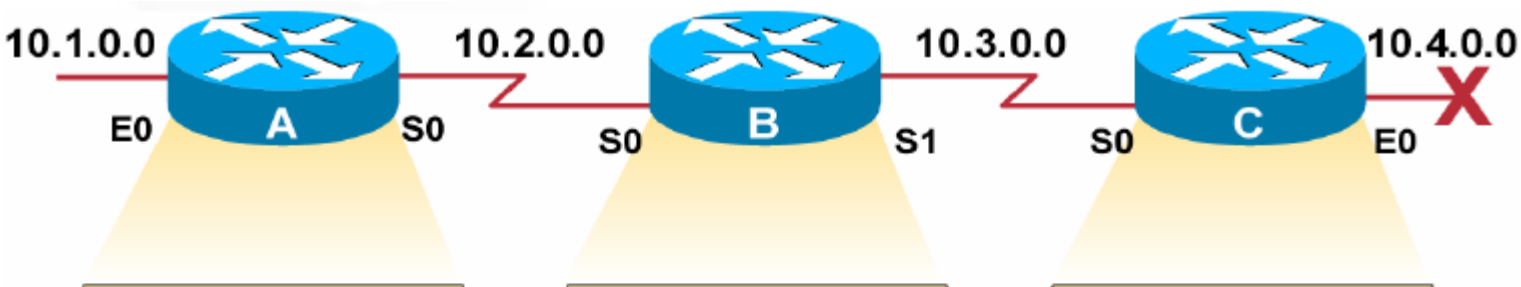
Distance Vector II.

Algoritmus

- Předpoklad:
  - každý směrovač zná pouze cestu a cenu ke svým sousedům
- Cíl:
  - v každém směrovači směrovací tabulka pro každý cíl
- Idea:
  - řekni sousedům svou představu směrovací tabulky
- Inicializace:
  - sousedé: známá cena
    - Distance Vector $= < cil, cena >$
  - ostatní: nekonečno
    - resp. hodnota definovaná jako nekonečno (pro RIP např. 16)
- Aktualizace:
  - pokud je cesta v získaném DV zvětšená o cenu cesty k danému sousedovi lepší než stávající uložená, aktualizuj tabulku

Distance Vector III.

Ilustrace problému pomalé konvergence



Routing Table		
10.1.0.0	E0	0
10.2.0.0	S0	0
10.3.0.0	S0	1
10.4.0.0	S0	2

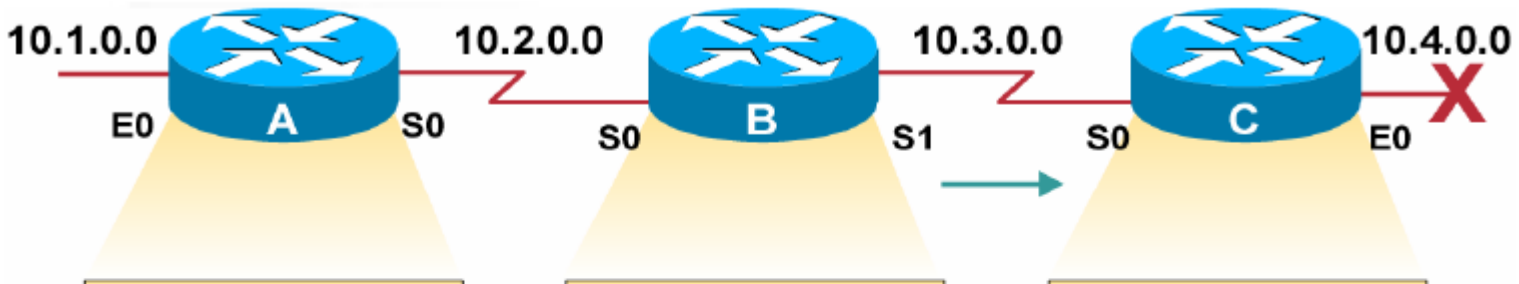
Routing Table		
10.2.0.0	S0	0
10.3.0.0	S1	0
10.4.0.0	S1	1
10.1.0.0	S0	1

Routing Table		
10.3.0.0	S0	0
10.4.0.0	E0	Down
10.2.0.0	S0	1
10.1.0.0	S0	2

- pomalá konvergence zapříčiní vznik nesprávných údajů ve směrovacích tabulkách

Distance Vector III.

Ilustrace problému pomalé konvergence



Routing Table		
10.1.0.0	E0	0
10.2.0.0	S0	0
10.3.0.0	S0	1
10.4.0.0	S0	2

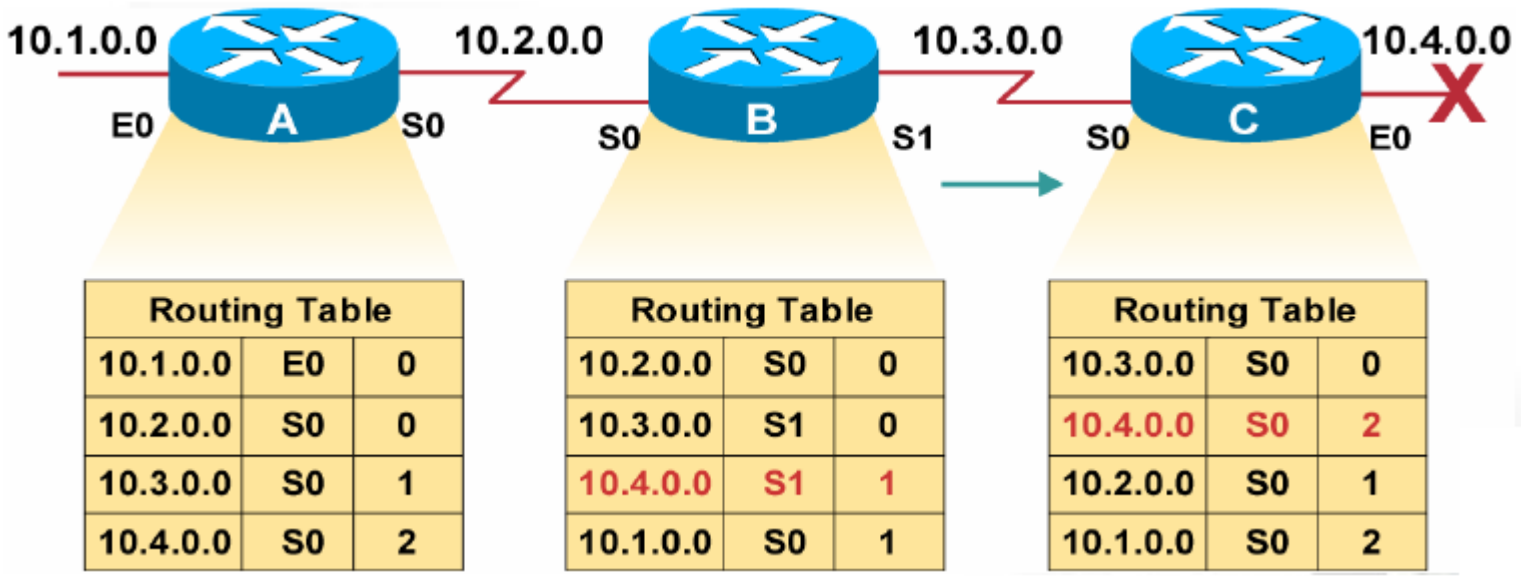
Routing Table		
10.2.0.0	S0	0
10.3.0.0	S1	0
10.4.0.0	S1	1
10.1.0.0	S0	1

Routing Table		
10.3.0.0	S0	0
10.4.0.0	S0	2
10.2.0.0	S0	1
10.1.0.0	S0	2

- směrovač C usoudí, že nejlepší cesta do sítě 10.4.0.0 je přes směrovač B

Distance Vector III.

Ilustrace problému pomalé konvergence



Routing Table		
10.1.0.0	E0	0
10.2.0.0	S0	0
10.3.0.0	S0	1
10.4.0.0	S0	2

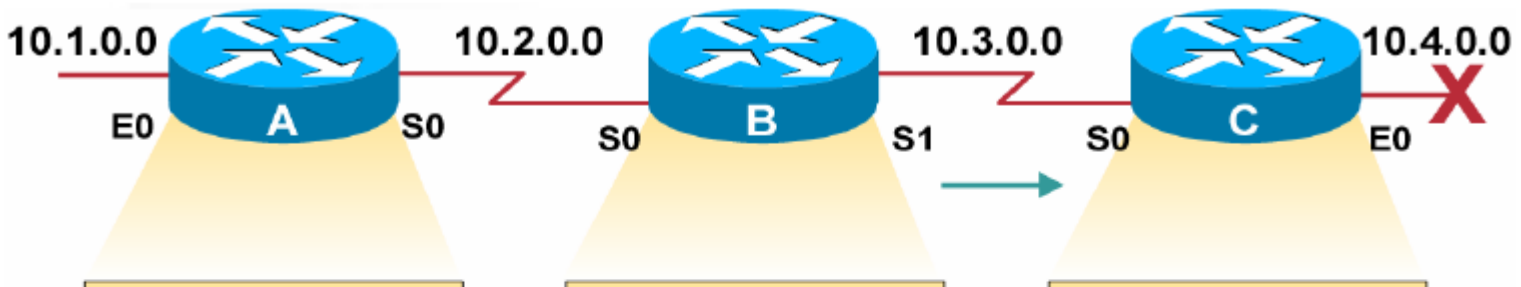
Routing Table		
10.2.0.0	S0	0
10.3.0.0	S1	0
10.4.0.0	S1	1
10.1.0.0	S0	1

Routing Table		
10.3.0.0	S0	0
10.4.0.0	S0	2
10.2.0.0	S0	1
10.1.0.0	S0	2

- směrovač A opraví svojí směrovací tabulku – chybně

Distance Vector III.

Ilustrace problému pomalé konvergence



Routing Table		
10.1.0.0	E0	0
10.2.0.0	S0	0
10.3.0.0	S0	1
10.4.0.0	S0	2

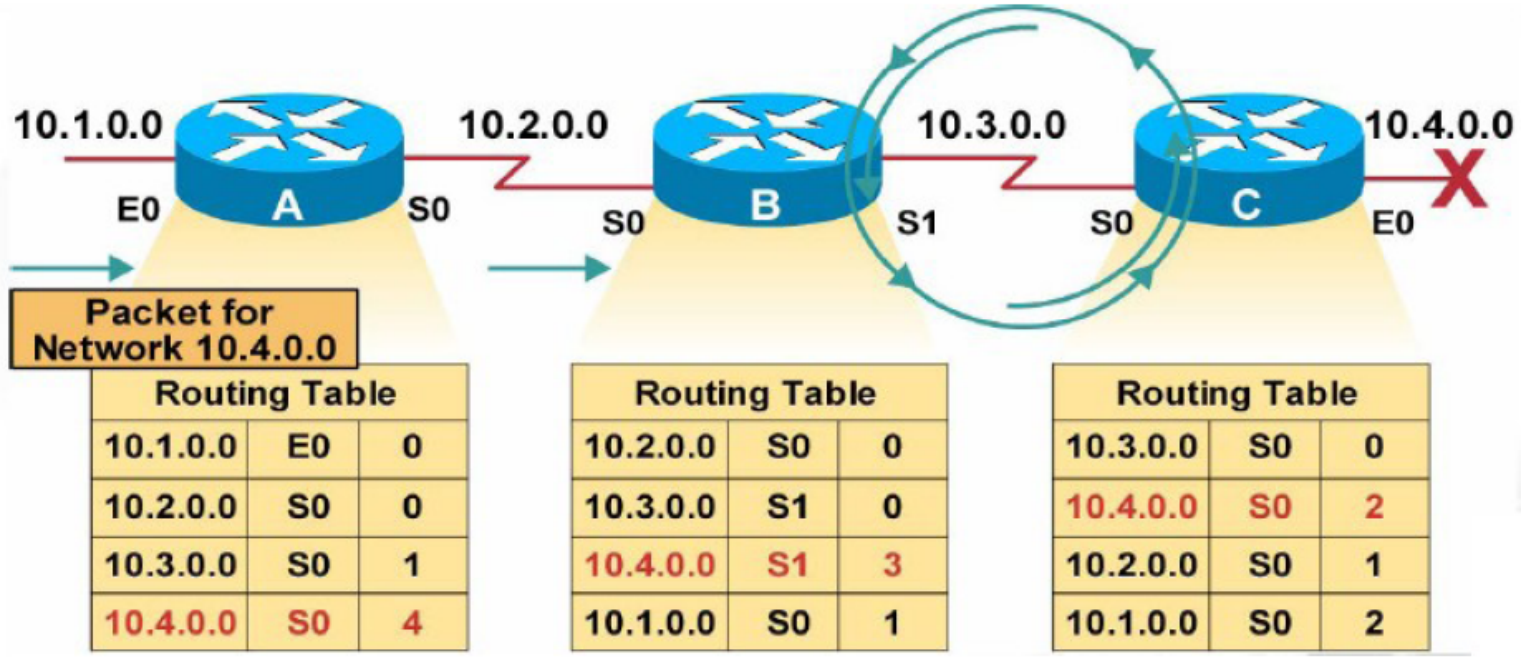
Routing Table		
10.2.0.0	S0	0
10.3.0.0	S1	0
10.4.0.0	S1	1
10.1.0.0	S0	1

Routing Table		
10.3.0.0	S0	0
10.4.0.0	S0	2
10.2.0.0	S0	1
10.1.0.0	S0	2

- metrika pro síť 10.4.0.0 roste do nekonečna (v rámci RIP do 16)

Distance Vector III.

Ilustrace problému pomalé konvergence



Routing Table		
10.1.0.0	E0	0
10.2.0.0	S0	0
10.3.0.0	S0	1
10.4.0.0	S0	4

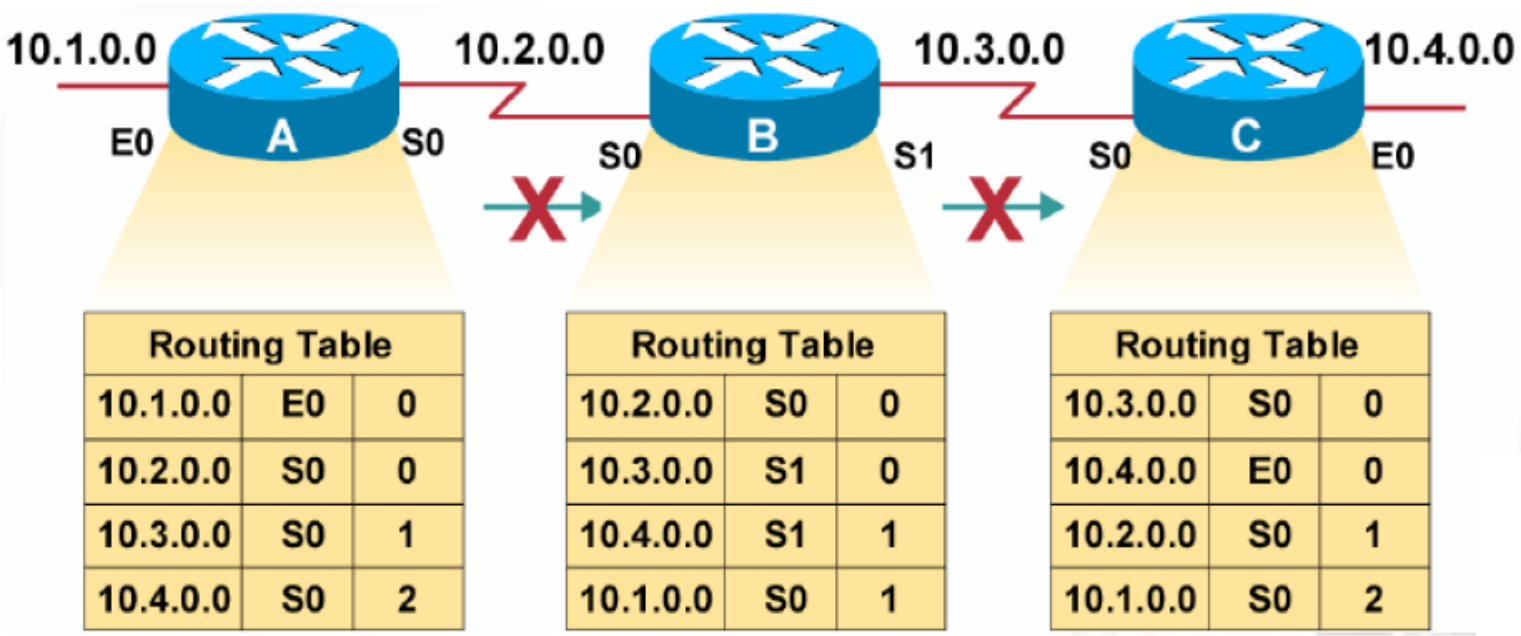
Routing Table		
10.2.0.0	S0	0
10.3.0.0	S1	0
10.4.0.0	S1	3
10.1.0.0	S0	1

Routing Table		
10.3.0.0	S0	0
10.4.0.0	S0	2
10.2.0.0	S0	1
10.1.0.0	S0	2

- Důsledek: vznik směrovací smyčky
  - paket pro síť 10.4.0.0 skáče mezi routery B a C

Distance Vector III.

Ilustrace problému pomalé konvergence



Routing Table		
10.1.0.0	E0	0
10.2.0.0	S0	0
10.3.0.0	S0	1
10.4.0.0	S0	2

Routing Table		
10.2.0.0	S0	0
10.3.0.0	S1	0
10.4.0.0	S1	1
10.1.0.0	S0	1

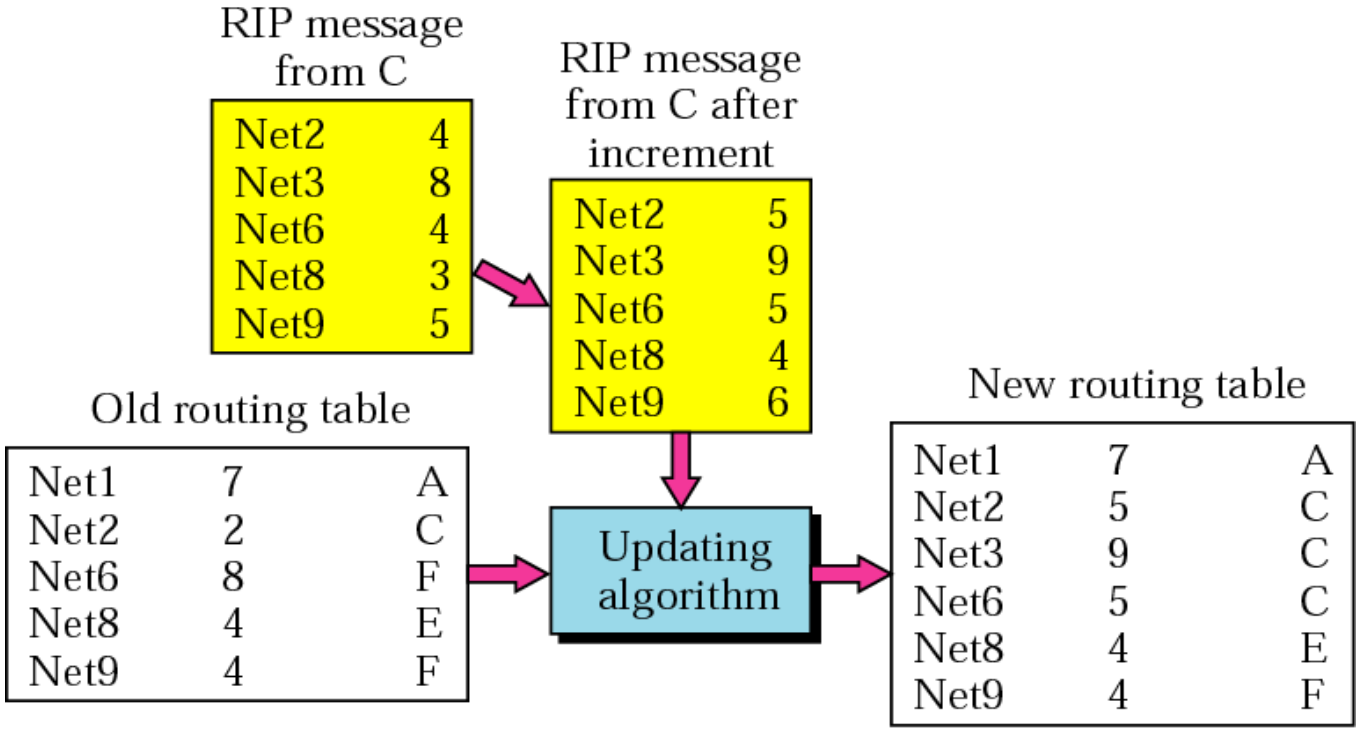
Routing Table		
10.3.0.0	S0	0
10.4.0.0	E0	0
10.2.0.0	S0	1
10.1.0.0	S0	2

- Řešení: *dělení horizontu*
  - směrovač nesděluje cestu zpět uzlu, od kterého se o ní dozvěděl
  - problém zůstává ve složitějších topologiích (navržena řada rozšíření)

Distance Vector IV. – protokol RIP I.

- hlavní představitel DV směrování
  - RIPv1 (RFC 1058)
  - RIPv2 (RFC 1723) – přidává např. autentizaci směrovacích informací
- sítě identifikovány s využitím mechanismu CIDR
- jako metrika se využívá počet hopů
  - přenos paketu mezi 2 sousedními směrovači má délku 1
  - nekonečno = 16
    - $\Rightarrow$ nelze použít pro sítě s minimálním počtem hopů mezi libovolnými dvěma směrovači > 15
- směrovače zasílají informaci každých 30 sekund
  - triggered update při změně stavu hrany
  - časový limit 180s (detekce chyb spojení)
- použití:
  - vhodné pro malé sítě a stabilní linky
  - není příliš vhodný pro redundantní sítě

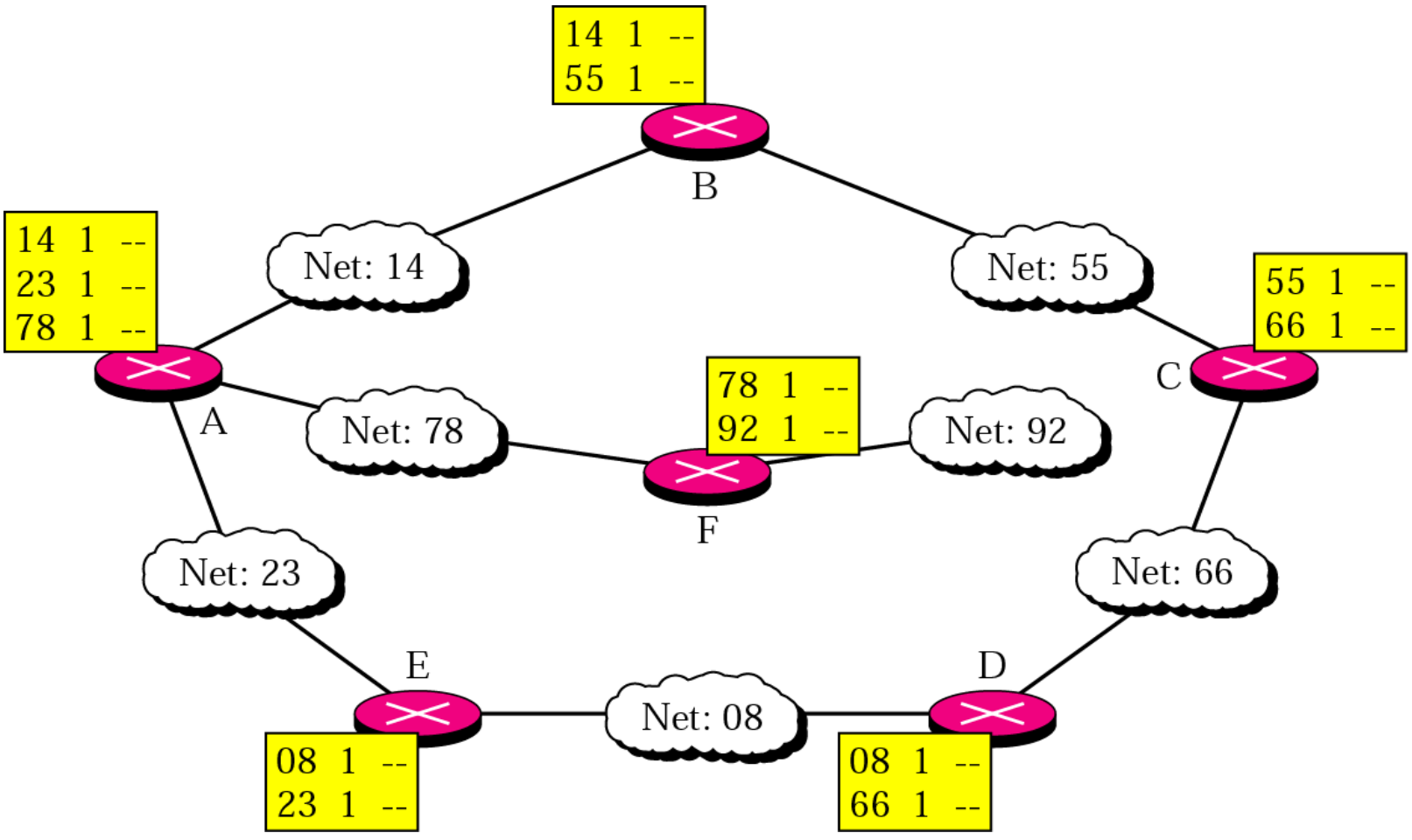
Distance Vector IV. – protokol RIP II.



Net1: No news, do not change
Net2: Same next hop, replace
Net3: A new router, add
Net6: Different next hop, new hop count smaller, replace
Net8: Different next hop, new hop count the same, do not change
Net9: Different next hop, new hop count larger, do not change

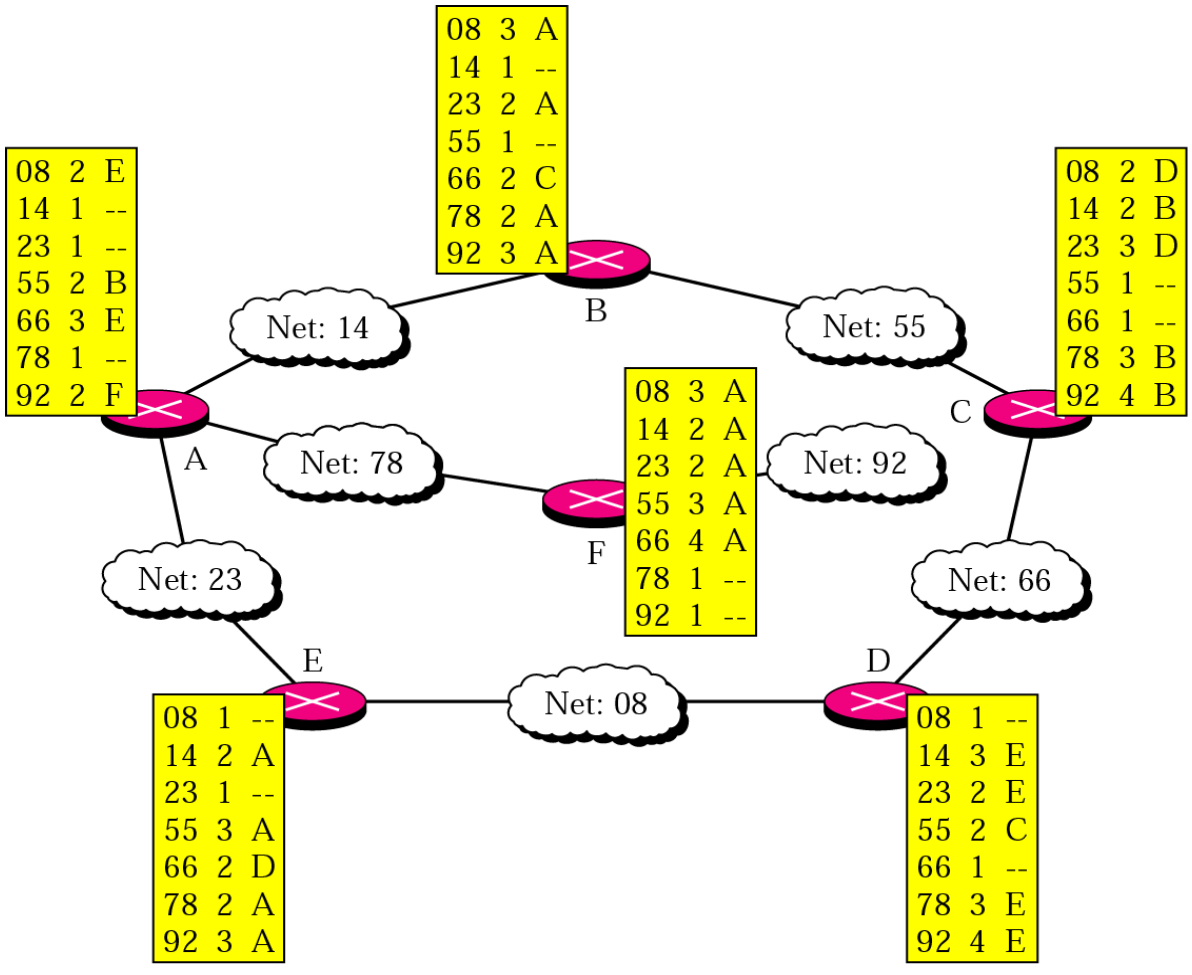
Obrázek: RIP – příklad aktualizace tabulek.

Distance Vector – protokol RIP III.



Obrázek: RIP – příklad: iniciální stav tabulek.

Distance Vector IV. – protokol RIP IV.



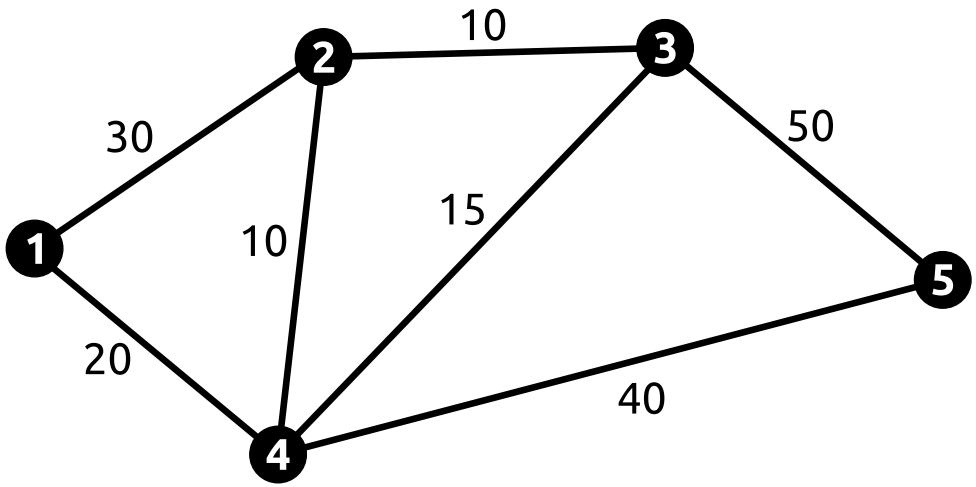
Obrázek: RIP – příklad: finální stav tabulek.

Kvalita služby

- základní parametry síťových toků:
  - *spolehlivost (reliability)* – požadavek plné spolehlivosti vs. tolerance definované ztrátovosti
  - *zpoždění (latency, delay)*
  - *rozptyl zpoždění (jitter)*
  - *přenosová kapacita (bandwidth)*

Směrování - hledání nejširší cesty

- Příklad grafu s vyznačenými šířkami pásma (ohodnocení hran)



- cesta šířka(1-2-3-5) = 10
- cesta šířka(1-4-2-3-5) = 10
- cesta šířka(1-4-3-5) = 15
- cesta šířka(1-4-5) = 20
- cesta šířka cesty je minimum šířek hran
- optimalizační funkce cesty je maximální šířka cest

Směrování - hledání nejširší cesty

- úprava optimalizační funkce - zavedení záporné šířky pásma
- optimalizační funkce cesty je minimalní šířka cest
- úprava známých algoritmů na hledání minimální cesty
  - Dijkstra
  - Bellman Ford

A* algoritmus

- Upravený Dijkstrův algoritmus.
- Používá se zejména k nalezení cesty do jednoho vrcholu – např. navigace.
- Krom délky nejkratší cesty do každého vrcholu bere při vyhledávání v úvahu i heuristický odhad jeho vzdálenosti od cíle.
- Každé cestě je přiřazen heuristický odhad délky jako součet ohodnocení jejích hran a heuristické ohodnocení koncového vrcholu.
- Do prioritní fronty nejsou ukládány vrcholy grafu, ale cesty.
  - Nejvyšší prioritu má cesta s nejnižším heuristickým ohodnocením.

A* algoritmus

- Pro každý vrchol je uloženo, je-li již „uzavřen“ či nikoliv, tzn., byl-li již navštíven, prozkoumán odebráním z fronty některé cesty v tomto vrcholu končící.

Dijkstrův algoritmus lze použít také k nalezení cety do jednoho vrcholu grafu. Obvykle ale projde mnoho neperspektivních vrcholů – např. při vyhledávání trasy v mapě jde i opačným směrem stejně daleko, jak správným.
A* tyto vrchly eliminuje pomocí heuristiky, je-li vhodně zvolena.

- Časová složitost záleží na kvalitě zvolené heuristiky – nejhůře může být exponenciální, nejlépe polynomiální, a to vůči délce optimální cesty.
- Paměťová složitost může být v nejhorším případě také exponenciální – existuje několik zlepšujících variant algoritmu.

Více informací v přednášce doc. Hliněného:
`http://video.fi.muni.cz/public/ITI/ITI2.avi`

A* algoritmus – pseudokód

Označme počáteční vrchol s, koncový c.

```
Označ všechny vrcholy jako neuzavřené.
Inicializuj prioritní frontu Q vrcholem (cestou) s.
Dokud Q není prázdná:
| Odstraň cestu p z Q.
| Nechť x je koncový vrchol p.
| Pokud x je uzavřený:
| | pokračuj další iterací.
| Pokud x = c:
| | Vrať cestu p jako výsledek.
| Uzavři x.
| Pro všechny hrany (x, y):
| | Vlož do fronty cestu p + (x, y)
Vrať zprávu o neexistenci cesty.
```

Floyd-Warshallův algoritmus

- Vypočítává nejkratší vzdálenost mezi všemi dvojicemi vrcholů v grafu.
- Graf může obsahovat záporně ohodnocené hrany, cykly s celkovým zápornym ohodnocením vedou k chybnému řešení.
- Mezi každými dvěma dvojicemi vrcholů postupně vylepšuje nejkratší známou vzdálenost.
- V každém kroku algoritmu je definována množina vrcholů, kterými je možno nejkratší cesty vést.
- Každou iterací je do této množiny přidán jeden vrchol.
- V každé z n iterací jsou aktualizovány cesty mezi všemi n^2 dvojicemi vrcholů. Časová složitost algoritmu je tedy $\mathcal{O}(n^3)$.
- Paměťová složitost algoritmu je $\mathcal{O}(n^2)$.

Floyd-Warshallův algoritmus – bližší popis

- Nechť jsou vrcholy grafu očíslovány $1 \ldots n$.
- Nejprve algoritmus uvažuje pouze hrany grafu. Následně prohledává cesty procházející pouze vrcholem 1. Poté cesty procházející pouze vrcholy 1, 2, atd.
- Mezi každými dvěma vrcholy u, v je v $k+1$. iteraci algoritmu známa cesta využívající vrcholů $1 \ldots k$.
- Pro nejkratší cestu mezi těmito vrcholy využívající vrcholů $1 \ldots k+1$ jsou dvě možnosti:
  - Vede opět pouze po vrcholech $1 \ldots k$.
  - Vede po vrcholech $1 \ldots k$ z u do vrcholu $k+1$ a z něj poté do$v$.
- Na konci výpočtu jsou známy nejkratší cety využívající všech vrcholůů grafu.

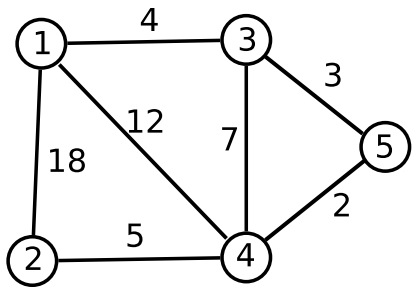
Floyd-Warshallův algoritmus – pseudokód

- V matici `d` na pozici `d[i][j]` je uložena vypočtená vzdálenost vrcholů i, j.
- Vstupem je graf v podobě matice sousednosti, kde jednotlivé prvky značí ohodnocení hrany nebo ∞

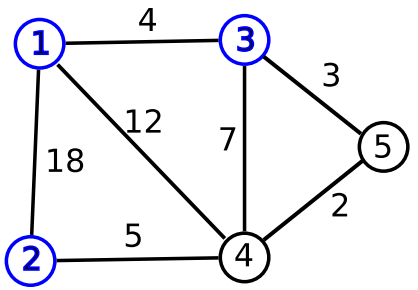
```
Pro všechna k od 1 do n:
| Pro všechna i od 1 do n:
| | Pro všechna j od 1 do n:
| | | d[i][j] = minimum(d[i][j], d[i][k] + d[k][j])
```

Floyd-Warshallův algoritmus – příklad

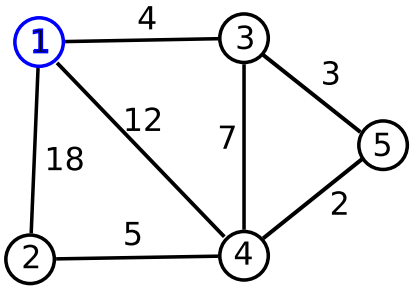
Obrázek: Vrcholy, kterými mohou vést cesty, jsou vyznačeny. Matice udává nejkratší nalezené vzdálenosti mezi dvojicemi vrcholů.



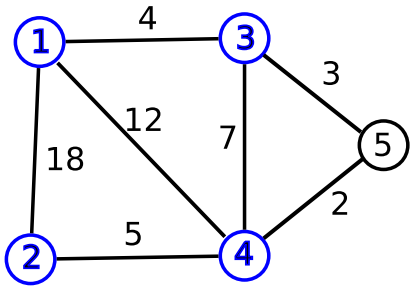
	1	2	3	4	5
1	0	18	4	12	?
2	18	0	?	5	?
3	4	?	0	7	3
4	12	5	7	0	2
5	?	?	3	2	0



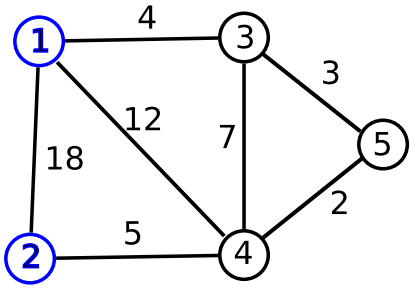
	1	2	3	4	5
1	0	18	4	11	7
2	18	0	22	5	25
3	4	22	0	7	3
4	11	5	7	0	2
5	7	25	3	2	0



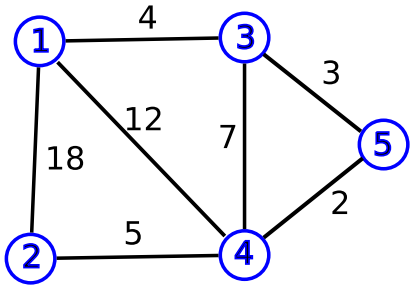
	1	2	3	4	5
1	0	18	4	12	?
2	18	0	22	5	?
3	4	22	0	7	3
4	12	5	7	0	2
5	?	?	3	2	0



	1	2	3	4	5
1	0	16	4	11	7
2	16	0	12	5	7
3	4	12	0	7	3
4	11	5	7	0	2
5	7	7	3	2	0



	1	2	3	4	5
1	0	18	4	12	?
2	18	0	22	5	?
3	4	22	0	7	3
4	12	5	7	0	2
5	?	?	3	2	0



	1	2	3	4	5
1	0	14	4	9	7
2	14	0	10	5	7
3	4	10	0	5	3
4	9	5	5	0	2
5	7	7	3	2	0

Distribuovaný Floyd-Warshall

Výhodou Floyd-Warshallova algoritmu je jeho snadná aplikace v distribuovaném prostředí – mezi autonomními jednotkami, které si mohou informace předávat jen pomocí zasílání zpráv po síti.

- Každý vrchol grafu vypočítává nejkratší cesty do všech ostatních vrcholů grafu.
- Na začátku zná každý vrchol jen cestu do svých sousedů.
- Stejně jako v sekvenční variantě algoritmu, každá iterace algoritmu přidává jeden vrchol, kterým mohou procházet hledané nejkratší cesty.
- Přidaný vrchol v každé iteraci rozešle svoji tabulku vzdáleností ostatním vrcholům grafu.
- Ostatní vrcholy pomocí své a přijaté tabulky aktualizují nejkratší cesty do všech vrcholů.

Distribuovaný Floyd-Warshall – pseudokód

Algoritmus je spuštěn ve vrcholu u.

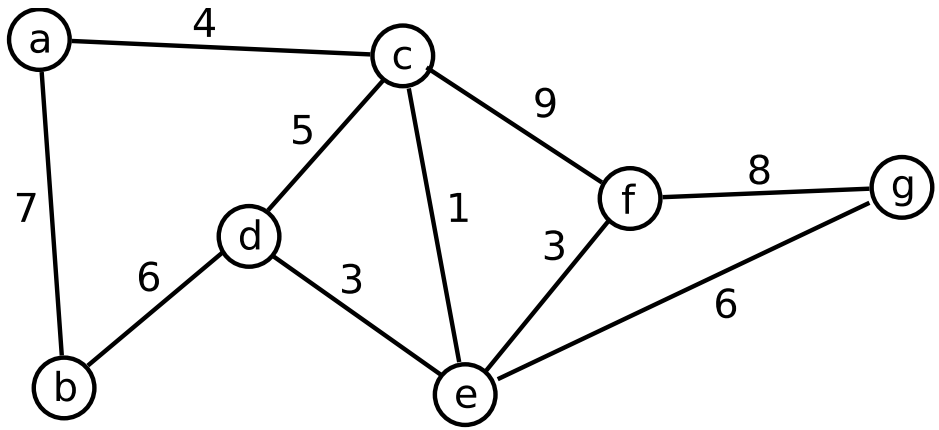
```
d[u] = 0; p[u] = nedef.
Pro všechny ostatní vrcholy v:
| Je-li v sousední vrchol:
| | d[v] = w(u,v); p[v] = v;
| Jinak:
| | d[v] = nekon.; p[v] = nedef;
Dokud nebyly vybrány všechny vrcholy:
| Vyber doposud nevybraný vrchol v.
| Pokud u = v:
| | Rozešli ostatním vrcholům pole d.
| Jinak:
| | Přijmi od vrcholu v jeho pole dv.
| Pro všechny vrcholy w:
| | d[w] = minimum(d[w], d[v]+dv[w]), uprav p[v].
```

Distribuovaný Floyd-Warshall

- Pro správnost algoritmu je nutné, aby všechny výpočetní uzly (vrcholy grafu) vybraly vždy stejný vrchol, který poté rozesílá svoji tabulku vzdáleností.
- Algoritmus je neefektivní z hlediska množství zasílaných dat. Pokud v některém vrcholu platí $\mathtt{d}[\mathtt{v}] = \infty$ pro právě vybraný vrchol v, jeho cesty se nijak neupraví a nemusí mu tedy být zasílána tabulka vzdáleností tohoto vrcholu.
- Před rozesláním tabulky vzdáleností se mohou vrcholy vzájemně informovat, které mají obdržet tuto tabulku s výrazně nižšími nároky na přenesená data $\Leftarrow$ Touegův algoritmus.

Cvičení

1. Na grafu níže vypočítejte nejkratší cesty použitím Dijkstrova, Bellman-Fordova a Floyd-Warshallova algoritmu. V případě prvních dvou použijte různé počáteční vrcholy.



2. Navrhněte způsob implementace nastíněného vylepšení distribuovaného Floyd-Warshallova algoritmu. Uvažujte, že výpočetní uzly mohou zasílat zprávy jen po hranách grafu (broadcasting je implementován přeposíláním zpráv mezi uzly).

Cvičení

3. Proč nepracuje Dijkstrův algoritmus korektně na grafech obsahujících záporně ohodnocené hrany? K jakým výsledkům může dojít, je-li na takovém grafu spuštěn?
4. Dokažte tvrzení, označované také jako trojúhelníková nerovnost v grafech:

$$\forall u, v, w \in V(G) : \delta(u, w) \leq \delta(u, v) + \delta(v, w)$$

5. Nechť vstupem Bellman-Fordova algoritmu je graf, v němž každá nejkratší cesta obsahuje nejvýše k hran. Navrhněte úpravu algoritmu umožňující ukončit výpočet po $k + 1$ iteracích.