PB165 – Grafy a sítě

Hledání nejkratších cest

Obsah přednášky

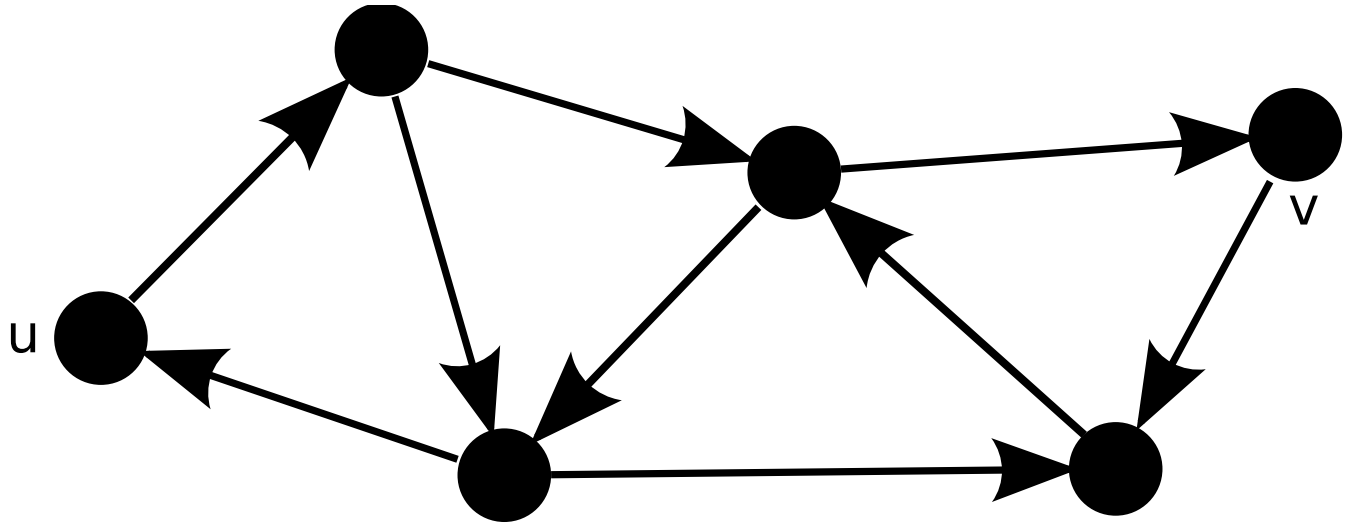
- Úvod
- Nejkratší cesty z jednoho vrcholu
  - Dijkstrův algoritmus
  - A* algoritmus
  - Bellman-Ford algoritmus
- Cesty mezi všemi vrcholy
  - Floyd-Warshallův algoritmus
  - Distribuovaný Floyd-Warshallův algoritmus

Vzdálenost v neohodnoceném 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: Vzdálenosti 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 smysl, v grafu nemůže existovat cyklus záporné délky.

Nejkratší cesty a cykly

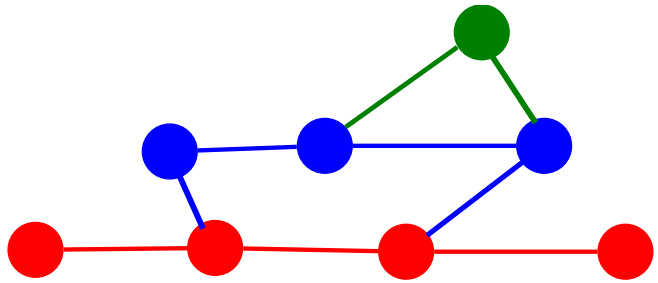
Věta

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. □

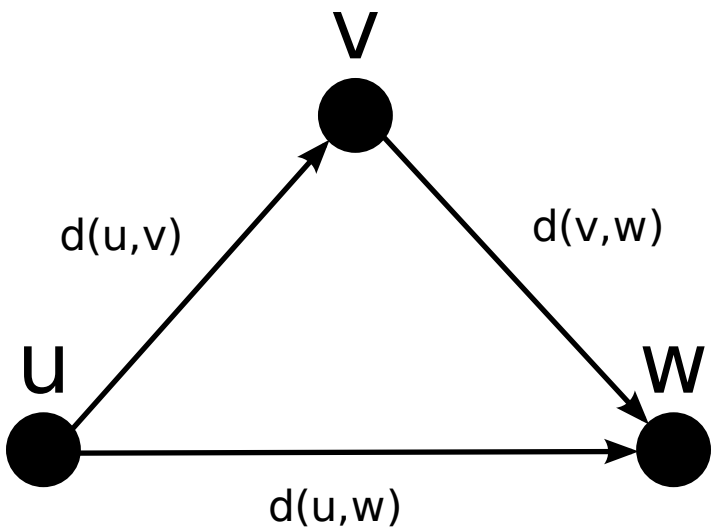
Obrázek: Nejkratší sled je vyznačen červeně. Delší sledy vedou i po modrých a zelených hranách a tvoří cykly.



Trojúhelníková nerovnost

Graf G splňuje *trojúhelníkovou nerovnost*, pokud pro libovolnou trojici jeho vrcholů u, v, w platí:

$$\delta(u, w) \leq \delta(u, v) + \delta(v, w)$$



Obecný graf tuto nerovnost nesplňuje. Příkladem, pro který trojúhelníková nerovnost platí, je např. graf nejkratších (nikoliv přímých) silničních vzdáleností mezi městy.

Dijkstrův algoritmus

- "Klasický" algoritmus hledání nejkratší cesty.
- Najde nejkratší cesty z jednoho vrcholu do všech ostatních.
- Pracuje pro orientovaný i neorientovaný graf.
- Vyžaduje nezáporné ohodnocení všech hran (nejen cyklů).
- 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 doposud 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 s.
  - 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 právě ty 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).

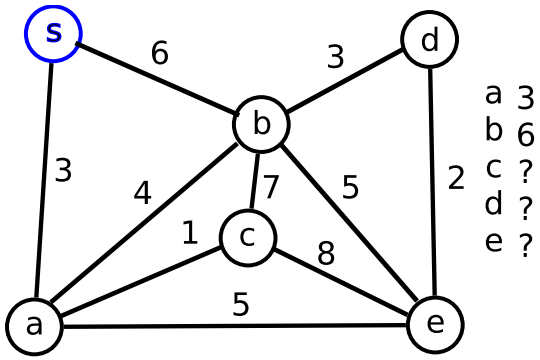
Dijkstrův algoritmus – pseudokód

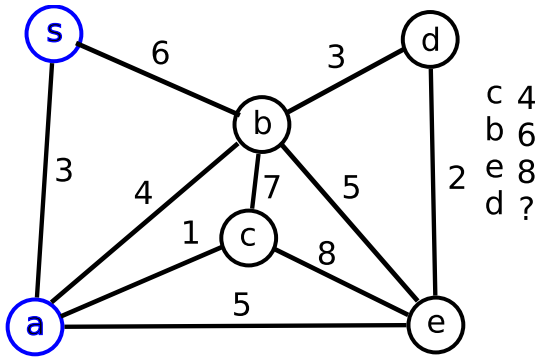
Vycházíme z počátečního vrcholu s.

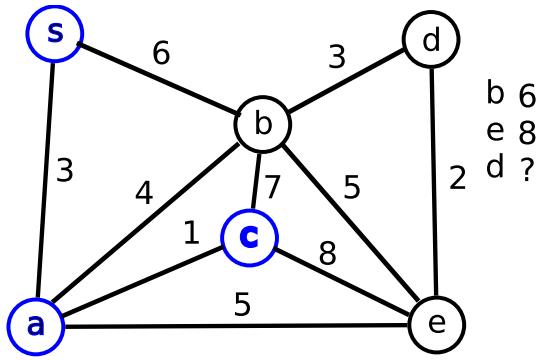
Vlož všechny vrcholy do Q
$d[s] \leftarrow 0$; $p[s] \leftarrow$ undef
for all $u \in V \backslash \{s\}$ **do**
  $d[u] \leftarrow \infty$
  $p[u] \leftarrow$ undef
end for
while $Q \neq \emptyset$ **do**
  Odstraň z Q vrchol u s nejvyšší prioritou
  for all Hrany (u, v) **do**
    if $d[v] > d[u] + w(u, v)$ **then**
      $d[v] \leftarrow d[u] + w(u, v)$
      $p[v] \leftarrow u$
    end if
  end for
end while

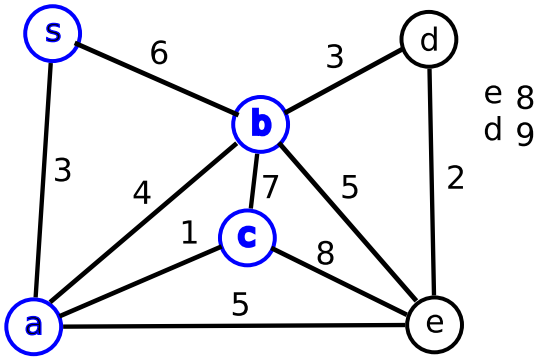
Dijkstrův algoritmus – příklad

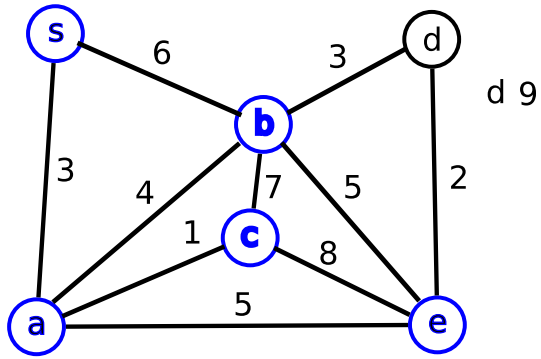
Obrázek: Vrcholy množiny S jsou vyznačeny modře. Napravo od grafu je znázorněn stav prioritní fronty.

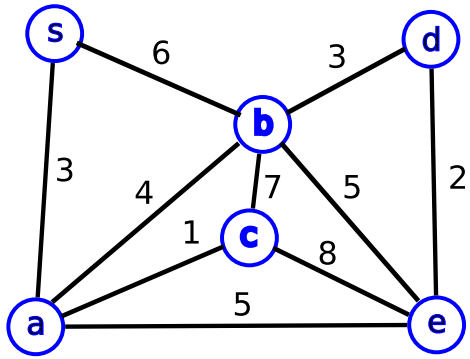












Dijkstrův algoritmus – animace/ilustrace výpočtu

- animace výpočtu na ukázkovém grafu
  - `http://www.unf.edu/~wkloster/foundations/DijkstraApplet/DijkstraApplet.htm`
- komentovaný výpočet
  - `http://www.youtube.com/watch?v=8Ls1RqHCOPw`
- výpočet s možností definice vlastního grafu
  - `http://www.cse.yorku.ca/~aaw/HFHuang/DijkstraStart.html`

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 složitost 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 zúčastní.

- Každý aktivní prvek periodicky rozesílá seznam sousedních vrcholů.
- Ten je propagován skrze síť ke všem aktivním prvkům.
- Každý aktivní prvek nezávisle na ostatních 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.

A* algoritmus

- Upravený Dijkstrův algoritmus.
- Používá se zejména k nalezení cesty do jednoho vrcholu – např. navigace.
- Kromě 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 F jako součet ohodnocení jejích hran G a heuristické ohodnocení koncového vrcholu H. F=G+H
- 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 cesty 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 vrcholy 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://www.video.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
while $Q \neq \emptyset$ **do**
  Odstraň cestu p z Q. Nechť x je její koncový vrchol
  if x je uzavřený **then**
    pokračuj další iterací
  end if
  if $x = c$ **then**
    Vrať cestu p jako výsledek
  end if
  Uzavři x
  for all hrany (x, y) začínající v x **do**
    Vlož do fronty cestu $p + (x, y)$
  end for
end while
Vrať zprávu o neexistenci cesty

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 – $\mathcal{O}(mn)$.

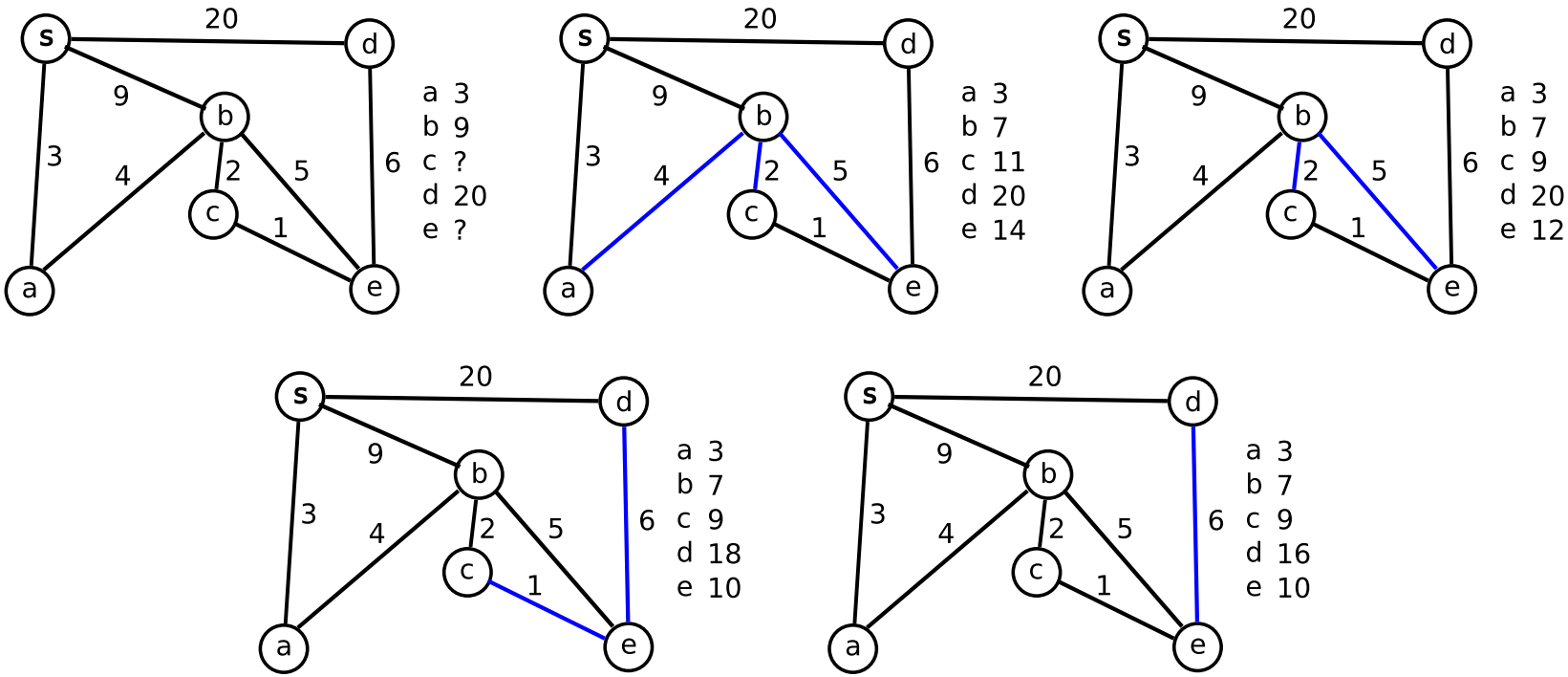
Bellman-Ford – pseudokód

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

$d[s] \leftarrow 0$; $p[s] \leftarrow$ undef
for all vrcholy v grafu vyjma počátečního **do**
  $d[u] = \infty$; $p[u] =$nedef.
end for
for $i = 1 \rightarrow n$ **do**
  for all hrany (u, v) **do**
    if $d[v] > d[u] + w(u, v)$ **then**
      $d[v] \leftarrow d[u] + w(u, v)$, $p[v] \leftarrow u$
    end if
  end for
end for
for all hrany (u, v) **do**
  if $d[v] > d[u] + w(u, v)$ **then**
    Chyba: graf obsahuje cyklus neg. váhy
  end if
end for

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.



* Modře by měly být i hrany (s,a), (s,b) a (s,d) na prvním obrázku.

Bellman-Ford – animace/ilustrace výpočtu

- komentovaná animace výpočtu (česky)
  - `http://www.youtube.com/watch?v=LrYL6akqpHU`
- komentovaná animace výpočtu (anglicky)
  - `http://www.csanimated.com/animation.php?t=Bellman-Ford_algorithm`

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.

Floyd-Warshallův algoritmus

- Vypočítává nejkratší vzdálenost mezi všemi dvojicemi vrcholů v grafu.
- Graf může obsahovat záporně ohodnocené hrany, cykly s celkovým záporným 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 $(k+1)$-ní 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 cesty využívající všech vrcholů grafu.

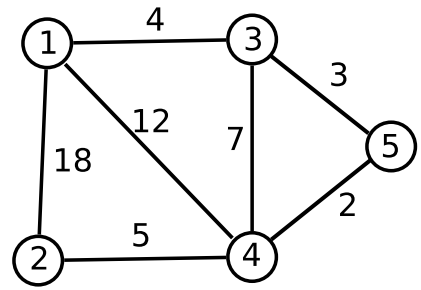
Floyd-Warshallův algoritmus – pseudokód

- 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 ∞

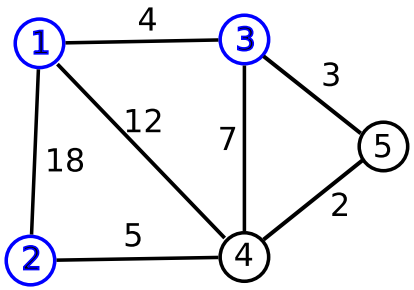
for all $k = 1 \rightarrow n$ **do**
  for all $i = 1 \rightarrow n$ **do**
    for all $j = 1 \rightarrow n$ **do**
      $d[i, j] \leftarrow \min(d[i, j], d[i, k] + d[k, j])$
    end for
  end for
end for

Floyd-Warshallův algoritmus – příklad

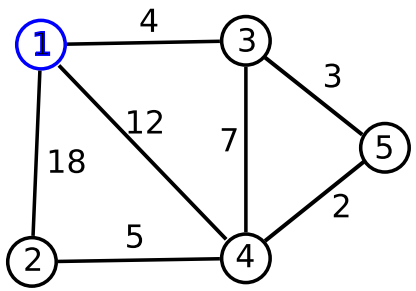
Obrázek: Vrcholy, kterými mohou vést cesty, jsou vyznačeny. Matice udává nejkratší nalezené vzdálenosti mezi dvojicemi vrcholů.



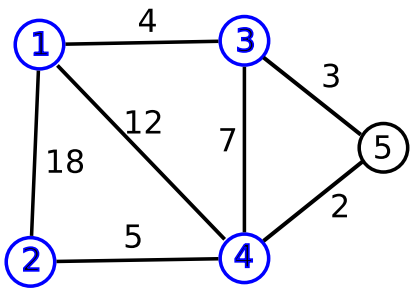
	1	2	3	4	5
1	0	18	4	12	?
2	18	0	?	5	?
3	4	?	0	7	3
4	12	5	7	0	2
5	?	?	3	2	0



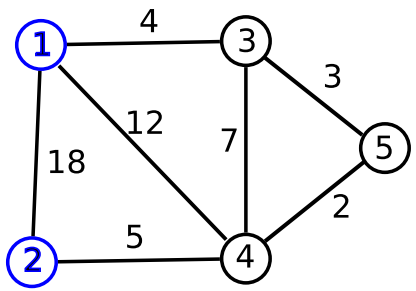
	1	2	3	4	5
1	0	18	4	11	7
2	18	0	22	5	25
3	4	22	0	7	3
4	11	5	7	0	2
5	7	25	3	2	0



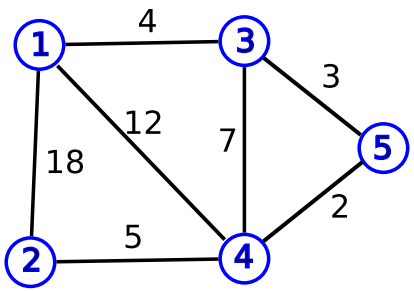
	1	2	3	4	5
1	0	18	4	12	?
2	18	0	22	5	?
3	4	22	0	7	3
4	12	5	7	0	2
5	?	?	3	2	0



	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.

Inicializace:

$d[u] \leftarrow 0$; $p[u] \leftarrow$ nedef
for all $v \in V \backslash \{u\}$ **do**
  if Existuje hrana (u, v) **then**
    $d[v] \leftarrow w(u, v)$; $p[v] \leftarrow u$
  else
    $d[v] \leftarrow \infty$; $p[v] \leftarrow$ nedef
  end if
end for

Hlavní cyklus:

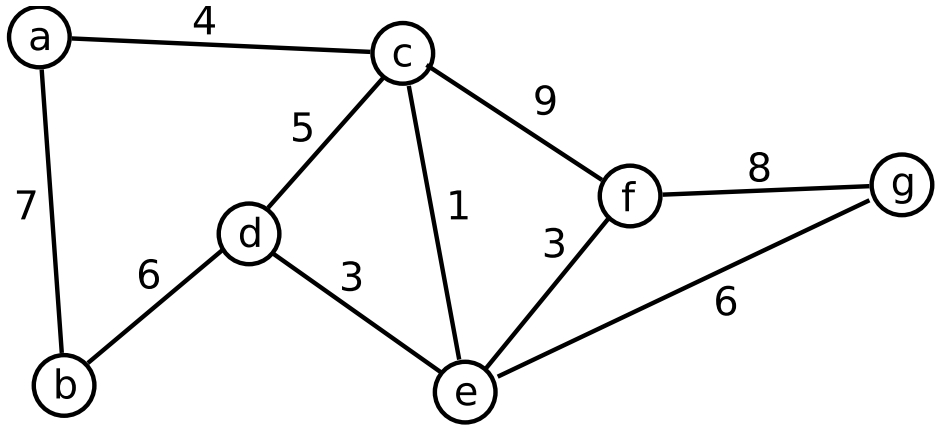
while nebyly vybrány všechny vrcholy **do**
  Vyber doposud nevybraný vrchol v
  if $u = v$ **then**
    Rozešli ostatním vrcholům pole d
  else
    Přijmi od vrcholu v jeho pole d'
  end if
  for all $w \in V$ **do**
    $d[w] \leftarrow \min(d[w], d[v] + d'[w])$
    uprav $p[v]$
  end for
end while

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 $\mathtt{d[v]} = \infty$ 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 $\Rightarrow$ Touegův algoritmus.
- Další informace:
  - Ajay D. Kshemkalyani, Mukesh Singhal. *Distributed Computing: Principles, Algorithms, and Systems*. Cambridge University Press, 2008. Str. 151-155

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. 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.