Matematika III – 8. přednáška
Grafy a algoritmy – cesty a souvislost

Michal Bulant

Masarykova univerzita
Fakulta informatiky

6. 11. 2007

Obsah přednášky

Doporučené zdroje

- Martin Panák, Jan Slovák, Drsná matematika, *e*-text.
- *Předmětové záložky v IS MU*
- Jiří Matoušek, Jaroslav Nešetřil, Kapitoly z diskrétní matematiky, Univerzita Karlova v Praze, Karolinum, Praha, 2000, 377 s.
- Petr Hliněný, Teorie grafů, studijní materiály, http://www.fi.muni.cz/~hlineny/Vyuka/GT/
- Donald E. Knuth, The Stanford GraphBase, ACM, New York, 1993 (`http://www-cs-faculty.stanford.edu/~knuth/sgb.html`).

Grafy jsou jazykem, ve kterém často formulujeme algoritmy. Rozumíme tím postup, kdy v nějakém orientovaném grafu přecházíme z uzlu do uzlu podél hran a přitom zpracováváme informace, které jsou určeny a ovlivněny:

- výsledkem předchozích operací,
- uzlem, ve kterém se zrovna nacházíme
- a hranou, kterou jsme do uzlu vstoupili.

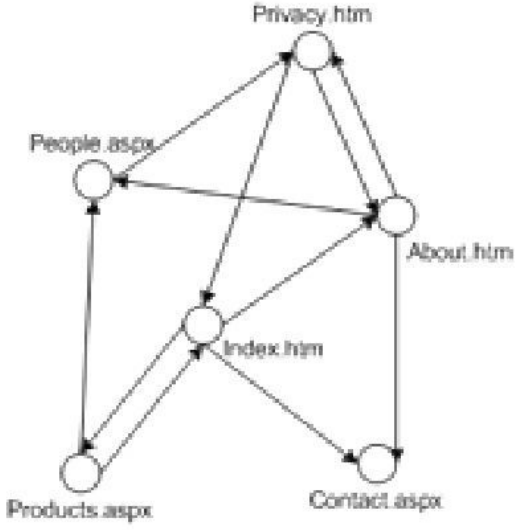
Při zpracování informace se zároveň rozhodujeme, kterými výstupními hranami budeme pokračovat a v jakém pořadí. Pokud je graf neorientovaný, můžeme všechny hrany považovat za dvojice hran orientované opačnými směry.

Abychom mohli algoritmy realizovat pomocí počítače, je třeba umět uvažovaný graf efektivně zadat. Uvedeme dva příklady:

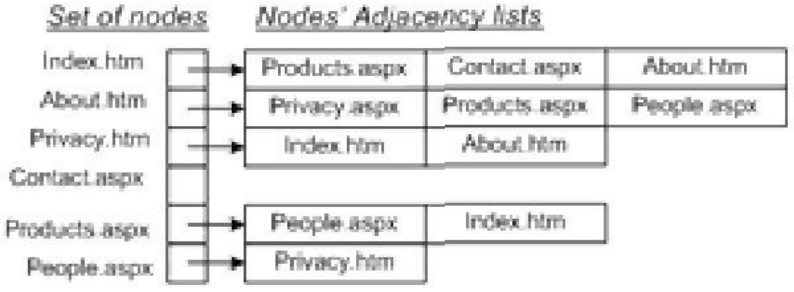
Definice (Hranový seznam/Edge List)

Graf $G = (V, E)$ si v něm reprezentujeme jako dva seznamy V a E propojené ukazateli tak, že každý vrchol ukazuje na všechny z něj vycházející hrany (případně také na všechny do něj vcházející hrany u orientovaných grafů) a každá hrana ukazuje na svůj počáteční a koncový vrchol.

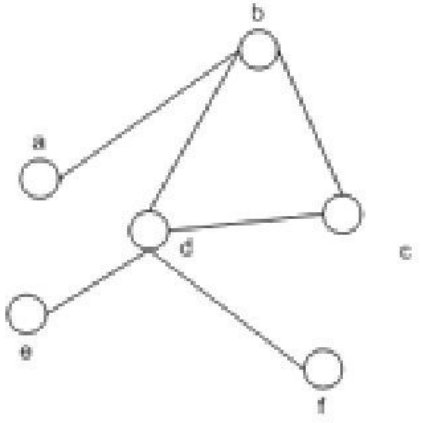
Je vidět, že pamět potřebná na uchování grafu je v tomto případě $O(|V| + |E|)$, protože na každou hranu ukazujeme právě dvakrát a na každý vrchol ukazujeme tolikrát, kolik je jeho stupeň a součet stupňů je také roven dvojnásobku počtu hran. Až na konstantní násobek jde tedy stále o optimální způsob uchovávání grafu v paměti.



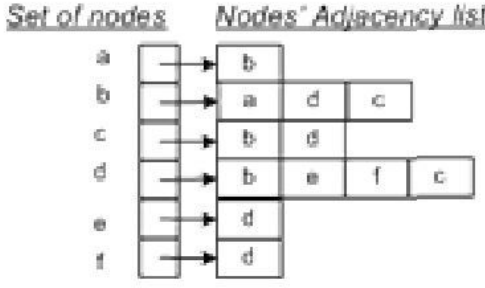
Directed Graph (a)



Adjacency List Representation (a)



Undirected Graph (b)



Adjacency List Representation (b)

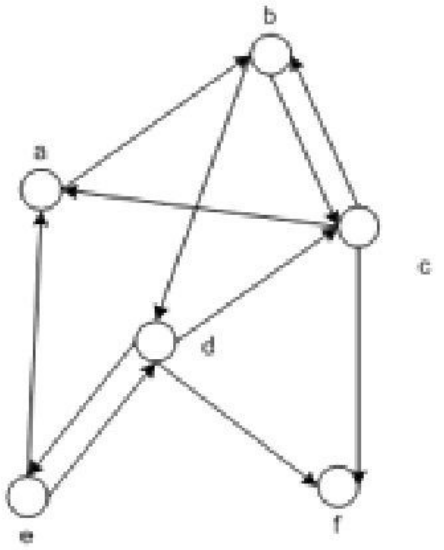
Definice (Matice sousednosti grafu)

Uvažme (neorientovaný) graf $G = (V, E)$, zvolme uspořádání jeho vrcholů $V = (v_1, \ldots, v_n)$ a definujme matici $A_G = (a_{ij})$ nad $\mathbb{Z}_2$ (tj. zaplněnou jen nulami a jedničkami) takto:

$$a_{ij} = \begin{cases} 1 & \text{jestliže je hrana } e_{ij} = \{v_i, v_j\} \text{ v } E \\ 0 & \text{jestliže není hrana } e_{ij} = \{v_i, v_j\} \text{ v } E \end{cases}$$

Matici A_G nazýváme matice sousednosti grafu G.

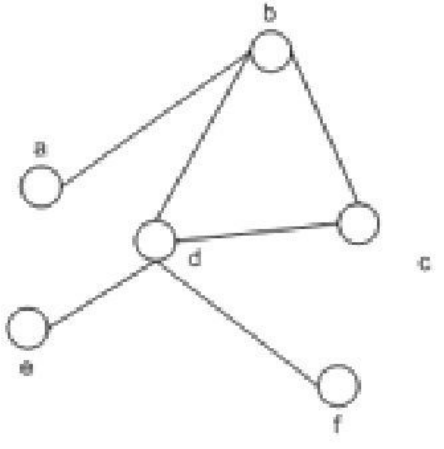
Zadání grafu pomocí matice sousednosti potřebuje vždy $O(n^2)$ místa v paměti. Pokud je ale v grafu málo hran, dostáváme tzv. řídkou matici se skoro všemi prvky nulovými. Takové lze naopak reprezentovat pomocí „hranových seznamů“ odpovídajících grafů a to i včetně obecných číselných hodnot pro jednotlivé hrany. Příklad pro procvičení – násobení matic pomocí reprezentace hranovým seznamem.



Directed Graph (a)

	a	b	c	d	e	f
a		✓				
b			✓	✓		
c		✓				✓
d			✓		✓	✓
e	✓			✓		
f						

Adjacency Matrix Representation (a)



Undirected Graph (b)

	a	b	c	d	e	f
a		✓				
b	✓		✓	✓		
c		✓		✓		
d		✓	✓		✓	✓
e				✓		
f				✓		

Adjacency Matrix Representation (b)

Definice (Základní operace nad grafem)

- *odebrání hrany*
- *přidání hrany*
- *přidání vrcholu*
- *odebrání vrcholu*
- *dělení hrany nově přidaným vrcholem*

Jak se projeví tyto operace v našich reprezentacích?

Jednoduchou aplikací maticového počtu je tvrzení:

Věta

Nechť $G = (V, E)$ je graf s uspořádanými vrcholy $V = (v_1, \ldots, v_n)$ a maticí sousednosti A_G. Označme $A_G^k = (a_{ij}^{(k)})$ prvky k-té mocniny matice A_G. Pak $a_{ij}^{(k)}$ je počet sledů délky k mezi vrcholy v_i a v_j.

Důkaz.

Indukcí. □

Důsledek

Jsou-li $G = (V, E)$ a A_G jako v předchozí větě, pak lze všechny dvojice vrcholů G spojit cestou právě tehdy, když má matice $(A + \mathbb{I}_n)^{n-1}$ samé nenulové členy (zde $\mathbb{I}_n$ označuje jednotkovou matici s n řádky a sloupci).

Prohledávání v grafech

Algoritmy bývají založeny na postupném prohledávání všech vrcholů v grafu. Zpravidla máme zadaný počáteční vrchol nebo si jej na začátku procesu zvolíme. V průběhu procesu pak v každém okamžiku jsou vrcholy

- *již zpracované*, tj. ty, kterými jsme již při běhu algoritmu prošli a definitivně je zpracovali;
- *aktivní*, tj. ty vrcholy, které jsou detekovány a připraveny pro zpracování;
- *spící*, tj. ty vrcholy, na které teprve dojde.

Zároveň si udržujeme přehled o již zpracovaných hranách.
V každém okamžiku musí být množiny vrcholů a/nebo hran v těchto skupinách disjunktním rozdělením množin vrcholů V a množin E hran grafu G a některý z aktivních vrcholů je aktuálně zpracováván.

Základní postup:

- Na počátku máme jeden aktivní vrchol a všechny ostaní vrcholy jsou spící.
- V prvním kroku projdeme všechny hrany vycházející z aktivního vrcholu a jejich příslušným koncovým vrcholům, které jsou spící, změníme stav na aktivní.
- V dalších krocích vždy u zpracovávaného vrcholu probíráme ty z něho vycházející hrany, které dosud nebyly probrány a jejich koncové vrcholy přidáváme mezi aktivní. Tento postup aplikujeme stejně u orientovaných i neorientovaných grafů, jen se drobně mění význam adjektiv koncový a počáteční u vrcholů.

V konkrétních úlohách se můžeme omezovat na některé z hran, které vychází z aktuálního vrcholu. Na principu to ale nic podstatného nemění.

Pro realizaci algoritmů je nutné se rozhodnout, v jakém pořadí zpracováváme aktivní vrcholy a v jakém pořadí zpracováváme hrany z nich vycházející. V zásadě přichází v úvahu dvě možnosti zpracovávání vrcholů:

1. vrcholy vybíráme pro další zpracování ve stejném pořadí, jak se stávaly aktivními (fronta – FIFO)
2. dalším vrcholem vybraným pro zpracování je poslední zaktivněný vrchol (zásobník – LIFO).

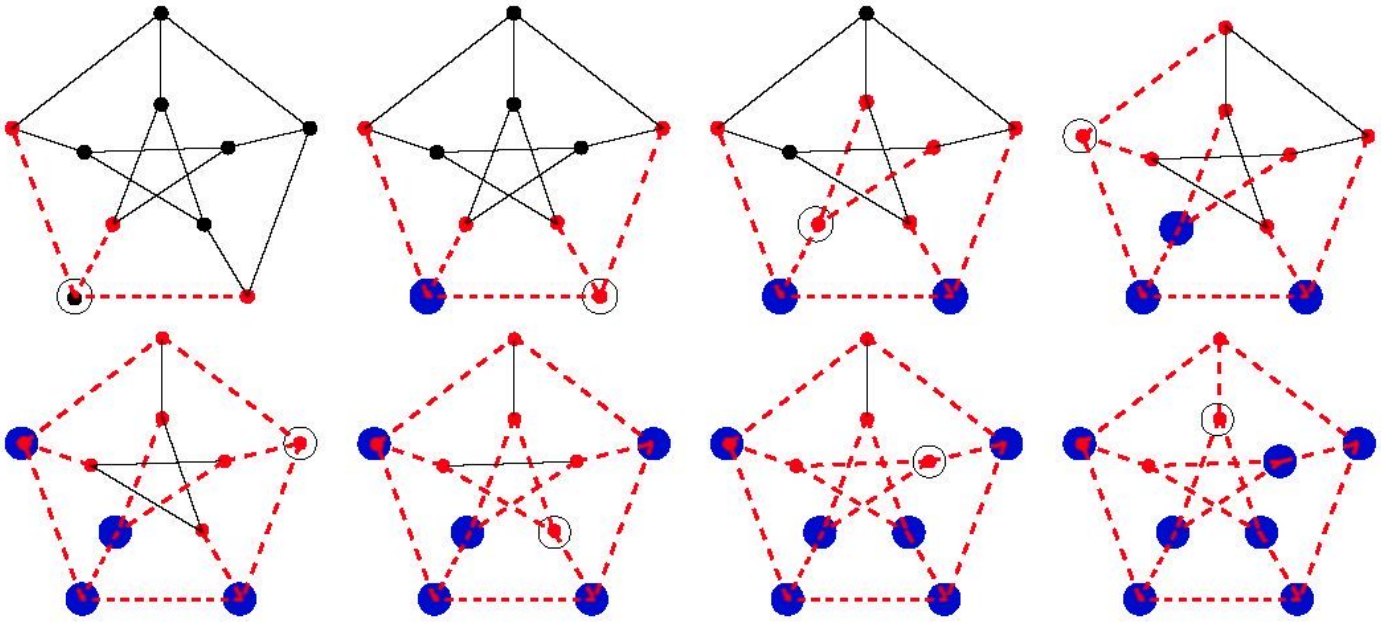
V prvním případě hovoříme o **prohledávání do šířky**, ve druhém o **prohledávání do hloubky**.

Na první pohled je zřejmá role volby vhodných datových struktur pro uchovávání údajů o grafu. Hranový seznam umožňuje projít všechny hrany vycházející z právě zpracovávaného vrcholu v čase lineárně úměrném jejich počtu. Každou hranu přitom diskutujeme nejvýše dvakrát, protože má právě dva konce. Zjevně tedy platí:

Věta

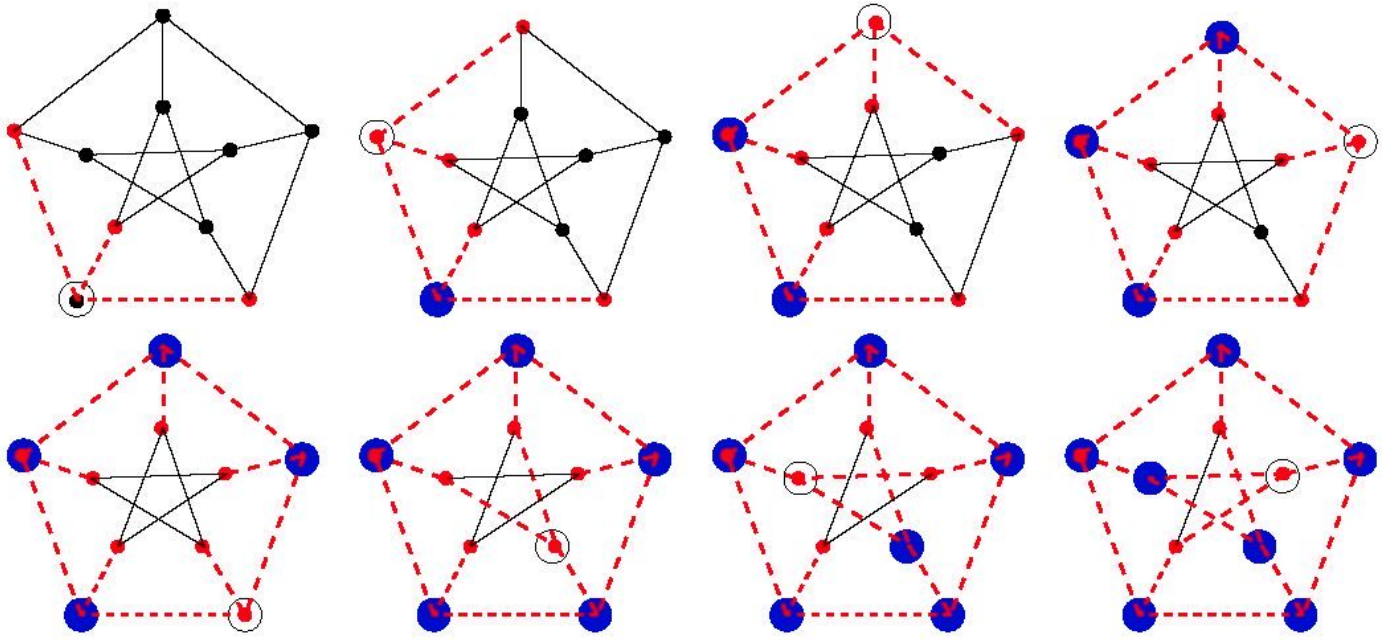
Celkový čas realizace vyhledávání do šířky i do hloubky je $O((n + m) * K)$, kde n je počet vrcholů v grafu, m je počet hran v grafu a K je čas potřebný na zpracování jedné hrany, resp. jednoho vrcholu.

Ilustrace prohledávání do šířky:



Zakroužkovaný vrchol je ten právě zpracovávaný, modré velké puntíky jsou již zpracované uzly, čárkované červené hrany jsou již zpracované a červené drobné uzly jsou ty aktivní (poznají se také podle toho, že do nich již vede některá zpracovaná hrana). Hrany zpracováváme v pořadí orientace proti hodinovým ručkám, přičemž za „první“ bereme směr „kolmo dolů“.

Totéž postupem „do hloubky“. Všimněte si, že první krok je stejný jako v předchozím případě.



Prohledávání „do hloubky“ odpovídá interpretaci rekurze v běžných imperativních jazycích (např. i v Maplu) – v těchto případech je ale *stavový graf* potenciálně nekonečný a prohledávání do hloubky tak samozřejmě nemusí nalézt všechny vrcholy dostupné z daného vrcholu po cestách konečné délky (narozdíl od prohledávání do šířky).

Prohledávání bludiště

Průchod bludištěm (s křídou) prostřednictvím prohledávání do hloubky v neorientovaném grafu – úkolem je najít východ nebo dokázat, že neexistuje:

- každou chodbu při prvním průchodu označíme čarou, při návratu přidáme druhou čáru
- při východu z místnosti preferujeme neoznačenou chodbu
- navštívenou místnost označíme, pokud je již označená, tak se ihned vrátíme stejnou chodbou zpět
- z dosud nenavštívené místnosti postupujeme dále
- jsou-li již všechny chodby vedoucí z místnosti použity, vracíme se zpět chodbou, která je označena jen jednou čarou
- pokud z dané místnosti vedou pouze chodby, které jsou označeny 2 čarami, hledání ukončíme (nutně jde o výchozí místnost, východ neexistuje a můžeme s klidem umřít).

Souvislé komponenty grafu

Definice

Nechť je $G = (V, E)$ neorientovaný graf. Na množině vrcholů grafu G zavedeme relaci $\sim$ tak, že $v \sim w$ právě když existuje cesta z v do w. Promyslete si, že tato relace je dobře definovaná a že se jedná o ekvivalenci. Každá třída $[v]$ této ekvivalence definuje indukovaný podgraf $G_{[v]} \subset G$ a disjunktní sjednocení těchto podgrafů je ve skutečnosti původní graf G. Podgrafům $G_{[v]}$ říkáme **souvislé komponenty grafu** G.
Pro orientované grafy postupujeme stejně, pouze u $\sim$ požadujeme aby cesta existovala z uzlu v do uzlu w nebo naopak z uzlu w do uzlu v.

Jinými slovy: Každý graf $G = (V, E)$ se přirozeně rozpadá na disjunktní podgrafy G_i takové, že vrcholy $v \in G_i$ a $w \in G_j$ jsou spojeny nějakou cestou právě, když $i = j$. To jsou právě souvislé komponenty grafu G.

Pro hledání všech souvislých komponent v grafu lze užít prohledávání — dodatečnou informací, kterou musíme zpracovávat je, kterou komponentu aktuálně procházíme.

Definice

Řekneme, že graf $G = (V, E)$ je

- **souvislý**, jestliže má právě jednu souvislou komponentu;
- **vrcholově k–souvislý**, jestliže má alespoň $k + 1$ vrcholů a bude souvislý po odebrání libovolné podmnožiny $k - 1$ vrcholů;
- **hranově k–souvislý**, jestliže bude souvislý po odebrání libovolné podmnožiny $k - 1$ hran.

Hranu, jejímž odebráním se zvýší počet souvislých komponent grafu G, nazýváme *mostem*.

Silnější souvislost grafu je žádoucí např. u síťových aplikací, kdy požadujeme značnou redundanci poskytovaných služeb v případě výpadku některých linek (tj. hran) nebo uzlů (tj. vrcholů).

Speciální případ: 2–souvislý graf je takový souvislý graf o alespoň třech vrcholech, kdy vynecháním libovolného vrcholu nenarušíme jeho souvislost.

Věta

Pro graf $G = (V, E)$ s alespoň třemi vrcholy jsou následující podmínky ekvivalentní:

- *G je (vrcholově) 2–souvislý;*
- *každé dva vrcholy v a w grafu G leží na společné kružnici;*
- *graf G je možné vytvořit z trojúhelníku K_3 pomocí postupných dělení hran.*

Obecnější je tzv. **Mengerova věta** (obd. „vrcholová verze“):

Věta

Pro každé dva vrcholy v a w v grafu $G = (V, E)$ je počet hranově různých cest z v do w roven minimálnímu počtu hran, které je třeba odstranit, aby se v a w ocitly v různých komponentách vzniklého grafu.

Metrika na grafech

Na každém (neorientovaném) grafu definujeme **vzdálenost uzlů** v a w jako číslo $d_G(v, w)$, která je rovna počtu hran v nejkratší možné cestě z v do w. Pokud cesta neexistuje, píšeme $d_G(v, w) = \infty$.
Budeme v dalším uvažovat pouze souvislý graf G. Pak pro takto zadanou funkci $d_G : V \times V \to \mathbb{N}$ platí obvyklé tři vlastnosti vzdálenosti:

- $d_G(v, w) \geq 0$ a přitom $d_G(v, w) = 0$ právě tehdy, když $v = w$;
- vzdálenost je symetrická, tj $d_G(v, w) = d_G(w, v)$;
- platí trojúhelníková nerovnost, tj. pro každou trojici vrcholů v, w, z platí

$$d_G(v, z) \leq d_G(v, w) + d_G(w, z).$$

Říkáme, že d_G je **metrika na grafu** G.

Metrika na grafu splňuje navíc:

- $d_G(v, w)$ má vždy nezáporné celočíselné hodnoty;
- je-li $d_G(v, w) > 1$, pak existuje nějaký vrchol z různý od v a w a takový, že $d_G(v, w) = d_G(v, z) + d_G(z, w)$.

Každá funkce d_G s výše uvedenými pěti vlastnostmi na $V \times V$ je metrikou nějakého grafu s vrcholy V.

Dijkstrův algoritmus

Nejkratší cestu v grafu, která vychází z daného uzlu v a končí v jiném uzlu w můžeme hledat pomocí prohledávání grafu do šířky. Při tomto typu prohledávání totiž postupně diskutujeme vrcholy, do kterých se umíme dostat z výchozího vrcholu po jediné hraně, poté projdeme všechny, které mají vzdálenost nejvýše 2 atd. Na této jednoduché úvaze je založen jeden z nejpoužívanějších grafových algoritmů – tzv. **Dijkstrův algoritmus**.
Tento algoritmus hledá nejkratší cesty za (reálného) předpokladu, kdy jednotlivé hrany e jsou ohodnoceny „vzdálenostmi“, tj. kladnými reálnými čísly $w(e)$. Kromě aplikace na hledání vzdáleností v silničních nebo jiných sítích to mohou být také výnosy, toky v sítích atd.

- Vstupem algoritmu je graf $G = (V, E)$ s ohodnocením hran a počáteční vrchol v_0.
- Výstupem je ohodnocení vrcholů čísly $d_w(v)$, která udávají nejmenší možný součet ohodnocení hran podél cest z vrcholu v_0 do vrcholu v.

Všimněte si, že místo původní úlohy (nalézt nejkratší cestu mezi **dvěma** vrcholy) řešíme i něco navíc – nalezneme nejkratší cestu z v do **všech** vrcholů.
Postup dobře funguje v orientovaných i neorientovaných grafech.
Je skutečně podstatné, že všechna naše ohodnocení jsou kladná.
Zkusme si rozmyslet třeba cestu P_3 se záporně ohodnocenou prostřední hranou. Při procházení sledu mezi krajními vrcholy bychom „vzdálenost“ zmenšovali každým prodloužením sledu o průchod prostřední hranou tam a zpět.

Dijkstrův algoristmus vyžaduje jen drobnou modifikaci obecného prohledávání do šířky:

- U každého vrcholu v budeme po celý chod algoritmu udržovat číselnou hodnotu $d(v)$, která bude horním odhadem skutečné vzdálenosti vrcholu v od vrcholu v_0.
- Množina již zpracovaných vrcholů bude v každém okamžiku obsahovat ty vrcholy, u kterých již nejkratší cestu známe, tj. $d(v) = d_w(v)$.
- Do množiny aktivních (právě zpracovávaných) vrcholů W zařadíme vždy právě ty vrcholy y z množiny spících vrcholů Z, pro které je $d(y) = \min\{d(z);\ z \in Z\}$.

Dijkstrův algoritmus

Předpokládáme, že graf G má alespoň dva vrcholy.

- *Inicializační krok:* Nastavíme hodnoty u všech $v \in V$,

$$d(v) = \begin{cases} 0 & \text{pro } v = v_0 \\ \infty & \text{pro } v \neq v_0, \end{cases}$$

  nastavíme $Z = V$, $W = \emptyset$.
- *Test cyklu:* Pokud není ohodnocení všech vrcholů $y \in Z$ rovno ∞, pokračujeme dalším krokem, v opačném případě algoritmus končí.

- *Aktualizace stavu vrcholů:*
  - Najdeme množinu N všech vrcholů $v \in Z$, pro které $d(v)$ nabývá nejmenší možné hodnoty

$$\delta = \min\{d(y);\ y \in Z\};$$

  - posledně zpracované aktivní vrcholy W přesuneme do množiny zpracovaných a za nové aktivní vrcholy zvolíme $W = N$ a odebereme je ze spících, tj. množina spících bude nadále $Z \setminus N$.
- *Tělo hlavního cyklu:* Pro všechny hrany v množině E_{WZ} všech hran vycházejících z některého aktivního vrcholu v a končících ve spícím vrcholu y opakujeme:
  - Vybereme dosud nezpracovanou hranu $e\{x, y\} \in E_{WZ}$;
  - Pokud je $d(x) + w(e) < d(y)$, nahradíme $d(y)$ touto menší hodnotou.

Věta

Pro všechny vrcholy v ležící v souvislé komponentě vrcholu s najde Dijsktrův algoritmus vzdálenosti $d_w(v)$. Vrcholy ostatních souvislých komponent zůstanou ohodnoceny $d(v) = \infty$. Algoritmus lze implementovat tak, že ukončí svoji práci v čase $O(n \log n + m)$, kde n je počet vrcholů a m je počet hran v grafu G.

Důkaz.

- během celého algoritmu platí $d(v) \geq d_w(v)$ pro všechna $v \in V$.
- po skončení algoritmu platí $d(v) \leq d_w(v)$ pro všechna $v \in V$ (Prostřednictvím silnějšího tvrzení: jsou-li $0 = d_1 < d_2 < \ldots < d_k < \infty$ všechny různé konečné vzdálenosti vrcholů od s a označíme-li $M_i = \{v \in V; d_w(v) = d_i\}$, pak se krok *Aktualizace stavu vrcholů* provede přesně k-krát a po jeho i-tém vykonání platí $N = M_i, \delta = d_i$ a $V \setminus Z = \cup_{k=1}^{i} M_i$.)

□

Modifikace algoritmu

- Pokud nás skutečně zajímá jen nejkratší cesta do konkrétního vrcholu, pak samozřejmě výpočet ukončíme poté, co tento vrchol přejde do množiny již zpracovaných.
- Můžeme využít dodatečné informace (při hledání nejkratší cesty z Brna do Prahy v silniční síti asi nepojedeme přes Ostravu) – *heuristika* $h : V \to \mathbb{R}$ splňující $w(\{x, y\}) \geq h(x) - h(y)$ pro každou hranu $\{x, y\}$. Doporučení: $h(v)$ je dolní odhad vzdálenosti v do cílového vrcholu (např. vzdálenost „vzdušnou čarou“). Místo minimalizace $d(y)$ pro $y \in Z$ tak v algoritmu minimalizujeme $d(y) + h(y)$.

Další algoritmy a aplikace

Principy **Dijkstrova algoritmu** se využívají v OSPF (Open Shortest Paths First) routovacím protokolu.

Bellman-Fordův algoritmus

- pracuje na stejném principu jako Dijkstrův; místo postupu po uzlech je zpracovává „naráz“ – cyklus *relaxace* probíhá $(|V| - 1)$ krát přes všechny hrany
- připouští záporné hrany a detekuje záporné cykly
- je obvykle časově náročnější než Dijkstrův
- distribuovaná verze je (či spíše byla) používána v *distance-vector routing protocol*

Floydův algoritmus (*all pairs shortest paths*)

- v čase $O(n^3)$ vypočte vzdálenosti mezi všemi vrcholy
- vychází z matice A délek hran a postupně počítá matice $U_0, U_1, \ldots, U_{|V|}$, kde $u_k(i,j)$ je délka nejkratší cesty z i do j, kde cesta prochází pouze vrcholy z $\{1, 2, \ldots, k\}$.
- výpočet vychází ze vztahu

$$u_k(i,j) = \min\{u_{k-1}(i,j), u_{k-1}(i,k) + u_{k-1}(k,j)\}.$$

- je efektivnější než „upravená mocnina“ A_G (násobení $\rightarrow$ sčítání, sčítání $\rightarrow$ minimum) – ta je totiž $O(n^4)$, resp. (optimálizovaná) $O(n^3 \log n)$.