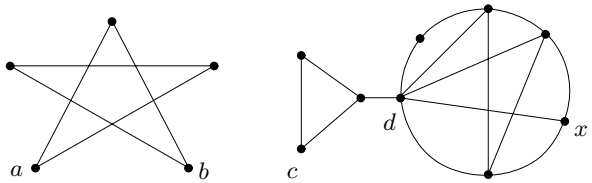
3 Vzdálenost a nejkratší cesty v grafech

Při procházení grafu je někdy prostá informace o souvislosti dostačující, ale většinou bychom rádi věděli i jak je z jednoho vrcholu do druhého daleko . . .



V jednodušším případě se při zjišťování grafové vzdálenosti díváme jen na minimální počet prošlých hran z vrcholu do vrcholu. V obecném případě však při určování vzdálenosti bereme do úvahy délky jednotlivých hran podél cesty (tyto délky musí být nezáporné!). □

Stručný přehled lekce

- Vzdálenost v grafech a její vlastnosti.
- Dynamický výpočet metriky grafu (Floyd–Warshall).
- Dijkstrův algoritmus pro nejkratší (ohodnocenou) cestu v grafu.
- Některé pokročilé myšlenky plánování cest.

3.1 Vzdálenost v grafu

Zopakujme si, že sledem délky n v grafu G rozumíme posloupnost vrcholů a hran $v_0, e_1, v_1, e_2, v_2, \ldots, e_n, v_n$, ve které hrana e_i má koncové vrcholy v_{i-1}, v_i.

Definice 3.1. Vzdálenost $d_G(u, v)$ dvou vrcholů u, v v grafu G je dána délkou nejkratšího sledu mezi u a v v G.
Pokud sled mezi u, v neexistuje, je vzdálenost $d_G(u, v) = \infty$. □

Neformálně řečeno, vzdálenost mezi u, v je rovna *nejmenšímu počtu hran*, které musíme projít, pokud se chceme dostat z u do v. Speciálně $d_G(u, u) = 0$.

Uvědomme si, že nejkratší sled je vždy cestou (vrcholy se neopakují) – Věta 2.2.

Fakt: V neorientovaném grafu je vzdálenost symetrická, tj. $d_G(u, v) = d_G(v, u)$. □

Lema 3.2. *Vzdálenost v grafech splňuje trojúhelníkovou nerovnost:*

$$\forall u, v, w \in V(G) \ : \quad d_G(u, v) + d_G(v, w) \geq d_G(u, w)\,.$$

Důkaz. Nerovnost snadno plyne ze zřejmého pozorování, že na sled délky $d_G(u, v)$ mezi u, v lze navázat sled délky $d_G(v, w)$ mezi v, w, čímž vznikne sled délky $d_G(u, v) + d_G(v, w)$ mezi u, w. Skutečná vzdálenost mezi u, w pak už může být jen menší. □

Základní zjištění vzdálenosti

Věta 3.3. *Nechť u, v, w jsou vrcholy souvislého grafu G takové, že $d_G(u,v) < d_G(u,w)$. Pak při algoritmu procházení grafu G do šířky z vrcholu u je vrchol v nalezen dříve než vrchol w.* □

Důkaz. Postupujeme indukcí podle vzdálenosti $d_G(u,v)$: Pro $d_G(u,v) = 0$, tj. $u = v$ je tvrzení jasné – vrchol u jako počátek prohledávání byl nalezen první. Proto nechť $d_G(u,v) = d > 0$ a označme v' souseda vrcholu v bližšího k u, tedy $d_G(u,v') = d-1$. Obdobně uvažme libovolného souseda w' vrcholu w. Pak

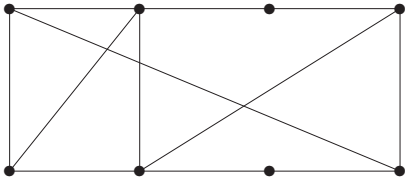
$$d_G(u,w') \geq d_G(u,w) - 1 > d_G(u,v) - 1 = d_G(u,v')\,,$$

a tudíž vrchol v' byl nalezen v prohledávání do šířky dříve než vrchol w' podle indukčního předpokladu. To znamená, že v' se dostal do fronty úschovny dříve než w'. Proto sousedé v' (mezi nimiž je v, ale ne w neboť $v'w$ není hranou) jsou při pokračujícím prohledávání také nalezeni dříve než w coby soused w'. □ □

Důsledek 3.4. *Algoritmus procházení grafu do šířky lze použít pro výpočet grafové vzdálenosti z daného vrcholu u.* □

Důsledek 3.4 „funguje“ jen pro vzdálenosti s jednotkovou délkou všech hran. My si dále ukážeme obecnější Dijkstrův algoritmus, který obdobným postupem počítá nejkratší vzdálenost při libovolně kladně ohodnocených délkách hran.

Některé další pojmy



Definice 3.5. Mějme graf G. Definujeme (vzhledem k G) následující pojmy a značení:

- *Excentricita* vrcholu $\mathrm{exc}(v)$ je nejdelší vzdálenost z v do jiného vrcholu grafu; $\mathrm{exc}(v) = \max_{x \in V(G)} d_G(v, x)$. □
- *Průměr* $\mathrm{diam}(G)$ grafu G je největší excentricita jeho vrcholů, naopak *poloměr* $\mathrm{rad}(G)$ grafu G je nejmenší excentricita jeho vrcholů. □
- *Centrem* grafu je množina vrcholů $U \subseteq V(G)$ takových, jejichž excentricita je rovna poloměru $\mathrm{rad}(G)$. □
- *Steinerova vzdálenost* mezi vrcholy libovolné podmnožiny $W \subseteq V(G)$ je rovna minimálnímu počtu hran souvislého podgrafu v G obsahujícího všechny vrcholy W.

3.2 Výpočet metriky

Definice: Metrikou grafu myslíme soubor vzdáleností mezi všemi dvojicemi vrcholů grafu. Jinak řečeno, *metrikou grafu G* je matice (dvourozměrné pole) `d[,]`, ve kterém prvek `d[i,j]` udává vzdálenost mezi vrcholy i a j. □

Metoda 3.6. Dynamický výpočet metriky skládáním cest
v grafu G na množině vrcholů $V(G) = \{v_0, v_1, \ldots, v_{N-1}\}$.

- Na počátku nechť `d[i,j]` udává 1 (případně délku hrany $\{v_i, v_j\}$), nebo ∞ pokud hrana mezi i, j není. □
- Po kroku $t \geq 0$ nechť platí, že `d[i,j]` udává délku nejkratšího sledu mezi v_i, v_j, který užívá pouze vnitřní vrcholy z množiny $\{v_0, v_1, \ldots, v_{t-1}\}$ (prázdné v $t = 0$). □
- Při přechodu z kroku t na následující krok $t+1$ upravujeme vzdálenost pro každou dvojici vrcholů v_i, v_j – jsou vždy pouhé dvě možnosti:
  - Buď je sled délky `d[i,j]` z předchozího kroku t stále nejlepší (tj. nově povolený vrchol v_t nám nepomůže),
  - nebo sled vylepšíme spojením přes nově povolený vrchol v_t, čímž získáme menší vzdálenost `d[i,t]+d[t,j]` $\rightarrow$ `d[i,j]`.

Věta 3.7. *Metoda 3.6 v poli* `d[i,j]` *správně vypočte vzdálenost mezi vrcholy v_i, v_j.*

Poznámka: V implementaci pro symbol ∞ použijeme velkou konstantu, třeba `MAX_INT/2`.

Algoritmus 3.8. Výpočet metriky grafu; Floyd–Warshall

vstup $<$ *matice sousednosti* `G[,]` *grafu na* `N` *vrcholech (číslovaných* `0...N-1`*),*
kde `G[i,j]=1` *pro hranu mezi* i, j *a* `G[i,j]=0` *jinak*;

```
for (i=0; i<N; i++) for (j=0; j<N; j++)
    d[i,j] = (i==j?0: (G[i,j]? 1: MAX_INT/2));
for (t=0; t<N; t++) {
    for (i=0; i<N; i++) for (j=0; j<N; j++)
        d[i,j] = min(d[i,j], d[i,t]+d[t,j]);
}
```

výstup '*Matice vzdáleností* `d[,]`'; □

Algoritmus 3.8 je implementačně velmi jednoduchý a provede zhruba N^3 kroků pro výpočet celé metriky.

Jeho jedinou (ale velkou) nevýhodou je, že vzdálenosti mezi všemi dvojicemi vrcholů je třeba *počítat najednou*. V praktických situacích však obvykle požadujeme zjištění vzdálenosti mezi jedinou dvojicí vrcholů, a pak celý zbytek výpočtu je k ničemu...

3.3 Vážená (ohodnocená) vzdálenost

Definice: Vážený graf je dvojice grafu G spolu s ohodnocením w hran reálnými čísly $w : E(G) \to \boldsymbol{R}$.
Kladně vážený graf (G, w) je takový, že $w(e) > 0$ pro všechny hrany e. □

Definice 3.9. (vážená vzdálenost) Mějme (kladně) vážený graf (G, w).
Délkou váženého sledu $S = (v_0, e_1, v_1, e_2, v_2, \ldots, e_n, v_n)$ v G myslíme součet

$$d^w_G(S) = w(e_1) + w(e_2) + \cdots + w(e_n)\,.$$

Váženou vzdáleností v (G, w) mezi dvěma vrcholy u, v pak myslíme

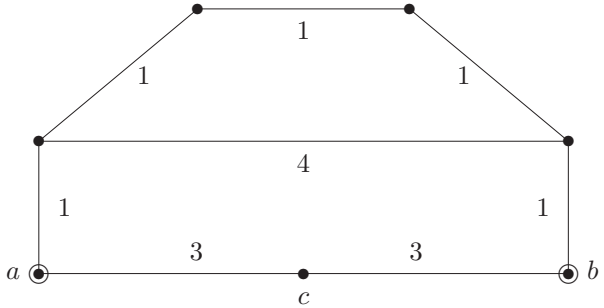
$$d^w_G(u, v) = \min\{d^w_G(S) : S \text{ je sled s konci } u, v\}\,.$$ □

Důležitým faktem je, že stejné definice a dobré vlastnosti zůstávají v platnosti i pro vzdálenost na orientovaných grafech. Obdobně Oddílu 3.1 nyní snadno dokážeme:

Fakt: Nejkratší sled v kladně váženém grafu je vždy cestou (příp. orientovanou). □

Lema 3.10. *Vážená vzdálenost v nezáporně vážených grafech (i orientovaných grafech) splňuje trojúhelníkovou nerovnost.*

Příklad 3.11. *Podívejme se na následující ohodnocený graf (čísla u hran udávají jejich váhy–délky.)*

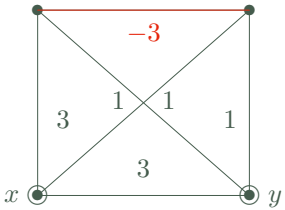


Vzdálenost mezi vrcholy a, c je 3, stejně tak mezi b, c. Co ale mezi a, b? Je jejich vzdálenost 6? Kdepak, vzdálenost a, b je 5, její cesta vede po „horních“ vrcholech.

Povšimněte si, že tento příklad také zároveň ukazuje, že postup prohledáváním do šířky není korektní pro hledání vzdáleností ve váženém grafu. □

Záporné délky hran

Doposud jsme důsledně vyžadovali nezápornost (lépe kladnost) ohodnocení délky hran; ale co je na hranách záporné délky tak „špatného“?



Takže jaká je v našem grafu vzdálenost mezi x, y? Je to 3 nebo 1? □
Ne, vzdálenost z x do y je $-\infty$.

Stejný problém nastane, kdykoliv je možno opakovat záporné hrany, tedy vždy v neorientovaných grafech a částečně i v orientovaných grafech obsahujících „záporné cykly“.

Uvedené příklady problémů se zápornými délkami sledů nás vedou k úplnému vyloučení záporných vah hran v našem výkladu. V jistých omezených případech přesto má smysl hledat nejkratší cesty i v záporných vahách, ale tomu se v našem textu nebudeme věnovat z důvodu nedostatku místa.

3.4 Hledání nejkratší cesty

Pro nalezení nejkratší (vážené) cesty mezi dvěma vrcholy kladně váženého grafu se používá tradiční *Dijkstrův algoritmus* či jeho vhodná vylepšení.

Takové algoritmy se například používají při vyhledávání vlakových spojení. Pravděpodobně se i vy někdy dostanete do situace, kdy budete nejkratší cestu hledat, proto si popsaný algoritmus včetně jeho vylepšení A^* zapamatujte.

Poznámka: Dijkstrův algoritmus je sice poněkud složitější než Algoritmus 3.8, ale na druhou stranu je *výrazně rychlejší*, pokud nás zajímá jen nejkratší vzdálenost z jednoho vrcholu místo všech dvojic vrcholů. □

Dijkstrův algoritmus

- Je variantou procházení grafu (skoro jako do šířky), kdy pro každý nalezený vrchol ještě máme *proměnnou udávající vzdálenost* – délku nejkratšího sledu (od počátku), kterým jsme se do tohoto vrcholu zatím dostali. □
- Z úschovny nalezených vrcholů vždy vybíráme vrchol s nejmenší vzdáleností (mezi uschovanými vrcholy) – do takového vrcholu se už lépe dostat nemůžeme, protože všechny jiné cesty by byly dle výběru delší. □
- Na konci zpracování tyto proměnné vzdálenosti udávají správně nejkratší vzdálenosti z počátečního vrcholu do ostatních.

Algoritmus 3.12. Dijkstrův pro nejkratší cestu v grafu

Tento algoritmus nalezne nejkratší cestu mezi vrcholy u a v kladně váženého grafu G.

vstup < *graf na* N *vrcholech daný maticí vzdáleností sousedů* del[,] ≥ 0,
kde del[i,j]$=\infty$ *pokud* j *není sousedem* i
vstup < u,v, *kde hledáme cestu z* u *do* v;□

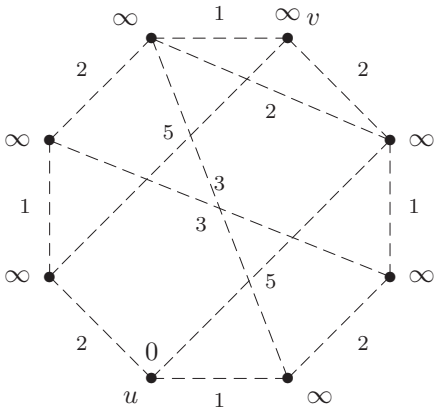
```
// stav[i] udává zpracovanost vrcholu, vzdal[i] zatím nalezenou vzdálenost
for (i=0; i<N; i++) { vzdal[i] = MAX; stav[i] = inic.; }
vzdal[u] = 0;□
while (stav[v]!=zprac.) {
    // zde nalezneme nejbližší nezpracovaný vrchol j
    m = argmin (vzdal[i]: 0≤i<N, stav[i]==inic.)
    if (vzdal[m]==MAX) return 'Cesta neexistuje';□
    // vrchol m poté zpracujeme, upravíme jeho sousedy
    foreach (k soused m)
        if (vzdal[m]+del[m,k]<vzdal[k]) {
            prich[k] = m;
            vzdal[k] = vzdal[m]+del[m,k];
        }
    stav[m] = zprac.;□
}
```

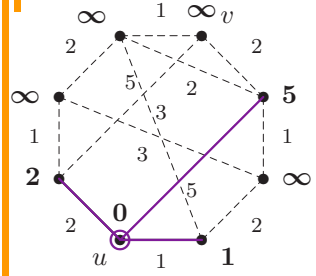
výstup 'Cesta délky vzdal[v], uložená zpětnými odkazy v poli prich[]';

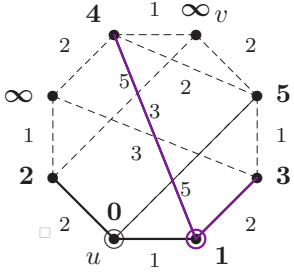
Poznámka: Uvědomme si, že pokud necháme tento algoritmus proběhnout až do zpracování všech vrcholů, získáme ve `vzdal[i]` nejkratší vzdálenosti z počátečního vrcholu do všech ostatních vrcholů i.

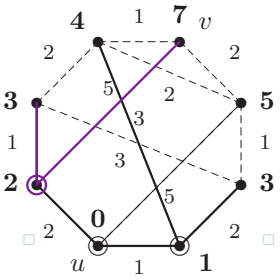
Všimněme si dále, že Algoritmus 3.12 počítá nejkratší cestu i v orientovaném grafu.

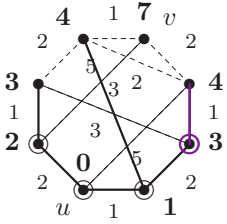
Příklad 3.15. *Ukázka běhu Dijkstrova Algoritmu 3.12 pro nalezení nejkratší cesty mezi vrcholy u, v v následujícím grafu.*

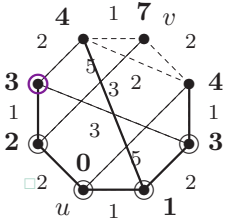


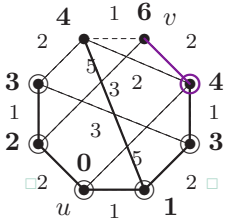

∞ 1 ∞ v
2 2
5 2
3
∞ 5
1 1
3
2 ∞
5
2 0 2
u 1 1

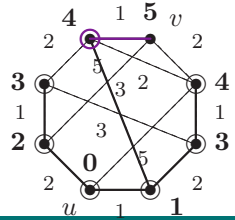

4 1 ∞ v
2 2
5 2
3
∞ 5
1 1
3
2 3
5
□ 2 0 2 □
u 1 1

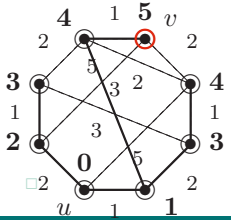

4 1 7 v
2 2
5 2
3
3 5
1 1
3
2 3
5
□ 2 0 2 □
u 1 1


4 1 7 v
2 2
3 2
3
3 4
1 1
3
2 3
5
2 0 2
u 1 1


4 1 7 v
2 2
3 2
3
3 4
1 1
3
2 3
5
□2 0 2
u 1 1


4 1 6 v
2 2
3 2
3
3 4
1 1
3
2 3
5
□2 0 2 □
u 1 1


4 1 5 v
2 2
3 2
3
3 4
1 1
3
2 3
5
2 0 2
u 1 1


4 1 5 v
2 2
3 2
3
3 4
1 1
3
2 3
5
□2 0 2
u 1 1

□

Fakt: Celkový počet kroků potřebný v Algoritmu 3.12 k nalezení nejkratší cesty z `u` do `v` je v základní „hloupé“ implementaci zhruba N^2, kde N je počet vrcholů grafu. □
Na druhou stranu, při lepší implementaci grafu seznamem sousedů a použití vhodné úschovny nezpracovaných vrcholů (implementované *haldou* s nalezenou vzdáleností jako klíčem) lze dosáhnout i mnohem rychlejšího běhu tohoto algoritmu na řídkých grafech – času téměř úměrného počtu hran grafu, přesněji $O\big(|E(G)| + N \log N\big)$. □

Věta 3.13. *V každé iteraci Algoritmu 3.12 (počínaje stavem po prvním průchodu cyklem* `while()` *) proměnná* `vzdal[i]` *udává nejkratší vzdálenost z vrcholu* `u` *do vrcholu* `i` *při cestě pouze po vnitřních vrcholech x, jejichž* `stav[x]==zprac`.□

Důkaz: Stručně matematickou indukcí:

- V prvním kroku algoritmu je jako vrchol ke zpracování vybrán první `j=u` a potom jsou jeho sousedům upraveny vzdálenosti od `u` podle délek hran z `u`. □
- V každém dalším kroku je vybrán jako vrchol `m` ke zpracování ten, který má ze všech nezpracovaných vrcholů nejkratší nalezenou vzdálenost od počátku `u`. To znamená, že žádná kratší cesta z `u` do `m` nevede, neboť každá „oklika“ přes jiné nezpracované vrcholy musí být delší dle výběru `m` a indukčního předpokladu.□

  Naopak každá nová nejkratší cesta z `u` do nezpracovaného vrcholu `k` procházející přes `m` musí mít `m` coby předposlední vrchol, tj. poslední hranu `mk`, a proto je upravená hodnota `vzdal[k]` správná i po přidání `m` mezi zpracované vrcholy.

□

3.5 Pokročilé plánování cest

- Třebaže je Dijkstrův algoritmus velmi rychlý a „téměř optimální“ mezi všemi způsoby hledání nejkratší cesty v nezáporně váženém grafu, stále zůstává příliš pomalým pro praktické nasazení třeba v přenosných navigačních zařízeních, jež obsahují mapová data v rozsahu desítek až stovek miliónů hran grafu. □
- Co však může být uděláno lépe? □
- Odpovědí je vhodné předzpracování grafu: Lze dobře očekávat, že samotný graf je relativně stabilní a pro jeho předzpracování je k dispozici dostatek času na výkonných strojích. □
- V tomto místě však vyvstává sekundární problém, kde uložit výsledky předzpracování? Určitě není reálné ukládat (např.) všechny nejkratší cesty mezi všemi dvojicemi vrcholů... □
- Dva z nejjednodušších úsporných přístupů k předzpracování grafu pro rychlé plánováni cest si nyní zběžně ukážeme.

Lépe než Dijkstrův algoritmus se chová vylepšený algoritmus A^*, který použitím vhodného potenciálu „směřuje“ celé prohledávání grafu ke správnému cíli a je skvěle použitelný ve všech situacích, kdy pojem „směr k cíli“ má význam. To je například při navigování v mapě.

Algoritmus A^*

- Je reimplementací Dijkstrova algoritmu s *„vhodně“ upravenými délkami* hran. □
- Nechť „*potenciál*“ $p_v(x)$ udává libovolný dolní odhad vzdálenosti z vrcholu x do cíle v. Například při navigaci v mapě může $p_v(x)$ udávat přímou (Euklidovskou) vzdálenost z bodu x do bodu v. □
- Každá (orientovaná!) hrana xy grafu (G, w) dostane nové délkové ohodnocení $\boldsymbol{w'(xy) = w(xy) + p_v(y) - p_v(x)}$. Potenciál p_v je *přípustný*, pokud všechna upravená ohodnocení jsou nezáporná, neboli $w(xy) \geq p_v(x) - p_v(y)$.

  Potenciál přímé vzdál. z x do cíle v je vždy přípustný podle trojúh. nerovnosti. □
- Upravená délka lib. sledu S z u do v pak je $d_G^{w'}(S) = d_G^w(S) + p_v(v) - p_v(u)$, což je konstantní rozdíl oproti původní délce S. Takže S je optimální pro původní délkové ohodnocení w, právě když je optimální pro nové w'.

  Dijkstrův algoritmus pro w' upravené potenciálem přímé vzdálenosti do v pak bude „silně preferovat“ hrany vedoucí ve směru k cíli v.

Myšlenka „dosahu“ (reach)

- Tento přístup těží z přirozeného poznatku, že valná většina hran reálné cestní sítě je pro globální plánování cest vpodstatě „bezvýznamná“.□

Definice: Nechť $S_{u,v}$ značí ve váženém grafu G optimální sled z u do v. Nechť dále pro $e \in E(S_{u,v})$ značí $prefix(S_{u,v}, e)$, $suffix(S_{u,v}, e)$ celý úsek sledu $S_{u,v}$ před (za) jeho hranou e. □ *Dosah hrany* $e \in E(G)$ je dán vztahem

$$reach_G(e) = \max \big\{ \min\big(d^w_G(prefix(S_{u,v}, e)), d^w_G(suffix(S_{u,v}, e)) \big) : \\ \forall u, v \in V(G) \wedge e \in E(S_{u,v}) \big\}.$$ □

Dosah hrany e přesně matematicky kvantifikuje „(bez)významnost“ e pro plánování cest; čím menší je $reach_G(e)$, tím blíže počátku nebo cíle opt. cesty musí hrana e být. Neboli, prakticky, hrany s malým $reach_G(e)$ lze pro většinu dotazů na optimální cesty ignorovat... □

Předpočítaného parametru dosahu hran lze přirozeně využít v *obousměrné variantě* Dijkstrova algoritmu (včetně A^*) ke vskutku radikální redukci počtu prohledávaných vrcholů při hledání nejkratší cesty:

- V řádku “`foreach (k soused m)`” (viz Algoritmus 3.12) prostě akceptujeme pouze ty sousedy `k` pro které platí $reach_G(\texttt{mk}) \geq \texttt{vzdal[m]}$.