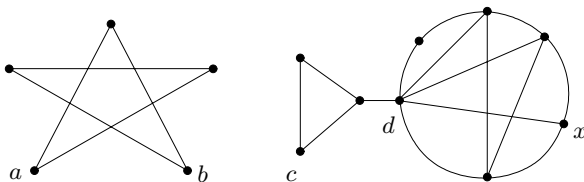


3 Vzdálenost a metrika v grafech

V minulé lekci jsme mluvili o souvislosti grafu, tj. o možnosti procházení z jednoho vrcholu do jiného. Někdy je prostá informace o souvislosti dostačující, ale většinou bychom rádi věděli i jak je to 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.
- Výpočet metriky grafu maticí.
- Dijkstrův algoritmus pro nejkratší (ohodnocenou) cestu v grafu.

3.1 Vzdálenost v grafu

Vzpomeňme si, že sledem délky n v grafu G rozumíme posloupnost vrcholů a hran $v_0, e_1, v_1, e_2, v_2, \dots, e_n, v_n$, 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 $d_G(u, v) = \infty$. \square

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.

Poznámka: V neorientovaném grafu je vzdálenost symetrická $d_G(u, v) = d_G(v, u)$. \square

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

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

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. \square

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 . □*

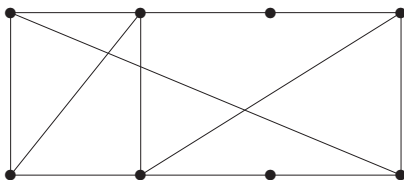
Důkaz. Postupujeme indukcí podle vzdálenosti $d_G(u, v)$: Pro $d_G(u, v) = 0$, tj. $u = v$ je tvrzení jasné – vrchol u jako počátek prohledávání byl nalezen první. Proto nechtě $d_G(u, v) = d > 0$ a označme v' souseda vrcholu v bližšího k u , tedy $d_G(u, v') = d - 1$. Stejně tak značme w' souseda vrcholu w bližšího k u , tedy $d_G(u, w') > d_G(u, v')$. Potom byl vrchol v' nalezen v prohledávání do šířky dříve než vrchol w' podle indukčního předpokladu. To znamená, že v' se dostal do fronty úschovny dříve než w' , a tudíž sousedé v' (mezi nimi v) budou při prohledávání nalezeni dříve než sousedé w' . □ □

Důsledek 3.4. *Základní algoritmus procházení grafu do šířky lze použít pro výpočet vzdálenosti z vrcholu u :*

Toto je poměrně jednoduchá aplikace, kdy počátečnímu vrcholu přiřadíme vzdálenost 0, a pak vždy každému dalšímu nalezenému vrcholu přiřadíme vzdálenost o 1 větší než byla vzdálenost vrcholu, ze kterého jsme jej našli. □

My si ale dále ukážeme obecnější Dijkstrův algoritmus, který počítá nejkratší vzdálenost při libovolně kladně ohodnocených délkách hran.

Další pojmy a fakta



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

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

Jedna zajímavost

Pro zajímavost (a zvědavé studenty) si uvedme následující vlastnost grafů. (Nebude třeba ke zkoušce.)

Definice: Graf je *vzdálenostně dědičný* (distance-hereditary), pokud vzdálenost každé dvojice jeho vrcholů u, v je stejná ve všech indukovaných souvislých podgrafech obsahujících u, v . □

Fakt. Grafy bez kružnic jsou vzdálenostně dědičné. □

Fakt. Následující konstrukce vytváří vzdálenostně dědičné grafy:

- Začneme z jediného vrcholu.
- Mějme dva grafy G_1, G_2 získané touto konstrukcí (rekurzivně). Vytvoříme jejich disjunktní sjednocení $G_1 \dot{\cup} G_2$.
- Mějme dva grafy G_1, G_2 získané touto konstrukcí (rekurzivně). Vytvoříme jejich disjunktní sjednocení $G_1 \dot{\cup} G_2$ a přidáme všechny hrany mezi $V(G_1)$ a $V(G_2)$. □

(Najděte vzdálenostně dědičný graf, který nelze takto sestavit. . .)

Věta 3.5. *Graf G je vzdálenostně dědičný právě když každá kružnice délky alespoň 5 má v G dvě „zkřížené“ tětivy.*

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. Iterativní výpočet metriky skládáním cest

- Na počátku nechť $d[i, j]$ udává 1 (případně délku hrany $\{i, j\}$), nebo ∞ pokud hrana mezi i, j není. □
- Po každém kroku $t \geq 0$ nechť $d[i, j]$ udává délku nejkratší cesty mezi i, j , která jde pouze přes vnitřní vrcholy z množiny $\{0, 1, 2, \dots, t - 1\}$. □
- Při přechodu z t na následující krok $t + 1$ upravujeme vzdálenost pro každou dvojici vrcholů – jsou vždy vždy pouze dvě možnosti:
 - Buď je cesta délky $d[i, j]$ z předchozího kroku stále nejlepší (tj. nově povolený vrchol t nám nepomůže),
 - **nebo** cestu vylepšíme spojením přes nově povolený vrchol t , čímž získáme menší vzdálenost $d[i, t] + d[t, j]$.

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

Algoritmus 3.7. Výpočet metriky grafu

vstup: *matice sousednosti* `G[] []` grafu na `N` vrcholech,
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]);
}
```

Poznámka: Algoritmus 3.7 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... □

Věta 3.8. Algoritmus 3.7 v poli `d[i, j]` správně vypočte vzdálenost mezi vrcholy `i, j`.

3.3 Vážená (ohodnocená) vzdálenost

Definice 3.9. **Vážený graf** je graf G spolu s ohodnocením w hran reálnými čísly $w : E(G) \rightarrow \mathbf{R}$.

Kladně vážený graf G, w je takový, že $w(e) > 0$ pro všechny hrany e . \square

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

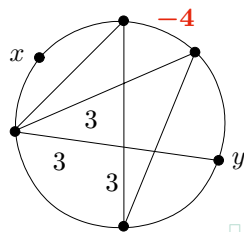
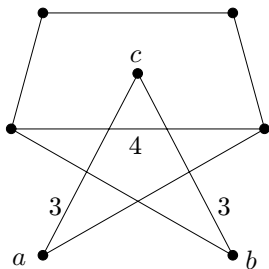
$$d_G^w(S) = w(e_1) + w(e_2) + \dots + w(e_n).$$

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

$$d_G^w(u, v) = \min\{d_G^w(S) : S \text{ je sled s konci } u, v\} . \square$$

Lema 3.10. *Vážená vzdálenost v kladně vážených grafech také splňuje **trojúhelníkovou nerovnost**.*

Podívejme se na následující graf vlevo. (Čísla u hran udávají jejich váhy, hrany bez čísel mají váhu 1.) Vážená vzdálenost mezi vrcholy a, c je 3, mezi b, c také 3. Jaká je vzdálenost mezi vrcholy a, b ? Ne, není to 6, ale najdeme kratší cestu délky 5.



V druhém příkladě vpravo je uvedena i hrana se zápornou vahou -4 . Nejkratší cesta mezi vrcholy x, y tak má délku -2 , ale pokud vezmeme sled, který hranu váhy -4 vícekrát zopakuje, dostaneme se na libovolně nízkou zápornou délku. To je samozřejmě nesmyslné, a proto se takovému problému radši vyhýbáme **zákazem záporných vah hran**.

3.4 Hledání nejkratší cesty

Pro nalezení nejkratší (vážené) cesty mezi dvěma vrcholy kladně váženého grafu se používá známý tzv. *Dijkstrův algoritmus*.

Poznámka: Dijkstrův algoritmus je sice složitější než Algoritmus 3.7, ale na druhou stranu je *výrazně rychlejší*, pokud nás zajímá jen nejkratší vzdálenost z jednoho vrcholu místo všech dvojic vrcholů.

Zrovna tento algoritmus se například používá při vyhledávání vlakových či autobusových spojení. Pravděpodobně se i vy někdy dostanete do situace, kdy budete nejkratší cestu hledat, proto si tento algoritmus zapamatujte. □

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 3.11. 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 seznamem sousedů $sous[] []$ a $del[] []$,
kde $sous[i][0], \dots, sous[i][st[i]-1]$ jsou sousedé vrcholu i stupně $st[i]$
a hrana z i do $sous[i][k]$ má délku $del[i][k] > 0$;

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_INT; stav[i] = 0; }
```

```
vzdal[u] = 0; □
```

```
while (stav[v]==0) {
```

```
    for (i=0, j=N; i<N; i++)
```

```
        if (stav[i]==0 && vzdal[i]<vzdal[j]) j = i;
```

```
    // zde jsme našli nejbližší nezpracovaný vrchol j, ten teď zpracujeme
```

```
    if (vzdal[j]==MAX_INT) return NENI_CESTA;
```

```
    stav[j] = 1; □
```

```
    for (k=0; k<st[j]; k++)
```

```
        if (vzdal[j]+del[j][k]<vzdal[sous[j][k]]) {
```

```
            pri[sous[j][k]] = j;
```

```
            vzdal[sous[j][k]] = vzdal[j]+del[j][k];
```

```
        }
```

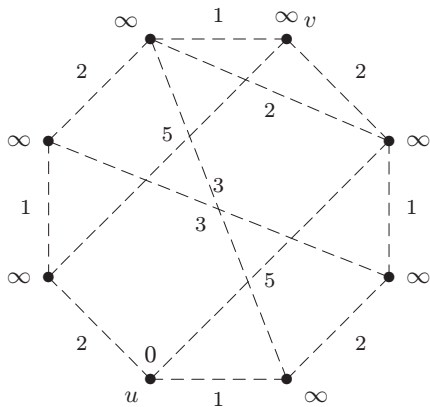
```
    }
```

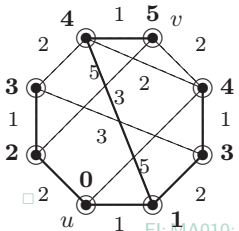
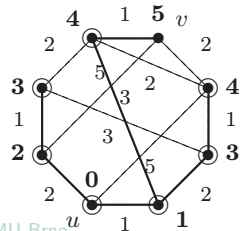
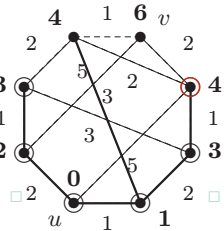
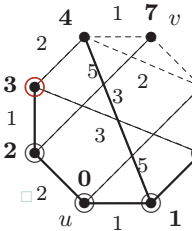
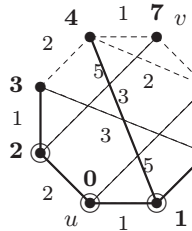
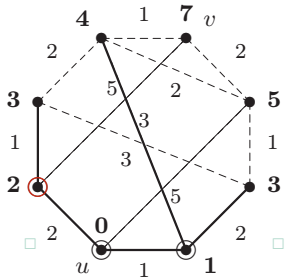
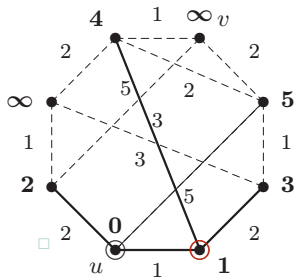
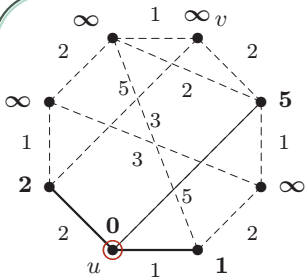
výstup 'Cesta délky $vzdal[v]$, uložená "pozpátku" v poli $pri[]$ ';

Poznámka: Uvědomme si, že pokud necháme tento algoritmus proběhnout až do zpracování všech vrcholů, získáme ve `vzda1[i]` nejkratší vzdálenosti z počátečního vrcholu do všech ostatních vrcholů.

Všimněme si dále, že Algoritmus 3.11 počítá stejně dobře nejkratší cestu i v **orientovaném grafu**.

Příklad 3.12. Ukázka běhu Dijkstrova Algoritmu 3.11 pro nalezení nejkratší cesty mezi vrcholy u, v v následujícím grafu.





Důkaz správnosti Algoritmu 3.11.

Klíčovým faktem k důkazu správnosti algoritmu je to, že v každém kroku (počínaje stavem po prvním průchodu cyklem `while()`) proměnná `vzda1[i]` udává nejkratší vzdálenost z vrcholu `u` do vrcholu `i` při cestě pouze po vnitřních vrcholech `x`, jejichž `stav[x]==1` (po zpracovaných vrcholech). □ 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 j ke zpracování ten, který má ve všech nezpracovaných vrcholech nejkratší nalezenou vzdálenost od počátku u . To ale znamená, že žádná kratší cesta už do j nevede, neboť každá „oklika“ přes jiné nezpracované vrcholy musí být delší dle výběru j . (V tomto bodě potřebujeme **nezápornost ohodnocení** `del[][]`.) Naopak můžeme podle délek hran vycházejících z j „vylepšit“ cesty do jeho sousedů. □

Závěrem zbývá ověřit, že nalezená nejkratší cesta z u do v je pozpátku uložená v poli `pri[]`, tj. předposlední vrchol před v je `pri[v]`, předtím `pri[pri[v]]`, atd... □ □

Fakt: Celkový počet kroků potřebný v Algoritmu 3.11 k nalezení nejkratší cesty z u do v je zhruba N^2 , kde N je počet vrcholů grafu. □

Na druhou stranu, při lepší implementaci úschovny nezpracovaných vrcholů (třeba *hal-dou* s nalezenou vzdáleností jako klíčem) lze dosáhnout i mnohem rychlejšího běhu tohoto algoritmu na řídkých grafech – času zhruba úměrného počtu hran grafu.