Jakýsi úvod do diskrétní matematiky

Áďa Klepáčovic

10. května 2023

Obsah

1 | Teorie grafů

Velkou část moderní matematiky (zcela jistě topologii, geometrii i algebru) tvoří studium „struktur“. Toto obecně nedefinované slovo obvykle značí množinu s nějakou další informací o vztahu mezi jejími prvky – tím obvykle bývá operace nebo třeba, jako v případě grafů, relace.

Tato kapitola zároveň značí jakýsi milník ve vývoji matematického myšlení, především algebraickým směrem. Můžeme se totiž začít bavit o speciálních zobrazeních, které zachovávají strukturu na množinách, mezi kterými vedou, tzv. *homomorfismech*; pochopit, že je dobré mít více popisů stejné struktury ekvivalentních v tom smyslu, že poskytují stejné množství informací, přestože se o žádné bijekci nedá formálně hovořit; uvidět, že je užitečné dva různé grafy (či obecně dvě různé struktury) považovat za stejné, když se liší pouze zanedbatelně.

Jednou, avšak zdaleka ne *jedinou*, motivací pro teorii grafů je schopnost analyzovat struktury tvořené množinou „uzlů“, mezi některými vedou „spojnice“. Takováto struktura úspěšně modeluje až neuvěřitelné množství přírodních i společenských úkazů. Mezi nimi jmenujmež

- návrhy elektrických obvodů, kde uzly jsou elektrická zařízení a spojnice jsou kabely mezi nimi vedoucí;
- lingvistické modely, kde uzly jsou slova a spojnice vede mezi těmi syntakticky souvisejícími;
- studium molekul, kde uzly jsou atomy a spojnice vazby mezi nimi;
- analýza šíření fámy v sociologii, kde uzly jsou lidské komunity a spojnice vede mezi komunitami s bezprostředním kontaktem.

Snad pro to, že uzly a spojnice grafu se obvykle kreslí jako body a úsečky v prostoru, ujaly se pro ně názvy *vrcholy* a *hrany* (jako v mnohoúhelnících), respektive. Struktura zvaná *graf* tedy sestává ze dvou údajů:

(1) množiny (obvykle konečné) vrcholů značené V a

(2) množiny hran E, která je spjata s množinou vrcholů; toto „sepětí“ se však definuje různě, v závislosti na vkusu a aplikaci. My si ukážeme tři z jistě většího množství různých definic.

Asi prvním přirozeným kandidátem pro „strukturu“ je množina s relací. To je také první způsob, jak si budeme definovat pojem *graf*. Je to také ten nejobecnější v tom smyslu, že jeho pouze drobné modifikace nám umožní definovat i obdobné struktury, jež také zmateně slují *grafy*, byť s připojeným atributem.

Abychom úsečky mezi body mohli popsat jako relaci, čili pomocí uspořádaných dvojic bodů, zcela jistě budeme požadovat, aby nevedly úsečky z bodu do něho samého. Úsečku délky 0 lze totiž triviálně ztotožnit s bodem. Dále, úsečka z bodu A do bodu B je jistě tatáž, kterak úsečka z bodu B do bodu A. Tento fakt musí rovněž relace E odrážet.

První vlastností relace se, snad nepřekvapivě, říká *antireflexivita*. Čili, relace E na V je *antireflexivní*, když hrana $(v,v) \notin E$ pro každý vrchol $v \in V$.

Druhou vlastnost už jsme potkali a nazvali ji symetrií. Požadujeme, aby s hranou $(v,w) \in E$ obsahovala E též hranu $(w,v) \in E$ pro každé dva vrcholy $v,w \in V$.

Definice 1.0.1 (Graf poprvé)**.** Dvojici $G := (V,E)$, kde V je konečná množina a E je relace na V, nazveme *grafem*, pokud je E **antireflexivní** a **symetrická**.

Poznámka. Z hlediska ryze formálního neodpovídá tato definice dokonale naší geometrické představě. My jsme totiž pouze požadovali, aby E obsahovala jak úsečku z v do w, tak úsečku z w do v, ale nikoli, aby se jednalo o *tutéž* úsečku. Tedy, jedna úsečka mezi body je v množině E reprezentována dvěma dvojicemi.

Nápravou by bylo definovat navíc ještě relaci R na E, kde (v,v') je v relaci R s (w,w') právě tehdy, když $(w,w') = (v',v)$ nebo $(w,w') = (v,v')$. Jinak řečeno, úsečka z bodu v do bodu v' je v relaci sama se sebou a s úsečkou z bodu v' do bodu v.

Uvážíme-li pak jako hrany grafu G nikoli množinu E, ale její třídy ekvivalence podle R (**Ověřte, že R je ekvivalence!**), dostaneme již přesnou množinovou paralelu bodů a úseček.

My však budeme v zájmu přehlednosti tento nedostatek ignorovat, protože není pro pochopení ani rozvoj teorie relevantní.

Cesta k druhé možné definici grafu není od první daleká. Stačí vlastně relaci E interpretovat trochu jinak. Přece, antireflexivní a symetrická relace je „totéž“ jako množina dvouprvkových podmnožin V.

Vskutku, vezměme nějakou $E' \subseteq \binom{V}{2}$. Relaci E z definice 1.0.1 sestrojíme tak, že z množiny $\{v,w\} \in E'$ vyrobíme dvojice (v,w) a (w,v). Protože prvky v množině nejsou uspořádané a nemohou se opakovat, dává tato konstrukce opravdu antireflexivní a symetrickou relaci. Vizuálně odpovídá rozdělení úsečky mezi v a w na šipku z v do w a šipku z w do v.

Z druhé strany, mějme nějakou antireflexivní a symetrickou relaci E na V. Protože s dvojicí (v,w) obsahuje E i dvojici (w,v), můžeme z těchto dvojic ztvárnit množinu $\{v,w\}$. Relace E je antireflexivní, čili se nemůže stát, že $v = w$, a množina $\{v,w\}$ je pročež vždy dvouprvková. Posbíráme-li všechny množiny $\{v,w\}$ do jedné velké množiny E', bude platit $E' \subseteq \binom{V}{2}$. Vizuálně odpovídá tato konstrukce slepení šipky z v do w a šipky z w do v do jedné úsečky mezi v a w.

Výstraha. Mezi množinami E a E' **nemůže existovat bijekce**, třeba jen pro to, že $\#E = 2\#E'$. Co konstrukce v předchozích dvou odstavcích ukazují, je pouze fakt, že pro naše účely definují E a E' stejnou strukturu na množině V.

Ovšem, uvážili-li bychom místo E pouze třídy ekvivalence jejích prvků podle relace R popsané v poznámce pod definicí 1.0.1, pak bychom skutečně tímto způsobem našli bijekci s množinou E'.

Definice 1.0.2 (Graf podruhé)**.** Dvojici $G := (V, E')$, kde V je konečná množina a $E' \subseteq \binom{V}{2}$, nazveme *grafem*.

Třetí pohled na hrany v grafu je více „kategoriální“. Zatímco množiny E a E' jsou závislé ve své definici na množině V, třetí množina hran E'', kterou si zde definujeme, bude libovolná konečná množina.

Tento popis grafové struktury bude odpovídat trochu jiné představě; konkrétně takové, kdy začínáme s množinou bodů V a s množinou šipek E (jež jsou od sebe naprosto odděleny) a oba konce každé šipky zapíchneme do dvou různých bodů z V. Toto „zapíchnutí“ realizují zobrazení $s, t : E'' \to V$ (z angl. **s**ource a **t**arget), která zobrazují šipky z E'' do bodů z V. Přičemž

budeme trvat na tom, aby $s(e) \neq t(e)$ pro všechny šipky $e \in E''$ a navíc, aby pro každou $e \in E''$ existovala šipka $e' \in E''$ taková, že $s(e) = t(e')$, $t(e) = s(e')$. Lidsky řečeno, nesmíme zapíchnout konce šipky do téhož vrcholu a, když zapíchneme začátek šipky do bodu v a její konec do bodu w, pak musíme vzít další šipku, jejíž začátek zapíchneme do w a konec do v.

Je snadné si rozmyslet, že z množiny šipek E'' zrekonstruujeme množinu E z definice 1.0.1 tak, že z šipky $e \in E''$ vytvoříme dvojici $(s(e), t(e)) \in E$. Podmínky kladené na zobrazení s a t zaručují, že vzniklá množina E je relace na V, která je antireflexivní a symetrická. V tomto případě dává uvedená konstrukce dokonce bijekci $E \cong E''$, čili jsme opět definovali tutéž strukturu na V.

Tato struktura bude zvlášť užitečná, až budeme probírat *toky v síti*.

Definice 1.0.3 (Graf potřetí)**.** Čtveřici $G := (V, E'', s, t)$, kde V a E'' jsou konečné množiny a s a t jsou zobrazení $E'' \to V$ nazveme *grafem*, pokud

(a) $s(e) \neq t(e)\ \forall e \in E''$ a

(b) $\forall e \in E''\ \exists e' \in E'' : s(e) = t(e') \wedge t(e) = s(e')$.

Poznámka. Opět, aby definice 1.0.3 odpovídala představě bodů a úseček (nebo oboustranných šipek), museli bychom definovat relaci R na E tak, aby e a e' byly v R, právě když $s(e) = s(e')$ a $t(e) = t(e')$ nebo $s(e) = t(e')$ a $t(e) = s(e')$. V takovém případě bychom mohli sestrojit bijekci mezi E'' a E' z definice 1.0.2.

Příklad. Ať $V := \{1, 2, 3, 4, 5\}$ a

(1) $E := \{(1,2),(2,1),(1,3),(3,1),(1,4),(4,1),(1,5),(5,1),(2,3),(3,2),(4,5),(5,4)\}$;

(2) $E' := \{\{1,2\},\{1,3\},\{1,4\},\{1,5\},\{2,3\},\{4,5\}\}$;

(3) $E'' := \{e_1, e_2, e_3, e_4, e_5, e_6, e_1', e_2', e_3', e_4', e_5', e_6'\}$, kde

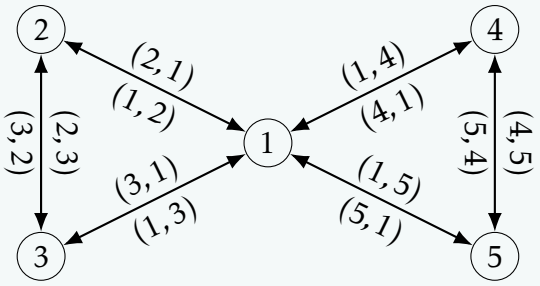
   (a) $s(e_1) = s(e_2) = s(e_3) = s(e_4) = 1$, $s(e_5) = 2$, $s(e_6) = 4$,

(b) $t(e_1) = 2$, $t(e_2) = t(e_5) = 3$, $t(e_3) = 4$, $t(e_4) = t(e_6) = 5$ a

(c) $(s(e_i), t(e_i)) = (t(e'_i), s(e'_i))$ pro všechna $i \leq 6$.

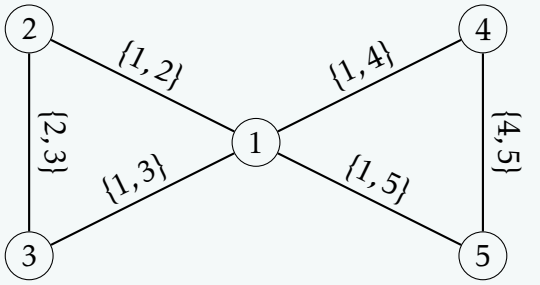
Není těžké nahlédnout, že E, E' i (E'', s, t) definují tutéž strukturu na V. Nakreslíme si grafy $G = (V, E), G' = (V, E')$ a $G'' = (V, E'', s, t)$. Přičemž hrany z E budeme kreslit jako oboustranné šipky, ty z E' jako prosté úsečky a ty z E'' rozdělíme na dvě protichůdné šipky, abychom vyjádřili rozdíly v interpretaci těchto grafových struktur.

Graf $G = (V, E)$ vypadá například takto. Pomněte, že například oboustranná šipka mezi vrcholy 1 a 2 představuje ve skutečnosti **dvě** dvojice – $(1, 2)$ a $(2, 1)$ z relace E.



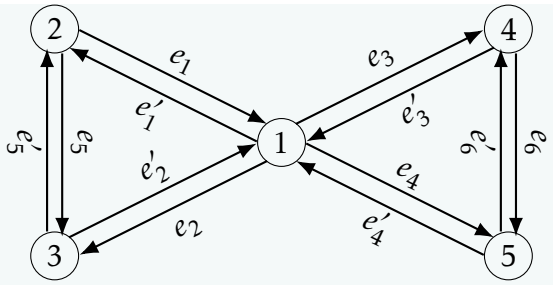
Obrázek 1: Graf jako množina V s relací E.

Zcela stejně vypadá i graf $G' = (V, E')$.



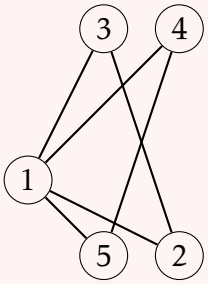
Obrázek 2: Graf jako množiny V a $E' \subseteq \binom{V}{2}$.

Konečně, $G'' = (V, E'', s, t)$ můžeme načrtnout taktéž velmi podobně.



Obrázek 3: Graf jako množina V s trojicí (E'', s, t).

Výstraha. Grafová struktura je obecně zcela nezávislá na jejím nakreslení. Například graf $G = (V, E')$ z předchozího příkladu lze ekvivalentně vyobrazit třeba následovně.



Obrázek 4: Graf $G = (V, E')$ nakreslený jinak.

V následujícím textu spojíme všechny tři interpretace dohromady a pro $v, w \in V$ budeme hranu mezi v a w značit zjednodušeně jako vw. Pokud nehrozí nedorozumění, budeme pod tímto zápisem rozumět hranu, jejíž začátek je v a konec w, čili $s(vw) = v$ a $t(vw) = w$. Avšak, kdykoli se nám to bude hodit, ztotožníme ji bez okolků s hranou wv s obrácenými konci.

Tento neformální přístup k popisu hran se může zdát jako nebezpečný, ale jak uvidíme, ve skutečnosti velmi zjednodušuje zápis a újma na rigorozitě je obecně minimální. Kompletněji řečeno, hranou mezi dvěma vrcholy $v, w \in V$ myslíme buď dvojici $(v, w) \in E$ nebo dvojici $(w, v) \in E$ nebo množinu $\{v, w\} \in E'$ nebo prvek $e \in E''$ takový, že $s(e) = v$ a $t(e) = w$, nebo prvek $e' \in E''$ takový, že $s(e') = w$ a $t(e') = v$, a je nám to u ...

Obecně, v teorii grafů se velmi často pracuje s konečnými posloupnostmi (či n-ticemi, chcete-li) vrcholů a hran. Zavedeme proto zjednodušené zna-

čení $x_1x_2\cdots x_n$ pro uspořádanou n-tici $(x_1,\dots,x_n)$. Kdyby hrozil konflikt se zápisem součinu prvků $x_1,\dots,x_n$, samozřejmě tento úzus dočasně opustíme.

Cvičení 1.0.1. Nakreslete graf $G=(V,E)$, kde

- $V=\{1,\dots,5\}, E=\{\{1,2\},\{1,3\},\{1,5\},\{2,3\},\{3,4\},\{4,5\}\}$.
- $V=\{1,\dots,5\}$, $E=\binom{V}{2}$.
- $V=\{1,\dots,8\}$, $E=\{e_1,\dots,e_8\}$ a
  - $t(e_i)=s(e_{i+1})=i+1$ pro všechna $i\leq 7$,
  - $t(e_8)=s(e_1)=1$.

Cvičení 1.0.2. Popište všechny grafy $G=(V,E)$, kde E je relace na V, která je antireflexivní, symetrická (to je součástí definice grafu) a navíc **transitivní**.

Cvičení 1.0.3. Ať V je konečná množina a E je relace na V, která je antireflexivní a symetrická. Definujme navíc na E další relaci $\sim$ předpisem

$$(v,v')\sim(w,w')\Leftrightarrow(v,v')=(w,w')\vee(v,v')=(w',w).$$

Dokažte, že pak existuje bijekce mezi $[E]_\sim$ a množinou

$$E':=\{\{v,v'\}\mid(v,v')\in E\},$$

čili mezi množinou tříd ekvivalence E podle $\sim$ a množinou, kterou dostanu tak, že z uspořádaných dvojic v E udělám neuspořádané dvojice, tj. dvouprvkové podmnožiny. Pro intuici vizte poznámku pod definicí 1.0.1.

Cvičení 1.0.4. Spočtěte, kolik existuje grafů na n vrcholech.

1.1 Pohyb v grafu

Jak jsme zmínili na začátku kapitoly, mnoho aplikací teorie grafů využívá interpretace této struktury jako množiny „uzlů“ se „spojnicemi“, po kterých se dá mezi uzly pohybovat. V této podsekci dáme pohybu po spojnicích formální tvář.

Nejzákladnějším typem pohybu v grafu je libovolná posloupnost vrcholů se zcela přirozenou podmínkou, že mezi vrcholy v posloupnosti bezprostředně za sebou musí vést hrana. Takové posloupnosti se říká *sled* (angl. *walk*).

Definice 1.1.1 (Sled poprvé). Ať $G = (V, E)$ je graf. Posloupnost vrcholů $v_1 v_2 \cdots v_n$ nazveme *sledem* v grafu G, pokud $v_i v_{i+1} \in E$ pro každý index $i \in \{1, \ldots, n-1\}$.

Je však užitečné si uvědomit, že každý sled lze ekvivalentně definovat jako posloupnost navazujících hran. Tento pohled (jak uvidíme později v kapitole) má své aplikace – často nás totiž zajímá, po kterých spojnicích chodíme, spíše než které uzly procházíme.

Máme-li sled $v_1 v_2 \cdots v_n$, tak každá dvojice $v_i v_{i+1}$ musí být hranou v G. To ovšem znamená, že zcela identickou informaci poskytuje i posloupnost hran $e_1 e_2 \cdots e_{n-1}$, kde $e_i = v_i v_{i+1}$.

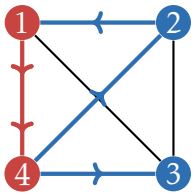
Poznámka. Tento princip, který prostupuje matematické struktury, je až překvapivě zásadního významu. Zobecníme-li trochu předchozí pozorování, uvědomíme si, že množina hran vlastně už v sobě obsahuje informaci o všech prvcích množiny V. Tedy bychom dokonce mohli definovat graf pouze jako množinu hran a množinu vrcholů bychom takto získali automaticky. Opak samozřejmě není pravdou.

Situace, kdy struktura na množině (či něčem složitějším) je dostatečně „hustá“, aby obsahovala kompletní informaci o této množině, je obecně velmi žádaná, jelikož struktura je často konstruována systematicky (a tedy jí rozumíme lépe) ve srovnání s náhodnou volbou její bázové množiny.

Pro nedostatek představivosti uvedeme příklad z homologické alge-

bry, kde injektivní a projektivní rezolventy modulů obsahují již kompletní informaci o daném modulu. Tedy člověku pro pochopení teorie modulů z homologického hlediska „stačí“ studovat injektivní a projektivní moduly, které jsou z definice více omezené než moduly obecné, pročež snadněji popsatelné.

Definice 1.1.2 (Sled podruhé)**.** Ať $G = (V, E)$ je graf. Posloupnost hran $e_1 e_2 \cdots e_n$ nazveme *sledem* v grafu G, pokud $t(e_i) = s(e_{i+1})$ pro všechna $i \in \{1, 2, \ldots, n-1\}$.



Obrázek 5: Příklad sledu 142143 v grafu. Jednou navštívené hrany a vrcholy jsou značené modře, dvakrát navštívené červeně.

Méně obecným pohybem v grafu je sled, ve kterém se nesmějí opakovat hrany. Takové sledy lze najít často třeba v běžné situaci, kdy si jako rozvůzce jídla plánujete cestu městem. Jako uzly si označíte například místa, která musíte objet, a hrany budou nejkratší trasy mezi nimi. Projet přes některá místa vícekrát vám příliš vadit nemusí, ale ztrácet čas cestováním po stejné trase sem a tam byste neradi.

Takovému sledu se říká *tah* (angl. *trail*). Jeho definice je velmi přirozená.

Definice 1.1.3 (Tah)**.** Ať $G = (V, E)$. *Tahem* v grafu G nazveme buď

(1) sled (vrcholů) $v_1 v_2 \cdots v_n$ takový, že $v_i v_{i+1} \neq v_j v_{j+1}$ pro všechna $i \neq j$, nebo

(2) sled (hran) $e_1 e_2 \cdots e_{n-1}$ takový, že $e_i \neq e_j$ pro všechna $i \neq j$.

Výstraha. Obě uvedené definice tahu jsou v našem (částečně neformálním) pojetí hran skutečně ekvivalentní. Napíšeme-li totiž $v_i v_{i+1} \neq$

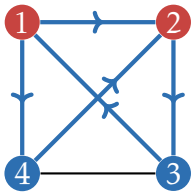
$v_j v_{j+1}$ myslíme tím vlastně dvě nerovnosti:

$$(v_i, v_{i+1}) \neq (v_j, v_{j+1}) \wedge (v_i, v_{i+1}) \neq (v_{j+1}, v_j);$$

nebo zápis můžeme též chápat jako

$$\{v_i, v_{i+1}\} \neq \{v_j, v_{j+1}\},$$

čili každou hranu chceme projít (kterýmkoli směrem) maximálně jednou.



Obrázek 6: Příklad tahu 142312 v grafu. Jednou navštívené hrany a vrcholy jsou značené modře, dvakrát navštívené červeně.

Posledním, a asi nejčastěji zkoumaným, typem pohybu grafem je *cesta* (angl. *path*), což je sled, ve kterém se nesmějí opakovat ani hrany ani vrcholy. Pro jednoduchost je užitečné si uvědomit, že z podmínky neopakování vrcholů automaticky plyne podmínka neopakování hran. Pokud bychom totiž chtěli po nějaké hraně přejít dvakrát, tak zároveň dvakrát projdeme její koncové vrcholy. Definovat cestu pomocí sledu vrcholů je tudíž přímočaré.

U sledu hran je to však horší. Protože každá hrana definuje dva vrcholy sledu, je třeba zařídit, aby se oba koncové body každé hrany lišily od obou koncových bodů jiné hrany, ale jenom tehdy, když hrany nejdou bezprostředně za sebou. To bohužel vede na trochu neintuitivní definici cesty přes hrany. Totiž, cesta obsahující n hran nutně obsahuje $n+1$ vrcholů, protože po sobě jdoucí hrany vždy sdílejí jeden vrchol. Rafinovaně se tedy cesta přes hrany dá vyjádřit jako tah, kde sjednocení přes všechny hrany (vnímané tentokrát jako množiny) má velikost právě $n+1$.

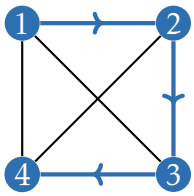
Studium cest má aplikace například právě v návrhu elektrických obvodů, kdy zcela jistě nechcete, aby do připojených zařízení šel proud z více, než jednoho místa.

Definice 1.1.4 (Cesta). Ať $G = (V, E)$ je graf. *Cestou* v grafu G nazveme buď

(1) sled (vrcholů) $v_1 v_2 \cdots v_n$ takový, že $v_i \neq v_j$ pro všechna $i \neq j$, nebo

(2) tah (hran) $e_1 e_2 \cdots e_{n-1}$ takový, že

$$\#\bigcup_{i=1}^{n-1} e_i = n.$$



Obrázek 7: Příklad cesty 1234 v grafu. Navštívené hrany a vrcholy jsou značené modře.

Posledním důležitým konceptem v grafu je tzv. *cyklus* (též *kružnice*, angl. *cycle*). Jedná se vlastně o „téměř cestu“, která končí tam, kde začala. Konkrétně je to tedy cesta prodloužená o svůj první vrchol (samozřejmě automaticky předpokládáme, že existuje hrana z posledního vrcholu do prvního).

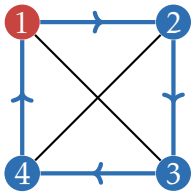
Definice 1.1.5 (Cyklus). Ať $G = (V, E)$ je graf. *Cyklem* v grafu nazveme buď

(1) sled (vrcholů) $v_1 v_2 \cdots v_n v_1$, pokud $v_1 v_2 \cdots v_n$ je cesta v G, nebo

(2) tah (hran) $e_1 e_2 \cdots e_{n-1} e_n$, kde $t(e_n) = s(e_1)$ a $e_1 e_2 \cdots e_{n-1}$ je cesta v G.

Příklad cyklu vidíte na obrázku 8. Jejich význam zatím necháme zahalen tajemstvím, jež má být odkryto v nejvíce dramatickou chvíli.

Cvičení 1.1.1. Ať $\mathcal{P}_1 = e_1^1 e_2^1 \cdots e_n^1$ a $\mathcal{P}_2 = e_1^2 e_2^2 \cdots e_n^2$ jsou cesty. Za předpokladu, že $t(e_n^1) = s(e_1^2)$, definujeme jejich *sloučení*, které zapíšeme třeba



Obrázek 8: Příklad cyklu 12341 v grafu. Jednou navštívené hrany a vrcholy jsou značené modře, dvakrát navštívené červeně.

jako $\mathcal{P}_1 \oplus \mathcal{P}_2$, přirozeně jako posloupnost hran

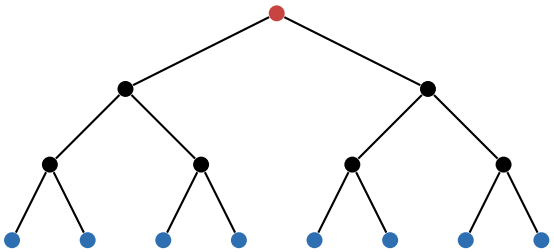
$$\mathcal{P}_1 \oplus \mathcal{P}_2 := e_1^1 e_2^1 \cdots e_n^1 e_1^2 e_2^2 \cdots e_n^2.$$

Určete, pro jaké cesty (v obecném grafu) $\mathcal{P}_1, \mathcal{P}_2$ platí, že

- $\mathcal{P}_1 \oplus \mathcal{P}_2 = \mathcal{P}_2 \oplus \mathcal{P}_1$;
- je $\mathcal{P}_1 \oplus \mathcal{P}_2$ cesta;
- je $\mathcal{P}_1 \oplus \mathcal{P}_2$ tah;
- je $\mathcal{P}_1 \oplus \mathcal{P}_2$ sled;
- je $\mathcal{P}_1 \oplus \mathcal{P}_2$ cyklus.

1.2 Stromy

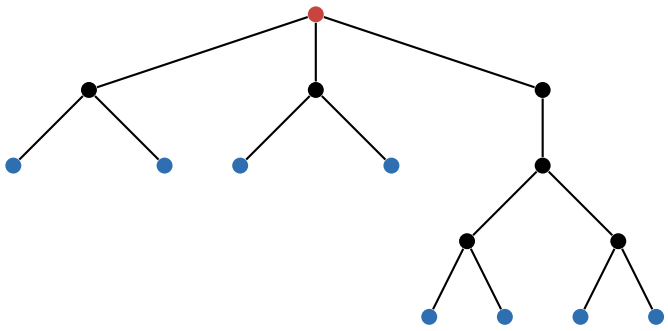
Chvíli se budeme bavit o stromech – ano, těch s listy a kořenem. Stromy jsou speciální typy grafů, které se takto nazývají ne nadarmo. Jsou to totiž grafy, u kterých si člověk může zvolit jakýsi „počáteční“ vrchol (zvaný *kořen*), z nějž se po cestě (ve smyslu definice 1.1.4) vždy dostane do jednoho z „koncových“ vrcholů, tzv. *listů*. Příklad stromu je na obrázku 9.



Obrázek 9: Příklad stromu. Kořen je značen červeně a listy modře.

Výstraha. Listy stromu jsou určeny jednoznačně jeho strukturou (jsou to ty jediné vrcholy, do nichž cesty od kořene mohou pouze vést a nikoli jimi procházet). Za kořen lze však volit libovolný vrchol, klidně i jeden z listů. Strom z obrázku 9 může proto vypadat i jak ukazuje obrázek 10.

Často není nutno o kořenu a listech stromu hovořit, pokud je v konkrétní situaci irelevantní rozlišovat jednotlivé vrcholy. Součástí definice stromu (kterou si záhy odvodíme) kořen ani listy nejsou.



Obrázek 10: Strom z obrázku 9 s jinou volbou kořene.

Nyní si rozmyslíme dvě ekvivalentní definice stromu.

Za první podmínku, abychom mohli graf nazvat stromem, budeme považovat fakt, že od kořene se dá dostat po hranách do každého z listů. Ekvivalentně, že z každého vrcholu se dá cestou dostat do každého vrcholu, protože za kořen lze, jak jsme nahlédli, volit kterýkoli vrchol, a cestu z kořene do vrcholu můžeme zkrátit tak, aby končila v nějakém vrcholu, jímž původně procházela. Grafy splňující tuto podmínku slují *souvislé*.

Definice 1.2.1 (Souvislý graf)**.** Graf $G = (V, E)$ nazveme *souvislým*, pokud pro každé dva vrcholy $v, w \in V$ existuje cesta z v do w, tedy cesta $v_1 v_2 \cdots v_n$, kde $v_1 = v$ a $v_n = w$.

Samotný název „strom“ plyne z faktu, že se jako graf pouze „větví“, čímž míníme, že při cestě směrem od (libovolného) kořene se jeden může pouze přibližovat k listům, ale nikoli se dostat zpět blíže ke kořeni. To lze snadno zařídit tak, že zakážeme cykly. Totiž, neexistuje-li v grafu cyklus, pak se po libovolné cestě ze zvoleného vrcholu můžeme od tohoto vrcholu pouze vzdalovat. Takové grafy nazveme, přirozeně, *acyklické*.

Definice 1.2.2 (Acyklický graf)**.** Graf $G = (V, E)$ nazveme *acyklický*, pokud neobsahuje cyklus o aspoň třech vrcholech (samotné vrcholy jsou totiž z definice vždy cykly).

Definice 1.2.3 (Strom)**.** Graf $G = (V, E)$ nazveme *stromem*, je-li souvislý a acyklický.

Na začátku sekce jsme slíbili ještě ekvivalentní definici stromu; ta činí začnou část důvodu užitečnosti stromů, především v informatice.

Ukazuje se totiž, že neexistence cyklů spolu se souvislostí způsobují, že mezi dvěma vrcholy vede vždy **přesně jedna cesta**. Po chvíli zamyšlení snad toto nepřichází jako nijak divoké tvrzení. Přeci, pokud by mezi vrcholy vedly cesty dvě, pak vrchol, kde se rozpojují, a vrchol, kde se opět spojují, by byly součástí cyklu uvnitř stromu, který jsme výslovně zakázali. Třeba překvapivější je fakt, že platí i opačná implikace.

Tvrzení 1.2.4 (Ekvivalentní definice stromu)**.** Graf $G = (V, E)$ je stromem ve smyslu definice 1.2.3 právě tehdy, když mezi každými dvěma

vrcholy G vede přesně jedna cesta.

Důkaz. Dokazujeme dvě implikace. Obě budeme dokazovat v jejich *kontrapozitivní* formě, tedy jako obrácenou implikaci mezi negacemi výroků. Lidsky, dokážeme, že (1) když existují vrcholy, mezi kterými nevede žádná nebo vede více než jedna cesta, pak G není strom, a (2) když G není strom, tak existují vrcholy, mezi kterými nevede žádná cesta nebo vede více než jedna.

(1) Pokud existují vrcholy, mezi kterými nevede cesta, pak G není souvislý, což odporuje definici stromu. Budeme tedy předpokládat, že existují vrcholy v, w, mezi kterými vedou různé cesty $v_1 \cdots v_n$ a $v'_1 \cdots v'_m$, kde $v_1 = v'_1 = v$ a $v_n = v'_m = w$. Myšlenka důkazu je najít cyklus obsahující vrchol, kde se cesty rozdělují, a vrchol, kde se opět spojují. Vizte obrázek 11a.

Ať r (od **r**ozpojení) je **největší** index takový, že $v_i = v'_i$ pro všechna $i \leq r$ (čili $v_r = v'_r$ je vrchol, ve kterém se cesty rozpojují). Ten určitě existuje, protože cesty se v nejhorším případě dělí už ve vrcholu $v = v_1$.

Podobně, ať s (od **s**pojení) je **nejmenší** index takový, že existuje $k \in \mathbb{Z}$ splňující $v_j = v'_{j+k}$ pro všechna $j \geq s$. Čili, vrchol $v_s = v'_{s+k}$ je vrchol, ve kterém se cesty opět spojily. Ovšem, mohlo se tak stát v okamžiku, kdy jsme po jedné cestě prošli více nebo méně vrcholů než po druhé – tento počet vyjadřuje ono číslo k. Takový vrchol jistě existuje, v nejhorším je to přímo koncový vrchol $w = v_n$.

Zřejmě platí $r < s$, jinak by cesty nebyly různé. Potom je ovšem například posloupnost vrcholů

$$(v_r, v_{r+1}, \ldots, v_s = v'_{s+k}, v'_{s+k-1}, \ldots, v'_r = v_r)$$

cyklem v G. Tedy ani v tomto případě G není strom.

(2) Když G není strom, tak není souvislý nebo obsahuje cyklus. Když G není souvislý, tak existují vrcholy, mezi nimiž nevede v G cesta, což protiřečí podmínce, aby mezi každým párem vrcholů vedla přesně jedna.

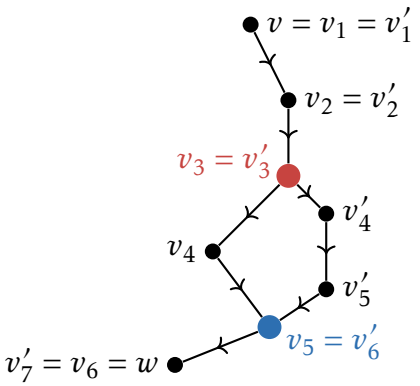
Budeme tedy předpokládat, že G obsahuje cyklus $v_1 v_2 \cdots v_n$ (tedy $v_n = v_1$ a $n \geq 3$). Pak ovšem pro libovolné indexy $i < j \leq n$ jsou

posloupnosti

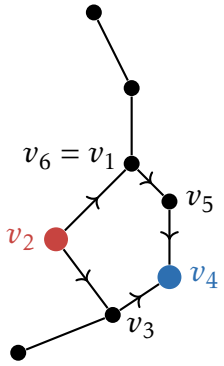
$$(v_i, v_{i+1}, \ldots, v_j) \quad \text{a} \quad (v_i, v_{i-1}, \ldots, v_1 = v_n, v_{n-1}, \ldots, v_j)$$

dvě různé cesty mezi v_i a v_j. Vizte obrázek 11b.

Tím je důkaz dokončen. □



(a) Část (1) důkazu tvrzení 1.2.4. Zde $r = 3, s = 5$ a $k = 1$. Sestrojený cyklus je $v_3v_4v_5v'_5v'_4v'_3$.



(b) Část (2) důkazu tvrzení 1.2.4. Zde $i = 2, j = 4$ a sestrojené cesty jsou $v_2v_3v_4$ a $v_2v_1v_5v_4$.

Obrázek 11: Ilustrace k důkazu tvrzení 1.2.4.

Cvičení 1.2.1. Dokažte, že je-li $T = (V, E)$ strom, pak $\#E = \#V - 1$.

Cvičení 1.2.2. Spočtěte, kolik existuje stromů na n vrcholech.

1.2.1 Minimální kostra

Ne všechny hrany jsou si rovny. Kterési se rodí krátké, jiné dlouhé; kterési štíhlé, jiné otylé; kterési racionální, jiné iracionální.

Často nastávají situace, kdy jeden potřebuje hranám grafu přiřadit nějakou hodnotu, obvykle číselnou, která charakterizuje klíčovou vlastnost této hrany. Při reprezentaci dopravní sítě grafem to může být délka silnice či její vytížení, při reprezentaci elektrických obvodů pak například odpor. V teorii grafů takové přiřazení hodnoty hranám grafu sluje *ohodnocení*.

Definice 1.2.5 (Ohodnocený graf)**.** Ať $G = (V, E)$ je graf. Libovolné zobrazení $w : E \to \mathbb{R}^+ = (0, \infty)$ nazveme *ohodnocením* grafu G. Trojici (V, E, w), kde w je ohodnocení G, nazveme *ohodnoceným grafem*.

Poznámka. Každý graf $G = (V, E)$ lze triviálně ztotožnit s ohodnoceným grafem (V, E, w), kde $w : E \to \mathbb{R}^+$ je konstantní zobrazení. Obvykle se volí konkrétně $w \equiv 1$, tedy zobrazení w takové, že $w(e) = 1$ pro každou $e \in E$.

Porozumění struktuře ohodnocených grafů může odpovědět na spoustu zajímavých (jak prakticky tak teoreticky) otázek. Můžeme se kupříkladu ptát, jak se nejlépe (vzhledem k danému ohodnocení) dostaneme cestou z jednoho vrcholu do druhého. Slovo „nejlépe“ zde chápeme pouze intuitivně. V závislosti na zpytovaném problému můžeme požadovat, aby cesta třeba minimalizovala či maximalizovala součet hodnot všech svých hran přes všechny možné cesty mezi danými vrcholy. Jsou však i případy, kdy člověk hledá cestu, která je nejblíže „průměru“.

Abychom pořád neříkali „součet přes všechny hrany cesty“, zavedeme si pro toto často zkoumané množství název *váha cesty*. Čili, je-li $\mathcal{P} := e_1 \cdots e_n$ cesta v nějakém ohodnoceném grafu G, pak její vahou rozumíme výraz

$$w(\mathcal{P}) := \sum_{i=1}^{n} w(e_i).$$

Zápis $w(\mathcal{P})$ můžeme vnímat buď jako zneužití zavedeného značení, nebo jako fakt, že jsme zobrazení w rozšířili z množiny všech hran na množinu všech cest v grafu G (kde samotné hrany jsou z definice též cesty).

Poznámka. Záměrně jsme užili slovního spojení *váha* cesty místo snad přirozenějšího *délka* cesty. V teorii grafů se totiž délkou cesty myslí obyčejně počet hran (nebo vrcholů), které obsahuje. Délka cesty $\mathcal{P} = e_1 \cdots e_n$ je tudíž n (resp. $n+1$), bo obsahuje n hran (resp. $n+1$ vrcholů).

Tento úzus svědčí účelu předchozí poznámky. Pokud totiž každý graf bez ohodnocení vnímáme vlastně jako ohodnocený graf, kde každá hrana má váhu přesně 1, pak váha každé cesty je rovna její délce.

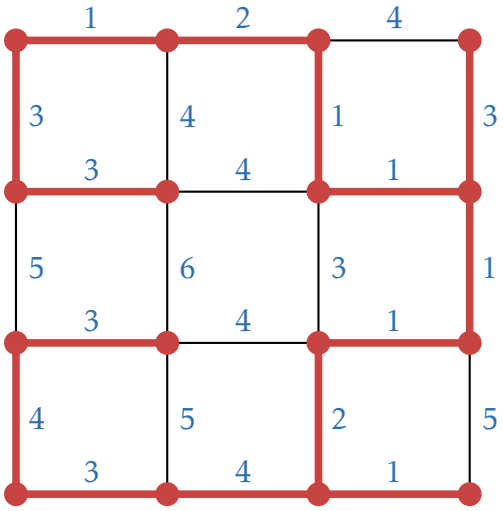
První (a nejjednodušší) problém, kterým se budeme zabývat, je nalezení

minimální kostry (angl. *spanning tree*).

Definice 1.2.6 (Minimální kostra)**.** Ať $G = (V, E, w)$ je **souvislý** ohodnocený graf. Ohodnocený graf $K = (V', E', w)$ nazveme *minimální kostrou* grafu G, pokud je souvislý, $V' = V$ (tedy K obsahuje všechny vrcholy G), $E' \subseteq E$ a

$$\sum_{e \in E'} w(e)$$

je minimální vzhledem ke všem možným volbám podmnožiny $E' \subseteq E$. Lidsky řečeno, graf K spojuje všechny vrcholy G tím „nejlevnějším“ způsobem vzhledem k ohodnocení w.



Obrázek 12: Minimální kostra grafu s ohodnocením w.

Pozorování. Minimální kostra ohodnoceného grafu je strom.

Důkaz. Kdyby minimální kostra nebyla strom, pak buď není souvislá, což jsme výslovně zakázali, nebo obsahuje cyklus. Tudíž se mezi nějakými dvěma vrcholy dá jít po více než jedné cestě, a proto můžeme přinejmenším jednu hranu z kostry odebrat. Protože každá hrana má kladné ohodnocení, snížili jsme tím součet hodnot všech hran. To je spor. □

Důsledek 1.2.7. Minimální kostra ohodnoceného stromu je s ním totožná.

Minimální kostra je zvlášť užitečná právě při návrhů elektrických obvodů, kdy je potřeba zařídit, aby všechna připojená zařízení čerpala co nejmenší množství energie. Protože elektřina proudí rychlostí světla, délka kabelu (pokud není zrovna mezigalaktický) nás příliš netrápí, ale právě odpor či kvalita/vodivost konkrétních spojů by mohly.

Další praktickou grafovou úlohou vedoucí na problém nalezení minimální kostry je potřeba spojit vzdálené servery. Korporace mají obvykle mnoho různých serverů rozmístěných po světě, jež spolu ale musejí sdílet data. Problém je v tom, že propojení mezi servery by nejen mělo vést k nejmenší možné prodlevě při přenosu dat (od toho **minimální**), ale nesmí ani obsahovat cykly (od toho **kostra**). Kdyby totiž cykly obsahovalo, pak by se při přenosu dat stalo, že by aspoň jeden server v tomto cyklu dostal aspoň dvakrát stejnou informaci z dvou různých zdrojů, ale v odlišný čas. Taková situace vede nezbytně dříve nebo později ke korupci dat; řekněme, když daný server už s obdrženou informací začal po přijetí provádět výpočet. Pro více detailů k tomuto využití minimálních koster vizte Spanning Tree Protocol.

1.2.2 Kruskalův algoritmus

Na problém nalezení minimální kostry souvislého ohodnoceného grafu existuje skoro až zázračně přímočarý algoritmus, pojmenovaný po americkém matematiku, Josephu B. Kruskalovi. Jeho základní myšlenkou je prostě začít s grafem K obsahujícím všechny vrcholy z V a přidávat hrany od těch s nejnižším ohodnocením po ty s nejvyšším tak dlouho, dokud nevznikne souvislý graf. Je potřeba pouze dávat pozor na cykly. K tomu stačí si pamatovat stromy (jako množiny vrcholů), které přidáváním hran vytvářím, a povolit přidání hrany jedině v případě, že spojuje dva různé stromy.

Pro zápis v pseudokódu vizte algoritmus 1. Manim s průběhem algoritmu s náhodně vygenerovaným hodnocením je k dispozici zde.

Algoritmus 1: Kruskalův algoritmus.

input : souvislý ohodnocený graf $G = (V, E, w)$, kde $V = \{v_1, \ldots, v_n\}$
output: množina hran E' minimální kostry grafu G

Inicializace;
$E' \leftarrow \emptyset$;
Množina hran, které nelze přidat (jinak by vznikl cyklus);
$X \leftarrow \emptyset$;
for $i \leftarrow 1$ **to** n **do**
    Každý strom nejprve obsahuje pouze jediný vrchol;
    $T_i \leftarrow \{v_i\}$;
Množina indexů pro pamatování sloučených stromů;
$I \leftarrow \{1, \ldots, n\}$;
Přidávám hrany, dokud mám pořád víc než jeden strom;
while $\#I > 1$ **do**
    $e \leftarrow$ libovolná hrana s minimální $w(e)$, která není v E' ani v X;
    $i \leftarrow$ index v I takový, že $s(e) \in T_i$;
    $j \leftarrow$ index v I takový, že $t(e) \in T_j$;
    if $i = j$ **then**
        Hrana spojuje vrcholy ve stejném stromě, jejím přidáním by vznikl cyklus;
        $X \leftarrow X \cup \{e\}$;
    else
        Hrana spojuje různé stromy. Přidávám ji do kostry;
        $E' \leftarrow E' \cup \{e\}$;
        $T_i \leftarrow T_i \cup T_j$;
        $I \leftarrow I \setminus \{j\}$;
return E';

Tvrzení 1.2.8. Kruskalův algoritmus je korektní.

Důkaz. Potřebujeme ověřit, že

(1) algoritmus provede pouze konečný počet kroků;

(2) algoritmus vrátí správnou odpověď.

Případ (1) je zřejmý, protože $\#E < \infty$, čili algoritmus přidá do E' pouze

konečně mnoho hran. Navíc, graf G je z předpokladu souvislý, a tedy vždy existuje hrana e spojující dva různé stromy T_i a T_j.

V případě (2) uvažme, že $K = (V, E', w)$ není minimální kostra G. V takovém případě se mohlo stát, že

(a) K není strom – tedy buď není souvislý nebo obsahuje cyklus;

(b) existuje podmnožina $E'' \subseteq E$ taková, že $K' = (V, E'', w)$ je strom a

$$\sum_{e \in E''} w(e) < \sum_{e \in E'} w(e).$$

Případ (a) lze vyloučit snadno, neboť souvislost K plyne ihned ze souvislosti G, tedy, jak již bylo řečeno v bodě (1), vždy lze nalézt hranu spojující do té doby dva různé stromy. Pokud K obsahuje cyklus, tak algoritmus musel v jednom kroku spojit hranou dva vrcholy ze stejného stromu, což je spor.

Pokud nastal případ (b), pak musejí existovat hrany $e'' \in E'' \setminus E'$ a $e' \in E' \setminus E''$ takové, že $w(e'') < w(e')$. Protože algoritmus zkouší přidávat hrany vždy počínaje těmi s nejmenší vahou, musel v nějakém kroku narazit na hranu e'' a zavrhnout ji. To ovšem znamená, že hrana e'' spojila dva různé vrcholy téhož stromu a graf $K' = (V, E'', w)$ obsahuje cyklus. To je spor s předpokladem, že K' je strom. Tedy, taková podmnožina $E'' \subseteq E$ nemůže existovat a K je vskutku minimální kostra G. □

Výstraha. Minimální kostra grafu **není jednoznačně určena**! Všimněte si, že na řádku 12 Kruskalova algoritmu volím **libovolnou** hranu, která má ze všech zatím nepřidaných nejnižší váhu a jejímž přidáním nevznikne cyklus.

Poznámka. V definici minimální kostry požadujeme, aby G byl souvislý graf. Pokud tomu však tak není, je pořád možné zkonstruovat minimální kostru pro každou část G, která souvislá je (pro každou jeho tzv. *komponentu souvislosti*), zkrátka tím, že Kruskalův algoritmus spouštíme opakovaně.

Cvičení 1.2.3 (Těžké ale fun)**.** Ať V je množina n bodů v rovině. Pro každé dva body $x, y \in V$ definujme váhu hrany xy jako vzdálenost bodů x a y. Čili, pokud $x = (x_1, x_2)$ a $y = (y_1, y_2)$, pak

$$w(xy) := \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}.$$

Vzniklý graf označme obyčejně G.

(a) Ukažte, že v každé minimální kostře G vede z každého vrcholu maximálně 6 hran.

(b) Ukažte, že existuje minimální kostra G, jejíž hrany se (jakožto úsečky v rovině) nekříží.

Cvičení 1.2.4. *Úplným grafem* na n vrcholech myslíme graf $G = (V, \binom{V}{2})$, tedy graf, mezi každým párem jehož vrcholů vede hrana. Takový graf se obvykle značí K_n. Najděte minimální kostru K_n a spočtěte její váhu (tj, součet vah všech jejích hran), je-li $V = \{1, \dots, n\}$ a

(a) $w(ij) = \max(i, j)$,

(b) $w(ij) = i + j$,

pro všechny páry $i, j \leq n$.

Cvičení 1.2.5. Dokažte, že když w (tj. ohodnocení G) je prosté zobrazení, pak je minimální kostra G určena jednoznačně.

1.3 Jordanovo centrum

V návaznosti na sekci o minimální kostře se rozhovoříme o jednom dalším optimalizačním problému – konkrétně hledání „centra“ ohodnoceného grafu.

Motivační úlohou je tzv. *facility location problem*, v přibližném překladu *úloha umístění střediska*. Jde o úlohu, kdy máte dánu dopravní síť sídlišť (obecně obydlených zón) a význačných uzlů, přes které se chtě nechtě musí jezdit (například velké křižovatky, Nuselák apod.). Hrany vedou mezi sídlišti či uzly, když od jednoho k druhému vede bezprostřední cesta (tedy cesta neprocházející žádným jiným sídlištěm nebo uzlem).

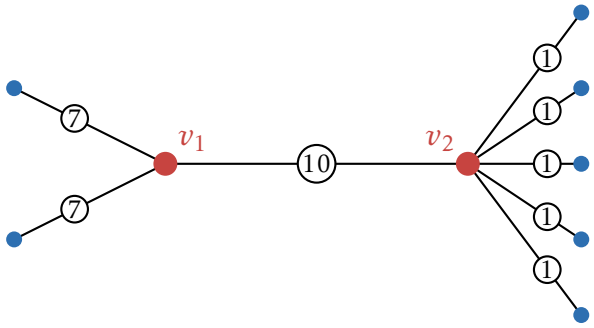
Pojďme si úlohu rozmyslet podrobně. Máme u nějakého uzlu v síti dopravních křižovatek postavit středisko. Bez ohledu na typ střediska nás pravděpodobně zajímá, aby se všichni z obydlených zón, které má toto středisko pokrýt, dostali k němu co nejsnáze. Ovšem, výraz „co nejsnáze“ je matematicky příliš vágní. Jistě hledáme optimální řešení, ale v jakém smyslu konkrétně?

Snad bude užitečné kýžený smysl optimality vyzkoumat na příkladě. Řekněme, že střediskem, jež potřebujeme umístit, je nemocnice. Co znamená, že je vzhledem k dané síti nemocnice *optimálně* umístěna? První, co by nás mohlo napadnout, je chtít, aby průměr vzdáleností od obydlených zón k nemocnici byl co nejmenší. To je přirozená myšlenka, ale jak si ihned rozmyslíme, vcelku morbidní.

U průměru totiž často nastává situace, že značný počet nízkých hodnot převáží nad zanedbatelným počtem hodnot vysokých. Uvažte síť uzlů a sídlišť danou grafem na obrázku 13, kde sídliště jsou značena modře, dopravní uzly červeně a ohodnocení hran, jak je vlastně zvykem, značí průměrnou dobu jízdy po těchto cestách.

V tomto případě by uzel v_2 jistě nabízel mnohem lepší průměrné řešení, neboť můžeme snadno spočítat, že průměrná doba jízdy od libovolného sídliště k němu je asi pět a půl minuty. Naopak, průměrná doba jízdy k uzlu v_1 činí těsně pod deset minut.

My se ale přesto rozhodneme postavit nemocnici v uzlu v_1. Proč? Totiž, při stavbě nemocnice nám nejde ani tolik o to, aby se do ní dostalo co nejvíce lidí co nejrychleji, ale **aby to nikdo neměl příliš daleko**. Kdybychom



Obrázek 13: Špatně vyvážená síť sídlišť a dopravních uzlů.

učinili opak a postavili nemocnici v uzlu v_2, pak by sice lidé ze sídlišť na pravé straně to měli do nejbližší nemocnice pouhou minutu, ale lidé z levých sídlišť by cestovali průměrně až sedmnáct minut. Do uzlu v_1 se dostane každý člověk nejpozději za jedenáct minut.

Tento způsob měření vhodnosti umístění nemocnice v dopravních sítích je skutečně v praxi používaný a je dozajista nepěkným příkladem volby menšího ze dvou zel. Tedy, není prioritou, aby se někomu dostalo pomoci velmi brzy, ale aby se nikomu nedostalo pomoci příliš pozdě.

Problém, jenž jsme právě zformulovali, je variantou výše zmíněného *facility location problem* (dále jen FLP), která sluje EFLP, tedy ***emergency*** *facility location problem*. Úlohou je nalézt takový uzel, jehož **maximální** vzdálenost ke všem ostatním uzlům je minimální, čili uzel pro umístění středisek jako jsou právě nemocnice a polikliniky, či obecněji „pohotovostní“ střediska.

Druhou známou variantou je SFLP, čili ***service*** *facility location problem* – úloha nalézt vhodný uzel pro umístění střediska „služeb“, kde je naopak žádoucí, aby nastala druhá z výše diskutovaných situací, aby se k němu přiblížilo co nejvíce lidí co nejrychleji. Ti, již cestují dlouho, budou na nejméně libý pád kalé nálady sedíce rozčileně ve voze a shrbení klejíce ustavičně v kolena, však nezhynou.

Formálně tedy hledáme uzel, který minimalizuje průměr všech vzdáleností od něj ke všem ostatním uzlům. Pro usnadnění výpočtu je dobré si uvědomit, že minimalizovat **průměr** všech vzdáleností je totéž, co minimalizovat **součet** všech vzdáleností, neboť počet uzlů se nemění (**Rozmyslete si to!**).

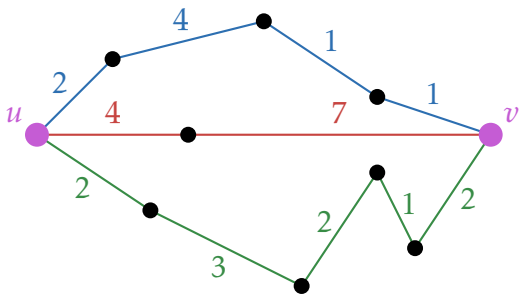
K formulaci obou úloh potřebujeme zavést základní pojem *vzdálenosti* mezi vrcholy v ohodnoceném grafu.

V zájmu strohosti vyjádření označíme pro libovolné dva vrcholy $u, v \in V$ grafu (V, E, w) symbolem $\mathcal{P}(u, v)$ množinu všech cest mezi u a v. Speciálně, $\mathcal{P}(v, v) = \{v\}$, čili cesta z vrcholu do něj samého obsahuje pouze tento jeden vrchol, a $\mathcal{P}(u, v) = \emptyset$, pokud mezi u a v nevede v G cesta.

Definice 1.3.1 (Vzdálenost v grafu). Ať $G = (V, E, w)$ je ohodnocený graf a $v, w \in V$. *Vzdálenost* mezi u a v v grafu G, značenou $d_G(u, v)$ (z angl. **d**istance), definujeme jako

$$d_G(u, v) := \begin{cases} \min_{\mathcal{P} \in \mathcal{P}(u,v)} w(\mathcal{P}), & \text{pokud } \mathcal{P}(u, v) \neq \emptyset; \\ \infty, & \text{pokud } \mathcal{P}(u, v) = \emptyset. \end{cases}$$

Lidsky řečeno, vzdáleností mezi vrcholy je váha nejkratší cesty mezi nimi vedoucí, pokud taková existuje.



Obrázek 14: Zde $d_G(u, v)$ je váha nejkratší, to jest modré, cesty.

Abychom našli pro daný graf $G = (V, E, w)$ řešení EFLP, musíme najít takový vrchol, který minimalizuje největší možnou vzdálenost od něj ke všem ostatní vrcholům G. Takový vrchol (nebo vrchol**y**?) nazveme *Jordanovým centrem* grafu G, po žabožroutím počtáři, Marie E. C. Jordanovi.

Samozřejmě, nezajímá-li nás vzdálenost ke **všem** vrcholům, jako je tomu v příkladě umístění nemocnice, uvážíme pouze maximum vzdáleností k relevantním vrcholům (tedy k sídlištím). Na principu úlohy to nic nemění.

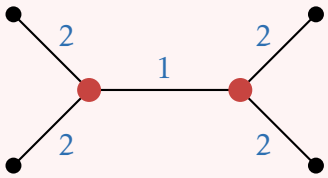
Formálně, *excentricita* vrcholu $v \in V$ je kvantita $e(v) := max_{u \in V} d_G(v, u)$, tedy maximum přes všechny vzdálenosti od něj k ostatním vrcholům. Toto číslo vyjadřuje, jak moc je vrchol vzdálen od „ideálního centra“ grafu,

tedy od bodu, od kterého by každý vrchol byl stejně daleko. Samozřejmě, toto ideální centrum málokdy existuje, takže hledáme pouze vrchol s nejmenší excentricitou, s nejmenší *odchylkou* od centra.

Definice 1.3.2 (Emergency Facility Location Problem). Ať $G = (V, E, w)$ je **souvislý** ohodnocený graf. Úlohu nalézt vrchol s minimální excentricitou nazveme EFLP. Jejím *řešením* je vrchol s touto vlastností, tedy vrchol $c \in V$ splňující

$$e(c) = \min_{v \in V} e(v).$$

Výstraha. Řešení EFLP **není jednoznačně určeno**! Vizte např. graf na obrázku 15.



Obrázek 15: Vrcholy s minimální excentricitou v grafu $G = (V, E, w)$.

Definice 1.3.3 (Jordanovo centrum). Množinu všech řešení EFLP pro graf $G = (V, E, w)$ nazýváme *Jordanovým centrem* grafu G.

Definice 1.3.4 (Poloměr grafu). Je-li c vrchol v Jordanově centru grafu $G = (V, E, w)$, pak hodnotu $e(c)$ nazýváme *poloměrem* grafu G a značíme ji $\rho(G)$.

Poznámka. V definici EFLP jsme požadovali, aby byl graf souvislý. To z ryze technického hlediska není nutné, protože excentricita vrcholu je definována i pro nesouvislý graf. Uvědomme si ale, že pro nesouvislý graf je excentricita každého vrcholu rovna ∞, tedy Jordanovým centrem je celý graf a úloha poněkud pozbývá smyslu.

Obdobným způsobem si formalizujeme i SFLP.

Nyní chceme nalézt vrchol, který minimalizuje průměr (nebo ekvivalent-

ně součet) vzdáleností od něj k, buď všem nebo pouze zajímavým, vrcholům. Názvosloví zde poněkud selhává a tomuto součtu přes vzdálenosti ke všem vrcholům se rovněž často přezdívá *excentricita*. Abychom pojmy odlišili, slovo „excentricita“ přeložíme a budeme říkat „výstřednost“. Tedy, *výstředností* vrcholu $v \in V$ v ohodnoceném grafu $G = (V, E, w)$ myslíme číslo

$$e'(v) := \sum_{u \in V} d_G(v, u).$$

Definice 1.3.5 (Service Facility Location Problem)**.** Ať $G = (V, E, w)$ je souvislý ohodnocený graf. Úlohu nalézt vrchol s minimální výstředností nazveme SFLP. Jejím řešením je vrchol $c' \in V$ s touto vlastností, tedy takový, že

$$e'(c') = \min_{v \in V} e'(v).$$

Pochopitelně, stejně jako EFLP, i SFLP může mít více řešení. Avšak, podle našeho nejlepšího vědomí se množině řešení SFLP nijak význačně neříká. Minimální výstřednost se občas nazývá *status* grafu G. Etymologie tohoto názvosloví je spjata s novodobým využitím SFLP při vyvažování nervových sítí a její zevrubné objasnění je nad rámec tohoto textu.

Definice 1.3.6 (Status grafu)**.** Ať $G = (V, E, w)$ je souvislý ohodnocený graf. Když $c' \in V$ je řešení SFLP, pak se hodnota $e'(c)$ nazývá *status* grafu G a značí se $\sigma(G)$.

Výstraha. Řešení EFLP a řešení SFLP mohou (ale **nemusí**) být disjunktní. Vrátíme-li se ke grafu na obrázku 13, pak řešením EFLP je pouze vrchol v_1, zatímco řešením SFLP je pouze vrchol v_2.

Následující podsekci dedikujeme obecnému algoritmu, pomocí nějž lze také hledat řešení jak EFLP, tak SFLP. Poznamenáme však, že existují mnohem efektivnější algoritmy, jež naleznou řešení těchto úloh; tyto ale vyžadují o poznání hlubší poznatky teorie grafů.

1.3.1 Floydův-Warshallův algoritmus

Vlastně jedinou výzvou na cestě k vyřešení FLP je umět aspoň rámcově efektivně najít vzdálenost mezi všemi dvojicemi vrcholů. Bohužel, žádná pěkná věta jako Pythagorova, která umožňuje okamžitě počítat vzdálenosti bodů v Eukleidovských prostorech, v teorii grafů neexistuje a existovat nemůže.

Připomeňme, že v ohodnoceném grafu $G = (V, E, w)$ je vzdálenost vrcholů $u, v \in V$ definována jako váha nejkratší cesty. Poznamenejme, že zde slovo „nejkratší“ chápeme ve smyslu *váhy* cesty (tedy součtu vah všech jejích hran), nikoli ve smyslu *délky* cesty (tedy počtu jejích hran). Asi bychom měli říkat „nejlehčí“ cesta, to je ale poněkud nepřirozené...

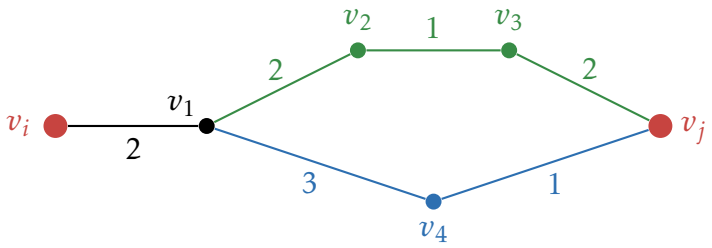
Floydův-Warshallův algoritmus nesouvisí přímo s FLP. Je to algoritmus, který nalezne vzdálenost (tedy váhu nejkratší cesty) mezi všemi dvojicemi vrcholů. Je však zřejmé, jak znalost této informace vede okamžitě k řešení EFLP, resp. SFLP. V moment, kdy známe vzdálenost každého vrcholu od každého, stačí pouze spočítat excentricitu, resp. výstřednost, každého vrcholu a vybrat pouze ty s minimální.

Jistě není překvapením, že zkoušet z každého vrcholu všechny možné cesty do všech ostatních vrcholů a z nich vybírat ty nejkratší, není dvakrát efektivní. Floydův-Warshallův algoritmus stojí na dvou principech, jimž se především v programování říká *rekurze* a *dynamické programování*. Jejich úplné pochopení a nabytí schopnosti využívat může být časově náročné, ale Floydův-Warshallův algoritmus jich využívá velmi přímočaře. Postupně si rozebereme, že ze znalosti váhy nejkratší cesty mezi dvěma vrcholy, **která využívá jen nějakou podmnožinu ostatních vrcholů**, lze zvětšováním této podmnožiny získat nakonec váhu nejkratší cesty mezi těmito vrcholy v celém grafu (odtud *rekurze*). Dále, ze znalosti vzdálenosti mezi určitými dvojicemi vrcholů můžeme rychle určit vzdálenost mezi párem, u kterého jsme ji zatím neznali (odtud *dynamické programování*).

Naší prací v této podsekci je dát předchozímu odstavci formální podobu. Ať $G = (V, E, w)$ je souvislý ohodnocený graf, kde $V = \{v_1, \ldots, v_n\}$. Definujme zobrazení $\delta : \{1, \ldots, n\}^3 \to \mathbb{R}^+$ následovně. Ať $\delta(i, j, k)$ je váha nejkratší cesty mezi v_i a v_j **využívající pouze vrcholy z podmnožiny vrcholů** $\{v_1, \ldots, v_k\}$ (samozřejmě, kromě počátečního v_i a koncového v_j). I když je G souvislý, tak taková cesta nemusí vždy existovat; v takovém případě je $\delta(i, j, k) = \infty$. Je snadné si uvědomit, že $d_G(v_i, v_j) = \delta(i, j, n)$, čili vzdále-

nost mezi vrcholy v grafu G je váha nejkratší cesty, která smí využít jeho všechny vrcholy.

Je zřejmé, že $\delta(i,j,k) \leq \delta(i,j,k-1)$, neboť máme jeden vrchol navíc, a přes ten může vést nějaká kratší cesta. Základní, a vlastně jedinou, myšlenkou Floydova-Warshallova algoritmu je pozorování, že když nastane situace, kdy $\delta(i,j,k) < \delta(i,j,k-1)$, pak ta kratší cesta musí využívat vrchol v_k. Ovšem, takovou cestu lze rozdělit na dvě – na cestu z v_i do v_k a cestu v_k do v_j. Původní cesta $v_i \cdots v_k \cdots v_j$ využívala pouze vrcholy z $\{v_1,\ldots,v_k\}$, takže obě její části, $v_1 \cdots v_k$ i $v_k \cdots v_j$ využívají pouze vrcholy z $\{v_1,\ldots,v_{k-1}\}$. Zároveň to musejí být právě ty nejkratší cesty mezi v_i a v_k a v_k a v_j, jinak by celková cesta $v_1 \cdots v_k \cdots v_j$ nebyla ta nejkratší. Mrkněte na obrázek 16.



Obrázek 16: Zde $\delta(i,j,3) = 7$, ale $\delta(i,j,4) = 6$.

Odtud plyne, že $\delta(i,j,k)$ je buď

- rovna $\delta(i,j,k-1)$, pokud přes vrchol v_k nevede žádná kratší cesta z v_i do v_j, nebo
- rovna součtu vah nejkratší cesty z v_i do v_k a nejkratší cesty z v_k do v_j, pokud přes vrchol v_k nějaká kratší cesta z v_i do v_j vede. V symbolech je tento součet roven $\delta(i,k,k-1) + \delta(k,j,k-1)$.

Předchozí body se dají shrnout do jednoho zápisu, neboť $\delta(i,j,k)$, jakožto váha **nejkratší** cesty, je vždy menším z obou množství. Konkrétně, vzoreček stojící za celým Floydovým-Warshallovým algoritmem je tento:

$$\delta(i,j,k) = \min(\delta(i,j,k-1), \delta(i,k,k-1) + \delta(k,j,k-1)). \qquad (\triangle)$$

Možná už vás napadá, jak samotný algoritmus bude fungovat. Přeci, čísla $\delta(i,j,k)$ mohu spočítat pro všechny dvojice $(i,j) \in \{1,\ldots,n\}^2$ v moment, kdy znám $\delta(i,j,k-1)$ opět pro všechny dvojice (i,j). Jenže, $\delta(i,j,0)$ je váha hrany

mezi v_i a v_j, pokud existuje, jinak ∞. Symbolicky,

$$\delta(i,j,0) = \begin{cases} w(v_iv_j), & \text{pokud } v_iv_j \in E, \\ \infty, & \text{jinak.} \end{cases}$$

To znamená, že znám hodnotu $\delta(i,j,0)$ pro všechny dvojice (i,j), a tedy postupným zvyšováním k umím spočítat i všechny hodnoty $\delta(i,j,k)$, a tím i nakonec, pro $k = n$, vzdálenosti mezi všemi páry vrcholů.

Nyní, když jsme myšlenku algoritmu doufám objasnili, zbývá jeho postup nějak rozumně zapsat. Naštěstí to lze velmi přímočaře, dokonce výrazně snadněji než například Kruskalův algoritmus.

Vlastně nám jde o to postupně nacházet kratší a kratší cesty mezi všemi dvojicemi vrcholů. Ten fakt, že hledáme vzdálenosti mezi vskutku **všemi** dvojicemi, nám umožňuje díky závěrům v předchozích odstavcích toto dělat efektivně.

Algoritmus vytváří zobrazení $D : V^2 \to [0,\infty]$, kterýmť si vlastně „pamatuje“ váhu zatím nejkratší nalezené cesty mezi každými dvěma vrcholy. Na začátku algoritmu je tudíž $D(v,v) := 0$ pro všechny $v \in V$ a $D(u,v) := w(uv)$ pro všechny dvojice $(u,v) \in V^2$, mezi kterými vede hrana. Pro všechny ostatní dvojice (u,v) pokládáme $D(u,v) := \infty$.

Opírajíce sebe o předšedší úvahy víme, že budeme muset učinit v algoritmu n kroků (to je počet vrcholů G) a v k-tém kroku vždy počítat hodnoty $\delta(i,j,k)$ pro všechny dvojice (i,j). Přejeme si tudíž, aby hodnoty z předchozího kroku, tj. $\delta(i,j,k-1)$, byly před spuštěním k-tého kroku všechny již zakódovány v zobrazení D, jakožto čísla $D(v_i,v_j)$, představující **zatím** nejkratší známé cesty mezi všemi dvojicemi vrcholů. Rovnost ($\triangle$) se pročež překládá (**v rámci k-tého kroku algoritmu**) do rovnosti

$$D(v_i,v_j) = \min(D(v_i,v_j), D(v_i,v_k) + D(v_k,v_j)).$$

Po skončení n-tého kroku bude tedy zobrazení D přesně odpovídat zobrazení d_G. Konečně můžeme přikročit k samotnému pseudokódu, jenž si prohlédněte níže.

Cvičení 1.3.1. Dokažte, že Floydův-Warshallův algoritmus selže, když připustíme i záporná ohodnocení hran.

Algoritmus 2: Floydův-Warshallův algoritmus.

input : souvislý ohodnocený graf $G = (V, E, w)$, kde $V = \{v_1, \ldots, v_n\}$
output: zobrazení $D : V^2 \to [0, \infty]$ takové, že $D \equiv d_G$

```
1  Inicializace;
2  for i ← 1 to n do
3      for j ← 1 to n do
4          if i = j then
5              Vzdálenost od vrcholu k němu samému je 0;
6              D(v_i, v_i) ← 0;
7          else
8              if v_i v_j ∈ E then
9                  Vzdálenost mezi různými vrcholy je váha hrany, pokud
                    existuje;
10                 D(v_i, v_j) ← w(v_i v_j);
11             else
12                 Jinak nastavíme hodnotu na ∞;
13                 D(v_i, v_j) ← ∞;
14 Postupně pro každé k od 1 do n budeme zmenšovat hodnoty D(v_i, v_j)
    pro všechny dvojice (i, j);
15 for k ← 1 to n do
16     for i ← 1 to n do
17         for j ← 1 to n do
18             D(v_i, v_j) ← min(D(v_i, v_j), D(v_i, v_k) + D(v_k, v_j));
19 return D;
```

1.4 Vzdálenost vrcholů

Sekci motivujeme úlohou „jako ze života“, která ve mně budí jistou míru nostalgie, jelikož jsem ji řešil na konci prvního ročníku v rámci zkoušky z programování. Dovolil jsem si ji ovšem literárně obohatit.

Úloha (Rodinný výlet)**.** Je prodloužený víkend a slezská rodina Koláčků plánuje cyklistický výlet oblastí Karviná. Jedou děda Koláček se svou chotí, babičkou Koláčkovou, a jejich čtyři uřvaná rozmazlená vnoučata – Matouš, Marek, Lukáš a Jan.

Z Třince do Orlové dánť jest směr a, i přes relativní nenáročnost terénu, vnoučata ustavičně fňukají, že chtějí jet tou nejkratší trasou. Děda Koláček, dobrodinec ten od kosti, snaží se vnoučatům vyhovět a nejkratší trasu úpěnlivě hledá. Do vřavy se přidává babička Koláčková, která ví, že trasa z Třince do Orlové vede přes mnoho malých vesnic, mnohože z nich hostí aspoň jednu hospodu.

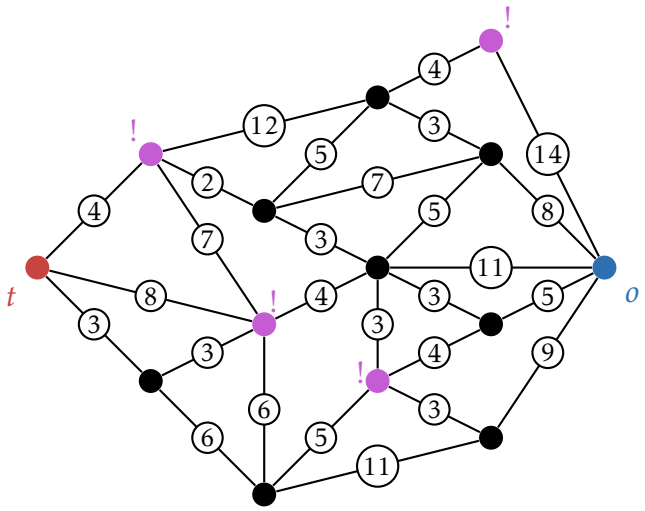
Vědouc velmi dobře, že každá hospoda zbrzdí cestu na nejméně půl hodiny a že i při mimoděčné zmínce o blízké hospodě si děda Koláček počíná mnouti pivní pupek, trvá babička Koláčková na tom, aby vybraná cesta z Třince do Orlové vedla přes vesnice, které jsou tak daleko od hospod, jak je to jen možné. Křik vnoučat brzy ji však přesvědčí, že délka trasy převažuje nad množstvím hospod, ke kterým se po cestě přiblíží.

Koláčkovic rodina tudíž stojí před nelehkým úkolem vybrat ze všech nejkratších cest z Třince do Orlové tu, která je hospodám co nejvíce vzdálena.

Brzy znaven, děda Koláček posílá nezkrotná vnoučata s konečně neosedlanými bicykly domů a výlet do Orlové se odkládá na příští školní prázdniny. Pomozme mu jej zorganizovat předem.

Úlohu si modelujeme ohodnoceným grafem $G = (V, E, w)$. Vrcholy představují vesnice, případně města, mezi Třincem a Orlovou, z nichž některé jsou označeny vykřičníkem mínícím přítomnost jedné či více hospod.

Nejprve se soustředíme na nejpodstatnější část úlohy, tou jest nalezení nejkratší cest mezi Třincem a Orlovou, kterážto města si v zájmu strohosti



Obrázek 17: Graf obcí mezi Třincem a Orlovou.

označíme písmeny $t, o \in V$.

První, co by člověka mohlo napadnout je spustit Floydův-Warshallův algoritmus. Zde je však nutno si uvědomit, že ten není možné modifikovat tak, aby hledal vzdálenost mezi dvěma konkrétními vrcholy. Totiž, z popisu jeho průběhu plyne, že váhu nejkratší cesty mezi dvěma vrcholy, která vede pouze přes vrcholy z množiny $\{v_1, \ldots, v_k\}$, umím spočítat teprve tehdy, když znám váhy nejkratších cest používajících pouze vrcholy $v_1, \ldots, v_{k-1}$ mezi **všemi dvojicemi vrcholů** v grafu $G = (\{v_1, \ldots, v_n\}, E, w)$. Odtud plyne, že Floydův-Warshallův algoritmus vždy najde vzdálenosti mezi všemi dvojicemi vrcholů.

Samozřejmě je též možné najít **všechny** cesty mezi danými dvěma vrcholy a z nich vybrat tu nejkratší. Tento postup je však ještě výrazně méně efektivní než Floydův-Warshallův algoritmus.

Není však radno zoufat, bo algoritmus pro relativně efektivní nalezení vzdálenosti mezi určenými dvěma vrcholy v grafu existuje. Nosí jméno svého tvůrce, holandského informatika a programátora Edsgera W. Dijkstry (čteno „dajkstry“). Není složitý, ale zcela jistě je méně přímočarý než Kruskalův i Floydův-Warshallův algoritmus.

Je ironické, že najít efektivní algoritmus na výpočet vzdálenosti mezi dvěma vrcholy je výrazně obtížnější než na výpočet vzdálenosti mezi všemi dvojicemi vrcholů.

1.4.1 Dijkstrův algoritmus

Dijkstrův algoritmus má mnoho společného s Floydovým-Warshallovým algoritmem, například rovněž v každém kroku zlepšuje zatím nejlepší známou vzdálenost od *zdrojového* vrcholu k ostatním. Jeho základní myšlenka je však jiná. Dijkstrův algoritmus využívá tzv. „průchod do šířky“, což znamená, že se „šíří“ od jednoho zvoleného *zdrojového vrcholu* nejprve do jeho sousedů (tj. do vrcholů s ním spojených hranou). Při průchodu si u každého vrcholu pamatuje zatím nejlepší nalezenou cestu ze zdrojového vrcholu do něj, a kdykoli vstoupí do dalšího vrcholu, ověří, zda cesta, po které se do něj dostal, není kratší než dosavad nejlepší.

Tento postup sám není nijak revoluční a průchod do šířky je asi tak starý jako grafy samotné. Myšlenka, se kterou Edsger Dijkstra přišel, však na svou dobu revoluční byla, neboť se jednalo o tehdy nejefektivnější způsob nalezení vzdálenosti mezi vrcholy v ohodnoceném grafu. U každého vrcholu si pamatoval váhu zatím nejkratší nalezené cesty od zdroje k němu. Uvědomil si, že **pokud vybírá vrcholy vždy od těch s minimální uloženou hodnotou**, tak v moment, **kdy projde všechny sousedy nějakého vrcholu**, pak už **je zapamatovaná hodnota v tomto vrcholu nejlepší možná**, tedy se jedná o skutečnou vzdálenost od zdroje k tomuto vrcholu a už se do něj nikdy nemusí znovu vracet.

Důvod, proč tomu tak je, není na první pohled zřejmý a objasníme ho podrobně v důkazu správnosti Dijkstrova algoritmu.

Ještě před slovním popisem a představením pseudokódu je dobré se zamyslet, o kolik více efektivní je Dijkstrův způsob oproti prostému průchodu do šířky. Pro úplnost:

- Průchod do šířky vždy putuje od vrcholu ke všem jeho sousedům, od těch zase k jejich sousedům (mezi nimiž je samozřejmě i onen původní vrchol) a tak dále.
- Dijkstrův algoritmus nahlédne z právě vybraného vrcholu do jeho sousedů, ale pak nepokračuje nutně jimi. Naopak, právě vybraný vrchol již „uzamkne“ a pokračuje nějakým s minimální uloženou zatím nejkratší cestou ze zdroje do něj.

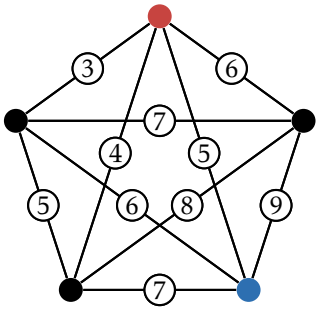
Kvůli tomu, že jeden ze sousedů každého vrcholu je přesně ten samý vrchol, projdu v naprosto nejhorším případě při běžném průchodu do šířky

každý vrchol tolikrát, kolik přes něj vede cest.

Naopak, Dijkstrův algoritmus *zakazuje* vracet se do vrcholů, jejichž všichni sousedi již byli navštíveni. Tedy, každý vrchol je navštíven v nejhorším tolikrát, kolik má sousedů. Je snadné si rozmyslet, o kolik je v průměrném případě počet sousedů vrcholu nižší než počet cest jím procházejících.

Největší časový rozdíl mezi obvyklým průchodem do šířky a Dijkstrovým algoritmem je vidět v tzv. *úplných grafech*, tedy grafech, ve kterých je každý vrchol spojen s každým. V úplném grafu na n vrcholech, který se často značí K_n, definuje libovolná podmnožina vrcholů cestu. Již dlouho víme, že všech podmnožin množiny o n prvcích je 2^n. Tedy, algoritmus průchodu do šířky udělá při hledání vzdálenosti mezi dvěma vrcholy v úplném grafu vždy 2^n kroků. Oproti tomu, není těžké dokázat (ale my to tu dělat nebudeme), že Dijkstrův algoritmus udělá v grafu na n vrcholech vždy nejvýše n^2 kroků.

Na obrázku 18 vidíte úplný graf na 5 vrcholech. Upřímně doporučuji, abyste si tvrzení předchozího odstavce vyzkoušeli v praxi.



Obrázek 18: Ohodnocený úplný graf na pěti vrcholech s vyznačeným zdrojovým a cílovým vrcholem.

Nyní přistoupíme ke slovnímu popisu Dijkstrova algoritmu. Algoritmus striktně vzato nepočítá vzdálenost mezi dvěma vrcholy, anobrž vzdálenost od *zdrojového vrcholu* ke všem ostatním.

Podobně jako Floydův-Warshallův algoritmus, i Dijkstrův algoritmus vytváří postupně zobrazení $d : V \to [0, \infty]$ takové, že po jeho skončení platí $d(v) = d_G(s, v)$ pro všechna $v \in V$ a nějaký pevně zvolený zdrojový vrchol $s \in V$. Na začátku algoritmu je pročež $d(s) = 0$ a $d(v) = \infty$ pro všechny $s \neq v \in V$. V každém kroku algoritmu navíc existuje množina X „zakázaných“ vrcholů, do kterých už není možné se dívat ani vracet. V moment, kdy pro-

cházím sousedy právě vybraného vrcholu, ignoruji ty, které již byly někdy zakázány.

Jeden krok algoritmu vypadá takto:

(1) Označ jako zvolený vrchol libovolný $v \in V$ **s minimální** $d(v)$.

(2) Pro každého souseda u zvoleného vrcholu v porovnej zatím nejkratší známou cestu z s do u, to jest $d(u)$, s váhou zatím nejkratší cesty z s do u vedoucí přes v, to jest $d(v) + w(uv)$. Je-li druhá hodnota menší, změň $d(u)$ na $d(v) + w(uv)$.

(3) Vrchol v označ jako zakázaný.

(4) Pokud ještě existuje vrchol, který není zakázaný, opakuj (1).

Jak jsme již psali – fakt, že jako další vrchol v pořadí volíme ten s minimální známou nejkratší cestou z s, umožňuje se nikdy do vrcholů, jejichž sousedy projdeme, nevracet, není samozřejmý. Výklad o obecném chodu Dijkstrova algoritmu zakončíme prezentací jeho pseudokódu pro **souvislý** ohodnocený graf $G = (V, E, w)$ s pevně zvoleným zdrojovým vrcholem $s \in V$. Ihned poté dokážeme jeho správnost.

Pro stručnost vyjádření zavedeme ještě značení

$$n(v) := \{u \in V \mid uv \in E\},$$

čili označíme výrazem $n(v)$ (od angl. ***n**eighbour*), množinu všech sousedů vrcholu $v \in V$.

Algoritmus 3: Dijkstrův algoritmus

input : souvislý ohodnocený graf $G = (V, E, w)$ s počátečním vrcholem $s \in V$
output: zobrazení $d : V \to [0, \infty]$ takové, že $d \equiv d_G(s, -)$

```
1  Inicializace;
2  d(s) ← 0;
3  for v ∈ V \ {s} do
4      d(v) ← ∞;
5  X ← ∅;
6  while X ≠ V do
7      v ← libovolný vrchol s minimální d(v) takový, že v ∉ X;
8      for u ∈ n(v) \ X do
9          if d(u) > d(v) + w(uv) then
10             d(u) ← d(v) + w(uv);
11     X ← X ∪ {v};
12 return d;
```

Tvrzení 1.4.1. Dijkstrův algoritmus je korektní.

Důkaz. Musíme ověřit, že Dijkstrův algoritmus je konečný a počítá správně.

Konečnost je zřejmá. Vrchol $v \in V$ s minimální $d(v)$ vždy existuje (i kdyby ta hodnota měla být ∞), a tedy nastane chvíle, kdy v X jsou všechny vrcholy grafu G. V ten moment $X = V$, cyklus končí a algoritmus vrací zobrazení d.

Korektnost ukážeme indukcí podle počtu zakázaných vrcholů, tedy podle $\#X$. Naším indukčním předpokladem bude, jako tomu obvykle v důkazech indukcí bývá, že algoritmus funguje. To přesně znamená, že v každém kroku

(a) je $d(v) = d_G(s, v)$ pro všechny vrcholy $v \in X$;

(b) je $d(u)$ váha nejkratší cesty z s do u vedoucí pouze přes vrcholy v X, pro všechna $u \in V \setminus X$.

Předpoklad (a) říká v lidské mluvě, že zakazování vrcholů si opravdu můžeme dovolit, tedy že do zakázaných vrcholů již nejde nalézt kratší cestu.

Význam předpokladu (b) je těžší dekódovat. Představme si spíše, co by se stalo, kdyby **neplatil**. Pak by existovala nějaká kratší cesta do nezakázaného vrcholu u přes zakázané vrcholy. Ovšem, přesně pro to, že tyto vrcholy jsou již zakázané, nemohu během zbytku průběhu algoritmu již nikdy tuto cestu objevit. Pokud by se stalo, že zrovna tato cesta je tou nejkratší, pak již zároveň nemohu nalézt hodnotu $d_G(s,u)$ a algoritmus selže.

Počáteční případ $\#X = 1$ je snadný. V tomto stádiu jsme zrovna zakázali zdrojový vrchol s a pro každého jeho souseda u jsme položili $d(u) = w(su)$, jelikož $d(s) = 0$ a $d(u)$ bývala hodnota ∞. Tedy, zakázaný vrchol je jedině s a $d(s) = d_G(s,s) = 0$, čímž jsme ověřili platnost (a). Jediné vrcholy, do nichž vede cesta přes zakázané vrcholy, tedy přes vrchol s, jsou sousedé s, a váha nejkratší cesty je zřejmě $w(su)$ pro každého souseda $u \in n(s)$. To dokazuje (b).

Nyní rozebereme případ pro obecnou $\#X$ s předpokladem platnosti (a) a (b) pro $\#X-1$. Nejprve si uvědomíme, že v tomto případě je výrok (b) platný automaticky. □